



El futuro digital
es de todos

MinTIC

«Misión
TIC 2022»

Formato de Informe de Seguimiento



Universidad de Caldas



Materia: Desarrollo de Aplicaciones Web – Ciclo 4A

Grupo: 12

Docente: Andrés Felipe Escallon Portilla.

Tutora: Yurley Katherine Echeverría Leal.

Sprint 3: Tienda Virtual de Tecnología.

Formato de Informe de Seguimiento

Equipo 2. Techno Team

Integrantes (Nombre completo)	Cédula	Rol	Nivel de participación (Alto, Medio, Bajo, Retirado)
BARRAGAN PLAZAS CARLOS EDUARDO	79536048	Líder de equipo. Administrador de Configuración.	Alto.
BARRAZA RIOS CRISTIAN	1045749373	Tester.	Alto.
BASTIDAS LAME LAURA MARCELA	1061774975	Diseñador UI.	Alto.
CAICEDO BELTRAN JONATHAN	1030579031	Diseñador de Software.	Alto.
CESPEDES RAMIREZ JOSE GIOVANNI	79854497	No se ha presentado.	No se ha presentado.

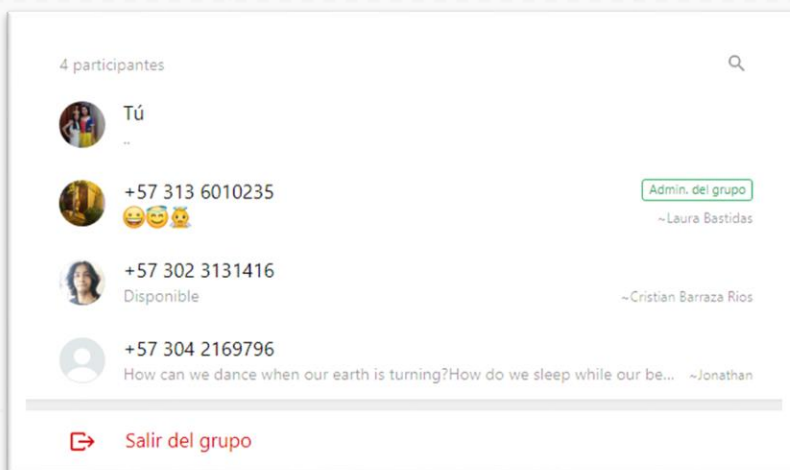


El futuro digital
es de todos

MinTIC

«Mision
TIC 2022»

Se realiza la reunión con el equipo, nos estamos comunicando por WhatsApp; donde se creó el grupo:



El señor José Giovanni Céspedes Ramírez, no se ha presentado.



Universidad de Caldas



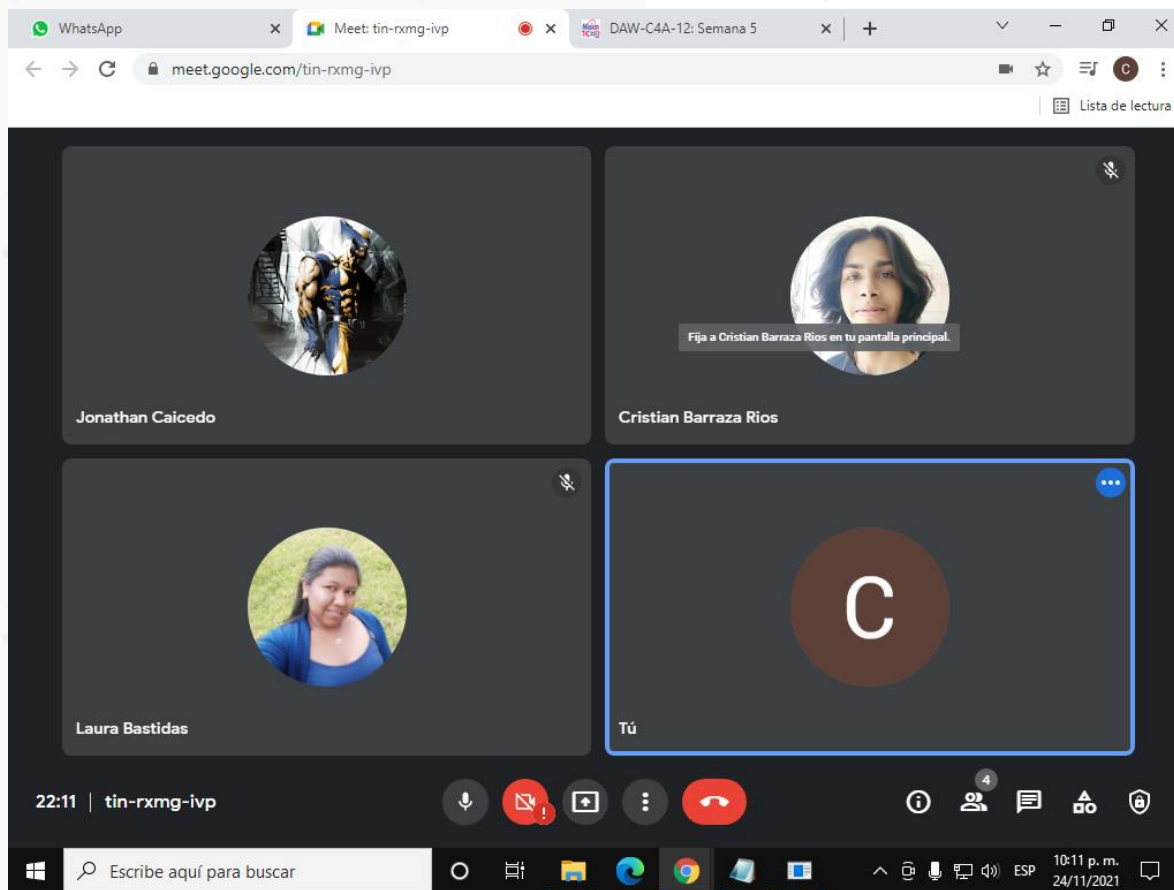
El futuro digital
es de todos

MinTIC

«Misión
TIC 2022»

Se anexa imagen de la reunión por Meet.

Se da inicio a la reunión previamente programada, donde hablamos sobre el trabajo entregado correspondiente al Sprint2 y empezamos a revisar el trabajo correspondiente al Sprint3.



Universidad de Caldas

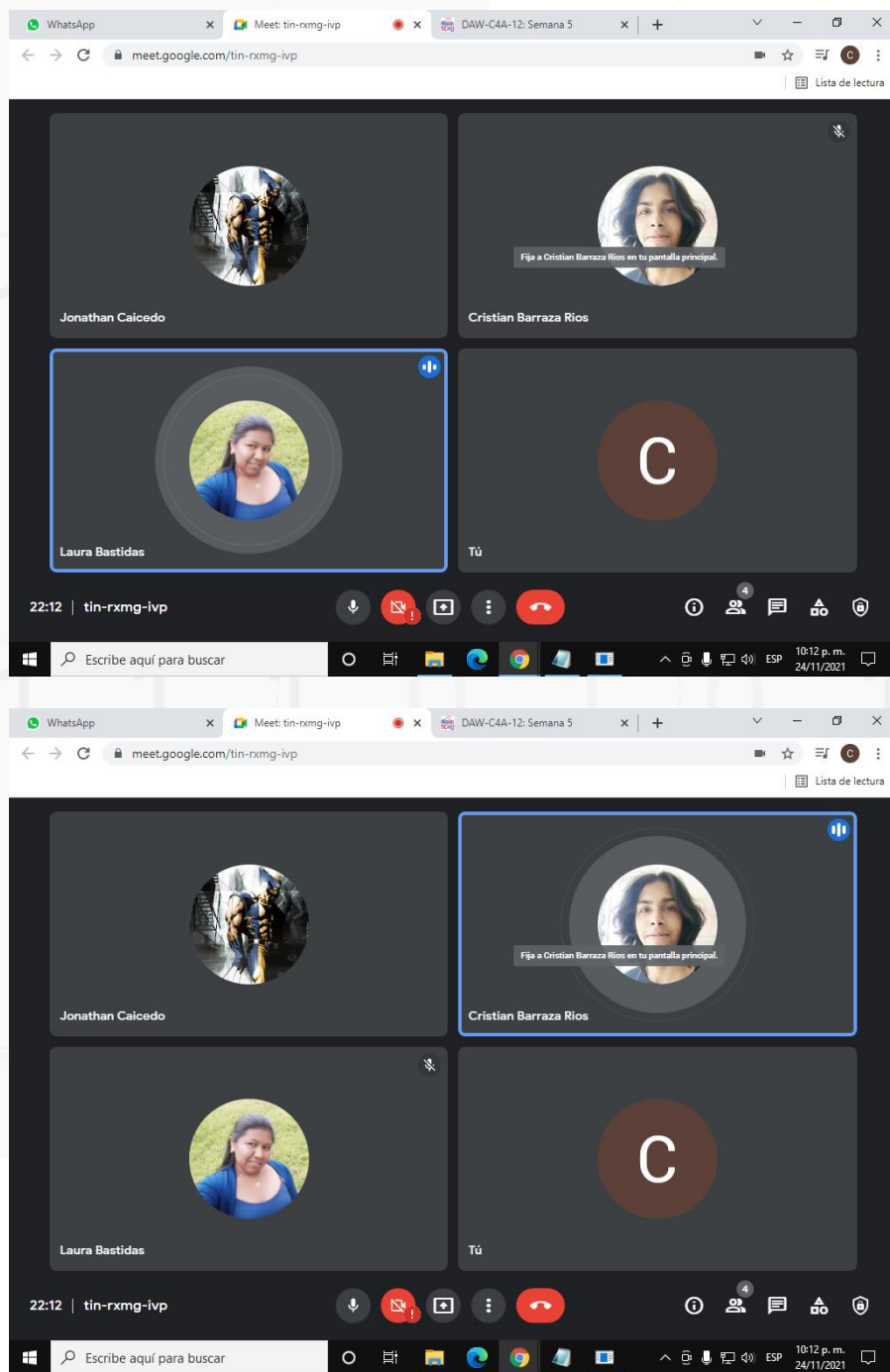


El futuro digital
es de todos

MinTIC

‘Mision
TIC2022’

En las imágenes que se anexan al trabajo podemos apreciar la participación de cada uno de los integrantes del grupo Techno Team, en cuanto refiere en la manera del desarrollo del trabajo que se va a presentar, como se va a recibir, verificar y organizar el trabajo.



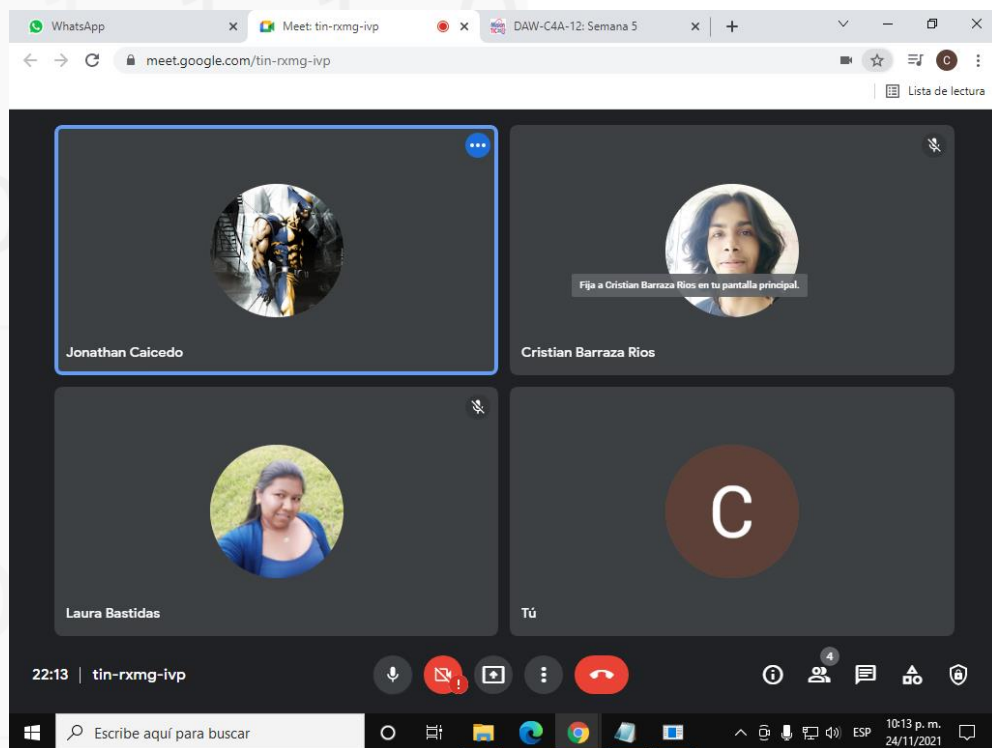
Universidad de Caldas



El futuro digital
es de todos

MinTIC

«Mision
TIC 2022»



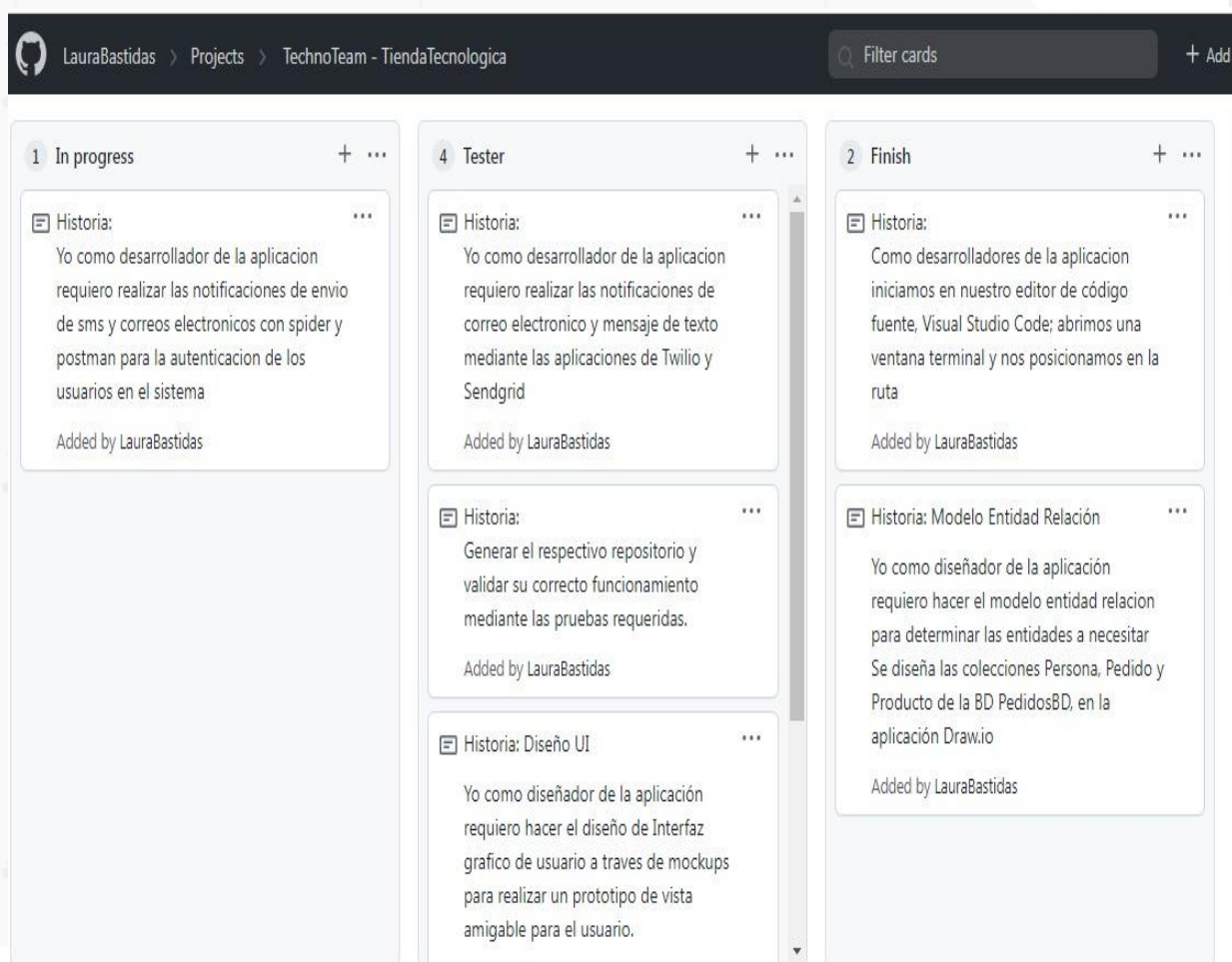
Universidad de Caldas



Tablero Kanban.

Actividades planteadas en el tablero Kanban, se actualiza el tablero Kanban con las actividades correspondientes al Sprint3, proyecto tienda virtual de tecnología.

En la siguiente imagen se puede observar cómo se visualiza el tablero:





Se procederá a agregar funcionalidades de notificación en Loopback, utilizaremos el servicio de correo electrónico y mensajería de texto.

Se modifica el archivo `autenticacion.services.ts`

```
src > controllers > TS persona.controller.ts > create > then() callback
61 let asunto = 'registro en la plataforma';
62 let contenido = `Hola ${persona.nombres}, su nombre de usuario es: ${persona.correo}
63 fetch('http://127.0.0.1:5000/envio-correo?correo_destino=${destino}&asunto=${asunto}
64 .then((data: any) => {
65   console.log(data);
66 })
67 return p;
68 }
69
70 @get('/personas/count')
71 @response(200, {
72   description: 'Persona model count',
73   content: { 'application/json': { schema: CountSchema } },
74 })
75 async count(
76   @param.where(Persona) where?: Where<Persona>,
77 ): Promise<Count> {
78   return this.personaRepository.count(where);
79 }
80
81 @get('/personas/')
```

```
> pedidos@0.0.1 build
> lb-tsc

> pedidos@0.0.1 start
> node -r source-map-support/register .

Server is running at http://[::1]:3000
Try http://[::1]:3000/ping
```

Se crean los métodos de generación y cifrado de contraseña

```
src > services > autenticacion.service.ts > AutenticacionService > GenerarClave
20
21 GenerarClave() {
22   let clave = generador(8, false);
23   return clave;
24 }
25
26
27 CifrarClave(clave: string) {
28   let claveCifrada = cryptoJS.MD5(clave).toString();
29   return claveCifrada;
30 }
31
```




Se crean las variables para las contraseñas, en el controlador de personas.

```
55 @post('/personas')
56 @response(200, {
57   description: 'Persona model instance',
58   content: {'application/json': {schema: getModelSchemaRef(Persona)}},
59 })
60 async create(
61   @requestBody({
62     content: {
63       'application/json': {
64         schema: getModelSchemaRef(Persona, {
65           title: 'NewPersona',
66           exclude: ['id'],
67         }),
68       },
69     },
70   }) persona: Omit<Persona, 'id'>,
71 ): Promise<Persona> {
72
73   let clave = this.servicioAutenticacion.GenerarClave();
74   let claveCifrada = this.servicioAutenticacion.CifrarClave(clave);
75   persona.clave = claveCifrada;
76   let p = await this.personaRepository.create(persona);
77 }
```

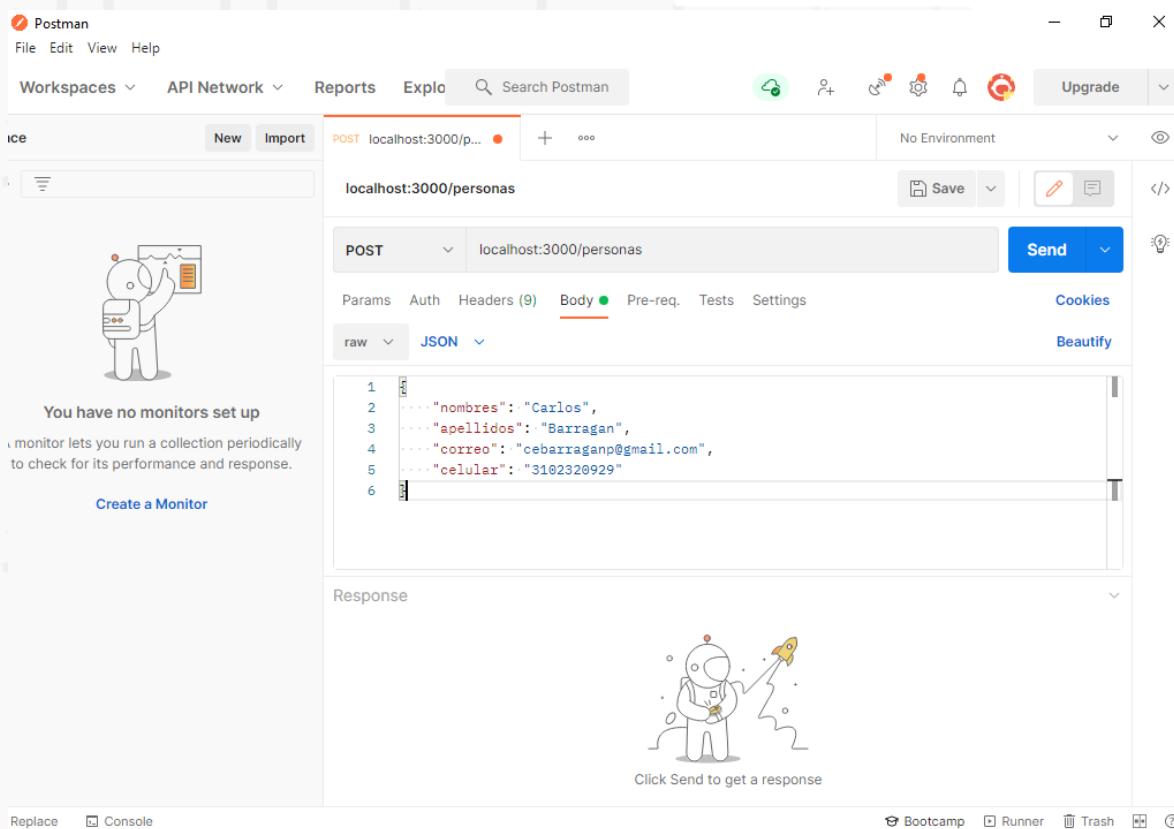
Se modifica el atributo clave para que el programa no la requiera

```
30
31 @property({
32   type: 'string',
33   required: true,
34 })
35 celular: string;
36
37 @property({
38   type: 'string',
39   required: false,
40 })
41 clave: string;
42
```



Se procede a realizar las pruebas de funcionamiento en el aplicativo POSTMAN para verificar que las claves se están generando de manera correcta.

Una vez que se inicia el aplicativo procedemos a diligenciar la información correspondiente a la prueba de notificación, como sabemos una vez que iniciamos el programa en loopback este se ejecuta en la URL localhost:3000/personas



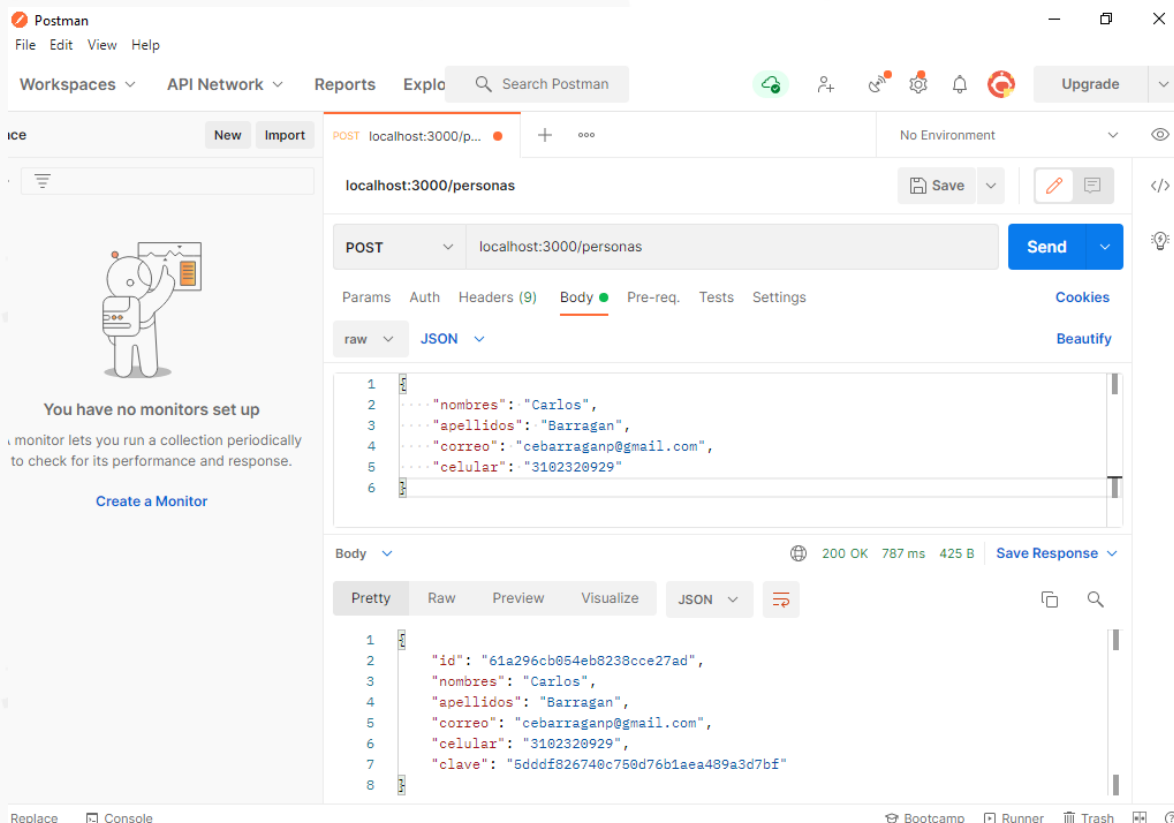


El futuro digital
es de todos

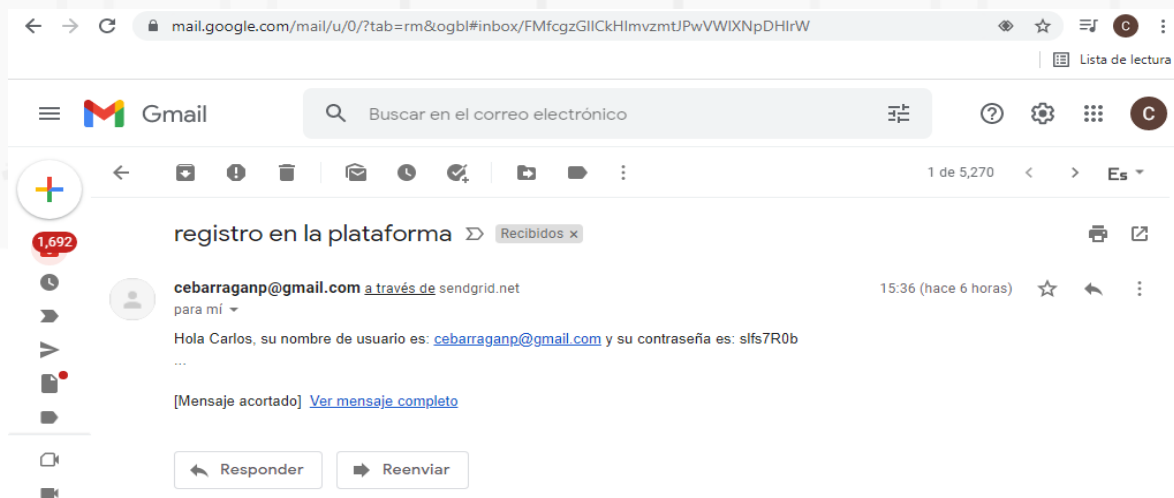
MinTIC

«Misión
TIC 2022»

Como se puede apreciar en la imagen generada por el sistema las pruebas de funcionamiento fueron exitosas:



Nos dirigimos al correo electrónico para verificar que se recibe la notificación:



Universidad de Caldas



Podemos verificar el mensaje en la aplicación Loopback.

The screenshot shows the Visual Studio Code interface with the 'personas.model.ts' file open. The Explorer sidebar on the left shows the project structure. The main editor area displays a REST client request and response. The request is a GET request to 'http://[::1]:3000/ping'. The response is a 200 OK status with a JSON body containing a 'url' and a 'status' of 200. The terminal at the bottom shows the command 'Try http://[::1]:3000/ping' and the response details.

```
src > models > TS personas.model.ts > Persona
27   required: true,
28   })
29   correo: string;
```

```
Try http://[::1]:3000/ping
Response {
  size: 0,
  timeout: 0,
  [Symbol(Body internals)]: {
    body: PassThrough {
      _readableState: [ReadableState],
      _events: [Object: null prototype],
      _eventsCount: 2,
      _maxListeners: undefined,
      _writableState: [WritableState],
      allowHalfOpen: true,
      [Symbol(kCapture)]: false,
      [Symbol(kCallback)]: null
    },
    disturbed: false,
    error: null
  },
  [Symbol(Response internals)]: {
    url: 'http://127.0.0.1:5000/envio-correo?correo_destino=cebarragan@gmail.com&asunto=registro%20en%20la%20plataforma&contenido=Hola%20Carlos,%20su%20nombre%20de%20usuario%20es:%20cebarragan@gmail.com%20y%20su%20contrase%C3%B1a%20es:%20s1fs7R0b',
    status: 200,
    statusText: 'OK',
    headers: Headers { [Symbol(map)]: [Object: null prototype] },
    counter: 0
  }
}
```

Estrategia Autenticación con JWT

En el archivo autenticacion.services.ts se crea el método para identificar una persona

The screenshot shows the Visual Studio Code interface with the 'autenticacion.services.ts' file open. The Explorer sidebar on the left shows the project structure. The main editor area displays the 'identificarPersona' method, which is a function that takes 'usuario' and 'clave' as parameters and returns a 'Persona' object if the credentials are valid, or false otherwise.

```
31
32
33   IdentificarPersona(usuario: string, clave: string) {
34     try {
35       let p = this.personaRepository.findOne({where: {correo: usuario, clave: clave}});
36       if (p) {
37         return p;
38       }
39       return false;
40     } catch {
41       return false;
42     }
43   }
```



Y se inyecta en el constructor, repositorio de persona.

```
1 import { /* inject, */ BindingScope, injectable } from '@loopback/core';
2 import { repository } from '@loopback/repository';
3 import { Llaves } from '../config/llaves';
4 import { Persona } from '../models';
5 import { PersonaRepository } from '../repositories';
6
7 const generador = require("password-generator");
8 const cryptoJS = require("crypto-js");
9 const jwt = require("jsonwebtoken");
10
11 @injectable({scope: BindingScope.TRANSIENT})
12 export class AutenticacionService {
13   constructor(
14     @repository(PersonaRepository)
15     public personaRepository: PersonaRepository { }
16   ) {}
17
18   GenerarClave() {}
19 }
```

Se crea el método para generar el token

```
43 GenerarTokenJWT(persona: Persona) {
44   let token = jwt.sign({
45     data: {
46       id: persona.id,
47       correo: persona.correo,
48       nombre: persona.nombres + " " + persona.apellidos
49     },
50     Llaves.claveJWT
51   }, Llaves.claveJWT);
52   return token;
53 }
```

Se instala npm i jsonwebtoken

se agrega una constante; la constante const jwt = require("jsonwebtoken");

```
1 import { /* inject, */ BindingScope, injectable } from '@loopback/core';
2 import { repository } from '@loopback/repository';
3 import { Llaves } from '../config/llaves';
4 import { Persona } from '../models';
5 import { PersonaRepository } from '../repositories';
6
7 const generador = require("password-generator");
8 const cryptoJS = require("crypto-js");
9 const jwt = require("jsonwebtoken");
```

Debemos generar o crear una carpeta y un archivo.

dentro de src crear carpeta con el nombre config

dentro de la carpeta config crear el archivo con el nombre llaves.ts

```
1 export namespace Llaves {
2   export const claveJWT = "JWT@2022";
3   export const urlServicioNotificaciones = "http://localhost:5000"
4 }
5
```



Modificar la línea fetch del controlador de persona luego del método POST

```
src > controllers > persona.controllers > PersonaController > identificarPersona
75 let claveCifrada = this.servicioAutenticacion.cifrarClave(clave);
76 persona.clave = claveCifrada;
77 let p = await this.personaRepository.create(persona);
78
79 //Notificar al usuario
80 let destino = persona.correo;
81 let asunto = 'Registro en la plataforma';
82 let contenido = `Hola ${persona.nombres}, su nombre de usuario es: ${persona.correo} y su contraseña es: ${claveCifrada}`;
83 fetch(`${Llaves.urlServicioNotificaciones}/envio-correo?correo_destino=${destino}&asunto=${asunto}&contenido=${contenido}`)
84   .then((data: any) => {
85     console.log(data);
86   })
87   return p;
88 }
```

importar archivo llaves

import {Llaves} from '../config/llaves';

```
src > services > autenticacion.service.ts > AutenticacionService > GenerarClave
1 import { /* inject, */ BindingScope, injectable } from '@loopback/core';
2 import { repository } from '@loopback/repository';
3 import {Llaves} from '../config/llaves';
4 import {Persona} from '../models';
5 import {PersonaRepository} from '../repositories';
6
```

Se crea el método para validar token

```
src > services > autenticacion.service.ts > AutenticacionService > GenerarClave
56 ValidarTokenJWT(token: string) {
57   try {
58     let datos = jwt.verify(token, Llaves.claveJWT);
59     return datos;
60   } catch {
61     return false;
62   }
63 }
64
```



En el controlador de persona se crea el método POST para identificar personas

```
20 public servicioAutenticacion: AutenticacionService
21 } {
22 }
23
24 @post("/identificarPersona", {
25   responses: {
26     '200': {
27       description: "Identificación de usuarios"
28     }
29   }
30 })
31
32 async identificarPersona(
33   @requestBody() credenciales: Credenciales
34 ) {
35   let p = await this.servicioAutenticacion.IdentificarPersona(credenciales.usuario, credenciales.clave);
36   if (p) {
37     let token = this.servicioAutenticacion.GenerarTokenJWT(p);
38     return {
39       datos: {
40         nombre: p.nombres,
41         correo: p.correo,
42         id: p.id
43       },
44       tk: token
45     }
46   } else {
47     throw new HttpErrors[401]("Datos inválidos");
48   }
49 }
50
51
52
53 }
```

Se crea un modelo para las Credenciales

lib4 model

nombre-> Credenciales

tipo-> model

prop. Adic? -> no

prop -> usuario

tipo -> string

es id? -> no

es necesario? -> si

prop -> clave

tipo -> string

es id? -> no

es necesario? -> si



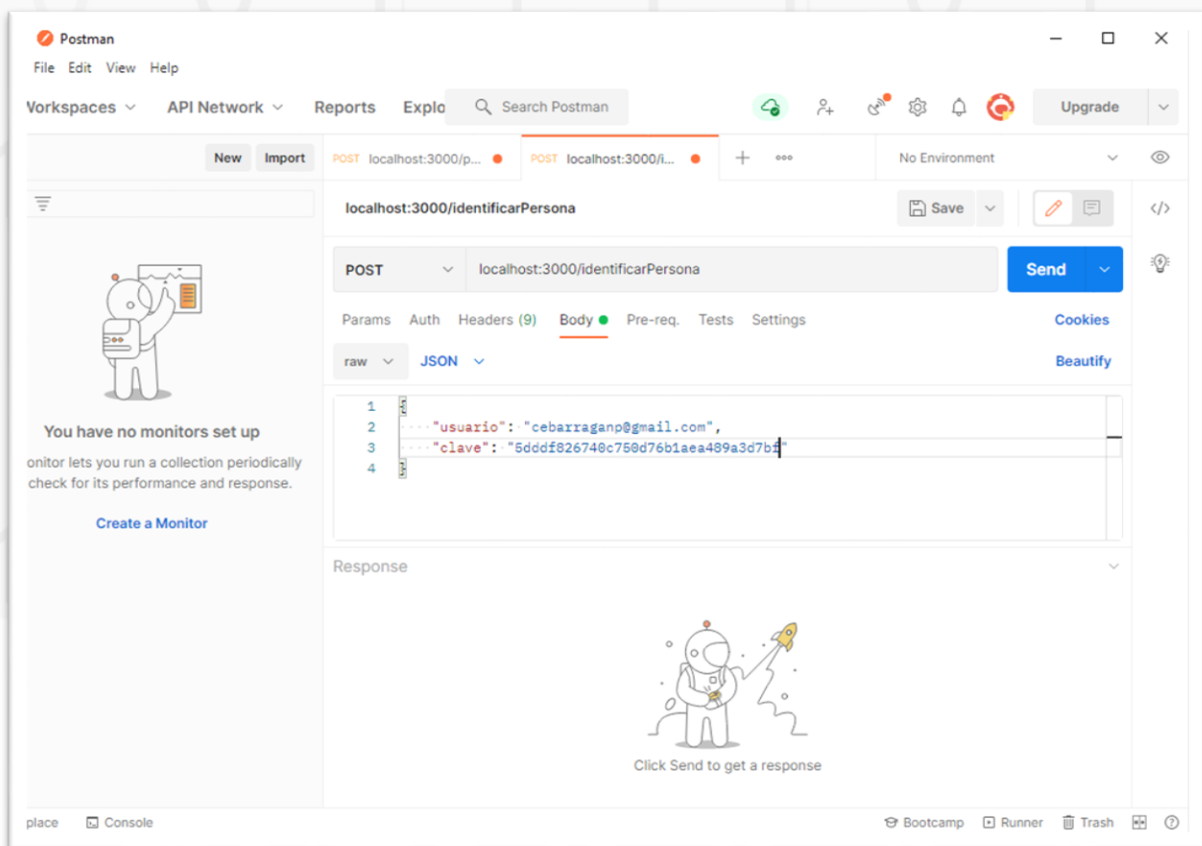
El futuro digital
es de todos

MinTIC

«Mision
TIC 2022»

```
1 import {Model, model, property} from '@loopback/repository';
2
3 @model()
4 export class Credenciales extends Model {
5   @property({
6     type: 'string',
7     required: true,
8   })
9   usuario: string;
10
11   @property({
12     type: 'string',
13     required: true,
14   })
15   clave: string;
16
17   constructor(data?: Partial<Credenciales>) {
18     super(data);
19   }
20 }
21
22 export interface CredencialesRelations {
23   // describe navigational properties here
24 }
25
26 export type CredencialesWithRelations = Credenciales & CredencialesRelations;
```

Se ejecuta y se hacen las pruebas en el aplicativo POSTMAN. Donde se utilizan los datos que se recibieron por correo.



Universidad de Caldas

[illegible]



SEGUNDA PARTE

Se procede a instalar los paquetes en la consola de git

```
npm i --force @loopback/authentication
```

```
npm i --force @loopback/security
```

El paquete correspondiente a loopback/authentication, provee todo lo que concierne a la captura de las solicitudes y la validación de las mismas.

El paquete correspondiente loopback/security, provee todo lo que concierne y permite el acceso al user profile, que corresponde al perfil de la cuenta del usuario.

crear carpeta y archivo

dentro de src crear una nueva carpeta con el nombre strategies

dentro de strategies crear un nuevo archivo con el nombre admin.strategies.ts

El proceso completo del JWT consta de dos pasos:

- El usuario de una aplicación web/móvil/desktop hace login con sus credenciales en el servidor donde está publicada el API.
- El usuario es validado en el servidor y se crea un nuevo Token JWT (usando nuestro "secret-key") para entregárselo al usuario.



El futuro digital
es de todos

MinTIC

«Mision
TIC 2022»

```
src > strategies > admin.strategies.ts > EstrategiaAdministrador > authenticate
1  import {AuthenticationStrategy} from '@loopback/authentication';
2  import {Service} from '@loopback/core';
3  import {HttpErrors, Request} from '@loopback/rest';
4  import {UserProfile} from '@loopback/security';
5  import {parseBearerToken} from 'parse-bearer-token';
6  import {AutenticacionService} from '../services';
7
8  export class EStrategiaAdministrador implements AuthenticationStrategy {
9    name: string = 'admin';
10
11    constructor(
12      @Service(AutenticacionService)
13      public servicioAutenticacion: AutenticacionService
14    ) {}
15
16    async authenticate(request: Request): Promise<UserProfile | undefined> {
17      let token = parseBearerToken(request);
18      if (token) {
19        let datos = this.servicioAutenticacion.ValidarTokenJWT(token);
20        if (datos) {
21          let perfil: UserProfile = Object.assign({
22            nombre: datos.data.nombre
23          });
24          return perfil;
25        } else {
26          throw new HttpErrors[401]("El token incluido no es válido.");
27        }
28      } else {
29        throw new HttpErrors[401]("No se ha incluido un token en la solicitud.");
30      }
31    }
32  }
```

instalar
npm i parse-bearer-token

En el archivo application.ts

```
42    nested: true,
43  },
44  },
45  },
46  registerAuthenticationStrategy(this, EStrategiaAdministrador);
47  this.component(AuthenticationComponent);
48  }
49  }
50  }
51  }
```

En el archivo producto.controller.ts

Se coloca antes del metodo POST

@authenticate("admin") para proteger todos los métodos con autenticación, al que no se quiera proteger se le coloca @authenticate.skip()