



PROYECTO MODULO ENCRIPTACION

Integrantes:

Báez Muñoz Ana Karen

Chávez Martínez José Guadalupe

Escobar Díaz Ulises Iván

Ruiz Tique Erick Alberto

EQUIPO DINAMITA - ENCRIPTACION

Materia : (OPT_2_RSD) Base de datos para dispositivos móviles

Profesora: Griselda Cortes Barrera

TECNOLOGICO DE ESTUDIOS SUPERIORES DE ECATEPEC
DIVISION DE INGENIERIA EN SISTEMAS COMPUTACIONALES

INDICE

Tabla de contenido

Resumen	2
Requerimientos	2
Algoritmo AES.....	2
Librería CryptoJS	2
Bootstrap	3
Problemática	3
Objetivos generales	4
Objetivo Especifico	4
Diseño	4
Primeras pruebas.....	12
Creacion de servidor MongoDB	13
Definir esquema del usuario.....	13
Encriptar contraseñas.....	14
Comparación de contraseñas encriptadas.....	14
Almacenar usuarios	15
Instalar y configurar Passport.js.....	16
De serializar usuarios y autenticación local	17
Instalar bodyParser a Express	17
Crear controlador usuario	18
Método login y logout.....	18
Verificación con Postman	19
Creación de BD, Colecciones, Vistas	19
Diseño de la base de datos	20
Importación de la información	21
Vistas (Consultas)	22
Frontend (Angular, BootStrap, NodeJs)	23
Inicio de sesión.....	23
Registro	24
Perfil de usuario	26
Proyecto en GitHub	27

Resumen

- Encriptar es el proceso por el cual se cifra un texto usando una clave, esta clave es un código de signos que se interpretan según determinadas reglas para que no pueda ser entendido por nadie.
- Desencriptar sería el proceso de transformar el mensaje encriptado a texto legible usando la clave generada al encriptar el mensaje, sin esta clave el mensaje sería imposible de volverlo a transformar a texto normal.

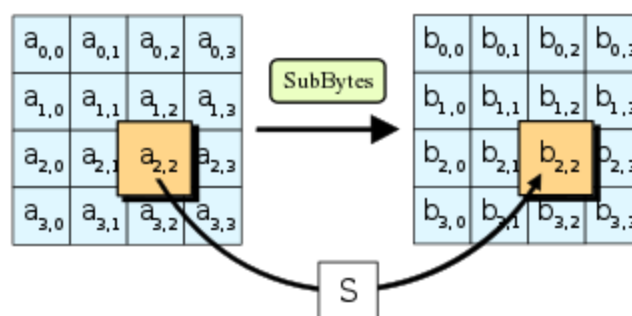
Requerimientos

Para desarrollar este proyecto usaremos unas herramientas determinadas:

Algoritmo AES

- Usaremos un algoritmo llamado AES – este algoritmo es uno de los más usados para esta función.

Advanced Encryption Standard (AES) es uno de los algoritmos de cifrado más utilizados y seguros actualmente disponibles. Es de acceso público, y es el cifrado que la NSA utiliza para asegurar documentos con la clasificación "top secret".



Librería CryptoJS

CryptoJS es una colección creciente de algoritmos criptográficos estándar y seguros implementados en JavaScript utilizando las mejores prácticas y patrones. Son rápidos y tienen una interfaz simple y consistente.

- Usaremos CryptoJs es una librería de algoritmos utilizados en criptografía y escritos en javascript.



Bootstrap

Es un framework de interfaz de usuario, de código abierto, creado para un desarrollo web más rápido y sencillo, su objetivo principal es crear sitios responsive. Permite que la interfaz de usuario de un sitio web funcione de manera óptima en todos los tamaños de pantalla, ya sea en teléfonos de pantalla pequeña o en dispositivos de escritorio de pantalla grande.



Problemática

Podemos considerarlo como el proceso por el cual la información legible se transforma, mediante un algoritmo, en ilegible, y se necesita de una “llave” especial para decodificarla. Este mecanismo nos permite aislar nuestra información de extraños y minimizar las consecuencias indeseadas que ellos pueden generar. El cifrado puede ser usado para proteger todo tipo de archivos; desde mails, información digital, documentos, videos, fotos, claves bancarias y personales, mensajes de texto, hasta el disco duro de una computadora.

Dicho esto, entonces, podemos pensar que cifrar o encriptar nuestra información no resultaría tan difícil, sobre todo si se encuentra en un disco duro al que solo uno tiene acceso. Sin embargo, y a pesar de lo común que se ha vuelto esto hoy en día, no existe en los usuarios individuales una verdadera conciencia de la importancia que reviste el cifrado como método de prevención y resguardo. Peor aún, hasta en el mundo corporativo, muchas empresas desestiman la importancia de asignar recursos para la implementación de verdaderas políticas de seguridad de la información.

Objetivos generales

El cifrado o la encriptación de mensajes sirve para hacer las comunicaciones más seguras, y lo mismo se puede decir a la hora de aplicarlo a Internet. La primera funcionalidad para conseguirlo es la de la confidencialidad de los mensajes, ya que, al no ir al descubierto, cuando tú le envías algo a otra persona, los algoritmos criptográficos de la aplicación ayudan a que no se pueda leer fácilmente si alguien lo intercepta en el camino.

Siendo la confidencialidad la primera de las ventajas que ofrece el cifrado de mensajes, la segunda podríamos decir que es la integridad. El encapsular un mensaje dentro de un sobre de cifrado, lo digo así para hacerse una mejor imagen mental, ayuda que todo lo que haya cifrado se mantenga correcto y completo.

También hay algoritmos criptográficos que proporcionan mecanismos para verificar la identidad de la persona que envía un mensaje. Además, hay métodos de cifrado que también ayudan a vincular un documento o transacción a una persona o sistema de gestión concretas.

Objetivo Especifico

Abarcar varios procedimientos, métodos y enfoques para proteger los datos confidenciales del acceso de terceros y realizar comunicaciones digitales seguras entre dos o más usuarios. La idea básica del cifrado es que los datos se convierten a un formato ilegible utilizando una clave antes de que se produzca un intercambio de información entre el remitente y el destinatario, o se almacenen los datos. El formato convertido se denomina texto cifrado y el formato legible se denomina texto sin formato. Sólo aquellos que conocen la clave (código) correcta para el algoritmo de cifrado tienen acceso al texto sin formato y pueden codificarlo en su forma original. Por lo tanto, el código debe mantenerse secreto o distribuirse de manera especial para que los datos puedan transmitirse o almacenarse de forma segura. El cifrado forma parte de la criptografía en lo que se refiere a la implementación técnica y a la seguridad de los diferentes métodos de cifrado.

Diseño

Los pasos a seguir son los siguientes:

- Abrimos nuestro cmd y creamos el proyecto a través del comando «ng new nombre-proyecto».
- Luego vamos instalamos las dependencias que usar nuestro proyecto:
- npm install crypto-js --save
- npm install bootstrap --save
- Iniciamos visual code atraves del cmd «Code .»

-Iniciamos el servidor para ver que la aplicación corre perfectamente «ng serve»

```
C:\Users\colosus\angular>ng new
? What name would you like to use for the new workspace and initial project? encriptar
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
```

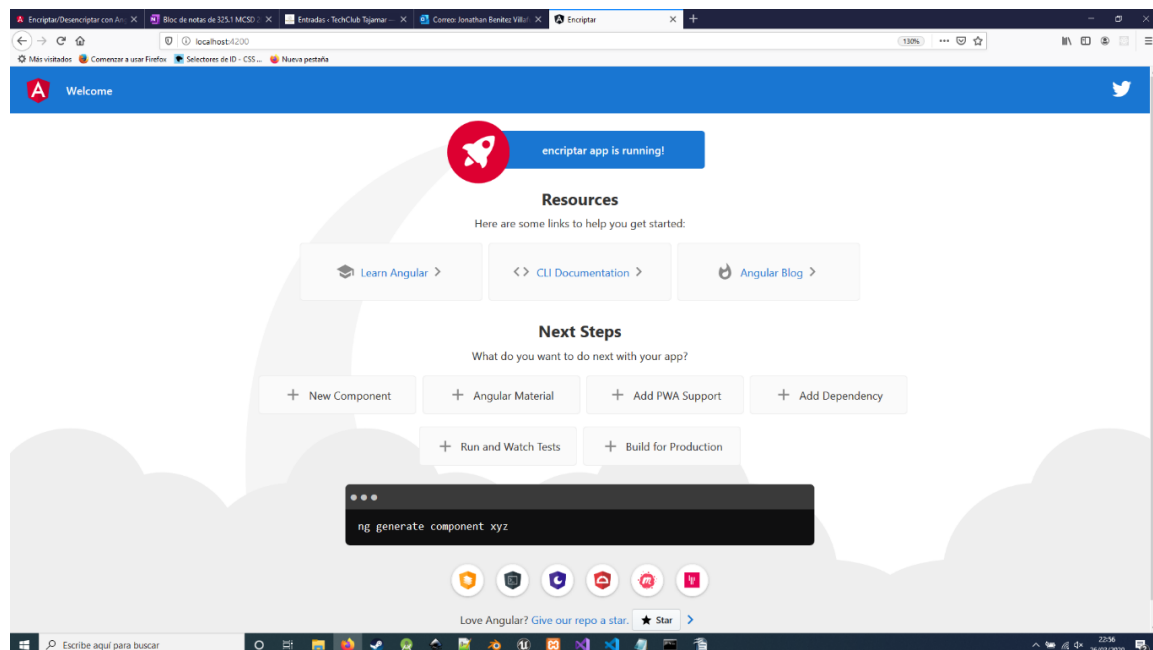
```
C:\Users\colosus\angular\encriptar>npm install crypto-js --save
```

```
C:\Users\colosus\angular\encriptar>npm install bootstrap -save
```

```
C:\Users\colosus\angular\encriptar>ng serve
? Would you like to share anonymous usage data about this project with the Angular Team at
Google under Google's Privacy Policy at https://policies.google.com/privacy? For more
details and how to change this setting, see http://angular.io/analytics. No
0% compiling
Compiling @angular/core : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
chunk {main} main.js, main.js.map (main) 57.7 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 9.71 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 2.7 MB [initial] [rendered]
Date: 2020-03-26T21:42:39.683Z - Hash: 05f36bdcc153b0d7f1cb - Time: 15478ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.

Date: 2020-03-26T21:42:40.894Z - Hash: 05f36bdcc153b0d7f1cb
5 unchanged chunks
Time: 767ms
: Compiled successfully.
```

Copiamos el local host en el navegador y nos abrirá el proyecto, si todo va bien veríamos esto:



Ahora agregamos los módulos en el **angular.json**

```
"styles": [  
  "src/styles.css",  
  "node_modules/bootstrap/dist/css/bootstrap.min.css"],  
  
"scripts": [  
  "node_modules/crypto-js.js",  
  "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"  
]
```

Con esto tenemos el proyecto configurado para usar los módulos agregados.

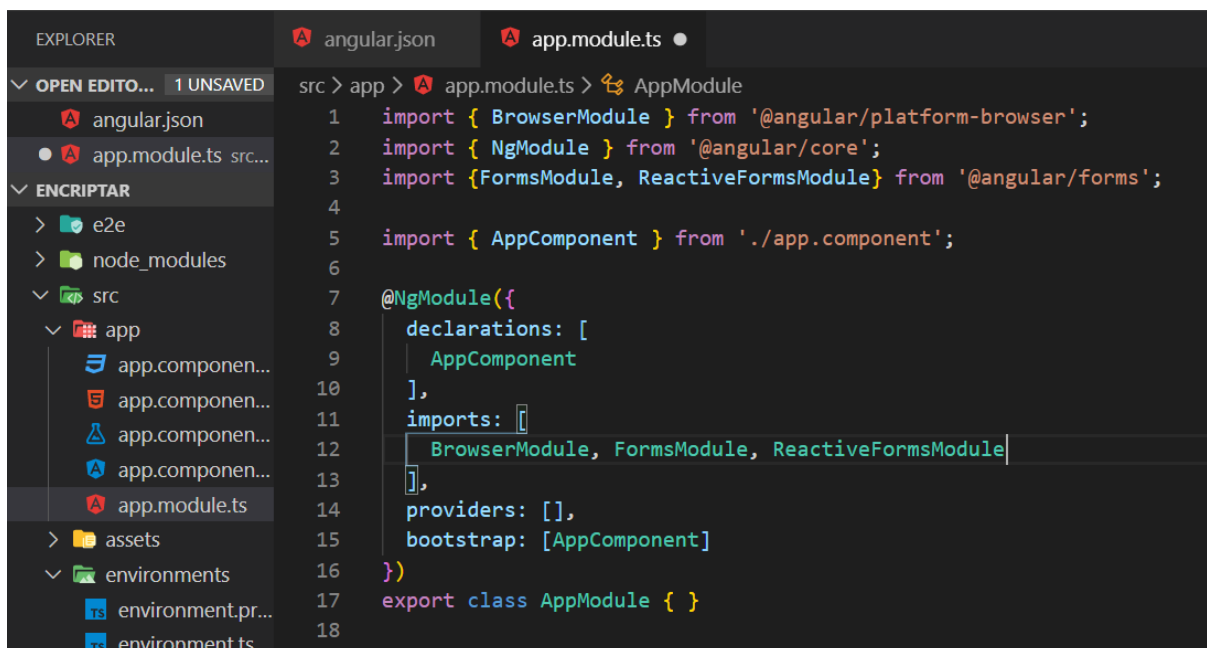
Ahora vamos a importar unas herramientas en el `app.module.ts` para poder usar formularios en nuestra app.

Hacemos un import de :

```
import {FormsModule, ReactiveFormsModule} from '@angular/forms';
```

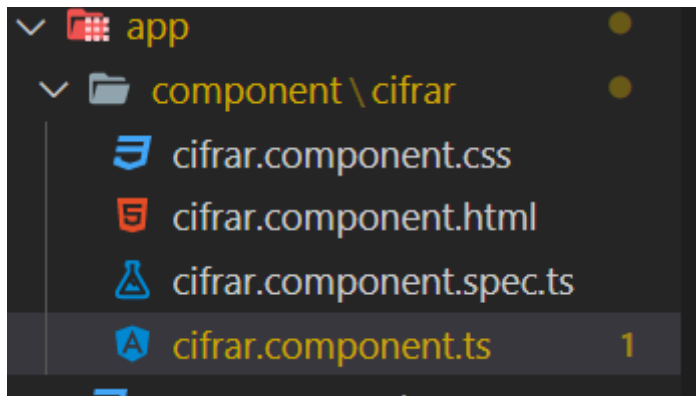
y tambien lo agregamos en el imports de `@NgModule`:

```
imports: [  
  BrowserModule, FormsModule, ReactiveFormsModule  
],
```



```
EXPLORER  
OPEN EDITOR... 1 UNSAVED  
angular.json  
app.module.ts src...  
ENCRIPTAR  
e2e  
node_modules  
src  
  app  
    app.component...  
    app.component...  
    app.component...  
    app.component...  
    app.module.ts  
  assets  
  environments  
    environment.pr...  
    environment.ts  
src > app > app.module.ts > AppModule  
1 import { BrowserModule } from '@angular/platform-browser';  
2 import { NgModule } from '@angular/core';  
3 import {FormsModule, ReactiveFormsModule} from '@angular/forms';  
4  
5 import { AppComponent } from './app.component';  
6  
7 @NgModule({  
8   declarations: [  
9     AppComponent  
10  ],  
11  imports: [  
12    BrowserModule, FormsModule, ReactiveFormsModule  
13  ],  
14  providers: [],  
15  bootstrap: [AppComponent]  
16 })  
17 export class AppModule { }  
18
```

Nos creamos un componente que va contendrá la lógica y la vista de la aplicación



Una vez creado el componente vamos a la vista del componente «cifrar.component.html»

y creamos un formulario que contendrá 2 partes:

La primera parte es la de cifrado que contendrá las siguientes partes:

- Una caja para el texto que vamos a cifrar
- Una caja para meter la clave
- Un textarea para mostrar el texto ya cifrado
- Un botón para llamar a la función de convertirTexto pasándole un valor «encriptar»

La segunda parte es la de descifrado y contendrá las siguientes partes

- Una caja para meter el texto que queremos descifrar
- Una caja para meter la clave usada en el cifrado y que usaremos en esta caja para descifrar el mensaje
- Un textarea que nos mostrara el mensaje descifrado
- Un botón para llamar a la función convertirTexto pasándole un valor «desencriptar»


```

<div>
  <div class="row">
    <div class="col-sm-6">
      <h1 style="text-align: center;">
        | Encriptar
      </h1>
      <br>
      <div class="form-group">
        <label>Texto </label>
        <input type="text" class="form-control" placeholder="Ingresa texto a encriptar" [(ngModel)]="encTexto">
      </div>

      <div class="form-group">
        <label for="txtPassword">Clave</label>
        <input type="password" class="form-control" placeholder="Ingresa llave de encriptación" [(ngModel)]="encPass">
      </div>
      <textarea class="form-control" readonly rows="3">{{textoEncriptado}}</textarea>
      <br>
      <button type="button" class="btn btn-success float-right" (click)="convertirTexto('encriptar')>Encriptar</button>
    </div>
    <div class="col-sm-6">
      <h1 style="text-align: center;">
        | Desencriptar
      </h1>
      <br>
      <div class="form-group">
        <label>Texto encriptado</label>
        <input type="text" class="form-control" placeholder="Ingrese el texto que quieres desencriptar" [(ngModel)]="desTexto">
      </div>

      <div class="form-group">
        <label for="txtPassword">Clave</label>
        <input type="password" class="form-control" placeholder="Ingrese la llave para desencriptar" [(ngModel)]="desPass">
      </div>
      <textarea class="form-control" readonly rows="3">{{textoDesencriptado}}</textarea>
      <br>
      <button type="button" class="btn btn-success float-right" (click)="convertirTexto('desencriptar')>Desencriptar</button>
    </div>
  </div>
</div>

```

Ahora vamos a ver en detalle que funcionalidad tiene cada etiqueta

En esta etiqueta definimos la clase que es de tipo formulario y **ngModel** que capturara el contenido que hay en la caja, en este caso sería el texto que vamos a introducir para cifrar:

```
<input type="text" class="form-control" placeholder="Ingresa texto a encriptar" [(ngModel)]="encTexto">
```

Esta es la misma que la anterior descrita y servirá para capturar la clave que usaremos para cifrar el texto:

```
<input type="password" class="form-control" placeholder="Ingresa llave de encriptación" [(ngModel)]="encPass">
```

Aquí usaremos un **textarea** para mostrar el texto cifrado, con **{{textoEncriptado}}** pintamos el resultado de cifrar el texto:

```
<textarea class="form-control" readonly rows="3">{{textoEncriptado}}</textarea>
```

Por ultimo tenemos el botón que contiene el evento **(click)="convertirTexto('encriptar')**, esto lo que hace es llamar a la función **«convertirTexto»** pasándole un parametro ('encriptar'):

```
<button type="button" class="btn btn-success float-right" (click)="convertirTexto('encriptar')>Encriptar</button>
```

Parte Descriptar:

En esta etiqueta vamos a introducir el resultado de encriptar el texto y lo capturamos con `ngModel`

```
<input type=»text» class=»form-control» placeholder=»Ingresa el texto que quieres desenscriptar» [(ngModel)]=»destexto»>
```

En esta etiqueta vamos a introducir la clave que hemos usado para cifrar y que ahora usaremos para descifrar el texto cifrado, capturamos la clave con `ngModel`

```
<input type=»password» class=»form-control» placeholder=»Ingresa la llave para desenscriptar» [(ngModel)]=»desPas s»>
```

En esta etiqueta se mostrare el texto ya descifrado y lo pintaremos con `{{textoDesenscriptado}}`

```
<textarea class=»form-control» readonly rows=»3»>{{textoDesenscriptado}}</textarea>
```

Por ultimo tenemos el botón que contiene el evento `(click)=»convertirTexto('desenscriptar')`, esto lo que hace es llamar a la función `«convertirTexto»` pasándole un parámetro que en este caso será el de `('desenscriptar')`

```
<button type=»button» class=»btn btn-success float-right» (click)=»convertirTexto('desenscriptar')»>Desenscriptar</button>
```

El resultado visual de este formulario es el siguiente:

Encriptar	Desenscriptar
<p><small>Texto</small></p> <input style="width: 90%;" type="text" value="Ingresa texto a encriptar"/>	<p><small>Texto encriptado</small></p> <input style="width: 90%;" type="text" value="Ingresa el texto que quieres desenscriptar"/>
<p><small>Clave</small></p> <input style="width: 90%;" type="password" value="Ingresa llave de encriptación"/>	<p><small>Clave</small></p> <input style="width: 90%;" type="password" value="Ingresa la llave para desenscriptar"/>
<div style="background-color: #6c757d; color: white; padding: 5px 15px; border-radius: 5px;">Encriptar</div>	<div style="background-color: #6c757d; color: white; padding: 5px 15px; border-radius: 5px;">Desenscriptar</div>

Ahora vamos a programar la parte lógica, nos iremos al `component.ts` donde este alojado nuestro html.

```
import { Component, OnInit } from '@angular/core';
import * as CryptoJS from 'crypto-js';

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

  encTexto: string;
  destexto: string;
  encPass: string;
  desPass: string;
  textoEncriptado: string;
  textoDesencriptado: string;
  constructor() { }

  convertirTexto(conversion: string) {
    if (conversion === 'encriptar') {
      this.textoEncriptado = CryptoJS.AES.encrypt(this.encTexto.trim(), this.encPass.trim()).toString();
    } else {
      this.textoDesencriptado = CryptoJS.AES.decrypt(this.destexto.trim(), this.desPass.trim()).toString(CryptoJS.enc.Utf8);
    }
  }

  ngOnInit(): void {
    this.convertirTexto();
  }
}
```

Vamos a seguir los pasos:

Primero hacemos un import de **CryptoJs** para realizar los procesos de encriptar y desencriptar el texto

```
import * as CryptoJS from 'crypto-js';
```

Definimos el tipo de datos de los **Ngmodel** que hemos usado para capturar los datos introducidos en las cajas en este caso todo lo que vamos a introducir es texto asi que será de tipo String

```
encTexto: string;
```

```
destexto: string;
```

```
encPass: string;
```

```
desPass: string;
```

```
textoEncriptado: string;
```

```
textoDesencriptado: string;
```

Lo siguiente es las funciones para realizar el proceso de encriptar y desencriptar

```
convertirTexto(conversion: string) {
```

```
  if (conversion === 'encriptar') {
```

```
    this.textoEncriptado = CryptoJS.AES.encrypt(this.encTexto.trim(), this.encPass.trim()).toString();
```

```
  } else {
```

```
    this.textoDesencriptado = CryptoJS.AES.decrypt(this.destexto.trim(), this.desPass.trim()).toString(CryptoJS.enc.Utf8);
```

```
}}
```

Vamos a ver en detalle esta función

La función se llama `ConvertirTexto` que es llamada cuando hacemos el click en el botón, definimos el tipo de dato que le pasamos como parámetro que en este caso es string

```
convertirTexto(conversion: string) {
```

A continuación, creamos un if donde vamos a comparar el parámetro pasado en este caso comparamos que botón hemos pulsado preguntado si conversión es igual a encriptar o desencriptar

```
if (conversion === 'encriptar') {
```

Si es encriptar ejecutara esta línea de código:

Esta línea lo que hace es escoger el texto de la caja que queremos encriptar y la clave que usaremos para encriptar, mediante el método `CryptoJS.AES.encrypt`, usamos el módulo `cryptojs` definimos el algoritmo que usaremos en este caso `Aes` y con `encrypt` procedemos a encriptarlo todo.

```
    this.textoEncriptado = CryptoJS.AES.encrypt(this.enctexto.trim(), this.encPass.trim()).toString();
```

Y si el botón pulsado es el de desencriptar se ejecutará la siguiente línea

La funcionalidad es la misma lo único que cambia es `decrypt` donde cojera el texto encriptado de la caja y la clave usada para encriptar y desencriptara el mensaje

```
    } else {
```

```
        this.textoDesencriptado = CryptoJS.AES.decrypt(this.destexto.trim(), this.dePass.trim()).toString(CryptoJS.enc.Utf8);
```

```
    }
```

Ahora vamos a ver si es funcional:

Metemos un texto

Metemos una clave que usaremos para encriptar y también para desencriptar

Primeras pruebas

Encriptar

Texto

Clave

Ahora vamos a Desencriptar

Metemos en la caja el texto que hemos encriptado

Metemos la clave usada en el cifrado para poder descifrarlo, tiene que ser la misma sino será imposible descifrar el texto

Desencriptar

Texto encriptado

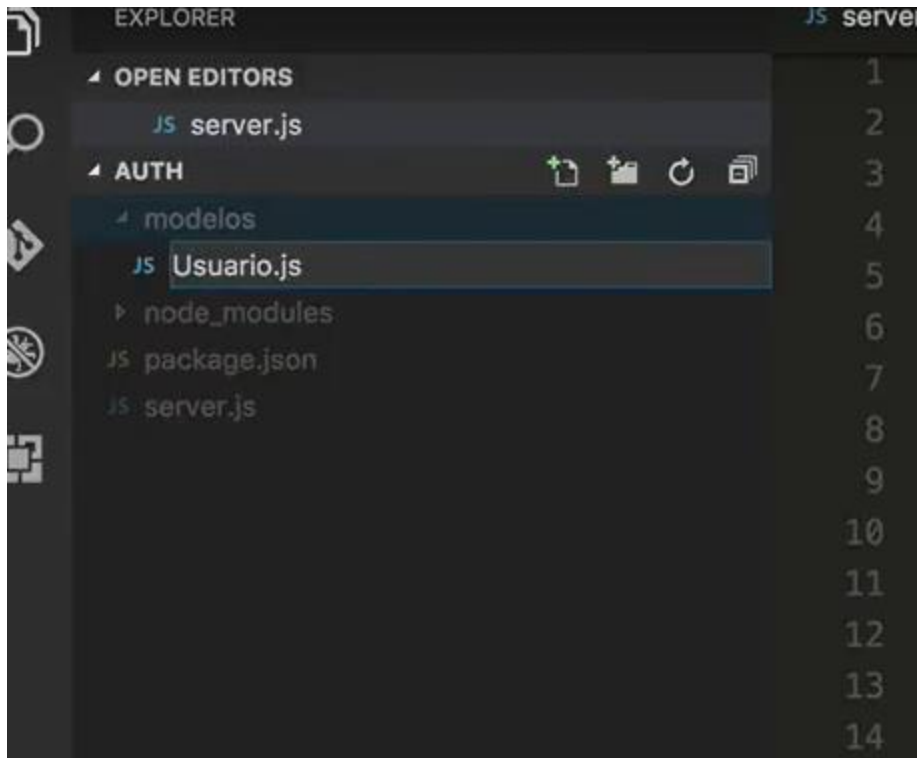
Clave

Creacion de servidor MongoDB

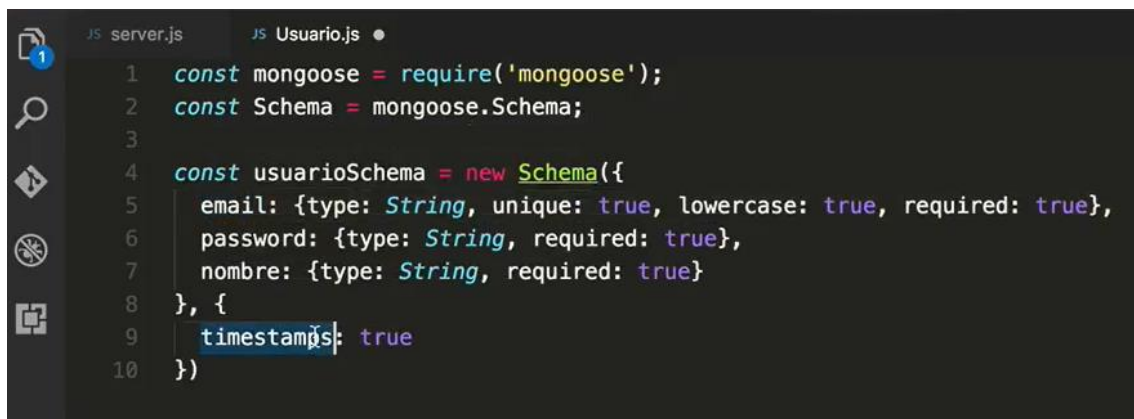
Definir esquema del usuario

Para el proceso de la definicion del usuario debemos crear una carpeta para poner los archivos dentro en este caso la carpeta modelo nos ayudara como contenedor.

Dentro de la carpeta crearemos nuestro archivo de usuario.js el cual es el modelo de nuestro usuario.



Una vez creado nuestro usuario debemos agregarle las propiedades correspondientes a el campo. En este caso ocupamos el email, password y el nombre. Esto para que el usuario tenga registro. Al igual que la propiedad de *timestamps* la cual no es mas que un registro de cuando fue creado el usuario dentro del sistema.



Encriptar contraseñas

La funcionalidad de encriptar las contraseñas nos permite no hacer una base de datos para almacenar las contraseñas de los usuarios y así mejorar la seguridad informática de estas.

De esta manera declaramos una constante la cual nos preguntara si el usuario modifico su contraseña o no y posteriormente se ejecutara el proceso de encriptacion de la informacion ingresada por el usuario.

Son 3 simples condicionales

```
JS server.js JS Usuario.js ●
7   password: {type: String, required: true},
8   nombre: {type: String, required: true}
9   }, {
10  timestamps: true
11  })
12
13  usuarioSchema.pre('save', function(next) {
14    const usuario = this;
15    if (!usuario.isModified('password')) {
16      return next();
17    }
18
19    bcrypt.genSalt(10, (err, salt) => {
20      if (err) {
21        next(err);
22      }
23      bcrypt.hash(usuario.password, salt, null, (err, hash) => {
24        if (err) {
25          next(err);
26        }
27        usuario.password = hash;
28        next();
29      })
30    })
31  })
```

Comparación de contraseñas encriptadas

Hay ciertas situaciones al ingresar las contraseñas de los usuarios en este caso debemos hacer una comparativa de las contraseñas ya que al momento de encriptarlas debemos tener la certeza que sean las mismas así como para verificarlas.

Si no pasamos por este proceso la informacion no podra ser procesada a la base de datos.

```
usuarioSchema.methods.compararPassword = function(password, cb) {
  bcrypt.compare(password, this.password, (err, sonIguales) => {
    if(err) {
      return cb(err);
    }
    cb(null, sonIguales);
  })
}

module.exports = mongoose.model('Usuario', {})
```

Almacenar usuarios

Los procedimientos de almacenamiento debemos de ser acuerdo a los datos y tipo de dato, en este caso debemos utilizar strings para crear los usuarios y las contraseñas.

```
const Usuario = require('./modelos/Usuario');
const u = new Usuario({
  email: 'ricardo@ric.com',
  nombre: 'Ricardo',
  password: '123456'
})

u.save()
  .then(() => {
    console.log('guardado')
  })
  .catch((error) => {
    console.log(error);
  })
```

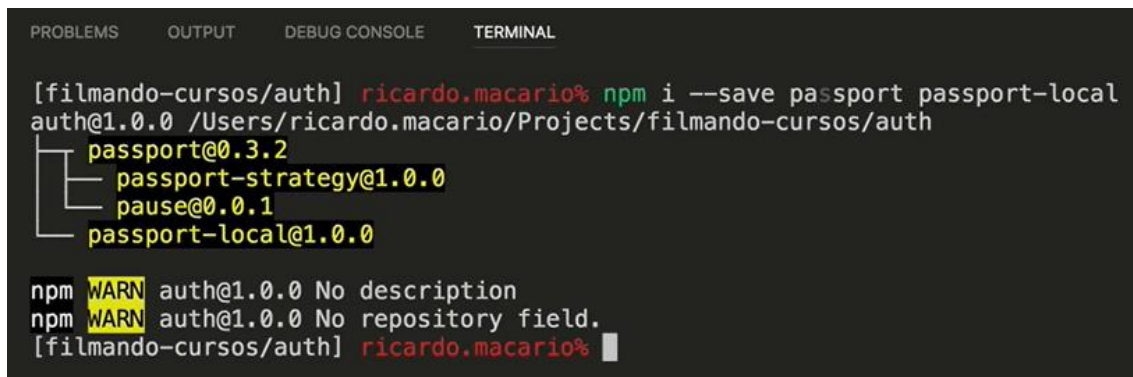
Para esto debemos de comprobar los scripts y ver si se ejecutan bien y cumplen con el almacenamiento.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

[filmando-cursos/auth] ricardo.macario% nodemon server
[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server index.js`
Escuchando en el puerto 3000
guardado
```


Instalar y configurar Passport.js

Instalamos la paquetería de passport.js



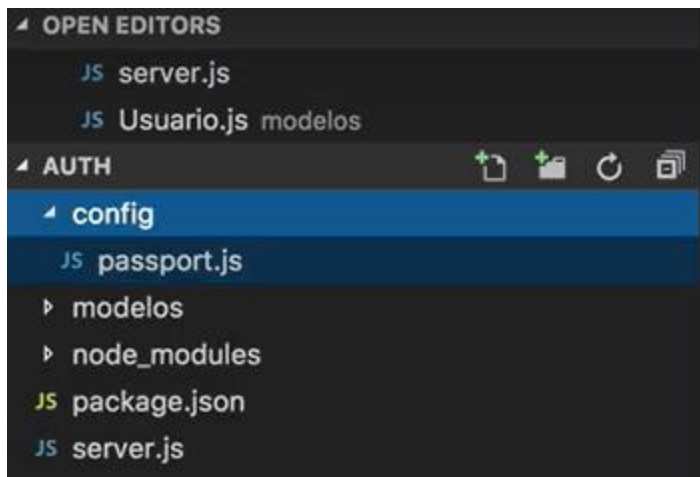
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

[filmando-cursos/auth] ricardo.macario% npm i --save passport passport-local
auth@1.0.0 /Users/ricardo.macario/Projects/filmando-cursos/auth
├── passport@0.3.2
│   ├── passport-strategy@1.0.0
│   └── pause@0.0.1
└── passport-local@1.0.0

npm WARN auth@1.0.0 No description
npm WARN auth@1.0.0 No repository field.
[filmando-cursos/auth] ricardo.macario%
```

Posterior a eso creamos una carpeta donde se guardaran los archivos de la instalacion.

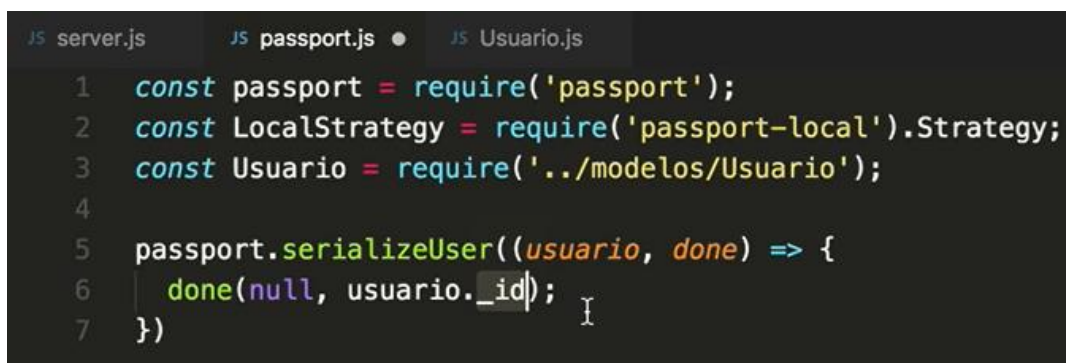
Dentro de esta carpeta guardamos el archivo.



```
OPEN EDITORS
JS server.js
JS Usuario.js modelos

AUTH
└─ config
    JS passport.js
  modelos
  node_modules
JS package.json
JS server.js
```

Por ultimo debemos de configurar el archivo que instalamos.



```
JS server.js  JS passport.js  JS Usuario.js

1  const passport = require('passport');
2  const LocalStrategy = require('passport-local').Strategy;
3  const Usuario = require('../modelos/Usuario');
4
5  passport.serializeUser((usuario, done) => {
6    done(null, usuario._id);
7  })
```

De serializar usuarios y autenticación local

Para que no haya confusión en cuanto a la serialización de la cantidad de usuarios registrados ocupamos una función para que no se empalmen los números de serial de los usuarios.

Así será más fácil localizar cualquier registro en la base de datos. Que al igual que la serialización ocupamos una función la cual nos busque el usuario que queramos modificar.

```
9 passport.deserializeUser((id, done) => {
10   Usuario.findById(id, (err, usuario) => {
11     done(err, usuario);
12   })
13 })
14
15 passport.use(new LocalStrategy(
16   {usernameField: 'email'},
17   (email, password, done) => {
18     Usuario.findOne({email}, (err, usuario) => {
19       if (!usuario) {
20         return done(null, false, {message: `Este email: ${email} no esta registrado`});
21       } else {
22         usuario.compararPassword(password, (err, sonIguales) => {
23           if (sonIguales) {
24             return done(null, usuario);
25           } else {
26             return done(null, false, {message: 'La contraseña no es válida'});
27           }
28         })
29       }
30     })
31   })
32 ))
```

Instalar bodyParser a Express

Esta función sirve para cambiar la sintaxis del código de manera express

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

[filmando-cursos/auth] ricardo.macario% npm i --save body-parser
auth@1.0.0 /Users/ricardo.macario/Projects/filmando-cursos/auth
└─ body-parser@1.16.1

npm WARN auth@1.0.0 No description
npm WARN auth@1.0.0 No repository field.
[filmando-cursos/auth] ricardo.macario%
```

Crear controlador usuario

Necesitamos crear un script llamado usuario para poder controlar todo lo que el usuario requiere o modifica.

```
JS server.js JS usuario.js ●
1  const passport = require('passport');
2  const Usuario = require('../modelos/Usuario');
3
4  exports.postSignup = (req, res, next) => {
5    const nuevoUsuario = new Usuario({
6      email: req.body.email,
7      nombre: req.body.nombre,
8      password: req.body.password
9    });
10
11    Usuario.findOne({email: req.body.email}, (err, usuarioExistente) => {
12      if (usuarioExistente) {
13        return res.status(400).send('Ya ese email esta registrado');
14      }
15      nuevoUsuario.save((err) => {
16        if (err) {
17          next(err);
18        }
19        req.login(nuevoUsuario, (err) => {
20          if (err) {
21            next(err);
22          }
23        })
24      })
25    })
26  })
27 }
```

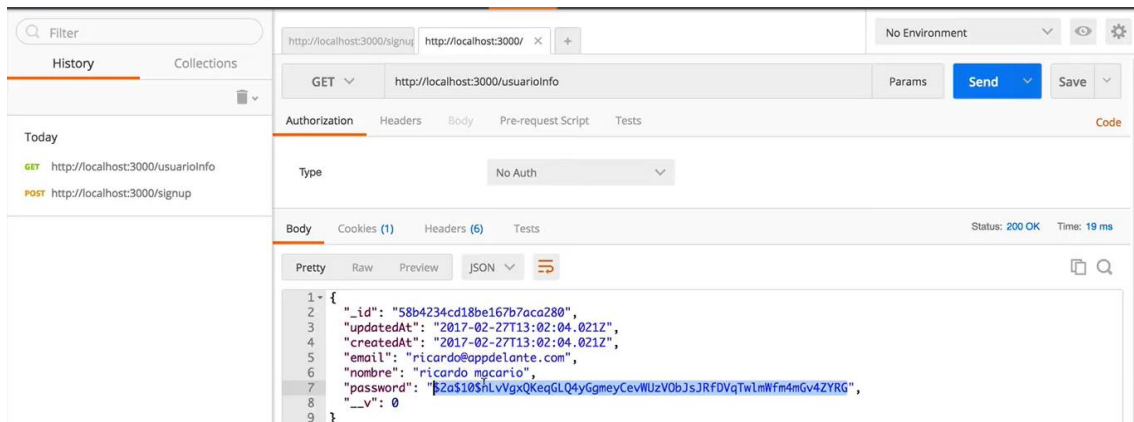
Método login y logout

Ocupamos la librería de passport para poder crear el metodo de logeo de esta manera el usuario tendra una interfaz mas amigable y entendible para poder crear sus registros.

```
exports.postLogin = (req, res, next) => {
  passport.authenticate('local', (err, usuario, info) => {
    if (err) {
      next(err);
    }
    if (!usuario) {
      return res.status(400).send('Email o contraseña no válidos');
    }
    req.login(usuario, (err) => {
      if (err) {
        next(err);
      }
      res.send('Login exitoso');
    })
  })(req, res, next);
}
```

Verificación con Postman

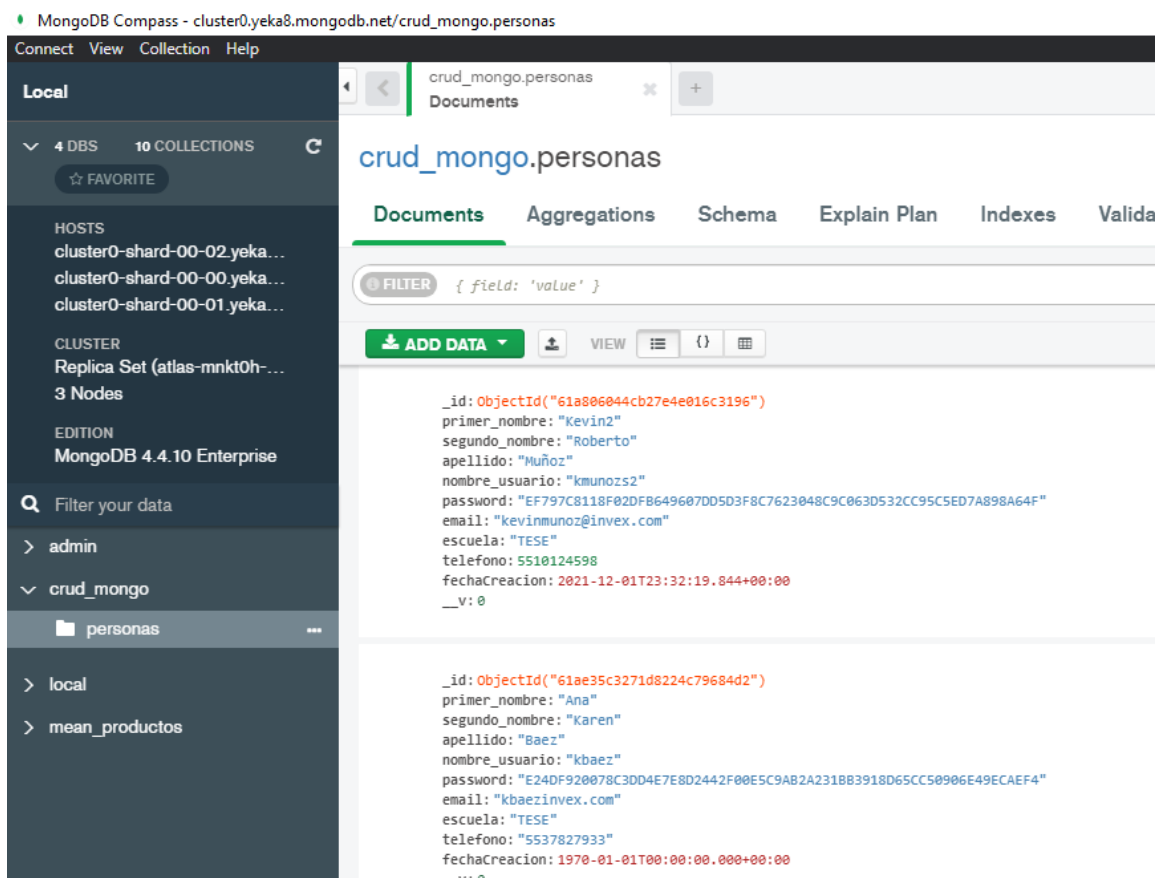
Con la ayuda de postman podemos ver el proceso de lo que fuimos generando tanto en la creacion de los usuarios asi como su almacenamiento de igual manera podemos observar la contraseña encriptada.



Mediante la direccion IP en nuestro localhost podemos ver todo nuestro proceso de los scrips como de la base de datos.

Creación de BD, Colecciones, Vistas

Primeramente vamos a consultar la base donde se alojara la informacion de registro y el campo que vamos a usar es "password "



Vamos a exportar la información en un CSV para poder visualizar la información de manera más certera

Export Collection crud_mongo.personas

Select Fields ⓘ + ADD FIELD

<input checked="" type="checkbox"/>	6	fechaCreacion
<input checked="" type="checkbox"/>	7	nombre_usuario
<input checked="" type="checkbox"/>	8	password
<input checked="" type="checkbox"/>	9	primer_nombre
<input checked="" type="checkbox"/>	10	segundo_nombre
<input checked="" type="checkbox"/>	11	telefono
	12	Add field

< BACK CANCEL SELECT OUTPUT

Vamos a separar los datos que vamos a usar en este caso dejaremos un CSV con el nombre del registro, asignando un ID

	A	B	C	D	E	F	G	H	
1	Id_P	primer_nombre	segundo_nombre	apellido	nombre_usuario	email	escuela	telefono	
2	1	Kevin	Roberto	Muñoz	kmunozs2	kevinmuno	TESE	5510124598	
3	2	Ana	Karen	Baez	kbaez	kbaezinvex	TESE	5537827933	
4									

Aquí vamos a separar el password que en este caso ya está encriptado manteniendo el ID para la identificación

	A	B	C	D	E	F	G	
1	Id_P	password						
2	1	EF797C8118F02DFB649607DD5D3F8C7623048C9C063D532CC95C5ED7A898A64F						
3	2	E24DF920078C3DD4E7E8D2442F00E5C9AB2A231BB3918D65CC50906E49ECAEF4						
4								

Diseño de la base de datos

Vamos a crear nuestra base con el nombre encriptación referente al módulo que se está trabajando

Database Name	Storage Size	Collections	Indexes	
Direcciones	905.2KB	4	4	
Encriptacion	8.2KB	2	2	

Se van a crear las tablas (Collection) con el nombre Registros y Password

Collection Name ^	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
Password	0	-	0.0 B	1	4.1 KB	
Registro	0	-	0.0 B	1	4.1 KB	

Importación de la información

Insertaremos los datos con su debida separación y de tipo String

Options

Select delimiter **SEMICOLON**

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_P String	<input checked="" type="checkbox"/> primer_nombre String	<input checked="" type="checkbox"/> segundo_nombre String	<input checked="" type="checkbox"/> apellido String	<input checked="" type="checkbox"/> nombre_usua String
1	1	Kevin2	Roberto	Muñoz	kmunozs2
2	2	Ana	Karen	Baez	kbaez

CANCEL

IMPORT

Aquí ya vemos los datos que se agregaron

Encriptacion.Registro

DOCUMENTS 2 TOTAL SIZE 411B AVG. SIZE 206B INDEXES 1 TOTAL SIZE 4.1KB AVG. SIZE 4.1KB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } **OPTIONS** **FIND** **RESET** **REFRESH**

ADD DATA **VIEW** **REFRESH**

Displaying documents 1 - 2 of 2

```
{
  "_id": ObjectId("61b3ca82b0e77da6a4dffc3a"),
  "Id_P": "1",
  "primer_nombre": "Kevin2",
  "segundo_nombre": "Roberto",
  "apellido": "Muñoz",
  "nombre_usuario": "kmunozs2",
  "email": "kevinmunoz@invex.com",
  "escuela": "TESE",
  "telefono": "5510124598"
}
```

```
{
  "_id": ObjectId("61b3ca82b0e77da6a4dffc3b"),
  "Id_P": "2",
  "primer_nombre": "Ana",
  "segundo_nombre": "Karen",
  "apellido": "Baez",
  "nombre_usuario": "kbaez",
  "email": "kbaez@invex.com",
  "escuela": "TESE",
  "telefono": "5537827933"
}
```

Ahora vamos a insertar los datos de password

Options

Select delimiter **SEMICOLON**

☒ Ignore empty strings

☐ Stop on errors

Specify Fields and Types

	<input checked="" type="checkbox"/> Id_P String	<input checked="" type="checkbox"/> password String
1	1	EF797C8118F02DFB649607DD5D3F8C7623048C9C063D532CC95C5ED7A898A64F
2	2	E24DF920078C3DD4E7E8D2442F00E5C9AB2A231BB3918D65CC50906E49ECAEF4

CANCEL

IMPORT

Ahora visualizaremos los password con sus respectivos ID

Encrptacion.Password

DOCUMENTS 2 TOTAL SIZE 226B AVG. SIZE 113B

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } **OPTIONS**

ADD DATA **VIEW** **LIST** **JSON** **GRID** **TABLE**

Displaying documents 1 -

```

_id: ObjectId("61b3caeb0e77da6a4dffc3e")
Id_P: "1"
password: "EF797C8118F02DFB649607DD5D3F8C7623048C9C063D532CC95C5ED7A898A64F"

_id: ObjectId("61b3caeb0e77da6a4dffc3f")
Id_P: "2"
password: "E24DF920078C3DD4E7E8D2442F00E5C9AB2A231BB3918D65CC50906E49ECAEF4"

```

Vistas (Consultas)

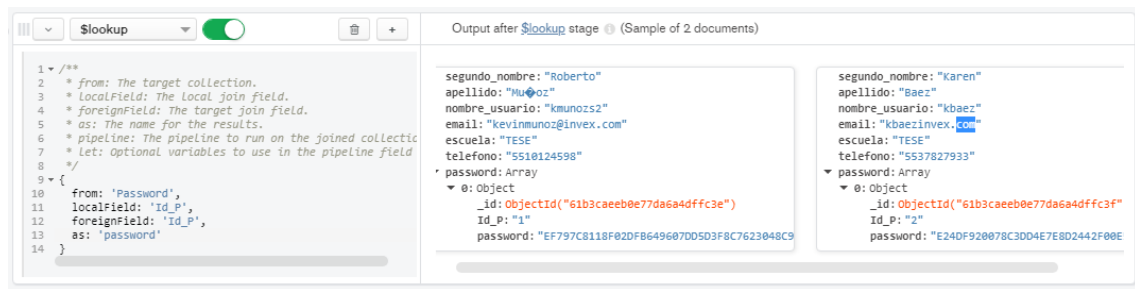
Ahora vamos a realizar todas las vistas del usuario para relacionarlas y poder hacer uso de lookup y llaves primarias para consultar ciertos datos

Registro-Password

```

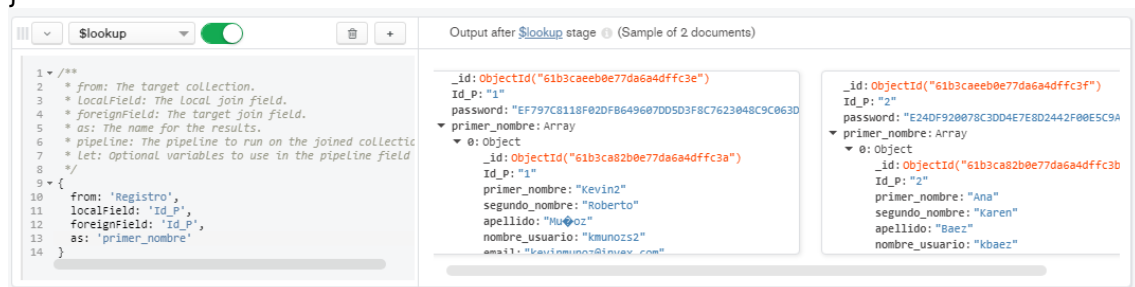
{
  from: 'Password',
  localField: 'Id_P',
  foreignField: 'Id_P',
  as: 'password'
}

```

Password-Registro

```
{
  from: 'Registro',
  localField: 'Id_P',
  foreignField: 'Id_P',
  as: 'primer_nombre'
}
```



Password-Registro

```
{
  from: 'Registro',
  localField: 'Id_P',
  foreignField: 'Id_P',
  as: 'email'
}
```



Frontend (Angular, BootStrap, NodeJs)

Inicio de sesión

```
<div>
  
</div>
<form #signInForm="ngForm" (ngSubmit)="signInForm.valid &&
onSubmit(signInForm)">
  <input type="text" name="email" #email="ngModel"
[(ngModel)]="model.email" placeholder="Email" [pattern]="emailRegex"
```



```

required
    [ngClass]="{'invalid-textbox' :signInForm.submitted &&
!email.valid }">
    <div *ngIf="signInForm.submitted && email.errors?.pattern">
        <label class="validation-message">Correo invalido.</label>
    </div>
    <input type="password" name="password" #password="ngModel"
[(ngModel)]="model.password" placeholder="Password" required
minlength="4"
    [ngClass]="{'invalid-textbox' :signInForm.submitted &&
!password.valid }">
    <div *ngIf="signInForm.submitted && password.errors?.minlength">
        <label class="validation-message">Minimo 4 caracteres.</label>
    </div>
    <input type="submit" value="Iniciar Sesion">
</form>
<!-- Error message -->
<div class="alert" *ngIf="serverErrorMessages">
    {{serverErrorMessages}}
</div>

```

INICIAR SESION REGISTRARSE

DE ESTUDIOS
TECNOLÓGICO SUPERIORES
DE ECATEPEC

Email

Password

INICIAR SESION

Registro

```

<div class="wrapper">
    <div id="formContent">
        <h2 class="underlineHover" routerLink="/login"
routerLinkActive="active"> Iniciar Sesion </h2>
        <h2 class="underlineHover" routerLink="/signup"


```

```
routerLinkActive="active"> Registrarse </h2>
  <router-outlet></router-outlet>
</div>
</div>
<div>
  
</div>
<form #signUpForm="ngForm" (ngSubmit)="signUpForm.valid &&
onSubmit(signUpForm)">
  <input type="text" #fullName="ngModel"
[(ngModel)]="userService.selectedUser.fullName" name="fullName"
placeholder="Nombre completo"
required [ngClass]="{'invalid-textbox' :signUpForm.submitted &&
!fullName.valid }">
  <div *ngIf="signUpForm.submitted && !fullName.valid">
    <label class="validation-message">Este campo es
obligatorio.</label>
  </div>
  <input type="text" #email="ngModel"
[(ngModel)]="userService.selectedUser.email" name="email"
placeholder="Email"
required [pattern]="emailRegex" [ngClass]="{'invalid-textbox'
:signUpForm.submitted && !email.valid }">
  <div *ngIf="signUpForm.submitted && email.errors">
    <label *ngIf="email.errors.required" class="validation-
message">Este campo es obligatorio.</label>
    <label *ngIf="email.errors.pattern" class="validation-
message">Correo invalido.</label>
  </div>
  <input type="password" #password="ngModel"
[(ngModel)]="userService.selectedUser.password" name="password"
placeholder="Password"
minlength="4" required [ngClass]="{'invalid-textbox'
:signUpForm.submitted && !password.valid }">
  <div *ngIf="signUpForm.submitted && password.errors">
    <label *ngIf="password.errors.required" class="validation-
message">Este campo es obligatorio.</label>
    <label *ngIf="password.errors.minlength" class="validation-
message">Introduce por lo menos 4 caracteres.</label>
  </div>
  <input type="submit" value="Registrarse">
</form>

<!-- Success message -->
<div class="success" *ngIf="showSucessMessage">
  Registro guardado
</div>

<!-- Error message -->
<div class="alert" *ngIf="serverErrorMessage">
  {{serverErrorMessage}}
</div>
```

INICIAR SESION REGISTRARSE



Nombre completo

Email

Password

REGISTRARSE

Perfil de usuario

```
<table *ngIf="userDetails" class="table-fill">
  <thead>
    <tr>
      <th colspan="2" class="text-center">Perfil de usuario</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Nombre</td>
      <td>{{userDetails.fullName.split(' ')[0]}}</td>
    </tr>
    <tr>
      <td>Apellido</td>
      <td>{{userDetails.fullName.split(' ')[1]}}</td>
    </tr>
    <tr>
      <td>Email</td>
      <td>{{userDetails.email}}</td>
    </tr>
    <tr>
      <td colspan="2" class="text-center">
```

```
                <input type="button" (click)="onLogout()" value="Cerrar
Sesion" />
            </td>
        </tr>
    </tbody>
</table>
```

Perfil de usuario	
Nombre	karen
Apellido	baez
Email	karenbm@tese.com
<div>CERRAR SESION</div>	

Proyecto en GitHub

<https://github.com/sandyCortes/front-generate-reports/tree/Dinamita>

Encriptación Sha256 **PROYECTO FUNCIONANDO**

```
_id: ObjectId("61a806044cb27e4e016c3196")
primer_nombre: "Kevin2"
segundo_nombre: "Roberto"
apellido: "Muñoz"
nombre_usuario: "kmunozs2"
password: "EF797C8118F02DFB649607DD5D3F8C7623048C9C063D532CC95C5ED7A898A64F"
email: "kevinmunoz@invex.com"
escuela: "TESE"
telefono: 5510124598
fechaCreacion: 2021-12-01T23:32:19.844+00:00
__v: 0
```

Página 28 | 28