Tim Scarlith Hernández

Josué Torres Nervaéz

Cristian Agüero

Bases de Datos 1

28/07/2020

Guía de implementación y de instalación de CRUD implementado en una base de datos nosql, MONGODB y Angular como interfaz gráfica.

En el siguiente artículo se especifican el conjunto de pasos que se deben de seguir al momento de crear una CRUD en una base de datos nosql como lo es Mongodb y como se puede interconectar este con Angular a través de Node js y su framework Expreess. Esto para crear una aplicación de administrador de contactos, implementando todas las funcionalidades básicas de cualquier aplicación que implemente un CRUD, (create, read, update, delete).

Primeramente si se desea implementar este proyecto se deben de instalar un conjunto de programas los cuales se citan a continuación:

- Node JS
- Angular 9 o Angular 10 con angular materaal
- Mongodb(en su versión de Cummunity Server)
- Editor de texto cualquiera preferiblemente visual code (por su facilidad en el manejo de terminales y compatibilidad con Angular)

Procesos de desarrollo

Backend

En primera instancia se iniciara con el desarrollo del backend.

Una vez instalados lo programas se debe proceder a crear una carpeta la cual será la carpeta correspondiente al proyecto, del actual se llama “Contactos” llamar lista de contactos como se puede visualizar en la imagen como se muestra en la figura 1.

figura 1

Seguidamente se debe de proceder a instalar un conjunto de dependencias para los programas referentes al backend, esto se puede hacer abriendo una terminal o una consola en el archivo

Contactos y seguidamente ingresar el comando **npm init -y,** esto para comenzar un archivo de tipo js que refiere a la instanciación de express en el proyecto, este archivo ayuda a manipular la instanciación de las dependencias que se utilizaran en el desarrollo del backend. En la siguientes líneas se muestran todas las dependencias necesarias para crear la base de datos, la conexión y las funcionalidades que proporcionadas por el CRUD.

$ npm i body-parser chalk express mongoose.

$npm i nodemon -g (este también puede ser instalado como dependencia del proyecto).

$npm i cors -d

una vez instanciados las dependencias anteriores se realiza una modificación en el archivo package.js esto para facilitar el arranque en consola y que la palabra clave esté más con la base de con el objetivo de iniciar un servidor. Vea la figura 2.

Base de datos Mongo

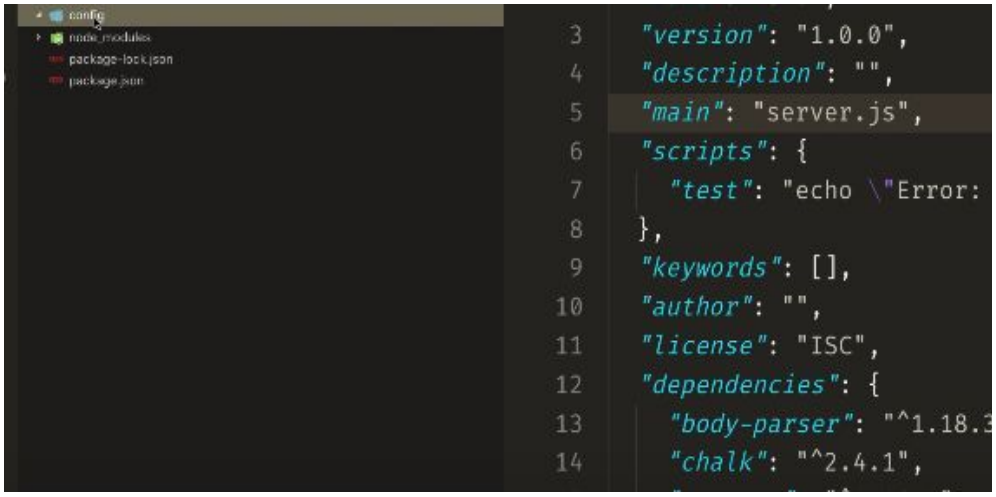


figura 2.

Seguidamente se debe de crear una carpeta de configuraciones de conexión con la base de datos llamada “config” dos archivos,(properties.js y database.js) para establecer la base de datos y todas las comunicaciones como url y puertos. Aprecie con mayor detenimiento en la figura 3

figura 3

Dentro del archivo properties.js debe de exportar su contenido, el cual es la instanciación del puerto de escucha de la base de datos y la dirección de esta misma, en el caso de la base de datos que crearemos se llamará crudNode, cómo se puede apreciar en la figura 4.

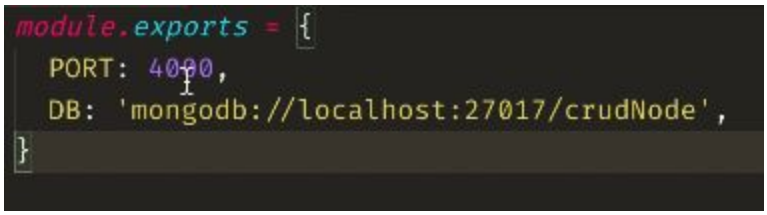


figura 4

El archivo de la base de datos debe de contener el código referente a la base de datos, debe de instanciar objetos de comunicación como uno de tipo propiedad el cual hereda de los métodos y atributos del archivo properties.js, también debe de exportar el módulo con todo el objeto y sus metod instanciados el cual realiza la coneccion de la de un externo con la base de datos. Vea ejemplificación en la figura 5.

```
const mongoose = require('mongoose');
const chalk = require('chalk');
const dbURL = require('./properties').DB;

const connected = chalk.bold.cyan;
const error = chalk.bold.red;

const termination = chalk.bold.magenta;

module.exports = ()=>{
    mongoose.set('useCreateIndex', true)
    mongoose.connect(dbURL,{useNewUrlParser: true, useUnifiedTopology: true})
    //mongoose.connect(dbURL,{useNewUrlParser: true})
    .then(() => console.log(connected('Mongo connected! on', dbURL)))
    .catch( err => console.log(error(`Conection has an error ${err}`)))
    process.on('SIGINT',()=>{
        mongoose.connection.close(()=>{
            console.log('Mongoose is disconected');
            process.exit(0);
        });
    });
}
```

figura 5.

Modulo de logica

Una vez terminado el procesos de programar el servidor se debe de crear la lógica que refiere a las funciones que debe de realizar nuestra aplicación, por lo cual se debe de crear una carpeta “customers” donde se almacenarán cuatro archivos,(customers.controller.js, customers.dao.js, customers.models.js y customers.routes.js), como se aprecia en la figura 6.

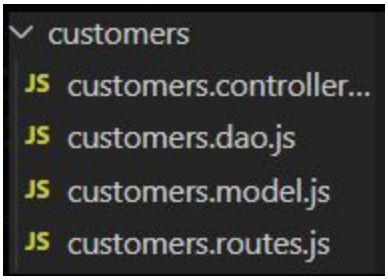
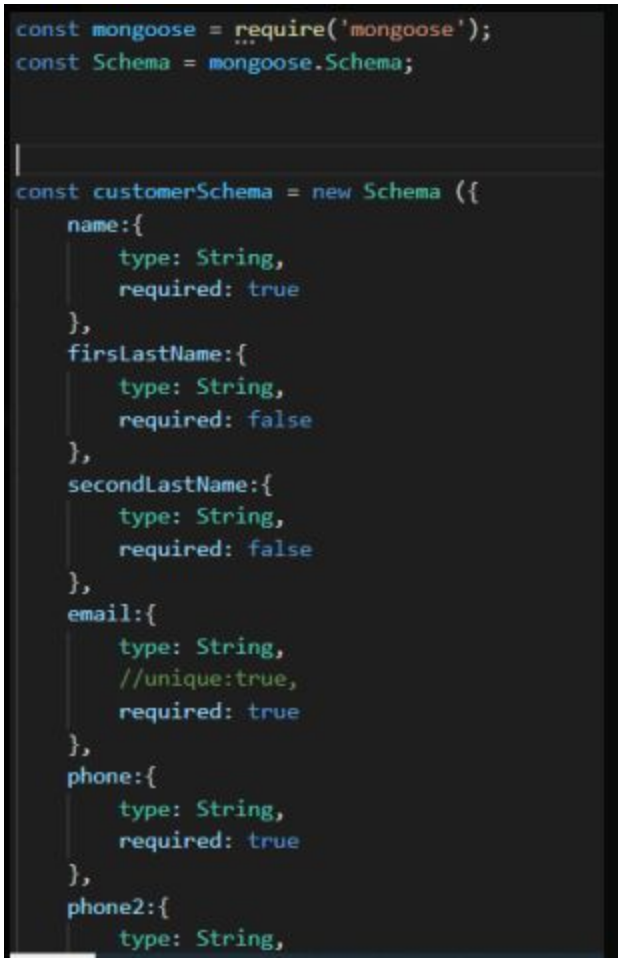


figura 6

Primeramente se debe de trabajar con el archivo de customer.models debido a que estos van a que en este archivo van a existir las plantillas de los objetos que contendrá la base de datos. El formato o modelo de los objetos que se pueden instanciar de esta clase será utilizado posteriormente para llamar a métodos de agregar y editar. Aprecie mejor lo antes mencionado en la figura 7.y figura 8.



```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;


const customerSchema = new Schema ({
    name:{
        type: String,
        required: true
    },
    firsLastName:{
        type: String,
        required: false
    },
    secondLastName:{
        type: String,
        required: false
    },
    email:{
        type: String,
        //unique:true,
        required: true
    },
    phone:{
        type: String,
        required: true
    },
    phone2:{
        type: String,
```

figura 7

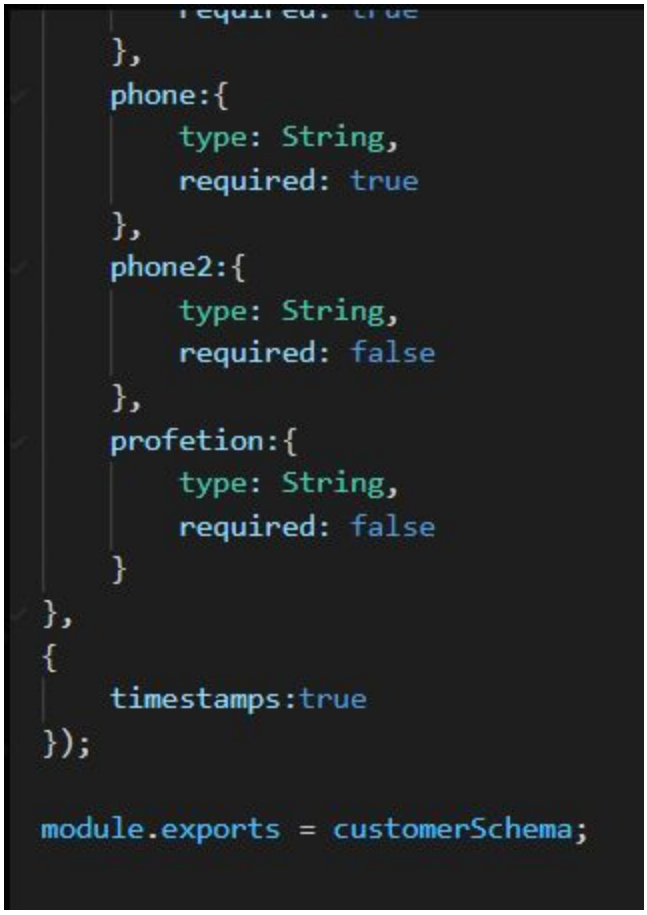


figura 8

Seguidamente se debe de trabajar en el archivo dao o data acces object, que es un documento donde se debe de crear las funciones CRUD, los tipos de parámetros que estas deben de resivir y que funcionalidad cumpliran(agreagar, borrar, encontar, etc), a este conjunto de instanciaciones se les llama esquema. Este archivo también debe de contener un módulo el cual exporte el esquema para que pueda ser implementado en el archivo controller correspondiente. Para mejorar el visualizacion de lo anterior mencionado vea la figura 9.

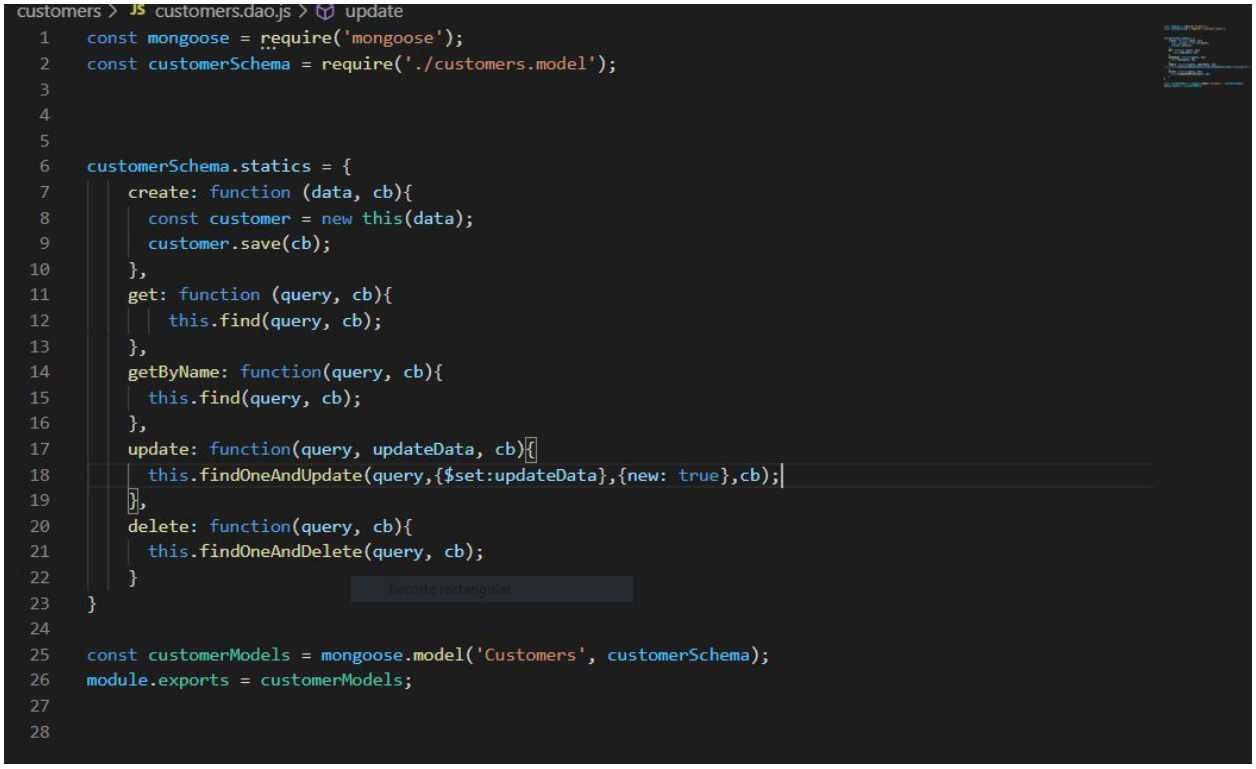


figura 9

Finalmente podremos implementar funcionalidad a los métodos del objeto que se exporta desde el archivo dao, como se puede apreciar en las figuras 10, 11, 12 13, 14

```
const  Customers =  require('./customers.dao');
exports.createCustumer = (req, res, next ) =>{
    const customers = {
        name: req.body.name,
        firsLastName: req.body.firsLastName,
        secondLastName: req.body.secondLastName,
        email: req.body.email,
        phone: req.body.phone,
        phone2: req.body.phone2,
        profetion: req.body.profetion
    };
    Customers.create(customers, (err, customers)=> {
        if(err) res.json({error:err});
        res.json({message: 'Customer created succesfully'});
    });
}

exports.getCustumers = (req, res, next)=>{
    Customers.get({},(err, customers)=>{
       if(err) res.json({error: err});
       res.json({Customers:customers});

    });

}

```

figura 10

```
18  exports.getCustumers = (req, res, next)=>{
19      Customers.get({},(err, customers)=>{
20          if(err) res.json({error: err});
21          res.json({Customers:customers});
22
23      });
24
25  }
26
27  exports.getCustumer = (req, res, next)=>{
28      Customers.getByName({name:req.params.name},(err, customer)=>{
29          if(err) res.json({error: err});
30          res.json({Customer: customer});
31
32      });
33  }
34
```

figura 11

```
exports.updateCustumer = (req, res, next)=>{
    const customer = {
        name: req.body.name,
        firsLastName: req.body.firsLastName,
        secondLastName: req.body.secondLastName,
        email: req.body.email,
        phone: req.body.phone,
        phone2: req.body.phone2,
        profetion: req.body.profetion
    };
    Customers.update({_id:req.params.id},customer,(err, customer)=>{
       if(err) res.json({error: err});
       res.json({message:'Customer update succesfully'});
    });
}

exports.removeCustumers = (req, res, next)=>{
    Customers.delete({_id:req.params.id},(err, customers)=>{
       if(err) res.json({error: err});
       res.json({message:'Customer deleted succesfully'});

    });

}
```

figura 12

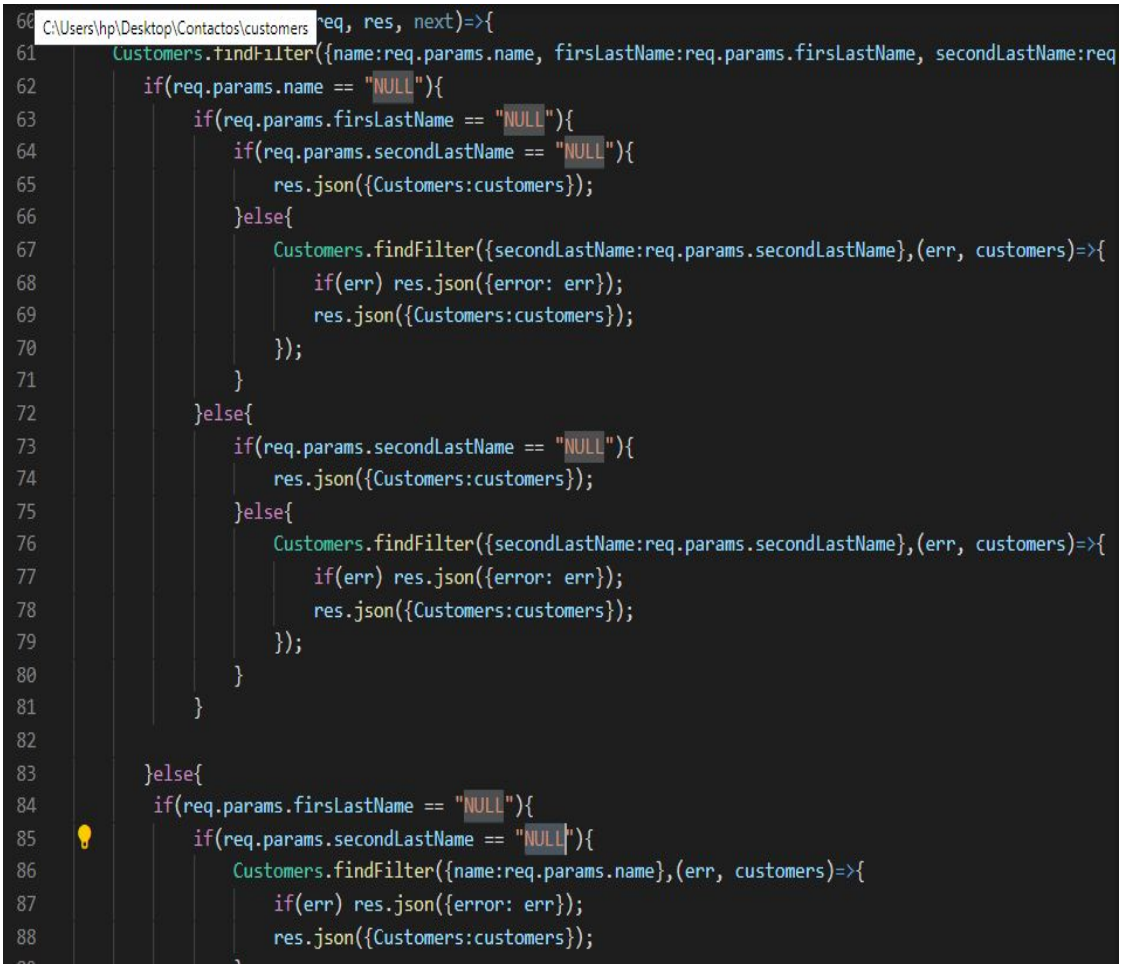
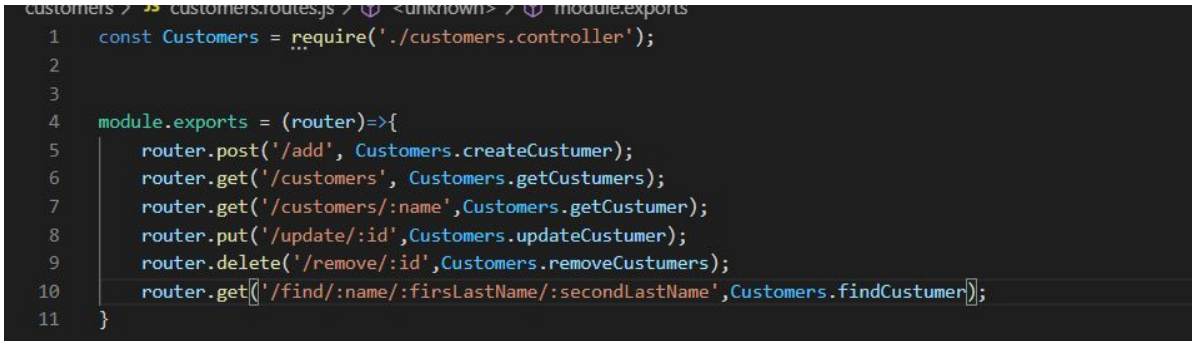

C:\Users\hp\Desktop\Contactos\customers
req, res, next)=>{
Customers.findFilter({name:req.params.name, firsLastName:req.params.firsLastName, secondLastName:req
if(req.params.name == "NULL"){
if(req.params.firsLastName == "NULL"){
if(req.params.secondLastName == "NULL"){
res.json({Customers:customers});
}else{
Customers.findFilter({secondLastName:req.params.secondLastName},(err, customers)=>{
if(err) res.json({error: err});
res.json({Customers:customers});
});
}
}else{
if(req.params.secondLastName == "NULL"){
res.json({Customers:customers});
}else{
Customers.findFilter({secondLastName:req.params.secondLastName},(err, customers)=>{
if(err) res.json({error: err});
res.json({Customers:customers});
});
}
}
}else{
if(req.params.firsLastName == "NULL"){
if(req.params.secondLastName == "NULL"){
Customers.findFilter({name:req.params.name},(err, customers)=>{
if(err) res.json({error: err});
res.json({Customers:customers});

figura 13

```
                );
            }else{
                Customers.findFilter({secondLastName:req.params.secondLastName},(err, customers)=>{
                    if(err) res.json({error: err});
                    res.json({Customers:customers});
                });
            }
        }else{
            if(req.params.secondLastName){
                Customers.findFilter({name:req.params.name,firsLastName:req.params.firsLastName},(err, cu
                    if(err) res.json({error: err});
                    res.json({Customers:customers});
                });
            }else{
                Customers.findFilter({name:req.params.name,firsLastName:req.params.firsLastName,secondLas
                    if(err) res.json({error: err});
                    res.json({Customers:customers});
                });
                }


        }
       }


    });

}

```

figura 14

Seguidamente se deben de instanciar las direcciones en el archivo de rutas, con las cuales podremos acceder a los métodos instanciados en el objeto de tipo customer qué se exportó del archivo customer.controller.js, además se debe de crear un módulo donde se exporte todas las rutas instanciadas. Vea lo anteriormente mencionado en la figura 15.



```
customers > JS customers.routes.js > <unknown> > module.exports
const Customers = require('./customers.controller');


module.exports = (router)=>{
    router.post('/add', Customers.createCustumer);
    router.get('/customers', Customers.getCustumers);
    router.get('/customers/:name',Customers.getCustumer);
    router.put('/update/:id',Customers.updateCustumer);
    router.delete('/remove/:id',Customers.removeCustumers);
    router.get('/find/:name/:firsLastName/:secondLastName',Customers.findCustumer);
}
```

figura 15

En el archivo principal se debe de crear un documento de extensión .js llamado server.js este documento es el cual iniciara nuestro servicio como servidor, inicia la coneccion con la base de datos y controla el trasiego de datos, aparte de eso es quien maneja los permisos de cuales son las páginas que pueden consumir de nuestro servicio de base de datos. Vea los detalles del código del servidor en la figura 16.

server.js > ...

```
1   const cors = require('cors');
2   const express = require('express');
3   const bodyParser = require('body-parser');
4   const properties = require('./config/properties');
5   const customersRoutes =  require('./customers/customers.routes');
6   const db = require('./config/database');
7   //Init db
8   db();
9   const bodyParserJSON = bodyParser.json();
10  const bodyParserURLEncoded = bodyParser.urlencoded({extended: true});
11  const app = express();
12  app.use(bodyParserJSON);
13  app.use(bodyParserURLEncoded);
14
15  app.use(cors());
16  //init ROUTER
17  const router = express.Router();
18  app.use('/api',router);
19  customersRoutes(router);
20  app.listen(properties.PORT,()=> console.log(`Server is running on ${properties.PORT}`));
```

figura 16

Frontend

Consta de dos partes principales, que son la lógica de comunicación y la interfaz gráfica realizada en angular 10, para la comunicación se utiliza una clase llamada service que implementa las librerías HttpCliente para comunicarse por medio de los protocolos de la red con nuestro servidor y HttpHeaders para darle un formato de un json a nuestro objeto enviado al servidor como se muestran en las figuras 17, 18 y 19

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
```

figura 17

```
const httpOption = {
  headers: new HttpHeaders({
    'Contend-Type': 'aplication/json'
  })
};
```

figura 18

```
constructor(
  private _http: HttpClient
){}

get(): Observable<Response> {
  return this._http.get<Response>(this.url);
}

add(cliente: Telefono):Observable<Response>{
  return this._http.post<Response>(this.urlAdd, cliente, httpOption);
}

delete(id: string):Observable<Response>{
  return this._http.delete<Response>(`${this.urlDelet}/${id}`);
}

edit(cliente: Telefono,id: string):Observable<Response>{
  return this._http.put<Response>(`${this.urlEdit}/${id}`, cliente, httpOption);
}

find(nombre: string, apellido1: string, apellido2: string) {
  console.log(`${this.urlFind}/${nombre}/${apellido1}/${apellido2}/`);
  return this._http.get<Response>(`${this.urlFind}/${nombre}/${apellido1}/${apellido2}/`, httpOption);
}
```

figura 19

Como se muestra en la imagen 19, se utilizan varias interface para almacenar los datos enviados y obtenidos de las distintas llamadas(Response y Telefono), estas interfaces los contienen las variables necesarias para el correcto almacenamiento de los datos.

También se hace uso de las url para enviar datos en vez de utilizar un objeto, normalmente este métodos solo se utiliza en momentos en que se ocupa enviar datos puntuales en vez de objetos enteros

La interfaz gráfica consta de una ventana principal que tendrá una tabla con toda la información pertinente con los contactos con se muestra en la figura 20.

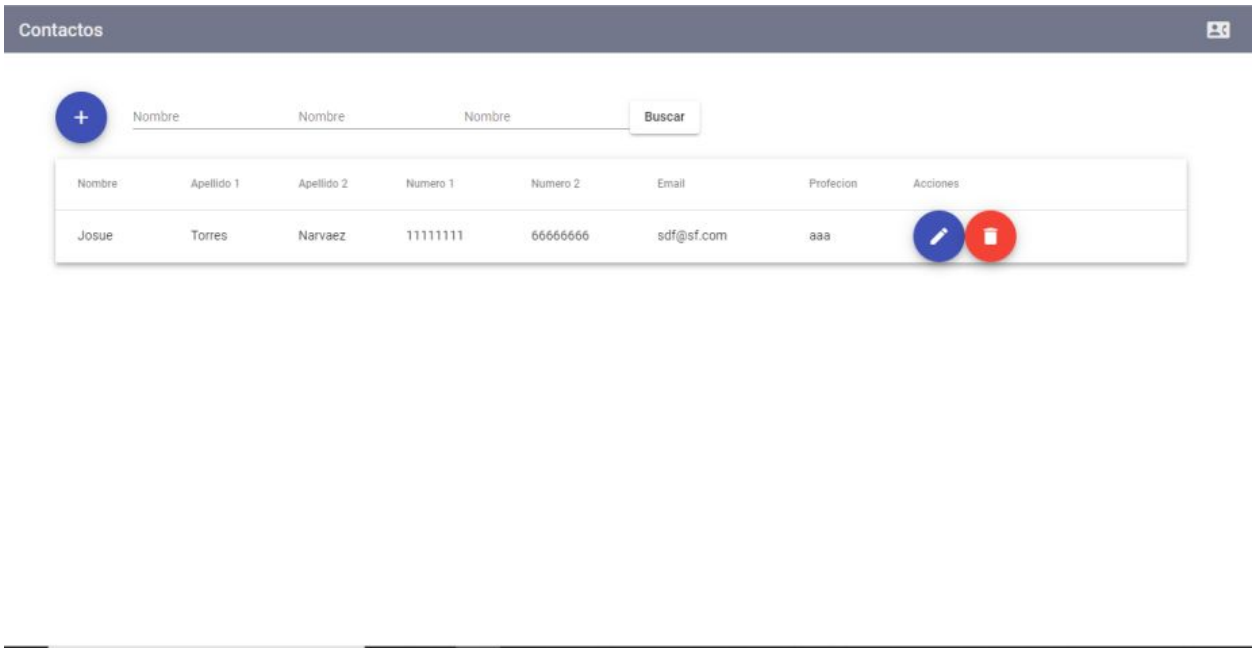


figura 20

El usuario podrá editar, eliminar, filtrar y agregar contactos como se muestra en las figuras 21, 22, 23, 24.

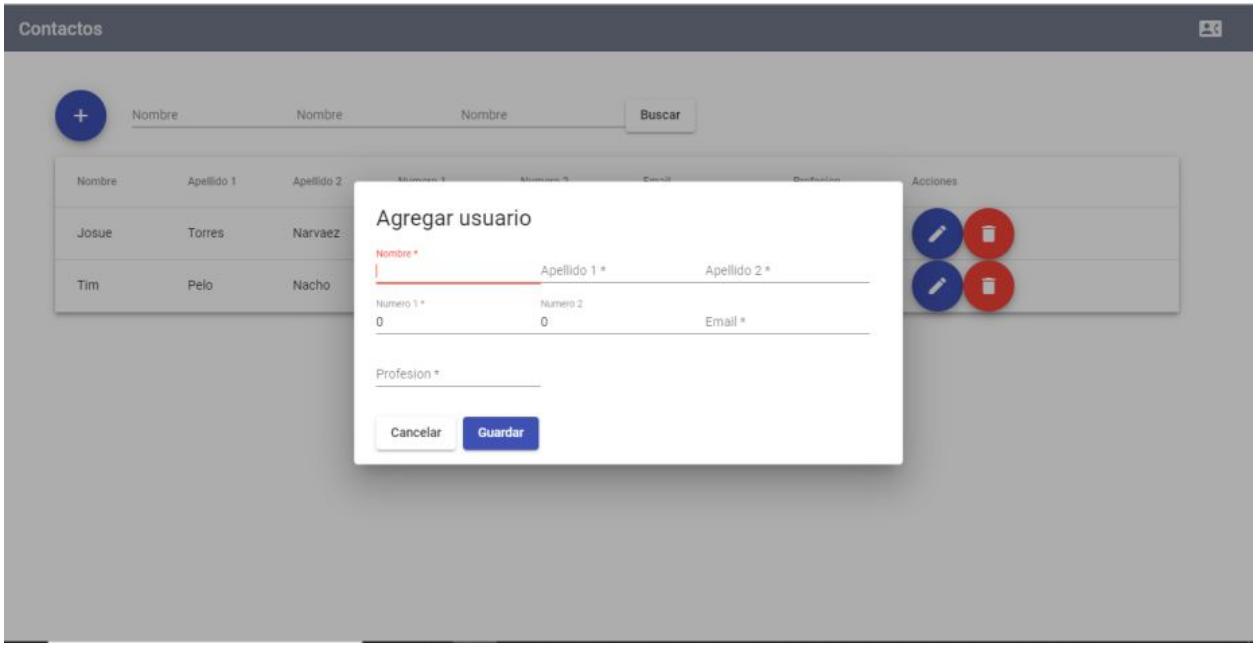


figura 21

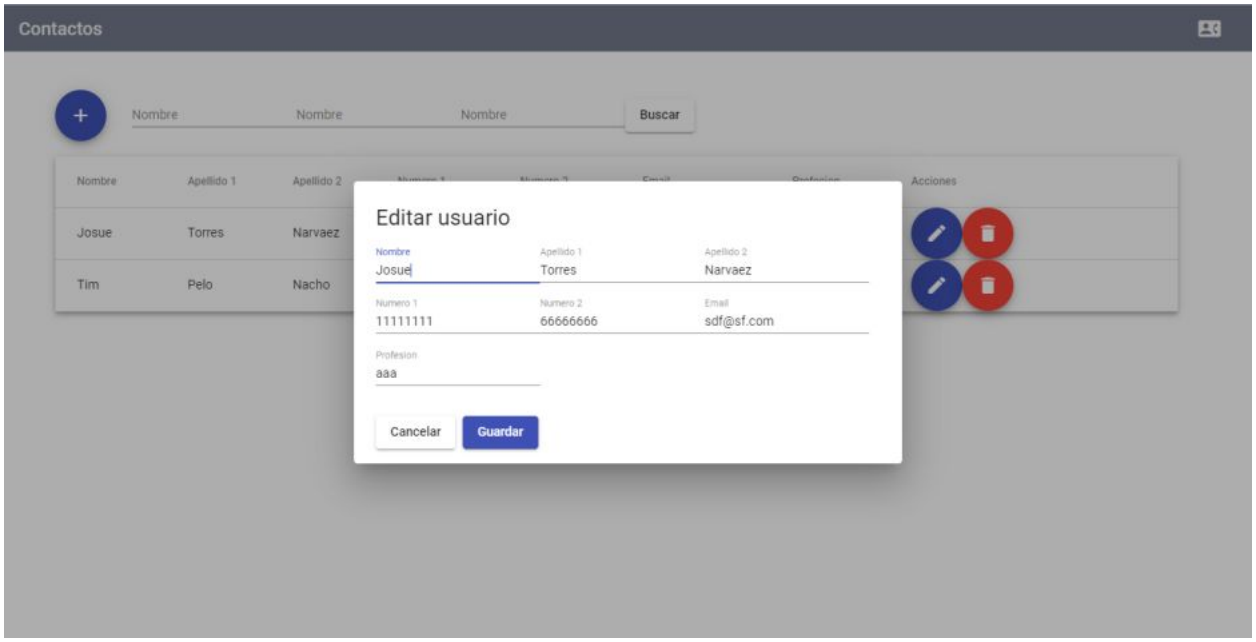


figura 22

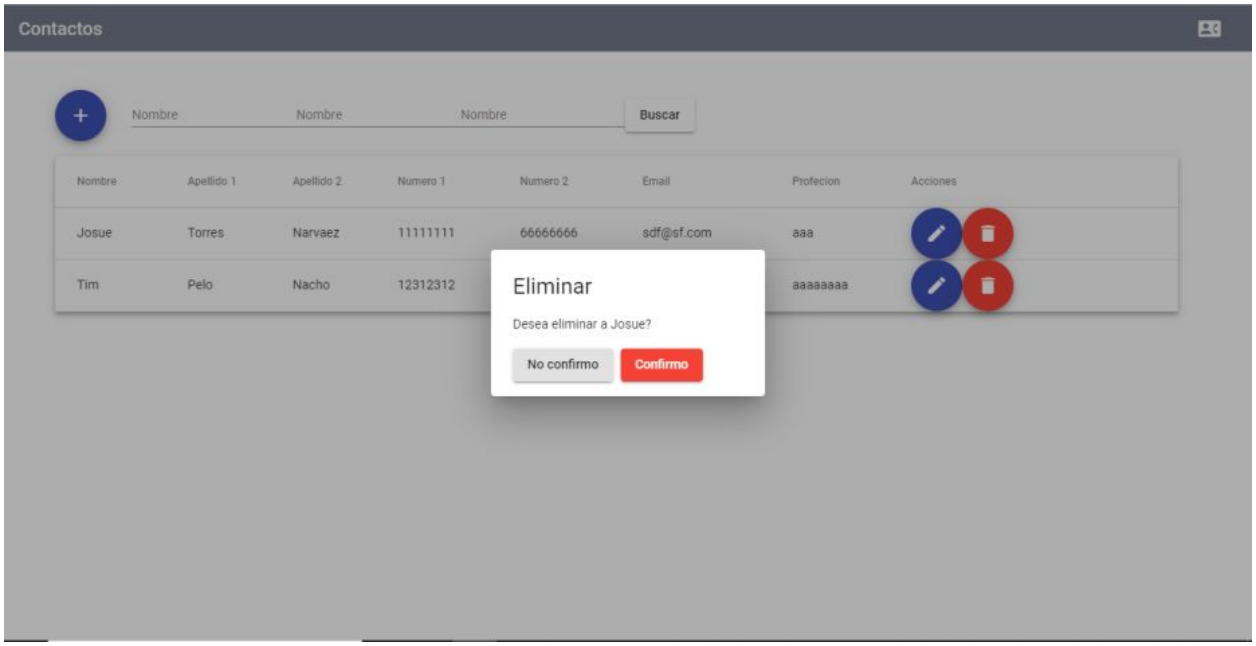


figura 23

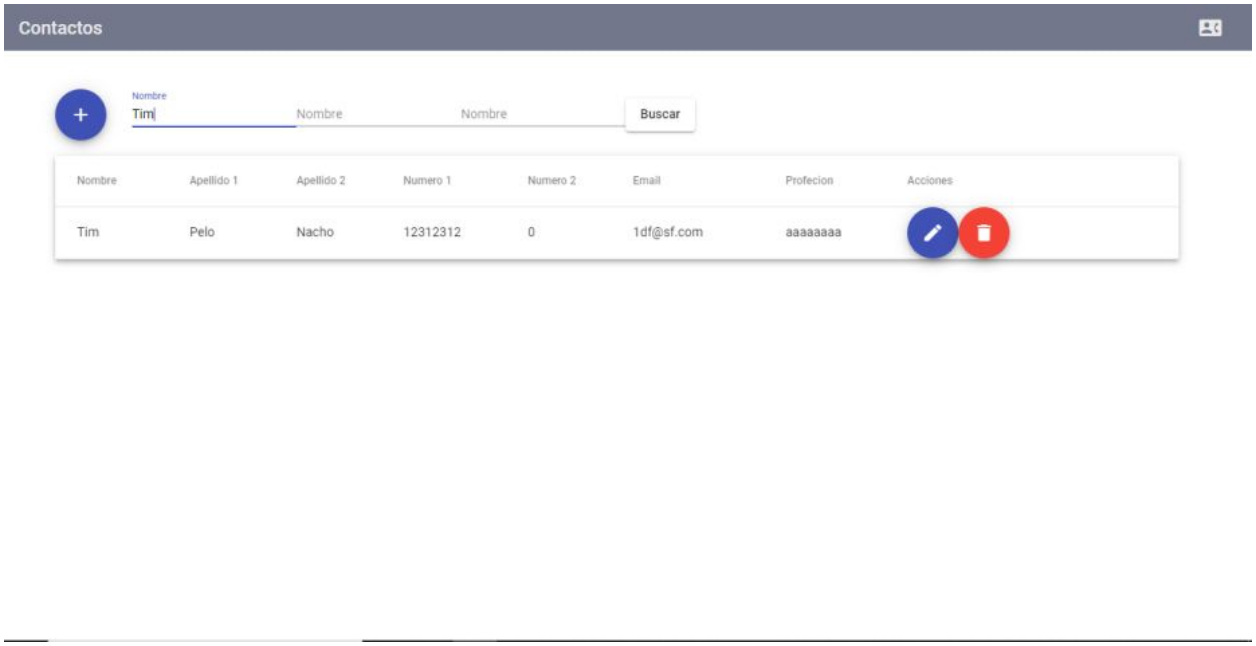


figura 24

Para la instalación de la aplicación web debe clonar o descargar el proyecto del repositorio github https://github.com/Macotosama/Investigacion.git**.** Una vez que se haya descargado o clonado se recomienda instalar la librería de angular matirial https://material.angular.io/guide/getting-started dentro del proyecto, abrimos la carpeta donde está guardado el proyecto con la terminal(puede ser la terminal de Node.js o la de su sistema operativo o la de su editor de código) y escribiremos la instrucción >ng s y esperaremos que se compile, !y listo ya podremos hacer uso de este proyecto¡.

Repositorio GIT

https://github.com/Macotosama/Investigacion.git

https://github.com/Macotosama/mongoCrud.git

Bibliografía

Pérez, B. (22 de 12 de 2018). *youtube*. Obtenido de youtube: https://www.youtube.com/watch?v=B9qy31FjdBg

Pérez, B. (22 de 12 de 2018). *YOUTUBE*. Obtenido de YOUTUBE: https://youtu.be/B9qy31FjdBg