

MÁSTER UNIVERSITARIO EN TECNOLOGÍAS WEB,
COMPUTACIÓN EN LA NUBE Y
APLICACIONES MÓVILES



VNIVERSITAT
DE VALÈNCIA

TRABAJO FIN DE MÁSTER

PLATAFORMA WEB PARA LA GESTIÓN DE UNA
CASA RURAL

AUTOR: ALBERTO HERRERO MARÍN

TUTOR: RAÚL PEÑA ORTIZ

JULIO 2025

REVISIÓN REALIZADA DESDE 27/6 A VERSIÓN 9/6

- Legendas:
- Correciones a abordar si o si
 - Correciones a abordar para no tener problemas en tribunal
 - Acuerdos relevantes
 - Notas al margen

→ OBJETIVO DE LA REVISIÓN: TODA LA MEMORIA

Te diré que no busques tanto detalle como en el TFG y me encanta una memoria de > 250 páginas - Bastaba con un ejemplo de cada tipo de implementación. Es la anticipación

los listados en general tienen poca resolución, ¿i con qué los has capturado? Modifica los que estén muy pixelados y tí genes. Por ejemplo, PlantUML puede generar SVG y de ahí convertirlos a PDF de alta calidad.

Así así, es un buen trabajo y no hay cosas de fondo que te vayan a impedir defender. Se vota la pedida profesional del Alberto del TFG al Sr. Alberto del Maestro.

Pero no por ello te relajes y aplaza los cambios que facilitan a base de pezón y pata



PD: Disculpa que haga tardadas así un mes en darte la risión. Las avaluaciones y los TFGS, que también el deadline de hoy, han podido cumplir.

Declaración de autoría:

Yo, Alberto Herrero Marín, declaro la autoría del Trabajo Fin de Máster (TFM) titulado “Plataforma web para la gestión de una casa rural” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual. El material no original que figura en este trabajo ha sido atribuido a sus legítimos autores.

Firmado:

scrónika

Resumen:

El presente Trabajo de Fin de Máster (TFM) consiste en el diseño e implementación de una plataforma web para la gestión de reservas en una casa rural. La solución incluye un sistema de autenticación, un calendario interactivo con integración de previsión meteorológica, y funcionalidades de reserva, modificación y cancelación de estancias. La arquitectura se basa en una interfaz desarrollada con Angular y un backend construido con Spring Boot y PostgreSQL. Además, se ha priorizado la escalabilidad, la experiencia de usuario y la seguridad del sistema. Este trabajo también incluye pruebas automáticas para garantizar la calidad del software. En conjunto, la plataforma busca ofrecer una herramienta eficiente tanto para los propietarios como para los huéspedes.

[] *preparado para un despliegue público.*

Abstract:

This Master's Thesis focuses on the design and development of a web platform for managing reservations in a rural house. The solution includes an authentication system, an interactive calendar with integrated weather forecast, and functionalities for booking, modifying, and canceling stays. The architecture is based on a frontend built with Angular and a backend developed using Spring Boot and PostgreSQL. The system emphasizes scalability, user experience, and security. Automated testing has also been implemented to ensure software quality. Overall, the platform aims to provide an efficient tool for both owners and guests.

L'utile a traduire tous astri-

Resum:

Aquest Treball de Fi de Màster (TFM) tracta sobre el disseny i desenvolupament d'una plataforma web per a la gestió de reserves en una casa rural. La solució inclou un sistema d'autenticació, un calendari interactiu amb la previsió meteòrològica integrada, i funcionalitats per a reservar, modificar i cancel·lar estades. L'arquitectura es basa en un frontend desenvolupat amb Angular i un backend amb Spring Boot i PostgreSQL. S'ha fet èmfasi en l'escalabilitat, l'experiència d'usuari i la seguretat. A més, s'han realitzat proves automàtiques per garantir la qualitat del programari. En conjunt, la plataforma pretén oferir una eina eficient tant per als propietaris com per als hostes.



Llurvelre a traducir tos capítols.

Agradecimientos:

En primer lugar, quiero agradecer a mi tutor/a por su dedicación, orientación y apoyo constante durante el desarrollo de este [TFM](#).

También agradezco a mi familia y amigos por su paciencia, motivación y confianza en mí a lo largo de todo el proceso.

Por último, agradezco ~~a los docentes del máster~~ por los conocimientos impartidos y las herramientas que me han permitido realizar este proyecto con éxito.

El profesorado del máster

yo cambie el orden y se dirige al final a los más importantes -

Índice general

1. Introducción	27
1.1. Introducción	27
1.2. Objetivos	29
1.2.1. Objetivo general	29
1.2.2. Objetivos específicos	29
1.3. Organización de la memoria	30
2. Estado del arte	33
2.1. Análisis de aplicaciones similares	33
2.1.1. Airbnb	33
2.1.2. Booking.com	34
2.1.3. Ruralka	36
2.1.4. Komoot	36
2.2. APIs y plataformas externas a utilizar	38
2.2.1. Portal turístico de Tuéjar	38
2.2.2. Portal turístico de Chelva	38
2.2.3. Web oficial del Ayuntamiento de Tuéjar	40
2.2.4. API de Instagram Graph	41
2.2.5. API meteorológica: Open-Meteo	41
2.2.6. API meteorológica: AEMET	41
2.3. Evaluación de tecnologías	42
2.3.1. Frontend	42
2.3.2. Backend	43
2.3.3. Contenerización y Orquestación	44
2.3.4. Bases de Datos	46
2.3.5. Servicios de plataformas en la nube	48
2.3.6. Tecnologías para validación y pruebas	50
2.3.7. Entorno de desarrollo, gestión de versiones y diseño de diagramas	50
2.3.8. Resumen de tecnologías seleccionadas	50

3. Requisitos, especificaciones, coste, riesgos, viabilidad	53
3.1. Requisitos	53
3.1.1. Descripción del sistema	53
3.1.2. Requisitos funcionales	53
3.1.3. Requisitos no funcionales	55
3.2. Especificaciones	56
3.2.1. Descripción general	56
3.2.2. Funcionalidades	56
3.3. Planificación y estimación de costes	58
3.4. Riesgos	60
3.5. Viabilidad	62
4. Análisis	65
4.1. Análisis	65
4.1.1. Diagrama de casos de uso	65
4.1.2. Diagramas de secuencia más relevantes	75
4.1.3. Diagrama de flujo del sistema de reservas	75
4.1.4. Diagrama de clases de primer nivel	76
5. Diseño	81
5.1. Arquitectura de componentes	81
5.1.1. Arquitectura de Componentes del Sistema Web	81
5.2. Sistema principal	84
5.3. Subsistema de recursos web	88
5.3.1. Extracción solicitada por el usuario	88
5.3.2. Extracción programada con scraping	89
5.4. Subsistema de publicaciones	90
5.5. Subsistema de previsión del tiempo	93
5.6. Diagrama de clases	94
5.7. Modelo de datos	94
5.7.1. Modelo físico de la base de datos relacional (<i>PostgreSQL</i>)	94
5.7.2. Modelo de datos de la base de datos no relacional <i>MongoDB</i>	96
5.8. Despliegue del sistema	97
5.9. Despliegue en entorno local	98
6. Implementación y pruebas	101
6.1. Implementación sistema principal	101
6.1.1. Seguridad	101

6.1.2. Gestión de reservas	104
6.1.3. Gestión de reseñas	109
6.1.4. Formulario de contacto	111
6.1.5. Persistencia y acceso a datos sistema principal	112
6.2. Subsistema de recursos web	113
6.2.1. Persistencia y acceso a datos subsistema de recursos web	115
6.3. Subsistema de publicaciones	118
6.4. Subsistema de extracción de datos meteorológicos	122
6.5. Aplicación Frontend y diseño de la interfaz de usuario	127
6.5.1. Componentes principales	127
6.6. Dockerización y despliegue de aplicaciones	134
6.6.1. Almacenamiento Persistente: PersistentVolume y PersistentVolumeClaim	136
6.6.2. Despliegues y Servicios de las Aplicaciones y Bases de Datos	138
6.6.3. Ingress Controller	144
6.7. Pruebas	146
6.7.1. Pruebas unitarias de la aplicación web	146
6.7.2. Pruebas funcionales de la aplicación web	149
6.8. Pruebas de seguridad	156
6.8.1. Configuración del entorno de pruebas	156
6.8.2. Pruebas de seguridad del backend	156
6.8.3. Pruebas de seguridad del frontend	157
6.9. Pruebas de rendimiento	158
6.9.1. Configuración del entorno de pruebas	158
6.9.2. Resultados obtenidos	160
6.10. Pruebas de accesibilidad y posicionamiento	160
7. Conclusiones	171
7.1. Revisión de costes	171
7.2. Conclusiones	172
7.3. Trabajo futuro	174
8. Bibliografía	177
A. Glosarios	181
A.1. Acrónimos y abreviaciones	181
A.2. Términos	184
A. Apéndices	187

A.1. Código fuente SpringBoot sobre seguridad	187
A.2. Código fuente SpringBoot sobre gestión de emails	188
A.3. Código fuente SpringBoot persistencia en el sistema principal	189
A.4. Código fuente SpringBoot scraping en el subsistema de extracción	199
A.5. Código fuente SpringBoot gestión de token de Instagram en subsistema de publicaciones	201
A.6. Código TypeScript del frontend de la aplicación	201
A.7. Código HyperText Markup Language (HTML) del frontend de la aplicación	220
A.8. Ficheros Docker del backend de la aplicación	240
A.9. Test unitarios del backend de la aplicación	240

Índice de figuras

1.1. Crecimiento del turismo rural en España	27
1.2. Interés por el turismo rural en la zona de la casa rural.	28
2.1. Ejemplo de experiencias ofrecidas en Airbnb.	34
2.2. Ejemplo de reseñas en Booking.com.	35
2.3. Ejemplo de recomendaciones turísticas en Booking.com.	35
2.4. Ejemplo de servicios ofrecidos en Ruralka.	36
2.5. Ejemplo de rutas personalizadas en Komoot.	37
2.6. Ejemplo de información turística extraída de la web de Tuéjar.	39
2.7. Ejemplo de rutas y eventos extraídos de la web de turismo de Chelva.	39
2.8. Contenido institucional relevante en la web del Ayuntamiento de Tuéjar.	40
3.1. Arquitectura de referencia del sistema.	57
3.2. Diagrama de Gantt del proyecto	61
4.1. Diagrama general de casos de uso.	66
4.2. Diagrama de clases de los actores que influyen en el sistema.	66
4.3. Diagrama de secuencia proceso de obtención de datos climáticos.	75
4.4. Diagrama de secuencia proceso de obtención de recursos web.	76
4.5. Diagrama de secuencia proceso de reserva de la casa rural.	77
4.6. Diagrama de actividades del ciclo de vida de una reserva.	78
4.7. Diagrama general de clases.	79
5.1. Diagrama de componentes del sistema.	82
5.2. Diagrama de secuencia: Confirmación de reserva.	85
5.3. Diagrama de secuencia: Obtener reservas (Gestor).	85
5.4. Diagrama de secuencia: Obtener reservas próximas.	86
5.5. Diagrama de secuencia: Actualización de estado.	86
5.6. Diagrama de secuencia: Obtener reservas por usuario.	87
5.7. Diagrama de secuencia del proceso de añadir una reseña.	87
5.8. Diagrama de secuencia del proceso de consulta de reseñas.	88

5.9.	Diagrama de secuencia del componente de contacto.	89
5.10.	Diagrama de secuencia para solicitudes de recursos por parte de usuarios.	90
5.11.	Diagrama de secuencia de la extracción programada de recursos web.	91
5.12.	Diagrama de secuencia de consulta, extracción y actualización de publicaciones desde la API (Application Programming Interface) de Instagram.	92
5.13.	Diagrama de secuencia del proceso de refresco del token de acceso (Subsistema de publicaciones).	92
5.14.	Diagrama de secuencia del componente de obtención de previsión del tiempo (Subsistema de previsión del tiempo).	93
5.15.	Diagrama de clases completo del sistema	95
5.16.	Modelo físico de la base de datos relacional PostgreSQL.	96
5.17.	Modelo físico de la base de datos no relacional MongoDB.	97
5.18.	Diagrama de despliegue del sistema basado en contenedores Docker sobre arquitectura Kubernetes.	99
5.19.	Diagrama de despliegue del sistema en entorno local.	100
6.1.	Diseño del componente <code>AdministradorComponent</code>	128
6.2.	Apartado para consultar reseñas existentes de <code>InicioComponent</code>	129
6.3.	Apartado con formulario para crear reseña de <code>InicioComponent</code>	130
6.4.	Apartado con formulario para crear una consulta de <code>InicioComponent</code>	130
6.5.	Carrusel de imágenes en el componente <code>LaCasaComponent</code>	131
6.6.	Vista general diseño del componente <code>ReservasComponent</code>	132
6.7.	Vista del resumen de la reserva en el componente <code>ReservasComponent</code>	133
6.8.	Vista del diseño del componente <code>ComoLlegarComponent</code>	134
6.9.	Vista del diseño del componente <code>RecomendacionesComponent</code>	135
6.10.	Vista del diseño del componente <code>EntornoComponent</code>	135
6.11.	Panel de control de Kubernetes mostrando el estado de los pods y servicios	145
6.12.	Ejecución de las pruebas unitarias realizadas en el Backend de la aplicación	149
6.13.	Ejecución de las pruebas de inicio de sesión	150
6.14.	Ejecución de las pruebas de registro	151
6.15.	Ejecución de las pruebas de reservas	152
6.16.	Ejecución de las pruebas de gestión de reservas	156
6.17.	Alertas iniciales detectadas en el backend por OWASP ZAP.	157
6.18.	Resultado inicial de las pruebas de seguridad del frontend.	158
6.19.	Resultado final de las pruebas de seguridad del frontend.	159
6.20.	Configuración del entorno de pruebas en JMeter	161
6.21.	Latencia media por endpoint — datos de <code>inicio1.csv</code>	162
6.22.	Latencia media por endpoint — datos de <code>reserva1.csv</code>	162

6.23. Latencia media por endpoint — datos de <code>visual1.csv</code>	163
6.24. Resultados Lighthouse — Página de inicio	164
6.25. Resultados Lighthouse — Página de entorno	165
6.26. Resultados Lighthouse — Página “La casa”	166
6.27. Resultados Lighthouse — Página de localizaciones	167
6.28. Resultados Lighthouse — Página de recomendaciones	168
6.29. Resultados Lighthouse — Página de reservas	169

Índice de tablas

2.1. Comparación de Plataformas para Despliegue de Aplicaciones	49
2.2. Tecnologías y herramientas utilizadas en el proyecto	51
3.1. Estimación de tareas del proyecto con asignación por sprint según planificación	59
3.2. Horario de dedicación	60
3.3. Estimación de la amortización de materiales	62
3.4. Estimación costes del proyecto	62
3.5. Plan de contingencias para los riesgos detectados	63
4.1. CU1 - Login	67
4.2. CU2 - Logout	67
4.3. CU3 - Registrarse	67
4.4. CU4 - Obtener información de la casa	68
4.5. CU5 - Contactar con gestor	68
4.6. CU6 - Obtener localización	68
4.7. CU7 - Obtener fiestas y gastronomía	69
4.8. CU8 - Obtener recomendaciones turísticas	69
4.9. CU9 - Obtener reseñas	69
4.10. CU10 - Obtener actividades	70
4.11. CU11 - Obtener fechas disponibles	70
4.12. CU12 - Reservar fechas	71
4.13. CU13 - Obtener clima	71
4.14. CU14 - Obtener tarifas	71
4.15. CU15 - Dejar reseña	72
4.16. CU16 - Consultar reservas pendientes	72
4.17. CU17 - Confirmar reserva	72
4.18. CU18 - Denegar reserva	73
4.19. CU19 - Raspar datos web	73
4.20. CU20 - Extraer datos actividades	73

4.21. CU21 - Extraer datos de redes sociales	74
6.1. Pruebas de casos de uso	152
6.1. Pruebas de casos de uso	155
7.1. Coste real de amortización de materiales para el desarrollo del proyecto . .	171
7.2. Costes reales del proyecto	172

Índice de listados de código

6.1.	Fichero de endpoints de creación de usuarios <code>SignupController.java</code>	102
6.2.	Fichero de servicio de creación de usuarios <code>SignupService.java</code>	102
6.3.	Fichero de configuración de seguridad <code>WebSecurityConfig.java</code>	103
6.4.	Fichero controlador de reservas <code>ReservaController.java</code>	105
6.5.	Fichero de servicio de reservas <code>ReservaService.java</code>	106
6.6.	Endpoints para la gestión de reseñas <code>ApplicationController.java</code>	110
6.7.	Servicio de gestión de reseñas <code>ReviewService.java</code>	110
6.8.	Controlador de contacto <code>ApplicationController.java</code>	111
6.9.	Servicio de contacto <code>ContactService.java</code>	111
6.10.	Definición de la tabla <code>reservas</code> en <code>schema.sql</code>	112
6.11.	Definición de la tabla <code>reviews</code> en <code>schema.sql</code>	112
6.12.	Definición de las tablas <code>users</code> , <code>roles</code> y <code>user_role</code> en <code>schema.sql</code>	113
6.13.	Fragmento del controlador de extracción de recursos <code>ExtraccionController.java</code>	114
6.14.	Servicio para la gestión de recursos <code>RecursoService.java</code>	114
6.15.	Modelo de documento MongoDB para recursos turísticos <code>Recurso.java</code>	116
6.16.	Modelo auxiliar de resultados de scraping <code>ScraperResult.java</code>	117
6.17.	Repositorio para la persistencia en MongoDB <code>RecursoRepository.java</code>	117
6.18.	Controlador principal para las publicaciones <code>MediaController.java</code>	118
6.19.	Servicio de extracción y almacenamiento de publicaciones <code>MediaService.java</code>	118
6.20.	Modelo de documento MongoDB para los medios <code>Media.java</code>	121
6.21.	Repositorio para la persistencia en MongoDB <code>MediaRepository.java</code>	122
6.22.	Fragmento del controlador de datos meteorológicos <code>WeatherController.java</code>	122
6.23.	Servicio para la gestión de datos meteorológicos <code>WeatherService.java</code>	123
6.24.	Modelo de documento MongoDB para datos meteorológicos <code>WeatherData.java</code>	126
6.25.	Repositorio de datos meteorológicos <code>WeatherDataRepository.java</code>	126
6.26.	Dockerfile del servicio de la aplicación principal <code>Dockerfile</code>	136
6.27.	PersistentVolume de PostgreSQL	136
6.28.	PersistentVolumeClaim de PostgreSQL	136
6.29.	PersistentVolume de MongoDB	137

6.30. PersistentVolumeClaim de MongoDB	137
6.31. PersistentVolume del token de Instagram	137
6.32. PersistentVolumeClaim del token de Instagram	137
6.33. Deployment de PostgreSQL	138
6.34. Service de PostgreSQL	139
6.35. Deployment de MongoDB	139
6.36. Service de MongoDB	140
6.37. Deployment de ElRinconDeEva	140
6.38. Service de ElRinconDeEva	141
6.39. Deployment de publicacionesapi	141
6.40. Service de publicacionesapi	142
6.41. Deployment de tiempo	143
6.42. Service de tiempo	143
6.43. Deployment de extraccion	144
6.44. Service de extraccion	144
6.45. Recurso Ingress para el acceso HTTPS	145
A.1. Fichero de filtro de autenticación <code>CustomAuthenticationFilter.java</code> . . .	187
A.2. Fichero de filtro de autorización <code>CustomAuthorizationFilter.java</code> . . .	187
A.3. Fichero de servicio para enviar emails <code>EmailService.java</code>	188
A.4. Repositorio de reservas <code>ReservaRepository.java</code>	189
A.5. Clase del dominio <code>Reserva.java</code>	190
A.6. Repositorio de reseñas <code>ReviewRepository.java</code>	192
A.7. Clase del dominio <code>Review.java</code>	192
A.8. Repositorio de usuarios <code>UserRepository.java</code>	194
A.9. Repositorio de roles <code>UserRepository.java</code>	194
A.10. Repositorio de relacion roles y usuarios <code>UserRepository.java</code>	194
A.11. Clase del dominio <code>MyUser.java</code>	195
A.12. Clase del dominio <code>UserRole.java</code>	198
A.13. Clase del dominio <code>Role.java</code>	198
A.14. Servicio de scraping de sitios web turísticos <code>ScraperService.java</code>	199
A.15. Servicio encargado de refrescar el token de acceso a la API de Instagram <code>InstagramTokenService.java</code>	201
A.16. Código del componente <code>AdministradorComponent</code>	201
A.17. Método <code>onSubmitReview</code> en el componente <code>InicioComponent</code>	204
A.18. Lógica del componente <code>LaCasaComponent</code>	205
A.19. Lógica del componente <code>ReservaComponent</code>	206

A.20.Lógica del componente ComoLlegarComponent	212
A.21.Lógica del componente RecomendacionesComponent	214
A.22.Lógica del componente EntornoComponent	215
A.23.Lógica del componente LoginComponent	216
A.24.Lógica del componente RegisterComponent	217
A.25.Lógica del servicio de autenticación AuthService	218
A.26.Tabla de visualización de reservas en administrador.component.html . .	220
A.27.Sección de Inicio en inicio.component.html	222
A.28.Sección de información la casa rural en la-casa.component.html	225
A.29.Sección de reservas de la casa rural en reservas.component.html	228
A.30.Sección como llegar en como-llegar.component.html	231
A.31.Sección de recomendaciones en recomendaciones.component.html	232
A.32.Sección de entorno en entorno.component.html	234
A.33.Sección de login en login.component.html	235
A.34.Sección de registro en register.component.html	238
A.35.Dockerfile del backend de la aplicación para la API de extracción web . .	240
A.36.Dockerfile del backend de la aplicación para la API de publicaciones . . .	240
A.37.Dockerfile del backend de la aplicación para la API de tiempo	240
A.38.Test unitario de la clase ReservaService en ReservaServiceTest.java .	240
A.39.Test unitario de la clase ReviewService en ReviewServiceTest.java . .	241
A.40.Test unitario de la clase UserService en UserServiceTest.java	243
A.41.Test unitario de la clase SignupService en SignupServiceTest.java . .	244
A.42.Test unitario de la clase MediaService en MediaServiceTest.java	246
A.43.Test unitario de la clase WeatherService en WeatherServiceTest.java .	246
A.44.Test unitario de la clase RecursoService en RecursoServiceTest.java .	248
A.45.Test funcional End to End (E2E) del menú de gestión de reservas	248
A.46.Test funcional E2E del menú de login	250
A.47.Test funcional E2E del menú de registro de usuario	250
A.48.Test funcional E2E del menú de reservas	251

En general está muy bien, sólo algún pequeño error que debes subsanar

Capítulo 1

Introducción

→ formato correcto del acrónimo

1.1. Introducción

En los últimos años, el sector del turismo rural ha experimentado un notable crecimiento, motivado por el interés creciente de los viajeros por destinos más tranquilos, naturales y alejados del turismo de masas, según el **Instituto Nacional de Estadística (INE)** [1]. La Figura 1.1 muestra cómo este crecimiento se intensifica sobre todo en las épocas menos señaladas del año, lo que influye en el interés de hacer visible y accesible este tipo de turismo a todo el mundo. Este contexto ha impulsado la digitalización de los servicios ofrecidos por las casas rurales, permitiendo mejorar la experiencia del huésped y optimizar la gestión por parte de los propietarios.

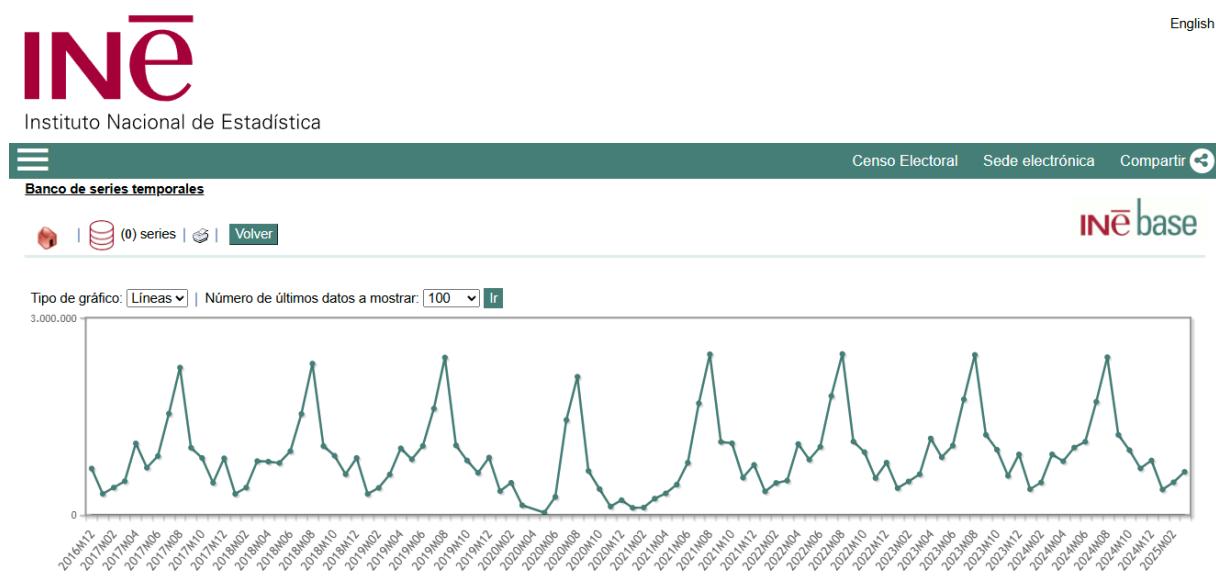


Figura 1.1: Crecimiento del turismo rural en España.

En el presente TFM, se va a realizar el desarrollo de una plataforma web para la gestión de una casa rural, ya que, realizando una búsqueda en Google Trends [2] sobre las poblaciones cercanas a las instalaciones de esta casa rural, se observa en la Figura 1.2 que existe interés por el turismo rural en la zona, especialmente en los meses de verano y durante las festividades. Por lo que se confirma la necesidad de contar con una herramienta que permita gestionar de forma eficiente la promoción del alojamiento.

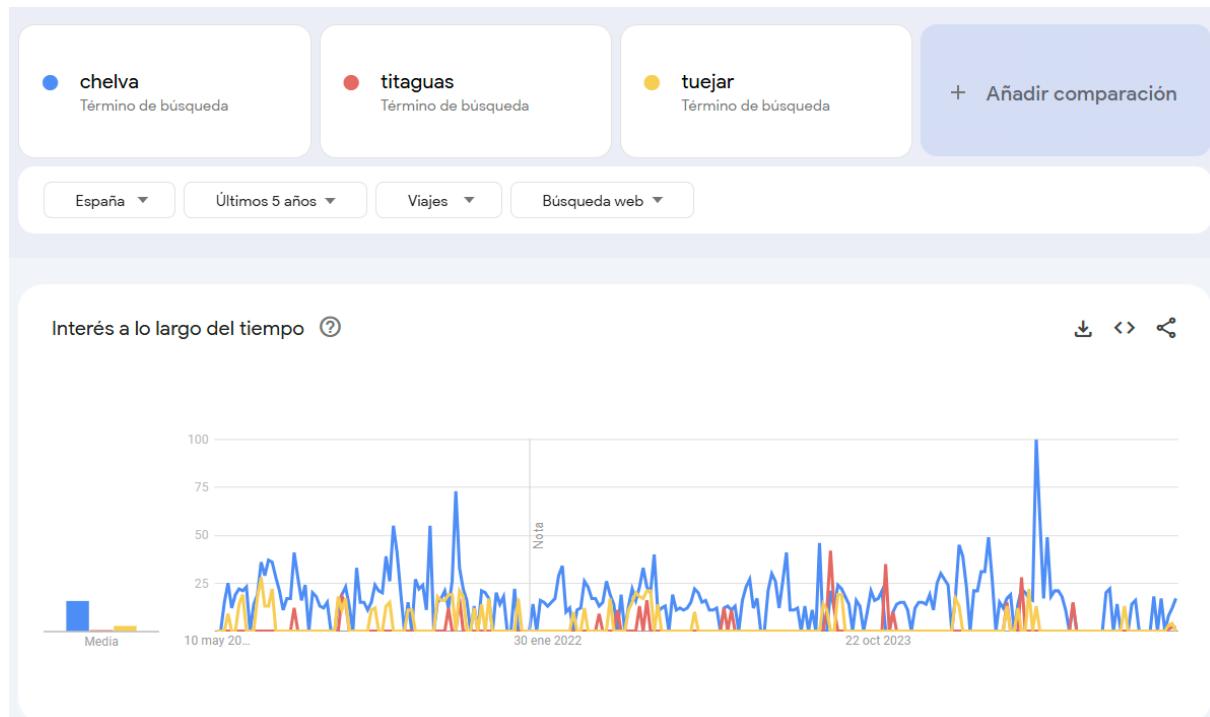


Figura 1.2: Interés por el turismo rural en la zona de la casa rural.

Concretamente, el desarrollo trata la creación de una plataforma web destinada a facilitar la administración y promoción de una casa rural. Esta aplicación, accesible desde dispositivos móviles y navegadores web, tiene como objetivo integrar diferentes funcionalidades: desde la reserva de estancias hasta la sugerencia de actividades turísticas personalizadas, aprovechando tecnologías modernas como los **microservicios**, bases de datos en la nube y sistemas de recomendaciones inteligentes basados en preferencias del usuario.

La aplicación propuesta incluye múltiples módulos, tales como un calendario de disponibilidad (**calendar**), recomendaciones contextuales basadas en geolocalización, formularios de contacto y un sistema de reseñas. Todo esto se implementará bajo una arquitectura modular y escalable, siguiendo los principios de diseño orientado a servicios.

Además, se han tenido en cuenta aspectos no funcionales relevantes como la accesibilidad, seguridad de los datos y compatibilidad entre dispositivos. Para garantizar la calidad del software, se utilizarán herramientas como **Docker** y **Kubernetes** para el despliegue y orquestación de contenedores, permitiendo una alta disponibilidad y una escalabilidad horizontal efectiva.

Con este trabajo se pretende no solo diseñar una herramienta útil para los propietarios de casas rurales, sino también aportar una visión sobre cómo las tecnologías emergentes pueden transformar el turismo rural, haciéndolo más sostenible, inclusivo y personalizado.

Motivación

El presente **TFM** está motivado por diversos factores, tanto personales como técnicos. La motivación principal surge a raíz de una necesidad real: un familiar del autor ha puesto en marcha una casa rural y requiere una aplicación web que le permita gestionar de forma eficiente tanto las reservas como la promoción del alojamiento. Esta situación concreta

ofrece una oportunidad ideal para aplicar los conocimientos adquiridos durante el máster en el desarrollo de una solución tecnológica útil, funcional y con aplicación inmediata en un contexto real.

A nivel técnico, este proyecto permite consolidar competencias clave en el ámbito del desarrollo de software moderno, como el diseño de arquitecturas de microservicios, el uso de tecnologías web actuales como Angular y Spring Boot, y la implementación de soluciones escalables y portables mediante contenedores Docker y orquestación con Kubernetes. Estas herramientas permiten construir un sistema distribuido, adaptable a distintos entornos de ejecución y preparado para crecer en funcionalidad en el futuro.

Además, existe un interés adicional en explorar la integración de fuentes de datos externas —como servicios meteorológicos o de eventos turísticos— que aporten valor añadido al usuario final. También se aborda el uso combinado de bases de datos relacionales y no relacionales, con el objetivo de optimizar el almacenamiento y procesamiento de información en función de su naturaleza.

En conjunto, este trabajo no solo representa un reto técnico y una oportunidad de aprendizaje, sino también una propuesta de valor real en el ámbito del turismo rural. El desarrollo de este sistema pretende demostrar cómo la tecnología puede acercarse a entornos tradicionalmente alejados de la digitalización, mejorando su competitividad, sostenibilidad y capacidad de adaptación a las nuevas demandas del mercado.

1.2. Objetivos

1.2.1. Objetivo general *principal*

Aplicar los conocimientos adquiridos en el Máster de [Tecnologías Web, Aplicaciones Móviles y Servicios \(TWCAM\)](#) a la creación de una plataforma web para una casa rural que permita gestionar reservas, ofrecer información turística y mejorar la experiencia del usuario mediante funcionalidades interactivas e integraciones externas basadas en microservicios.

1.2.2. Objetivos específicos

- Facilitar la consulta de la disponibilidad y la reserva en línea del alojamiento.
- Mostrar información relevante sobre la casa rural y su entorno.
- Integrar contenidos multimedia y datos procedentes de fuentes externas.
- Permitir la gestión de usuarios con distintos roles y funcionalidades asociadas.
- Asegurar compatibilidad con distintos dispositivos y navegadores web.
- Incorporar valoraciones, actividades y recomendaciones personalizadas al entorno.
- Garantizar la seguridad, escalabilidad y accesibilidad del sistema.

1.3. Organización de la memoria

Capítulo 1: Introducción

Este capítulo presenta el contexto general del proyecto, así como la motivación personal y técnica que ha llevado a su desarrollo. Se definen el objetivo general y los objetivos específicos que se persiguen con este trabajo. Finalmente, se expone la estructura de la memoria, anticipando brevemente el contenido de los capítulos posteriores.

Capítulo 2: Estado del arte

Se realiza un análisis de aplicaciones existentes similares a la propuesta. Además, se evalúan distintas tecnologías disponibles para el desarrollo del proyecto, incluyendo herramientas de frontend, backend, bases de datos, ~~contenedores~~ y servicios de nube, justificando las elecciones tomadas.

en la
nube

↳ componentes basados en
contenedores

Capítulo 3: Requisitos, especificaciones, coste, riesgos y viabilidad

Este capítulo recoge la descripción funcional y no funcional del sistema propuesto, así como las especificaciones técnicas. También incluye una planificación temporal, una estimación de costes, el análisis de riesgos potenciales y un estudio de viabilidad del proyecto.

Capítulo 4: Análisis

Se lleva a cabo un análisis detallado del sistema mediante diagramas de casos de uso, clases de primer nivel y secuencia. Esta fase permite establecer una base sólida para el diseño posterior del sistema, identificando los principales actores, funcionalidades y relaciones entre componentes.

Capítulo 5: Diseño

Se describe la arquitectura del sistema, dividiéndolo en subsistemas según su funcionalidad: recursos web, publicaciones y previsión del tiempo. Se presentan los diagramas de clases y el modelo de datos, así como los detalles del despliegue y organización de los componentes del sistema.

Capítulo 6: Implementación y pruebas

Este capítulo se centra en la implementación técnica de los diferentes módulos del sistema, incluyendo el desarrollo del Frontend, Backend y la dockerización para su des-

pliegue. Además, se detallan las pruebas realizadas, tanto unitarias como funcionales, para asegurar el correcto funcionamiento del sistema.

Capítulo 7: Conclusiones

Se presentan las conclusiones obtenidas tras la realización del proyecto, así como una revisión de los costes. Finalmente, se propone una línea de trabajo futuro para continuar mejorando el sistema o extender sus funcionalidades.

El trabajo del capitán también es breve,
para leer errores o corregir y no me convienen
algunas cosas como la resolución de los jueces.

Capítulo 2

Estado del arte

2.1. Análisis de aplicaciones similares

En el contexto de la gestión de casas rurales, así como de las actividades y eventos cercanos a éstas, existen diversas aplicaciones web destacadas que ofrecen funcionalidades avanzadas para la administración y promoción de propiedades. En este apartado se presenta un análisis comparativo de algunas de las plataformas más relevantes, destacando sus principales fortalezas y debilidades.

2.1.1. Airbnb

Airbnb [3] es una de las aplicaciones de referencia en el sector de alquiler de viviendas y alojamientos turísticos. Esta plataforma ofrece una interfaz amigable y moderna que facilita tanto la búsqueda de propiedades como la gestión de reservas. Además, dispone de herramientas de marketing como la promoción de ofertas y la personalización de sugerencias de viaje, basadas en las preferencias y el historial de los usuarios.

Puntos fuertes:

- Interfaz intuitiva y fácil de usar, lo que mejora la experiencia del usuario.
- Amplias funcionalidades de recomendación y personalización, adaptadas a los gustos y necesidades de los usuarios.
- Sistema de experiencias que permite a los anfitriones ofrecer actividades locales, enriqueciendo la estancia del visitante. La Figura 2.1 muestra un ejemplo de cómo se presentan estas experiencias en la plataforma.
- Integración con múltiples plataformas de pago, aumentando la comodidad para el cliente.

Puntos débiles:

- Comisiones elevadas para los anfitriones, lo cual puede resultar una desventaja para pequeños propietarios de casas rurales.
- Requiere de una conexión constante para la sincronización de datos, lo cual puede no ser óptimo en zonas rurales con cobertura limitada.

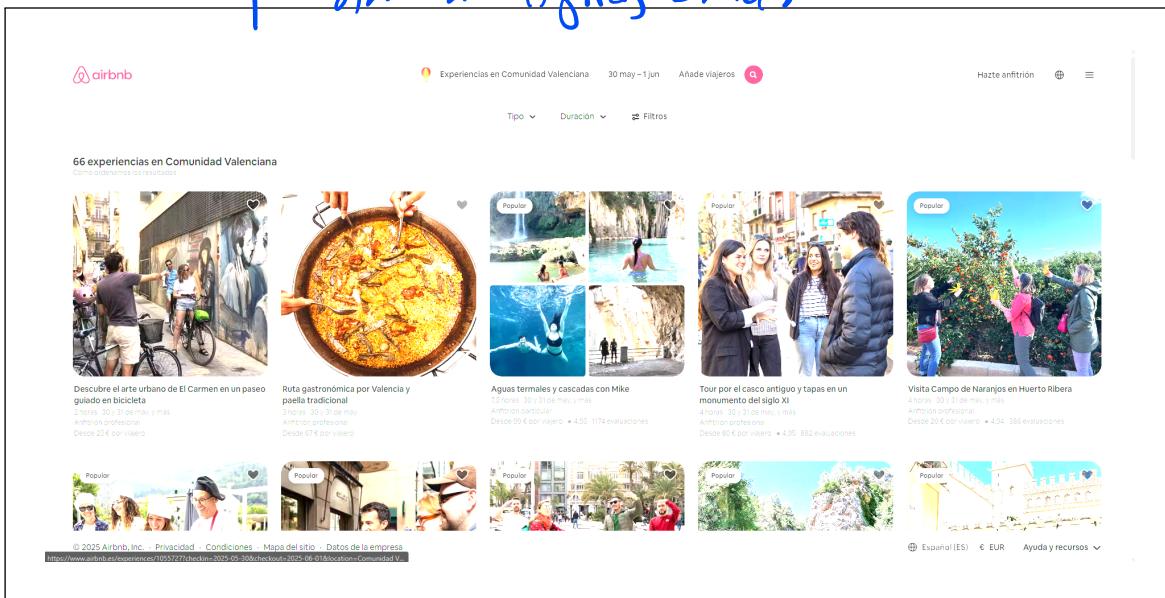


Figura 2.1: Ejemplo de experiencias ofrecidas en Airbnb.

- Seguridad y privacidad del usuario, ya que existe el riesgo de compartir demasiada información personal en línea.

2.1.2. Booking.com

Booking.com [4] es otra plataforma ampliamente utilizada para la reserva de alojamientos. Ofrece a los propietarios de casas rurales una plataforma robusta para la gestión de reservas y pagos.

Puntos fuertes:

- Amplia visibilidad en el mercado, atrayendo una gran cantidad de usuarios.
- Sistema de reseñas con filtros que permiten valorar distintos tipos de experiencia y consultar opiniones relevantes según las fechas deseadas. La Figura 2.2 muestra un ejemplo de cómo se presentan estas reseñas en la plataforma.
- Integración de un sistema de mapas y recomendaciones turísticas locales. La Figura 2.3 muestra un ejemplo de cómo se presentan estas recomendaciones en la plataforma.
- Permite la gestión de propiedades múltiples desde un mismo perfil de anfitrión.

Puntos débiles:

- Costos de servicio elevados y comisiones en cada reserva.
- Dificultades en la personalización de la experiencia del usuario, limitando las opciones de personalización para el anfitrión.
- Falta de opciones de promoción específicas para propietarios de casas rurales, comparado con otras plataformas.

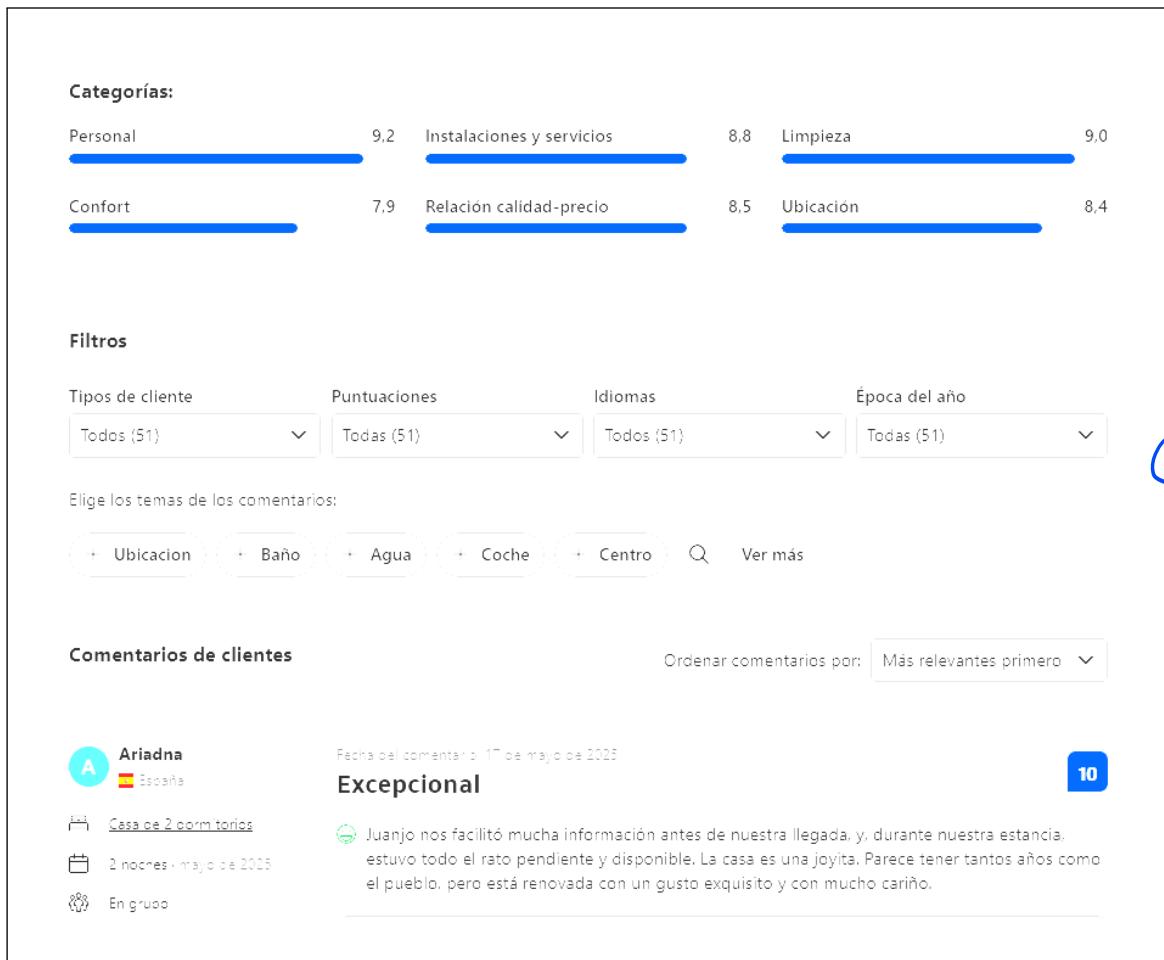


Figura 2.2: Ejemplo de reseñas en Booking.com.

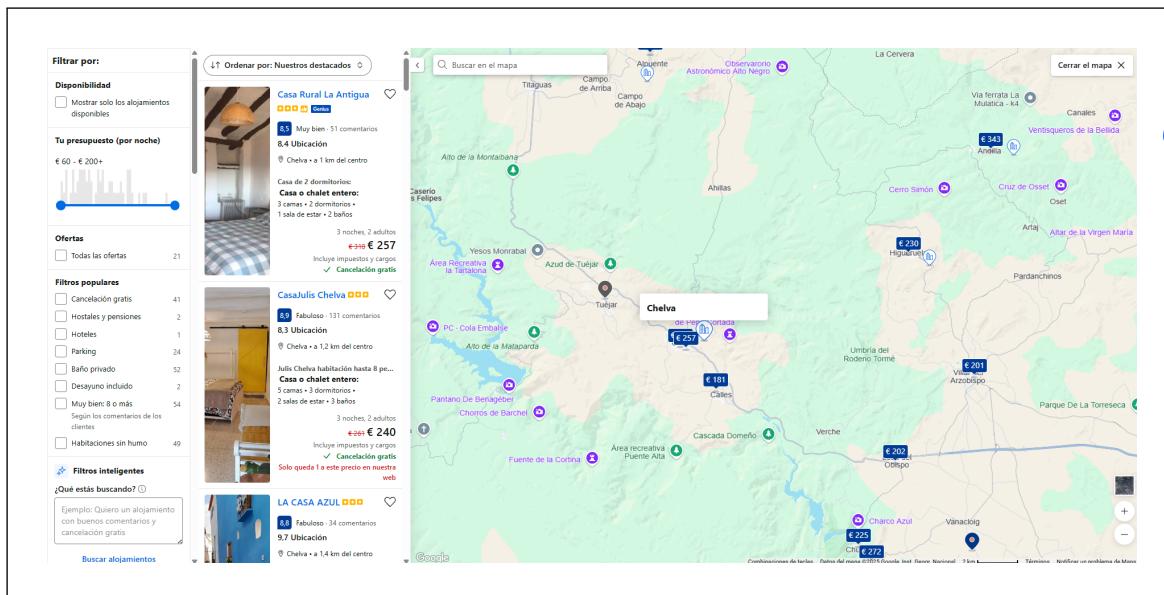


Figura 2.3: Ejemplo de recomendaciones turísticas en Booking.com.

2.1.3. Ruralka

Ruralka [5] es una plataforma específica para el alquiler de casas rurales en España, enfocada en ofrecer experiencias únicas en entornos naturales.

Puntos fuertes:

- Enfoque en la experiencia rural, destacando actividades al aire libre y experiencias culturales personalizadas por el anfitrión.
- Sistema sencillo de servicios adaptado a la experiencia buscada en casas rurales. La Figura 2.4 muestra un ejemplo de cómo se presentan estos servicios en la plataforma.
- Selección cuidada de propiedades que garantiza una calidad estandarizada.
- Proporciona una mayor visibilidad a pequeños propietarios rurales, con tarifas de servicio más bajas en comparación con plataformas generalistas.

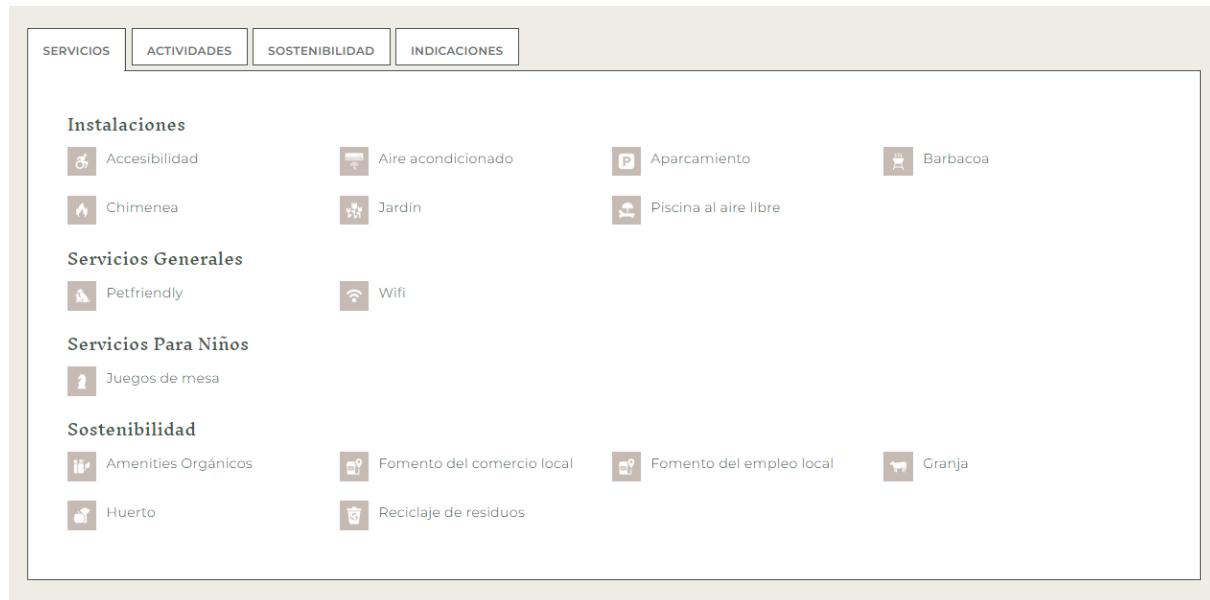


Figura 2.4: Ejemplo de servicios ofrecidos en Ruralka.

Puntos débiles:

- Menor alcance en términos de tráfico y usuarios comparado con grandes plataformas internacionales.
- Limitación en opciones de pago y personalización de propiedades, lo cual puede ser un inconveniente para algunos usuarios.

2.1.4. Komoot

Komoot [6] es una plataforma especializada en la planificación y descubrimiento de rutas para actividades al aire libre como senderismo, ciclismo, running o excursiones en la naturaleza. Se ha consolidado como una de las aplicaciones más utilizadas en Europa

para explorar rutas personalizadas y obtener recomendaciones según el nivel de dificultad, el tipo de terreno o el paisaje.

Puntos fuertes:

- Amplia base de datos de rutas, mapas y puntos de interés generados por la comunidad de usuarios.
- Posibilidad de personalizar rutas según actividad, condición física y tipo de terreno. Esto se puede observar en la Figura 2.5, donde se muestran ejemplos de rutas personalizadas y cómo podemos planificarlas.
- Navegación guiada por voz incluso sin conexión a Internet, ideal para zonas rurales con cobertura limitada.
- Integración con relojes GPS y dispositivos de ciclismo para registrar y compartir rutas.

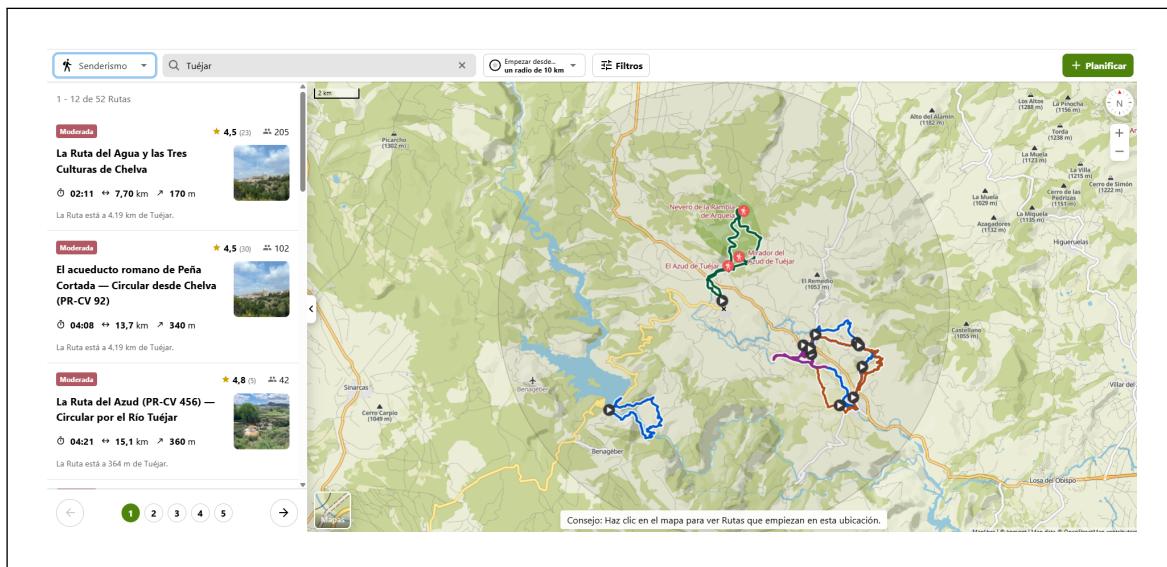


Figura 2.5: Ejemplo de rutas personalizadas en Komoot.

Puntos débiles:

- Algunas funcionalidades avanzadas requieren suscripción de pago (por ejemplo, mapas detallados offline por región).
- Interfaz menos atractiva en comparación con otras aplicaciones de planificación de viajes.
- Menor presencia en países no europeos, lo que puede limitar la oferta de rutas si la casa rural se encuentra en una región menos explorada por la comunidad.

Este análisis de aplicaciones muestra tanto las oportunidades como los desafíos que estas plataformas enfrentan en el mercado actual. Nuestra aplicación de gestión y marketing para una casa rural busca combinar las mejores prácticas de estas plataformas —alojamiento, actividades al aire libre y eventos culturales— y adaptarlas en una solución integral que facilite tanto la visibilidad del alojamiento como la promoción de

experiencias completas. Al ofrecer una gestión automatizada y sin necesidad de conocimientos técnicos, se potencia la autonomía del propietario y se mejora la experiencia del visitante desde el descubrimiento hasta la participación en actividades locales.

2.2. APIs y plataformas externas a utilizar

  **Objetivo:**

Para el correcto funcionamiento del sistema de gestión de la casa rural, se integrarán diversas **APIs** y plataformas externas. Estas fuentes permitirán extraer información turística, contenidos multimedia y datos meteorológicos relevantes para enriquecer la experiencia del usuario. A continuación, se detallan las principales herramientas seleccionadas, su funcionalidad y un análisis de sus ventajas e inconvenientes.

2.2.1. Portal turístico de Tuéjar

El portal turístico oficial de Tuéjar [7] ofrece información actualizada sobre rutas, actividades y patrimonio natural y cultural del municipio. Este sitio se utilizará como fuente para el raspado automático de contenidos relacionados con eventos locales y lugares de interés. En la Figura 2.6 se muestra un ejemplo de cómo se presenta la información turística en la web de Tuéjar.

Puntos fuertes:

- Información específica y localizada sobre el entorno rural.
- Actualización frecuente de eventos locales y propuestas turísticas.
- Fuente confiable y mantenida por el ayuntamiento o entes locales.

Puntos débiles:

- Estructura web variable que puede dificultar la automatización del raspado.
- Falta de API oficial, lo que obliga a depender del web scraping.

2.2.2. Portal turístico de Chelva

El sitio web de turismo de Chelva [8] se utilizará como fuente secundaria para obtener eventos, fiestas populares, rutas y otras actividades cercanas a Tuéjar. El objetivo es ampliar el abanico de recomendaciones disponibles para los visitantes. En la Figura 2.7 se muestra un ejemplo de cómo se presentan y están accesibles de forma sencilla monumentos, espacios naturales, aldeas y fiestas en la web de Chelva.

Puntos fuertes:

- Información turística de un municipio cercano, complementaria a la de Tuéjar.
- Contenido visual útil para la promoción de actividades (imágenes, carteles de eventos, etc.).

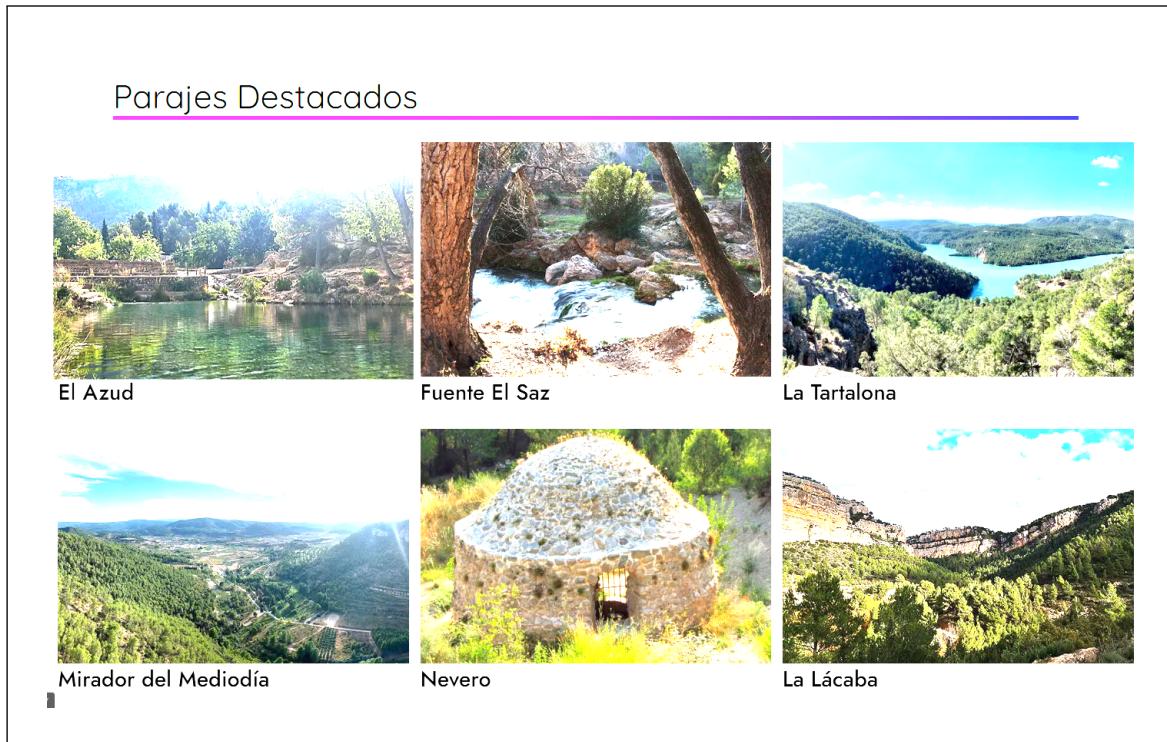


Figura 2.6: Ejemplo de información turística extraída de la web de Tuéjar.

Puntos débiles:

- Dificultad de estructuración para el raspado, ya que su diseño web no sigue patrones estándar.
- Contenido limitado a eventos puntuales.



Figura 2.7: Ejemplo de rutas y eventos extraídos de la web de turismo de Chelva.

2.2.3. Web oficial del Ayuntamiento de Tuéjar

La web institucional del Ayuntamiento [9] contiene convocatorias, noticias municipales y eventos oficiales que pueden ser relevantes para los usuarios de la plataforma, especialmente en el contexto de fiestas patronales o ferias. Como se observa en la Figura 2.8, la web del Ayuntamiento de Tuéjar ofrece información sobre eventos locales y festividades de las cuales extraer los recursos para la plataforma.

Puntos fuertes:

- Información oficial y validada por el consistorio.
- Acceso a contenido de interés general para residentes y visitantes.

Puntos débiles:

- Página orientada a trámites administrativos, no siempre con contenido turístico directo.
- Variabilidad en la estructura del contenido que complica el scraping sistemático.

- [17 de Enero: Hogueras de San Antón](#)
- [Febrero: Carnavales](#)
- [Fin de semana posterior a San José: Fallas de Tuéjar](#)
- [Semana Santa](#)
- [30 de Abril: Noche de los Mayos](#)
- [Semana del 15 de Agosto: Fiestas Patronales](#)
- [13 de Noviembre: San Diego](#)
- [8 de Diciembre: La Purísima](#)

FIESTAS GORDAS: [Fiesta de Interés Turístico Autonómico de la Comunitat Valenciana](#). Celebradas los años terminados en 0 y en 5, donde destaca el "Entramoro" y la "Rodá de la Bandera".



Figura 2.8: Contenido institucional relevante en la web del Ayuntamiento de Tuéjar.

2.2.4. API de Instagram Graph

La *Instagram Graph API* [10] permitirá obtener publicaciones relacionadas con la casa rural, eventos locales y fotografías etiquetadas en la zona. Será utilizada para enriquecer la sección multimedia de la plataforma, conectando contenido generado por el gestor sobre su perfil de Instagram.

Puntos fuertes:

- Acceso a publicaciones recientes mediante hashtags o localizaciones.
 - Contenido auténtico y visual, generado por el gestor.
 - Posibilidad de integración con galerías dinámicas en la plataforma.

Puntos débiles:

- Requiere autenticación y permisos para acceder a ciertos datos.
 - Cambios frecuentes en las políticas de uso y limitaciones en las llamadas a la API.

2.2.5. API meteorológica Open-Meteo

La API (Application Programming Interface) de Open-Meteo [11] será la fuente principal para obtener datos meteorológicos en la plataforma. Esta API (Application Programming Interface) pública ofrece información precisa y en tiempo real, sin necesidad de autenticación mediante token, lo que simplifica su integración y uso en aplicaciones automatizadas.

Puntos fuertes:

- No requiere tokenización ni registro, facilitando el acceso directo a los datos.
 - Permite obtener datos históricos de predicción, lo que posibilita aplicar modelos propios para prever condiciones futuras en función de patrones pasados.
 - Proporciona estimaciones muy precisas basadas en modelos meteorológicos de última generación y ajustadas a la ubicación exacta mediante coordenadas geográficas.
 - Respuesta en formato JSON clara y fácilmente integrable con el sistema.

Puntos débiles:

- Aunque ofrece predicciones detalladas, no incluye avisos oficiales sobre fenómenos adversos como lo hace AEMET.

2.2.6. API meteorológica: AEMET

La API (Application Programming Interface) de Agencia Estatal de Meteorología (AEMET) [12] fue evaluada como posible alternativa, ya que proporciona datos oficiales en el ámbito nacional. Sin embargo, se descartó como fuente principal por las siguientes razones:

- La ubicación de la casa rural no cuenta con una estación meteorológica de AEMET cercana, por lo que los datos disponibles son menos representativos de las condiciones reales.
- Requiere autenticación mediante token y tiene limitaciones de uso que complican su automatización continua.

2.3. Evaluación de tecnologías

Para el desarrollo de aplicaciones web, existen múltiples tecnologías, Frameworks y bases de datos disponibles. A continuación, se describen algunas de las alternativas más comunes, destacando sus características y diferencias. Finalmente, se justifica la elección de Angular [13], Spring Boot [14], Docker [15] y Kubernetes (K8S) [16] como las herramientas de desarrollo principales para esta aplicación, junto con el uso de distintas bases de datos adaptadas a cada caso de uso que se detallará más adelante. La principal motivación para elegir estas tecnologías es continuar y profundizar en los conocimientos adquiridos en el programa del máster.

2.3.1. Frontend

Angular [13] es un Framework de desarrollo de aplicaciones web de Google basado en TypeScript, que permite construir interfaces de usuario dinámicas y altamente interactivas. Comparado con otras tecnologías como React o Vue.js, Angular ofrece un ecosistema completo que incluye enrutamiento, inyección de dependencias y herramientas de pruebas.

Ventajas:

- Arquitectura robusta basada en componentes y módulos.
- Excelente rendimiento y herramientas para el desarrollo de aplicaciones a gran escala.
- Comunidad activa y soporte continuo de Google.

Desventajas:

- Curva de aprendizaje pronunciada debido a su estructura compleja.
- Tamaño relativamente grande del Framework en comparación con alternativas como Vue.js.

React [17] es una biblioteca de JavaScript desarrollada por Meta, centrada en la construcción de interfaces de usuario a través de componentes. A diferencia de Angular, React es una biblioteca más ligera y flexible, aunque se requiere la integración de herramientas adicionales para lograr funcionalidades completas.

Ventajas:

- Flexibilidad para integrarse con otras bibliotecas y herramientas.

- Alta velocidad de renderizado gracias al uso de un DOM virtual.
- Gran popularidad y extensa comunidad, con abundante documentación y recursos.

Desventajas:

- Mayor necesidad de configuración e integración de herramientas externas (por ejemplo, enrutamiento y gestión de estado).
- La flexibilidad y falta de estructura puede resultar confusa para proyectos grandes.

¿Qué son los beneficios?

2.3.2. Backend

Spring Boot [14] es un marco de trabajo para el desarrollo de aplicaciones **Backend** en Java. A diferencia de alternativas como Express (Node.js) o Django (Python), Spring Boot permite crear microservicios escalables y altamente configurables.

Ventajas:

- Gran capacidad de integración con sistemas empresariales.
- Alto rendimiento en aplicaciones de gran escala y fácil escalabilidad.
- Amplio ecosistema de herramientas y bibliotecas, como Spring Data para la gestión de bases de datos y Spring Security para la autenticación y autorización.
- Amplio soporte para programación reactiva y arquitectura de microservicios. Esto permitirá la utilización de extensiones como Spring WebFlux [18] para construir aplicaciones reactivas y eficientes.

Desventajas:

- Consumo de memoria elevado, lo cual puede ser un inconveniente en aplicaciones de recursos limitados.
- Complejidad en la configuración inicial, especialmente para desarrolladores principiantes.

Django [19] es un **Framework** de desarrollo web en Python que permite construir aplicaciones web de manera rápida y eficiente, utilizando una arquitectura basada en el patrón **Modelo Vista Controlador (MVC)**. A diferencia de otros **Frameworks** como Flask[20] (Python) o Express (Node.js), Django proporciona una gran cantidad de herramientas y componentes listos para usar, facilitando el desarrollo de aplicaciones complejas.

Ventajas:

- Gran cantidad de funcionalidades integradas, incluyendo autenticación, administración y protección contra ataques comunes, lo que acelera el desarrollo.
- Arquitectura escalable y organizada, ideal para aplicaciones de mediana y gran escala.

- Fuerte comunidad de soporte y documentación exhaustiva, que facilita la resolución de problemas y el aprendizaje.

Desventajas:

- Puede ser excesivo para aplicaciones pequeñas o simples, debido a su estructura robusta.
- Curva de aprendizaje para aquellos que no están familiarizados con Python o con el enfoque MVC de Django.
- Limitaciones en la personalización de ciertas partes del Framework, lo cual puede ser un inconveniente en proyectos altamente específicos.

Express(Node.js) [21] es un Framework minimalista para Node.js que permite desarrollar aplicaciones Backend con JavaScript o TypeScript. Su simplicidad y capacidad para construir APIs Representational State Transfer (REST) lo hacen ideal para aplicaciones pequeñas y medianas.

Ventajas:

- Ligero y rápido, ideal para aplicaciones que requieren una alta velocidad de respuesta.
- Ecosistema de módulos y paquetes que facilita la ampliación de funcionalidades.
- Compatibilidad total con JavaScript en el Frontend, lo que permite un desarrollo full-stack unificado.

Desventajas:

- No ofrece tantas funcionalidades integradas como Frameworks más robustos como Spring Boot, lo cual requiere el uso de bibliotecas adicionales.
- Menor rendimiento en aplicaciones de alta carga comparado con alternativas basadas en Java.

¿ Cuál es la conclusión ?

2.3.3. Contenerización y Orquestación

Docker [15] es una tecnología de contenedorización que permite empaquetar aplicaciones y sus dependencias en contenedores ligeros, portables y aislados. Gracias a su amplio ecosistema y facilidad de uso, se ha convertido en el estándar de facto en entornos de desarrollo y despliegue moderno. A diferencia de soluciones como LXC [22], Docker proporciona una capa de abstracción más amigable al desarrollador, centrada en la construcción y distribución de aplicaciones mediante imágenes reutilizables.

Ventajas:

- Portabilidad entre distintos entornos gracias al empaquetado en imágenes.
- Rapidez en el despliegue y facilidad para replicar entornos.

- Amplia comunidad y disponibilidad de herramientas complementarias.

Desventajas:

- Puede consumir más recursos que soluciones de virtualización más ligeras como LXC.
- Requiere una adecuada configuración de seguridad para entornos de producción.

Kubernetes [16] (o [K8S](#)) es una plataforma de orquestación de contenedores que automatiza el despliegue, escalado y gestión de aplicaciones en contenedores. Se integra comúnmente con Docker como motor de contenedores, aunque también soporta otras alternativas. En comparación con soluciones más simples como Docker Swarm [\[23\]](#), Kubernetes ofrece un mayor control sobre redes, almacenamiento y políticas de autosanación.

Ventajas:

- Escalabilidad horizontal y alta disponibilidad mediante replicación automática.
- Orquestación avanzada con balanceo de carga, monitoreo y autoescalado.
- Gran ecosistema y soporte por parte de los principales proveedores cloud.

Desventajas:

- Curva de aprendizaje elevada debido a su complejidad.
- Requiere mayor inversión en infraestructura y conocimientos técnicos.

LXC (Linux Containers) [22] es una tecnología de virtualización a nivel de sistema operativo que permite ejecutar múltiples entornos Linux aislados dentro de un mismo kernel. A diferencia de Docker, que opera a un nivel más alto centrado en las aplicaciones, LXC ofrece un control más detallado sobre el entorno del sistema, siendo más parecido a una máquina virtual ligera.

Ventajas:

- Mayor control sobre los procesos y el entorno del sistema operativo.
- Menor sobrecarga de recursos al compartir el mismo kernel del host.

Desventajas:

- Menor facilidad de uso y adopción en comparación con Docker.
- Integración limitada con herramientas modernas de orquestación como [K8S](#).

Docker Swarm [23] es la solución de orquestación nativa de Docker, diseñada para desplegar y gestionar clústeres de contenedores de forma más simple que Kubernetes. Aunque ofrece una integración fluida con el ecosistema Docker, su uso ha disminuido frente al crecimiento y madurez de [K8S](#).

Ventajas:

- Sencillez de configuración y despliegue en comparación con Kubernetes.
- Integración directa con Docker CLI y Docker Compose.

Desventajas:

- Menor escalabilidad y robustez en entornos de producción complejos.
- Comunidad y soporte más reducidos, con menor ritmo de desarrollo.

¿ Aquí no hay conclusiones?

2.3.4. Bases de Datos

Para el desarrollo de la aplicación, es necesario almacenar datos en diferentes tipos de bases de datos, según el caso de uso específico. A continuación, se analizan las bases de datos NoSQL, destinadas al almacenamiento de contenido multimedia etiquetado; las bases de datos SQL, utilizadas para la gestión de autenticación y reservas; y las bases de datos en memoria, implementadas para optimizar el rendimiento en el acceso a datos temporales o frecuentemente consultados. Cabe destacar que todas las bases de datos comparadas son compatibles o tienen equivalencias con servicios de plataforma en la nube, tanto a nivel de [Infraestructure as a Service \(IaaS\)](#) como de [Platform as a Service \(PaaS\)](#), los cuales se compararán en el apartado siguiente.

Bases de datos NoSQL Para almacenar contenido multimedia y etiquetas de categorización, se evaluarán MongoDB y Cassandra.

MongoDB [24] es una base de datos orientada a documentos que almacena datos en formato BSON, ideal para aplicaciones que requieren flexibilidad en el esquema y escalabilidad horizontal.

Ventajas:

- Modelo de datos flexible y basado en documentos, ideal para almacenar contenido multimedia y metadatos asociados.
- Facilidad para realizar consultas complejas en estructuras de datos anidadas.
- Escalabilidad horizontal, con soporte nativo para fragmentación de datos.

Desventajas:

- Mayor consumo de memoria y almacenamiento.
- Rendimiento limitado en consultas a gran escala comparado con Cassandra.

Cassandra [25] es una base de datos distribuida orientada a columnas, diseñada para manejar grandes volúmenes de datos en múltiples nodos, ideal para aplicaciones de alta disponibilidad.

Ventajas:

- Escalabilidad masiva y resistencia a fallos, gracias a su arquitectura distribuida.
- Alta eficiencia en escritura y lectura de grandes cantidades de datos.

Desventajas:

- Curva de aprendizaje y configuración compleja.
- Menor flexibilidad en la estructura de datos en comparación con MongoDB.

Conclusión: En este proyecto, MongoDB resulta ser la elección más apropiada para almacenar contenido multimedia con etiquetas, debido a su flexibilidad.

Bases de datos SQL Para la gestión de autenticación y reservas, se evaluarán PostgreSQL[26] y MySQL[27], así como Oracle~~oracle~~ y MariaDB~~mariadb~~.

PostgreSQL [26] es una base de datos relacional con soporte avanzado de SQL y capacidades de extensibilidad.

Ventajas:

- Potente soporte para operaciones complejas y consultas avanzadas.
- Extensible y con un sólido soporte de integridad referencial y transacciones.

MySQL [27] es una base de datos relacional conocida por su simplicidad y eficiencia en el procesamiento de transacciones de lectura-escritura.

Ventajas:

- Amplia compatibilidad y facilidad de uso.
- Buen rendimiento para aplicaciones de lectura intensiva.

Oracle~~oracle~~ es una base de datos relacional empresarial, con características avanzadas orientadas a entornos críticos y escalables.

Ventajas:

- Gran capacidad para gestión de grandes volúmenes de datos y alta concurrencia.
- Funcionalidades avanzadas de seguridad y replicación.

MariaDB~~mariadb~~ es un fork de MySQL que ofrece mejoras en rendimiento y nuevas características manteniendo compatibilidad.

Ventajas:

- Mejoras en rendimiento y en almacenamiento en comparación con MySQL.

- Comunidad activa y desarrollo abierto.

Conclusión: Tanto PostgreSQL como MySQL cumplen con los requisitos de tablas sencillas para la gestión de autenticación y reservas, por lo que se puede optar por cualquiera. Además, la selección de estas bases de datos se ha realizado también pensando en el entorno de desarrollo backend utilizado durante el máster, donde se estudió la integración principalmente con PostgreSQL y MySQL, facilitando así la compatibilidad y el soporte durante el desarrollo. Por este motivo, Oracle y MariaDB, aunque válidas, no fueron seleccionadas para este proyecto.

Bases de datos en memoria Para mejorar el rendimiento en el acceso a datos temporales o frecuentemente consultados, se consideran bases de datos en memoria. Aquí se compararán Redis y Memcached.

Redis [28] es una base de datos en memoria basada en un modelo de estructura de datos clave-valor con soporte para varios tipos de estructuras de datos.

Ventajas:

- Alta velocidad de acceso a datos.
- Soporte para estructuras de datos complejas y persistencia opcional.

Memcached [29] es una solución de almacenamiento en caché distribuido que también utiliza un modelo clave-valor, optimizado para operaciones de lectura y escritura de alta velocidad.

Ventajas:

- Rápido y ligero, ideal para almacenamiento en caché simple.

Conclusión: Redis es la opción preferida en este proyecto para datos en memoria, gracias a su soporte para estructuras complejas y manejo eficiente de datos temporales. Además, de ser una herramienta utilizada durante el desarrollo del máster.

2.3.5. Servicios de plataformas en la nube

A la hora de desplegar aplicaciones web, existen diferentes proveedores de plataformas en la nube que ofrecen servicios variados dependiendo de las necesidades de escalabilidad, rendimiento y costos. A continuación, en la Tabla 2.1 se presenta una comparación entre los principales proveedores para ayudarte a elegir la opción más adecuada para el despliegue de tu aplicación.

Descripción de los servicios

- **Amazon Web Services (AWS)**[30]: AWS ofrece una amplia variedad de servicios, incluyendo instancias [Elastic Compute Cloud \(EC2\)](#), Lambda, [Simple Storage Service \(S3\)](#) y [Relational Database Service \(RDS\)](#). No cuenta con un plan gratuito, y los precios varían según el uso, con almacenamiento en [S3](#) desde \$0.023 por GB al mes. No incluye dominio gratuito, pero permite desplegar tanto el [Backend](#) como el [Frontend](#), y es totalmente compatible con [K8S](#) a través de Amazon [Elastic Kubernetes Service \(EKS\)](#).

Plataforma	Servicio gratuito	Dominio Gratis	Frontend	Backend	Base de datos	Kubernetes
AWS	No	No	Sí	Sí	Sí	Sí
GCP	No	No	Sí	Sí	Sí	Sí
Azure	No	No	Sí	Sí	Sí	Sí
Vercel	Sí para uso básico	Sí (vercel.app)	Sí	No	No	No
Netlify	Sí para uso básico	Sí (netlify.app)	Sí	No	No	No
Render	Sí para uso básico	No	Sí	Sí	Sí(limitado)	No
Firebase	Sí para uso básico	Sí (firebaseapp.com)	No	No	Sí	No

Tabla 2.1: Comparación de Plataformas para Despliegue de Aplicaciones

- **Microsoft Azure[31]:** Similar a AWS, Azure proporciona máquinas virtuales, App Services, bases de datos y almacenamiento. Tampoco ofrece un plan gratuito y el precio de almacenamiento es de aproximadamente \$0.0208 por GB al mes. No incluye un dominio gratuito, pero permite desplegar tanto Backend como Frontend, además de ser compatible con K8S mediante el servicio Azure K8S Service.
- **Google Cloud Platform (GCP)[32]:** GCP proporciona herramientas potentes como Google Compute Engine (ComputeEngine), Google Cloud Functions (CloudFunctions) y Google Cloud Storage (CloudStorage). No cuenta con un servicio gratuito y el almacenamiento cuesta aproximadamente \$0.020 por GB al mes. Similar a AWS y Azure, no ofrece dominio gratuito, pero permite desplegar Backend, Frontend, y bases de datos, además de ser compatible con K8S mediante Google K8S Engine.
- **Vercel[33]:** Vercel es una plataforma centrada en el despliegue de aplicaciones Frontend, como aplicaciones estáticas o con funciones Serverless. Ofrece un plan gratuito y el almacenamiento tiene un costo desde aproximadamente \$0.02 por GB al mes. Incluye un dominio gratuito en el plan básico.
- **Netlify[34]:** Similar a Vercel, Netlify está diseñado principalmente para el despliegue de aplicaciones Frontend. Ofrece un plan gratuito, y el almacenamiento cuesta aproximadamente \$0.03 por GB al mes. También incluye un dominio gratuito en el plan básico.
- **Render[35]:** Render es una plataforma que permite desplegar tanto Backend como Frontend y bases de datos. Tiene un plan gratuito para aplicaciones básicas y el costo de almacenamiento varía dependiendo del servicio. Destaca en su facilidad para desplegar aplicaciones web dockerizadas y tiene conectores para Redis y PostgreSQL, por lo que puede ser muy útil para el despliegue del backend en un entorno primitivo, y coincide con las herramientas seleccionadas para las bases de datos.
- **Firebase[36]:** Firebase es una plataforma de Google especializada en Backend y bases de datos para aplicaciones en tiempo real. Ofrece un plan gratuito con opciones de pago en función del almacenamiento y otros recursos.

Conclusión: Durante el desarrollo, se utilizará una solución gratuita para el desarrollo y la aprobación de los diseños por parte del cliente, por lo que se utilizará Netlify en el lado del Frontend, Render mediante despliegue de contenedores Docker para base de datos y Backend. En el futuro, se procederá a desplegar la aplicación en una plataforma como AWS, Azure o GCP. La preferencia recae sobre GCP, dado que he realizado un curso de especialización en esta plataforma y el precio no difiere significativamente de otras opciones, especialmente teniendo en cuenta que la web tiene un alcance reducido

y no requiere un gran volumen de procesamiento, lo cual no generará una diferencia considerable en el coste mensual.

2.3.6. Tecnologías para validación y pruebas

En el proceso de validación de esta aplicación, se utilizarán diversas tecnologías para garantizar su correcto funcionamiento, rendimiento y seguridad. Para las pruebas unitarias del *Backend* se emplearán JUnit [37] y Mockito [38], mientras que Postman [39] se utilizará para realizar pruebas manuales de los endpoints. Las pruebas funcionales del *Frontend* se automatizarán mediante Cypress [40], y las pruebas de rendimiento se llevarán a cabo con Apache JMeter [41]. Por último, para detectar posibles vulnerabilidades tanto en el *Backend* como en el *Frontend*, se aplicará la herramienta OWASP ZAP [42].

2.3.7. Entorno de desarrollo, gestión de versiones y diseño de diagramas

Para el desarrollo de la codificación del proyecto se ha utilizado Visual Studio Code [43], un editor de código ligero y altamente configurable que soporta múltiples lenguajes de programación, destacando en este caso el uso de Java gracias a sus extensiones específicas.

La gestión de versiones se ha llevado a cabo mediante la plataforma GitHub [44], que ofrece un sistema distribuido de control de versiones y facilita la integración continua y el trabajo colaborativo.

En cuanto al diseño de diagramas, se ha optado por PlantUML [45] como herramienta para la creación de diagramas UML. Esta elección se basa en su facilidad de integración con Visual Studio Code y su capacidad para generar diagramas a partir de texto, lo que agiliza su edición y mantenimiento en el proyecto.

Para el desarrollo de la memoria se ha utilizado LaTeX [46], compilado mediante la herramienta arara [47] ejecutada dentro de un contenedor Docker, lo que garantiza un entorno de compilación reproducible y aislado. La edición del código LaTeX se realizó principalmente en Visual Studio Code, complementado con paquetes específicos para la elaboración de diagramas de Gantt, como pgfgantt [48], que se utilizó para planificar y visualizar las tareas definidas durante el proyecto.

Este entorno de trabajo combinado ha permitido una gestión eficiente tanto del código como de la documentación, asegurando la calidad y la trazabilidad del proyecto.

2.3.8. Resumen de tecnologías seleccionadas

En resumen, en la Tabla 2.2 las tecnologías seleccionadas para el desarrollo de la aplicación de gestión y marketing para una casa rural son:

Tipo	Necesidad	Componente
Frontend	Construir interfaz dinámica	Angular
Backend	Servicio robusto y eficiente	Spring Boot
Contenerización	Empaquetar y facilitar despliegue	Docker
Orquestación	Gestión del ciclo de vida de contenedores	Kubernetes
Bases de datos	Contenido multimedia	MongoDB
Bases de datos	Autenticación y reservas	PostgreSQL / MySQL
Bases de datos	Datos en memoria	Redis
Plataforma en la nube	Despliegue escalable y alta disponibilidad	Google Cloud Platform (GCP)
Entorno de desarrollo	Editor de código	Visual Studio Code
Gestión de versiones	Control de versiones	GitHub
Diagramas UML	Diseño de diagramas UML	PlantUML
Pruebas	Pruebas unitarias backend	JUnit y Mockito
Pruebas	Pruebas funcionales backend	Postman
Pruebas	Pruebas funcionales frontend	Cypress
Pruebas	Pruebas de seguridad	OWASP ZAP
Pruebas	Pruebas de rendimiento	JMeter
Documentación	Redacción de memoria	LaTeX

Tabla 2.2: Tecnologías y herramientas utilizadas en el proyecto

Podrías no repetir Tipo agrupando las tecnologías.

Y lo sabes que de esta tabla solo ve si tiene de tecnologías muy importante.

El trabajo del capitán también es breve,
por lo que errores a corregir y no me convienen
algunas cosas como la resolución de los juegues.

Capítulo 3

Requisitos, especificaciones, coste, riesgos, viabilidad

3.1. Requisitos

En este capítulo se presentan los requisitos funcionales y no funcionales del sistema propuesto para la gestión de una casa rural, detallando las funcionalidades necesarias desde el punto de vista de los usuarios y del propietario. También se incluyen la estimación de costes, la planificación temporal del proyecto y los recursos utilizados para su desarrollo.

3.1.1. Descripción del sistema

El sistema desarrollado permitirá gestionar de manera integral una casa rural, proporcionando una plataforma intuitiva y funcional tanto para los gestores como para los usuarios. Ofrecerá un menú principal con información relevante sobre las instalaciones y actividades disponibles, así como un módulo de reservas en línea que facilitará la interacción entre los huéspedes y el gestor.

La plataforma contará con un sistema de recomendaciones que, de manera automática, ofrecerá sugerencias de sitios turísticos y eventos en la zona, personalizando dichas recomendaciones según las preferencias del usuario. Asimismo, se integrarán mapas interactivos para mostrar rutas y facilitar el acceso a la casa rural, así como información útil sobre la gastronomía y actividades al aire libre.

Además, el sistema brindará la posibilidad de contactar al propietario a través de un formulario de contacto y permitirá a los usuarios dejar reseñas sobre su experiencia, enriqueciendo así la interacción con futuros huéspedes. También se ofrecerá un pronóstico del clima para ayudar a los usuarios a planificar sus actividades, y un feed actualizado de las redes sociales para estar al tanto de las novedades de la casa rural.

3.1.2. Requisitos funcionales

- **RF1:** La plataforma debe contar con un módulo que muestre el clima actual y el pronóstico del tiempo para los días seleccionados en la reserva, específicamente para la zona de la casa rural.

- **RF2:** La plataforma debe ofrecer una sección con sugerencias de actividades al aire libre, como senderismo y ciclismo, que incluya descripción de las actividades, mapas interactivos y niveles de dificultad.
- **RF3:** La plataforma debe incluir un calendario visual que muestre la disponibilidad de la casa rural.
- **RF4:** La plataforma debe incluir un apartado para visualizar reseñas que han apor-tado clientes de la casa rural.
- **RF5:** La plataforma debe tener un apartado para visualizar los feeds de las publi-caciones de redes sociales de la casa rural.
- **RF6:** La plataforma debe contener recomendaciones sobre sitios, actividades y even-tos cercanos a la localización de la casa rural.
- **RF7:** La plataforma debe mostrar un menú principal con información general sobre la casa rural, incluyendo imágenes, vídeos y una descripción de las instalaciones.
- **RF8:** La plataforma debe proporcionar información sobre la gastronomía y otros aspectos turísticos del pueblo.
- **RF9:** La plataforma debe incluir un mapa interactivo que facilite la llegada a la casa rural, utilizando servicios de mapas integrados de Google Maps.
- **RF10:** La plataforma debe tener un apartado para realizar reservas, permitiendo a los usuarios registrados seleccionar una fecha en línea.
- **RF11:** La base de datos debe almacenar información sobre las reservas y los usuarios que las han realizado, para que esta información esté disponible en línea para todos los usuarios.
- **RF12:** Cada registro de reserva debe incluir el rango de fechas, el número de per-sonas, el precio total, el estado de la reserva y el usuario que realizó la reserva.
- **RF13:** Cada registro de usuario debe contener información como DNI/NIE/Pasa-porte, nombre, apellidos, e-mail, teléfono, dirección, fecha de nacimiento, contraseña (cifrada), fecha de registro, fecha de baja y estado de la cuenta.
- **RF14:** El sistema debe permitir a los usuarios registrados, con al menos una ex-periencia en la casa, dejar reseñas sobre su experiencia en la casa rural, y estas opiniones deben ser mostradas en la página.
- **RF15:** El sistema debe permitir a los usuarios contactar a la gerencia de la casa mediante un formulario que envíe un correo al gestor. Este formulario debe incluir los campos de nombre, e-mail, teléfono y mensaje.
- **RF16:** El sistema debe integrar las APIs de una aplicación de rutas para obtener y alimentar el menú de actividades al aire libre.
- **RF17:** El sistema debe implementar servicios de extracción de datos de páginas web de ayuntamientos y entidades cercanas que expongan blogs con eventos, tarjetas de recomendaciones de lugares o datos sobre climatología de forma automatizada.

SCRINING

- **RF18:** El sistema debe integrar las APIs de las redes sociales de la casa rural para obtener publicaciones, que se almacenarán en una base de datos documental. Estas publicaciones se mostrarán en la plataforma en tiempo real.
- **RF19:** El sistema debe utilizar los hashtags identificados en las publicaciones de redes sociales, almacenados junto a los feeds, para determinar en qué menús mostrar las publicaciones, llenando así los menús de recomendaciones, eventos, actividades e imágenes del alojamiento en la plataforma.
- **RF20:** El sistema debe avisar al usuario gestor por correo cuando se realice una solicitud de reserva, tras ser almacenada en el sistema.
- **RF21:** El sistema debe avisar al usuario cliente por correo cuando se aprueba o deniega una solicitud de reserva.
- **RF22:** El sistema debe estar diseñado para tres tipos de usuarios: el gestor, el usuario cliente registrado y el usuario no registrado.
- **RF23:** El sistema debe implementar un sistema de autenticación y autorización para los usuarios, con roles de gestor, cliente y usuario registrado.
- **RF24:** El usuario cliente, además de tener acceso a todo el contenido de la plataforma, podrá realizar reservas en la casa rural. También tendrá acceso a un menú donde podrá consultar sus reservas, gestionar su cuenta y dejar una reseña tras su estancia. Su rol se mantendrá como "cliente" hasta que complete la reseña correspondiente.
- **RF25:** El usuario gestor, además de tener acceso a todo el contenido de la plataforma, podrá revisar las peticiones de reservas y decidir si las aprueba o las deniega.
- **RF26:** El usuario no registrado podrá acceder a las secciones de la plataforma que no están restringidas a usuarios registrados o al gestor.

3.1.3. Requisitos no funcionales

- **RNF01:** La página debe cargarse en menos de 3 segundos en redes de velocidad media (50 a 200 Mbps).
- **RNF02:** El sistema debe proteger la información de los usuarios con protocolo [Transport Layer Security \(TLS\)](#) y medidas de seguridad en formularios de contacto.
- **RNF03:** El sistema debe ser escalable horizontalmente para manejar picos de tráfico y aumentos en la cantidad de datos dinámicos.
- **RNF04:** El sistema debe contar con un código modular y documentado para facilitar futuras actualizaciones y correcciones.
- **RNF05:** El sistema debe ser compatible con los principales navegadores (Chrome, Firefox, Safari, Edge) y dispositivos móviles (Android y iOS).
- **RNF06:** El sistema debe tener una interfaz de usuario intuitiva y accesible, con navegación sencilla entre los diferentes menús.

El proyecto es grande,
 no de error de compilación
 y lo he ejecutado cuando
 he cambiado
 otras cosa
 de la plantilla

- **RNF07:** El sistema debe implementar copias de seguridad automáticas para los datos almacenados en la base de datos temporal.
- **RNF08:** El sistema debe tener configuraciones de monitorización continua y registro de errores para la detección y resolución rápida de problemas.
- **RNF09:** El sistema debe ser desplegado mediante contenedores utilizando Docker para facilitar la portabilidad y la consistencia entre entornos de desarrollo, prueba y producción.
- **RNF10:** El despliegue del sistema debe ser gestionado y orquestado mediante [K8S](#) para mejorar la escalabilidad, la resiliencia y la administración de los microservicios en los servicios con más demanda.

3.2. Especificaciones

Una vez se han definido los requisitos funcionales y no funcionales del sistema, se procede a especificar el sistema a partir de ellos. En este apartado se muestran características del sistema, sin entrar en detalles de diseño e implementación, que permiten entender el problema al que nos enfrentamos pero no cómo lo vamos a afrontar (etapa de diseño).

3.2.1. Descripción general

El sistema a desarrollar es una plataforma para una casa rural que ofrece alojamiento y actividades turísticas en un entorno natural, tal y como representa la Figura 3.1. La aplicación web debe mostrar información general de la casa rural, recomendaciones de sitios turísticos y eventos cercanos, información sobre la gastronomía y otros aspectos turísticos del pueblo, un formulario de contacto, un calendario de disponibilidad, un sistema de reservas, mapas y rutas, el clima y pronóstico del tiempo, actividades al aire libre, reseñas de clientes y un feed de redes sociales.

3.2.2. Funcionalidades

- **Menú principal:** Muestra información general de la casa rural, incluyendo imágenes, vídeos y descripción de las instalaciones.
- **Recomendaciones y eventos:** Servicio que muestra recomendaciones de sitios turísticos y eventos cercanos, actualizados automáticamente.
- **Información turística:** Sección que muestra información sobre la gastronomía y otros aspectos turísticos del pueblo.
- **Formulario de contacto:** Permite a los usuarios contactar al dueño de la casa mediante un formulario que incluye campos de nombre, email, teléfono y mensaje.
- **Calendario de disponibilidad:** Calendario visual que muestra la disponibilidad de la casa rural.

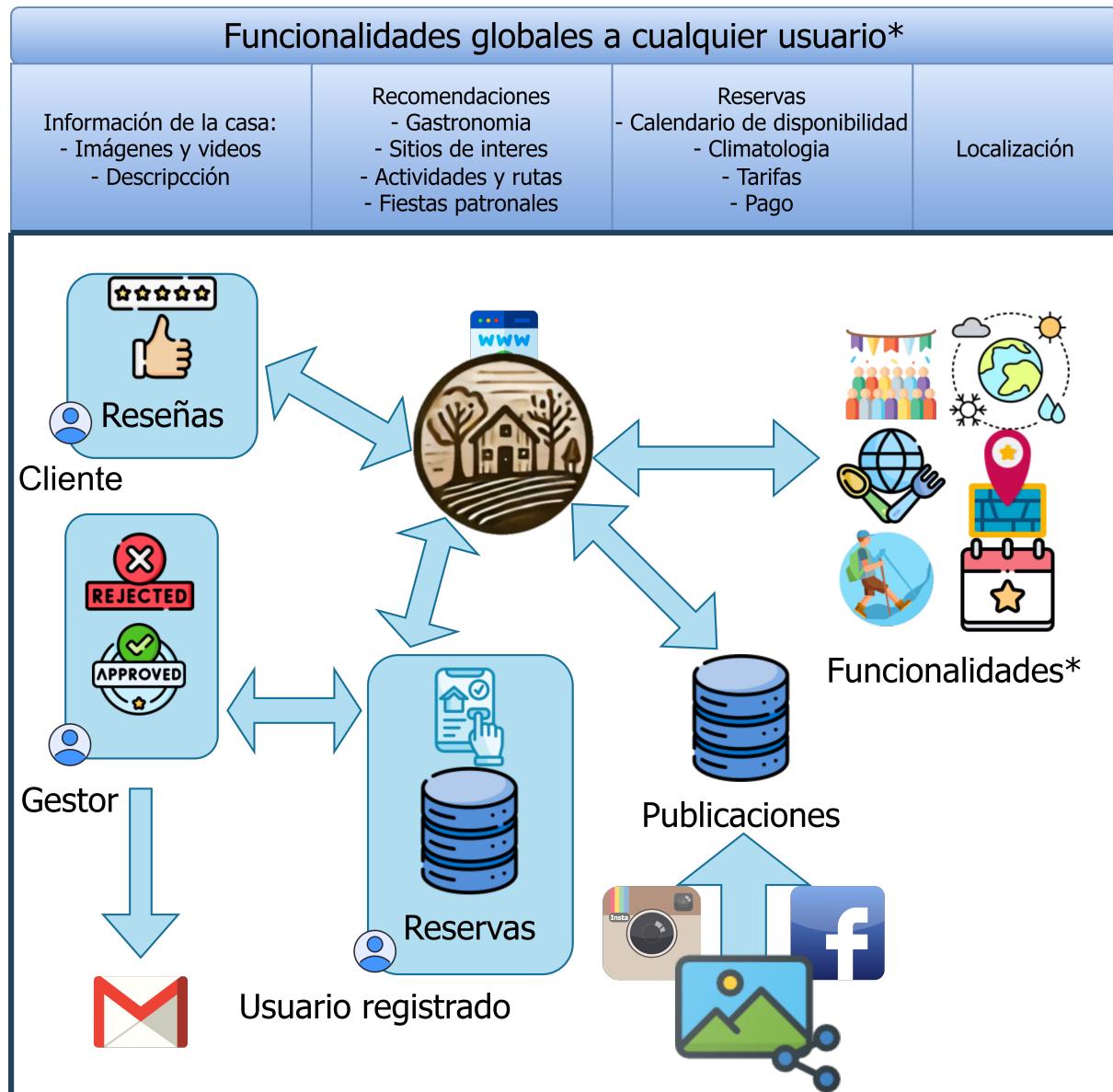


Figura 3.1: Arquitectura de referencia del sistema.

- **Sistema de reservas:** Permite al usuario reservar una fecha en línea y gestiona la aprobación o denegación de la reserva, concluyendo con un correo de confirmación o rechazo al usuario.
- **Mapas y rutas:** Muestra mapas y rutas para facilitar la llegada a la casa rural, utilizando integración con servicios de mapas.
- **Clima y pronóstico:** Módulo que muestra el clima actual y el pronóstico del tiempo para los días seleccionados en la reserva en la zona de la casa rural.
- **Actividades:** Sección con sugerencias de actividades al aire libre como senderismo y ciclismo, con mapas interactivos y niveles de dificultad, actualizados automáticamente según la ubicación.
- **Reseñas de huéspedes:** Permite a los huéspedes dejar reseñas sobre su experiencia en la casa rural y muestra estas opiniones en la página.

- **Feed de redes sociales:** Integra un feed en tiempo real de las publicaciones de redes sociales de la casa rural.

3.3. Planificación y estimación de costes

En este apartado se plantea la planificación del proyecto, abarcando el ciclo de vida a seguir, las tareas en las que se descompondrá y el coste temporal asociado.

Dado el carácter individual del proyecto, se ha optado por una metodología ágil adaptada [49], basada en *sprints* cortos. Aunque las metodologías ágiles suelen estar diseñadas para equipos, su capacidad para priorizar tareas y validar requisitos de forma iterativa se ajusta a las necesidades de este desarrollo. La periodicidad de las revisiones permite garantizar un progreso continuo y la incorporación de mejoras basadas en las necesidades específicas del proyecto.

Para la estimación del tiempo necesario para cada tarea, se ha utilizado la técnica *Program Evaluation and Review Technique* [50] basada en la *distribución Beta*. Este enfoque permite calcular una estimación ponderada considerando tiempos optimistas, pesimistas y el más probable. Se ha optado por la *distribución Beta* frente a otras alternativas, como la clásica o la triangular, debido a su capacidad para dar un mayor peso al tiempo más probable, lo cual es especialmente útil en proyectos donde las tareas presentan incertidumbres moderadas pero es posible realizar una estimación razonable de su duración central. Este método asegura un equilibrio entre escenarios extremos (optimista y pesimista) y la realidad del desarrollo, permitiendo obtener una planificación más ajustada.

Además, este enfoque resulta particularmente adecuado al tratarse de un único recurso, donde métodos como Planning Poker [51] o consultas a expertos no tienen aplicabilidad directa. Al ser una aplicación web personalizada con múltiples integraciones, no es un proyecto repetitivo que permita apoyarse en datos históricos, lo que refuerza la idoneidad del enfoque probabilístico de *Program Evaluation and Review Technique*. → ~~PERT~~

La planificación seguirá un enfoque secuencial, con actividades organizadas en función de sus dependencias, aunque algunas podrán ejecutarse en paralelo para optimizar el tiempo de desarrollo. Las tareas se agrupan en fases como análisis, diseño, implementación y pruebas, y se estima su duración en horas. Estas estimaciones se convertirán posteriormente a días para facilitar la planificación general. Este desglose permitirá evaluar el esfuerzo necesario para completar el proyecto y garantizar que se cumplan los plazos establecidos. La estimación de tiempos se calcula tal y como define la Ecuación 3.1.

$$\text{Estimación} = \frac{O + P + (4 \cdot M)}{6}$$

(3.1)

Fórmula para la estimación de tiempos basada en la técnica PERT, donde O es el tiempo optimista, P el pesimista y M el más probable.

Considerando la metodología ágil adoptada, la Tabla 3.1 detalla las tareas a realizar, desglosadas por fase y con sus correspondientes estimaciones de tiempo, calculadas mediante la metodología *Program Evaluation and Review Technique*. Además, cada tarea se ha asignado a uno de los siete sprints planificados, reflejando la organización incremental e iterativa del trabajo. De este modo, se puede visualizar no solo el esfuerzo estimado en horas, sino también la distribución de las tareas a lo largo de los diferentes ciclos de desarrollo.

Sprint	Fase	Descripción	T.Optimista (h)	T.Pesimista (h)	T.Mejor (h)	Estimación (h)
1	Documentación	Estado del arte	8	12	10	10
		Análisis Requisitos	8	12	9	9.33
		Análisis Especificación técnica	8	12	9	9.33
	Análisis	Diagramas y casos de uso	8	10	9	9.33
		Diseño del usuario	8	10	9	9.33
		Compatibilidad de tecnologías	8	11	9	9.33
	Diseño	Infraestructura	8	12	10	10
		Backend	8	10	9	9.33
2	Implementación	Front End: Menú principal	8	10	9	9.33
		Formulario de contacto	2	4	3	3
		Autenticación y autorización	8	12	9	9.33
	Implementación	Backend: Lógica y base de datos reseñas	16	22	18	18.33
		Base de datos de usuarios y roles	2	4	3	3
		Pruebas Unitarias: obtención de reseñas	2	3	2.5	2.5
3	Implementación	Microservicio: contenido de redes sociales	8	12	10	10
		Recomendaciones y eventos	3	5	4	4
		Información turística	3	5	4	4
	Implementación	Integración datos: menú principal	8	12	10	10
		Integración datos: información turística	2	4	3	3
		Pruebas Datos: menú principal	2	3	2.5	2.5
	Pruebas	Datos: información turística	2	3	2.5	2.5
4	Implementación	Calendario de disponibilidad	6	10	8	8
		Sistema de reservas: Front End	5	8	6.5	6.5
		Base de datos de reservas	2	4	3	3
	Implementación	Sistema de reservas: Backend	10	15	12	12.33
		Pruebas Unitarias: sistema de reservas	3	5	4	4
		Mapas interactivos	1	2	1.5	1.5
	Implementación	Pronóstico del clima	1	2	1.5	1.5
5	Implementación	Microservicio: clima API	6	10	8	8
		Microservicio: ayuntamientos API	12	18	14	14.33
	Implementación	Actividades	2	4	3	3
		Microservicio: actividades API	8	12	10	10
	Pruebas	Datos: actividades al aire libre	1	2	1.5	1.5
		Datos: recomendaciones y eventos	1	2	1.5	1.5
6	Implementación	Backend: Feed redes sociales	4	7	5.5	5.5
		Pruebas Usuario final	8	12	10	10
	Pruebas	Seguridad	2	3	2.5	2.5
		Backend Dockerización	16	22	18	18.33
7	Producción	Evaluación soluciones y despliegue	8	12	10	10
		Total	225	322	264.5	263.92

Tabla 3.1: Estimación de tareas del proyecto con asignación por sprint según planificación.

El quadacíz más visto es resultar los sprints de alguna forma.

El ciclo de vida estimado para el proyecto es de aproximadamente **14 semanas**, lo cual corresponde a un total de **263.92 horas** de trabajo. Debido a la combinación de actividades laborales y estudios, se estima una dedicación de **22 horas semanales**, distribuidas en **2 horas diarias entre semana** y **6 horas los fines de semana**, tal y como se muestra en la Tabla 3.2.

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
2h	2h	2h	2h	2h	6h	6h

Figura 3.2: Horario de dedicación

Seguidamente, en la Figura 3.2 se encuentra el diagrama de Gantt, generado con el paquete Pgfgantt [48], ilustra gráficamente la planificación temporal del proyecto. En él, cada sprint corresponde a un ciclo de trabajo de dos semanas (14 días), lo que permite observar claramente la secuencia y solapamiento de tareas en el tiempo, así como el avance progresivo del proyecto conforme a los principios de la gestión ágil.

A continuación, se va a estimar el coste de personal del proyecto. Para este proyecto se ha destinado 1 persona durante 291 horas (264 estimadas y un 10 % margen). Basándome en el estudio Michael Page de salarios [52] para un perfil de desarrollador Back End y Front End en Valencia con menos de dos años de experiencia el salario bruto esta entre los 22.000€ y los 40.000€ por lo tanto se va a coger la media de estas dos cifras 31.000€ al año que convertido a salario mensual da un total de 2.583,40€ por mes. Por lo tanto, el salario por hora contando un trabajo a jornada completa de 40 horas es de 16,15€/h. Multiplicado por el número de horas estimadas, da un total de 4.699,95€ al cual se le debe sumar la correspondiente cotización en la seguridad social [53] que en este caso es del 28,3 %. Por lo tanto, se obtendría un coste del proyecto de 6.015,94 €.

Otro aspecto a tener en cuenta son los costes materiales y la amortización que se va a hacer de ellos durante el desarrollo del proyecto. Lo primero que se va a calcular es la amortización de los equipos informáticos a utilizar en el desarrollo del proyecto. El porcentaje de amortización vigente para sistemas informáticos según la Agencia Estatal de Administración Tributaria (AEAT) [54] es del 26 % y su vida útil estará entre 4 y 8 años por lo que se selecciona una vida útil para todos los componentes de 4 años (48 meses) para una mayor amortización, aplicando la Fórmula 3.2.

$$\frac{\text{Coste del material}}{4} * \text{Años amortización (uso material)} \quad (3.2)$$

La Tabla 3.3 muestra los costes a imputar de cada uno de los componentes utilizados. En este caso sólo se dispone de un componente ya que los demás componentes forman parte de la infraestructura del proyecto y pasan a ser costes directos. Ya que el único componente de coste directo sería la utilización de un entorno de plataforma en la nube PaaS y que el coste de este variará en función de su utilización, no es posible añadirlo a la estimación.

Para finalizar, la Tabla 3.4 muestra el coste de cada activo a utilizar durante el proyecto, incluyendo materiales, equipos de desarrollo y personal:

3.4. Riesgos

En el transcurso del desarrollo existe una gran cantidad de módulos a implementar que posteriormente deben encajar y esto es posiblemente lo más costoso del proyecto y

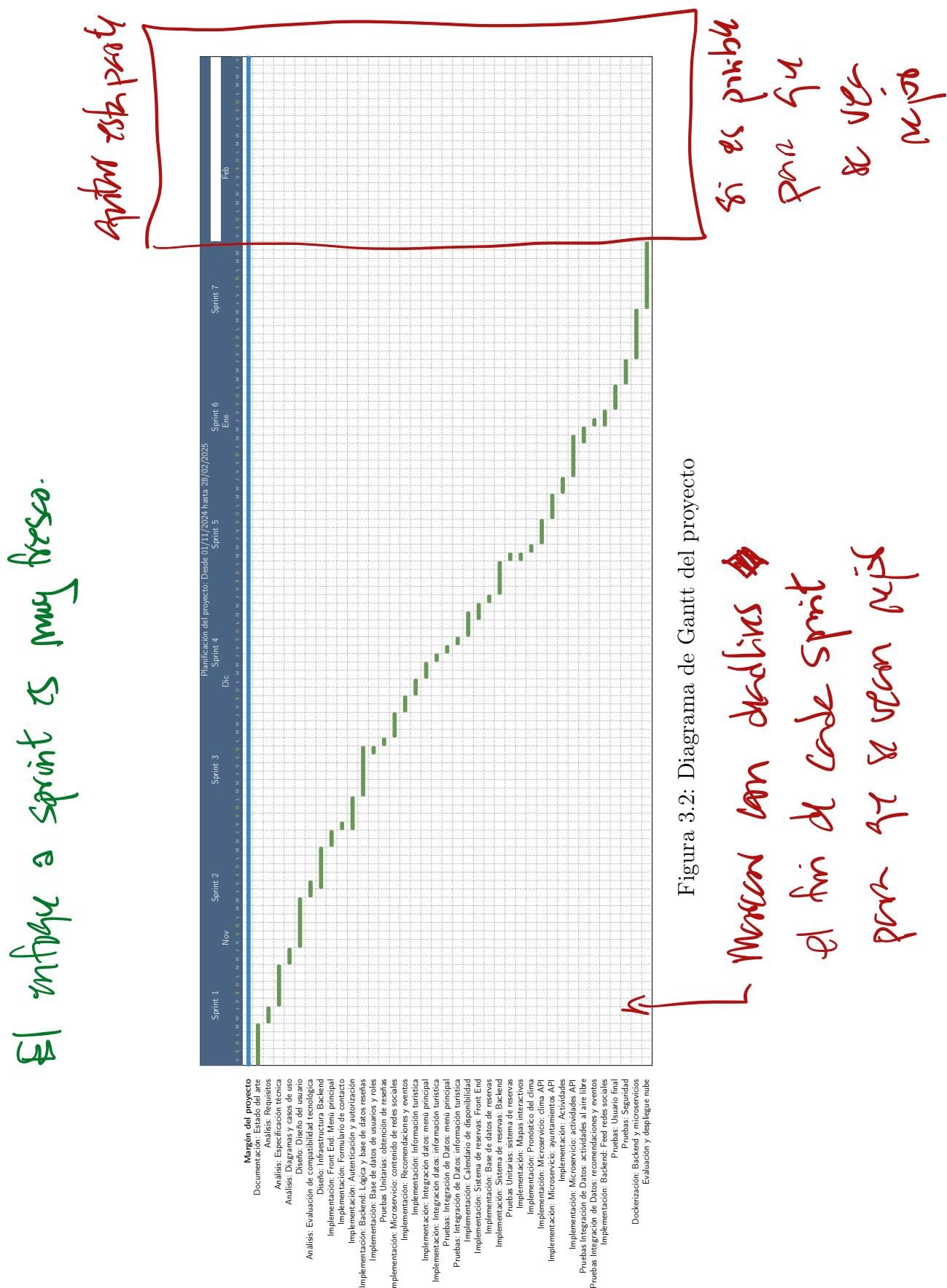


Figura 3.2: Diagrama de Gantt del proyecto

Llegan hechas de otra parte pero
dím así no se arriba de vez bien

Elemento	Coste	Meses de utilización	Coste de Amortización
Ordenador para desarrollo	890,00 €	4	74,16 €

Tabla 3.3: Estimación de la amortización de materiales

Elemento	Coste (€)
Amortización portátil para desarrollo	74,16
Recursos humanos	6.015,94
Total	6.090,11 €

Tabla 3.4: Estimación costes del proyecto

por lo que se va a dedicar tanto tiempo al desarrollo del sistema principal y a implantarlo con los demás subsistemas. Esta planificación por tanto lleva asociados unos riesgos que podrían retrasar el ciclo de vida del proyecto, los cuales se enumerarán a continuación:

- 1. Falta de tiempo por actividad laboral. *Proyecto scrum incorrecto*
- 2. Problemas de compatibilidad entre las API (Application Programming Interface) a utilizar.
- 3. Problemas de compatibilidad con plataformas de despliegue en la nube utilizadas.

Una vez analizados los posibles riesgos detectados, se plantea el plan de contingencia de la Tabla 3.5.

Además de las soluciones planteadas anteriormente para cada riesgo detectado, en la planificación del proyecto se ha tenido en cuenta un margen de error del 10% (1 último sprint y una semana respecto a lo estimado que equivale a 27 horas) de margen para poder solucionar los riesgos valorados anteriormente.

3.5. Viabilidad

Teniendo en cuenta los puntos observados anteriormente se puede decir que el proyecto está planificado para su entrega en el mes de marzo de 2025. Empezando el desarrollo en el mes de noviembre de 2024 y finalizándolo durante el mes de febrero de 2025. En cuanto a su coste, se ha estimado **6.090,11 €**. Es un coste medio por lo que en caso del desarrollo de un proyecto real con esta estimación no habría problema a la hora de asumirlo. Su coste temporal es de 98 días teniendo como margen 21 días más, por lo que podría alargarse el proyecto hasta principios de marzo. Es un proyecto que podría realizarse en paralelo si se tuviesen varios desarrolladores, en concreto uno encargado del Frontend y otro para Backend. Pero en este caso se ha optado por un desarrollo secuencial. Dado que las tecnologías a utilizar son conocidas sería bastante realista al decir que la planificación estimada se va a cumplir dentro de los márgenes establecidos.

Riesgo	Coste	Solución
Falta de tiempo por actividad laboral.	2 horas por semana	Incrementar el tiempo dedicado el fin de semana de 6 a 8 horas para compensar días con mayor carga de trabajo entre semana
Problemas de compatibilidad entre las API (Application Programming Interface) a utilizar	1 semana	A la hora de integrar distintas APIs pueden surgir incompatibilidades por cambios en sus versiones, diferencias en el formato de los datos o limitaciones de acceso. Este tipo de problemas podría suponer un retraso considerable, especialmente si afecta a funcionalidades clave. Para reducir el impacto, he analizado previamente varias alternativas para cada API (Application Programming Interface) principal. Así, si surge algún inconveniente grave, podré cambiar rápidamente a otra opción sin necesidad de rehacer gran parte del trabajo. También realizaré pruebas de integración desde las primeras fases del desarrollo para detectar posibles incompatibilidades lo antes posible.
Problemas de compatibilidad con plataformas de despliegue en la nube utilizadas.	1 semana	Requisito opcional: El uso de servicios en la nube puede provocar incidencias relacionadas con la configuración de la infraestructura, incompatibilidades entre entornos o restricciones en los servicios contratados. Si llegara a ocurrir algún problema que impida el despliegue en la nube, el proyecto podría retrasarse o no completarse según lo previsto. Por eso, tengo previsto como solución de contingencia realizar el despliegue final en un entorno local (en mi propio equipo o red local), asegurando que la funcionalidad del sistema pueda ser demostrada aunque el entorno cloud no esté disponible a tiempo. Durante el desarrollo, procuraré mantener siempre operativa esta alternativa local como respaldo.

Tabla 3.5: Plan de contingencias para los riesgos detectados

El trabajo del capítulo también es breve,
por la ley errores a corregir y no me convienen
algunas cosas como la resolución de los errores.

Capítulo 4

Análisis

4.1. Análisis

En esta sección se va a analizar los casos de uso necesarios para la implementación de nuestra aplicación, relacionándolos con los requisitos deducidos en el apartado anterior. Además, también se va a definir los actores que interactúan con el sistema y un primer modelo de datos que seguirá la aplicación.

4.1.1. Diagrama de casos de uso

En la Figura 4.1 se observan todos los casos de uso identificados con respecto a la toma de requisitos realizada. En este caso se han identificado los siguientes actores, detallándolos de menos a más privilegios dentro de la aplicación web como se aprecia en la Figura 4.2 que incluye un diagrama de clases de herencia de actores funcionales en la plataforma web.

- **Usuario:** Este actor es un usuario que accede a la aplicación web sin registrarse, podrá realizar solo labores de consulta de información sobre la aplicación.
- **Usuario registrado:** Este actor se identificara cuando el actor usuario se haya registrado o haya iniciado sesión correctamente y podrá realizar las acciones necesarias para reservar la casa rural.
- **Cliente:** Este actor será un actor de tipo Cliente al que se le haya asignado una reserva y esta haya finalizado, lo que le dará derecho a dejar una reseña del alojamiento.
- **Gestor:** Este actor podrá realizar las acciones de aprobación y denegación de reservas realizadas por los clientes.
- **Sistema:** Este es un actor de tipo no funcional que realizara todas las labores del sistema, como son el refresco de los datos, el envío de correos de reserva y la extracción y actualización periódica de entidades externas.

A continuación, se van a definir cada uno de los casos de uso ilustrados en la Figura 4.1 y los requisitos con los que están involucrados. Inicialmente, el usuario puede acceder al sistema de forma anónima, registrarse o iniciar sesión. En la Tabla 4.1 se observa el caso de

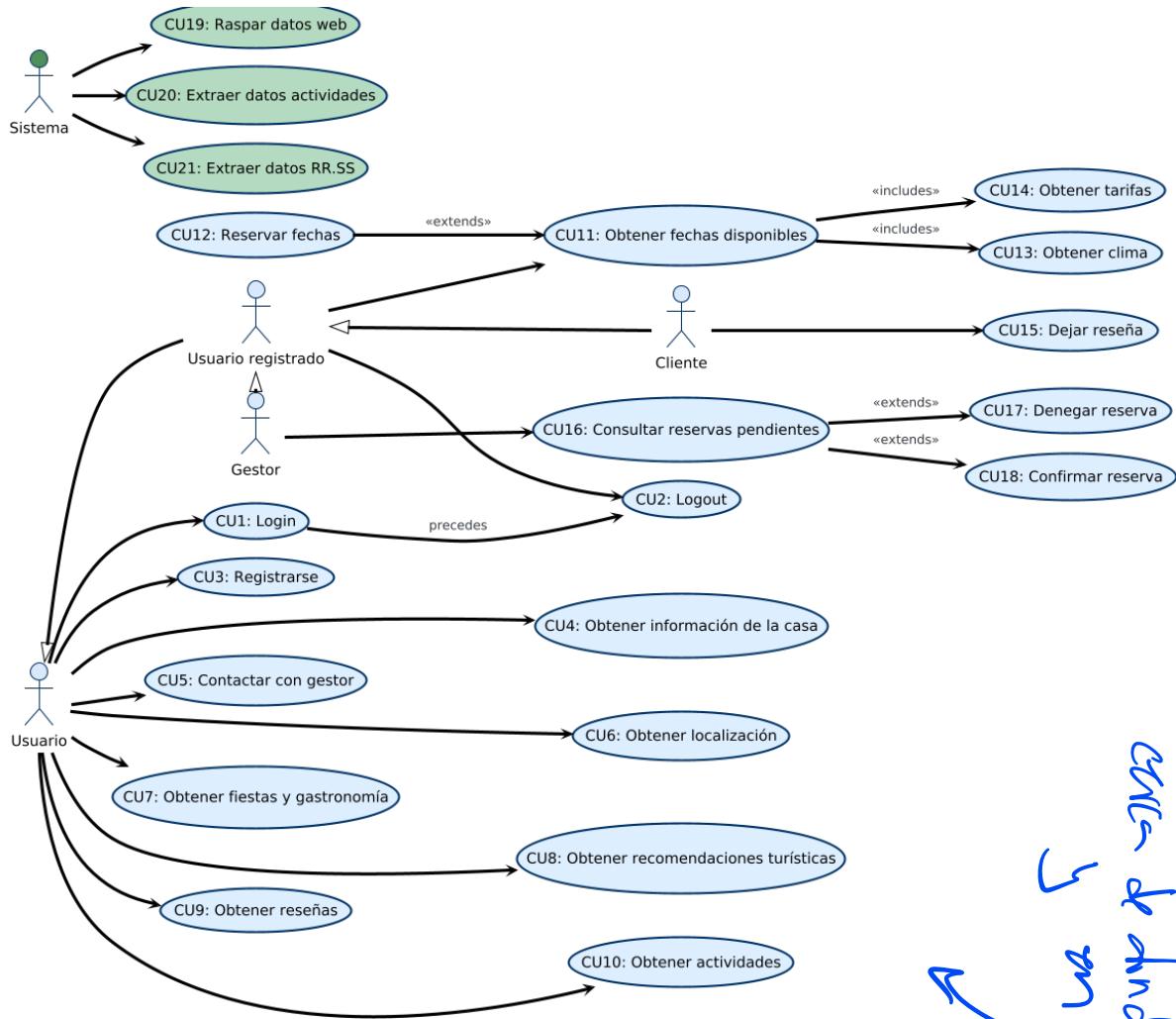
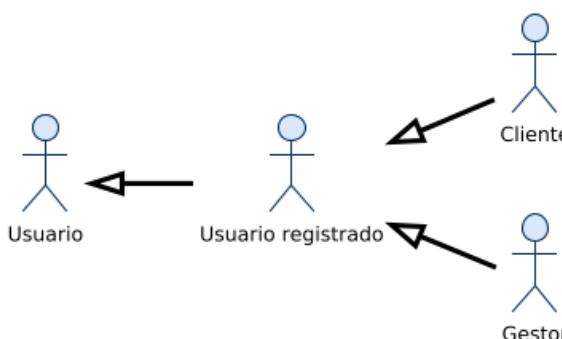


Figura 4.1: Diagrama general de casos de uso.

He cambiado el orden para siempre
tienen que ir apareciendo lo más
principales de arriba hacia abajo
en ese orden

Figura 4.2: Diagrama de clases de los actores que influyen en el sistema.
interactúan

uso relacionado con el inicio de sesión. Una vez se ha iniciado sesión como cliente el usuario podrá cerrar sesión causando el efecto contrario sobre la interfaz web como se observa en la Tabla 4.2. En caso de que el usuario no esté registrado como se observa en la Tabla 4.3, se le ofrecerá la opción de realizarlo. Una vez se han revisado los casos de uso referentes a la autenticación, en la Tabla 4.4 se observa uno de los casos de uso accesibles por cualquier usuario de la aplicación web encargado de obtener los datos de la casa rural, incluyendo

Número	CU-1
Nombre	Login
Actores	Usuario
Descripción	Autenticarse en la interfaz web mediante usuario y contraseña.
Flujo de datos	<ol style="list-style-type: none"> 1. Rellenar el formulario de inicio de sesión. 2. Iniciar sesión como cliente. 3. Validar formulario en el cliente. 4. Validar formulario en el servidor.
Pre-condiciones	Debe de existir el cliente en el sistema.
Post-condiciones	El usuario está autenticado en la sesión. El usuario tiene acceso a la sección de reservas.
Requisitos	RNF02, RF10, RF11, RF23

Tabla 4.1: CU1 - Login

Número	CU-2
Nombre	Logout
Actores	Cliente
Descripción	Cerrar sesión como cliente.
Flujo de datos	<ol style="list-style-type: none"> 1. Cerrar sesión como cliente.
Pre-condiciones	Debe de existir el cliente en el sistema. El cliente debe estar logueado.
Post-condiciones	El usuario volverá a ser anonimo. Se eliminará cualquier estado de sesión almacenada. Se limitará la posibilidad de realizar una reserva.
Requisitos	RNF02, RF10, RF11, RF23

Tabla 4.2: CU2 - Logout

Número	CU-3
Nombre	Registrarse
Actores	Usuario
Descripción	Registro de un nuevo cliente en la aplicación web, almacenando sus datos en la base de datos
Flujo de datos	<ol style="list-style-type: none"> 1. Rellenar los formularios con los datos del nuevo usuario. 2. Validar el formulario de datos en el cliente. 3. Enviar el formulario y validararlo en el servidor. <ul style="list-style-type: none"> a) Mostrar un mensaje satisfactorio y redirigir a la pantalla de inicio con la sesión del nuevo usuario registrada. b) Mostrar un mensaje de error por existencia de los datos del usuario en el servidor.
Pre-condiciones	No debe de existir el email del cliente en el sistema.
Post-condiciones	El usuario estará registrado en el sistema. Se añadirá un nuevo estado de sesión almacenada. Se redirigirá al usuario al menú principal.
Requisitos	RF13, RNF02, RF10, RF11, RF23

Tabla 4.3: CU3 - Registrarse

las descripciones y contenido multimedia necesario. En la Tabla 4.5 se puede observar el caso de uso que permite el contacto con el administrador desde el menú principal. En la

Número	CU-4
Nombre	Obtener información de la casa
Actores	Usuario
Descripción	Visualización de datos referentes a las instalaciones y los servicios ofrecidos.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “La casa”. 2. Visualización de datos y contenido multimedia. 3. Enlace a página para reservas.
Pre-condiciones	El servidor web se encuentre en funcionamiento.
Post-condiciones	El usuario visualizará los datos y el contenido multimedia relacionado con la casa rural.
Requisitos	RF5, RF7, RF19, RF26

Tabla 4.4: CU4 - Obtener información de la casa

Número	CU-5
Nombre	Contactar con gestor
Actores	Usuario
Descripción	Rellenar un formulario de contacto que concluira con un correo enviado al gestor de la casa rural.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Inicio”. 2. Rellenar el formulario de contacto incluyendo nombre, email y el mensaje. 3. Enviar.
Pre-condiciones	El servidor web se encuentre en funcionamiento.
Post-condiciones	En el correo del administrador se debe visualizar un nuevo correo.
Requisitos	RF15, RF26

Tabla 4.5: CU5 - Contactar con gestor

Tabla 4.6 se puede observar el caso de uso que permite al usuario visualizar la localización de la casa rural. En la Tabla 4.7 se puede observar el caso de uso que permite al usuario

Número	CU-6
Nombre	Obtener localización
Actores	Usuario
Descripción	Visualizar la localización de la casa rural mediante un mapa interactivo.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Dónde estamos”. 2. Visualizar y interactuar con el mapa interactivo.
Pre-condiciones	El servidor web se encuentre en funcionamiento.
Post-condiciones	Se debe de visualizar un mapa interactivo situado sobre el inmueble.
Requisitos	RF9, RF26

Tabla 4.6: CU6 - Obtener localización

visualizar una sección que incluye contenido multimedia estático y dinámico sobre fiestas y gastronomía de los pueblos cercanos. En la Tabla 4.8 se puede observar el caso de uso que permite al usuario visualizar el contenido multimedia de lugares cercanos a visitar durante su estancia. En la Tabla 4.9 se puede observar el caso de uso que permite al usuario visualizar las reseñas aportadas por huéspedes de la casa rural. Para finalizar con los casos de uso que podrán realizar todos los usuarios y que son de solo lectura, en la Tabla 4.10 se puede observar el caso de uso que permite al usuario visualizar un conjunto de actividades a realizar en la naturaleza. A continuación, se presentarán los casos de uso

Número	CU-7
Nombre	Obtener fiestas y gastronomía
Actores	Usuario
Descripción	Visualizar una sección con dulces y platos típicos y otra sección que incluya las festividades del pueblo.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Recomendaciones”. 2. Visualizar el contenido textual y multimedia.
Pre-condiciones	El servidor web se encuentre en funcionamiento.
Post-condiciones	Se debe visualizar contenido tanto estático, como extraido de las redes sociales.
Requisitos	RF8, RF19, RF26

Tabla 4.7: CU7 - Obtener fiestas y gastronomía

Número	CU-8
Nombre	Obtener recomendaciones turísticas
Actores	Usuario
Descripción	Visualizar el contenido multimedia de lugares a visitar con una pequeña descripción.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Entorno”. 2. Visualizar el contenido multimedia con una pequeña descripción. <ol style="list-style-type: none"> a) Navegar a la descripción detallada de la información turística en el sistema origen.
Pre-condiciones	El servidor web se encuentre en funcionamiento.
Post-condiciones	Se debe visualizar contenido tanto estático, como extraido de las redes sociales. Si se hace click sobre alguna se debe redirigir al sistema origen del cual se extrajo ese lugar.
Requisitos	RF6, RF17, RF19, RF26

Tabla 4.8: CU8 - Obtener recomendaciones turísticas

Número	CU-9
Nombre	Obtener reseñas
Actores	Usuario
Descripción	Visualizar una lista de reseñas sobre la casa rural.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Inicio”. 2. Visualizar una lista de reseñas que contenga puntuación, reseña y fecha.
Pre-condiciones	El servidor web se encuentre en funcionamiento. Debe haber al menos una reseña almacenada en el servidor.
Post-condiciones	Se debe visualizar una lista de reseñas que incluirán una puntuación final sobre la estancia.
Requisitos	RF4, RF26

Tabla 4.9: CU9 - Obtener reseñas

que van relacionados con un usuario registrado y que ha iniciado sesión en el sistema. En la Tabla 4.11 se observa el caso encargado de la obtención de las fechas disponibles para reservar la casa rural. Seguidamente, será posible realizar diversas acciones sobre esa sección de reservas que son las incluidas en los casos de uso 13 4.13 y 14 4.14 y que finalizará en la consecución de una reserva si se desea en el caso de uso 12 4.12.

En cuanto al rol de huésped se ha definido un único caso de uso que lo diferencia de todos los casos que pueden realizar los usuarios. En la Tabla 4.15 se encuentra especificado el caso encargado de dejar las reseñas. En la Tabla 4.16 se observa el caso de uso que

Número	CU-10
Nombre	Obtener actividades
Actores	Usuario
Descripción	Visualizar un menu interactivo con actividades al aire libre a realizar cerca de la casa rural.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Entorno”. 2. Visualizar un conjunto de actividades a realizar en la naturaleza con descripción y dificultad de la actividad. <ol style="list-style-type: none"> a) Navegar a la descripción detallada de la actividad en el sistema origen.
Pre-condiciones	El servidor web se encuentre en funcionamiento. La api que proporciona las actividades debe estar en funcionamiento.
Post-condiciones	Se debe visualizar un conjunto de actividades a realizar en la naturaleza con descripción y dificultad de la actividad. Si se hace click sobre alguna se debe redirigir al sistema origen del cual se extrajo esa actividad.
Requisitos	RF16, RF26

Tabla 4.10: CU10 - Obtener actividades

Número	CU-11
Nombre	Obtener fechas disponibles
Actores	Cliente
Descripción	Visualizar un calendario con las fechas disponibles.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Reservas”. 2. Visualizar un calendario interactivo con las fechas disponibles. 3. El cliente podrá seleccionar fechas a futuro.
Pre-condiciones	El servidor web se encuentre en funcionamiento. Solo se mostrarán fechas a futuro. Las fechas no disponibles se marcarán con un color diferente.
Post-condiciones	Se debe visualizar un calendario interactivo con las fechas disponibles y las ocupadas.
Requisitos	RF3, RF10

Tabla 4.11: CU11 - Obtener fechas disponibles

permite al administrador visualizar las reservas pendientes de confirmación.

En la Tabla 4.17 se observa el caso de uso que permite al administrador aprobar una reserva pendiente.

En la Tabla 4.18 se observa el caso de uso que permite al administrador rechazar una reserva pendiente. Por último, se han definido los casos de uso que realiza el sistema internamente. En la Tabla 4.19 se observa el caso de uso que permite realizar la extracción de datos turísticos para mostrarlos en la aplicación web. En la Tabla 4.20 se observa el caso de uso que permite realizar la extracción de la API para obtener actividades a realizar y para mostrarlos en la aplicación web. En la Tabla 4.21 se observa el caso de uso que permite realizar la extracción de datos de redes sociales para mostrarlas en la aplicación web.

Número	CU-12
Nombre	Reservar fechas
Actores	Cliente
Descripción	Seleccionar u conjunto de fechas disponibles y formalizar una reserva.
Flujo de datos	<ol style="list-style-type: none"> 1. Selección de un conjunto de fechas. 2. Validación por parte del servidor de las fechas. 3. Envío correo con solicitud de reserva al administrador. 4. Nuevo registro de reserva en la base de datos.
Pre-condiciones	<p>Se deben cumplir las condiciones y el flujo del CU11.</p> <p>El servidor web se encuentre en funcionamiento.</p> <p>Solo se podrán seleccionar fechas a futuro.</p> <p>Las fechas no disponibles no se podrán seleccionar.</p> <p>Deberá existir una base de datos que almacene las reservas.</p>
Post-condiciones	<p>Se debe visualizar un mensaje de éxito de creación de la reserva.</p> <p>Se debe mostrar como reserva pendiente de confirmación en el perfil del cliente.</p>
Requisitos	RF3, RF10, RF12

Tabla 4.12: CU12 - Reservar fechas

Número	CU-13
Nombre	Obtener clima
Actores	Cliente
Descripción	Visualizar una sección con las condiciones climatológicas resumidas que habrá cuando se seleccione una fecha en el CU11.
Flujo de datos	<ol style="list-style-type: none"> 1. Selección de un conjunto de fechas. 2. Llamada al servicio interno de obtención de condiciones climáticas. 3. Visualización de respuesta de datos meteorológicos para cada día seleccionado.
Pre-condiciones	<p>Se deben cumplir las condiciones y el flujo del CU11.</p> <p>El servidor web se encuentre en funcionamiento.</p> <p>El servicio externo de tiempo debe estar disponible.</p>
Post-condiciones	Se debe visualizar una sección con el pronóstico de tiempo para cada uno de los días seleccionados.
Requisitos	RF1

Tabla 4.13: CU13 - Obtener clima

Número	CU-14
Nombre	Obtener tarifas
Actores	Cliente
Descripción	Visualizar las tarifas asociadas a las fechas seleccionadas en el CU11.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Reservas”. 2. Visualizar las tarifas asociadas a las fechas seleccionadas en el CU11.
Pre-condiciones	El servidor web se encuentre en funcionamiento.
Post-condiciones	Se debe visualizar las tarifas asociadas a las fechas seleccionadas.
Requisitos	RF12

Tabla 4.14: CU14 - Obtener tarifas

Número	CU-15
Nombre	Dejar reseña
Actores	Huésped
Descripción	Permite dejar una reseña a un usuario que cumpla la condición de huésped.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Menu principal”. 2. Rellenar el formulario de nueva reseña con los campos de puntuación y descripción. 3. Enviar la reseña.
Pre-condiciones	El servidor web se encuentre en funcionamiento. El cliente registrado debe tener asignado el rol de huésped, que se otorga cuando se completa una reserva y la fecha de esta es anterior a la fecha de hoy.
Post-condiciones	Se debe visualizar un mensaje de confirmación de la reseña. Se debe visualizar en el último elemento de la lista de la sección la nueva reseña.
Requisitos	RF14, RF24

Tabla 4.15: CU15 - Dejar reseña

Número	CU-16
Nombre	Consultar reservas pendientes
Actores	Administrador
Descripción	Permite al administrador visualizar una lista de reservas pendientes de confirmación.
Flujo de datos	<ol style="list-style-type: none"> 1. Navegación a pestaña de “Reservas pendientes”. 2. Visualizar una lista de reservas pendientes con detalles como fechas, cliente y estado.
Pre-condiciones	El servidor web se encuentre en funcionamiento. Debe haber al menos una reserva pendiente en el sistema.
Post-condiciones	Se debe visualizar una lista de reservas pendientes de confirmación.
Requisitos	RF20, RF25

Tabla 4.16: CU16 - Consultar reservas pendientes

Número	CU-17
Nombre	Confirmar reserva
Actores	Administrador
Descripción	Permite al administrador aprobar una reserva pendiente de confirmación.
Flujo de datos	<ol style="list-style-type: none"> 1. Seleccionar una reserva pendiente de la lista. 2. Aprobar la reserva. 3. Enviar notificación de aprobación al cliente.
Pre-condiciones	El servidor web se encuentre en funcionamiento. Debe haber al menos una reserva pendiente en el sistema.
Post-condiciones	La reserva seleccionada cambiará su estado a aprobada. El cliente recibirá una notificación de aprobación.
Requisitos	RF20, RF21

Tabla 4.17: CU17 - Confirmar reserva

Número	CU-18
Nombre	Denegar reserva
Actores	Administrador
Descripción	Permite al administrador rechazar una reserva pendiente de confirmación.
Flujo de datos	<ol style="list-style-type: none"> 1. Seleccionar una reserva pendiente de la lista. 2. Rechazar la reserva. 3. Enviar notificación de rechazo al cliente.
Pre-condiciones	El servidor web se encuentre en funcionamiento. Debe haber al menos una reserva pendiente en el sistema.
Post-condiciones	La reserva seleccionada cambiará su estado a rechazada. El cliente recibirá una notificación de rechazo.
Requisitos	RF20, RF22

Tabla 4.18: CU18 - Denegar reserva

Número	CU-19
Nombre	Raspar datos web
Actores	Sistema
Descripción	Permite al sistema extraer datos turísticos para mostrarlos en la aplicación web.
Flujo de datos	<ol style="list-style-type: none"> 1. Acceder a aplicaciones web de ayuntamientos. 2. Extraer los datos relevantes. 3. Almacenar los datos en la base de datos. 4. Actualizar la información mostrada en la aplicación web.
Pre-condiciones	Las aplicaciones web de los ayuntamientos deben estar disponibles.
Post-condiciones	Los datos extraídos se almacenarán y se mostrarán en la aplicación web.
Requisitos	RF6, RF8, RF17

Tabla 4.19: CU19 - Raspar datos web

Número	CU-20
Nombre	Extraer datos actividades
Actores	Sistema
Descripción	Permite al sistema extraer datos de actividades al aire libre para mostrarlos en la aplicación web.
Flujo de datos	<ol style="list-style-type: none"> 1. Conectar con las APIs de actividades al aire libre. 2. Extraer los datos relevantes. 3. Almacenar los datos en la base de datos. 4. Actualizar la información mostrada en la aplicación web.
Pre-condiciones	Las APIs de actividades al aire libre deben estar disponibles.
Post-condiciones	Los datos extraídos se almacenarán y se mostrarán en la aplicación web.
Requisitos	RF16

Tabla 4.20: CU20 - Extraer datos actividades

Número	CU-21
Nombre	Extraer datos de redes sociales
Actores	Sistema
Descripción	Permite al sistema extraer datos de redes sociales para mostrarlas en la aplicación web.
Flujo de datos	<ol style="list-style-type: none">1. Conectar con las APIs de las redes sociales.2. Extraer los datos relevantes.3. Almacenar los datos en la base de datos.4. Actualizar la información mostrada en la aplicación web.
Pre-condiciones	Las APIs de las redes sociales deben estar disponibles.
Post-condiciones	Los datos extraídos se almacenarán y se mostrarán en la aplicación web.
Requisitos	RF5, RF18

Tabla 4.21: CU21 - Extraer datos de redes sociales

Te he llevado un nuevo página, donde no sea muy ortodoxo, para que los ojos de lectura descansen de tanto tabla.

4.1.2. Diagramas de secuencia más relevantes

En las Figuras 4.3, 4.4 y 4.5 se muestran los diagramas de secuencia de primer nivel que se han diseñado para la aplicación web. En estos diagramas se pueden observar las interacciones entre los actores y el sistema para los procesos de obtención de datos climáticos, obtención de recursos web y reserva de la casa rural. Se han considerado estos tres diagramas ya que son los que componen la principal lógica de negocio de la plataforma. Existen otros componentes como la realización de formularios, o el registro e inicio de sesión que se han considerado de más bajo nivel y no se especificarán en esta sección. O otros diagramas como el de obtención de publicaciones que es exactamente igual al de obtención de recursos web, ya que se trata de una extracción de datos. La única diferencia es que la API (Application Programming Interface) envía datos formateados directamente y los recursos web se deben formatear a un formato entendible por el Frontend, y esto son detalles a más bajo nivel.

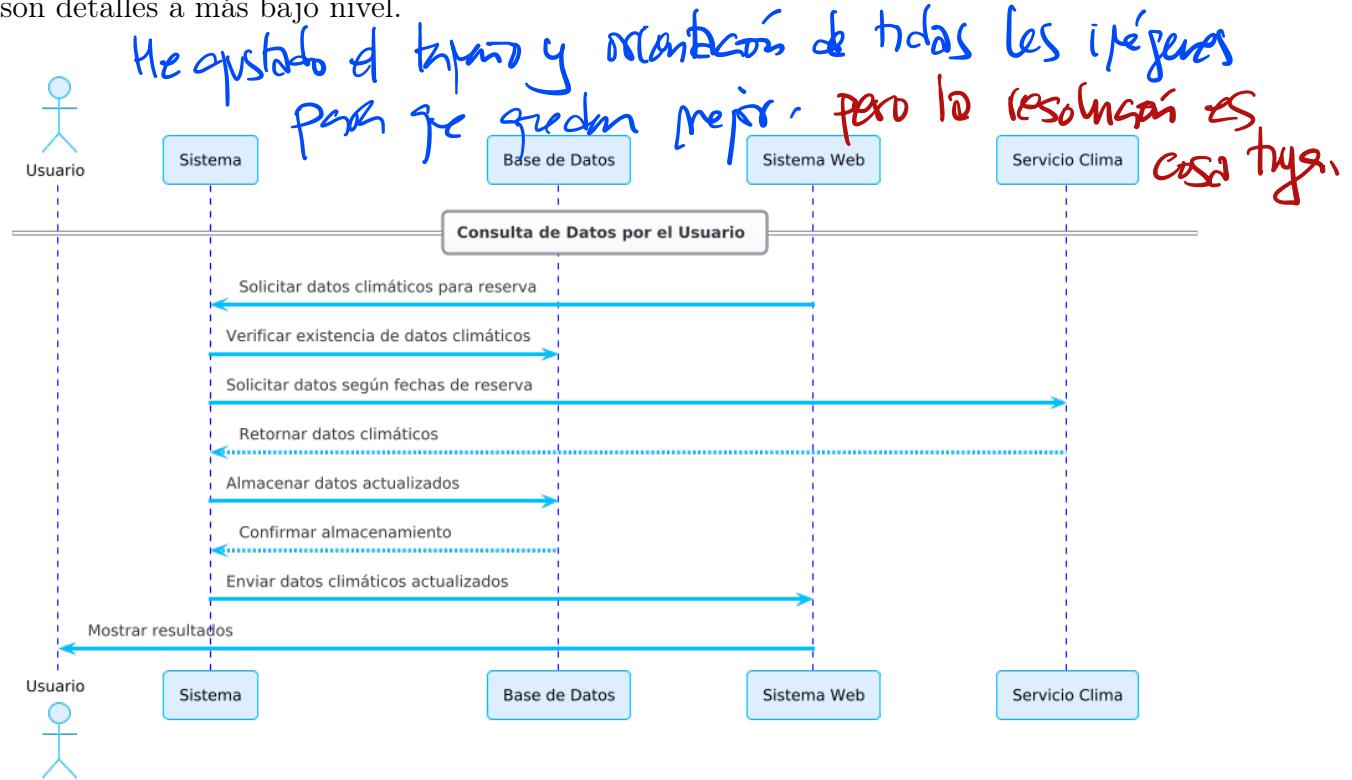


Figura 4.3: Diagrama de secuencia proceso de obtención de datos climáticos.

4.1.3. Diagrama de flujo del sistema de reservas

En la aplicación web se ha implementado un sistema de reservas que permite a los usuarios solicitar fechas para alojarse en la casa rural. Este sistema está diseñado para gestionar el ciclo de vida de una reserva, desde su creación hasta su finalización. A continuación, se describe el diagrama de actividades que representa este proceso. La Figura 4.6 representa el diagrama de actividades que modela el ciclo de vida de una reserva dentro del sistema de gestión de la casa rural. Este diagrama ilustra de forma secuencial los estados por los que pasa una reserva desde su solicitud inicial hasta su finalización, incluyendo la interacción entre el usuario cliente y el gestor.

El proceso se inicia cuando un usuario envía una solicitud de reserva. Automáticamente

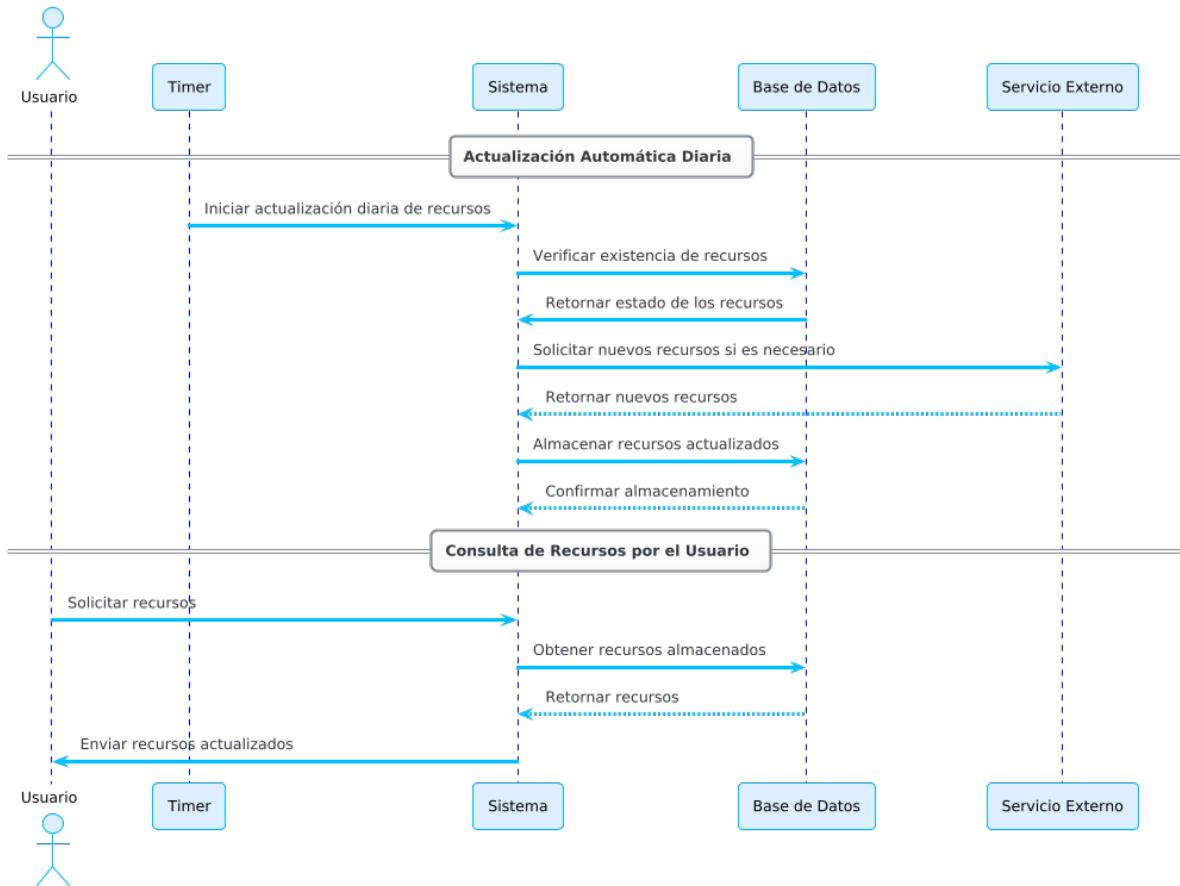


Figura 4.4: Diagrama de secuencia proceso de obtención de recursos web.

te, el sistema envía un correo de confirmación al cliente indicando que su solicitud ha sido registrada, y otro al gestor notificando la nueva petición. La reserva queda registrada en estado *PENDIENTE*.

A continuación, el gestor revisa las reservas pendientes. Si decide aceptarla, el sistema cambia el estado a *CONFIRMADA* y envía un correo al cliente con la confirmación y las instrucciones para realizar el pago. Si el gestor la deniega, el estado cambia a *CANCELADA* y se notifica al cliente incluyendo el motivo de la denegación.

Una vez confirmada, el sistema entra en un ciclo de espera hasta que se reciba el pago. Cuando este se efectúa, el estado pasa a *PAGADA*. A partir de ese momento, se activa una comprobación diaria en segundo plano para evaluar si la fecha actual ha superado el rango de fechas de la reserva.

Cuando la fecha de fin ha sido alcanzada, el sistema actualiza automáticamente el estado de la reserva a *FINALIZADA* y cambia el rol del usuario a *Cliente*. Este cambio permite al usuario visualizar un formulario para valorar su experiencia. Tras enviar la reseña, el rol del usuario se revierte a *Usuario registrado*, concluyendo así el proceso.

4.1.4. Diagrama de clases de primer nivel

En la Figura 4.7 se muestra el diagrama de clases de primer nivel que se ha diseñado para la aplicación web. En este diagrama se pueden observar las clases principales que se han definido para el desarrollo de la aplicación web y sus relaciones, sin tener en cuenta

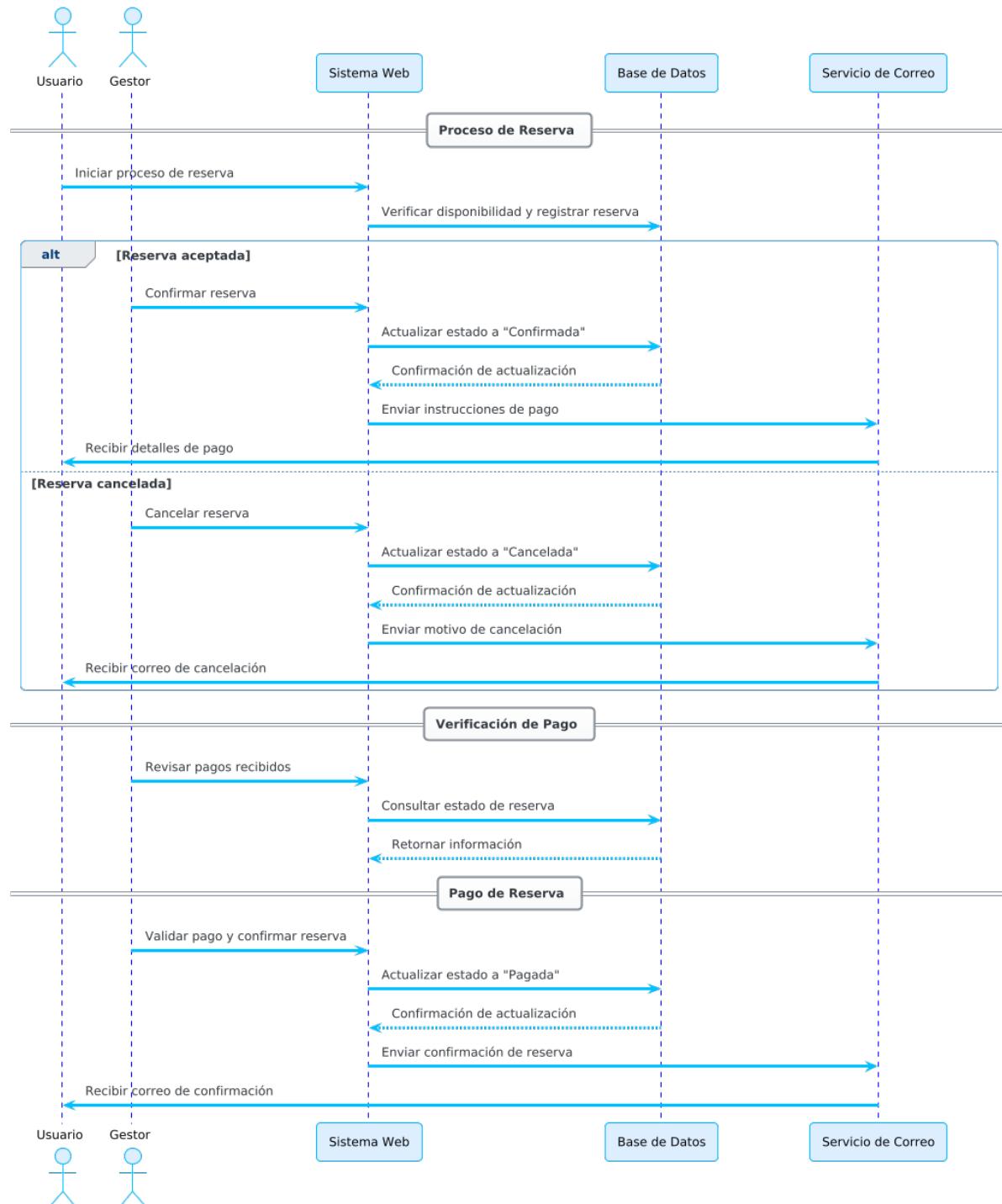


Figura 4.5: Diagrama de secuencia proceso de reserva de la casa rural.

los posibles campos necesarios a añadir en la fase de diseño técnico.

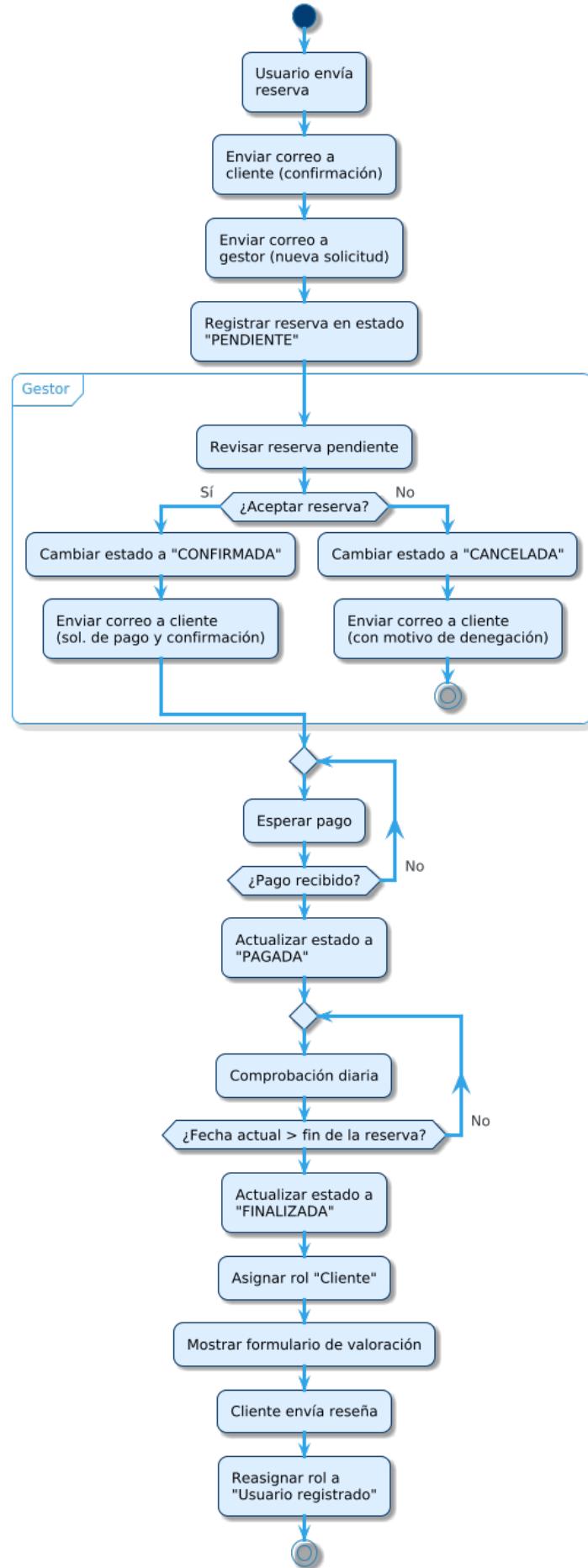


Figura 4.6: Diagrama de actividades del ciclo de vida de una reserva.

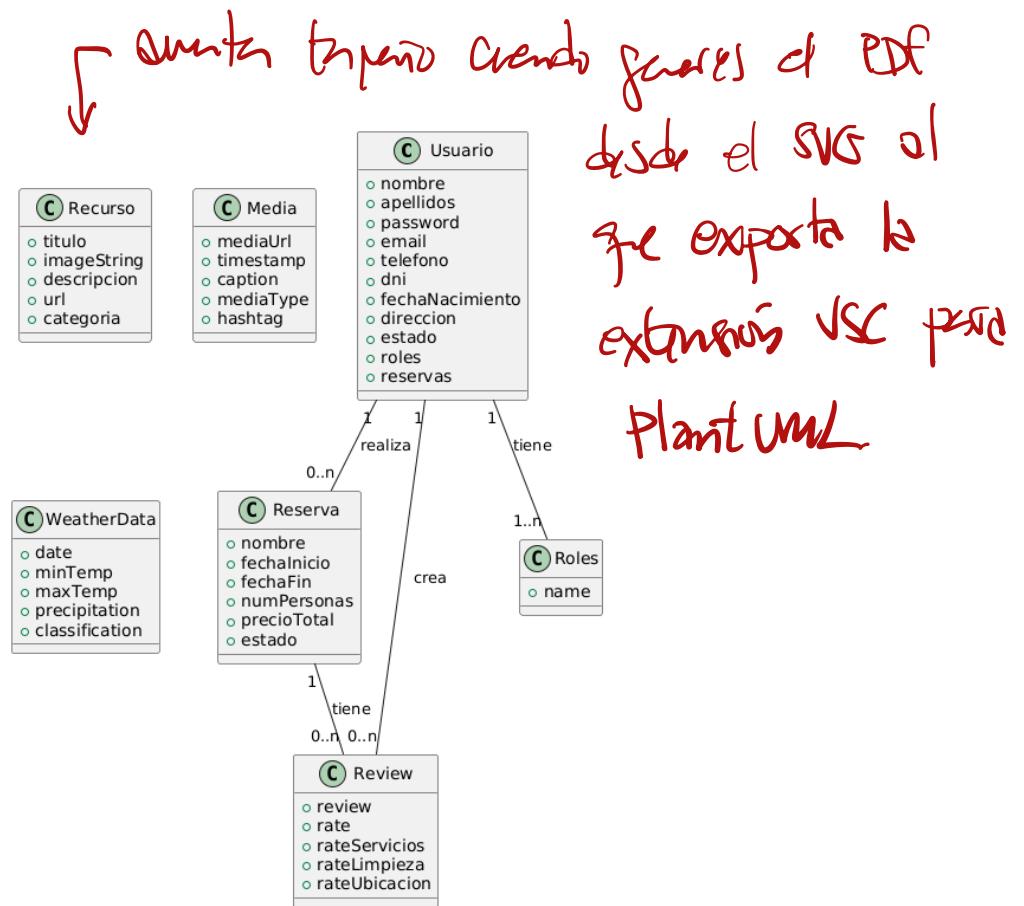


Figura 4.7: Diagrama general de clases.

El diseño te lo quedado finca gente "BONITO", con unos diagramas de componentes y despliegue muy scrittos, pero ley errores a corregir y no me convencen algunas cosas como la resolución de los errores (resolución y hiperos de diagramas de secuencia).

Capítulo 5

Diseño

En esta sección se va a introducir el diseño o arquitectura del sistema mediante diagramas de componentes de cada uno de los componentes lógicos que forman los subsistemas de la aplicación. Además, se va a especificar diagramas de secuencia más precisos para cada una de las funcionalidades de los componentes de cada subsistema. Para finalizar, se muestra un diagrama de despliegue en el que se podrá observar los nodos a desplegar y el empaquetamiento de cada componente.

5.1. Arquitectura de componentes

5.1.1. Arquitectura de Componentes del Sistema Web

La Figura 5.1 muestra el diagrama de componentes del sistema desarrollado, el cual está estructurado en tres capas tecnológicas principales: **Frontend Web**, **Backend Principal** y **Subsistemas especializados y servicios externos**. Esta arquitectura modular permite una escalabilidad horizontal clara, así como la integración sencilla de nuevas funcionalidades.

1. Plataforma Web Frontend

Implementada con Angular y tecnologías estándar como HTML5, la interfaz de usuario está compuesta por varios componentes funcionales:

- **Login, Register, Gestor, Inicio, Reservas, LaCasa, Entorno, Recomendaciones, ComoLlegar:** Son los componentes visuales que representan cada una de las vistas principales.
- **Servicios asociados:** cada vista se comunica con el backend a través de servicios como:
 - AuthService: gestión de autenticación y login.
 - RegistroService: registro de usuarios.
 - ReserveService: gestión de reservas.
 - WeatherService: consulta de clima.

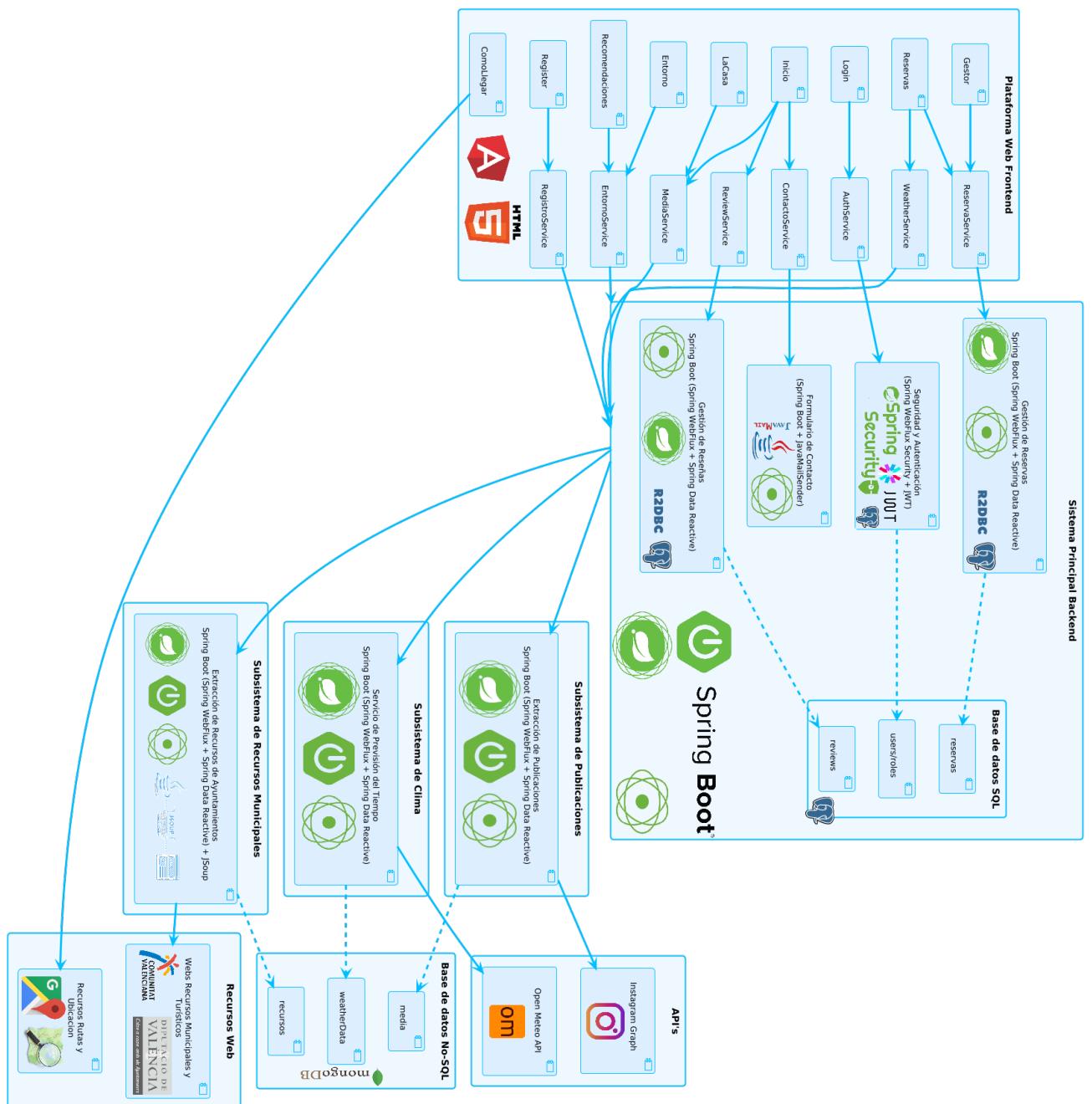


Figura 5.1: Diagrama de componentes del sistema.

Lo he visto y recordado para no quedarme lejos
porque esto es super acortado.

- **ContactoService:** envío de formularios de contacto.
- **ReviewService:** envío y recuperación de reseñas.
- **MediaService:** carga de publicaciones desde redes sociales.
- **EntornoService:** obtención de datos de recursos turísticos.

2. Sistema Principal Backend

Este bloque implementa la lógica de negocio central mediante Spring Boot, Spring WebFlux, Spring Security y Spring Data Reactive. Sus componentes son:

- **Gestión de Reservas:** permite crear, actualizar y cancelar reservas, accediendo a la base de datos PostgreSQL mediante R2DBC.
- **Seguridad y Autenticación:** implementada con Spring Security y JWT, garantiza la protección del acceso a recursos.
- **Formulario de Contacto:** usa JavaMailSender para el envío de mensajes desde la web.
- **Gestión de Reseñas:** permite a los usuarios enviar valoraciones sobre su experiencia.

Además, toda la persistencia en el sistema principal está delegada en la **base de datos SQL PostgreSQL**, que incluye las tablas `reservas`, `users/roles` y `reviews`.

3. Subsistemas especializados

Los microservicios siguientes se encargan de tareas específicas, todos desarrollados con Spring Boot y tecnologías reactivas:

- **Subsistema de Publicaciones:**
 - Conecta con la **API de Instagram Graph** para importar publicaciones según hashtags.
 - Almacena los datos en la colección `media` de MongoDB.
- **Subsistema de Clima:**
 - Obtiene predicciones meteorológicas usando **Open Meteo API**.
 - Los datos se almacenan en la colección `weatherData` en MongoDB.
- **Subsistema de Recursos Municipales:**
 - Utiliza JSoup para extraer información desde **webs municipales y turísticas**.
 - Almacena los resultados en la colección `recursos` en MongoDB.

4. Integraciones Externas

- **Bases de datos NoSQL:** gestionadas con MongoDB, organizadas en colecciones específicas para medios, clima y entorno.
- **APIs externas:** el sistema accede a fuentes de datos abiertas como **Instagram Graph API** y **Open Meteo API**.
- **Recursos Web:** portales oficiales de entidades como *Turisme Comunitat Valenciana* y *Diputació de València*.

5.2. Sistema principal

En el sistema principal, el componente de reservas gestiona el flujo completo asociado a la realización y gestión de reservas por parte de los usuarios y del gestor. A continuación, se describen los diferentes procesos soportados, ilustrados mediante diagramas de secuencia.

- En la Figura 5.2, se presenta el proceso de **confirmación de reserva**. Un **Usuario** envía una solicitud con los datos necesarios para registrar una reserva. El controlador valida los datos y consulta si el usuario existe en la base de datos. Tras almacenar la reserva en el repositorio, se envía un correo de confirmación mediante el servicio de correo electrónico. El sistema responde con un código 201 **CREATED**.
- La Figura 5.3 describe la **consulta de reservas por parte del gestor**. El **Gestor** solicita todas las reservas ordenadas por fecha de inicio. La lógica del servicio consulta el repositorio y retorna un **FluxReserva** con los resultados.
- En la Figura 5.4, se representa el proceso de **obtención de reservas próximas**. En este caso, cualquier **Usuario** puede consultar las reservas a partir de la fecha actual. El sistema devuelve un **FluxTuple2LocalDate**, **LocalDate** con los intervalos de fechas de las reservas, para que se visualicen los días reservados en el calendario por otros usuarios.
- La Figura 5.5 detalla el flujo de **actualización del estado de una reserva**. El **Gestor** envía una petición para modificar el estado. Dependiendo del nuevo estado, se ejecutan acciones distintas: si se marca como **PAGADA**, se confirma el pago y se envía un correo de confirmación; si se marca como **CANCELADA**, se notifica la cancelación. En ambos casos, se actualiza la entidad y se guarda en la base de datos.
- Por último, la Figura 5.6 ilustra la **consulta de reservas por usuario**. A través del correo electrónico, un **Usuario** puede recuperar sus reservas. El sistema filtra por email y responde con un **FluxReserva**.

Después, se encuentra el componente de gestión de reseñas. En este módulo, los usuarios que hayan realizado una reserva (rol **CLIENTE**) podrán dejar reseñas en la aplicación, las cuales podrán ser consultadas por cualquier otro usuario. Este componente se divide en dos funcionalidades principales: añadir una reseña y consultar las reseñas existentes.

En la Figura 5.7 se representa el flujo completo para el proceso de añadir una reseña. El usuario autenticado, con rol **CLIENTE**, envía una solicitud **POST** con los datos de la

Exceso de detalle técnico si también añades todo el código. Cuando ayer los ojos estaban malos (y el tablero estaba
 85 diluido) voy a ver cajitas azules.

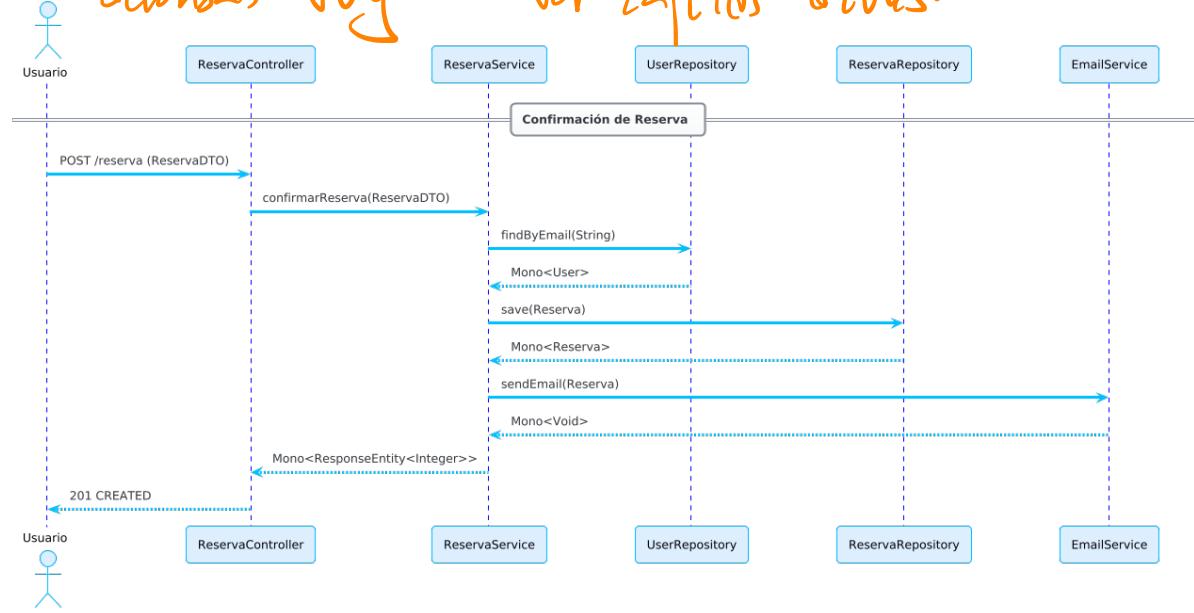


Figura 5.2: Diagrama de secuencia: Confirmación de reserva.

↳ repiti en grises en todos

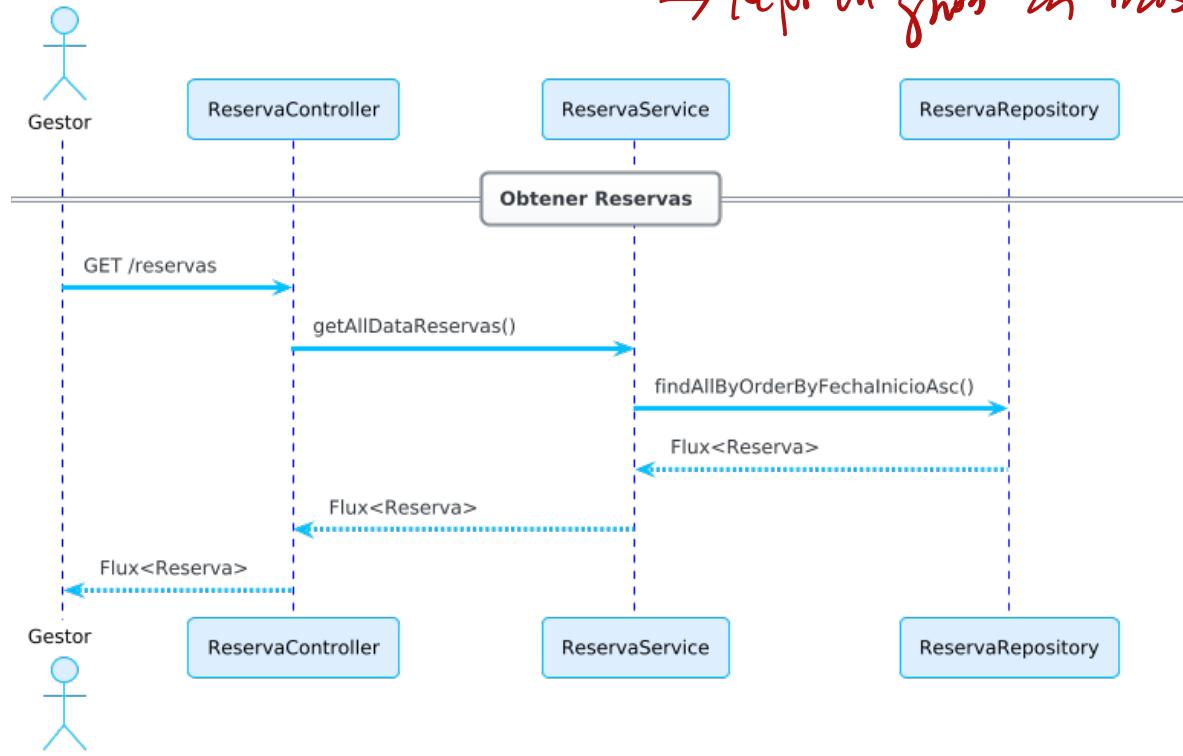


Figura 5.3: Diagrama de secuencia: Obtener reservas (Gestor).

reseña. El controlador delega la petición al servicio, que identifica al usuario por medio del sistema de autenticación de Spring y lo inyecta en la petición. Luego, se consulta el repositorio de usuarios para recuperar la entidad correspondiente y se elimina su rol de CLIENTE, dado que este rol es transitorio y únicamente se asigna tras una estancia validada. Posteriormente, se guarda la reseña en el repositorio de reseñas y se devuelve una respuesta de tipo MonoResponseEntity<Review> al cliente.

En la Figura 5.8 se ilustra el proceso de consulta de reseñas. En este caso, cualquier

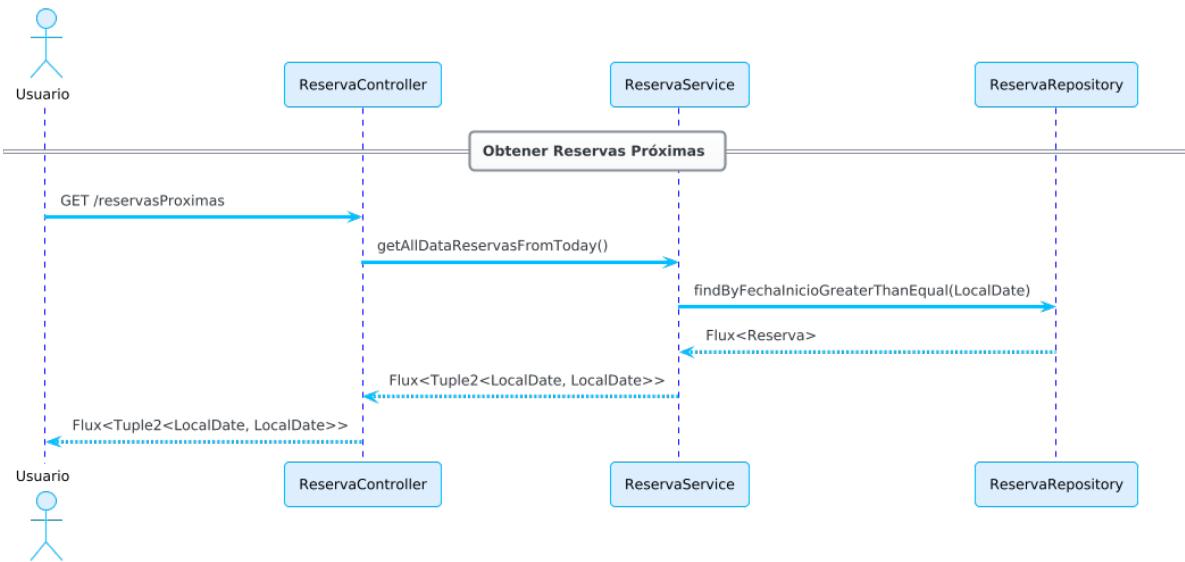


Figura 5.4: Diagrama de secuencia: Obtener reservas próximas.

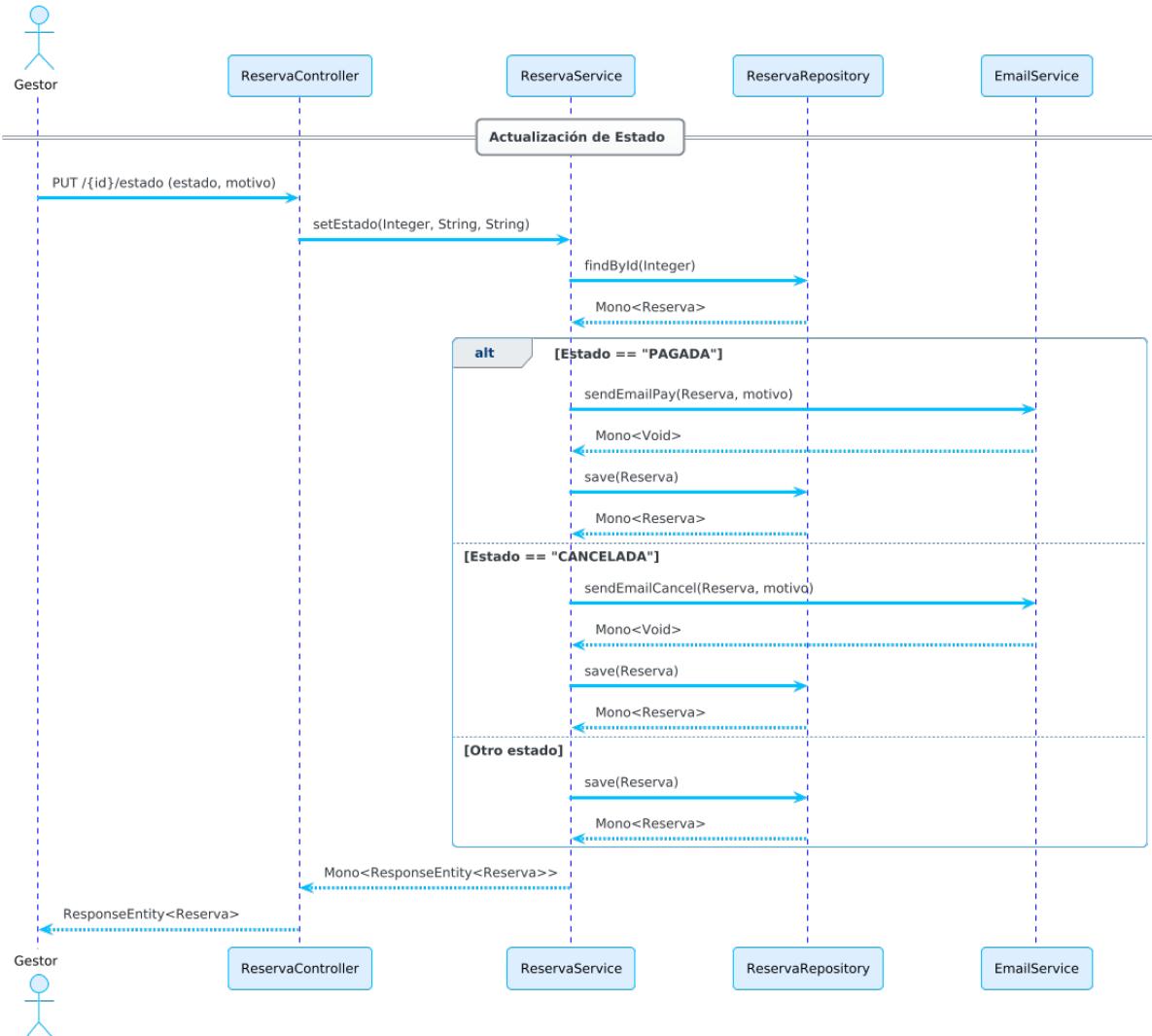


Figura 5.5: Diagrama de secuencia: Actualización de estado.

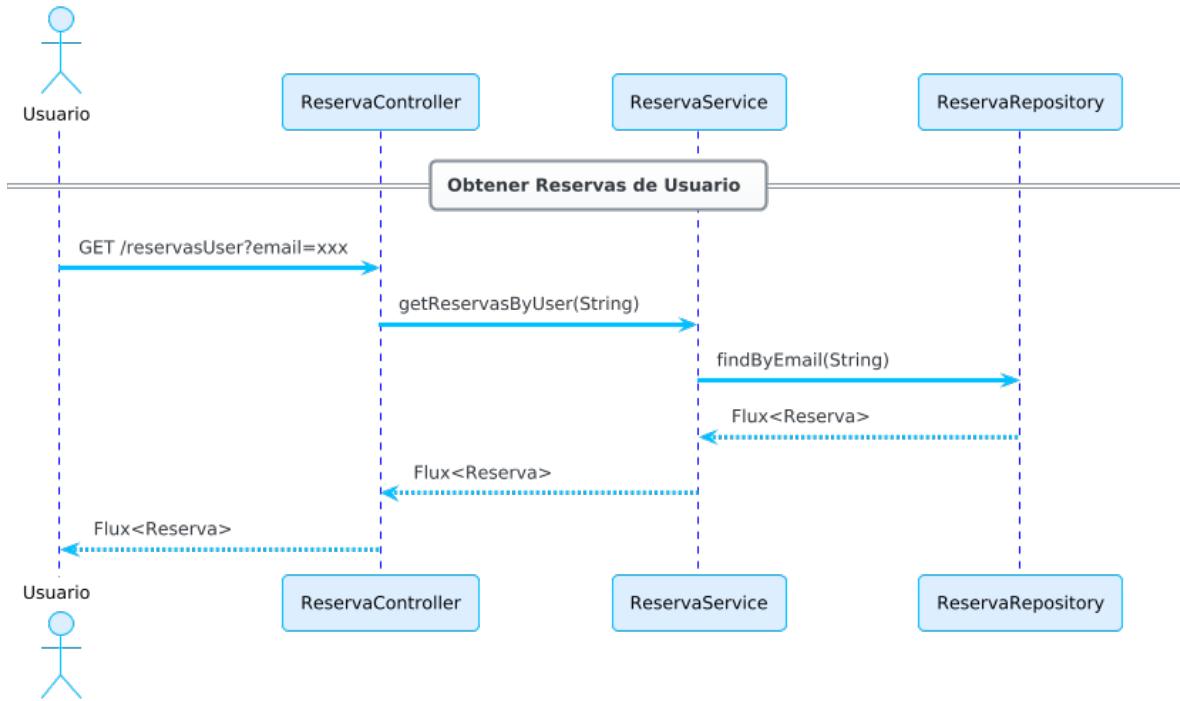


Figura 5.6: Diagrama de secuencia: Obtener reservas por usuario.

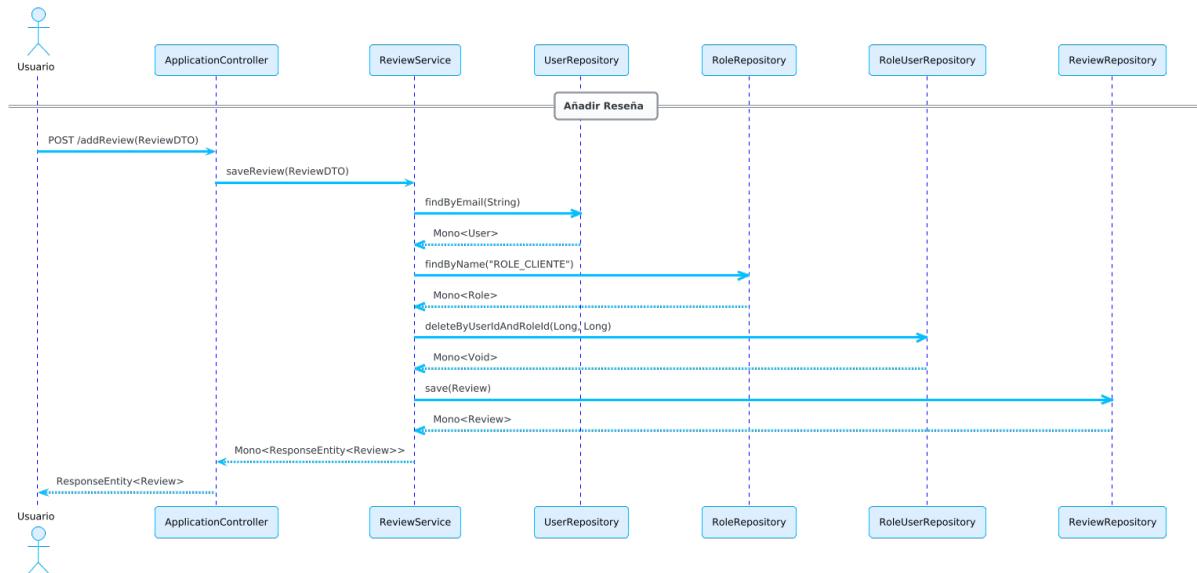


Figura 5.7: Diagrama de secuencia del proceso de añadir una reseña.

usuario puede enviar una solicitud GET para recuperar las reseñas almacenadas. El controlador redirige la petición al servicio, que a su vez consulta el repositorio de reseñas mediante una operación reactiva que devuelve un `FluxReview` ordenado por fecha de creación en orden descendente. La respuesta se entrega finalmente al cliente en forma de una lista de reseñas disponibles.

A continuación, se encuentra el componente de contacto, el cual permite a los usuarios enviar mensajes directamente al administrador de la casa rural a través de un formulario. En la Figura 5.9 se observa el diseño de la lógica de este proceso. El usuario envía una solicitud POST con los datos del formulario, incluyendo nombre, email y mensaje. El con-

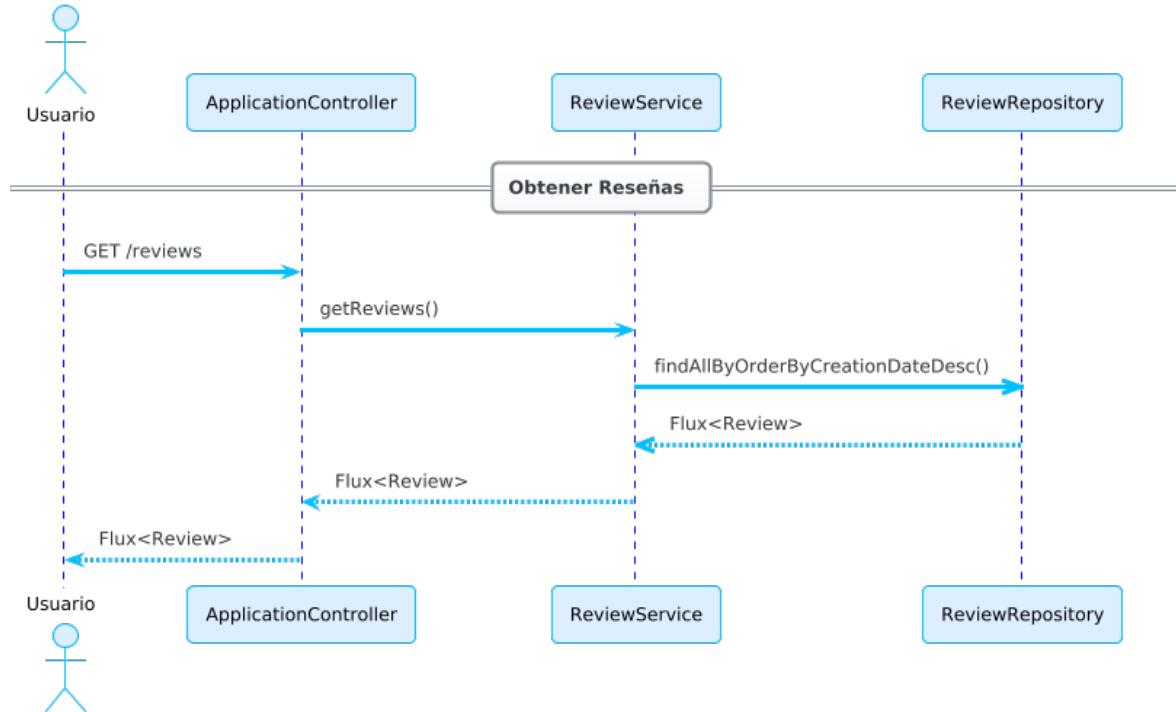


Figura 5.8: Diagrama de secuencia del proceso de consulta de reseñas.

trolador recibe la petición y delega el envío del correo al servicio correspondiente.

El `ContactService` construye un mensaje de tipo `SimpleMailMessage` utilizando los datos proporcionados por el usuario y lo envía mediante el componente `JavaMailSender`, el cual está configurado en el sistema. Una vez enviado el correo electrónico, el servicio devuelve una respuesta de tipo `MonoContact>`, que es encapsulada en un objeto `ResponseEntity` para ser devuelta al usuario con un estado HTTP 200 si todo ha ido correctamente, o 400 en caso de error.

Finalmente, se encuentra el componente encargado de la seguridad el cual se especifica mediante configuración y infraestructura. Por lo tanto, se detallará en la sección de despliegue 5.8 y en la implementación 6.1

5.3. Subsistema de recursos web

En el subsistema de recursos web, uno de los componentes fundamentales es el servicio de raspado automatizado de contenidos turísticos. Este componente permite tanto a los usuarios acceder a los recursos almacenados como al sistema actualizar periódicamente la información mediante técnicas de *web scraping*.

5.3.1. Extracción solicitada por el usuario

En la Figura 5.10 se ilustra el comportamiento del sistema cuando un usuario interacciona con el `ExtraccionController` para consultar los recursos disponibles. Se representan dos casos diferenciados:

- Una solicitud GET a /api/v1/extraccion desencadena la invocación del méto-

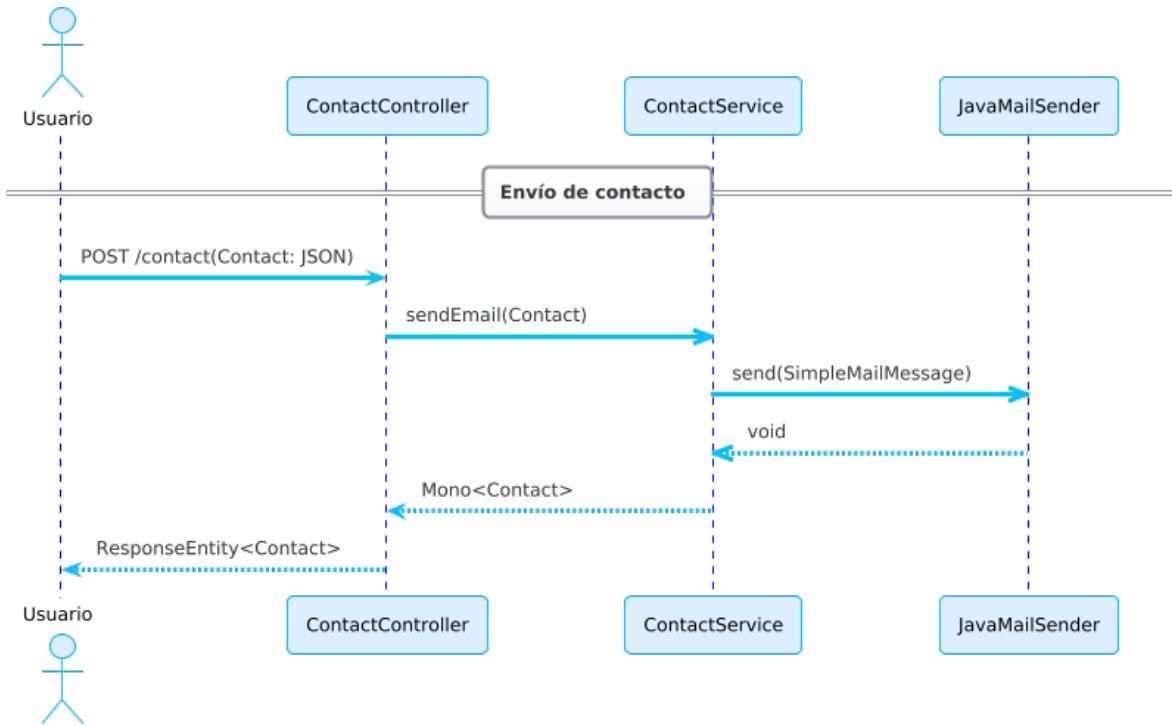


Figura 5.9: Diagrama de secuencia del componente de contacto.

do `extractAllResources()` del servicio `RecursoService`, que accede a todos los documentos de la colección mediante el repositorio reactivo `RecursoRepository` y retorna un `Flux<Recurso>`.

- En el segundo caso, cuando el usuario realiza una solicitud GET a `/api/v1/extraccionFiestas`, el sistema filtra los resultados por la categoría "fiestas" mediante el método `findByCategoria()`.

5.3.2. Extracción programada con scraping

En la Figura 5.11 se representa la lógica que ejecuta la tarea de extracción programada mediante un `@Scheduled` en el servicio `RecursoService`. Esta ejecución se realiza automáticamente cada día a las 00:00 horas y tiene como objetivo extraer y almacenar información actualizada desde diversas fuentes externas.

Durante este proceso:

- Se invoca secuencialmente el método `scrape()` del `ScraperService`, pasando como parámetros la URL base, el tipo de recurso (como "monumentos", "fiestas") y la localización (como "Chelva" o "Tuejar").
- Internamente se llama a `ensureCollectionExists()` para asegurar que la colección `recursos` existe en MongoDB usando `ReactiveMongoTemplate`.
- Se realiza una conexión con cada URL mediante `JSoup` y se procesan los resultados con métodos específicos como `processChelvaMonumentosAldeas()`, `processTuejarPatrimonioNacional()` o `processTuejarFiestas()`.

infosis : (enph)

[foto marc hilera]

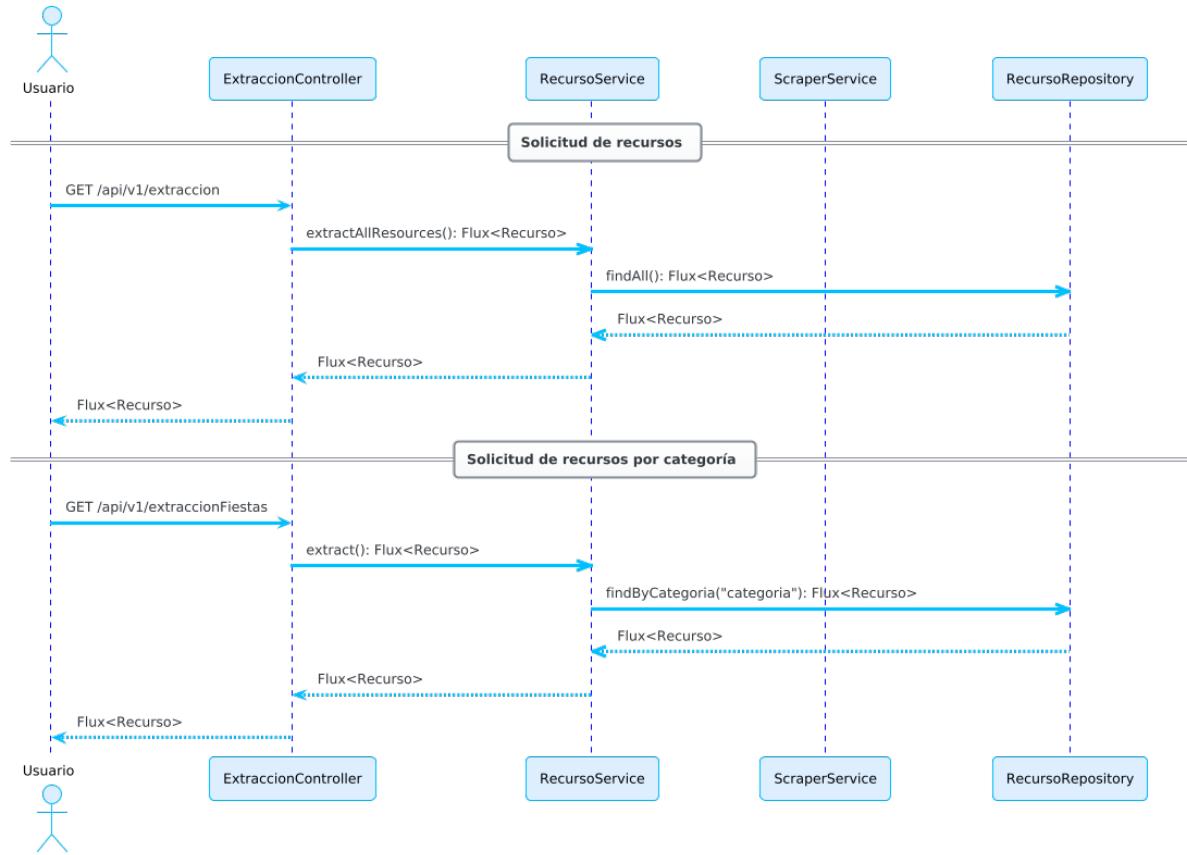


Figura 5.10: Diagrama de secuencia para solicitudes de recursos por parte de usuarios.

- Cada recurso se almacena en MongoDB a través del RecursoRepository.
- Finalmente, se consulta la base de datos para verificar el estado actualizado de todos los recursos.

5.4. Subsistema de publicaciones

En el subsistema de publicaciones el componente a especificar es el encargado de realizar la llamada a la API (Application Programming Interface) de Instagram para obtener publicaciones públicas. En la Figura 5.12 se representa cómo interactúan los distintos elementos del sistema para gestionar consultas, así como la extracción y actualización de publicaciones desde la red social.

Cuando un **usuario** realiza una solicitud GET a /api/v1/media, el MediaController llama al MediaService, que busca las publicaciones en el repositorio (MediaRepository). Si no se encuentra ninguna entrada relevante, el servicio puede invocar la API (Application Programming Interface) de Instagram a través del cliente HTTP reactivo (WebClient), para recuperar y actualizar los datos de forma dinámica.

Además, este proceso de sincronización se ejecuta automáticamente cada día a las 00:00 mediante un temporizador (Timer). El servicio obtiene el token de acceso desde el sistema de archivos mediante el InstagramTokenService, realiza la solicitud a la API (Application Programming Interface) de Instagram y, en función de si la publicación ya existe en la base de datos o no, realiza una de las siguientes acciones:

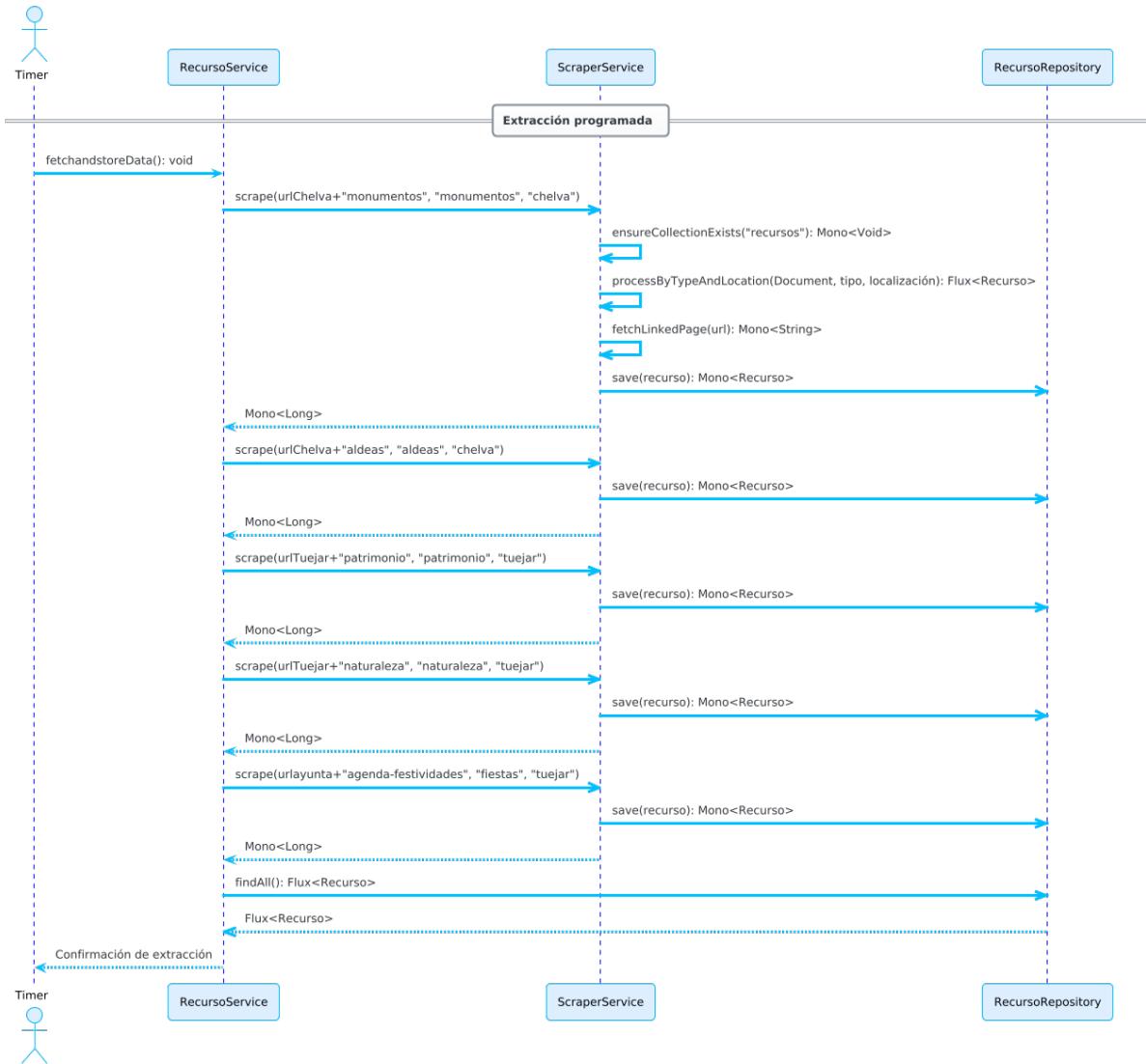


Figura 5.11: Diagrama de secuencia de la extracción programada de recursos web.

- **Guardar nueva publicación:** Si el identificador no se encuentra registrado, se almacena la nueva publicación.
- **Actualizar publicación existente:** Si la publicación ya existe, se actualiza su URL de medios.

Este comportamiento condicional se resume mediante una estructura `alt` que permite distinguir entre ambos flujos. El resultado del proceso se devuelve al `Timer` como indicativo de finalización.

Por otro lado, la Figura 5.13 muestra el procedimiento automatizado de renovación del token de acceso, esencial para mantener la conexión con la [API \(Application Programming Interface\)](#). Cada 30 días, el `InstagramTokenService` accede al sistema de archivos (`Filesystem`) para leer el token actual y realiza una solicitud a la ruta `refresh_access_token`. Una vez recibido el nuevo token, lo guarda de nuevo en el sistema de archivos. Este proceso garantiza la operatividad continua del sistema sin necesidad de intervención manual.

→ Revisa todos los acronymos, aquí y arriba, para que estén en su punto correcto.

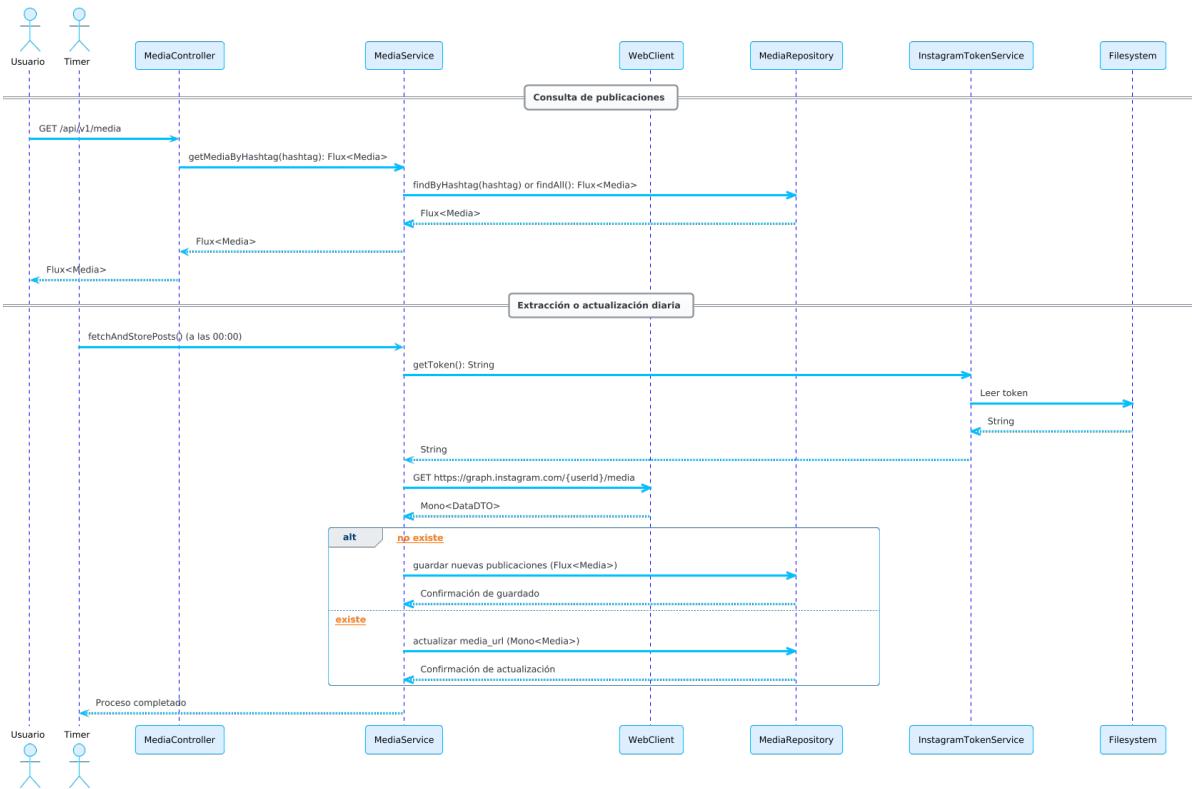


Figura 5.12: Diagrama de secuencia de consulta, extracción y actualización de publicaciones desde la API (Application Programming Interface) de Instagram.

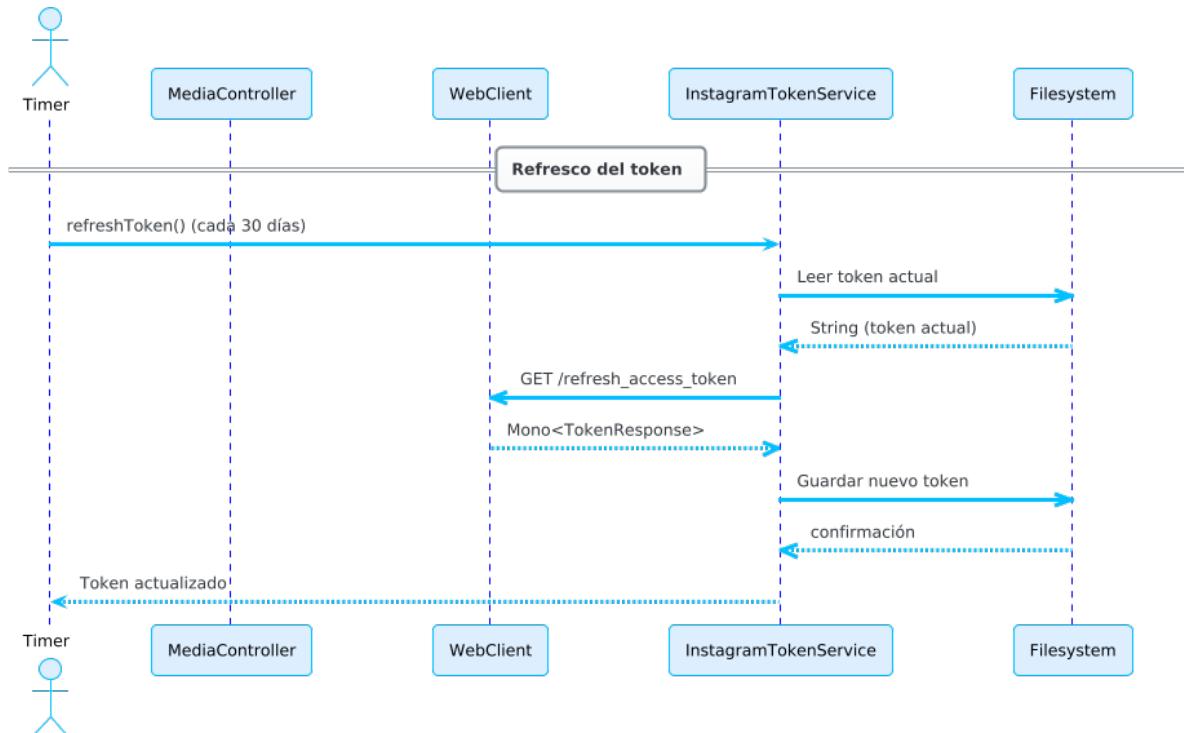


Figura 5.13: Diagrama de secuencia del proceso de refresco del token de acceso (Subsistema de publicaciones).

5.5. Subsistema de previsión del tiempo

En el subsistema de previsión del tiempo, el componente a especificar es el encargado de obtener información meteorológica de cara a la reserva del usuario. En la Figura 5.14 se describe el proceso completo llevado a cabo cuando un usuario realiza una solicitud para consultar el pronóstico del tiempo en un rango de fechas determinado.

El flujo comienza cuando el **usuario** realiza una petición GET a /api/v1/weather, incluyendo dos parámetros de fecha. El WeatherController delega esta solicitud al WeatherService, que realiza una iteración sobre cada fecha individual del rango indicado. Para cada una de estas fechas, primero intenta recuperar los datos desde la base de datos documental (WeatherDataRepository). Ruta nra linea

En caso de que los datos no se encuentren almacenados, el sistema evalúa si la fecha es histórica o si corresponde a una predicción futura. Si se trata de una fecha histórica (pasada o más allá de los próximos 15 días), se realiza una petición a la API de historial de Open Meteo. Si es una predicción (hasta 15 días en el futuro), se consulta la API de pronóstico. En ambos casos, la respuesta se procesa y se clasifica (por ejemplo: “Lluvioso”, “Frío”, “Caluroso”...), y los datos resultantes se guardan en el repositorio para su reutilización futura.

Finalmente, el servicio devuelve todos los resultados al controlador, que responde al usuario con la lista de datos meteorológicos correspondiente.

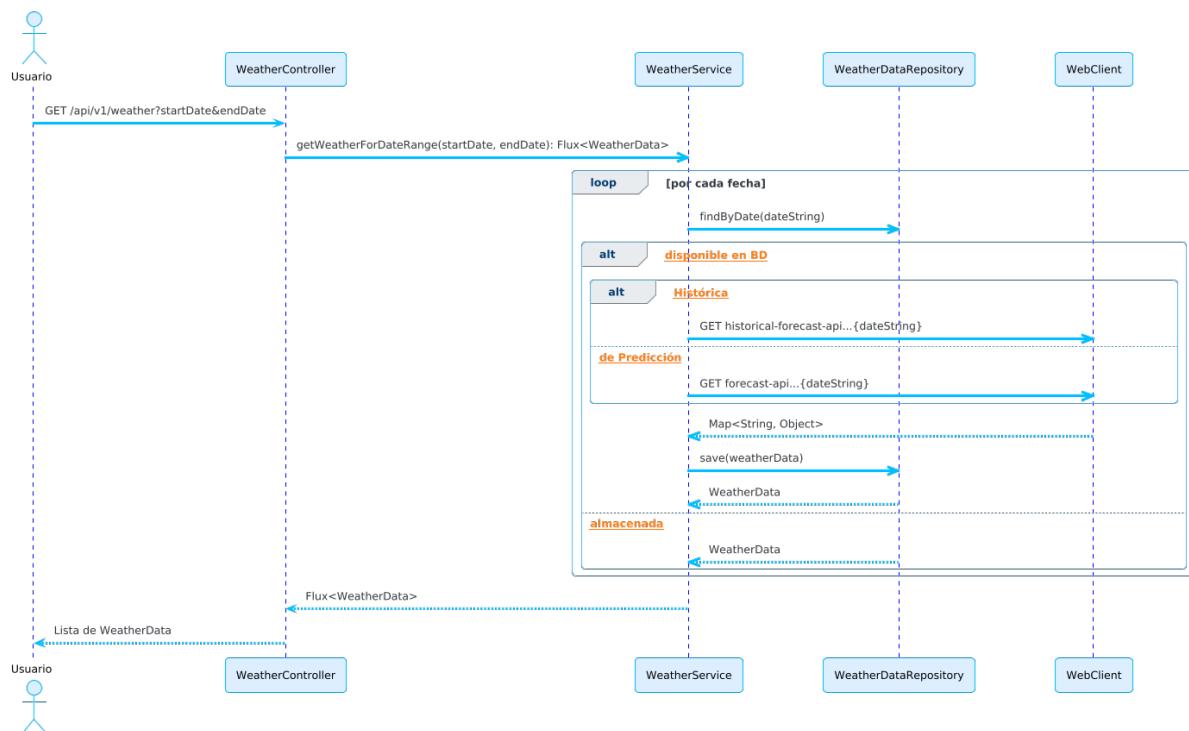


Figura 5.14: Diagrama de secuencia del componente de obtención de previsión del tiempo (Subsistema de previsión del tiempo).

5.6. Diagrama de clases

A continuación, en la Figura 5.15 se presenta el diagrama de clases que describe las principales entidades de un sistema de gestión de usuarios, reservas, opiniones y recursos. Este diagrama es fundamental para la comprensión de cómo interactúan las diferentes clases en el sistema.

Este diagrama muestra la estructura y relaciones entre las clases principales del sistema. La clase **Usuario** contiene atributos como nombre, apellidos, email, teléfono, entre otros. Esta clase también tiene métodos relacionados con el registro y obtención de usuarios y roles. La clase **Reserva** se encarga de la gestión de reservas, incluyendo la confirmación, cancelación y obtención de datos relacionados con las mismas. La clase **Review** gestiona las opiniones de los usuarios sobre los servicios, permitiendo registrar y obtener reseñas. Además, la clase **Recurso** gestiona los recursos de tipo información, como eventos y actividades, con métodos para obtener datos mediante scraping. La clase **Media** se encarga de manejar la información multimedia, como fotos y publicaciones en redes sociales, y la clase **WeatherData** almacena información relacionada con el clima, que puede ser consultada por rango de fechas.

Las relaciones entre las clases están representadas con las líneas de asociación. Por ejemplo, un **Usuario** puede tener múltiples **Reservas** y **Reviews**, mientras que una **Reserva** puede tener varias **Reviews**. Además, cada **Usuario** tiene uno o más **Roles**, los cuales determinan sus permisos dentro del sistema.

5.7. Modelo de datos

En cuanto a modelos de datos físicos se han definido tres modelos de datos para determinar las colecciones a generar en MongoDB y uno para las tablas relacionales en PostgreSQL.

5.7.1. Modelo físico de la base de datos relacional (PostgreSQL)

La Figura 5.16 muestra el modelo físico de datos de la base de datos relacional, implementado en PostgreSQL. El diagrama entidad-relación describe la estructura de las tablas, los campos principales y las relaciones de integridad referencial existentes en el sistema.

El modelo se compone de las siguientes tablas:

users: Contiene los datos de los usuarios del sistema. Incluye campos como **id** (clave primaria, tipo SERIAL), **fecha_nacimiento**, **estado**, **fecha_registro**, **fecha_modificacion**, **fecha_baja**, **nombre**, **password**, **apellidos**, **email**, **telefono**, **dni** y **direccion**.
Nota: El campo **password** se almacena cifrado tanto a nivel de base de datos como a nivel de aplicación, siguiendo buenas prácticas de seguridad y protección de la información sensible de los usuarios. Esta tabla se encuentra referenciada por otras entidades.

roles: Define los posibles roles de usuario en el sistema. Está formada por el campo **id** (clave primaria, tipo SERIAL) y **name** (tipo VARCHAR, único).

user_role: Tabla intermedia que implementa la relación muchos-a-muchos entre usuarios y roles. Incluye los campos **user_id** y **role_id**, ambos claves foráneas que referencian

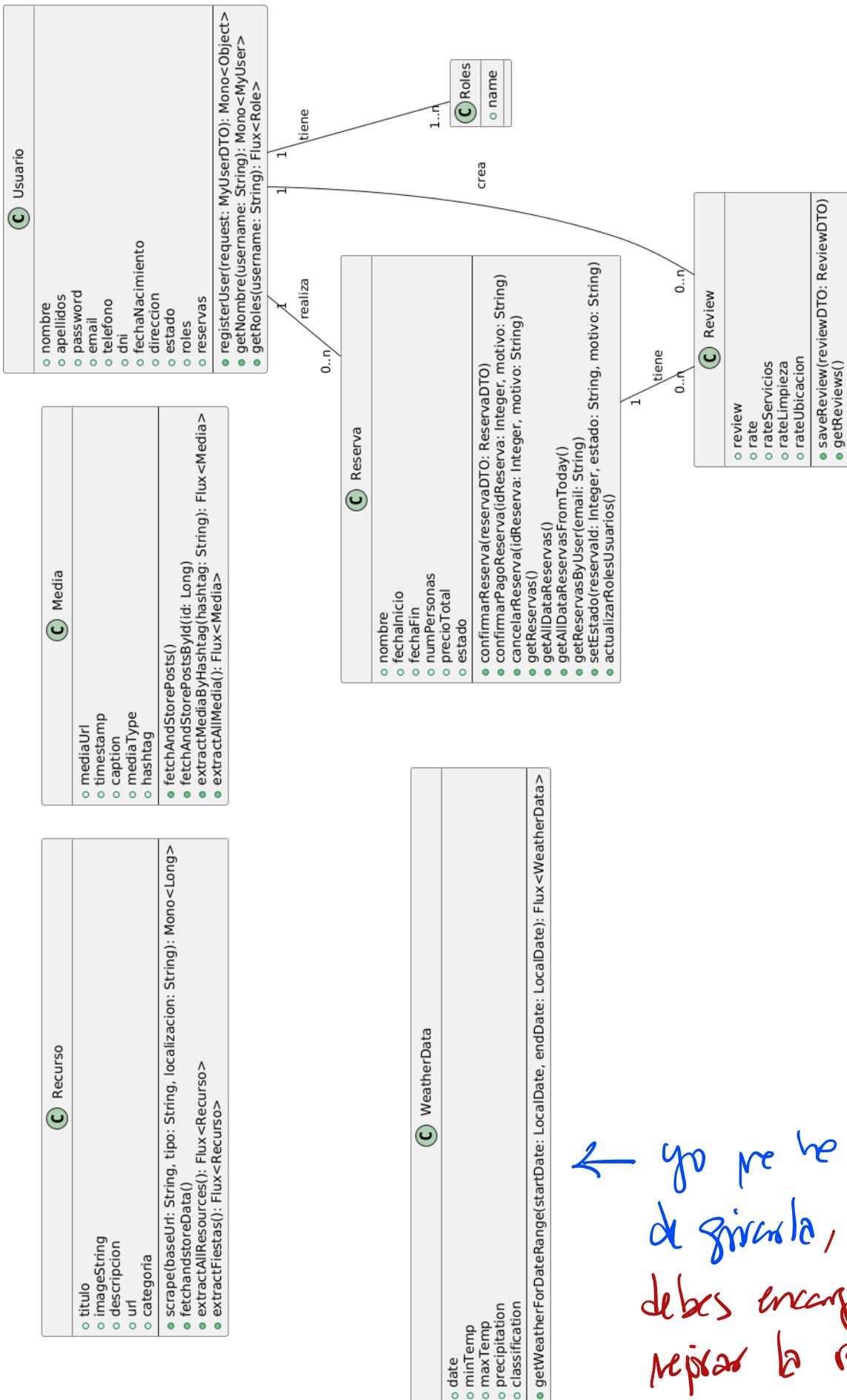


Figura 5.15: Diagrama de clases completo del sistema

↑ yo me he encargado
de girarla, tú te
debes encargar de
mejorar la respuesta

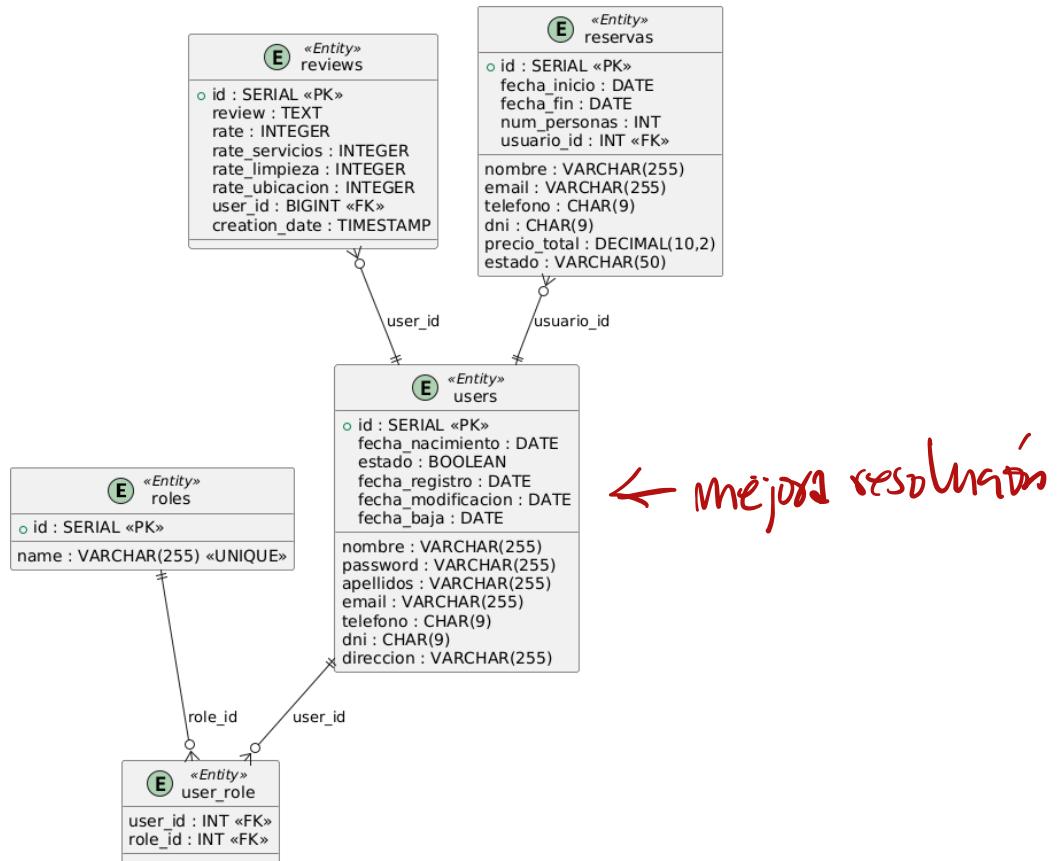


Figura 5.16: Modelo físico de la base de datos relacional PostgreSQL.

a las tablas `users` y `roles`, respectivamente.

`reviews`: Almacena las valoraciones de los usuarios. Incluye `id` (clave primaria), `review` (texto), `rate`, `rate_servicios`, `rate_limpieza`, `rate_ubicacion` (todos de tipo `INTEGER`), `user_id` (clave foránea a `users`) y `creation_date` (`timestamp`).

`reservas`: Recoge la información de las reservas realizadas. Sus campos principales son `id` (clave primaria), `fecha_inicio`, `fecha_fin`, `num_personas`, `usuario_id` (clave foránea a `users`), `nombre`, `email`, `telefono`, `dni`, `precio_total` y `estado`.

El diagrama incluye las relaciones de integridad referencial entre las tablas mediante claves foráneas (`user_id`, `role_id` y `usuario_id`), asegurando la consistencia de los datos a través de las asociaciones indicadas.

5.7.2. Modelo de datos de la base de datos no relacional MongoDB

La Figura 5.17 muestra el modelo físico de datos correspondiente a las colecciones de la base de datos MongoDB utilizadas en el sistema. El diagrama representa tres colecciones principales: `recursos`, `media` y `weather_data`, cada una con su respectiva estructura de campos.

`recursos`: Esta colección contiene documentos identificados por el campo `_id` (clave primaria), y almacena los siguientes atributos: `title`, `categoria`, `imageString`, `url`, `description` y `localizacion`, todos de tipo `String`.

`media`: Los documentos de la colección `media` disponen de los campos `_id` (clave

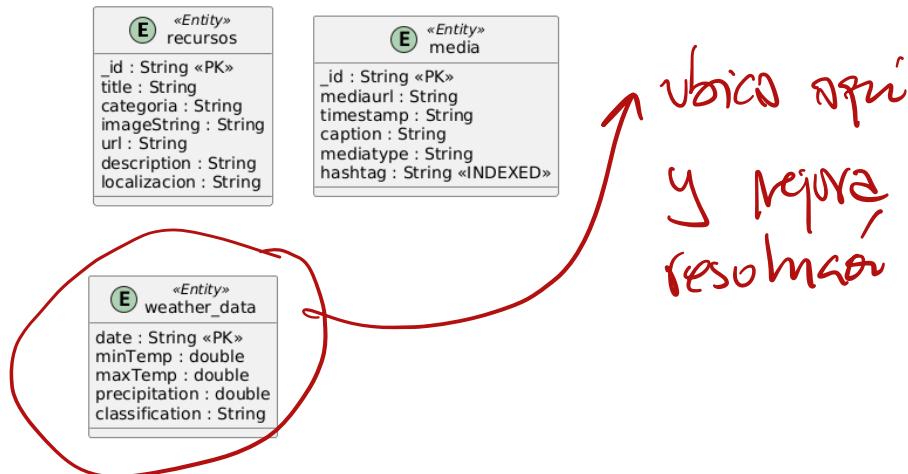


Figura 5.17: Modelo físico de la base de datos no relacional MongoDB.

primaria), **mediaurl**, **timestamp**, **caption**, **mediatype** y **hashtag**, todos de tipo **String**. El campo **hashtag** está indexado, ya que en el desarrollo se van a realizar la mayoría de búsquedas a través de él, por lo que al indexarlo mejorará la velocidad en la búsqueda de documentos.

weather_data: Cada documento en esta colección se identifica por el campo **date** (clave primaria, de tipo **String**), y contiene los campos **minTemp**, **maxTemp**, **precipitation** (de tipo **double**) y **classification** (de tipo **String**).

En el modelo presentado no existen relaciones explícitas entre colecciones, ya que MongoDB utiliza un enfoque orientado a documentos donde cada colección es independiente y puede escalar de manera autónoma. Los tipos de datos y claves principales están reflejados en el propio diagrama.

5.8. Despliegue del sistema

El presente sistema está diseñado para ejecutarse sobre una arquitectura de contenedores orquestada con [Kubernetes](#), de forma que pueda desplegarse en cualquier proveedor de servicios en la nube. En este apartado se detalla la infraestructura de despliegue propuesta, representada visualmente en la Figura 5.18 mediante un diagrama de despliegue.

El sistema se compone de una aplicación principal, encapsulada en un contenedor Docker bajo el nombre de [elrincondeeva](#), desplegada como un [Deployment](#) en Kubernetes. Esta aplicación central es responsable de coordinar la lógica de negocio y canalizar las peticiones hacia los distintos [microservicios](#) del sistema: [Publicaciones](#), [PrevisionDeTiempo](#) y [RecursosWeb](#), también empaquetados como contenedores Docker y desplegados como pods independientes dentro del clúster.

La comunicación externa con el sistema se realiza a través de un [Ingress Controller](#) que expone el servicio principal mediante protocolo [HyperText Transfer Protocol Secure \(HTTPS\)](#). Este [Ingress Controller](#) actúa como único punto de entrada al sistema, redirigiendo las peticiones entrantes al contenedor correspondiente del servicio principal, aunque se mantiene la flexibilidad para exponer directamente otros servicios si fuera necesario en un futuro.

En cuanto al almacenamiento persistente, el sistema utiliza dos bases de datos. Por un lado, una instancia de [PostgreSQL](#) para la gestión de datos relacionales, asociada

exclusivamente al contenedor principal. Esta base de datos se monta sobre un volumen persistente ([PersistentVolume](#)) y un reclamo ([PersistentVolumeClaim](#)) propio, garantizando la durabilidad de los datos. Por otro lado, los microservicios de [Publicaciones](#), [PrevisionDeTiempo](#) y [RecursosWeb](#) comparten una base de datos [MongoDB](#) desplegada como un único contenedor y montada igualmente sobre su propio par PV/PVC.

Una particularidad del microservicio [Publicaciones](#) es que requiere almacenar de forma persistente un [token](#) de acceso para consultar la [API \(Application Programming Interface\)](#) de Instagram. Para ello, se ha montado un volumen persistente específico ([token-pv](#)) a través de un PVC, que permite mantener dicho archivo de manera duradera. Esta solución evita el uso de [Secrets](#) o [ConfigMaps](#), que no serían adecuados debido a la necesidad de escritura dinámica sobre el archivo.

La Figura 5.18 resume gráficamente la organización de los servicios, contenedores, volúmenes persistentes y conexiones internas en el clúster de Kubernetes.

5.9. Despliegue en entorno local

Durante la fase de desarrollo, se ha optado por un despliegue en entorno local que permite una ejecución sencilla de cada uno de los componentes del sistema sin depender de herramientas de orquestación como [Kubernetes](#) ni de contenedores [Docker](#). En este modelo, cada microservicio se ejecuta como una aplicación independiente en la misma máquina o red local, y se comunican entre sí mediante llamadas HTTP directas a través de diferentes puertos.

La Figura 5.19 muestra el diagrama de despliegue utilizado durante el desarrollo local. En este esquema, cada componente desarrollado con [Spring Boot](#) ([SistemaPrincipal](#), [Publicaciones](#), [PrevisionDeTiempo](#) y [RecursosWeb](#)) se ejecuta como una aplicación independiente, expuesta a través de su propio puerto. Asimismo, las bases de datos [PostgreSQL](#) y [MongoDB](#) se ejecutan como servicios locales tradicionales.

A diferencia de la arquitectura en la nube descrita anteriormente, donde las comunicaciones están encapsuladas y controladas mediante un [Ingress Controller](#), en el entorno local cada servicio es accesible desde el exterior a través de [localhost](#) y su respectivo puerto. Esto implica que cualquier consumidor que conozca el puerto puede acceder directamente a cualquiera de los servicios, lo cual reduce el aislamiento y la seguridad del sistema. En el modelo de nube, sin embargo, solo se permite el acceso desde Internet al servicio principal ([SistemaPrincipal](#)) gracias al uso del [Ingress Controller](#) y protocolos seguros como [HTTPS](#).

Este enfoque local permite realizar pruebas y desarrollos de manera más rápida y sin necesidad de infraestructura adicional, aunque se recomienda restringir su uso a entornos controlados, dado que carece de mecanismos de seguridad avanzados.

→ si presentas la figura en
el principio luego no la pides
dejar
al
final.

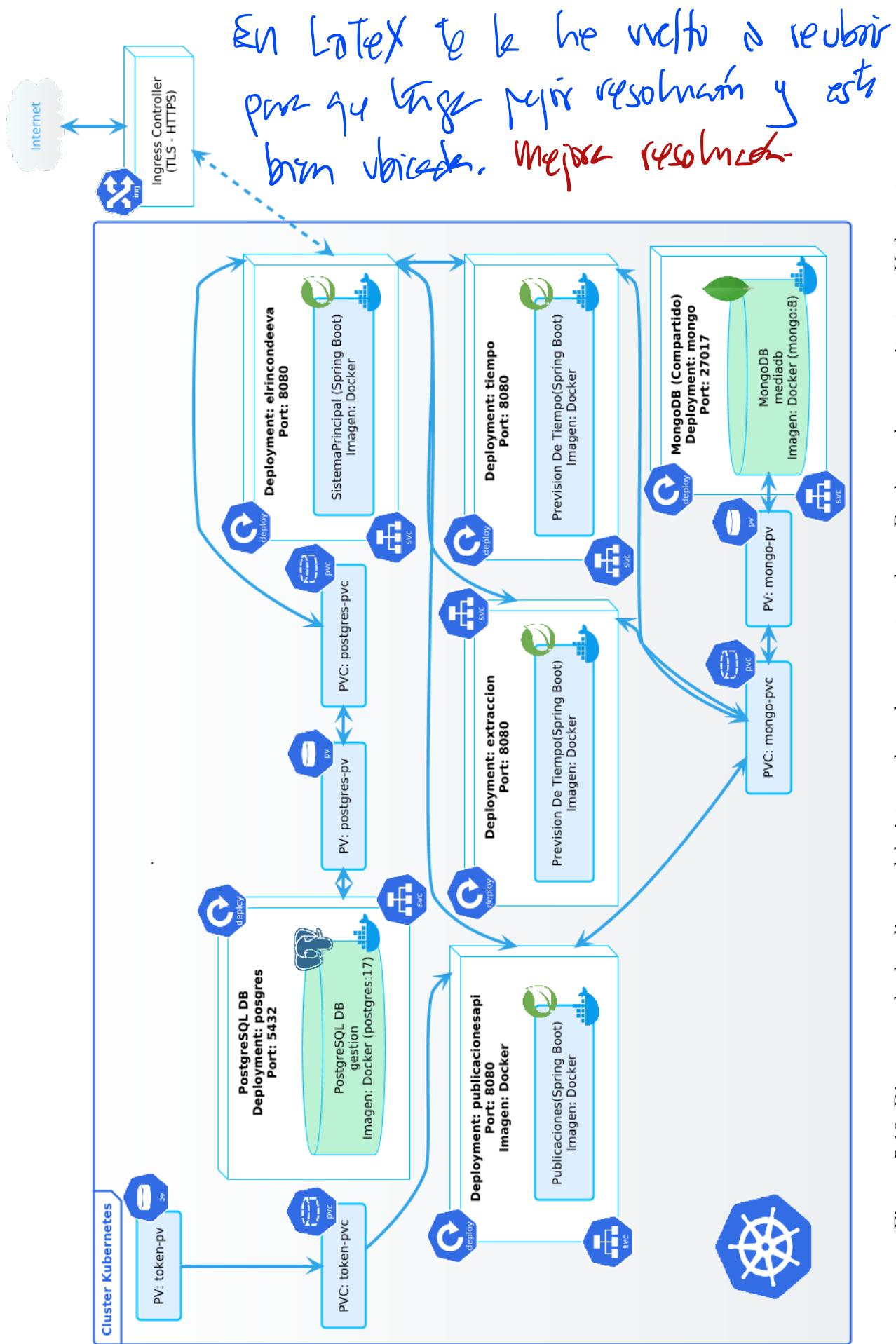


Figura 5.18: Diagrama de despliegue del sistema basado en contenedores Docker sobre arquitectura Kubernetes.

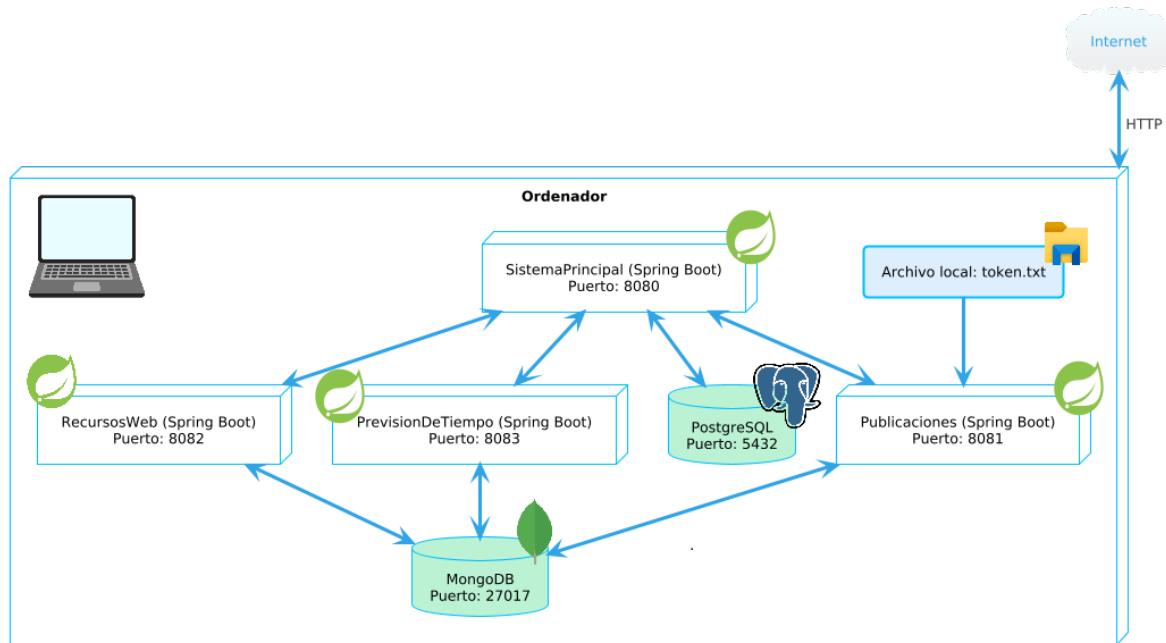


Figura 5.19: Diagrama de despliegue del sistema en entorno local.

Se ve que has hecho un gran trabajo, y no
sólo implementado, por tanto listo de antiguo
"implementación". Además, hay errores a corregir y
no me comentas algunas cosas como la resolución
Capítulo 6
de los errores.

Implementación y pruebas

En este capítulo se describe la implementación de la aplicación. Se presenta la arquitectura del sistema, los subsistemas que componen el sistema principal y el subsistema de recursos web. Además, se describen los endpoints y servicios más relevantes, así como la persistencia y acceso a datos de cada uno de los subsistemas. También, se presentan detalles finales de cómo ha quedado la aplicación y su despliegue en el servidor. Finalmente, se presentan las pruebas realizadas para verificar el correcto funcionamiento de la aplicación y su despliegue en el servidor.

6.1. Implementación sistema principal

En esta sección se describen los subsistemas que componen el sistema principal de la aplicación. Se describen los subsistemas y sus interacciones y se muestran los fragmentos de código más importantes para el funcionamiento de la aplicación.

6.1.1. Seguridad

La aplicación cuenta con un sistema de autenticación y autorización de usuarios. Para ello se ha utilizado [Spring Security \[55\]](#). Este subsistema permite gestionar los roles de los usuarios y las restricciones de acceso a los diferentes recursos de la aplicación. Se han definido los siguientes roles:

- **ADMIN**: Requiere autenticación. Tiene acceso a todas las funcionalidades de la aplicación. Además, puede gestionar las reservas de los usuarios.
- **CLIENTE**: Requiere autenticación. Tiene acceso a las funcionalidades de hacer una nueva reserva y dejar reseñas en la aplicación. Accede a todo el contenido y a contenido propio dónde puede visualizar sus reservas actuales o pasadas. Además, podrá dejar una reseña cada vez que realice una reserva y esta se encuentre en estado finalizada.
- **USER**: Requiere autenticación. Tiene acceso a la funcionalidad de hacer una nueva reserva en la aplicación. Accede a todo el contenido y a contenido propio dónde puede visualizar sus reservas actuales o pasadas.

- SIN USUARIO: este no es un rol definido dentro de las tablas del sistema. Dentro de el se encuentran solo las funcionalidades de solo lectura que no requieren autenticación y el sistema de registro o inicio de sesión.

El sistema permite la creación de usuarios, para esto se ha habilitado un endpoint de acceso libre desde el cual cualquier usuario anónimo, puede registrarse para obtener así el rol de USUARIO y un perfil en la aplicación. El Listado 6.1 muestra el endpoint habilitado, en el cual se realiza una implementación reactiva del sistema de registro con gestión de errores que serán enviados al frontend para notificar al usuario. Además, también ha un servicio que se lanza inicialmente desde la clase de configuración `TestDataBaseConfiguration` que hace la misma función que el de usuarios pero para crear al usuario administrador.

Listado 6.1: Fichero de endpoints de creación de usuarios `SignupController.java`

```

20  @RestController
21  @RequestMapping("api/v1/signup")
22  public class SignupController {
23
24      @Autowired
25      private SignupService signupService;
26
27      @PostMapping("")
28      public Mono<ResponseEntity<Map<String, String>>> registro(@RequestBody MyUserDTO entity) {
29
30          return signupService.registerUser(entity)
31              .map(user -> {
32
33                  Map<String, String> response = new HashMap<>();
34                  response.put("mensaje", "Usuario registrado exitosamente: " + user.toString());
35                  return ResponseEntity.ok(response);
36              })
37              .onErrorResume(e -> {
38
39                  Map<String, String> errorResponse = new HashMap<>();
40                  errorResponse.put("error", "Error al registrar usuario: " + e.getMessage());
41                  return Mono.just(ResponseEntity.badRequest().body(errorResponse));
42              });
43      }
44  }
45 }
```

Este endpoint llamará a un servicio que se encargara de crear el usuario en la base de datos. El Listado 6.2 muestra el servicio que se encarga de crear el usuario en la base de datos y enviar el correo al usuario. En la línea 32 se puede observar como se controla que no se pueda duplicar el usuario en el sistema, pese a que en base de datos también se encuentra incluida esta restricción. En la línea 37 se puede observar como se procede a cifrar la contraseña antes de almacenarla en la base de datos. Finalmente, en la línea 53 se observa el proceso de guardado del usuario con los datos provenientes del objeto **Data Transfer Object (DTO)** que han sido transformados a un objeto de tipo **USER** que es el que se almacena en la base de datos mediante el repositorio y el dominio generado.

Listado 6.2: Fichero de servicio de creación de usuarios `SignupService.java`

```

29  @Transactional
30  public Mono<Object> registerUser(MyUserDTO request) {
31
32      return userRepository.findByEmail(request.getEmail())
33          .flatMap(existingUser -> Mono.error(new Exception("El email de usuario
34          "+existingUser.getEmail()+" ya está en uso.")))
35          .switchIfEmpty(Mono.defer(() -> {
36
37              MyUser newUser = new MyUser();
```

Q: ¿Por qué se usa switchIfEmpty?

y ate la 36?

```

38     newUser.setPassword(passwordEncoder.encode(request.getPassword()));
39     newUser.setNombre(request.getNombre());
40     newUser.setApellidos(request.getApellidos());
41     newUser.setEmail(request.getEmail());
42     newUser.setTelefono(request.getTelefono());
43     newUser.setDni(request.getDni());
44     newUser.setFechaNacimiento(request.getFechaNacimiento());
45     newUser.setDireccion(request.getDireccion());
46     newUser.setEstado(true);
47     newUser.setFechaRegistro(java.time.LocalDate.now());
48     newUser.setFechaModificacion(java.time.LocalDate.now());

49
50
51     return userRepository.save(newUser)
52         .flatMap(savedUser ->
53
54             userRepository.findFirstByName("ROLE_USER")
55                 .flatMap(role -> {
56                     UserRole roleAssign = new UserRole();
57                     roleAssign.setUserId(savedUser.getId());
58                     roleAssign.setRoleId(role.getId());
59
60                     return roleUserRepository.save(roleAssign)
61                         .then(Mono.just(savedUser));
62                 })
63             );
64         );

```

↓ Form con S3

El sistema de autenticación y autorización se basa en el uso de [JSON Web Token \(JWT\)](#) para la gestión de los tokens de acceso. El [Listado 6.3](#) muestra el fichero de configuración del sistema de seguridad, donde se observa como se han definido las rutas que no requieren autenticación y las rutas que requieren autenticación. En las líneas 80 a 91 se observa como están los diferentes endpoints con restricciones especiales en la aplicación. Los primeros que aparecen son los endpoints a los cuales todo usuario incluso los no registrados podrán acceder. En la línea 90 se observa el endpoint para añadir reseña que solo lo puede realizar un usuario autenticado que tenga el rol de [CLIENTE](#) y en la 91 se especifica que todos los demás endpoints a los que se llame deben hacerse con un usuario autenticado independientemente de su rol. Por otro lado, destacar que se ha incluido la política de [Cross-Origin Resource Sharing \(CORS\)](#) para permitir solo los métodos utilizados y una política especial en la línea 103 para exponer los tokens, roles y distintos campos que son necesarios para que desde el [Frontend](#) se puedan consultar en el momento que se necesite. Finalmente, se observa que se ha decidido realizar una reimplementación de las clases de Spring Security [55] de forma reactiva para poder comprobar la autoría de los tokens con los usuarios almacenados en la base de datos. En el Apéndice A.1, en el Listado A.1, se puede observar la implementación personalizada que se ha realizado validando el token [JWT](#) con un objeto [USER](#) el cual internamente valida mediante un repositorio si existe en la base de datos. Finalmente, en el [Apéndice A.1](#), en el [Listado A.2](#), se realiza el filtro de autorización en caso de no ser un inicio de sesión y se extraen los roles que van implícitos en las propias peticiones y se obtiene el usuario directamente del [JWT](#) para así identificar unívocamente cada petición con el usuario y acceder solamente a la información de este.

Listado 6.3: Fichero de configuración de seguridad `WebSecurityConfig.java`

```

36     public WebSecurityConfig(CustomUserDetailsService userDetailsService,
37                             AuthenticationManager authenticationManager,
38                             SecurityContextRepository securityContextRepository,
39                             JwtService jwtService) {
40         this.userDetailsService = userDetailsService;
41         this.authenticationManager = authenticationManager;
42         this.securityContextRepository = securityContextRepository;
43         this.jwtService = jwtService;
44     }

```

```

45
46     @Bean
47     public PasswordEncoder passwordEncoder() {
48         return new BCryptPasswordEncoder();
49     }
50
51     @Bean
52     public ReactiveAuthenticationManager reactiveAuthenticationManager() {
53         var authManager = new UserDetailsServiceReactiveAuthenticationManager(userDetailsService);
54         authManager.setPasswordEncoder(passwordEncoder());
55         return authManager;
56     }
57
58     @Bean
59     public SecurityWebFilterChain securityFilterChain(ServerHttpSecurity http) {
60         var authenticationFilter = new CustomAuthenticationFilter(reactiveAuthenticationManager(),
61             jwtService);
62         var authorizationFilter = new CustomAuthorizationFilter(jwtService);
63
64         return http
65             .csrf(csrf -> csrf.disable()) // [ADVERTENCIA] Habilitar si usas cookies/session en lugar de
66             .JWT
67             .authenticationManager(authenticationManager)
68             .securityContextRepository(securityContextRepository)
69
70             .headers(headers -> headers
71                 .contentSecurityPolicy(csp -> csp
72                     .policyDirectives("default-src 'self'; script-src 'self'; style-src 'self'; font-src
73                         'self'; img-src 'self' data:"))
74                 .contentTypeOptions(config -> {}) // X-Content-Type-Options: nosniff
75             )
76
77             .authorizeExchange(exchanges -> exchanges
78                 .pathMatchers("/api/v1/login/**").permitAll()
79                 .pathMatchers("/api/v1/signup").permitAll()
80                 .pathMatchers("/api/v1/video", "/api/v1/contact", "/api/v1/media", "/api/v1/entorno",
81                     "/api/v1/fiestas", "/api/v1/gastronomia", "/api/v1/reviews").permitAll()
82                 .pathMatchers("/v3/api-docs**", "/swagger-ui/**").permitAll()
83                 .pathMatchers("/api/v1/addReview").hasAuthority("ROLE_CLIENTE")
84                 .anyExchange().authenticated()
85             )
86
87             .addFilterAt(authenticationFilter, SecurityWebFiltersOrder.AUTHENTICATION)
88             .addFilterBefore(authorizationFilter, SecurityWebFiltersOrder.AUTHORIZATION)
89             .cors(cors -> cors.configurationSource(corsConfigurationSource()))
90             .build();
91     }
92
93     @Bean
94     public CorsConfigurationSource corsConfigurationSource() {
95         CorsConfiguration config = new CorsConfiguration();
96         config.setAllowedOrigins(Collections.singletonList("*"));
97         config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
98         config.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type", "X-Requested-With",
99             "*"));
100        config.setExposedHeaders(Arrays.asList("Authorization", "Content-Type",
101            "Access-Control-Allow-Origin", "access_token", "refresh_token", "username", "roles"));
102
103        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
104        source.registerCorsConfiguration("/**", config);
105        return source;
106    }
107}

```

*índice reservas
línea en blanco*

6.1.2. Gestión de reservas

En este apartado se describe el subsistema de gestión de reservas. Este subsistema permite gestionar las reservas de los usuarios y la disponibilidad de la casa rural. El

desarrollo de este subsistema se divide en endpoints dedicados a los usuarios con rol USER y, por otro lado, endpoints dedicados a los usuarios con rol ADMIN, que permiten gestionar las peticiones de reserva generadas por los usuarios registrados.

Los usuarios autenticados con el rol USER podrán realizar nuevas reservas a través del endpoint /reserva (línea 33 del Listado 6.4), que recoge los datos de una reserva y los procesa de forma reactiva mediante el servicio correspondiente. También podrán consultar las reservas que han realizado previamente mediante el endpoint /reservasUser, utilizando su dirección de correo electrónico como identificador (línea 77 del mismo listado), la cuál ha sido extradida por el Frontend con la llamada a un endpoint de información del usuario (/me) en función del token de autenticación.

En cuanto a los usuarios con el rol ADMIN, estos disponen de varios endpoints protegidos mediante la anotación @PreAuthorize("hasRole('ADMIN')"), lo cual restringe su uso exclusivamente a administradores. Uno de ellos es /reservas, que permite consultar todas las reservas realizadas por los usuarios (línea 44), y otro es /reservasProximas, que proporciona un listado de las reservas activas próximas a la fecha actual (línea 55). Además, el endpoint /{id}/estado (línea 65) permite modificar el estado de una reserva concreta, proporcionando tanto el nuevo estado como el motivo de la modificación.

El Listado 6.4 muestra el fichero completo ReservaController.java, el cual agrupa todos los endpoints descritos. En él se observa cómo se hace uso de la programación reactiva, y cómo se gestionan los errores de manera asíncrona, devolviendo respuestas adecuadas con los códigos de estado correspondientes.

Listado 6.4: Fichero controlador de reservas ReservaController.java

```

26  @RestController
27  @RequestMapping("/api/v1")
28  public class ReservaController {
29
30      @Autowired
31      private ReservaService reservaService;
32
33      @PostMapping("/reserva")
34      public Mono<ResponseEntity<Integer>> confirmarReserva(@RequestBody ReservaDTO reservaDTO) {
35          return reservaService.confirmarReserva(reservaDTO)
36              .map(reserva -> new ResponseEntity<>(reserva, HttpStatus.CREATED))
37              .onErrorResume(e -> {
38                  if(e.getMessage().contains("ERROR_USER")) {
39                      return Mono.just(new ResponseEntity<>(HttpStatus.NOT_FOUND));
40                  } else if(e.getMessage().contains("ERROR_RESERVA")) {
41                      return Mono.just(new ResponseEntity<>(HttpStatus.CONFLICT));
42                  } else {
43                      System.err.println("Error confirmacion de reserva: " + e.getMessage());
44                      return Mono.just(new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR));
45                  }
46
47              });
48      }
49
50      @GetMapping("/reservas")
51      @PreAuthorize("hasRole('ROLE_ADMIN')")
52      public Flux<Reserva> getReservas() {
53          return reservaService.getAllDataReservas()
54              .onErrorResume(e -> {
55
56                  System.err.println("Error fetching reservas: " + e.getMessage());
57                  return Flux.empty();
58              });
59      }
60
61      @GetMapping("/reservasProximas")
62
63      public Flux<Tuple2<LocalDate, LocalDate>> getReservasFromDate() {
```



```

64     return reservaService.getAllDataReservasFromToday()
65         .onErrorResume(e -> {
66
67             System.err.println("Error fetching reservas: " + e.getMessage());
68             return Flux.empty();
69         });
70     }
71     @PutMapping("/{id}/estado")
72     @PreAuthorize("hasRole('ROLE_ADMIN')")
73     public Mono<ResponseEntity<Reserva>> actualizarEstado(@PathVariable Integer id, @RequestParam String
74     estado, @RequestBody String motivo) {
75         return reservaService.setEstado(id, estado, motivo)
76             .map(result -> ResponseEntity.ok().body(result))
77             .switchIfEmpty(Mono.just(ResponseEntity.notFound().build()))
78             .onErrorResume(e -> {
79                 return Mono.just(ResponseEntity.internalServerError().build());
80             });
81     }
82
83     @GetMapping("/reservasUser")
84     public Flux<Reserva> getReservasUsuario(@RequestParam String email) {
85         return reservaService.getReservasByUser(email);
86     }
87
88
89
90
91 }

```

Sólo entre paréntesis cuando no es la primera
vez que lo citas

Los usuarios autenticados con el rol USER podrán realizar nuevas reservas a través del endpoint /reserva (línea 17 del Listado 6.5), que recoge los datos de una reserva y los procesa de forma reactiva mediante el servicio correspondiente. También podrán consultar las reservas que han realizado previamente mediante el endpoint /reservasUser, utilizando su dirección de correo electrónico como identificador (línea 52 del mismo listado).

En cuanto a los usuarios con el rol ADMIN, estos disponen de varios endpoints protegidos mediante la anotación @PreAuthorize("hasRole('ADMIN')"), lo cual restringe su uso exclusivamente a administradores. Uno de ellos es /reservas, que permite consultar todas las reservas realizadas por los usuarios (línea 46), y otro es /reservasProximas, que proporciona un listado de las reservas activas próximas a la fecha actual (línea 55). Además, el endpoint /{id}/estado (línea 65) permite modificar el estado de una reserva concreta, proporcionando tanto el nuevo estado como el motivo de la modificación.

El Listado 6.5 muestra el fichero completo ReservaService.java, el cual implementa la lógica de negocio relacionada con la gestión de reservas. Este servicio es invocado por el controlador ReservaController (Listado 6.4) para realizar las operaciones sobre las reservas, incluyendo su creación, modificación, cancelación, confirmación de pago y actualización de roles de usuario.

¡Tener en

Listado 6.5: Fichero de servicio de reservas ReservaService.java

```

25     private final ReservaRepository reservaRepository;
26     private final EmailService emailService;
27     private final UserRepository userRepository;
28     private final RoleUserRepository roleUserRepository;
29     private final RoleRepository roleRepository;
30     public ReservaService(ReservaRepository reservaRepository, EmailService emailService, UserRepository
31     userRepository, RoleUserRepository roleUserRepository, RoleRepository roleRepository) {
32         this.reservaRepository = reservaRepository;
33         this.emailService = emailService;
34         this.userRepository = userRepository;
35         this.roleUserRepository = roleUserRepository;
36         this.roleRepository = roleRepository;

```

```
37
38     public Mono<Integer> confirmarReserva(ReservaDTO reservaDTO) {
39         Reserva nuevaReserva = new Reserva();
40
41         nuevaReserva.setFechaInicio(reservaDTO.getStartDate());
42         nuevaReserva.setFechaFin(reservaDTO.getEndDate());
43
44         nuevaReserva.setNumPersonas(reservaDTO.getNumPersonas());
45         nuevaReserva.setPrecioTotal(reservaDTO.getPrecio());
46         nuevaReserva.setEstado(Reserva.EstadoReserva.CONFIRMADA);
47         nuevaReserva.setEmail(reservaDTO.getEmail());
48         nuevaReserva.setNombre(reservaDTO.getNombre());
49         nuevaReserva.setDni(reservaDTO.getDni());
50         nuevaReserva.setTelefono(reservaDTO.getTelefono());
51
52         return userRepository.findByEmail(reservaDTO.getEmailUsuario())
53             .switchIfEmpty(Mono.error(new RuntimeException("ERROR_USER")))
54             .flatMap(user -> {
55                 nuevaReserva.setUsuarioId(user.getId());
56                 return reservaRepository.existeReservaEnFechas(
57                     nuevaReserva.getFechaInicio(),
58                     nuevaReserva.getFechaFin()
59
60                     ).flatMap(existe -> {
61                         if (!existe) {
62                             return reservaRepository.save(nuevaReserva)
63                                 .doOnSuccess(savedReserva -> {
64                                     emailService.sendEmail(savedReserva)
65                                         .doOnSuccess(_ -> System.out.println("Email enviado"))
66                                         .doOnError(error -> new RuntimeException("ERROR_EMAIL"))
67                                         .subscribe();
68
69                                     .map(Reserva::getId);
70
71                         } else {
72                             return Mono.error(new RuntimeException("EXISTE_RESERVA"));
73                         }
74                     });
75                 });
76             });
77
78     public Mono<Reserva> confirmarPagoReserva(Integer idReserva, String motivo) {
79
80         return reservaRepository.findById(idReserva)
81             .doOnSuccess(reservaPagada -> {
82
83                 emailService.sendEmailPay(reservaPagada, motivo)
84                     .doOnSuccess(_ -> System.out.println("Email enviado"))
85                     .doOnError(error -> System.out.println("Error al enviar email: " +
86                         error.getMessage()))
87                     .subscribe();
88
89             });
90
91     public Mono<Reserva> cancelarReserva(Integer idReserva, String motivo) {
92
93         return reservaRepository.findById(idReserva)
94             .doOnSuccess(reservaPagada -> {
95
96
97                 emailService.sendEmailCancel(reservaPagada, motivo)
98                     .doOnSuccess(_ -> System.out.println("Email enviado"))
99                     .doOnError(error -> System.out.println("Error al enviar email: " +
100                         error.getMessage()))
101                     .subscribe();
102
103             };
104
105     }
106
107
108     public Flux<Integer> getReservas() {
109         return reservaRepository.findAllId();
```

```
110     }
111
112     public Flux<Reserva> getAllDataReservas() {
113         return reservaRepository.findAllByOrderByFechaInicioDesc();
114     }
115
116     public Flux<Tuple2<LocalDate, LocalDate>> getAllDataReservasFromToday() {
117         return reservaRepository.findByFechaInicioGreaterThanOrEqualTo(LocalDate.now())
118             .map(reserva -> Tuples.of(reserva.getFechaInicio(), reserva.getFechaFin()));
119     }
120
121
122     public Flux<Reserva> getReservasByUser(String email) {
123         return reservaRepository.findByEmail(email);
124     }
125
126     public Mono<Reserva> setEstado(Integer reservaid, String estado, String motivo) {
127         return reservaRepository.findById(reservaid)
128             .flatMap(reserva -> {
129                 reserva.setEstado(Reserva.EstadoReserva.valueOf(estado));
130                 if (estado.equals("PAGADA")) {
131                     return confirmarPagoReserva(reservaid, motivo)
132                         .then(reservaRepository.save(reserva));
133                 } else if (estado.equals("CANCELADA")) {
134                     return cancelarReserva(reservaid, motivo)
135                         .then(reservaRepository.save(reserva));
136                 } else {
137                     return reservaRepository.save(reserva);
138                 }
139             });
140     }
141
142     @Scheduled(cron = "0 0 2 * * ?")
143     public void actualizarRolesUsuarios() {
144         LocalDate hoy = LocalDate.now();
145
146         reservaRepository.findByEstadoAndFechaInicioLessThanEqual(Reserva.EstadoReserva.PAGADA, hoy)
147             .flatMap(reserva ->
148                 roleUserRepository.findById(reserva.getUsuarioId()) // Obtenemos los roles del usuario
149                     .collectList() // Convertimos los roles a una lista
150                     .flatMap(roles ->
151                         roleRepository.findByName("ROLE_CLIENTE") // Buscamos el rol CLIENTE de manera reactiva
152                             .flatMap(roleCliente -> {
153                                 // Verificamos si la lista de roles ya contiene ROLE_CLIENTE
154                                 boolean tieneClienteRole = roles.stream()
155                                     .anyMatch(role -> role.getRoleId().equals(roleCliente.getId()));
156                                 System.out.println(tieneClienteRole);
157                                 if (tieneClienteRole) {
158                                     return Mono.empty(); // No hacer nada si ya tiene el rol CLIENTE
159                                 } else {
160                                     // Si no tiene el rol, lo asignamos
161                                     UserRole roleAssign = new UserRole();
162                                     roleAssign.setUserId(reserva.getUsuarioId());
163                                     roleAssign.setRoleId(roleCliente.getId());
164                                     System.out.println("Asignando rol CLIENTE al usuario " +
165                                         roleAssign.getUserId() + " con id " + roleAssign.getRoleId());
166                                     // Actualizamos el estado de la reserva a FINALIZADA
167                                     reserva.setEstado(Reserva.EstadoReserva.FINALIZADA);
168                                     return reservaRepository.save(reserva)
169                                         .then(roleUserRepository.save(roleAssign));
170                                 }
171                             })
172             )
173             .subscribe(
174                 _ -> System.out.println("Roles actualizados correctamente."),
175                 error -> System.err.println("Error al actualizar roles: " + error.getMessage())
176             );
177
178
179
180
181     }
182
183 }
```

A lo largo de la implementación se observa cómo se emplea programación reactiva gracias a los tipos `Mono<T>` y `Flux<T>`, lo que permite manejar flujos de datos de forma asíncrona y no bloqueante.

- El método `confirmarReserva` crea una nueva reserva a partir de los datos proporcionados en un `ReservaDTO`, la asocia al usuario correspondiente a partir de su email, la guarda en la base de datos y finalmente lanza un correo de confirmación utilizando `EmailService` que se muestra en el Apéndice A.2, Listado A.3.
- El método `confirmarPagoReserva` se utiliza para marcar una reserva como pagada y enviar un correo de confirmación del pago.
- De manera similar, `cancelarReserva` notifica por correo electrónico que una reserva ha sido cancelada.
- El método `setEstado` permite cambiar el estado de una reserva a través de su identificador, y según el nuevo estado, desencadena acciones adicionales como el envío de correos o la modificación de datos en la base.
- `getReservas` devuelve un flujo de identificadores de todas las reservas almacenadas, mientras que `getAllDataReservas` proporciona todos los datos completos de cada una de ellas.
- `getAllDataReservasFromToday` filtra aquellas reservas cuya fecha de inicio es igual o posterior al día actual, devolviendo únicamente sus intervalos de fechas, lo que puede ser útil para mostrar disponibilidad en un calendario.
- `getReservasByUser` permite obtener las reservas asociadas a un usuario concreto mediante su correo electrónico.
- Finalmente, el método programado `actualizarRolesUsuarios`, que se ejecuta automáticamente cada día a las 2:00h gracias a la anotación `@Scheduled`, actualiza el rol de los usuarios que hayan realizado reservas ya finalizadas, asignándoles el rol de cliente (`CLIENTE`) si todavía no lo tienen. Esta lógica garantiza que los usuarios activos que han completado reservas reciban un rol adecuado para su perfil y puedan dejar una reseña.

6.1.3. Gestión de reseñas

El subsistema de reseñas permite a los usuarios compartir valoraciones sobre su experiencia en la casa rural. Está compuesto por dos endpoints principales: uno para la consulta pública de reseñas y otro restringido para la creación de nuevas valoraciones.

El primer endpoint, `/reviews` (línea 76 del Listado 6.6), permite obtener todas las reseñas almacenadas en la base de datos, ordenadas de forma descendente según la fecha de creación. Este endpoint está disponible para cualquier usuario, sin necesidad de autenticación previa, lo que facilita la visibilidad de las opiniones existentes sobre el alojamiento.

El segundo endpoint, `/addReview` (línea 80), permite a los usuarios autenticados con el rol `CLIENTE` enviar una nueva reseña mediante un objeto `ReviewDTO`. Para ello, se obtiene el contexto de seguridad reactivo a fin de identificar al usuario autenticado, y se valida su existencia en la base de datos. A continuación, se construye un nuevo objeto `Review` con

los datos proporcionados, incluyendo la puntuación general y las valoraciones específicas sobre limpieza, ubicación y servicios. Como paso final, se elimina la relación del usuario con el rol ROLE_CLIENTE (línea 51), indicando que ya ha realizado su valoración, y se guarda la reseña en la base de datos. En caso de producirse un error durante este proceso, se responde con un código de estado Bad Request.

El Listado 6.6 muestra parte del fichero `ApplicationController.java` que contiene los endpoints para la gestión de reseñas, mientras que el Listado 6.7 recoge el contenido del servicio `ReviewService`, encargado de la lógica de negocio para almacenar y recuperar reseñas.

Listado 6.6: Endpoints para la gestión de reseñas `ApplicationController.java`

```

75  @GetMapping(value="reviews", produces="application/json")
76  public Flux<Review> getReviews() {
77      return reviewService.getReviews();
78  }
79
80  @PostMapping(value="addReview", produces="application/json")
81  public Mono<ResponseEntity<Review>> addReview(@RequestBody ReviewDTO reviewDTO) {
82      return reviewService.saveReview(reviewDTO)
83          .map(savedReview -> ResponseEntity.ok(savedReview))
84          .onErrorResume(e -> Mono.just(ResponseEntity.badRequest().build()));
85  }

```

Listado 6.7: Servicio de gestión de reseñas `ReviewService.java`

```

18  @Service
19  public class ReviewService {
20      @Autowired ReviewRepository reviewRepository;
21      @Autowired RoleUserRepository roleUserRepository;
22      @Autowired RoleRepository roleRepository;
23      @Autowired UserRepository userRepository;
24      public Mono<Review> saveReview(ReviewDTO reviewDTO) {
25          System.out.println("ReviewDTO: " + reviewDTO.getRate());
26
27          return ReactiveSecurityContextHolder.getContext()
28              .flatMap(ctx -> {
29                  Authentication authentication = ctx.getAuthentication();
30                  String userId = (String) authentication.getPrincipal();
31                  System.out.println("Principal: " + userId);
32                  return Mono.just(userId);
33              })
34              .flatMap(userId ->
35                  userRepository.findByEmail(userId)
36                      .switchIfEmpty(Mono.error(new RuntimeException("User not found")))
37              )
38              .flatMap(user -> {
39                  Review review = new Review();
40                  review.setReview(reviewDTO.getReview());
41                  review.setRate(reviewDTO.getRate());
42                  review.setRateLimpieza(reviewDTO.getRateLimpieza());
43                  review.setRateServicios(reviewDTO.getRateServicios());
44                  review.setRateUbicacion(reviewDTO.getRateUbicacion());
45                  review.setUserId(user.getId());
46                  review.setCreationDate(LocalDateTime.now());
47                  return roleRepository.findByName("ROLE_CLIENTE")
48                      .flatMap(role -> {
49                          return roleUserRepository.deleteByUserIdAndRoleId(user.getId(), role.getId())
50                              .then(reviewRepository.save(review));
51                      });
52
53      });
54  }
55
56  public Flux<Review> getReviews() {
57
58

```

→ si quitas espacios igual estás en página

```

59     return reviewRepository.findAllByOrderByCreationDateDesc();
60 }
61 }
```

6.1.4. Formulario de contacto

La aplicación incorpora un mecanismo de contacto que permite a cualquier usuario enviar un mensaje al propietario de la casa rural. Este mecanismo está disponible a través del endpoint `/contact`, que acepta solicitudes POST con un objeto `Contact` en formato JSON (línea 93 del Listado 6.8).

El controlador delega la lógica de envío al servicio `ContactService`, el cual se encarga de construir y enviar un mensaje de correo electrónico a la dirección configurada en el sistema (a través de la propiedad `spring.email.email`). El contenido del mensaje incluye el nombre del remitente, su dirección de correo electrónico (usada como `reply-to`) y el mensaje de consulta proporcionado por el usuario.

Este servicio permite una comunicación directa entre los usuarios y el responsable del alojamiento, facilitando la resolución de dudas o la solicitud de información adicional sin necesidad de registro o autenticación previa.

A continuación, se muestran en el Listado 6.8 las líneas relevantes del controlador `ApplicationController.java`, y en el Listado 6.9 el fragmento correspondiente del servicio `ContactService.java`.

Listado 6.8: Controlador de contacto `ApplicationController.java`

```

93 @PostMapping("/contact")
94 public Mono<ResponseEntity<Contact>> sendEmail(@RequestBody Contact c) {
95     return contactService.sendEmail(c)
96         .map(ResponseEntity::ok)
97         .onErrorResume(IllegalArgumentException.class,
98             e -> Mono.just(ResponseEntity.badRequest().body(null)));
99 }
100 }
```

→ no puedes pasar a antiguación creando ya lo has q todo
previamente

Listado 6.9: Servicio de contacto `ContactService.java`

```

12 import reactor.core.publisher.Mono;
13
14 @Service
15 public class ContactService {
16
17
18     private final JavaMailSender mailSender;
19
20     @Value("${spring.email.email}")           // tu buzón destino
21     private String to;
22
23     public ContactService(JavaMailSender mailSender) {
24         this.mailSender = mailSender;
25     }
26
27     public Mono<Contact> sendEmail(Contact contact) {
```

↑
No vueltes todo si
no muestra más código.

6.1.5. Persistencia y acceso a datos sistema principal

Por último, se presenta la base de datos del sistema principal. Esta base de datos está compuesta por varias tablas que almacenan la información necesaria para el funcionamiento de la aplicación. A continuación, se describen las tablas más relevantes y los repositorios de Spring Data que permiten interactuar con ellas. Para dar soporte a las operaciones del sistema, se han definido diversas entidades persistentes que se almacenan en una base de datos relacional PostgreSQL. Las tablas se crean a través del script `schema.sql`, el cual se ejecuta al iniciar la aplicación si las tablas no existen. A continuación se describe la estructura de cada una de las tablas principales y su relación con las clases del dominio y los repositorios correspondientes.

- Entidad Reserva: La entidad `Reserva` representa una solicitud de reserva por parte de un usuario, e incluye información sobre las fechas, el número de personas, el estado de la reserva y los datos de contacto. Su representación en base de datos se define mediante la tabla `reservas`, incluida en el Listado 6.10. Esta tabla establece una relación de clave foránea con la tabla `users`, de forma que cada reserva está asociada a un usuario registrado.

El acceso a esta entidad se realiza mediante el repositorio `ReservaRepository`, que extiende la interfaz `R2dbcRepository` y proporciona métodos para consultar reservas por identificador, estado, fecha de inicio o dirección de correo electrónico (véase Apéndice A.3, Listado A.4). La clase del dominio asociada es `Reserva`, que contiene los campos mapeados mediante anotaciones de Spring Data R2DBC (véase Apéndice A.3, Listado A.5).

Listado 6.10: Definición de la tabla `reservas` en `schema.sql`

```

42 | CREATE TABLE IF NOT EXISTS reservas (
43 |   id SERIAL PRIMARY KEY,
44 |   nombre VARCHAR(255) NOT NULL CHECK (char_length(nombre) >= 1),
45 |   fecha_inicio DATE NOT NULL,
46 |   fecha_fin DATE NOT NULL,
47 |   num_personas INT NOT NULL CHECK (num_personas > 0),
48 |   email VARCHAR(255) NOT NULL,
49 |   telefono CHAR(9) NOT NULL CHECK (telefono ~ '^[0-9]{9}$'),
50 |   dni CHAR(9) NOT NULL CHECK (dni ~ '^\d{8}[A-Za-z]$'),
51 |   precio_total DECIMAL(10, 2) NOT NULL CHECK (precio_total >= 0),
52 |   estado VARCHAR(50) NOT NULL CHECK (estado IN ('CONFIRMADA', 'PAGADA',
53 |   'CANCELADA', 'FINALIZADA')),
54 |   usuario_id INT NOT NULL,
55 |   FOREIGN KEY (usuario_id) REFERENCES users(id) ON DELETE CASCADE
      );

```

- Entidad Review: La tabla `reviews` permite almacenar valoraciones realizadas por los usuarios sobre su experiencia en la casa rural. Cada reseña incluye un comentario textual, puntuaciones detalladas (servicios, limpieza y ubicación) y una relación directa con el usuario que la creó. Esta tabla también define una clave foránea a la tabla `users`, lo que asegura la integridad referencial (Listado 6.11).

El repositorio correspondiente, así como su dominio `Review`, se incluyen en el Apéndice A.3, Listado A.7 y Listado A.6 para consulta detallada.

Listado 6.11: Definición de la tabla `reviews` en `schema.sql`

```

30 | CREATE TABLE if not exists reviews (
31 |   id SERIAL PRIMARY KEY,
32 |   review TEXT NOT NULL,

```

```

33     rate INTEGER NOT NULL CHECK (rate >= 1 AND rate <= 5),
34     rate_servicios INTEGER NOT NULL CHECK (rate >= 1 AND rate <= 5),
35     rate_limpieza INTEGER NOT NULL CHECK (rate >= 1 AND rate <= 5),
36     rate_ubicacion INTEGER NOT NULL CHECK (rate >= 1 AND rate <= 5),
37     user_id BIGINT NOT NULL,
38     creation_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
39     FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
40 );

```

- Entidad User y control de roles:

Los usuarios del sistema se representan mediante la tabla `users`, la cual contiene información personal, de contacto y de estado de actividad. La relación de roles se gestiona mediante las tablas `roles` y `user_role`, que permiten definir una estructura de autorización flexible mediante una relación de muchos a muchos (Listado 6.12). Además, ya que Spring R2DBC no admite campos en los que se guarden tipos de datos complejos, esta es la única forma de implementar esta relación de roles y usuarios.

Who's

Esta estructura es fundamental para la gestión de accesos a diferentes funcionalidades del sistema, como se ha visto en las secciones anteriores. Los repositorios correspondientes a estas entidades (`UserRepository`, `RoleRepository`, etc.) también se incluyen en el Apéndice A.3.

Listado 6.12: Definición de las tablas `users`, `roles` y `user_role` en `schema.sql`

```

1 CREATE TABLE IF NOT EXISTS roles (
2     id SERIAL PRIMARY KEY,
3     name VARCHAR(255) NOT NULL UNIQUE
4 );
5
6 CREATE TABLE IF NOT EXISTS users (
7     id SERIAL PRIMARY KEY,
8     nombre VARCHAR(255) NOT NULL CHECK (char_length(nombre) >= 1),
9     password VARCHAR(255) NOT NULL CHECK (char_length(password) >= 6),
10    apellidos VARCHAR(255) NOT NULL CHECK (char_length(apellidos) >= 1),
11    email VARCHAR(255) NOT NULL,
12    telefono CHAR(9) NOT NULL CHECK (telefono ~ '^[0-9]{9}$'),
13    dni CHAR(9) NOT NULL CHECK (dni ~ '^\d{8}[A-Za-z]$'),
14    fecha_nacimiento DATE NOT NULL,
15    direccion VARCHAR(255) NOT NULL CHECK (char_length(direccion) >= 10),
16    estado BOOLEAN NOT NULL,
17    fecha_registro DATE NOT NULL,
18    fecha_modificacion DATE NOT NULL,
19    fecha_baja DATE
20 );
21
22
23 CREATE TABLE IF NOT EXISTS user_role (
24     user_id INT,
25     role_id INT,
26     FOREIGN KEY (user_id) REFERENCES users(id),
27     FOREIGN KEY (role_id) REFERENCES roles(id)
28 );

```

6.2. Subsistema de recursos web

La aplicación incluye un mecanismo automatizado de extracción de recursos turísticos que permite mantener actualizada la información sobre sitios de interés cercanos a la casa rural. Este sistema está accesible mediante el controlador `ExtraccionController`, disponible en el endpoint `/extraccion`, y se apoya en el servicio `RecursoService` para obtener y devolver los datos en formato reactivo (líneas 23 a 28 del Listado 6.13).

El controlador ofrece tres rutas distintas:

/extraccion para obtener todos los recursos turísticos conocidos,

/extraccionFiestas para obtener únicamente fiestas locales extraídas de sitios web oficiales,

/extraccion/test para forzar manualmente el scraping y almacenamiento de datos, útil para pruebas o actualizaciones no programadas.

La lógica principal reside en el servicio `RecursoService`, el cual ejecuta una tarea programada diariamente mediante una anotación `@Scheduled`. Este servicio llama al `ScraperService`, que se encarga de realizar la extracción asíncrona de información desde diversas URLs predefinidas, seleccionadas tras un estudio previo de fuentes confiables, utilizando expresiones regulares de búsqueda [Cascading Style Sheets \(CSS\)](#) o [XML Path Language \(XPath\)](#) y la librería JSoup [56]. La información extraída se procesa y guarda automáticamente en colecciones MongoDB[24] utilizando un repositorio reactivo.

Posteriormente, el propio `RecursoService` consulta estas colecciones para proporcionar datos actualizados al usuario final, distinguiendo entre fiestas locales y otros eventos o lugares de interés.

A continuación, se muestran los fragmentos relevantes de implementación. En el Listado 6.13 se presenta el controlador de extracción, en el Listado 6.14 la lógica del servicio de recursos, y en el Apéndice A.4, Listado A.14 el servicio encargado del scraping.

Listado 6.13: Fragmento del controlador de extracción de recursos `ExtraccionController.java`

```

15 public class ExtraccionController {
16
17     @Autowired
18     private RecursoService imgService;
19
20
21
22     @GetMapping("extraccion")
23     public Flux<Recurso> getResources( ) {
24         return imgService.extractAllResources();
25
26     }
27
28
29     @GetMapping("extraccionFiestas")
30     public Flux<Recurso> getFiestas( ) {
31         return imgService.extractFiestas();
32
33     }
34
35
36     @GetMapping("extraccion/test")
37     public void test() {
38         imgService.fetchandstoreData();
39     }
40 }
41

```

Listado 6.14: Servicio para la gestión de recursos `RecursoService.java`

```

13 @Service
14 public class RecursoService {
15     @Autowired
16     private ScraperService scraperService;
17     @Value("${url.tuejar}")
18     private String urlTuejar;

```

```

19     @Value("${url.chelva}")
20     private String urlChelva;
21     @Value("${url.aytuejar}")
22     private String urlAyuntamiento;
23     @Autowired
24     private RecursoRepository sRepository;
25     @Scheduled(cron = "0 0 0 * * ?")
26     public void fetchAndStoreData() {
27
28         scraperService.scrape(urlChelva + "monumentos", "monumentos", "chelva").subscribe(
29             null,
30             error -> System.err.println("Error en el scraper para Chelva Monumentos: " +
31                 error.getMessage()),
32             () -> System.out.println("Scraping de Chelva Monumentos completado")
33         );
34         scraperService.scrape(urlChelva + "aldeas", "aldeas", "chelva").subscribe(
35             null,
36             error -> System.err.println("Error en el scraper para Chelva Aldeas: " + error.getMessage()),
37             () -> System.out.println("Scraping de Chelva Aldeas completado")
38         );
39
40         scraperService.scrape(urlTuejar + "patrimonio", "patrimonio", "tuejar").subscribe(
41             null,
42             error -> System.err.println("Error en el scraper para Tuejar Patrimonio: " +
43                 error.getMessage()),
44             () -> System.out.println("Scraping de Tuejar Patrimonio completado")
45         );
46
47         scraperService.scrape(urlTuejar + "naturaleza", "naturaleza", "tuejar").subscribe(
48             null,
49             error -> System.err.println("Error en el scraper para Tuejar Naturaleza: " +
50                 error.getMessage()),
51             () -> System.out.println("Scraping de Tuejar Naturaleza completado")
52         );
53
54         scraperService.scrape(urlAyuntamiento + "agenda-festividades", "fiestas", "tuejar").subscribe(
55             null,
56             error -> System.err.println("Error en el scraper para Tuejar Fiestas: " +
57                 error.getMessage()),
58             () -> System.out.println("Scraping de Tuejar Fiestas completado:" + urlAyuntamiento +
59                 "agenda-festividades")
60         );
61
62
63     public Flux<Recurso> extractAllResources() {
64         return sRepository.findAll();
65     }
66
67     public Flux<Recurso> extractFiestas() {
68         return sRepository.findByCategoria("fiestas");
69     }
70
71 }
72 }
```

6.2.1. Persistencia y acceso a datos subsistema de recursos web

La persistencia de los recursos extraídos se realiza automáticamente mediante la integración con MongoDB a través de Spring Data Reactive. La entidad principal utilizada es `Recurso`, representada como un documento en la colección `recursos`, definida mediante la anotación `@Document` en la clase correspondiente (Listado 6.15).

Cada recurso incluye atributos como título, descripción, URL, imagen codificada en

base64, localización y categoría. Para garantizar la unicidad de los registros, el identificador del documento se genera concatenando el título y la localización, lo que evita duplicados si se extrae un mismo sitio desde diferentes fuentes.

Además, la clase auxiliar `ScraperResult` encapsula una lista de objetos `Recurso` obtenidos tras realizar el proceso de extracción desde fuentes externas. Esta estructura intermedia permite manejar de forma más flexible los resultados antes de su inserción en la base de datos (Listado 6.16).

El acceso a la base de datos se gestiona mediante el repositorio reactivo `RecursoRepository`, que extiende la interfaz `ReactiveMongoRepository`. Este repositorio expone métodos personalizados como `findByTitle` y `findByCategoria`, que permiten realizar búsquedas específicas dentro de la colección `recursos` sin necesidad de definir consultas manualmente (Listado 6.17).

Gracias al uso de estas herramientas, las colecciones se crean automáticamente al insertar el primer documento, eliminando la necesidad de una definición previa o scripts de inicialización.

A continuación, se presentan los fragmentos relevantes de las clases de modelo y repositorio.

Listado 6.15: Modelo de documento MongoDB para recursos turísticos `Recurso.java`

```
6  @Document(collection = "recursos")
7  public class Recurso {
8
9      @Id
10     private String id;
11
12     private String title;
13     private String categoria;
14     private String imageString;
15     private String url;
16     private String description;
17     private String localizacion;
18
19     public Recurso() {
20     }
21
22     public Recurso(String title, String imageString, String url, String description, String categoria,
23     String localizacion) {
24         this.title = title;
25         this.imageString = imageString;
26         this.url = url;
27         this.description = description;
28         this.localizacion = localizacion;
29         this.categoria = categoria;
30         this.id = title + "-" + localizacion;
31     }
32
33     public String getId() {
34         return id;
35     }
36
37     public String getTitle() {
38         return title;
39     }
40
41     public void setTitle(String title) {
42         this.title = title;
43     }
44
45     public String getImageString() {
46         return imageString;
47     }
48
49     public void setImageString(String imageString) {
```

```

49     this.imageString = imageString;
50 }
51
52     public String getUrl() {
53         return url;
54     }
55
56     public void setUrl(String url) {
57         this.url = url;
58     }
59
60     public String getDescription() {
61         return description;
62     }
63
64     public void setDescription(String description) {
65         this.description = description;
66     }
67
68     public String getCategoría() {
69         return categoría;
70     }
71
72     public void setCategoría(String categoría) {
73         this.categoría = categoría;
74     }
75
76     public String getLocalización() {
77         return localización;
78     }

```

Listado 6.16: Modelo auxiliar de resultados de scraping `ScraperResult.java`

```

6     public class ScraperResult {
7         private List<Recurso> links;
8
9         public ScraperResult() {
10             this.links = new ArrayList<>();
11         }
12         public ScraperResult(List<Recurso> links) {
13             this.links = links;
14         }
15
16         public List<Recurso> getLinks() {
17             return links;
18         }
19         public void setLinks(List<Recurso> links) {
20             this.links = links;
21         }
22
23         public void add(Recurso recurso) {
24             links.add(recurso);
25         }
26     }

```

Listado 6.17: Repositorio para la persistencia en MongoDB `RecursoRepository.java`

```

5 import es.uv.hemal.extraccion.extraccion.models.Recurso;
6 import reactor.core.publisher.Flux;
7
8 public interface RecursoRepository extends ReactiveMongoRepository<Recurso, String> {
9     Flux<Recurso> findByTitle(String title);
10    Flux<Recurso> findByCategoría(String categoría);
11 }

```

6.3. Subsistema de publicaciones

Este subsistema está diseñado para obtener publicaciones desde la API de Instagram, procesarlas y almacenarlas en una base de datos MongoDB. Para ello se utilizan controladores y servicios desarrollados en Spring Boot utilizando Spring WebFlux [18] de forma reactiva.

Controlador de publicaciones

El controlador principal expone los endpoints para la consulta y extracción de publicaciones, así como la actualización de información asociada. El código se muestra en el [listado 6.18](#).

Listado 6.18: Controlador principal para las publicaciones `MediaController.java`

```

14  @RestController
15  @RequestMapping("api/v1")
16  public class MediaController {
17
18      @Autowired
19      private MediaService mediaService;
20
21      @Autowired
22      private InstagramTokenService instagramTokenRefresher;
23
24      @GetMapping("media")
25      public Flux<Media> getMediaByHashtag(@RequestParam(required = false) String hashtag) {
26          if (hashtag != null && !hashtag.isEmpty()) {
27
28              return mediaService.extractMediaByHashtag(hashtag);
29          } else {
30
31              return mediaService.extractAllMedia();
32          }
33      }
34
35
36      @GetMapping("media/test")
37      public void test() {
38          mediaService.fetchAndStorePosts();
39      }
40
41      @GetMapping("mediaToken/test")
42      public void testToken() {
43          instagramTokenRefresher.refreshToken();
44      }
45 }
```

Entre sus funcionalidades destacadas se encuentran:

- `/media`: devuelve todas las publicaciones almacenadas.
- `/media/hashtag/{hashtag}`: permite filtrar publicaciones por hashtag.
- `/media/test`: ejecuta manualmente la extracción desde Instagram.

Servicio de extracción de publicaciones

El servicio contiene la lógica de negocio encargada de conectarse con la API de Instagram, extraer las publicaciones, procesar su contenido y guardarlos en la base de datos. Su implementación se detalla en el [listado 6.19](#).

Listado 6.19: Servicio de extracción y almacenamiento de publicaciones
MediaService.java

```

23
24 @Service
25 public class MediaService {
26
27     @Autowired
28     private MediaRepository mediaRepository;
29     @Autowired
30     private InstagramTokenService instagramTokenRefresher;
31
32     private String accessToken = "";
33     @Value("${instagram.user-id}")
34     private String userId = "YOUR_USER_ID";
35
36     @Scheduled(cron = "0 0 0 * * ?")
37     public void fetchAndStorePosts() {
38         WebClient webClient = WebClient.create("https://graph.instagram.com");
39         try {
40             accessToken = instagramTokenRefresher.getToken();
41         } catch (IOException e) {
42             System.out.println("Error al leer el token de acceso: " + e.getMessage());
43         }
44         webClient.get()
45             .uri(uriBuilder -> uriBuilder
46                 .path("/"+userId+"/media")
47                 .queryParam("fields", "id,caption,media_type,media_url,timestamp")
48                 .queryParam("access_token", accessToken)
49                 .build())
50             .retrieve()
51             .bodyToMono(DataDTO.class)
52             .doOnNext(response -> {
53
54                 System.out.println("Respuesta cruda: " + response);
55             })
56             .flatMapMany(dataDTO -> Flux.fromIterable(dataDTO.getData()))
57             .concatMap(this::processAndStorePost)
58             .doOnTerminate(() -> System.out.println("Proceso de actualización de posts finalizado"))
59             .doOnError(e -> System.err.println("Error al actualizar posts: " + e.getMessage()))
60             .subscribe();
61     }
62
63
64     public void fetchAndStorePostsById(Long id) {
65         WebClient webClient = WebClient.create("https://graph.instagram.com");
66         try {
67             accessToken = instagramTokenRefresher.getToken();
68         } catch (IOException e) {
69             System.out.println("Error al leer el token de acceso: " + e.getMessage());
70         }
71         webClient.get()
72             .uri(uriBuilder -> uriBuilder
73                 .path("/" + userId + "/media")
74                 .queryParam("fields", "id,caption,media_type,media_url,timestamp")
75                 .queryParam("access_token", accessToken)
76                 .build())
77             .retrieve()
78             .bodyToMono(DataDTO.class)
79             .doOnNext(response -> System.out.println("Respuesta cruda: " + response))
80             .flatMapMany(dataDTO -> Flux.fromIterable(dataDTO.getData()))
81             .filter(post -> post.getId().equals(id))
82             .flatMap(this::updateMediaUrlById)
83             .doOnTerminate(() -> System.out.println("Proceso de actualización de posts finalizado"))
84             .doOnError(e -> System.err.println("Error al actualizar posts: " + e.getMessage()))
85             .subscribe();
86     }
87
88     private Mono<Media> updateMediaUrlById(MediaDTO postDTO) {
89         return mediaRepository.findById(postDTO.getId())
90             .flatMap(media -> {
91                 media.setMediaurl(postDTO.getMediaUrl());
92                 media.setTimestamp(postDTO.getTimestamp());
93                 System.out.println("Actualizando URL de la media: " + postDTO.getMediaUrl());
94             })
95     }

```

```
94         return mediaRepository.save(media);
95     })
96     .switchIfEmpty(Mono.error(new RuntimeException("Post no encontrado con id: " +
97         postDTO.getId())));
98 }
99
100 private Mono<Void> processAndStorePost(MediaDTO post) {
101     return mediaRepository.existsById(post.getId())
102         .flatMap(exists -> {
103             if (!exists) {
104                 List<String> hashtags = extractHashtags(post.getCaption());
105                 return savePostWithHashtags(post, hashtags);
106             } else {
107                 return mediaRepository.findById(post.getId())
108                     .flatMap(media -> {
109                         media.setMediaurl(post.getMediaUrl());
110                         media.setTimestamp(post.getTimestamp());
111
112                         return mediaRepository.save(media);
113                     })
114                     .then();
115             }
116         });
117 }
118
119 private List<String> extractHashtags(String caption) {
120     if (caption == null || caption.isEmpty()) {
121         return Collections.emptyList();
122     }
123     Pattern pattern = Pattern.compile("#(\\w+)");
124     Matcher matcher = pattern.matcher(caption);
125     List<String> hashtags = new ArrayList<>();
126     while (matcher.find()) {
127         hashtags.add(matcher.group(1));
128     }
129     return hashtags;
130 }
131
132 private Mono<Void> savePostWithHashtags(MediaDTO post, List<String> hashtags) {
133     if (hashtags.isEmpty()) {
134         return Mono.empty();
135     }
136
137     return Flux.fromIterable(hashtags)
138         .flatMap(hashtag -> {
139             Media media = new Media();
140             System.out.println(post.getMediaType()+" "+post.getMediaUrl());
141             media.setId(post.getId());
142             media.setMediaurl(post.getMediaUrl());
143             media.setTimestamp(post.getTimestamp());
144             media.setCaption(post.getCaption());
145             media.setMediatype(post.getMediaType());
146             media.setHashtag(hashtag);
147
148             return mediaRepository.save(media);
149         })
150         .then();
151 }
152
153 public Flux<Media> extractMediaByHashtag(String hashtag) {
154
155     return mediaRepository.findByHashtag(hashtag);
156 }
157
158 public Flux<Media> extractAllMedia() {
159     return mediaRepository.findAll();
160 }
161
162
163
164 }
165 }
```

En este servicio se pueden destacar los siguientes métodos clave:

- `fetchAndStorePosts()`: realiza la conexión con Instagram y obtiene las publicaciones.
- `processAndStorePost(post)`: comprueba si ya existe en la base de datos, y en caso contrario, la almacena o si existe actualiza el timestamp y la url, ya que al guardarse las imágenes en una [Content Delivery Network \(CDN\)](#), Instagram las elimina cada cierto tiempo y hay que renovar la URL.
- `extractHashtags(caption)`: extrae todos los hashtags de una publicación mediante expresiones regulares.
- `savePostWithHashtags(post, hashtags)`: almacena la publicación junto con los hashtags en MongoDB.

Cuando se invoca el método `fetchAndStorePosts()`, el servicio realiza una petición a la API de Instagram Graph [57] obteniendo los datos en formato [JavaScript Object Notation \(JSON\)](#). A continuación, por cada publicación:

1. Se extraen los campos relevantes: ID, texto, medios, y hashtags.
2. Se verifica si esa publicación ya existe en la base de datos.
3. Si es nueva, se guarda junto con los hashtags en un documento MongoDB.

Por último, existe una clase encargada de refrescar el token de acceso a la API de Instagram, que se invoca cada 50 días para garantizar el acceso continuo a los datos. Estos tokens serán gestionados en un fichero cuya ruta estará definida en las propiedades de cada entorno ([Véase en Apéndice A.5, Listado A.15](#)).

Persistencia y acceso a datos subsistema de publicaciones

La persistencia de los medios extraídos de Instagram se realiza automáticamente mediante la integración con MongoDB a través de Spring Data Reactive. La entidad principal utilizada es `Media`, representada como un documento en la colección `media`, definida mediante la anotación `@Document` en la clase correspondiente ([Listado 6.20](#)).

Cada medio incluye atributos como `mediaurl` (URL del medio), `timestamp` (fecha y hora de publicación), `caption` (descripción del medio), `mediatype` (tipo de medio) y `hashtag` (etiqueta asociada). Para garantizar la unicidad de los registros, el identificador del documento es el id generado por el sistema origen (Instagram).

La clase auxiliar `MediaRepository` extiende `ReactiveMongoRepository`, lo que permite realizar operaciones reactivas sobre la base de datos sin necesidad de escribir consultas SQL tradicionales. Este repositorio expone métodos personalizados como `findByHashtag`, que permiten realizar búsquedas específicas dentro de la colección `media` ([Listado 6.21](#)).

El acceso a la base de datos es completamente reactivo, lo que optimiza la interacción con MongoDB y permite manejar grandes volúmenes de datos de manera eficiente.

A continuación, se presentan los fragmentos relevantes de las clases de modelo y repositorio.

[Listado 6.20: Modelo de documento MongoDB para los medios Media.java](#)

```

6  @Document(collection = "media")
7  public class Media {
8
9      @Id
10     private String id;
11
12     private String mediaurl;
13     private String timestamp;
14     private String caption;
15     private String mediatype;
16     @Indexed
17     private String hashtag;
18     public String getId() {
19         return id;
20     }
21     public void setId(String id) {
22         this.id = id;
23     }
24
25     public String getTimestamp() {
26         return timestamp;
27     }
28     public void setTimestamp(String timestamp) {
29         this.timestamp = timestamp;
30     }

```

Listado 6.21: Repositorio para la persistencia en MongoDB `MediaRepository.java`

```

5
6  import org.springframework.data.mongodb.repository.ReactiveMongoRepository;
7
8  public interface MediaRepository extends ReactiveMongoRepository<Media, String> {
9      Flux<Media> findByHashtag(String hashtag);

```

6.4. Subsistema de extracción de datos meteorológicos

El sistema de extracción de datos meteorológicos obtiene información sobre el clima de una región específica en base a un rango de fechas proporcionado por el usuario. Este subsistema está accesible mediante el controlador `WeatherController`, disponible en el endpoint `/weather`, y se apoya en el servicio `WeatherService` para obtener y devolver los datos en formato reactivo (Listado 6.22).

El controlador ofrece una ruta principal:

`/weather` para obtener datos meteorológicos de un rango de fechas proporcionado por el usuario, usando parámetros `startDate` y `endDate` en formato de fecha International Organization for Standardization (ISO).

La lógica principal reside en el servicio `WeatherService`, el cual se encarga de obtener los datos meteorológicos desde la API de Open-Meteo [58]. Si los datos no están disponibles en la base de datos, el servicio consulta la API para obtener la información histórica o pronosticada, dependiendo de la fecha proporcionada. Los datos obtenidos incluyen la temperatura máxima, mínima y la precipitación, y se clasifican como “Lluvioso”, “Caluroso”, “Frío” o “Normal”, según las condiciones climáticas.

A continuación, se presentan los fragmentos relevantes de implementación. En el Listado 6.22 se presenta el controlador de datos meteorológicos, en el Listado 6.23 la lógica del servicio de extracción de datos meteorológicos, y en el Listado 6.25 el repositorio encargado de la persistencia en MongoDB [24].

Listado 6.22: Fragmento del controlador de datos meteorológicos WeatherController.java

```

15     private final WeatherService weatherService;
16
17     public WeatherController(WeatherService weatherService) {
18         this.weatherService = weatherService;
19     }
20
21     @GetMapping("/weather")
22     public Flux<WeatherData> getWeather(
23         @RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate startDate,
24         @RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate endDate) {
25
26         if (startDate.isAfter(endDate)) {
27             return Flux.error(new IllegalArgumentException("La fecha de inicio no puede ser posterior a la fecha de fin."));
28         }
29
30         return weatherService.getWeatherForDateRange(startDate, endDate);
31     }
32 }
```

Listado 6.23: Servicio para la gestión de datos meteorológicos WeatherService.java

```

18 @Service
19 public class WeatherService {
20
21     @Autowired
22     private WeatherDataRepository weatherDataRepository;
23
24     private final WebClient webClientForecast =
25         WebClient.create("https://api.open-meteo.com/v1/forecast");
26     private final WebClient webClientHistorical = WebClient.create("https://hi_"
27         storical-forecast-api.open-meteo.com/v1/forecast");
28
29     public Flux<WeatherData> getWeatherForDateRange(LocalDate startDate,
30         LocalDate endDate) {
31
32         return Flux.range(0, (int) (endDate.toEpochDay() -
33             startDate.toEpochDay() + 1))
34             .map(startDate::plusDays)
35             .flatMap(this::checkAndFetchWeather);
36     }
37
38     private Mono<WeatherData> checkAndFetchWeather(LocalDate date) {
39
40         LocalDate today = LocalDate.now();
41         boolean isHistorical = date.isBefore(today) ||
42             date.isAfter(today.plusDays(15));
43         if (isHistorical) {
44             date = date.minusYears(1);
45         }
46         String dateString =
47             date.format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
48
49         return weatherDataRepository.findById(dateString)
50             .switchIfEmpty(fetchAndStoreWeather(date, dateString));
51     }
52 }
```

```
45     }
46
47     private Mono<WeatherData> fetchAndStoreWeather(LocalDate date, String
48         dateString) {
49         LocalDate today = LocalDate.now();
50         boolean isHistorical = date.isBefore(today) ||
51             date.isAfter(today.plusDays(15));
52
53         WebClient selectedClient = isHistorical ? webClientHistorical :
54             webClientForecast;
55         System.out.println("isHistorical: " + isHistorical);
56
57         if (isHistorical) {
58             date = date.minusYears(1);
59         }
60
61         String formattedDate =
62             date.format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
63
64         String url = String.format(
65             "?latitude=40.598&longitude=-0.457&daily=temperature_2m_max,pr_"
66             "ecipitation_sum,temperature_2m_min&start_date=%s&end_date=%s",
67             formattedDate, formattedDate
68         );
69         System.out.println(formattedDate);
70         return selectedClient.get()
71             .uri(url)
72             .retrieve()
73             .bodyToMono(new ParameterizedTypeReference<Map<String,
74                 Object>>() {})
75             .flatMap(response -> processWeatherResponse(response))
76             .flatMap(weatherDataRepository::save);
77     }
78
79     private Mono<WeatherData> processWeatherResponse(Map<String, Object>
80         response) {
81         Map<String, Object> daily = (Map<String, Object>)
82             response.get("daily");
83         List<String> times = (List<String>) daily.get("time");
84         List<Double> maxTemperatures = (List<Double>)
85             daily.get("temperature_2m_max");
86         List<Double> minTemperatures = (List<Double>)
87             daily.get("temperature_2m_min");
88         List<Double> precipitation = (List<Double>)
89             daily.get("precipitation_sum");
90
91         if (times.isEmpty() || maxTemperatures.isEmpty() ||
92             minTemperatures.isEmpty() || precipitation.isEmpty()) {
93             return Mono.error(new RuntimeException("No se han encontrado datos
94                 para el día solicitado"));
95         }
96
97         WeatherData weatherData = new WeatherData();
98         weatherData.setTimes(times);
99         weatherData.setMaxTemperatures(maxTemperatures);
100        weatherData.setMinTemperatures(minTemperatures);
101        weatherData.setPrecipitation(precipitation);
102        return Mono.just(weatherData);
103    }
104}
```

```

85     String time = times.get(0);
86     double precip = precipitation.get(0);
87     double avgTemperature = (maxTemperatures.get(0) +
88         minTemperatures.get(0)) / 2;
89     double minTemperature = (minTemperatures.get(0));
90     double maxTemperature = (maxTemperatures.get(0));
91     double totalPrecipitation = precip;
92
92     String classification = classifyDay(avgTemperature,
93         totalPrecipitation);
93     WeatherData weatherData = new WeatherData(time,
94         minTemperature, maxTemperature, totalPrecipitation, classification);
94
95
96     return Mono.just(weatherData);
97 }
98
99     private String classifyDay(double temperature, double precipitation) {
100         if (precipitation > 0) {
101             return "Lluvioso";
102         } else if (temperature >= 25) {
103             return "Caluroso";
104         } else if (temperature < 10) {
105             return "Frío";
106         } else {
107             return "Normal";
108         }
109     }
110 }
111 }
```

Los lenguajes te han quedado muy bien - Yo te he puesto el label del caption en castellano.

Persistencia y acceso a datos en el subsistema de extracción de datos meteorológicos

La persistencia de los datos meteorológicos se realiza automáticamente mediante la integración con MongoDB a través de Spring Data Reactive. La entidad principal utilizada es `WeatherData`, representada como un documento en la colección `weather_data`, definida mediante la anotación `@Document` en la clase correspondiente (Listado 6.24).

Cada entrada de datos meteorológicos incluye atributos como fecha, temperatura mínima, temperatura máxima, precipitación y clasificación del día. El identificador del documento se corresponde con la fecha, lo que garantiza que no haya duplicados para una misma fecha.

El acceso a la base de datos se gestiona mediante el repositorio reactivo `WeatherDataRepository`, que extiende la interfaz `ReactiveMongoRepository`. Este repositorio permite realizar búsquedas específicas dentro de la colección `weather_data` mediante consultas como `findByDate`, sin necesidad de definir consultas manualmente (Listado 6.25).

Gracias a esta integración, los datos meteorológicos se guardan y recuperan de forma eficiente, permitiendo consultas rápidas y actualizadas.

A continuación, se presentan los fragmentos relevantes de las clases de modelo y repositorio.

Listado 6.24: Modelo de documento MongoDB para datos meteorológicos *WeatherData.java*

```
6  @Document(collection = "weather_data")
7  public class WeatherData {
8
9      @Id
10     private String date;
11     private double minTemp;
12     private double maxTemp;
13     private double precipitation;
14     private String classification;
15
16
17     public WeatherData() {}
18
19     public WeatherData(String date, double minTemp, double maxTemp, double precipitation, String
20                         classification) {
21         this.date = date;
22         this.minTemp = minTemp;
23         this.maxTemp = maxTemp;
24
25         this.precipitation = precipitation;
26         this.classification = classification;
27     }
28
29
30     public String getDate() {
31         return date;
32     }
33
34     public void setDate(String date) {
35         this.date = date;
36     }
37
38
39     public double getMinTemp() {
40         return minTemp;
41     }
42
43     public void setMinTemp(double minTemp) {
44         this.minTemp = minTemp;
45     }
46
47     public double getMaxTemp() {
48         return maxTemp;
49     }
50
51     public void setMaxTemp(double maxTemp) {
52         this.maxTemp = maxTemp;
53     }
54
55     public double getPrecipitation() {
56         return precipitation;
57     }
58
59     public void setPrecipitation(double precipitation) {
60         this.precipitation = precipitation;
61     }
62
63     public String getClassification() {
64         return classification;
65     }
66
67     public void setClassification(String classification) {
68         this.classification = classification;
69     }
70 }
```

Listado 6.25: Repositorio de datos meteorológicos WeatherDataRepository.java

```

1 package hemal.uv.es.tiempo.repositories;
2
3
4
5 import org.springframework.data.mongodb.repository.Query;
6 import org.springframework.data.mongodb.repository.ReactiveMongoRepository;
7
8 import hemal.uv.es.tiempo.domain.WeatherData;
9 import reactor.core.publisher.Mono;
10
11
12 public interface WeatherDataRepository extends ReactiveMongoRepository<WeatherData, String> {
13     @Query("{ 'id' : ?0 }")
14     Mono<WeatherData> findByDate(String date);
15 }
```

6.5. Aplicación Frontend y diseño de la interfaz de usuario

En esta sección se describen los principales componentes que conforman la implementación del **Frontend** de la aplicación, incluyendo tanto los componentes como los servicios desarrollados. Asimismo, se presenta el diseño final de dichos elementos, mostrando el resultado visual y funcional alcanzado tras su integración.

6.5.1. Componentes principales

La aplicación **Frontend** está desarrollada utilizando Angular [13]. A continuación, se describen los principales componentes que conforman la aplicación, así como su funcionalidad y diseño.

Componente AdministradorComponent

El componente **AdministradorComponent** representa la interfaz administrativa de la aplicación, permitiendo gestionar las reservas realizadas por los usuarios. Está compuesto por una lógica en TypeScript encargada de manejar eventos, peticiones al backend y control de estado, y por una plantilla **HyperText Markup Language 5.0 (HTML5)** que muestra una tabla interactiva con las reservas y botones de acción.

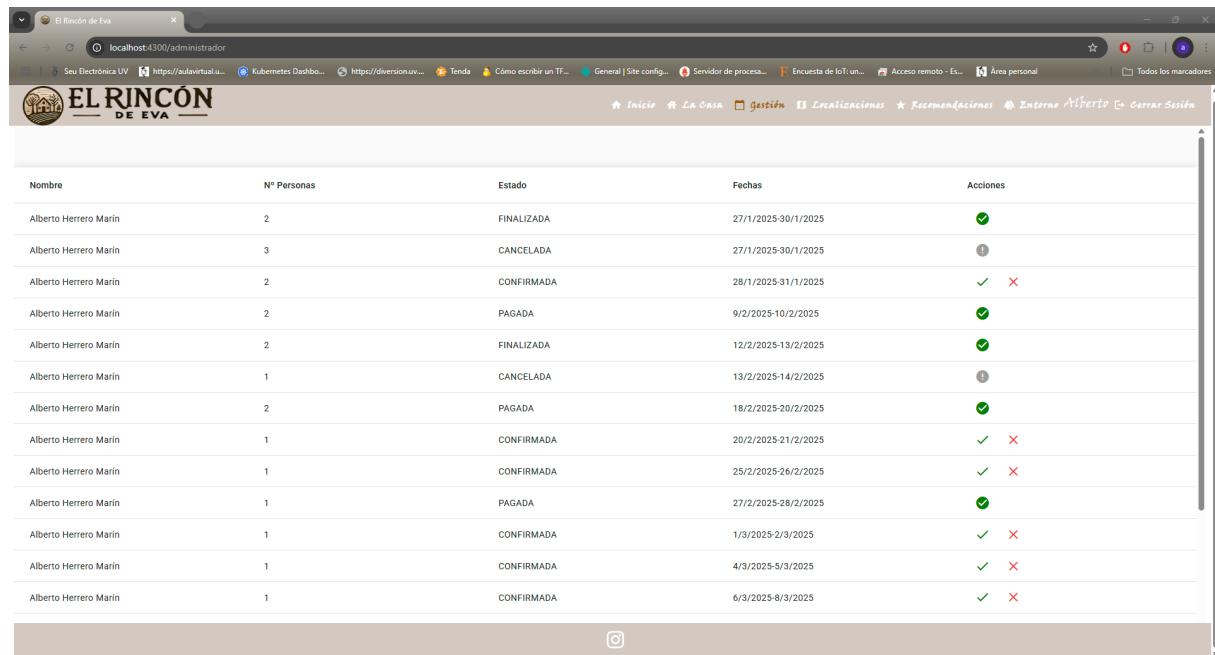
La lógica de negocio se implementa en el archivo **administrador.component.ts**, donde destacan los métodos **aprobarPagoReserva** y **cancelarReserva**. Ambos abren un diálogo para recoger un motivo por parte del administrador y actúan en función de la respuesta del usuario, comunicándose con el backend para actualizar el estado de la reserva. Esto puede verse en el Apéndice A.6, Listado A.16.

Por otra parte, el archivo de plantilla **administrador.component.html** define la estructura visual del componente mediante una tabla de Angular Material (**mat-table**) donde se muestran las columnas: nombre, número de personas, estado, fechas y acciones. Las acciones varían en función del estado de la reserva, mostrando botones para aprobar o cancelar únicamente si la reserva está pendiente. Esta lógica de presentación puede consultarse en el Apéndice A.7, Listado A.26.

esta sección se lee muy bien
y todo el código se encuentra en
apéndices!

Fase sc

Para finalizar, en la Figura 6.1 se muestra el diseño final del componente `AdministradorComponent`, que incluye una tabla con las reservas y botones de acción para gestionar cada una de ellas.



The screenshot shows a web browser window titled 'El Rincón de Eva'. The URL is 'localhost:4300/administrador'. The page has a header with the logo 'EL RINCON DE EVA' and navigation links: 'Inicio', 'La Casa', 'Gestión', 'Localizaciones', 'Recomendaciones', 'Internos Alberto', and 'Cerrar Sesión'. Below the header is a table with the following columns: 'Nombre', 'Nº Personas', 'Estado', 'Fechas', and 'Acciones'. The table contains 15 rows of reservation data. The 'Acciones' column includes a green checkmark icon and a red X icon for each row. The data in the table is as follows:

Nombre	Nº Personas	Estado	Fechas	Acciones
Alberto Herrero Marín	2	FINALIZADA	27/1/2025-30/1/2025	✓
Alberto Herrero Marín	3	CANCELADA	27/1/2025-30/1/2025	ⓘ
Alberto Herrero Marín	2	CONFIRMADA	28/1/2025-31/1/2025	✓ ✘
Alberto Herrero Marín	2	PAGADA	9/2/2025-10/2/2025	✓
Alberto Herrero Marín	2	FINALIZADA	12/2/2025-13/2/2025	✓
Alberto Herrero Marín	1	CANCELADA	13/2/2025-14/2/2025	ⓘ
Alberto Herrero Marín	2	PAGADA	18/2/2025-20/2/2025	✓
Alberto Herrero Marín	1	CONFIRMADA	20/2/2025-21/2/2025	✓ ✘
Alberto Herrero Marín	1	CONFIRMADA	25/2/2025-26/2/2025	✓ ✘
Alberto Herrero Marín	1	PAGADA	27/2/2025-28/2/2025	✓
Alberto Herrero Marín	1	CONFIRMADA	1/3/2025-2/3/2025	✓ ✘
Alberto Herrero Marín	1	CONFIRMADA	4/3/2025-5/3/2025	✓ ✘
Alberto Herrero Marín	1	CONFIRMADA	6/3/2025-8/3/2025	✓ ✘

Figura 6.1: Diseño del componente `AdministradorComponent`

Componente `InicioComponent`

El componente `InicioComponent` gestiona la visualización de reseñas, la interacción del usuario con las reseñas (expandiéndolas y cargando más), así como el envío de reseñas y mensajes de contacto, con validación y manejo de errores.

La lógica de negocio se implementa en el archivo `inicio.component.ts`, donde destacan los métodos `onSubmitReview` y `onSubmit`. El método `onSubmitReview` se encarga de enviar una nueva reseña al backend, validando previamente el formulario de reseñas. Si la reseña es válida, se procesa y se envía al servicio `ReviewService`, mostrando un mensaje de confirmación al usuario mediante un `snackBar`. Después de enviar la reseña, se actualiza el estado de la sesión del usuario y se recargan las reseñas. Por otro lado, el método `onSubmit` maneja el envío de un formulario de contacto, validando los campos requeridos y, en caso de ser válido, enviando el mensaje al backend a través del `MailService`. Al igual que con las reseñas, si el contacto se envía correctamente, se muestra una notificación al usuario. Ambos métodos interactúan con el backend para actualizar la información, como puede verse en el Apéndice A.6, Listado A.17. El archivo de plantilla `inicio.component.html` utiliza varios componentes de Angular Material para estructurar la interfaz, pero lo más relevante en cuanto a la interacción con el backend se encuentra en el manejo de reseñas y el formulario de contacto.

En la sección de reseñas, se emplea el `ngFor` para iterar sobre las reseñas disponibles en el backend. Para cada reseña, se muestra una calificación con estrellas utilizando el componente `mat-icon`, y los detalles adicionales de la reseña, como limpieza, ubicación y servicios, se cargan dinámicamente mediante la función `getStars()`. Las reseñas se obtienen a través de un servicio que consulta el backend, y la lógica de visualización de

las mismas está implementada en el archivo `inicio.component.ts`. El usuario también puede cargar más reseñas mediante un botón que llama a la función `loadMoreReviews()`.

Además, el componente incluye un formulario para que los usuarios dejen sus reseñas. Este formulario está vinculado a un `FormGroup` en el archivo `inicio.component.ts`, y su envío activa la función `onSubmitReview()`, que envía la información al backend para ser guardada. La validación del formulario se realiza de forma reactiva, asegurando que se cumplan los requisitos antes de permitir el envío.

En la sección de contacto, se incluye otro formulario que permite a los usuarios enviar un mensaje. Este formulario está también vinculado a un `FormGroup` y, al ser enviado, ejecuta la función `onSubmit(contactForm)`, que se encarga de enviar los datos al backend para su procesamiento.

Estos componentes de Angular interactúan con el backend mediante servicios que gestionan las solicitudes HTTP y las respuestas correspondientes. La implementación detallada de estas interacciones puede consultarse en el Apéndice A.7, Listado A.27.

Para finalizar, en la Figura 6.2 se muestra el diseño del apartado para revisar las reseñas existentes, en la Figura 6.3 se muestra como crearía un cliente reciente una nueva reseña (solo se muestra si la reserva del cliente ha finalizado y no ha dejado una reseña después de ésta) y en la Figura 6.4 se muestra el apartado para crear una consulta al propietario de la casa rural. En este último apartado, el cliente puede dejar su nombre, correo electrónico y mensaje, que serán enviados al propietario de la casa rural a través del servicio de correo electrónico implementado en el backend.

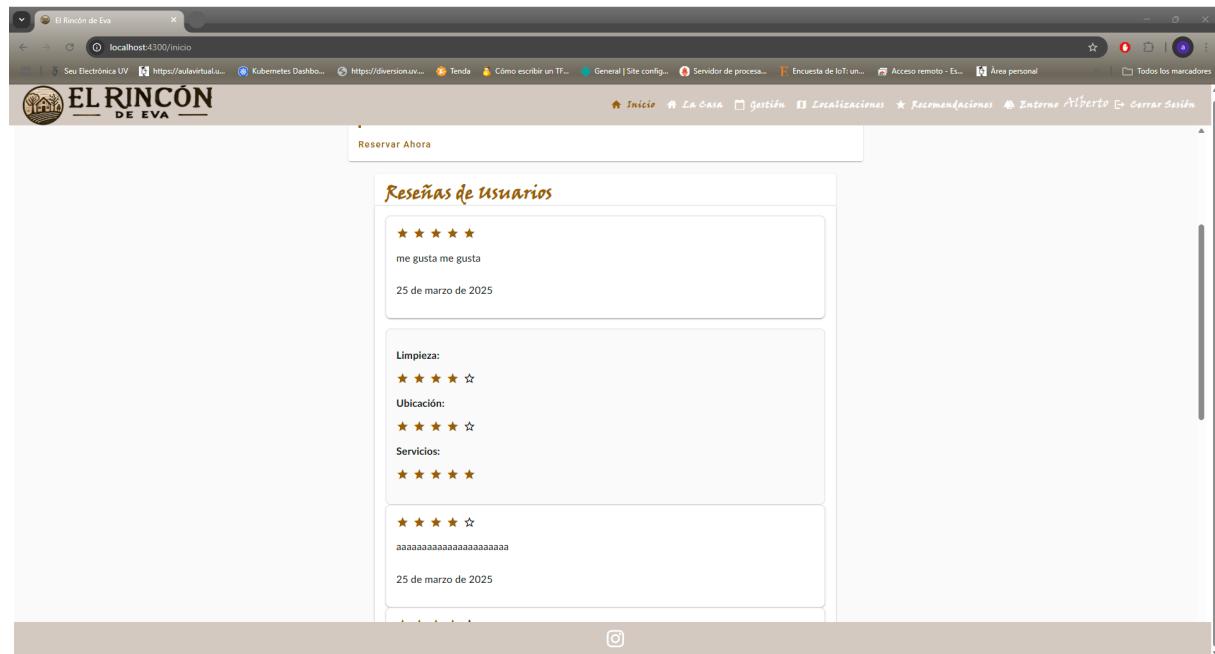


Figura 6.2: Apartado para consultar reseñas existentes de `InicioComponent`

Componente `LaCasaComponent`

El componente `LaCasaComponent` gestiona la visualización de contenido multimedia relacionado con la casa rural. Su propósito es proporcionar una experiencia inmersiva

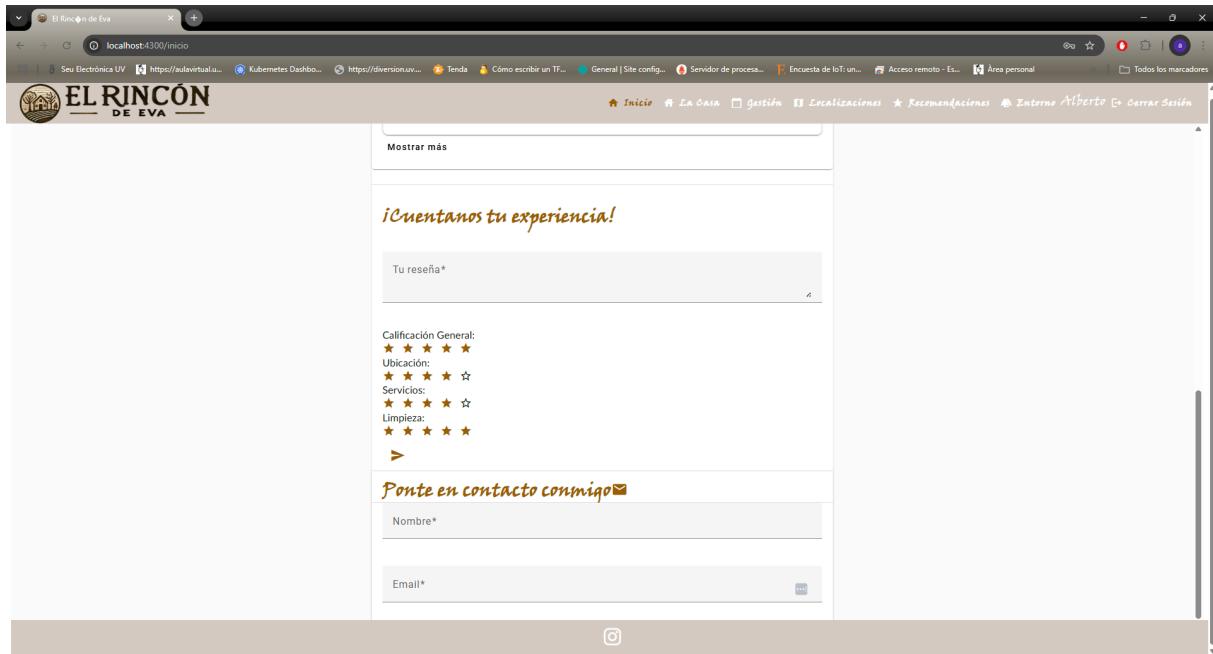


Figura 6.3: Apartado con formulario para crear reseña de `InicioComponent`

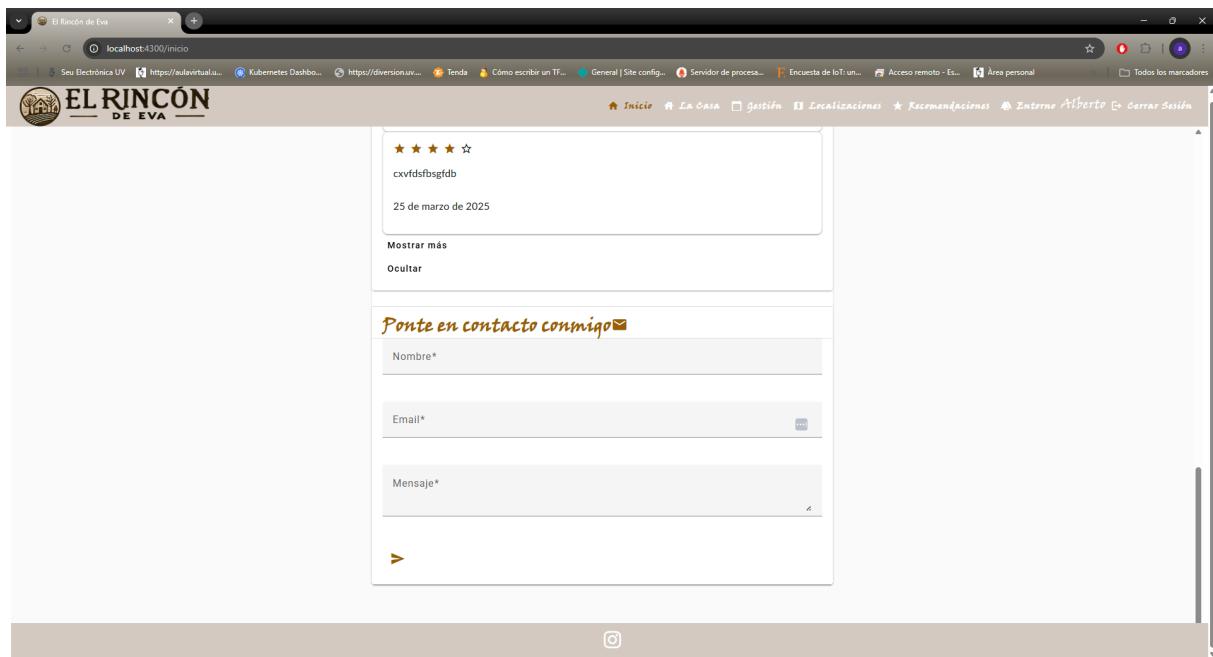


Figura 6.4: Apartado con formulario para crear una consulta de `InicioComponent`

a través de un video introductorio, un carrusel de imágenes que representan distintos espacios del alojamiento y una descripción textual detallada sobre las instalaciones, habitaciones y servicios ofrecidos.

La lógica de negocio se implementa en el archivo `la-casa.component.ts`, donde destacan los métodos `ngOnInit`, `fetchMedia`, `prevSlide` y `nextSlide`. En el método `ngOnInit` se inicializa la URL del video introductorio concatenando la base del backend con la ruta al recurso. Asimismo, se llama al método `fetchMedia()`, encargado de recuperar las imágenes del backend a través del servicio `MediaService`, que las devuelve como una colección de objetos `Media`. Estas imágenes se almacenan en el array `slides` para ser

utilizadas en el carrusel.

Los métodos `prevSlide()` y `nextSlide()` permiten la navegación manual entre las imágenes del carrusel, modificando el índice `currentSlide` que controla qué conjunto de imágenes se muestra actualmente. El carrusel está configurado para mostrar dos imágenes por página, utilizando la constante `imagesPerSlide`.

El archivo de plantilla `la-casa.component.html` estructura la interfaz del usuario en varias secciones claramente diferenciadas. En primer lugar, se encuentra el video introductorio, que se obtiene dinámicamente desde el backend y se presenta con controles para su reproducción. A continuación, se encuentra el carrusel de imágenes, implementado con un contenedor que itera sobre `slides` utilizando `ngFor` y calcula el subconjunto de imágenes a mostrar en función de `currentSlide`. La implementación detallada del componente puede consultarse en el Apéndice A.6, Listado A.18, mientras que la estructura HTML5 se encuentra en el Listado A.28 del Apéndice A.7.

En la Figura 6.5 se muestra la sección del carrusel de imágenes (obtenidas de la integración de PublicacionesAPI) y el vídeo (obtenido del backend estático).

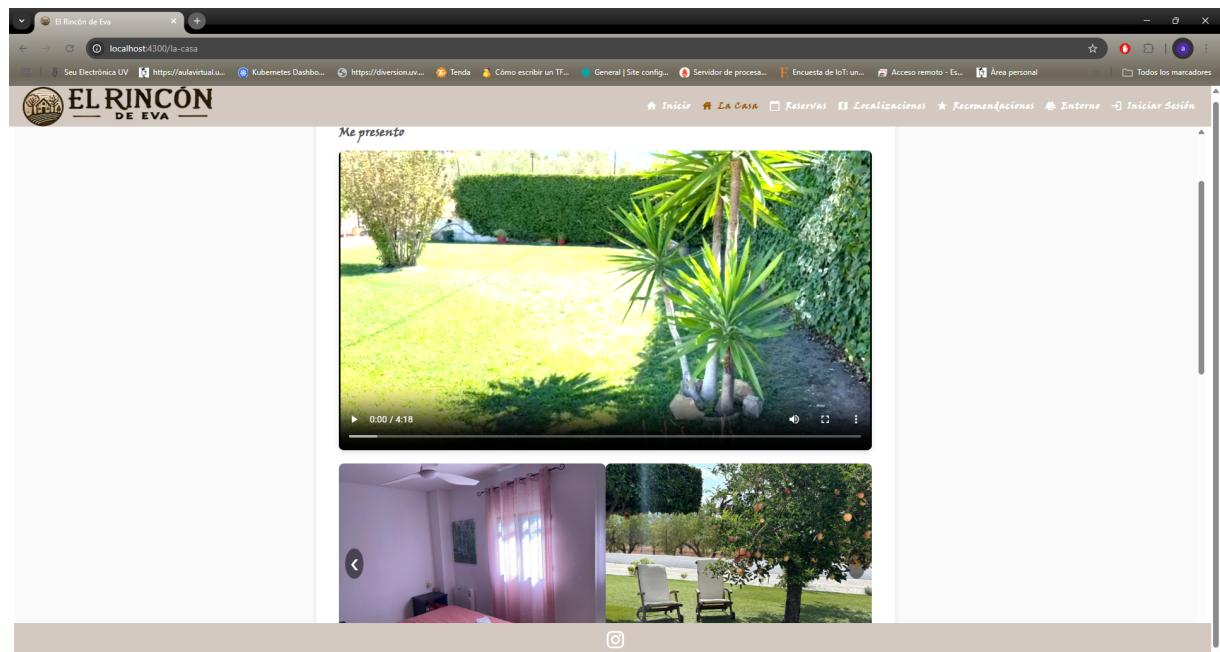


Figura 6.5: Carrusel de imágenes en el componente `LaCasaComponent`

Componente ReservaComponent

El componente `ReservaComponent` se encarga de gestionar todo el proceso de reserva de la casa rural. Para ello, se ha integrado un calendario interactivo mediante el componente `mwl-calendar-month-view`, cuya vista ha sido personalizada para adaptarse a los requisitos del negocio. En este calendario, los usuarios pueden visualizar las fechas ya reservadas, las cuales se cargan mediante el método `initializeEvents()`, que obtiene dicha información desde el servicio `reservaService`.

Además, el componente permite seleccionar rangos de fechas no reservadas. Una vez seleccionado un rango válido, se ejecuta el método `fetchWeather()`, que envía ese intervalo al `Backend` para obtener la previsión meteorológica de cada día del periodo. Esta

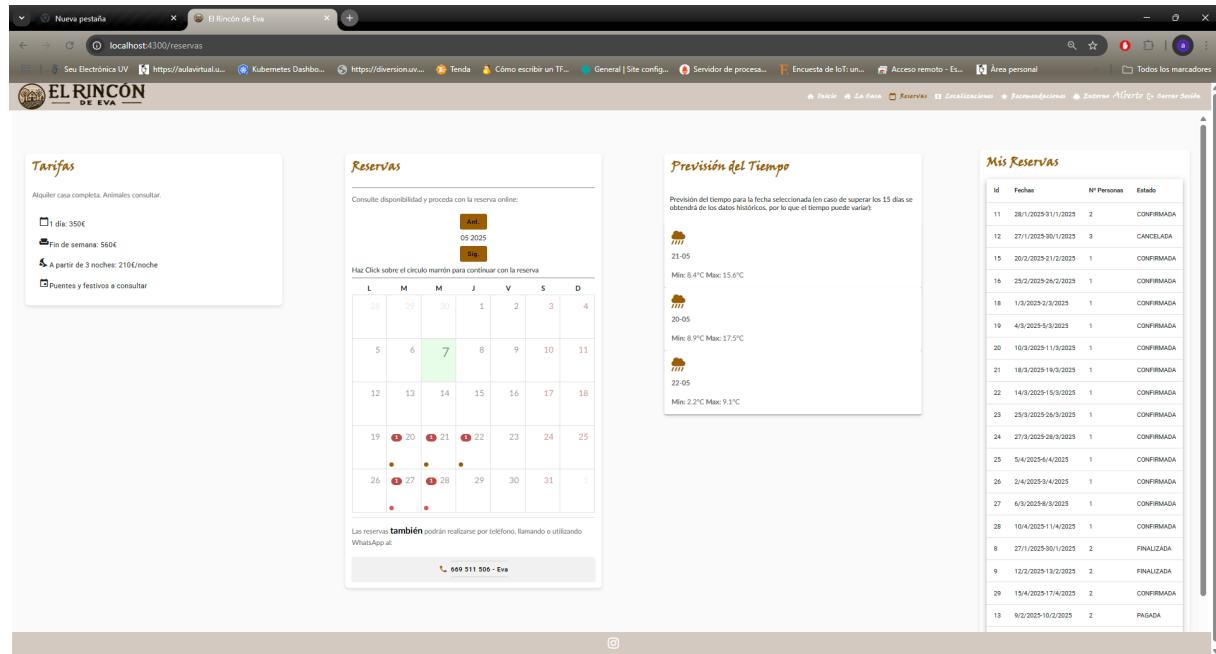


Figura 6.6: Vista general diseño del componente ReservasComponent

información se encapsula en objetos `WeatherData`, los cuales se representan visualmente en un `MatCard` contiguo. Si el usuario hace clic sobre uno de los días seleccionados (marcados con un punto marrón), se invoca el método `onEventClicked()`, que despliega un formulario para completar los datos necesarios para la reserva, en caso de que no se puedan obtener directamente del usuario autenticado.

Una vez completado el formulario, se muestra un resumen de la reserva en un cuadro de diálogo (`MatDialog`), heredado del componente padre. Este resumen incluye el precio final calculado automáticamente en función de las tarifas aplicables. Si el usuario confirma, la reserva se envía de forma síncrona al `Backend` a través del servicio de reservas.

Adicionalmente, el componente permite consultar las reservas anteriores y futuras del usuario mediante una tabla. Esta tabla se alimenta de los datos proporcionados por el método `getReservasUser(email)` del servicio `reservaService`, y muestra entradas de tipo `Reserva`.

La implementación completa de este componente se detalla en el Apéndice A.6, Listado A.19, mientras que su estructura `HTML5` puede consultarse en el Listado A.29 del Apéndice A.7. En la Figura 6.6 se muestra una vista completa del diseño de este componente. En la Figura 6.7 se puede observar el cuadro de diálogo para finalizar la reserva el cual es generado por el componente `resumen-reserva.component`.

Componente ComoLlegarComponent

El componente `ComoLlegarComponent` está diseñado para mostrar un mapa con la ubicación exacta de la casa rural, usando los mapas de Google. Además, muestra un mapa interactivo con rutas turísticas alrededor de la casa rural, aprovechando la biblioteca `Leaflet` y datos obtenidos mediante el servicio `OverpassService`, que utiliza la API de `OpenStreetMap`. Al inicializarse, se carga un mapa centrado en la ubicación de la casa rural, y se solicitan las rutas disponibles en la zona mediante el método `loadRoutes()`.

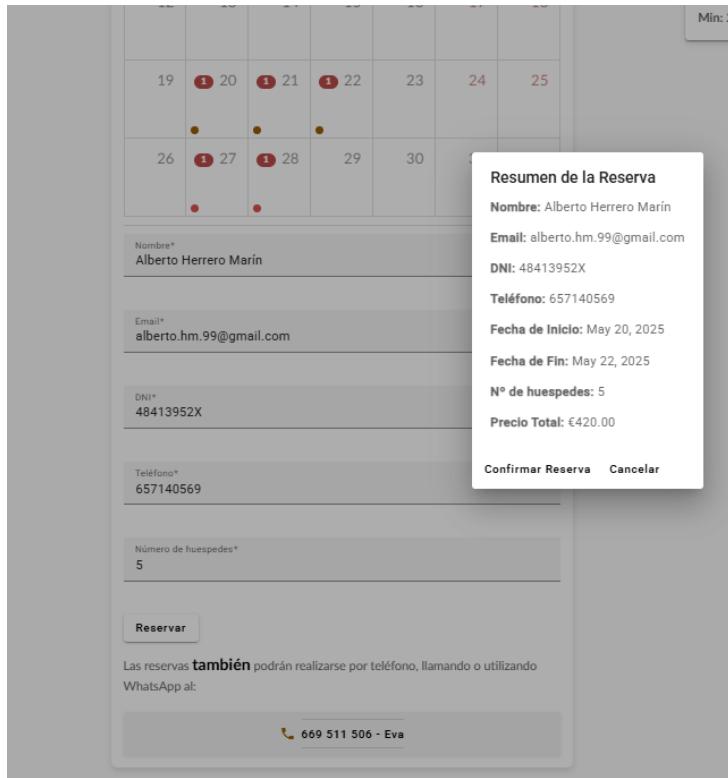


Figura 6.7: Vista del resumen de la reserva en el componente `ReservasComponent`

Este método procesa los elementos recibidos, almacenando los nodos en una caché interna mediante `filterNodes()`, y dibujando cada ruta como una polilínea sobre el mapa. Cada ruta es interactiva: al hacer clic sobre ella se ejecuta `showRouteDetails()`, que despliega un popup con opciones para visualizar la ruta en OpenStreetMap o descargarla en formato GPX. Esta descarga se genera dinámicamente mediante `downloadGPX()`, que transforma los datos de la ruta con el método `convertToGPX()`.

El mapa ajusta automáticamente el encuadre para mostrar todas las rutas encontradas, y permite una experiencia de navegación intuitiva y enriquecida para el usuario.

La implementación completa del componente puede consultarse en el Apéndice A.6, Listado A.20, mientras que su estructura HTML5 se encuentra en el Listado A.30 del Apéndice A.7. En la Figura 6.8 se muestra una vista completa del diseño de este componente.

Componente `RecomendacionesComponent`

El componente `RecomendacionesComponent` permite mostrar recomendaciones relacionadas con la gastronomía y las fiestas del entorno. En su inicialización, recupera listas de objetos de tipo `Entorno` y `Media` mediante los servicios `EntornoService` y `MediaService`, que se asigna al array `slidesFiestas` y `slidesComida` respectivamente.

El componente define dos carruseles: uno para imágenes de comida y otro para eventos festivos. La navegación entre imágenes se gestiona mediante métodos que actualizan los índices de desplazamiento de cada carrusel (`prevSlideComida()`, `nextSlideComida()`, `prevSlideFiesta()`, `nextSlideFiesta()`), mostrando los elementos en bloques de tres.

El código fuente se incluye en el Apéndice A.6, Listado A.21, y su correspondiente

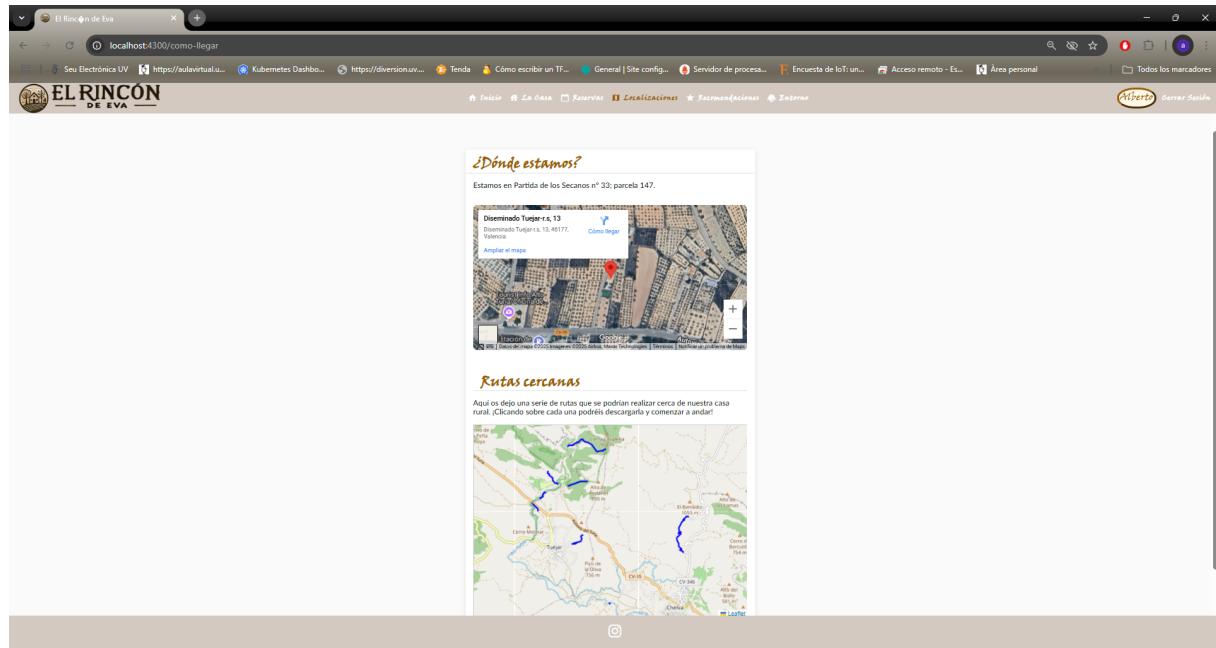


Figura 6.8: Vista del diseño del componente ComoLlegarComponent

plantilla [HTML5](#) puede consultarse en el Listado A.31 del Apéndice A.7.

En la Figura 6.9 se muestra una vista completa del diseño de este componente.

Componente EntornoComponent

El componente `EntornoComponent` es responsable de mostrar los principales puntos de interés turístico de las localidades de Tuéjar y Chelva. Para ello, durante la inicialización se realiza una petición al backend mediante el servicio `EntornoService`, que obtiene un array de objetos `Entorno` a través del método `getEntorno()`. A partir de los datos obtenidos, se filtran dos subconjuntos diferenciados: uno para la localidad de Tuéjar y otro para Chelva, excluyendo en ambos casos los elementos cuya categoría corresponde a festividades.

Cada uno de estos subconjuntos se utiliza para poblar dinámicamente una sección distinta del componente, en la que se representa la información mediante tarjetas (`mat-card`) que muestran la imagen, título, descripción y un enlace al recurso externo asociado a cada lugar. La maquetación se presenta en formato de cuadrícula adaptable a través del uso de clases personalizadas.

La implementación detallada del componente puede consultarse en el Apéndice A.6, Listado A.22, mientras que la estructura [HTML5](#) se encuentra en el Listado A.32 del Apéndice A.7. En la Figura 6.10 se muestra una vista completa del diseño de este componente.

6.6. ~~Dockerización y despliegue de aplicaciones~~

básico en contenedores

La dockerización y el despliegue de la aplicación se han llevado a cabo exclusivamente para la parte correspondiente al Backend. Para ello, se han generado previamente los archivos Dockerfile para cada una de las cuatro aplicaciones desarrolladas con Spring

básico en contenedores Docker

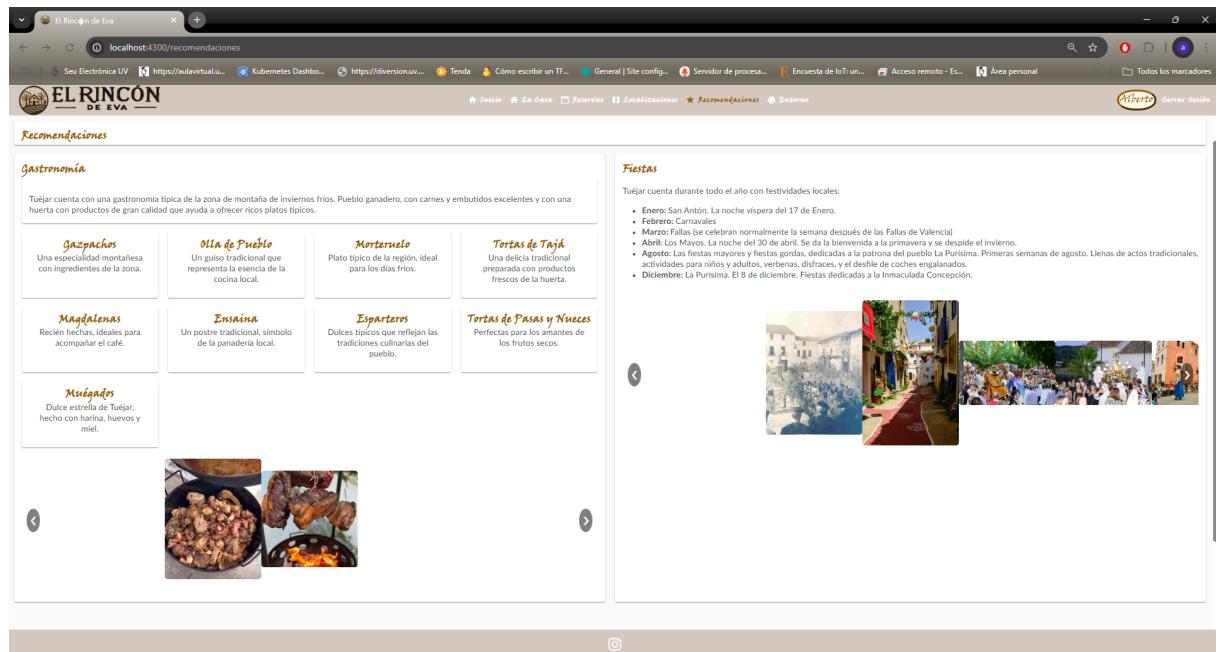


Figura 6.9: Vista del diseño del componente RecomendacionesComponent

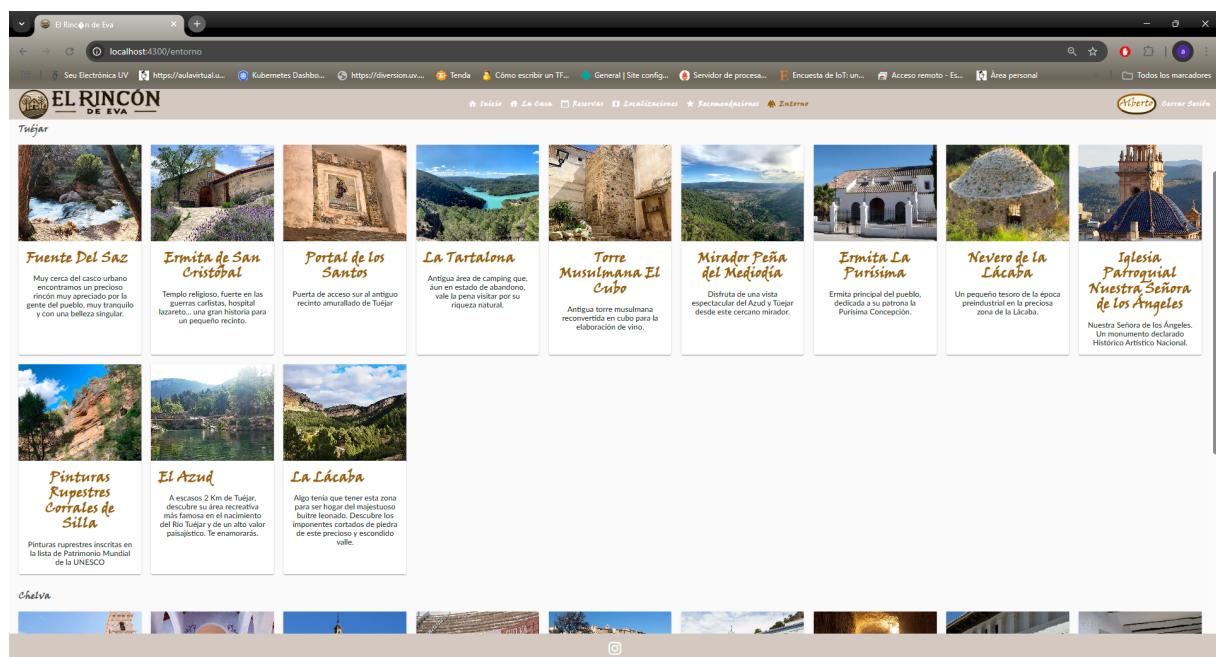


Figura 6.10: Vista del diseño del componente EntornoComponent

Boot.

En el Listado 6.26 se presenta el Dockerfile correspondiente al servicio principal de la aplicación. En dicho archivo, se puede observar cómo se transfiere el paquete generado mediante la herramienta Maven [59], a través del comando `mvn clean package`, al entorno de ejecución definido en Docker. Posteriormente, se procede a la ejecución del archivo `.jar` utilizando el perfil denominado `despliegue`, el cual ha sido configurado con variables de entorno específicas en el entorno Spring Boot. Estas variables serán sobreescritas posteriormente mediante los archivos de configuración utilizados en los despliegues de Kubernetes.

Los restantes Dockerfile presentan una estructura similar, diferenciándose únicamente en el nombre de la aplicación correspondiente. Para mayor detalle, en el Apéndice A.8 se incluyen los Dockerfile de las otras tres aplicaciones del sistema.

Listado 6.26: Dockerfile del servicio de la aplicación principal Dockerfile

```

1 FROM eclipse-temurin:24-jdk-alpine-3.20
2 WORKDIR /app
3 COPY *.jar elrincondeeva.jar
4 EXPOSE 8080
5 CMD ["java", "-Dspring.profiles.active=despliegue", "-jar", "elrincondeeva.jar"]

```

Perfecto sólo un error

Posteriormente, estas imágenes se han subido a un registro privado de Docker, donde se encuentran disponibles para su despliegue en el clúster de Kubernetes del entorno de producción.

A continuación, se describen cada uno de los elementos que componen el despliegue de la aplicación en el clúster de Kubernetes, los cuales se han generado mediante ficheros YAML Ain't Markup Language (YAML):

6.6.1. Almacenamiento Persistente: PersistentVolume y PersistentVolumeClaim

- PostgreSQL:

- Se define un volumen persistente `postgres-pv` para el almacenamiento de datos de la base de datos.
- El volumen se reclama mediante `postgres-pvc`.

Todo listo se dice esto

Listado 6.27: PersistentVolume de PostgreSQL

```

5 apiVersion: v1
6 kind: PersistentVolume
7 metadata:
8   name: postgres-pv
9 spec:
10   capacity:
11     storage: 1Gi
12   accessModes:
13     - ReadWriteOnce
14   hostPath:
15     path: "/data/postgres"

```

Listado 6.28: PersistentVolumeClaim de PostgreSQL

```

17 apiVersion: v1
18 kind: PersistentVolumeClaim
19 metadata:
20   name: postgres-pvc
21 spec:
22   accessModes:
23     - ReadWriteOnce
24   resources:
25     requests:
26       storage: 1Gi

```

■ MongoDB:

- Se declara el volumen persistente mongo-pv.
- Se reclama con mongo-pvc para uso compartido por varias aplicaciones.

Listado 6.29: PersistentVolume de MongoDB

```

30 apiVersion: v1
31 kind: PersistentVolume
32 metadata:
33   name: mongo-pv
34 spec:
35   capacity:
36     storage: 1Gi
37   accessModes:
38     - ReadWriteOnce
39   hostPath:
40     path: "/data/mongo"

```

Listado 6.30: PersistentVolumeClaim de MongoDB

```

42 apiVersion: v1
43 kind: PersistentVolumeClaim
44 metadata:
45   name: mongo-pvc
46 spec:
47   accessModes:
48     - ReadWriteOnce
49   resources:
50     requests:
51       storage: 1Gi

```

■ Token de Instagram (publicacionesapi):

- Volumen token-pv para almacenar el fichero token.txt.
- PersistentVolumeClaim asociado: token-pvc.

Listado 6.31: PersistentVolume del token de Instagram

```

55 apiVersion: v1
56 kind: PersistentVolume
57 metadata:
58   name: token-pv
59 spec:
60   capacity:
61     storage: 1Mi
62   accessModes:
63     - ReadWriteOnce
64   hostPath:
65     path: /mnt/data/token-store

```

Listado 6.32: PersistentVolumeClaim del token de Instagram

```

69  apiVersion: v1
70  kind: PersistentVolumeClaim
71  metadata:
72    name: token-pvc
73  spec:
74    accessModes:
75      - ReadWriteOnce
76    resources:
77      requests:
78        storage: 1Mi

```

6.6.2. Despliegues y Servicios de las Aplicaciones y Bases de Datos

El despliegue de las aplicaciones y bases de datos se realiza mediante recursos de tipo **Deployment** y **Service**. A continuación, se describen los elementos más relevantes de cada uno de ellos.

Componentes del Despliegue en Kubernetes

PostgreSQL

- El **Deployment de PostgreSQL** (6.33) define:
 - Una única réplica del contenedor con la imagen oficial de `postgres:17`.
 - Variables de entorno para configurar el nombre de la base de datos, usuario y contraseña.
 - Montaje de un volumen persistente a `/var/lib/postgresql/data`, usando el `PersistentVolumeClaim` llamado `postgres-pvc`.
- El **Service de PostgreSQL** (6.34) permite el acceso interno al contenedor:
 - Expone el puerto `Transmission Control Protocol (TCP)` 5432.
 - Usa `clusterIP: None` para permitir la detección por `Domain Name System (DNS)` entre pods.

Listado 6.33: Deployment de PostgreSQL

```

84  apiVersion: apps/v1
85  kind: Deployment
86  metadata:
87    name: postgres
88  spec:
89    replicas: 1
90    selector:
91      matchLabels:
92        app: postgres
93    template:
94      metadata:
95        labels:
96          app: postgres
97    spec:
98      containers:
99        - name: postgres
100       image: postgres:17
101      ports:
102        - containerPort: 5432

```

```

103     env:
104         - name: POSTGRES_DB
105             value: gestion
106         - name: POSTGRES_USER
107             value: admin
108         - name: POSTGRES_PASSWORD
109             value: admin_password
110     volumeMounts:
111         - name: postgres-storage
112             mountPath: /var/lib/postgresql/data
113     volumes:
114         - name: postgres-storage
115             persistentVolumeClaim:
116                 claimName: postgres-pvc

```

*Who's
who & return*

Listado 6.34: Service de PostgreSQL

```

118 apiVersion: v1
119 kind: Service
120 metadata:
121     name: postgres
122 spec:
123     selector:
124         app: postgres
125     ports:
126         - protocol: TCP
127             port: 5432
128             targetPort: 5432
129     clusterIP: None

```

MongoDB

- El Deployment de MongoDB (6.35) configura:
 - Un pod basado en la imagen oficial mongo:8.
 - Montaje de volumen persistente para los datos en /data/db, enlazado a mongo-pvc.
- El Service de MongoDB (6.36) facilita el acceso interno:
 - Expone el puerto estándar 27017.
 - Usa clusterIP: None, como el caso de PostgreSQL.

Listado 6.35: Deployment de MongoDB

```

133 apiVersion: apps/v1
134 kind: Deployment
135 metadata:
136     name: mongo
137 spec:
138     replicas: 1
139     selector:
140         matchLabels:
141             app: mongo
142     template:
143         metadata:
144             labels:
145                 app: mongo
146     spec:
147         containers:
148             - name: mongo
149                 image: mongo:8
150                 ports:
151                     - containerPort: 27017

```

```

152     volumeMounts:
153       - name: mongo-data
154         mountPath: /data/db
155   volumes:
156     - name: mongo-data
157       persistentVolumeClaim:
158         claimName: mongo-pvc

```

*Colección
colección
retener*

Listado 6.36: Service de MongoDB

```

160 apiVersion: v1
161 kind: Service
162 metadata:
163   name: mongo
164 spec:
165   selector:
166     app: mongo
167   ports:
168     - port: 27017
169       targetPort: 27017
170   clusterIP: None

```

Aplicación ElRinconDeEva

- **El Deployment de ElRinconDeEva**, que se detalla en el Listado 6.37, incluye:
 - Una aplicación Spring Boot contenida en la imagen `albherre/elrincondeeva-app:v3`.
 - Variables de entorno para definir las URLs de conexión a servicios como `PostgreSQL`, `publicacionesapi`, `tiempo` y `extracción`.
 - Un contenedor de inicialización (`initContainer`) que espera a que `PostgreSQL` esté disponible antes de iniciar la aplicación principal.
- **El Service de ElRinconDeEva**, que se detalla en el Listado 6.38, permite la comunicación interna y externa:
 - Redirige el tráfico del puerto 80 al 8080 del contenedor.
 - Usa tipo `LoadBalancer` por defecto para comunicar con el exterior por medio del Ingress.

Listado 6.37: Deployment de ElRinconDeEva

```

176 apiVersion: apps/v1
177 kind: Deployment
178 metadata:
179   name: elrincondeeva
180 spec:
181   replicas: 1
182   selector:
183     matchLabels:
184       app: elrincondeeva
185   template:
186     metadata:
187       labels:
188         app: elrincondeeva
189   spec:
190     initContainers:
191       - name: wait-for-postgres
192         image: busybox
193         command:
194           - sh

```

```

195      - -c
196      - "until nc -z -v -w30 postgres 5432; do echo 'Waiting for PostgreSQL...'; sleep 5; done;" 
197  containers:
198    - name: elrincondeeva
199      image: albherre/elrincondeeva-app:v8
200      ports:
201        - containerPort: 8080
202      env:
203        - name: SPRING_DATASOURCE_URL
204          value: r2dbc:postgresql://postgres:5432/gestion
205        - name: FRONTEND_URL
206          value: http://localhost:4200
207        - name: SPRING_DATASOURCE_USERNAME
208          value: admin
209        - name: SPRING_DATASOURCE_PASSWORD
210          value: admin_password
211        - name: PUBLICACIONESAPI_URL
212          value: http://publicacionesapi:8080
213        - name: TIEMPO_URL
214          value: http://tiempo:8080
215        - name: EXTRACCION_URL
216          value: http://extraccion:8080

```

elrinc de eva.

Listado 6.38: Service de ElRinconDeEva

```

219 ---
220
221 apiVersion: v1
222 kind: Service
223 metadata:
224   name: elrincondeeva
225 spec:
226   selector:
227     app: elrincondeeva
228   ports:
229     - port: 80

```

publicacionesapi

- El **Deployment de publicacionesapi**, que se detalla en el Listado 6.39, contiene:
 - Una aplicación Spring Boot con la imagen `albherre/publicacionesapi-app:v2`.
 - Uso de `initContainers` para:
 - Esperar a que `MongoDB` esté disponible.
 - Crear un archivo `token.txt` en un volumen persistente si no existe.
 - Montaje de un volumen persistente desde `token-pvc`.
- El **Service de publicacionesapi**, que se detalla en el Listado 6.40, proporciona acceso interno:
 - Redirige el puerto estándar 8080.

Listado 6.39: Deployment de publicacionesapi

```

233 # publicacionesapi
234 ---
235 apiVersion: apps/v1
236 kind: Deployment
237 metadata:
238   name: publicacionesapi
239 spec:

```

```

240     replicas: 1
241     selector:
242       matchLabels:
243         app: publicacionesapi
244     template:
245       metadata:
246         labels:
247           app: publicacionesapi
248     spec:
249       volumes:
250         - name: token-storage
251           persistentVolumeClaim:
252             claimName: token-pvc # Refiere al PVC para el almacenamiento persistente
253     initContainers:
254
255       - name: wait-for-mongo
256         image: busybox
257         command:
258           - sh
259           - -c
260           - "until nc -z -v -w30 mongo 27017; do echo 'Waiting for MongoDB...'; sleep 5; done;"
261       - name: init-token
262         image: busybox
263         command:
264           - "sh"
265           - "-c"
266           - |
267             if [ ! -f /app/token.txt ]; then
268               echo
269               "IGAAIPJxg21SpBZAE1qYlo1dmNualJ6ZAGRyTEw2eG9qZA1VJcTB3S1pRZAWdCWGNQWmlKamJGeUFyQVQ2dmZA0d2lPMnpGY3YzV3ItM
270             fi
271     volumeMounts:
272       - name: token-storage
273         mountPath: /app # Monta el volumen persistente donde se guardará el token.txt
274
275   containers:
276     - name: publicacionesapi
277       image: albherrre/publicacionesapi-app:v8
278       ports:
279         - containerPort: 8080
280       env:
281         - name: SPRING_DATA_MONGODB_URI
282           value: mongodb://mongo:27017/mediadb
283         - name: INSTAGRAM_TOKEN_FILE_PATH
284           value: /data/
285     volumeMounts:

```

cohorte ciclo retorna.

Listado 6.40: Service de publicacionesapi

```

288
289   ---
290   apiVersion: v1
291   kind: Service
292   metadata:
293     name: publicacionesapi
294   spec:
295     selector:
296       app: publicacionesapi
297     ports:
298       - port: 8080

```

tiempo

- El Deployment de tiempo, que se detalla en el Listado 6.41, define:

- El contenedor de la aplicación meteorológica, expuesto en el puerto 8080.
 - Uso de `initContainers` para esperar a que `MongoDB` esté disponible.
 - Variables de entorno de conexión al servicio `MongoDB` para almacenamiento de datos meteorológicos.
- El **Service de tiempo**, que se detalla en el Listado 6.42, habilita el acceso interno entre pods:
- Enruta tráfico al puerto 8080.

Listado 6.41: Deployment de tiempo

```

302 # tiempo
303 ---
304 apiVersion: apps/v1
305 kind: Deployment
306 metadata:
307   name: tiempo
308 spec:
309   replicas: 1
310   selector:
311     matchLabels:
312       app: tiempo
313   template:
314     metadata:
315       labels:
316         app: tiempo
317   spec:
318     initContainers:
319       - name: wait-for-mongo
320         image: busybox
321         command:
322           - sh
323           - -c
324           - "until nc -z -v -w30 mongo 27017; do echo 'Waiting for MongoDB...'; sleep 5; done;"
325     containers:
326       - name: tiempo
327         image: albherrre/tiempo-app:v8
328         ports:
329           - containerPort: 8080
330         env:

```

MIRE CERCA REFERENCIAS

Listado 6.42: Service de tiempo

```

332           value: mongodb://mongo:27017/mediadb
333   ---
334 apiVersion: v1
335 kind: Service
336 metadata:
337   name: tiempo
338 spec:
339   selector:
340     app: tiempo
341   ports:
342     - port: 8080

```

extraccion

- El **Deployment de extraccion**, que se detalla en el Listado 6.43, contiene:
 - El contenedor de la aplicación que realiza tareas de scraping y transformación de datos.

- Uso de `initContainers` para esperar a que MongoDB esté disponible.
- Variables de entorno para conexión con MongoDB para almacenar resultados extraídos.
- El Service de extraccion, que se detalla en el Listado 6.44, configura su exposición:
 - Utiliza el puerto 8080.

Listado 6.43: Deployment de extraccion

```

346 # extraccion
347 ---
348 apiVersion: apps/v1
349 kind: Deployment
350 metadata:
351   name: extraccion
352 spec:
353   replicas: 1
354   selector:
355     matchLabels:
356       app: extraccion
357   template:
358     metadata:
359       labels:
360         app: extraccion
361     spec:
362       initContainers:
363         - name: wait-for-mongo
364           image: busybox
365           command:
366             - sh
367             - -c
368             - "until nc -z -v -w30 mongo 27017; do echo 'Waiting for MongoDB...'; sleep 5; done;"
369       containers:
370         - name: extraccion
371           image: albherre/extraccion-app:v8
372           ports:
373             - containerPort: 8080
374           env:

```

↗ Muy cerca ya

Listado 6.44: Service de extraccion

```

376   value: mongodb://mongo:27017/mediadb
377 ---
378 apiVersion: v1
379 kind: Service
380 metadata:
381   name: extraccion
382 spec:
383   selector:
384     app: extraccion
385   ports:
386     - port: 8080

```

6.6.3. Ingress Controller

- Se define un recurso de tipo Ingress que enruta el tráfico HTTPS hacia el servicio principal.
- Se utiliza TLS a través de un secret denominado `elrincondeeva-tls`, así se prepara para su despliegue en la nube en un futuro.

→ Referencia!

- La ruta principal redirige al servicio `elrincondeeva`.

Listado 6.45: Recurso Ingress para el acceso HTTPS

```

390
391 # 5. Ingress Controller (nginx ingresado previamente)
392 ---
393 apiVersion: networking.k8s.io/v1
394 kind: Ingress
395 metadata:
396   name: elrincondeeva
397   annotations:
398     nginx.ingress.kubernetes.io/ssl-redirect: "true"
399 spec:
400   tls:
401     - hosts:
402       - elrincondeeva.local
403       secretName: elrincondeeva-tls
404   rules:
405     - host: elrincondeeva.local
406       http:
407         paths:
408           - path: /
409             pathType: Prefix
410             backend:
411               service:

```

Una vez desplegada la infraestructura se comprueba graficamente mediante el panel de control de [Kubernetes](#) que todos los pods se encuentran en estado `Running` y que los servicios están correctamente configurados. En la Figura 6.11 se muestra una captura de pantalla del panel de control de [Kubernetes](#) donde se puede observar el estado de los pods y servicios desplegados.

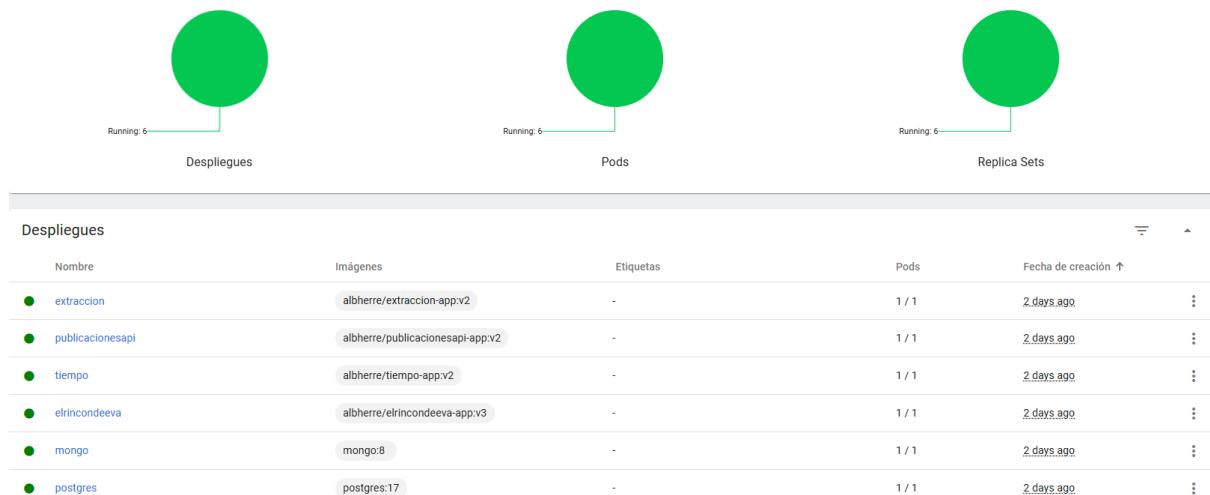


Figura 6.11: Panel de control de Kubernetes mostrando el estado de los pods y servicios

6.7. Pruebas

6.7.1. Pruebas unitarias de la aplicación web

Estas pruebas se han realizado exclusivamente en la parte del [Backend](#) de la aplicación, se ha utilizado Postman [39] para realizar llamadas a cada uno de los endpoints generados y validar su funcionamiento y se ha utilizado las herramientas JUnit [37] y Mockito [38] para la ejecución de pruebas unitarias sobre algunos de los servicios más relevantes y cuya lógica es mas compleja dentro de la aplicación. A continuación se describen las pruebas más relevantes realizadas en el [Backend](#) de la aplicación:

- **Pruebas de servicios de reservas:** Se han creado pruebas para verificar que los servicios de reservas funcionan correctamente, incluyendo la creación y modificación de reservas. En el Apéndice A.9, Listado A.38, se puede consultar el código fuente de las pruebas unitarias realizadas en el servicio de reservas.

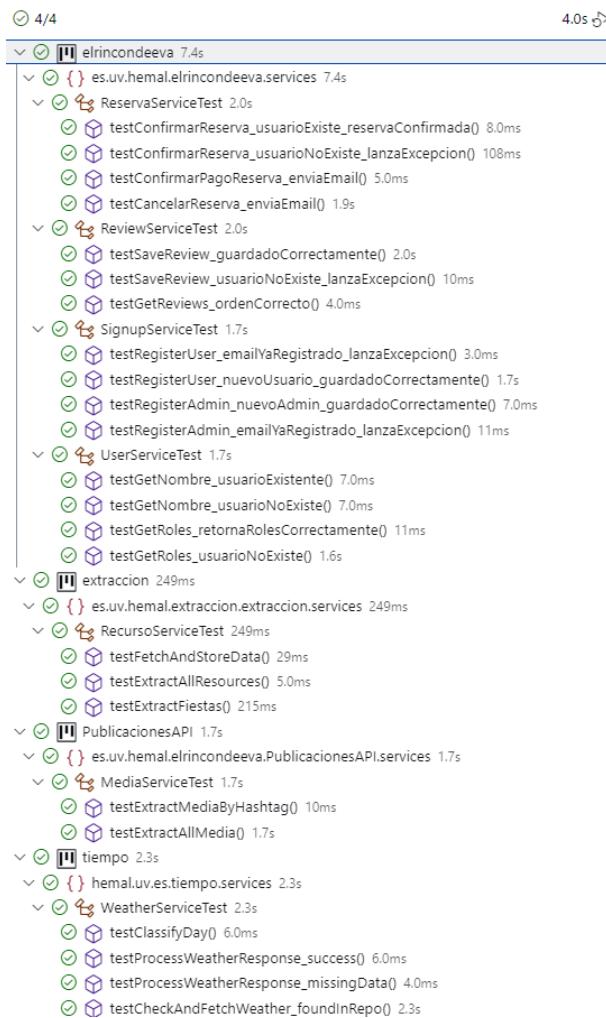
- **testConfirmarReserva_usuarioExiste_reservaConfirmada**
 - Simula el caso en que el usuario ya existe en el sistema.
 - Crea una reserva y la guarda correctamente.
 - Verifica que se envía un correo de confirmación.
 - Comprueba que el ID devuelto por la reserva es el esperado.
- **testConfirmarReserva_usuarioNoExiste_lanzaExpcion**
 - Simula el caso en que el usuario no existe en el sistema.
 - Verifica que se lanza una excepción con el mensaje “Usuario no encontrado”.
- **testConfirmarPagoReserva_enviaEmail**
 - Simula la confirmación del pago de una reserva existente.
 - Verifica que se recupera correctamente la reserva por ID.
 - Comprueba que se envía un correo de confirmación del pago.
- **testCancelarReserva_enviaEmail**
 - Simula la cancelación de una reserva existente.
 - Verifica que se recupera correctamente la reserva por ID.
 - Comprueba que se envía un correo de cancelación con el motivo indicado.
- **Pruebas de servicios de usuarios:** Se han diseñado pruebas para verificar que los servicios de gestión de usuarios funcionan correctamente, incluyendo el registro, inicio de sesión y recuperación de contraseñas. En el Apéndice A.9, Listado A.40 y Listado A.41 se puede consultar el código fuente de las pruebas unitarias realizadas en el servicio de usuario.
 - **testGetNombre_usuarioExistente**
 - Simula la existencia de un usuario con un correo electrónico dado.
 - Verifica que se devuelve correctamente el objeto MyUser correspondiente.
 - Comprueba que el email del usuario es el esperado.
 - **testGetNombre_usuarioNoExiste**
 - Simula la ausencia de un usuario con el correo electrónico indicado.

- Verifica que se devuelve `null`.
- **testGetRoles _ retornaRolesCorrectamente**
 - Simula un usuario con dos roles asociados.
 - Verifica que se recuperan correctamente los roles del usuario mediante sus identificadores.
 - Comprueba que se devuelven los nombres de roles esperados: `ROLE_USER` y `ROLE_ADMIN`.
- **testGetRoles _ usuarioNoExiste**
 - Simula que el usuario no existe en la base de datos.
 - Verifica que la lista de roles devuelta es vacía.
- **testRegisterUser _ emailYaRegistrado _ lanzaExpcion**
 - Simula que el correo electrónico ya está registrado.
 - Verifica que se lanza una excepción con un mensaje indicando que el email ya está en uso.
- **testRegisterUser _ nuevoUsuario _ guardadoCorrectamente**
 - Simula el registro de un nuevo usuario con datos válidos.
 - Verifica que se encripta la contraseña, se guarda el usuario y se le asigna el rol `ROLE_USER`.
 - Confirma que el resultado no es nulo.
- **testRegisterAdmin _ nuevoAdmin _ guardadoCorrectamente**
 - Simula el registro de un nuevo administrador.
 - Verifica que se guarda el usuario y se le asigna el rol `ROLE_ADMIN`.
 - Comprueba que el proceso finaliza correctamente.
- **testRegisterAdmin _ emailYaRegistrado _ lanzaExpcion**
 - Simula que el correo electrónico del administrador ya está registrado.
 - Verifica que se lanza una excepción con el mensaje correspondiente.
- **Pruebas de servicios de reseñas:** Se han creado pruebas para verificar que los servicios de gestión de reseñas funcionan correctamente, incluyendo la creación y recuperación de reseñas. En el Apéndice A.9, Listado A.39 se puede consultar el código fuente de las pruebas unitarias realizadas en el servicio de reseñas.
 - **testSaveReview _ guardadoCorrectamente**
 - Simula que un usuario autenticado con rol `ROLE_CLIENTE` guarda una reseña.
 - Verifica que la reseña se guarda correctamente en el repositorio.
 - Comprueba que se asigna correctamente el ID del usuario a la reseña.
 - **testSaveReview _ usuarioNoExiste _ lanzaExpcion**
 - Simula que el usuario autenticado no existe en la base de datos.
 - Verifica que se lanza una excepción con el mensaje `User not found`.
 - **testGetReviews _ ordenCorrecto**
 - Simula la recuperación de reseñas ordenadas por fecha de creación descendente.

- Verifica que las reseñas se devuelven en el orden correcto.
- **Pruebas de servicios de API publicaciones:** Se han creado pruebas para verificar que los servicios de gestión de publicaciones funcionan correctamente, incluyendo la recuperación de datos desde Instagram y la gestión de los mismos. En el Apéndice A.9, Listado A.42 se puede consultar el código fuente de las pruebas unitarias realizadas en el servicio de publicaciones.
 - **testExtractMediaByHashtag**
 - Simula la búsqueda de contenido multimedia por un hashtag específico ("playa").
 - Verifica que se devuelve correctamente el contenido asociado al hashtag.
 - **testExtractAllMedia**
 - Simula la recuperación de todos los elementos multimedia de la base de datos.
 - Verifica que se devuelven correctamente todos los resultados esperados.
- **Pruebas de servicios de API meteorología:** Se han diseñado pruebas para verificar que los servicios de gestión de datos meteorológicos funcionan correctamente, incluyendo la recuperación de datos desde la API de OpenWeather y la gestión de los mismos. En el Apéndice A.9, Listado A.43 se puede consultar el código fuente de las pruebas unitarias realizadas en el servicio de meteorología.
 - **testClassifyDay**
 - Verifica la clasificación del día según la temperatura y la precipitación.
 - Prueba los posibles resultados: "Caluroso", "Frío", "Normal" y "Lluvioso".
 - **testProcessWeatherResponse_success**
 - Simula una respuesta de datos meteorológicos válida.
 - Verifica que se procesa correctamente y se clasifica como "Normal".
 - **testProcessWeatherResponse_missingData**
 - Simula una respuesta vacía del servicio meteorológico.
 - Verifica que se lanza una excepción con el mensaje "No se han encontrado datos".
 - **testCheckAndFetchWeather_foundInRepo**
 - Comprueba si la información meteorológica para una fecha ya está almacenada.
 - Verifica que se recupera del repositorio sin hacer nuevas peticiones externas.
- **Pruebas de servicios de API recursos:** Se han creado pruebas para verificar que los servicios de gestión de datos del entorno funcionan correctamente, incluyendo la recuperación de datos desde la API de OpenStreetMap y la gestión de los mismos. En el Apéndice A.9, Listado A.44 se puede consultar el código fuente de las pruebas unitarias realizadas en el servicio de entorno.
 - **testFetchAndStoreData**
 - Simula el comportamiento de `scraperService.scrape` y `recursoRepository.findAll()`.

- Verifica que se hacen las llamadas esperadas al `scraperService` con URLs y categorías correctas.
- **testExtractAllResources**
 - Simula la devolución de un recurso desde el repositorio.
 - Verifica que se llama a `findAll()` correctamente.
- **testExtractFiestas**
 - Simula la devolución de un recurso con la categoría `fiestas`.
 - Verifica que se llama a `findByCategoria("fiestas")` correctamente.

En la Figura 6.12 se muestra la ejecución de todas las pruebas unitarias realizadas en el `Backend` de la aplicación. Comprobando que todas las pruebas ejecutadas han pasado correctamente y no se han producido errores en la ejecución de las mismas.



Te ha quedado
genial pt
el código est'
en apéndice

Figura 6.12: Ejecución de las pruebas unitarias realizadas en el `Backend` de la aplicación

6.7.2. Pruebas funcionales de la aplicación web

Las pruebas funcionales de la aplicación web se han realizado utilizando la herramienta Cypress [40], que permite realizar pruebas E2E en aplicaciones web. Estas pruebas se han diseñado para verificar el correcto funcionamiento de las funcionalidades más relevantes de

la aplicación, asegurando que los usuarios puedan interactuar con ella de manera efectiva y sin errores. Las pruebas se han estructurado en diferentes archivos, cada uno de los cuales se centra en un aspecto específico de la aplicación. A continuación, se describen las pruebas más relevantes realizadas con Cypress:

- **Pruebas de inicio de sesión y registro:** Se han creado pruebas para verificar que los usuarios pueden iniciar sesión correctamente con credenciales válidas y que el sistema maneja adecuadamente los errores de inicio de sesión. También se han probado los flujos de registro, asegurando que los nuevos usuarios pueden registrarse sin problemas. En el Apéndice A.9, Listado A.46 y Listado A.47 se pueden consultar los códigos fuente de las pruebas funcionales realizadas en el inicio de sesión y registro. En estas pruebas se ha validado lo siguiente:
 - **Debería iniciar sesión con credenciales válidas:** Verifica que, al ingresar un correo y una contraseña válidos, el usuario es redirigido a la página de inicio (/inicio).
 - **Debería mostrar error si los datos son inválidos:** Verifica que, si se ingresan credenciales inválidas, el sistema muestra un mensaje de error indicando que el usuario o la contraseña no existen.

El resultado de estas pruebas para el Login y el Registro se muestran en la Figura 6.13 y la Figura 6.15 respectivamente.

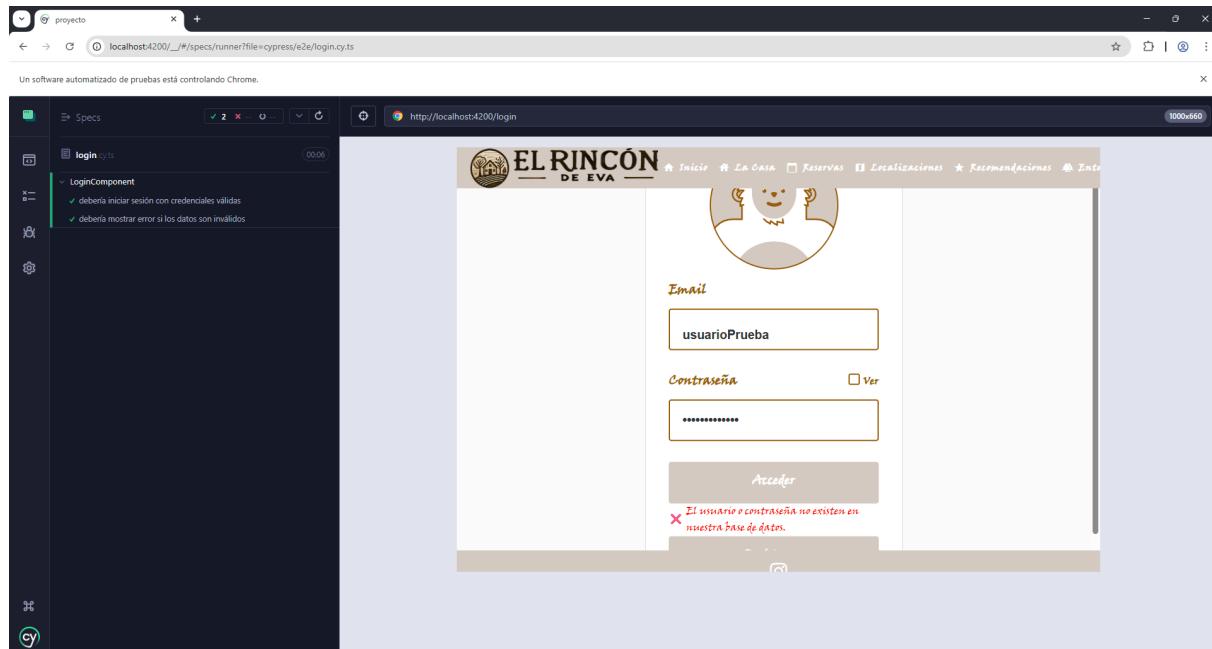


Figura 6.13: Ejecución de las pruebas de inicio de sesión

- **Pruebas de reservas:** Se han diseñado pruebas para verificar que los usuarios pueden realizar reservas correctamente, incluyendo la selección de fechas, la introducción de datos personales y la confirmación de la reserva. También se han probado los flujos de cancelación y modificación de reservas. En el Apéndice A.9, Listado A.48 se puede consultar el código fuente de las pruebas funcionales realizadas en la gestión de reservas. En estas pruebas se ha validado lo siguiente:

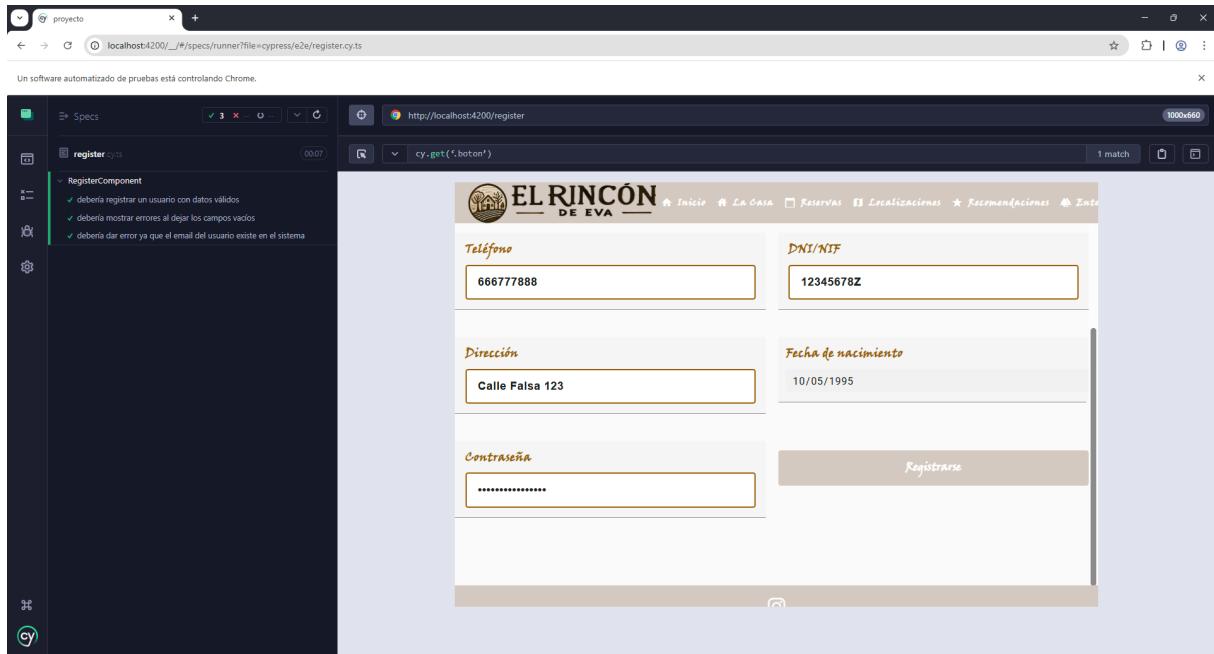


Figura 6.14: Ejecución de las pruebas de registro

- **Debe mostrar un botón para iniciar sesión cuando el email está vacío:** Verifica que se muestre un botón de inicio de sesión cuando el campo de email esté vacío.
- **Debe mostrar el calendario de reservas cuando el email está presente:** Verifica que se muestra el calendario de reservas al iniciar sesión correctamente.
- **Debe mostrar el tiempo para las fechas seleccionadas:** Verifica que se muestre el tiempo para las fechas seleccionadas en el calendario.
- **Debe mostrar el formulario para completar la reserva rellenado desde el backend:** Verifica que el formulario de reserva esté correctamente llenado con datos provenientes del backend.

El resultado de estas pruebas se muestra en la Figura 6.15.

- **Pruebas de gestión de reservas:** Se han diseñado pruebas para verificar que el administrador puede realizar gestiones sobre las reservas probando los flujos de cancelación, aprobación de reservas. En el Apéndice A.9, Listado A.45 se puede consultar el código fuente de las pruebas funcionales realizadas en la gestión de reservas. En estas pruebas se ha validado lo siguiente:

- **Debería mostrar la tabla de reservas con encabezados correctos:** Verifica que la tabla de reservas es visible y que los encabezados incluyen: 'Nombre', 'Número Personas', 'Estado', 'Fechas', 'Acciones'.
- **Debería tener al menos una fila de datos:** Asegura que la tabla contiene al menos una fila con datos.
- **Debería mostrar los botones adecuados según el estado de la reserva:** Verifica que los botones de la tabla cambian dependiendo del estado de la reserva:
 - Si el estado es PAGADA o FINALIZADA, el botón debería mostrar el icono check_circle.

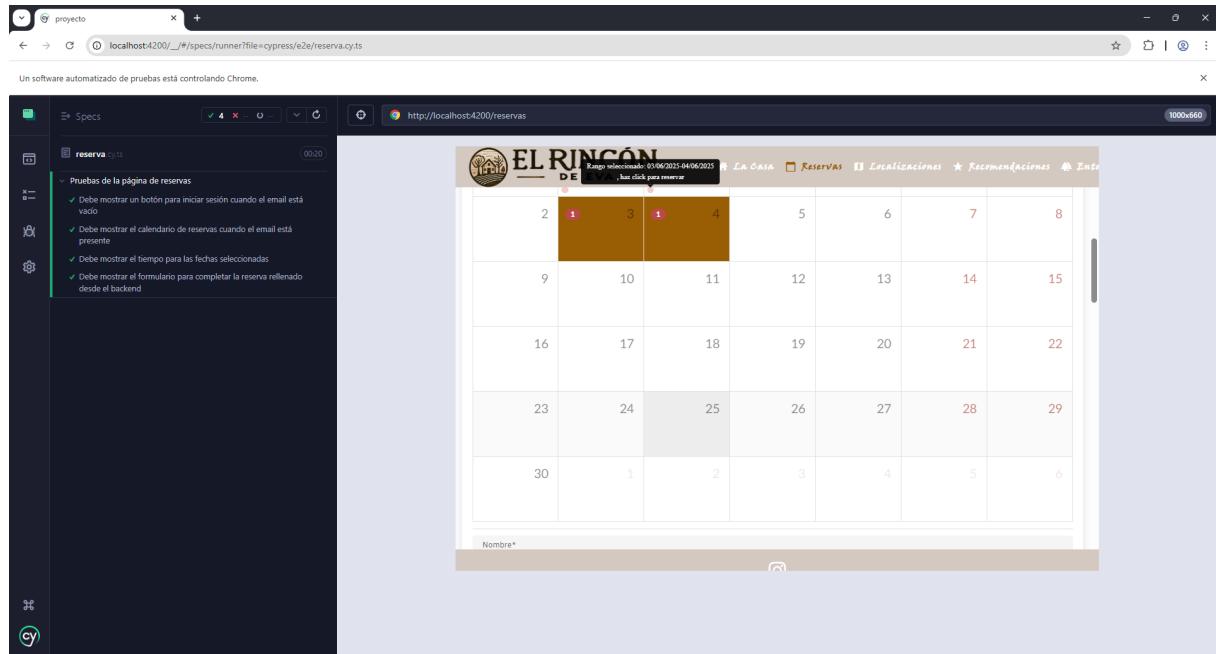


Figura 6.15: Ejecución de las pruebas de reservas

- Si el estado es CANCELADA, el botón debería mostrar el icono error.
- En otros estados, debería haber botones con iconos check y close.
- Debería aprobar una reserva CONFIRMADA, llenar el motivo y confirmar:
 - Hace clic en el botón de una reserva con estado CONFIRMADA.
 - Espera a que se abra el diálogo, escribe el motivo en el textarea y hace clic en Aceptar.
 - Verifica que se muestre un mensaje de éxito: Reserva confirmada con éxito!.

El resultado de estas pruebas se muestra en la Figura 6.16.

Con estas pruebas y las realizadas durante las pruebas unitarias se han validado cada uno de los casos de uso definidos durante la sección de Análisis 4.1.1. En la Tabla 6.1 se muestran los casos de uso y las pruebas realizadas para cada uno de ellos.

Tabla 6.1: Pruebas de casos de uso

Caso de uso	Prerrequisitos	Prueba	Resultado esperado
CU-1 - Login	El cliente debe existir en el sistema.	Rellenar formulario de login y enviarlo. Validación en cliente y servidor.	El usuario queda autenticado y puede acceder a la sección de reservas al completo para poder realizar una nueva reserva.

Continúa en la siguiente página

Caso de uso	Prerrequisitos	Prueba	Resultado esperado
CU-2 Logout	Debe de existir el cliente en el sistema. El cliente debe estar logueado.	Cerrar sesión como cliente y verificar que el estado de sesión se elimina.	El usuario volverá a ser anónimo. Se eliminará cualquier estado de sesión almacenada. Se limitará la posibilidad de realizar una reserva.
CU-3 Registrarse	No debe de existir el email del cliente en el sistema.	Rellenar el formulario de registro con datos válidos y enviarlo.	El usuario estará registrado en el sistema. Se añadirá un nuevo estado de sesión almacenada. Se redirigirá al usuario al menú principal.
CU-4 Obtener información de la casa	El servidor web se encuentre en funcionamiento.	Acceder a la pestaña “La casa” y verificar que se muestran los datos y contenido multimedia.	El usuario visualizará los datos y el contenido multimedia relacionado con la casa rural.
CU-5 Contactar con gestor	El servidor web se encuentre en funcionamiento.	Rellenar el formulario de contacto con datos válidos y enviarlo.	En el correo del administrador se debe visualizar un nuevo correo.
CU-6 Obtener localización	El servidor web se encuentre en funcionamiento.	Acceder a la pestaña “Dónde estamos” y verificar que se muestra un mapa interactivo.	Se debe visualizar un mapa interactivo situado sobre el inmueble.
CU-7 Obtener fiestas y gastronomía	El servidor web se encuentre en funcionamiento.	Acceder a la pestaña “Recomendaciones” y verificar que se muestra contenido textual y multimedia.	Se debe visualizar contenido tanto estático como extraído de las redes sociales.
CU-8 Obtener recomendaciones turísticas	El servidor web se encuentre en funcionamiento.	Acceder a la pestaña “Entorno” y verificar que se muestra contenido multimedia con descripciones.	Se debe visualizar contenido tanto estático como extraído de las redes sociales. Si se hace clic sobre alguna, se debe redirigir al sistema origen.
CU-9 Obtener reseñas	El servidor web se encuentre en funcionamiento. Debe haber al menos una reseña almacenada en el servidor.	Acceder a la pestaña “Inicio” y verificar que se muestra una lista de reseñas.	Se debe visualizar una lista de reseñas que incluirán una puntuación final sobre la estancia.

Continúa en la siguiente página

Caso de uso	Prerrequisitos	Prueba	Resultado esperado
CU-10 Obtener actividades	El servidor web se encuentre en funcionamiento. La API que proporciona las actividades debe estar en funcionamiento.	Acceder a la pestaña “Entorno” y verificar que se muestran actividades con descripción y dificultad.	Se debe visualizar un conjunto de actividades con descripción y dificultad. Si se hace clic sobre alguna, se debe redirigir al sistema origen.
CU-11 Obtener fechas disponibles	El servidor web se encuentre en funcionamiento. Solo se mostrarán fechas a futuro. Las fechas no disponibles se marcarán con un color diferente.	Acceder a la pestaña “Reservas” y verificar que se muestra un calendario interactivo con las fechas disponibles.	Se debe visualizar un calendario interactivo con las fechas disponibles y las ocupadas.
CU-12 Reservar fechas	Se deben cumplir las condiciones y el flujo del CU11. El servidor web se encuentre en funcionamiento. Solo se podrán seleccionar fechas a futuro. Las fechas no disponibles no se podrán seleccionar. Deberá existir una base de datos que almacene las reservas.	Seleccionar un conjunto de fechas disponibles, formalizar la reserva y verificar que se registra correctamente.	Se debe visualizar un mensaje de éxito de creación de la reserva. Se debe mostrar como reserva pendiente de confirmación en el perfil del cliente.
CU-13 Obtener clima	Se deben cumplir las condiciones y el flujo del CU11. El servidor web se encuentre en funcionamiento. El servicio externo de tiempo debe estar disponible.	Seleccionar un conjunto de fechas y verificar que se muestra el pronóstico del tiempo.	Se debe visualizar una sección con el pronóstico de tiempo para cada uno de los días seleccionados.
CU-14 Obtener tarifas	El servidor web se encuentre en funcionamiento.	Seleccionar fechas en la pestaña “Reservas” y verificar que se muestran las tarifas asociadas.	Se debe visualizar las tarifas asociadas a las fechas seleccionadas.

Continúa en la siguiente página

Caso de uso	Prerrequisitos	Prueba	Resultado esperado
CU-15 Dejar reseña	El servidor web se encuentre en funcionamiento. El cliente registrado debe tener asignado el rol de cliente.	Rellenar el formulario de nueva reseña con datos válidos y enviarlo.	Se debe visualizar un mensaje de confirmación de la reseña. La nueva reseña debe aparecer en la lista de reseñas.
CU-16 Consultar reservas pendientes	El servidor web se encuentre en funcionamiento. Debe haber al menos una reserva pendiente en el sistema. Debe estar logueado como administrador.	Acceder a la pestaña “Gestión” y verificar que se muestra una lista de reservas.	Se debe visualizar una lista de reservas pendientes de confirmación.
CU-17 Confirmar reserva	El servidor web se encuentre en funcionamiento. Debe haber al menos una reserva pendiente en el sistema. Debe estar logueado como administrador.	Seleccionar una reserva pendiente y aprobarla.	La reserva seleccionada cambiará su estado a aprobada. El cliente recibirá una notificación de aprobación.
CU-18 Denegar reserva	El servidor web se encuentre en funcionamiento. Debe haber al menos una reserva pendiente en el sistema. Debe estar logueado como administrador.	Seleccionar una reserva pendiente y rechazarla.	La reserva seleccionada cambiará su estado a rechazada. El cliente recibirá una notificación de rechazo.
CU-19 Raspar datos web	Las aplicaciones web de los ayuntamientos deben estar disponibles.	Ejecutar el proceso de extracción de datos y verificar que se almacenan correctamente.	Los datos extraídos se almacenarán y se mostrarán en la aplicación web.
CU-20 Extraer datos actividades	Las APIs de actividades al aire libre deben estar disponibles.	Ejecutar el proceso de extracción de datos y verificar que se almacenan correctamente.	Los datos extraídos se almacenarán y se mostrarán en la aplicación web.
CU-21 Extraer datos de redes sociales	Las APIs de las redes sociales deben estar disponibles.	Ejecutar el proceso de extracción de datos y verificar que se almacenan correctamente.	Los datos extraídos se almacenarán y se mostrarán en la aplicación web.

Tabla 6.1: Pruebas de casos de uso

1 Te ha quedado genial!

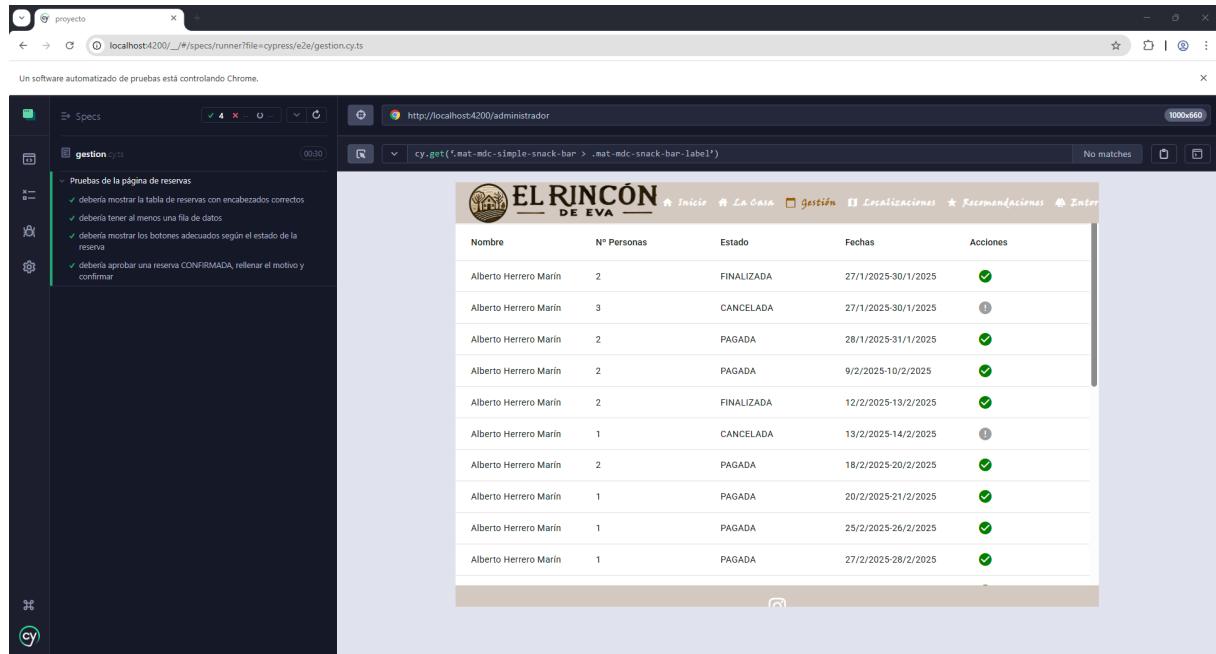


Figura 6.16: Ejecución de las pruebas de gestión de reservas

6.8. Pruebas de seguridad

Para garantizar la seguridad de la aplicación web desarrollada, se han llevado a cabo pruebas de seguridad utilizando la herramienta OWASP ZAP [42], reconocida por su capacidad para detectar vulnerabilidades comunes en entornos web como Cross-Site Scripting (XSS), Inyección de SQL (SQL Injection), problemas de cabeceras HTTP, entre otras. La evaluación se ha realizado sobre dos vectores principales: el Backend de la aplicación, a través de su especificación OpenAPI generada mediante Swagger [60], y el Frontend, accediendo a la URL dónde se encuentra el servidor de Angular en funcionamiento.

6.8.1. Configuración del entorno de pruebas

La herramienta OWASP ZAP se ha ejecutado en modo GUI, atacando dos URLs principales:

- localhost:8089 correspondiente al frontend desplegado en un servidor embebido.
- localhost:8080/api/v1 endpoint principal del Backend.

Para el análisis del Backend, se ha importado el archivo de especificación OpenAPI, `api-docs.yaml` generado automáticamente por la extensión Swagger. Esto ha permitido una exploración exhaustiva y estructurada de los endpoints.

6.8.2. Pruebas de seguridad del backend

Análisis inicial

El análisis inicial realizado con OWASP ZAP sobre el Backend identificó dos alertas relevantes, tal y como mostró la figura 6.17:

- **Format String Error:** Esta alerta apareció en el endpoint POST /api/v1/contact. Indica que hay una posible vulnerabilidad relacionada con el tratamiento inadecuado de cadenas, potencialmente explotable para provocar errores o comportamientos no deseados si no se validan o escapan adecuadamente los datos introducidos por el usuario.
- **Inyección XSLT:** Sugiere que ciertos parámetros pueden ser susceptibles a técnicas de inyección de transformaciones XML (XSLT). Esto representa un riesgo cuando se procesan entradas eXtensible Markup Language (XML) sin validación o filtrado.

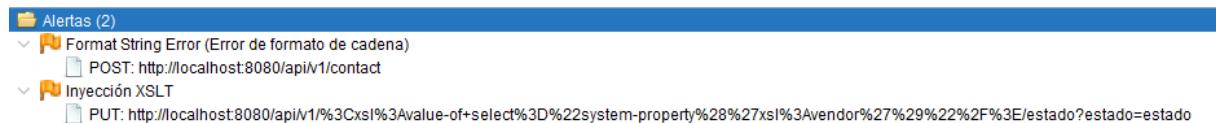


Figura 6.17: Alertas iniciales detectadas en el backend por OWASP ZAP.

Medidas aplicadas

Para solventar las vulnerabilidades detectadas se aplicaron las siguientes medidas:

- Se configuraron cabeceras de seguridad en el `WebSecurityConfigurerAdapter`.
- Se validaron y sanearon los campos de entrada en formularios, especialmente en el `ContactService`.

Resultado final

Tras aplicar las mejoras mencionadas, se repitieron las pruebas con OWASP ZAP y ya no se observó ninguna alerta.

6.8.3. Pruebas de seguridad del frontend

Análisis inicial

Durante el análisis inicial del frontend se detectaron varias alertas de seguridad que ponían en riesgo la integridad de la aplicación web, ~~entre ellas~~ tal y como pone la frase 6/8:

- Falta de definición de la directiva `default-src` en la política Content Security Policy (CSP).
- Ausencia total o configuración incorrecta de la política Content-Security-Policy.
- Falta de cabecera `X-Frame-Options` para prevenir ataques de clickjacking.
- Divulgación de información mediante la cabecera HTTP `X-Powered-By`.
- Falta de la cabecera `X-Content-Type-Options`.
- Inclusión de archivos JavaScript externos de dominios cruzados.
- Comentarios HTML con información sensible en el código fuente.

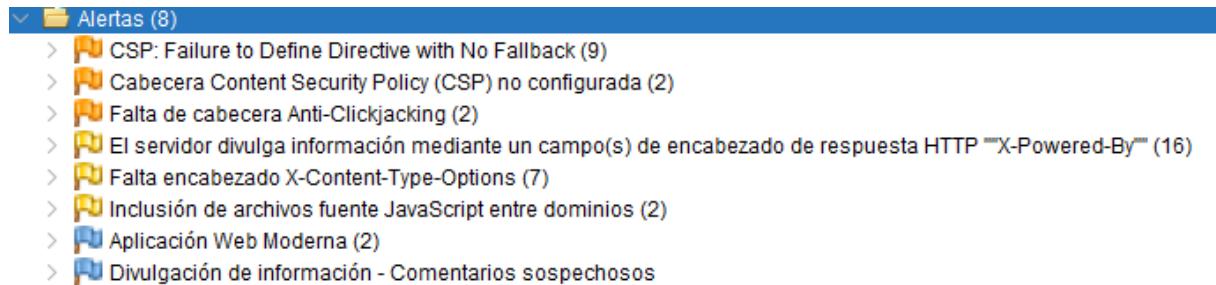


Figura 6.18: Resultado inicial de las pruebas de seguridad del frontend.

Medidas aplicadas

- Se implementó una cabecera **Content-Security-Policy** que limita las fuentes de scripts, estilos e imágenes a dominios de confianza.
- Se añadió la cabecera **X-Frame-Options: DENY** para evitar la incrustación de la aplicación en iframes.
- Se configuró la cabecera **X-Content-Type-Options: nosniff** para prevenir ataques de tipo MIME sniffing.
- Se eliminó la cabecera **X-Powered-By** del backend para evitar divulgación innecesaria.
- Se revisaron y limpiaron comentarios sospechosos o informativos en el HTML de las plantillas.
- Se deshabilitó la carga de archivos JS desde dominios cruzados no controlados.

Resultado final

El segundo análisis evidenció una reducción significativa de las vulnerabilidades detectadas. En la Figura 6.19 se observa una situación mucho más controlada y segura. Solo quedarán dos alertas inevitables dado que Angular inyecta los estilos en formato **inline** y se ha realizado la configuración de **CSP** más restrictiva posible.

6.9. Pruebas de rendimiento

Para evaluar el rendimiento de la aplicación web en condiciones de carga, se ha utilizado Apache JMeter [41], una herramienta ampliamente reconocida para pruebas de carga y estrés. Estas pruebas permiten identificar cuellos de botella, tiempos de respuesta y comportamiento ante múltiples usuarios concurrentes.

6.9.1. Configuración del entorno de pruebas

Se ha elaborado un plan de pruebas, representado mediante un archivo **.jmx**, con el objetivo de simular múltiples usuarios concurrentes interactuando con los principales

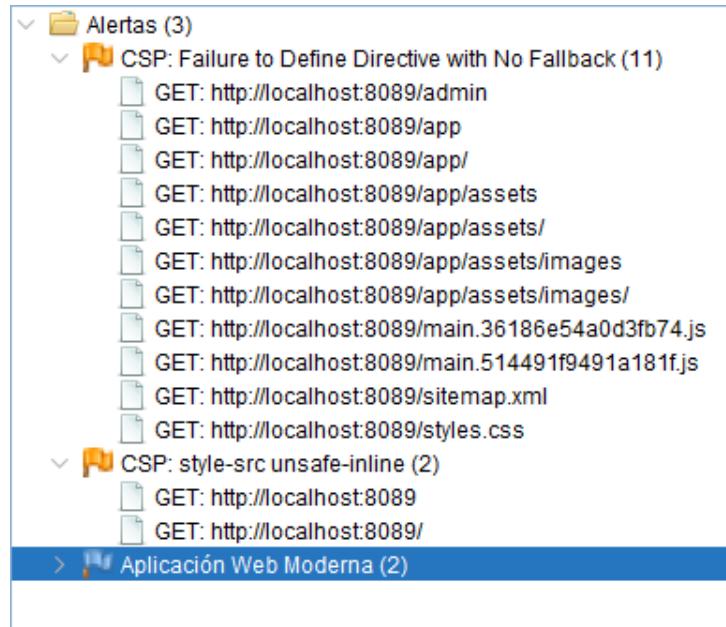


Figura 6.19: Resultado final de las pruebas de seguridad del frontend.

servicios expuestos por la **API (Application Programming Interface)** REST del **Backend**. Con el fin de reproducir un entorno lo más cercano posible al de producción, dichas pruebas se han ejecutado sobre el clúster de **Kubernetes** previamente desplegado, al cual se accede a través de un túnel seguro establecido desde el servidor local que aloja la aplicación web del **Frontend**. Para estas pruebas tanto el **Frontend** como el cluster estarán cifrados mediante protocolo **Secure Sockets Layer (SSL)**, asemejándose así a un escenario real. Los escenarios contemplados en este plan de pruebas son los siguientes:

- Inicio de sesión de usuarios.
- Realización de reservas.
- Consulta de recomendaciones, actividades y entorno.

Para cada escenario se ha considerado una prueba de carga diferente. Ya que hay llamadas a **APIs** externas que pueden denegar el acceso si se realizan demasiadas llamadas. Además, por funcionalidad de aplicación hay pruebas de rendimiento que se realizarán siendo realistas con la finalidad de la plataforma.

Por lo tanto, para el escenario en que se prueba un gran número de accesos concurrentes, el entorno se configuró con:

- 100 usuarios virtuales concurrentes.
- Ramp-up de 1 segundos. Ya que no sale de la aplicación y se puede medir el rendimiento sin riesgo a que **APIs** externas denieguen el acceso.
- Duración de prueba: 5 iteraciones.

Para el escenario de generación de reservas, se configuró el entorno con:

- 10 usuarios virtuales concurrentes.

- Ramp-up de 3 segundos. Ya que se quiere probar el caso en que varios usuarios realizan reservas al mismo tiempo, pero no se quiere saturar el sistema.
- Duración de prueba: 5 iteraciones.

Por último, para el escenario de consulta de recomendaciones, actividades y entorno, se configuró el entorno con:

- 100 usuarios virtuales concurrentes.
- Ramp-up de 1 segundos. Ya que no se realizan cambios en la base de datos y se pueden realizar muchas consultas sin riesgo a que APIs externas denieguen el acceso.
- Duración de prueba: 5 iteraciones.

Se utilizó un Thread Group para cada uno de los escenarios, permitiendo así una ejecución independiente y controlada de cada prueba. Se utilizó un HTTP(S) Test Script Recorder para capturar las peticiones HTTPS realizadas por el navegador al interactuar con la aplicación web. Esto permite simular de manera precisa las acciones de los usuarios reales. En la Figura 6.20 se muestra la configuración del entorno de pruebas en JMeter.

6.9.2. Resultados obtenidos

Durante la ejecución de las pruebas de carga se ha medido la latencia media por cada endpoint, extrayendo estos datos del Reporte de Resumen generado de cada Thread Group.

La Figura 6.21 representa los resultados obtenidos durante el proceso de inicio de sesión. En ella se observa un comportamiento eficiente con tiempos de respuesta bajos, lo que indica que las operaciones de autenticación están bien optimizadas y no dependen de servicios externos.

En la Figura 6.22, correspondiente al proceso de gestión de reservas, se identifican endpoints con una mayor latencia. Esta diferencia se debe principalmente a que estas operaciones requieren lógica adicional, como la comprobación de fechas disponibles o el envío de correos electrónicos, lo que implica llamadas a APIs externas.

Por último, la Figura 6.23 muestra la latencia media durante la carga de vistas y elementos del frontend. De nuevo, los endpoints con mayor tiempo de respuesta son aquellos que consultan servicios externos, como la obtención de datos turísticos o recomendaciones personalizadas.

Los resultados obtenidos evidencian que algunos endpoints presentan una latencia significativamente mayor debido a su dependencia de servicios externos. Como trabajo futuro, se propone analizar estos casos en profundidad y aplicar mecanismos de optimización, como la implementación de caché, reducción de llamadas redundantes o estrategias de precarga, con el objetivo de mejorar el rendimiento global de la aplicación.

Podrás arrancar pronto de acceso con la otra página

6.10. Pruebas de accesibilidad y posicionamiento

Con el fin de evaluar la calidad técnica de la aplicación web en aspectos de accesibilidad, posicionamiento (Search Engine Optimization (SEO)), y rendimiento general, se ha

en inglés

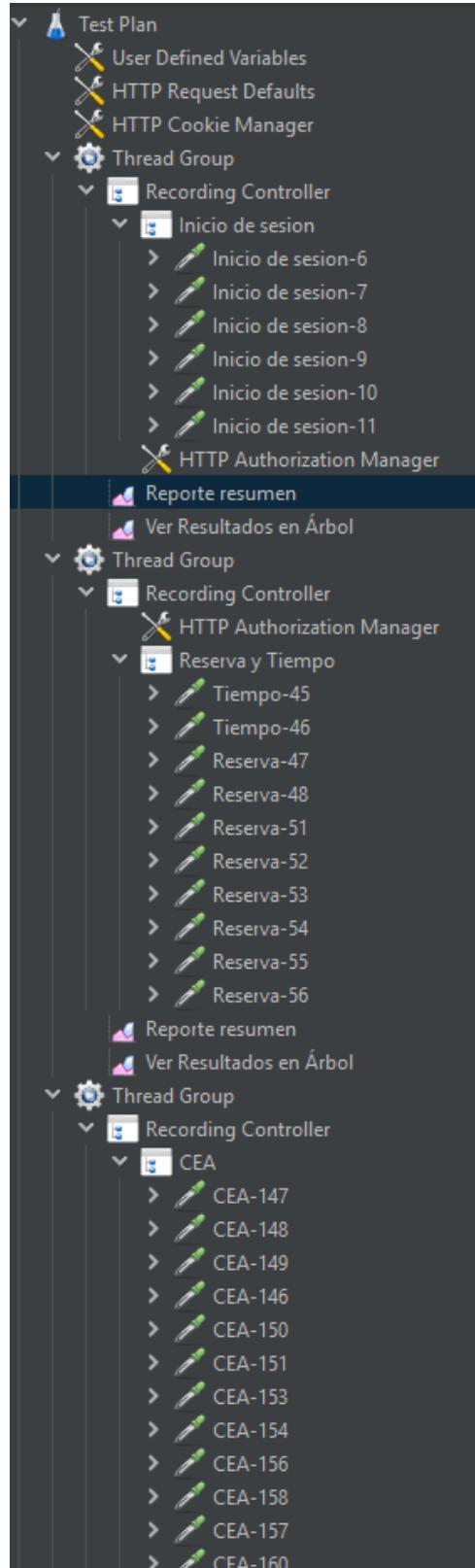


Figura 6.20: Configuración del entorno de pruebas en JMeter

ejecutado Lighthouse [61] sobre las principales páginas del sitio. A continuación, se resumen los resultados obtenidos, acompañados de los gráficos generados automáticamente por la herramienta.

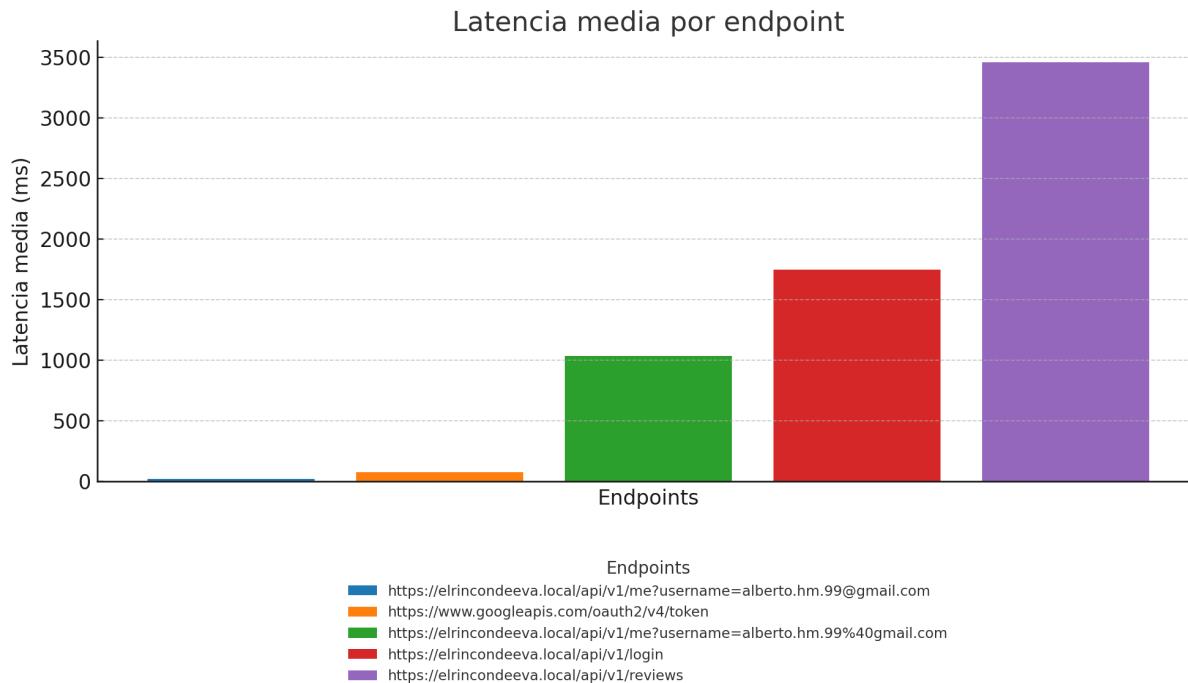


Figura 6.21: Latencia media por endpoint — datos de `inicio1.csv`

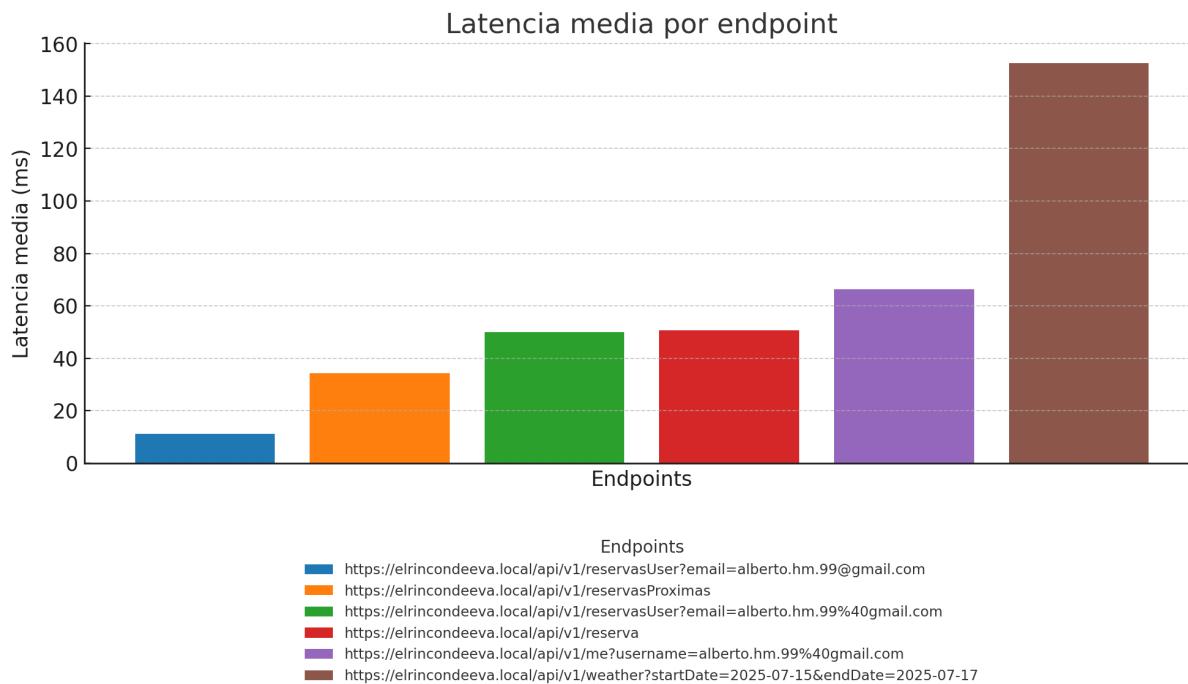


Figura 6.22: Latencia media por endpoint — datos de `reserva1.csv`

De Naran que operando así
Página de inicio

La página de inicio obtiene una puntuación destacada en SEO (100) y buenas prácticas (96), mientras que la accesibilidad se sitúa en un 86. Se detectan algunos problemas menores en botones y enlaces que carecen de etiquetas accesibles, así como contrastes de color mejorables. El rendimiento es aceptable (74), aunque se puede mejorar la carga inicial mediante compresión y reducción de JavaScript innecesario.

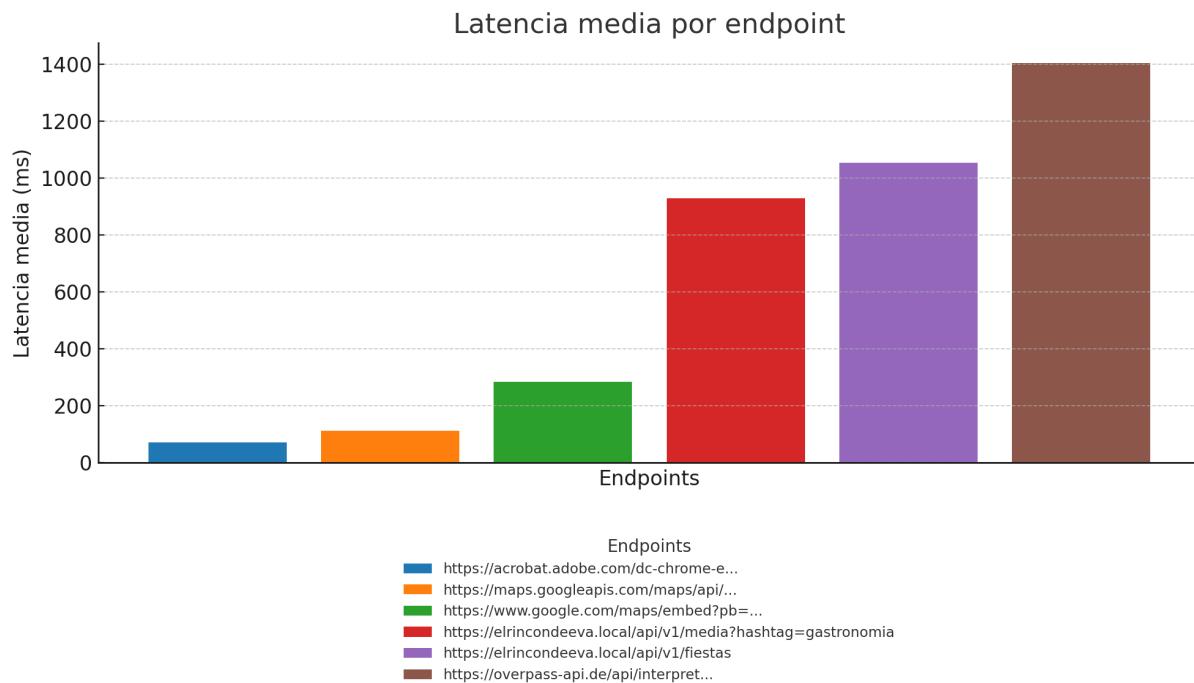


Figura 6.23: Latencia media por endpoint — datos de `visual1.csv`

Página de entorno

→ *accesibilidad*

En este caso, la accesibilidad alcanza un 90 y el SEO mantiene el 100. Sin embargo, el rendimiento baja a 59, con una LCP (Largest Contentful Paint) cercana a los 7 segundos. Se recomienda optimizar las imágenes, precargar las que son clave para el renderizado y revisar la carga de recursos bloqueantes. También se observan advertencias sobre contraste de color y etiquetas ALT redundantes.

Página “La casa”

El análisis muestra una accesibilidad de 92 y puntuación perfecta en SEO. El rendimiento es bajo (63), principalmente debido a la carga tardía de elementos visuales. Las recomendaciones apuntan a reducir CSS y JavaScript no utilizados, definir tamaños explícitos para imágenes y adoptar formatos modernos como WebP.

Página de localizaciones

Esta página obtiene un 87 en accesibilidad y un 93 en prácticas recomendadas. El rendimiento alcanza un 71, con buena respuesta inicial pero presencia de tareas largas y recursos que bloquean el renderizado. Mejorar la compresión de texto, reducir el uso de librerías pesadas y controlar los cambios de diseño puede incrementar la puntuación.

Página de recomendaciones

Aunque destaca en accesibilidad (92) y SEO (100), el rendimiento global es bajo (61). Se identifican problemas relacionados con el renderizado de imágenes, animaciones

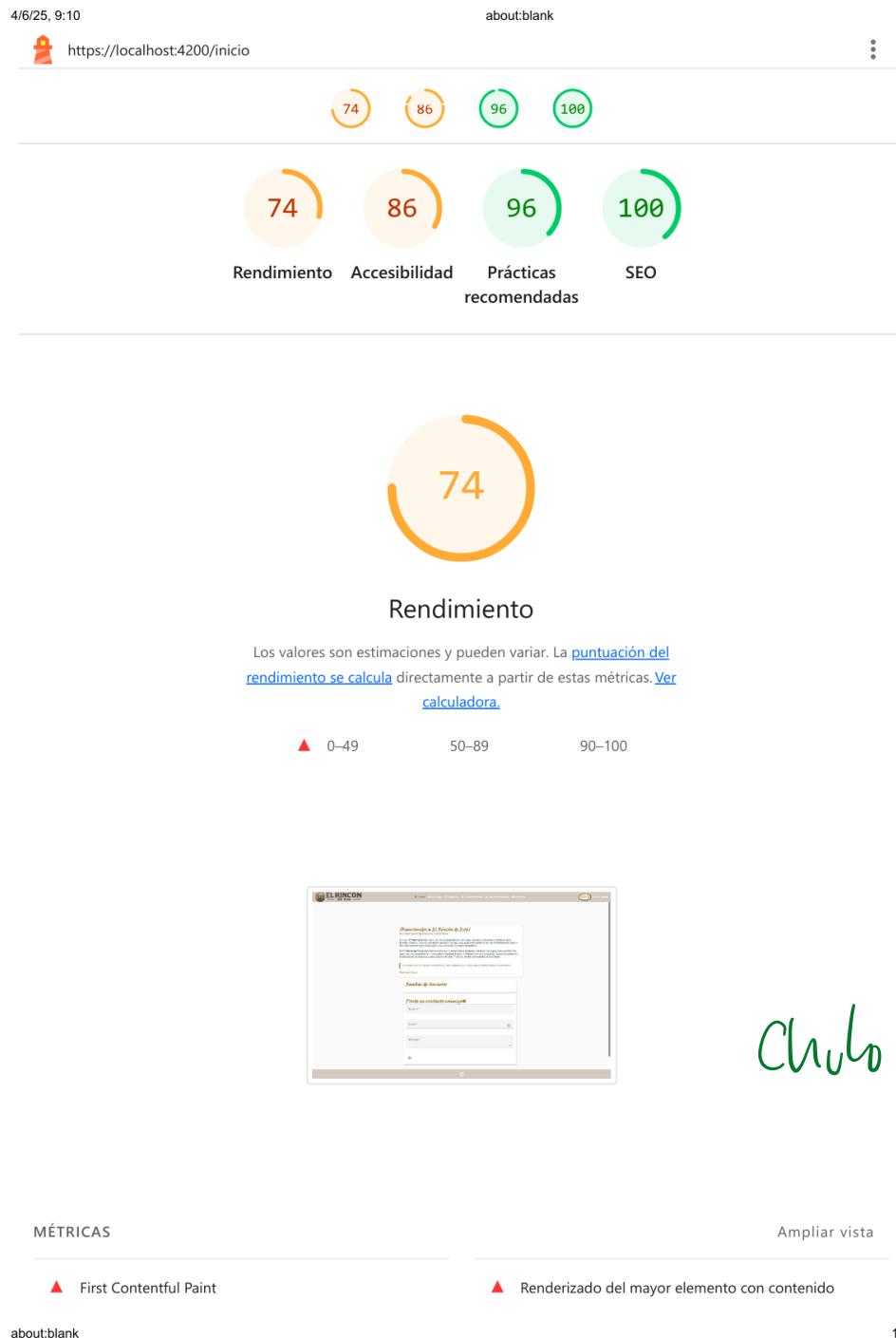


Figura 6.24: Resultados Lighthouse — Página de inicio

y encadenamiento de solicitudes. También se recomienda reducir el tamaño de las cargas útiles y mejorar el uso de caché.

Página de reservas

La página de reservas obtiene una puntuación de 93 en accesibilidad, 100 en SEO y 96 en buenas prácticas. No obstante, el rendimiento es limitado (62) debido a cambios de diseño acumulados y bloqueo del hilo principal. Se sugiere revisar las tareas largas y

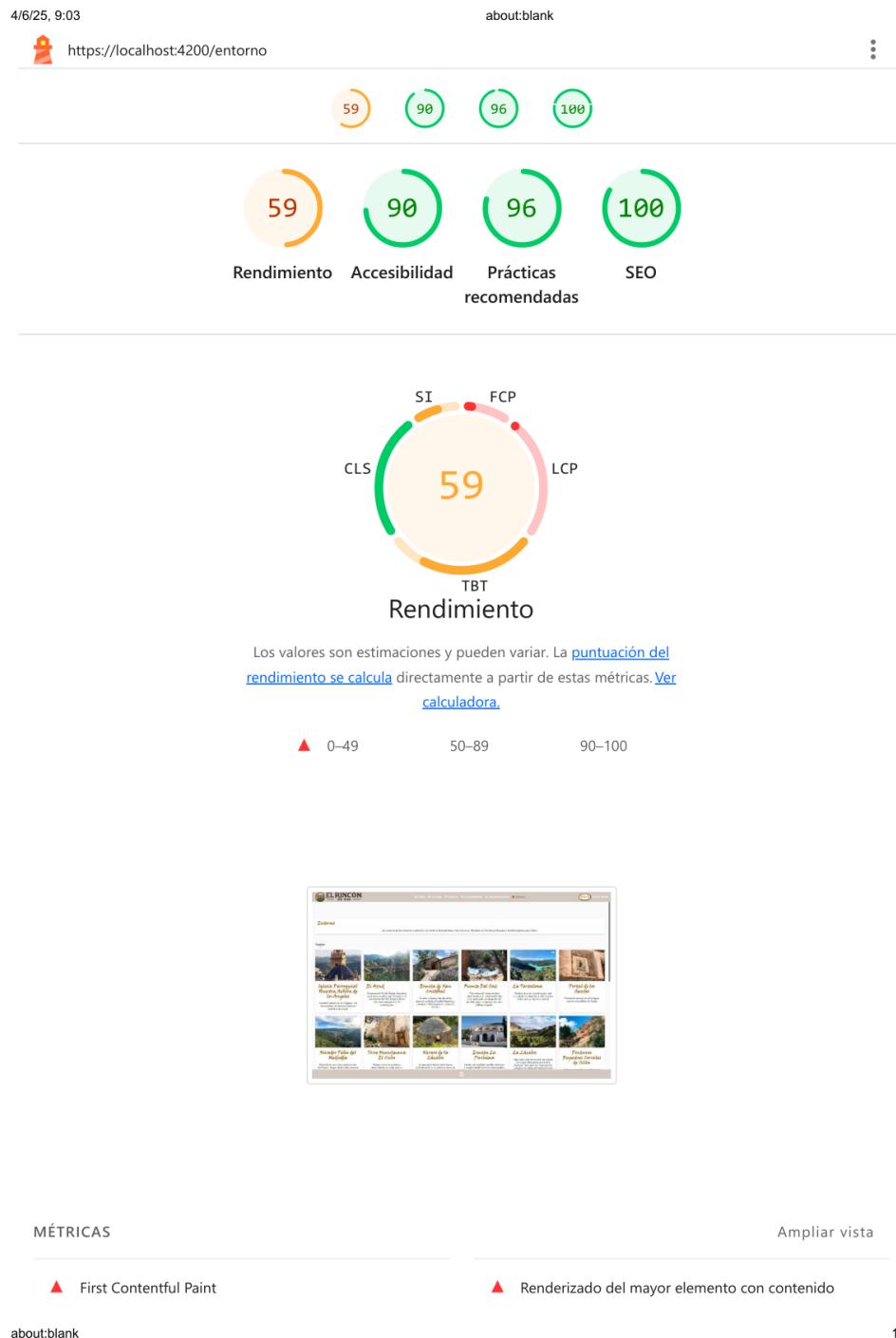


Figura 6.25: Resultados Lighthouse — Página de entorno

evitar imágenes sin compresión adecuada.

Conclusiones

En términos generales, todas las páginas analizadas presentan una **excelente puntuación en SEO (100)** y **muy buena accesibilidad (>86)**, con oportunidades de mejora principalmente en el área de rendimiento. Las recomendaciones comunes incluyen:

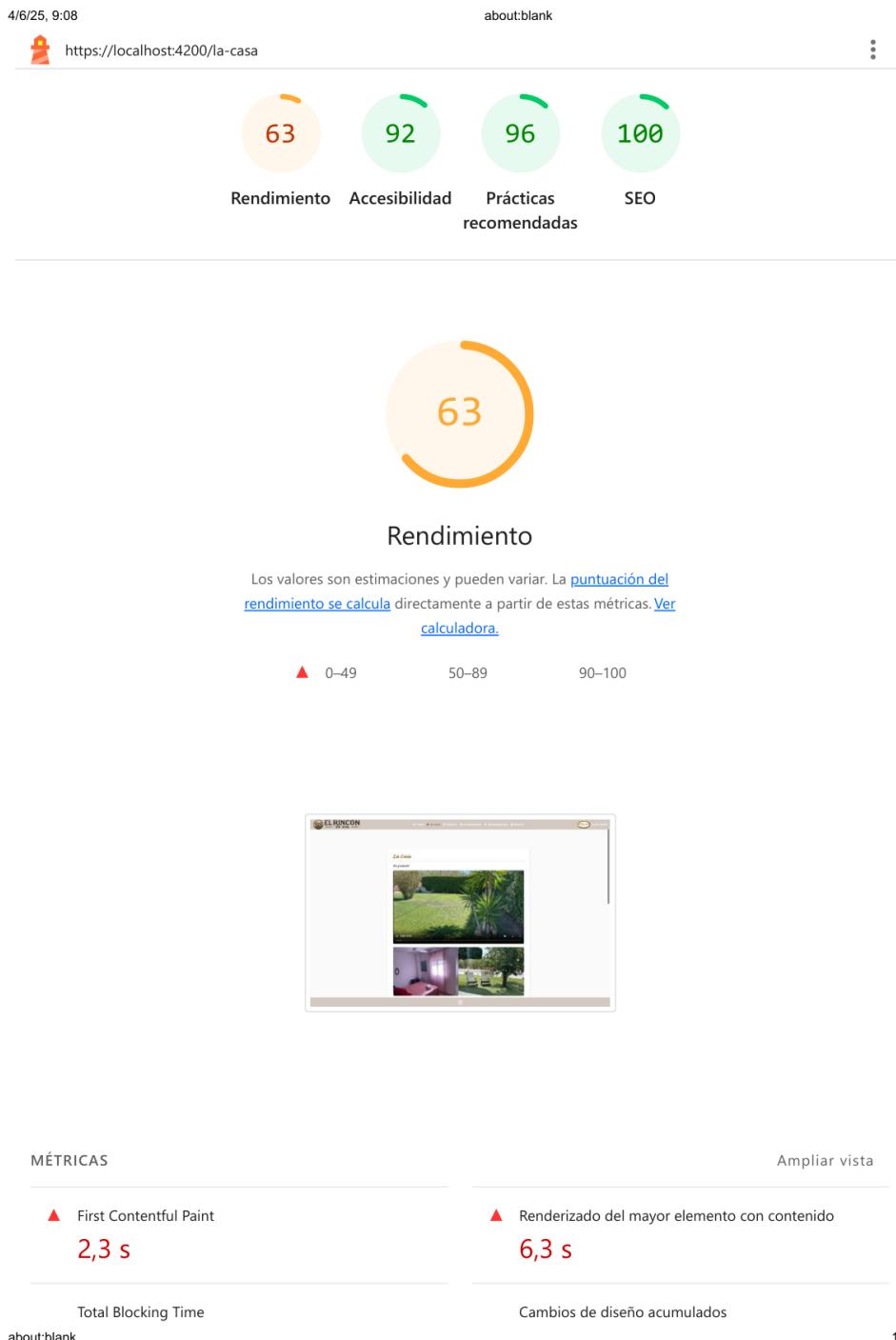


Figura 6.26: Resultados Lighthouse — Página “La casa”

- Optimizar y precargar imágenes clave.
- Reducir [CSS](#) y [JavaScript](#) no utilizados.
- Eliminar recursos que bloqueen el renderizado.
- Aplicar formatos modernos y mejorar la caché de recursos.

Estas acciones quedan planteadas como línea de trabajo futuro para incrementar la eficiencia y la experiencia de usuario del sitio web.

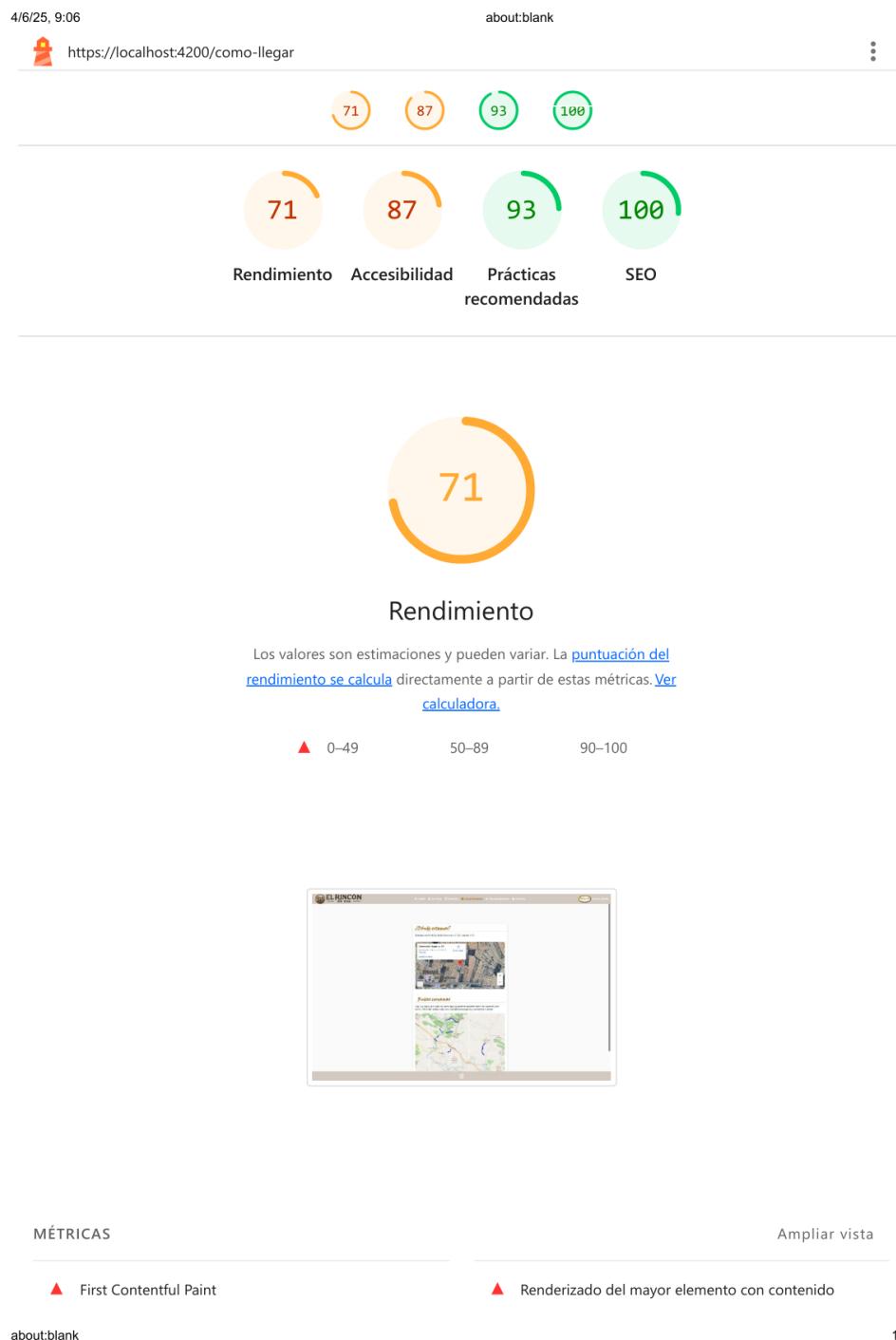


Figura 6.27: Resultados Lighthouse — Página de localizaciones

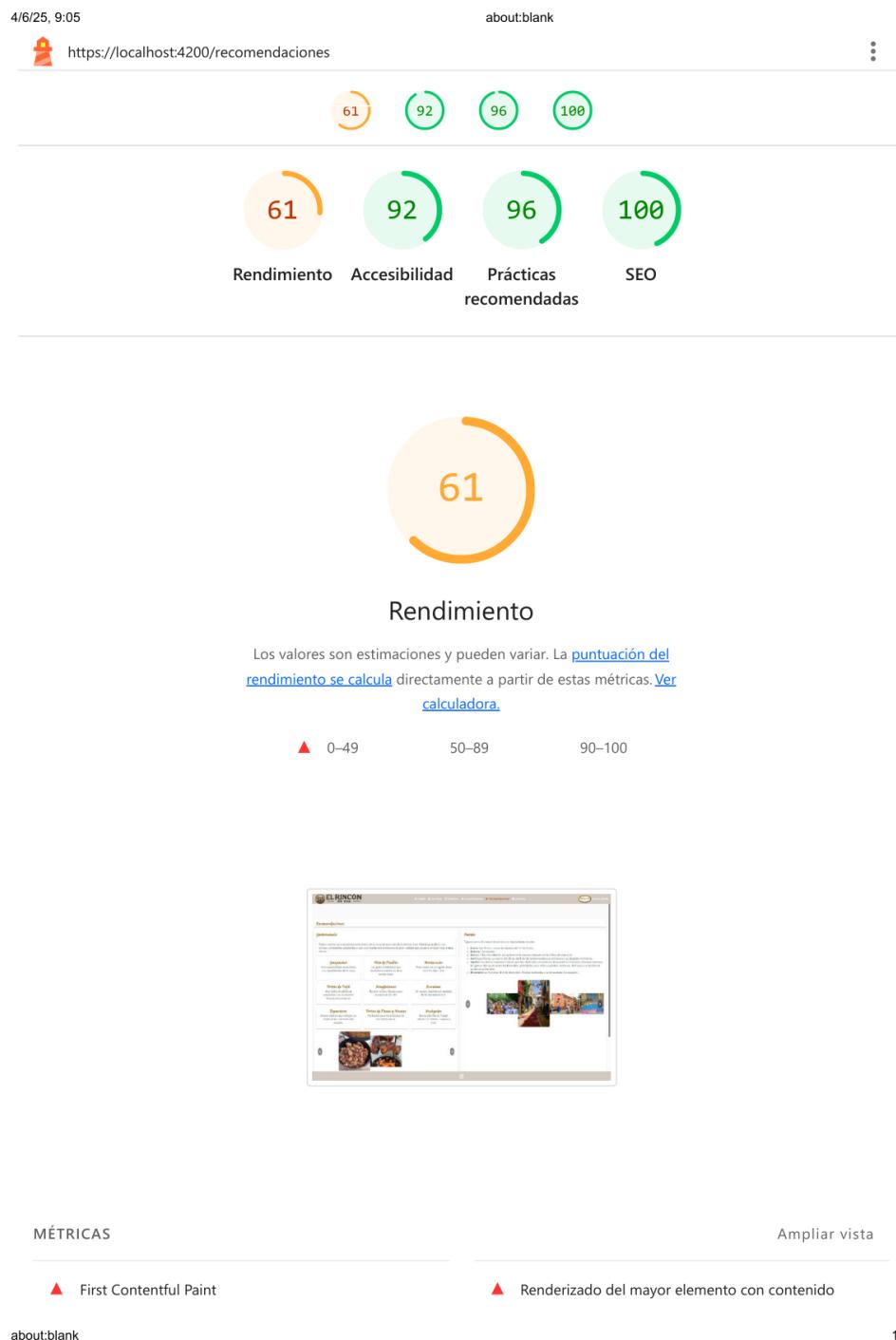


Figura 6.28: Resultados Lighthouse — Página de recomendaciones

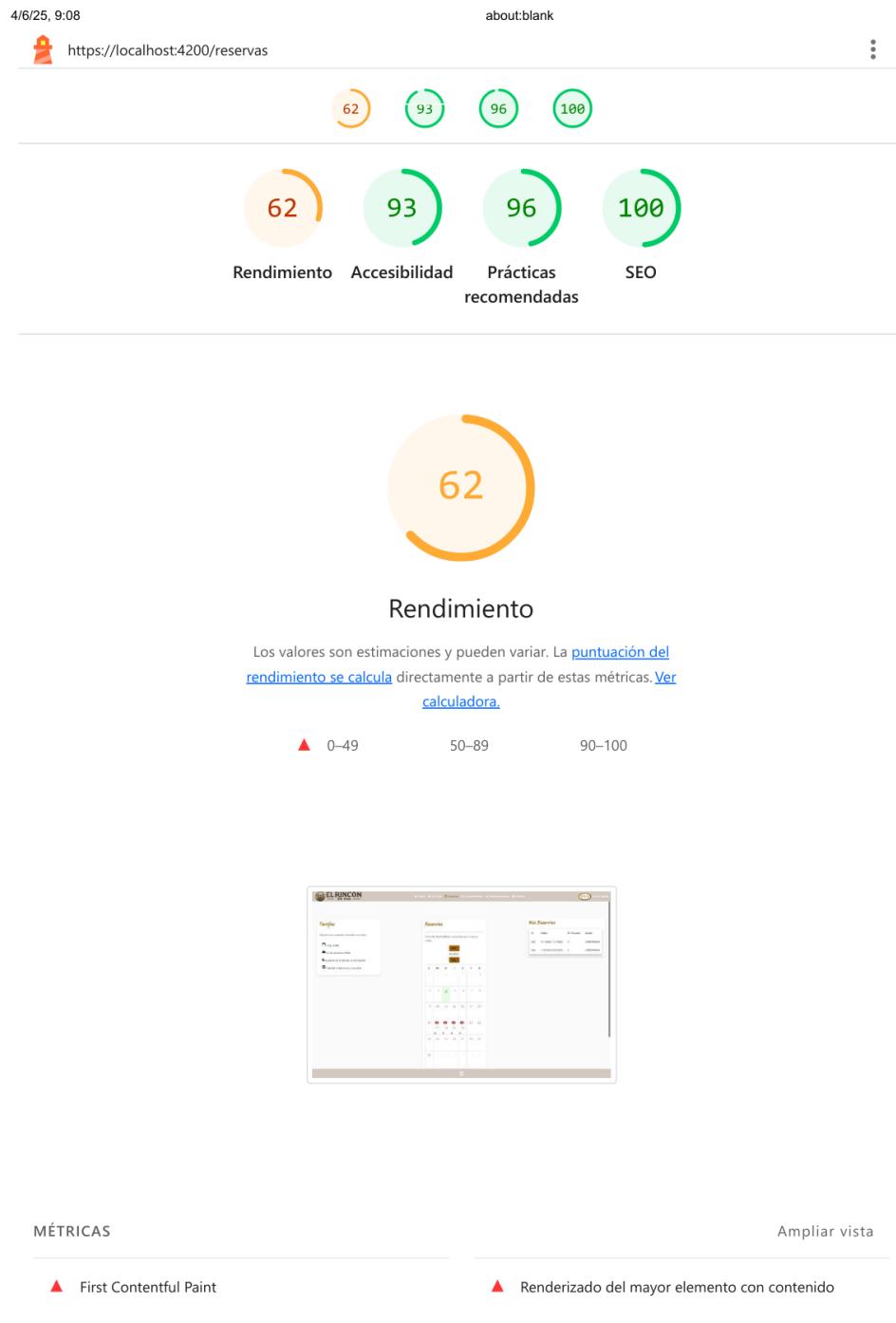


Figura 6.29: Resultados Lighthouse — Página de reservas

Capítulo 7

Conclusiones

7.1. Revisión de costes

Cuando finaliza un proyecto es necesario evaluar nuevamente los costes reales que éste ha tenido. Estos costes se dividen en costes temporales y costes en personal y recursos.

En cuanto a costes temporales, este TFM tenía estimada una duración de 17 semanas lo que equivale a unos 4 meses y medio comenzando el desarrollo en noviembre del año 2024 y terminándolo en marzo del año 2025. En esta estimación se había planteado una serie de riesgos y medidas para que el proyecto contase con un margen de error para posibles contratiempos. Se han encontrado contratiempos en las tareas relacionadas con servicios externos. En concreto, se gastó más tiempo en la intención de obtener algunos recursos web como son rutas cercanas (implementado parcialmente) o eventos, que finalmente se tomó la decisión de dejar para trabajo futuro ya que no era viable realizar una extracción de datos de forma automática. En este caso, se estima que el tiempo perdido fue de 1 semana(22 horas) lo cual está contemplado dentro del margen estipulado. Por lo que el tiempo total de desarrollo fue de 17 semanas, que es el tiempo estimado inicialmente.

Por otro lado, se encuentran los costes en personal. En este aspecto se estimaron 6.015,94€ para las 17 semanas (que incluyen el margen consumido) de trabajo siendo el salario por hora de 16,15€ . En este caso, el desarrollo terminó dentro del plazo acordado por lo que no se incurrió en costes adicionales.

El segundo coste a tener en cuenta es el coste en materiales. En la Tabla 7.1 se muestra el coste real de amortización de los materiales utilizados para el desarrollo del proyecto.

Elemento	Coste	Meses de utilización	Amortización (Formula (3.2))
Ordenador para desarrollo	890 €	0.33	73,43€

Tabla 7.1: Coste real de amortización de materiales para el desarrollo del proyecto

Para finalizar, la Tabla 7.2 muestra el coste final de cada activo utilizado durante el proyecto, incluyendo materiales, equipos de desarrollo y personal. Finalmente, la diferencia obtenida en costes respecto a lo estimado es de 0€.

Elemento	Coste (€)
Amortización portátil para desarrollo	74,16
Recursos humanos	6.015,94
Total	6.090,11 €

Tabla 7.2: Costes reales del proyecto

7.2. Conclusiones

El desarrollo del presente [TFM](#) ha permitido diseñar e implementar un sistema distribuido y desplegable en la nube para la gestión de una casa rural. Durante el proyecto, se han integrado diversas tecnologías estudiadas en el máster, abordando tanto la parte de infraestructura como la de desarrollo web, visualización de contenido y análisis de datos.

El primer paso consistió en realizar una valoración de las tecnologías disponibles para el desarrollo y despliegue de sistemas distribuidos modernos. Para ello, se analizaron herramientas de virtualización ligera como Docker, sistemas de orquestación como Kubernetes, y plataformas cloud, aplicando conocimientos adquiridos especialmente en las asignaturas de *Centro de datos y virtualización* y *Computación en la nube*. Esto permitió definir una arquitectura escalable, segura y eficiente, adecuada para la naturaleza de los microservicios propuestos.

Posteriormente, se procedió a la especificación de requisitos funcionales y no funcionales del sistema, integrando prácticas profesionales aprendidas en la asignatura de *Métodos de producción de software*. Esta fase fue clave para definir correctamente los casos de uso, los flujos de usuario y las necesidades técnicas del sistema, alineando los objetivos del proyecto con las necesidades reales de un sistema de gestión de apartamentos turísticos o casas rurales.

Una vez definidos los requisitos, se realizó la planificación temporal del proyecto, incluyendo estimación de tareas, coste y distribución del esfuerzo. Para ello se emplearon herramientas y técnicas aprendidas en la asignatura de *Métodos de producción de software*, permitiendo generar un cronograma ajustado a las fases de desarrollo del [TFM](#).

La fase de análisis y diseño se centró en estructurar correctamente la solución desde una perspectiva modular, lo cual fue posible gracias a las asignaturas de *Desarrollo basado en componentes distribuidos y servicios*, *Persistencia relacional y no relacional de datos* y *Seguridad*. En esta etapa se definieron los componentes, servicios, bases de datos, endpoints y mecanismos de comunicación entre módulos, garantizando una arquitectura cohesiva, mantenible y segura.

Durante la etapa de implementación se consolidaron y aplicaron muchos de los conocimientos prácticos del máster. Se desarrolló el [Backend](#) utilizando paradigmas de programación reactiva con tecnologías modernas, como Spring Boot, y bases de datos relacionales y no relacionales, aplicando lo aprendido en *Programación del lado del servidor* y *Persistencia relacional y no relacional de datos*. El frontend se implementó con Angular, incorporando técnicas de desarrollo web aprendidas en *Programación del lado del cliente y visualización* y se diseñó una experiencia de usuario optimizada para dispositivos móviles, integrando contenidos multimedia y funcionalidades interactivas, apoyado en la asignatura de *Gestión y distribución de contenido multimedia*.

También se implementó una [API \(Application Programming Interface\)](#) de recomendación.



Te falta la de Componentes distribuidos de forma independiente

ción de integración de redes sociales y un sistema de predicción meteorológica, haciendo uso de técnicas aprendidas en *Análisis de datos web y sociales*. La aplicación ha sido diseñada con una interfaz intuitiva, accesible y con un enfoque responsive, gracias a los conocimientos adquiridos en *Dispositivos móviles y realidad aumentada* y *Programación del lado del cliente y visualización*.

El despliegue final del sistema se realizó utilizando contenedores Docker y orquestación con Kubernetes, quedo pendiente realizar este despliegue final sobre la infraestructura cloud seleccionada durante el estudio, que se realizaría integrando scripts de automatización y configuración avanzada, poniendo en práctica las asignaturas de *Administración de recursos y automatización de operaciones* y *Computación en la nube*. Además, se tuvieron en cuenta aspectos relacionados con la ciberseguridad, tanto en la comunicación como en el almacenamiento de datos, en línea con lo aprendido en la asignatura de *Seguridad*.

*¿Lo he
seguido
lo visto
en una
aproximación?*

Finalmente, se ejecutaron distintas pruebas para garantizar el correcto funcionamiento del sistema. Se realizaron pruebas unitarias, funcionales, de seguridad y de rendimiento, que permitieron validar el sistema para una futura puesta en producción. Esta etapa se apoyó en las metodologías de pruebas estudiadas en *Métodos de producción de software* y *Seguridad*.

En cuanto al cumplimiento de los objetivos planteados al inicio del TFM, se puede concluir que:

- *Aplicar los conocimientos adquiridos en las asignaturas de desarrollo Frontend, Backend y computación en la nube* se ha logrado mediante la implementación de una plataforma web funcional para una casa rural, que integra reservas, información turística e interacciones avanzadas.
- *Facilitar la consulta de la disponibilidad y la reserva en línea del alojamiento* se ha cumplido mediante un sistema de reservas accesible desde la interfaz de usuario.
- *Mostrar información relevante sobre la casa rural y su entorno* se ha conseguido mediante secciones dedicadas a instalaciones, gastronomía y recomendaciones turísticas.
- *Integrar contenidos multimedia y datos procedentes de fuentes externas* se ha llevado a cabo a través de la incorporación de imágenes, vídeos, mapas interactivos y datos de redes sociales y meteorología.
- *Permitir la gestión de usuarios con distintos roles y funcionalidades asociadas* se ha implementado con autenticación y autorización diferenciada para administradores y huéspedes.
- *Incorporar valoraciones, actividades y recomendaciones personalizadas* se ha realizado con éxito, aunque este aspecto admite mejoras futuras en cuanto a personalización basada en preferencias del usuario.
- *Garantizar la seguridad, escalabilidad y accesibilidad del sistema* se ha trabajado mediante cifrado SSL, arquitectura basada en contenedores y buenas prácticas de desarrollo web accesible.
- *Asegurar compatibilidad con distintos dispositivos y navegadores web* se ha validado correctamente, obteniendo resultados satisfactorios en entornos móviles y de escritorio.

Si bien todos los objetivos planteados han sido cumplidos, se reconoce que cada uno de ellos ofrece margen de mejora. Estas posibles mejoras se abordan en detalle en sección 7.3 de trabajo futuro.

7.3. Trabajo futuro

El desarrollo de este TFM ha permitido adquirir una serie de conocimientos y habilidades en el ámbito del desarrollo web, la computación en la nube y la gestión de proyectos. Sin embargo, también ha puesto de manifiesto áreas que requieren atención y mejora. A continuación, se presentan las principales conclusiones y recomendaciones para el trabajo futuro, divididas en dos secciones: mejoras en el sistema Backend y Frontend y mejoras en la infraestructura del sistema, incluyendo la gestión de contenedores Docker y el despliegue en la nube:

- Mejoras en el backend:

- Añadir imágenes a las reseñas.
- Incorporar nuevos tipos de orígenes de datos para extraer eventos, recomendaciones o lugares de interés.
- Mejorar la gestión de usuarios, incluyendo el registro, el cambio de contraseña y la administración de usuarios administradores.
- Optimizar los procesos de reserva y los mensajes de correo electrónico, haciendolos más profesionales e incluyendo facturas en PDF.
- Implementar una mejor gestión de errores y excepciones para mejorar la experiencia del usuario y la seguridad del sistema.
- Incorporar orígenes de datos para poder importar eventos, dado que actualmente no se ha encontrado una fuente de datos que permita la importación de eventos de forma automática.
- Mejorar el rendimiento del sistema, optimizando las consultas a la base de datos y el uso de caché para reducir la carga en el servidor y mejorar los tiempos de respuesta.

- Mejoras en el frontend:

- Añadir filtros para eventos o lugares de interés por tipo, fecha o distancia.
- Incluir funcionalidades de cambio de contraseña y creación de nuevos usuarios administradores.
- Añadir filtros en la pantalla de gestión de reservas y en la vista de reservas de cada usuario.
- Incorporar un sistema de notificaciones para avisar al usuario de cambios en su reserva o en su cuenta en la propia plataforma.
- Mejorar la interfaz de usuario para hacerla más atractiva, accesible y fácil de usar, cumpliendo con las normas de accesibilidad y usabilidad web.
- Evaluar con detenimiento el SEO, mejorando el rendimiento de la página para mayor posicionamiento posicionamiento y usabilidad en la aplicación, realizando pruebas con usuarios reales y con herramientas de análisis estudiadas, para identificar áreas de mejora.

- Adaptar la aplicación a dispositivos móviles, asegurando que todas las funcionalidades sean accesibles y usables en pantallas pequeñas, consiguiendo generar una aplicación PWA funcional.
- Realizar pruebas de usabilidad con usuarios reales para identificar áreas de mejora en la experiencia de usuario y la interfaz.

Por otro lado, se valoró en el estudio del estado del arte realizar el despliegue en una plataforma en la nube, pero dado que la plataforma todavía debe ser revisada por el cliente se ha decidido no realizarlo hasta que el sistema esté validado y se pueda realizar un despliegue en producción. En el futuro se puede valorar la posibilidad de realizar un despliegue en una plataforma en la nube como AWS [61] o Google Cloud [62]. Se deberá implantar los ficheros de Kubernetes ya generados y adaptarlos a la plataforma de nube seleccionada o utilizar una solución común, usando tecnologías como Terraform [62] (la cual se vio durante el desarrollo del Máster) y regenerar las imágenes Docker con los últimos cambios, para posteriormente realizar un despliegue en la nube. Esto permitiría una mayor escalabilidad y flexibilidad del sistema, así como una mejor gestión de los recursos y una mayor seguridad. En este punto también se generarán los certificados para exponer tanto el Backend como el Frontend mediante el protocolo HTTPS.

Tu CAM
acabas
Si y le das
ant/ no le
urdos o stop.

Capítulo 8

Bibliografía

→ → false fecha consulta

- [1] Instituto Nacional de Estadística, *Turismo rural*, mayo de 2025. dirección: <https://www.ine.es/consul/serie.do?s=EOT1145&c=2&nult=100> (citado en página 27).
 - [2] Google, *Google Trends*, mayo de 2025. dirección: <https://trends.google.es/trends/explore?cat=67&date=today%205-y&geo=ES&q=chelva,titagras,tuejar&hl=es> (citado en página 27).
 - [3] Airbnb, *Airbnb - Vacation Rentals, Cabins, Beach Houses, Unique Homes & Experiences*, 2024. visitado nov. de 2024. dirección: <https://www.airbnb.com> (citado en página 33).
 - [4] Booking.com, *Booking.com / Official site / The best hotels, flights, car rentals & accommodations*, 2024. visitado nov. de 2024. dirección: <https://www.booking.com> (citado en página 34).
 - [5] Ruralka, *Ruralka - Hoteles con encanto en entornos rurales*, 2024. visitado nov. de 2024. dirección: <https://www.ruralka.com> (citado en página 36).
 - [6] Koomot.com, *Todo lo que necesitas para salir a explorar*, 2024. visitado mayo de 2025. dirección: <https://www.komoot.com/es-es/features> (citado en página 36).
 - [7] Ayuntamiento de Tuéjar, *Turismo Tuéjar*, <https://tuejarturismo.es/>, Consultado el 27 de mayo de 2025, 2025 (citado en página 38).
 - [8] Ayuntamiento de Chelva, *Turismo Chelva*, <https://www.turismochelva.es/>, Consultado el 27 de mayo de 2025, 2025 (citado en página 38).
 - [9] Ayuntamiento de Tuéjar, *Web oficial del Ayuntamiento de Tuéjar*, <https://www.tuejar.es/>, Consultado el 27 de mayo de 2025, 2025 (citado en página 40).
 - [10] Meta Platforms Inc., *Instagram Graph API*, <https://developers.facebook.com/docs/instagram-api/>, Consultado el 27 de mayo de 2025, 2025 (citado en página 41).
 - [11] Open-Meteo, *Open-Meteo API*, <https://open-meteo.com/>, Consultado el 27 de mayo de 2025, 2025 (citado en página 41).
 - [12] Agencia Estatal de Meteorología (AEMET), *AEMET OpenData API*, <https://opendata.aemet.es/centrodedescargas/inicio>, Consultado el 27 de mayo de 2025, 2025 (citado en página 41).
 - [13] Google, *Angular*, 2024. visitado nov. de 2024. dirección: <https://angular.io> (citado en páginas 42, 127).
 - [14] VMware, *Spring Boot*, 2024. visitado nov. de 2024. dirección: <https://spring.io/projects/spring-boot> (citado en páginas 42, 43).

- [15] Docker, Inc., *Docker*, 2024. visitado nov. de 2024. dirección: <https://www.docker.com> (citado en páginas 42, 44).
- [16] The Kubernetes Authors, *Kubernetes*, 2024. visitado nov. de 2024. dirección: <https://kubernetes.io> (citado en páginas 42, 45).
- [17] Meta, *React*, 2024. visitado nov. de 2024. dirección: <https://react.dev> (citado en página 42).
- [18] VMware, *Spring WebFlux*, 2024. visitado nov. de 2024. dirección: <https://docs.spring.io/spring-framework/reference/web/webflux.html> (citado en páginas 43, 118).
- [19] Django Software Foundation, *Django - The Web framework for perfectionists with deadlines*, 2024. visitado nov. de 2024. dirección: <https://www.djangoproject.com> (citado en página 43).
- [20] Flask, *Flask*, feb. de 2024. visitado feb. de 2024. dirección: <https://flask.palletsprojects.com/en/2.0.x/> (citado en página 43).
- [21] Express Contributors, *Express*, 2024. visitado nov. de 2024. dirección: <https://expressjs.com> (citado en página 44). 
- [22] LXC Project, *Linux Containers (LXC)*, ~~Accedido el~~ 26 de mayo de 2025, 2024. dirección: <https://linuxcontainers.org/> (citado en páginas 44, 45). 
- [23] Docker Inc., *Docker Swarm Overview*, ~~Accedido el~~ 26 de mayo de 2025, 2024. dirección: <https://docs.docker.com/engine/swarm/> (citado en páginas 45, 46).
- [24] MongoDB, Inc., *MongoDB*, 2024. visitado nov. de 2024. dirección: <https://www.mongodb.com> (citado en páginas 46, 114, 122).
- [25] Apache Software Foundation, *Apache Cassandra*, 2024. visitado nov. de 2024. dirección: <https://cassandra.apache.org> (citado en página 47).
- [26] PostgreSQL Global Development Group, *PostgreSQL*, 2024. visitado nov. de 2024. dirección: <https://www.postgresql.org> (citado en página 47).
- [27] Oracle, *MySQL*, 2024. visitado nov. de 2024. dirección: <https://www.mysql.com> (citado en página 47).
- [28] Redis Ltd., *Redis*, 2024. visitado nov. de 2024. dirección: <https://redis.io> (citado en página 48).
- [29] Danga Interactive, *Memcached*, 2024. visitado nov. de 2024. dirección: <https://memcached.org> (citado en página 48).
- [30] Amazon Web Services, *Amazon Web Services*, 2024. visitado nov. de 2024. dirección: <https://aws.amazon.com> (citado en páginas 48, 175).
- [31] Microsoft, *Microsoft Azure*, 2024. visitado nov. de 2024. dirección: <https://azure.microsoft.com> (citado en página 49).
- [32] Google Cloud Platform, *Google Cloud Platform*, 2024. visitado nov. de 2024. dirección: <https://cloud.google.com> (citado en páginas 49, 175).
- [33] Vercel, *Vercel*, 2024. visitado nov. de 2024. dirección: <https://vercel.com> (citado en página 49).
- [34] Netlify, *Netlify*, 2024. visitado nov. de 2024. dirección: <https://www.netlify.com> (citado en página 49).
- [35] Render, *Render*, 2024. visitado nov. de 2024. dirección: <https://render.com> (citado en página 49).

- [36] Firebase, *Firebase*, 2024. visitado nov. de 2024. dirección: <https://firebase.google.com> (citado en página 49).
- [37] JUnit 5, *JUnit 5*, mayo de 2025. dirección: <https://junit.org/junit5/> (citado en páginas 50, 146).
- [38] Open Source Initiative, *Mockito*, mayo de 2025. dirección: <https://site.mockito.org/> (citado en páginas 50, 146).
- [39] Inc. 2025 Postman, *Postman*, mayo de 2025. dirección: <https://www.postman.com/> (citado en páginas 50, 146).
- [40] Cypress.io, *Cypress*, mayo de 2025. dirección: <https://www.cypress.io/> (citado en páginas 50, 149). Visita
- Visitado* [41] Apache Software Foundation, *Apache JMeter*, <https://jmeter.apache.org/>, Último acceso: junio de 2025, 2024 (citado en páginas 50, 158).
- Visitado* [42] Checkmarx, *Zap*, 2025. dirección: <https://www.zaproxy.org/> (citado en páginas 50, 156).
- [43] *Visual Studio Code*, <https://code.visualstudio.com/>, Versión utilizada para el desarrollo del proyecto, 2025. visitado 26 de mayo de 2025 (citado en página 50).
- [44] *GitHub*, <https://github.com/>, Plataforma para gestión de versiones y colaboración, 2025. visitado 26 de mayo de 2025 (citado en página 50).
- Faltó* [45] *PlantUML - UML Tool*, <https://plantuml.com/>, Herramienta para creación de diagramas UML mediante texto, 2025. visitado 26 de mayo de 2025 (citado en página 50).
- Visitado* [46] Leslie Lamport, *LaTeX: A Document Preparation System*, Sistema de composición tipográfica utilizado para la memoria, 1994. dirección: <https://www.latex-project.org/> (citado en página 50).
- Visitado* [47] *arara: A TeX automation tool*, <https://ctan.org/pkg/arara>, Herramienta para automatizar la compilación de documentos LaTeX, 2025. visitado 26 de mayo de 2025 (citado en página 50).
- Visitado* [48] CTAN, *Comprehensive TEX Archive Network*, Visitado en Febrero de 2021, nov. de 2024. dirección: <https://www.ctan.org/pkg/pgfpgant> (citado en páginas 50, 60).
- [49] Sylvia Ilieva, Penko Ivanov y Eliza Stefanova, “Analyses of an agile methodology implementation,” en *Proceedings. 30th Euromicro Conference, 2004.*, IEEE, 2004, páginas 326-333. DOI: [10.1109/EURMIC.2004.1333387](https://doi.org/10.1109/EURMIC.2004.1333387) (citado en página 58).
- [50] Mete Mazlum y Ali Fuat Güneri, “CPM, PERT and project management with fuzzy logic technique and implementation on a business,” *Procedia-Social and Behavioral Sciences*, volumen 210, páginas 348-357, 2015. DOI: [10.1016/j.sbspro.2015.11.378](https://doi.org/10.1016/j.sbspro.2015.11.378) (citado en página 58). Visita
- Visitado* [51] Asana, *Planning poker: la estrategia integral para la estimación ágil*, ~~Accedido~~ el 26 de mayo de 2025, 2025. dirección: <https://asana.com/es/resources/planning-poker> (citado en página 58). Visita
- Visitado* [52] Michael Page, *Estudio de remuneración 2024*, ~~Visitado~~ en Noviembre de 2024, nov. de 2024. dirección: https://www.michaelpage.es/sites/michaelpage.es/files/estudio_remuneracion_tecnologia_2024.pdf (citado en página 60).

1 2 referencias
no web!

- [53] Gobierno de España, *Inicio Trabajadores Cotización / Recaudación de Trabajadores Bases y tipos de cotización 2024*, Visitado en Noviembre de 2024, nov. de 2024. dirección: <https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537> (citado en página 60).
- [54] Agencia Tributaria, *Tabla de amortización simplificada*, Visitado en Noviembre de 2024, nov. de 2024. dirección: https://sede.agenciatributaria.gob.es/Sede/ayuda/manuales-videos-folletos/manuales-practicos/folleto-actividades-economicas/3-impuesto-sobre-renta-personas-fisicas/3_5-estimacion-directa-simplificada/3_5_4-tabla-amortizacion-simplificada.html (citado en página 60).
- [55] VMware, *Spring Security*, 2024. visitado nov. de 2024. dirección: <https://spring.io/projects/spring-security> (citado en páginas 101, 103).
- [56] Jonathan Hedley y jsoup contributors, *jsoup: Java HTML Parser*, 2025. dirección: <https://jsoup.org> (citado en página 114).
- [57] Facebook, *Instagram API with Instagram Login Overview*, Página web, Accedido el 23 de noviembre de 2024, 2024. dirección: <https://developers.facebook.com/docs/instagram-platform/instagram-api-with-instagram-login/overview> (citado en página 121).
- [58] Open-meteo, *Weather API*, Visitado en Abril de 2025, abr. de 2025. dirección: <https://open-meteo.com/en/docs> (citado en página 122).
- [59] Apache Software Foundation, *Maven*, mayo de 2025. dirección: <https://maven.apache.org/> (citado en página 136).
- [60] Swagger, *Swagger*, 2025. dirección: <https://swagger.io/why-swagger/> (citado en página 156).
- [61] Google, *Lighthouse*, <https://developer.chrome.com/docs/lighthouse/overview?hl=es-419>, Último acceso: junio de 2025, 2025 (citado en página 161).
- [62] Hashicorp, *Angular*, 2024. visitado mayo de 2025. dirección: <https://developer.hashicorp.com/terraform> (citado en página 175).

¡ De 62 referencias únicamente 2 no
son web. Debemos abrir libros, artículos
o informes técnicos para todo aquello que
quieras !

Apéndice A

Glosarios

A.1. Acrónimos y abreviaciones

AEAT

Agencia Estatal de Administración Tributaria. [60](#)

AEMET

Agencia Estatal de Meteorología. [41](#), [42](#)

AWS

Amazon Web Services. [48](#), [49](#), [175](#)

CDN

Content Delivery Network. [121](#)

CloudFunctions

Google Cloud Functions. [49](#)

CloudStorage

Google Cloud Storage. [49](#)

ComputeEngine

Google Compute Engine. [49](#)

CORS

Cross-Origin Resource Sharing. [103](#)

CSP

Content Security Policy. [157](#), [158](#)

CSS

Cascading Style Sheets. [114](#), [163](#), [166](#)

DNS

Domain Name System. [138](#)

DTO

Data Transfer Object. [102](#)

E2E

End to End. [25](#), [149](#), [248](#), [250](#), [251](#)

EC2

Elastic Compute Cloud. [48](#)

EKS

Elastic Kubernetes Service. [48](#)

GCP

Google Cloud Platform. [49](#)

HTML

HyperText Markup Language. [16](#), [185](#), [220](#)

HTML5

HyperText Markup Language 5.0. [127](#), [131](#), [132](#), [133](#), [134](#)

HTTPS

HyperText Transfer Protocol Secure. [97](#), [98](#), [144](#), [160](#), [175](#)

IaaS

Infraestructure as a Service. [46](#)

INE

Instituto Nacional de Estadística. [27](#)

ISO

International Organization for Standardization. [122](#)

JSON

JavaScript Object Notation. [121](#)

JWT

JSON Web Token. [103](#)

K8S

Kubernetes. [42](#), [45](#), [46](#), [48](#), [49](#), [56](#)

MVC

Modelo Vista Controlador. [43](#)

PaaS

Platform as a Service. [46](#), [60](#)

RDS

Relational Database Service. [48](#)

REST

Representational State Transfer. [44](#), [159](#)

S3

Simple Storage Service. [48](#)

SEO

Search Engine Optimization. [160](#), [162](#), [163](#), [164](#), [165](#), [174](#)

SQL Injection

Inyección de SQL. [156](#)

SSL

Secure Sockets Layer. [159](#)

TCP

Transmission Control Protocol. [138](#)

TFM

Trabajo Fin de Máster. [3](#), [11](#), [27](#), [28](#), [171](#), [172](#), [173](#), [174](#)

TLS

Transport Layer Security. [55](#), [144](#)

TWCAM

Tecnologías Web, Aplicaciones Móviles y Servicios. [29](#)

XML

eXtensible Markup Language. [157](#)

XPath

XML Path Language. [114](#)

XSS

Cross-Site Scripting. [156](#)

YAML

YAML Ain't Markup Language. [136](#)

A.2. Términos

API (Application Programming Interface)

Interfaz de programación que permite la comunicación y el intercambio de datos entre diferentes sistemas o aplicaciones. [18](#), [41](#), [44](#), [62](#), [63](#), [75](#), [90](#), [91](#), [92](#), [98](#), [159](#), [160](#), [172](#)

Azure

Microsoft Azure, una plataforma de servicios en la nube de Microsoft.. [49](#)

Backend

Parte de una aplicación web que maneja la lógica de negocio, la gestión de datos y la comunicación con el frontend. [18](#), [30](#), [43](#), [44](#), [48](#), [49](#), [50](#), [62](#), [131](#), [132](#), [134](#), [146](#), [149](#), [156](#), [159](#), [172](#), [173](#), [174](#), [175](#)

distribución Beta

Distribución de probabilidad utilizada en el análisis PERT para modelar incertidumbres en tiempos de ejecución de tareas, considerando los valores optimista, pesimista y más probable para generar una estimación ponderada.. [58](#)

Docker

Una plataforma para desarrollar, enviar y ejecutar aplicaciones dentro de contenedores, lo que permite un entorno aislado y replicable.. [16](#), [98](#), [174](#), [240](#)

Framework

Conjunto de herramientas y librerías que proporciona una estructura de desarrollo para simplificar la construcción y el mantenimiento de aplicaciones. [42](#), [43](#), [44](#)

Frontend

Parte de una aplicación web que interactúa directamente con el usuario y controla la interfaz de usuario y la experiencia visual. [13](#), [15](#), [30](#), [42](#), [44](#), [48](#), [49](#), [50](#), [62](#), [75](#), [103](#), [105](#), [127](#), [156](#), [159](#), [173](#), [174](#), [175](#)

full-stack

Hace referencia a un desarrollador o un sistema que trabaja con todas las capas de desarrollo de una aplicación, desde el front-end (interfaz de usuario) hasta el back-end (servidores y bases de datos). [44](#)

Gantt

Un diagrama de Gantt es una herramienta de gestión de proyectos que se utiliza para representar visualmente el cronograma de un proyecto. Cada tarea se representa como una barra horizontal, cuyo tamaño y posición dependen de su duración y fecha de inicio. El diagrama permite visualizar la relación entre las diferentes tareas, identificar los plazos y controlar el progreso de un proyecto. Es utilizado ampliamente en la planificación de proyectos para asegurar que las tareas se completen dentro del tiempo estimado.. [60](#)

HyperText Markup Language 5.0

Ultima versión de [HTML](#). [131](#), [132](#), [133](#), [134](#), [182](#)

Ingress Controller

Componente de Kubernetes que gestiona el acceso externo a los servicios del clúster, normalmente HTTP/HTTPS, mediante reglas de enrutamiento. [97](#), [98](#)

JavaScript

Un lenguaje de programación de alto nivel, interpretado y orientado a objetos, utilizado principalmente para crear interactividad en sitios web.. [42](#), [44](#), [163](#), [166](#)

Kubernetes

Un sistema de orquestación de contenedores que automatiza el despliegue, la escalabilidad y la gestión de aplicaciones en contenedores.. [97](#), [98](#), [136](#), [145](#), [159](#), [175](#)

Meta

Una compañía de tecnología que desarrolla productos como Facebook, Instagram, WhatsApp y otras aplicaciones relacionadas con la realidad virtual y aumentada..
[42](#)

microservicio

Estilo de arquitectura donde la funcionalidad se divide en pequeños servicios independientes que se comunican entre sí, normalmente a través de APIs. [97](#)

MongoDB

Una base de datos NoSQL orientada a documentos que permite almacenar datos en formato BSON, ideal para aplicaciones con esquemas flexibles.. [98](#), [141](#), [143](#), [144](#)

OpenAPI

Especificación para construir APIs RESTful, que define un formato estándar para describir los endpoints, parámetros, respuestas y otros aspectos de una API. Permite la generación automática de documentación y clientes para interactuar con la API..
[156](#)

PersistentVolume

Recurso de almacenamiento en Kubernetes que representa un volumen físico o lógico existente, gestionado por el administrador del clúster. [15](#), [98](#), [136](#)

PersistentVolumeClaim

Solicitud de almacenamiento persistente en Kubernetes, que un contenedor realiza para vincularse a un volumen disponible. [15](#), [98](#), [136](#), [137](#)

PostgreSQL

Una base de datos relacional de código abierto, conocida por su fiabilidad, robustez y características avanzadas de gestión de datos.. [49](#), [97](#), [98](#), [139](#), [140](#)

Program Evaluation and Review Technique

Técnica de Revisión y Evaluación de Programas (Program Evaluation and Review Technique), una herramienta de gestión de proyectos que se utiliza para planificar y coordinar tareas dentro de un proyecto. Utiliza estimaciones de tiempos optimistas, más probables y pesimistas para calcular el tiempo esperado para completar una tarea.. [58](#)

PWA

Una aplicación web progresiva (PWA) es un tipo de aplicación que combina lo mejor de las aplicaciones web y móviles. Se caracteriza por ser rápida, confiable y atractiva, permitiendo a los usuarios interactuar con ella incluso sin conexión a Internet. Las PWAs utilizan tecnologías web modernas para ofrecer una experiencia similar a la de una aplicación nativa, incluyendo notificaciones push, acceso a la cámara y almacenamiento local. [symbol](#). [175](#)

Redis

Modelo de base de datos en memoria de tipo clave-valor, utilizado para almacenamiento en caché y manejo de datos temporales. [49](#)

Serverless

Arquitectura de computación en la que los desarrolladores crean y ejecutan aplicaciones sin tener que gestionar los servidores. El proveedor de la nube es responsable de la infraestructura y del escalado, lo que permite a los desarrolladores concentrarse en la lógica de la aplicación. Aunque el término "serverless" implica que no hay servidores, en realidad los servidores están siendo gestionados por el proveedor de la nube, y los desarrolladores solo pagan por el uso de recursos cuando se ejecutan sus funciones.. [49](#)

Spring Boot

Un marco de trabajo basado en Java que facilita la creación de aplicaciones backend escalables, incluyendo microservicios.. [98](#), [134](#), [136](#)

sprint

Periodo de tiempo corto, típicamente de una a cuatro semanas, durante el cual un equipo ágil desarrolla y entrega un incremento del producto que es funcional y revisable.. [58](#)

token

Archivo o valor secreto utilizado para autenticación o autorización segura entre servicios. [98](#)

TypeScript

Un superset de JavaScript que añade tipado estático y otras características a JavaScript.. [42](#)

Apéndice A

Apéndices

A.1. Código fuente SpringBoot sobre seguridad

Listado A.1: Fichero de filtro de autenticación `CustomAuthenticationFilter.java`

```
30 protected Mono<Void> onAuthenticationSuccess(Authentication authentication, WebFilterExchange
31   webFilterExchange) {
32
33     User user = (User) authentication.getPrincipal();
34     String access_token = jwtService.generateAccessToken(user.getUsername(),
35               user.getAuthorities().stream().map(GrantedAuthority::getAuthority).coll
36               ect(Collectors.toList()));
37     String refresh_token = jwtService.generateRefreshToken(user.getUsername(),
38               user.getAuthorities().stream().map(GrantedAuthority::getAuthority).co
39               llect(Collectors.toList()));
40
41     for (GrantedAuthority authority : authentication.getAuthorities()) {
42       System.out.println(authority.getAuthority());
43     }
44
45     webFilterExchange.getExchange().getResponse().getHeaders().add("access_token",
46       access_token);
47     webFilterExchange.getExchange().getResponse().getHeaders().add("refresh_token",
48       refresh_token);
49     webFilterExchange.getExchange().getResponse().getHeaders().add("username",
50       authentication.getName());
51     webFilterExchange.getExchange().getResponse().getHeaders().add("roles",
52       authentication.getAuthorities().stream().map(GrantedAuthority::getAuthority).collect(Col
53       lectors.toList()).toString());
54     return Mono.empty();
55   }
56 }
```

Listado A.2: Fichero de filtro de autorización `CustomAuthorizationFilter.java`

```
33 public Mono<Void> filter(ServerWebExchange exchange, WebFilterChain chain) {
34   String path = exchange.getRequest().getPath().value();
35
36   if ("/api/v1/login".equals(path)) {
37     return chain.filter(exchange);
38   } else {
39     String authHeader = exchange.getRequest().getHeaders().getFirst("Authorization");
40     if (authHeader != null && authHeader.startsWith("Bearer ")) {
41       try {
42         String token = jwtService.getTokenFromHeader(authHeader);
43         String username = jwtService.getUsernameFromToken(token);
44
45         Collection<SimpleGrantedAuthority> authorities =
46           jwtService.getAuthoritiesFromToken(token);
47
48         UsernamePasswordAuthenticationToken auth = new
49           UsernamePasswordAuthenticationToken(username, null, authorities);
50       }
51     }
52   }
53 }
```

```

48     SecurityContextHolder.getContext().setAuthentication(auth);
49     if (SecurityContextHolder.getContext().getAuthentication().isAuthenticated()) {
50         for (GrantedAuthority authority :
51             SecurityContextHolder.getContext().getAuthentication().getAuthorities()) {
52             System.out.println("- " + authority.getAuthority());
53         }
54         return chain.filter(exchange);
55     } else {
56         return handleException(exchange, new Exception("Forbidden"));
57     }
58
59
60     } catch (Exception exception) {
61         return handleException(exchange, exception);
62     }
63 } else {
64     return chain.filter(exchange);
65 }
66
67 }
68
69
70 private Mono<Void> handleException(ServerWebExchange exchange, Throwable throwable) {
71     exchange.getResponse().getHeaders().set("error", throwable.getMessage());
72     exchange.getResponse().setRawStatusCode(403);
73     Map<String, String> error = Map.of("error_msg", throwable.getMessage());
74     exchange.getResponse().getHeaders().setContent-Type(MediaType.APPLICATION_JSON);
75     try {
76         return exchange.getResponse().writeWith(Mono
77                                         .just(exchange.getResponse().bufferFactory().wrap(new
78                                         ObjectMapper().writeValueAsBytes(error))));
79     } catch (Exception e) {
80         return Mono.error(e);
81     }
82 }
```

A.2. Código fuente SpringBoot sobre gestión de emails

Listado A.3: Fichero de servicio para enviar emails EmailService.java

```

12 @Service
13 public class EmailService {
14     @Autowired
15     private JavaMailSender emailSender;
16     @Value("${spring.email.email}")
17     private String email;
18     public Mono<Reserva> sendEmail(Reserva reserva) {
19         SimpleMailMessage message = new SimpleMailMessage();
20         message.setSubject("Reserva número: " + reserva.getId()+" confirmada
21         "+reserva.getNombre()+"|"+reserva.getDni());
22         message.setText("Reserva confirmada a nombre de: " + reserva.getNombre()+" con número de
23         reserva: " + reserva.getId() + " para el día " + reserva.getFechaInicio() + " hasta el día " +
24         reserva.getFechaFin() + " para " + reserva.getNumPersonas() + " personas. El precio total es de "
25         + reserva.getPrecioTotal() + "€. Gracias por confiar en nosotros."+"Pongase en contacto con el
26         administrador para realizar el pago de otra forma al 669 511 506 o proceda con la realización del
27         pago mediante transferencia al siguiente número de cuenta:. Ponga en el concepto su nombre de la
28         reserva y el número de la reserva que le hemos facilitado.");
29         message.setTo(reserva.getEmail());
30         message.setFrom(email);
31         emailSender.send(message);
32         return Mono.just(reserva);
33     }
34
35     public Mono<Reserva> sendEmailPay(Reserva reserva, String motivo) {
36         SimpleMailMessage message = new SimpleMailMessage();
37         message.setSubject("Reserva número: " + reserva.getId()+" pagada
38         "+reserva.getNombre()+"|"+reserva.getDni());
```

```

31     message.setText("Reserva pagada. Cantidad recibida: " + reserva.getPrecioTotal() + "€" + " a nombre
32     de: " + reserva.getNombre() + " con número de reserva: " + reserva.getId() + " para el día " +
33     reserva.getFechaInicio() + " hasta el día " + reserva.getFechaFin() + " para " +
34     reserva.getNumPersonas() + " personas. A continuación le enviamos las instrucciones detalladas
35     para hacer check-in: "+motivo+" Gracias por confiar en nosotros.");
36     message.setTo(reserva.getEmail());
37     message.setFrom(email);
38     emailSender.send(message);
39     return Mono.just(reserva);
40   }
41
42   public Mono<Reserva> sendEmailCancel(Reserva reserva, String motivo) {
43     SimpleMailMessage message = new SimpleMailMessage();
44     message.setSubject("Reserva número: " + reserva.getId() + " cancelada
45     "+reserva.getNombre()+"|"+reserva.getDni());
46     message.setText("Reserva cancelada." + "A nombre de: " + reserva.getNombre() + " con número de
47     reserva: " + reserva.getId() + " para el día " + reserva.getFechaInicio() + " hasta el día " +
        reserva.getFechaFin() + " para " + reserva.getNumPersonas() + " personas. Lamentamos las
        molestias ocasionadas, esperamos volver a contar con vosotros. A continuación, le indicamos el
        motivo de la cancelación de su reserva: "+motivo);
48     message.setTo(reserva.getEmail());
49     message.setFrom(email);
50     emailSender.send(message);
51     return Mono.just(reserva);
52   }
53 }
```

A.3. Código fuente SpringBoot persistencia en el sistema principal

Listado A.4: Repositorio de reservas ReservaRepository.java

```

1 package es.uv.hemal.elrincondeeva.repositories;
2
3
4 import java.time.LocalDate;
5 import java.time.LocalDateTime;
6
7 import org.springframework.cglib.core.Local;
8 import org.springframework.data.r2dbc.repository.Query;
9 import org.springframework.data.r2dbc.repository.R2dbcRepository;
10
11 import org.springframework.stereotype.Repository;
12
13 import es.uv.hemal.elrincondeeva.domain.Reserva;
14 import reactor.core.publisher.Flux;
15 import reactor.core.publisher.Mono;
16
17
18 @Repository
19 public interface ReservaRepository extends R2dbcRepository<Reserva, Integer> {
20
21   @Query("SELECT id FROM reservas")
22   Flux<Integer> findAllId();
23
24
25   Flux<Reserva> findByEmail(String email);
26   @SuppressWarnings("null")
27   Mono<Reserva> findById(Integer id);
28
29   Flux<Reserva> findAllByOrderByFechaInicioDesc();
30   Flux<Reserva> findByFechaInicioGreaterThanOrEqualTo(LocalDate fechaInicio);
31   Flux<Reserva> findByEstadoAndFechaInicioLessThanOrEqualTo(Reserva.EstadoReserva estado, LocalDate fecha);
32
33   @Query("""
34     SELECT CASE
35       WHEN COUNT(r.* ) > 0 THEN TRUE
36       ELSE FALSE
37     END
38   """)
```

```
38     FROM reservas r
39     WHERE (:startDate BETWEEN r.fecha_inicio AND r.fecha_fin)
40       OR (:endDate BETWEEN r.fecha_inicio AND r.fecha_fin)
41       OR (r.fecha_inicio BETWEEN :startDate AND :endDate)
42     """")
43 Mono<Boolean> existeReservaEnFechas(LocalDate startDate, LocalDate endDate);
44
45
46 }
```

Listado A.5: Clase del dominio Reserva.java

```
1 package es.uv.hemal.elrincondeeva.domain;
2
3
4 import java.time.LocalDate;
5
6 import org.springframework.data.annotation.Id;
7 import org.springframework.data.relational.core.mapping.Column;
8 import org.springframework.data.relational.core.mapping.Table;
9
10
11
12 @Table("reservas")
13 public class Reserva {
14
15     @Id
16     private Integer id;
17
18     @Column("nombre")
19     private String nombre;
20     @Column("email")
21     private String email;
22     @Column("dni")
23     private String dni;
24     @Column("telefono")
25     private String telefono;
26
27     @Column("fecha_inicio")
28     private LocalDate fechaInicio;
29
30     @Column("fecha_fin")
31     private LocalDate fechaFin;
32
33     @Column("num_personas")
34     private int numPersonas;
35
36     @Column("precio_total")
37     private Double precioTotal;
38
39     @Column("estado")
40     private EstadoReserva estado;
41
42     @Column("usuario_id")
43     private Integer usuarioId;
44
45     // Constructor, Getters y Setters
46
47     public Reserva() {}
48
49
50     public Reserva(Integer id, String nombre, String email, String dni, String telefono, LocalDate
51         fechaInicio,
52         LocalDate fechaFin, int numPersonas, Double precioTotal, EstadoReserva estado, Integer
53         usuarioId) {
54         this.id = id;
55         this.nombre = nombre;
56         this.email = email;
57         this.dni = dni;
58         this.telefono = telefono;
59         this.fechaInicio = fechaInicio;
60         this.fechaFin = fechaFin;
61         this.numPersonas = numPersonas;
```

```
60     this.precioTotal = precioTotal;
61     this.estado = estado;
62     this.usuarioid = usuarioid;
63 }
64
65
66     public String getNombre() {
67         return nombre;
68     }
69
70
71     public void setNombre(String nombre) {
72         this.nombre = nombre;
73     }
74
75
76     public String getEmail() {
77         return email;
78     }
79
80
81     public void setEmail(String email) {
82         this.email = email;
83     }
84
85
86     public String getDni() {
87         return dni;
88     }
89
90
91     public void setDni(String dni) {
92         this.dni = dni;
93     }
94
95
96     public String getTelefono() {
97         return telefono;
98     }
99
100
101    public void setTelefono(String telefono) {
102        this.telefono = telefono;
103    }
104
105
106    public Integer getId() {
107        return id;
108    }
109
110    public void setId(Integer id) {
111        this.id = id;
112    }
113
114    public LocalDate getFechaInicio() {
115        return fechaInicio;
116    }
117
118    public void setFechaInicio(LocalDate fechaInicio) {
119        this.fechaInicio = fechaInicio;
120    }
121
122    public LocalDate getFechaFin() {
123        return fechaFin;
124    }
125
126    public void setFechaFin(LocalDate fechaFin) {
127        this.fechaFin = fechaFin;
128    }
129
130    public int getNumPersonas() {
131        return numPersonas;
132    }
133
134    public void setNumPersonas(int numPersonas) {
```

```

135     this.numPersonas = numPersonas;
136 }
137
138     public Double getPrecioTotal() {
139         return precioTotal;
140     }
141
142     public void setPrecioTotal(Double precioTotal) {
143         this.precioTotal = precioTotal;
144     }
145
146     public EstadoReserva getEstado() {
147         return estado;
148     }
149
150     public void setEstado(EstadoReserva estado) {
151         this.estado = estado;
152     }
153
154     public Integer getUserId() {
155         return usuarioId;
156     }
157
158     public void setUserId(Integer userId) {
159         this.usuarioId = userId;
160     }
161
162
163     public enum EstadoReserva {
164         CONFIRMADA,
165         PAGADA,
166         CANCELADA,
167         FINALIZADA
168     }
169 }
```

Listado A.6: Repositorio de reseñas ReviewRepository.java

```

1 package es.uv.hemal.elrincondeeva.repositories;
2
3
4 import org.springframework.data.r2dbc.repository.R2dbcRepository;
5 import org.springframework.stereotype.Repository;
6
7 import es.uv.hemal.elrincondeeva.domain.Review;
8 import reactor.core.publisher.Flux;
9
10
11 @Repository
12 public interface ReviewRepository extends R2dbcRepository<Review, Integer> {
13     Flux<Review> findAllByOrderByCreationDateDesc();
14 }
15
```

Listado A.7: Clase del dominio Review.java

```

1 package es.uv.hemal.elrincondeeva.domain;
2
3 import java.time.LocalDateTime;
4
5 import javax.validation.constraints.NotNull;
6
7 import org.springframework.data.annotation.Id;
8 import org.springframework.data.relational.core.mapping.Column;
9 import org.springframework.data.relational.core.mapping.Table;
10
11 @Table("reviews")
12 public class Review {
13
14     @Id
15     private Long id;
```

```
16
17     @Column
18     @NotNull(message = "El nombre no puede ser nulo")
19     private String review;
20
21     @Column()
22     @NotNull(message = "La puntuación no puede ser nulo")
23     private int rate;
24
25     @Column("rate_servicios")
26     @NotNull(message = "La puntuación no puede ser nulo")
27     private int rateServicios;
28     @Column("rate_limpieza")
29     @NotNull(message = "La puntuación no puede ser nulo")
30     private int rateLimpieza;
31     @Column("rate_ubicacion")
32     @NotNull(message = "La puntuación no puede ser nulo")
33     private int rateUbicacion;
34
35
36
37     public int getRateServicios() {
38         return rateServicios;
39     }
40
41     public void setRateServicios(int rateServicios) {
42         this.rateServicios = rateServicios;
43     }
44
45     public int getRateLimpieza() {
46         return rateLimpieza;
47     }
48
49     public void setRateLimpieza(int rateLimpieza) {
50         this.rateLimpieza = rateLimpieza;
51     }
52
53     public int getRateUbicacion() {
54         return rateUbicacion;
55     }
56
57     public void setRateUbicacion(int rateUbicacion) {
58         this.rateUbicacion = rateUbicacion;
59     }
60
61     @Column("creation_date")
62     @NotNull(message = "La fecha de creación no puede ser nulo")
63     private LocalDateTime creationDate;
64
65     @Column("user_id")
66     @NotNull(message = "El usuario no puede ser nulo")
67     private Integer userId;
68
69     // Getters y Setters
70     public Long getId() {
71         return id;
72     }
73
74     public void setId(Long id) {
75         this.id = id;
76     }
77
78     public String getReview() {
79         return review;
80     }
81
82     public void setReview(String review) {
83         this.review = review;
84     }
85
86     public int getRate() {
87         return rate;
88     }
89
90     public void setRate(int rate) {
```

```

91     this.rate = rate;
92 }
93
94     public LocalDateTime getCreationDate() {
95         return creationDate;
96     }
97
98     public void setCreationDate(LocalDateTime creationDate) {
99         this.creationDate = creationDate;
100    }
101
102    public Integer getUserId() {
103        return userId;
104    }
105
106    public void setUserId(Integer userId) {
107        this.userId = userId;
108    }
109
110 }
```

Listado A.8: Repositorio de usuarios UserRepository.java

```

1 package es.uv.hemal.elrincondeeva.repositories;
2
3
4 import org.springframework.data.r2dbc.repository.Query;
5 import org.springframework.data.r2dbc.repository.R2dbcRepository;
6 import org.springframework.data.repository.query.Param;
7 import org.springframework.stereotype.Repository;
8
9 import es.uv.hemal.elrincondeeva.domain.MyUser;
10 import es.uv.hemal.elrincondeeva.domain.Role;
11 import reactor.core.publisher.Flux;
12 import reactor.core.publisher.Mono;
13
14 @Repository
15 public interface UserRepository extends R2dbcRepository<MyUser, Integer> {
16
17     Mono<MyUser> findByEmail(String username);
18
19     @Query("SELECT r.* FROM roles r INNER JOIN user_role ur ON r.id = ur.role_id WHERE ur.user_id = :userId")
20     Flux<Role> findRolesByUserId(Integer userId);
21
22     @Query("SELECT r.* FROM roles r WHERE r.name = :name")
23     Mono<Role> findFirstByName(@Param("name") String name);
24
25     @Query("SELECT id FROM users WHERE username = :username")
26     Mono<Integer> findByUsername(@Param("username") String username);
27
28 }
```

Listado A.9: Repositorio de roles UserRepository.java

```

1 package es.uv.hemal.elrincondeeva.repositories;
2
3 import org.springframework.data.r2dbc.repository.R2dbcRepository;
4 import org.springframework.stereotype.Repository;
5
6 import es.uv.hemal.elrincondeeva.domain.Role;
7 import reactor.core.publisher.Mono;
8
9 @Repository
10 public interface RoleRepository extends R2dbcRepository<Role, Integer> {
11     Mono<Role> findByName(String name);
12     Mono<String> findNameById(Integer id);
13 }
```

Listado A.10: Repositorio de relacion roles y usuarios UserRepository.java

```

1 package es.uv.hemal.elrincondeeva.repositories;
2
3
4 import org.springframework.data.r2dbc.repository.R2dbcRepository;
5
6 import org.springframework.stereotype.Repository;
7
8 import es.uv.hemal.elrincondeeva.domain.UserRole;
9 import reactor.core.publisher.Flux;
10 import reactor.core.publisher.Mono;
11
12
13 @Repository
14 public interface RoleUserRepository extends R2dbcRepository<UserRole, Integer> {
15     Flux<UserRole> findByUserId(Integer userId);
16     Flux<UserRole> findByRoleId(Integer roleId);
17     Mono<Void> deleteByUserIdAndRoleId(Integer userId, Integer roleId);
18 }
```

Listado A.11: Clase del dominio MyUser.java

```

1 package es.uv.hemal.elrincondeeva.domain;
2
3 import org.springframework.data.annotation.Id;
4 import org.springframework.data.annotation.Transient;
5 import org.springframework.data.relational.core.mapping.Column;
6 import org.springframework.data.relational.core.mapping.Table;
7
8 import javax.validation.constraints.Email;
9 import javax.validation.constraints.NotNull;
10 import javax.validation.constraints.Pattern;
11 import javax.validation.constraints.Size;
12 import java.time.LocalDate;
13
14 import java.util.List;
15
16 @Table("users")
17
18
19
20 public class MyUser {
21     @Id
22     private Integer id;
23
24     @Column("nombre")
25     @NotNull(message = "El nombre no puede ser nulo")
26     @Size(min = 1, message = "El nombre debe tener al menos 1 caracter")
27     private String nombre;
28
29
30     @Column("password")
31     @NotNull(message = "La contraseña no puede ser nula")
32     @Size(min = 6, message = "La contraseña debe tener al menos 6 caracteres")
33     private String password;
34
35     @Column("apellidos")
36     @NotNull(message = "Los apellidos no pueden ser nulos")
37     @Size(min = 1, message = "Los apellidos deben tener al menos 1 caracter")
38     private String apellidos;
39
40     @Column("email")
41     @NotNull(message = "El email no puede ser nulo")
42     @Email(message = "El email debe ser válido")
43     private String email;
44
45     @Column("telefono")
46     @NotNull(message = "El teléfono no puede ser nulo")
47     @Pattern(regexp = "[0-9]{9}", message = "El teléfono debe tener 9 dígitos")
48     private String telefono;
49
50
51     @Column("dni")
52     @NotNull(message = "El DNI no puede ser nulo")
```

```
53     @Pattern(regexp = "^\d{8}[A-Z-a-z]$", message = "El DNI debe tener 8 dígitos seguidos de una letra")
54     private String dni;
55
56     @Column("fecha_nacimiento")
57     @NotNull(message = "La fecha de nacimiento no puede ser nula")
58     private LocalDate fechaNacimiento;
59
60     @Column("direccion")
61     @NotNull(message = "La dirección no puede ser nula")
62     @Size(min = 10, message = "La dirección debe tener al menos 10 caracteres")
63     private String direccion;
64         @Column("estado")
65     @NotNull(message = "El estado no puede ser nula")
66     @Size(min = 10, message = "La dirección debe tener al menos 10 caracteres")
67     private Boolean estado;
68     @Column("fecha_registro")
69     @NotNull(message = "La fecha de registro no puede ser nula")
70     private LocalDate fechaRegistro;
71     @Column("fecha_modificacion")
72     @NotNull(message = "La fecha de modificación no puede ser nula")
73     private LocalDate fechaModificacion;
74     @Column("fecha_baja")
75     private LocalDate fechaBaja;
76
77     @Transient
78     private List<Role> roles;
79
80     public MyUser() {
81     }
82     public MyUser(
83         @NotNull(message = "El nombre no puede ser nulo") @Size(min = 1, message = "El
84             nombre debe tener al menos 1 carácter") String nombre,
85         @NotNull(message = "El nombre de usuario no puede ser nulo") @Size(min = 4,
86             message = "El nombre de usuario debe tener al menos 4 caracteres") String
87             username,
88         @NotNull(message = "La contraseña no puede ser nula") @Size(min = 6, message =
89             "La contraseña debe tener al menos 6 caracteres") String password,
90         @NotNull(message = "Los apellidos no pueden ser nulos") @Size(min = 1, message =
91             "Los apellidos deben tener al menos 1 carácter") String apellidos,
92         @NotNull(message = "El email no puede ser nulo") @Email(message = "El email debe
93             ser válido") String email,
94         @NotNull(message = "El teléfono no puede ser nulo") @Pattern(regexp =
95             "^[0-9]{9}$", message = "El teléfono debe tener 9 dígitos") String telefono,
96         @NotNull(message = "El DNI no puede ser nulo") @Pattern(regexp =
97             "^\d{8}[A-Z-a-z]$", message = "El DNI debe tener 8 dígitos seguidos de una
98             letra") String dni,
99         @NotNull(message = "La fecha de nacimiento no puede ser nula") LocalDate
100            fechaNacimiento,
101         @NotNull(message = "La dirección no puede ser nula") @Size(min = 10, message =
102             "La dirección debe tener al menos 10 caracteres") String direccion,
103         @NotNull(message = "El estado no puede ser nula") @Size(min = 10, message = "La
104             dirección debe tener al menos 10 caracteres") Boolean estado,
105         @NotNull(message = "La fecha de registro no puede ser nula") LocalDate
106            fechaRegistro,
107         @NotNull(message = "La fecha de modificación no puede ser nula") LocalDate
108            fechaModificacion
109     ) {
110         this.nombre = nombre;
111         this.password = password;
112         this.apellidos = apellidos;
113         this.email = email;
114         this.telefono = telefono;
115         this.dni = dni;
116         this.fechaNacimiento = fechaNacimiento;
117         this.direccion = direccion;
118         this.estado = estado;
119         this.fechaRegistro = fechaRegistro;
120         this.fechaModificacion = fechaModificacion;
121         this.fechaBaja = fechaBaja;
122         this.roles = roles;
123     }
124     public Integer getId() {
125         return id;
126     }
```

```
114 |     public void setId(Integer id) {
115 |         this.id = id;
116 |     }
117 |     public String getNombre() {
118 |         return nombre;
119 |     }
120 |     public void setNombre(String nombre) {
121 |         this.nombre = nombre;
122 |     }
123 |     public String getPassword() {
124 |         return password;
125 |     }
126 |     public void setPassword(String password) {
127 |         this.password = password;
128 |     }
129 |     public String getApellidos() {
130 |         return apellidos;
131 |     }
132 |     public void setApellidos(String apellidos) {
133 |         this.apellidos = apellidos;
134 |     }
135 |     public String getEmail() {
136 |         return email;
137 |     }
138 |     public void setEmail(String email) {
139 |         this.email = email;
140 |     }
141 |     public String getTelefono() {
142 |         return telefono;
143 |     }
144 |     public void setTelefono(String telefono) {
145 |         this.telefono = telefono;
146 |     }
147 |     public String getDni() {
148 |         return dni;
149 |     }
150 |     public void setDni(String dni) {
151 |         this.dni = dni;
152 |     }
153 |     public LocalDate getFechaNacimiento() {
154 |         return fechaNacimiento;
155 |     }
156 |     public void setFechaNacimiento(LocalDate fechaNacimiento) {
157 |         this.fechaNacimiento = fechaNacimiento;
158 |     }
159 |     public String getDireccion() {
160 |         return direccion;
161 |     }
162 |     public void setDireccion(String direccion) {
163 |         this.direccion = direccion;
164 |     }
165 |     public Boolean getEstado() {
166 |         return estado;
167 |     }
168 |     public void setEstado(Boolean estado) {
169 |         this.estado = estado;
170 |     }
171 |     public LocalDate getFechaRegistro() {
172 |         return fechaRegistro;
173 |     }
174 |     public void setFechaRegistro(LocalDate fechaRegistro) {
175 |         this.fechaRegistro = fechaRegistro;
176 |     }
177 |     public LocalDate getFechaModificacion() {
178 |         return fechaModificacion;
179 |     }
180 |     public void setFechaModificacion(LocalDate fechaModificacion) {
181 |         this.fechaModificacion = fechaModificacion;
182 |     }
183 |     public LocalDate getFechaBaja() {
184 |         return fechaBaja;
185 |     }
186 |     public void setFechaBaja(LocalDate fechaBaja) {
187 |         this.fechaBaja = fechaBaja;
188 |     }
```

```
189     public List<Role> getRoles() {
190         return roles;
191     }
192     public void setRoles(List<Role> roles) {
193         this.roles = roles;
194     }
195
196
197
198
199 }
200 }
```

Listado A.12: Clase del dominio UserRole.java

```
1 package es.uv.hemal.elrincondeeva.domain;
2
3 import org.springframework.data.relational.core.mapping.Column;
4 import org.springframework.data.relational.core.mapping.Table;
5 import javax.validation.constraints.NotNull;
6
7 @Table("user_role")
8 public class UserRole {
9
10
11     @Column("user_id")
12     @NotNull
13     private Integer userId;
14
15     @Column("role_id")
16     @NotNull
17     private Integer roleId;
18
19
20
21
22     public Integer getUserId() {
23         return userId;
24     }
25
26     public void setUserId(Integer userId) {
27         this.userId = userId;
28     }
29
30     public Integer getRoleId() {
31         return roleId;
32     }
33
34     public void setRoleId(Integer roleId) {
35         this.roleId = roleId;
36     }
37
38 }
39 }
```

Listado A.13: Clase del dominio Role.java

```
1 package es.uv.hemal.elrincondeeva.domain;
2
3 import org.springframework.data.annotation.Id;
4 import org.springframework.data.relational.core.mapping.Column;
5 import org.springframework.data.relational.core.mapping.Table;
6
7 import java.util.List;
8
9 @Table("roles")
10 public class Role {
11     @Id
12     private Integer id;
13
14     @Column("name")
```

```

15     private String name;
16
17
18
19     public Integer getId() {
20         return id;
21     }
22
23     public void setId(Integer id) {
24         this.id = id;
25     }
26
27     public String getName() {
28         return name;
29     }
30
31     public void setName(String name) {
32         this.name = name;
33     }
34 }

```

A.4. Código fuente SpringBoot scraping en el subsistema de extracción

Listado A.14: Servicio de scraping de sitios web turísticos ScraperService.java

```

15 @Service
16 public class ScraperService {
17
18     @Autowired
19     private RecursoRepository recursoRepository;
20
21     @Autowired
22     private ReactiveMongoTemplate mongoTemplate;
23
24     public Mono<Long> scrape(String baseUrl, String tipo, String localizacion) {
25         System.out.println("Raspando: " + baseUrl + " para tipo: " + tipo + ", localización: " +
26             localizacion);
27
28         return ensureCollectionExists("recursos")
29             .then(Mono.fromCallable(() -> Jsoup.connect(baseUrl).get()))
30             .flatMapMany(document -> processByTypeAndLocation(document, tipo, localizacion))
31             .flatMap(recurso -> recursoRepository.save(recurso)
32                 .doOnSuccess(saved -> System.out.println("Guardado en MongoDB: " + saved))
33                 .doOnError(error -> System.err.println("Error al guardar: " +
34                     error.getMessage())))
35             .count()
36             .doOnNext(count -> System.out.println("Total de recursos procesados: " + count))
37             .doOnError(error -> System.err.println("Error en el proceso de scraping: " +
38                 error.getMessage()));
39     }
40
41     private Mono<Void> ensureCollectionExists(String collectionName) {
42         return mongoTemplate.collectionExists(collectionName)
43             .flatMap(exists -> exists
44                 ? Mono.empty()
45                 : mongoTemplate.createCollection(collectionName).then())
46             .doOnNext(unused -> System.out.println("Colección asegurada: " + collectionName));
47     }
48
49     private Flux<Recurso> processByTypeAndLocation(Document document, String tipo, String localizacion) {
50         if (("monumentos".equals(tipo) || "aldeas".equals(tipo)) && "chelva".equals(localizacion)) {
51             return processChelvaMonumentosAldeas(document, tipo, localizacion);
52         } else if (("patrimonio".equals(tipo) || "naturaleza".equals(tipo)) &&
53             "tuejar".equals(localizacion)) {
54             return processTuejarPatrimonioNaturaleza(document, tipo, localizacion);
55         } else if ("fiestas".equals(tipo) && "tuejar".equals(localizacion)) {
56

```

```
53     return processTuejarFiestas(document, tipo, localizacion);
54 }
55 System.out.println("Sin coincidencia para tipo: " + tipo + ", localización: " + localizacion);
56 return Flux.empty();
57 }

58 private Flux<Recurso> processChelvaMonumentosAldeas(Document document, String tipo, String
59 localizacion) {
60     return Flux.fromIterable(document.select("div.qodef-tours-destination-item-holder a"))
61         .map(imgLink -> {
62             String pageUrl = imgLink.attr("href");
63             String imageUrl = imgLink.select("div.qodef-tours-destination-item-image
64 img").attr("src");
65             String title = imgLink.select("h3.qodef-tours-destination-item-title").text();
66
67             if (pageUrl.isEmpty()) {
68                 System.err.println("URL de la página vacía: " + imgLink);
69                 return null;
70             }
71             return new Recurso(title, imageUrl, pageUrl, "", tipo, localizacion);
72         })
73         .filter(recurso -> recurso != null)
74         .doOnNext(recurso -> System.out.println("Procesado recurso: " + recurso));
75 }

76 private Flux<Recurso> processTuejarPatrimonioNaturaleza(Document document, String tipo, String
77 localizacion) {
78     return Flux.fromIterable(document.select("a[href] img[src]"))
79         .filter(imgLink -> imgLink.absUrl("src").contains(".jpg"))
80         .flatMap(imgLink -> {
81             String imageUrl = imgLink.absUrl("src");
82             String imageAlt = imgLink.attr("alt");
83             String linkUrl = imgLink.parent().absUrl("href");
84
85             if (imageUrl.isEmpty() || imageAlt.isEmpty() || linkUrl.isEmpty()) {
86                 System.err.println("Datos faltantes: img=" + imageUrl + ", alt=" + imageAlt + ",
87                     link=" + linkUrl);
88                 return Mono.empty();
89             }
90
91             return fetchLinkedPage(linkUrl)
92                 .map(description -> new Recurso(imageAlt, imageUrl, linkUrl, description,
93                     tipo, localizacion))
94                 .defaultIfEmpty(new Recurso(imageAlt, imageUrl, linkUrl, "", tipo,
95                     localizacion));
96         })
97         .doOnNext(recurso -> System.out.println("Procesado recurso: " + recurso));
98 }
99 private Flux<Recurso> processTuejarFiestas(Document document, String tipo, String localizacion) {
100     return Flux.fromIterable(document.select(".image-style-imagen-120-x-120"))
101         .flatMap(imgLink -> {
102             String imageUrl = imgLink.parent().absUrl("href");
103             String url = imgLink.parent().absUrl("href");
104             String linkUrl = imgLink.parent().absUrl("href");
105
106             String[] parts = url.split("/");
107             String imageAlt = parts[parts.length - 1];
108
109
110
111             if (imageUrl.isEmpty() || linkUrl.isEmpty()) {
112                 System.err.println("Datos faltantes: img=" + imageUrl + ", alt=" + imageAlt + ",
113                     link=" + linkUrl);
114                 return Mono.empty();
115             }
116
117             return fetchLinkedPage(linkUrl)
118                 .map(description -> new Recurso(imageAlt, imageUrl, linkUrl, description,
119                     tipo, localizacion))
```

```

119         .defaultIfEmpty(new Recurso(imageAlt, imageUrl, linkUrl, "", tipo,
120                         localizacion));
121     })
122     .doOnNext(recurso -> System.out.println("Procesado recurso: " + recurso));
123 }
124
125 private Mono<String> fetchLinkedPage(String linkUrl) {
126     return Mono.fromCallable(() -> Jsoup.connect(linkUrl).get())
127         .map(linkedPage -> {
128             Elements titleElements =
129                 linkedPage.selectXpath("//*[@id='cs-content']/div[1]/div/div/div[1]/div[2]");
130             return titleElements.isEmpty() ? "" : titleElements.get(0).text();
131         })
132         .doOnError(e -> System.err.println("Error accediendo a: " + linkUrl))
133         .onErrorResume(e -> Mono.empty());
134 }

```

A.5. Código fuente SpringBoot gestión de token de Instagram en subsistema de publicaciones

Listado A.15: Servicio encargado de refrescar el token de acceso a la API de Instagram `InstagramTokenService.java`

```

6 import com.fasterxml.jackson.annotation.JsonProperty;
7
8 import java.io.IOException;
9 import java.nio.file.Files;
10 import java.nio.file.Path;
11 import java.nio.file.Paths;
12
13 @Service
14 public class InstagramTokenService {
15
16     private final WebClient webClient = WebClient.create("https://graph.instagram.com");
17
18
19     @Value("${instagram.token-file-path}")
20     private String tokenFilePath;
21
22     public String getToken() throws IOException {
23
24         Path path = Paths.get(System.getProperty("user.dir"), tokenFilePath, "token.txt");
25
26         return Files.readAllLines(path).get(0).trim();
27     }
28
29     public void saveToken(String newToken) throws IOException {

```

A.6. Código TypeScript del frontend de la aplicación

Listado A.16: Código del componente `AdministradorComponent`

```

1 import { Component, OnInit, ViewEncapsulation } from '@angular/core';
2 import { MatTableDataSource } from '@angular/material/table';
3 import { MatDialog } from '@angular/material/dialog';
4 import { DialogMotivoComponent } from '../dialog-motivo/dialog-motivo.component';
5 import { ReservaService } from '../services/reserva.service';
6 import { MatSnackBar } from '@angular/material/snack-bar';
7 import { HttpErrorResponse } from '@angular/common/http';
8 import { Reserva } from '../compartido/Reserva';

```

```
9
10
11 @Component({
12   selector: 'app-administrador',
13   templateUrl: './administrador.component.html',
14   styleUrls: ['./administrador.component.scss']
15 })
16 export class AdministradorComponent implements OnInit {
17   displayedColumns: string[] = ['nombre', 'numeroPersonas', 'estado', 'fechas', 'acciones'];
18   dataSource = new MatTableDataSource<Reserva>();
19   finalizadas = new MatTableDataSource<Reserva>(); // Para reservas finalizadas
20   reservasOk: Reserva[] = []; // Para almacenar las reservas filtradas
21   reservasFinalizadas: Reserva[] = []; // Para almacenar las reservas finalizadas
22   filtros = {
23     nombre: '',
24     dni: '',
25     estado: ''
26   };
27
28   constructor(
29     private reservaService: ReservaService,
30     private dialog: MatDialog,
31     private snackBar: MatSnackBar
32   ) {}
33
34   ngOnInit(): void {
35     this.dataSource.filterPredicate = (reserva: Reserva, filtroStr: string): boolean => {
36       const f = JSON.parse(filtroStr);
37
38       const coincideNombre = !f.nombre || reserva.nombre?.toLowerCase().includes(f.nombre.toLowerCase());
39       const coincideDni = !f.dni || reserva.dni?.toLowerCase().includes(f.dni.toLowerCase());
40
41       const coincideEstado = !f.estado || reserva.estado === f.estado;
42       return coincideNombre && coincideDni && coincideEstado;
43     };
44
45     this.cargarReservas();
46   }
47
48   applyFilter(): void {
49     const filtroStr = JSON.stringify(this.filtros);
50     this.dataSource.filter = filtroStr;
51   }
52
53
54   cargarReservas(): void {
55
56
57   this.reservaService.obtenerReservas().subscribe(reservas => {
58     this.reservasOk = reservas.filter(reserva => {
59       return reserva.estado !== 'FINALIZADA' ;
60     });
61
62     this.dataSource.data = this.reservasOk;
63     this.reservasFinalizadas = reservas.filter(reserva => reserva.estado === 'FINALIZADA');
64     this.finalizadas.data = this.reservasFinalizadas;
65   });
66
67
68 }
69
70   aprobarPagoReserva(reserva: Reserva): void {
71
72     const dialogRef = this.dialog.open(DialogMotivoComponent, {
73       width: '300px',
74       data: { reservaInfo: reserva, title: 'Aprobar pago de la reserva', placeholder: 'Mensaje de confirmación', motivo: '' }
75     });
76
77     dialogRef.afterClosed().subscribe(result => {
78       if (result) {
79         if (result.motivo && result.motivo.trim() !== '') {
80           console.log(result.reservaInfo);
81           this.reservaService.pagarReserva(result.reservaInfo, result.motivo).subscribe({
82             next: (response: Reserva) => {
```

```
83         this.snackBar.open('Reserva confirmada con éxito!', 'Cerrar', {
84             duration: 3000,
85         });
86         this.cargarReservas();
87     },
88     error: (error: HttpErrorResponse) => {
89         this.snackBar.open('Error al confirmar la reserva. Inténtalo de nuevo.', 'Cerrar', {
90             duration: 3000,
91         });
92     },
93 });
94 } else {
95     this.snackBar.open('El motivo no puede estar vacío.', 'Cerrar', {
96         duration: 3000,
97         verticalPosition: 'top',
98         horizontalPosition: 'center',
99     });
100 }
101 } else {
102     this.snackBar.open('No se proporcionó motivo. Operación cancelada.', 'Cerrar', {
103         duration: 3000,
104         verticalPosition: 'top',
105         horizontalPosition: 'center',
106     });
107 }
108 );
109 }
110
111 cancelarReserva(reserva: Reserva): void {
112     const dialogRef = this.dialog.open(DialogMotivoComponent, {
113         width: '300px',
114         data: { reservaInfo: reserva, title: 'Denegar Reserva', placeholder: 'Motivo de rechazo', motivo: '' }
115     });
116
117 dialogRef.afterClosed().subscribe(result => {
118     if (result) {
119         if (result.motivo && result.motivo.trim() !== '') {
120             this.reservaService.cancelarReserva(result.reservaInfo, result.motivo).subscribe({
121                 next: (response: Reserva) => {
122
123
124                     this.snackBar.open('Reserva cancelada con éxito!', 'Cerrar', {
125                         duration: 3000,
126                     });
127                     this.cargarReservas();
128
129
130                 },
131                 error: (error: HttpErrorResponse) => {
132
133
134                     this.snackBar.open('Error al cancelar la reserva. Inténtalo de nuevo.', 'Cerrar', {
135                         duration: 3000,
136                     });
137                 },
138             });
139         }
140     }
141     } else {
142         this.snackBar.open('El motivo no puede estar vacío.', 'Cerrar', {
143             duration: 3000,
144             verticalPosition: 'top',
145             horizontalPosition: 'center',
146         });
147     }
148 }
149 } else {
150     this.snackBar.open('No se proporcionó motivo. Operación cancelada.', 'Cerrar', {
151         duration: 3000,
152         verticalPosition: 'top',
153         horizontalPosition: 'center',
154     });
155 }
156 );
```

```
157 |     }
158 | }
```

Listado A.17: Método onSubmitReview en el componente InicioComponent

```
1 import { Component, OnInit, ViewEncapsulation } from '@angular/core';
2 import { FormBuilder, FormGroup, Validators } from '@angular/forms';
3 import { ReviewService } from '../services/review.service';
4 import { MailService } from '../services/mail.service';
5 import { MatSnackBar } from '@angular/material/snack-bar';
6 import { AuthService } from '../services/auth.service';
7 import { switchMap } from 'rxjs';
8 @Component({
9   selector: 'app-inicio',
10  templateUrl: './inicio.component.html',
11  styleUrls: ['./inicio.component.scss']
12})
13 export class InicioComponent implements OnInit {
14  contactForm: FormGroup;
15  reviewForm: FormGroup;
16  reviews: any[] = [];
17  expandedIndex: number | null = null;
18  isClientRole: boolean = false;
19  visibleReviewsCount = 3;
20  constructor(private fb: FormBuilder, private reviewService: ReviewService, private mailService: MailService, private authService: AuthService,
21    private snackBar: MatSnackBar)
22  {
23    // Formulario de contacto
24    this.contactForm = this.fb.group({
25      name: ['', [Validators.required]],
26      email: ['', [Validators.required, Validators.email]],
27      message: ['', [Validators.required]],
28    });
29
30    // Formulario de reseñas
31    this.reviewForm = this.fb.group({
32      review: ['', [Validators.required, Validators.minLength(10)]],
33      rate: [0, [Validators.required, Validators.min(1), Validators.max(5)]],
34      rateUbicacion: [0, [Validators.required, Validators.min(1), Validators.max(5)]],
35      rateServicios: [0, [Validators.required, Validators.min(1), Validators.max(5)]],
36      rateLimpieza: [0, [Validators.required, Validators.min(1), Validators.max(5)]],
37    });
38  }
39
40  ngOnInit() {
41    this.getReviews();
42    this.checkUserRole();
43  }
44
45  checkUserRole() {
46    const userRolesString = sessionStorage.getItem('Roles')?.replace(/\[\]/g, '') || '';
47    const userRoles = userRolesString.split(',').map(role => role.trim());
48    this.isClientRole = userRoles.includes('ROLE_CLIENTE');
49  }
50
51  getReviews() {
52    this.reviewService.fetchReviews().subscribe({
53      next: (reviews: any[]) => {
54        this.reviews = reviews.map(review => {
55          if (review.creationDate && Array.isArray(review.creationDate)) {
56            const [year, month, day] = review.creationDate;
57            const creationDate = new Date(year, month - 1, day, 0, 0, 0);
58            if (!isNaN(creationDate.getTime())) {
59              const opciones: Intl.DateTimeFormatOptions = {
60                day: 'numeric',
61                month: 'long',
62                year: 'numeric',
63              };
64              review.creationDate = creationDate.toLocaleDateString('es-ES', opciones);
65            } else {
66              console.warn('Fecha no válida:', review.creationDate);
67            }
68          }
69        });
70      }
71    });
72  }
73}
```

```

68         review.creationDate = 'Fecha no válida';
69     }
70   } else {
71     review.creationDate = 'Fecha no disponible';
72   }
73   return review;
74 });
75 },
76 error: (error: any) => console.error('Error al cargar reseñas:', error),
77 });
78 }
79
80 toggleExpanded(index: number): void {
81   this.expandedIndex = this.expandedIndex === index ? null : index;
82 }
83
84 getStars(rate: number): ('filled' | 'empty')[] {
85   return Array.from({ length: 5 }, (_, i) => (i < rate ? 'filled' : 'empty'));
86 }
87 loadMoreReviews(): void {
88
89   this.visibleReviewsCount = Math.min(this.visibleReviewsCount + 3, this.reviews.length);
90 }
91 hideReviews(): void {
92
93   this.visibleReviewsCount = 3;
94 }
95 onSubmitReview() {
96
97   if (this.reviewForm.valid) {
98     this.reviewService.addReview(this.reviewForm.value).pipe(
99       switchMap((response) => {
100         console.log('Reseña enviada correctamente:', response);
101         this.snackBar.open('Reseña enviada correctamente', 'Cerrar');
102         this.reviewForm.reset();
103         return this.authService.checkRoles();
104       })
105     ).subscribe({
106       next: () => {
107         console.log('Sesión actualizada correctamente');
108         this.checkUserRole();
109         this.getReviews();
110       },
111       error: (error) => console.error('Error al procesar la reseña o actualizar la sesión:', error)
112     });
113   } else {
114     console.warn('Formulario de reseñas no válido');
115   }
116 }
117 }
118
119 onSubmit(form: FormGroup) {
120   if (form.valid) {
121     this.mailService.addContact(this.contactForm.value).subscribe({
122       next: () => {
123         console.log('Contacto enviado correctamente');
124         this.contactForm.reset();
125       },
126       error: (error) => console.error('Error al enviar el formulario:', error),
127     });
128   } else {
129     console.warn('Formulario de contacto no válido');
130   }
131 }
132 }
133 }

```

Listado A.18: Lógica del componente LaCasaComponent

```

1 import { Component, ViewEncapsulation } from '@angular/core';
2 import { Media } from '../compartido/media';
3 import { Inject } from '@angular/core';
4 import { MediaService } from '../services/media.service';

```

```
5  @Component({
6    selector: 'app-la-casa',
7
8    templateUrl: './la-casa.component.html',
9    styleUrls: ['./la-casa.component.scss'
10  })
11 export class LaCasaComponent {
12
13   slides : Media[] = [];
14   videoUrl: string | null = null;
15
16   currentSlide: number =0;
17   imagesPerSlide: number =2;
18
19   constructor( @Inject('baseURL') public baseURL: string, private mediaService: MediaService) {}
20
21   ngOnInit() {
22     this.videoUrl = this.BaseURL + 'video';
23     //this.fetchMediaByHashtag('habitacionprincipal');
24     //this.fetchMediaByHashtag('jardin');
25     this.fetchMedia();
26   }
27   fetchMedia() {
28     this.mediaService.getMedia().subscribe({
29       next: (data: Media[]) => {
30         const filtrados = data.filter(media => media.hashtag !== 'gastronomia');
31         this.slides.push(...filtrados);
32       },
33       error: (err: any) => {
34         console.error('Error al obtener media:', err);
35       }
36     });
37   }
38   fetchMediaByHashtag(hashtag: string): void {
39     this.mediaService.getMediaByHashtag(hashtag).subscribe({
40       next: (data: Media[]) => {
41
42         this.slides.push(...data);
43       },
44       error: (err: any) => {
45         console.error('Error al obtener media:', err);
46       }
47     });
48   }
49
50
51   prevSlide() {
52     this.currentSlide =
53       this.currentSlide > 0
54       ? this.currentSlide - this.imagesPerSlide
55       : Math.max(this.slides.length - this.imagesPerSlide, 0);
56   }
57
58   nextSlide() {
59     this.currentSlide =
60       this.currentSlide <
61       this.currentSlide <
62       this.slides.length - this.imagesPerSlide
63       ? this.currentSlide + this.imagesPerSlide
64       : 0;
65   }
66
67   getTransformStyle(): string {
68     const percent = this.currentSlide * (100 / this.imagesPerSlide);
69     return `translateX(-${percent}%)`;
70   }
71
72   getFlexStyle(): string {
73     return `0 0 ${100 / this.imagesPerSlide}%`;
74   }
75 }
```

Listado A.19: Lógica del componente ReservaComponent

```
1 import {
2   Component,
3   ChangeDetectorRef,
4   OnInit,
5   ChangeDetectionStrategy,
6   TemplateRef,
7   ViewChild,
8   LOCALE_ID,
9 } from '@angular/core';
10 import { FormBuilder, FormGroup, Validators } from '@angular/forms';
11 import { CalendarDateFormatter, CalendarEvent, CalendarMonthViewBeforeRenderEvent, CalendarMonthViewDay, CalendarView, CalendarWeekViewBeforeRenderEvent } from 'angular-calendar';
12 import {
13   addDays,
14   addMonths,
15   addWeeks,
16   addYears,
17   endOfDay,
18   format,
19   isAfter,
20   isBefore,
21   isEqual,
22   startOfDay,
23   subDays,
24   subMonths,
25 } from 'date-fns';
26
27 import { MyUser } from '../compartido/MyUser';
28 import { ReservaService } from '../services/reserva.service';
29 import { MatDialog } from '@angular/material/dialog';
30 import { ResumenReservaComponent } from '../resumen-reserva/resumen-reserva.component';
31 import { Reserva } from '../compartido/Reserva';
32 import { MatSnackBar } from '@angular/material/snack-bar';
33 import { CustomDateFormatter } from '../reservas/custom-date-formatter.provider';
34 import { MatPaginator } from '@angular/material/paginator';
35 import { WeatherService, WeatherWithIcon } from '../services/weather.service';
36 @Component({
37   selector: 'app-reservas',
38   templateUrl: './reservas.component.html',
39   changeDetection: ChangeDetectionStrategy.Default,
40   providers: [
41     {
42       provide: CalendarDateFormatter,
43       useClass: CustomDateFormatter,
44     },
45   ],
46   styleUrls: ['./reservas.component.scss'],
47 })
48 export class ReservasComponent implements OnInit {
49
50   view: CalendarView = CalendarView.Month;
51   viewDate: Date = new Date();
52   events: CalendarEvent[] = [];
53   actualEvent: CalendarEvent[] = [];
54   nombre!: string;
55   weatherData: WeatherWithIcon[] = [];
56   aviso!: string;
57
58   selectedDays: Set<Date> = new Set();
59   teléfono!: string;
60   dni!: string;
61   apellidos!: string;
62   email ='';
63   reservedDates: { start: Date; end: Date }[] = [];
64   startDate: Date | null = null;
65   endDate: Date | null = null;
66   @ViewChild('dayEventsTemplate', { static: true })
67   dayEventsTemplate!: TemplateRef<any>;
68   selectedDay: any;
69   today!: Date;
70   currentMonthYear: string = '';
71   contactForm: FormGroup;
72   reservas: Reserva[] = [];
73   displayedColumns: string[] = ['id', 'fecha', 'numPersonas', 'estado'];
```

```
75  minDate!: Date;
76  maxDate!: Date;
77
78
79
80  user!: MyUser;
81  constructor(private dialog: MatDialog, private cdr: ChangeDetectorRef, private fb: FormBuilder, private
82    reservaService: ReservaService, private snackBar: MatSnackBar, private weatherService: WeatherService)
83  {
84    this.contactForm = this.fb.group({
85      name: ['', [Validators.required]],
86      email: ['', [Validators.required, Validators.email]],
87      emailUsuario: ['', []],
88      telefono: ['', [Validators.required, Validators.pattern(/^[0-9]{9}$/)]],
89      dni: ['', [Validators.required, Validators.pattern(/^\\d{8}[A-Za-z]$/)]],
90      numPersonas: ['', [Validators.required, Validators.min(1), Validators.max(7)]],
91      startDate: ['', []],
92      endDate: ['', []],
93      precio: ['', []],
94
95    });
96  }
97  seleccionado!: boolean;
98  ngOnInit(): void {
99    if(sessionStorage.getItem('Email') === null){
100      this.email='';
101    }
102    else{
103      this.email = sessionStorage.getItem('Email')!;
104    }
105    if (this.email) {
106      this.today = startOfDay(new Date());
107      this.initializeEvents();
108      this.updateMonthYear();
109
110      this.reservaService.getReservasUser(this.email).subscribe({
111        next: (data:Reserva[]) => this.reservas = data,
112        error: (err: any) => console.error('Error fetching reservas', err)
113      });
114    }
115    const today = new Date();
116
117    this.minDate = new Date(today.setDate(today.getDate() + 7));
118    this.maxDate = new Date(today.setFullYear(today.getFullYear() + 1));
119  }
120  initializeEvents(): void {
121
122    this.reservaService.getReservedDates().subscribe((dates: { start: Date; end: Date }[]) => {
123      this.reservedDates = dates;
124
125      this.events = [
126        ...this.events,
127        ...this.reservedDates.map((dateRange) => ({
128          start: dateRange.start,
129          end: dateRange.end,
130          title: `Reservado: ${format(dateRange.start, 'dd/MM/yyyy')} - ${format(
131            dateRange.end,
132            'dd/MM/yyyy'
133          )}`,
134          color: { primary: '#d9534f', secondary: '#d9534f' },
135          allDay: true,
136          draggable: false,
137          resizable: { beforeStart: false, afterEnd: false },
138        })),
139      ];
140    });
141    this.cdr.detectChanges();
142  }
143
144
145  dateIsValid(date: Date): boolean {
146    return date >= this.minDate && date <= this.maxDate;
```

```
148     }
149
150
151     updateMonthYear() {
152         this.currentMonthYear = format(this.viewDate, 'MM yyyy');
153         this.cdr.detectChanges();
154     }
155
156     onDayClicked({ day }: { day: any }): void {
157         this.seleccionado = false;
158         this.selectedDay = this.selectedDay === day.date ? null : day.date;
159         this.aviso = '';
160         // Si no se ha seleccionado un inicio, lo asignamos
161         if (!this.startDate) {
162             this.aviso = 'Selecciona la fecha final para tu reserva';
163             this.startDate = day.date;
164             this.selectedDays.add(this.startDate!); // Marcamos el día como seleccionado
165         } else if (!this.endDate) {
166             // Si el final está vacío, validamos y asignamos la fecha final
167             if (day.date >= this.startDate) {
168                 this.endDate = day.date;
169                 this.addEvent();
170                 this.selectedDays.add(this.endDate!); // Marcamos también el fin como seleccionado
171                 this.fetchWeather();
172                 this.aviso = 'Haz Click sobre el circulo marrón para continuar con la reserva';
173             } else {
174                 this.snackBar.open('La fecha final debe ser posterior a la fecha de inicio.', 'Cerrar', {
175                     duration: 3000,
176                 });
177             }
178         } else {
179             // Si ya se tiene un rango, reiniciamos la selección
180             this.removeEvent(this.startDate, this.endDate);
181             this.startDate = day.date;
182             this.endDate = null;
183             this.aviso = 'Selecciona la fecha final de tu reserva';
184             this.selectedDays.clear(); // Limpiamos la selección
185             this.selectedDays.add(this.startDate!); // Volvemos a seleccionar el nuevo inicio
186         }
187     }
188
189     this.cdr.detectChanges();
190 }
191
192     addEvent(): void {
193         const today = startOfDay(new Date());
194         const initOfRange = addWeeks(today, 1);
195         const endOfRange = addWeeks(addYears(today, 1), 1);
196
197         if (
198             this.startDate &&
199             this.endDate &&
200             this.startDate <= this.endDate &&
201             this.startDate > today &&
202             this.startDate >= initOfRange &&
203             this.endDate <= endOfRange
204         ) {
205
206
207             const isOverlapping = this.reservedDates.some(
208                 (reserved) =>
209                     // Caso 1: Empieza dentro del rango y termina dentro del rango
210                     (isAfter(this.startDate!, reserved.start) && isBefore(this.startDate!, reserved.end)) ||
211                     (isAfter(this.endDate!, reserved.start) && isBefore(this.endDate!, reserved.end)) ||
212                     //Caso 2: Empieza antes y termina dentro del rango
213                     (isBefore(this.startDate!, reserved.start) && isAfter(this.endDate!, reserved.start) &&
214                     isBefore(this.endDate!, reserved.end)) ||
215                     (isBefore(this.startDate!, reserved.start) && isAfter(this.endDate!, reserved.start) &&
216                     isEqual(this.endDate!, reserved.end)) ||
217                     // Caso 3: Empieza antes y termina después (engloba la reserva existente)
218                     (isBefore(this.startDate!, reserved.start) && isAfter(this.endDate!, reserved.end)) ||
219
220                     // Caso 4: Empieza en la misma fecha de inicio de una reserva existente y termina dentro del
221                     rango
222             )
223         )
224     }
225 }
```

```
220     (isEqual(this.startDate!, reserved.start) && isBefore(this.endDate!, reserved.end)) ||
221     // Caso 5: Empieza en la misma fecha de inicio de una reserva existente y termina fuera del
222     // rango
223     (isEqual(this.startDate!, reserved.start) && isAfter(this.endDate!, reserved.end)) ||
224     // Caso 6: Empieza dentro del rango y termina exactamente en la fecha de fin de una reserva
225     // existente
226     (isAfter(this.startDate!, reserved.start) && isEqual(this.endDate!, reserved.end)) ||
227
228     // Caso 7: Mismo rango exacto que una reserva existente
229     (isEqual(this.startDate!, reserved.start) && isEqual(this.endDate!, reserved.end))
230
231     );
232     if (!isOverlapping) {
233       this.events = [
234         ...this.events,
235         {
236           start: this.startDate,
237           end: this.endDate,
238
239           title:
240             'Rango seleccionado: ' +
241             format(this.startDate, 'dd/MM/yyyy') +
242             '_' +
243             format(this.endDate, 'dd/MM/yyyy') +
244             ' , haz click para reservar',
245             color: { primary: '#995d03', secondary: '#995d03' },
246
247           allDay: true,
248           resizable: {
249             beforeStart: true,
250             afterEnd: true,
251           },
252           draggable: false,
253         },
254       ];
255     } else {
256       this.snackBar.open('El rango seleccionado se superpone con una reserva existente.', 'Cerrar', {
257         duration: 3000,
258       });
259       this.cdr.detectChanges();
260
261     }
262     console.log(this.events);
263     this.cdr.detectChanges();
264     this.viewDate = new Date(this.viewDate);
265   } else if (this.startDate! <= today) {
266
267     this.snackBar.open('La fecha de inicio debe ser una fecha futura.', 'Cerrar', {
268       duration: 3000,
269     });
270   }
271 }
272
273 removeEvent(startDate: Date, endDate: Date): void {
274   this.events = this.events.filter(
275     (event) =>
276       !(
277         event.start.getTime() === startDate.getTime() &&
278         event.end?.getTime() === endDate.getTime()
279       )
280     );
281 }
282
283 changeMonth(offset: number) {
284   this.viewDate =
285     offset > 0
286       ? addMonths(this.viewDate, offset)
287       : subMonths(this.viewDate, -offset);
288   this.updateMonthYear();
289 }
290
291 onEventClicked(event: CalendarEvent): void {
292   if (
```

```
293     event.start.getTime() === this.startDate?.getTime() &&
294     event.end?.getTime() === this.endDate?.getTime()
295   )
296   {
297     this.seleccionado = true;
298     this.reservaService.obtenerDatosUsuario().subscribe({
299       next: (data: MyUser|null) => {
300         if (data) {
301           this.nombre = data.nombre ;
302           this.apellidos = data.apellidos;
303           this.email = data.email;
304           this.telefono = data.telefono;
305           this.dni = data.dni;
306
307           this.contactForm.patchValue({
308             name: `${this.nombre} ${this.apellidos}`,
309             email: this.email,
310             telefono: this.telefono,
311             dni: this.dni,
312           });
313         } else {
314           console.error('Data is null or undefined');
315         }
316       },
317       error: (error: any) => {
318         console.error('Error al obtener los datos del usuario:', error);
319       }
320     });
321   }
322 );
323
324 else {
325   this.seleccionado = false;
326 }
327 }
328
329 onSubmit(form: FormGroup) {
330   if (form.valid) {
331     const emailValue = form.get('email')?.value;
332     const nombreValue = form.get('name')?.value;
333     const dni = form.get('dni')?.value;
334     const telefono = form.get('telefono')?.value;
335     const numPersonas = form.get('numPersonas')?.value;
336     const startDateValue = this.startDate;
337     const endDateValue = this.endDate;
338
339     if (!startDateValue || !endDateValue) {
340       console.error('Fecha de inicio o fin no válida');
341       return;
342     }
343
344     const diffTime = Math.abs(endDateValue.getTime() - startDateValue.getTime());
345     const diffDays = Math.ceil(diffTime / (1000 * 3600 * 24));
346     let precioTotal: number = 0;
347     const precioPorDia = 350;
348     const precioFinDeSemana = 560;
349     const precioPorNocheExtra = 210;
350
351     if (diffDays <= 1) {
352
353       precioTotal = precioPorDia;
354     } else if (this.esFinDeSemana(startDateValue, endDateValue)) {
355
356       precioTotal = precioFinDeSemana;
357     } else if (diffDays >= 2) {
358
359       precioTotal = diffDays * precioPorNocheExtra;
360     }
361
362
363
364     const dialogRef = this.dialog.open(ResumenReservaComponent, {
365       data: {
366         nombre: nombreValue,
```

```

368     email: emailValue,
369     emailUsuario: sessionStorage.getItem('Email'),
370     dni: dni,
371     telefono: telefono,
372     numPersonas: numPersonas,
373     startDate: startDateValue,
374     endDate: endDateValue,
375     precio: precioTotal
376   }
377 });
378
379
380 } else {
381   this.snackBar.open('Formulario de reserva incorrecto. Rellene todos los datos necesarios',
382   'Cerrar', {
383     duration: 3000,
384   });
385 }
386
387
388 esFinDeSemana(startDate: Date, endDate: Date): boolean {
389   const startDay = startDate.getDay();
390   const endDay = endDate.getDay();
391
392
393   return (
394     (startDay === 5 && endDay === 6) ||
395     (startDay === 6 && endDay === 0) ||
396     (startDay === 5 && endDay === 0)
397   );
398 }
399
400 fetchWeather(): void {
401   if (!this.startDate || !this.endDate) {
402     alert('Por favor, ingresa ambas fechas.');
403     return;
404   }
405
406   this.weatherService
407     .getWeather(this.startDate, this.endDate)
408     .subscribe((data) => {
409
410     this.weatherData = data.map(item => {
411
412       item.date = new Date(item.date);
413       return item;
414     });
415   });
416 }
417 }

```

Listado A.20: Lógica del componente ComoLlegarComponent

```

1 import { AfterViewInit, Component, OnInit, ViewEncapsulation } from '@angular/core';
2 import { OverpassService } from '../services/overpass.service';
3 import * as L from 'leaflet';
4 import * as js2xmlparser from 'js2xmlparser';
5
6 @Component({
7   selector: 'app-como-llegar',
8   templateUrl: './como-llegar.component.html',
9   styleUrls: ['./como-llegar.component.scss']
10 })
11 export class ComoLlegarComponent implements OnInit, AfterViewInit {
12   private map: any;
13   private nodeCache: { [key: number]: any } = {};
14
15   constructor(private overpassService: OverpassService) {}
16
17   ngAfterViewInit(): void {
18     this.initMap();
19     this.loadRoutes();

```

```
20  }
21
22  ngOnInit(): void {}
23
24
25  initMap(): void {
26    this.map = L.map('map', { zoomControl: false }).setView([39.7699306, -1.030388], 10);
27    L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png').addTo(this.map);
28  }
29
30  loadRoutes(): void {
31    this.overpassService.getRoutes(39.7699306, -1.030388).subscribe((response) => {
32      this.filterNodes(response.elements);
33
34      let allRouteCoordinates: L.LatLng[] = [];
35
36      response.elements.forEach((element: any) => {
37        if (element.type === 'way' && element.nodes && element.nodes.length > 0) {
38          const routeCoordinates = this.getRouteCoordinates(element);
39
40          const polyline = L.polyline(routeCoordinates, { color: 'blue', weight: 3 }).addTo(this.map);
41
42          polyline.on('click', () => {
43            this.showRouteDetails(element);
44          });
45
46        }
47      });
48
49      allRouteCoordinates = allRouteCoordinates.concat(routeCoordinates);
50    }
51  });
52
53
54  if (allRouteCoordinates.length > 0) {
55    this.map.fitBounds(L.latLngBounds(allRouteCoordinates));
56  }
57);
58}
59
60
61 filterNodes(elements: any[]): void {
62  elements.forEach((element: any) => {
63    if (element.type === 'node' && element.lat && element.lon) {
64      this.nodeCache[element.id] = { lat: element.lat, lon: element.lon };
65    }
66  });
67}
68
69
70 getRouteCoordinates(route: any): L.LatLng[] {
71  const coordinates: L.LatLng[] = [];
72  route.nodes.forEach((nodeId: number) => {
73    const node = this.nodeCache[nodeId];
74    if (node) {
75      coordinates.push(L.latLng(node.lat, node.lon));
76    }
77  });
78  return coordinates;
79}
80
81
82 showRouteDetails(route: any): void {
83  const popupContent = `
84    <strong>Detalles:</strong><br>
85    ${route.tags.name}<br>
86    <button id="btnVer" class="btn-ver">Ver</button>
87    <button id="btnDescargar" class="btn-ver">Descargar</button>
88  `;
89
90  const popup = L.popup()
91    .setLatLng(this.map.getCenter())
92    .setContent(popupContent)
93    .openOn(this.map);
94}
```

```

95  setTimeout(() => {
96    const btnVer = document.getElementById('btnVer');
97    const btnDescargar = document.getElementById('btnDescargar');
98
99    btnVer?.addEventListener('click', () => {
100      window.open(`https://www.openstreetmap.org/way/${route.id}`, '_blank');
101    });
102
103    btnDescargar?.addEventListener('click', () => {
104      this.downloadGPX(route);
105    });
106  }, 0);
107}
108
109 downloadGPX(route: any): void {
110   const gpxData = this.convertToGPX(route);
111   const blob = new Blob([gpxData], { type: 'application/gpx+xml' });
112   const url = window.URL.createObjectURL(blob);
113   const a = document.createElement('a');
114   a.href = url;
115   a.download = `${route.id}.gpx`;
116   document.body.appendChild(a);
117   a.click();
118   document.body.removeChild(a);
119   window.URL.revokeObjectURL(url);
120 }
121
122
123
124 convertToGPX(route: any): string {
125
126   return `<?xml version="1.0"?>
<gpx version="1.1" creator="Leaflet">
  <trk><name>${route.tags.name}</name>
  <trkseg>
    ${route.nodes.map((nodeId: number) => {
131     const node = this.nodeCache[nodeId];
132     return node ? `<trkpt lat="${node.lat}" lon="${node.lon}"></trkpt>` : '';
133   }).join('')}
  </trkseg>
  </trk>
</gpx>`;
137 }
138 }
```

Listado A.21: Lógica del componente RecomendacionesComponent

```

1 import { Component, Inject } from '@angular/core';
2 import { Entorno } from '../compartido/entorno';
3 import { EntornoService } from '../services/entorno.service';
4 import { Media } from '../compartido/media';
5 import { MediaService } from '../services/media.service';
6 @Component({
7   selector: 'app-recomendaciones',
8
9   templateUrl: './recomendaciones.component.html',
10  styleUrls: ['./recomendaciones.component.scss'],
11})
12 export class RecomendacionesComponent {
13   videoUrl: string | null = null;
14
15   slidesComida: Media[] = [];
16
17   slidesFiestas: Entorno[] = [];
18
19   currentSlideComida = 0;
20   currentSlideFiesta = 0;
21   imagesPerSlideComida = 3;
22   imagesPerSlideFiesta = 3;
23
24   constructor( @Inject('baseURL') public baseURL: string, private entornoService: EntornoService, private mediaService: MediaService ) {}
```

```

26  ngOnInit(): void {
27    this.videoUrl = this.BaseURL + 'gastronomia';
28
29    this.entornoService.getFiestas().subscribe((places: Entorno[]) => {
30
31      console.log('Fiestas obtenidas:', places);
32      this.slidesFiestas = places;
33
34    });
35    this.fetchMediaByHashtag('gastronomia');
36    for (let i = 0; i < this.slidesComida.length; i++) {
37      console.log(this.slidesComida[i].mediaurl);
38    }
39
40    this.slidesComida.push (
41      { id: '1',
42        mediaurl: this.videoUrl|| '',
43        timestamp: new Date().toISOString(),
44        caption: 'Un gazpacho tradicional de Tuejar',
45        mediatype: 'video',
46        hashtag: 'gastronomia',}
47      );
48
49
50  }
51  prevSlideComida() {
52    this.currentSlideComida =
53    this.currentSlideComida > 0
54      ? this.currentSlideComida - this.imagesPerSlideComida
55      : Math.max(this.slidesComida.length - this.imagesPerSlideComida, 0);
56  }
57
58  nextSlideComida() {
59    this.currentSlideComida =
60    this.currentSlideComida <
61    this.slidesComida.length - this.imagesPerSlideComida
62      ? this.currentSlideComida + this.imagesPerSlideComida
63      : 0;
64  }
65
66
67  prevSlideFiesta() {
68    this.currentSlideFiesta =
69    this.currentSlideFiesta > 0
70      ? this.currentSlideFiesta - this.imagesPerSlideFiesta
71      : Math.max(this.slidesFiestas.length - this.imagesPerSlideFiesta, 0);
72  }
73
74  nextSlideFiesta() {
75    this.currentSlideFiesta =
76    this.currentSlideFiesta <
77    this.slidesFiestas.length - this.imagesPerSlideFiesta
78      ? this.currentSlideFiesta + this.imagesPerSlideFiesta
79      : 0;
80  }
81
82  fetchMediaByHashtag(hashtag: string): void {
83    this.mediaService.getMediaByHashtag(hashtag).subscribe({
84      next: (data: Media[]) => {
85
86        this.slidesComida.push(...data);
87      },
88      error: (err: any) => {
89        console.error('Error al obtener media:', err);
90      }
91    });
92  }
93
94 }
95

```

Listado A.22: Lógica del componente EntornoComponent

```
1 | import { Component, OnInit, ViewEncapsulation } from '@angular/core';
```

```

2 import { EntornoService } from '../services/entorno.service';
3 import { Entorno } from '../compartido/entorno';
4
5 @Component({
6   selector: 'app-entorno',
7   templateUrl: './entorno.component.html',
8   styleUrls: ['./entorno.component.scss']
9 })
10 export class EntornoComponent implements OnInit {
11   tuejarPlaces: Entorno[] = [];
12   chelvaPlaces: Entorno[] = [];
13
14   constructor(private entornoService: EntornoService) {}
15
16   ngOnInit(): void {
17     this.entornoService.getEntorno().subscribe((places: Entorno[]) => {
18
19       this.tuejarPlaces = places.filter(place => place.localizacion === 'tuejar' && place.categoría !==
20         'fiestas');
21       this.chelvaPlaces = places.filter(place => place.localizacion === 'chelva' && place.categoría !==
22         'fiestas');
23     });
24   }
25 }

```

Listado A.23: Lógica del componente LoginComponent

```

1 // login.component.ts
2 import { Component, OnInit, Renderer2, ViewEncapsulation } from '@angular/core';
3 import { FormBuilder, FormGroup, Validators } from '@angular/forms';
4 import { AuthService } from '../services/auth.service';
5 import { Router } from '@angular/router';
6 import { catchError, of } from 'rxjs';
7 @Component({
8   selector: 'app-login',
9
10  templateUrl: './login.component.html',
11  styleUrls: ['./login.component.scss']
12})
13 export class LoginComponent implements OnInit {
14   loginForm: FormGroup;
15   authError: string | null = null;
16
17   constructor(private fb: FormBuilder, private authService: AuthService, private renderer: Renderer2, private router: Router) {
18
19     this.loginForm = this.fb.group({
20       username: ['', Validators.required],
21       password: ['', Validators.required]
22     });
23
24   }
25   ngOnInit(): void {
26     const script = this.renderer.createElement('script');
27     script.src = 'app/assets/js/login.js';
28     console.log(script);
29     script.type = 'text/javascript';
30     this.renderer.appendChild(document.body, script);
31   }
32   onSubmit(): void {
33     if (this.loginForm.valid) {
34       const { username, password } = this.loginForm.value;
35       this.closeError();
36       this.authService.login(username, password).pipe(
37         catchError((error) => {
38
39           this.authError = 'El usuario o contraseña no existen en nuestra base de datos.';
40           return of(null);
41         })
42       ).subscribe({
43         next: (response) => {
44           if (response) {
45             console.log('Login exitoso');
46           }
47         }
48       });
49     }
50   }
51 }

```

```

46         this.router.navigate(['/inicio']);
47     }
48   });
49 }
50 } else {
51   this.authError = 'El usuario o contraseña no cumplen con las especificaciones.';
52 }
53 }
54
55 closeError(): void {
56   this.authError = null;
57 }
58 goToRegister() {
59   this.router.navigateByUrl('/register');
60 }
61 }

```

Listado A.24: Lógica del componente RegisterComponent

```

1 import { Component, OnInit } from '@angular/core';
2 import { FormGroup, FormBuilder, Validators, AbstractControl, ValidationErrors } from '@angular/forms';
3 import { Router } from '@angular/router';
4 import { RegistroService } from '../services/registro.service';
5 import { MatSnackBar } from '@angular/material/snack-bar';
6 import { HttpErrorResponse } from '@angular/common/http';
7 @Component({
8   selector: 'app-register',
9
10  templateUrl: './register.component.html',
11  styleUrls: ['./register.component.scss'],
12})
13 export class RegisterComponent implements OnInit {
14   registerForm: FormGroup;
15
16   constructor(
17     private fb: FormBuilder,
18     private registerService: RegistroService,
19     private router: Router ,
20     private snackBar: MatSnackBar
21   ) {
22     this.registerForm = this.fb.group({
23       password: ['', [Validators.required, Validators.minLength(6)]],
24       nombre: ['', [Validators.required, Validators.minLength(2)]],
25       apellidos: ['', [Validators.required, Validators.minLength(2)]],
26       email: ['', [Validators.required, Validators.email]],
27       telefono: ['', [Validators.required, Validators.pattern(/^[0-9]{9}$/)]],
28       dni: ['', [Validators.required, Validators.pattern(/^\d{8}[A-Z]{2}$/)]],
29       fechaNacimiento: ['', [Validators.required]],
30       direccion: ['', [Validators.required, Validators.minLength(10)]]
31     });
32   }
33 }
34
35 ngOnInit(): void {}
36
37 onSubmit(): void {
38
39   if (this.registerForm.valid) {
40     const formData = this.registerForm.value;
41
42     this.registerService.registerUser(formData).subscribe(
43       response => {
44         this.snackBar.open('Usuario registrado correctamente', 'Cerrar', {
45           duration: 3000,
46         });
47         this.router.navigate(['/login']);
48       },
49       error => {
50
51         this.snackBar.open(`$ {error.message}`, 'Cerrar', {
52           duration: 3000,
53         });
54

```

```
55     }
56   );
57 } else {
58   this.snackBar.open('Formulario invalido rellene correctamente los datos.', 'Cerrar', {
59     duration: 3000,
60   });
61 }
62 }
63
64
65 }
66 }
```

Listado A.25: Lógica del servicio de autenticación AuthService

```
1 import { HttpClient, HttpHeaders, HttpResponse } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { BehaviorSubject, Observable, tap, catchError, map, throwError } from 'rxjs';
4 import { baseURL } from '../compartido/baseurl';
5 import { Router } from '@angular/router';
6 import { MatSnackBar } from '@angular/material/snack-bar';
7
8 @Injectable({
9   providedIn: 'root'
10 })
11 export class AuthService {
12   private isAuthenticated = new BehaviorSubject<boolean>(false);
13   private userNameSubject = new BehaviorSubject<string | null>(null);
14
15   user$ = this.isAuthenticated.asObservable();
16   userName$ = this.userNameSubject.asObservable();
17   private accessToken: string | null = null;
18   private refreshToken: string | null = null;
19   private isAdmin = new BehaviorSubject<boolean>(false);
20
21   constructor(private http: HttpClient, private router: Router, private snackBar: MatSnackBar) {
22
23     const storedToken = sessionStorage.getItem('AccessToken');
24     const storedUsername = sessionStorage.getItem('Username');
25     const storedRoles = sessionStorage.getItem('Roles');
26
27
28     if (storedToken && storedUsername && storedRoles) {
29       this.isAuthenticated.next(true);
30       this.userNameSubject.next(storedUsername);
31       this.checkAdminRole(storedRoles);
32
33     }
34   }
35
36   login(username: string, password: string): Observable<any> {
37     const headers = new HttpHeaders({
38       'Authorization': `Basic ${btoa(`${username}:${password}`)}`,
39       'Content-Type': 'application/json',
40     });
41
42     const body = { username, password };
43
44     return this.http.post<any>(`${baseURL}login`, body, { headers, observe: 'response' }).pipe(
45       tap(response => {
46         const accessToken = response.headers.get('access_token');
47         const roles = response.headers.get('roles');
48         const refreshToken = response.headers.get('refresh_token');
49         console.log('Access Token:', accessToken);
50         console.log('Roles:', roles);
51         console.log('Refresh Token:', refreshToken);
52         const username = response.headers.get('username');
53         console.log('Username:', username);
54
55         if (accessToken && roles && username && refreshToken) {
56           sessionStorage.setItem('AccessToken', accessToken);
57           sessionStorage.setItem('RefreshToken', refreshToken);
58           sessionStorage.setItem('Roles', roles);
59         }
60       })
61     );
62   }
63
64   checkAdminRole(roles: string): void {
65     if (roles === 'admin') {
66       this.isAdmin.next(true);
67     } else {
68       this.isAdmin.next(false);
69     }
70   }
71
72   logout(): void {
73     this.router.navigate(['/login']);
74     this.isAuthenticated.next(false);
75     this.userNameSubject.next(null);
76     this.isAdmin.next(false);
77     sessionStorage.removeItem('AccessToken');
78     sessionStorage.removeItem('RefreshToken');
79     sessionStorage.removeItem('Roles');
80   }
81 }
```

```
59      this.http.get(baseURL + 'me?username=' + username, this.getAuthHeaders()).subscribe({
60        next: (data: any) => {
61          this.isAuthenticated.next(true);
62          console.log('Datos del usuario:', data);
63          sessionStorage.setItem('Username', data.nombre);
64          sessionStorage.setItem('Email', data.email);
65          this.userNameSubject.next(data.nombre);
66          this.checkAdminRole(roles);
67
68        },
69        error: (error) => {
70          console.error('Error al obtener los datos del usuario:', error);
71          this.isAuthenticated.next(false);
72          this.userNameSubject.next(null);
73        },
74      });
75    },
76  ),
77  catchError(error => {
78    throw error;
79  })
80 );
81 );
82 }
83 checkRoles(): Observable<any> {
84   return this.http.get<any>(baseURL + 'login/refresh', {
85     ...this.getAuthHeaders(),
86     observe: 'response'
87   }).pipe(
88     map(response => {
89       const accessToken = response.headers.get('access_token');
90       const refreshToken = response.headers.get('refresh_token');
91       const roles = response.headers.get('roles');
92
93       console.log('Roles aquí:', roles);
94
95       if (accessToken && refreshToken && roles) {
96         sessionStorage.setItem('AccessToken', accessToken);
97         sessionStorage.setItem('RefreshToken', refreshToken);
98         sessionStorage.setItem('Roles', roles);
99         this.checkAdminRole(roles);
100       }
101
102       return response.body;
103     }),
104     catchError(error => {
105       console.error('Error en checkRoles:', error);
106       return throwError(() => error);
107     })
108   );
109 }
110
111
112
113
114
115 sessionExpired(error: any): void {
116   if (error.status === 403 && error.error?.error_msg?.includes('The Token has expired')) {
117     this.snackBar.open('Su sesión ha expirado. Será redirigido al inicio de sesión.', 'Cerrar', {
118       duration: 5000,
119       horizontalPosition: 'center',
120       verticalPosition: 'top',
121       panelClass: ['session-expired-snackbar'],
122     });
123     this.logout();
124     setTimeout(() => {
125       this.router.navigate(['/login']);
126     }, 5000);
127   }
128 }
129
130 getAuthHeaders(): { headers: HttpHeaders } {
131   const token = sessionStorage.getItem('AccessToken');
132   return {
133     headers: new HttpHeaders({
```

```

134     Authorization: `Bearer ${token}`,
135     'Content-Type': 'application/json',
136   },
137 }
138 }

140 getAccessToken(): string | null {
141   return this.accessToken;
142 }
143

144 setAccessToken(token: string) {
145   this.accessToken = token;
146 }
147

148 setRefreshToken(token: string) {
149   this.refreshToken = token;
150 }
151

152 logout(): void {
153   sessionStorage.removeItem('AccessToken');
154   sessionStorage.removeItem('Roles');
155   sessionStorage.removeItem('Username');
156   sessionStorage.removeItem('Email');
157   this.isAuthenticated.next(false);
158   this.userNameSubject.next(null);
159   this.isAdmin.next(false);
160 }
161

162 isLoggedIn(): boolean {
163   return this.isAuthenticated.value;
164 }
165

166

167 private checkAdminRole(roles: string): void {
168   const userRolesString = sessionStorage.getItem('Roles')?.replace(/\[\]/g, '') || '';
169   const userRoles = userRolesString.split(',').map(role => role.trim());
170   console.log('Roles:', userRoles);
171   this.isAdmin.next(userRoles.includes('ROLE_ADMIN'));
172   console.log(this.isAdmin.value);
173 }
174

175

176

177

178 isAdmin$(): Observable<boolean> {
179   return this.isAdmin.asObservable();
180 }
181 }
182 }
```

A.7. Código HTML del frontend de la aplicación

Listado A.26: Tabla de visualización de reservas en `administrador.component.html`

```

1 <div class="filtros" >
2   <mat-form-field>
3     <input matInput placeholder="Filtrar por nombre" [(ngModel)]="filtros.nombre"
4       (keyup)="applyFilter()">
5   </mat-form-field>
6
6   <mat-form-field>
7     <input matInput placeholder="Filtrar por DNI" [(ngModel)]="filtros.dni" (keyup)="applyFilter()">
8   </mat-form-field>
9
10  <mat-form-field>
11    <mat-label>Filtrar por estado</mat-label>
12    <mat-select [(ngModel)]="filtros.estado" (selectionChange)="applyFilter()">
13      <mat-option value="">Todos</mat-option>
14      <mat-option value="CONFIRMADA">CONFIRMADA</mat-option>
15      <mat-option value="CANCELADA">CANCELADA</mat-option>
```

```

16     <mat-option value="PAGADA">PAGADA</mat-option>
17   </mat-select>
18 </mat-form-field>
19
20
21
22 <mat-accordion multi>
23
24   <!-- Panel de Reservas Activas -->
25   <mat-expansion-panel [expanded]="true">
26     <mat-expansion-panel-header>
27       <mat-panel-title>Reservas Activas</mat-panel-title>
28       <mat-panel-description>Ver u ocultar las reservas no finalizadas</mat-panel-description>
29     </mat-expansion-panel-header>
30
31     <mat-table [dataSource]="dataSource" class="mat-elevation-z8">
32       <!-- Nombre -->
33       <ng-container matColumnDef="nombre">
34         <mat-header-cell *matHeaderCellDef> Nombre </mat-header-cell>
35         <mat-cell *matCellDef="let reserva"> {{reserva.nombre}} </mat-cell>
36       </ng-container>
37
38       <!-- Número de Personas -->
39       <ng-container matColumnDef="numeroPersonas">
40         <mat-header-cell *matHeaderCellDef> Nº Personas </mat-header-cell>
41         <mat-cell *matCellDef="let reserva"> {{reserva.numPersonas}} </mat-cell>
42       </ng-container>
43
44       <!-- Estado -->
45       <ng-container matColumnDef="estado">
46         <mat-header-cell *matHeaderCellDef> Estado </mat-header-cell>
47         <mat-cell *matCellDef="let reserva"> {{reserva.estado}} </mat-cell>
48       </ng-container>
49
50       <!-- Fechas -->
51       <ng-container matColumnDef="fechas">
52         <mat-header-cell *matHeaderCellDef> Fechas </mat-header-cell>
53         <mat-cell *matCellDef="let reserva">
54           {{reserva.fechaInicio[0]}}/{{reserva.fechaInicio[1]}}/{{reserva.fechaInicio[0]}} -
55           {{reserva.fechaFin[2]}}/{{reserva.fechaFin[1]}}/{{reserva.fechaFin[0]}}
56         </mat-cell>
57       </ng-container>
58
59       <!-- Acciones -->
60       <ng-container matColumnDef="acciones">
61         <mat-header-cell *matHeaderCellDef> Acciones </mat-header-cell>
62         <mat-cell *matCellDef="let reserva">
63           <button *ngIf="reserva.estado === 'PAGADA' || reserva.estado === 'FINALIZADA'" mat-icon-button
64             [ngStyle]="{'color': 'green'}" disabled>
65             <mat-icon>check_circle</mat-icon>
66           </button>
67
68           <button *ngIf="reserva.estado === 'CANCELADA'" mat-icon-button color="warn" disabled>
69             <mat-icon>error</mat-icon>
70           </button>
71
72           <button *ngIf="reserva.estado !== 'PAGADA' && reserva.estado !== 'CANCELADA' && reserva.estado
73             !== 'FINALIZADA'" mat-icon-button [ngStyle]="{'color': 'green'}"
74             (click)="aprobarPagoReserva(reserva)">
75             <mat-icon>check</mat-icon>
76           </button>
77           <button *ngIf="reserva.estado !== 'PAGADA' && reserva.estado !== 'CANCELADA' && reserva.estado
78             !== 'FINALIZADA'" mat-icon-button color="warn" (click)="cancelarReserva(reserva)">
79             <mat-icon>close</mat-icon>
80           </button>
81         </mat-cell>
82       </ng-container>
83
84       <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
85       <mat-row *matRowDef="let row; columns: displayedColumns;"></mat-row>
86     </mat-table>
87   </mat-expansion-panel>
88
89   <!-- Panel de Reservas Finalizadas -->
90   <mat-expansion-panel>
```

```

87 <mat-expansion-panel-header>
88   <mat-panel-title>Reservas Finalizadas</mat-panel-title>
89   <mat-panel-description>Ver u ocultar las reservas completadas</mat-panel-description>
90 </mat-expansion-panel-header>
91
92 <mat-table [dataSource]="finalizadas" class="mat-elevation-z8">
93   <!-- Nombre -->
94   <ng-container matColumnDef="nombre">
95     <mat-header-cell *matHeaderCellDef> Nombre </mat-header-cell>
96     <mat-cell *matCellDef="let reserva"> {{reserva.nombre}} </mat-cell>
97   </ng-container>
98
99   <!-- Número de Personas -->
100  <ng-container matColumnDef="numeroPersonas">
101    <mat-header-cell *matHeaderCellDef> Nº Personas </mat-header-cell>
102    <mat-cell *matCellDef="let reserva"> {{reserva.numPersonas}} </mat-cell>
103  </ng-container>
104
105  <!-- Estado -->
106  <ng-container matColumnDef="estado">
107    <mat-header-cell *matHeaderCellDef> Estado </mat-header-cell>
108    <mat-cell *matCellDef="let reserva"> {{reserva.estado}} </mat-cell>
109  </ng-container>
110
111  <!-- Fechas -->
112  <ng-container matColumnDef="fechas">
113    <mat-header-cell *matHeaderCellDef> Fechas </mat-header-cell>
114    <mat-cell *matCellDef="let reserva">
115      {{reserva.fechaInicio[0]}}/{{reserva.fechaInicio[1]}}/{{reserva.fechaInicio[0]}} -
116      {{reserva.fechaFin[2]}}/{{reserva.fechaFin[1]}}/{{reserva.fechaFin[0]}}
117    </mat-cell>
118  </ng-container>
119
120  <!-- Acciones (solo icono verde) -->
121  <ng-container matColumnDef="acciones">
122    <mat-header-cell *matHeaderCellDef> Acciones </mat-header-cell>
123    <mat-cell *matCellDef="let reserva">
124      <button mat-icon-button [ngStyle]="{'color': 'green'}" disabled>
125        <mat-icon>check_circle</mat-icon>
126      </button>
127    </mat-cell>
128  </ng-container>
129
130  <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
131  <mat-row *matRowDef="let row; columns: displayedColumns;"></mat-row>
132 </mat-table>
133 </mat-expansion-panel>
134
135 </mat-accordion>
136 </div>

```

Listado A.27: Sección de Inicio en inicio.component.html

```

1 <div class="container">
2   <!-- Card de bienvenida -->
3   <mat-card class="welcome-card">
4     <mat-card-header>
5       <mat-card-title>¡Bienvenidos a El Rincón de Eva!</mat-card-title>
6       <mat-card-subtitle>
7         >Un lugar para desconectar y disfrutar</mat-card-subtitle>
8       >
9     </mat-card-header>
10
11   <mat-card-content>
12     <p>
13       La casa <strong>"El Rincón de Eva"</strong> está ubicada en la población
14       de Tuéjar, provincia de Valencia. Reserva de la Biosfera Unesco. Está
15       situada fuera del casco urbano, lo que permite disfrutar de una total
16       intimidad, pero a tan solo cinco minutos del pueblo y muy cerca de las
17       zonas recreativas.
18     </p>
19     <p>
20       En <strong>El Rincón de Eva</strong> podrás desconectar de tu estrés

```

```
21     diario, relajarte y disfrutar de la paz y tranquilidad del lugar. Su
22     casa acogedora y su maravilloso jardín te invitan a disfrutar del sol y
23     la piscina. Bonitos atardeceres tumbados en las hamacas y enamorarte del
24     cielo "Tuejano" repleto de estrellas al anochecer.
25   </p>
26   <mat-divider></mat-divider>
27   <blockquote class="quote">
28     "Los otoños con sus colores maravillosos y unos inviernos que, aunque
29     fríos, también tienen sus encantos."
30   </blockquote>
31 </mat-card-content>
32
33 <mat-card-actions>
34   <button mat-button class="reservarahora" [routerLink]="/reservas">
35     Reservar Ahora
36   </button>
37 </mat-card-actions>
38 </mat-card>
39
40 <mat-card class="review-card">
41   <mat-card-header>
42
43   <mat-card-title>Reseñas de Usuarios</mat-card-title>
44 </mat-card-header>
45
46   <mat-divider></mat-divider>
47
48   <mat-card class="review-item">
49 <mat-card-content>
50
51   <!-- Mostrar mensaje si no hay reseñas -->
52   <div *ngIf="!reviews || reviews.length === 0" class="no-reviews-message">
53     <p><strong>Haz tu reserva con nosotros y serás el primero en contarnos qué te ha parecido nuestra
54       casa.</strong></p>
55 </div>
56
57   <!-- Mostrar reseñas si existen -->
58   <div *ngIf="reviews && reviews.length > 0">
59     <div *ngFor="let review of reviews | slice:0:visibleReviewsCount; let i = index"
60       class="review-block">
61       <!-- Resumen principal -->
62       <mat-card class="review-summary" (click)="toggleExpanded(i)">
63         <div>
64           <div class="star-rating">
65             <ng-container *ngFor="let star of getStars(review.rate)">
66               <mat-icon class="filled" *ngIf="star==='filled'">star</mat-icon>
67               <mat-icon *ngIf="star === 'empty'">star_outline</mat-icon>
68             </ng-container>
69           </div>
70         <p class="review-text">{{ review.review }}</p>
71         <p>{{ review.creationDate }}</p>
72       </mat-card>
73
74       <!-- Detalles adicionales -->
75       <div *ngIf="expandedIndex === i" class="detailed-rates">
76         <div class="rate-detail">
77           <p><strong>Limpieza:</strong></p>
78           <div class="star-rating">
79             <ng-container *ngFor="let star of getStars(review.rateLimpieza)">
80               <mat-icon class="filled" *ngIf="star==='filled'">star</mat-icon>
81               <mat-icon *ngIf="star === 'empty'">star_outline</mat-icon>
82             </ng-container>
83           </div>
84         <div class="rate-detail">
85           <p><strong>Ubicación:</strong></p>
86           <div class="star-rating">
87             <ng-container *ngFor="let star of getStars(review.rateUbicacion)">
88               <mat-icon class="filled" *ngIf="star==='filled'">star</mat-icon>
89               <mat-icon *ngIf="star === 'empty'">star_outline</mat-icon>
90             </ng-container>
91           </div>
92         </div>
93       <div class="rate-detail">
```

```
94     <p><strong>Servicios:</strong></p>
95     <div class="star-rating">
96       <ng-container *ngFor="let star of getStars(review.rateServicios)">
97         <mat-icon class="filled" *ngIf="star==='filled'">star</mat-icon>
98         <mat-icon *ngIf="star === 'empty'">star_outline</mat-icon>
99       </ng-container>
100     </div>
101   </div>
102 </div>
103 </div>
104
105   <!-- Botón para mostrar más -->
106   <div *ngIf="visibleReviewsCount < reviews.length">
107     <button mat-button (click)="loadMoreReviews()">Mostrar más</button>
108   </div>
109
110   <!-- Botón para ocultar -->
111   <div *ngIf="visibleReviewsCount > 3">
112     <button mat-button (click)="hideReviews()">Ocultar</button>
113   </div>
114 </div>
115
116 </mat-card-content>
117 </mat-card>
118
119
120   <mat-divider *ngIf="isClientRole"></mat-divider>
121
122   <mat-card-content *ngIf="isClientRole">
123     <form [formGroup]="reviewForm" (ngSubmit)="onSubmitReview()">
124       <div><h3>¡Cuentanos tu experiencia!</h3></div>
125       <mat-form-field appearance="fill">
126
127         <mat-label>Tu reseña</mat-label>
128         <textarea matInput formControlName="review"></textarea>
129         <mat-error *ngIf="reviewForm.get('review')?.hasError('required')">
130           Reseña es requerida
131         </mat-error>
132         <mat-error *ngIf="reviewForm.get('review')?.hasError('minlength')">
133           La reseña debe tener al menos 10 caracteres
134         </mat-error>
135       </mat-form-field>
136
137       <div>
138         <label>Calificación General:</label>
139         <app-star-rating
140           [rating]="reviewForm.get('rate')?.value"
141           (ratingChange)="reviewForm.get('rate')?.setValue($event)"
142         ></app-star-rating>
143       </div>
144
145       <div>
146         <label>Ubicación:</label>
147         <app-star-rating
148           [rating]="reviewForm.get('rateUbicacion')?.value"
149           (ratingChange)="reviewForm.get('rateUbicacion')?.setValue($event)"
150         ></app-star-rating>
151       </div>
152
153       <div>
154         <label>Servicios:</label>
155         <app-star-rating
156           [rating]="reviewForm.get('rateServicios')?.value"
157           (ratingChange)="reviewForm.get('rateServicios')?.setValue($event)"
158         ></app-star-rating>
159       </div>
160
161       <div>
162         <label>Limpieza:</label>
163         <app-star-rating
164           [rating]="reviewForm.get('rateLimpieza')?.value"
165           (ratingChange)="reviewForm.get('rateLimpieza')?.setValue($event)"
166         ></app-star-rating>
167       </div>
168 
```

```

169     <button class="reservarahora" mat-icon-button type="submit" color="accent">
170       <mat-icon>send</mat-icon>
171     </button>
172   </form>
173 </mat-card-content>
174
175   <mat-divider></mat-divider>
176   <!-- Card de contacto -->
177 <div class="contact-form-container">
178   <mat-card class="contact-card">
179     <mat-card-header>
180       <mat-icon class="header-icon" aria-hidden="false">email</mat-icon>
181       <mat-card-title>Ponte en contacto conmigo</mat-card-title>
182     </mat-card-header>
183
184   <mat-divider></mat-divider>
185
186   <mat-card-content>
187     <form [formGroup]="contactForm" (ngSubmit)="onSubmit(contactForm)">
188       <mat-form-field appearance="fill">
189         <mat-label>Nombre</mat-label>
190         <input matInput formControlName="name" />
191         <mat-error *ngIf="contactForm.get('name')?.hasError('required')">
192           Nombre es requerido
193         </mat-error>
194       </mat-form-field>
195
196       <mat-form-field appearance="fill">
197         <mat-label>Email</mat-label>
198         <input matInput formControlName="email" type="email" />
199         <mat-error *ngIf="contactForm.get('email')?.hasError('required')">
200           Email es requerido
201         </mat-error>
202         <mat-error *ngIf="contactForm.get('email')?.hasError('email')">
203           Introduzca un email válido
204         </mat-error>
205       </mat-form-field>
206
207       <mat-form-field appearance="fill">
208         <mat-label>Mensaje</mat-label>
209         <textarea matInput formControlName="message"></textarea>
210         <mat-error *ngIf="contactForm.get('message')?.hasError('required')">
211           Mensaje es requerido
212         </mat-error>
213       </mat-form-field>
214
215       <button class="reservarahora" mat-icon-button type="submit">
216         <mat-icon>send</mat-icon>
217       </button>
218     </form>
219   </mat-card-content>
220 </mat-card>
221 </div>
222
223
224
225 </mat-card>
226 </div>

```

Listado A.28: Sección de información la casa rural en la-casa.component.html

```

1 <div class="house-container">
2   <mat-card class="house-card">
3     <mat-card-header>
4       <mat-card-title>La Casa</mat-card-title>
5     </mat-card-header>
6
7     <mat-divider></mat-divider>
8     <mat-card-content>
9       <mat-card-subtitle>Me presento</mat-card-subtitle>
10      <div class="video">
11        <video *ngIf="videoUrl" class="house-tour-video" [src]="videoUrl" controls controlslist
12          preload="auto">

```

```
12     Tu navegador no soporta la reproducción de video.
13   </video>
14
15   </div>
16
17   <div class="carousel-container">
18     <button class="carousel-button prev" (click)="prevSlide()">
19       &#10094;
20     </button>
21     <div class="carousel-slides">
22       <div
23         class="slide"
24         *ngFor="let slide of slides; let i = index"
25
26         [ngStyle]="{
27           transform: getTransformStyle(),
28           flex: getFlexStyle()
29         }"
30       >
31
32         <img [src]="slide.mediaurl" [alt]="slide.caption" />
33       </div>
34     </div>
35     <button class="carousel-button next" (click)="nextSlide()">
36       &#10095;
37     </button>
38   </div>
39
40   <mat-divider></mat-divider>
41
42   <p>
43     La casa cuenta con un total de <strong>cuatro habitaciones</strong>,
44     tres dobles y una individual. Baño completo y cocina totalmente equipada
45     (lavavajillas, microondas). Además cuenta con un paellero para disfrutar
46     de la cocina, con todo lo necesario para hacer brasas, paellas y una
47     cocina de gas de cinco fuegos y horno de gas (medidas).
48   </p>
49
50   <mat-divider></mat-divider>
51
52   <p>
53     Su bonito jardín de <strong>césped artificial</strong> rodeando la
54     piscina y una zona de césped natural. Nuestro preciado manzano de más de
55     50 años que sigue regalándonos su sombra y sus deliciosas manzanas.
56   </p>
57
58   <mat-divider></mat-divider>
59
60   <mat-card-subtitle>Detalles de las Habitaciones</mat-card-subtitle>
61   <p>
62     Las habitaciones todas ellas equipadas con ventilador de techo y
63     radiadores. Una de ellas cuenta también con aire acondicionado. En el
64     salón, chimenea, ventilador de techo y aire acondicionado. En el baño,
65     radiador y secador de pelo.
66   </p>
67
68   <mat-card-subtitle>Servicios y Equipamiento</mat-card-subtitle>
69
70   <ul class="sin-puntos">
71     <li>
72       <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" width="20px"><path d="M5.361 6.625a6.75
7.75 0 0 1 7.413 6.721 6.93 6.93 0 0 0 6.087-6.988V2.125a1.5 1.5 0 0 0-2.4-1.21-1.8
1.35h.9L13.461.7c-.8-.6-1.9-.6-2.7 0l-2.1 1.575h.91-1.8-1.35a1.5 1.5 0 0 0-2.4 1.2zm1.5 0v-4.511.8
1.35c.267.2.633.2.9 0l2.1-1.575a.75.75 0 0 1 .9 0l2.1 1.575c.267.2.633.2.9 0l1.8-1.35V6.37a5.43 5.43
0 0 1-4.754 5.486 5.25 5.25 0 0 1-5.746-5.23zm4.5 6v10.5a.75.75 0 0 0 1.5 0v-10.5a.75.75 0 0 0-1.5
0m-5.359 3.811c1.473.285 2.458 1.077 2.374
1.51-0.84432-1.292.801-2.765.516-1.473-.284-2.458-1.076-2.374-1.51.084-.432 1.292-.8
2.765-.516m.285-1.473c-2.179-.42-4.233.206-4.523 1.705s1.383 2.846 3.562 3.267 4.233-.205
4.523-1.705-1.383-2.846-3.562-3.267m14.699 2.09c.084434-.9 1.226-2.374-1.51 1.473-.285 2.681.084
2.765.517zm1.473-.284c-.29-1.5-2.344-2.126-4.523-1.705s-3.851 1.767-3.562 3.267 2.344 2.126 4.523
1.705 3.852-1.767 3.562-3.267"></path></svg>
    <b>Exterior</b>
    <ul class="sin-puntos">
      <li><mat-icon>check</mat-icon> Jardín</li>
```

```

76      <li><mat-icon>check</mat-icon> Piscina</li>
77      <li><mat-icon>check</mat-icon> Terraza</li>
78      <li><mat-icon>check</mat-icon> Zona de aparcamiento</li>
79      <li><mat-icon>check</mat-icon> Zona de juegos (baloncesto-tenis de mesa)</li>
80    </ul>
81  </li>
82
83  <li>
84    <img alt="SVG icon representing a basketball hoop and a tennis table." data-bbox="158 168 878 271"/>
85    <ul class="sin-puntos">
86      <li><mat-icon>check</mat-icon> Aire acondicionado en salón-comedor y habitación principal</li>
87      <li><mat-icon>check</mat-icon> Calefacción: radiadores eléctricos en todas las habitaciones y
88        toallero eléctrico en el baño</li>
89      <li><mat-icon>check</mat-icon> Ventiladores de techo en todas las estancias</li>
90      <li><mat-icon>check</mat-icon> Lavadora</li>
91      <li><mat-icon>check</mat-icon> Lavavajillas</li>
92      <li><mat-icon>check</mat-icon> Secadora</li>
93      <li><mat-icon>check</mat-icon> Microondas</li>
94      <li><mat-icon>check</mat-icon> Colección de juegos</li>
95      <li><mat-icon>check</mat-icon> TV Smart</li>
96    </ul>
97  </li>
98  <li>
99    <img alt="SVG icon representing a bathroom." data-bbox="158 458 878 538"/>
100   <ul class="sin-puntos">
101     <li><mat-icon>check</mat-icon> Papel higiénico</li>
102     <li><mat-icon>check</mat-icon> Toallas</li>
103     <li><mat-icon>check</mat-icon> Bidé</li>
104     <li><mat-icon>check</mat-icon> Baño privado</li>
105     <li><mat-icon>check</mat-icon> WC</li>
106     <li><mat-icon>check</mat-icon> Artículos de aseo gratis</li>
107     <li><mat-icon>check</mat-icon> Secador de pelo</li>
108     <li><mat-icon>check</mat-icon> Ducha</li>
109   </ul>
110 </li>
111 <li>
112   <img alt="SVG icon representing a kitchen." data-bbox="158 721 878 831"/>
113   <ul class="sin-puntos">
114     <li><mat-icon>check</mat-icon> Horno + encimera a gas (5 fuegos)</li>
115     <li><mat-icon>check</mat-icon> Paellero a leña para paellas y asar carne</li>
116   </ul>
117 </li>
118 </ul>
119 </li>
120 </ul>
121 <mat-card-subtitle>Condiciones</mat-card-subtitle>
122 <p>
```

```
124     Para garantizar la estancia, se deberá abonar una fianza de 150 euros, que será reembolsada al  
125     finalizar la estancia, siempre que no se hayan producido desperfectos en la propiedad.  
126 </p>  
127 <p>  
128     El check-in podrá acordarse con el gestor de la casa, para asegurar una llegada cómoda y sin  
129     contratiempos. Por otro lado, el check-out debe realizarse antes de las 17:00 h del día de salida.  
130 </p>  
131 </mat-card-content>  
132 </mat-card>  
133 </div>
```

Listado A.29: Sección de reservas de la casa rural en `reservas.component.html`

```
1 <div class="reservas-container">  
2   <mat-card class="tarifas-card" >  
3     <mat-card-header>  
4       <mat-card-title>Tarifas</mat-card-title>  
5     </mat-card-header>  
6     <mat-divider></mat-divider>  
7     <mat-card-content>  
8       <p>Alquiler casa completa. Animales consultar.</p>  
9  
10    <mat-list>  
11      <mat-list-item>  
12        <mat-icon matListIcon>calendar_today</mat-icon>  
13        <span class="tarifa-item">1 día: 350€</span>  
14      </mat-list-item>  
15  
16      <mat-list-item>  
17        <mat-icon matListIcon>weekend</mat-icon>  
18        <span class="tarifa-item">Fin de semana: 560€</span>  
19      </mat-list-item>  
20  
21      <mat-list-item>  
22        <mat-icon matListIcon>nights_stay</mat-icon>  
23        <span class="tarifa-item">A partir de 3 noches: 210€/noche</span>  
24      </mat-list-item>  
25  
26      <mat-list-item>  
27        <mat-icon matListIcon>event</mat-icon>  
28        <span class="tarifa-item">Puentes y festivos a consultar</span>  
29      </mat-list-item>  
30    </mat-list>  
31    <p *ngIf="email===''">Inicie sesión o <span>regístrate</span> para poder consultar el calendario de  
32    reservas y realizar una nueva reserva.  
33    <button mat-raised-button routerLink="/login" class="mat-button">Iniciar sesión /  
34    Registrarse</button></p>  
35  
36  </mat-card-content>  
37 </mat-card>  
38 <mat-card class="reservas-card" *ngIf="email!==''">  
39   <mat-card-header>  
40     <mat-card-title>Reservas</mat-card-title>  
41   </mat-card-header>  
42  
43   <mat-divider></mat-divider>  
44  
45   <mat-card-content>  
46     <hr />  
47     <p>Consulte disponibilidad y proceda con la reserva online:</p>  
48     <div class="calendar-controls">  
49       <button mat-button (click)="changeMonth(-1)">Ant.</button>  
50       <span>{{ currentMonthYear }}</span>  
51       <button mat-button (click)="changeMonth(1)">Sig.</button>  
52     </div>  
53     <div *ngIf="aviso !== ''"><span>{{aviso}}</span></div>  
54     <hr />  
55     <ng-container *ngIf="events">  
56       <mwl-calendar-month-view  
57         [viewDate]="viewDate"
```

```
58 [events]="events"
59 [weekStartsOn]="1"
60 (dayClicked)="onDayClicked($event)"
61 [activeDayIsOpen]="false"
62
63 (eventClicked)="onEventClicked($event.event)"
64 >
65 <ng-template #cellTemplate let-day="day">
66   <div [ngClass]="{{'cal-disabled': !dateIsValid(day.date)}}>
67     {{ day.date.getDate() }}
68   </div>
69 </ng-template>
70
71
72   </mwl-calendar-month-view>
73 </ng-container>
74 <mat-divider></mat-divider>
75 <div *ngIf="seleccionado === true">
76   <form [formGroup]="contactForm" (ngSubmit)="onSubmit(contactForm)">
77     <mat-form-field appearance="fill">
78       <mat-label>Nombre</mat-label>
79       <input matInput formControlName="name" />
80       <mat-error *ngIf="contactForm.get('name')?.hasError('required')">
81         Nombre es requerido
82       </mat-error>
83     </mat-form-field>
84
85     <mat-form-field appearance="fill">
86       <mat-label>Email</mat-label>
87       <input matInput formControlName="email" type="email" />
88       <mat-error *ngIf="contactForm.get('email')?.hasError('required')">
89         Email es requerido
90       </mat-error>
91       <mat-error *ngIf="contactForm.get('email')?.hasError('email')">
92         Introduzca un email válido
93       </mat-error>
94     </mat-form-field>
95     <mat-form-field appearance="fill">
96       <mat-label>DNI</mat-label>
97       <input matInput formControlName="dni" />
98       <mat-error *ngIf="contactForm.get('dni')?.hasError('required') &&
99         contactForm.get('dni')?.touched">
100        El DNI/NIF es obligatorio
101      </mat-error>
102      <mat-error *ngIf="contactForm.get('dni')?.hasError('pattern') &&
103        contactForm.get('dni')?.touched">
104        El DNI/NIF debe tener un formato válido
105      </mat-error>
106    </mat-form-field>
107
108    <mat-form-field appearance="fill">
109      <mat-label>Teléfono</mat-label>
110      <input matInput formControlName="telefono" />
111      <mat-error *ngIf="contactForm.get('telefono')?.hasError('required')">
112        Teléfono es requerido
113      </mat-error>
114      <mat-error *ngIf="contactForm.get('telefono')?.hasError('pattern') &&
115        contactForm.get('telefono')?.touched">
116        El teléfono debe tener un formato válido
117      </mat-error>
118    </mat-form-field>
119    <mat-form-field appearance="fill">
120      <mat-label>Número de huéspedes</mat-label>
121      <input matInput formControlName="numPersonas" type="number" min="1" max="7" />
122      <mat-error *ngIf="contactForm.get('numPersonas')?.hasError('required')">
123        El número de huéspedes es requerido
124      </mat-error>
125      <mat-error *ngIf="contactForm.get('numPersonas')?.hasError('min')">
126        El número mínimo de huéspedes es 1
127      </mat-error>
128      <mat-error *ngIf="contactForm.get('numPersonas')?.hasError('max')">
129        El número máximo de huéspedes es 7
130      </mat-error>
131    </mat-form-field>
```

```
130
131     <button class="reservarahora" mat-raised-button type="submit">
132         Reservar
133     </button>
134   </form>
135 </div>
136 <p>
137     Las reservas <span class="remarcar">también</span> podrán realizarse por teléfono, llamando o
138     utilizando
139     WhatsApp al:
140 </p>
141
142 <mat-card class="contact-card">
143     <a mat-button href="tel:+34669511506">
144         <mat-divider></mat-divider>
145         <mat-icon class="contact-icon">phone</mat-icon>669 511 506 - Eva
146         <mat-divider></mat-divider>
147     </a>
148 </mat-card>
149 </mat-card-content>
150 </mat-card>
151
152 <mat-card class="weather-card" *ngIf="email !== '' && weatherData.length > 0">
153     <mat-card-header>
154         <mat-card-title>Previsión del Tiempo</mat-card-title>
155     </mat-card-header>
156     <mat-divider></mat-divider>
157
158
159 <div class="weather-info">
160     <p class="prevision">Previsión del tiempo para la fecha seleccionada (en caso de superar los 15
161     días se obtendrá de los datos históricos, por lo que el tiempo puede variar):</p>
162
163     <mat-card class="weather-container" *ngFor="let weather of weatherData">
164         <a [href]="'https://www.aemet.es/es/eltiempo/prediccion/municipios/tuejar-id46247'">
165             <mat-card-header class="weather-text">
166                 <mat-card-title> <i [class]="weather.icon" [ngStyle]="{["font-size:": ["2rem"], ["display:": ["block"], ["margin:": ["0 auto"]}]}></i></mat-card-title>
167             </mat-card-header>
168             <mat-card-content class="text-center">
169                 <p class="weather-text">{{ weather.date | date:'dd-MM' }}</p>
170                 <p><strong>Min:</strong> {{ weather.minTemp }}°C<strong> Max:</strong> {{ weather.maxTemp }}°C</p>
171
172             </mat-card-content>
173         </a>
174     </mat-card>
175
176 </div>
177
178 <p *ngIf="weatherData.length === 0 && (startDate || endDate)">
179     No se encontraron datos para el rango de fechas especificado.
180 </p>
181 </mat-card>
182
183
184
185 </mat-card>
186
187
188
189
190
191 <mat-card class="mis-reservas-card" *ngIf="email!=='' && reservas && reservas.length > 0">
192     <mat-card-header>
193         <mat-card-title>Mis Reservas</mat-card-title>
194     </mat-card-header>
195     <mat-divider></mat-divider>
196     <mat-card-content>
197         <table mat-table [dataSource]="reservas" class="mat-elevation-z8">
198             <ng-container matColumnDef="id">
199                 <th mat-header-cell *matHeaderCellDef> Id </th>
200                 <td mat-cell *matCellDef="let reserva"> {{reserva.id}} </td>
```

```

202     </ng-container>
203     <ng-container matColumnDef="fecha">
204       <th mat-header-cell *matHeaderCellDef> Fechas </th>
205       <td mat-cell *matCellDef="let reserva">
206         {{reserva.fechaInicio[2]+'/'+reserva.fechaInicio[1]+'/'+reserva.fechaInicio[0]}}
207         - {{reserva.fechaFin[2]+'/'+reserva.fechaFin[1]+'/'+reserva.fechaFin[0]}}
208       </td>
209     </ng-container>
210     <ng-container matColumnDef="numPersonas">
211       <th mat-header-cell *matHeaderCellDef> N° Personas </th>
212       <td mat-cell *matCellDef="let reserva"> {{reserva.numPersonas}} </td>
213     </ng-container>
214     <ng-container matColumnDef="estado">
215       <th mat-header-cell *matHeaderCellDef> Estado </th>
216       <td mat-cell *matCellDef="let reserva"> {{reserva.estado}} </td>
217     </ng-container>
218     <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
219     <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
220   </table>
221 </mat-card-content>
222 </mat-card>
223
224 <mat-card *ngIf="user && (!reservas || reservas.length === 0)">
225   <mat-card-header>
226     <mat-card-title>No tienes reservas</mat-card-title>
227   </mat-card-header>
228   <mat-card-content>
229     <p>Aún no has realizado ninguna reserva. <span class="remarcar">¡Hazlo ahora!</span></p>
230   </mat-card-content>
231 </mat-card>
232 </div>

```

Listado A.30: Sección como llegar en `como-llegar.component.html`

```

1 <div class="container">
2   <mat-card class="location-card">
3     <mat-card-header>
4       <mat-card-title>¡Dónde estamos?</mat-card-title>
5     </mat-card-header>
6     <mat-divider></mat-divider>
7
8     <mat-card-content>
9       <p>Estamos en Partida de los Secanos nº 33; parcela 147.</p>
10
11    <iframe class="map-iframe"
12      src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d889.3937124543116!2d-1.03858802698026!
13      52!3d39.77419015348232!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0xd60baf49313354d%3A0xa!
14      5c393c5f8a9b855!2sDiseminado%20Tuejar-r.s%2C%2013%2C%2046177%20%20Valencia!5e1!3m2!1ses!2ses!4v1
15      725265838818!5m2!1ses!2ses"
16      width="100%"
17      height="300"
18
19      loading="lazy"
20      referrerpolicy="no-referrer-when-downgrade"
21    ></iframe>
22
23    <div class="rutas-section">
24      <mat-card-header>
25        <mat-card-title>Rutas cercanas</mat-card-title>
26      </mat-card-header>
27      <mat-divider></mat-divider>
28
29      <p>
30        Aquí os dejo una serie de rutas que se podrían realizar cerca de nuestra casa rural. ¡Clicando
31        sobre cada una podréis descargarla y comenzar a andar!
32      </p>
33      <div class="map-container">
34        <div id="map"></div>

```

35 | </div>

Listado A.31: Sección de recomendaciones en `recomendaciones.component.html`

```
1  <!-- recomendaciones.component.html -->
2  <div class="recomendaciones-container">
3      <mat-card class="recomendaciones-header-card">
4          <mat-card-header>
5              <mat-card-title>Recomendaciones</mat-card-title>
6          </mat-card-header>
7          <mat-divider></mat-divider>
8      </mat-card>
9
10     <div class="subsections-container">
11         <!-- Gastronomía -->
12         <mat-card class="subsection-card">
13             <mat-card-header>
14                 <mat-card-title>Gastronomía</mat-card-title>
15             </mat-card-header>
16             <mat-card-content>
17                 <section>
18                     <mat-card>
19                         <mat-card-content>
20                             <p>Tuéjar cuenta con una gastronomía típica de la zona de montaña de inviernos fríos. Pueblo ganadero, con carnes y embutidos excelentes y con una huerta con productos de gran calidad que ayuda a ofrecer ricos platos típicos.</p>
21                         </mat-card-content>
22                     </mat-card>
23                 </section>
24             </mat-card-content>
25         </mat-card>
26
27         <div class="recetas-container">
28             <mat-card class="receta-card">
29                 <mat-card-title>Gazpachos</mat-card-title>
30                 <mat-card-content>Una especialidad montañesa con ingredientes de la zona.</mat-card-content>
31             </mat-card>
32
33             <mat-card class="receta-card">
34                 <mat-card-title>Olla de Pueblo</mat-card-title>
35                 <mat-card-content>Un guiso tradicional que representa la esencia de la cocina local.</mat-card-content>
36             </mat-card>
37
38             <mat-card class="receta-card">
39                 <mat-card-title>Morteruelo</mat-card-title>
40                 <mat-card-content>Plato típico de la región, ideal para los días fríos.</mat-card-content>
41             </mat-card>
42
43             <mat-card class="receta-card">
44                 <mat-card-title>Tortas de Tajá</mat-card-title>
45                 <mat-card-content>Una delicia tradicional preparada con productos frescos de la huerta.</mat-card-content>
46             </mat-card>
47
48             <mat-card class="receta-card">
49                 <mat-card-title>Magdalenas</mat-card-title>
50                 <mat-card-content>Recién hechas, ideales para acompañar el café.</mat-card-content>
51             </mat-card>
52
53             <mat-card class="receta-card">
54                 <mat-card-title>Ensaimina</mat-card-title>
55                 <mat-card-content>Un postre tradicional, símbolo de la panadería local.</mat-card-content>
56             </mat-card>
57
58             <mat-card class="receta-card">
59                 <mat-card-title>Esparteros</mat-card-title>
60                 <mat-card-content>Dulces típicos que reflejan las tradiciones culinarias del pueblo.</mat-card-content>
61             </mat-card>
62
63             <mat-card class="receta-card">
64                 <mat-card-title>Tortas de Pasas y Nueces</mat-card-title>
```

```
64         <mat-card-content>Perfectas para los amantes de los frutos secos.</mat-card-content>
65     </mat-card>
66
67     <mat-card class="receta-card">
68         <mat-card-title>Muégados</mat-card-title>
69         <mat-card-content>Dulce estrella de Tuéjar, hecho con harina, huevos y
70             miel.</mat-card-content>
71     </mat-card>
72 </div>
73 </section>
74
75     <!-- app.component.html -->
76     <div class="carousel-container">
77         <button class="carousel-button prev" (click)="prevSlideComida()">
78             &#10094;
79         </button>
80         <div class="carousel-slides">
81             <div
82                 class="slide"
83                 *ngFor="let slide of slidesComida; let i = index"
84                 [style.transform]="'translateX(-' + currentSlideComida * (100 / imagesPerSlideComida) + '%)'"
85                 [style.flex]="'0 0 ' + 100 / imagesPerSlideComida + '%'"
86             >
87                 <ng-container *ngIf="slide.mediatype === 'IMAGE'; else videoTemplate">
88                     <a [href]="slide.mediaurl" target="_blank">
89                         <img [src]="slide.mediaurl" [alt]="slide.caption" />
90                     </a>
91                 </ng-container>
92
93                 <ng-template #videoTemplate>
94                     <a [href]="slide.mediaurl" target="_blank">
95                         <video>
96                             <source [src]="slide.mediaurl" type="video/mp4" />
97                             Tu navegador no soporta la reproducción de vídeo.
98                         </video>
99                     </a>
100                </ng-template>
101            </div>
102
103            </div>
104            <button class="carousel-button next" (click)="nextSlideComida()">
105                &#10095;
106            </button>
107        </div>
108    </mat-card-content>
109 </mat-card>
110
111
112     <!-- Fiestas -->
113     <mat-card class="subsection-card">
114         <mat-card-header>
115             <mat-card-title>Fiestas</mat-card-title>
116         </mat-card-header>
117         <mat-card-content>
118             <p>Tuéjar cuenta durante todo el año con festividades locales:</p>
119             <ul>
120                 <li>
121                     <strong>Enero:</strong> San Antón. La noche víspera del 17 de Enero.
122                 </li>
123                 <li><strong>Febrero:</strong> Carnavales</li>
124                 <li>
125                     <strong>Marzo:</strong> Fallas (se celebran normalmente la semana
126                     después de las Fallas de Valencia)
127                 </li>
128                 <li>
129                     <strong>Abril:</strong> Los Mayos. La noche del 30 de abril. Se da
130                     la bienvenida a la primavera y se despide el invierno.
131                 </li>
132                 <li>
133                     <strong>Agosto:</strong> Las fiestas mayores y fiestas gordas,
134                     dedicadas a la patrona del pueblo La Purísima. Primeras semanas de
135                     agosto. Llenas de actos tradicionales, actividades para niños y
136                     adultos, verbenas, disfraces, y el desfile de coches engalanados.
137                 </li>

```

```
138     <li>
139         <strong>Diciembre:</strong> La Purísima. El 8 de diciembre. Fiestas
140             dedicadas a la Inmaculada Concepción.
141         </li>
142     </ul>
143
144     <!-- Carousel de imágenes de fiestas -->
145     <div class="carousel-container">
146         <button class="carousel-button prev" (click)="prevSlideFiesta()">
147             &#10094;
148         </button>
149         <div class="carousel-slides">
150             <div
151                 class="slide"
152                 *ngFor="let slide of slidesFiestas; let i = index"
153                 [style.transform]="
154                     translateX(-{{ currentSlideFiesta * 100 / imagesPerSlideFiesta }}) +
155                     0
156                 "
157                 [style.flex]="'0 0 ' + 100 / imagesPerSlideFiesta + '%'"
158             >
159                 <a [href]="slide.url"><img [src]="slide.imageString" [alt]="slide.title" /></a>
160             </div>
161         </div>
162         <button class="carousel-button next" (click)="nextSlideFiesta()">
163             &#10095;
164         </button>
165     </div>
166     </mat-card-content>
167 </mat-card>
168 </div>
169 </div>
170 </div>
```

Listado A.32: Sección de entorno en `entorno.component.html`

```
1 <div class="entorno-container">
2     <!-- Encabezado del entorno -->
3     <mat-card class="entorno-header-card">
4         <mat-card-header>
5
6             <mat-card-title>Entorno</mat-card-title>
7         </mat-card-header>
8         <mat-divider></mat-divider>
9         <mat-card-content>
10            <p>
11                La comarca de los serranos cuenta con un entorno de naturaleza impresionante. Rodeada de
12                frondosos bosques y bonitos lugares para visitar.
13            </p>
14        </mat-card-content>
15    </mat-card>
16
17    <!-- Apartado de Tuéjar -->
18    <div class="section">
19        <h2 class="seccion">Tuéjar</h2>
20        <div class="grid">
21            <mat-card class="place-card" *ngFor="let place of tuejarPlaces">
22                <a [href]="place.url">
23                    <img mat-card-image [src]="place.imageString" [alt]="place.title" />
24                <mat-card-header>
25                    <mat-card-title>{{ place.title }}</mat-card-title>
26
27                </mat-card-header>
28                <mat-card-content>
29                    <p>{{ place.description }}</p>
30                </mat-card-content>
31            </a>
32            </mat-card>
33
34        </div>
35    </div>
36
37    <!-- Apartado de Chelva -->
```

```

38   <div class="section">
39     <h2 class="seccion">Chelva</h2>
40     <div class="grid">
41
42       <mat-card class="place-card" *ngFor="let place of chelvaPlaces">
43         <a [href]="place.url">
44           <img mat-card-image [src]="place.imageString" [alt]="place.title" />
45           <mat-card-header>
46             <mat-card-title>{{ place.title }}</mat-card-title>
47
48           </mat-card-header>
49           <mat-card-content>
50             <p>{{ place.description }}</p>
51           </mat-card-content>
52         </a>
53       </mat-card>
54     </div>
55   </div>
56 </div>

```

Listado A.33: Sección de login en login.component.html

```

1 <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
2   <div class="svgContainer">
3     <div>
4       <svg class="mySVG" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
5         viewBox="0 0 200 200">
6         <defs>
7           <circle id="armMaskPath" cx="100" cy="100" r="100" />
8         </defs>
9         <clipPath id="armMask">
10           <use xlink:href="#armMaskPath" overflow="visible" />
11         </clipPath>
12         <circle cx="100" cy="100" r="100" fill="#d4c9c0" />
13       <g class="body">
14         <path class="bodyBGchanged" [ngStyle]="{display: 'none'}" fill="#FFFFFF"
15           d="M200,122h-35h-14.9V72c0-27.6-22.4-50-50s-50,22.4-50,50v50H35.8H010,91h200L200,122z" />
16         <path class="bodyBNormal" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
17           stroke-linejoin="round" fill="#FFFFFF" d="M200,158.5c0-20.2-14.8-36.5-35-36.5h-14.9V72.8c0-27
18             .4-21.7-50.4-49.1-50.8c-28-0.5-50.9,22.1-50.9,50v50
19             H35.8C16,122,0,138,0,157.8L0,213h200L200,158.5z" />
20         <path fill="#d4c9c0"
21           d="M100,156.4c-22.9,0-43,11.1-54.1,27.7c15.6,10,34.2,15.9,54.1,15.9s38.5-5.8,54.1-15.9
22             C143,167.5,122.9,156.4,100,156.4z" />
23       </g>
24       <g class="earL">
25         <g class="outerEar" fill="#d4c9c0" stroke="#995d03" stroke-width="2.5">
26           <circle cx="47" cy="83" r="11.5" />
27           <path d="M46.3 78.9c-2.3 0-4.1 1.9-4.1 4.1 0 2.3 1.9 4.1 4.1 4.1" stroke-linecap="round"
28             stroke-linejoin="round" />
29         </g>
30         <g class="earHair">
31           <rect x="51" y="64" fill="#FFFFFF" width="15" height="35" />
32           <path d="M53.4 62.8C48.5 67.4 45 72.2 42.8 77c3.4-1 6.8-1 10.1-1-4 3.7-6.8 7.6-8.2 11.6 2.1
33             0 4.2 0 6.3-2.2 4.1-3.8 8.3-3.7 12.5 1.2-7 3.4-1.4 5.2-1.9" fill="#fff" stroke="#995d03"
34             stroke-width="2.5" stroke-linecap="round" stroke-linejoin="round" />
35         </g>
36       </g>
37       <g class="earR">
38         <g class="outerEar">
39           <circle fill="#d4c9c0" stroke="#995d03" stroke-width="2.5" cx="153" cy="83" r="11.5" />
40           <path fill="#d4c9c0" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
41             stroke-linejoin="round" d="M153.7,78.9 c2.3,0,4.1,1.9,4.1,4.1c0,2.3-1.9,4.1-4.1,4.1" />
42         </g>
43         <g class="earHair">
44           <rect x="134" y="64" fill="#FFFFFF" width="15" height="35" />
45           <path fill="#FFFFFF" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
46             stroke-linejoin="round" d="M146.6,62.8 c4.9,4.6,8.4,9.4,10.6,14.2c-3.4-0.1-6.8-0.1-10.1,0.1c
47             4.3,7.6,8.2,11.6c-2.1,0-4.2,0-6.3,0.2c2.6,4.1,3.8,8.3,3.7,12.5
48             c-1.2-0.7-3.4-1.4-5.2-1.9" />
49         </g>
50       </g>
51     </div>
52   </form>

```

```

38   <path class="chin" d="M84.1 121.6c2.7 2.9 6.1 5.4 9.8 7.5l.9-4.5c2.9 2.5 6.3 4.8 10.2 6.5
39   0-1.9-.1-3.9-.2-5.8 3 1.2 6.2 2 9.7 2.5-.3-2.1-.7-4.1-1.2-6.1" fill="none" stroke="#995d03"
40   stroke-width="2.5" stroke-linecap="round" stroke-linejoin="round" />
41   <path class="face" fill="#d4c9c0"
42   d="M134.5,46v35.5c0,21.815-15.446,39.5-34.5,39.5s-34.5-17.685-34.5-39.5V46" />
43   <path class="hair" fill="#FFFFFF" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
44   stroke-linejoin="round" d="M81.457,27.929 c1.755-4.084,5.51-8.262,11.253-11.77c0.979,2.565,1.883
45   ,5.14,2.712,7.723c3.162-4.265,8.626-8.27,16.272-11.235
46   c-0.737,3.293-1.588,6.573-2.554,9.837c4.857-2.116,11.049-3.64,18.428-4.156c-2.403,3.23-5.021,6.3
47   91-7.852,9.474" />
48   <g class="eyebrow">
49   <path fill="#FFFFFF"
50   d="M138.142,55.064c-4.93,1.259-9.874,2.118-14.787,2.599c-0.336,3.341-0.776,6.689-1.322,10.037
51   c-4.569-1.465-8.909-3.222-12.996-5.226c-0.98,3.075-2.07,6.137-3.267,9.179c-5.514-3.067-10.559-
52   6.545-15.097-10.329
53   c-1.806,2.889-3.745,5.73-5.816,8.515c-7.916-4.124-15.053-9.114-21.296-14.738l1.107-11.768h73.4
54   75V55.064z" />
55   <path fill="#FFFFFF" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
56   stroke-linejoin="round" d="M63.56,55.102 c6.243,5.624,13.38,10.614,21.296,14.738c2.071-2.785,4
57   .01-5.626,5.816-8.515c4.537,3.785,9.583,7.263,15.097,10.329
58   c1.197-3.043,2.287-6.104,3.267-9.179c4.087,2.004,8.427,3.761,12.996,5.226c0.545-3.348,0.986-6.
59   696,1.322-10.037 c4.913-0.481,9.857-1.34,14.787-2.599" />
60   </g>
61   <g class="eyeL">
62   <circle cx="85.5" cy="78.5" r="3.5" fill="#995d03" />
63   <circle cx="84" cy="76" r="1" fill="#fff" />
64   </g>
65   <g class="eyeR">
66   <circle cx="114.5" cy="78.5" r="3.5" fill="#995d03" />
67   <circle cx="113" cy="76" r="1" fill="#fff" />
68   </g>
69   <g class="mouth">
70   <path class="mouthBG" fill="#617E92"
71   d="M100.2,101c-0.4,0-1.4,0-1.8,0c-2.7-0.3-5.3-1.1-8-2.5c-0.7-0.3-0.9-1.2-0.6-1.8 c0.2-0.5,0.7-
0.7,1.2-0.7c0.2,0,0.5,0.1,0.6,0.2c3,1.5,5.8,2.3,8.6,2.3s5.7-0.7,8.6-2.3c0.2-0.1,0.4-0.2,0.6-0.2
c0.5,0,1,0.3,1.2,0.7c0.4,0.7,0.1,1.5-0.6,1.9c-2.6,1.4-5.3,2.2-7.9,2.5C101.7,101,100.5,101,100.
2,101z" />
72   <path [ngStyle]={"display: 'none'"} class="mouthSmallBG" fill="#617E92"
73   d="M100.2,101c-0.4,0-1.4,0-1.8,0c-2.7-0.3-5.3-1.1-8-2.5c-0.7-0.3-0.9-1.2-0.6-1.8 c0.2-0.5,0.7-
0.7,1.2-0.7c0.2,0,0.5,0.1,0.6,0.2c3,1.5,5.8,2.3,8.6,2.3s5.7-0.7,8.6-2.3c0.2-0.1,0.4-0.2,0.6-0.2
c0.5,0,1,0.3,1.2,0.7c0.4,0.7,0.1,1.5-0.6,1.9c-2.6,1.4-5.3,2.2-7.9,2.5C101.7,101,100.5,101,100.
2,101z" />
74   <path [ngStyle]={"display: 'none'"} class="mouthMediumBG"
75   d="M95,104.2c-4.5,0-8.2-3.7-8.2-8.2v-2c0-1.2,1-2.2,2.2-2.2h22c1.2,0,2.2,1,2.2,2.2v2
c0,4.5-3.7,8.2-8.2,8.2H95z" />
76   <path [ngStyle]={"display: 'none'"} class="mouthLargeBG" d="M100 110.2c-9 0-16.2-7.3-16.2-16.2
0-2.3 1.9-4.2 4.2-4.2h24c2.3 0 4.2 1.9 4.2 4.2 0 9-7.2 16.2-16.2 16.2z" fill="#617e92"
77   stroke="#995d03" stroke-linejoin="round" stroke-width="2.5" />
78   <defs>
79   <path id="mouthMaskPath"
80   d="M100.2,101c-0.4,0-1.4,0-1.8,0c-2.7-0.3-5.3-1.1-8-2.5c-0.7-0.3-0.9-1.2-0.6-1.8
c0.2-0.5,0.7-0.7,1.2-0.7c0.2,0,0.5,0.1,0.6,0.2c3,1.5,5.8,2.3,8.6,2.3s5.7-0.7,8.6-2.3c0.2-0.1
,0.4-0.2,0.6-0.2
c0.5,0,1,0.3,1.2,0.7c0.4,0.7,0.1,1.5-0.6,1.9c-2.6,1.4-5.3,2.2-7.9,2.5C101.7,101,100.5,101,100.
2,101z" />
81   </defs>
82   <clipPath id="mouthMask">
83   <use xlink:href="#mouthMaskPath" overflow="visible" />
84   </clipPath>
85   <g clip-path="url[#mouthMask)">
86   <g class="tongue">
87   <circle cx="100" cy="107" r="8" fill="#cc4a6c" />
88   <ellipse class="tongueHighlight" cx="100" cy="100.5" rx="3" ry="1.5" opacity=".1"
fill="#fff" />
89   </g>
90   </g>
91   <path clip-path="url[#mouthMask)" class="tooth" [ngStyle]={"fill:'#FFFFFF'"}
92   d="M106,97h-4c-1.1,0-2-0.9-2-2v-2h8v2C108,96.1,107.1,97,106,97z" />
93   <path class="mouthOutline" fill="none" stroke="#995d03" stroke-width="2.5"
stroke-linejoin="round"
94   d="M100.2,101c-0.4,0-1.4,0-1.8,0c-2.7-0.3-5.3-1.1-8-2.5c-0.7-0.3-0.9-1.2-0.6-1.8 c0.2-0.5,0.7-
0.7,1.2-0.7c0.2,0,0.5,0.1,0.6,0.2c3,1.5,5.8,2.3,8.6,2.3s5.7-0.7,8.6-2.3c0.2-0.1,0.4-0.2,0.6-0.2
c0.5,0,1,0.3,1.2,0.7c0.4,0.7,0.1,1.5-0.6,1.9c-2.6,1.4-5.3,2.2-7.9,2.5C101.7,101,100.5,101,100.
2,101z" />
```

```

72   </g>
73   <path class="nose" d="M97.7 79.9h4.7c1.9 0 3 2.2 1.9 3.71-2.3 3.3c-.9 1.3-2.9 1.3-3.8
74     01-2.3-3.3c-1.3-1.6-2.3-3.7 1.8-3.7z" fill="#995d03" />
75   <g class="arms" clip-path="url[#armMask)">
76     <g class="armL" [ngStyle]={`${ visibility : 'hidden' }`}>
77       <polygon fill="#d4c9c0" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
78         stroke-linejoin="round" stroke-miterlimit="10" points="121.3,98.4 111,59.7 149.8,49.3
79         169.8,85.4" />
80       <path fill="#d4c9c0" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
81         stroke-linejoin="round" stroke-miterlimit="10"
82         d="M134.4,53.5119.3-5.2c2.7-0.7,5.4,0.9,6.1,3.5v0c0.7,2.7-0.9,5.4-3.5,6.11-10.3,2.8" />
83       <path fill="#d4c9c0" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
84         stroke-linejoin="round" stroke-miterlimit="10"
85         d="M150.9,59.4126-7c2.7-0.7,5.4,0.9,6.1,3.5v0c0.7,2.7-0.9,5.4-3.5,6.11-21.3,5.7" />
86
87     <g class="twoFingers">
88       <path fill="#d4c9c0" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
89         stroke-linejoin="round" stroke-miterlimit="10"
90         d="M158.3,67.8123.1-6.2c2.7-0.7,5.4,0.9,6.1,3.5v0c0.7,2.7-0.9,5.4-3.5,6.11-23.1,6.2" />
91       <path fill="#d4c9c0"
92         d="M180.1,6512.2-0.6c1.1-0.3,2.2,0.3,2.4,1.4v0c0.3,1.1-0.3,2.2-1.4,2.41-2.2,0.6L180.1,65z"
93         />
94       <path fill="#d4c9c0" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
95         stroke-linejoin="round" stroke-miterlimit="10"
96         d="M160.8,77.5119.4-5.2c2.7-0.7,5.4,0.9,6.1,3.5v0c0.7,2.7-0.9,5.4-3.5,6.11-18.3,4.9" />
97       <path fill="#d4c9c0" d="M178.8,75.712.2-0.6c1.1-0.3,2.2,0.3,2.4,1.4v0c0.3,1.1-0.3,2.2-1.4,_
98         2.41-2.2,0.6L178.8,75.7z" />
99     </g>
100   <path fill="#d4c9c0" d="M175.5,55.912.2-0.6c1.1-0.3,2.2,0.3,2.4,1.4v0c0.3,1.1-0.3,2.2-1.4,2.1
101     41-2.2,0.6L175.5,55.9z" />
102   <path fill="#d4c9c0" d="M152.1,50.412.2-0.6c1.1-0.3,2.2,0.3,2.4,1.4v0c0.3,1.1-0.3,2.2-1.4,2.1
103     41-2.2,0.6L152.1,50.4z" />
104   <path fill="#FFFFFF" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
105     stroke-linejoin="round" d="M123.5,97.8
106     c-41.4,14.9-84.1,30.7-108.2,35.5L1.2,81c33.5-9.9,71.9-16.5,111.9-21.8" />
107   <path fill="#FFFFFF" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
108     stroke-linejoin="round" d="M108.5,60.4 c7.7-5.3,14.3-8.4,22.8-13.2c-2.4,5.3-4.7,10.3-6.7,15.1
109     c4.3,0.3,8.4,0.7,12.3,1.3c-4.2,5-8.1,9.6-11.5,13.9
110     c3.1,1.1,6,2.4,8.7,3.8c-1.4,2.9-2.7,5.8-3.9,8.5c2.5,3.5,4.6,7.2,6.3,11c-4.9-0.8-9-0.7-16.2-2
111     .7" />
112   <path fill="#FFFFFF" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
113     stroke-linejoin="round" d="M94.5,103.8 c-0.6,4-3.8,8.9-9.4,14.7c-2.6-1.8-5-3.7-7.2-5.7c-2.5,_
114     4.1-6.6,8.8-12.2,14c-1.9-2.2-3.4-4.5-4.5-6.9c-4.4,3.3-9.5,6.9-15.4,10.8
115     c-0.2-3.4,0.1-7.1,1.1-10.9" />
116   <path fill="#FFFFFF" stroke="#995d03" stroke-width="2.5" stroke-linecap="round"
117     stroke-linejoin="round" d="M97.5,63.9 c-1.7-2.4-5.9-4.1-12.4-5.2c-0.9,2.2-1.8,4.3-2.5,6.5c-3
118     .8-1.8-9.4-3.1-17-3.8c0.5,2.3,1.2,4.5,1.9,6.8c-5-0.6-11.2-0.9-18.4-1 c2,2.9,0.9,3.5,3.9,6.2"
119     />
120   </g>
121   <g class="armR" [ngStyle]={`${ visibility: 'hidden' }`}>
122     <path fill="#d4c9c0" stroke="#995d03" stroke-linecap="round" stroke-linejoin="round"
123       stroke-miterlimit="10" stroke-width="2.5" d="M265.4 97.3110.4-38.6-38.9-10.5-20 36.1z" />
124     <path fill="#d4c9c0" stroke="#995d03" stroke-linecap="round" stroke-linejoin="round"
125       stroke-miterlimit="10" stroke-width="2.5" d="M252.4 52.4L233 47.2c-2.7-.7-5.4-9-6.1 3.5-.7
126       2.7.9 5.4 3.5 6.1110.3 2.8M226 76.41-19.4-5.2c-2.7-.7-5.4-9-6.1 3.5-.7 2.7.9 5.4 3.5 6.1118.3
127       4.9M228.4 66.71-23.1-6.2c-2.7-.7-5.4-9-6.1 3.5-.7 2.7.9 5.4 3.5 6.1123.1 6.2M235.8
128       58.31-26.7c-2.7-.7-5.4-9-6.1 3.5-.7 2.7.9 5.4 3.5 6.1121.3 5.7" />
129     <path fill="#d4c9c0" d="M207.9 74.71-2.2-6c-1.1-.3-2.2.3-2.4 1.4-.3 1.1.3 2.2 1.4 2.412.2.6
130       1-3.8zM206.7 641-2.2-6c-1.1-.3-2.2.3-2.4 1.4-.3 1.1.3 2.2 1.4 2.412.2.6 1-3.8zM211.2
131       54.81-2.2-6c-1.1-.3-2.2.3-2.4 1.4-.3 1.1.3 2.2 1.4 2.412.2.6 1-3.8zM234.6
132       49.41-2.2-6c-1.1-.3-2.2.3-2.4 1.4-.3 1.1.3 2.2 1.4 2.412.2.6 1-3.8z" />
133     <path fill="#fff" stroke="#995d03" stroke-linecap="round" stroke-linejoin="round"
134       stroke-width="2.5" d="M263.3 96.7c41.4 14.9 84.1 30.7 108.2 35.5l14-52.3C352 70 313.6 63.5
135       273.6 58.1" />
136     <path fill="#fff" stroke="#995d03" stroke-linecap="round" stroke-linejoin="round"
137       stroke-width="2.5" d="M278.2 59.31-18.6-10 2.5 11.9-10.7 6.5 9.9 8.7-13.9 6.4 9.1 5.9-13.2
138       9.2 23.1-.9M284.5 100.1c-.4 4 1.8 8.9 6.7 14.8 3.5-1.8 6.7-3.6 9.7-5.5 1.8 4.2 5.1 8.9 10.1
139       14.1 2.7-2.1 5.1-4.4 7.1-6.8 4.1 3.4 9 7 14.7 11 1.2-3.4 1.8-7 1.7-10.9M314 66.7s5.4-5.7
140       12.6-7.4c1.7 2.9 3.3 5.7 4.9 8.6 3.8-2.5 9.8-4.4 18.2-5.7.1 3.1.1 6.1 0 9.2 5.5-1 12.5-1.6
141       20.8-1.9-1.4 3.9-2.5 8.4-2.5 8.4" />
142   </g>
143   </g>
144 </svg>

```

```
103     </div>
104   </div>
105
106   <div class="inputGroup inputGroup1">
107     <label for="loginEmail" id="loginEmailLabel">Email</label>
108     <input type="email" id="loginEmail" formControlName="username" maxlength="254" />
109   </div>
110   <div class="inputGroup inputGroup2">
111     <label for="loginPassword" id="loginPasswordLabel">Contraseña</label>
112     <input type="password" formControlName="password" id="loginPassword" />
113     <label id="showPasswordToggle" for="showPasswordCheck">Ver
114       <input id="showPasswordCheck" type="checkbox" />
115       <div class="indicator"></div>
116     </label>
117   </div>
118   <div class="inputGroup inputGroup3">
119
120     <button id="login" type="submit">Acceder</button>
121     <mat-error *ngIf="authError" class="auth-error">
122       <span>{{ authError }}</span>
123
124     </mat-error>
125     <button id="registro" (click)="goToRegister()">Registrarse</button>
126   </div>
127 </form>
128
129
```

Listado A.34: Sección de registro en register.component.html

```
1 <mat-card-content>
2 <form [formGroup]="registerForm" (ngSubmit)="onSubmit()">
3   <mat-card-header class="custom-card-header">
4
5     <mat-card-title>Formulario de registro</mat-card-title>
6   </mat-card-header>
7   <div class="inputGroup inputGroup1">
8     <mat-form-field appearance="fill">
9       <label for="loginNombre" id="loginNombreLabel">Nombre</label>
10      <input matInput type="text" id="loginNombre" formControlName="nombre" maxlength="254" />
11      <mat-error *ngIf="registerForm.get('nombre')?.hasError('required') &&
12        registerForm.get('nombre')?.touched">
13        El nombre es obligatorio
14      </mat-error>
15      <mat-error *ngIf="registerForm.get('nombre')?.hasError('minlength') &&
16        registerForm.get('nombre')?.touched">
17        El nombre debe tener al menos 4 caracteres
18      </mat-error>
19    </mat-form-field>
20  </div>
21
22  <div class="inputGroup inputGroup1">
23    <mat-form-field appearance="fill">
24      <label for="loginApellidos" id="loginEmailLabel">Apellidos</label>
25      <input matInput type="text" id="loginApellidos" formControlName="apellidos" maxlength="254" />
26      <mat-error *ngIf="registerForm.get('apellidos')?.hasError('required') &&
27        registerForm.get('apellidos')?.touched">
28        Los apellidos son obligatorios
29      </mat-error>
30      <mat-error *ngIf="registerForm.get('apellidos')?.hasError('minlength') &&
31        registerForm.get('apellidos')?.touched">
32        Los apellidos deben tener al menos 4 caracteres
33      </mat-error>
34    </mat-form-field>
35  </div>
36
37  <div class="inputGroup inputGroup1">
38    <mat-form-field appearance="fill">
39      <label for="loginEmail" id="loginEmailLabel">Email</label>
40      <input matInput type="email" id="loginEmail" formControlName="email" maxlength="254" />
41      <mat-error *ngIf="registerForm.get('email')?.hasError('required') &&
42        registerForm.get('email')?.touched">
```

```
40     El email es obligatorio
41   </mat-error>
42   <mat-error *ngIf="registerForm.get('email')?.hasError('email') &&
43     registerForm.get('email')?.touched">
44     El email no tiene un formato válido
45   </mat-error>
46 </mat-form-field>
47 </div>
48
49 <div class="inputGroup inputGroup1">
50   <mat-form-field appearance="fill">
51     <label for="loginTelefono" id="loginTelefonoLabel">Teléfono</label>
52     <input matInput type="number" id="loginTelefono" formControlName="telefono" maxlength="25" />
53     <mat-error *ngIf="registerForm.get('telefono')?.hasError('required') &&
54       registerForm.get('telefono')?.touched">
55       El teléfono es obligatorio
56     </mat-error>
57     <mat-error *ngIf="registerForm.get('telefono')?.hasError('pattern') &&
58       registerForm.get('telefono')?.touched">
59       El teléfono debe tener un formato válido
60     </mat-error>
61   </mat-form-field>
62 </div>
63
64 <div class="inputGroup inputGroup1">
65   <mat-form-field appearance="fill">
66     <label for="loginNif" id="loginNifLabel">DNI/NIF</label>
67     <input matInput type="text" id="loginNif" formControlName="dni" maxlength="10" />
68     <mat-error *ngIf="registerForm.get('dni')?.hasError('required') &&
69       registerForm.get('dni')?.touched">
70       El DNI/NIF es obligatorio
71     </mat-error>
72     <mat-error *ngIf="registerForm.get('dni')?.hasError('pattern') &&
73       registerForm.get('dni')?.touched">
74       El DNI/NIF debe tener un formato válido
75     </mat-error>
76   </mat-form-field>
77 </div>
78
79 <div class="inputGroup inputGroup1">
80   <mat-form-field appearance="fill">
81     <label for="loginDireccion" id="loginDireccionLabel">Dirección</label>
82     <input matInput type="text" id="loginDireccion" formControlName="direccion" maxlength="254" />
83     <mat-error *ngIf="registerForm.get('direccion')?.hasError('required') &&
84       registerForm.get('direccion')?.touched">
85       La dirección es obligatoria
86     </mat-error>
87     <mat-error *ngIf="registerForm.get('direccion')?.hasError('minlength') &&
88       registerForm.get('direccion')?.touched">
89       La dirección debe tener al menos 10 caracteres
90     </mat-error>
91   </mat-form-field>
92 </div>
93
94 <div class="inputGroup inputGroup1">
95   <mat-form-field appearance="fill">
96     <label for="loginFecha" id="loginFechaLabel">Fecha de nacimiento</label>
97     <input matInput type="date" id="loginFecha" formControlName="fechaNacimiento" name="fecha"
98       class="styled-date">
99
100    <mat-error *ngIf="registerForm.get('fechaNacimiento')?.hasError('required') &&
101      registerForm.get('fechaNacimiento')?.touched">
102      La fecha de nacimiento es obligatoria
103    </mat-error>
104
105    </mat-form-field>
106 </div>
107
108 <div class="inputGroup inputGroup2">
109   <mat-form-field appearance="fill">
110     <label for="loginPassword" id="loginPasswordLabel">Contraseña</label>
111     <input matInput type="password" id="loginPassword" formControlName="password" />
112     <mat-error *ngIf="registerForm.get('password')?.hasError('required') &&
113       registerForm.get('password')?.touched">
```

```

105     La contraseña es obligatoria
106     </mat-error>
107     <mat-error *ngIf="registerForm.get('password')?.hasError('minlength') &&
108       registerForm.get('password')?.touched">
109       La contraseña debe tener al menos 6 caracteres
110     </mat-error>
111   </mat-form-field>
112 </div>
113
114   <button class="boton" type="submit">Registrarse</button>
115
116 </form>
117 </mat-card-content>
```

A.8. Ficheros Docker del backend de la aplicación

Listado A.35: Dockerfile del backend de la aplicación para la API de extracción web

```

1 FROM eclipse-temurin:24-jdk-alpine-3.20
2 WORKDIR /app
3 COPY *.jar extraccion.jar
4 EXPOSE 8080
5 CMD ["java", "-Dspring.profiles.active=despliegue", "-jar", "extraccion.jar"]
```

Listado A.36: Dockerfile del backend de la aplicación para la API de publicaciones

```

1 FROM eclipse-temurin:24-jdk-alpine-3.20
2 WORKDIR /app
3 COPY *.jar publicacionesapi.jar
4 EXPOSE 8080
5 CMD ["java", "-Dspring.profiles.active=despliegue", "-jar", "publicacionesapi.jar"]
```

Listado A.37: Dockerfile del backend de la aplicación para la API de tiempo

```

1 FROM eclipse-temurin:24-jdk-alpine-3.20
2 WORKDIR /app
3 COPY *.jar tiempo.jar
4 EXPOSE 8080
5 CMD ["java", "-Dspring.profiles.active=despliegue", "-jar", "tiempo.jar"]
```

A.9. Test unitarios del backend de la aplicación

Listado A.38: Test unitario de la clase ReservaService en ReservaServiceTest.java

```

23 import java.time.LocalDate;
24
25 public class ReservaServiceTest {
26     private ReservaRepository reservaRepository;
27     private EmailService emailService;
28     private UserRepository userRepository;
29     private RoleUserRepository roleUserRepository;
30     private RoleRepository roleRepository;
31
32     private ReservaService reservaService;
33
34     @BeforeEach
35     public void setUp() {
36         reservaRepository = mock(ReservaRepository.class);
37         emailService = mock(EmailService.class);
38         userRepository = mock(UserRepository.class);
```

```

39     roleUserRepository = mock(RoleUserRepository.class);
40     roleRepository = mock(RoleRepository.class);
41
42     reservaService = new ReservaService(reservaRepository, emailService, userRepository,
43                                         roleUserRepository, roleRepository);
44 }
45
46 @Test
47 public void testConfirmarReserva_usuarioExiste_reservaConfirmada() {
48     ReservaDTO dto = new ReservaDTO(null, null, null, null, null, null, null, 0, 0);
49     dto.setStartDate(LocalDate.of(2025, 1, 1));
50     dto.setEndDate(LocalDate.of(2025, 1, 3));
51     dto.setNumPersonas(2);
52     dto.setPrecio(200.0);
53     dto.setEmail("cliente@email.com");
54     dto.setNombre("Juan");
55     dto.setDni("12345678A");
56     dto.setTelefono("600000000");
57     dto.setEmailUsuario("cliente@email.com");
58
59     MyUser user = new MyUser();
60     user.setId(99);
61
62     Reserva reservaGuardada = new Reserva();
63     reservaGuardada.setId(123);
64     reservaGuardada.setUsuarioId(99);
65
66     when(userRepository.findByEmail("cliente@email.com")).thenReturn(Mono.just(user));
67     when(reservaRepository.existeReservaEnFechas(dto.getStartDate(), dto.getEndDate()))
68         .thenReturn(Mono.just(false));
69     when(reservaRepository.save(any(Reserva.class))).thenReturn(Mono.just(reservaGuardada));
70     when(emailService.sendEmail(any(Reserva.class))).thenReturn(Mono.empty());
71
72     Integer id = reservaService.confirmarReserva(dto).block();
73
74     assertNotNull(id);
75     assertEquals(123, id);
76     verify(reservaRepository, times(1)).save(any(Reserva.class));
77     verify(emailService, times(1)).sendEmail(reservaGuardada);
78 }
79
80 @Test
81 public void testConfirmarReserva_usuarioNoExiste_lanzaExcepcion() {
82     ReservaDTO dto = new ReservaDTO(null, null, null, null, null, null, null, 0, 0);
83     dto.setEmailUsuario("noexiste@email.com");
84
85     when(userRepository.findByEmail("noexiste@email.com")).thenReturn(Mono.empty());
86
87     RuntimeException ex = assertThrows(RuntimeException.class, () -> {
88         reservaService.confirmarReserva(dto).block();
89     });
90
91     assertEquals("ERROR_USER", ex.getMessage());
92 }
93
94 @Test
95 public void testCancelarReserva_enviaEmail() {
96     Reserva reserva = new Reserva();
97     reserva.setId(1);
98     when(reservaRepository.findById(1)).thenReturn(Mono.just(reserva));
99     when(emailService.sendEmailCancel(reserva, "Cancelación por motivo")).thenReturn(Mono.empty());
100
101    Reserva result = reservaService.cancelarReserva(1, "Cancelación por motivo").block();
102
103    assertNotNull(result);
104    assertEquals(reserva, result);
105    verify(emailService, times(1)).sendEmailCancel(reserva, "Cancelación por motivo");
106 }

```

Listado A.39: Test unitario de la clase ReviewService en ReviewServiceTest.java

```
28
29     private ReviewRepository reviewRepository;
30     private RoleUserRepository roleUserRepository;
31     private RoleRepository roleRepository;
32     private UserRepository userRepository;
33
34     private ReviewService reviewService;
35
36     @BeforeEach
37     void setUp() {
38         reviewRepository = mock(ReviewRepository.class);
39         roleUserRepository = mock(RoleUserRepository.class);
40         roleRepository = mock(RoleRepository.class);
41         userRepository = mock(UserRepository.class);
42
43         reviewService = new ReviewService();
44         reviewService.reviewRepository = reviewRepository;
45         reviewService.roleRepository = roleRepository;
46         reviewService.roleUserRepository = roleUserRepository;
47         reviewService.userRepository = userRepository;
48     }
49
50     @Test
51     void testSaveReview_guardadoCorrectamente() {
52         // Datos simulados
53         ReviewDTO dto = new ReviewDTO();
54         dto.setReview("Muy buena experiencia");
55         dto.setRate(4);
56         dto.setRateLimpieza(5);
57         dto.setRateServicios(4);
58         dto.setRateUbicacion(3);
59
60         MyUser user = new MyUser();
61         user.setId(1);
62         user.setEmail("test@correo.com");
63
64         Role role = new Role();
65         role.setId(2);
66         role.setName("ROLE_CLIENTE");
67
68         Review savedReview = new Review();
69         savedReview.setReview(dto.getReview());
70         savedReview.setUserId(1);
71
72         // Mocks de seguridad
73         Authentication authentication = mock(Authentication.class);
74         when(authentication.getPrincipal()).thenReturn("test@correo.com");
75
76         SecurityContext securityContext = mock(SecurityContext.class);
77         when(securityContext.getAuthentication()).thenReturn(authentication);
78
79         try (MockedStatic<ReactiveSecurityContextHolder> contextHolder =
80             mockStatic(ReactiveSecurityContextHolder.class)) {
81             contextHolder.when(ReactiveSecurityContextHolder::getContext)
82                 .thenReturn(Mono.just(securityContext));
83
84             // Mocks de repositorios
85             when(userRepository.findByEmail("test@correo.com")).thenReturn(Mono.just(user));
86             when(roleRepository.findByName("ROLE_CLIENTE")).thenReturn(Mono.just(role));
87             when(roleUserRepository.deleteByUserIdAndRoleId(1, 2)).thenReturn(Mono.empty());
88             when(reviewRepository.save(any(Review.class))).thenReturn(Mono.just(savedReview));
89
90             Review result = reviewService.saveReview(dto).block();
91
92             assertNotNull(result);
93             assertEquals(dto.getReview(), result.getReview());
94             assertEquals(1, result.getUserId());
95         }
96
97         @Test
98         void testSaveReview_usuarioNoExiste_lanzaExpcion() {
99             ReviewDTO dto = new ReviewDTO();
100            dto.setReview("Experiencia mala");
101            dto.setRate(1);
```

```

102
103     Authentication authentication = mock(Authentication.class);
104     when(authentication.getPrincipal()).thenReturn("noexiste@correo.com");
105
106     SecurityContext securityContext = mock(SecurityContext.class);
107     when(securityContext.getAuthentication()).thenReturn(authentication);
108
109     try (MockedStatic<ReactiveSecurityContextHolder> contextHolder =
110          mockStatic(ReactiveSecurityContextHolder.class)) {
111         contextHolder.when(ReactiveSecurityContextHolder::getContext)
112             .thenReturn(Mono.just(securityContext));
113
114         when(userRepository.findByEmail("noexiste@correo.com")).thenReturn(Mono.empty());
115
116         RuntimeException exception = assertThrows(RuntimeException.class,
117               () -> reviewService.saveReview(dto).block());
118
119         assertEquals("User not found", exception.getMessage());
120     }
121
122     @Test
123     void testGetReviews_ordenCorrecto() {
124         Review r1 = new Review();
125         r1.setReview("Reseña 1");
126
127         Review r2 = new Review();
128         r2.setReview("Reseña 2");
129
130         when(reviewRepository.findAllByOrderByCreationDateDesc()).thenReturn(Flux.just(r2, r1));
131
132         List<Review> reviews = reviewService.getReviews().collectList().block();
133
134         assertNotNull(reviews);
135         assertEquals(2, reviews.size());
136         assertEquals("Reseña 2", reviews.get(0).getReview()); // orden descendente esperado
137     }
138 }
```

Listado A.40: Test unitario de la clase UserService en `UserServiceTest.java`

```

19 class UserServiceTest {
20
21     private UserRepository userRepository;
22     private RoleUserRepository roleUserRepository;
23     private RoleRepository roleRepository;
24
25     private UserService userService;
26
27     @BeforeEach
28     void setUp() {
29         userRepository = mock(UserRepository.class);
30         roleUserRepository = mock(RoleUserRepository.class);
31         roleRepository = mock(RoleRepository.class);
32
33         userService = new UserService();
34         userService.userRepository = userRepository;
35         userService.roleUserRepository = roleUserRepository;
36         userService.roleRepository = roleRepository;
37     }
38
39     @Test
40     void testGetNombre_usuarioExistente() {
41         String email = "test@correo.com";
42         MyUser user = new MyUser();
43         user.setEmail(email);
44
45         when(userRepository.findByEmail(email)).thenReturn(Mono.just(user));
46
47         MyUser result = userService.getNombre(email).block();
48
49         assertNotNull(result);
50         assertEquals(email, result.getEmail());
```

```

51    }
52
53    @Test
54    void testGetNombre_usuarioNoExiste() {
55        String email = "noexiste@correo.com";
56
57        when(userRepository.findByEmail(email)).thenReturn(Mono.empty());
58
59        MyUser result = userService.getNombre(email).block();
60
61        assertNull(result);
62    }
63
64    @Test
65    void testGetRoles_retornaRolesCorrectamente() {
66        String email = "usuario@correo.com";
67        MyUser user = new MyUser();
68        user.setId(1);
69        user.setEmail(email);
70
71        UserRole ur1 = new UserRole();
72        ur1.setRoleId(100);
73        UserRole ur2 = new UserRole();
74        ur2.setRoleId(200);
75
76        Role r1 = new Role();
77        r1.setId(100);
78        r1.setName("ROLE_USER");
79        Role r2 = new Role();
80        r2.setId(200);
81        r2.setName("ROLE_ADMIN");
82
83        when(userRepository.findByEmail(email)).thenReturn(Mono.just(user));
84        when(roleUserRepository.findById(1)).thenReturn(Flux.just(ur1, ur2));
85        when(roleRepository.findById(100)).thenReturn(Mono.just(r1));
86        when(roleRepository.findById(200)).thenReturn(Mono.just(r2));
87
88        List<Role> roles = userService.getRoles(email).collectList().block();
89
90        assertNotNull(roles);
91        assertEquals(2, roles.size());
92        assertEquals("ROLE_USER", roles.get(0).getName());
93        assertEquals("ROLE_ADMIN", roles.get(1).getName());
94    }
95
96    @Test
97    void testGetRoles_usuarioNoExiste() {
98        String email = "invalido@correo.com";
99
100       when(userRepository.findByEmail(email)).thenReturn(Mono.empty());
101
102       List<Role> roles = userService.getRoles(email).collectList().block();
103
104       assertNotNull(roles);
105       assertTrue(roles.isEmpty());
106    }
107}

```

Listado A.41: Test unitario de la clase SignupService en SignupServiceTest.java

```

19 public class SignupServiceTest {
20
21     private UserRepository userRepository;
22     private RoleUserRepository roleUserRepository;
23     private PasswordEncoder passwordEncoder;
24
25     private SignupService signupService;
26
27     @BeforeEach
28     void setUp() {
29         userRepository = mock(UserRepository.class);
30         roleUserRepository = mock(RoleUserRepository.class);
31         passwordEncoder = mock(PasswordEncoder.class);

```

```
32     signupService = new SignupService(userRepository, roleUserRepository, passwordEncoder);
33 }
34
35 private MyUserDTO crearUsuarioDTO(String email) {
36     MyUserDTO dto = new MyUserDTO();
37     dto.setEmail(email);
38     dto.setPassword("pass123");
39     dto.setNombre("Juan");
40     dto.setApellidos("Pérez");
41     dto.setTelefono("123456789");
42     dto.setDni("12345678A");
43     dto.setFechaNacimiento(LocalDate.of(2000, 1, 1));
44     dto.setDireccion("Calle Falsa 123");
45     return dto;
46 }
47
48 @Test
49 void testRegisterUser_emailYaRegistrado_lanzaExcepcion() {
50     String email = "usuario@correo.com";
51     MyUserDTO dto = crearUsuarioDTO(email);
52     MyUser existing = new MyUser();
53     existing.setEmail(email);
54
55     when(userRepository.findByEmail(email)).thenReturn(Mono.just(existing));
56
57     Exception ex = assertThrows(Exception.class, () -> signupService.registerUser(dto).block());
58
59     assertTrue(ex.getMessage().contains("ya está en uso"));
60 }
61
62 @Test
63 void testRegisterUser_nuevoUsuario_guardadoCorrectamente() {
64     MyUserDTO dto = crearUsuarioDTO("nuevo@correo.com");
65
66     when(userRepository.findByEmail(dto.getEmail())).thenReturn(Mono.empty());
67     when(passwordEncoder.encode(dto.getPassword())).thenReturn("encodedpass");
68
69     MyUser saved = new MyUser();
70     saved.setId(1);
71     when(userRepository.save(any(MyUser.class))).thenReturn(Mono.just(saved));
72
73     Role role = new Role();
74     role.setId(10);
75     when(userRepository.findFirstByName("ROLE_USER")).thenReturn(Mono.just(role));
76
77     when(roleUserRepository.save(any(UserRole.class))).thenReturn(Mono.just(new UserRole()));
78
79     Object result = signupService.registerUser(dto).block();
80
81     assertNotNull(result);
82     verify(userRepository).save(any(MyUser.class));
83     verify(roleUserRepository).save(any(UserRole.class));
84 }
85
86 @Test
87 void testRegisterAdmin_nuevoAdmin_guardadoCorrectamente() {
88     MyUserDTO dto = crearUsuarioDTO("admin@correo.com");
89
90     when(userRepository.findByEmail(dto.getEmail())).thenReturn(Mono.empty());
91     when(passwordEncoder.encode(dto.getPassword())).thenReturn("encodedpass");
92
93     MyUser saved = new MyUser();
94     saved.setId(2);
95     when(userRepository.save(any(MyUser.class))).thenReturn(Mono.just(saved));
96
97     Role role = new Role();
98     role.setId(20);
99     when(userRepository.findFirstByName("ROLE_ADMIN")).thenReturn(Mono.just(role));
100
101    when(roleUserRepository.save(any(UserRole.class))).thenReturn(Mono.just(new UserRole()));
102
103    Object result = signupService.registerAdmin(dto).block();
104
105    assertNotNull(result);
106    verify(userRepository).save(any(MyUser.class));
```

```

107     verify(roleUserRepository).save(any(UserRole.class));
108 }
109
110 @Test
111 void testRegisterAdmin_emailYaRegistrado_lanzaExcepcion() {
112     String email = "admin@correo.com";
113     MyUserDTO dto = crearUsuarioDTO(email);
114     MyUser existing = new MyUser();
115     existing.setEmail(email);
116
117     when(userRepository.findByEmail(email)).thenReturn(Mono.just(existing));
118
119     Exception ex = assertThrows(Exception.class, () -> signupService.registerAdmin(dto).block());
120
121     assertTrue(ex.getMessage().contains("ya está en uso"));
122 }
123 }
```

Listado A.42: Test unitario de la clase MediaService en MediaServiceTest.java

```

13 class MediaServiceTest {
14
15     @InjectMocks
16     private MediaService mediaService;
17
18     @Mock
19     private MediaRepository mediaRepository;
20
21     @Mock
22     private InstagramTokenService instagramTokenRefresher;
23
24     @BeforeEach
25     void setUp() {
26         MockitoAnnotations.openMocks(this);
27         ReflectionTestUtils.setField(mediaService, "userId", "dummyUser");
28     }
29
30
31     @Test
32     void testExtractMediaByHashtag() {
33         Media media1 = new Media();
34         media1.setId(
35             "1");
36         media1.setHashtag("playa");
37
38         when(mediaRepository.findByHashtag("playa")).thenReturn(Flux.just(media1));
39
40         StepVerifier.create(mediaService.extractMediaByHashtag("playa"))
41             .expectNext(media1)
42             .verifyComplete();
43     }
44
45     @Test
46     void testExtractAllMedia() {
47         Media media = new Media();
48         media.setId("2");
49
50         when(mediaRepository.findAll()).thenReturn(Flux.just(media));
51
52         StepVerifier.create(mediaService.extractAllMedia())
53             .expectNext(media)
54             .verifyComplete();
55     }
56
57 }
```

Listado A.43: Test unitario de la clase WeatherService en WeatherServiceTest.java

```

17 class WeatherServiceTest {
18
19     @Mock
```

```
20     private WeatherDataRepository weatherDataRepository;
21
22     @InjectMocks
23     private WeatherService weatherService;
24
25     @BeforeEach
26     void setup() {
27         MockitoAnnotations.openMocks(this);
28     }
29
30     @Test
31     void testClassifyDay() {
32         String result1 = invokeClassifyDay(26.0, 0.0);
33         String result2 = invokeClassifyDay(8.0, 0.0);
34         String result3 = invokeClassifyDay(18.0, 0.0);
35         String result4 = invokeClassifyDay(15.0, 5.0);
36
37         assert result1.equals("Caluroso");
38         assert result2.equals("Frio");
39         assert result3.equals("Normal");
40         assert result4.equals("Lluvioso");
41     }
42
43     private String invokeClassifyDay(double temp, double prec) {
44         return org.springframework.test.util.ReflectionTestUtils
45             .invokeMethod(weatherService, "classifyDay", temp, prec);
46     }
47
48     @Test
49     void testProcessWeatherResponse_success() {
50         Map<String, Object> response = Map.of(
51             "daily", Map.of(
52                 "time", List.of("2023-04-01"),
53                 "temperature_2m_max", List.of(22.0),
54                 "temperature_2m_min", List.of(12.0),
55                 "precipitation_sum", List.of(0.0)
56             )
57         );
58
59         Mono<WeatherData> result = org.springframework.test.util
60             .ReflectionTestUtils.invokeMethod(weatherService, "processWeatherResponse", response);
61
62         StepVerifier.create(result)
63             .expectNextMatches(data -> data.getClassification().equals("Normal"))
64             .verifyComplete();
65     }
66
67     @Test
68     void testProcessWeatherResponse_missingData() {
69         Map<String, Object> response = Map.of(
70             "daily", Map.of(
71                 "time", List.of(),
72                 "temperature_2m_max", List.of(),
73                 "temperature_2m_min", List.of(),
74                 "precipitation_sum", List.of()
75             )
76         );
77
78         Mono<WeatherData> result = org.springframework.test.util
79             .ReflectionTestUtils.invokeMethod(weatherService, "processWeatherResponse", response);
80
81         StepVerifier.create(result)
82             .expectErrorMatches(err -> err.getMessage().contains("No se han encontrado datos"))
83             .verify();
84     }
85
86     @Test
87     void testCheckAndFetchWeather_foundInRepo() {
88         LocalDate date = LocalDate.now();
89         String dateStr = date.format(java.time.format.DateTimeFormatter.ofPattern("yyyy-MM-dd"));
90         WeatherData mockData = new WeatherData(dateStr, 10, 20, 0, "Normal");
91
92         when(weatherDataRepository.findByDate(dateStr)).thenReturn(Mono.just(mockData));
93
94         StepVerifier.create(weatherService.getWeatherForDateRange(date, date))
```

```

95         .expectNextMatches(wd -> wd.getClassification().equals("Normal"))
96         .verifyComplete();
97
98     verify(weatherDataRepository).findByDate(dateStr);
99 }
100 }
```

Listado A.44: Test unitario de la clase RecursoService en RecursoServiceTest.java

```

16 @ExtendWith(SpringExtension.class)
17 class RecursoServiceTest {
18
19     @InjectMocks
20     private RecursoService recursoService;
21
22     @Mock
23     private ScraperService scraperService;
24
25     @Mock
26     private RecursoRepository recursoRepository;
27
28     private final String urlChelva = "https://chelva.test/";
29     private final String urlTuejar = "https://tuejar.test/";
30     private final String urlAytuejar = "https://aytuejar.test/";
31
32     @BeforeEach
33     void setUp() {
34         ReflectionTestUtils.setField(recursoService, "urlChelva", urlChelva);
35         ReflectionTestUtils.setField(recursoService, "urlTuejar", urlTuejar);
36         ReflectionTestUtils.setField(recursoService, "urlAyunt", urlAytuejar);
37     }
38
39     @Test
40     void testFetchAndStoreData() {
41         when(scraperService.scrape(anyString(), anyString(), anyString()))
42             .thenReturn(Mono.empty());
43
44         when(recursoRepository.findAll()).thenReturn(Flux.empty());
45
46         recursoService.fetchandstoreData();
47
48         verify(scraperService).scrape(urlChelva + "monumentos", "monumentos", "chelva");
49         verify(scraperService).scrape(urlChelva + "aldeas", "aldeas", "chelva");
50         verify(scraperService).scrape(urlTuejar + "patrimonio", "patrimonio", "tuejar");
51         verify(scraperService).scrape(urlTuejar + "naturaleza", "naturaleza", "tuejar");
52         verify(scraperService).scrape(urlAytuejar + "agenda-festividades", "fiestas", "tuejar");
53
54         verify(recursoRepository).findAll();
55     }
56
57     @Test
58     void testExtractAllResources() {
59         Recurso recurso = new Recurso();
60         when(recursoRepository.findAll()).thenReturn(Flux.just(recurso));
61
62         recursoService.extractAllResources().collectList().block();
63
64         verify(recursoRepository).findAll();
65     }
66
67     @Test
68     void testExtractFiestas() {
69         Recurso recurso = new Recurso();
70         when(recursoRepository.findByCategoria("fiestas")).thenReturn(Flux.just(recurso));
71
72         recursoService.extractFiestas().collectList().block();
73
74         verify(recursoRepository).findByCategoria("fiestas");
75     }
76 }
```

Listado A.45: Test funcional E2E del menú de gestión de reservas

```

1 | describe('Pruebas de la página de reservas', () => {
2 |
3 |     beforeEach(() => {
4 |         cy.visit('http://localhost:4200/login');
5 |
6 |         cy.get('#loginEmail').type('alberto.hm.99@gmail.com');
7 |         cy.get('#loginPassword').type('alberto12345');
8 |         cy.get('button[type="submit"]').click();
9 |         cy.wait(4000);
10 |        cy.visit('http://localhost:4200/administrador');
11 |    });
12 |
13 |    it('debería mostrar la tabla de reservas con encabezados correctos', () => {
14 |        // Verifica que la tabla está visible
15 |        cy.get('mat-table').should('be.visible');
16 |
17 |        // Verifica que los encabezados existen
18 |        cy.get('mat-header-cell').then((headers) => {
19 |            const texts = [...headers].map(h => h.textContent?.trim());
20 |            expect(texts).to.include.members([
21 |                'Nombre',
22 |                'Nº Personas',
23 |                'Estado',
24 |                'Fechas',
25 |                'Acciones'
26 |            ]);
27 |        });
28 |    });
29 |
30 |    it('debería tener al menos una fila de datos', () => {
31 |        cy.get('mat-row').should('have.length.greaterThan', 0);
32 |    });
33 |
34 |    it('debería mostrar los botones adecuados según el estado de la reserva', () => {
35 |        // Iteramos sobre las filas
36 |        cy.get('mat-row').each(($row) => {
37 |            cy.wrap($row).within(() => {
38 |                cy.get('mat-cell').eq(2).invoke('text').then((estado) => {
39 |                    const estadoTrim = estado.trim();
40 |                    if (estadoTrim === 'PAGADA' || estadoTrim === 'FINALIZADA') {
41 |                        cy.get('button mat-icon').should('contain.text', 'check_circle');
42 |                    } else if (estadoTrim === 'CANCELADA') {
43 |                        cy.get('button mat-icon').should('contain.text', 'error');
44 |                    } else {
45 |                        cy.get('button mat-icon').then((icons) => {
46 |                            const textos = [...icons].map(i => i.textContent?.trim());
47 |                            expect(textos).to.include.members(['check', 'close']);
48 |                        });
49 |                    }
50 |                });
51 |            });
52 |        });
53 |    });
54 |
55 |    it('debería aprobar una reserva CONFIRMADA, rellenar el motivo y confirmar', () => {
56 |        // 1. Clic sobre el botón "check" de una reserva "CONFIRMADA"
57 |        cy.get('mat-row')
58 |            .contains('CONFIRMADA')
59 |            .parents('mat-row')
60 |            .within(() => {
61 |                cy.get('button')
62 |                    .find('mat-icon')
63 |                    .contains('check')
64 |                    .click({ force: true });
65 |            });
66 |
67 |        // 2. Esperar a que se abra el diálogo y llenar el motivo
68 |        cy.get('mat-dialog-container') // también puede ser mat-mdc-dialog-container
69 |            .should('exist')
70 |            .within(() => {
71 |                // Escribe en el textarea
72 |                cy.get('textarea[matinput]')
73 |                    .should('be.visible')
74 |                    .type('Motivo de prueba desde Cypress', { force: true });
75 |

```

```

76     // 3. Hacer clic en Aceptar
77     cy.contains('button', 'Aceptar')
78         .should('be.visible')
79         .click({ force: true });
80     });
81
82 // 4. Esperar a que se muestre el mensaje de éxito
83 cy.get('.mat-mdc-simple-snack-bar > .mat-mdc-snack-bar-label').contains('Reserva confirmada con éxito!', {
84     timeout: 20000
85 });
86
87
88
89
90
91 });
92

```

Listado A.46: Test funcional E2E del menú de login

```

1 describe('LoginComponent', () => {
2     it('debería iniciar sesión con credenciales válidas', () => {
3         cy.visit('http://localhost:4200/login');
4
5         cy.get('#loginEmail').type('alberto.hm.99@gmail.com');
6         cy.get('#loginPassword').type('alberto12345');
7         cy.get('button[type="submit"]').click();
8
9         // Verifica que redirige a /inicio tras el login
10        cy.url().should('include', '/inicio');
11    });
12
13    it('debería mostrar error si los datos son inválidos', () => {
14        cy.visit('http://localhost:4200/login');
15
16        cy.get('#loginEmail').type('usuarioPrueba');
17        cy.get('#loginPassword').type('contraseña123');
18        cy.get('button[type="submit"]').click();
19
20        cy.contains('El usuario o contraseña no existen').should('be.visible');
21    });
22 });

```

Listado A.47: Test funcional E2E del menú de registro de usuario

```

1 describe('RegisterComponent', () => {
2     const url = 'http://localhost:4200/register'; // ajusta si es diferente
3
4     it('debería registrar un usuario con datos válidos', () => {
5         cy.visit(url);
6
7         cy.get('#loginNombre').type('Juan');
8         cy.get('#loginApellidos').type('Pérez');
9         cy.get('#loginEmail').type('pepito3@yopmail.com');
10        cy.get('#loginTelefono').type('666777888');
11        cy.get('#loginNif').type('12345678Z');
12        cy.get('#loginDireccion').type('Calle Falsa 123');
13        cy.get('#loginFecha')
14            .should('have.attr', 'type', 'date')
15            .invoke('val', '1995-05-10')
16            .trigger('input');
17        cy.get('#loginPassword').type('contrasenaSegura');
18
19        cy.get('button[type="submit"]').click();
20
21
22        cy.url().should('include', '/login');
23    });
24
25    it('debería mostrar errores al dejar los campos vacíos', () => {

```

```

26     cy.visit(url);
27     cy.get('button[type="submit"]').click();
28
29     cy.get('#loginNombre').focus().blur();
30     cy.contains('El nombre es obligatorio').should('be.visible');
31
32   });
33
34   it('debería dar error ya que el email del usuario existe en el sistema', () => {
35     cy.visit(url);
36
37     cy.get('#loginNombre').type('Juan');
38     cy.get('#loginApellidos').type('Pérez');
39     cy.get('#loginEmail').type('alberto.hm.99@gmail.com');
40     cy.get('#loginTelefono').type('666777888');
41     cy.get('#loginNif').type('12345678Z');
42     cy.get('#loginDireccion').type('Calle Falsa 123');
43     cy.get('#loginFecha')
44       .should('have.attr', 'type', 'date')
45       .invoke('val', '1995-05-10')
46       .trigger('input');
47     cy.get('#loginPassword').type('contrasenaSegura');
48     cy.get('button[type="submit"]').click();
49     cy.get('.mat-mdc-simple-snack-bar > .mat-mdc-snack-bar-label')
50       .invoke('text')
51       .should('match', /ya\s*est[aá]\s*en\s*uso/i);
52
53   });
54 });
55

```

Listado A.48: Test funcional E2E del menú de reservas

```

1 describe('Pruebas de la página de reservas', () => {
2
3   beforeEach(() => {
4
5     cy.visit('http://localhost:4200/reservas');
6   });
7
8
9
10  it('Debe mostrar un botón para iniciar sesión cuando el email está vacío', () => {
11    cy.get('p').contains('Inicie sesión o regístrate');
12    cy.get('button').contains('Iniciar sesión / Registrarse');
13  });
14
15  it('Debe mostrar el calendario de reservas cuando el email está presente', () => {
16
17    cy.visit('http://localhost:4200/login'); // Visita la página
18
19    // Simula un login
20    cy.get('#loginEmail').type('alberto.hm.99@gmail.com');
21    cy.get('#loginPassword').type('alberto12345');
22    cy.get('button[type="submit"]').click();
23    cy.wait(2000); // Espera 2 segundos para que se complete la redirección
24    cy.visit('http://localhost:4200/reservas');
25    cy.get('.reservas-card').should('be.visible');
26
27
28
29
30
31  });
32
33  it('Debe mostrar el tiempo para las fechas seleccionadas', () => {
34
35    cy.visit('http://localhost:4200/login'); // Visita la página
36
37    // Simula un login
38    cy.get('#loginEmail').type('alberto.hm.99@gmail.com');
39    cy.get('#loginPassword').type('alberto12345');
40    cy.get('button[type="submit"]').click();

```

```
41     cy.wait(2000); // Espera 2 segundos para que se complete la redirección
42 cy.visit('http://localhost:4200/reservas');
43
44
45     cy.wait(2000); // Espera 2 segundos para que se complete la redirección
46
47 cy.get('.reservas-card').should('be.visible');
48
49
50 cy.get('.calendar-controls > :nth-child(3) > .mdc-button__label').click();
51 cy.get(':nth-child(2) > .cal-cell-row > :nth-child(2) > .cal-cell-top').click();
52 cy.get(':nth-child(2) > .cal-cell-row > :nth-child(3) > .cal-cell-top').click();
53 cy.get('.weather-card')
54   .scrollIntoView(); // Desplazar la vista hasta que el elemento sea visible
55
56   cy.get('.weather-card').should('be.visible');
57 });
58
59 it('Debe mostrar el formulario para completar la reserva rellenado desde el backend', () => {
60
61   cy.visit('http://localhost:4200/login'); // Visita la página
62
63 // Simula un login
64 cy.get('#loginEmail').type('alberto.hm.99@gmail.com');
65   cy.get('#loginPassword').type('alberto12345');
66   cy.get('button[type="submit"]').click();
67   cy.wait(2000); // Espera 2 segundos para que se complete la redirección
68 cy.visit('http://localhost:4200/reservas');
69
70
71   cy.wait(2000); // Espera 2 segundos para que se complete la redirección
72
73 cy.get('.reservas-card').should('be.visible');
74
75
76 cy.get('.calendar-controls > :nth-child(3) > .mdc-button__label').click();
77 cy.get(':nth-child(2) > .cal-cell-row > :nth-child(2) > .cal-cell-top').click();
78 cy.get(':nth-child(2) > .cal-cell-row > :nth-child(3) > .cal-cell-top').click();
79
80   cy.get(':nth-child(2) > .cal-cell-row > :nth-child(3) > .cal-events > .cal-event').click();
81
82   cy.get('.mat-mdc-form-field.ng-tns-c508571215-8 > .mat-mdc-text-field-wrapper >
83     .mat-mdc-form-field-flex > .mat-mdc-form-field-infix')
84     .find('input') // Suponiendo que sea un campo de entrada dentro del contenedor
85     .should('not.have.value', ''); // Verifica que el campo no esté vacío
86   });
87
88 });
89
```