Calorie REST Api Doc

By Vladimir Slav

Contents

Introduction

This document describes how to use calorie rest api. The calorie API provides CRUD functionality for users and their meals.

Endpoints and their brief descriptions

The API provides following endpoints. Detailed descriptions follow after.

Method	Endpoint	Function
GET	/doc	Download this documentation
GET	/users/{user_id}	Get Specific User
GET	/users/list/{amount}/{page}	Get User List
POST	/users/	Create a new user
POST	/users/auth/	Authenticate a user
PUT	/users/{user_id}	Change User Details
PUT	/users/reset	Reset User Password (and send it to user's mail)
DELETE	/users/{user_id}	Delete User
GET	/logout	Log Out current user
POST	/meals/{owner_id}	Create a new meal for User with {owner_id}
GET	/meals/{meal_id}	Get Specific Meal
GET	/meals/list/{owner_id}	Get meals of User with {owner_id}
GET	/meals/filter/{owner_id}	Get meals filtered by parameters
DELETE	/meals/{meal_id}	Delete meal with specificē id
PUT	/meals/{meal_id}	Update specificē meal

Detailed Endpoint Descriptions

In the following rest endpoint descriptions all parameters are mandatory unless stated otherwise.

- The rest api always returns json. Structure always contains keys: 'app_code', 'http_code'
- App_code is zero in case of success
- Code 200 is received upon success. 'message'(string) is present in case of an error.
- Other codes (4xx & 5xx) are returned upon various errors. The application also sends its own app code for some specific errors.

Users

The following endpoints are used to perform user-related actions.

Get Specific User – GET /users/{user_id}

Description	Gets the user with specificē id
Requirements	Need to be logged on Simple User can only get information about himself Administrator / Moderator can get info about any user
Successful Response	**message** : Request successful **user** : serialized json user data. Each record contains fields: • id, • email, • name, • role (string – "administrator" or "moderator" or "user"), • daily_calories (int) • reg_date (dd.mm.YYYY format, days/months with leading zeroes)

Get User List – GET /users/{amount}/{page}

Description	Gets the user list with defined amount (cannot exceed maximum – 100), page Is used to go through the whole list. Pages start from zero. I.e. GET /users/100/3 gets users #301-400
Requirements	User needs to be logged on Role should either be moderator or administrator
Successful Response	**message** : Request successful **page(int):** Current Page **amount(int):** Amount Retrieved **last(string literal):** ('true' or 'false') – is this a last page? **users** : serialized json array with user data containing fields: • id, • email, • name, • role (string – "administrator" or "moderator" or "user"), • daily_calories (int) • reg_date (dd.mm.YYYY format, days/months with leading zeroes)

Create a New User – POST /users/

Description	Creates a new user. User permission role is set to 'User'
Requirements	For parameters: Unique email should be provided Name must be longer than 1 character Password length must be at least 6 characters
POST Parameters	**'email', 'password', 'name'**
Successful Response	**message** : User Created

Authenticate a user – POST /users/auth/

Description	Authenticates already existing user
POST Parameters	**'email', 'password**
Successful Response	**message** : Authentication Successful **user** : serialized json user data containing fields: • id, • email, • name, • role (string – "administrator" or "moderator" or "user"), • daily_calories (int) • reg_date (dd.mm.YYYY format, days/months with leading zeroes)

Change User Details – PUT /users/{user_id}

Description	Edits existing user details
Requirements/Constraints	Needs to be authenticated Simple User can update name, calories and password Moderator can set those values for any user Administrator can also change other user roles and passwords
PUT Parameters	• **'name',** • **'daily_calories',** • **'password' (optional),** • **'role' (optional, admin only, values: 'user', 'moderator', 'administrator')**
Successful Response	**message** : User Data Change Successful **user** : serialized modified json user data containing fields: • id, • email, • name, • role (string – "administrator" or "moderator" or "user"), • daily_calories (int) • reg_date (dd.mm.YYYY format, days/months with leading zeroes)

Reset User Password – PUT /users/reset/

Description	Resets the password of already existing user. If user exists – a new 10-character password is generated and sent to the users email
POST Parameters	**'email'**
Successful Response	**message** : Password reset successful

Reset User Password – PUT /users/reset/

Description	Authenticates already existing user. If user exists – a new 10-character password is generated and sent to the users email
POST Parameters	**'email'**
Successful Response	**message** : Password reset successful

Delete User – DELETE /users/{user_id}

Description	Deletes a specified user from the system
Requirements	Simple users can only delete themselves Moderators can delete any simple user Administrators can delete any user
Successful Response	**message** : Password reset successful

Meals

To make actions with meals – user authentication is necessary.

POST	/meals/{owner_id}	Create a new meal for User with {owner_id}
GET	/meals/{meal_id}	Get Specific Meal
GET	/meals/list/{owner_id}	Get meals of User with {owner_id}
GET	/meals/filter/{owner_id}	Get meals filtered by parameters
DELETE	/meals/{meal_id}	Delete meal with specificē id
PUT	/meals/{meal_id}	Update specificē meal

Create a New Meal – POST /meals/{owner_id}

Description	Creates a new meal for specified user
Requirements/constraints	Simple User can only create meals for himself Moderators and administrators can create meals for any user Date cannot exceed todays date
POST Parameters	**'text'(optional)** – meal name or comment, up to 255 chars **'date'(string)** – format 'dd.mm.YYYY' (with leading zeroes). i.e – 17.02.2013 **time'(string) –** time in 24hour format 'HH:mm' (with leading zeroes). Ie. 02:07, 15:48 **'calories'(int) –** calories, between 1 and 5000
Successful Response	**message** : Meal Added **meal:** serialized json array with meal data: 'owner' (owner id), 'text' (meal comment/name), 'date' (date in dd.mm.YYYY format), 'time' (time in HH:mm format), 'calories' – calories returned • 'owner' (owner id), • 'text' (meal comment/name), • 'date' (date in dd.mm.YYYY format), • 'time' (time in HH:mm format), • 'calories' – meal calories

Get a specific meal – GET /meals/{meal_id}

Description	Gets a specific meal
Requirements/constraints	Simple Users can only get information about their own meals Administrator / Moderator can get info about any meal
Successful Response	**meal:** serialized json array with meal data: 'owner' (owner id), 'text' (meal comment/name), 'date' (date in dd.mm.YYYY format), 'time' (time in HH:mm format), 'calories' – calories returned • 'owner' (owner id), • 'text' (meal comment/name), • 'date' (date in dd.mm.YYYY format), • 'time' (time in HH:mm format), • 'calories' – meal calories

Get Meal List – GET /meals/list/{owner_id}

Description	Gets the meal list of the specific user
GET Parameters	**'amount'(int)** – how many meals to retrieve with this request (max 100) **'page'(int)** – page is used to iterate through the list, starts with 0. i.e. &amount=100&page=1 gets meals #101-200
Requirements	Simple User can only get information about their own meals Administrator / Moderator can get info about any meal
Successful Response	**page(int):** Current Page **amount(int):** Amount Retrieved **last(string literal):** ('true' or 'false') – is this a last page? **meals:** serialized json array with meals. Each record contains data: • 'owner' (owner id), • 'text' (meal comment/name), • 'date' (date in dd.mm.YYYY format), • 'time' (time in HH:mm format), • 'calories' – calories returned

Get Meals by Filter– GET /meals/filter/{owner_id}

Description	Gets the meals within defined criteria.
Restrictions	startdate should be lesser or equal to enddate starttime should be lesser or equal to endtime you can only request 2-month data max
GET Parameters	**startdate(string) –** starting date (inclusive) from which meals will be selected, in dd.mm.YYYY format **enddate(string) –** final date (inclusive) from which meals will be selected, in dd.mm.YYYY format **starttime(string, optional)** – starting time from which meals will be selected, in HH:mm format, hours and minutes leading by zero **endtime(string, optional) –** end time to which meals will be selected, in HH:mm format, hours and minutes leading by zero
Requirements	Simple Users can only get information about their own meals Administrator / Moderator can get info about any meal
Successful Response	**from(string):** startdate **to(int):** enddate **last(string literal):** ('true' or 'false') – is this a last page? **meals:** serialized json array with meals. Each record contains data: • 'owner' (owner id), • 'text' (meal comment/name), • 'date' (date in dd.mm.YYYY format), • 'time' (time in HH:mm format), • 'calories' – calories returned

Change Meal Details – PUT /meals/{meal_id}

Description	Edits existing meal details
Requirements/Constraints	Simple Users can update their own meals only Moderator and Administrator can update any meals
PUT Parameters	**'text'(optional)** – meal name or comment, up to 255 chars **'date'(string)** – format 'dd.mm.YYYY' (with leading zeroes). i.e – 17.02.2013 **time'(string) –** time in 24hour format 'HH:mm' (with leading zeroes). Ie. 02:07, 15:48 **'calories'(int) –** calories, between 1 and 5000
Successful Response	**message** : User Data Change Successful **user** : serialized modified json user data containing fields: • id, • email, • name, • role (string – "administrator" or "moderator" or "user"), • daily_calories (int) • reg_date (dd.mm.YYYY format, days/months with leading zeroes)

Delete Specific Meal – DELETE /meals/{meal_id}

Description	Deletes a specified meal from the system
Requirements	Simple users can only delete their own meals Moderators and administrators can delete any meal
Successful Response	**message** : Meal Deleted

ENDPOINT ERROR CODES

Most of the endpoints give back their own app error codes, so that error handling can be better automated.

Generic

Those error codes can come from any endpoint and are explained below.

Error Code Value	Explanation
0	Operation Successful
1	Wrong Arguments Passed in URI or Request
3	Internal DB Error (server-side request failed)
4	Auth Expired – need to Authenticate again
5	User is not authenticated
6	Auth Wrong (wrong authentication token given0
7	User Not Found
8	Forbidden operation (usually when user does not have rights to access endpoint with his parameters)
9	Missing Parameter in the request
10	Extra arguments passed in URI
11	Mail could not be sent due to internal error

User Endpoint Codes

Those errors can be returned when calling to /users/ endpoints

Error Code Value	Explanation
-100	Email Is Missing From the Request (on registration / password reset)
-101	Given Name parameter is too short (on registration/update)
-102	Bad Password (password does not match on login)
-103	User with this Email already exists (on registration)
-104	Bad Page requestes (in user list)
-105	Bad amount requested (in user list)

Meal Endpoint Codes

Error Code Value	Explanation
-200	Wrong Calorie Amount Given (on creation/update)
-201	Meal comment too long (on creation/update) – Limit is 255 characters
-202	Bad date given (on filter / creation / update)
-204	Bad Time given (on filter / creation / update)
-205	Bad amount given on meal list
-206	Meal not found
-207	Too much data requested (filter can request up to two month data)