Manual for Rest-Server

Johann Haag, Tarik Hosic und Simon Blaha

20.12.2018, Leonding

Project Name	Smart Organizer
Project Leader	Simon Blaha
Version	1.0
Document state	In process

Contents

1 Installation

1.1 Setting up the MongoDB

A MongoDB can be installed on the computer by default. It is also possible to operate the MongoDB via Docker. If you want to install MongoDB on your computer, you will see `https://www.mongodb.com` on the following page. If you want to use Docker, you will see `https://hub.docker.com/_/mongo` on the following page.

1.2 Installation of the required packages for the rest server

Change to the root directory of the remainder server and enter npm i in the terminal. Thus, all required packages are installed. For information about npm, see `https://www.npmjs.com/get-npm`. Node is used to operate the server, for information see url https://nodejs.org/

2 Configuration

In the server's /database/conifg.json file, database settings of the server can be changed. Here you can specify a different URI under dbURI if you do not use a localhost. The default installation always includes localhost and the default port for MongoDB. Furthermore, it is also possible to change the error outputs.

3 Commissioning of the rest server

There are several ways to start the server. On the one hand, they can switch to the root directory of the server via CMD and execute node index.js or You run the script run. However, this requires the scripting language Bash. During commissioning, the required collections are automatically created on the MongoDB. If everything starts up correctly, the server will listen on port 8080 and that a connection to MongoDB was made. The server can be easily terminated with STRG + C in the terminal.

4 Using the Servers

It is assumed that the server is running on localhost. If something goes wrong, it will always return a JSON object with the error and code attributes. Code has the value of the error code and the attribute error contains the error message. If a query succeeds, a JSON object with a result attribute is returned.

4.1 User registration

Join a new user on the server. To do this, a JSON object must be sent to the server by post request. The path for the post-request is `http://localhost:8080/user/register`. Structure of the JSON object: "username": "<username>", "password": "<password>" } The password is stored encrypted on the server.

4.2 User login

Login of a user on the server. When logging in, a token is created for the user. To do this, a JSON object must be sent to the server by post request. The path for the post-request is `http://localhost:8080/user/login`. Structure of the JSON object: "username": "<username>", "password": "<password>" } Upon successful login, a token is returned. The token is required as authorization for some functions on the server.

4.3 Logout

Logging out a user on the server. To do this, a JSON object must be sent to the server by post request. The path for the post-request is `http://localhost:8080/user/logout`. Structure of the JSON object: "username": "<username>", "token": "<token>" If successful, the user will no longer have a valid token.

4.4 Check if user exists

Check if a username is already taken. To do this, a JSON object must be sent to the server by post request. The path for the post-request is `http://localhost:8080/user/check`. Structure of the JSON object: "username": "<username>"

4.5 Create Appointment

Create an appointment for a user. To do this, a JSON object must be sent to the server by post request. The path for the post-request is `http://localhost:8080/appoimentnt/create`. Structure of the JSON object: "appointment": "username": "<username>", "date": "<date>", "duration": "<number>", "name": "< name> "}," token ":" <token> "} The token must be a valid token of the user whose username was used in the appointment.

4.6 Delete Appointment

Delete an appointment of a user. To do this, a JSON object must be sent to the server by post request. The path for the post-request is `http://localhost:8080/appoitnment/delete`. Structure of the JSON object: "username": "<username>", "token": "<token>", "appointmentid": "<appointmentid>" } The transferred username must be the owner of the appointment and the token must be valid.

4.7 Edit Appointment

Editing an appointment of a user. To do this, a JSON object must be sent to the server by post request. The path for the post request is `http://localhost:8080/appoitnment/edit`. Structure of the JSON object: "appointment": "username": "<username>" [, "date": "<date>"] [, "duration": "<number>"] [, "name ":" <name> "]," token ":" <token> " The attributes written in [] are optional!

4.8 Get all appointments

Query all appointments of a user. To do this, a JSON object must be sent to the server by post request. The path for the post-request is http://localhost:8080/appoitnment/get. Structure of the JSON object: "username": "<username>", "token": "<token>"

4.9 Create group

Create a group. To do this, a JSON object must be sent to the server by post request. The path for the post-request is `http://localhost:8080/group/create`.

Structure of the JSON object: "group": "owner": "<username>", "name": "<groupName>", "token": "<token>" The passed token must be a valid token for the given username.

4.10 Delete group

Create a group. To do this, a JSON object must be sent to the server by post request. The path for the post-request is `http://localhost:8080/group/delete`. Structure of the JSON object: "group": "owner": "<username>", "name": "<groupName>", "token": "<token>"

4.11 Get groups

All groups get the user for the owner. To do this, a JSON object must be sent to the server by post request. The path for the post-request is `http://localhost:8080/group/get`. Structure of the JSON object: "username": "<username>", "token": "<token>"