TECNOLOGICO DE ESTUDIOS SUPERIORES DE ECATEPEC

DIVISIÓN DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

Proyecto

Módulo QR

Integrantes:
Cano Tabla Aarón Israel
Juárez Mendoza Leticia Guadalupe
Tlapanco Melo Cesar Gabriel

Profesora:
Griselda Cortes Barrera

Grupo: 5851

(ESP_SID) BASE DE DATOS PARA DISPOSITIVOS MOVILES

ECATEPEC DE MORELOS, EDO. DE MÉXICO FEBRERO / 2022

Antes de comenzar con los requerimientos veamos ¿qué es un código QR y para qué sirve?

Un código QR (abreviado de Quick Response code) es un tipo de código de barras de matriz (o código de barras bidimensional) diseñado por primera vez en 1994 para la industria automotriz en Japón. Un código de barras es una etiqueta óptica legible por máquina que contiene información sobre el artículo al que está adherido. En la práctica, los códigos QR a menudo contienen datos para un localizador, identificador o rastreador que apunta a un sitio web o aplicación. Un código QR utiliza cuatro modos de codificación estandarizados (numérico, alfanumérico, byte / binario y kanji) para almacenar datos de manera eficiente.

El sistema Quick Response se hizo popular fuera de la industria automotriz debido a su rápida legibilidad y mayor capacidad de almacenamiento en comparación con los códigos de barras UPC estándar. Las aplicaciones incluyen seguimiento de productos, identificación de artículos, seguimiento del tiempo, gestión de documentos y marketing general.

Requerimientos y necesidades del módulo a trabajar

Antes de comenzar, necesita algunos requisitos previos:

- Conocimientos básicos de TypeScript. En particular, la familiaridad con conceptos orientados a objetos como clases y decoradores de TypeScript.
- Una máquina de desarrollo local con **Node 10+**, junto con **NPM 6+** instalado. La CLI de Angular requiere un nodo como la mayoría de las herramientas frontend en la actualidad. Simplemente puede ir a la página de descargas del sitio web oficial y descargar los binarios para su sistema operativo. También puede consultar las instrucciones específicas de su sistema sobre cómo instalar Node usando un administrador de paquetes. Sin embargo, la forma recomendada es usar NVM - Node Version Manager - un script bash compatible con POSIX para administrar múltiples versiones activas de Node.js.

Problemática

Crear un proyecto desde cero en Angular que contengan diferentes funcionalidades entre ellas surge la necesidad de investigar herramientas incluyentes que permitan generar nuevos métodos de validación de información a un bajo costo tales como los códigos QR que se basan en una tecnología que almacena una cadena de información que puede ser legible desde un escáner o un teléfono inteligente.

Objetivos generales

Construir un prototipo de códigos QR en Angular para saber cómo se estructuran y funcionan dentro de la aplicación.

Objetivos específicos

Generar un código único de identificación por medio de la representación bidimensional QR que permita facilitar la verificación de la originalidad del mismo.

Implementar una funcionalidad de lectura de información con código QR a través del uso de una cámara que permita la visualización de dicha información original.

Diseño: Describir de manera textual y gráfica la propuesta de solución de lo descrito anteriormente

Algunas propiedades del código QR

El código QR ofrece varias propiedades que se pueden asociar con qr-code> directiva.

- Valor: El tipo de propiedad toma un valor Sting y convierte String en código QR; El valor predeterminado es ".
- Talla: Esta propiedad configura la altura y el ancho del componente de código QR, el tipo de accesorio es número y el valor predeterminado se establece en 100.
- Nivel: El tipo de accesorio es String; el valor predeterminado se establece en L, utilizado principalmente para el nivel de corrección QR ('L', 'M', 'Q', 'H').
- Antecedentes: El tipo de accesorio es String; el valor predeterminado es blanco. Se utiliza principalmente para configurar el color de fondo.
- BackgroundAlpha: Se utiliza para establecer la opacidad del fondo, definida en forma numérica, y el valor predeterminado es 1.0.
- Primer plano: Se utiliza para ajustar el color de primer plano, el tipo de propiedad es Cadena y el valor predeterminado es negro.
- Primer plano Alfa: Configura la opacidad del primer plano. El valor predeterminado es 1.0 y se define en forma numérica.

- Mímica: El tipo de valor es String, que se utiliza para configurar el tipo de mime para la imagen de salida. Además, el valor predeterminado es image / png.
- Relleno: Configura principalmente el relleno en código QR; el número es el tipo de propiedad con el valor predeterminado establecido en nulo.
- Lienzo: El tipo de valor es booleano y se usa para generar un elemento de lienzo si se establece en verdadero. Sin embargo, el valor predeterminado se establece en falso.

Para construir el código QR es necesario seguir una serie de pasos que se enlistaran a continuación:

PASOS:

Se tiene que crear un nuevo proyecto en angular, instalación de routing y formato CSS, seguido de eso instalamos el Bootstrap con el siguiente comando "*npm install --save bootstrap jquery @popperjs/core*"

Esperamos y nos vamos a abrir nuestro proyecto para editar el angular.json, una vez encontrado el json, nos vamos a styles y scripts para inicializar el Bootstrap con lo siguiente

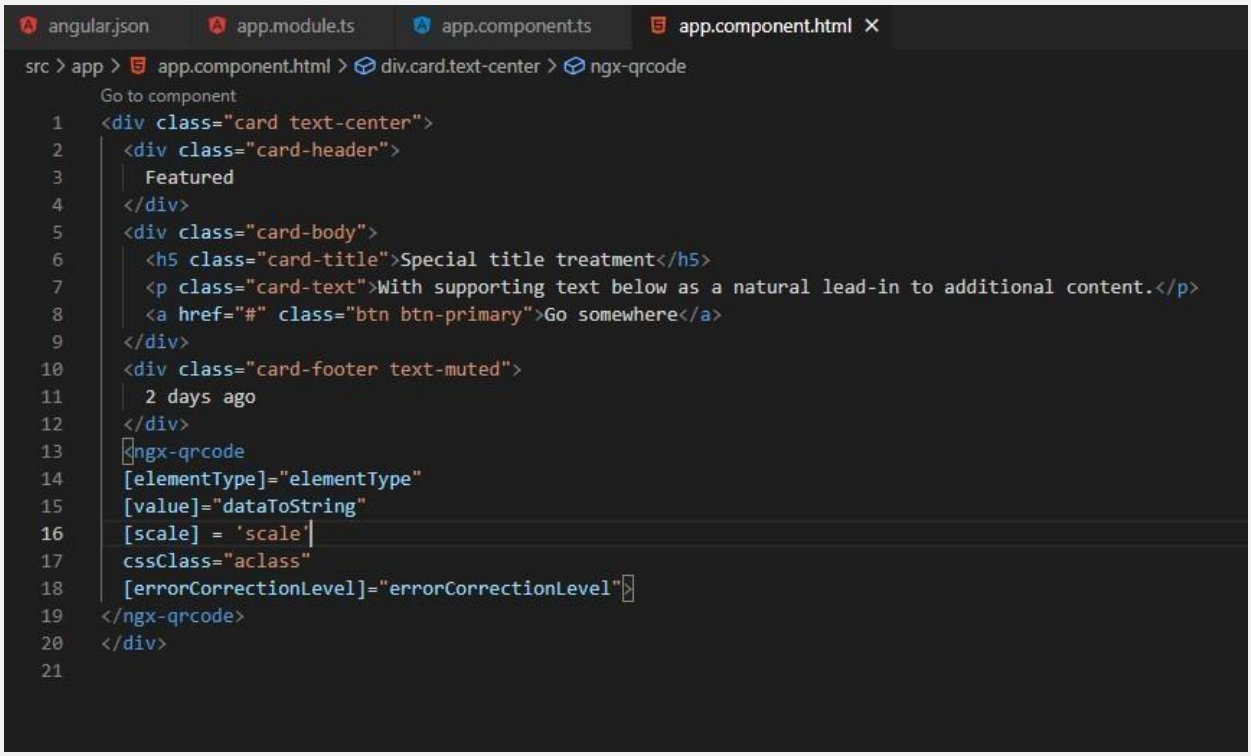
```
“"styles": [
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "src/styles.scss"
],
"scripts":                                    [
  "node_modules/jquery/dist/jquery.min.js",
  "node_modules/@popperjs/core/dist/umd/popper.min.js",
  "node_modules/bootstrap/dist/js/bootstrap.min.js"
]”
```

Una vez instalado y editado el json, necesitamos instalar la librería que genera el Qr el cual será con el siguiente comando, hay que recordar que todo esto es dentro de la carpeta que se creo a la hora de hacer el proyecto.

‘‘npm install @techiediaries/ngx-qrcode –save”

Lo siguiente es irnos a nuestro app.module.ts y importamos la librería como se muestra acontinuacion.

Agregamos una Card de Bootstrap para simular el Qr y tengamos todo listo, así que nos vamos a nuestro app.component.html y hacemos la card y anexamos dentro de ella la etiqueta de nuestro QR



Usamos lo que la librería nos deja usar el [elemntType] nos sirve para decirle que tipo de qr es el que vamos a necesitar que hay 3 que son una imagen, canvas y url.
[value] = es el parámetro que nosotros le vamos a pasar, en este caso utilizamos un string, ya que guardaremos la información en un json para poder visualizar la información del usuario mediante un qr
[scale] = este nos ayuda a redimensionar el qr al tamaño que nosotros queramos en números enteros
cssClass = es el estilo propio del Qr
[errorCorrectionLevel] = es el nivel de corrección que nosotros esperamos
Y todo esto lo vamos a establecer en el typescript de app.components.ts

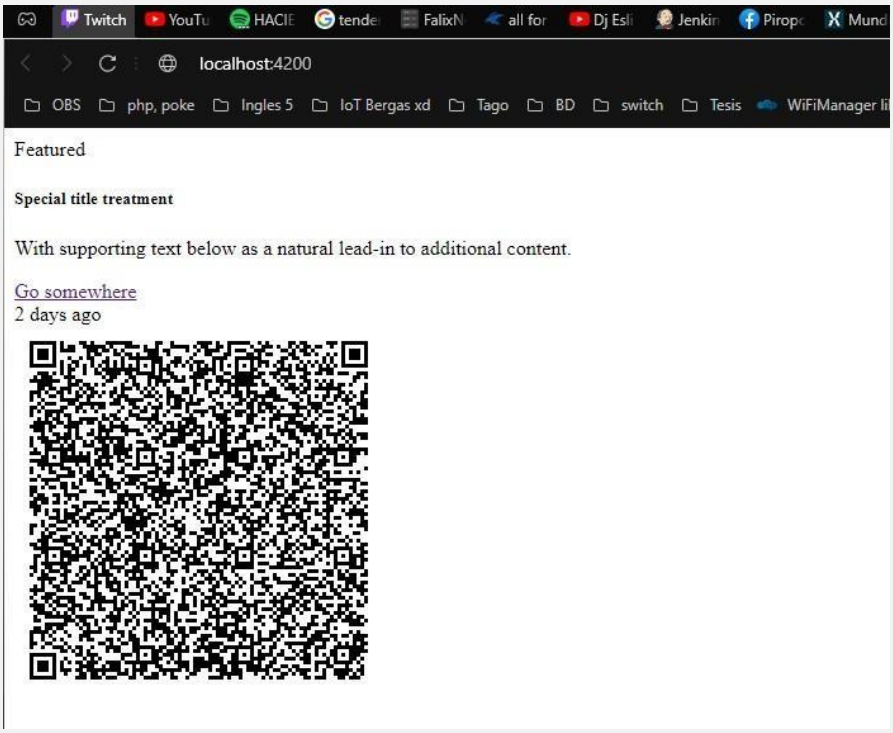
angular.json | app.module.ts | app.component.ts | app.component.html

src > app > app.component.ts > AppComponent > scale

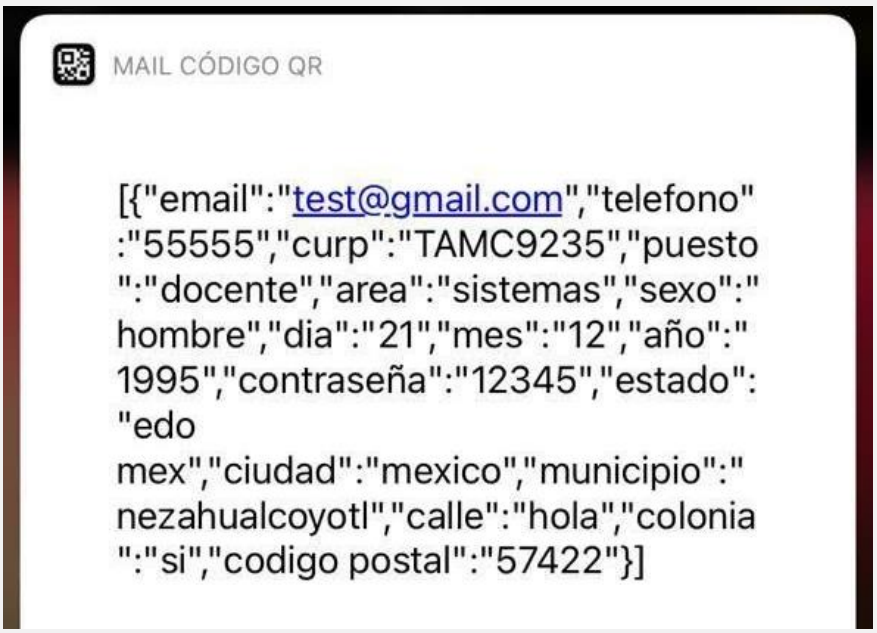
```ts
import { Component } from '@angular/core';
import { NgxQrcodeElementTypes, NgxQrcodeErrorCorrectionLevels } from '@techiediaries/ngx-qrcode';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'projectQR';

  data = [{
    'email': 'test@gmail.com',
    'telefono': '55555',
    'curp': 'TAMC9235',
    'puesto': 'docente',
    'area':'sistemas',
    'sexo': 'hombre',
    'dia': '21',
    'mes': '12',
    'año': '1995',
    'contraseña': '12345',
    'estado': 'edo mex',
    'ciudad': 'mexico',
    'municipio': 'nezahualcoyotl',
    'calle': 'hola',
    'colonia': 'si',
    'codigo postal' : '57422'

  }]
  dataToString = JSON.stringify(this.data);
  url = 'dataToString';
  profile = 'routeToMyProfile';
  elementType = NgxQrcodeElementTypes.URL;
  errorCorrectionLevel = NgxQrcodeErrorCorrectionLevels.HIGH;
  scale = 3
  value = this.url + this.profile;
}
```

Aquí en la clase Appcomponent, inicializamos el titulo de nuestro qr, seguido de la data o los datos que tendrá el json a partir de el Excel propuesto por mis compañeros de esa parte, una vez teniendo el ejemplo, usamos una librería llamada JSONPACK, la cual nos ayudara a convertir el JSON en un string para poder transformarlo a un qr, con la siguiente línea en consola lo instalamos: "*npm install jsonpack*"

Así que creamos un dataToString para pasar de json a un string y la variable url fue creada para instar un Url de pagina pero en este caso para fines prácticos se remplazo con el dataToString.
Profile: es el nombre que tendrá el Qr solo fue escrito así por fines prácticos
Y las demás variables fueron explicadas en anteriores pasos y con ellos tendríamos nuestro Qr funcionando y creado mediante un JSON.



Y si lo escaneamos con un celular obtendríamos la siguiente cadena:



Ahora tenemos que restructurar todo el código para poner routing y con ello poder hacer un componente mucho más limpio y que a la hora de que se agregue al proyecto final no haya problema.

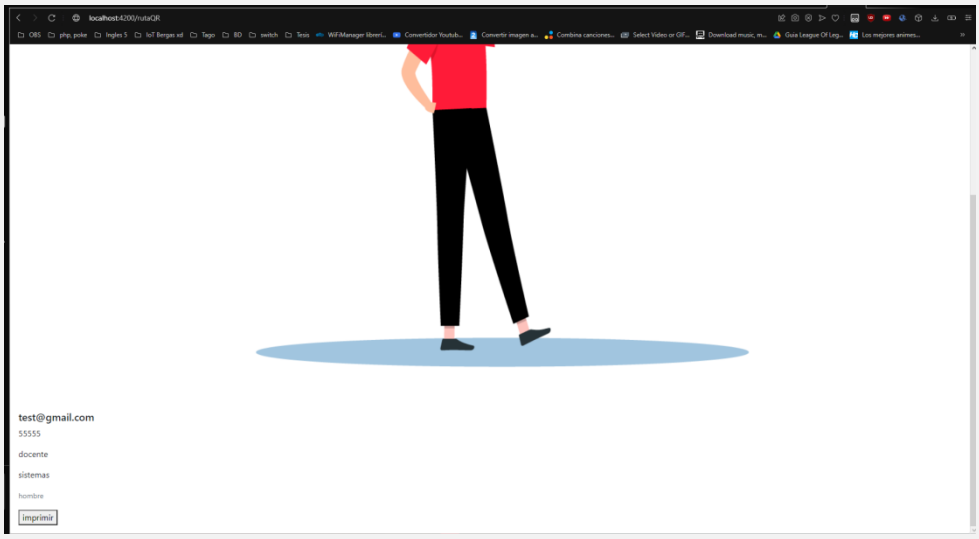
Hacemos de ng g c qr y ng g c imprimir

El qr tendrá lo que será la instancia en donde nosotros generaremos los QRs con el fin de tener solo la información relevante e importante a la hora de generar el gafete con su QR.

Una vez teniendo todo esto en mente ruteamos los componentes anteriores en el app-routing.module.ts

```typescript
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { ImprimirComponent } from './imprimir/imprimir.component';
import { QrComponent } from './qr/qr.component';

const routes: Routes = [
  {
    path: 'rutaQR', component: QrComponent
  },
  {
    path: 'rutaImprimir', component: ImprimirComponent
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Teniendo esto vamos a nuestro HTML del QR para empezar agregar un card en donde se vea una pre visualización de nuestro gafete con todo y su foto del empleado que generará un gafete de su propia sesión, esto solo sirve como ejemplo ya que no tenemos algo como para personalizar todavía debido a que no tenemos una cuenta propia en esta simulación.



Como se observa en la imagen anterior, tenemos un usuario que mediante un JSON del cual se extraen los datos importantes que son: el correo, numero, que puesto ocupa, de que área es y que sexo es, también podemos agregar el nombre de dicho personal, pero pensamos que es mejor que lo tengan en el QR para que tengan un poco más deprivacidad a la hora de portarlo y que en la entrada lo escaneen y sapan quien es el que ingresa y a donde va.

Dando con ello, usamos aquí un ciclo For para leer los datos y no andar copiando el Card cada que tengamos una nueva persona, obviamente una vez teniendo algo más sólido con una base de datos, se puede manejar mejor la información para que sea más personalizada y que cada persona tenga la opción de imprimir su gafete.

```
Go to component
<div class="card-group"> <!-- para comentar es ALT+SHIF+A  -->
  <!--  --><div class="card" *ngFor="let d of datos">  <!-- Aqui se usa un array que es un foreach -->
    <img [src]="d.foto" class="card-img-top" alt="..."> <!-- el src en corchetes hace alusion a que toma un array -->
    <div class="card-body">
      <h5 class="card-title">{{d.email}}</h5> <!-- otra manera de tomar un array -->
      <p class="card-text">{{d.telefono}}</p>
      <p class="card-text">{{d.puesto}}</p>
      <p class="card-text">{{d.area}}</p>
      <p class="card-text"><small class="text-muted">{{d.sexo}}</small></p>
      <button [routerLink]="['/imprimir/rutaImprimir']">imprimir</button>
      <!-- el array esta creado en datos-personales.component.ts -->
    </div>
  </div>
  <!--  <ngx-qrcode
  [elementType]="elementType"
  [value]="dataToString"
  [scale] = 'scale'
  cssClass="aclass"
  [errorCorrectionLevel]="errorCorrectionLevel">
</ngx-qrcode>-->
</div>
```

A continuación viene lo que se usa en el card para mostrar la información. En la siguiente imagense puede observar como tomamos los datos en el qr.component.ts

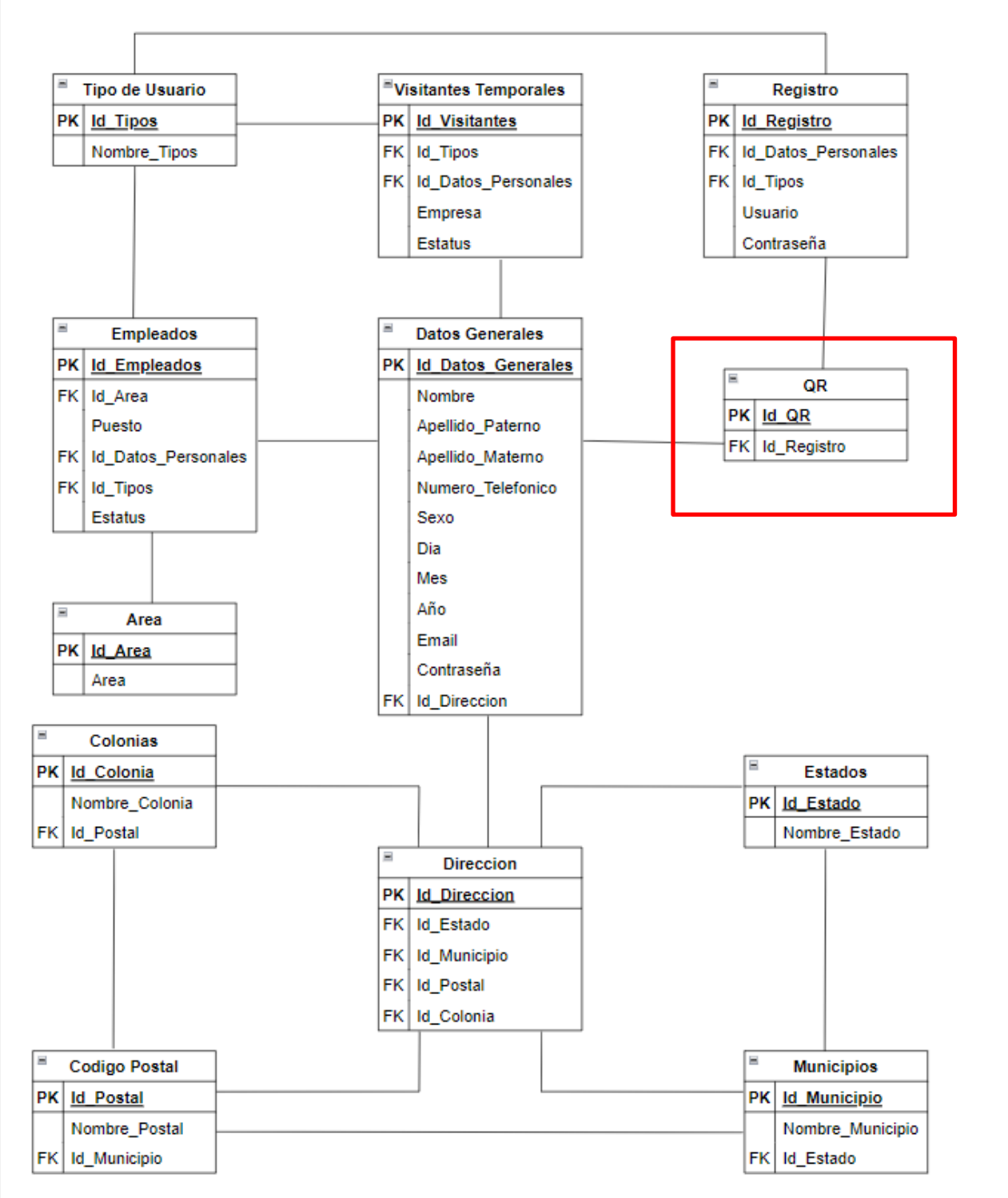
```
@Component({
  selector: 'app-qr',
  templateUrl: './qr.component.html',
  styleUrls: ['./qr.component.css']
})
export class QrComponent implements OnInit {

  datos: any [] = [
    {
    foto: './assets/persona3.png',
    email: 'test@gmail.com',
    telefono: '55555',
    puesto: 'docente',
    area:'sistemas',
    sexo: 'hombre',
    // 'curp': 'TAMC9235',
    //'dia': '21',
    //'mes': '12',
    //'año': '1995',
    //'contraseña': '12345',
    //'estado': 'edo mex',
    //'ciudad': 'mexico',
    //'municipio': 'nezahualcoyotl',
    //'calle': 'hola',
    //'colonia': 'si',
    //'codigo postal' : '57422'

  },
    {
    foto: './assets/persona4.png',
    email: 'test1@gmail.com',
    telefono: '111',
    puesto: 'alumno',
    area:'sistemas',
    sexo: 'mujer',
    }
];
```

Una vez teniendo eso, necesitaremos que el botón imprimir que se encuentra en el HTML se routee con el componente imprimir para que con ello se pueda empezar a imprimir y podamos simular la impresión.

```
<button [routerLink]="['/imprimir/rutaImprimir']">imprimir</button>
```

Tecnológico de Estudio Superiores de Ecatepec

Diagrama E-R
~MongoDb~



Creación de BD con mongo altlas.

```
"scripts": {
  "dev": "nodemon src/index.js",
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "watch": "ng build --watch --configuration development",
  "test": "ng test"

},
"private": true,
"dependencies": {
  "@angular/animations": "~12.2.0",
  "@angular/common": "~12.2.0",
  "@angular/compiler": "~12.2.0",
  "@angular/core": "~12.2.0",
  "@angular/forms": "~12.2.0",
  "@angular/platform-browser": "~12.2.0",
  "@angular/platform-browser-dynamic": "~12.2.0",
  "@angular/router": "~12.2.0",
  "@popperjs/core": "^2.11.0",
  "@techiediaries/ngx-qrcode": "^9.1.0",
  "bootstrap": "^5.1.3",
  "express": "^4.17.2",
  "jquery": "^3.6.0",
  "jsonpack": "^1.1.5",
  "mongoose": "^6.1.8",
  "morgan": "^1.10.0",
  "nodemon": "^2.0.15",
  "rxjs": "~6.6.0",
  "tslib": "^2.3.0",
  "underscore": "^1.13.2",
  "zone.js": "~0.11.4"
},
"devDependencies": {
  "@angular-devkit/build-angular": "~12.2.5",
  "@angular/cli": "~12.2.5",
  "@angular/compiler-cli": "~12.2.0",
  "@types/jasmine": "~3.8.0",
```

Se tiene que instalar Morgan, mongoose, nodemon, express y underscore para el manejor, conexión, implementación, altas, bajas, etc.

Y se crea el script de “nodemon src/index.js” con el fin de que se este actualizando cada que nosotros guardemos.

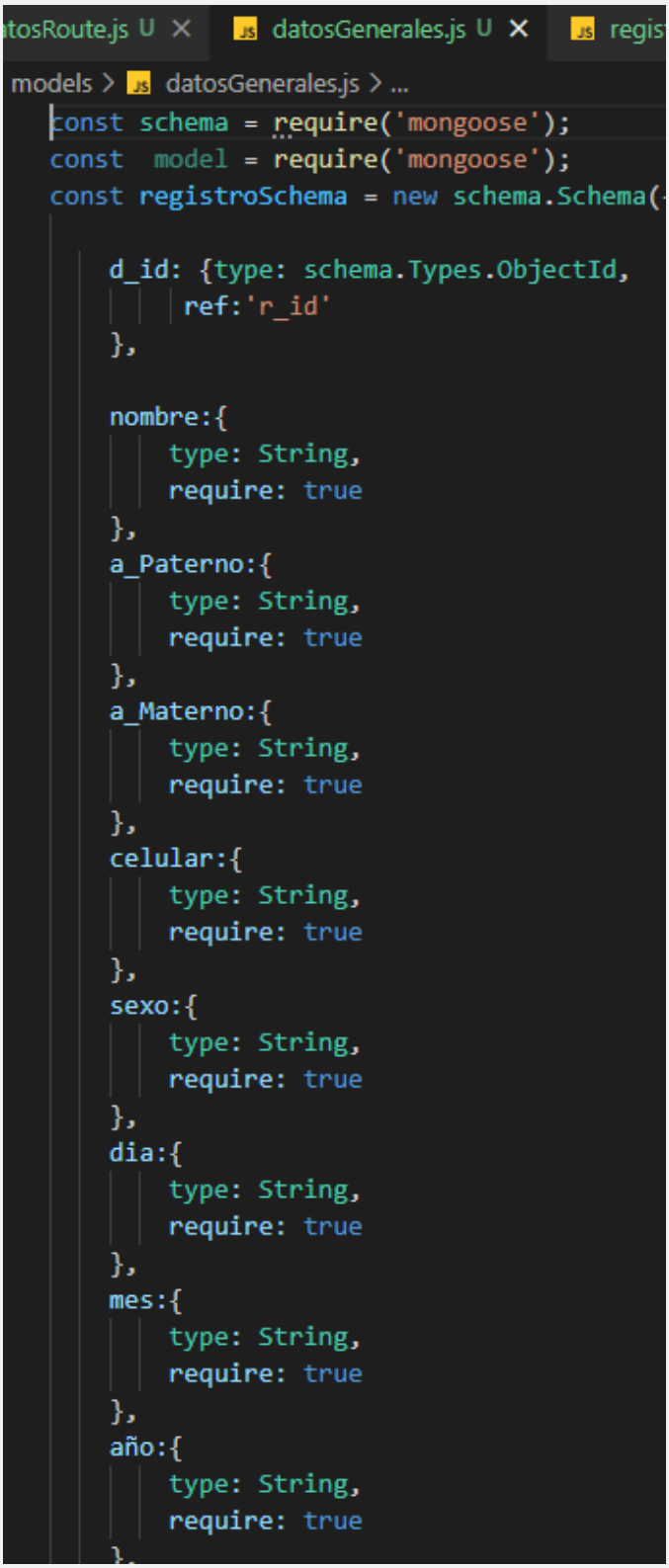
datosRoute.js U | datosGenerales.js U | registro.js U | qr.js U | db.js U

configuration > db.js > ...

```
1  const mongoose = require('mongoose');
2  const cadena = 'mongodb+srv://tlaps:12345@clustertest.de0wf.mongodb.net/DatabaseQR';
3
4  mongoose.connect(cadena, {
5      useNewUrlParser:true,
6      useUnifiedTopology:true
7  })
8
9  .then(dato =>{
10     console.log('conectado');
11 })
12 .catch(error =>{
13     console.log(error);
14 })
```

Creación de JS con la finalidad de hacer la estructura en JSON para la base de datos.

Datos generales.



Tecnológico de Estudio Superiores de Ecatepec

Registro

QR

```
c > models > qr.js > ...
1   const schema = require('mongoose');
2   const  model = require('mongoose');
3   const registro = require('./registro');
4   const qrSchema = new schema.Schema({
5
6       q_id: {type: schema.Types.ObjectId,
7           ref:'id'
8       },
9
10      pk:{
11          type: schema.Schema.Types.ObjectId,
12          ref: "id"
13      }
14  })
15
16  const qr = schema.model ('qr', qrSchema);
17  module.exports = qr;
```

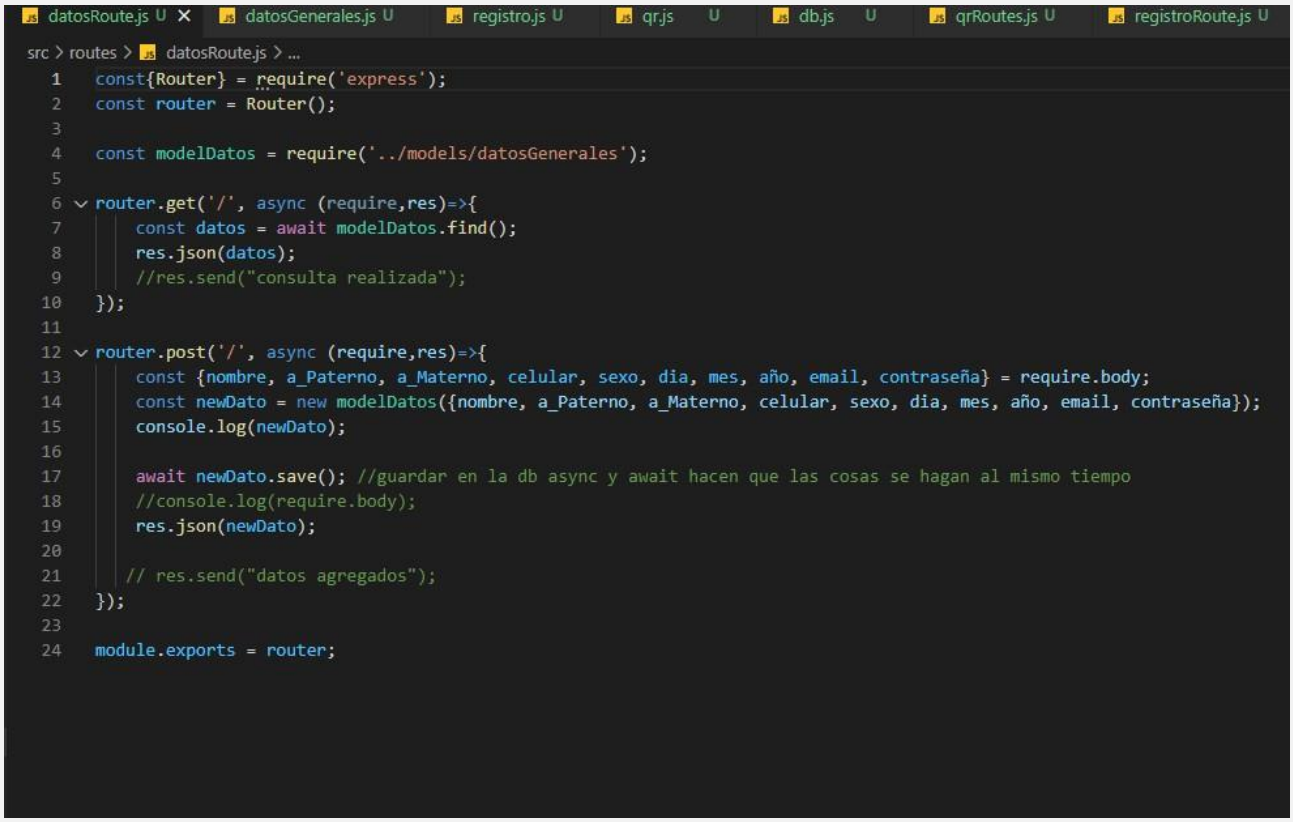
Rutas de los js para la conexión de BD por medio de Postman

```
datosRoute.js U | datosGenerales.js U | registro.js U | qr.js U | db.js U | qrRoutes.js U | registroRoute.js U
src > index.js > app.listen() callback
1   const express = require('express');
2   const app = express();
3   const morgan = require('morgan');
4
5   require('../configuration/db')
6
7   //configuraciones
8   app.set('port', process.env.PORT || 1919); //usar un puerto en especifico o el que nosotros damos
9
10  app.set('json spaces', 2); // darle espacios al json en el explorador
11
12  app.use(morgan('dev')); // para que se reinicie el servidor desde el package.json automaticamente
13
14  app.use(express.urlencoded({extended:false}));
15  app.use(express.json()); //aqui le decimos que se trabaje con JSON
16
17  app.use(require('./routes/main'));
18  app.use('/api/movies', require('./routes/movies'));//cambiamos el nombre de la ruta
19  app.use('/api/consulta', require('./routes/userRoute'));
20  app.use('/api/guardar', require('./routes/userRoute'));
21
22  //QR rutas
23
24  app.use('/api/datos', require('./routes/datosRoute'));
25  app.use('/api/registro', require('./routes/registroRoute'));
26  app.use('/api/qr', require('./routes/qrRoutes'));
27
28
29  app.listen(app.get('port'), ()=>{
30      //codigo ascci de backstick alt+96
31      console.log(`servidor en puerto ${app.get('port')}`);
32  });
33
```

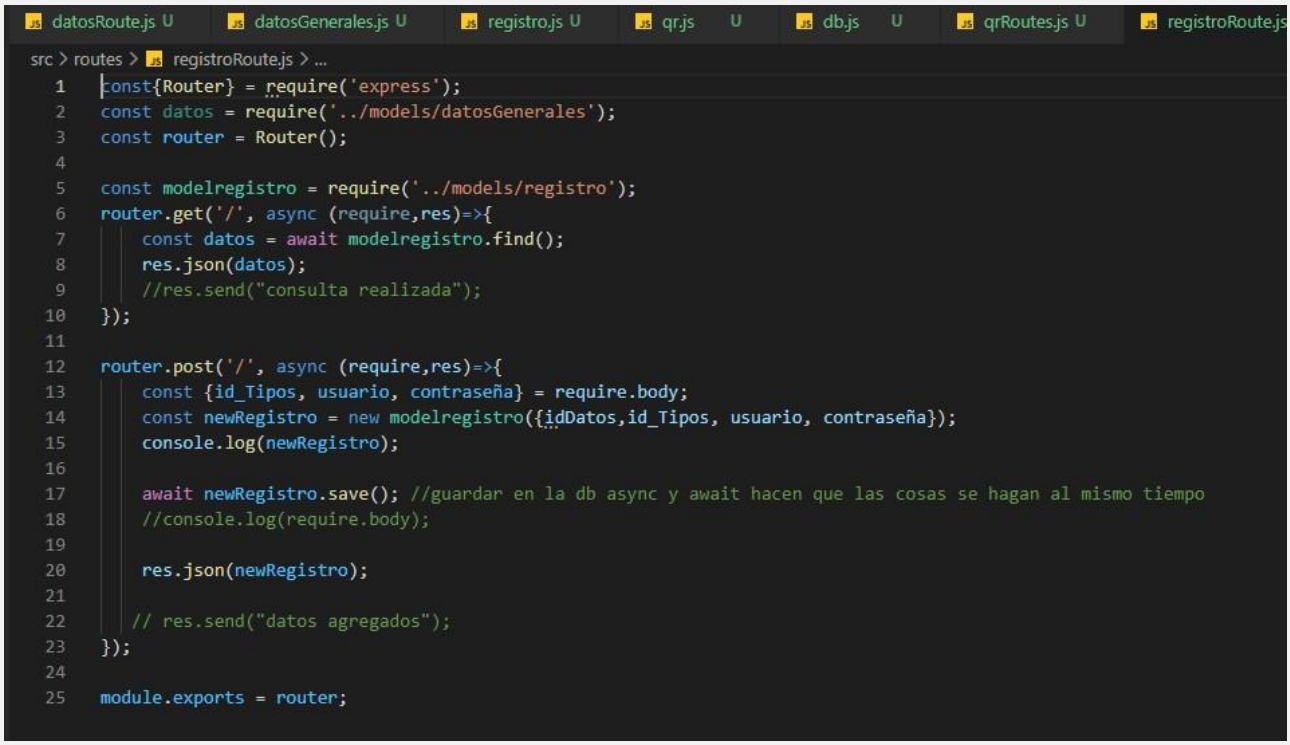
Tecnológico de Estudio Superiores de Ecatepec

Altas y Consultas a la BD.

Datos

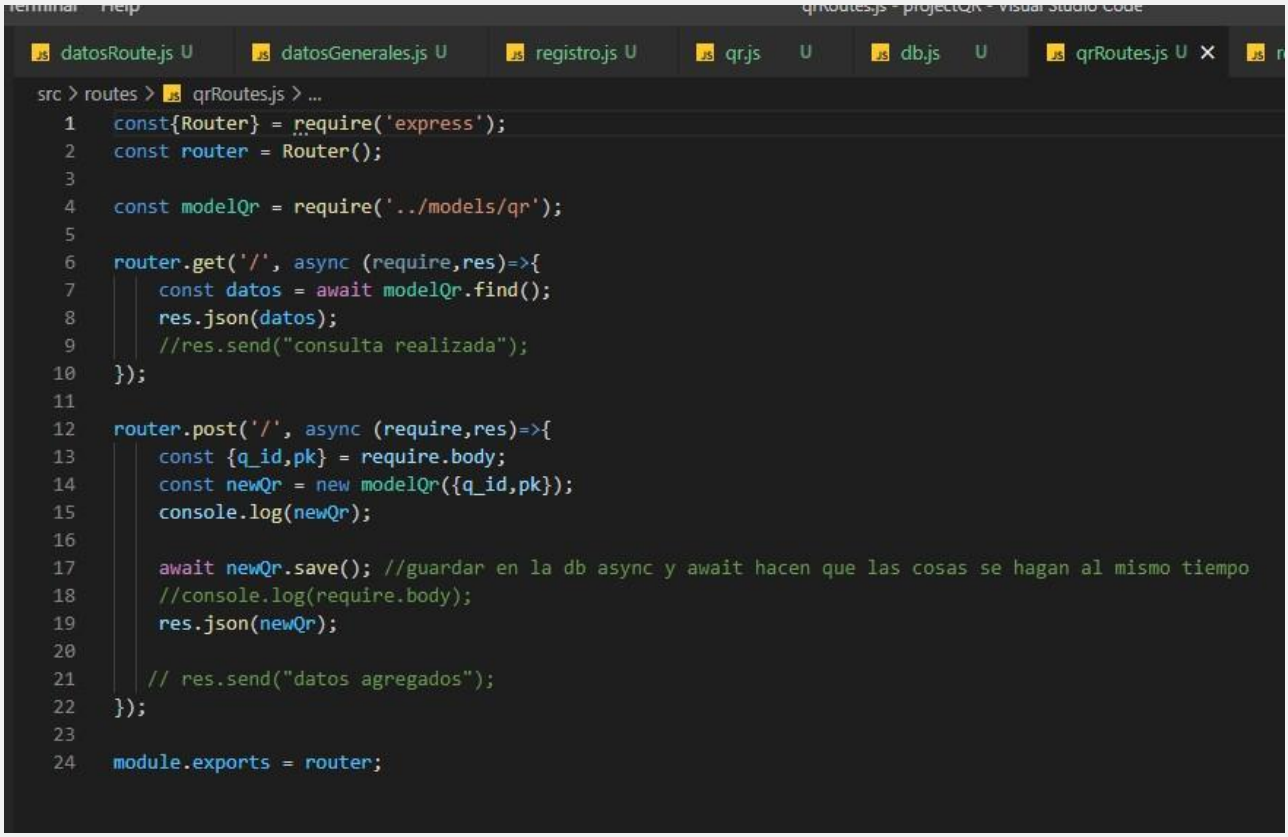


Registro

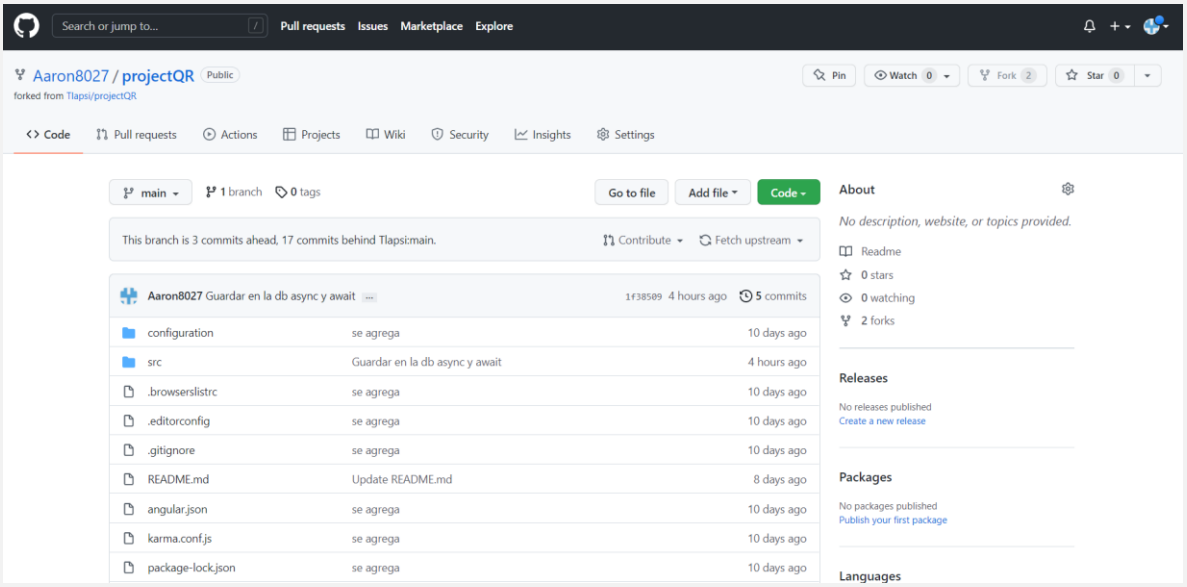


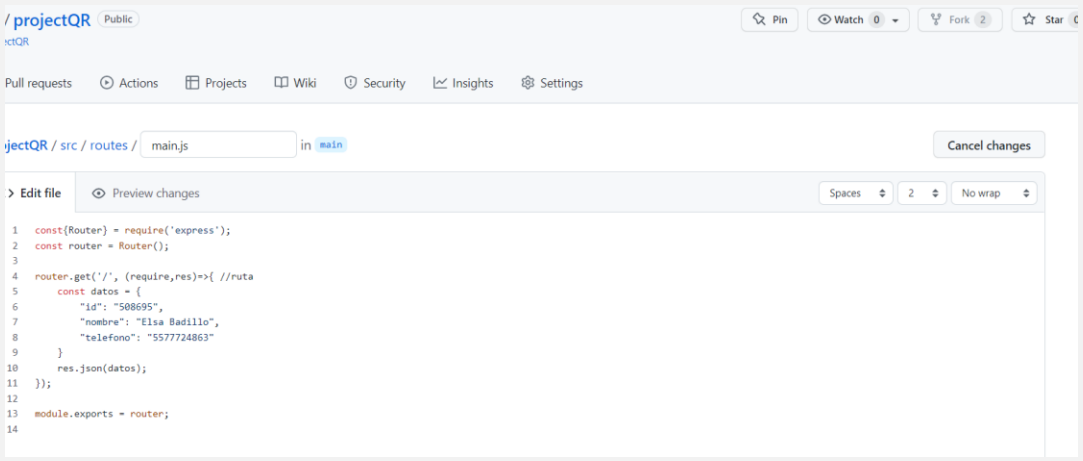
Tecnológico de Estudio Superiores de Ecatepec

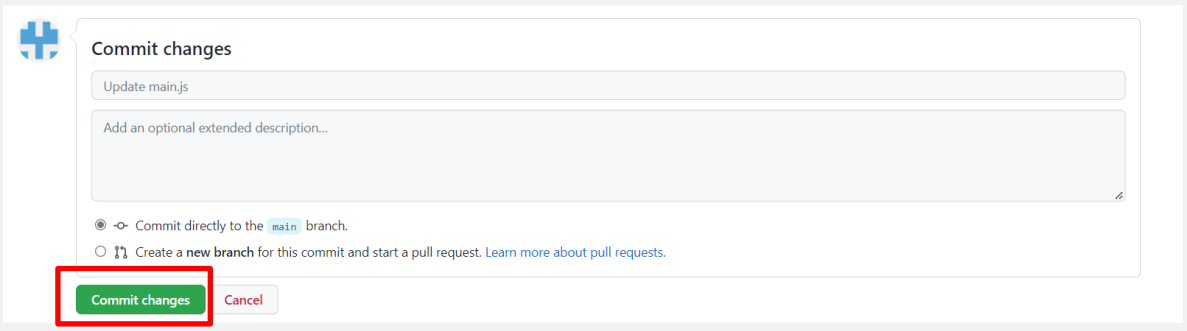
QR



Empleamos también una herramienta llamada Git Hub para facilitar el trabajo en equipo y realizar modificaciones simultáneas por medio de la elaboración de commits, que posteriormente se suben al proyecto y así se actualizan los repositorios de cada integrante del equipo.





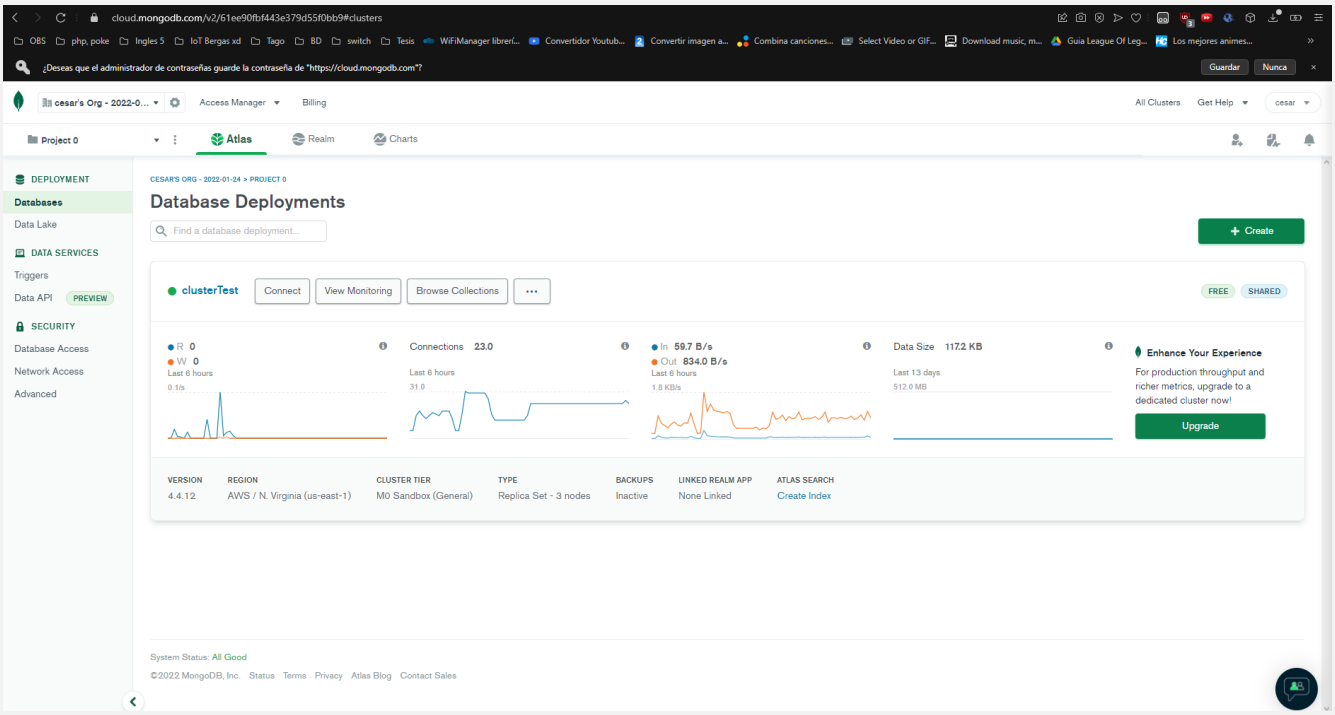


Tecnológico de Estudio Superiores de Ecatepec

Visualización en Postman, MongoDB Atlas y MongoDB Compass

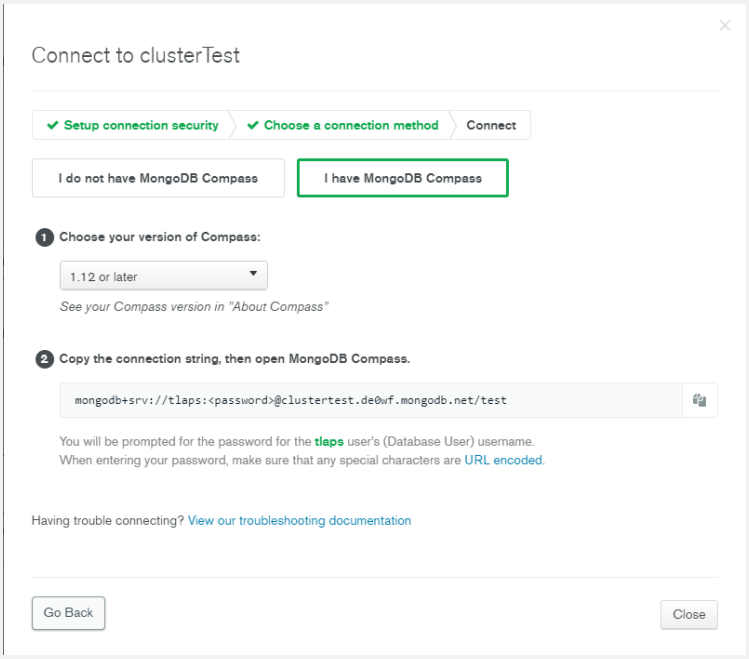
-Mongo Atlas

Como podemos ver, se creo una cuenta en mongoDB Atlas con el fin de tener un Cluster el cual nos ayudará con la conexión entre mongoDB Compass y Postman para poder hacer, Get, Post, Delete en sus respectivas colleciones que nosotros creemos mediante los JS

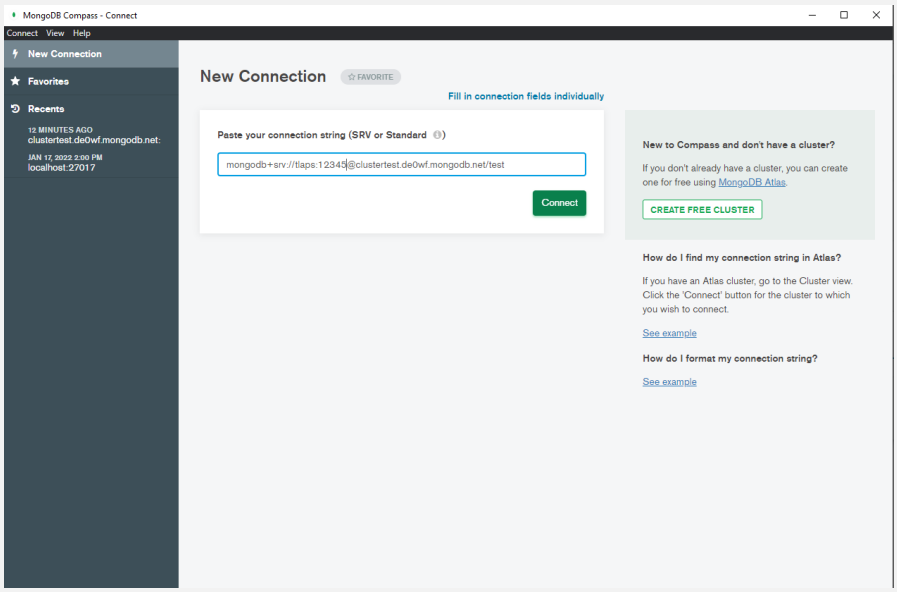


MongoDB es una base de datos orientada a documentos. Esto quiere decir que en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

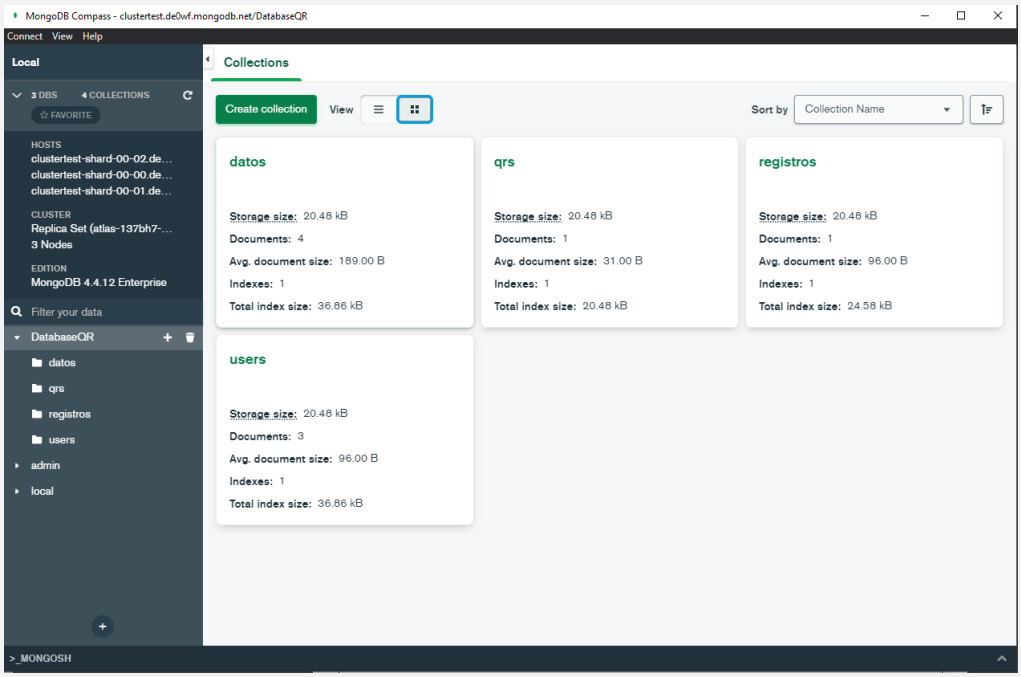
A la hora de conectar mongoDB Compass con el Cluster es necesario que tengamos el URL encoded que nos servira con esto hacer la conexion en donde nosotros queramos, para ellos necesitamos un password el cual es obtenido en cuanto se crea al Cluster

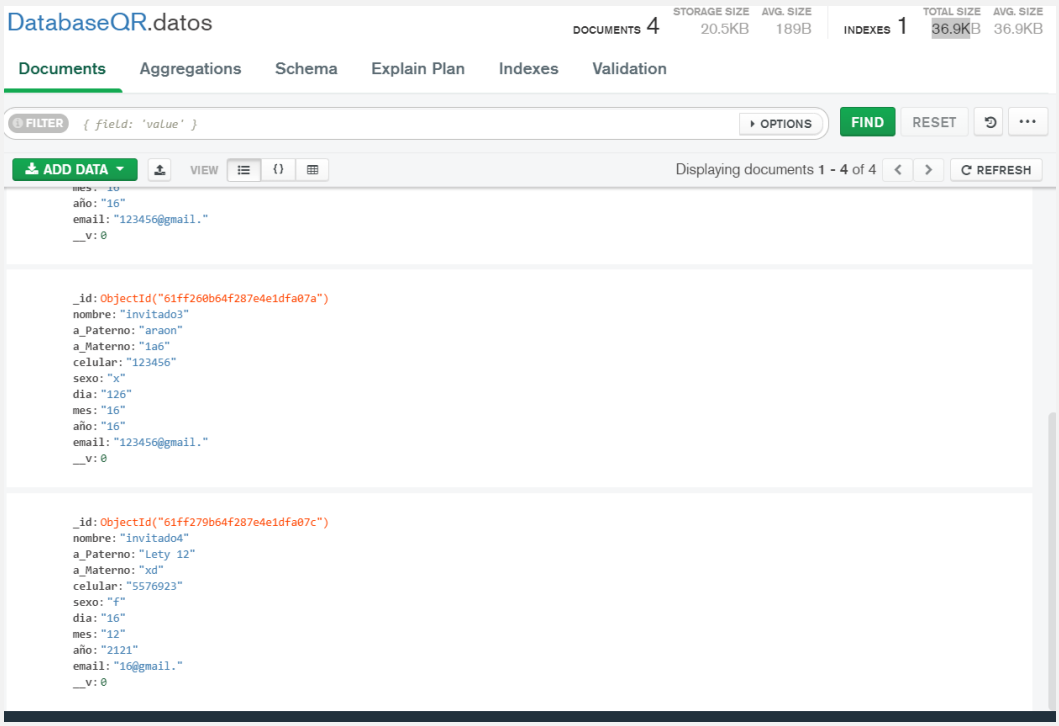


En cuanto nosotross abrimos el mongoDB Compass nos pedira que ingresemos la conexion string y el URL encoded que nos dio Atlas, lo usamos aquí con su contraseña y con ello podremos ingresar a la Base de Datos y poder ver las colleciones que tenemos.



-MongoDB

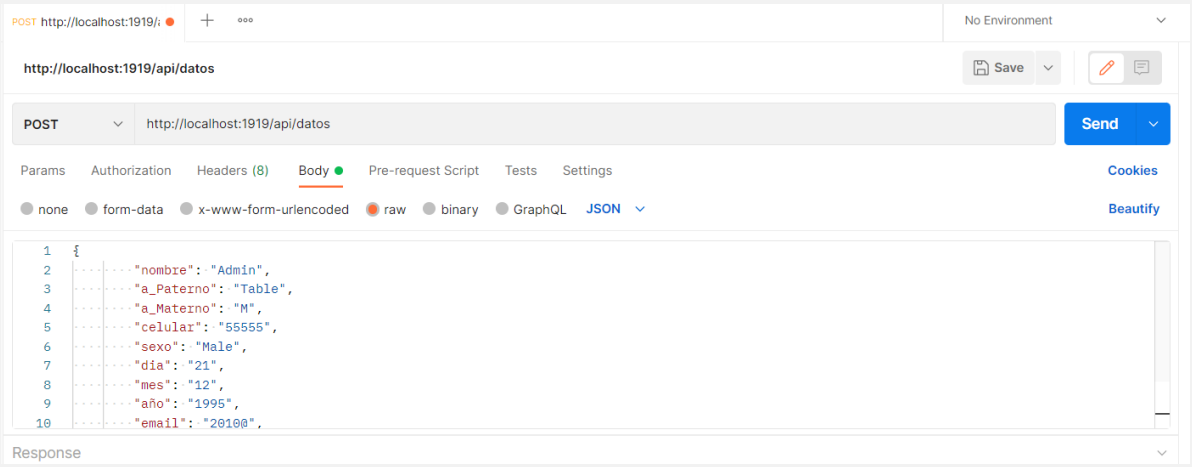

MongoDB Compass - clustertest.de0wf.mongodb.net/DatabaseQR
Connect View Help
Local
3 DBS 4 COLLECTIONS
FAVORITE
HOSTS
clustertest-shard-00-02.de...
clustertest-shard-00-00.de...
clustertest-shard-00-01.de...
CLUSTER
Replica Set (atlas-137bh7-...
3 Nodes
EDITION
MongoDB 4.4.12 Enterprise
Filter your data
DatabaseQR
datos
qrs
registros
users
admin
local
>_MONGOSH
Collections
Create collection
View
Sort by
Collection Name
datos
Storage size: 20.48 kB
Documents: 4
Avg. document size: 189.00 B
Indexes: 1
Total index size: 36.86 kB
qrs
Storage size: 20.48 kB
Documents: 1
Avg. document size: 31.00 B
Indexes: 1
Total index size: 20.48 kB
registros
Storage size: 20.48 kB
Documents: 1
Avg. document size: 96.00 B
Indexes: 1
Total index size: 24.58 kB
users
Storage size: 20.48 kB
Documents: 3
Avg. document size: 96.00 B
Indexes: 1
Total index size: 36.86 kB

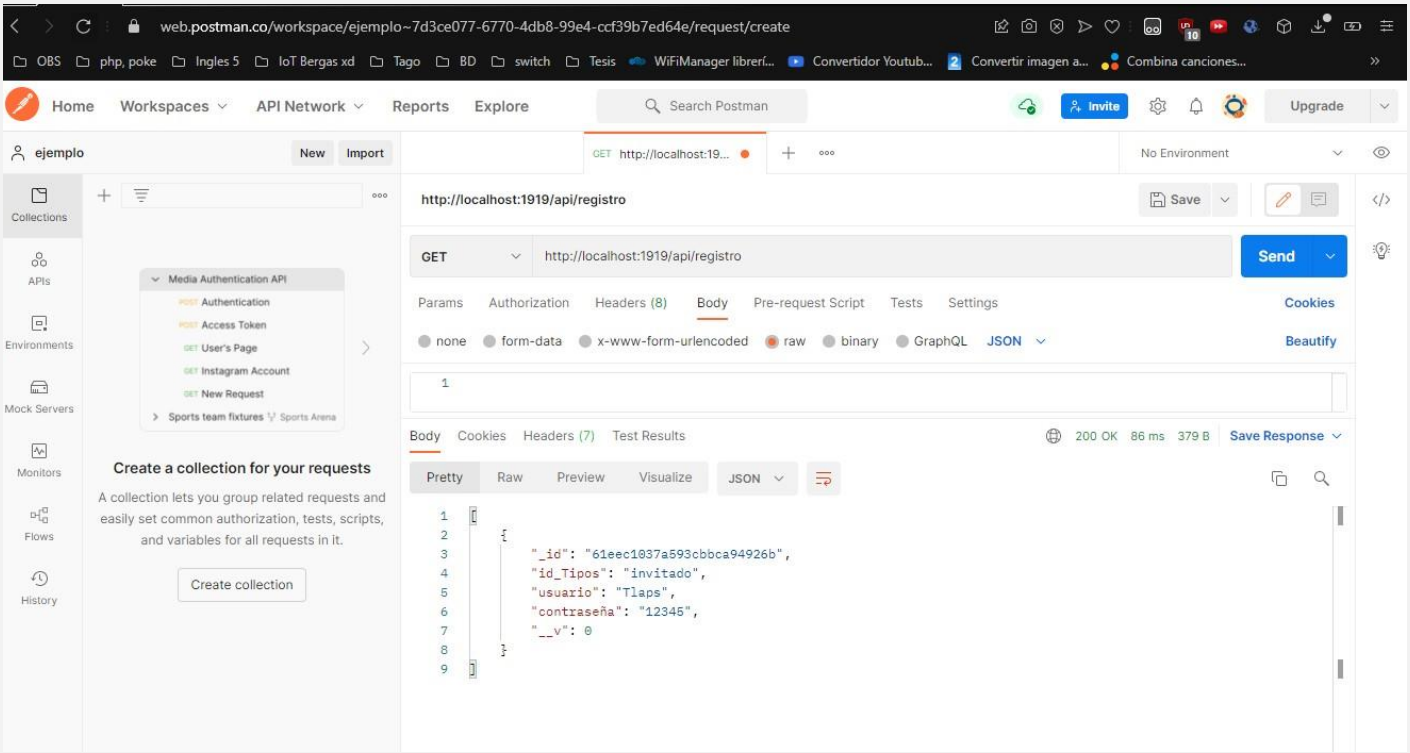

DatabaseQR.datos
DOCUMENTS 4
STORAGE SIZE 20.5KB
AVG. SIZE 189B
INDEXES 1
TOTAL SIZE 36.9KB
AVG. SIZE 36.9KB
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
FILTER { field: 'value' }
OPTIONS
FIND
RESET
ADD DATA
VIEW
Displaying documents 1 - 4 of 4
REFRESH
año: "16"
email: "123456@gmail."
__v: 0
_id: ObjectId("61ff260b64f287e4e1dfa07a")
nombre: "invitado3"
a_Paterno: "araon"
a_Materno: "1a6"
celular: "123456"
sexo: "x"
dia: "126"
mes: "16"
año: "16"
email: "123456@gmail."
__v: 0
_id: ObjectId("61ff279b64f287e4e1dfa07c")
nombre: "invitado4"
a_Paterno: "Lety 12"
a_Materno: "xd"
celular: "5576923"
sexo: "f"
dia: "16"
mes: "12"
año: "2121"
email: "16@gmail."
__v: 0

Tecnológico de Estudio Superiores de Ecatepec

Postaman

Desde esta herramienta nos conectamos al Localhost para realizar operaciones de tipo POST y así enviar datos a la Base de conocimientos.





Conclusiones:

Concluimos que mediante la elaboración de este proyecto, adquirimos muchos conocimientos nuevos sobre lo que es el Cloud Computing, así como a utilizar BD de tipo NoSQL. Manejar una base de datos de éste tipo presentó varios retos, desde la cadena de conexión, hasta la consulta de datos, ya que como sabemos, trabajar una base de datos orientada a Documentos, es muy distinto al lenguaje estructurado que habitualmente manejamos. Pudimos aprender a programar en Angular, así como la manipulación de commits en los apartados de git hub que es una herramienta muy agradable para trabajar en equipo, implementamos diseño con el marco de trabajo de Bootstrap y logramos obtener una interfaz muy agradable con la generación de código QR correspondiente.

Éste proyecto nos ayudó a aprender a lo largo de su desarrollo primero el cómo trabaja Angular y como bien sabemos es de código abierto para poder crear aplicaciones en el cual implementamos la funcionalidad principal y con el conjunto de las bibliotecas de TypeScript que con ellas llevamos a cabo la importación a sus aplicaciones.
Dicho proyecto se trabajó con MongoBD que bien, es una base de datos orientada a documentos, esto con la finalidad de que los registros creados de nuestro modulo se puedan ir guardando los datos que se utilizan para al momento de generar el código QR. El módulo también se complementó con Postman que nos permitió realizar pruebas de AP, esto para que a través de una interfaz gráfica de usuario podamos obtener diferentes tipos de respuestas que como objetivo son validadas. Gracias al desarrollo de este proyecto me permitió adquirir nuevos conocimientos que si bien me servirán en el mundo laboral, ya que es un proyecto bastante completo y con herramientas que como programadores son de suma importancia, y con un nivel de complejidad más alto, al ser un proyecto robusto y que se dividió entre muchos equipos de trabajo.