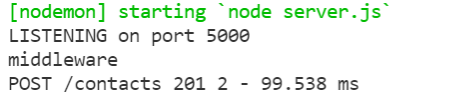
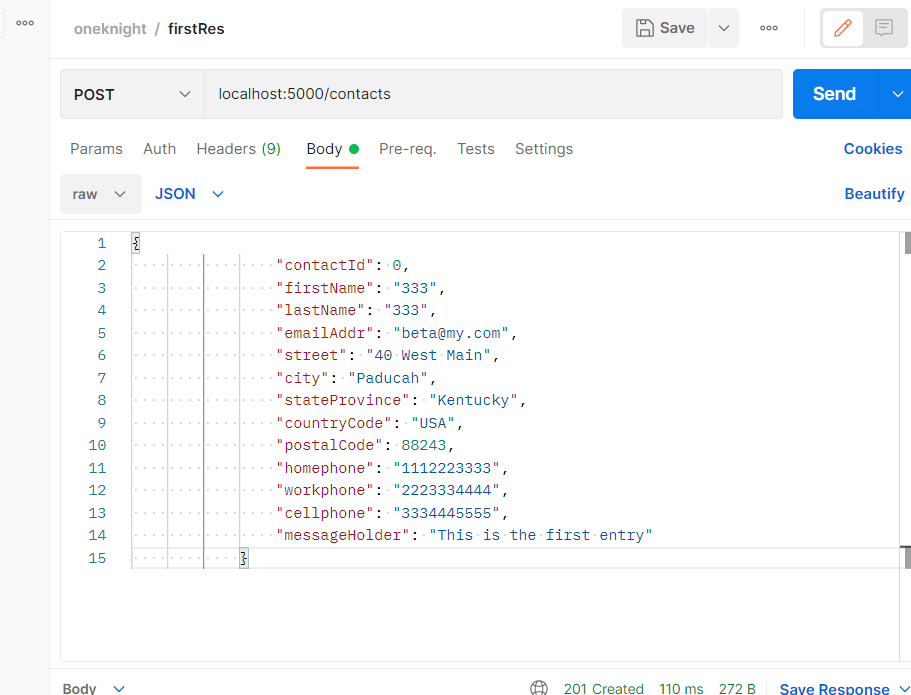
This objective of this lesson is to POST data to the SQL database, creating a new record for each POST. The HTTP PUT method will be used to communicate from the client application to the server and an SQL INSERT will be generated to insert the information into the database. Each successful, INSERT to the database will result in the receipt of the code 201. Below is a reflection of a successful insert as posted from the ‘morgan’ monitoring package.



Each insert is expected to have the column ‘contactId = 0’ so that the software will autoincrement ‘contactId’ which is the primary key of the ‘contacts’ table.  
Test of the Post Command is done with POSTMAN



1) In dbOperations.js, create the interface function called ‘addContact’ so that we can insert the information into the database via SQL STORED PROCEDURE. The SQL-SERVER STORED PROCEDURE was generated with the CREATE PROCEDURE command in SQL-Server. It is called in this code, but first the input is matched with the correct field name and then ‘.execute’ is called to start the operation for inserting the data. The function entitled: InsertOrUpdateContact is the stored procedure made during the creation of the database.

const addContact = async(contact) => {

    try {

       let pool = await sql.connect(config); // Log on to database

       let addedContact = await pool.request()

           .input('firstName', sql.VarChar,contact.firstName)

           .input('lastName', sql.VarChar, contact.lastName)

           .input('emailAddr', sql.VarChar, contact.emailAddr)

           .input('street', sql.VarChar, contact.street)

           .input('city', sql.VarChar, contact.city)

           .input('stateProvince', sql.VarChar, contact.stateProvince)

           .input('countryCode', sql.VarChar, contact.countryCode)

           .input('postalCode', sql.Int, contact.postalCode )

           .input('homephone', sql.VarChar, contact.homephone )

           .input('workphone', sql.VarChar, contact.workphone )

           .input('cellphone', sql.VarChar, contact.cellphone )

           .input('messageHolder', sql.VarChar, contact.messageHolder )

            .execute('InsertOrUpdateContact');

        return addedContact.recordsets;

    }  // end try block

    catch(error) {

        console.log("DATABASE Post ERROR IN     addContact  Error Posted Next Line");

        console.log(error);

    } // end catch

}  // end addContact

// Export getThisContact so that ‘allroutes’ is aware of its existence

module.exports = { getContacts, getThisContact}

The complete ‘allroutes.js’ and ‘dbOperations.js’ files are listed below. Save all files and test with an ID number that you see in your specific SQL-SERVER table. Regarding the json information which is displays on the Chrome Browser: see the Special Note on the OUTPUT that is listed below.

=========== =========== =========== =========== =========== =========== =========

Complete files:

**// file: dboperations.js page 1**

const config = require('./dbconfig'); // access database configuration

const sql = require('mssql');

const getContacts = async() => {

try {

let pool = await sql.connect(config); // Log on to database

let theseContacts = await pool.request().query(

"SELECT \* FROM Contacts"

);

return theseContacts;

} // end try block

catch(error) {

console.log("DATABASE CONNECTION ERROR IN getThisContact Error Posted Next Line");

console.log(error);

} // end catch

} // end getContacts

const getThisContact = async(contactId) => {

try {

let pool = await sql.connect(config); // Log on to database

let theseContacts = await pool.request()

.input('input\_parameter', sql.Int, contactId)

.query( "SELECT \* FROM Contacts where contactId = @input\_parameter");

return theseContacts;

} // end try block

catch(error) {

console.log("DATABASE Query ERROR IN getThisContact Error Posted Next Line");

console.log(error);

} // end catch

} // end getThisContact

**// file: dboperations.js page 2 of 2**

const addContact = async(contact) => {

try {

let pool = await sql.connect(config); // Log on to database

let addedContact = await pool.request()

.input('firstName', sql.VarChar,contact.firstName)

.input('lastName', sql.VarChar, contact.lastName)

.input('emailAddr', sql.VarChar, contact.emailAddr)

.input('street', sql.VarChar, contact.street)

.input('city', sql.VarChar, contact.city)

.input('stateProvince', sql.VarChar, contact.stateProvince)

.input('countryCode', sql.VarChar, contact.countryCode)

.input('postalCode', sql.Int, contact.postalCode )

.input('homephone', sql.VarChar, contact.homephone )

.input('workphone', sql.VarChar, contact.workphone )

.input('cellphone', sql.VarChar, contact.cellphone )

.input('messageHolder', sql.VarChar, contact.messageHolder )

.execute('InsertOrUpdateContact');

return addedContact.recordsets;

} // end try block

catch(error) {

console.log("DATABASE Post ERROR IN addContact Error Posted Next Line");

console.log(error);

} // end catch

} // end addContact

module.exports = { getContacts, getThisContact, addContact}

======================================================================

// file allroutes.js

const express = require('express');

const router = express.Router();

// imports to implement api's

var bodyParser = require('express');

var cors = require('cors');

const app = express();  // to create an object of express

const dboperations = require('../dbfiles/dboperations')

app.use(bodyParser.urlencoded({extended:true}));

app.use(bodyParser.json());

app.use(cors());

router.use((request, response, next) => {

  console.log('middleware');   // for authentication software etc

  next();

})

router.route('/contacts').**get**((request, response)=> {

  dboperations.getContacts().then(result => {

       console.log(result);

       response.json(result);

   })

})

router.route('/contacts/:contactId').**get**((request, response)=> {

  dboperations.getThisContact(request.params.contactId).then(result => {

     response.send(result);

   //   response.json(result);

  })

})

router.route('/contacts').**post**((request, response)=> {

  let contact = request.body

      dboperations.addContact(contact).then(result => {

      response.status(201).json(result);

      })

})

router.route('/user').**get**((request, response)=> {

        console.log(" console    This is your first routed api call");

        response.send(" This is your first routed api call");

    })

    router.route('/test').**get**((request, response)=> {

      console.log(" console    This is a call to test");

      response.send(" This is a test");

  })

module.exports = router;    // export all of the routes

=================================== ========================================

// file allroutes.js

const express = require('express');

const router = express.Router();

// imports to implement api's

var bodyParser = require('express');

var cors = require('cors');

const app = express();  // to create an object of express

const dboperations = require('../dbfiles/dboperations')

app.use(bodyParser.urlencoded({extended:true}));

app.use(bodyParser.json());

app.use(cors());

router.use((request, response, next) => {

  console.log('middleware');   // for authentication software etc

  next();

})

router.route('/contacts').get((request, response)=> {

  dboperations.getContacts().then(result => {

       console.log(result);

       response.json(result);

   })

})

router.route('/contacts/:contactId').get((request, response)=> {

  dboperations.getThisContact(request.params.contactId).then(result => {

     response.send(result);

   //   response.json(result);

  })

})

router.route('/user').get((request, response)=> {

        console.log(" console    This is your first routed api call");

        response.send(" This is your first routed api call");

    })

    router.route('/test').get((request, response)=> {

      console.log(" console    This is a call to test");

      response.send(" This is a test");

  })

module.exports = router;    // export all of the routes

=========== =========== =========== =========== =========== =========== =========  
Special Note on the OUTPUT

Why do I receive 2 responses for each HTTP request coming in to nodemon.

Example:

Using Chrome web browser was used to generate Get request (Get the info in the database for a certain id.)

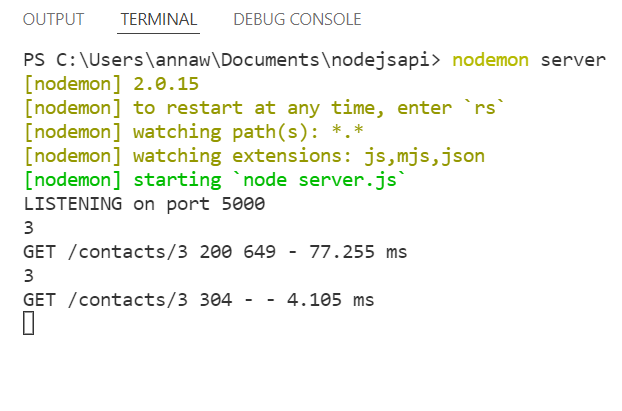
<http://localhost:5000/3>

was sent,

The console shows that, as a result of the single request, two request messages were sent to the back end server.

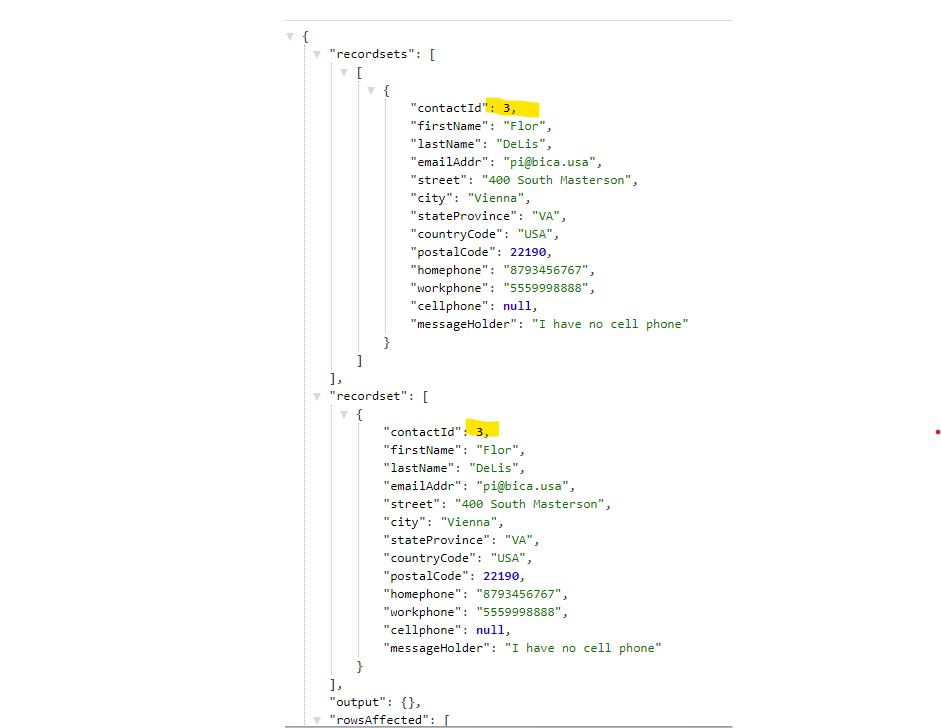
The first request (receiving a status code of 200) is called a CORS preflight message in which the backend software checks that all is well with the request so the message is s.

The second request (receiving a status code of 30) sends back the complete response (data including the data from the SQL Server)



The data received by the Chrome Browser is as follows

:



CORS: the *Cross Origin Resource Sharing* is used to allow node to connect to Express middleware. CORS will allow access to clients from different systems. Consequently, the first response sent to the browser should be a code 200: “I have checked your ‘header’ and everything is appropriate. I will now process the HTTP data, at which time the HTTP response will then now transmit the data in the second reply with the data that you requested.

Example:

