**Alankrit Bhardwaj**

**Nurukurthi Prakash**

**MDITS ASSINGMENT**

For this assignment we have successfully tried to demonstrate how to build CRUD APIS using Node.js, express.js with MySql. And authentication was implemented with JWT.

**Graphical user interface, diagram

Description automatically generated**

The given picture describes how the connection between client and database is made through REST API. The client (**postman** in our case) uses the rest API (**Node.js**) to pass a query to database (**MySQl**) which in turn send the data to the REST API interface and then back to client.

Let’s see how to use Node.js to create CRUD APIs-

To create the complete the project we need to install certain libraries like:

1) **bcrypt-** It is used to encrypt the password.

Text

Description automatically generated

2) **jsonwebtoken(JWT)-** To generate a token so that user can be authenticated

Text

Description automatically generated

Writing the code for NODE JS:

Before going to the coding part let’s understand some of the basic terminology:

a) **Services** - The services contain the database queries and returning objects or throwing errors.

**b) Controllers** - The controllers handles all the logic behind validating request parameters, query, Sending Responses with correct codes.

c) **Routes** - The API routes maps to the Controllers.

Let’s start with connection between mysql and Node.js

Step1: Create a file with database.js.

Step2: Write the following code to connect with the mysql database present on your device-

Text

Description automatically generated

Let’s make the service file for our application:

The following lines of code will insert data inside our database(**test**) inside table **registration**.

Text

Description automatically generated

The **module.exports** is a special object which is included in every JavaScript file in the Node.js application by default. The module is a variable that represents the current module, and exports is an object that will be exposed as a module. So, whatever you assign to **module.exports** will be exposed as a module.

**Pool.query** will allow you to execute a basic single query when you need to execute from a client that would be accessed from the pool of client threads. So pool.query could be used to directly run the query rather than acquiring a client and then running the query with that client. It takes in 3 parameter-

a) String query which needs to be run on the mysql server.

b) An array of value which needs to be passed through the query.

c) It contains a function to return an error or and result from the database.

After creating all the example api the file will look like this-

const pool=require("../../config/database");

module.exports={

create:(data,callBack)=>{

pool.query(

'insert into registration(id,firstName,lastName,gender,email,password,number) values(?,?,?,?,?,?,?)',

[data.id,

data.firstName,

data.lastName,

data.gender,

data.email,

data.password,

data.number],

(error,results,fields)=>{

if(error){

return callBack(error);

}

return callBack(null,results);

}

);

},

getUsers:callBack=>{

pool.query(

'select id,firstName,lastName,gender,email,password,number from registration',

[],

(error,results,fields)=>{

if(error){

return callBack(error);

}

return callBack(null,results);

}

);

},

getUsersbyUserId:(id,callBack)=>{

pool.query(

'select id,firstName,lastName,gender,email,number from registration where id=?',

[id],

(error,results,fields)=>{

if(error){

return callBack(error);

}

return callBack(null,results[0]);

}

);

},

updateUserById:(data,callBack)=>{

pool.query(

'update registration set firstName=?,lastName=?,gender=?,email=?,password=?,number=? where id=?',

[

data.firstName,

data.lastName,

data.gender,

data.email,

data.password,

data.number,data.id],

(error,results,fields)=>{

if(error){

return callBack(error);

}

return callBack(null,results);

}

);

},

deleteByUserId:(data,callBack)=>{

pool.query(

'delete from registration where id=?',

[data.id],

(error,results,fields)=>{

if(error){

return callBack(error);

}

return callBack(null,results);

}

);

},

getUserByUserEmail: (email, callBack) => {

pool.query(

`select \* from registration where email = ?`,

[email],

(error, results, fields) => {

if (error) {

callBack(error);

}

return callBack(null, results[0]);

}

);

}

}

Now let’s create the controller class which will run a function on the data received from the database through service class-

Text

Description automatically generated