Hilfeseite

Hier befindet sich eine Übersicht und Erklärungen zu den verschiedenen Funktionen der Anwendung.

Folgende Seiten stehen zur Verfügung:

- Index.html – Übersicht über alle Sportler des Trainers.
- SportlerErstellen.html – Erstellung neuer Sportler.
- SportlerErstellenBulk.html – Erstellung neuer Sportler per CSV-Datei.
- SportlerBearbeiten.html – Bearbeitung der Sportlerdaten.
- LeistungErstellen.html – Erfassung neuer Leistungen.
- LeistungErstellenBulk.html – Erfassung neuer Leistungen mehrerer Sportler per CSV.
- Leistungsuebersicht.html – Zeigt die Leistungen eines Sportlers.
- SchwimmnachweisErstellen.html – Verwaltung von Schwimmnachweisen.
- RegelnUebersicht.html – Übersicht und Aktualisierung der Regeln.
- RegelnAktualisierung.html – Regeln per CSV-Datei aktualisieren.
- Anmeldung.html – Anmeldungsseite.
- Registrierung.html – Registrierungsseite.
- BackendDown.html – Anzeige bei Serverproblemen.
- Dokumentation des Backends – Überblick über die ComPeteHub API.

Startseite *(Index.html)*

Diese Seite dient als Startpunkt und zeigt eine Übersicht aller dem Trainer zugewiesenen Sportler.

Buttons und Funktionen:

- **Sortieren:** Dropdown-Menü, um die Sportler nach Vorname, Nachname oder Geburtstag zu sortieren.
- **Regeln:** Button, um die aktuellen Regeln einzusehen und zu aktualisieren.
- **Leistungsexport:** Button, um die Leistungen ausgewählter Sportler als CSV-Datei zu exportieren.
- **Sportler erstellen:** Button, um einen neuen Sportler manuell hinzuzufügen.
- **Sportler Massenerstellung:** Button, um mehrere Sportler per CSV-Datei hinzuzufügen.

Leistungsexport (CSV)

Mit dem Button **Leistungsexport** können die Leistungen ausgewählter Sportler als CSV-Datei exportiert werden.

- Nach Klick öffnet sich ein Modal, in dem alle Sportler aufgelistet sind.
- Sie können einzelne oder alle Sportler auswählen, deren Leistungen exportiert werden sollen.
- Nach Klick auf **Leistungen exportieren** wird eine CSV-Datei mit folgendem Format erstellt:

```
Nachname;Vorname;Geschlecht;Geburtsjahr;Geburtstag;Übung;Kategorie;Datum;Ergebnis;Punkte
```

Gültige Werte:

- **Geschlecht:** m, w, d
- **Geburtsjahr:** yyyy
- **Geburtstag/Datum:** dd.mm.yyyy
- **Ergebnis:** abhängig von der Übung:
  - Distanzen: Meter (m), Zentimeter (cm)

    - Punkte: Ganze Zahlen
    - Bestanden: 0 = nein, 1 = ja
    - Zeit: MM:SS, M:SS, SS, S
- **Punkte:** 0-3

Das Modal zeigt eine Tabelle mit allen Sportlern, Checkboxen zur Auswahl und einen Export-Button. Nach dem Export wird die CSV-Datei automatisch heruntergeladen.

Sportler erstellen *(SportlerErstellen.html)*

Hier können neue Sportler manuell hinzugefügt werden.

Inputfelder und Funktionen:

- **Vorname:** Eingabefeld, um den Vornamen des neuen Sportlers einzugeben.
- **Nachname:** Eingabefeld, um den Nachnamen des neuen Sportlers einzugeben.
- **Email:** Eingabefeld, um die E-Mail-Adresse des neuen Sportlers einzugeben.
- **Geburtstag:** Eingabefeld, um das Geburtsdatum des neuen Sportlers einzugeben.
- **Geschlecht:** Radio-Buttons, um das Geschlecht des neuen Sportlers auszuwählen.

Buttons:

- **Speichern:** Speichert den neuen Sportler.
- **Abbrechen:** Bricht die Erstellung ab und kehrt zur Startseite zurück.

Sportler Massenerstellung *(SportlerErstellenBulk.html)*

Auf dieser Seite können mehrere Sportler gleichzeitig über eine CSV-Datei erstellt werden.

Inputfelder und Funktionen:

- **CSV-Datei:** Eingabefeld, um eine CSV-Datei mit den Sportlerdaten hochzuladen.
  Das Format der Datei muss den Vorgaben entsprechen:
  `Vorname;Nachname;Email;Geburtsdatum;Geschlecht`

Gültige Werte:

- **Geschlecht:** m, w, d
- **Geburtsdatum:** dd.mm.yyyy
- **Email:** gültige E-Mail-Adresse

Buttons:

- **Speichern:** Lädt die CSV-Datei hoch und erstellt die Sportler in der Datenbank.
- **Abbrechen:** Bricht den Vorgang ab und kehrt zur Startseite zurück.

Nach dem Hochladen wird eine Rückmeldung angezeigt, ob alle Sportler erfolgreich eingetragen wurden oder welche Einträge Fehler enthielten.

Sportler bearbeiten *(SportlerBearbeiten.html)*

Hier können die Daten eines bestehenden Sportlers bearbeitet werden.

Inputfelder und Funktionen:

- **Vorname:** Eingabefeld, um den Vornamen des Sportlers zu ändern.
- **Nachname:** Eingabefeld, um den Nachnamen des Sportlers zu ändern.
- **Email:** Eingabefeld, um die E-Mail-Adresse des Sportlers zu ändern.
- **Geburtstag:** Eingabefeld, um das Geburtsdatum des Sportlers zu ändern.
- **Geschlecht:** Radio-Buttons, um das Geschlecht des Sportlers zu ändern.

Buttons:

- **Speichern:** Speichert die Änderungen.
- **Abbrechen:** Bricht die Bearbeitung ab und kehrt zur Startseite zurück.

Leistung erfassen *(LeistungErstellen.html)*

Auf dieser Seite können neue Leistungen für einen Sportler erfasst werden.

Inputfelder und Funktionen:

- **Kategorie:** Dropdown-Menü, um die Kategorie der Leistung auszuwählen.
- **Datum:** Eingabefeld, um das Datum der Leistung festzulegen.
- **Übung:** Dropdown-Menü, um die spezifische Übung auszuwählen.
- **Gemessener Wert:** Eingabefeld, um den Wert der Leistung (z. B. Zeit, Punkte) einzugeben.

Buttons:

- **Speichern:** Speichert die eingegebene Leistung.
- **Abbrechen:** Bricht die Eingabe ab und kehrt zur Leistungsübersicht zurück.

Leistungen Massenerstellung *(LeistungErstellenBulk.html)*

Auf dieser Seite können Leistungen für mehrere Sportler gleichzeitig über eine CSV-Datei erstellt werden.

Inputfelder und Funktionen:

- **CSV-Datei:** Eingabefeld, um eine CSV-Datei mit den Leistungsdaten hochzuladen.
  Das Format der Datei muss den Vorgaben entsprechen:
  `Nachname;Vorname;Geschlecht;Geburtsjahr;Geburtstag;Übung;Kategorie;Datum;Ergebnis;Punkte`

Gültige Werte:

- **Geschlecht:** m, w, d
- **Geburtstag/Datum:** dd.mm.yyyy
- **Ergebnis:** abhängig von der Übung:
  - Distanzen: Meter (m), Zentimeter (cm)

    - Punkte: Ganze Zahlen
    - Bestanden: 0 = nein, 1 = ja
    - Zeit: MM:SS, M:SS, SS, S
- **Punkte:** 0-3

Buttons:

- **Speichern:** Lädt die CSV-Datei hoch und erstellt die Leistungen in der Datenbank.
- **Abbrechen:** Bricht den Vorgang ab und kehrt zur Startseite zurück.

Nach dem Hochladen wird eine Rückmeldung angezeigt, ob alle Leistungen erfolgreich eingetragen wurden oder welche Einträge Fehler enthielten.

Leistungsübersicht *(Leistungsuebersicht.html)*

Hier sind die Leistungen eines Sportlers in den Kategorien **Ausdauer, Schnelligkeit, Kraft** und **Koordination** einsehbar.

Buttons und Funktionen:

- **Leistungen eintragen:** Button, um eine neue Leistung für den Sportler zu erfassen.
- **Anzeigen ab:** Eingabefeld, um das Startdatum für die angezeigten Leistungen festzulegen.
- **Schwimmnachweis eintragen:** Button, um einen neuen Schwimmnachweis hochzuladen.
- **Schwimmnachweise herunterladen:** Button, um alle Schwimmnachweise des Sportlers als ZIP-Datei herunterzuladen.

Kategorien und Tabellen:

- Die Seite zeigt für jede Kategorie (**Ausdauer, Schnelligkeit, Kraft, Koordination**) einen eigenen Abschnitt an.
- Jede Kategorie kann durch Klick auf die Überschrift aufgeklappt oder zugeklappt werden, um die zugehörigen Leistungen anzuzeigen oder auszublenden.
- In den Tabellen werden folgende Informationen zu den einzelnen Leistungen angezeigt:
    - **Übung:** Name der absolvierten Übung
    - **Datum:** Datum der Leistung
    - **Wert:** Erzieltes Ergebnis (z. B. Zeit, Punkte, Distanz)
    - **Medaille:** Erreichte Medaille (Gold, Silber, Bronze oder kein Erfolg)
- In der Kopfzeile jeder Kategorie werden zusätzlich der letzte Wert und die zuletzt erreichte Medaille angezeigt.

Hinweis: Durch das Auf- und Zuklappen der Kategorien behalten Sie die Übersicht und können gezielt die gewünschten Leistungsdaten einsehen.

Schwimmnachweis *(SchwimmnachweisErstellen.html)*

Hier können Schwimmnachweise für Sportler hochgeladen und verwaltet werden.

Inputfelder und Funktionen:

- **Nachweis hochladen:** Eingabefeld, um eine Datei mit dem Schwimmnachweis hochzuladen.
- **Datum:** Eingabefeld, um das Datum des Nachweises festzulegen.

Buttons:

- **Hochladen:** Lädt den Schwimmnachweis hoch und speichert ihn in der Datenbank.

- **Abbrechen:** Bricht den Upload ab und kehrt zur vorherigen Seite zurück.

Regeln Übersicht *(RegelnUebersicht.html)*

Diese Seite bietet eine Übersicht über die aktuellen Regeln.

Inputfelder und Funktionen:

- **Kategorie:** Dropdown-Menü, um die Kategorie der Regeln auszuwählen.
- **Übung:** Dropdown-Menü, um die spezifische Übung auszuwählen.
- **Jahr:** Eingabefeld, um das Jahr der Regeln festzulegen.

Buttons:

- **Regeln aktualisieren:** Weiterleitung zur Seite *RegelnAktualisierung.html*, um Regeln per CSV-Datei zu aktualisieren.

Regeln aktualisieren *(RegelnAktualisierung.html)*

Auf dieser Seite können die Regeln des Systems per CSV-Datei aktualisiert werden.

Inputfelder und Funktionen:

- **CSV-Datei:** Eingabefeld, um eine CSV-Datei mit den neuen oder aktualisierten Regeln hochzuladen. Das Format der Datei muss den Vorgaben entsprechen: `Regelwerk Jahr; Disziplin Name; Übung Name; Einheit; Geschlecht; Altersgruppe Start; Altersgruppe Ende; Bronze; Silber; Gold; Beschreibung`

Hinweise zum Format:

- **Einheit:** meter, centimeter, second, minute, bool, point
- **Geschlecht:** m, w, d
- **Bronze/Silber/Gold:** abhängig von der Übung (z. B. Distanzen in cm, Punkte als ganze Zahlen, Bestanden als 0/1, Zeiten in ms)

Buttons:

- **Aktualisieren:** Lädt die CSV-Datei hoch und aktualisiert die Regeln im System.
- **Abbrechen:** Bricht den Vorgang ab und kehrt zur Regelübersicht zurück.

Nach dem Hochladen wird eine Rückmeldung angezeigt, ob die Regeln erfolgreich aktualisiert wurden oder ob Fehler aufgetreten sind.

Anmeldung *(Anmeldung.html)*

Auf dieser Seite kann man sich mit E-Mail und Passwort anmelden.

Inputfelder und Funktionen:

- **E-Mail:** Eingabefeld, um die E-Mail-Adresse einzugeben.
- **Passwort:** Eingabefeld, um das Passwort einzugeben.
- **Angemeldet bleiben:** Checkbox, um die Sitzung über einen längeren Zeitraum aktiv zu halten.

Buttons:

- **Anmelden:** Meldet den Benutzer an und leitet zur Startseite weiter.

Registrierung *(Registrierung.html)*

Auf dieser Seite kann man sich mit E-Mail und Passwort registrieren.

Inputfelder und Funktionen:

- **E-Mail:** Eingabefeld, um die E-Mail-Adresse einzugeben.
- **Passwort:** Eingabefeld, um ein sicheres Passwort einzugeben.
- **Passwort bestätigen:** Eingabefeld, um das Passwort erneut einzugeben.

Buttons:

- **Registrieren:** Erstellt ein neues Benutzerkonto und leitet zur Anmeldeseite weiter.

Server nicht erreichbar *(BackendDown.html)*

Diese Seite wird angezeigt, wenn der Server nicht erreichbar ist.

Funktionen:

- **Warnmeldung:** Zeigt eine Warnung an, dass der Server derzeit nicht erreichbar ist.
- **Automatische Weiterleitung:** Falls der Server wieder verfügbar ist, wird der Benutzer automatisch zur Startseite weitergeleitet.

Stand: 21.05.2025

Swagger
Supported by SMARTBEAR

/docs/swagger.json Explore

ComPeteHub API 1.0

[Base URL: localhost:8080]
/docs/swagger.json

This API allows managing athletes, disciplines and performances in the ComPeteHub system.

Team Reissdorf - Website
Send email to Team Reissdorf
MIT License

Schemes
HTTP

Authorize

Athlete Management

POST /v1/athlete/bulk-create Bulk creates new athletes from csv file

Upload a CSV file to create multiple athlete profiles. If an athlete already exists, the process will continue, and the response will indicate which athletes already exist.

Parameters Try it out

Name	Description
Athletes * required file *(formData)*	CSV file containing details of multiple athletes to create profiles Datei auswählen Keine ausgewählt
Authorization string *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
201	Creation successful

Example Value | Model


```
{
  "already_existing_athletes": [
    {
      "birth_date": "string",
      "createdAt": "string",
      "deletedAt": "string",
      "email": "string",
      "first_name": "string",
```

Code	Description


```
      "id": 0,
      "last_name": "string",
      "sex": "string",
      "trainer_email": "string",
      "updatedAt": "string"
    }
  ],
  "message": "Creation successful"
}
```


400 Invalid request body

Example Value | Model


```
{
  "error": "<task> failed"
}
```


401 The token is invalid

Example Value | Model


```
{
  "error": "<task> failed"
}
```


409 All athletes already exist; none have been created

Example Value | Model


```
{
  "already_existing_athletes": [
    {
      "birth_date": "string",
      "createdAt": "string",
      "deletedAt": "string",
      "email": "string",
      "first_name": "string",
      "id": 0,
      "last_name": "string",
      "sex": "string",
      "trainer_email": "string",
      "updatedAt": "string"
    }
  ],
  "message": "Creation successful"
}
```


500 Internal server error

Example Value | Model


```
{
  "error": "<task> failed"
}
```


POST `/v1/athlete/create` Creates a new athlete profile

Creates a new athlete profile with the given data. Duplicate email and birthdate combinations are not allowed.

Parameters Try it out

Name	Description
Athlete * required `object` *(body)*	Details of an athlete to create a profile **Example Value** | Model

Name	Description

```
{
  "birth_date": "YYYY-MM-DD",
  "email": "bob.alice@example.com",
  "first_name": "Bob",
  "last_name": "Alice",
  "sex": "<m|f|d>"
}
```

Parameter content type

application/json

Authorization
string
(header)

Access JWT is sent in the Authorization header or set as a http-only cookie

Authorization

Responses

Response content type: application/json

Code	Description

201 — Creation successful

Example Value | Model

```
{
  "message": "<task> successful"
}
```

400 — Invalid request body

Example Value | Model

```
{
  "error": "<task> failed"
}
```

401 — The token is invalid

Example Value | Model

```
{
  "error": "<task> failed"
}
```

409 — Athlete already exists

Example Value | Model

```
{
  "error": "<task> failed"
}
```

500 — Internal server error

Example Value | Model

```
{
  "error": "<task> failed"
}
```

DELETE `/v1/athlete/delete/{AthleteId}` Deletes the given athlete profile

Deletes the given athlete profile.

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	Delete the given athlete AthleteId
Authorization `string` *(path)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Deletion successful **Example Value** | Model `{ "message": "<task> successful" }`
400	Invalid request parameter **Example Value** | Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** | Model `{ "error": "<task> failed" }`
404	Athlete could not be found for this trainer **Example Value** | Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** | Model `{ "error": "<task> failed" }`

PUT `/v1/athlete/edit` Edits an existing athlete profile

Edits an existing athlete profile with the given details. Duplicate email and birthdate combinations are not allowed.

Parameters Try it out

Name	Description
Athlete * required `object` *(body)*	Edited details of an athlete **Example Value** | Model `{ "athlete_id": 1, "birth_date": "YYYY-MM-DD", "email": "bob.alice@example.com", "first_name": "Bob", "last_name": "Alice", "sex": "<m|f|d>", "swim_cert": true }` **Parameter content type** application/json
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Edited successful **Example Value** | Model `{ "message": "<task> successful" }`
400	Invalid request body **Example Value** | Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** | Model `{ "error": "<task> failed" }`
404	Athlete could not be found for this trainer **Example Value** | Model

Code	Description

```
{
  "error": "<task> failed"
}
```

409 Athlete already exists

Example Value | Model

```
{
  "error": "<task> failed"
}
```

500 Internal server error

Example Value | Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/athlete/get-all` Returns all athlete profiles and swim certificates

All athlete profiles and swim certificates of the given trainer are returned

Parameters Try it out

Name	Description
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description

200 Request successful

Example Value | Model

```
{
  "athletes": [
    {
      "athlete_id": 1,
      "birth_date": "YYYY-MM-DD",
      "email": "bob.alice@example.com",
      "first_name": "Bob",
      "last_name": "Alice",
      "sex": "<m|f|d>",
      "swim_cert": true
    }
  ],
  "message": "Request successful"
}
```

401 The token is invalid

Example Value | Model

Code	Description

```
{
  "error": "<task> failed"
}
```

500 Internal server error

Example Value | Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/athlete/get/{AthleteId}` Returns one athlete profile

One athlete profile with given id and of the given trainer gets returned

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	Get the athlete with the given id AthleteId
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description

200 Request successful

Example Value | Model

```
{
  "athletes": [
    {
      "athlete_id": 1,
      "birth_date": "YYYY-MM-DD",
      "email": "bob.alice@example.com",
      "first_name": "Bob",
      "last_name": "Alice",
      "sex": "<m|f|d>",
      "swim_cert": true
    }
  ],
  "message": "Request successful"
}
```

401 The token is invalid

Example Value | Model

```
{
  "error": "<task> failed"
}
```

Code	Description
404	Athlete not found **Example Value** | Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** | Model `{ "error": "<task> failed" }`

Settings

POST `/v1/backendSettings/change-log-level` ChangeLogLevel changes the log level of the server

Changes the log level of the server to the specified level 0-4 (Debug-Fatal).

Parameters Try it out

Name	Description
Authorization `string` *(header)*	Settings access JWT is sent in the Authorization header or set as a http-only cookie Authorization
Log-level * required `object` *(body)*	Log level **Example Value** | Model `{ "log_level": 1 }` **Parameter content type** application/json

Responses Response content type application/json

Code	Description
200	Log level changed **Example Value** | Model `{ "message": "<task> successful" }`
400	Invalid request body

Code	Description

Example Value | Model

```
{
  "error": "<task> failed"
}
```

401 The token is invalid

Example Value | Model

```
{
  "error": "<task> failed"
}
```

500 Internal server error

Example Value | Model

```
{
  "error": "<task> failed"
}
```

HealthCheck

GET `/v1/coffee` Returns a 418 I'm a teapot response

Simple endpoint to return a 418 I'm a teapot response.

Parameters Try it out

No parameters

Responses Response content type application/json

Code	Description

418 Error: I'm a teapot

Example Value | Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/ping` Returns a pong response

Simple ping-pong endpoint to check if the server is running properly.

Parameters Try it out

No parameters

Responses Response content type: application/json

Code	Description
200	Pong response

Example Value | Model

```
{
  "message": "<task> successful"
}
```

Discipline Management

GET `/v1/discipline/get-all` Returns all discipline names

All discipline names are returned from the database

Parameters Try it out

Name	Description
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type: application/json

Code	Description
200	Request successful

Example Value | Model

```
{
  "discipline_names": [
    "Discipline name"
  ],
  "message": "Request successful"
}
```

Code	Description
401	The token is invalid

Example Value | Model

```
{
  "error": "<task> failed"
}
```

Code	Description
500	Internal server error

Example Value | Model

Code	Description

```
{
  "error": "<task> failed"
}
```

Exercise Management

GET `/v1/exercise/get/{DisciplineName}` Returns the exercises

All exercises of the given discipline will be returned. When the athlete id is given, the age specific description will be returned with the exercise.

Parameters Try it out

Name	Description
DisciplineName * required `string` *(path)*	Get the exercises with the given discipline name DisciplineName
athlete-id `integer` *(query)*	Get the exercise_specifics for the given athletes age and sex athlete-id
performance-date `string` *(query)*	Date in YYYY-MM-DD format to get the exercises according to the ruleset of the given year performance-date
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Request successful **Example Value** | Model

```
{
  "exercises": [
    {
      "age_specifics": "Age specific description",
      "discipline_name": "Discipline",
      "exercise_id": 1,
      "name": "Exercise",
      "unit": "minutes"
    }
  ],
  "message": "Request successful"
}
```

401	The token is invalid **Example Value** | Model

Code	Description

```
{
  "error": "<task> failed"
}
```

404 Discipline, athlete or ruleset year does not exist

Example Value | Model

```
{
  "error": "<task> failed"
}
```

500 Internal server error

Example Value | Model

```
{
  "error": "<task> failed"
}
```

Performance Management

POST `/v1/performance/bulk-create` Bulk create performance entries from CSV

Upload a .csv file to bulk-create performance entries.

Parameters Try it out

Name	Description
Performances * required `file` *(formData)*	CSV file; must have extension .csv; columns: lastName;firstName;gender;birthYear;birthDate;exercise;category;date;result;points Datei auswählen Keine ausgewählt
Authorization `string` *(header)*	Bearer JWT token Authorization

Responses Response content type application/json

Code	Description

201 Bulk creation successful

Example Value | Model

```
{
  "failed_entries": [
    {
      "reason": "Invalid athlete ID",
      "row": 2
    }
  ],
  "message": "Bulk Creation successful"
}
```

Code	Description
400	Bad request: missing file / invalid CSV / wrong extension **Example Value** | Model `{ "error": "<task> failed" }`
401	Unauthorized: invalid or missing token **Example Value** | Model `{ "error": "<task> failed" }`
409	Conflict: all entries failed, none created **Example Value** | Model `{ "error": "<task> failed" }`
500	Internal server error (DB failure or file read error) **Example Value** | Model `{ "error": "<task> failed" }`

POST `/v1/performance/create` Creates a new performance entry

Creates a new performance entry with the given data. Maximum 3 entries/discipline/athlete/day are allowed (performance limit).

Parameters Try it out

Name	Description
Performance * required `object` *(body)*	Details of a performance (valid units are: <millisecond, centimeter, point, bool>) **Example Value** | Model `{ "athlete_id": 1, "date": "YYYY-MM-DD", "exercise_id": 1, "points": 1 }` **Parameter content type** application/json
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses

Response content type application/json

Code	Description
201	Creation successful

Example Value | Model

```
{
  "medal_status": "Medal status",
  "message": "Creation successful"
}
```

Code	Description
400	Invalid request body

Example Value | Model

```
{
  "error": "<task> failed"
}
```

Code	Description
401	The token is invalid

Example Value | Model

```
{
  "error": "<task> failed"
}
```

Code	Description
404	Athlete or exercise does not exist

Example Value | Model

```
{
  "error": "<task> failed"
}
```

Code	Description
409	Performance limit reached

Example Value | Model

```
{
  "error": "<task> failed"
}
```

Code	Description
500	Internal server error

Example Value | Model

```
{
  "error": "<task> failed"
}
```

PUT `/v1/performance/edit` Edits an existing performance entry

Edits an existing performance entry with the given details.

Parameters Try it out

Name	Description
Performance * required `object`	Edited details of a performance entry

Name	Description
(body)	**Example Value** | Model

```
{
  "date": "YYYY-MM-DD",
  "exercise_id": 1,
  "performance_id": 1,
  "points": 1
}
```

Parameter content type

application/json

Authorization
`string`
(header)

Access JWT is sent in the Authorization header or set as a http-only cookie

Authorization

Responses — **Response content type** application/json

Code	Description
200	Edited successful **Example Value** | Model

```
{
  "message": "<task> successful"
}
```

400	Invalid request body **Example Value** | Model

```
{
  "error": "<task> failed"
}
```

401	The token is invalid **Example Value** | Model

```
{
  "error": "<task> failed"
}
```

404	Performance entry or goals not found **Example Value** | Model

```
{
  "error": "<task> failed"
}
```

500	Internal server error **Example Value** | Model

```
{
  "error": "<task> failed"
}
```

POST `/v1/performance/export` Exports all performance entries of the specified athletes as a csv file

Exports all performance entries of the specified athletes as a csv file

Parameters Try it out

Name	Description
json * required `object` *(body)*	JSON payload in the format: **Example Value** | Model `{ "athlete_ids": [ 1 ] }` **Parameter content type** application/json
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type text/csv

Code	Description
200	CSV file **Example Value** | Model `(no example available)`
400	Invalid request body **Example Value** | Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** | Model `{ "error": "<task> failed" }`
404	One or more athletes do not exist **Example Value** | Model `{ "error": "<task> failed" }`
500	Internal server error

Code	Description

Example Value | Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/performance/get-latest/{AthleteId}` Returns the latest performance entry

Returns the latest performance entry from the database

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	Get the latest performance entry of the given athlete_id AthleteId
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Request successful

Example Value | Model

```
{
  "message": "Request successful",
  "performance_entry": {
    "athlete_id": 1,
    "date": "YYYY-MM-DD",
    "exercise_id": 1,
    "medal": "gold",
    "performance_id": 1,
    "points": 1,
    "unit": "meter"
  }
}
```

401 The token is invalid

Example Value | Model

```
{
  "error": "<task> failed"
}
```

404 Athlete or performance entry not found

Example Value | Model

```
{
  "error": "<task> failed"
}
```

Code	Description
500	Internal server error

Example Value | Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/performance/get/{AthleteId}` Returns all performance entries of the given athlete

Returns all performance entries of the given athlete and can be filtered using the 'since' query parameter

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	Get all performance entries of the given athlete AthleteId
since `string` *(query)*	Date in YYYY-MM-DD format to get all entries since then (including the entries from that day) since
date `string` *(query)*	Date in YYYY-MM-DD format to get all entries from a specific day date
best `boolean` *(query)*	If true, only the latest best performance entries for each discipline will be returned --
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as an http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Request successful

Example Value | Model

```
{
  "message": "Request successful",
  "performance_entries": [
    {
      "athlete_id": 1,
      "date": "YYYY-MM-DD",
      "exercise_id": 1,
      "medal": "gold",
      "performance_id": 1,
      "points": 1,
      "unit": "meter"
    }
  ]
}
```

Code	Description
400	Date parameter is before the since-parameter **Example Value** | Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** | Model `{ "error": "<task> failed" }`
404	Athlete does not exist **Example Value** | Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** | Model `{ "error": "<task> failed" }`

Ruleset Management

POST `/v1/ruleset/create` Creates new ruleset entries from csv file

Upload a CSV file to create multiple ruleset entries. Needs to contain 11 columns.

Parameters Try it out

Name	Description
RulesetEntries * required `file` *(formData)*	CSV file containing details of the ruleset Datei auswählen Keine ausgewählt
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Creation successful **Example Value** | Model `{ "message": "<task> successful" }`
400	Invalid request body **Example Value** | Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** | Model `{ "error": "<task> failed" }`
409	All ruleset entries already exist; none have been created **Example Value** | Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** | Model `{ "error": "<task> failed" }`

GET `/v1/ruleset/get` Returns the rulesets

Retrieves the rulesets of a specific exercise with the given exercise_id and year.

Parameters Try it out

Name	Description
year * required `integer` *(query)*	Year of the ruleset year
exercise_id * required `integer` *(query)*	ID of the exercise exercise_id

Responses Response content type application/json

Code	Description
200	Request successful

Example Value | Model


```
{
  "message": "Request successful",
  "ruleset_goals": [
    {
      "bronze": 0,
      "createdAt": "string",
      "deletedAt": "string",
      "description": "string",
      "from_age": 0,
      "gold": 0,
      "id": 0,
      "rulesetId": 0,
      "sex": "string",
      "silver": 0,
      "to_age": 0,
      "updatedAt": "string"
    }
  ]
}
```


400 Invalid query parameters

Example Value | Model


```
{
  "error": "<task> failed"
}
```


404 Rulesets not found

Example Value | Model


```
{
  "error": "<task> failed"
}
```


500 Internal server error

Example Value | Model


```
{
  "error": "<task> failed"
}
```


Swim Certificate

POST `/v1/swimCertificate/create/{AthleteId}` Uploads a swim certificate for an athlete.

A Trainer can upload a swim certificate as a file for an specified athlete.

Parameters Try it out

Name	Description
file * required `file` *(formData)*	File to upload Datei auswählen Keine ausgewählt

Name	Description
AthleteId * required `integer` *(path)*	Get the latest performance entry of the given athlete_id AthleteId
Authorization `string` *(header)*	JWT Token Authorization

Responses Response content type: application/json

Code	Description
200	Upload successful **Example Value** | Model

```
{
  "message": "<task> successful"
}
```

400	Invalid request **Example Value** | Model

```
{
  "error": "<task> failed"
}
```

401	Unauthorized **Example Value** | Model

```
{
  "error": "<task> failed"
}
```

500	Internal server error **Example Value** | Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/swimCertificate/download-all/{AthleteId}` Download all swim certificates for an athlete

A trainer can download all swim certificates for a specified athlete. The result is a ZIP file containing all documents.

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	ID of the athlete AthleteId

Name	Description
Authorization `string` *(header)*	JWT token Authorization

Responses Response content type: application/zip

Code	Description
200	ZIP archive with all swim certificates **Example Value** | Model `(no example available)`
400	Invalid request **Example Value** | Model `{ "error": "<task> failed" }`
401	Unauthorized **Example Value** | Model `{ "error": "<task> failed" }`
404	Swim certificates not found or athlete not found **Example Value** | Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** | Model `{ "error": "<task> failed" }`

User Management

POST `/v1/user/login` Login to an existing user account

Logs the user into the system with the provided credentials and returns a refresh-JWT.

Parameters Try it out

Name	Description
User * required `object` *(body)*	The user's email address and password, along with a 'remember_me' field. If set to false or left empty, the refresh token cookie will not have a maxAge flag, causing the browser to automatically delete it when the session ends. **Example Value** | Model `{ "email": "bob.alice@example.com", "password": "<password>", "remember_me": true }` **Parameter content type** application/json

Responses Response content type: application/json

Code	Description
200	Login successful **Example Value** | Model `{ "access-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8", "refresh-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8" }`
400	Invalid request body **Example Value** | Model `{ "error": "<task> failed" }`
401	Wrong credentials **Example Value** | Model `{ "error": "<task> failed" }`
404	User not found **Example Value** | Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** | Model `{ "error": "<task> failed" }`

POST `/v1/user/logout` Logout of a user account

Logs the user out by clearing cookies from the client's browser.

Parameters Try it out

No parameters

Responses Response content type application/json

Code	Description
200	Logout successful **Example Value** | Model `{ "message": "<task> successful" }`
500	Internal server error **Example Value** | Model `{ "error": "<task> failed" }`

POST `/v1/user/register` Register a new user

Creates a new user account in the database with the provided details.

Parameters Try it out

Name	Description
User * required `object` *(body)*	The user's email address and password, along with a 'remember_me' field. If set to false or left empty, the refresh token cookie will not have a maxAge flag, causing the browser to automatically delete it when the session ends. **Example Value** | Model `{ "email": "bob.alice@example.com", "password": "<password>", "remember_me": true }` **Parameter content type** application/json

Responses

Response content type application/json

Code	Description

201	Registration successful
	Example Value | Model
	{
	"access-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8",
	"refresh-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8"
	}

400	Invalid request body
	Example Value | Model
	{
	"error": "<task> failed"
	}

409	User already exists
	Example Value | Model
	{
	"error": "<task> failed"
	}

500	Internal server error
	Example Value | Model
	{
	"error": "<task> failed"
	}

POST `/v1/user/start-session` StartSession generates an access token for the user.

Uses the refresh token to generate a new access-JWT for the user, if the refresh token is still valid.

Parameters Try it out

Name	Description
Authorization `string` *(header)*	Refresh JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Session start successful **Example Value** | Model

Code	Description


```
{
  "access-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8"
}
```


401 The token is invalid

Example Value | Model


```
{
  "error": "<task> failed"
}
```


500 Internal server error

Example Value | Model


```
{
  "error": "<task> failed"
}
```


Models

athleteManagement.AlreadyExistingAthletesResponse {

- already_existing_athletes: [**databaseUtils.Athlete** {
  - birth_date: string
  - createdAt: string
  - deletedAt: string
  - email: string
  - first_name: string
  - id: integer
  - last_name: string
  - sex: string
  - trainer_email: string
  - updatedAt: string
  }]
- message: string
  example: Creation successful

}

athleteManagement.AthleteBody {

- birth_date: string
  example: YYYY-MM-DD
- email: string
  example: bob.alice@example.com
- first_name: string
  example: Bob
- last_name: string
  example: Alice
- sex: string
  example: <m|f|d>

}

athleteManagement.AthleteBodyWithId {

```
athlete_id      integer
                example: 1
birth_date      string
                example: YYYY-MM-DD
email           string
                example: bob.alice@example.com
first_name      string
                example: Bob
last_name       string
                example: Alice
sex             string
                example: <m|f|d>
swim_cert       boolean
```

}

athleteManagement.AthletesResponse {

```
athletes        [athleteManagement.AthleteBodyWithId {
                  athlete_id      integer
                                  example: 1
                  birth_date      string
                                  example: YYYY-MM-DD
                  email           string
                                  example: bob.alice@example.com
                  first_name      string
                                  example: Bob
                  last_name       string
                                  example: Alice
                  sex             string
                                  example: <m|f|d>
                  swim_cert       boolean
                }]
message         string
                example: Request successful
```

}

backendSettings.ChangeLogLevelRequest {

```
log_level       integer
                example: 1
```

}

databaseUtils.Athlete {

```
birth_date      string
createdAt       string
deletedAt       string
email           string
first_name      string
id              integer
last_name       string
sex             string
trainer_email   string
updatedAt       string
```

}

databaseUtils.ExerciseGoal {

bronze	integer Time: ms; Distance: cm; Points; Bool: <0|1>
createdAt	string
deletedAt	string
description	string
from_age	integer
gold	integer Time: ms; Distance: cm; Points; Bool: <0|1>
id	integer
rulesetId	integer
sex	string
silver	integer Time: ms; Distance: cm; Points; Bool: <0|1>
to_age	integer
updatedAt	string

}

disciplineManagement.DisciplinesResponse {

discipline_names	[*example: List ["Discipline name"]* string]
message	string *example: Request successful*

}

endpoints.ErrorResponse {

error	string *example: <task> failed*

}

endpoints.SuccessResponse {

message	string *example: <task> successful*

}

exerciseManagement.ExerciseBodyWithId {

age_specifics	string *example: Age specific description*
discipline_name	string *example: Discipline*
exercise_id	integer *example: 1*
name	string *example: Exercise*
unit	string *example: minutes*

}

exerciseManagement.ExercisesResponse {

- **exercises** [exerciseManagement.ExerciseBodyWithId {
  - **age_specifics** string
    example: Age specific description
  - **discipline_name** string
    example: Discipline
  - **exercise_id** integer
    example: 1
  - **name** string
    example: Exercise
  - **unit** string
    example: minutes

  }]
- **message** string
  example: Request successful

}

performanceManagement.BulkCreatePerformanceResponse {

- **failed_entries** [performanceManagement.FailedPerformanceEntry {
  - **reason** string
    example: Invalid athlete ID
  - **row** integer
    example: 2

  }]
- **message** string
  example: Bulk Creation successful

}

performanceManagement.CreatePerformanceResponse {

- **medal_status** string
  example: Medal status
- **message** string
  example: Creation successful

}

performanceManagement.ExportRequest {

- **athlete_ids** [
  example: List [1]
  integer]

}

performanceManagement.FailedPerformanceEntry {

- **reason** string
  example: Invalid athlete ID
- **row** integer
  example: 2

}

performanceManagement.PerformanceBody {

- **athlete_id** integer
  example: 1
- **date** string
  example: YYYY-MM-DD
- **exercise_id** integer
  example: 1
- **points** integer
  example: 1

}

performanceManagement.PerformanceBodyEdit {

date	string	*example: YYYY-MM-DD*
exercise_id	integer	*example: 1*
performance_id	integer	*example: 1*
points	integer	*example: 1*

}

performanceManagement.PerformanceBodyWithId {

athlete_id	integer	*example: 1*
date	string	*example: YYYY-MM-DD*
exercise_id	integer	*example: 1*
medal	string	*example: gold*
performance_id	integer	*example: 1*
points	integer	*example: 1*
unit	string	*example: meter*

}

performanceManagement.PerformanceResponse {

message	string	*example: Request successful*
performance_entry	**performanceManagement.PerformanceBodyWithId** {	
	athlete_id	integer *example: 1*
	date	string *example: YYYY-MM-DD*
	exercise_id	integer *example: 1*
	medal	string *example: gold*
	performance_id	integer *example: 1*
	points	integer *example: 1*
	unit	string *example: meter*
	}	

}

performanceManagement.PerformancesResponse {

message	string	*example: Request successful*
performance_entries	[**performanceManagement.PerformanceBodyWithId** {	
	athlete_id	integer *example: 1*
	date	string *example: YYYY-MM-DD*
	exercise_id	integer *example: 1*
	medal	string *example: gold*
	performance_id	integer *example: 1*
	points	integer *example: 1*
	unit	string *example: meter*
	}]	

}

rulesetManagement.RulesetResponse ⌄ {

```
    message                 string
                            example: Request successful
    ruleset_goals            ⌄ [databaseUtils.ExerciseGoal ⌄ {
                                bronze                  integer

                                                        Time: ms; Distance: cm; Points; Bool: <0|1>

                                createdAt               string
                                deletedAt               string
                                description             string
                                from_age                integer
                                gold                    integer

                                                        Time: ms; Distance: cm; Points; Bool: <0|1>

                                id                      integer
                                rulesetId               integer
                                sex                     string
                                silver                  integer

                                                        Time: ms; Distance: cm; Points; Bool: <0|1>

                                to_age                  integer
                                updatedAt               string
                              }]
}
```

userManagement.AccessTokenHolder ⌄ {

```
    access-token            string
                            example:
                            eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-
                            IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8
}
```

userManagement.DoubleTokenHolder ⌄ {

```
    access-token            string
                            example:
                            eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-
                            IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8
    refresh-token           string
                            example:
                            eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-
                            IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8
}
```

userManagement.UserBody ⌄ {

```
    email*                  string
                            example: bob.alice@example.com
    password*               string
                            example: <password>
    remember_me             boolean
                            example: true
}
```

/docs/swagger.json Explore

ComPeteHub API 1.0

[Base URL: localhost:8080]
/docs/swagger.json

This API allows managing athletes, disciplines and performances in the ComPeteHub system.

Team Reissdorf - Website
Send email to Team Reissdorf
MIT License

Schemes
HTTP

Authorize

Athlete Management

POST /v1/athlete/bulk-create Bulk creates new athletes from csv file

Upload a CSV file to create multiple athlete profiles. If an athlete already exists, the process will continue, and the response will indicate which athletes already exist.

Parameters Try it out

Name	Description
Athletes * required `file` *(formData)*	CSV file containing details of multiple athletes to create profiles Datei auswählen Keine ausgewählt
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
201	Creation successful **Example Value** Model


```
{
  "already_existing_athletes": [
    {
      "birth_date": "string",
      "createdAt": "string",
      "deletedAt": "string",
      "email": "string",
      "first_name": "string",
```

Code	Description


```
      "id": 0,
      "last_name": "string",
      "sex": "string",
      "trainer_email": "string",
      "updatedAt": "string"
    }
  ],
  "message": "Creation successful"
}
```


400 Invalid request body

Example Value Model


```
{
  "error": "<task> failed"
}
```


401 The token is invalid

Example Value Model


```
{
  "error": "<task> failed"
}
```


409 All athletes already exist; none have been created

Example Value Model


```
{
  "already_existing_athletes": [
    {
      "birth_date": "string",
      "createdAt": "string",
      "deletedAt": "string",
      "email": "string",
      "first_name": "string",
      "id": 0,
      "last_name": "string",
      "sex": "string",
      "trainer_email": "string",
      "updatedAt": "string"
    }
  ],
  "message": "Creation successful"
}
```


500 Internal server error

Example Value Model


```
{
  "error": "<task> failed"
}
```


POST `/v1/athlete/create` Creates a new athlete profile

Creates a new athlete profile with the given data. Duplicate email and birthdate combinations are not allowed.

Parameters Try it out

Name	Description
Athlete * required `object` *(body)*	Details of an athlete to create a profile **Example Value** Model

Name	Description


```
{
  "birth_date": "YYYY-MM-DD",
  "email": "bob.alice@example.com",
  "first_name": "Bob",
  "last_name": "Alice",
  "sex": "<m|f|d>"
}
```


Parameter content type

application/json

Authorization
`string`
(header)

Access JWT is sent in the Authorization header or set as a http-only cookie

Authorization

Responses

Response content type application/json

Code	Description

201 Creation successful

Example Value Model


```
{
  "message": "<task> successful"
}
```


400 Invalid request body

Example Value Model


```
{
  "error": "<task> failed"
}
```


401 The token is invalid

Example Value Model


```
{
  "error": "<task> failed"
}
```


409 Athlete already exists

Example Value Model


```
{
  "error": "<task> failed"
}
```


500 Internal server error

Example Value Model


```
{
  "error": "<task> failed"
}
```


DELETE `/v1/athlete/delete/{AthleteId}` Deletes the given athlete profile

Deletes the given athlete profile.

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	Delete the given athlete AthleteId
Authorization `string` *(path)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Deletion successful **Example Value** Model `{ "message": "<task> successful" }`
400	Invalid request parameter **Example Value** Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** Model `{ "error": "<task> failed" }`
404	Athlete could not be found for this trainer **Example Value** Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** Model `{ "error": "<task> failed" }`

PUT `/v1/athlete/edit` Edits an existing athlete profile

Edits an existing athlete profile with the given details. Duplicate email and birthdate combinations are not allowed.

Parameters Try it out

Name	Description

Athlete * required
`object`
(body)
	Edited details of an athlete
	Example Value Model
	<pre>{
	"athlete_id": 1,
	"birth_date": "YYYY-MM-DD",
	"email": "bob.alice@example.com",
	"first_name": "Bob",
	"last_name": "Alice",
	"sex": "<m|f|d>",
	"swim_cert": true
	}</pre>**Parameter content type**
	application/json

Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description

200	Edited successful
	Example Value Model
	<pre>{
	"message": "<task> successful"
	}</pre>

400	Invalid request body
	Example Value Model
	<pre>{
	"error": "<task> failed"
	}</pre>

401	The token is invalid
	Example Value Model
	<pre>{
	"error": "<task> failed"
	}</pre>

404	Athlete could not be found for this trainer **Example Value** Model

Code	Description

```
{
  "error": "<task> failed"
}
```

409 Athlete already exists

Example Value Model

```
{
  "error": "<task> failed"
}
```

500 Internal server error

Example Value Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/athlete/get-all` Returns all athlete profiles and swim certificates

All athlete profiles and swim certificates of the given trainer are returned

Parameters Try it out

Name	Description
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description

200 Request successful

Example Value Model

```
{
  "athletes": [
    {
      "athlete_id": 1,
      "birth_date": "YYYY-MM-DD",
      "email": "bob.alice@example.com",
      "first_name": "Bob",
      "last_name": "Alice",
      "sex": "<m|f|d>",
      "swim_cert": true
    }
  ],
  "message": "Request successful"
}
```

401 The token is invalid

Example Value Model

Code	Description

```
{
  "error": "<task> failed"
}
```

500 Internal server error

Example Value Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/athlete/get/{AthleteId}` Returns one athlete profile

One athlete profile with given id and of the given trainer gets returned

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	Get the athlete with the given id AthleteId
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description

200 Request successful

Example Value Model

```
{
  "athletes": [
    {
      "athlete_id": 1,
      "birth_date": "YYYY-MM-DD",
      "email": "bob.alice@example.com",
      "first_name": "Bob",
      "last_name": "Alice",
      "sex": "<m|f|d>",
      "swim_cert": true
    }
  ],
  "message": "Request successful"
}
```

401 The token is invalid

Example Value Model

```
{
  "error": "<task> failed"
}
```

Code	Description
404	Athlete not found **Example Value** Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** Model `{ "error": "<task> failed" }`

Settings

POST `/v1/backendSettings/change-log-level` ChangeLogLevel changes the log level of the server

Changes the log level of the server to the specified level 0-4 (Debug-Fatal).

Parameters Try it out

Name	Description
Authorization `string` *(header)*	Settings access JWT is sent in the Authorization header or set as a http-only cookie Authorization
Log-level * required `object` *(body)*	Log level **Example Value** Model `{ "log_level": 1 }` **Parameter content type** application/json

Responses Response content type application/json

Code	Description
200	Log level changed **Example Value** Model `{ "message": "<task> successful" }`
400	Invalid request body

Code	Description

Example Value Model

```
{
  "error": "<task> failed"
}
```

401 The token is invalid

Example Value Model

```
{
  "error": "<task> failed"
}
```

500 Internal server error

Example Value Model

```
{
  "error": "<task> failed"
}
```

HealthCheck

GET `/v1/coffee` Returns a 418 I'm a teapot response

Simple endpoint to return a 418 I'm a teapot response.

Parameters Try it out

No parameters

Responses Response content type application/json

Code	Description

418 Error: I'm a teapot

Example Value Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/ping` Returns a pong response

Simple ping-pong endpoint to check if the server is running properly.

Parameters Try it out

No parameters

Responses Response content type application/json

Code	Description
200	Pong response **Example Value** Model `{ "message": "<task> successful" }`

Discipline Management

GET `/v1/discipline/get-all` Returns all discipline names

All discipline names are returned from the database

Parameters Try it out

Name	Description
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Request successful **Example Value** Model `{ "discipline_names": [ "Discipline name" ], "message": "Request successful" }`
401	The token is invalid **Example Value** Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** Model

Code	Description

```
{
  "error": "<task> failed"
}
```

Exercise Management

GET `/v1/exercise/get/{DisciplineName}` Returns the exercises

All exercises of the given discipline will be returned. When the athlete id is given, the age specific description will be returned with the exercise.

Parameters Try it out

Name	Description
DisciplineName * required `string` *(path)*	Get the exercises with the given discipline name DisciplineName
athlete-id `integer` *(query)*	Get the exercise_specifics for the given athletes age and sex athlete-id
performance-date `string` *(query)*	Date in YYYY-MM-DD format to get the exercises according to the ruleset of the given year performance-date
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Request successful **Example Value** Model

```
{
  "exercises": [
    {
      "age_specifics": "Age specific description",
      "discipline_name": "Discipline",
      "exercise_id": 1,
      "name": "Exercise",
      "unit": "minutes"
    }
  ],
  "message": "Request successful"
}
```

401	The token is invalid **Example Value** Model

Code	Description

```
{
  "error": "<task> failed"
}
```

404 Discipline, athlete or ruleset year does not exist

Example Value Model

```
{
  "error": "<task> failed"
}
```

500 Internal server error

Example Value Model

```
{
  "error": "<task> failed"
}
```

Performance Management

POST `/v1/performance/bulk-create` Bulk create performance entries from CSV

Upload a .csv file to bulk-create performance entries.

Parameters Try it out

Name	Description
Performances * required `file` *(formData)*	CSV file; must have extension .csv; columns: lastName;firstName;gender;birthYear;birthDate;exercise;category;date;result;points Datei auswählen Keine ausgewählt
Authorization `string` *(header)*	Bearer JWT token Authorization

Responses Response content type application/json

Code	Description

201 Bulk creation successful

Example Value Model

```
{
  "failed_entries": [
    {
      "reason": "Invalid athlete ID",
      "row": 2
    }
  ],
  "message": "Bulk Creation successful"
}
```

Code	Description
400	Bad request: missing file / invalid CSV / wrong extension **Example Value** Model `{ "error": "<task> failed" }`
401	Unauthorized: invalid or missing token **Example Value** Model `{ "error": "<task> failed" }`
409	Conflict: all entries failed, none created **Example Value** Model `{ "error": "<task> failed" }`
500	Internal server error (DB failure or file read error) **Example Value** Model `{ "error": "<task> failed" }`

POST `/v1/performance/create` Creates a new performance entry

Creates a new performance entry with the given data. Maximum 3 entries/discipline/athlete/day are allowed (performance limit).

Parameters Try it out

Name	Description
Performance * required `object` *(body)*	Details of a performance (valid units are: <millisecond, centimeter, point, bool>) **Example Value** Model `{ "athlete_id": 1, "date": "YYYY-MM-DD", "exercise_id": 1, "points": 1 }` **Parameter content type** application/json
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses

Response content type application/json

Code	Description
201	Creation successful **Example Value** Model `{ "medal_status": "Medal status", "message": "Creation successful" }`
400	Invalid request body **Example Value** Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** Model `{ "error": "<task> failed" }`
404	Athlete or exercise does not exist **Example Value** Model `{ "error": "<task> failed" }`
409	Performance limit reached **Example Value** Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** Model `{ "error": "<task> failed" }`

PUT `/v1/performance/edit` Edits an existing performance entry

Edits an existing performance entry with the given details.

Parameters Try it out

Name	Description
Performance * required `object`	Edited details of a performance entry

Name	Description
(body)	**Example Value** Model

```
{
  "date": "YYYY-MM-DD",
  "exercise_id": 1,
  "performance_id": 1,
  "points": 1
}
```

Parameter content type

application/json

Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Edited successful **Example Value** Model

```
{
  "message": "<task> successful"
}
```

400	Invalid request body **Example Value** Model

```
{
  "error": "<task> failed"
}
```

401	The token is invalid **Example Value** Model

```
{
  "error": "<task> failed"
}
```

404	Performance entry or goals not found **Example Value** Model

```
{
  "error": "<task> failed"
}
```

500	Internal server error **Example Value** Model

```
{
  "error": "<task> failed"
}
```

POST `/v1/performance/export` Exports all performance entries of the specified athletes as a csv file

Exports all performance entries of the specified athletes as a csv file

Parameters Try it out

Name	Description
json * required `object` *(body)*	JSON payload in the format: **Example Value** Model `{ "athlete_ids": [ 1 ] }` **Parameter content type** application/json
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type text/csv

Code	Description
200	CSV file **Example Value** Model `(no example available)`
400	Invalid request body **Example Value** Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** Model `{ "error": "<task> failed" }`
404	One or more athletes do not exist **Example Value** Model `{ "error": "<task> failed" }`
500	Internal server error

Code	Description

Example Value Model

```
{
  "error": "<task> failed"
}
```

GET `/v1/performance/get-latest/{AthleteId}` Returns the latest performance entry

Returns the latest performance entry from the database

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	Get the latest performance entry of the given athlete_id AthleteId
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Request successful

Example Value Model

```
{
  "message": "Request successful",
  "performance_entry": {
    "athlete_id": 1,
    "date": "YYYY-MM-DD",
    "exercise_id": 1,
    "medal": "gold",
    "performance_id": 1,
    "points": 1,
    "unit": "meter"
  }
}
```

401 The token is invalid

Example Value Model

```
{
  "error": "<task> failed"
}
```

404 Athlete or performance entry not found

Example Value Model

```
{
  "error": "<task> failed"
}
```

Code	Description
500	Internal server error **Example Value** Model `{ "error": "<task> failed" }`

GET `/v1/performance/get/{AthleteId}` Returns all performance entries of the given athlete

Returns all performance entries of the given athlete and can be filtered using the 'since' query parameter

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	Get all performance entries of the given athlete AthleteId
since `string` *(query)*	Date in YYYY-MM-DD format to get all entries since then (including the entries from that day) since
date `string` *(query)*	Date in YYYY-MM-DD format to get all entries from a specific day date
best `boolean` *(query)*	If true, only the latest best performance entries for each discipline will be returned --
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as an http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Request successful **Example Value** Model `{ "message": "Request successful", "performance_entries": [ { "athlete_id": 1, "date": "YYYY-MM-DD", "exercise_id": 1, "medal": "gold", "performance_id": 1, "points": 1, "unit": "meter" } ] }`

Code	Description
400	Date parameter is before the since-parameter **Example Value** Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** Model `{ "error": "<task> failed" }`
404	Athlete does not exist **Example Value** Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** Model `{ "error": "<task> failed" }`

Ruleset Management

POST `/v1/ruleset/create` Creates new ruleset entries from csv file

Upload a CSV file to create multiple ruleset entries. Needs to contain 11 columns.

Parameters Try it out

Name	Description
RulesetEntries * required `file` *(formData)*	CSV file containing details of the ruleset Datei auswählen Keine ausgewählt
Authorization `string` *(header)*	Access JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Creation successful **Example Value** Model `{ "message": "<task> successful" }`
400	Invalid request body **Example Value** Model `{ "error": "<task> failed" }`
401	The token is invalid **Example Value** Model `{ "error": "<task> failed" }`
409	All ruleset entries already exist; none have been created **Example Value** Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** Model `{ "error": "<task> failed" }`

GET `/v1/ruleset/get` Returns the rulesets

Retrieves the rulesets of a specific exercise with the given exercise_id and year.

Parameters Try it out

Name	Description
year * required `integer` *(query)*	Year of the ruleset year
exercise_id * required `integer` *(query)*	ID of the exercise exercise_id

Responses Response content type application/json

Code	Description
200	Request successful **Example Value** Model


```
{
  "message": "Request successful",
  "ruleset_goals": [
    {
      "bronze": 0,
      "createdAt": "string",
      "deletedAt": "string",
      "description": "string",
      "from_age": 0,
      "gold": 0,
      "id": 0,
      "rulesetId": 0,
      "sex": "string",
      "silver": 0,
      "to_age": 0,
      "updatedAt": "string"
    }
  ]
}
```


Code	Description
400	Invalid query parameters **Example Value** Model


```
{
  "error": "<task> failed"
}
```


Code	Description
404	Rulesets not found **Example Value** Model


```
{
  "error": "<task> failed"
}
```


Code	Description
500	Internal server error **Example Value** Model


```
{
  "error": "<task> failed"
}
```


Swim Certificate

POST `/v1/swimCertificate/create/{AthleteId}` Uploads a swim certificate for an athlete.

A Trainer can upload a swim certificate as a file for an specified athlete.

Parameters Try it out

Name	Description
file * required `file` *(formData)*	File to upload Datei auswählen Keine ausgewählt

Name	Description
AthleteId * required `integer` *(path)*	Get the latest performance entry of the given athlete_id AthleteId
Authorization `string` *(header)*	JWT Token Authorization

Responses Response content type: application/json

Code	Description
200	Upload successful **Example Value** Model `{ "message": "<task> successful" }`
400	Invalid request **Example Value** Model `{ "error": "<task> failed" }`
401	Unauthorized **Example Value** Model `{ "error": "<task> failed" }`
500	Internal server error **Example Value** Model `{ "error": "<task> failed" }`

GET `/v1/swimCertificate/download-all/{AthleteId}` Download all swim certificates for an athlete

A trainer can download all swim certificates for a specified athlete. The result is a ZIP file containing all documents.

Parameters Try it out

Name	Description
AthleteId * required `integer` *(path)*	ID of the athlete AthleteId

Name	Description
Authorization `string` *(header)*	JWT token Authorization

Responses Response content type: application/zip

Code	Description
200	ZIP archive with all swim certificates **Example Value** Model (no example available)
400	Invalid request **Example Value** Model { "error": "<task> failed" }
401	Unauthorized **Example Value** Model { "error": "<task> failed" }
404	Swim certificates not found or athlete not found **Example Value** Model { "error": "<task> failed" }
500	Internal server error **Example Value** Model { "error": "<task> failed" }

User Management

POST `/v1/user/login` Login to an existing user account

Logs the user into the system with the provided credentials and returns a refresh-JWT.

Parameters Try it out

Name	Description

User * required
`object`
(body)
	The user's email address and password, along with a 'remember_me' field. If set to false or left empty, the refresh token cookie will not have a maxAge flag, causing the browser to automatically delete it when the session ends.
	
	Example Value Model
	
	{
	"email": "bob.alice@example.com",
	"password": "<password>",
	"remember_me": true
	}
	
	Parameter content type
	application/json

Responses

Response content type: application/json

Code	Description

200	Login successful
	
	Example Value Model
	
	{
	"access-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8",
	"refresh-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8"
	}

400	Invalid request body
	
	Example Value Model
	
	{
	"error": "<task> failed"
	}

401	Wrong credentials
	
	Example Value Model
	
	{
	"error": "<task> failed"
	}

404	User not found
	
	Example Value Model
	
	{
	"error": "<task> failed"
	}

500	Internal server error
	
	Example Value Model
	
	{
	"error": "<task> failed"
	}

POST `/v1/user/logout` Logout of a user account

Logs the user out by clearing cookies from the client's browser.

Parameters Try it out

No parameters

Responses Response content type application/json

Code	Description
200	Logout successful **Example Value** Model `{ "message": "<task> successful" }`
500	Internal server error **Example Value** Model `{ "error": "<task> failed" }`

POST `/v1/user/register` Register a new user

Creates a new user account in the database with the provided details.

Parameters Try it out

Name	Description
User * required `object` *(body)*	The user's email address and password, along with a 'remember_me' field. If set to false or left empty, the refresh token cookie will not have a maxAge flag, causing the browser to automatically delete it when the session ends. **Example Value** Model `{ "email": "bob.alice@example.com", "password": "<password>", "remember_me": true }` **Parameter content type** application/json

Responses

Response content type application/json

Code	Description

201	Registration successful
	Example Value Model
	{
	"access-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8",
	"refresh-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8"
	}

400	Invalid request body
	Example Value Model
	{
	"error": "<task> failed"
	}

409	User already exists
	Example Value Model
	{
	"error": "<task> failed"
	}

500	Internal server error
	Example Value Model
	{
	"error": "<task> failed"
	}

POST `/v1/user/start-session` StartSession generates an access token for the user.

Uses the refresh token to generate a new access-JWT for the user, if the refresh token is still valid.

Parameters Try it out

Name	Description
Authorization `string` *(header)*	Refresh JWT is sent in the Authorization header or set as a http-only cookie Authorization

Responses Response content type application/json

Code	Description
200	Session start successful **Example Value** Model

Code	Description


```
{
  "access-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8"
}
```


401 The token is invalid

Example Value Model


```
{
  "error": "<task> failed"
}
```


500 Internal server error

Example Value Model


```
{
  "error": "<task> failed"
}
```


Models

athleteManagement.AlreadyExistingAthletesResponse {

already_existing_athletes	[databaseUtils.Athlete {	
birth_date		string
createdAt		string
deletedAt		string
email		string
first_name		string
id		integer
last_name		string
sex		string
trainer_email		string
updatedAt		string
}]		
message	string	
	example: Creation successful	

}

athleteManagement.AthleteBody {

birth_date	string	
	example: YYYY-MM-DD	
email	string	
	example: bob.alice@example.com	
first_name	string	
	example: Bob	
last_name	string	
	example: Alice	
sex	string	
	example: <m|f|d>	

}

athleteManagement.AthleteBodyWithId {

athlete_id	integer *example: 1*
birth_date	string *example: YYYY-MM-DD*
email	string *example: bob.alice@example.com*
first_name	string *example: Bob*
last_name	string *example: Alice*
sex	string *example: <m|f|d>*
swim_cert	boolean

}

athleteManagement.AthletesResponse {

athletes [**athleteManagement.AthleteBodyWithId** {

athlete_id	integer *example: 1*
birth_date	string *example: YYYY-MM-DD*
email	string *example: bob.alice@example.com*
first_name	string *example: Bob*
last_name	string *example: Alice*
sex	string *example: <m|f|d>*
swim_cert	boolean

}]

message	string *example: Request successful*

}

backendSettings.ChangeLogLevelRequest {

log_level	integer *example: 1*

}

databaseUtils.Athlete {

birth_date	string
createdAt	string
deletedAt	string
email	string
first_name	string
id	integer
last_name	string
sex	string
trainer_email	string
updatedAt	string

}

databaseUtils.ExerciseGoal {

```
    bronze                  integer

                            Time: ms; Distance: cm; Points; Bool: <0|1>

    createdAt               string
    deletedAt               string
    description             string
    from_age                integer
    gold                    integer

                            Time: ms; Distance: cm; Points; Bool: <0|1>

    id                      integer
    rulesetId               integer
    sex                     string
    silver                  integer

                            Time: ms; Distance: cm; Points; Bool: <0|1>

    to_age                  integer
    updatedAt               string
}
```

disciplineManagement.DisciplinesResponse {

```
    discipline_names
                                [
                            example: List [ "Discipline name" ]
                            string]
    message                 string
                            example: Request successful
}
```

endpoints.ErrorResponse {

```
    error                   string
                            example: <task> failed
}
```

endpoints.SuccessResponse {

```
    message                 string
                            example: <task> successful
}
```

exerciseManagement.ExerciseBodyWithId {

```
    age_specifics           string
                            example: Age specific description
    discipline_name         string
                            example: Discipline
    exercise_id             integer
                            example: 1
    name                    string
                            example: Exercise
    unit                    string
                            example: minutes
}
```

exerciseManagement.ExercisesResponse {

- **exercises** [**exerciseManagement.ExerciseBodyWithId** {
  - **age_specifics** string
    example: Age specific description
  - **discipline_name** string
    example: Discipline
  - **exercise_id** integer
    example: 1
  - **name** string
    example: Exercise
  - **unit** string
    example: minutes

  }]
- **message** string
  example: Request successful

}

performanceManagement.BulkCreatePerformanceResponse {

- **failed_entries** [**performanceManagement.FailedPerformanceEntry** {
  - **reason** string
    example: Invalid athlete ID
  - **row** integer
    example: 2

  }]
- **message** string
  example: Bulk Creation successful

}

performanceManagement.CreatePerformanceResponse {

- **medal_status** string
  example: Medal status
- **message** string
  example: Creation successful

}

performanceManagement.ExportRequest {

- **athlete_ids** [
  example: List [1]
  integer]

}

performanceManagement.FailedPerformanceEntry {

- **reason** string
  example: Invalid athlete ID
- **row** integer
  example: 2

}

performanceManagement.PerformanceBody {

- **athlete_id** integer
  example: 1
- **date** string
  example: YYYY-MM-DD
- **exercise_id** integer
  example: 1
- **points** integer
  example: 1

}

performanceManagement.PerformanceBodyEdit {

```
    date                    string
                            example: YYYY-MM-DD
    exercise_id             integer
                            example: 1
    performance_id          integer
                            example: 1
    points                  integer
                            example: 1
}
```

performanceManagement.PerformanceBodyWithId {

```
    athlete_id              integer
                            example: 1
    date                    string
                            example: YYYY-MM-DD
    exercise_id             integer
                            example: 1
    medal                   string
                            example: gold
    performance_id          integer
                            example: 1
    points                  integer
                            example: 1
    unit                    string
                            example: meter
}
```

performanceManagement.PerformanceResponse {

```
    message                 string
                            example: Request successful
    performance_entry
```

performanceManagement.PerformanceBodyWithId {

```
        athlete_id          integer
                            example: 1
        date                string
                            example: YYYY-MM-DD
        exercise_id         integer
                            example: 1
        medal               string
                            example: gold
        performance_id      integer
                            example: 1
        points              integer
                            example: 1
        unit                string
                            example: meter
    }
}
```

performanceManagement.PerformancesResponse {

```
    message                 string
                            example: Request successful
    performance_entries
```

[performanceManagement.PerformanceBodyWithId {

```
        athlete_id          integer
                            example: 1
        date                string
                            example: YYYY-MM-DD
        exercise_id         integer
                            example: 1
        medal               string
                            example: gold
        performance_id      integer
                            example: 1
        points              integer
                            example: 1
        unit                string
                            example: meter
     }]
}
```

rulesetManagement.RulesetResponse {

```
    message                 string
                            example: Request successful
    ruleset_goals               [databaseUtils.ExerciseGoal    {
                                bronze                  integer

                                                        Time: ms; Distance: cm; Points; Bool: <0|1>

                                createdAt               string
                                deletedAt               string
                                description             string
                                from_age                integer
                                gold                    integer

                                                        Time: ms; Distance: cm; Points; Bool: <0|1>

                                id                      integer
                                rulesetId               integer
                                sex                     string
                                silver                  integer

                                                        Time: ms; Distance: cm; Points; Bool: <0|1>

                                to_age                  integer
                                updatedAt               string
                             }]
}
```

userManagement.AccessTokenHolder {

```
    access-token            string
                            example:
                            eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-
                            IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8
}
```

userManagement.DoubleTokenHolder {

```
    access-token            string
                            example:
                            eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-
                            IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8
    refresh-token           string
                            example:
                            eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI8dXNlci1pZD4iLCJuYW1lIjoiPHRva2VuLXR5cGU-
                            IiwiaWF0IjoxNzM0Njk4NzEwfQ.hzvbcP77EO8dnEyy5i-OgoOp8MYYwslfwKx32ZKgrH8
}
```

userManagement.UserBody {

```
    email*                  string
                            example: bob.alice@example.com
    password*               string
                            example: <password>
    remember_me             boolean
                            example: true
}
```