

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

27. 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	3
3	Technologien	3
3.1	Übersicht verwendeter Technologien, Software und Formate	3
3.2	Architektur	5
3.3	Schnittstellen	7
3.4	Sicherheit	7
4	Bedienungsanleitung des Clients	9
4.1	Build	9
4.2	Registrieren und Login	10
4.3	TimeTracken	11
5	Feedback und Kritik am Praktikum	15
6	Anhang	15



1 Über TimeTracker

1.1 Inhalt der Anwendung

TimeTracker ist eine Anwendung, die es ermöglicht, Zeitinvestitionen zu managen. Ein registrierter Nutzer kann das System nutzen, um den Zeitaufwand bestimmter Aktivitäten aufzuzeichnen. Er kann dazu Activities anlegen und Stoppuhren starten, welche die Dauer der Aufgaben aufzeichnen.

Dem Nutzer wird eine Übersicht über seine aufgenommenen Zeiten angeboten. Dadurch bekommt er einen besseren Überblick darüber, wie viel Zeit er in welche Aufgaben investiert hat. Das Tool soll den Nutzer dabei unterstützen, Hotspots oder Muster in seinem Verhalten zu identifizieren um gegebenenfalls seine Prioritäten anzupassen.

Zusätzlich bietet die Plattform eine anonymisierte globale Statistik, die alle Zeitinvestitionen aller Nutzer repräsentiert und nur von eingeloggten Nutzern eingesehen werden kann.

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 Sinthujan Thanabalasingam (Master Informatik, 4. Semester) und Wieland Strauß (Diplom Informatik, 7. Semester).

Sinthujan beschäftigte sich schwerpunktmäßig mit Kozeptionierung und Implementation der Architektur des Backends sowie Entwurf und dem Design des Frontends. Bei Wieland lag der Fokus bei der Implementation im Pair-Programming, der Dokumentation sowie allen organisatorischen Aufgaben.

Die Implementation erfolgte phasenweise im Pair-Programming. Zu Beginn wurde der Fokus auf die benötigte Backend-Funktionalität gelegt, es gab jedoch vorwiegend 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 Test Driven Development (TDD) betrieben. Eine API-Definition wurde zu Beginn mit *Swagger* erstellt und parallel zur Entwicklung des Backends erweitert, wenn neue Anforderungen identifiziert wurden. Da die vorgeschlagene Architektur sich Micro-Services zu nutze macht, wurde bereits während der frühen Entwicklungs- und Testphase Docker als Containertechnik eingesetzt. Parallel zur Backend-Entwicklung wurde ein auf React basierendes Frontend geschaffen.

Zuletzt wurde die Anwendung auf einem V-Server von DigitalOcean ausgerollt, mit HTTPS abgesichert und unter der Domain <https://iamtrent.de> verfügbar gemacht.

Der Arbeitsablauf organisierte sich aus mehreren Treffen. Bei diesen wurde die Planung durchgeführt, ein Großteil der Anwendung geschrieben und Aufgaben für die selbstständige Durchführung festgelegt. Aufgrund der unterschiedlichen Expertise der Entwickler wurde Pair-Programming praktiziert. Weite Teile der Backend-Entwicklung fanden auch nach Absprache bei den Treffen selbstständig, zum Beispiel zu Hause statt.

3 Technologien

3.1 Übersicht verwendeter Technologien, Software und Formate

Die Anwendung besteht aus verteilten Micro-Services, die mit Hilfe von Docker containerisiert wurden. Jeder Micro-Service basiert auf einem Spring-Boot Container und nutzt den Netflix-Stack (Eureka und Zuul), um sich bei einer Service-Registry zu registrieren. Die nun auffindbaren Services können gegenseitig miteinander kommunizieren und ein API-Gateway (Zuul) delegiert Requests an die

entsprechenden Services. Die Micro-Services bieten jeweils eine eigene REST-API an, die mit Hilfe des Web-Frontends konsumiert werden kann. Die in der Anwendung genutzten Technologien aus Tabelle 1 sind in Abbildung 1 noch einmal schematisch zugeordnet. 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

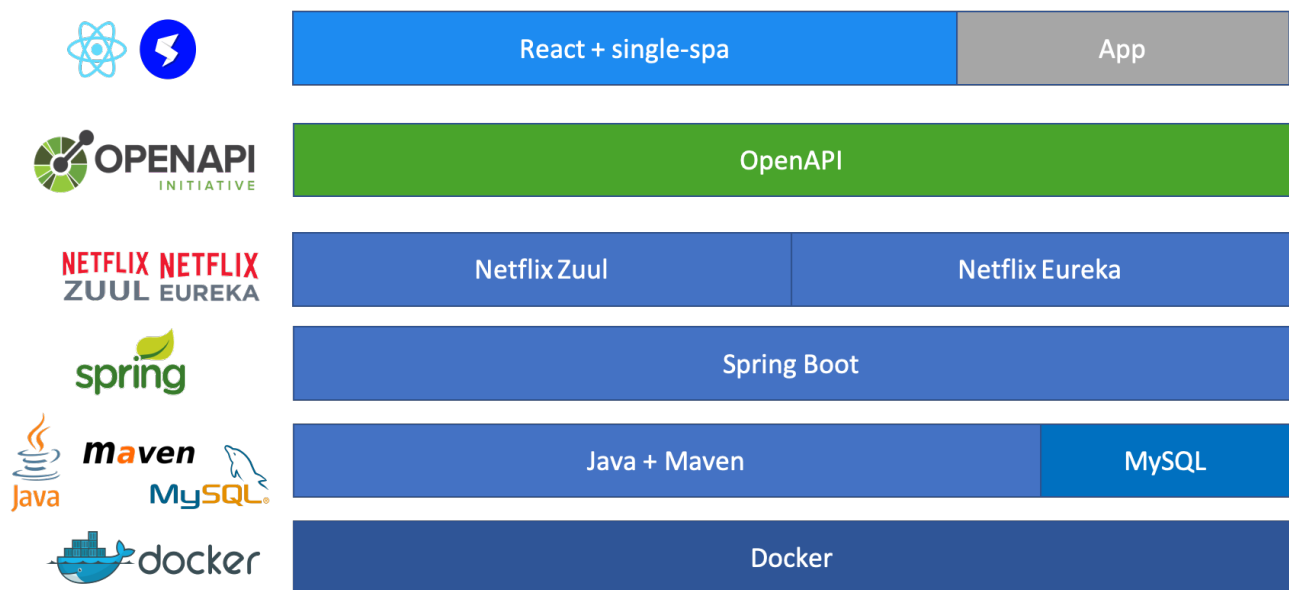


Abbildung 1: Technologie Stack

3.2 Architektur

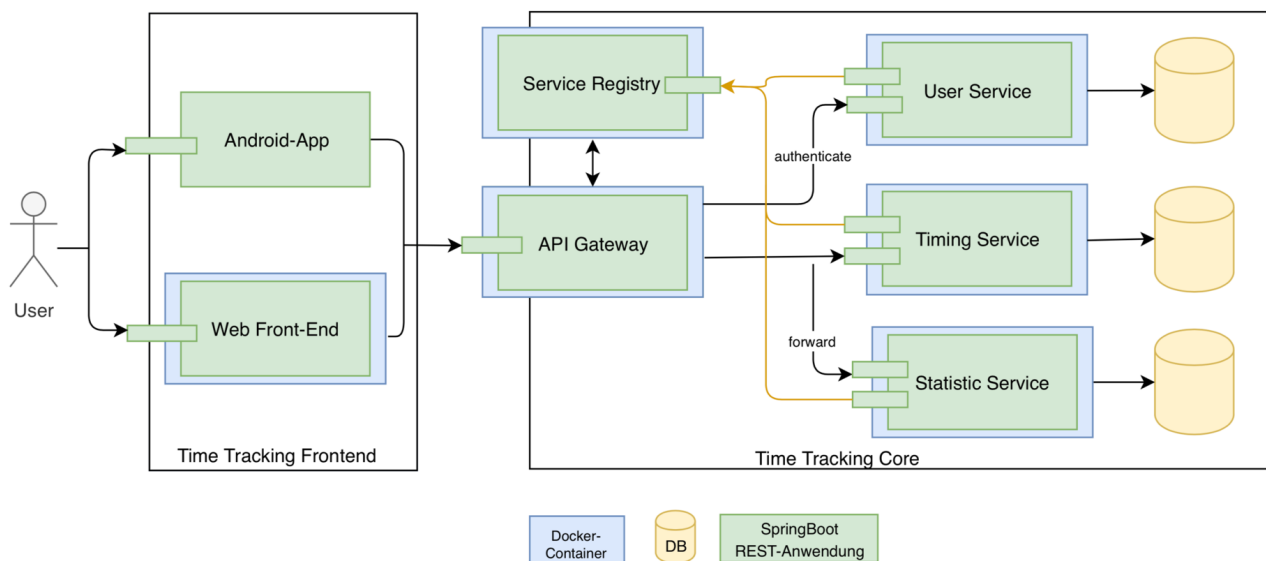


Abbildung 2: Architektur von TimeTracker

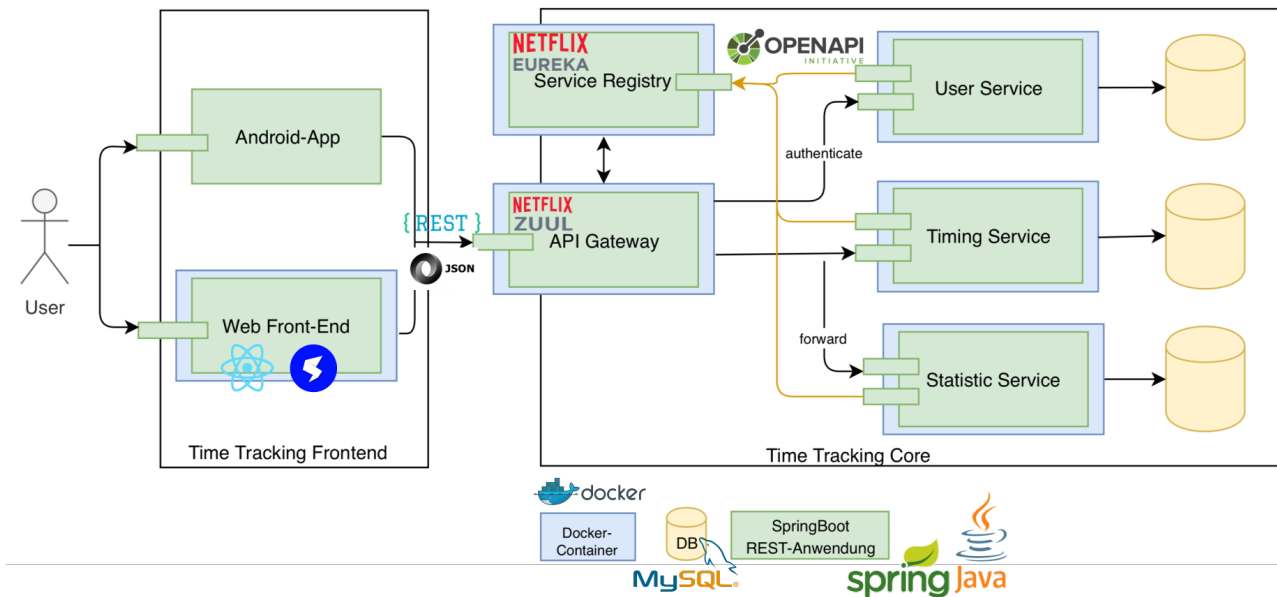


Abbildung 3: Architektur mit zugeordneten Technologien

Funktionalität 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 entsprechenden Service

Die Dockerisierung und Nutzung von docker compose sowie docker stack ermöglicht das Ausrollen kontinuierlicher Updates.

3.3 Schnittstellen

Die API wurde mit Hilfe von OpenAPI und Swagger definiert und entwickelt. Mit Hilfe eines Swagger-Plugins für Maven wurden automatisch Java-Interfaces generiert, welche dann von Spring-REST-Controllern implementiert werden konnten. Eine Auflistung aller Methoden ist über die folgenden Links zu erreichen. Zusätzlich befindet sich im Anhang (Seite 16) 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 sichere und authentifizierte Kommunikation mit den Micro-Services wurde mit Hilfe von **JSON Web Token (JWT)** und HTTPS umgesetzt.

Bei erfolgreichem Login erhält der Benutzer ein Token, mit dem er sich bei den Diensten authentifizieren kann. Das Token wird im localStorage des Browsers gespeichert und bei jedem Request im Header mitgeführt. Dank Spring ist jedem Dienst dabei frei überlassen, wie das Token interpretiert wird, da ein entsprechendes Interface für jeden Service frei implementiert werden kann. So kann

zum Beispiel Autorisierung auf Basis von Rollen realisiert werden. In unserer Anwendung findet dieses Feature jedoch keine Bedeutung, so dass jeder Micro-Service für sich nur die Validität des Tokens überprüft.

Des Weiteren wird die Sicherheit der Verbindung mittels der Transportverschlüsselung **HTTPS** gewährleistet. Dazu wird im Produktionsbetrieb der gesamten Architektur ein nginx-Server vorgeschoben, der HTTPS-Requests entgegen nimmt, TLS-Termination vornimmt und die dann entschlüsselte Kommunikation an das API-Gateway weiterleitet. Die Kommunikation innerhalb des V-Servers, also zwischen den Micro-Services selbst, ist folglich weiterhin unverschlüsselt (HTTP). Wir haben uns bewusst dazu entschieden, da Verteilung von Zertifikaten sowie das ständige Ver- und Entschlüsseln der Kommunikation die Performance sichtlich beeinflusste.

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 <https://iamtrent.de>

Lokale Installationen sind zu erreichen unter `localhost`


4.1 Build

Um die Anwendung zu bauen und lokal hochzufahren, wechseln Sie bitte in das Wurzelverzeichnis und führen Sie folgende Kommandos aus:

```
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.

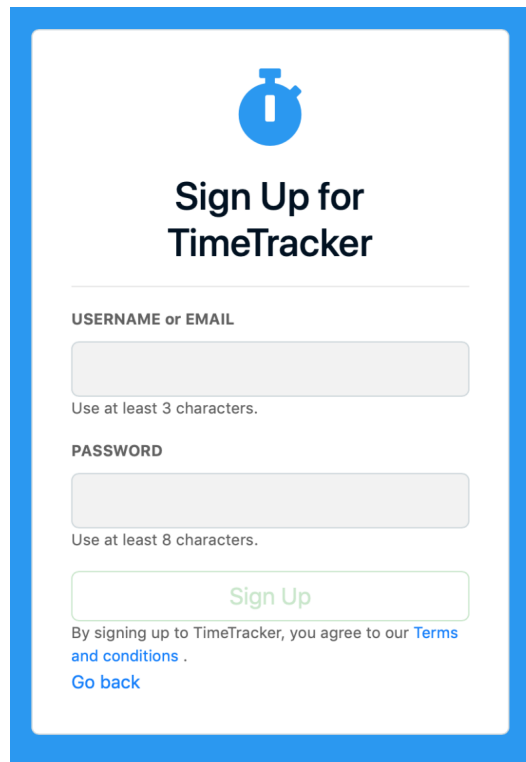
The image shows a registration form for 'TimeTracker' enclosed in a blue border. At the top center is a blue circular icon with a white stopwatch symbol. Below the icon, the text 'Sign Up for TimeTracker' is displayed in a bold, black font. A horizontal line separates the title from the input fields.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

Abbildung 5: Registrieren

Um sich zu registrieren, wähle einen Usernamen sowie ein Passwort und klicke dann auf „Sign Up“. Beachte bitte, dass das Passwort mindestens 8 Zeichen lang sein muss. Bitte merke Dir deine ausgewählten Credentials, da bislang noch keine Registrierungsbestätigung per Mail versendet wird. Nachdem Du dich erfolgreich registriert haben, wirst Du wieder auf unsere Startseite (Abb. 4) geleitet, wo Du dich ab sofort mit Deinen Nutzerdaten anmelden können.

Wenn Du einmal angemeldet bist, kannst Du dich jederzeit 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 alltägliche Aktionen, für die Du die Zeit trackst, die Du in sie investierst. Zum Beispiel Lernen, Reisen, Einkaufen, Sport treiben und so weiter. Füge einfach eine neue Aktivität hinzu, in dem Du auf *+ New Activity* klickst

(Abb. 6).

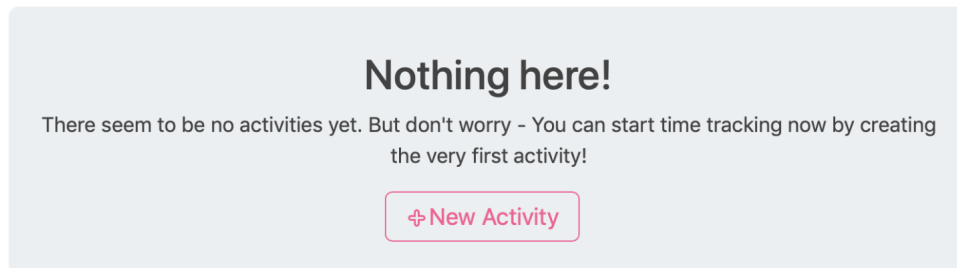


Abbildung 6: Keine Aktivitäten angelegt

A screenshot of a mobile app interface for creating a new activity. At the top is a blue circular icon with a white clock face. Below it, the title 'Create a new Activity' is centered in a bold, black, sans-serif font. The form consists of three main sections: 1. 'NAME OF NEW ACTIVITY' with a text input field containing 'Vorlesung SCC' and a small note below it: 'Use at least 3 characters.' 2. 'DESCRIPTION' with a text input field containing 'Vorlesung Service and Cloud Computing, Dienstag 3. DS' and a note below it: 'Describe shortly what the activity is all about. Use at least 10 characters.' 3. A dropdown menu labeled 'Studies' with 'Studies' selected and a small downward arrow icon. Below the dropdown is a note: 'Tag your activity.' At the bottom of the form are two buttons: a green 'Create activity' button and a red 'Cancel' button.

Abbildung 7: Aktivität anlegen

Nun kannst Du die Aktivität genauer beschreiben (Abb. 7. Gib ihr einen Namen und eine Beschreibung, damit Du später weißt, wofür Du sie angelegt hast. Der Name darf keine Umlaute wie „Ä, Ö, Ü“ enthalten und die Beschreibung

muss mindestens 10 Zeichen lang sein. Danach verbindest Du deine Aktivität mit einem Tag. Tags helfen Dir dabei, deine Activities zu kategorisieren. 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.

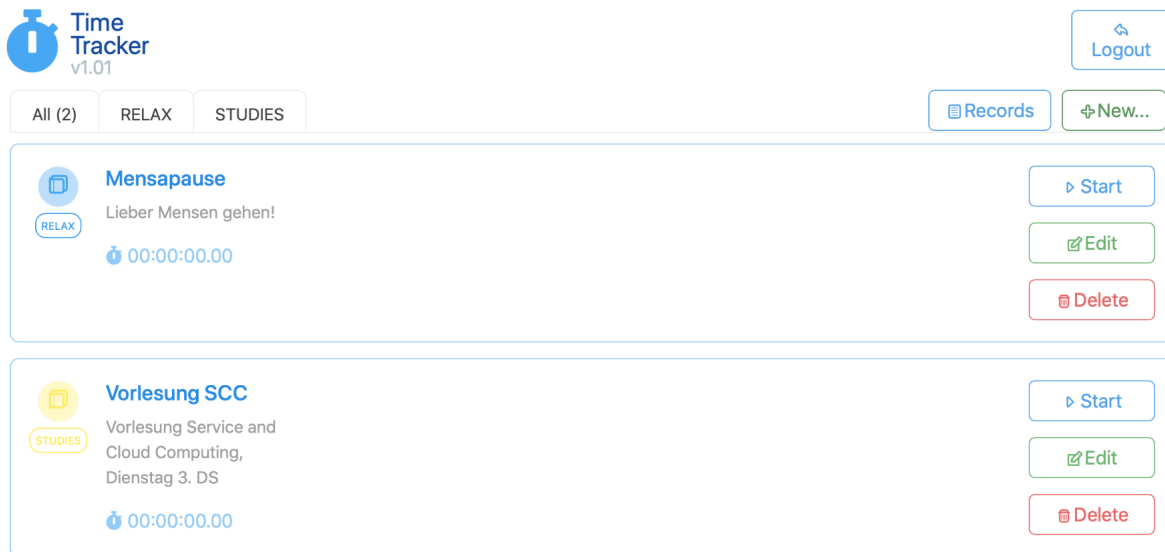


Abbildung 8: Übersicht: Aktivitäten

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, um investierte Zeit zu tracken. Summiert ergeben sie die gesamte Zeit, die du für eine Aktivität aufbringst. Nach einem Klick auf *Start* wirst Du sehen, dass die Stopuhr blinkt und die Zeit läuft. Mit einem Klick auf den selben Button stoppst du das Tracking deiner Aktivität und beendest den Record. Die Zeit des Records wird dir nun angezeigt. 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 kannst Du Namen, Beschreibung und Tag deiner Activities anpassen.

Wenn du auf einem Desktop TimeTracker nutzt, kannst Du über die Reiter

oben die Ansicht nach Tags filtern.

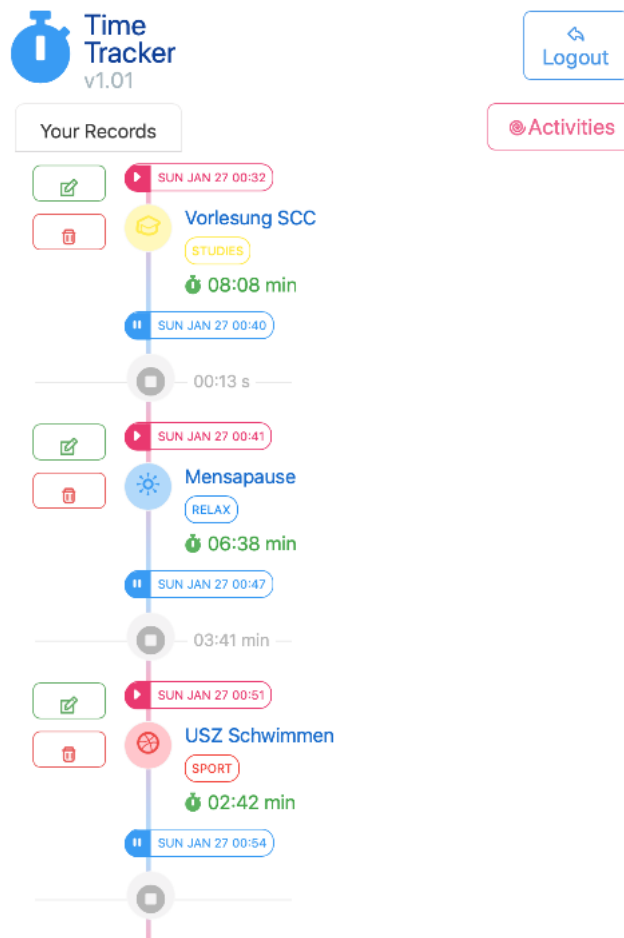


Abbildung 9: Aufgenommene Records

Wenn du *Records* anwählst, kommst Du zu einer Übersicht all deiner Zeitaufnahmen (Abb. 9). Hier kannst Du genau sehen, wie viel Zeit Du für welche Aktivität investiert hast. Außerdem kannst Du von hier aus auch Records löschen.

5 Feedback und Kritik am Praktikum

Allgemein: Insgesamt machte es viel Spaß, diese Anwendung zu entwickeln. Ein gutes Klima im Team trug maßgeblich dazu bei. Aufgrund des unterschiedlicher praktischer Erfahrung fand ein reger Wissensaustausch statt. Es entwickelte sich mit der Zeit eine Art Mentoring zwischen beiden Teilnehmern. Bei der Umsetzung wurde sehr deutlich, wie das Praktikum die Vorlesungsinhalte vertieft und in direkten praktischen Bezug dazu tritt. Das in der Vorlesung erlangte Wissen konnte anschaulich umgesetzt werden. Die im Rahmen des Praktikums entstandene Anwendung wird weiterentwickelt und genutzt werden.

Spezifikation der Anforderungen: Durch die in manchen Teilen unspezifische Aufgabenstellung ließ sich schwer abschätzen, welche Anforderungen genau erfüllt werden müssen. Positiv daran ist die Möglichkeit der freien Auswahl von Vorgehensweise, Umsetzung und Tools. Die Zwischenpräsentation und das Feedback darauf hat jedoch maßgeblich zum besseren Verständnis der Aufgabenstellung und fokussierterem Arbeiten geführt.

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: 1.01

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)

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://helloeverb.com>
Contact Info: hello@helloeverb.com
Version: 1.01
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}](#)
- [DELETE /activities/records/{recordId}](#)
- [GET /activities/](#)
- [GET /activities/{activityId}](#)
- [GET /activities/{activityId}/records](#)
- [GET /activities/stats](#)
- [GET /activities/records](#)
- [PUT /activities/](#)

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](#)

DELETE /activities/records/{recordId}

[Up](#)

(deleteActivityRecord)

Deletes a record with the given id,

Path parameters

recordId (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](#)

[Up](#)

GET /activities/

(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](#)

[Up](#)

GET /activities/{activityId}

(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](#)

[Up](#)

GET /activities/{activityId}/records

(getActivityRecords)

Returns all records for an activity

Path parameters

activityId (required)

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

state (optional)

Query Parameter — The state of the record

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/

(updateActivity)

Updates the activity meta data.

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](#)

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

state (optional)

[String](#) The state of the activity

duration (optional)

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

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: 1.01-oas3

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\[UiService\]*](#)