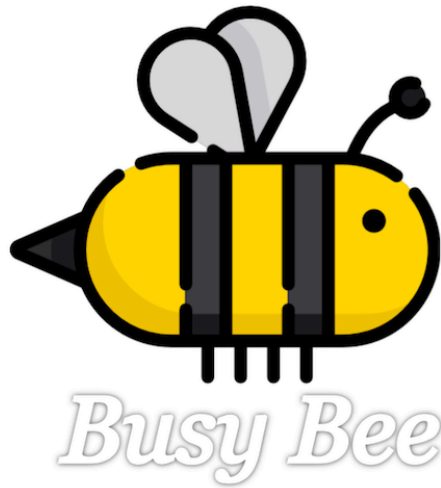


Student ID: 01263502
Name: Kay Erika Findo
Team: 301

Microservice 2



Core Design of the Microservice

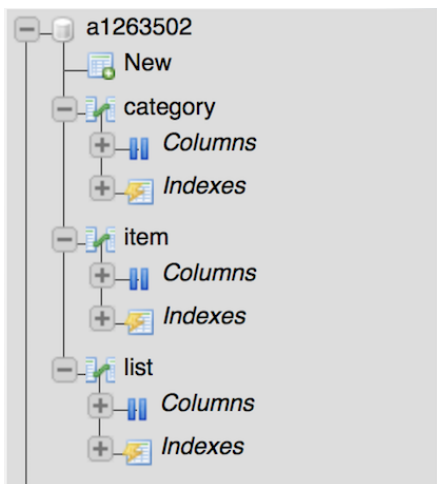
- How does the service's external interface look like? How can your colleagues use your service? Which protocol(s) and technologies are used?

MS2- To-do-List-Service is implemented using the programming language java script. This service provides the ability to create, remove and update To-Do lists. 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



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 More
<input type="checkbox"/>	2 name	varchar(500)	utf8_general_ci		No				Change Drop Primary Unique Index Spatial Fulltext More
<input type="checkbox"/>	3 done	tinyint(1)			No	None			Change Drop Primary Unique Index Spatial Fulltext More
<input type="checkbox"/>	4 list_id	int(11)		UNSIGNED	No	None			Change Drop Primary Unique Index Spatial Fulltext More

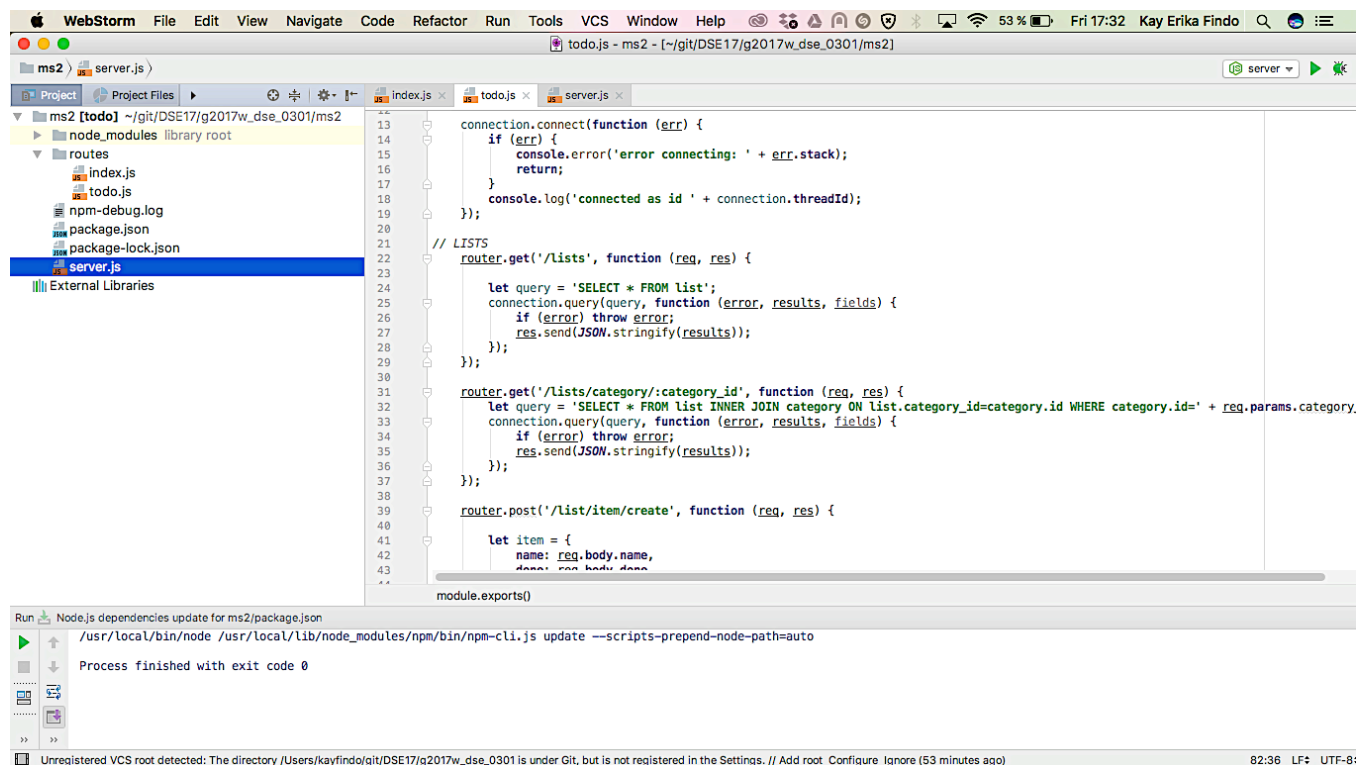
List:

#	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 More
<input type="checkbox"/>	2 user_id	int(11)			No	None			Change Drop Primary Unique Index Spatial More
<input type="checkbox"/>	3 name	varchar(300)	utf8_general_ci		No				Change Drop Primary Unique Index Spatial More
<input type="checkbox"/>	4 created	date			No	None			Change Drop Primary Unique Index Spatial More
<input type="checkbox"/>	5 category_id	int(10)		UNSIGNED	No	None			Change Drop Primary Unique Index Spatial More

MS2 consists of a package called routes, which has two important classes index.js and todo.js. In todo.js are implemented all the functionalities required:

- Show all lists
- Show lists by category
- Create item
- Delete item by item_id
- Update item
- Show all categories
- Show items by list_id

Server.js realizes the connection with the server through HTTP protocol. All the functionalities above are tested through PostMan API, in order to check if the request or the connection with the database is successfully accomplished. The data are presented in PostMan in JSON-Format.



```
13 connection.connect(function (err) {
14   if (err) {
15     console.error('error connecting: ' + err.stack);
16     return;
17   }
18   console.log('connected as id ' + connection.threadId);
19 });
20
21 // LISTS
22 router.get('/Lists', function (req, res) {
23
24   let query = 'SELECT * FROM list';
25   connection.query(query, function (error, results, fields) {
26     if (error) throw error;
27     res.send(JSON.stringify(results));
28   });
29 });
30
31 router.get('/Lists/category:category_id', function (req, res) {
32   let query = 'SELECT * FROM list INNER JOIN category ON list.category_id=category.id WHERE category.id=' + req.params.category_id;
33   connection.query(query, function (error, results, fields) {
34     if (error) throw error;
35     res.send(JSON.stringify(results));
36   });
37 });
38
39 router.post('/List/item/create', function (req, res) {
40
41   let item = {
42     name: req.body.name,
43     data: req.body.data
44   };
45
46   module.exports()
```

Your thoughts on the service's deployment

- How are you deploying your Microservice? How can it be reached via the network (i.e. IP/hostname and port(s))?

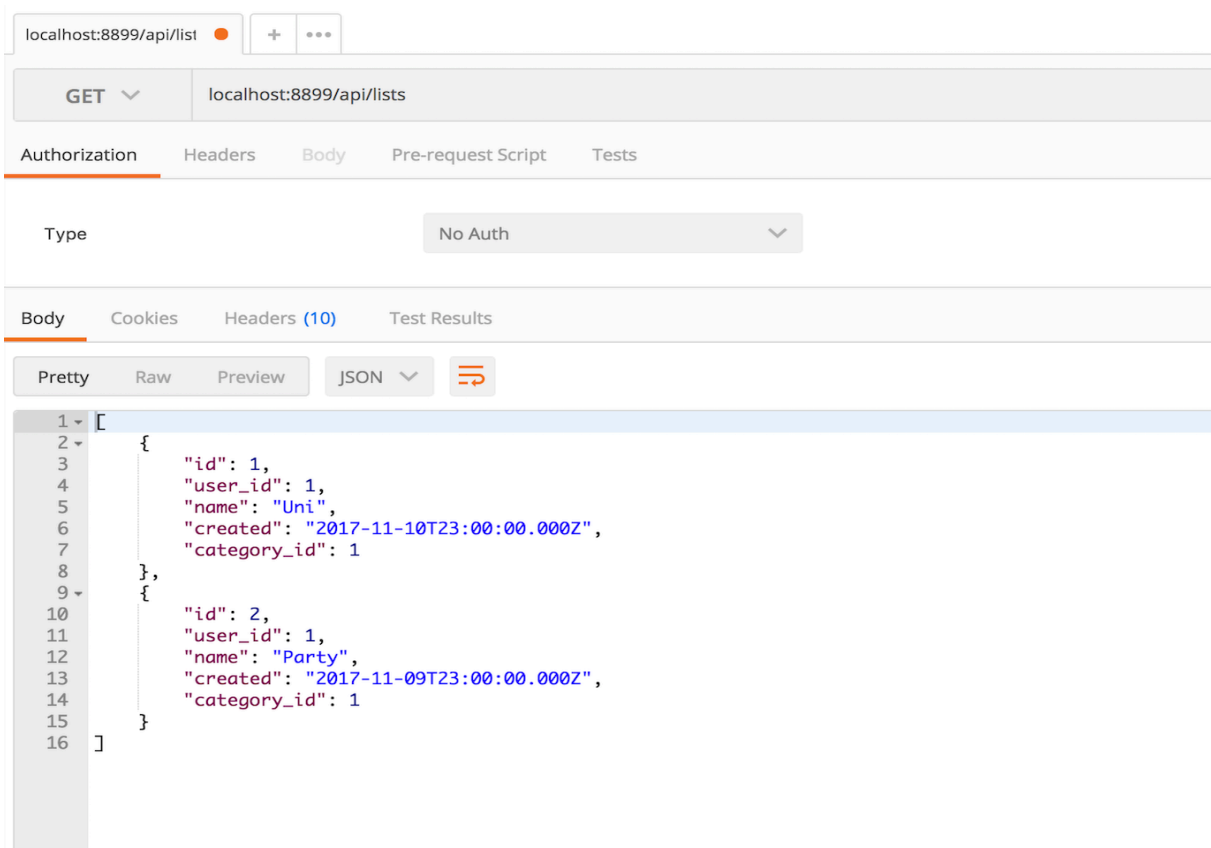
First of all modules and libraries must be installed through npm command line. npm is a package manager for the JavaScript programming language and the JavaScript runtime environment Node.js. It consists of a command line client, also called npm, and an online database of public and paid-for private packages, called the npm registry.

Ideas for testing and presentation:

1. How are you going to test your service when other services you have to depend upon are not available? How can you demonstrate the functionality of your service in such situations?

As mention before, all of the functionalities implemented are tested through PostMan API. This gives me the possibility of demonstrating my functionalities without needing the User Interface. Below I have tested and documented the functionalities which I have implemented.

- Show all the lists



- Show the lists by category id

localhost:8899/api/list

GET localhost:8899/api/lists/category/1

Authorization Headers Body Pre-request Script Tests

Type No Auth

Body Cookies Headers (10) Test Results

Pretty Raw Preview JSON

```
1 [
2   {
3     "id": 1,
4     "user_id": 1,
5     "name": "Important",
6     "created": "2017-11-10T23:00:00.000Z",
7     "category_id": 1
8   },
9   {
10    "id": 1,
11    "user_id": 1,
12    "name": "Important",
13    "created": "2017-11-09T23:00:00.000Z",
14    "category_id": 1
15  }
16 ]
```

- Create Item

localhost:8899/api/list

POST localhost:8899/api/list/item/create

Authorization Headers (1) Body Pre-request Script Tests

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

```
1 {
2   "name": "Buy a book",
3   "done": 0,
4   "list_id": 1
5 }
```

- Delete Item

localhost:8899/api/list + ...

DELETE ▼ localhost:8899/api/list/item/delete/4

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

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

1 ▼ {

- Update Item

POST ▼ localhost:8899/api/list/item/update/2

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

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

1 ▼ {
2 "name": "Find a DJ",
3 "done": 1,
4 "list_id": 2
5 }

- Show categories

localhost:8899/api/cat + ...

GET ▼ localhost:8899/api/categories

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

Type No Auth

Body Cookies Headers (10) Test Results

Pretty Raw Preview **JSON** ▼ ≡

1 ▼ [
2 {
3 "id": 1,
4 "name": "Important"
5 }
6]

The current state of your implementation:

- What has been achieved so far? What's missing? What works? What doesn't?

The implementation is not yet finished, but so far we can create items, delete them and also update them and the connection with the databases is established. The tests show that the implementation is correctly done but is still missing the option to create new lists, update and remove them. In the section above it is shown how the Use Cases work and the interaction with the MySQL database gives back the correct information and it also delivers the new items created.

Our focus till the next deadline is to see how the three ms are going to communicate with each other.

The connection with the OPEN VPN is also successfully established.

