

IES ARROYO HARNINA

Técnico Superior en Desarrollo de Aplicaciones Multiplataformas

PROYECTO

GESTIÓN DE RESERVAS: CASA RURAL “LOS MARAGATOS”

Autor: M.^a Lourdes Cano Barrero.

Curso Académico: 2º DAM.

ÍNDICE

1. INTRODUCCIÓN	3
2. OBJETIVOS	4
3. MARCO TEORICO.....	5
4. DESARROLLO DEL PROYECTO	6
4.1 Diseño	6
4.2 Implementación.....	10
5. CONCLUSIONES	15
6.BIBLIOGRAFÍA	16

1. INTRODUCCIÓN

Para mejorar la eficiencia en la gestión de reservas de la casa rural “Los Maragatos”, localizada en el municipio de Entrín Alto, se ha desarrollado una aplicación de gestión utilizando tecnologías como Spring, Angular y MySQL.

Este proyecto ofrece la creación de un sistema en línea con el fin de facilitar y agilizar el proceso de reserva del alojamiento.

Este trabajo se ha desarrollado en dos componentes principales, una aplicación web fronted basada en Angular y un backend implementado en Java, utilizando el framework Spring.

El principal objetivo de este proyecto es proporcionar a los posibles clientes de esta casa rural una plataforma intuitiva, además, se ha implementado un sistema de inicio de sesión que permite gestionar sus reservas de forma segura.

En resumen, este trabajo brinda una visión del desarrollo del sistema de reserva para una casa rural y muestra los resultados obtenidos.

2. OBJETIVOS

En cuanto a los objetivos marcados para poder realizar este proyecto, se han establecido los siguientes:

1. Registro de usuarios y login: los usuarios tendrán que crear una cuenta para acceder a las a toda la aplicación.
2. Reservas: los usuarios podrían realizar reservas para la casa seleccionando las fechas que desean reservar y el número de personas que se alojarán.
3. Confirmación de reserva: una vez que se realiza la reserva, comprobando primero que haya disponibilidad, se enviaría al usuario una confirmación de reserva por correo electrónico.
4. Cancelación de reservas: los usuarios podrían cancelar una reserva existente si lo desean.
5. Comentarios: los usuarios podrían dejar comentarios sobre la casa después de su estancia.
6. Contacto: los clientes podrán contactar con el administrador del alojamiento mediante un formulario que se enviará vía email
7. Seguridad: JWT. A los usuarios se les asignará un token válido durante 24 horas y será validado en el backend.

3. MARCO TEORICO

Para el desarrollo de esta aplicación, se utilizarán varias tecnologías que se explicarán a continuación:

Spring: framework de Java que se utiliza para el desarrollo de aplicaciones empresariales. Proporciona una amplia gama de características, como la inyección de dependencias y la gestión de transacciones, que hacen que el desarrollo de aplicaciones sea más eficiente y escalable.

Angular: Es un framework de JavaScript que se utiliza para el desarrollo de aplicaciones web. Angular proporciona una estructura sólida para el desarrollo de aplicaciones web y ofrece características como la vinculación de datos y la inyección de dependencias.

MySQL: Es un sistema de gestión de bases de datos relacionales que se utiliza para almacenar y gestionar datos. MySQL es ampliamente utilizado en el desarrollo de aplicaciones web debido a su capacidad de manejar grandes cantidades de datos.

En la aplicación que se desarrollará, se utilizarán estas tecnologías para crear una plataforma que permita a los clientes reservar fácilmente una casa rural y a los propietarios estar en contacto con sus clientes. Se utilizará Spring para crear el backend que maneje la lógica de la aplicación y se integrará con una base de datos MySQL. Angular se utilizará para crear una interfaz de usuario que permita a los clientes reservar una casa rural y a los propietarios gestionar las reservas. En conjunto, estas tecnologías permitirán la creación de una aplicación de gestión de reservas eficiente para una casa rural.

4. DESARROLLO DEL PROYECTO

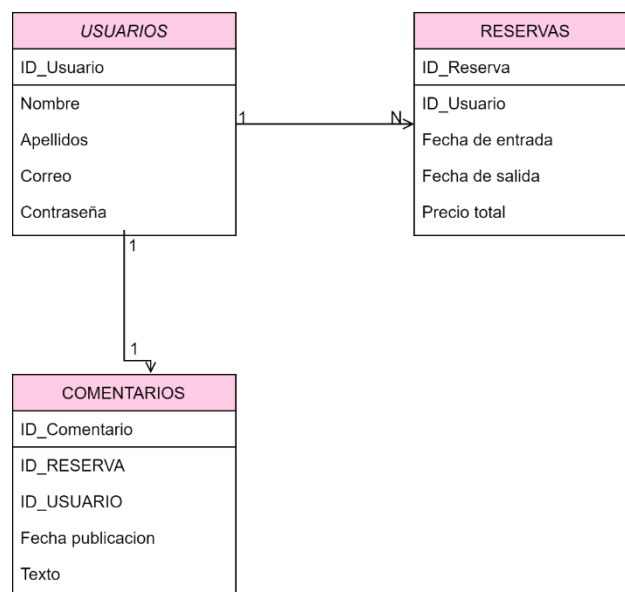
En este apartado se detallará el proceso de desarrollo del sistema de reservas.

4.1 DISEÑO

Se ha diseñado una arquitectura modular que permite la integración de nuevos módulos y funcionalidades con vistas al futuro. He implementado una interfaz con formularios claros y sencillos y procesos de reservas simples con la utilización de módulos de PrimeNG.

Asimismo, se implementó una seguridad básica en la aplicación con JWT, donde se almacena en ‘Session Storage’ datos no sensibles de los usuarios.

Diseño de la base de datos:

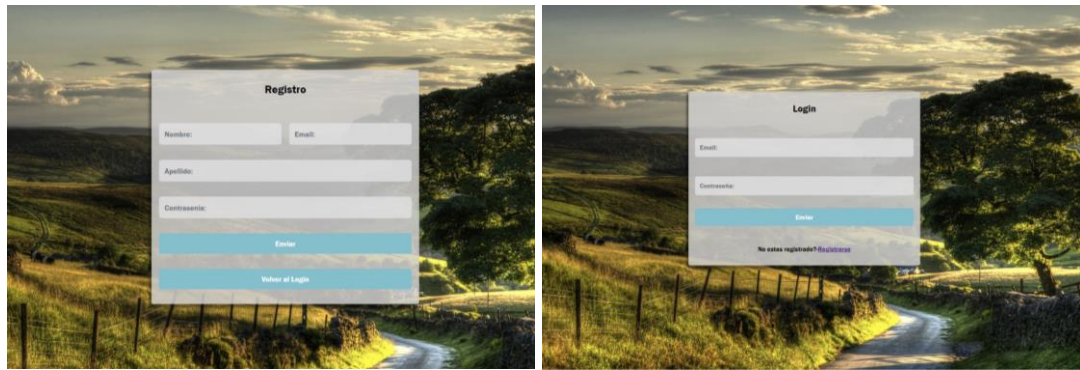


En cuanto al diseño de la BD, está formado por tres tablas con relacion 1->N entre Usuarios-Reservas y 1->1 entre Comentarios-Usuarios.

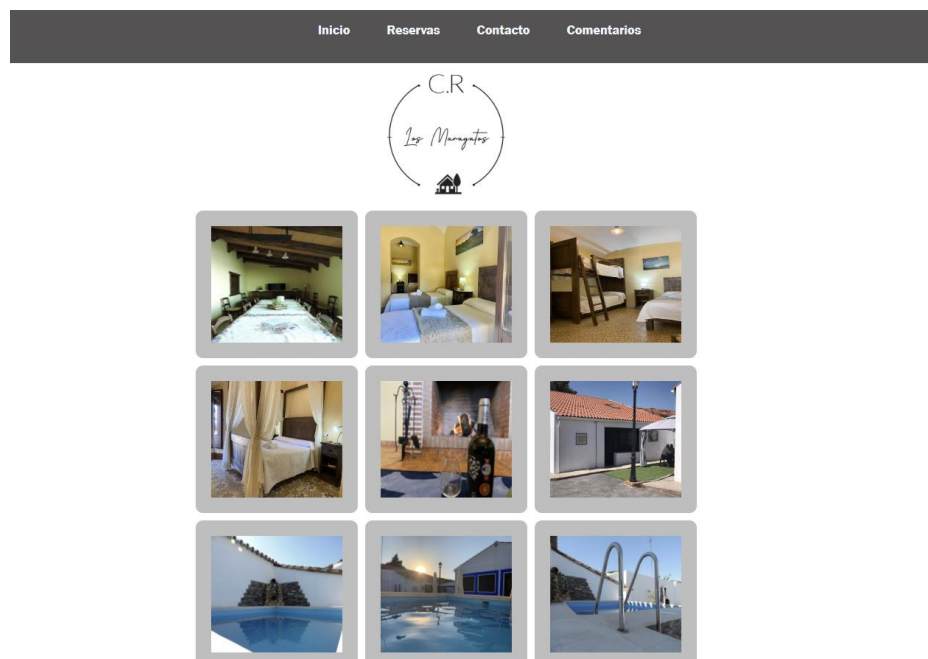
Diseño de los componentes:

- **Login y Registro de usuarios:** el diseño de los formularios pertenece a la biblioteca de PrimeNG, la cual proporciona componentes de interfaz de usuarios predefinidos y funcionales como estos.

Proyecto - Técnico Superior en Administración de Sistemas Informáticos
GESTIÓN DE RESERVAS:
CASRA RURAL “LOS MARAGATOS”

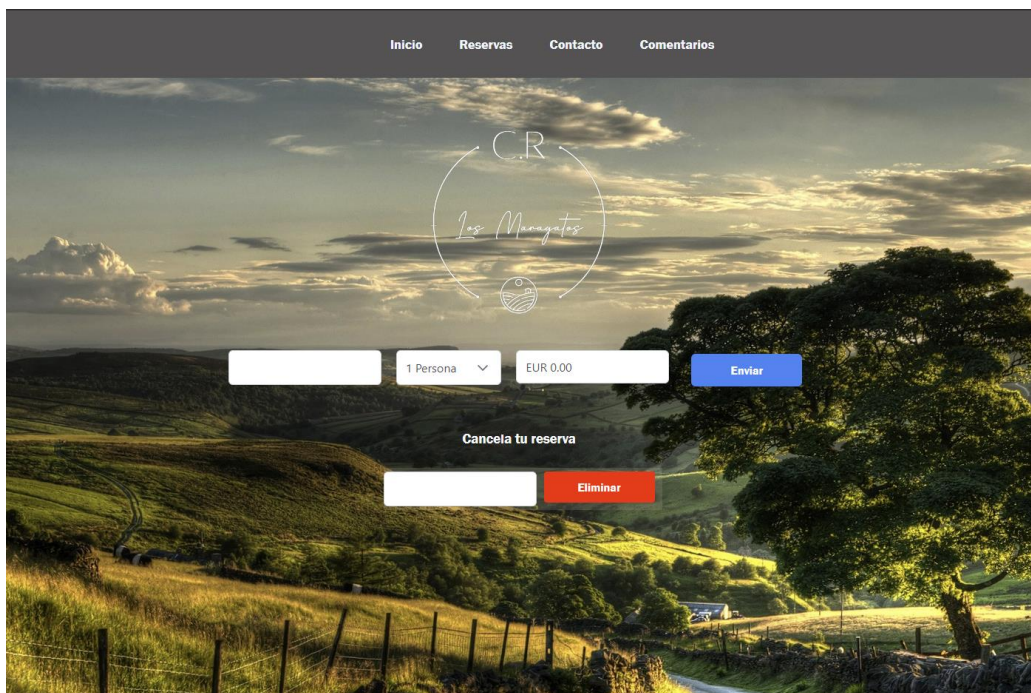


- El **inicio** de la app: se trata de una estructura de HTML estática, con un contenedor principal que contiene varias etiquetas de imágenes.

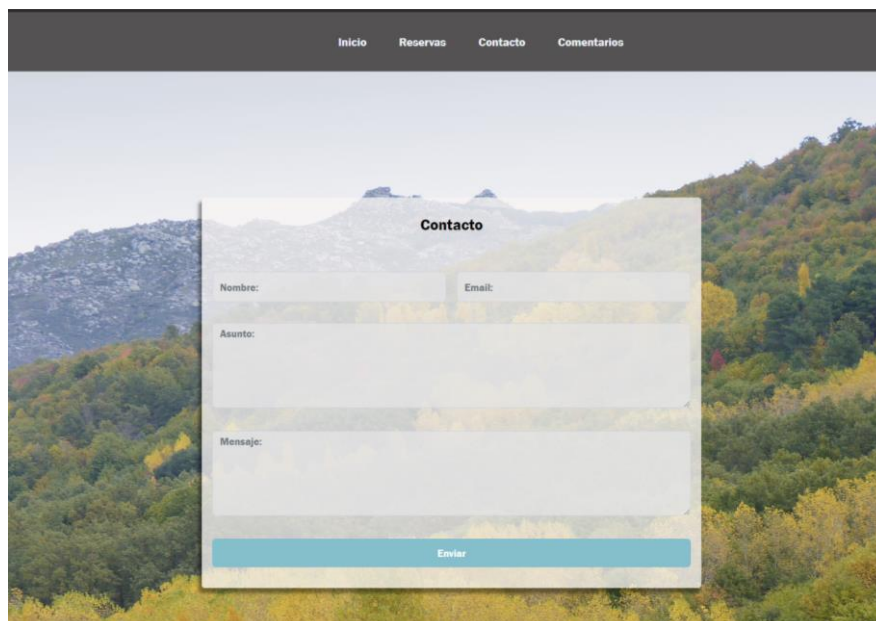


- Realizar **reservas**: el diseño de las reservas permite a los usuarios seleccionar fechas, indicar el número de personas y confirmar su reserva. También ofrece la funcionalidad de cancelar una reserva existente al ingresar el ID correspondiente y hacer clic en el botón de eliminación.

Proyecto - Técnico Superior en Desarrollo de Aplicaciones Multiplataformas
GESTIÓN DE RESERVAS:
CASRA RURAL “LOS MARAGATOS”

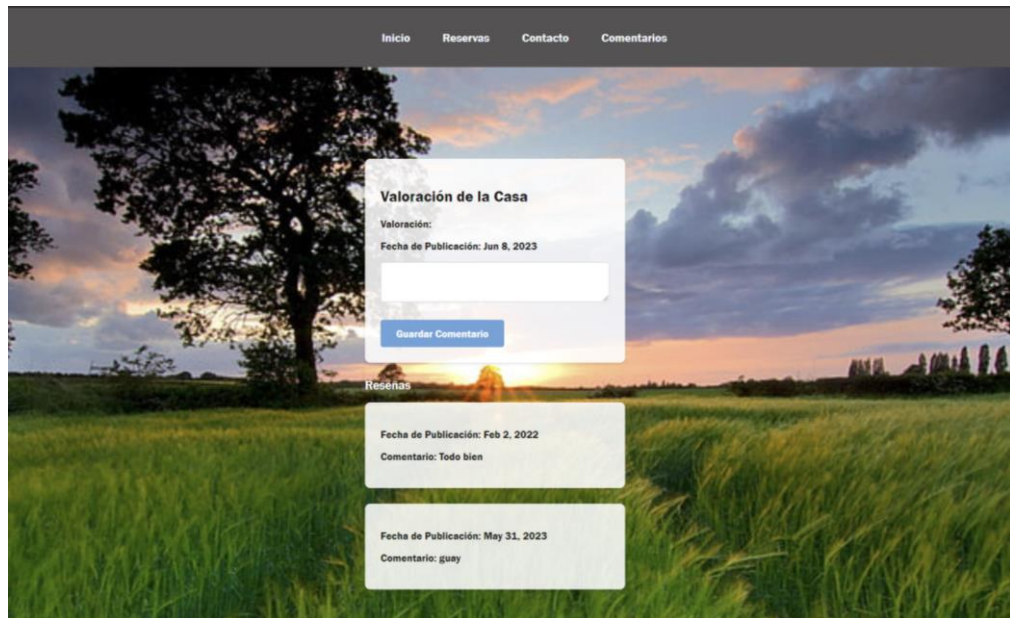


- **Contactar** con el alojamiento: se realiza también a través de un formulario de la librería PrimeNG. Dentro de un contenedor principal, se encuentra una ‘card’ que contiene el formulario. El formulario consta de campos para ingresar el nombre, correo electrónico, asunto y mensaje.



Proyecto - Técnico Superior en Administración de Sistemas Informáticos
GESTIÓN DE RESERVAS:
CASRA RURAL “LOS MARAGATOS”

- **Valoraciones** de los usuarios: contiene un formulario con un único campo para la opinión que han tenido los usuarios sobre su estancia. Después del formulario, hay una sección de lista de comentarios.



The screenshot displays a web interface for a rural accommodation. At the top, a navigation bar includes links for 'Inicio', 'Reservas', 'Contacto', and 'Comentarios'. The main content area features a 'Valoración de la Casa' (House Rating) form. This form includes a 'Valoración:' label, a 'Fecha de Publicación: Jun 8, 2023', a text input field, and a 'Guardar Comentario' button. Below the form, a section titled 'Reservas' (Reviews) lists two entries. The first entry, dated 'Feb 2, 2022', has a comment of 'Todo bien'. The second entry, dated 'May 31, 2023', has a comment of 'guay'. The entire interface is overlaid on a background image of a lush green field with a large tree and a sunset sky.

4.2 IMPLEMENTACIÓN

Para el backend, se utilizó el framework Spring de Java con la implementación de Rest controller para gestionar las solicitudes provenientes del frontend, y se desarrollaron servicios para procesar la lógica de negocio y la interacción con la base de datos.

- Rest-Controller de **Registro** y **Login** de usuarios: para manejar solicitudes HTTP POST. Para registrar nuevos usuarios: recibe un objeto ‘Usuario’ en el cuerpo de la solicitud y crea un nuevo usuario utilizando los datos proporcionados. Luego, verifica si se pudo guardar correctamente en la base de datos utilizando el servicio ‘usuarioService’. Para hacer login: recibo un ‘Usuario DTO’ con dos atributos, correo y contraseña y los convierte en un ‘UsuarioLoginReturnDTO’ para mandar al front la información del usuario y el token que se genera. Con ‘usuarioservice’ comprueba si el correo y la contraseña son válidos y se encuentran registrados en la BD.

```
no usages
@PostMapping("/api/user/create")
public ResponseEntity<?> registrarUser(@RequestBody Usuario user){

    //GUARDAR DATOS String nombre, String apellido, String correo, String contrasenia
    Usuario usuario=new Usuario(user.getNombre(),user.getApellido(),user.getCorreo(),user.getContrasenia());
    if(usuarioService.isSave(usuario)==true) {
        return new ResponseEntity<Usuario>(user, HttpStatus.OK);
    }
    return new ResponseEntity<>({ body: "Error", HttpStatus.NOT_FOUND);
}

no usages
@PostMapping("/api/user/login")
public ResponseEntity<?> login(@RequestBody UsuarioDTO loginRequest) {
    String correo = loginRequest.getCorreo();
    String contrasenia = loginRequest.getContrasenia();
    Usuario userLogged= usuarioService.loginUser(correo, contrasenia);
    if (userLogged != null) {
        // Generar el token JWT
        String token = jwtUtils.generateJwtToken(userLogged.getCorreo());

        // Validar el token JWT
        if (jwtUtils.validateJwtToken(token, userLogged.getCorreo())) {
            UsuarioLoginReturnDTO userReturn = new UsuarioLoginReturnDTO(userLogged.getCorreo(), token, userLogged.getId_usuario());
            return new ResponseEntity<>(userReturn, HttpStatus.OK);
        } else {
            // Generar un nuevo token
            String newToken = jwtUtils.generateJwtToken(userLogged.getCorreo());
            UsuarioLoginReturnDTO userReturn = new UsuarioLoginReturnDTO(userLogged.getCorreo(), newToken, userLogged.getId_usuario());
            return new ResponseEntity<>(userReturn, HttpStatus.OK);
        }
    } else {
        return new ResponseEntity<>({ body: "Usuario inválido", HttpStatus.NOT_FOUND);
    }
}
```

- Rest-Controller de **reservas**: realiza una solicitud POST donde llega como parámetro una ‘ReservaDTO’ con los campos que se indican en la vista, se transforma a un objeto de la clase ‘Reservas’, que es nuestro modelo en el backend y comprueba si las fechas están disponibles a través de métodos implementados en el Servicio de las reservas. Cuando la reserva está realizada, manda un mail de confirmación al usuario de la reserva con los datos de la reserva más el ‘id’ de la reserva para posteriormente tener la oportunidad de cancelar dicha reserva.

```
no usages
@PostMapping("api/reserva/reservas")
public ResponseEntity<?> confirmacionReserva(@RequestBody ReservaDTO reserva) throws MessagingException {
    // Lógica para registrar la reserva

    Usuario usuario=reservasService.findUserByReserva(reserva);
    Reserva reserva1=reservasService.convertToEntity(reserva);
    List<Reserva> reservasenFecha=reservasService.obtenerReservasEntreFechas(reserva1.getFecha_Entrada(), reserva1.getFecha_Salida());
    if(!reservasenFecha.isEmpty()){
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Las fechas seleccionadas no están disponibles.");
    }
    //GUARDAR LA RESERVA
    reservasService.createReserva(reserva);
    Long id= reservasService.findID(reserva1);
    System.out.println( reservasService.findID(reserva1));
    //MANDAR MAIL DE CONFIRMACION DE RESERVA
    MimeMessage message = mailSender.createMimeMessage();
    MimeMessageHelper helper = new MimeMessageHelper(message, multipart: true);
    message.setFrom(new InternetAddress("casarurallosmaragatos@gmail.com"));
    helper.setTo(usuario.getCorreo());
    helper.setSubject("Confirmación de reserva");
    String content ="<html><body><h1>Confirmación de reserva</h1><p>Datos de la reserva:</p><ul><li>Nombre: " + usuario.getNombre() +
        "</li><li>Fecha Entrada: " + reserva1.getFecha_Entrada() +
        "</li><li>Fecha Salida: " + reserva1.getFecha_Salida()+
        "</li><li>Importe: " + reserva1.getImporteTotal()+
        "</li><li>Código de Identificación de la reserva: " + id+
        "</li></ul>"+"El importe se abonará en el establecimiento"+
        "</li></ul></body></html>";
    helper.setText(content, true);
    mailSender.send(message);

    return ResponseEntity.ok(reserva);
}
```

- Rest-Controller de **contacto**: la solicitud POST recibe como parámetro un ‘Contacto Formulario’ que sería la clase modelo que utiliza en Angular y simplemente manda un mail con los datos del usuario al correo de la casa rural.

Proyecto - Técnico Superior en Desarrollo de Aplicaciones Multiplataformas
GESTIÓN DE RESERVAS:
CASRA RURAL “LOS MARAGATOS”

```
no usages
@CrossOrigin(origins = "http://localhost:4200")
@RestController
public class ContactoController {

    2 usages
    @Autowired
    JavaMailSender javaMailSender;

    no usages
    @PostMapping("/api/contact")
    public void mandarEmail(@RequestBody ContactoFormulario form){

        String nombre=form.getNombre();
        String correo= form.getCorreo();
        String asunto= form.getAsunto();
        String mensaje=form.getMensaje();

        SimpleMailMessage simpleMailMessage=new SimpleMailMessage();
        simpleMailMessage.setTo("casarurallosmaragatos@gmail.com");
        simpleMailMessage.setSubject("Nuevo mensaje de contacto");
        simpleMailMessage.setText("Nombre: " + nombre + "\nCorreo: "
            + correo + "\nAsunto "
            + asunto + "\nMensaje: " + mensaje);

        javaMailSender.send(simpleMailMessage);
        //COPIA DEL CORREO
        SimpleMailMessage userMail = new SimpleMailMessage();
        userMail.setTo(correo);
        userMail.setSubject("Copia del mensaje");
        userMail.setText("Gracias por contactarnos. Aquí tienes una copia de tu mensaje:"
            + "\nAsunto " + asunto + "\n" + mensaje);

        javaMailSender.send(userMail);
    }
}
```

- Rest-Controller de **comentarios**: se trata de una solicitud POST la cual recibe un ‘Comentario DTO’ y lo transforma al objeto modelo que tenemos en nuestra BD para poder guardarlo sin tener problemas. Confirma con el Servicio que ese usuario no tenga otro comentario ya registrado. Tenemos otra solicitud GET que manda la lista de comentarios guardados al front para poder mostrar todas las valoraciones de la casa.

```
no usages
@CrossOrigin(origins = "http://localhost:4200")
@RestController
public class ComentarioController {

    4 usages
    @Autowired
    ComentarioService comentarioService;

    no usages
    @GetMapping ("api/comentarioAll")
    public ResponseEntity<List<ComentarioDTO>> guardarComentario(){
        List<ComentarioDTO> comentarioDTOList=comentarioService.getAllComentarios();
        return new ResponseEntity<>(comentarioDTOList, HttpStatus.OK);
    }

    no usages
    @PostMapping ("api/comentario")
    public ResponseEntity<?> guardarComentario(@RequestBody ComentarioDTO comentario){

        Comentario comentario1=comentarioService.convertToEntity(comentario);

        if(comentarioService.isNullComent(comentario1, comentario1.getUsuario())==true){
            comentarioService.saveComentario(comentario1);
            return ResponseEntity.ok(comentario1);
        }

        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Ya tiene un comentario de la estancia");
    }
}
```

En cuanto a la implementación en Angular tenemos los siguientes servicios y componentes:

- Servicio de **registro y login**: manda en la solicitud los datos del usuario tanto en el login como en el registro, y almacena la información con el token en la ‘Session Storage’.

```
@Injectable({
  providedIn: 'root',
})
export class AuthService {
  private _user!: UsuarioLogged;

  constructor(private http: HttpClient) {
    this._user = JSON.parse(sessionStorage.getItem('user') || "{}")
  }

  login(loginForm: Login): Observable<any> {
    return this.http
      .post<any>(`${base_url}/login`, loginForm)
      .pipe(
        tap((response) => {
          this._user = response;
          sessionStorage.setItem('user', JSON.stringify(this._user))
          sessionStorage.setItem('token', this._user.token.toString())
        })
      );
  }

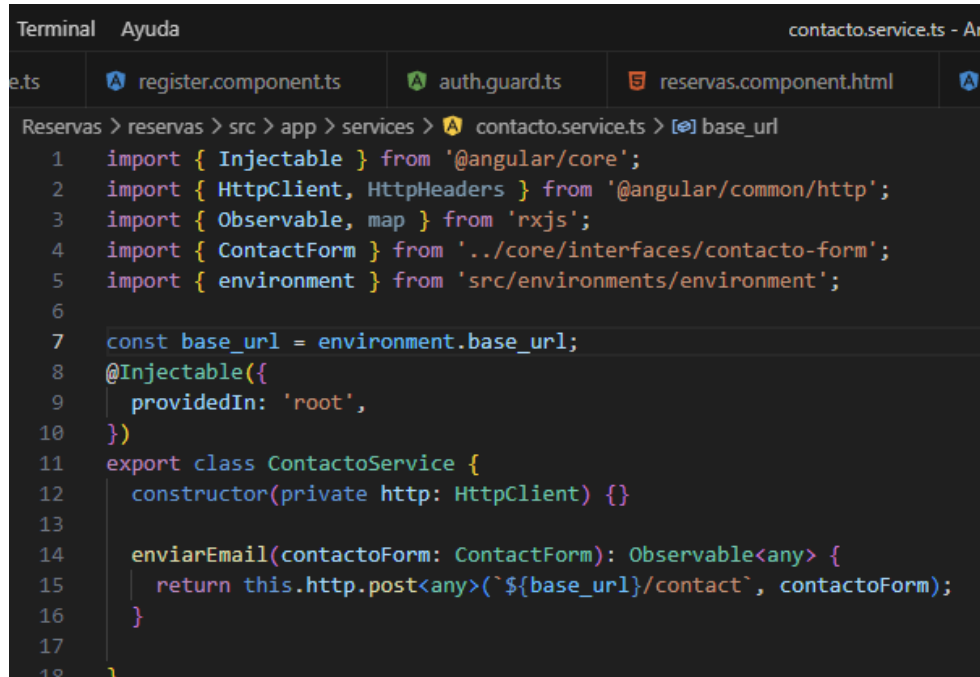
  register(createForm: Usuario): Observable<any> {
    return this.http.post<any>(`${base_url}/create`, createForm)
      .pipe(
        tap((response) => {
          this._user = response;
          sessionStorage.setItem('user', JSON.stringify(this._user))
          sessionStorage.setItem('token', this._user.token.toString())
        })
      );
  }
}
```

- Servicio de **comentarios**: manda un objeto de tipo ‘Comentario’ al back de Spring, y realiza una petición para obtener la lista de comentarios guardados por todos los usuarios.

```
Reservas > reservas > src > app > services > comentario.service.ts > [e] base_url
1  import { HttpClient } from '@angular/common/http';
2  import { Injectable } from '@angular/core';
3  import { environment } from 'src/environments/environment';
4  import { Comentarios } from '../core/models/comentarios';
5  import { Observable } from 'rxjs';
6
7  const base_url = environment.base_url;
8  @Injectable({
9    providedIn: 'root'
10 })
11
12 export class ComentarioService{
13
14   constructor(private http:HttpClient) {}
15
16
17   guardarComentarios(comentario: Comentarios): Observable<any>{
18     return this.http.post<any>(`${base_url}/comentario`,comentario)
19   }
20   obtenerComentarios() {return this.http.get<any>(`${base_url}/comentarioAll`)
21   }
22
23 }
```

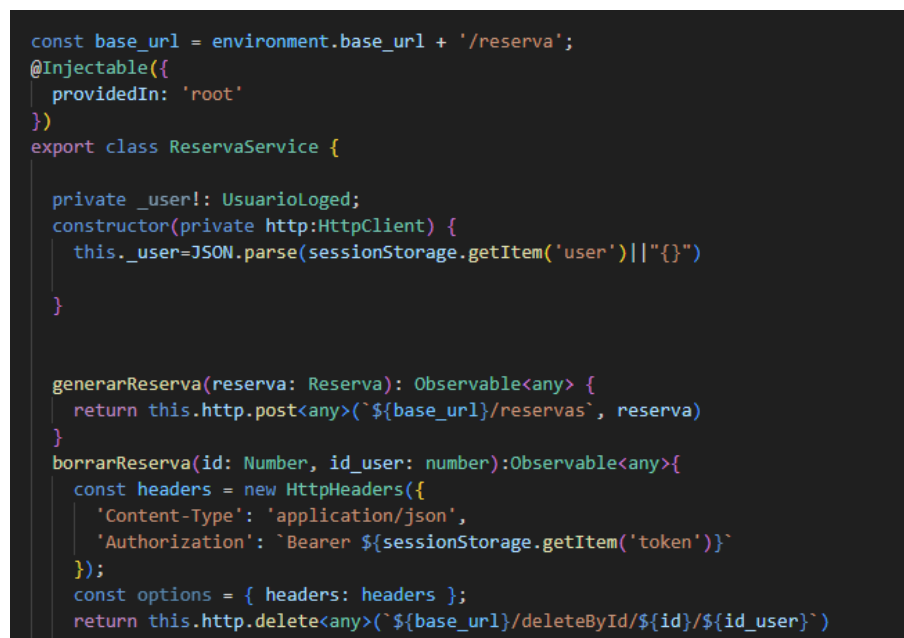
Proyecto - Técnico Superior en Desarrollo de Aplicaciones Multiplataformas
GESTIÓN DE RESERVAS:
CASRA RURAL “LOS MARAGATOS”

- Servicio de **contacto**: realiza solamente una petición POST mandando los datos del formulario de contacto.



```
Terminal  Ayuda  contacto.service.ts - Ar
e.ts  register.component.ts  auth.guard.ts  reservas.component.html
Reservas > reservas > src > app > services > contacto.service.ts > base_url
1  import { Injectable } from '@angular/core';
2  import { HttpClient, HttpHeaders } from '@angular/common/http';
3  import { Observable, map } from 'rxjs';
4  import { ContactForm } from '../core/interfaces/contacto-form';
5  import { environment } from 'src/environments/environment';
6
7  const base_url = environment.base_url;
8  @Injectable({
9    providedIn: 'root',
10 })
11 export class ContactoService {
12   constructor(private http: HttpClient) {}
13
14   enviarEmail(contactoForm: ContactForm): Observable<any> {
15     return this.http.post<any>(`${base_url}/contact`, contactoForm);
16   }
17
18 }
```

- Servicio de **reservas**: Tenemos una propiedad ‘_user’ que almacena los datos del usuario logueado. Se inicializa en el constructor utilizando el valor almacenado en la sesión del navegador. Para ello, se utiliza el método JSON.parse para convertir el valor almacenado en formato JSON a un objeto de angular. Hay dos solicitudes, POST, para guardar la reserva del usuario y un DELETE para que el usuario pueda borrar su reserva.



```
const base_url = environment.base_url + '/reserva';
@Injectable({
  providedIn: 'root'
})
export class ReservaService {

  private _user!: UsuarioLogado;
  constructor(private http:HttpClient) {
    this._user=JSON.parse(sessionStorage.getItem('user')||"{}")
  }

  generarReserva(reserva: Reserva): Observable<any> {
    return this.http.post<any>(`${base_url}/reservas`, reserva)
  }
  borrarReserva(id: Number, id_user: number):Observable<any>{
    const headers = new HttpHeaders({
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${sessionStorage.getItem('token')}`
    });
    const options = { headers: headers };
    return this.http.delete<any>(`${base_url}/deleteById/${id}/${id_user}`)
  }
}
```

5. CONCLUSIONES

Como conclusión de este trabajo realizado, decir que se consiguió realizar una aplicación web para la gestión de esta casa rural, teniendo la oportunidad de seguir desarrollándola en un futuro para introducir nuevas funcionalidades.

La intención de este proyecto es desarrollar una solución que pueda ser utilizada en un futuro cercano por el propietario de esta casa rural. Es una herramienta fácil de utilizar para gestionar las reservas y mejorar la experiencia de los posibles clientes de esta casa.

Este trabajo me ha brindado una oportunidad de enfrentar desafíos como es diseñar esta aplicación prácticamente funcional y buscar soluciones efectivas. Ha sido un proceso de aprendizaje rápido pero muy enriquecedor.

6. BIBLIOGRAFÍA

Direcciones Web.

- <https://jwt.io/libraries>
- <https://primeng.org/virtualscroller>
- <https://gustavoipeiretti.com/spring-boot-enviar-emails/>
- <https://github.com/jwt/jwt>
- <https://mvnrepository.com/>
- Enlace proyecto en GitHub: https://github.com/Lur162/Gestion_Reservas