A thorough documentation of the Hulan Beer API located at https://intake-api.hulan.nl.

Available endpoints

Authentication

- Login : `POST /authentication/`
- Validate : `POST /authentication/validate`
- Logout : `POST /authentication/logout`

Users

- Create : `POST /users/`
- Get single : `GET /users/:id`
- Get many : `GET /users/:ids`
- Get all : `GET /users/`
- Overwrite : `PUT /users/`
- Merge : `PATCH /users/`
- Delete : `DELETE /users/:id`

Brewers

- Create : `POST /brewers/`
- Get single : `GET /brewers/:id`
- Get many : `GET /brewers/:ids`
- Get all : `GET /brewers/`
- Overwrite : `PUT /brewers/`
- Merge : `PATCH /brewers/`
- Delete : `DELETE /brewers/:id`

Beers

- Create : `POST /beers/`
- Get single : `GET /beers/:id`
- Get many : `GET /beers/:ids`
- Get all : `GET /beers/`
- Overwrite : `PUT /beers/`
- Merge : `PATCH /beers/`
- Delete : `DELETE /beers/:id`

Query parameters

The following query parameters can be used on every `GET` request in the application.

Populate parameter

This parameter allows you to populate references to other collections in the response.

Key : `populate`

Value : A comma separated value of the fields you want populated. You can deep populate values by separating fields using a dot (f.e. `brewers.beers`). If no value has been defined, the algorithm will populate all references in the root of the results.

Example : `../beers?populate=brewers.beers`

Filter parameter

This parameter allows you to filter the response data on one or more fields on specific values.

Key : `filter[:field]`

Value : The value you want the field defined in the key to be filtered on. If the schema type of the field is defined as a string, the filter will check if the given value is a part of the field value (case insensitive).

Example : `../beers?filter[name]=uinnes&filter[percentage]=5`

Search parameter

This parameter allows you to search through all fields of the response using the same value. Results matching the value in at least one of the fields will be shown in the response. You can limit the fields which fields in which the algorithm searches using the search scope parameter.

Key : `search`

Value : The value you want the field defined in the key to be filtered on. If the schema type of the field is defined as a string, the filter will check if the given value is a part of the field value (case insensitive).

Example : `../beers?search=sour`

Search scope parameter

This parameter allows you to sort the response data on one or more fields in the desired order.

Key : `searchScope`

Value : A comma separated value of all fields on which the search needs to be executed. If no search scope it set, the search will be conducted on all fields of the model.

Example : `../beers?search=sour&searchScope=name,beerType`

Pick parameter

This parameter allows you to define which fields you want the results to contain. If one or more fields have been picked for a layer, the remaining layers will be omitted from the response. You can deep pick fields by separating fields using a dot (f.e. `brewers.name`).

Key : `pick`

Value : A comma separated value of all fields which you want returned in the response.

Example : `../beers?pick=name,beerType`

Sort parameter

This parameter allows you to sort the response data on one or more fields in the desired order.

Key : `sort`

Value : A comma separated value of all fields on which the response needs to be sorted. A field can be prefixed with a hyphen (`-`) in order to by it in descending order.

Example : `../beers?sort=-percentage,name`

Offset parameter

This parameter allows you to skip the first *n* number of results.

Key : `offset`

Value : An integer value which defines how many results you want to skip.

Example : `../beers?offset=10`

Limit parameter

This parameter allows you to skip the first *n* number of results.

Key : `limit`

Value : An integer value which defines the maximum number of results you want returned.

Example : `../beers?limit=5`

Endpoint definitions

Authentication endpoints

Login

Log into the application. The token received in the response can be used in the `Authorization` header to authorize for endpoints in the application.

Path : `/authentication/`

Method : `POST`

Authorization : Open

Request body

```
Credentials: {
  username: string;
  password: string;
}
```

Success response

```
SessionToken: {
  id: string;
  token: string;
  timestamp: Date;
  userId: string;

  // references
  user: User;
}
```

Validate

Validate the used session. The session can be defined by using the `Authorization` header in the request.

Path : `/authentication/validate`

Method : `POST`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
SessionToken: {
  id: string;
  token: string;
  timestamp: Date;
  userId: string;

  // references
  user: User;
}
```

Logout

Log out of the application. The session can be defined by using the `Authorization` header in the request.

Path : `/authentication/logout`

Method : `POST`

Authorization : Open

User endpoints

Create user

Create a User document in the database.

Path : `/users/`

Method : `POST`

Authorization : `Role.ADMIN`

Request body

```
User: {
  username: string;
  role: Role;
}
```

References: Role

Success response

```
User: {
  id: string;
  username: string;
  role: Role;
}
```

References: Role

Get single user

Get a specific User document by its ID from the database.

Path : `/users/:id`

Method : `GET`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
User: {
  id: string;
  username: string;
  role: Role;
}
```

References: Role

Get many users

Get a specific set of User documents by their ID's from the database.

Path : `/users/:ids`

Method : `GET`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
User: {
  id: string;
  username: string;
  role: Role;
}[]
```

References: Role

Get all users

Get all User documents you have access to from the database.

Path : `/users/`

Method : `GET`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
User: {
  id: string;
  username: string;
  role: Role;
}[]
```

References: Role

Overwrite user

Overwrite a User document in the database.

Path : `/users/`

Method : `PUT`

Authorization : `Role.ADMIN`

Request body

```
User: {
  id: string
  username: string;
  role: Role;
}
```

References: Role

Success response

```
User: {
  id: string;
  username: string;
  role: Role;
}
```

References: Role

Merge user

Merge a User document in the database.

Path : `/users/`

Method : `PATCH`

Authorization : `Role.ADMIN`

Request body

```
User: {
  id: string
  username?: string;
  role?: Role;
}
```

References: Role

Success response

```
User: {
  id: string;
  username: string;
  role: Role;
}
```

References: Role

Delete user

Delete a specific User document from the database.

Path : `/users/:id`

Method : `DELETE`

Authorization : `Role.ADMIN`

Success response

```
User: {
  id: string;
  username: string;
  role: Role;
}
```

References: Role

Brewer endpoints

Create brewer

Create a Brewer document in the database.

Path : `/brewers/`

Method : `POST`

Authorization : `Role.ADMIN, Role.USER`

Request body

```
Brewer: {
  name: string;
  city: string;
  beerIds: string[];
}
```

Success response

```
Brewer: {
  id: string
  name: string;
  city: string;
  beerIds: string[];

  // references
  beers?: Beer[];
}
```

Get single brewer

Get a specific Brewer document by its ID from the database.

Path : `/brewers/:id`

Method : `GET`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
Brewer: {
  id: string
  name: string;
  city: string;
  beerIds: string[];

  // references
  beers?: Beer[];
}
```

Get many brewers

Get a specific set of Brewer documents by their ID's from the database.

Path : `/brewers/:ids`

Method : `GET`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
Brewer: {
  id: string
  name: string;
  city: string;
  beerIds: string[];

  // references
  beers?: Beer[];
}[]
```

Get all brewers

Get all Brewer documents you have access to from the database.

Path : `/brewers/`

Method : `GET`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
Brewer: {
  id: string
  name: string;
  city: string;
  beerIds: string[];

  // references
  beers?: Beer[];
}[]
```

Overwrite brewer

Overwrite a Brewer document in the database.

Path : `/brewers/`

Method : `PUT`

Authorization : `Role.ADMIN, Role.USER`

Request body

```
Brewer: {
  id: string
  name: string;
  city: string;
  beerIds: string[];
}
```

Success response

```
Brewer: {
  id: string
  name: string;
  city: string;
  beerIds: string[];

  // references
  beers?: Beer[];
}
```

Merge brewer

Merge a Brewer document in the database.

Path : `/brewers/`

Method : `PATCH`

Authorization : `Role.ADMIN, Role.USER`

Request body

```
Brewer: {
  id: string
  name?: string;
  city?: string;
  beerIds?: string[];
}
```

Success response

```
Brewer: {
  id: string
  name: string;
  city: string;
  beerIds: string[];

  // references
  beers?: Beer[];
}
```

Delete brewer

Delete a specific Brewer document from the database.

Path : `/brewers/:id`

Method : `DELETE`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
Brewer: {
  id: string
  name: string;
  city: string;
  beerIds: string[];

  // references
  beers?: Beer[];
}
```

Beer endpoints

Create beer

Create a Beer document in the database.

Path : `/beers/`

Method : `POST`

Authorization : `Role.ADMIN, Role.USER`

Request body

```
Beer: {
  name: string;
  description: string;
  type: BeerType;
  percentage: number;
}
```

References: BeerType

Success response

```
Beer: {
  id: string;
  name: string;
  description: string;
  type: BeerType;
  percentage: number;
}
```

References: BeerType

Get single beer

Get a specific Beer document by its ID from the database.

Path : `/beers/:id`

Method : `GET`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
Beer: {
  id: string;
  name: string;
  description: string;
  type: BeerType;
  percentage: number;

  // references
  brewers?: Brewer[];
}
```

References: BeerType

Get many beers

Get a specific set of Beer documents by their ID's from the database.

Path : `/beers/:ids`

Method : `GET`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
Beer: {
  id: string;
  name: string;
  description: string;
  type: BeerType;
  percentage: number;

  // references
  brewers?: Brewer[];
}[]
```

References: BeerType

Get all beers

Get all Beer documents you have access to from the database.

Path : `/beers/`

Method : `GET`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
Beer: {
  id: string;
  name: string;
  description: string;
  type: BeerType;
  percentage: number;

  // references
  brewers?: Brewer[];
}[]
```

References: BeerType

Overwrite beer

Overwrite a Beer document in the database.

Path : `/beers/`

Method : `PUT`

Authorization : `Role.ADMIN, Role.USER`

Request body

```
Beer: {
  id: string;
  name: string;
  description: string;
  type: BeerType;
  percentage: number;
}
```

References: BeerType

Success response

```
Beer: {
  id: string;
  name: string;
  description: string;
  type: BeerType;
  percentage: number;
}
```

References: BeerType

Merge beer

Merge a Beer document in the database.

Path : `/beers/`

Method : `PATCH`

Authorization : `Role.ADMIN, Role.USER`

Request body

```
Beer: {
  id: string;
  name?: string;
  description?: string;
  type?: BeerType;
  percentage?: number;
}
```

References: BeerType

Success response

```
Beer: {
  id: string;
  name: string;
  description: string;
  type: BeerType;
  percentage: number;
}
```

References: BeerType

Delete beer

Delete a specific Beer document from the database.

Path : `/beers/:id`

Method : `DELETE`

Authorization : `Role.ADMIN, Role.USER`

Success response

```
Beer: {
  id: string;
  name: string;
  description: string;
  type: BeerType;
  percentage: number;
}
```

References: BeerType

Enumerables

Role enumerable

```
enum Role {
  USER,
  ADMIN
}
```

Beer type enumerable

```
enum BeerType {
  PILSNER,
  IPA,
  WHEAT,
  BROWN,
  PORTER,
  STOUT,
  SOUR
}
```