



---

---

**TECNOLÓGICO DE ESTUDIOS SUPERIORES DE  
ECATEPEC**

**DIVISIÓN DE INGENIERÍA EN SISTEMAS  
COMPUTACIONALES**

**BASE DE DATOS PARA DISPOSITIVOS MÓVILES**

**AUTENTIFICACIÓN, REGISTRO Y BLOQUEO DE  
USUARIOS**

**GRUPO: 5851**

**EQUIPO ##**

**PRESENTA:**

López Alcántara Alan Jesús

**PROFESOR:**

Cortes Barrera Griselda

**Febrero 09, 2022**

# PROPUESTA INICIAL DEL MÓDULO

## AUTENTIFICACIÓN, REGISTRO Y BLOQUEO DE USUARIOS

### REQUERIMIENTOS Y NECESIDADES DEL MÓDULO A TRABAJAR

El presente módulo a trabajar surge de la necesidad de contemplar el acceso restringido a determinado tipo de usuarios que cuenten con los permisos y privilegios para la visualización y uso del espacio de trabajo de sitios específicos de la aplicación en curso. Así mismo como el **bloqueo** de usuarios que resulten denominados por la empresa por y estos no puedan tener ningún tipo de acceso a sitios de la aplicación y por tanto resguardar la información de la empresa.

Es posible enlistar las necesidades como:

- I. Bloqueo de usuarios en el sistema.
- II. Notificación de usuario no registrado (logueado).
- III. Restringir el acceso a usuarios que no cuenten con permisos establecidos para determinados sitios.

Los requerimientos del módulo engloban principalmente:

- ♣ Una base de datos que describa los usuarios bloqueados con su identificador (correo).
- ♣ Un módulo de Login en donde implementar la funcionalidad del bloqueo en el proceso de acceso de la cuenta.
- \* Un servicio que permita bloquear el acceso de los usuarios a las rutas de redirección de los sitios.

### PROBLEMÁTICA

La empresa que solicita el diseño de la aplicación nos describe la necesidad de mantener un acceso protegido a la información de la página por medio del acceso de las cuentas de usuarios. Es decir, es necesario mantener un registro de los usuarios que pueden tener acceso a determinados sitios de la aplicación y usuarios que definitivamente no puedan acceder al sistema por medio de un bloqueo.

Tomando en cuenta que la aplicación trata con diferentes tipos de usuario, se identifica que cada uno de ellos tiene acceso a determinado tipo de información y de espacio de trabajo, por lo tanto la funcionalidad del módulo permitirá que los usuarios únicamente tengan posibilidad de acceder a sus espacios correspondientes de trabajo. Así mismo como identificar usuarios que se encuentren bloqueados y no cuenten con acceso alguno al sistema.

### OBJETIVO GENERAL

Crear un módulo en el cual se administran usuarios ya sea de un portal escolar, empresa etc. Etiquetando a personal con ciertos permisos y restricciones dependiendo de su posición que se le otorgue.

## OBJETIVOS ESPECIFICOS

- Identificar que funciones son las que ocuparemos para la implementación del modulo
- Diseñar un módulo de pruebas de dichas funciones
- Codificar para el proyecto final

## DISEÑO DE LA PROPUESTA

Nuestra propuesta principal consiste en la implementación de una función en Angular denominada “Guard”, este tipo de middlewares funcionan como un bloqueador de rutas, es decir que cuando queremos acceder a algún sitio que nos redirija a otro sitio el Guard podrá bloquear esta ruta y evitar que el usuario tenga acceso. En sencillos pasos Guard funciona como un resultado booleano, responde true o false, si la respuesta es true se da acceso, de lo contrario se notifica al usuario que no puede acceder a ese sitio en general.

### ¿Qué es Guard?

Los Guards en Angular, son de alguna manera: middlewares que se ejecutan antes de cargar una ruta y determinan si se puede cargar dicha ruta o no. Existen 4 tipos diferentes de Guards (o combinaciones de estos) que son los siguientes:

1. **(CanActivate)** Antes de cargar los componentes de la ruta.
2. **(CanLoad)** Antes de cargar los recursos (assets) de la ruta.
3. **(CanDeactivate)** Antes de intentar salir de la ruta actual (usualmente utilizado para evitar salir de una ruta, si no se han guardado los datos).
4. **(CanActivateChild)** Antes de cargar las rutas hijas de la ruta actual.

### ¿Por qué utilizar Guards?

Una de las ventajas de utilizarlos, es que al no cargar la ruta evitamos que los usuarios vean una interfaz a la que no tienen acceso (aunque sea por unos pocos milisegundos). Por otra parte, con estos componentes es posible estructurar el código de la aplicación de una manera más organizada y donde la lógica de la ruta está en la ruta en sí y la lógica de permisos y acceso a recursos se encuentra aislada en estos componentes.

### ¿Cómo utiliza Angular los Guards?

Empezaremos con la creación de un Guard. Para crear un Guard es posible hacerlo de manera manual, pero angular-cli provee un generador muy bueno y que de manera automática genera el Guard junto a las pruebas del mismo

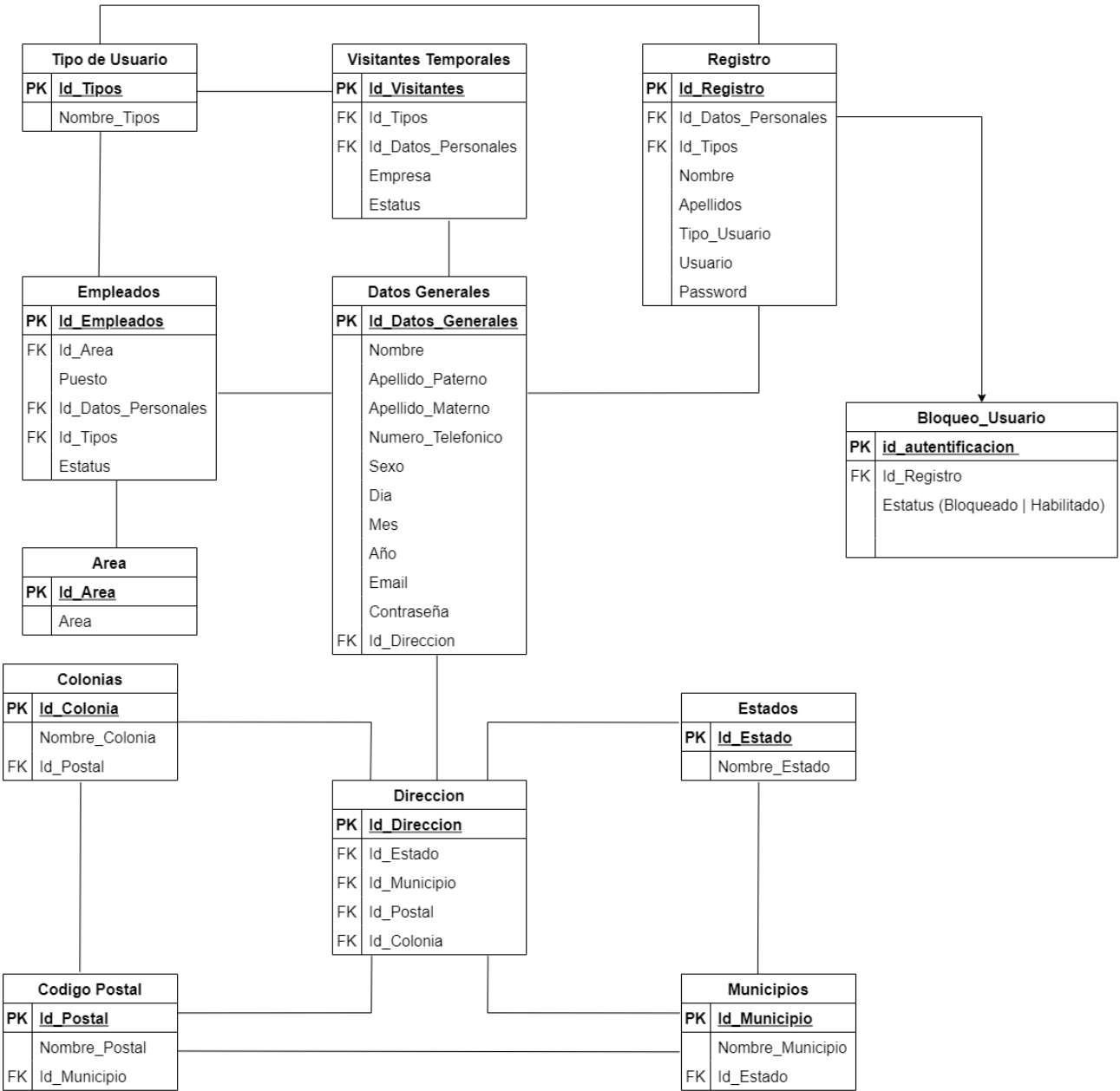


# DISEÑO MODELO E-R DEL PROYECTO Y ADECUACIÓN CON RESPECTO AL MÓDULO

## DIAGRAMA E-R DE LA BASE DE DATOS DEL PROYECTO

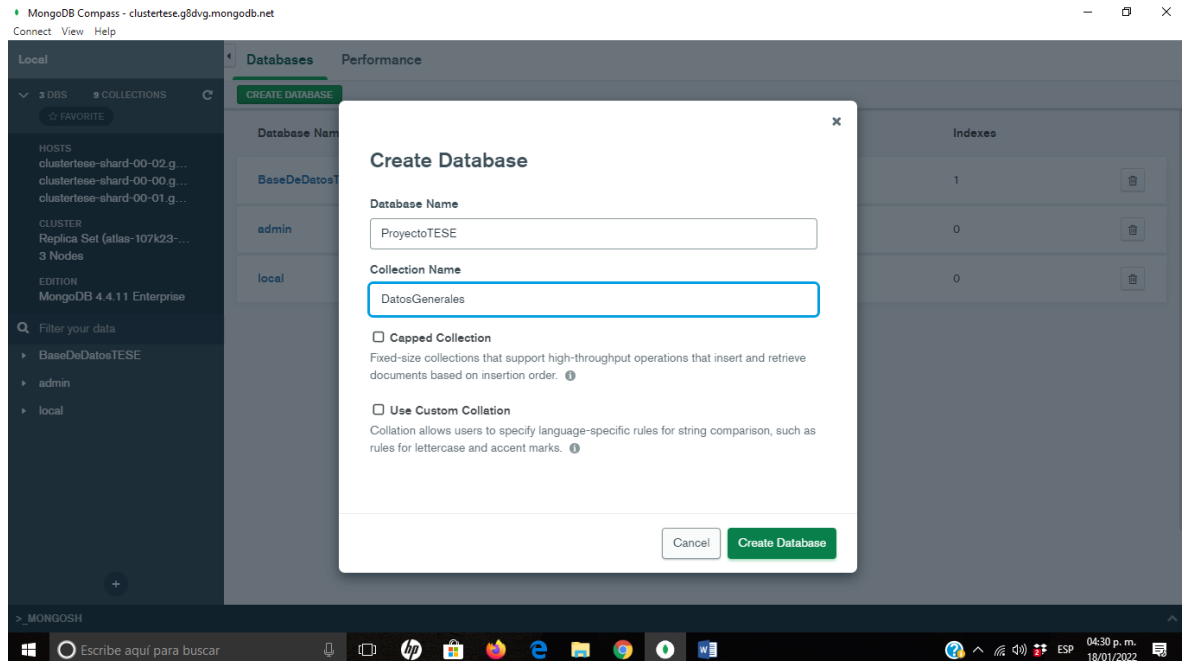
### Módulo autenticación, registro y bloqueo de usuarios

Para la construcción del módulo consideramos necesario la implementación de una tabla adicional Bloqueo\_Usuario la cual pueda contener el id\_Registro de cada uno de los usuarios, con este elemento es posible confirmar mediante la vista realizada conocer los datos generales y el tipo de usuario. Se solicitan datos iniciales como el nombre y los apellidos para contar con información básica de referencia. Esta tabla contiene el PK que refiere al id\_autenticación el cual asigna el token de registro, la FK del id\_registro y un dato adicional que describe si el usuario se encuentra en un estatus de (Bloqueo o Habilitado) que en la aplicación cumple la función de True o False en cuanto a su autenticación y acceso.

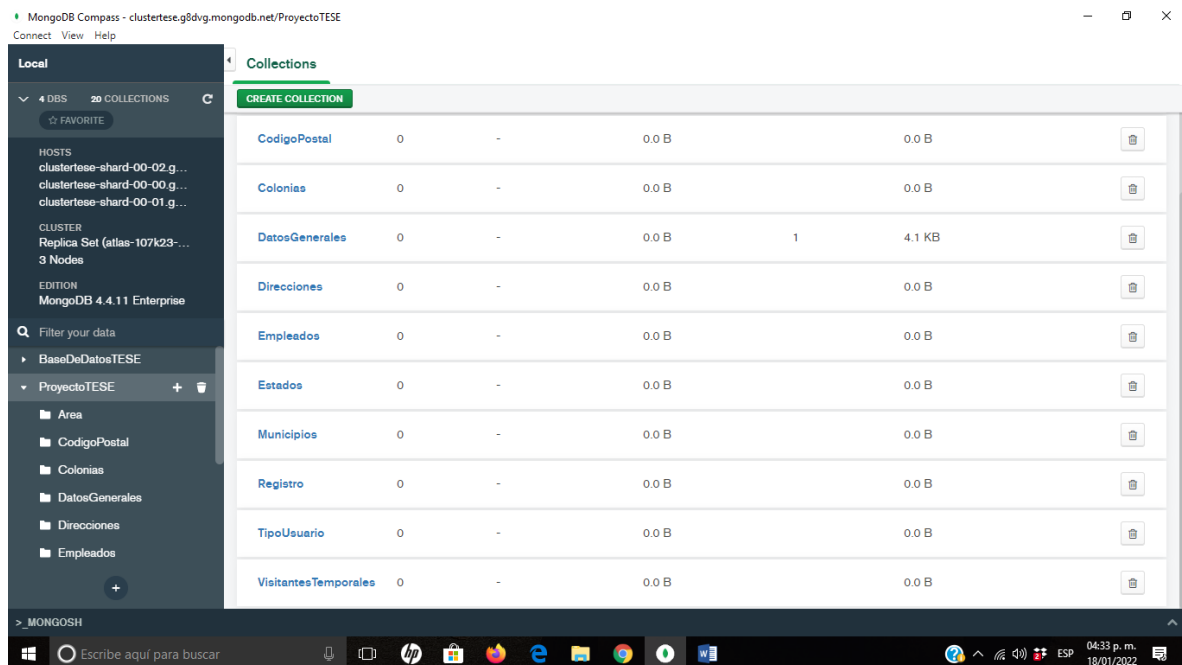


# DISEÑO E IMPLEMENTACIÓN DE LA BASE DE DATOS DEL PROYECTO

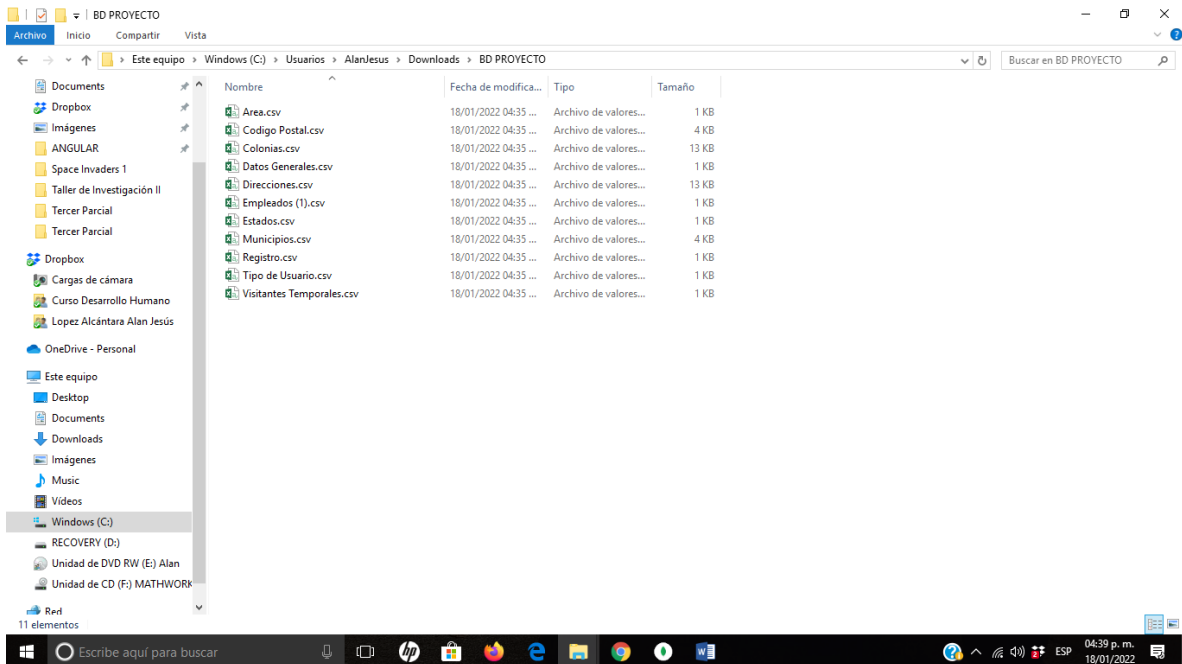
## CREACIÓN DE BASE DE DATOS DEL PROYECTO



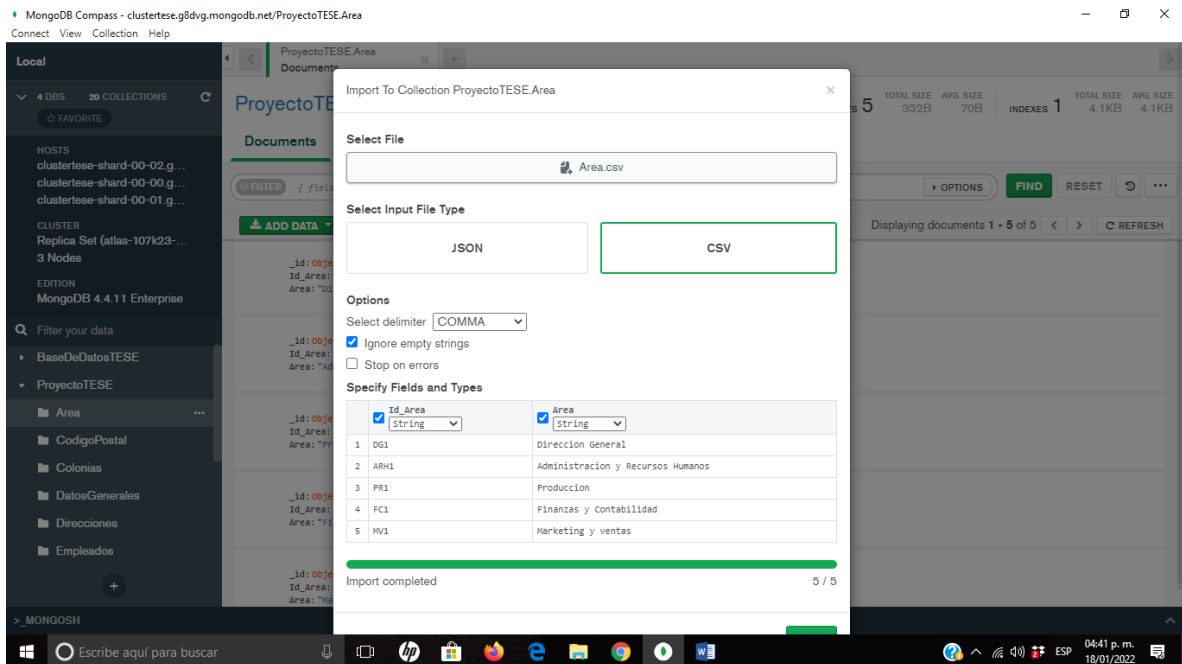
## CREACIÓN DE COLECCIONES



## Realizamos la importación de cada una de las colecciones desde Excel



## COLECCIÓN ÁREA



## COLECCIÓN CÓDIGO POSTAL

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESE.CodigoPostal

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

BaseDatosTESE

ProyectoTESE

- Area
- CodigoPostal
- Colonias
- DatosGenerales
- Direcciones
- Empleados

ADD DATA

Documents

Codigo Postal.csv

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_Postal String	<input checked="" type="checkbox"/> Nombre_Postal String	<input checked="" type="checkbox"/> Id_Municipio String
1	43824	28367	1001
2	52832	28040	1001
3	45700	28927	1005
4	51944	22476	2004
5	51811	22120	2004
6	40675	23080	3003
7	41465	23085	3003
8	49456	24085	4002
9	55621	25902	527
10	41535	25093	530

Import completed 212 / 212

DONE

Displaying documents 1 - 20 of 212

## COLECCIÓN COLONIAS

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESE.Colonias

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

BaseDatosTESE

ProyectoTESE

- Area
- CodigoPostal
- Colonias
- DatosGenerales
- Direcciones
- Empleados

ADD DATA

Documents

Colonias.csv

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_Colonia String	<input checked="" type="checkbox"/> Nombre_Colonia String	<input checked="" type="checkbox"/> Id_Postal String
1	76637	2 de Octubre	43824
2	83141	Che Guevara	43824
3	91973	Altavista	52832
4	91367	Curtidores	52832
5	89508	Ampliación La Cañada	45700
6	82987	La Cañada	45700
7	89997	Ojos de Agua	45700
8	81054	10 de Mayo	51944
9	92864	Granjas Familiares Unidas	51944
10	92501	18 de Marzo	51811

Import completed 438 / 438

DONE

Displaying documents 1 - 20 of 438

## COLECCIÓN DATOS GENERALES

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESEDatosGenerales

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

BaseDeDatosTESE

ProyectoTESE

- Area
- CodigoPostal
- Colonias
- DatosGenerales
- Direcciones
- Empleados

ADD DATA

Import To Collection ProyectoTESE.DatosGenerales

Select File

Datos Generales.csv

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_Datos_Generales	<input checked="" type="checkbox"/> Nombre	<input checked="" type="checkbox"/> Apellido_Paterno	<input checked="" type="checkbox"/> Apellido_Materno	<input checked="" type="checkbox"/> Nur
1	8456	Cesar Librado	Rangel	Reyes	553554
2	7681	Pedro Antonio	Ponce	Flores	553481
3	5251	Luis Enrique	Zenteno	Calderon	550997
4	5739	Alan Fernando	Yañez	Sandoval	551234
5	8445	Maria Fernanda	Rangel	Haro	235841
6	5400	Nara	Yañez	Quintero	989761

Import completed 6 / 6

## COLECCIÓN DIRECCIONES

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESEDirecciones

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

BaseDeDatosTESE

ProyectoTESE

- Area
- CodigoPostal
- Colonias
- DatosGenerales
- Direcciones
- Empleados

ADD DATA

Import To Collection ProyectoTESE.Direcciones

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_Direccion	<input checked="" type="checkbox"/> Id_estado	<input checked="" type="checkbox"/> Id_Municipio	<input checked="" type="checkbox"/> Id_Postal	<input checked="" type="checkbox"/> Id_colonia
1	111580	1	1001	43824	76637
2	111694	1	1001	43824	83141
3	111792	1	1001	52832	91973
4	111607	1	1001	52832	91367
5	111863	1	1005	45700	89508
6	111980	1	1005	45700	82987
7	111690	1	1005	45700	89997
8	111496	2	2004	51944	81054
9	111925	2	2004	51944	92864
10	111150	2	2004	51811	92501

Import completed 438 / 438



## COLECCIÓN EMPLEADOS

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESEmpleados

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

BaseDeDatosTESE

ProyectoTESE

- Area
- CodigoPostal
- Colonias
- DatosGenerales
- Direcciones
- Empleados

Import To Collection ProyectoTESE.Empleados

Select File

Empleados (1).csv

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_Empleados String	<input checked="" type="checkbox"/> Id_Area String	<input checked="" type="checkbox"/> Puesto String	<input checked="" type="checkbox"/> Id_datos_Generales String
1	E14480	ARH1	Gerente de recursos humanos	8456
2	E12745	PR1	Gerente de distribucion	7681
3	E14986	MV1	Gerente de ventas	5251

Import completed 3 / 3

DONE

## COLECCIÓN ESTADOS

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESEstados

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

CodigoPostal

Colonias

DatosGenerales

Direcciones

Empleados

Estados

Municipios

Registro

Import To Collection ProyectoTESE.Estados

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_Estado String	<input checked="" type="checkbox"/> Nombre_Estado String
1	1	Aguascalientes
2	2	Baja California
3	3	Baja California
4	4	Campeche
5	5	Coahuila de Zaragoza
6	6	Colima
7	7	Chiapas
8	8	Chihuahua
9	9	Distrito Federal
10	10	Durango

Import completed 33 / 33

DONE

## COLECCIÓN MUNICIPIOS

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESE.Municipios

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

- CodigoPostal
- Colonias
- DatosGenerales
- Direcciones
- Empleados
- Estados
- Municipios
- Registro

Documents

ProjectoTESE

Import To Collection ProyectoTESE.Municipios

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_Municipio String	<input checked="" type="checkbox"/> Nombre_Municipio String	<input checked="" type="checkbox"/> Id_Estado String
1	1001	Aguascalientes	1
2	1005	Jesús María	1
3	2004	Tijuana	2
4	3003	La Paz	3
5	4002	Campeche	4
6	527	Ramos Arizpe	5
7	530	Saltillo	5
8	6002	Colima	6
9	610	Villa de Álvarez	6
10	7101	Tuxtla Gutiérrez	7

Import completed 162 / 162

DONE

04:47 p.m. 18/01/2022

## COLECCIÓN REGISTRO

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESE.Registro

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

- CodigoPostal
- Colonias
- DatosGenerales
- Direcciones
- Empleados
- Estados
- Municipios
- Registro

Documents

ProjectoTESE.Registro

Import To Collection ProyectoTESE.Registro

Select File

Registro.csv

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_Registro String	<input checked="" type="checkbox"/> Id_Datos_Personales String	<input checked="" type="checkbox"/> Id_Tipos String	<input checked="" type="checkbox"/> Usuario String	<input checked="" type="checkbox"/>
1	22208	8456	386	Cesar Rangel	L4
2	24583	7681	386	Pedro Ponce Flores	PF
3	21817	5251	386	Luis Enrique Zenteno	ZI
4	22804	5739	364	Fernando Yáñez Sandoval	AS
5	22128	8445	364	Fernanda Rangel	11
6	22269	5400	364	Nara Quinteros	10

Import completed 6 / 6

04:47 p.m. 18/01/2022

## COLECCIÓN TIPO DE USUARIO

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESE.TipoUsuario

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

- Direcciones
- Empleados
- Estados
- Municipios
- Registro
- TipoUsuario
- Visitantes Temporales

admin

Import To Collection ProyectoTESE.TipoUsuario

Select File

Tipo de Usuario.csv

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_Tipos	<input checked="" type="checkbox"/> Nombre_Tipos
1	364	Visitante
2	386	Empleado

Import completed 2 / 2

DONE

## COLECCIÓN VISITANTES TEMPORALES

MongoDB Compass - clustertese.g8dvg.mongodb.net/ProyectoTESE.VisitantesTemporales

Connect View Collection Help

Local

4 DBS 20 COLLECTIONS

HOSTS

- clustertese-shard-00-02.g...
- clustertese-shard-00-00.g...
- clustertese-shard-00-01.g...

CLUSTER

Replica Set (atlas-107k23-...)

3 Nodes

EDITION

MongoDB 4.4.11 Enterprise

Filter your data

- Direcciones
- Empleados
- Estados
- Municipios
- Registro
- TipoUsuario
- Visitantes Temporales

admin

Import To Collection ProyectoTESE.VisitantesTemporales

Select File

Visitantes Temporales.csv

Select Input File Type

JSON CSV

Options

Select delimiter COMMA

☒ Ignore empty strings

☐ Stop on errors

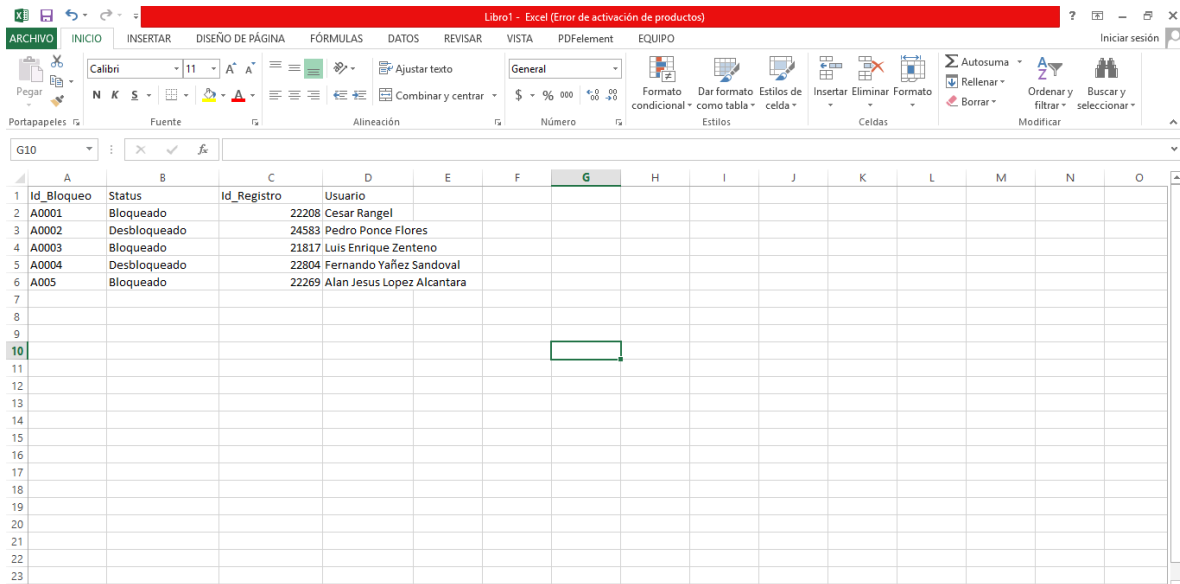
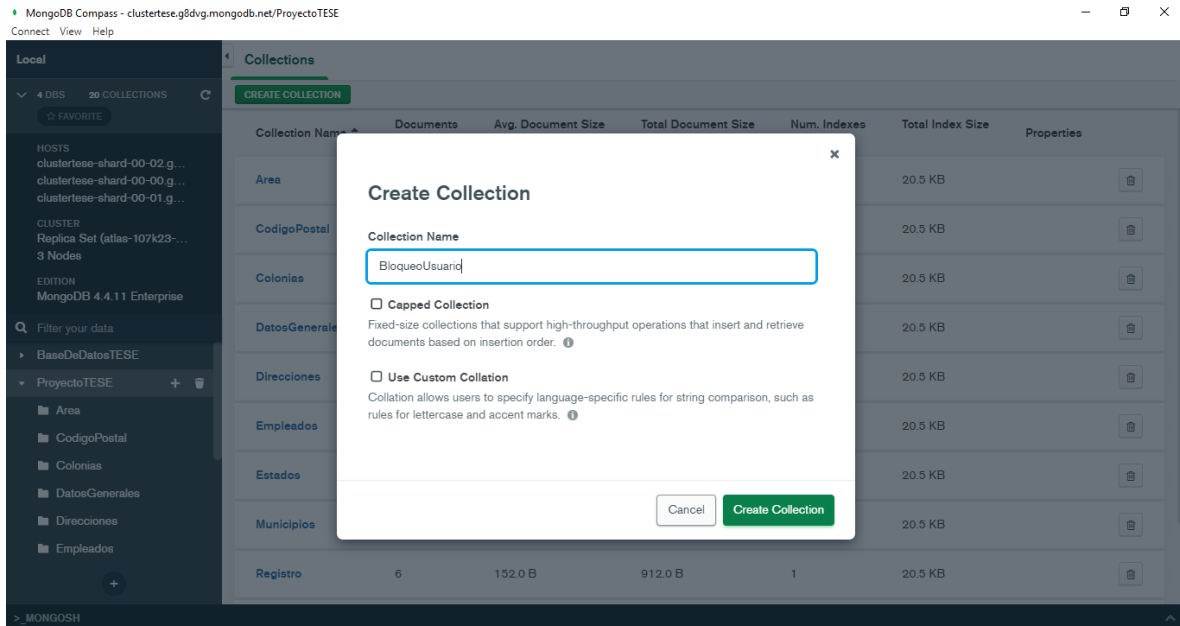
Specify Fields and Types

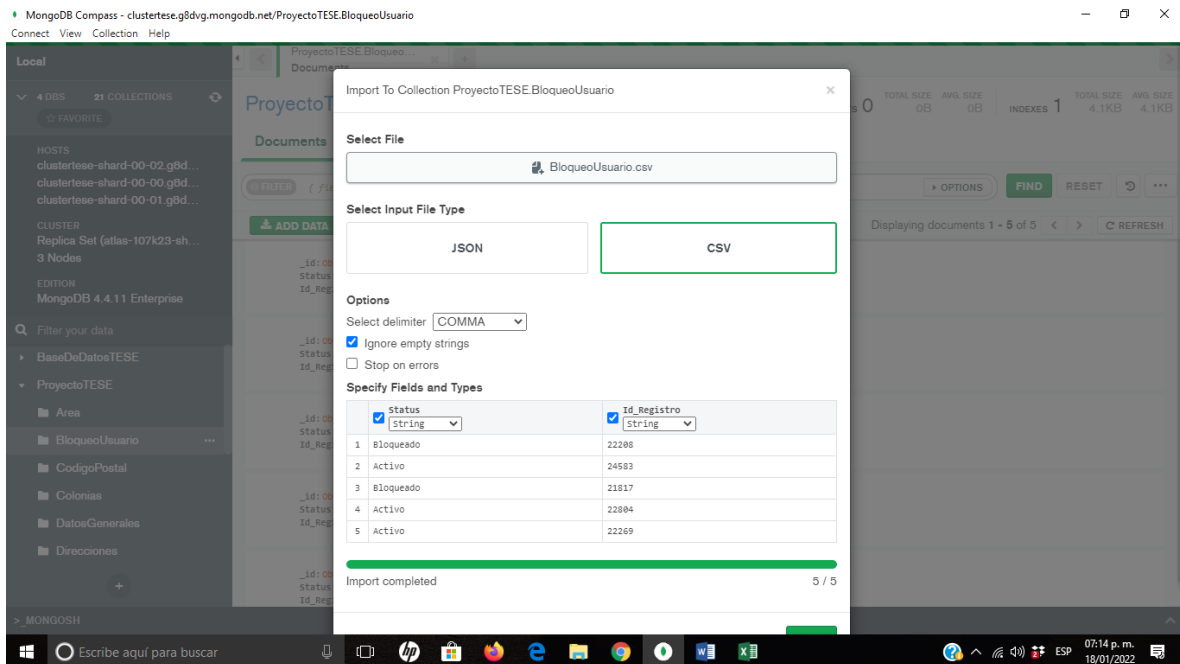
	<input checked="" type="checkbox"/> Id_Visitantes	<input checked="" type="checkbox"/> Id_Tipos	<input checked="" type="checkbox"/> Id_Datos_Personales	<input checked="" type="checkbox"/> Empresa	<input checked="" type="checkbox"/> Estado
1	V19499	364	5739	Simbo	NULL
2	V16478	364	8445	Penex	NULL
3	V17304	364	5400	Alpla	NULL

Import completed 3 / 3

DONE

## IMPLEMENTACIÓN DE LA TABLA DE BLOQUEO PARA NUESTRO MODULO





## VISTA ÚTIL DE USUARIOS BLOQUEADOS Y ACTIVOS EN LA BASE DE DATOS

Mediante la consulta en la colección del registro es posible identificar los datos del usuario por medio de la misma colección, así como el estatus de este usuario por medio de la colección de **BloqueoUsuario**.

```
{
  from: 'Registro',
  localField: 'Id_Registro',
  foreignField: 'Id_Registro',
  as: 'DatosUsuario'
}
```

## RESULTADOS DE LA BÚSQUEDA

Output after `$lookup` stage (Sample of 5 documents)

```
_id: ObjectId("61e765e39a3122f6b4f8ca43")
Status: "Bloqueado"
Id_Registro: "22208"
▼ DatosUsuario: Array
  ▼ 0: Object
    _id: ObjectId("61e7438e9a3122f6b4f8ca1f")
    Id_Registro: "22208"
    Id_Datos_Personales: "8456"
    Id_Tipos: "386"
```

```
_id: ObjectId("61e765e39a3122f6b4f8ca44")
Status: "Activo"
Id_Registro: "24583"
▼ DatosUsuario: Array
  ▼ 0: Object
    _id: ObjectId("61e7438e9a3122f6b4f8ca1f")
    Id_Registro: "24583"
    Id_Datos_Personales: "7681"
    Id_Tipos: "386"
```

```
Id_Registro: "22208"
▼ DatosUsuario: Array
  ▼ 0: Object
    _id: ObjectId("61e7438e9a3122f6b4f8ca1f")
    Id_Registro: "22208"
    Id_Datos_Personales: "8456"
    Id_Tipos: "386"
    Usuario: "Cesar Rangel"
    Contraseña: "L0812160"
```

```
Id_Registro: "24583"
▼ DatosUsuario: Array
  ▼ 0: Object
    _id: ObjectId("61e7438e9a3122f6b4f8ca1f")
    Id_Registro: "24583"
    Id_Datos_Personales: "7681"
    Id_Tipos: "386"
    Usuario: "Pedro Ponce Flores"
    Contraseña: "PFlores022"
```

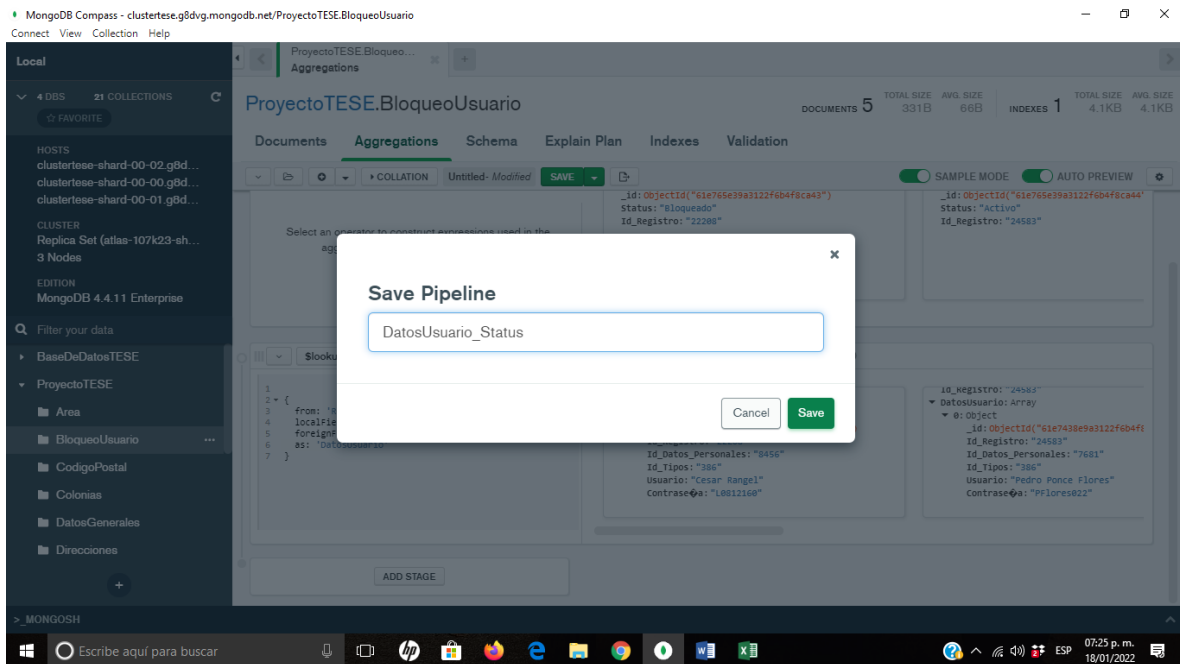
```
_id: ObjectId("61e765e39a3122f6b4f8ca46")
Status: "Activo"
Id_Registro: "22804"
▼ DatosUsuario: Array
  ▼ 0: Object
    _id: ObjectId("61e7438e9a3122f6b4f8ca22")
    Id_Registro: "22804"
    Id_Datos_Personales: "5739"
    Id_Tipos: "364"
```

```
_id: ObjectId("61e765e39a3122f6b4f8ca47")
Status: "Activo"
Id_Registro: "22269"
▼ DatosUsuario: Array
  ▼ 0: Object
    _id: ObjectId("61e7438e9a3122f6b4f8ca24")
    Id_Registro: "22269"
    Id_Datos_Personales: "5400"
    Id_Tipos: "364"
```

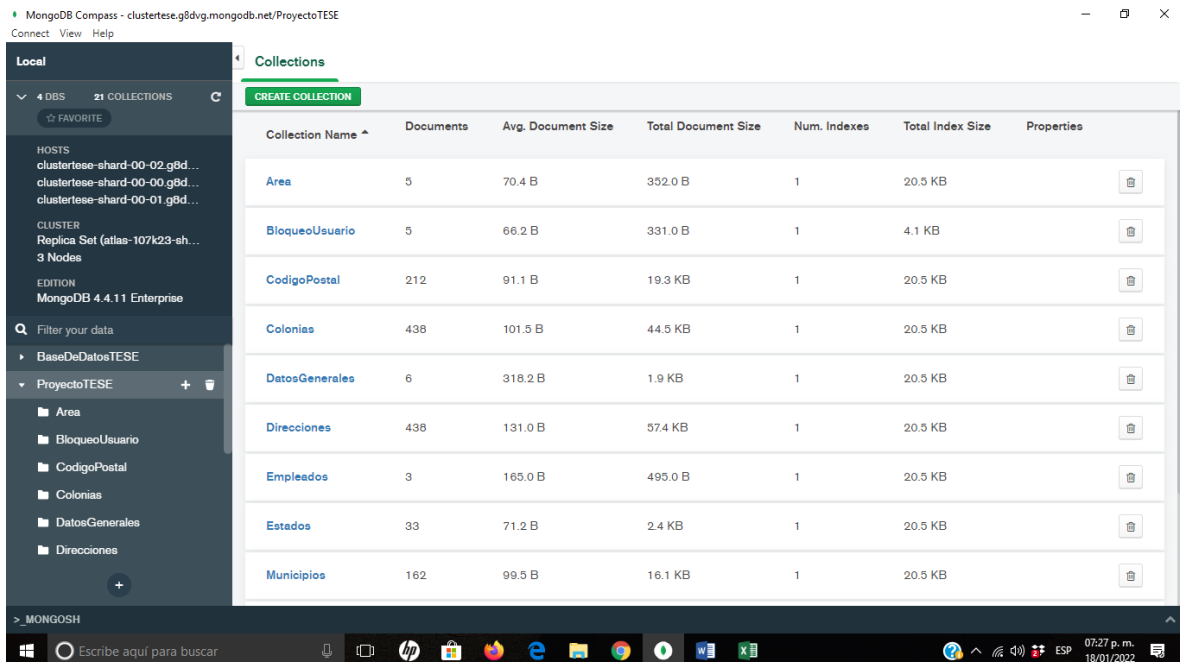
```
Id_Registro: "22804"
DatosUsuario: Array
  ▼ 0: Object
    _id: ObjectId("61e7438e9a3122f6b4f8ca22")
    Id_Registro: "22804"
    Id_Datos_Personales: "5739"
    Id_Tipos: "364"
    Usuario: "Fernando Yañez Sandoval"
    Contraseña: "As13144135"
```

```
Id_Registro: "22269"
▼ DatosUsuario: Array
  ▼ 0: Object
    _id: ObjectId("61e7438e9a3122f6b4f8ca24")
    Id_Registro: "22269"
    Id_Datos_Personales: "5400"
    Id_Tipos: "364"
    Usuario: "Alan Jesus Lopez Alcantara"
    Contraseña: "nyq980912"
```

## GUARDAMOS LA VISTA DEL MÓDULO



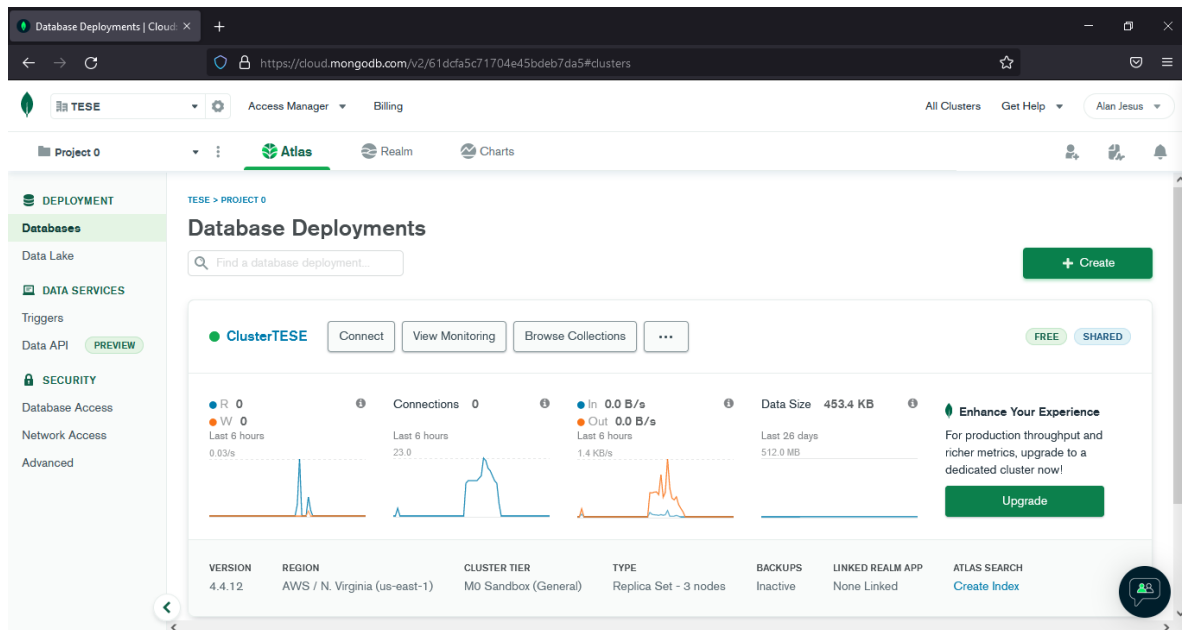
## REGISTROS ALMACENADOS EN LAS COLECCIONES



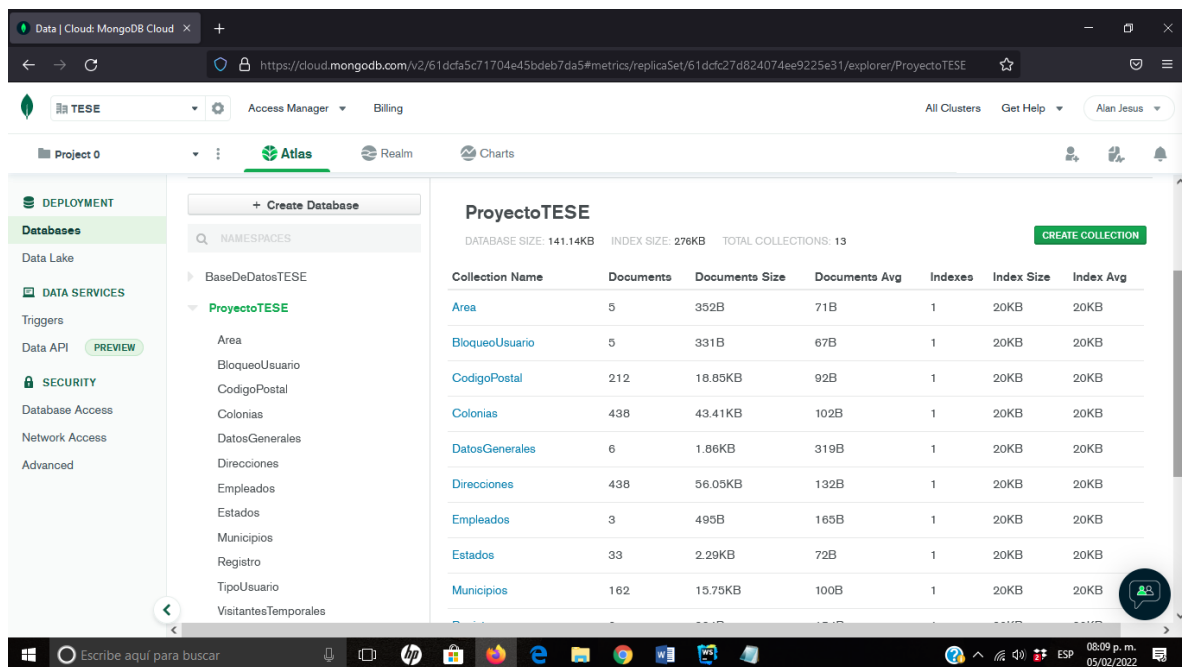




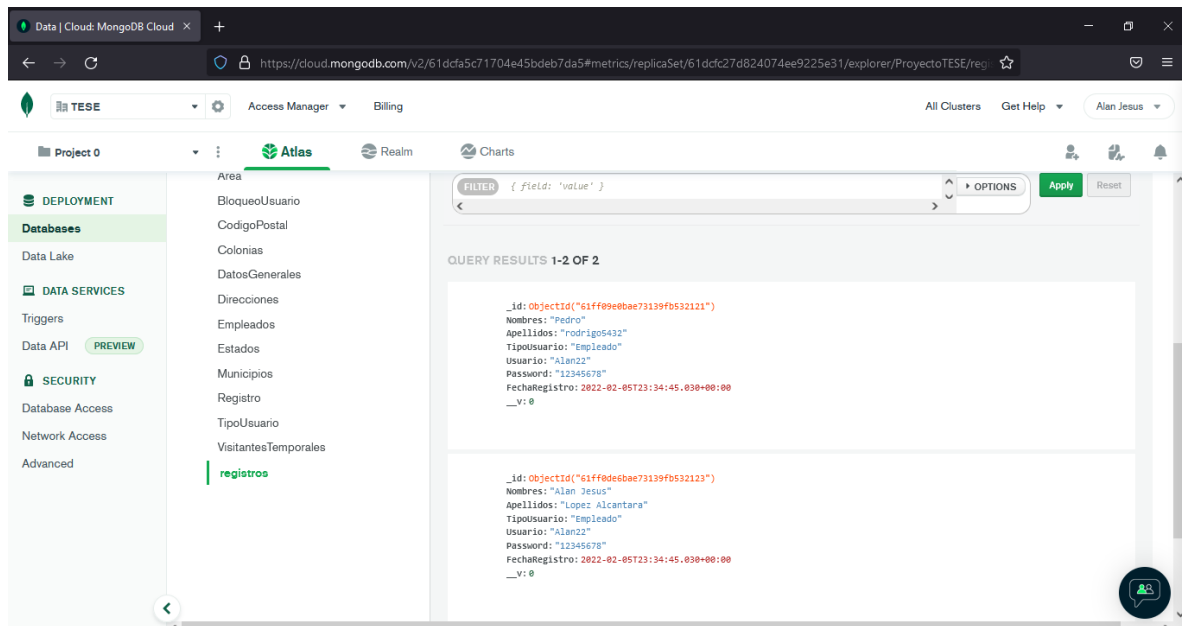
## Entorno de Mongo DB Atlas



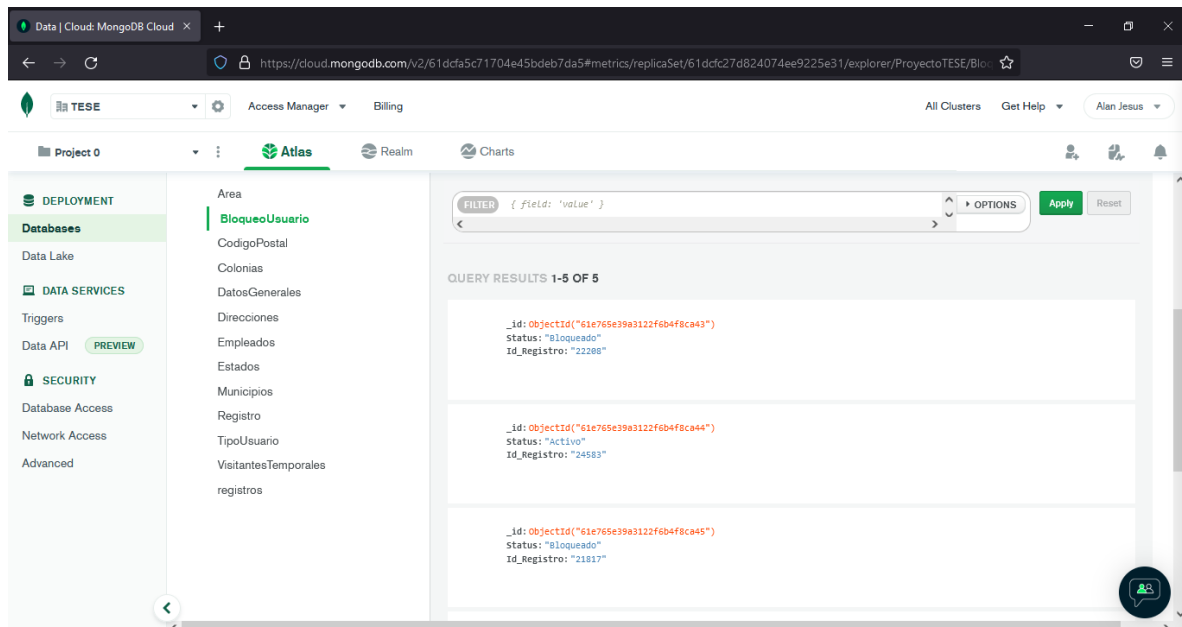
## Cluster TESE >> ProyectoTESE



## Registros del Back-end desde el Cluster TESE en Mongo Atlas



## Registros del Bloqueo\_Usuario desde el Cluster TESE en Mongo Atlas



### URL DEL CLUSTER

<https://cloud.mongodb.com/v2/61dcfa5c71704e45bdeb7da5#metrics/replicaSet/61dcfc27d824074ee9225e31/explorer/ProyectoTESE>

# AUTENTIFICACIÓN, REGISTRO Y BLOQUEO DE USUARIO

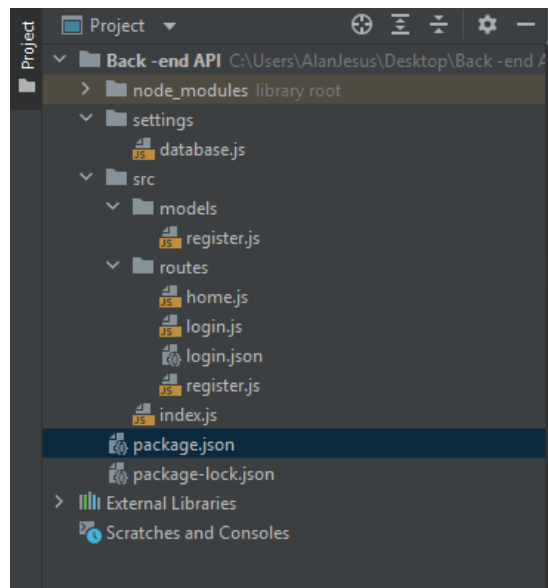
## BACK-END

Para el inicio del desarrollo del módulo asignado es importante comenzar la inicializador de nuestro servidor de express, el cual permitirá establecer la conexión con la base de datos del proyecto por medio de Mongo DB. Para esto creamos una carpeta de configuraciones que contiene todas las configuraciones de acuerdo a nuestra base de datos, en este caso la base de datos se encuentra alojada en un Cluster de Mongo DB Atlas.

### Autenticación basada en token

Para la implementación de autenticación de usuarios por medio de roles, es decir de protección de rutas de acuerdo al usuario que se registra lo hacemos mediante (JWT) en comparación con la autenticación basada en sesión que necesita almacenar la sesión de una cookie, la gran ventaja de la autenticación basada en token es que almacenamos el token es que almacenamos el token web JSON en el lado del cliente.

### Estructura del proyecto



### Settings/database.js

```
const mongoose = require ('mongoose');
const url =
'mongodb+srv://Alan11:knotslip@clustertese.g8dvg.mongodb.net/ProyectoTESE
';
mongoose.connect(url, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then (dato =>{
  console.log('Base de datos conectada');
})
.catch (error => {
  console.log('Error de conexión', error)
})
```

## Conexión de la Base de datos

```
Servidor en puerto 4500
Base de datos conectada
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
Version Control  TODO  Problems  Terminal
```

También para el desarrollo del Back –end es importante establecer los archivos que alojan cada una de las direcciones o rutas, por ejemplo en el caso del módulo de autenticación, registro y bloqueo se generan tres direcciones, Login, home y register. Por lo tanto creamos cada uno de los archivos js.

### routes/home

```
const {Router} = require ('express');
const router = Router ();

router.get('/', (req, res)=>{
  res.send('BIENVENIDO AL HOME');
});
module.exports = router;
```

### routes/Login

```
const {Router} = require('express');
const router = Router();
const bloqueo = require('underscore');
const singin = require('../routes/login.json');
router.get('/login', (req, res)=>{
  res.json(singin);
});
router.post('/login', (req,res)=>{
  const {username, password} =(req.body);
  if(username && password){
    const nuevoUsuario = {...req.body};
    singin.push(nuevoUsuario);
    res.json(singin);
  }else{
    res.send('USUARIO NO REGISTRADO');
  }
});
router.delete('./login/:username', (req,res)=>{
  const { username } = req.params;
  bloqueo.each(singin, (usuario, i)=>{
    if(usuario.username==username){
      singin.splice(i,1);
      res.json(singin);
      console.log('SE BLOQUEO EL USUARIO');
    }else{
      res.send('ERROR AL BLOQUEAR');
    }
  });
});
```

```

});
router.put('./login:/username', (req,res)=>{
  const {username} = req.params;
  const {password} = req.body;
  if(usuario.username==username){
    bloqueo.each(singin, (usuario, i)=>{
      if(usuario.username==username){
        singin.username=username;
        singin.password=password;
        res.json(singin);
        console.log('SE ACTUALIZÓ USUARIO');
      }else{
        res.send('ERROR AL ACTUALIZAR');
      }
    });
  }
});
module.exports = router;

```

## routes/register

```

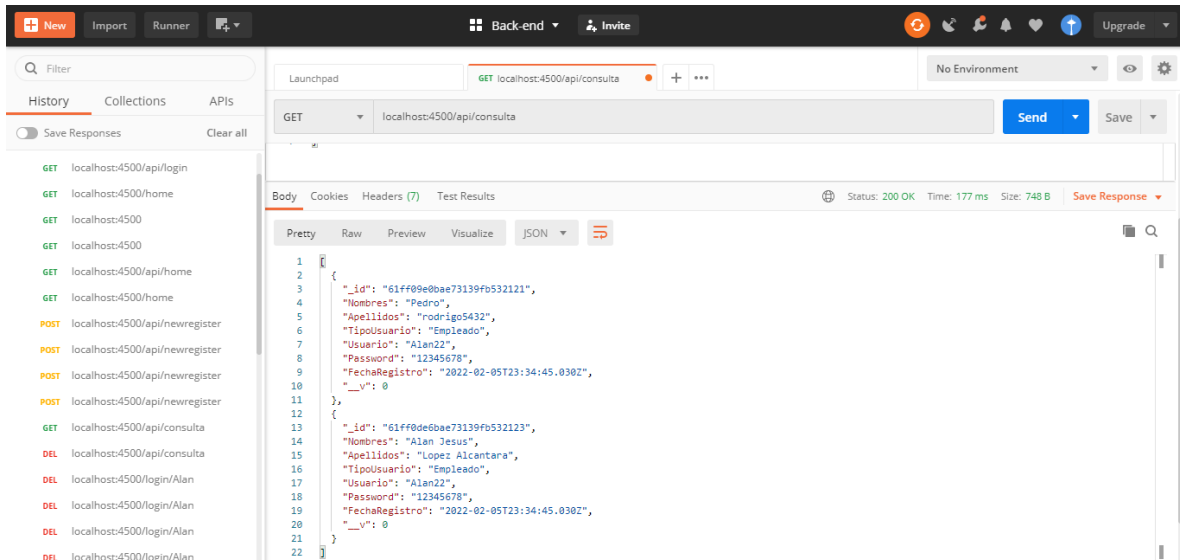
const Router = require ('express');
const router = Router();
const modeloRegistro = require ('../models/register');

router.get('/', async (req,res)=>{
  const usuariosRegistro = await modeloRegistro.find();
  res.json(usuariosRegistro);
  res.send('CONSULTA DE REGISTROS')
});
router.post('/', async (req,res)=>{
  const {Nombres, Apellidos, TipoUsuario, Usuario, Password} =
req.body;
  const nuevoUsuario = new modeloRegistro({Nombres, Apellidos,
TipoUsuario, Usuario, Password} );
  console.log(nuevoUsuario);
  await nuevoUsuario.save();

  //console.log(req.body);
  res.json(nuevoUsuario);
  //res.send('REGISTRO DE USUARIO EXITOSO');
});
module.exports = router;

```

## Verificación de registros desde POSTMAN



Estas rutas son funcionales y es importante que sean declaradas en el archivo index.js, el cual contiene cada una de las rutas, así como la inicialización del servidor y la base de datos.

### Index.js

```
const express = require ('express');
const app = express();
const morgan = require ('morgan');
require('../settings/database');

//Configuraciones
app.set('port', process.env.PORT ||4500);
app.set('json spaces', 2);
app.use(morgan('dev'));
app.use(express.urlencoded({extended: false}));
app.use(express.json());

//Configuracion de Rutas
app.use(require('./routes/home'));
app.use('/api/login', require('./routes/login'));
app.use('/api/consulta', require('./routes/register'));
app.use('/api/newregister', require ('./routes/register'));

app.listen (app.get('port'), ()=>{
  console.log(`Servidor en puerto ${app.get('port')}`);
});
```

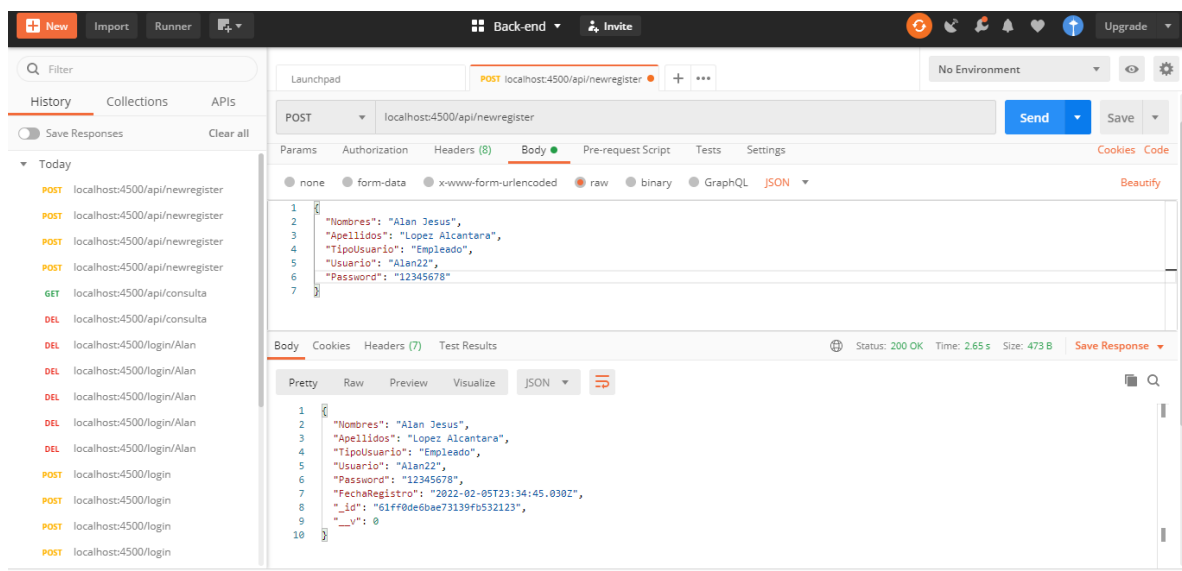
Ahora finalmente, para comprobar que el registro de un nuevo usuario es posible es necesario constatar que el modelo que se diseña en el back –end corresponda a los mismos valores que el front-end, así mismo como se encuentre especificado en el modelo de la base de datos, esto debido a que se requiere la creación de un esquema que contiene los valores que se importarán desde front-end al servidor y posteriormente a la base de datos.

## Model/register

```
const {Schema, model} = require ('mongoose');
const registerSchema = new Schema({
  Nombres: {
    type: String,
    require: true
  },
  Apellidos: {
    type: String,
    require: true
  },
  TipoUsuario: {
    type: String,
    require: true
  },
  Usuario: {
    type: String,
    require: true
  },
  Password: {
    type: String,
    require: true
  },
  FechaRegistro: {
    type: Date,
    default: Date.now()
  }
})
module.exports = model('registro', registerSchema);
```

Finalmente para corroborar que la base de datos tiene contacto con el servidor y aloja los datos podemos hacerlo desde POSTMAN, así mismo como el la consola de comandos y finalmente verificando que los registros se almacenan en la base de datos del proyecto desde el Cluster.

## Verificación de registro de usuario en POSTMAN

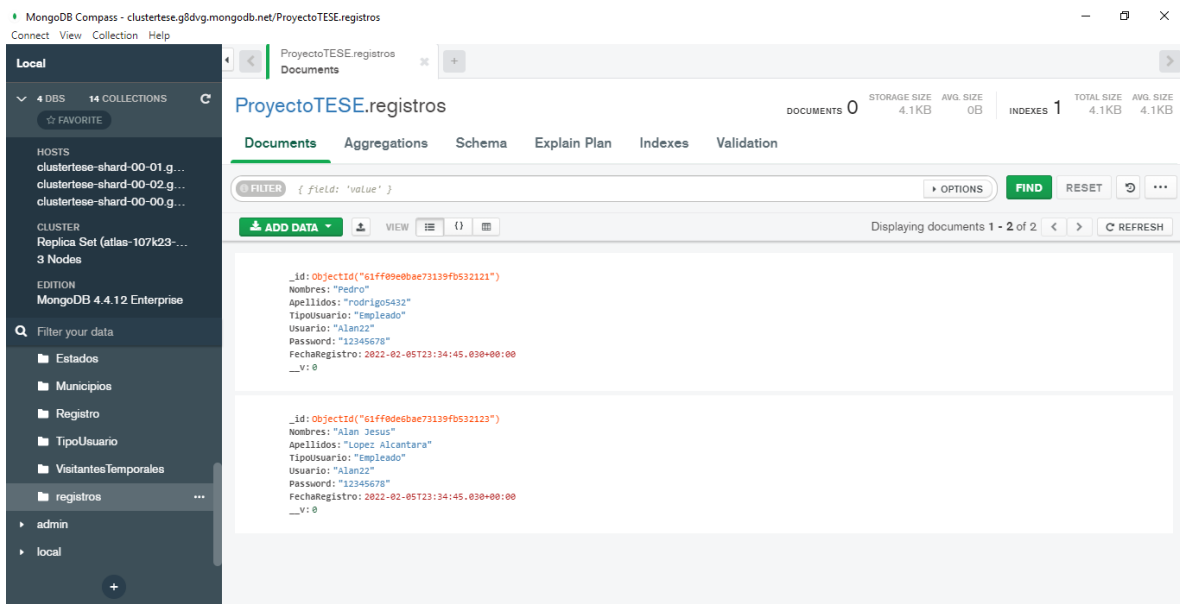


## Verificación en la consola de comandos de Webstorm

```
Terminal: Local x + v
POST /api/newregister 200 1496.863 ms - 228
{
  Nombres: 'Alan Jesus',
  Apellidos: 'Lopez Alcantara',
  TipoUsuario: 'Empleado',
  Usuario: 'Alan22',
  Password: '12345678',
  FechaRegistro: 2022-02-05T23:34:45.030Z,
  _id: new ObjectId("61ff0de6bae73139fb532123")
}
POST /api/newregister 200 960.764 ms - 237
```

```
Terminal: Local x + v
{
  Nombres: 'Pedro',
  Apellidos: 'rodrigo5432',
  TipoUsuario: 'Empleado',
  Usuario: 'Alan22',
  Password: '12345678',
  FechaRegistro: 2022-02-05T23:26:44.017Z,
  _id: new ObjectId("61ff07dba9ab02065f710839")
}
[nodemon] restarting due to changes...
[nodemon] starting 'node src/index.js'
Servidor en puerto 4500
```

## Verificación en la BD de Mongo DB del Proyecto





## FRONT – END

Para la integración del módulo, que en particular era la implementación de una protección de rutas o bloqueo de usuario se realizó por medio de la funcionalidad de autenticación de un usuario por medio de su registro previo al sistema y su posterior intento de acceso por medio de su nombre de usuario y contraseña. De esta manera la protección de las rutas se ejerce por medio de la autenticación de estos parámetros. Si la información registrada inicialmente del usuario es correcta al loguear, este tendrá acceso al Home, de lo contrario el sistema negará el acceso o lo bloqueará en caso de ser necesario.

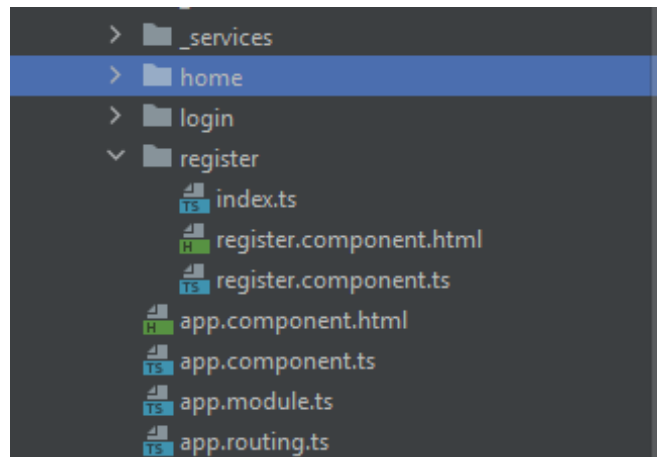
Para la implementación de esta funcionalidad a nuestro módulo se utilizó la estructura principal del módulo que menciona el uso de GUARDS, que son interfaces que permiten la protección de las rutas y especifican que tipo de usuarios pueden tener acceso a cierta dirección y que usuarios no.

Pero para probar esta funcionalidad, se realizó el **Frontend** la creación de un formulario tanto de **Login**, como de **Registro**, esto para la extracción de los datos del usuario y que en el **Backend** se realicen todas las operaciones de autenticación necesarias para la protección de las rutas.

Se crearon directorios que contienen alertas y elementos visuales que permiten que la interfaz con la que interactúa el usuario sea más entendible.

Pero enfocándonos en los directorios más importantes, tenemos tres de ellos, que son: Home, Login y Register.

Estos directorios cuentan cada uno de ellos con sus tres archivos principales como el index, el html y el component.ts.



### DIRECTORIO HOME

#### *Home.component.html*

Estas funciones van a permitirnos dar la bienvenida al usuario, registrar los usuarios que se encuentran registrados y nos servirá como ruta principal de protección para usuarios no logueados.

```
<h1>¡BIENVENIDO! {{currentUser.firstName}}</h1>
<p>CONTROL DE REGISTRO DE USUARIOS</p>
```

```

<h3>USUARIOS REGISTRADOS:</h3>

<ul>
  <li *ngFor="let user of users">
    {{user.username}} ({{user.firstName}} {{user.lastName}})
    ({{user.usertype}})
    - <a (click)="deleteUser(user.id)" class="text-danger">Bloquear
    Usuario</a>
    - <a (click)="enableUser(user.id)" class="text-danger">Habilitar
    Usuario</a>
  </li>
</ul>

```

### ***Home.component.ts***

En este bloque se van a alojar los datos de los usuarios que se registran en el sistema y que pueden ser manipulados directamente desde el sitio para su eliminación.

```

import { Component, OnInit } from '@angular/core';
import { first } from 'rxjs/operators';

import { User } from '@/_models';
import { UserService, AuthenticationService } from '@/_services';

@Component({ templateUrl: 'home.component.html' })
export class HomeComponent implements OnInit {
  currentUser: User;
  users = [];

  constructor(
    private authenticationService: AuthenticationService,
    private userService: UserService
  ) {
    this.currentUser = this.authenticationService.currentUserValue;
  }

  ngOnInit() {
    this.loadAllUsers();
  }

  deleteUser(id: number) {
    this.userService.delete(id)
      .pipe(first())
      .subscribe(() => this.loadAllUsers());
  }

  private loadAllUsers() {
    this.userService.getAll()
      .pipe(first())
      .subscribe(users => this.users = users);
  }
}

```

Para el archivo index únicamente exportamos el contenido del componente.

```
export * from './home.component';
```

## DIRECTORIO LOGIN

### *Login.component.html*

En este archivo se contienen todas las etiquetas de la interfaz, los campos a llenar y algunas alertas y restricciones que se le asignan al usuario, como la asignación del llenado de los campos o del número mínimo de caracteres del password. También contiene las redirecciones al registro de nuevos usuarios.

```
</div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" formControlName="password" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.password.errors }" />
    <div *ngIf="submitted && f.password.errors" class="invalid-
feedback">
      <div *ngIf="f.password.errors.required">La contraseña es
necesaria</div>
    </div>
  </div>
  <div class="form-group">
    <button [disabled]="loading" class="btn btn-primary">
      <span *ngIf="loading" class="spinner-border spinner-border-sm
mr-1"></span>
      Login
    </button>
    <a routerLink="/register" class="btn btn-link">Registrarse</a>
  </div>
</form>
```

### *Login.component.ts*

En este archivo se contienen los elementos de retorno y parámetros de autenticación, se liga con las alertas y se cumplen las funcionalidades, en caso de que el usuario y el password sean correctos se redirigirá a el Home o página principal y en caso de ser incorrectos aparecerán las alertas y los mensajes que indiquen que el usuario o contraseña son incorrectas.

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AlertService, AuthenticationService } from '@/_services';

@Component({ templateUrl: 'login.component.html' })
export class LoginComponent implements OnInit {
  loginForm: FormGroup;
  loading = false;
  submitted = false;
  returnUrl: string;

  constructor(
    private formBuilder: FormBuilder,
    private route: ActivatedRoute,
    private router: Router,
```

```

        private authenticationService: AuthenticationService,
        private alertService: AlertService
    ) {
        // redirigir a Home si ya ha iniciado sesión
        if (this.authenticationService.currentUserValue) {
            this.router.navigate(['/']);
        }
    }

    ngOnInit() {
        this.loginForm = this.formBuilder.group({
            username: ['', Validators.required],
            password: ['', Validators.required]
        });

        // obtener la URL de retorno de los parámetros de la ruta o por defecto a '/'
        this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';
    }

    // captador de conveniencia para un fácil acceso a los campos de formulario
    get f() { return this.loginForm.controls; }

    onSubmit() {
        this.submitted = true;

        // restablecer alertas al enviar
        this.alertService.clear();

        // deténgase aquí si el formulario no es válido
        if (this.loginForm.invalid) {
            return;
        }

        this.loading = true;
        this.authenticationService.login(this.f.username.value, this.f.password.value)
            .pipe(first())
            .subscribe(
                data => {
                    this.router.navigate([this.returnUrl]);
                },
                error => {
                    this.alertService.error(error);
                    this.loading = false;
                }
            );
    }
}

```

Y de igual forma el index contiene la exportación del componente.

```
export * from './login.component';
```

## DIRECTORIO REGISTRO

Inicialmente, de igual manera que con los otros componentes, este se exporta en el index.

```
export * from './register.component';
```

### *Register.component.html*

Este archivo contiene todos los elementos necesarios para el registro de los usuarios, los campos en donde el usuario añade sus datos y los botones de retorno o de registro. El usuario recibirá las alertas sobre los campos que son obligatorios o las sentencias requeridas, así como mensajes que puedan determinar al usuario si su registro fue exitoso o si el usuario ya se encuentra registrado.

```
</div>
  </div>
  <div class="form-group">
    <label for="username">Usuario</label>
    <input type="text" formControlName="username" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.username.errors }" />
    <div *ngIf="submitted && f.username.errors" class="invalid-
feedback">
      <div *ngIf="f.username.errors.required">Obligatorio</div>
    </div>
  </div>

  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" formControlName="password" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.password.errors }" />
    <div *ngIf="submitted && f.password.errors" class="invalid-
feedback">
      <div *ngIf="f.password.errors.required">Obligatorio</div>
      <div *ngIf="f.password.errors.minlength">El password debe
contener al menos 6 caracteres</div>
    </div>
  </div>

  <div class="form-group">
    <button [disabled]="loading" class="btn btn-primary">
      <span *ngIf="loading" class="spinner-border spinner-border-sm
mr-1"></span>
      Registrarse
    </button>
    <a routerLink="/login" class="btn btn-link">Cancelar</a>
  </div>
</form>
```

### ***Register.component.ts***

Este archivo contiene los elementos necesarios que validen los campos y los datos que se introducen en ellos, así como su extracción y registro de usuarios. Es el encargado de enviar los mensajes al usuario sobre su registro y almacena la información necesaria para la autenticación.

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AlertService, UserService, AuthenticationService } from
'@/_services';

@Component({ templateUrl: 'register.component.html' })
export class RegisterComponent implements OnInit {
  registerForm: FormGroup;
  loading = false;
  submitted = false;

  constructor(
    private formBuilder: FormBuilder,
    private router: Router,
    private authenticationService: AuthenticationService,
    private userService: UserService,
    private alertService: AlertService
  ) {
    //redirigir a Home si ya ha iniciado sesión
    if (this.authenticationService.currentUserValue) {
      this.router.navigate(['/']);
    }
  }

  ngOnInit() {
    this.registerForm = this.formBuilder.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      username: ['', Validators.required],
      usernameType: ['', Validators.required],
      password: ['', [Validators.required,
Validators.minLength(6)]]
    });

    //captador de conveniencia para un fácil acceso al campo de
formulario
    get f() { return this.registerForm.controls; }

    onSubmit() {
      this.submitted = true;

      // restablecer alertas al enviar
      this.alertService.clear();

      // deténgase aquí si el formulario no es válido
      if (this.registerForm.invalid) {
```

```

        return;
    }

    this.loading = true;
    this.userService.register(this.registerForm.value)
        .pipe(first())
        .subscribe(
            data => {
                this.alertService.success('Registro exitoso', true);
                this.router.navigate(['/login']);
            },
            error => {
                this.alertService.error(error);
                this.loading = false;
            });
    }
}

```

## AUTENTIFICACIÓN DEL USUARIO

Para ello como lo habíamos mencionado inicialmente, se implementó un GUARD, esto con CanActivate, esta sentencia funciona con valores booleanos, se verifica que inicialmente estamos en la ruta del Login, se introducen los datos, si estos se encuentran registrados en el sistema y el usuario y password coinciden, retorna un true y redirige al usuario al HOME, de lo contrario retorna un false y lo redirige nuevamente al LOGIN.

```

import { Injectable } from '@angular/core';
import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot }
    from '@angular/router';

import { AuthenticationService } from '@/_services';

@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {
    constructor(
        private router: Router,
        private authenticationService: AuthenticationService
    ) {}

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        const currentUser = this.authenticationService.currentUserValue;
        if (currentUser) {
            // Si se encuentra autorizado retorna un true
            return true;
        }

        // si no se encuentra logueado o no tiene acceso se retorna un
        false y se redirige a el LOGIN
        this.router.navigate(['/login'], { queryParams: { returnUrl:
            state.url } });
        return false;
    }
}

```

Finalmente tenemos los archivos generales de la aplicación como lo son el html, component.ts, module.ts y el archivo de routing.ts

#### ***App.component.html***

```
<nav class="navbar navbar-expand navbar-dark bg-dark"
*ngIf="currentUser">
  <div class="navbar-nav">
    <a class="nav-item nav-link" routerLink="/">Home</a>
    <a class="nav-item nav-link" (click)="logout()">Salir</a>
  </div>
</nav>

<div class="jumbotron">
  <div class="container">
    <div class="row">
      <div class="col-sm-6 offset-sm-3">
        <alert></alert>
        <router-outlet></router-outlet>
      </div>
    </div>
  </div>
</div>
```

#### ***App.module.ts***

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule, HTTP_INTERCEPTORS } from
'@angular/common/http';

// utilizado para crear backend falso
import { fakeBackendProvider } from './_helpers';

import { appRoutingModule } from './app.routing';
import { JwtInterceptor, ErrorInterceptor } from './_helpers';
import { AppComponent } from './app.component';
import { HomeComponent } from './home';
import { LoginComponent } from './login';
import { RegisterComponent } from './register';
import { AlertComponent } from './_components';

@NgModule({
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    HttpClientModule,
    appRoutingModule
  ],
  declarations: [
    AppComponent,
    HomeComponent,
    LoginComponent,
    RegisterComponent,
```



```

        AlertComponent
    ],
    providers: [
        { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi:
true },
        { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi:
true },

        // proveedor utilizado para crear backend falso
        fakeBackendProvider
    ],
    bootstrap: [AppComponent]
})
export class AppModule { };

```

### ***App.component.ts***

```

import { Component } from '@angular/core';
import { Router } from '@angular/router';

import { AuthenticationService } from '../_services';
import { User } from '../_models';

import '../_content/app.less';

@Component({ selector: 'app', templateUrl: 'app.component.html' })
export class AppComponent {
    currentUser: User;

    constructor(
        private router: Router,
        private authenticationService: AuthenticationService
    ) {
        this.authenticationService.currentUser.subscribe(x =>
this.currentUser = x);
    }

    logout() {
        this.authenticationService.logout();
        this.router.navigate(['/login']);
    }
}

```

### ***App.routing.ts***

```

import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from '../home';
import { LoginComponent } from '../login';
import { RegisterComponent } from '../register';
import { AuthGuard } from '../_helpers';

const routes: Routes = [
    { path: '', component: HomeComponent, canActivate: [AuthGuard] },
    { path: 'login', component: LoginComponent },
    { path: 'register', component: RegisterComponent },

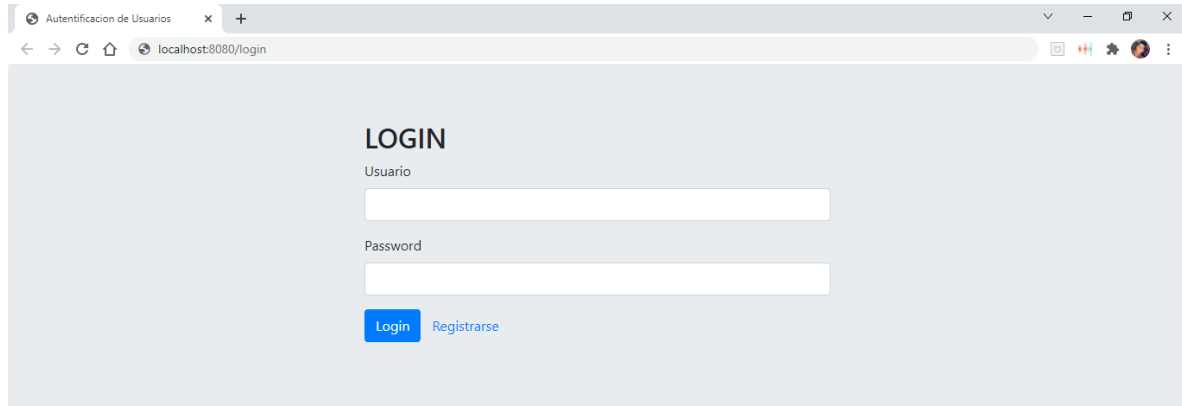
```

```
// Redirecciona a Home
{ path: '**', redirectTo: '/' }
];

export const appRoutingModule = RouterModule.forRoot(routes);
```

En este apartado mostramos la interfaz con la que el usuario interactúa, sin identificar las funcionalidades. Para esto mostramos inicialmente las siguientes pantallas.

### INTERFAZ LOGIN

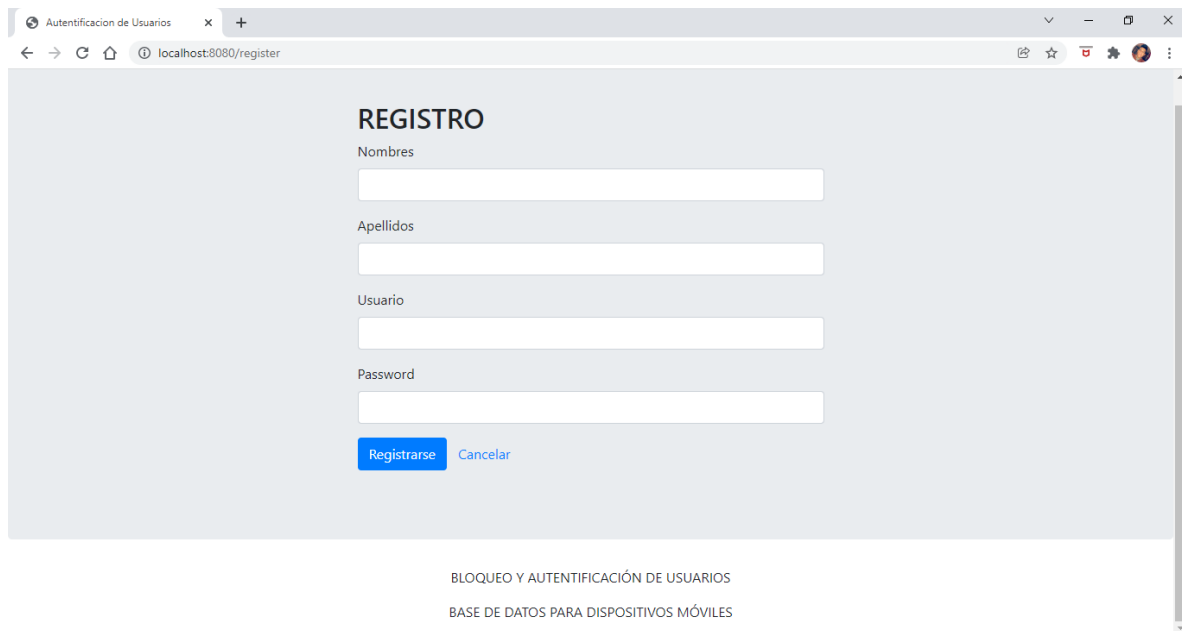


A screenshot of a web browser showing the 'LOGIN' page. The browser's address bar displays 'localhost:8080/login'. The page has a light gray background. At the top, the word 'LOGIN' is centered in a bold, black font. Below it, the label 'Usuario' is followed by a white input field. Further down, the label 'Password' is followed by another white input field. At the bottom, there are two buttons: a blue 'Login' button and a blue 'Registrarse' button.

BLOQUEO Y AUTENTIFICACIÓN DE USUARIOS

BASE DE DATOS PARA DISPOSITIVOS MÓVILES

### INTERFAZ REGISTRO



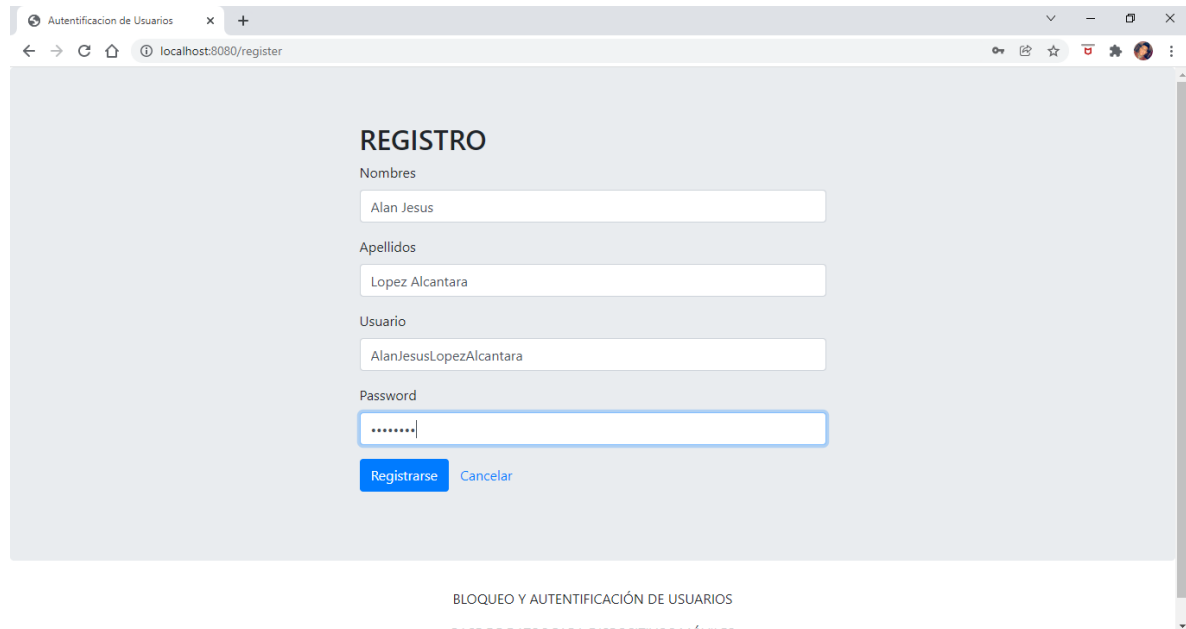
A screenshot of a web browser showing the 'REGISTRO' page. The browser's address bar displays 'localhost:8080/register'. The page has a light gray background. At the top, the word 'REGISTRO' is centered in a bold, black font. Below it, there are four labels with corresponding white input fields: 'Nombres', 'Apellidos', 'Usuario', and 'Password'. At the bottom, there are two buttons: a blue 'Registrarse' button and a blue 'Cancelar' button.

BLOQUEO Y AUTENTIFICACIÓN DE USUARIOS

BASE DE DATOS PARA DISPOSITIVOS MÓVILES

Estas pantallas nos muestran los campos de acceso y de registro al sistema. Ahora identificaremos el funcionamiento mediante una prueba de registro de un usuario.

## REGISTRO USUARIO



Autenticación de Usuarios x +

localhost:8080/register

### REGISTRO

Nombres

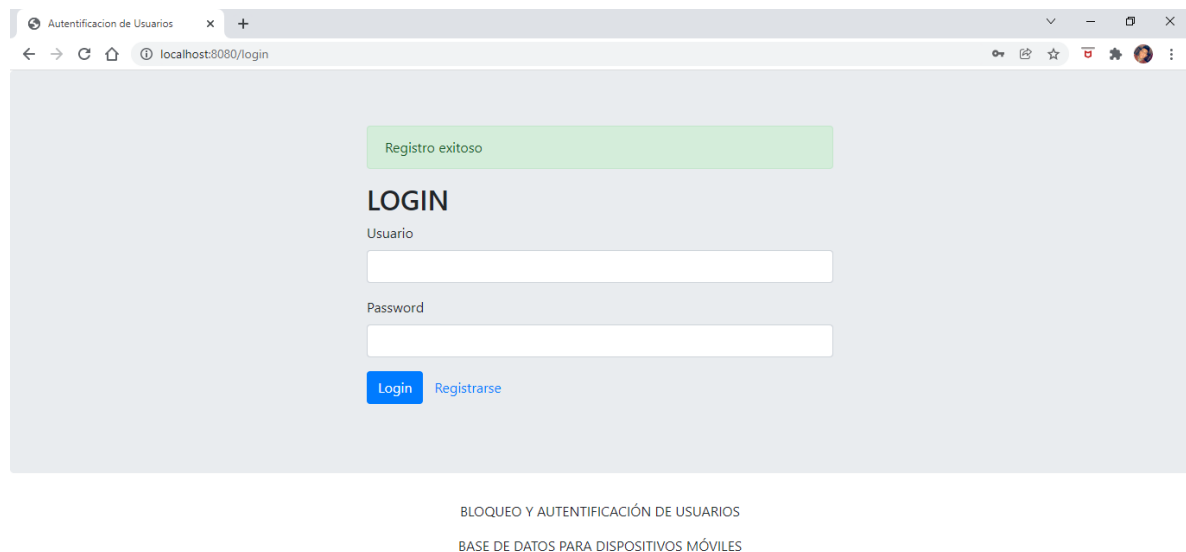
Apellidos

Usuario

Password

[Registrarse](#) [Cancelar](#)

BLOQUEO Y AUTENTIFICACIÓN DE USUARIOS  
BASE DE DATOS PARA DISPOSITIVOS MÓVILES



Autenticación de Usuarios x +

localhost:8080/login

Registro exitoso

### LOGIN

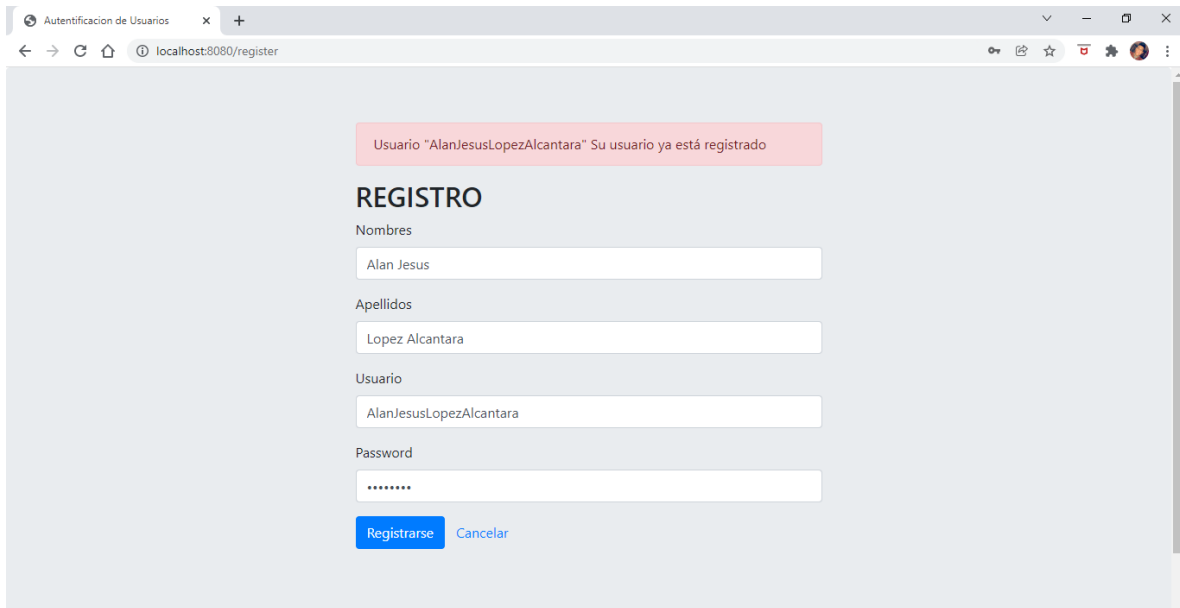
Usuario

Password

[Login](#) [Registrarse](#)

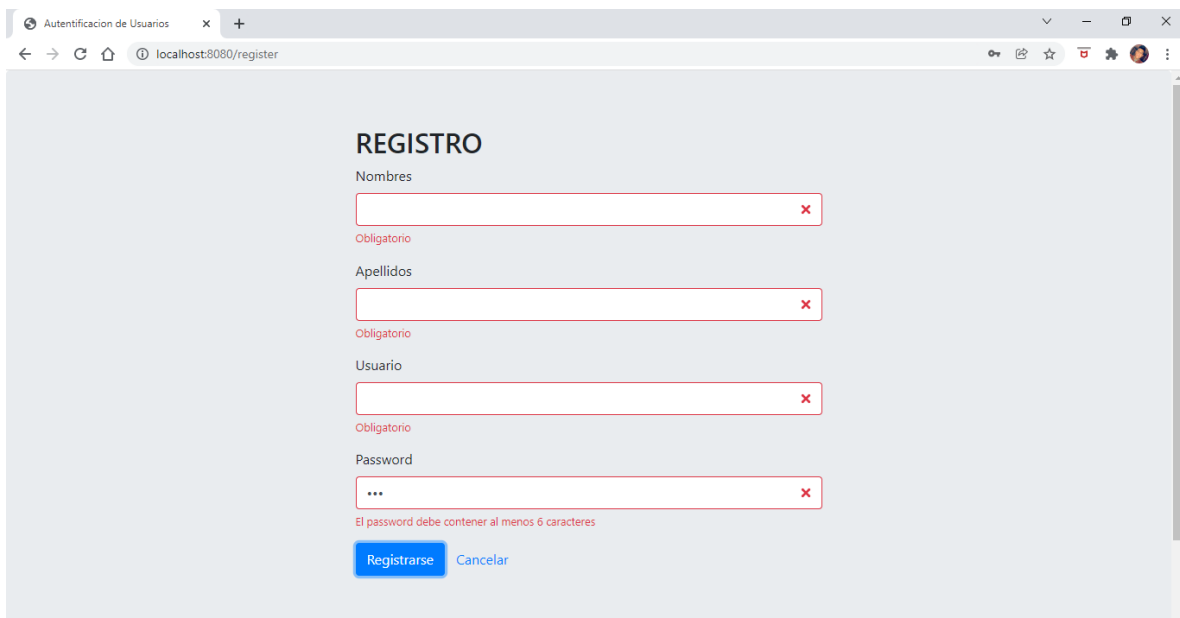
BLOQUEO Y AUTENTIFICACIÓN DE USUARIOS  
BASE DE DATOS PARA DISPOSITIVOS MÓVILES

## REGISTRO DE USUARIO YA ALMACENADO



A screenshot of a web browser window with the title "Autenticación de Usuarios". The address bar shows "localhost:8080/register". A pink message box at the top states: "Usuario 'AlanJesusLopezAlcantara' Su usuario ya está registrado". Below this, the heading "REGISTRO" is followed by four input fields: "Nombres" (containing "Alan Jesus"), "Apellidos" (containing "Lopez Alcantara"), "Usuario" (containing "AlanJesusLopezAlcantara"), and "Password" (containing "\*\*\*\*\*"). At the bottom are two buttons: "Registrarse" (blue) and "Cancelar" (grey).

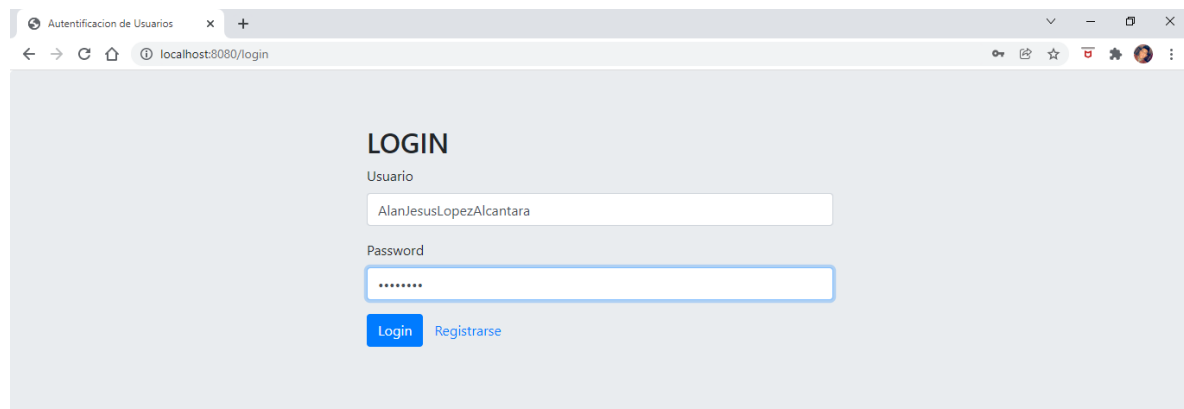
## ALERTAS DE REGISTRO



A screenshot of the same web browser window, but with validation alerts. Each input field is outlined in red and has a red "x" icon in the top right corner. Below each field is the word "Obligatorio" in red. The "Password" field has a red message below it: "El password debe contener al menos 6 caracteres". The "Registrarse" and "Cancelar" buttons remain at the bottom.

Ahora comprobaremos que el acceso puede darse mediante el Login.

## LOGIN DE USUARIO REGISTRADO



BLOQUEO Y AUTENTIFICACIÓN DE USUARIOS

BASE DE DATOS PARA DISPOSITIVOS MÓVILES

En la siguiente pantalla podemos comprobar que el acceso a la ruta mediante el Login fue exitoso y de igual manera se almacenaron tanto nuestros datos de usuario, como los nombres y Apeidos. (Aparecen tres registros por pruebas preliminares).



BLOQUEO Y AUTENTIFICACIÓN DE USUARIOS

BASE DE DATOS PARA DISPOSITIVOS MÓVILES

Pero al dar clic en Eliminar Usuario podemos sacarlo del sistema. Y posteriormente al intentar acceder por medio de este mismo el sistema solicitará que vuelva a registrarse.

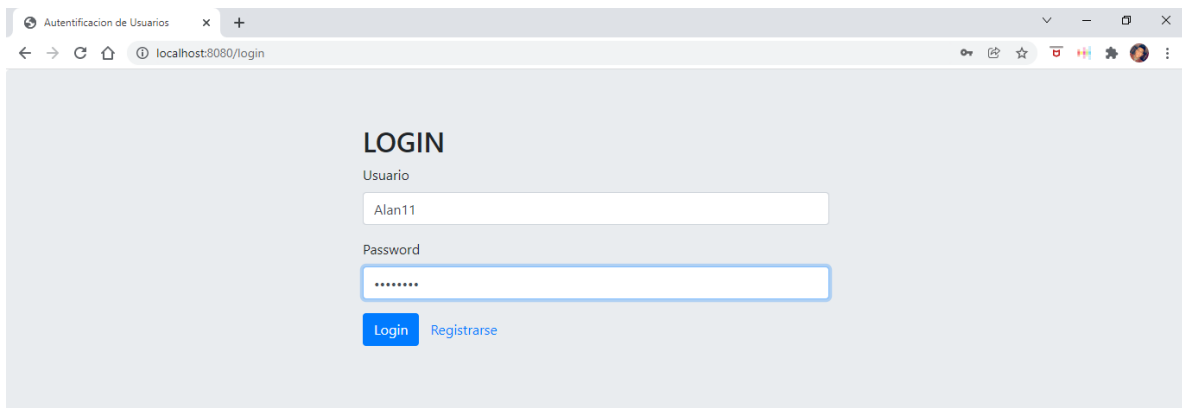
## BLOQUEO DE USUARIO



BLOQUEO Y AUTENTICACIÓN DE USUARIOS

BASE DE DATOS PARA DISPOSITIVOS MÓVILES

## Intento de acceso de usuario bloqueado.



BLOQUEO Y AUTENTICACIÓN DE USUARIOS

BASE DE DATOS PARA DISPOSITIVOS MÓVILES

Autenticación de Usuarios x +

localhost:8080/login

Su usuario se encuentra bloqueado, comuníquese con soporte técnico

## LOGIN

Usuario

Password

[Login](#) [Registrarse](#)

BLOQUEO Y AUTENTICACIÓN DE USUARIOS

BASE DE DATOS PARA DISPOSITIVOS MÓVILES

## ACCESO USUARIO REGISTRADO CON PASSWORD O USUARIO INCORRECTO

Autenticación de Usuarios x +

localhost:8080/login

Usuario o password incorrecta

## LOGIN

Usuario

Password

[Login](#) [Registrarse](#)

BLOQUEO Y AUTENTICACIÓN DE USUARIOS

BASE DE DATOS PARA DISPOSITIVOS MÓVILES

## CONCLUSIÓN

Finalmente puedo concluir que la implementación de la protección de las rutas, fue un tanto más eficaz por medio de la autenticación de los usuarios por medio de los parámetros necesarios. La creación del Login o del formulario de registro no eran módulos que correspondían a nuestro equipo, sin embargo para poder ejemplificar la funcionalidad de la autenticación de un GUARD, era importante contar con estos módulos para que el frontend fuera una interfaz más asemejada a un contexto de la vida diaria.

La implementación de estas autenticaciones tiene que ver con la extracción de los datos (POST) y posteriormente el análisis y comparación de los mismos (GET), es decir la realización de una consulta que verifique que los datos de los parámetros requeridos son los mismos y que mediante esto se puede o no tener acceso a una ruta específica, en este caso el HOME, también se permite al tener acceso al HOME la eliminación de los usuarios (DELETE), todos estos mismos casos que usamos en prácticas anteriores con POSTMAN.

El añadir un GUARD, especifica la autenticación de estos datos por medio de la consulta y actúa como un condicional que retorna valores booleanos, es decir que si es verdadero se puede dar acceso al usuario, de lo contrario no tendrá acceso al sistema y puede corresponder a su bloqueo inmediato.

Con respecto al back-end muestra la funcionalidad del registro de los usuarios por medio de los campos requeridos para los datos necesarios, por medio del uso de POSTMAN fue posible hacer las pruebas de registro y posterior a esto la administración del servidor para que este se pueda conectar a la base de datos en Mongo DB por medio del Clúster creado en clase.

Una vez que el back fue funcional me pareció adecuado el desarrollo del front con las mismas características que representaran al usuario la interfaz de acceso y de introducción de sus datos. En el front se contiene la protección de las rutas y la autenticación de los usuarios por medio del token que se le asigne de acuerdo a su registro. Es importante mencionar que los registros de las contraseñas en el front contienen la función de encriptación del password, es decir que este no será visible para ninguno de los usuarios que manipulen la aplicación y se incrementa la seguridad de los datos de los usuarios así mismo como la seguridad de la empresa por la autenticación de usuarios y la manipulación de ellos desde el acceso al HOME.

La funcionalidad dio resultado y nos permitió agregar, administrar y eliminar usuarios. Así como asignar permisos y protección a las rutas solo para usuarios registrados en el sistema. Inicialmente teníamos una idea distinta en la formación de la base de datos mediante la creación de una tabla de bloqueo, pero para ello era necesario contar con los módulos correspondientes y complicaba más el desarrollo de la implementación así que decidimos trabajar específicamente en el GUARD y la autenticación.