

Final report

Micro service 2



Team 301

**Kay Erika Findo
01263502**

A discussion of how you eventually had to deviate from your original plan

The missing parts were implemented as planned, including here create items, delete items and also update items and lists and the connection with the databases was established successfully. The implementation is correctly done and every use case is completed as required. The interaction with the MySQL database gives back the correct information and it also delivers the expected results. The communication between microservices works perfectly using the Cross-Origin Resource, which is the only deviation from the plan.

A description of your service's interface.

1. Technology stack

- **Npm**
Npm is the package manager for JavaScript and the world's largest software registry. It is the default package manager for the JavaScript runtime environment Node.js.
- **NodeJs**
I decided to use NodeJs as a server, because it builds fast, scalable network applications and easy to start. Compared to java servers like Tomcat or Glassfish you do not need to deploy your project. I use hot deployment, which takes less than 5 seconds to restart a server.
- **Webpack**
For controlling the packages of NodeJs server, I decided to use webpack manager. Main file is package.json, which contains information about microservices, plugins, versions of plugins and other dependencies.
- **Plugins:**
 - a. **Mysql.**
Native Mysql Client for NodeJs, which realizes the communication between application and mysql database.
 - b. **Express.**
Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It allows to set up middlewares to respond to HTTP Requests and defines a routing table which is used to perform different actions based on HTTP Method and URL.

2. Server Configuration

In order to make reliable rest API some configurations shall be done. The use of the Access-Control-Allow-Origin opens a door for cross-origin access by specific requesting origins. The application can be accessed by any domain.

```
app.use(function (req, res, next) {
```

```

res.setHeader('Content-Type', 'application/json');
res.setHeader('Access-Control-Allow-Origin', '*');
res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');
res.setHeader('Access-Control-Allow-Credentials', true);
next();
});

```

The port 8899 is also opened for HTTP requests.

```
app.listen(8899);
```

3. Database Architecture Description

I have first created a database (a1263502) in MySQL

<https://www.univie.ac.at/phpmyadmin02/> offered by the Universita of Vienna.

Database host: a1263502.mysql.univie.ac.at

Username: a1263502

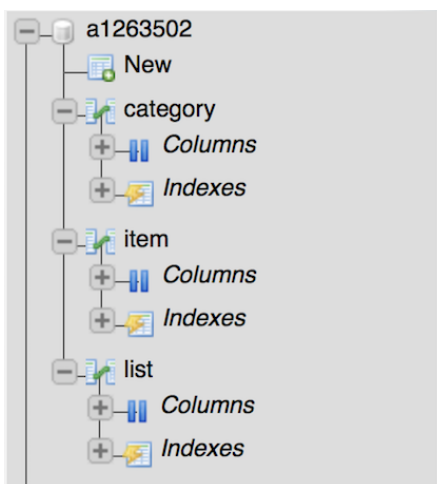
Password: kay2devi

```

let connection = mysql.createConnection({
  host: 'a1263502.mysql.univie.ac.at',
  user: 'a1263502',
  password: 'kay2devi',
  database: 'a1263502',
  multipleStatements: true
});

```

Setup connection in nodejs



The database contains 3 Tables: Category, Item and List. Category has an auto-incremented ID and name (Important, not Important, not very Important). On the other hand Item has these attributes: id also auto-incremented, name, done and list_id as a foreign key. All the to-do items have a description like *attend the VO at 16.00*, which is the name of this attribute and done shows if this to-do item is already accomplished or not. Lists must have a certain category regarding their importance, which is referenced through the category_id as foreign key and the items contained in a list through the list_id mentioned before. Table List also has an auto incremented id, user_id, name, created.

Category:

| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|--------------------------|------|------|--------------|-----------------|------|---------|----------|----------------|--|
| <input type="checkbox"/> | 1 | id | int(11) | UNSIGNED | No | None | | AUTO_INCREMENT | Change Drop Primary Unique Index Spatial Fulltext More |
| <input type="checkbox"/> | 2 | name | varchar(400) | utf8_general_ci | No | | | | Change Drop Primary Unique Index Spatial Fulltext More |

Item:

| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|--------------------------|------|---------|--------------|-----------------|------|---------|----------|----------------|---|
| <input type="checkbox"/> | 1 | id | int(11) | UNSIGNED | No | None | | AUTO_INCREMENT | Change Drop Primary Unique Index Spatial Fulltext |
| <input type="checkbox"/> | 2 | name | varchar(500) | utf8_general_ci | No | None | | | Change Drop Primary Unique Index Spatial Fulltext |
| <input type="checkbox"/> | 3 | done | tinyint(1) | | No | None | | | Change Drop Primary Unique Index Spatial Fulltext |
| <input type="checkbox"/> | 4 | list_id | int(11) | UNSIGNED | No | None | | | Change Drop Primary Unique Index Spatial Fulltext |

List:

| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|--------------------------|------|-------------|--------------|-----------------|------|---------|----------|----------------|---|
| <input type="checkbox"/> | 1 | id | int(11) | UNSIGNED | No | None | | AUTO_INCREMENT | Change Drop Primary Unique Index Spatial |
| <input type="checkbox"/> | 2 | user_id | int(11) | | No | None | | | Change Drop Primary Unique Index Spatial |
| <input type="checkbox"/> | 3 | name | varchar(300) | utf8_general_ci | No | None | | | Change Drop Primary Unique Index Spatial |
| <input type="checkbox"/> | 4 | created | date | | No | None | | | Change Drop Primary Unique Index Spatial |
| <input type="checkbox"/> | 5 | category_id | int(10) | UNSIGNED | No | None | | | Change Drop Primary Unique Index Spatial |

4. Rest API Interface Description

- **Create list for user**

Following rest call will create a list for particular user. Every List must have a specific category (important, not important, very important).

Body parameters:

User_id: integer

Name: string

Created: (not needed, will be created automatically)

Category_id: integer

Example call:

POST: /list/create

localhost:8899/api/list

POST

localhost:8899/api/list/create

Authorization Headers (1) **Body** Pre-request Script Tests

☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary **JSON (application/json)**

1

2 {

3 "name": "Testee1",

4 "user_id": 1,

5 "category_id": 2

6 }

7

Content-Type: application/json

Response: HTTP RESPONSE 200

- **Create category**

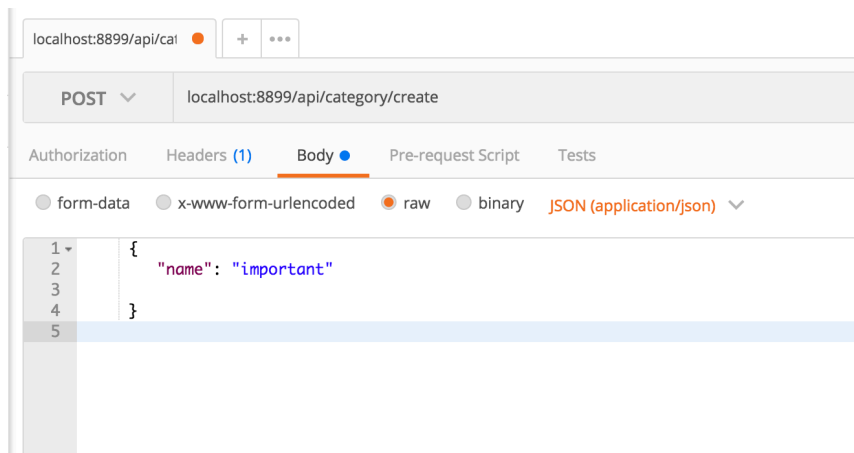
A category can be important, not important or very important. There will only be these 3 categories available and each list will be part of a specific category.

Body parameters:

Name: string

Example call:

POST: category/create



Content-Type: application/json

Response: HTTP RESPONSE 200

- **Create item**

Following rest call will create a item for a particular list. A list consists of one or many items.

Body parameters:

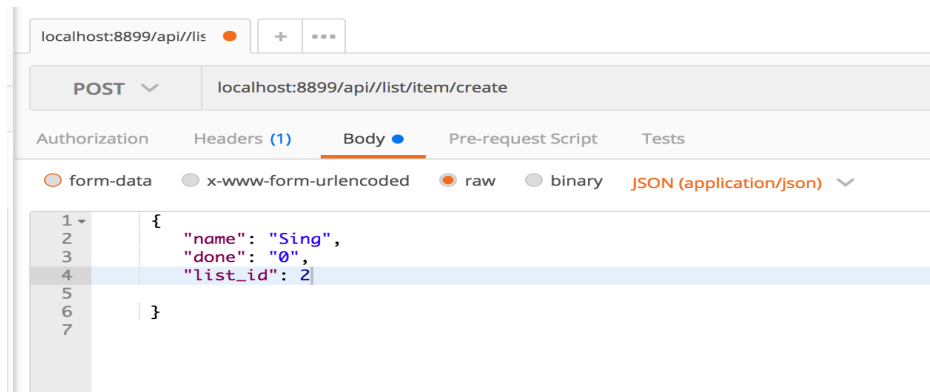
Name: string

Done: integer (0-false, 1-true)

List_id: integer

Example call:

POST: /list/item/create



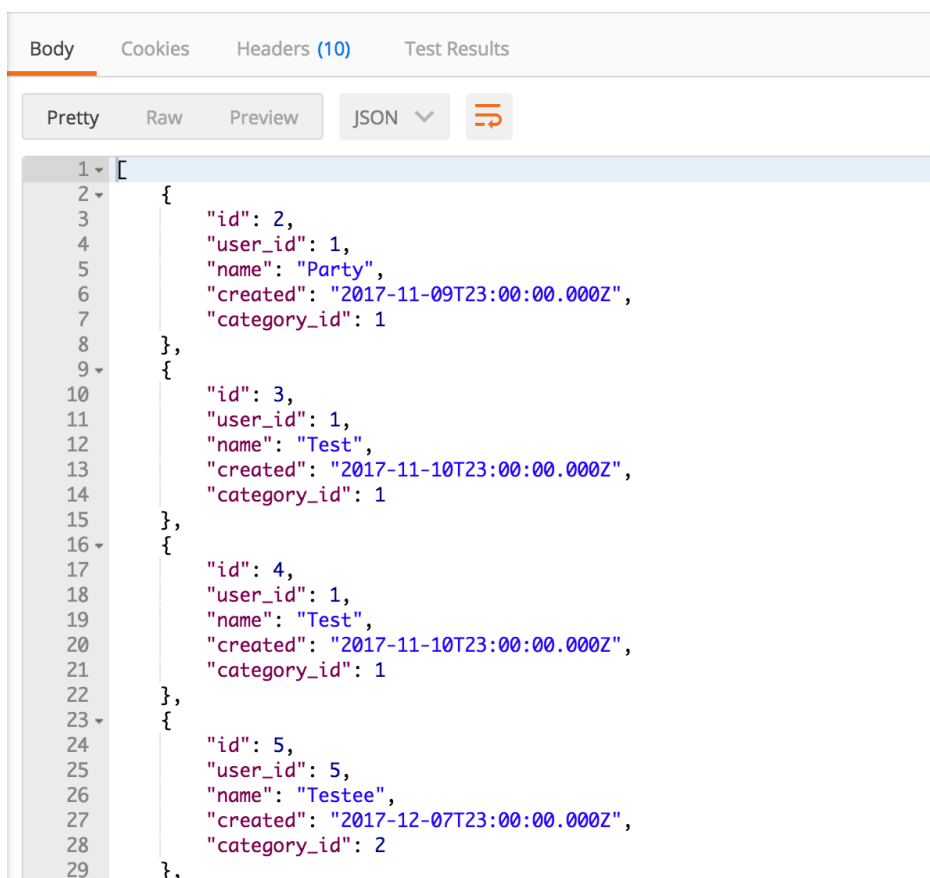
Content-Type: application/json
Response: HTTP RESPONSE 200

- **Show all lists**

Following rest call shows us all the lists in the database.

Example call:

GET: /lists/



Content-Type: application/json
Response: HTTP RESPONSE 200

- **Show all lists of a category**

Following rest call shows all lists of a category.

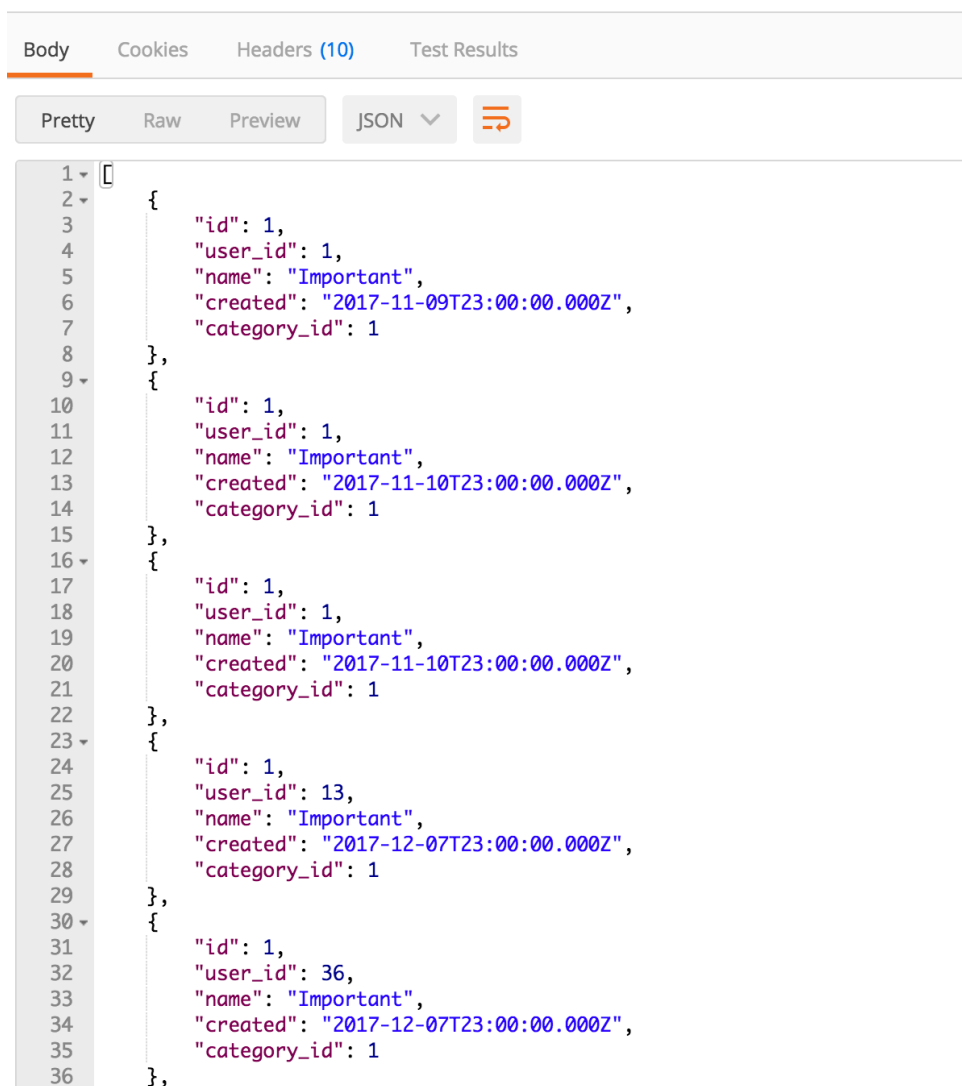
Parameters:

Category_id: integer

Example call:

GET: lists/category/1

In this example are shown all the lists of the category important.



The screenshot shows a REST client interface with the following components:

- Top tabs: Body, Cookies, Headers (10), Test Results. The 'Body' tab is selected.
- Below the tabs: A row of buttons for 'Pretty', 'Raw', 'Preview', and a dropdown menu set to 'JSON'. To the right is a refresh icon.
- The main area displays a JSON array of four objects, formatted with line numbers 1 through 36 on the left margin.

```
1 {
2   {
3     "id": 1,
4     "user_id": 1,
5     "name": "Important",
6     "created": "2017-11-09T23:00:00.000Z",
7     "category_id": 1
8   },
9   {
10    "id": 1,
11    "user_id": 1,
12    "name": "Important",
13    "created": "2017-11-10T23:00:00.000Z",
14    "category_id": 1
15  },
16  {
17    "id": 1,
18    "user_id": 1,
19    "name": "Important",
20    "created": "2017-11-10T23:00:00.000Z",
21    "category_id": 1
22  },
23  {
24    "id": 1,
25    "user_id": 13,
26    "name": "Important",
27    "created": "2017-12-07T23:00:00.000Z",
28    "category_id": 1
29  },
30  {
31    "id": 1,
32    "user_id": 36,
33    "name": "Important",
34    "created": "2017-12-07T23:00:00.000Z",
35    "category_id": 1
36  },
37 }
```

Content-Type: application/json

Response: HTTP RESPONSE 200

- **Show all lists of an user**

Following rest call shows all lists of an user.

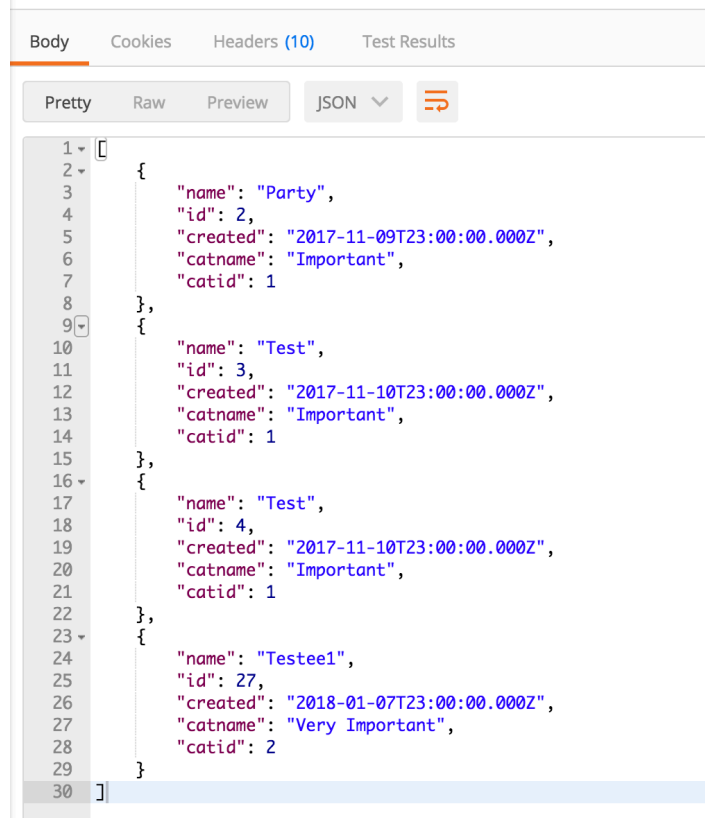
Parameters:

User_id: integer

Example call:

GET: lists/user/1

In this example are shown all the lists of the user with the id 1.



Content-Type: application/json

Response: HTTP RESPONSE 200

- **Show the list with the required id**

Following rest call shows the list depending on the list Id.

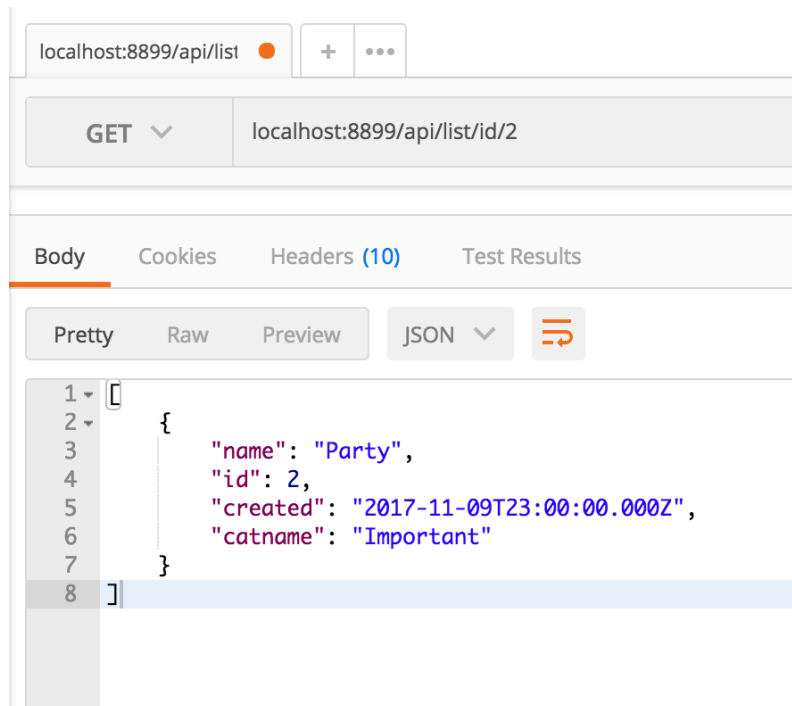
Parameters:

List_id: integer

Example call:

GET: lists/id/2

In these example is displayed the list with the id 2.



Content-Type: application/json

Response: HTTP RESPONSE 200

- **Show the list with the required name**

Following rest call shows the list/s depending on the list name.

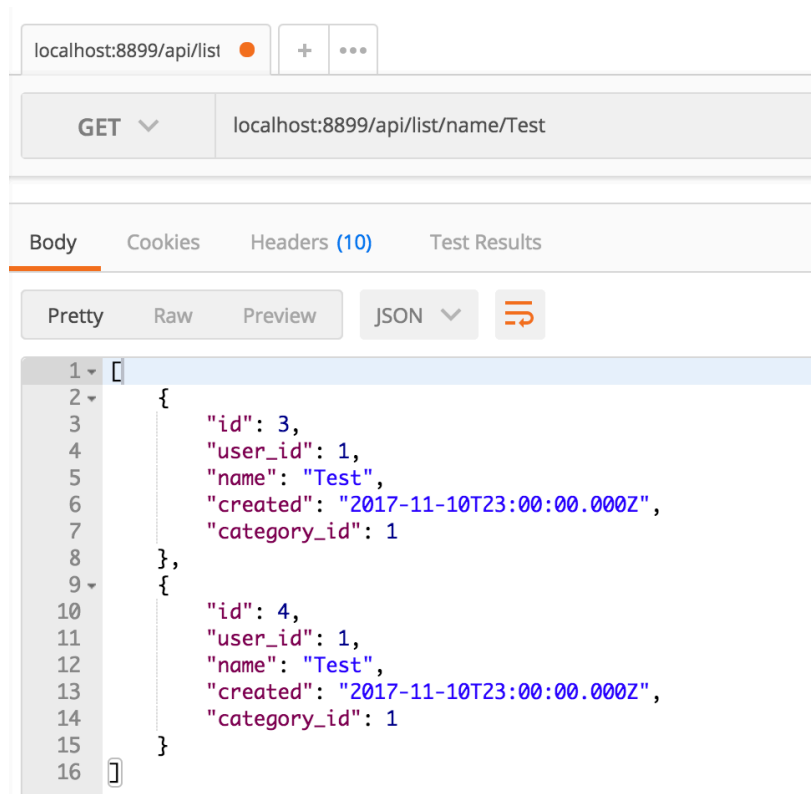
Parameters:

List_name: string

Example call:

GET: lists/name/Test

In these example is displayed the list with the name Test.



Content-Type: application/json
Response: HTTP RESPONSE 200

- **Show all categories**

Following rest call shows all the categories in the database.

Example call:

GET: /categories



Content-Type: application/json
Response: HTTP RESPONSE 200

- **Show items by list id**

Following rest call shows all the items that belong to a list.

Parameters:

List_id: integer

Example call:

GET: /list/items/2

In this example are displayed the items, which belong to the list with the id 2.



Content-Type: application/json

Response: HTTP RESPONSE 200

- **Update item**

Following rest call allows us to update an item's name, list id, the attribute completed which gives us information if the to do item done is or not.

Parameter:

Item_id: integer

Body parameters:

Name: string

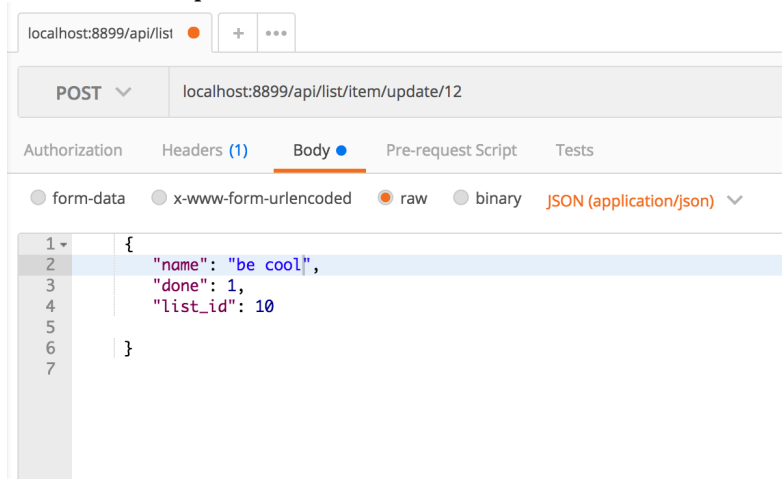
Done: integer

List_id: integer

Example call:

POST: /list/item/update/12

In this example the item's attributes (name, done, list id) with item id 12 are updated with the new parameters.



Content-Type: application/json

Response: HTTP RESPONSE 200

- **Update list name**

Following rest call allows us to update a required list's name.

Parameter:

List_id: integer

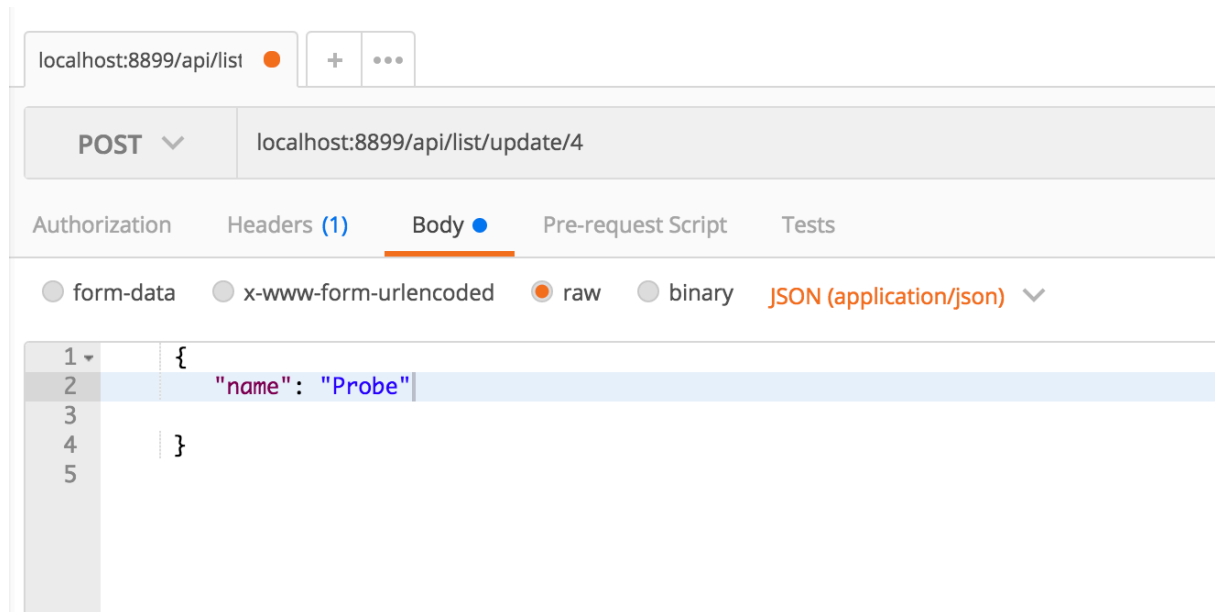
Body parameters:

Name: string

Example call:

POST: /list/update/4

In this example the list's name with id 4 is updated with the new parameter.



Content-Type: application/json
Response: HTTP RESPONSE 200

- **Delete item from list**

Following rest call allows us to delete a item of the list.

Parameters:

Item_id: integer

Example call:

DELETE: /list/item/delete/11

In this example the item with id 11, buy stuff will be deleted.

Before:

| | | | | | | | | | | |
|--------------------------|--|------|--|------|--|--------|----|-----------|---|----|
| <input type="checkbox"/> | | Edit | | Copy | | Delete | 11 | buy stuff | 0 | 13 |
| <input type="checkbox"/> | | Edit | | Copy | | Delete | 12 | be cool | 1 | 10 |
| <input type="checkbox"/> | | Edit | | Copy | | Delete | 14 | bla bla | 0 | 7 |

After:

| | | | | | | | | | | |
|--------------------------|---|----------------------|---|----------------------|---|------------------------|----|---------------|---|----|
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 12 | be cool | 1 | 10 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 14 | bla bla | 0 | 7 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 19 | Omilab Abgabe | 1 | 15 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 20 | Das del... | 1 | 15 |

Content-Type: application/json
Response: HTTP RESPONSE 200

- **Delete list**

Following rest call allows us to delete a list with the belonging items.

Parameters:
















List_id: integer

Example call:













DELETE: /list/delete/11

In this example the list with id 11, Important Test will be deleted.

Before:

| | | | | | | | | | | | |
|--------------------------|---|----------------------|---|----------------------|---|------------------------|----|----|----------------|------------|---|
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 10 | 13 | be very smart | 2017-12-08 | 2 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 11 | 13 | important test | 2017-12-08 | 1 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 12 | 13 | but bue bue | 2017-12-08 | 3 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 13 | 36 | be pretty | 2017-12-08 | 1 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 15 | 38 | UNI-Stuff | 2017-12-12 | 2 |

After:

| | | | | | | | | | | | |
|--------------------------|---|----------------------|---|----------------------|---|------------------------|----|----|---------------|------------|---|
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 10 | 13 | be very smart | 2017-12-08 | 2 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 12 | 13 | but bue bue | 2017-12-08 | 3 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 13 | 36 | be pretty | 2017-12-08 | 1 |
| <input type="checkbox"/> |  | Edit |  | Copy |  | Delete | 15 | 38 | UNI-Stuff | 2017-12-12 | 2 |

Content-Type: application/json
Response: HTTP RESPONSE 200

A “quick start tutorial” that describes how one can use your service.

Pre Requirements:

1. Npm version 5.5.1 or higher
2. NodeJs version 8.9.1 or higher
3. BigIpEdge connected or be under University of Vienna network
4. TunnelBlick to make application server accessible via VPN to other users.

Installation:

1. Clone MS2 git project
2. Type in terminal: “npm instal” and wait until packages and other dependencies will be installed and stored in node_modules folder.
3. Execute command “node server.js”. Server will start and mysql client will be connected with the database. Don’t close a terminal window with a nodejs server.

Usage:

Now server is deployed on port 8899 and you can test it locally. To perform test you need to open Postman (or browser) and make simple GET Request to check if everything ok.

`localhost:8899/api/lists`

You will get a list of lists.

If everything works you can start to test other API endpoints in order to check if it works.