

# Tecnológico de Estudios Superiores de Ecatepec

Ing. Sistemas Computacionales

**Integración del módulo Visitantes temporales, configuración de backend y base de datos en Mongo DB**



Tecnológico de Estudios Superiores de Ecatepec

Ing. Sistemas Computacionales

**Presentado por:**

VICTOR RAMSES CAMACHO MARTINEZ

YARELI GABINO PAREDES

ALANJESUS RODRIGUEZ JARAMILLO

## INTRODUCCION

Un sistema de información es una de las herramientas principales en la administración y operación de una empresa ya que mediante estos es posible tener el control de las principales actividades de la empresa reduciendo así su vulnerabilidad respecto a la seguridad tanto de su información como del capital humano de la empresa.

En el presente documento se documenta el desarrollo del módulo "REGISTRO DE VISITANTES TEMPORALES" el cual compone una serie de módulos que en conjunto conforman el sistema de acceso para los empleados y visitantes de una empresa, de esta forma se tendrá el registro y ubicación de las personas que se encuentran dentro de la empresa y así en caso de ser necesario localizar a los visitantes de forma más ágil.

En el siguiente documento se ejemplifica el desarrollo y configuración del backend desarrollado en `spring` y el desarrollo de la base de datos del módulo en cuestión la cual se desarrollo con ayuda de las herramientas `Mongo DB` y `POSTMAN` , así mismo se integran las pruebas realizadas en su funcionamiento.

## Requerimientos y Necesidades

El módulo asignado requiere el desarrollo de las diferentes vistas y formularios que ayuden a generar un registro de los visitantes que acuden a una empresa sin una cita previa, para lo cual es necesario contar con las siguientes herramientas de desarrollo y requerimientos técnicos

### Requerimientos Técnicos

- ✓ **Procesador Intel Core i5 o superior**
- ✓ **8 gb memoria RAM**
- ✓ **HDD de 250 gb o superior**
- ✓ **Sistema operativo Windows 8 o superior**

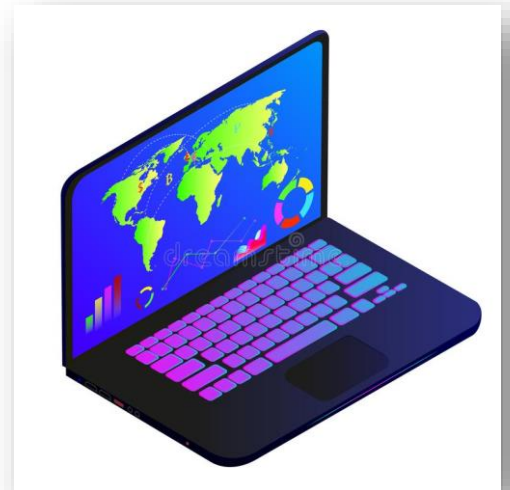
### Herramientas de desarrollo

#### WEBSTORM



WebStorm es un entorno de desarrollo integrado para JavaScript y las tecnologías relacionadas. Al igual que otros IDE de JetBrains, hace que su experiencia de desarrollo sea más agradable, automatiza las tareas repetitivas y le ayuda a gestionar las tareas complejas con facilidad.

Webstorm es un IDE profesional JavaScript que es compatible con una amplia gama de tecnologías modernas relacionadas con el lenguaje de programación JavaScript, HTML y CCS, además ofrece la experiencia completa para el desarrollo Web productivo.





## **BOOTSTRAP**

Bootstrap es un framework front-end utilizado para desarrollar aplicaciones web y sitios mobile first, o sea, con un layout que se adapta a la pantalla del dispositivo utilizado por el usuario.

tiene varios recursos para configurar los estilos de los elementos de la página de una manera simple y eficiente, además de facilitar la construcción de páginas que, al mismo tiempo, están adaptadas para la web y para dispositivos móviles.

## **ANGULAR**

es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

Angular se basa en clases tipo "Componentes", cuyas propiedades son las usadas para hacer el binding de los datos. En dichas clases tenemos propiedades (variables) y métodos (funciones a llamar).

Angular es la evolución de AngularJS, aunque incompatible con su predecesor.



## POSTMAN

Postman es una herramienta que se utiliza, sobre todo, para el testing de API REST, aunque también admite otras funcionalidades que se salen de lo que engloba el testing de este tipo de sistemas.

Gracias a esta herramienta, además de testear, consumir y depurar API REST, podremos monitorizarlas, escribir pruebas automatizadas para ellas, documentarlas, mockearlas, simularlas, etc.



## MONGO DB

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



## DESARROLLO

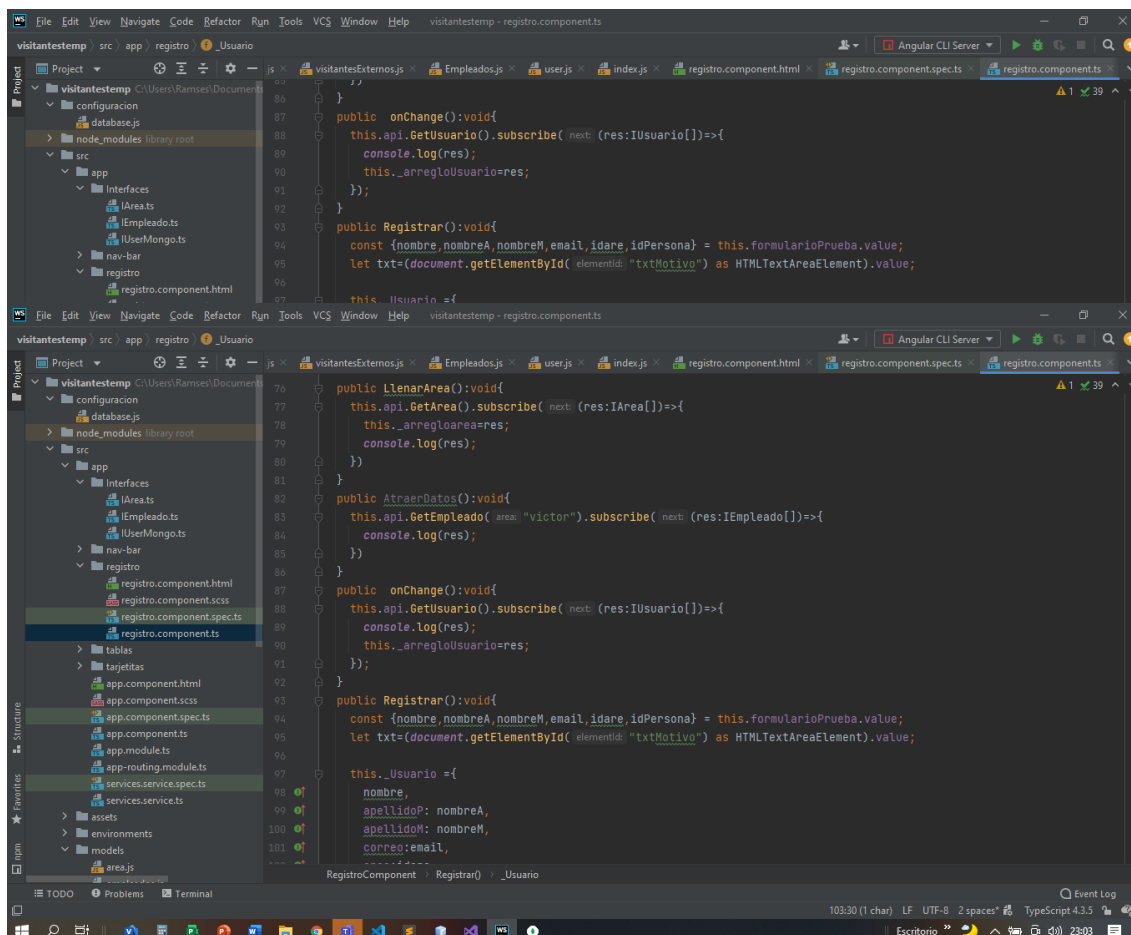
Un módulo es uno de los elementos principales con los que podemos organizar el código de las aplicaciones en Angular. No deben ser desconocidos hasta este momento del Manual de Angular, puesto que nuestra aplicación básica ya disponía de uno.

Sin embargo, en lugar de colocar el código de todos los componentes, directivas o pipes en el mismo módulo principal, lo adecuado es desarrollar diferentes módulos y agrupar distintos elementos en unos u otros. El orden se realizará de una manera lógica, atendiendo a nuestras propias preferencias, el modelo de negocio o las preferencias del equipo de desarrollo.

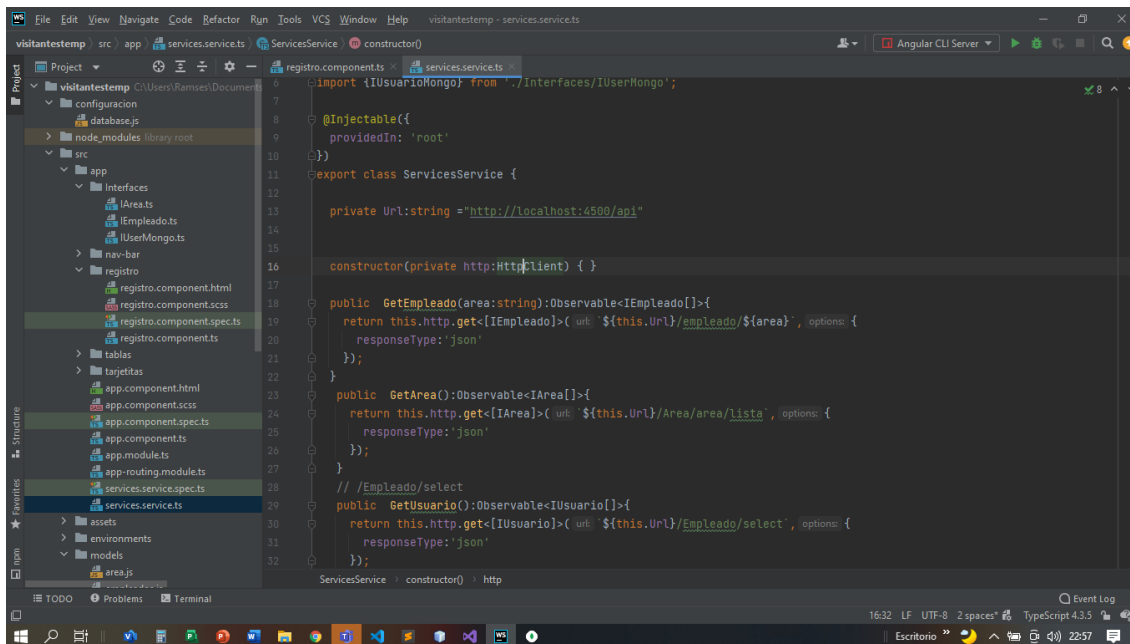
Durante el desarrollo del backend del modulo visitantes temporales determinamos la creación de los eventos requeridos para la vinculación con la base de datos de acuerdo al diagrama entidad relación previamente realizado.

Creación de los eventos que utilizaremos en este caso serán cuatro principales:

- ✓ IEmpleado: Que retorna los datos de nuestros trabajadores en la base.
- ✓ IArea: Que retorna los daos de nuestros trabajadores en la base
- ✓ IUsuario: Que retorna los datos completos en la base.
- ✓ IUsuarioMongo: Que guarda los datos de la página a la base.



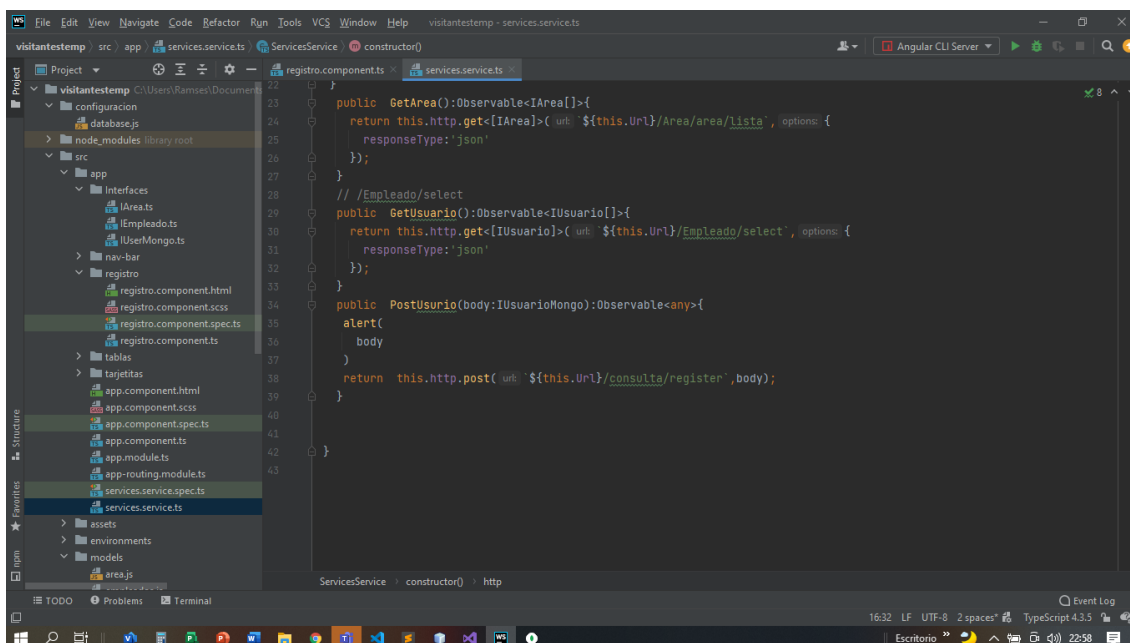
La función principal de los eventos creados es permitir una correcta interacción entre el backend, frontend y la base de datos ya que mediante estos se le permitirá al usuario ingresar datos, guardar los datos y consultarlos posteriormente.



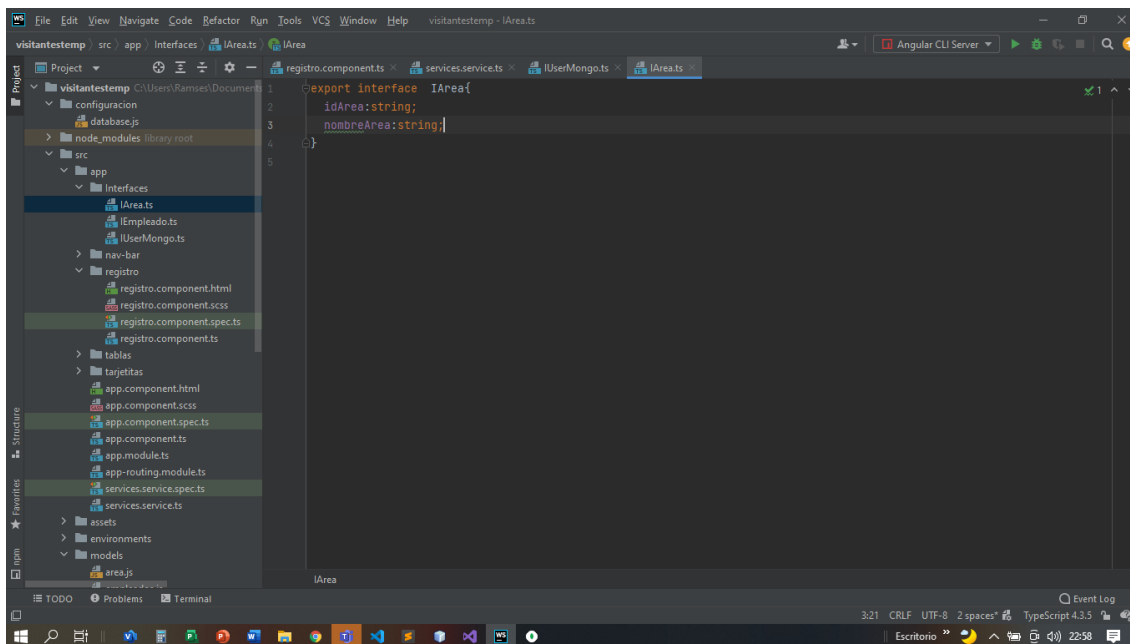
```
6 import { IUserMongo } from '../Interfaces/IUserMongo';
7
8 @Injectable({
9   providedIn: 'root'
10 })
11 export class ServicesService {
12
13   private Url:string = "http://localhost:4500/api"
14
15   constructor(private http:HttpClient) {}
16
17   public GetEmpleado(area:string):Observable<IEmpleado[]>{
18     return this.http.get<IEmpleado[]>({ url: `${this.Url}/Empleado/${area}`, options: {
19       responseType: 'json'
20     }});
21   }
22
23   public GetArea():Observable<IArea[]>{
24     return this.http.get<IArea[]>({ url: `${this.Url}/Area/area/lista`, options: {
25       responseType: 'json'
26     }});
27   }
28   // /Empleado/select
29   public GetUsuario():Observable<IUsuario[]>{
30     return this.http.get<IUsuario[]>({ url: `${this.Url}/Empleado/select`, options: {
31       responseType: 'json'
32     }});
33   }
34 }
```

Así mismo desarrollamos 3 interfaces con las cuales el usuario tendrá contacto, de tal forma serán las encargadas de exportar los datos de cada operación y corroborar su correcta distribución con la base de datos , corroborando que se han almacenado correctamente, en este caso serán:

- ✓ IArea
- ✓ IEmpleados
- ✓ IUserMongo.

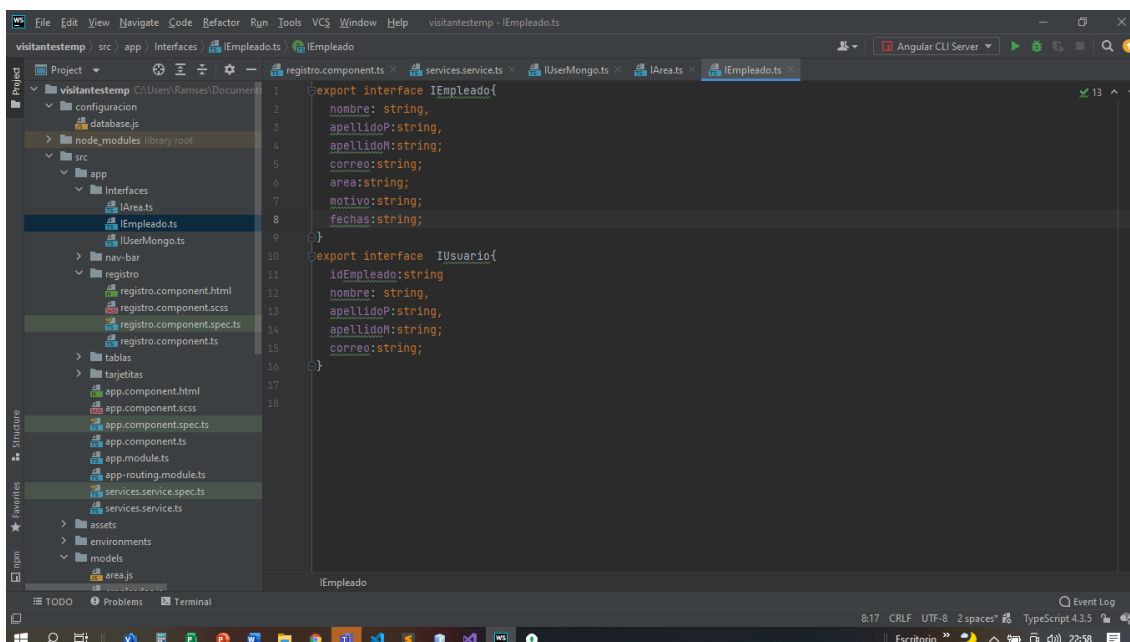


```
23 public GetArea():Observable<IArea[]>{
24   return this.http.get<IArea[]>({ url: `${this.Url}/Area/area/lista`, options: {
25     responseType: 'json'
26   }});
27 }
28 // /Empleado/select
29 public GetUsuario():Observable<IUsuario[]>{
30   return this.http.get<IUsuario[]>({ url: `${this.Url}/Empleado/select`, options: {
31     responseType: 'json'
32   }});
33 }
34 public PostUsuario(body:IUsuarioMongo):Observable<any>{
35   alert(
36     body
37   )
38   return this.http.post({ url: `${this.Url}/consulta/register`, body});
39 }
40
41 }
42 }
```



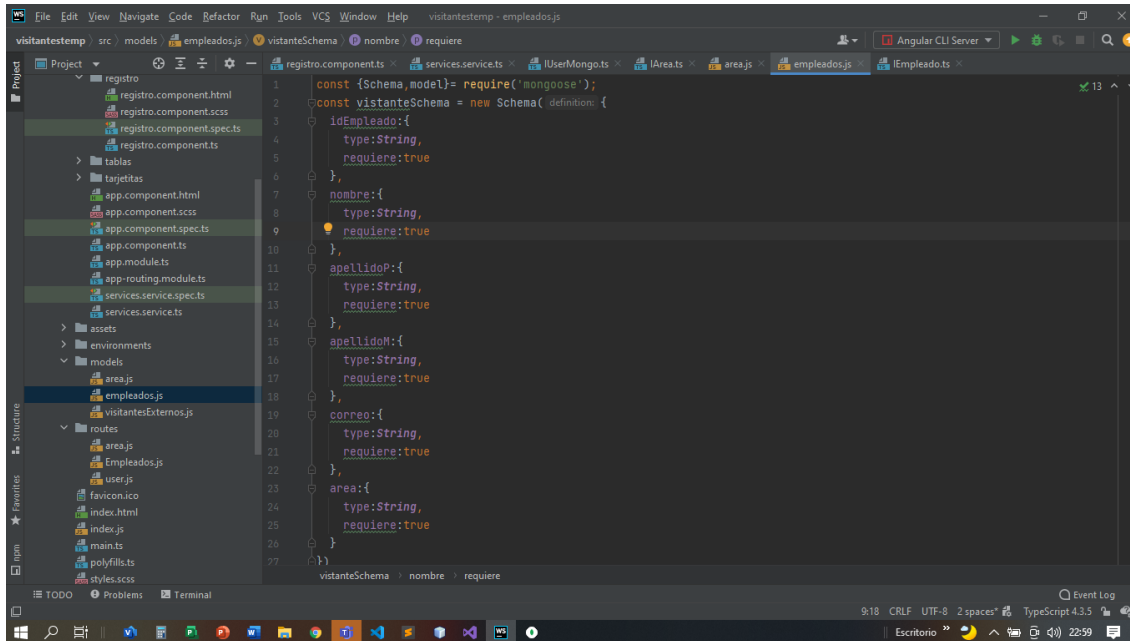
posteriormente creamos los models que serán los encargados de indicar los campos a llenar en la base de datos, entremos uno por cada proceso

Derivado a la interacción directa de este modelo con el usuario final es de suma importancia realizar una interfaz amigable y fácil de comprender por el usuario final , ya que de esto dependerá obtener la información correcta para cada campo.





Los modelos representan un objeto con propiedades que permite la utilización de la misma desde clases externas, evitando la reiteración de código, evitando así la redundancia de datos y una mejor administración de la información recopilada en la base de datos, de esta forma en un futuro las consultas en la base se realizarán de una forma más eficiente.

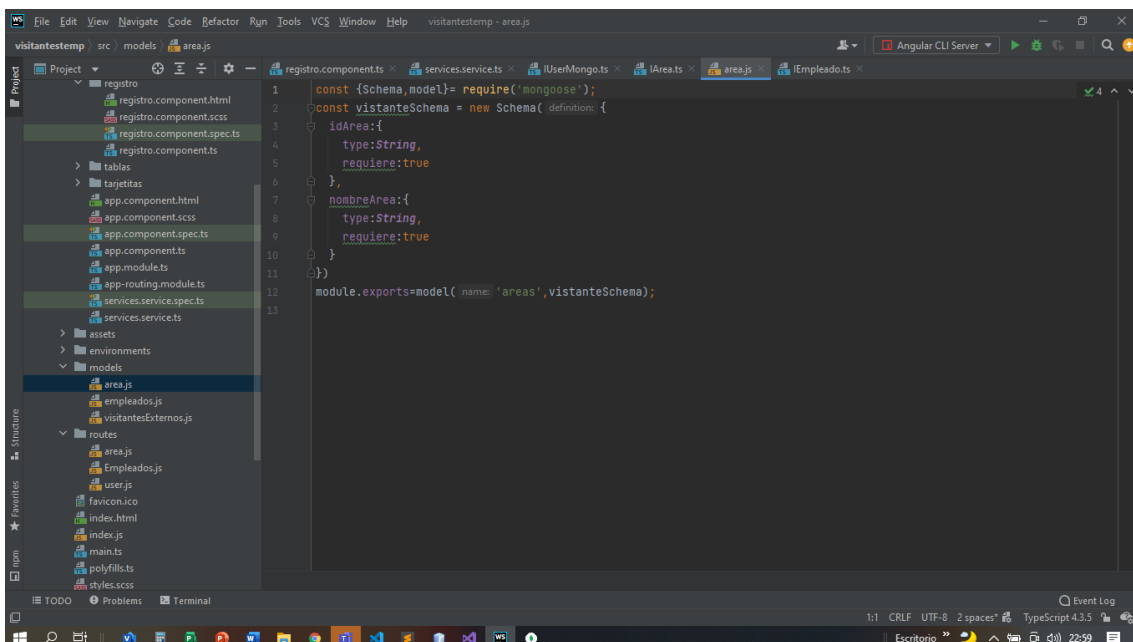


The screenshot shows the Visual Studio Code editor with the 'empleados.js' file open in the 'models' directory. The code defines a Mongoose schema for the 'empleados' model. The schema includes fields for 'idEmpleado', 'nombre', 'apellidoP', 'apellidoM', and 'correo', all of type 'String' and required. The 'idEmpleado' field is also marked as a 'required' field. The schema is named 'vistanteSchema' and is exported as 'empleados'.

```
const {Schema,model}= require('mongoose');
const vistanteSchema = new Schema( definitions: {
  idEmpleado:{
    type:String,
    require:true
  },
  nombre:{
    type:String,
    require:true
  },
  apellidoP:{
    type:String,
    require:true
  },
  apellidoM:{
    type:String,
    require:true
  },
  correo:{
    type:String,
    require:true
  },
  area:{
    type:String,
    require:true
  }
})
module.exports=model( name: 'empleados',vistanteSchema);
```

los modelos implementados en el backend de visitantes temporales son:

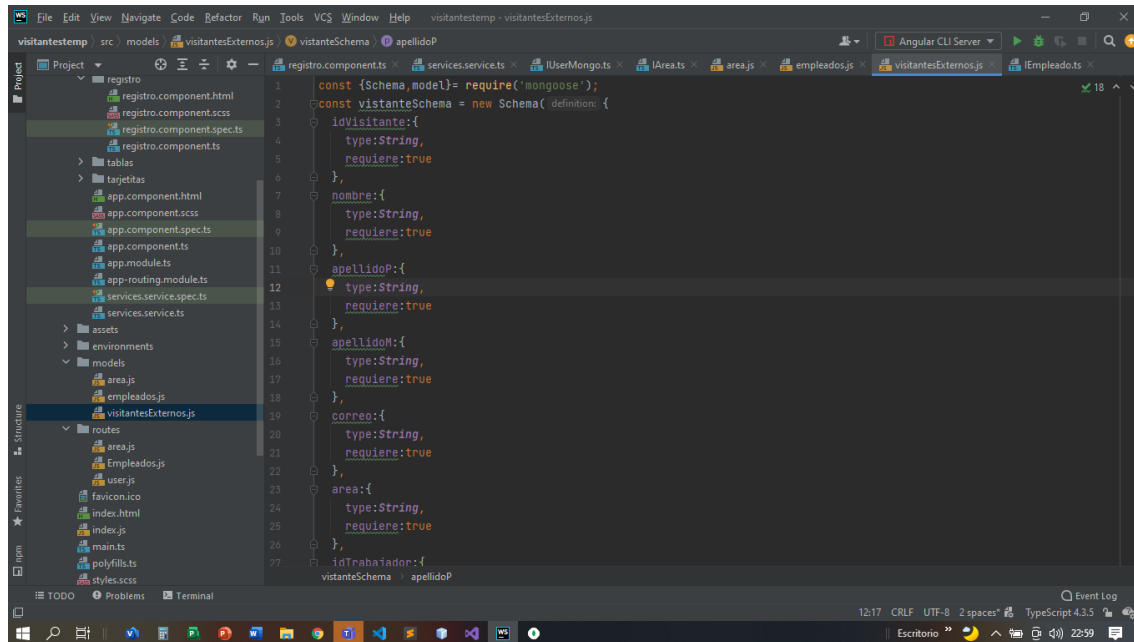
- ✓ área
- ✓ empleados
- ✓ visitantes externos



The screenshot shows the Visual Studio Code editor with the 'area.js' file open in the 'models' directory. The code defines a Mongoose schema for the 'area' model. The schema includes fields for 'idArea' and 'nombreArea', both of type 'String' and required. The 'idArea' field is also marked as a 'required' field. The schema is named 'vistanteSchema' and is exported as 'area'.

```
const {Schema,model}= require('mongoose');
const vistanteSchema = new Schema( definitions: {
  idArea:{
    type:String,
    require:true
  },
  nombreArea:{
    type:String,
    require:true
  }
})
module.exports=model( name: 'areas',vistanteSchema);
```

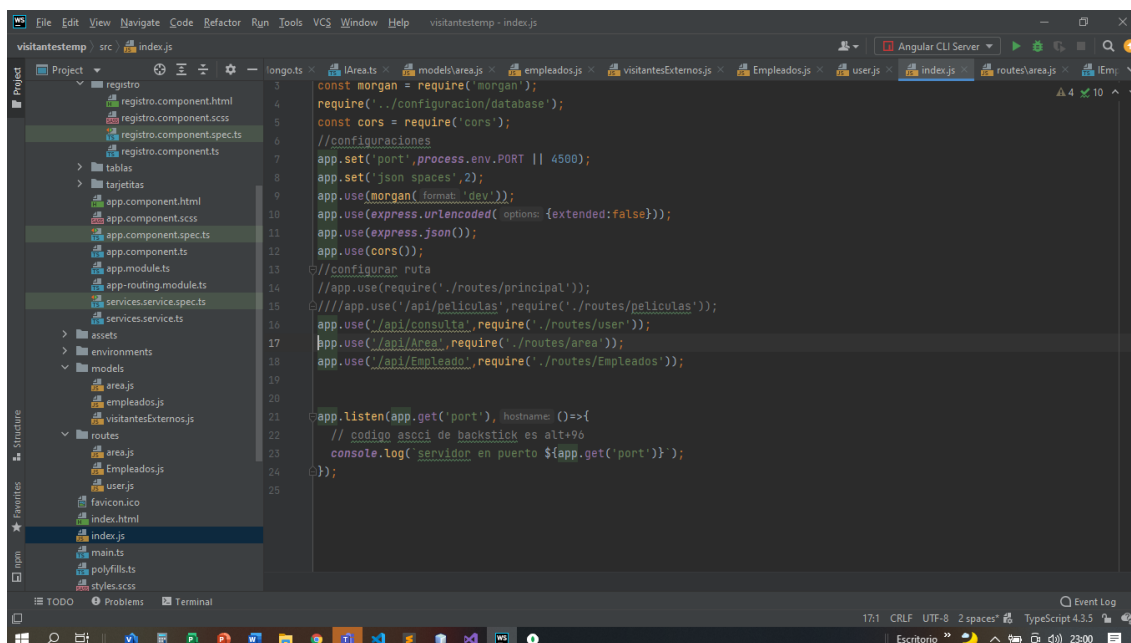
todos los campos que conforman cada uno de los modelos son campos obligatorios , en los cuales el usuario que generara el nuevo registro de un visitante temporal deben ingresar la información correcta y correspondiente de cada campo , de lo contrario el frontend enviara alertas indicando los campos requeridos para completar el registro.



```
1 const {Schema,model}= require('mongoose');
2 const visitanteSchema = new Schema( definition: {
3   idVisitante:{
4     type:String,
5     require:true
6   },
7   nombre:{
8     type:String,
9     require:true
10  },
11  apellidoP:{
12    type:String,
13    require:true
14  },
15  apellidoM:{
16    type:String,
17    require:true
18  },
19  correo:{
20    type:String,
21    require:true
22  },
23  area:{
24    type:String,
25    require:true
26  },
27  idTrabajador:{
28    type:String,
29    require:true
30  }
31 },
32 {
33   timestamps: true
34 }
35 );
36 module.exports = mongoose.model('visitante', visitanteSchema);
```

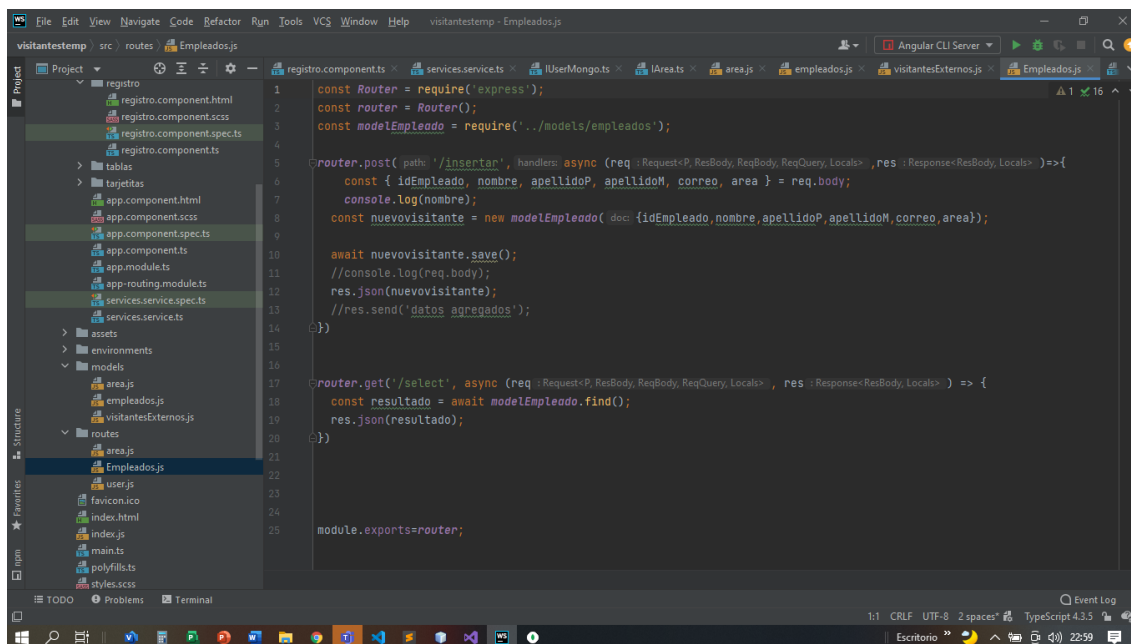
Posteriormente crearemos nuestras rutas para poder ser generadas en la pagina web y funciones con el llenado de datos para que puedan ser enviados a la base de datos.

Las Rutas de navegación sirven, en primera instancia, para conocer cómo interactúan cada usuario con una página web y todos sus apartados. Ayudan a ver cuáles son las partes más frecuentadas, cuáles las que menos, conocer el tiempo de duración de las visitas, qué contenidos son los más atractivos, qué funciones se aprovechan y cuáles no.



```
1 const express = require('express');
2 const mongoose = require('mongoose');
3 constmorgan = require('morgan');
4 const bodyParser = require('body-parser');
5 const cors = require('cors');
6 //configuraciones
7 app.set('port',process.env.PORT || 4500);
8 app.set('json spaces',2);
9 app.use(morgan('format:dev'));
10 app.use(express.urlencoded({ extended:false}));
11 app.use(express.json());
12 app.use(cors());
13 //configurar ruta
14 app.use(require('./routes/principal'));
15 app.use('/api/peliculas',require('./routes/peliculas'));
16 app.use('/api/consulta',require('./routes/user'));
17 app.use('/api/Area',require('./routes/area'));
18 app.use('/api/Empleado',require('./routes/Empleados'));
19
20 //configurar servidor
21 app.listen(app.get('port'), hostname() =>{
22   // codigo ascii de backstick es alt+96
23   console.log('servidor en puerto ${app.get('port')}');
24 });
```

En definitiva, sirven para que las compañías sepan cómo se navega a través de sus webs y puedan actuar en consecuencia, mejorando el diseño o los contenidos en caso de que el tiempo sea reducido y reordenador si se desea conseguir una mayor afluencia de visitas en determinados apartados.



```
const Router = require('express');
const router = Router();
const modelEmpleado = require('../models/empleados');

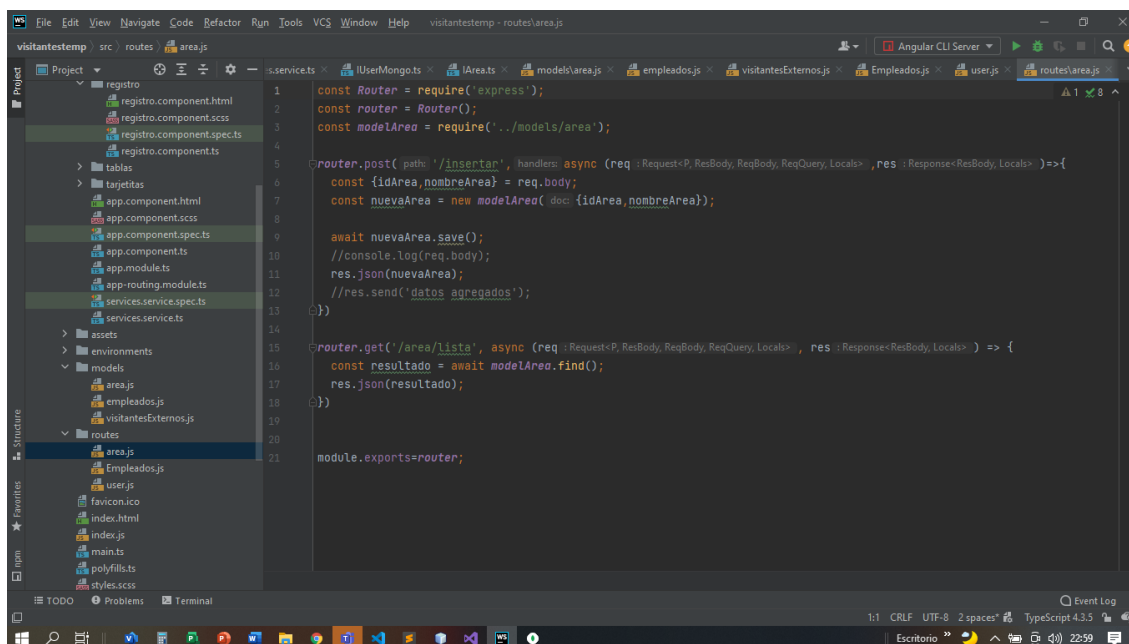
router.post('/insertar', handlers: async (req: Request<P, ResBody, ReqBody, ReqQuery, Locals>, res: Response<ResBody, Locals>) => {
  const { idEmpleado, nombre, apellidoP, apellidoM, correo, area } = req.body;
  console.log(nombre);
  const nuevovisitante = new modelEmpleado({ doc: { idEmpleado, nombre, apellidoP, apellidoM, correo, area }});

  await nuevovisitante.save();
  //console.log(req.body);
  res.json(nuevovisitante);
  //res.send('datos agregados');
});

router.get('/select', async (req: Request<P, ResBody, ReqBody, ReqQuery, Locals>, res: Response<ResBody, Locals>) => {
  const resultado = await modelEmpleado.find();
  res.json(resultado);
});

module.exports=router;
```

Después tendremos que dar las acciones de cada ruta que son los campos que va a tomar y mandar a la base de datos, estas acciones determinan hacia que portal se dirigira el usuario al hacer uso del evento click , de esta forma podrá interactuar con todos los portales de la pagina web.



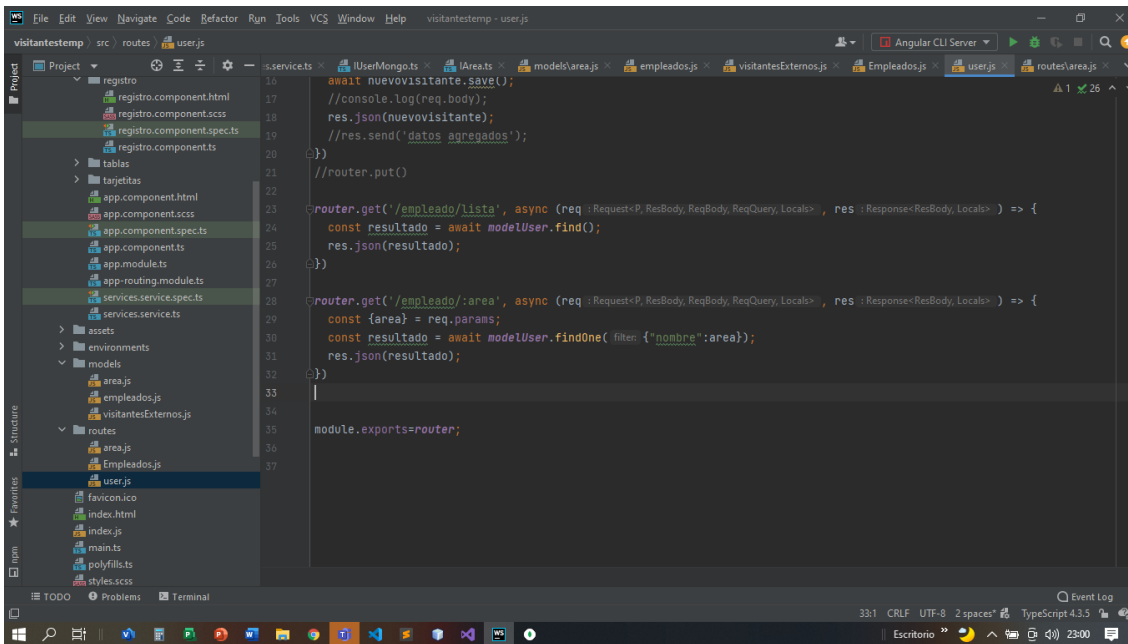
```
const Router = require('express');
const router = Router();
const modelArea = require('../models/area');

router.post('/insertar', handlers: async (req: Request<P, ResBody, ReqBody, ReqQuery, Locals>, res: Response<ResBody, Locals>) => {
  const { idArea, nombreArea } = req.body;
  const nuevaArea = new modelArea({ doc: { idArea, nombreArea }});

  await nuevaArea.save();
  //console.log(req.body);
  res.json(nuevaArea);
  //res.send('datos agregados');
});

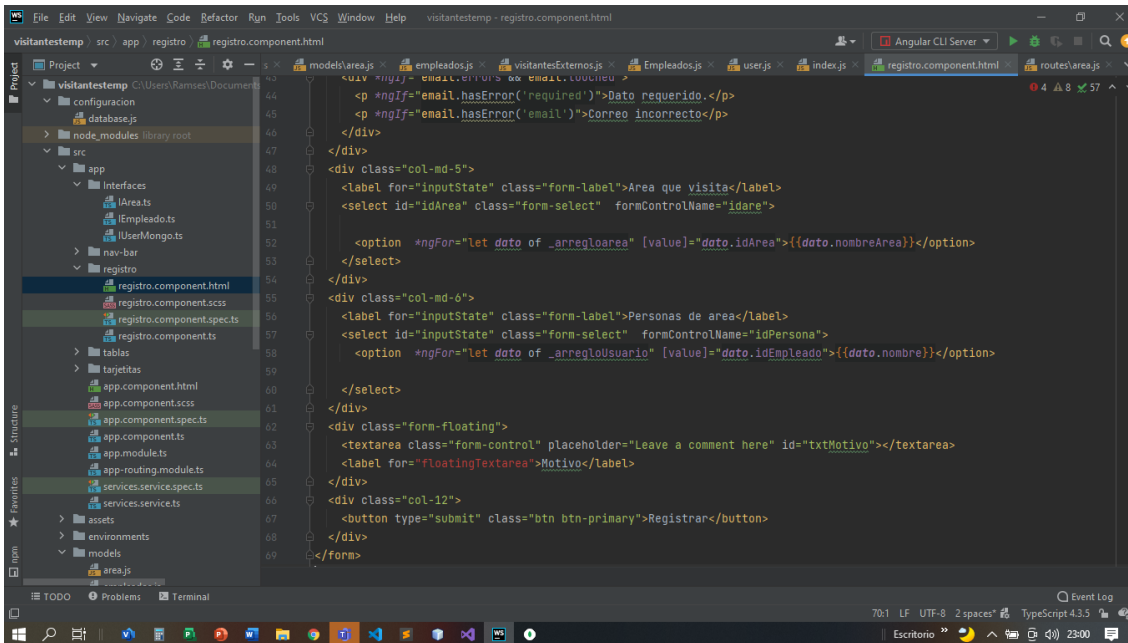
router.get('/area/lista', async (req: Request<P, ResBody, ReqBody, ReqQuery, Locals>, res: Response<ResBody, Locals>) => {
  const resultado = await modelArea.find();
  res.json(resultado);
});

module.exports=router;
```



```
16 await nuevovisitante.save();
17 //console.log(req.body);
18 res.json(nuevovisitante);
19 //res.send('datos agregados');
20 })
21 //router.put()
22
23 router.get('/empleado/lista', async (req : Request<P, ResBody, ReqQuery, Local>, res : Response<ResBody, Local>) => {
24   const resultado = await modelUser.find();
25   res.json(resultado);
26 })
27
28 router.get('/empleado:area', async (req : Request<P, ResBody, ReqQuery, Local>, res : Response<ResBody, Local>) => {
29   const {area} = req.params;
30   const resultado = await modelUser.findOne( {filter: {"nombre":area}});
31   res.json(resultado);
32 })
33
34
35 module.exports=router;
```

Por último, solo tendremos que extraer la información de nuestras áreas y empleados para que se puedan reflejar en las listas despegables.



```
44 <div *ngIf="email.hasError('required')">Data requerido.</div>
45 <div *ngIf="email.hasError('email')">Correo incorrecto.</div>
46 </div>
47 </div>
48 <div class="col-md-5">
49   <label for="inputState" class="form-label">Area que visita</label>
50   <select id="idArea" class="form-select" formControlName="idArea">
51     <option *ngFor="let dato of _arregloarea" [value]="dato.idArea">{{dato.nombreArea}}</option>
52   </select>
53 </div>
54 <div class="col-md-6">
55   <label for="inputState" class="form-label">Personas de area</label>
56   <select id="inputState" class="form-select" formControlName="idPersona">
57     <option *ngFor="let dato of _arreglousuario" [value]="dato.idEmpleado">{{dato.nombre}}</option>
58   </select>
59 </div>
60 <div class="form-floating">
61   <textarea class="form-control" placeholder="Leave a comment here" id="txtMotivo"></textarea>
62   <label for="floatingTextarea">Motivo</label>
63 </div>
64 <div class="col-12">
65   <button type="submit" class="btn btn-primary">Registrar</button>
66 </div>
67 </div>
68 </form>
```

## Pruebas del funcionamiento de la página web con nuestra conexión a mongoDB

A continuación, se muestran las pruebas realizadas con la conexión del frontend, backend y la base de datos, de acuerdo a estas pruebas se corrobora el correcto funcionamiento del enrutado entre los potales asi como la captura y almacenamiento de la información correspondiente al registro de los visitantes temporales.

(15) WhatsApp

porque puerto entra imap - Busc

Postman

Visitantes temporales

+

localhost:4200/enlaceregistro

navbar Registro

Search

Nombre

Apellido Paterno

Apellido Materno

ricardo

escamilla

villasana

Dirección de correo electrónico

victorcamacho1171@gmail.com

Inserte su correo

Area que visita

Recursos humanos

Personas de area

wero

Motivo

hjkjhkhjk

Registrar

MongoDB Compass - cluster:visitas.9vj2v.mongodb.net/visitas.usuarios

Connect View Collection Help

Local

3 DBS 11 COLLECTIONS

☆ FAVORITE

HOSTS

cluster:visitas-shard-00-01...

cluster:visitas-shard-00-02...

cluster:visitas-shard-00-00...

CLUSTER

Replica Set (atlas-52x9r2-...

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

admin

local

visitas

areas

empleados

usuarios

> MONGOSH

visitas.usuarios

Documents

DOCUMENTS 12

STORAGE SIZE 20.5KB

AVG. SIZE 188B

INDEXES 1

TOTAL SIZE 36.9KB

AVG. SIZE 36.9KB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER [ field: 'value' ]

ADD DATA VIEW

Displaying documents 1 - 12 of 12

Refresh

\_id: ObjectId("61ea3785fa521e905a199b94")

nombre: "Victor Ramses"

apellido: "Camacho"

apellido: "Martinez"

correo: "victorcamacho1171@gmail.com"

area: "3"

idTrabajador: "2"

motivo: "hjkjhkhjk"

fechas: 2022-01-21T04:31:22.245+00:00

\_\_v: 0

\_id: ObjectId("61ea403d790fc47709ca3600")

nombre: "Ricardo"

apellido: "Escamilla"

apellido: "Villasana"

correo: "victorcamacho1171@gmail.com"

area: "3"

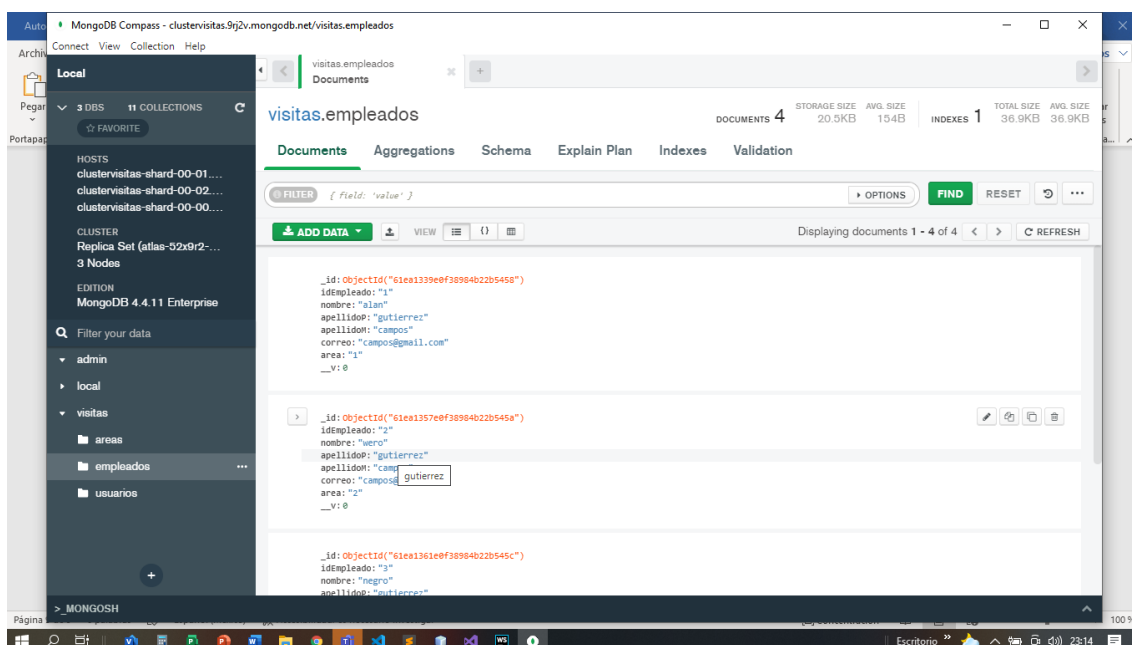
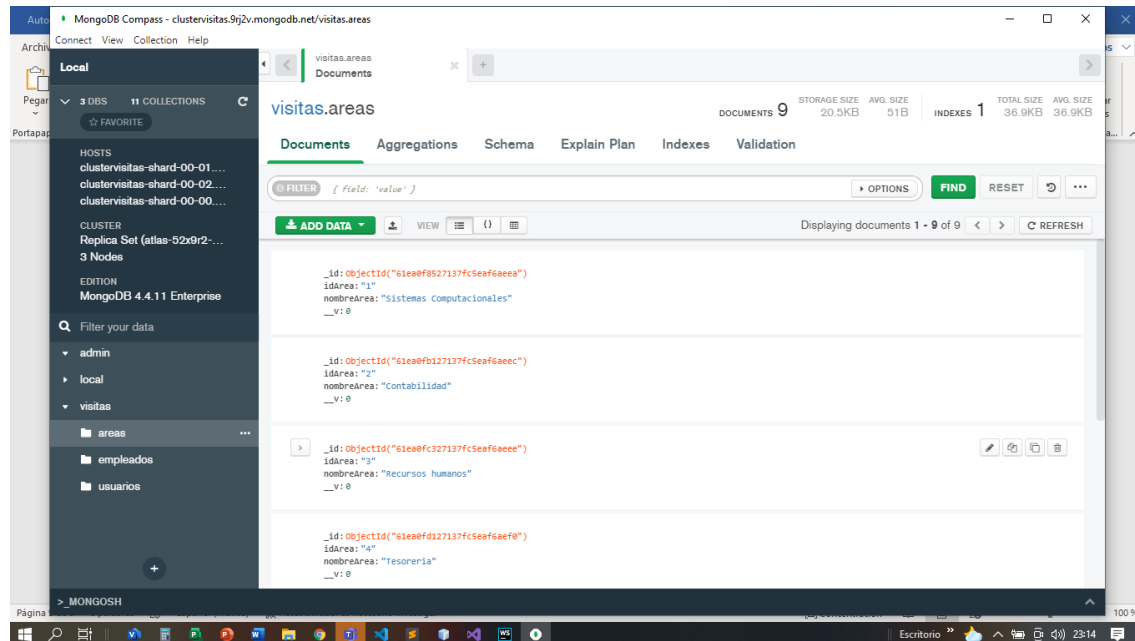
idTrabajador: "2"

motivo: "hjkjhkhjk"

fechas: 2022-01-21T04:34:23.569+00:00

\_\_v: 0

Al realizar inserción de datos en el frontend podemos consultar mediante Mongo DB y postman que los datos se han guardado en la base de datos. Asi podremos realizar consultas de datos posteriormente.



## **CONCLUSION**

Durante los últimos años los sistemas de información constituyen uno de los principales ámbitos de estudio en el área de organización de empresas. El entorno donde las compañías desarrollan sus actividades se vuelve cada vez más complejo. La creciente globalización, el proceso de internacionalización de la empresa, el incremento de la competencia en los mercados de bienes y servicios, la rapidez en el desarrollo de las tecnologías de información, el aumento de la incertidumbre en el entorno y la reducción de los ciclos de vida de los productos originan que la información se convierta en un elemento clave para la gestión, así como para la supervivencia y crecimiento de la organización empresarial. Si los recursos básicos analizados hasta ahora eran tierra, trabajo y capital, ahora la información aparece como otro insumo fundamental a valorar en las empresas.