Information Engineering and Technology Faculty

German University in Cairo

NETW903: Network and Services

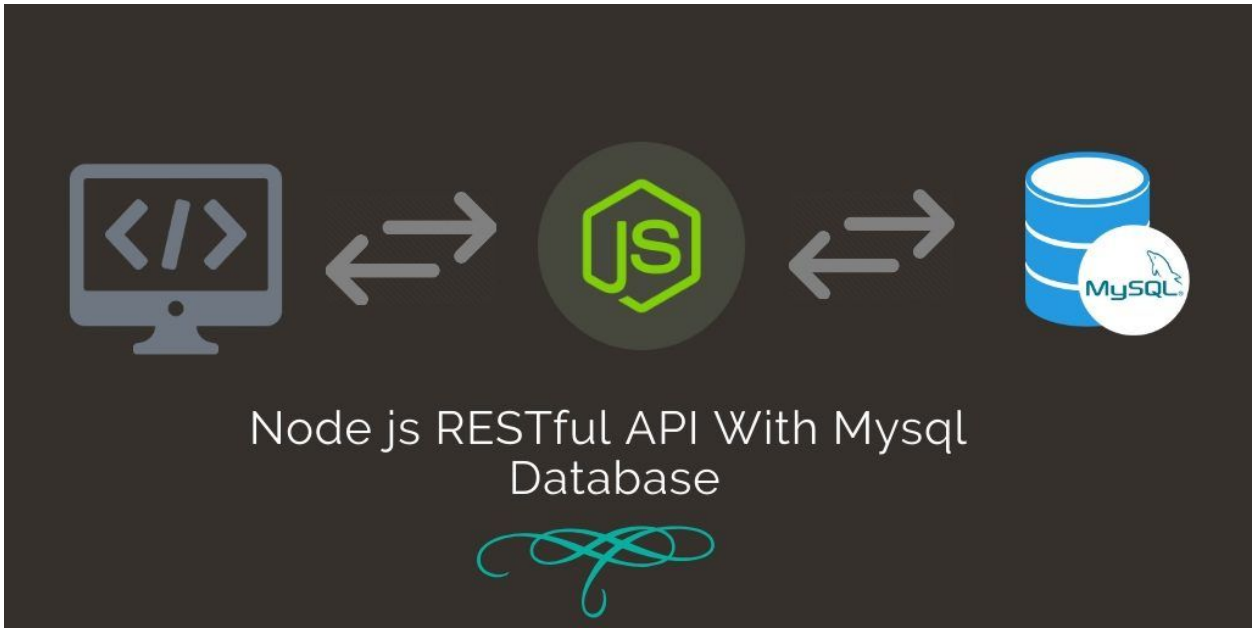
Project Report

Karim Adel Zohair 37-5409
Abdelhameed Emad 37-16451
Michael Masry 37-3657

Submitted to: Dr. Mohamed Abdel Wahab

Overview

In this project, we are implementing a to-do list application that uses RESTful web service to help end-users create, read, update and delete their tasks through a PC Python-based software. we used Express.js which is an open-source web application framework that provides a robust set of features for Node.js, integrated with MySQL database.



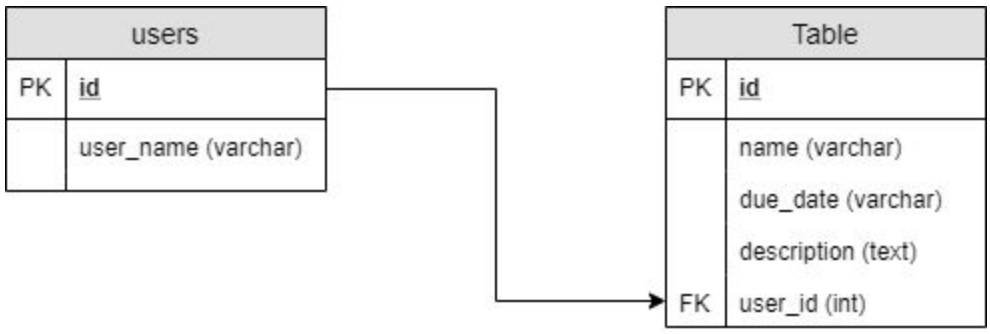
Implementation

1 - Deciding the features needed by the end-user in the application:

- User Sign up
- Create tasks.
- Read a single task.
- Read all of his tasks.
- Update tasks
- Delete tasks

Representing CRUD (Create, Read, Update, Delete) which are the four basic operations that any stored data is subjected to, The equivalent HTTP methods are POST, GET, PUT, DELETE which will be used by the client to communicate with the web service.

2 - Designing the database which will contain the users' information and the users' tasks, we used MySQL Database as mentioned before:



We didn't include user passwords in this testing database design to make testing a bit easier.

3 - Implementing the Express.js server:

First, the user signs up by sending an HTTP POST request to the server that contains the username desired, example request:

POST /signup?username=Michael
Host: 127.0.0.1:3000/

```
//Sign Up
app.post('/signup', (req, res) => {
    const user_name = req.query.username
    let user = {user_name:`${user_name}`};
    let sql = 'INSERT INTO users SET ?';
    let query = db.query(sql, user, (err, result) => {
        if(err){
            console.log(err)
            throw err;
        }
        console.log(result);
        res.send(JSON.stringify({ id: result.insertId }));
    });
});
```

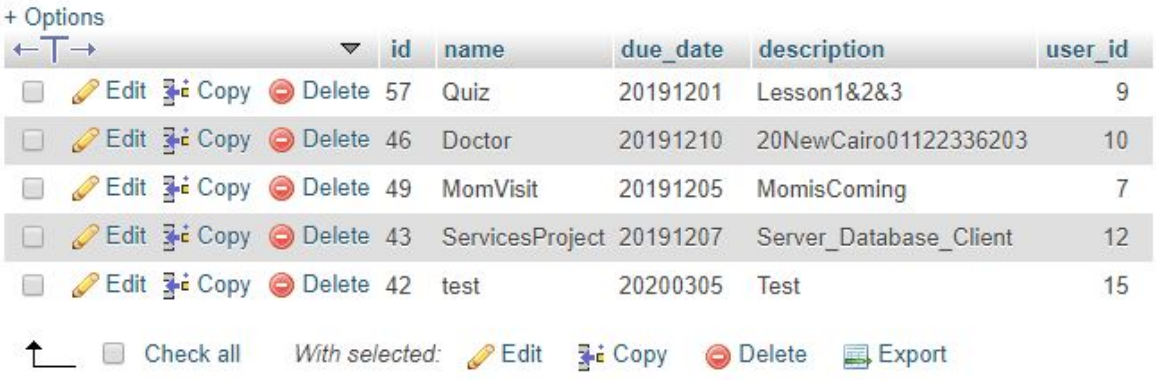
The server gets the username from the query, saves it in the table (users) in the database, and returns a JSON response contains the user id in the database, in this case, {“id”: 15} is returned to the user.

Second, the user can create a task by sending an HTTP POST request that contains all the info of the task, request example:

POST /addnewtask?name=test&date=20200305&description=Test&user_id=15
Host: 127.0.0.1:3000/

```
// Create task
app.post('/addnewtask', (req, res) => {
    const task_name = req.query.name
    const task_date = req.query.date
    const task_description = req.query.description
    const user_id = req.query.user_id
    let task = {name:`${task_name}`, due_date:`${task_date}`,
     description:`${task_description}`,user_id:`${user_id}`};
    let sql = 'INSERT INTO tasks SET ?';
    let query = db.query(sql, task, (err, result) => {
        if(err){
            console.log(err)
            throw err;
        }
        console.log(result);
        res.send(JSON.stringify({ id: result.insertId }));
    });
```

The server gets the task information from the query, saves it in the table (tasks) in the database, and returns a JSON response contains the task id in the database, in this case, {“id”: 42} is returned to the user.
Ps: we save the date in string format as follows YYYYMMDD



+ Options

	id	name	due_date	description	user_id
Edit Copy Delete	57	Quiz	20191201	Lesson1&2&3	9
Edit Copy Delete	46	Doctor	20191210	20NewCairo01122336203	10
Edit Copy Delete	49	MomVisit	20191205	MomisComing	7
Edit Copy Delete	43	ServicesProject	20191207	Server_Database_Client	12
Edit Copy Delete	42	test	20200305	Test	15

Check all *With selected:* Edit Copy Delete Export

Third, the user can read all his tasks by sending an HTTP GET request that contains the user's id, request example:

GET /allmytasks?user_id=15
Host: 127.0.0.1:3000/

```
// Get all the user's tasks
app.get('/allmytasks', (req, res) => {
    const id = req.query.user_id
    let sql = `SELECT name, due_date, description, id FROM tasks WHERE user_id=${id};`;
    let query = db.query(sql, (err, results) => {
        if(err){
            console.log(err)
            throw err;
        }
        console.log(results);
        res.send(JSON.stringify(results));
    });
});
```

The server gets the user id from the query, retrieves the information of all the tasks of this user id from the database, and sends it to the user in JSON format.

Edit Copy Delete	31	Go to the supermarket	20191225	get the ingredients for the lunch	8
Edit Copy Delete	32	Go to the library	20191230	Return the data analysis book	8
Edit Copy Delete	33	Visit your family	20200110	get some gifts with you from the mall	8


```
[{"name":"Go to the supermarket","due_date":"20191225","description":"get the ingredients for the
lunch","id":31},{"name":"Go to the library","due_date":"20191230","description":"Return the data analysis
book","id":32},{"name":"Visit your family","due_date":"20200110","description":"get some gifts with you from the
mall","id":33}]
```


The user can also retrieve a single task using an HTTP GET request that contains the task's id in the database, example:

GET /rvtask?id=31
Host: 127.0.0.1:3000/

```
// Select single task
app.get('/rvtask', (req, res) => {
    id = req.query.id
    let sql = `SELECT name, due_date, description FROM tasks WHERE id=${id}`;
    let query = db.query(sql, (err, result) => {
        if(err){
            console.log(err)
            throw err;
        }
        console.log(result);
        res.send(JSON.stringify(result));
    });
});
```

The server returns the task in JSON format

```
[{"name":"Go to the supermarket","due_date":"20191225","description":"get the ingredients for the lunch"}]
```

The user can update a task using an HTTP PUT request that contains the task's id in the query and the new name, due_date, or description in the body of the request (you can only update one field at a time), example:

PUT /updatetask?id=2
Host: 127.0.0.1:3000/

```
// Update post
app.put('/updatetask', (req, res) => {
    let newTitle = req.body.name;
    let due_date = req.body.due_date;
    let description = req.body.description;
    console.log(newTitle)
    console.log(due_date)
    console.log(description)
    if(!(req.body.name === "0")){
        console.log("I am at new title")
        let sql = `UPDATE tasks SET name = ? WHERE id = ${req.query.id}`;
        let query = db.query(sql, newTitle, (err, result) => {
            if(err) throw err;
            console.log(result);
            res.send('task updated...');
        });
    }
```

```
    else if (!(req.body.due_date === "0")) {
        console.log("I am at due date")
        let sql = `UPDATE tasks SET due_date = ? WHERE id = ${req.query.id}`;
        let query = db.query(sql, due_date,(err, result) => {
            if(err) throw err;
            console.log(result);
            res.send('task updated...');
        });
    }
    else if (!(req.body.description === "0")) {
        console.log("I am at description")
        let sql = `UPDATE tasks SET description = ? WHERE id = ${req.query.id}`;
        let query = db.query(sql, description,(err, result) => {
            if(err) throw err;
            console.log(result);
            res.send('task updated...');
        });
```

The server updates the database and returns “task updated”.

Finally, the user can delete tasks by sending an HTTP DELETE request that contains the task’s id in the database, example:

DELETE /deletetask?id=31
Host: 127.0.0.1:3000/

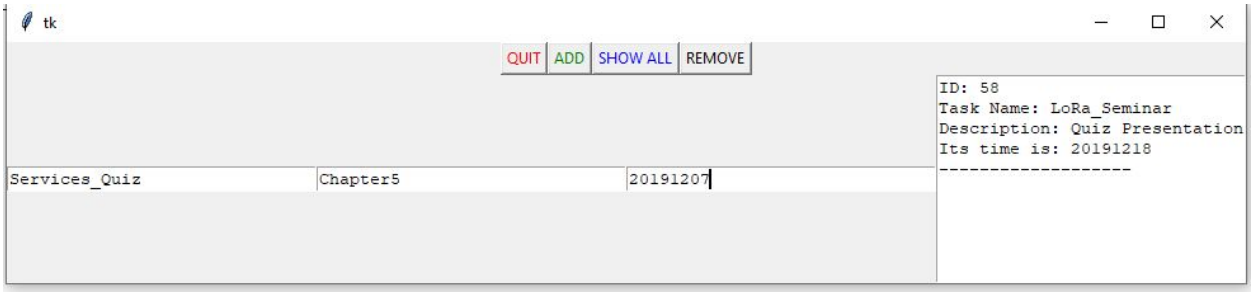
```
// Delete task
app.delete('/deletetask', (req, res) => {
    let sql = `DELETE FROM tasks WHERE id = ${req.query.id}`;
    let query = db.query(sql, (err, result) => {
        if(err){
            console.log(err)
            throw err;
        }
        console.log("done");
        res.send("done");
    });
});
```

The server will delete the task and reply by sending “done”.

PS: We are hosting the web service locally for testing.

4 - We made a simple Python GUI to test the web service.

When we click “SHOW ALL”, it sends a GET request to the server and displays all the tasks of this user, then we will add another task by clicking “ADD” which will send a POST request to the server to add the task.



After adding, we will delete the task “LoRa_Seminar” by writing its id in the first input space and clicking “REMOVE” which will send a DELETE request to the server.