Andrés Felipe Robles C.C. 1023960270

Carlos Andres Pineda C.C. 80150569

Edwin Cardozo C.C. 80833120

Luis Ariel Rojas C.C. 1052393702

Santiago Moreno C.C. 1113698439

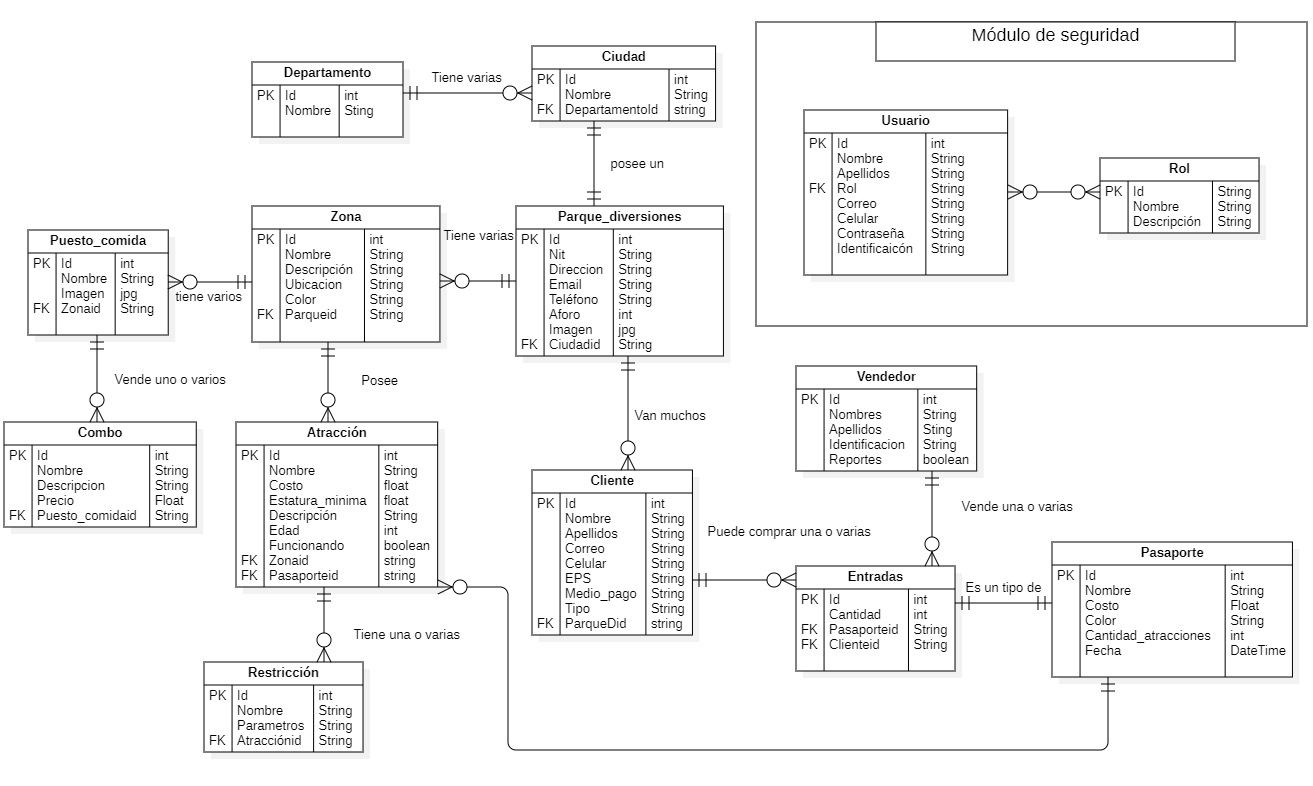
**Sprint 1**

**Grupo 5 – HALOS**

**Adventure Park**

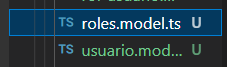
1. **Actualización de la arquitectura.**

De acuerdo a lo nombrado a inicio de semana, se hizo la respectiva corrección del diagrama de entidad relación que se había venido trabajando, añadiendo la tabla de rol con sus respectivos campos identificados. El modelo se seguridad se plantea aparte de las relaciones del modelo.



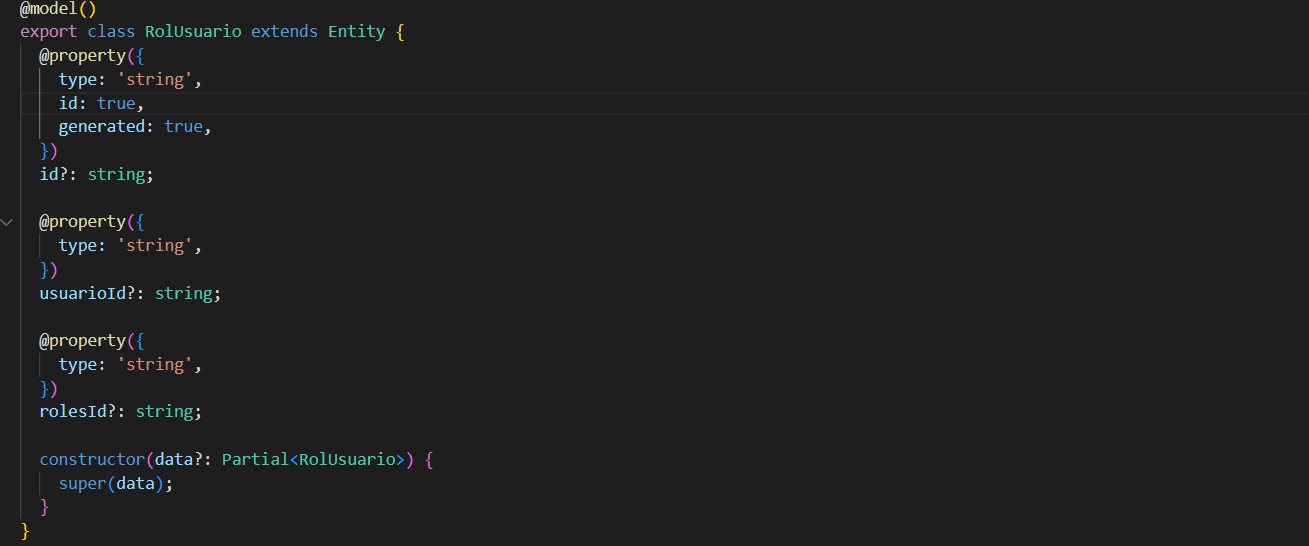
1. **Creación del modelo de seguridad**
   1. **Creación de los modelos de usuario y rol**

Como se muestra en el modelo de Entidad-Relación planteado la creación de usuario y rol son indispensables para realizar el modelo de seguridad requerido por la aplicación de Adventure Park.

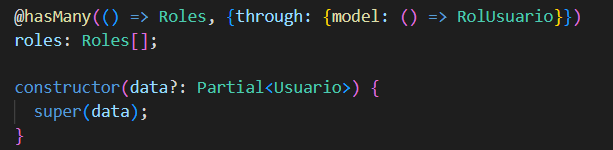


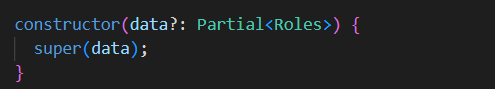
* 1. **Creación de la relación**

De acuerdo a lo visto en clase, las dos tablas anteriores tendrán una relación de muchos a muchos teniendo en cuenta que un usuario puede tener muchos roles y un rol puede tener múltiples usuarios. Debido a que es una relación de muchos a muchos, es necesario crear una tabla intermedia que en este caso se va a llamar Usuario-Rol y nos va a permitir hacer la relación pertinente.



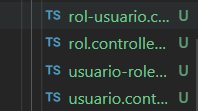
La imagen anterior muestra el modelo creado para la tabla intermedia RolUsuario, la cual solo contó con un parámetro al momento de su creación, pero al momento de crear la relación correspondiente se integraron dos campos más.





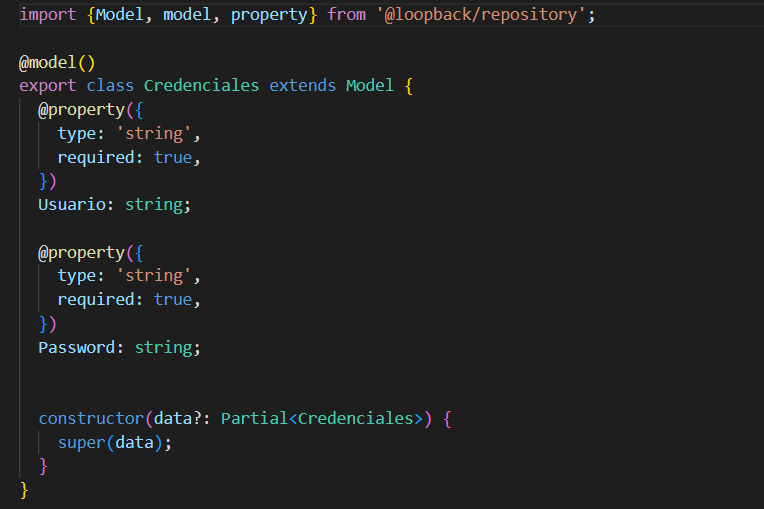
* 1. **Creación de los controladores**

Una vez finalizada la creación de los modelos necesarios para el módulo de seguridad se crean los controladores que van a ser necesarios para el mismo.



* 1. **Creación del modelo credenciales**

El modelo “Credenciales” va a ser un modelo intermedio que se encargará de recibir la información del usuario y de la contraseña y es indispensable para la lógica de nuestro negocio. Este no va a enviar información a la base de datos y por lo tanto no es necesaria la creación de los controladores o relaciones, va a ser únicamente para nuestra lógica de negocio. El modelo solo tendrá dos campos, el usuario y la contraseña que serán útiles para el momento en el cual se realice el login.



* 1. **Creación del método de identificación.**

Como en la aplicación que se está realizando el módulo de seguridad se encuentra únicamente relacionado con dos modelos, usuario y rol, la identificación del usuario se va a realizar por medio de un método POST en el controlador que de usuario.

  @post('/Login',{

    responses:{

      '200':{

        description:"Identificacion de las personas"

      }

    }

  })

  async identificar(

    @requestBody() credenciales:Credenciales

  ):Promise<Usuario | null>{

    let clavecifrada=this.servicioautenticacion.EncriptarPassword(credenciales.Password);

    let usuario = await this.usuarioRepository.findOne({

      where:{

        correo:credenciales.Usuario,

        clave:clavecifrada

      }

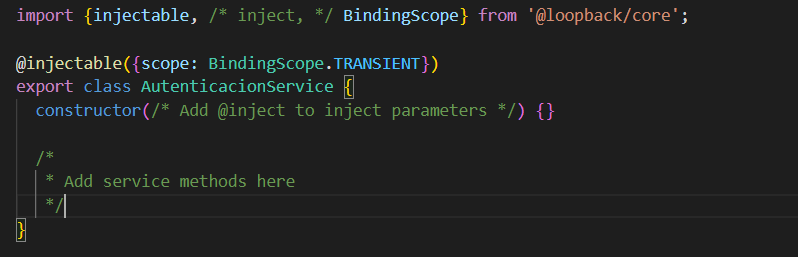
    });

    return usuario;

  }

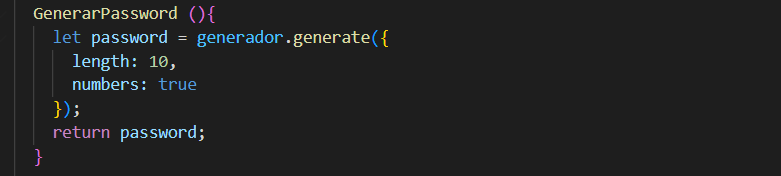
* 1. **Creación del servicio de autenticación**

Antes de enviar los datos del usuario a la base de datos es necesario consumir un microservicio que genere las contraseñas, además estas deben ser cifradas ya que si alguien tiene acceso a las bases de datos tendría acceso a las contraseñas.



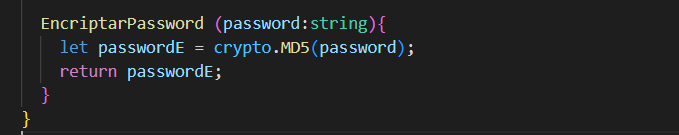
* 1. **Generación de la contraseña**

La generación de la contraseña se realizar gracias a uso de un microservicio llamado generate-password, el cual se usa para generar la contraseña del usuario de manera automática. Este se instala de manera local y se llama por medio de la declaración de una constante en el servicio de autenticación.



* 1. **Encriptación de la contraseña**

Una vez se ha generado la contraseña como se muestra anteriormente, se continúa con la encriptación de la misma consumiendo un microservicio llamado Crypto JS. Este se instala de manera local y se llama por medio de la declaración de una constante en el servicio de autenticación.



Se usa el método de encriptación MD5, según lo visto en clase.

* 1. **Llamado del servicio de autenticación en el modelo usuario**

Por medio de la función import, se llama el servicio creado anteriormente que contiene dos de los métodos creados hasta el momento.

import { AutenticacionService } from '../services';

En el constructor se hace el llamado correspondiente al servicio de autenticación gracias a:

@service(AutenticacionService)

    public servicioautenticacion:AutenticacionService

* 1. **Actualización del método post de crear**

Por motivos de seguridad, antes de crear el usuario se tiene que hacer uso de los microservicios previamente instalados y llamados. Para realizar esto se llaman los microservicios para generar y encriptar la contraseña generada.



* 1. **Proceso de notificación.**

Usando Anaconda Navigator y las variables de entorno creadas para el sprint 2, se realiza el proceso de notificación al usuario para confirmar que este haya recibido la información correspondiente a la contraseña. Para hacer el consumo de esta información es necesario hacer la instalación local del fetch dentro de la API, este nos permite usar otro servidor dentro de la aplicación.

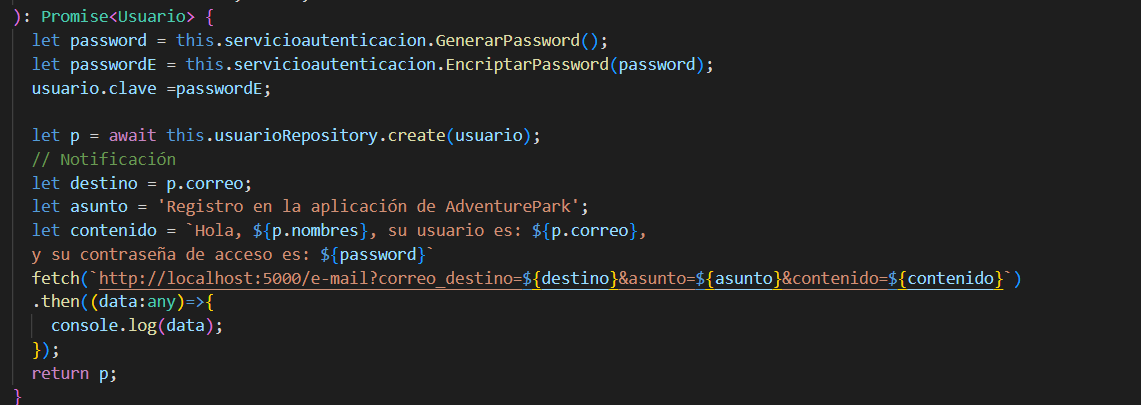
    fetch(`http://localhost:5000/e-mail?correo\_destino=${destino}&asunto=${asunto}&contenido=${contenido}`)

    .then((data:any)=>{

      console.log(data);

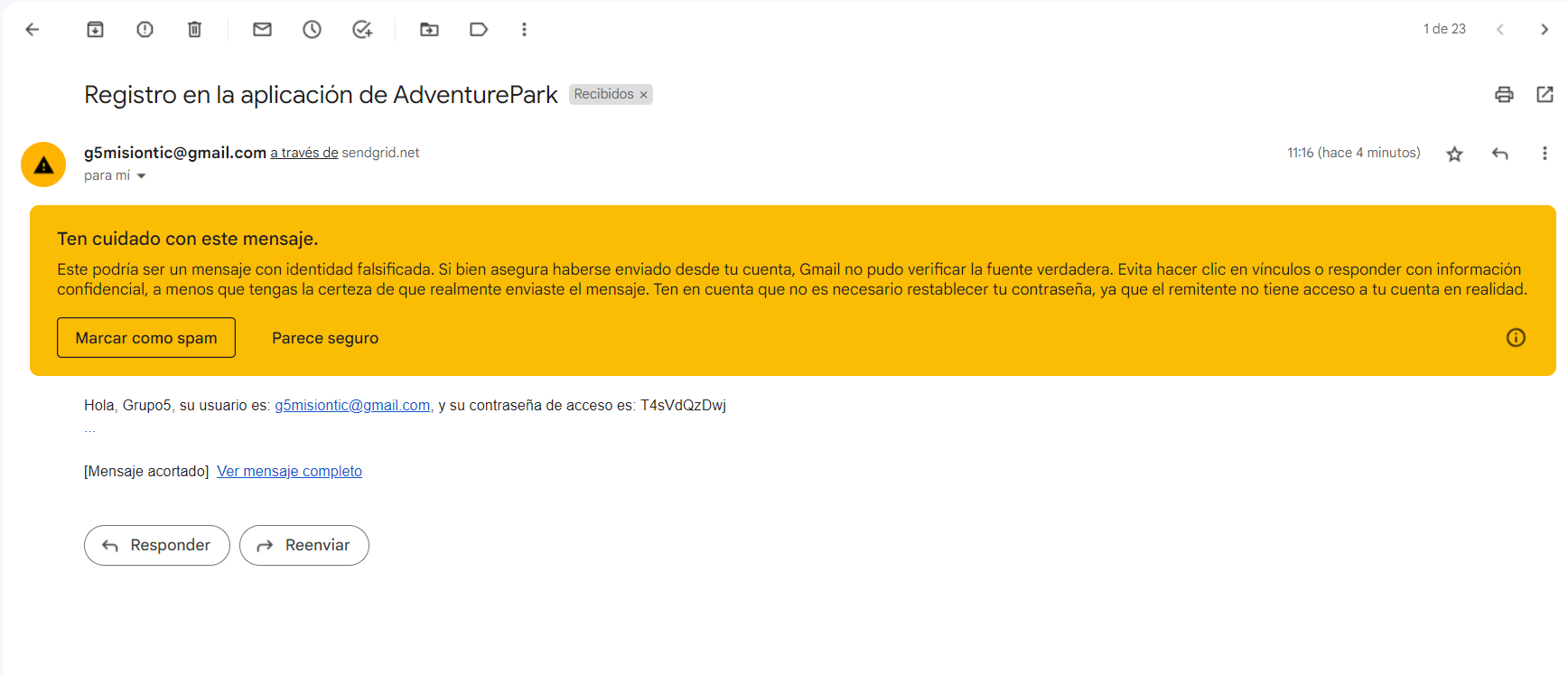
    });

Se usa el puerto 5000 ya que spider se ejecuta en este dominio e-mail (como se trabajó en el sprint anterior).



En la imagen anterior se muestra el uso de un “String template” para el envío del correo electrónico con la clave de acceso.

El correo con la contraseña de acceso se ve de la siguiente forma:



Al revisar el mongodb, se puede encontrar el usuario recién creado con la contraseña cifrada

