

WEB SERVICE CON NODEJS

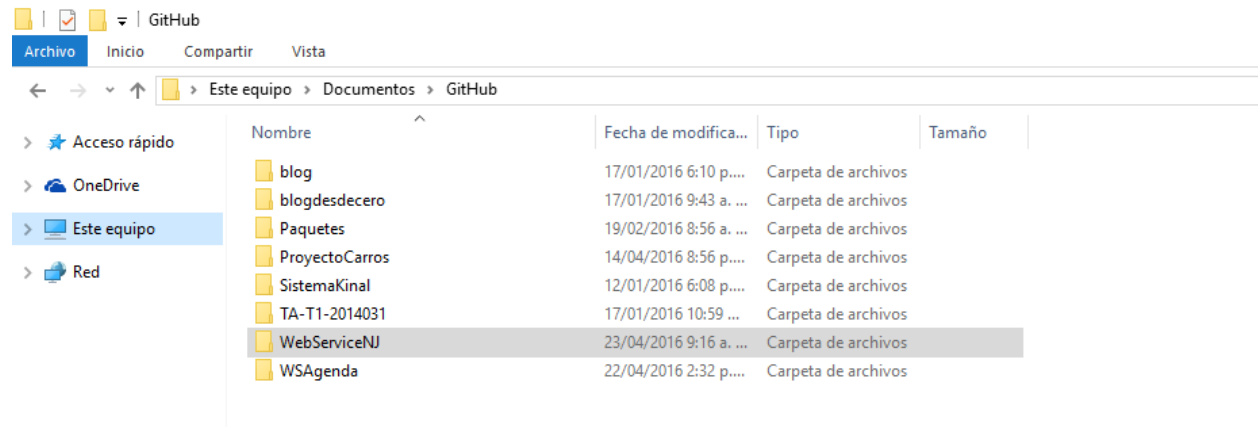
Josue Us

UJOSUE [Instagram.com/UJosue](https://www.instagram.com/UJosue)

CREAR EL SERVIDOR

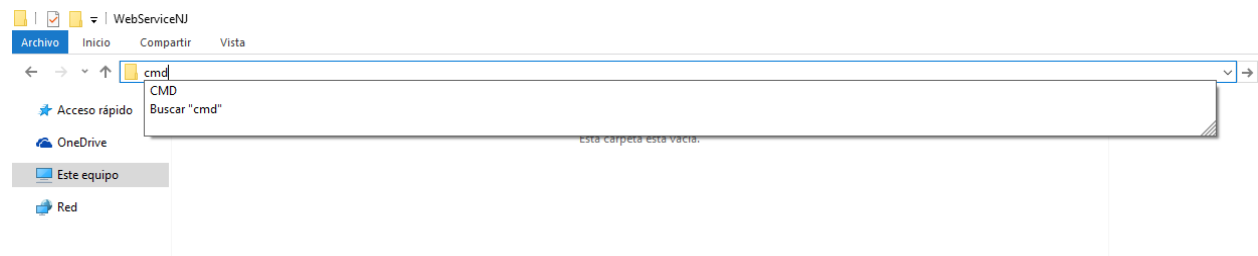
PASO 1: CARPETA

Lo primero que vamos a hacer es crear una carpeta en donde trabajaremos nuestro servidor.

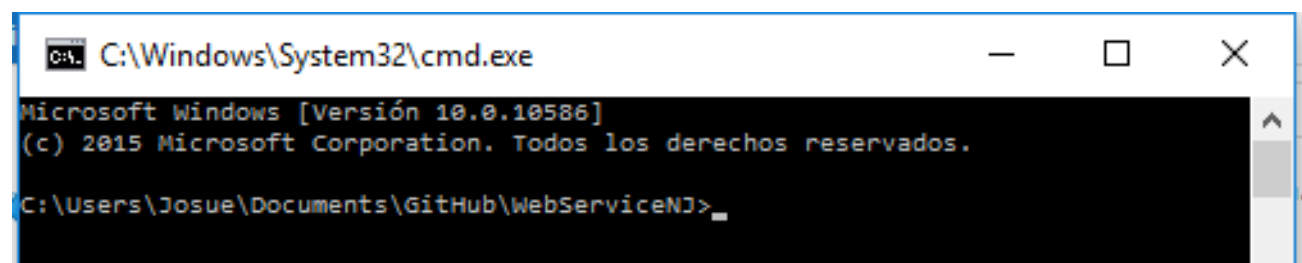


PASO 2: CREAR LOS MÓDULOS

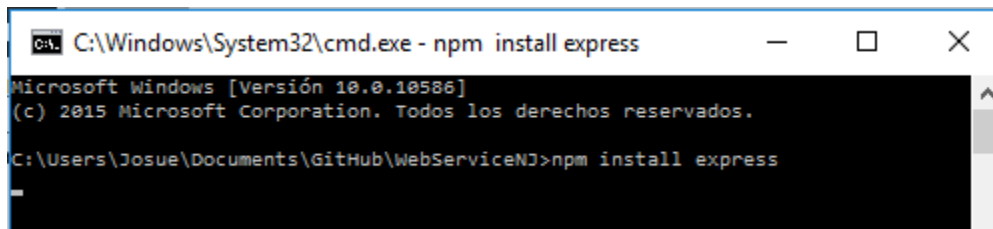
Ahora debemos de abrir la consola, y dirigirnos al directorio de nuestra carpeta, para esto entraremos a la carpeta y escribiremos "CMD" en la barra de direcciones del explorador.



Le daremos *enter* y abrirá la consola dentro del directorio..



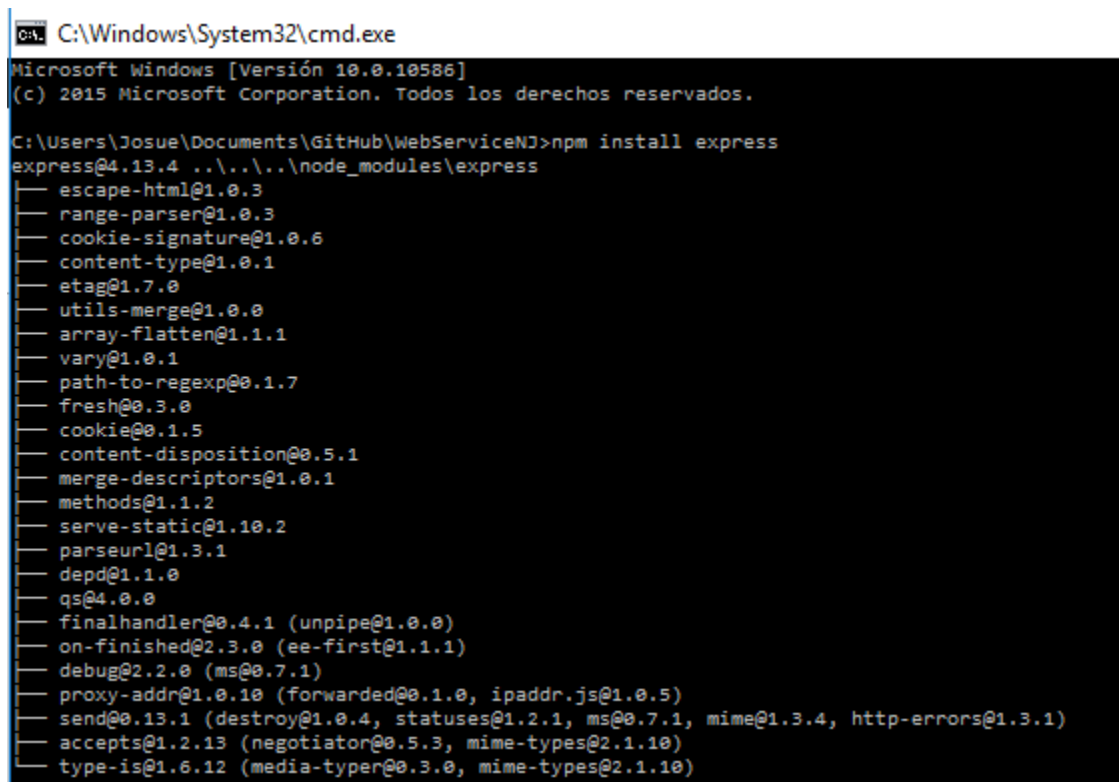
Ahora escribiremos los siguientes comandos:



```
C:\Windows\System32\cmd.exe - npm install express
Microsoft Windows [Versión 10.0.10586]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Josue\Documents\GitHub\WebServiceNJ>npm install express
```

Para instalar los módulos básicos de Node JS, y nos dará el siguiente resultado.



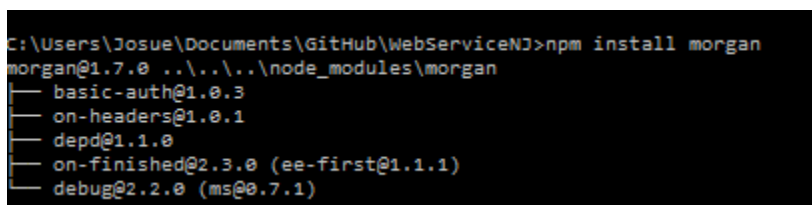
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.10586]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Josue\Documents\GitHub\WebServiceNJ>npm install express
express@4.13.4 ..\..\..\node_modules\express
├── escape-html@1.0.3
├── range-parser@1.0.3
├── cookie-signature@1.0.6
├── content-type@1.0.1
├── etag@1.7.0
├── utils-merge@1.0.0
├── array-flatten@1.1.1
├── vary@1.0.1
├── path-to-regexp@0.1.7
├── fresh@0.3.0
├── cookie@0.1.5
├── content-disposition@0.5.1
├── merge-descriptors@1.0.1
├── methods@1.1.2
├── serve-static@1.10.2
├── parseurl@1.3.1
├── depd@1.1.0
├── qs@4.0.0
├── finalhandler@0.4.1 (unpipe@1.0.0)
├── on-finished@2.3.0 (ee-first@1.1.1)
├── debug@2.2.0 (ms@0.7.1)
├── proxy-addr@1.0.10 (forwarded@0.1.0, ipaddr.js@1.0.5)
├── send@0.13.1 (destroy@1.0.4, statuses@1.2.1, ms@0.7.1, mime@1.3.4, http-errors@1.3.1)
├── accepts@1.2.13 (negotiator@0.5.3, mime-types@2.1.10)
└── type-is@1.6.12 (media-typer@0.3.0, mime-types@2.1.10)
```

Ahora para el web service necesitaremos los siguientes módulos adicionales:

Instalaremos los módulos de morgan, body-parser y mysql, necesarios para la ejecución del web service.

Para Morgan ingresaremos el siguiente comando:



```
C:\Users\Josue\Documents\GitHub\WebServiceNJ>npm install morgan
morgan@1.7.0 ..\..\..\node_modules\morgan
├── basic-auth@1.0.3
├── on-headers@1.0.1
├── depd@1.1.0
├── on-finished@2.3.0 (ee-first@1.1.1)
└── debug@2.2.0 (ms@0.7.1)
```

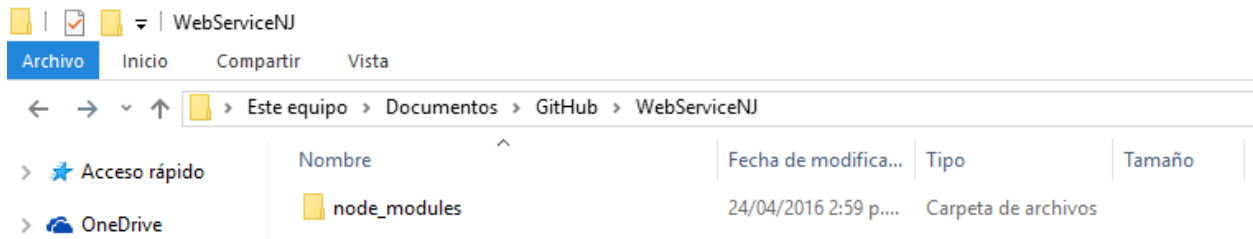
Luego instalaremos body-parser:

```
C:\Users\Josue\Documents\GitHub\WebServiceNJ>npm install body-parser
body-parser@1.15.0 ..\..\..\node_modules\body-parser
├── content-type@1.0.1
├── bytes@2.2.0
├── depd@1.1.0
├── qs@6.1.0
├── on-finished@2.3.0 (ee-first@1.1.1)
├── raw-body@2.1.6 (unpipe@1.0.0, bytes@2.3.0)
├── debug@2.2.0 (ms@0.7.1)
├── http-errors@1.4.0 (inherits@2.0.1, statuses@1.2.1)
├── iconv-lite@0.4.13
└── type-is@1.6.12 (media-typer@0.3.0, mime-types@2.1.10)
```

Por último instalamos la librería de mysql:

```
C:\Users\Josue\Documents\GitHub\WebServiceNJ>npm install mysql
mysql@2.10.2 ..\..\..\node_modules\mysql
├── bignumber.js@2.1.4
└── readable-stream@1.1.14 (string_decoder@0.10.31, isarray@0.0.1, inherits@2.0.1, core-util-is@1.0.2)
```

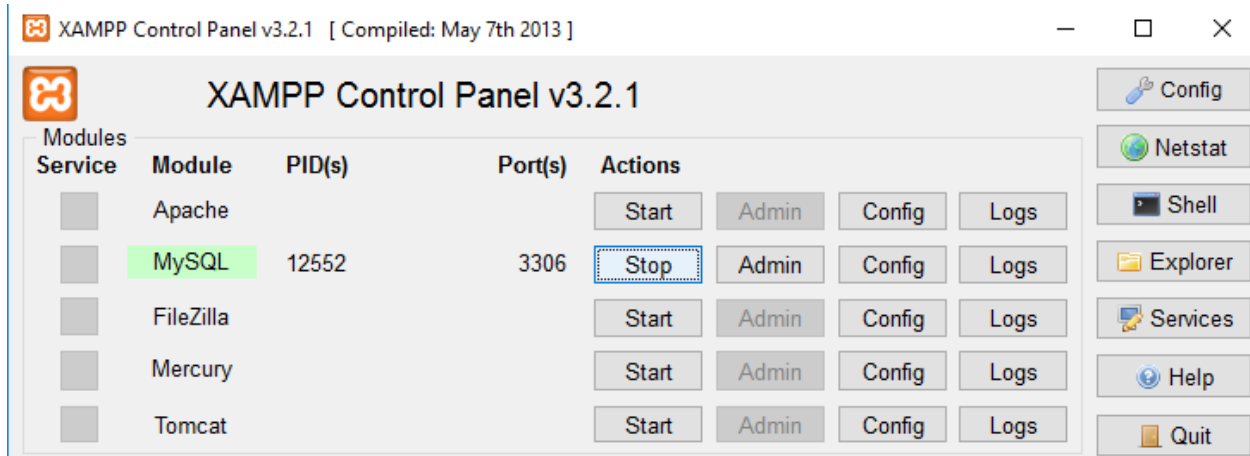
Al terminar en la carpeta se creará una carpeta que contendrá todos los módulos instalados.



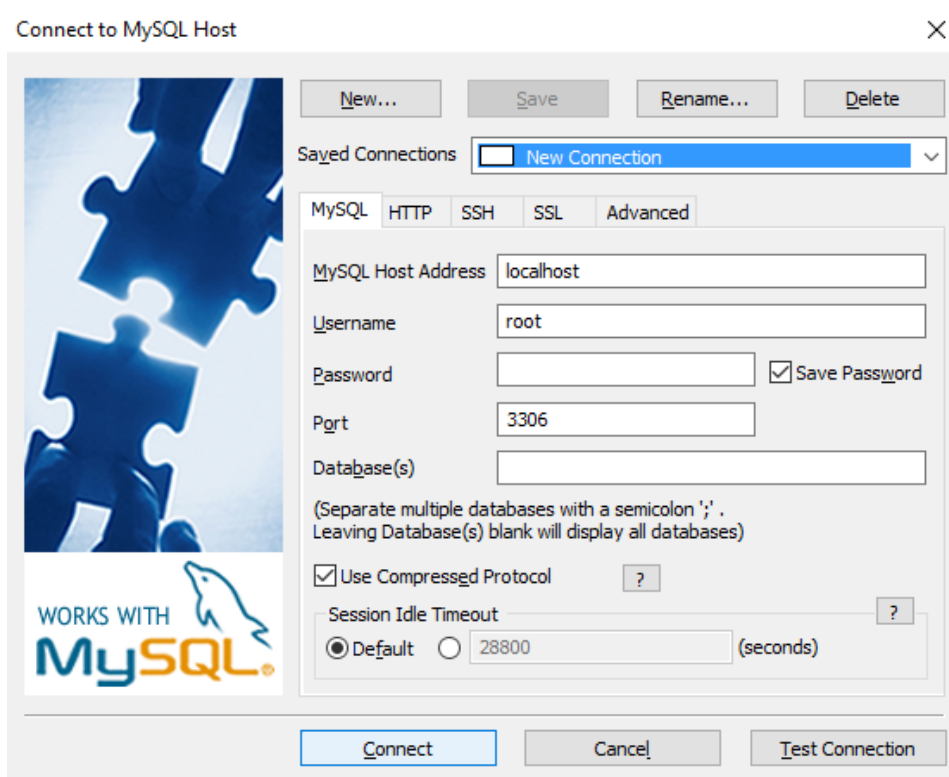
BASE DE DATOS

CREANDO LA BASE DE DATOS

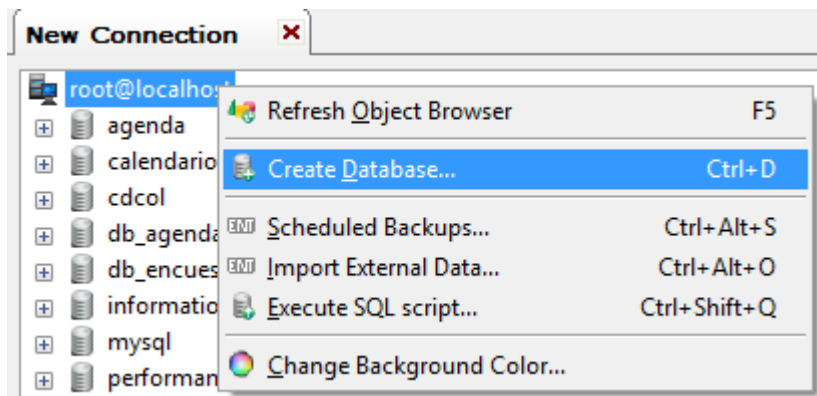
Abriremos *XAMPP* e iniciaremos *MySQL*.



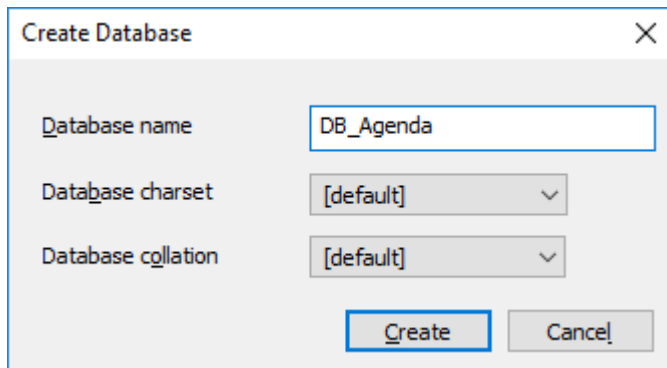
Abrimos SQLyog y nos conectaremos dándole en *connect*, siempre y cuando no hayamos modificado nada con anterioridad.



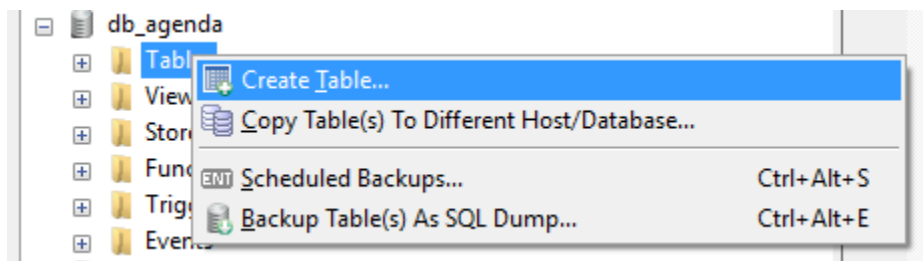
Ahora crearemos una nueva base de datos:



Y le daremos un nombre.



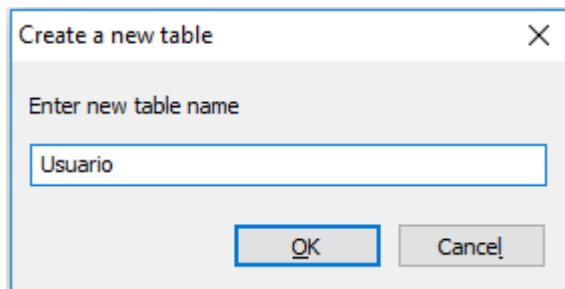
Luego crearemos la tabla, como en este caso haremos una agenda sencilla solo tendrá de usuario y contacto.



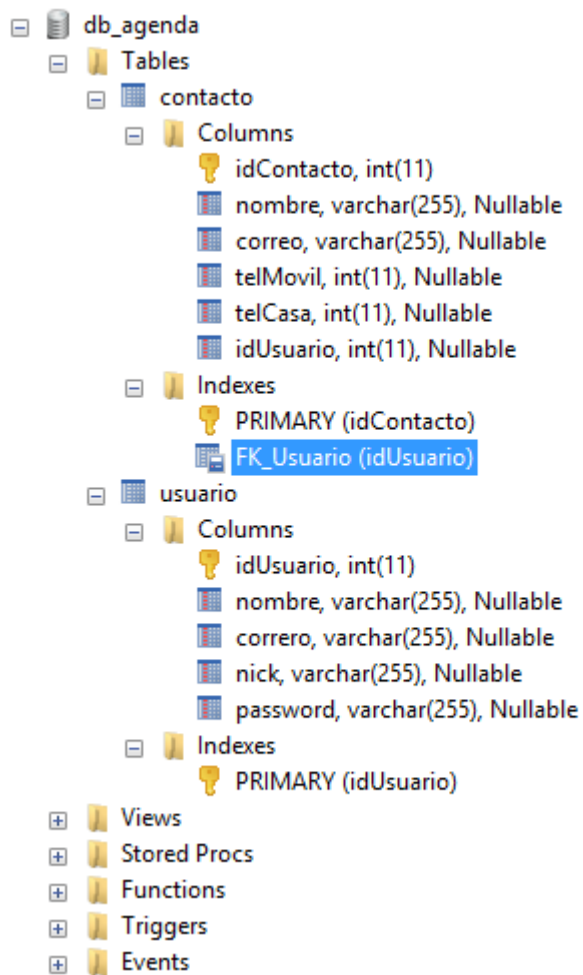
Primero haremos el de usuario, para ello escribimos los campos que necesitaremos, y al primero, que será la llave principal le seleccionaremos *pk* y *autoincrement*, para que se genere automáticamente.

Field Name	Datatype	Len	Default	PK?	Not Null?	Unsigned?	Auto Incr?	Zerofill?	Comment
* idUsuario	int	11		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
nombre	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
correo	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
nick	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
password	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Ahora la crearemos le ponemos el nombre cuando lo solicite y listo, haremos lo mismo con contacto.



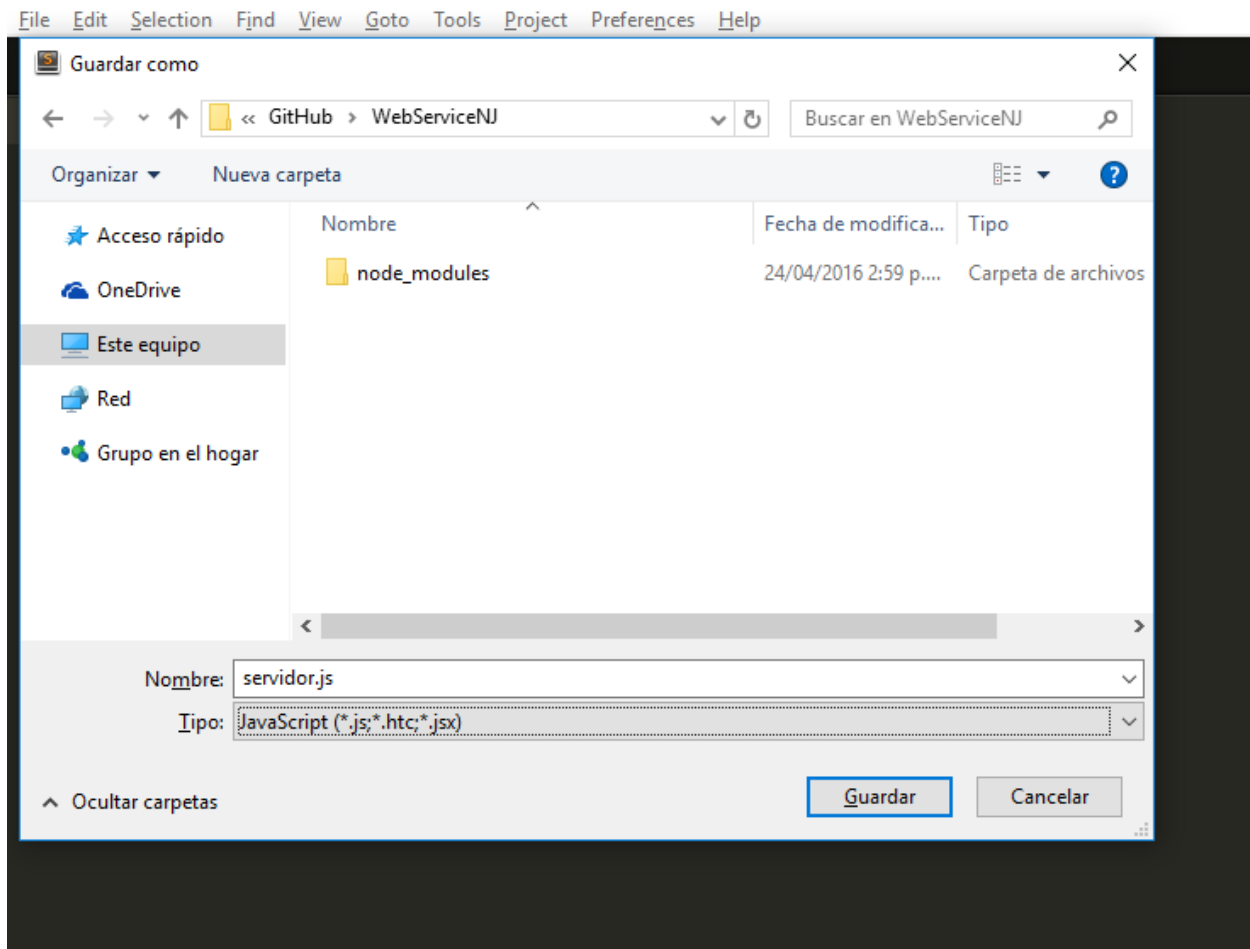
Quedándonos de la siguiente forma la base de datos.



CONFIGURAR EL SERVIDOR

SERVER.JS

Abriremos el editor de texto, en este caso utilizaré sublime text 3, sin embargo cualquier otro es funcional. Crearemos un nuevo archivo y lo guardaremos con algún nombre, en mi caso le pondré "server", este será el archivo que utilizaremos para configurar el servidor, por lo que lo guardaremos como tipo java script.



Ahora colocaremos el siguiente código:

```
servidor.js
1  (function() {
2    // body...
3  })();
```

En donde dice "// body" será donde escribiremos lo que queremos que haga.

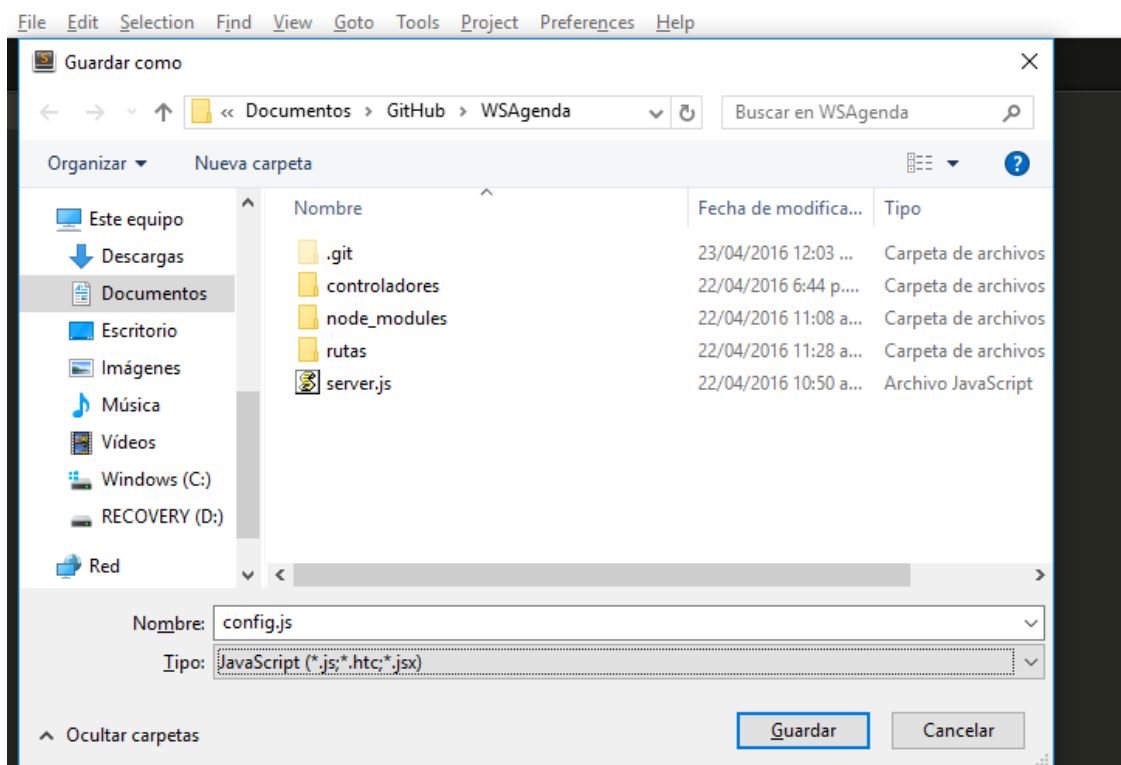
Lo siguiente es exportar las librerías que utilizaremos, para esto creamos una variable y le asignamos `"require(Nombre_De_La_Librería);"`. Para este ejemplo utilizaremos las librerías de express, Morgan, body-parser y mysql.

```
servidor.js
1 (function() {
2   // body...
3   let express = require('express');
4   let bodyParser = require('body-parser');
5   let morgan = require('morgan');
6   let mysql = require('mysql');
7 })();
```

Ahora configuraremos el puerto en donde correrá nuestro servidor, en este caso lo pondremos en el 3000, pero puede ser colocado en cualquier otro, siempre y cuando no esté siendo utilizado.

```
7 let puerto = 3000;
```

Luego le indicaremos la configuración para que se conecte a nuestra base de datos my sql, para esto crearemos un nuevo archivo, llamado `config.js`, en donde indicaremos la configuración.



Dentro de este archivo colocaremos lo siguiente:

```
config.js
server.js
1 module.exports={
2   //configuración
3 }
```

En lugar de *//configuración* colocaremos los parámetros, como el límite de conexiones, el nombre del host, o dirección IP si es remota, en el que se encuentra el servidor mysql, nombre de usuario y contraseña, y ya por último el nombre de la base de datos.

```
1 module.exports={
2   //configuración
3   'database':{
4     connectionLimit : 10,
5     host      : 'localhost',
6     user      : 'root',
7     password  : '',
8     database  : 'db_agenda'
9   }
10 }
```

Regresando al archivo de *servidor.js*, indicaremos la dirección del archivo que acabamos de crear.

```
8 let conf = require('./config');
```

Ahora asignamos la configuración de *database* en el archivo mediante una variable.

```
9 let pool = mysql.createPool(conf.database);
```

Creamos una variable llamada *app* que contendrá todos los métodos de *express*.

```
10 let app = express();
```

Luego le indicamos a *app* dónde está el pool de conexiones, y de esta forma lo hacemos accesible para todo el servidor.

```
11 app.set('pool', pool);
```

Ahora seguimos diciéndole que sólo utilizara un método de envío de datos, en este caso JSON.

```
12     app.use(bodyParser.urlencoded({
13         extended:false
14     }));
15     app.use(bodyParser.json());
```

Luego indicaremos que está en modo de desarrollo.

```
16     app.use(morgan('dev'));
```

Por último le indicamos como serán las rutas url, y le indicaremos de qué archivo él debe de obtener, en este caso le indicamos la carpeta *rutas*, que luego crearemos.

```
17     app.use('/api/v1/', require('./rutas')(app));
```

Y para finalizar agregaremos el siguiente código que nos indicará que todo funcionó correctamente.

```
19     app.listen(puerto, function(){
20         console.log("Your server are in the port: " + puerto);
21     })
```

El código nos queda de la siguiente forma:

```
servidor.js
4   let bodyParser = require('body-parser');
5   let morgan = require('morgan');
6   let mysql = require('mysql');
7   let puerto = 3000;
8   let conf = require('./config');
9   let pool = mysql.createPool(conf.database);
10  let app = express();
11  app.set('pool', pool);
12  app.use(bodyParser.urlencoded({
13    extended: false
14  }));
15  app.use(bodyParser.json());
16  app.use(morgan('dev'));
17  app.use('/api/v1/', require('./rutas')(app));
18
19  app.listen(puerto, function(){
20    console.log("Your server are in the port: " + puerto);
21  })
22  })();
```

CONTROLADORES DE USUARIO Y CONTACTO

Creamos la carpeta *controladores* en donde crearemos los controladores.



Ahora abriremos el *ControladorContacto.js* y escribiremos lo siguiente, entre las llaves escribiremos el código.

```
ControladorUsuario.js
1  module.exports = function(app) {
2
3  }
```

Dentro colocamos un *return* para que nos devuelva las funciones.

```

2      return {
3      |
4      }

```

Empezamos con registro, lo declaramos como una *function*, y le declaramos la conexión a la base de datos, utilizando el pool, le hacemos un *catch* a los posibles errores de conexión.

```

3      registro: function(peticion, respuesta) {
4          var pool = app.get('pool');
5          pool.getConnection(function(err, connection) {
6              if (err) {
7                  connection.release();
8                  respuesta.json({
9                      "code": 100,
10                     "status": "Error al conectar a la base de datos"
11                 });
12             }

```

Si no hubo ningún error llamamos al procedimiento almacenado *registro*, nuevamente hacemos un *catch* a los posibles errores, y de no darse ninguno reportamos que se produjo con éxito.

```

13      connection.query("CALL sp_registrousuario('" + peticion.body.hombre + "','" + peticion.body.correo + "','" + peticion.body.nick + "','" + peticion.body.contrasena + "','");", function(err) {
14          if (err)
15              throw err;
16          else
17              respuesta.send({
18                  "mensaje": "Registro insertado correctamente",
19                  "status": "200"
20              });
21          connection.release();
22      });

```

Seguimos con el *login*, para ello volvemos a declararlo igual que el *registro*

```

26      login: function(peticion, respuesta) {
27      |
28      }

```

Nuevamente llamamos a la conexión y hacemos *catch* a los errores.

```

3      registro: function(peticion, respuesta) {
4          var pool = app.get('pool');
5          pool.getConnection(function(err, connection) {
6              if (err) {
7                  connection.release();
8                  respuesta.json({
9                      "code": 100,
10                     "status": "Error al conectar a la base de datos"
11                 });
12             }

```

Ahora al igual que con *registro* llamamos al procedimiento *login* y sí no hubo ningún error lo reportamos.

```
36▼      connection.query("CALL sp_autenticarUsuario('" + petition.body.nombre + "', '" + petition.body.contrasena + "');",
37      if (err)
38          throw err;
39      else
40          respuesta.json(row);
41      connection.release();
42  });
```

Quedándonos el código de la siguiente forma:

```
1  module.exports = function(app) {
2      return {
3          registro: function(petition, respuesta) {
4              var pool = app.get('pool');
5              pool.getConnection(function(err, connection) {
6                  if (err) {
7                      connection.release();
8                      respuesta.json({
9                          "code": 100,
10                         "status": "Error al conectar a la base de datos"
11                     });
12                 }
13                 connection.query("CALL sp_registroUsuario('" + petition.body.nombre + "','" + petition.body.correo + "','" + petition.body.nick + "','" + petition.body.contrasena + "');", function(err) {
14                     if (err)
15                         throw err;
16                     else
17                         respuesta.send({
18                             "mensaje": "Registro insertado correctamente",
19                             "status": "200"
20                         });
21                     connection.release();
22                 });
23             });
24         },
25         login: function(petition, respuesta) {
26             var pool = app.get('pool');
27             pool.getConnection(function(err, connection) {
28                 if (err) {
29                     connection.release();
30                     res.json({
31                         "code": 100,
32                         "status": "Error al conectar a la base de datos"
33                     });
34                 }
35                 connection.query("CALL sp_autenticarUsuario('" + petition.body.nombre + "', '" + petition.body.contrasena + "');", function(err, row) {
36                     if (err)
37                         throw err;
38                     else
39                         respuesta.json(row);
40                     connection.release();
41                 });
42             });
43         }
44     }
45 }
46 }
```

Ahora para contacto nos saltaremos el principio, pues es igual que el de usuario, e iremos directo a las funciones, en este caso empezamos con *agregar*, luego de probar la conexión ejecutamos el procedimiento almacenado de *agregar* y devolvemos el resultado.

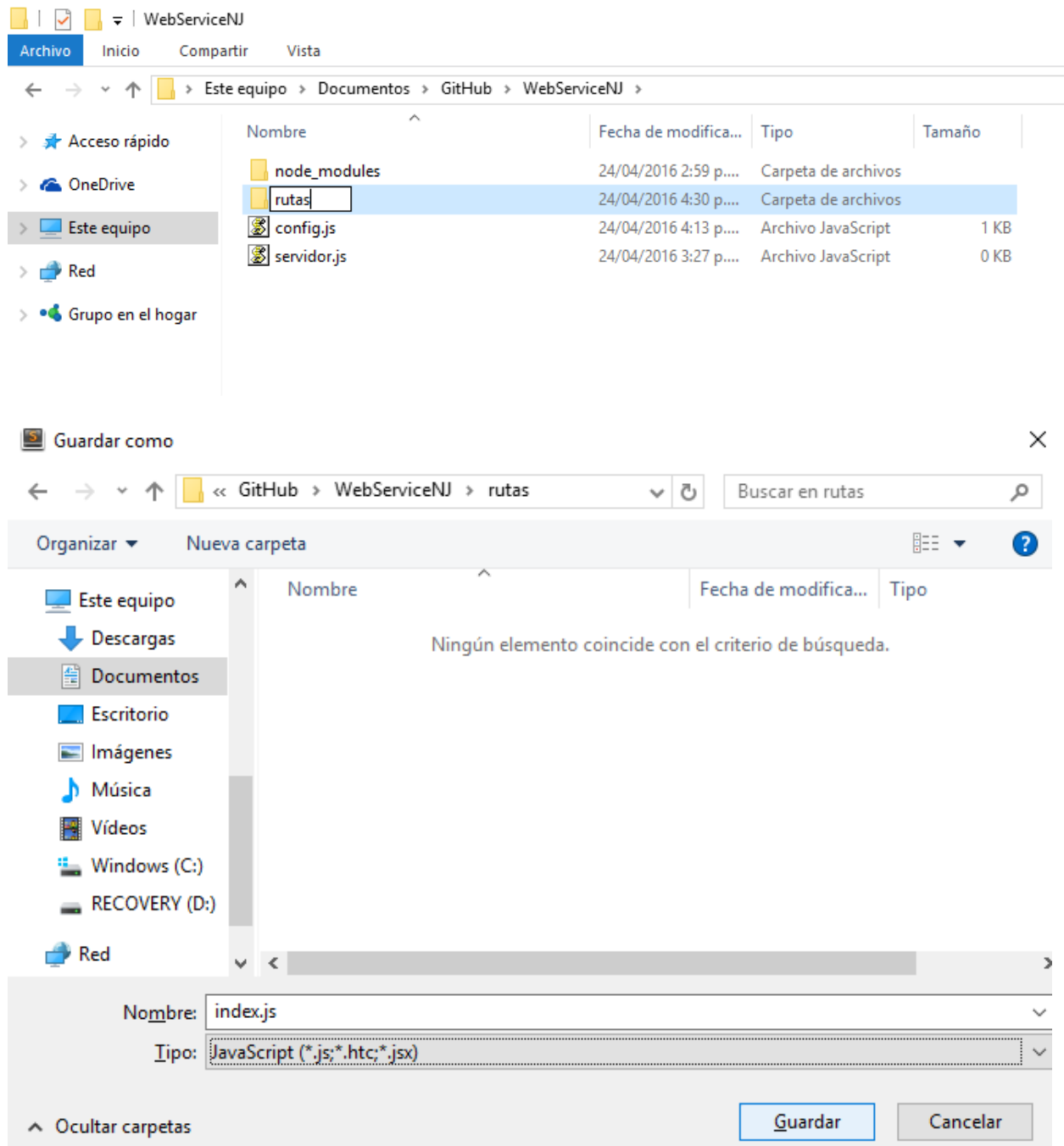
```
13      connection.query("INSERT INTO contacto VALUES (NULL,'" + req.body.nombre + "','" + req.body.telCasa + "','" + req.body.telMovil + "','" + req.body.direccion +
14      "','" + req.body.correo + "','" + req.body.idusuario + "');", function(err, row) {
15          if (err)
16              throw err;
17          else
18              res.json({
19                  "mensaje": "Contacto Agregado"
20              });
21          connection.release();
22      });
```

Volveremos a hacer lo mismo con los métodos *eliminar*, *listar* y *editar*. Quedándonos el código de la siguiente forma.

```
1 module.exports = function(app) {
2   return {
3     add: function(req, res) {
4       var pool = app.get('pool');
5       pool.getConnection(function(err, connection) {
6         if (err) {
7           connection.release();
8           res.json({
9             "code": 100,
10            "status": "Error al conectar a la base de datos"
11          });
12        }
13        connection.query("INSERT INTO contacto VALUES (NULL,'" + req.body.nombre + "','" + req.body.telCasa + "','" + req.body.telMovil + "','" + req.body.direccion + "','" + req.body.correo + "','" + req.body.idusuario + "')", function(err, row) {
14          if (err) {
15            throw err;
16          } else {
17            res.json({
18              "mensaje": "Contacto Agregado"
19            });
20            connection.release();
21          }
22        });
23      });
24    },
25    delete: function(req, res) {
26      var pool = app.get('pool');
27      pool.getConnection(function(err, connection) {
28        if (err) {
29          connection.release();
30          res.json({
31            "code": 100,
32            "status": "Error al conectar a la base de datos"
33          });
34        }
35        connection.query("Delete from contacto where idContacto=" + req.body.idcontacto, function(err, row) {
36          if (err) {
37            throw err;
38          } else {
39            res.json({
40              "mensaje": "Contacto eliminado"
41            });
42            connection.release();
43          }
44        });
45      });
46    },
47    list: function(req, res) {
48      var pool = app.get('pool');
49      pool.getConnection(function(err, connection) {
50        if (err) {
51          connection.release();
52          res.json({
53            "code": 100,
54            "status": "Error al conectar a la base de datos"
55          });
56        }
57        connection.query("Select * from contacto where idusuario=" + req.query.idusuario, function(err, row) {
58          if (err) {
59            throw err;
60          } else {
61            res.json(row);
62            connection.release();
63          }
64        });
65      });
66    },
67    edit: function(req, res) {
68      var pool = app.get('pool');
69      pool.getConnection(function(err, connection) {
70        if (err) {
71          connection.release();
72          res.json({
73            "code": 100,
74            "status": "Error al conectar a la base de datos"
75          });
76        }
77        connection.query("UPDATE contacto set nombre='" + req.body.nombre + "',telefonoCasa=" + req.body.telCasa + ",telefonoMovil=" + req.body.telMovil + ",direccion=" + req.body.direccion + ",correo=" + req.body.correo + " where idContacto=" + req.body.idContacto, function(err, row) {
78          if (err) {
79            throw err;
80          } else {
81            res.json({
82              "mensaje": "Contacto editado"
83            });
84            connection.release();
85          }
86        });
87      });
88    }
89  }
90 }
```

RUTAS DEL SERVIDOR

Como ya hemos indicado en el archivo *servidor.js* las rutas se obtendrán de la carpeta *rutas*, dentro de esta carpeta el servidor, por defecto, buscará el archivo *index.js*, entonces crearemos ambos objetos.



Importaremos *router* del módulo *express*.

```
1 let ruta = require('express').Router();
```

Agregamos el siguiente código, y en vez de *//body* colocaremos las instrucciones.


```
2 module.exports = (function(app) {  
3     // body...  
4 });
```

Declararemos la dirección raíz, que será como la homepage de la siguiente forma.

```
4     ruta.get('/', function(req, res){  
5         respuesta.send("Service started!")  
6     });
```

Importamos a los controladores de usuario y contacto, colocamos dos puntos antes de la barra para regresar un directorio, pues el index está en otra carpeta.

```
8     var usuario = require('../controladores/ControladorUsuario.js')(app);  
9     var contacto = require('../controladores/ControladorContacto.js')(app);
```

Luego para usuario creamos las rutas, e indicamos mediante que método http se podrá acceder al recurso.

```
14     ruta.post('/usuario/registro', usuario.registro);  
15     ruta.post('/usuario/login', usuario.login);
```

Hacemos lo mismo con *contacto*.

```
17     //Rutas de contacto  
18     ruta.get('/contacto', contacto.list);  
19     ruta.post('/contacto', contacto.add);  
20     ruta.put('/contacto', contacto.edit);  
21     ruta.delete('/contacto', contacto.delete);
```

Y por último ponemos un return.

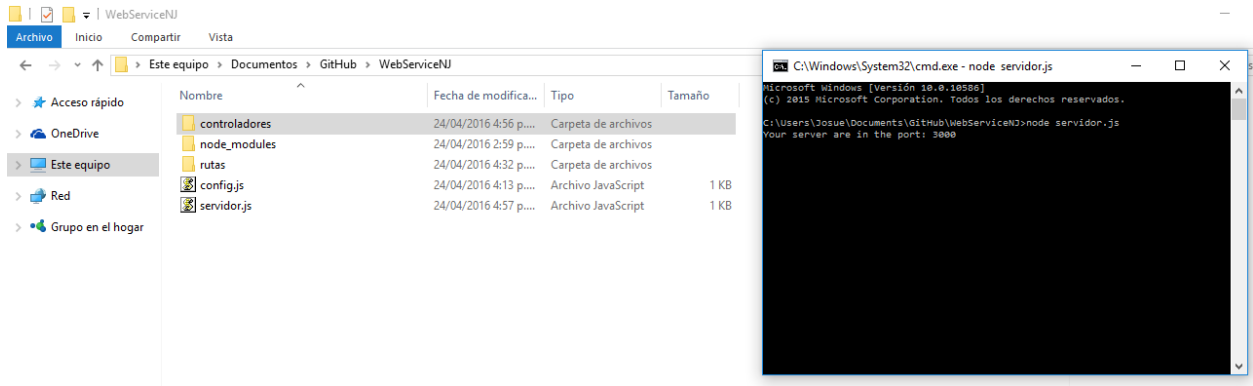
```
23     return ruta;
```

Y el código nos queda así.

```
1 var ruta = require('express').Router();
2 module.exports = (function(app) {
3   // body...
4   ruta.get('/', function(req, res){
5     respuesta.send("Service started!")
6   });
7
8   var usuario = require('../controladores/ControladorUsuario.js')(app);
9   var contacto = require('../controladores/ControladorContacto.js')(app);
10
11
12   //Rutas de usuario
13   // [METHOD : POST]sql
14   ruta.post('/usuario/registro', usuario.registro);
15   ruta.post('/usuario/login', usuario.login);
16
17   //Rutas de contacto
18   ruta.get('/contacto', contacto.list);
19   ruta.post('/contacto', contacto.add);
20   ruta.put('/contacto', contacto.edit);
21   ruta.delete('/contacto', contacto.delete);
22
23   return ruta;
24 });
```

EJECUCIÓN DEL SERVIDOR

LÍSTO HAZ ACABADO CON EL SERVIDOR AHORA SOLO PONLO A FUNCIONAR CON EL SIGUIENTE COMANDO.



Si quieres ver el repositorio puedes encontrarlo en:



GITHUB.COM/UJOSUE/WEBSERVICENJ

Sígueme en:

Instagram.com/UJosue10

Twitter.com/UJosue10

Ujosue.wordpress.com

Facebook.com/josueus2