

TU DRESDEN

SERVICE AND CLOUD COMPUTING

PRAKTIKUM

TimeTracker

Eine Microservice-Anwendung für das bessere Zeitmanagement

Autoren:

Sinthujan THANABALASINGAM

Wieland STRAUSS

Dozentin:

Dr.-Ing. Iris BRAUN

22. Januar 2019



Inhaltsverzeichnis

1	Über TimeTracker	2
1.1	Inhalt der Anwendung	2
1.2	Einordnung der Arbeit	2
2	Arbeitsablauf	2
2.1	Team	2
2.2	Vorgehensweise	2
3	Technologien	3
3.1	Übersicht verwendeter Technologien, Software und Formate	3
3.2	Architektur	5
3.3	Schnittstellenbeschreibung	7
3.4	Sicherheit	7
4	Bedienungsanleitung des Clients	8
4.1	Build	8
4.2	Registrieren und Login	9
4.3	TimeTracken	10
5	Feedback und Kritik am Praktikum	14
6	Anhang	14



1 Über TimeTracker

1.1 Inhalt der Anwendung

TimeTracker ist eine Anwendung um die Zeit im Überblick zu behalten. Wenn der Nutzer registriert ist, kann er sich jederzeit auf den Server einloggen und die Aufnahme einer Aktivität starten. Nach beenden seiner Aufgabe wird die Aufnahme gestoppt.

Dem Benutzer wird eine Übersicht über seine aufgenommene Zeit angeboten. Dadurch bekommt er einen besseren Überblick über seine aufgebrauchte Zeit und kann sich in Zukunft besser einteilen.

Zusätzlich bietet die Plattform eine globale Statistik über alle Benutzer.

1.2 Einordnung der Arbeit

Die Anwendung wurde als Prüfungsvorleistung im Rahmen einer praktischen Übung der Lehrveranstaltung *Service and Cloud Computing* des Lehrstuhls Rechnernetze der TU Dresden entwickelt. Dozentin und Betreuerin ist Frau Dr.-Ing. Iris Braun.

2 Arbeitsablauf

2.1 Team

Das Entwicklerteam besteht aus den Studenten Sinthujanan Thanabalasingam (Master Informatik, 4. Semester)und Wieland Strauß (Diplom Informatik, 7. Semester).

Schwerpunkt von Sinthu war die Kozeptionierung und der Entwurf, bei Wieland das Organisatorische und alle Belange drum herum.

Die Implementation erfolgte weitgehend im Pair Programming. Es gab keine spezifische Aufteilung zwischen Backend- und Frontend-Entwicklung.

2.2 Vorgehensweise

Nach der Ideenfindung wurde zuerst die Architektur entworfen. Die Entwicklung wurde nahezu vollständig nach dem Prinzip des TDD (Test Driven Development)

betrieben. Die API wurde parallel mit *Swagger* erstellt. Auch die Verwendung von Docker und die Entwicklung des Frontends wurden zeitgleich umgesetzt. Zuletzt wurde die Anwendung auf einen Server von DigitalOcean ausgerollt

Der Arbeitsablauf organisierte sich aus mehreren Treffen. Bei diesem wurden die Planung durchgeführt, der Großteil der Anwendung geschrieben und Aufgaben für die selbstständige Durchführung festgelegt. Aufgrund der verschiedenen Expertise wurde Pair Programming durchgeführt.

3 Technologien

3.1 Übersicht verwendeter Technologien, Software und Formate

Die in der Anwendung implementierten Technologien aus Tabelle 1 sind in Abbildung 1 noch einmal schematisch zugeordnet. Die Anwendung (blau) läuft in einem Docker-Container (dunkelblau). Über die API (grün) kommuniziert die Weboberfläche (hellblau) mit der Anwendung. Die sich in Planung befindende App hat den gleichen Zugriffspunkt.

Eine Zuordnung der Technologien zur Architektur erfolgt in Abbildung 3

Anmerkung: Die Andoid-App wurde nicht im Rahmen der Lehrveranstaltung fertig gestellt und ist somit noch ausstehend. Da sie aber für eine spätere Nutzung angedacht ist und sie ebenfalls über die REST-Schnittstelle kommuniziert wurde sie im Entwurf berücksichtigt.

Tabelle 1: Technologien

Name	Version	Verwendung
Java	JDK 1.8.0	Verwendete Programmiersprache
Maven	3.1.0	Build-Management-Tool
Spring Boot	2.0.2	Java Framework für Backends
MySQL	5.7.0	Relationale Datenbank
Netflix Zuul	1.3.1	Edge Service für dynamisches Routen, Monitoring und Sicherheit
Netflix Eureka	1.9.2	Service-Registry
JSON	-	Datenaustauschformat
REST	-	Programmierparadigma für Webservices
React	16.1.1	JavaScript Bibliothek für User Interfaces
single-spa	2.6.0	JavaScript Framework für Front-End Micro-services
OpenAPI (Swagger)	3.0	Definition der REST-Schnittstelle; Automatisches Generieren von Java-Interfaces
Postman	6.5.3	API Development Environment, Testen der API
Docker	18.09.0	Umgebung für Container, Deployment

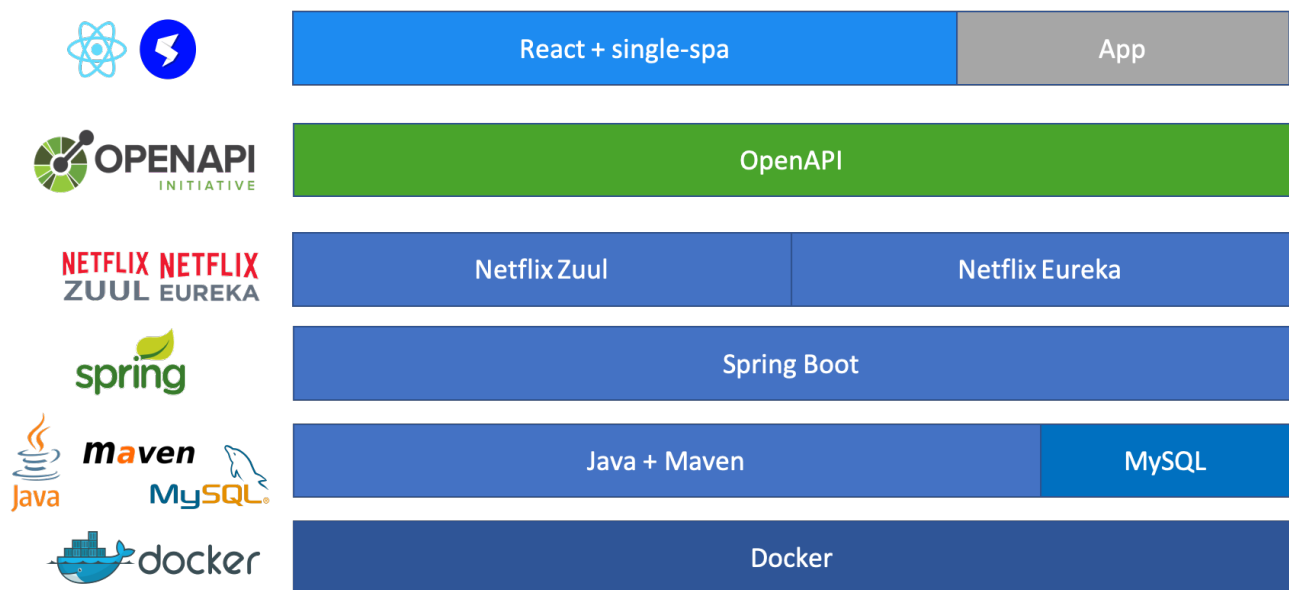


Abbildung 1: Technologie Stack

3.2 Architektur

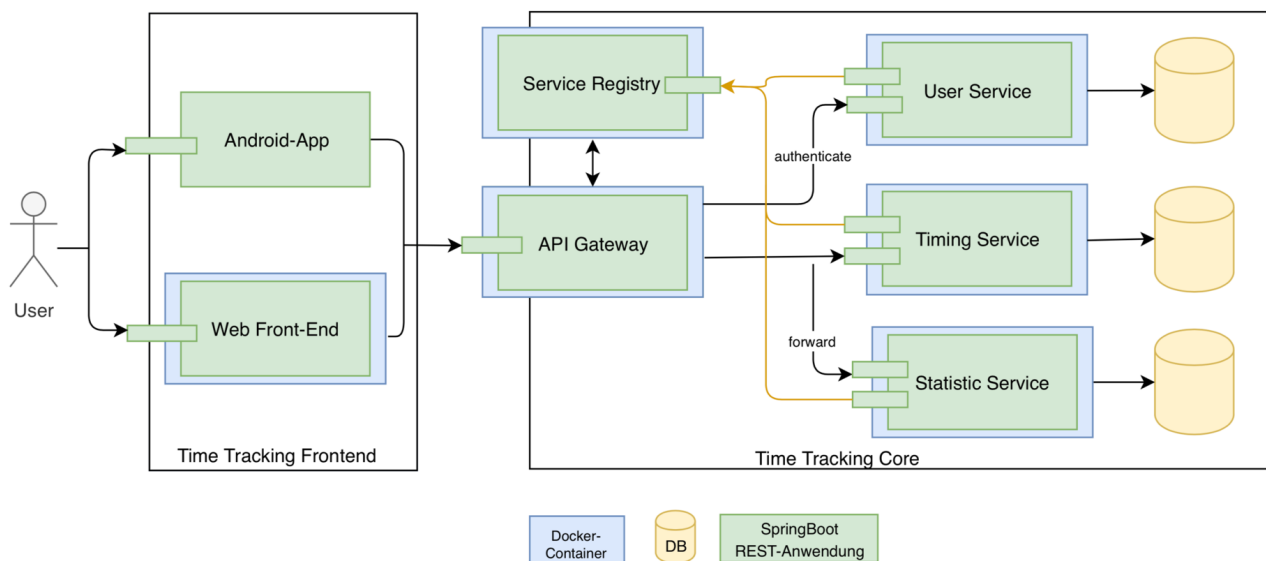


Abbildung 2: Architektur von TimeTracker

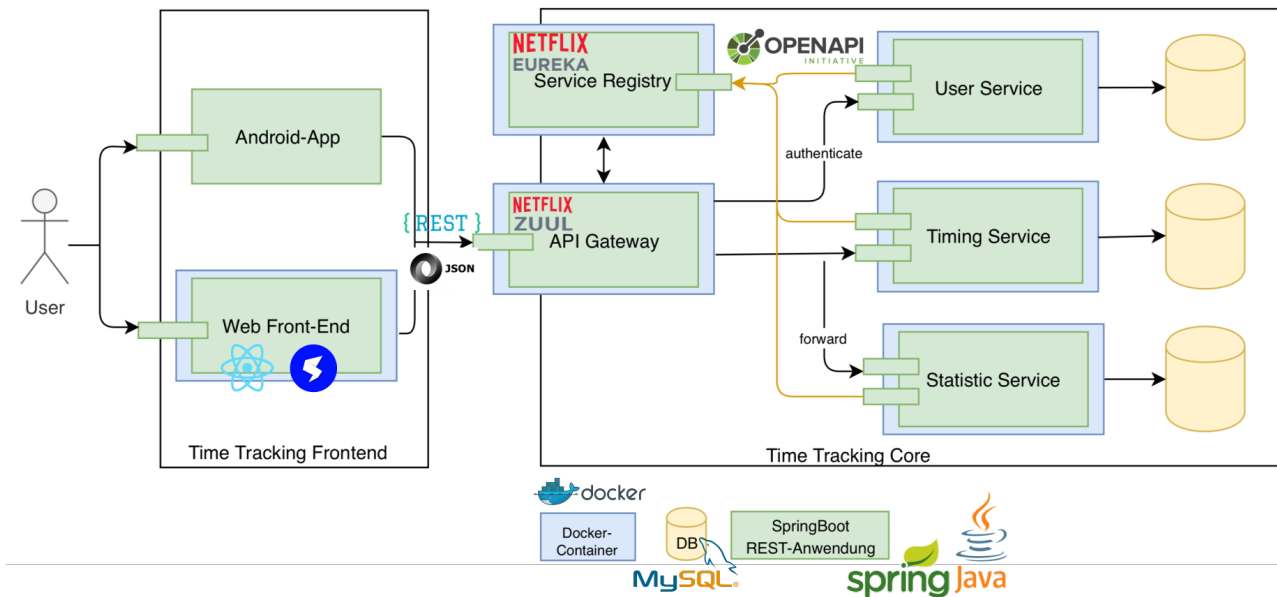


Abbildung 3: Architektur mit zugeordneten Technologien

Die Funktionen der Services

- User Service
 - Registrierung
 - Login
- Timing Service
 - Anlegen/ Abruf von Aktivitäten
 - Anlegen/ Abruf von Records
 - Abruf von Statistiken
- Frontend Service
 - Aufruf von UI
- Service Registry
 - Hält Services vor

- API Gateway
 - Nutzt Service Registry
 - Mapping der Requests auf Service

Durch die Micro Service Architektur war es möglich rolling Updates zu machen, was insbesondere im Bereich der UI zum Einsatz kam.

3.3 Schnittstellen

Die API wurde nach dem OpenAPI-Standard entwickelt und generierte automatisch ein Interface in Java. Eine Auflistung aller Methoden ist über die folgenden Links zu erreichen. Zusätzlich befindet sich im Anhang (Seite 15) ein Ausdruck der automatisch generierten Dokumentation.

- frontend-service:
https://app.swaggerhub.com/apis/SCC_Group4/frontend-service/
- timing-service:
https://app.swaggerhub.com/apis/SCC_Group4/timing-service/
- user-service:
https://app.swaggerhub.com/apis/SCC_Group4/user-service/

3.4 Sicherheit

Die Sicherheit wurde mittels **JSON Web Token (JWT)** und umgesetzt. Dabei erhält der Benutzer beim Login einen Token, mittels derer er sich gegenüber den Diensten authentifizieren kann. Das Token wird im localStorage des Browsers gespeichert und bei jedem Request im Header mitgeführt.

Des Weiteren wird die Sicherheit der Verbindung mittels der Transportverschlüsselung **HTTPS** gewährleistet.

4 Bedienungsanleitung des Clients

Your Time is precious.

Track your investments!

TimeTracker bietet folgende Funktionen:

- Time Tracking
- Anlegen und Löschen von Aktivitäten
- Tracking für verschiedene Aktivitäten
- Zuordnen mit Tags
- Abruf von Statistiken (Benutzer/Global)

Der Dienst ist online zu erreichen unter <http://iamtrent.de>

Lokale Installationen sind zu erreichen unter localhost:9123


4.1 Build

Zum Build bitte folgende Zeilen Ausführen:

```
mvn clean install
docker-compose -f docker/docker-compose.yml up -d
```

4.2 Registrieren und Login






Your time is precious. Track your investments!

TimeTracker is a tool that allows you to track the amount of time for different activities that you take part in daily in your life. TimeTracker helps to find hot spots or recognize patterns in the way you spend your time.

We offer the following features:

- Create Activities and categories
- Store time records of your activities
- See statistics and find patterns in your time management.



Sign in to TimeTracker

USERNAME or EMAIL

PASSWORD

[Sign in](#)

[New here? Create an account!](#)

By signing up you accept the TimeTracker privacy policies and general terms.

Abbildung 4: Login

Willkommen bei TimeTracker! Auf unserer Startseite (Abb. 4) kannst du dich mit deinen Benutzerdaten einloggen. Du hast noch kein Benutzerkonto? Kein Problem, unterhalb des *Sign In* Buttons ist ein Link, der zur Registrierung führt.

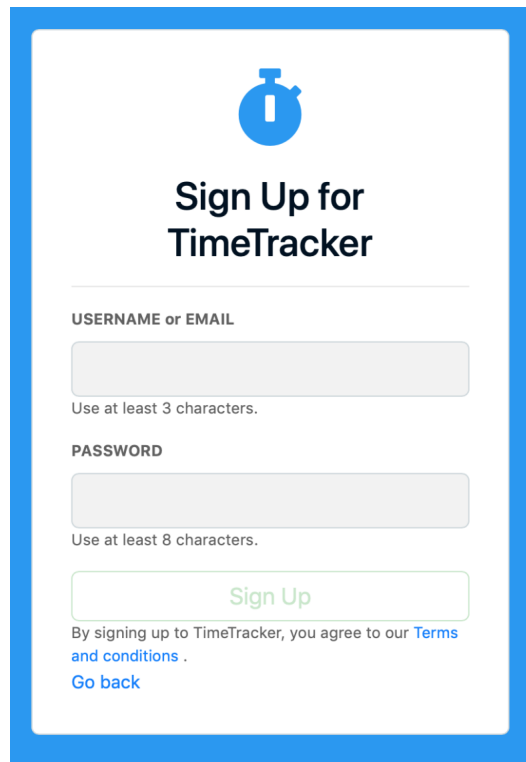
The image shows a registration form for 'TimeTracker' enclosed in a blue border. At the top center is a blue circular icon with a white stopwatch symbol. Below the icon, the text 'Sign Up for TimeTracker' is displayed in a bold, black font. A horizontal line separates the title from the input fields. The first input field is labeled 'USERNAME or EMAIL' in a small, grey font. Below this field is a light grey rectangular input box, and underneath it, the text 'Use at least 3 characters.' is shown in a small, grey font. The second input field is labeled 'PASSWORD' in a small, grey font. Below this field is another light grey rectangular input box, and underneath it, the text 'Use at least 8 characters.' is shown in a small, grey font. Below the password field is a green rectangular button with the text 'Sign Up' in a white font. At the bottom of the form, there is a line of text: 'By signing up to TimeTracker, you agree to our [Terms and conditions](#) .', followed by a blue link 'Go back'.

Abbildung 5: Registrieren

Um die zu registrieren musst du nur einen Usernamen oder eine Email, sowie ein Passwort eintragen. Beachte, dass das Passwort mindestens 8 Zeichen haben muss, vorher ist der Button nicht anklickbar! Nach dem du dich erfolgreich registriert hast wirst du wieder auf unsere Startseite (Abb. 4) geleitet, wo du dich ab sofort mit deinen Nutzerdaten anmelden kannst.

Wenn du einmal angemeldet bist kannst du dich jeder Zeit mit einem Klick auf *Logout* oben rechts wieder abmelden.

4.3 TimeTracken

Wenn du dich das erste mal einloggst, sind noch keine Aktivitäten vorhanden. Aktivitäten sind die Aktionen, für die du deine Zeit trackst. Füge einfach eine neue Aktivität hinzu, in dem du auf *+ New Activity* klickst (Abb. 6).

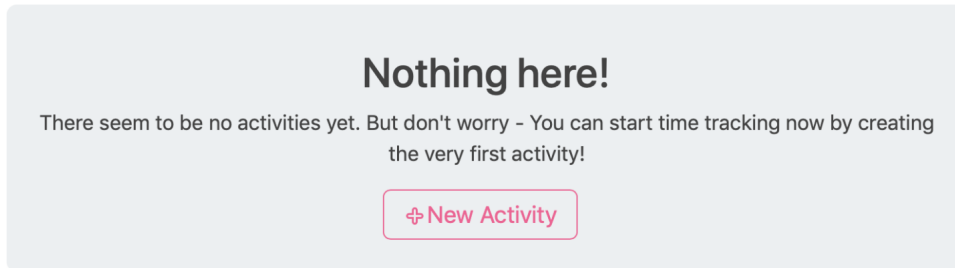


Abbildung 6: Keine Aktivitäten angelegt

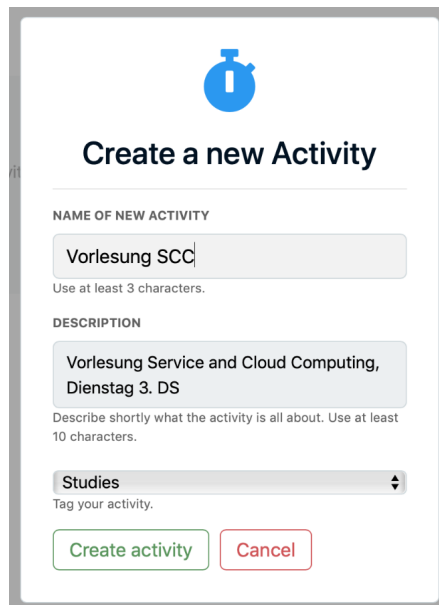
A screenshot of the "Create a new Activity" form in the Time Tracker app. The form has a white background with a blue header containing the app's logo. The title "Create a new Activity" is centered at the top. Below the title, there are three input fields: "NAME OF NEW ACTIVITY" with the text "Vorlesung SCC", "DESCRIPTION" with the text "Vorlesung Service and Cloud Computing, Dienstag 3. DS", and a dropdown menu for "Tag" with "Studies" selected. Each input field has a small instruction below it: "Use at least 3 characters." for the name, "Describe shortly what the activity is all about. Use at least 10 characters." for the description, and "Tag your activity." for the tag. At the bottom of the form are two buttons: "Create activity" (green) and "Cancel" (red).

Abbildung 7: Aktivität anlegen

Nun kannst du eine Aktivität anlegen (Abb. 7. Gibst ihr einen Namen und eine Beschreibung, damit du später weißt, wofür du sie angelegt hast. Die Beschreibung muss mindestens 10 Zeichen lang sein. Danach verbindest du deine Aktivität mit einem Tag. Tags sind sowas wie Kategorien. Jede Aktivität hat ein Tag und dadurch eine Zuordnung zu einem deiner Lebensbereiche.

Tags sind beispielsweise Studies, Sport und Relax. Mit Hilfe der Tags kannst du später schauen, wie viel Sport du gemacht hast, auch wenn du deine Zeit in den Aktivitäten SSchwimmen und Rad fahren aufgenommen hast.

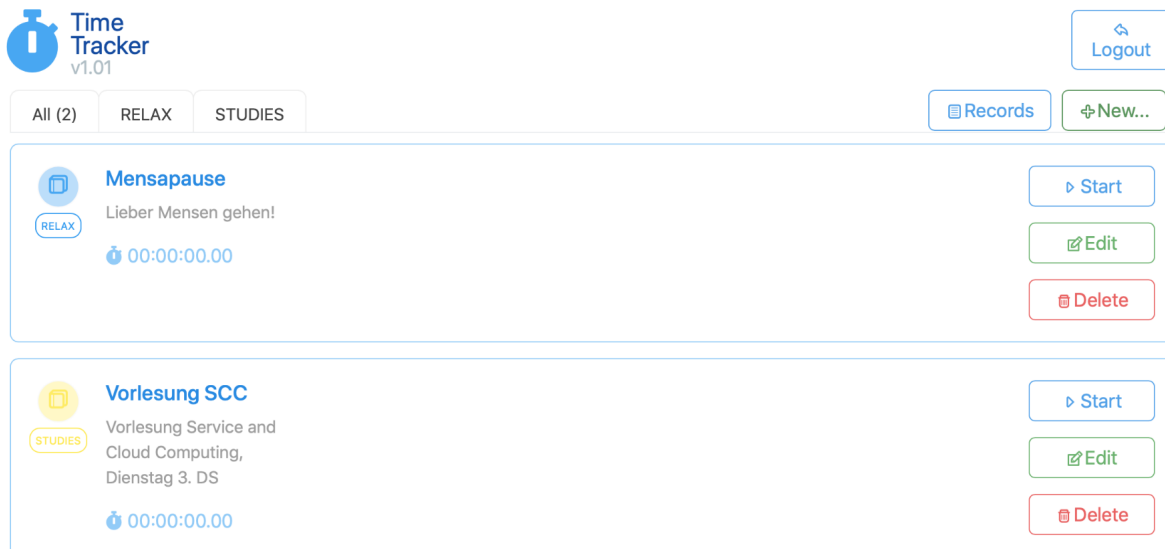


Abbildung 8: Übersicht: Aktivitäten

Nun hast du Aktivitäten und eine Übersicht (Abb. 8). Hier kannst du mit einem Klick auf *Start* ein Record beginnen. Records sind die einzelnen Aufnahmen, die du machst. Zusammen addiert ergeben sie die gesamte Zeit, die du für eine Aktivität aufbringst. Wie du siehst, blinkt nun die Uhr und die Zeit läuft. Mit einem Klick auf den selben Button stoppst du das Tracking deiner Aktivität und beendest den Record. Du kannst dich zwischendurch auch abmelden oder von einem anderen Gerät aus einloggen, die Aufnahme geht einfach weiter.

Mit dem Button *+ New* kannst du weitere Aktivitäten anlegen und mit dem Button *Delete* die jeweilige auch wieder löschen. In dem du auf *Edit* klickst oder einfach auf den Titel/ die Beschreibung kannst du die Inhalte updaten.

Über die Reiter oben kannst du deine Aktivität nach den von dir genutzten Tags filtern.

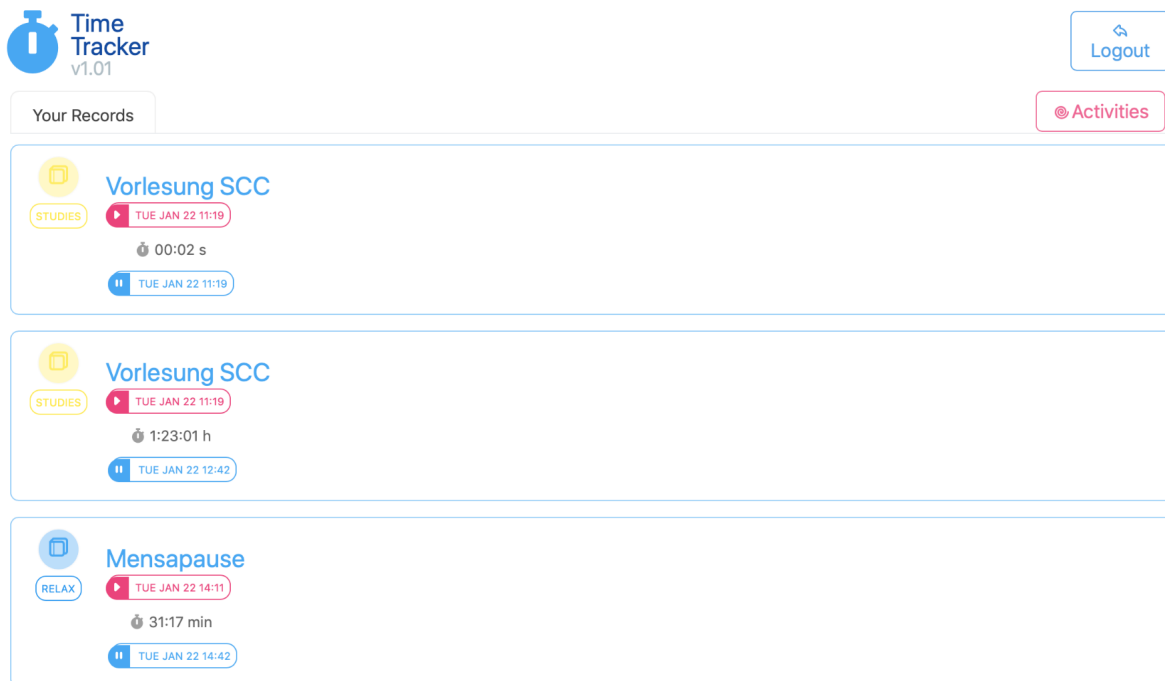


Abbildung 9: Aufgenommene Records

Wenn du *Records* anwählst kommst du zu einer Übersicht all deiner Zeitaufnahmen (Abb. 9). Hier steht für welche Aktivität, von wann bis wann und wie lange die Aufnahmen war.

5 Feedback und Kritik am Praktikum

Allgemein: Es machte uns sehr viel Spaß diese Anwendung zu entwickeln. Ein gutes Klima im Team trug dazu bei. Aufgrund des Unterschiedes in den Bereichen der praktischen Erfahrung konnte bei uns auch gut Wissen ausgetauscht werden, beziehungsweise entwickelte sich mit der Zeit eine Art Mentoring. Bei der Umsetzung wurde sehr deutlich, wie das Praktikum die Vorlesungsinhalte vertieft und im direkten Bezug dazu tritt. Das Wissen wurde dadurch anschaulicher und praktisch umgesetzt.

Spezifikation der Anforderungen: Durch die in manchen Teilen unspezifische Aufgabenstellung lässt sich schwer Abschätzen, was es genau zu erreichen gilt. Positiv daran ist die Möglichkeit der freien Auswahl der Vorgehensweise, Umsetzung und Tools. Die Zwischenpräsentation und das Feedback darauf hat aber maßgeblich zum Verständnis bei geführt, was erwartet wird.

6 Anhang

- Generierte API-Dokumentation User Service
- Generierte API-Dokumentation Timing Service
- Generierte API-Dokumentation Frontend Service

User Service

No description provided (generated by Swagger Codegen <https://github.com/swagger-api/swagger-codegen>)

More information: <https://helloverb.com>

Contact Info: hello@helloverb.com

Version: 0.0.1

BasePath: /

All rights reserved

<http://apache.org/licenses/LICENSE-2.0.html>

Access

1.

Methods

[[Jump to Models](#)]

Table of Contents

[Default](#)

- [GET /users/{uuid}](#)
- [POST /users/info](#)
- [GET /users/{name}/{uuid}](#)
- [POST /signup](#)

Default

GET /users/{uuid}

[Up](#)

(getUserName)

Path parameters

uuid (required)

Path Parameter —

Return type

[UserData](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [UserData](#)

POST /users/info

[Up](#)

(getUserNamesFromUuidList)

Consumes

This API call consumes the following media types via the Content-Type request header:

- application/json

Request body

body [UuidData](#) (required)

Body Parameter —

Return type

array[[UserData](#)]

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK

GET /users/{name}/uuid

[Up](#)

(getUserUuid)

Path parameters

name (required)

Path Parameter —

Return type

[UuidData](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [UuidData](#)

POST /signup

[Up](#)

(signup)

Consumes

This API call consumes the following media types via the Content-Type request header:

- application/json

Request body

body [UserData](#) (required)

Body Parameter —

Return type

[RegistrationStatus](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [RegistrationStatus](#)

Models

[[Jump to Methods](#)]

Table of Contents

1. [RegistrationStatus](#)
2. [UserData](#)
3. [UuidData](#)
4. [operationResponse](#)

RegistrationStatus

[Up](#)

UserData

[Up](#)

Data that is transferred to register a new or update an existing user.

username (optional)

[String](#) The user name.

password (optional)

[String](#) The hashed password of the user

role (optional)

[String](#)

UuidData

[Up](#)

uuids (optional)

[array\[String\]](#)

operationResponse

[Up](#)

error (optional)

[String](#) Used to indicate error messages.

data (optional)

[array\[Object\]](#)

Timing Service

No description provided (generated by Swagger Codegen <https://github.com/swagger-api/swagger-codegen>)

More information: <https://helloreverb.com>

Contact Info: hello@helloreverb.com

Version: 0.0.1

BasePath: /

All rights reserved

<http://apache.org/licenses/LICENSE-2.0.html>

Access

1.

Methods

[[Jump to Models](#)]

Table of Contents

[Default](#)

- [POST /activities/{activityId}/records](#)
- [POST /activities/](#)
- [DELETE /activities/{activityId}](#)
- [GET /activities/](#)
- [GET /activities/{activityId}](#)
- [GET /activities/{activityId}/records](#)
- [GET /activities/stats](#)
- [GET /activities/records](#)
- [PUT /activities/{activityId}](#)

Default

POST /activities/{activityId}/records

[Up](#)

(addActivityRecord)

Triggers start or end of a record for a given activityID

Path parameters

activityId (required)

Path Parameter —

Return type

[operationResponse](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [operationResponse](#)

POST /activities/

[Up](#)

(createActivity)

Consumes

This API call consumes the following media types via the Content-Type request header:

- application/json

Request body

body [activity](#) (required)

Body Parameter —

Return type

[operationResponse](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [operationResponse](#)

DELETE /activities/{activityId}

[Up](#)

(deleteActivity)

Removes an activity with given id and all its associated records from the database.

Path parameters

activityId (required)

Path Parameter —

Return type

[operationResponse](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [operationResponse](#)

GET /activities/

[Up](#)

(getActivities)

Returns all activities

Return type[operationResponse](#)**Produces**

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [operationResponse](#)**GET** /activities/{activityId}[Up](#)**(getActivity)**

Returns activity with given id

Path parameters**activityId (required)***Path Parameter —***Return type**[operationResponse](#)**Produces**

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [operationResponse](#)**GET** /activities/{activityId}/records[Up](#)**(getActivityRecords)**

Returns all records for an activity

Path parameters**activityId (required)***Path Parameter —***Return type**[operationResponse](#)**Produces**

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [operationResponse](#)

GET /activities/stats

[Up](#)

(getGlobalActivityStats)

Returns statistics for this application

Return type

[operationResponse](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [operationResponse](#)

GET /activities/records

[Up](#)

(getRecords)

Returns all records for a user

Query parameters

tag (optional)

Query Parameter – The tag filter for user records

activityUuid (optional)

Query Parameter – The uuid of the activity of interest

Return type

[operationResponse](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [operationResponse](#)

PUT /activities/{activityId}

[Up](#)

(updateActivity)

Updates the activity meta data.

Path parameters

activityId (required)

Path Parameter –

Return type

[operationResponse](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [operationResponse](#)

Models

[[Jump to Methods](#)]

Table of Contents

1. [activity](#)
2. [activityRecord](#)
3. [operationResponse](#)

activity

[Up](#)

An activity that can be tracked

name (optional)

[String](#) The name of the activity

tag (optional)

[String](#) The tag of the activity

uuid (optional)

[String](#) The uuid of the activity

description (optional)

[String](#) The description of the activity

owneruuid (optional)

[String](#) The uuid of the user that owns this activity

activityRecord

[Up](#)

A record of an activity that a user has submitted

activityuuid (optional)

[String](#) The uuid of the activity.

activityName (optional)

[String](#) The name of the activity.

time (optional)

[Date](#) The point in time a record update was issued (e.g. the request was sent to the server) format: date-time

startTime (optional)

[Date](#) The point in time this record was created format: date-time

endTime (optional)

[Date](#) The point in time this record ended format: date-time

uuid (optional)

[String](#) The uuid of this record

duration (optional)

[*Integer*](#) The duration of the record format: int32

tag (optional)

[*String*](#) The tag of the activity

state (optional)

[*String*](#)

Enum:

STARTED

ENDED

operationResponse

[Up](#)

error (optional)

[*String*](#) Used to indicate error messages. Null if no error occurred

data (optional)

[*array\[Object\]*](#) Attached data of response

Platform Frontend Service

No description provided (generated by Swagger Codegen <https://github.com/swagger-api/swagger-codegen>)

More information: <https://helloverb.com>

Contact Info: hello@helloverb.com

Version: 0.1-oas3

BasePath: /

All rights reserved

<http://apache.org/licenses/LICENSE-2.0.html>

Access

Methods

[[Jump to Models](#)]

Table of Contents

[Default](#)

- [GET /uiservices](#)

Default

GET /uiservices

[Up](#)

(getUiServices)

Return type

[UiServices](#)

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses

200

OK [UiServices](#)

Models

[[Jump to Methods](#)]

Table of Contents

1. [UiService](#)
2. [UiServices](#)

UiService

[Up](#)

Services with name and url.

serviceId (optional)

[*String*](#)

serviceName (optional)

[*String*](#)

serviceAddress (optional)

[*String*](#)

UiServices

[Up](#)

services (optional)

[*array\[UiService\]*](#)