

## SkyStay Hotels



Eduardo Merino Fernández

## ÍNDICE

1. INTRODUCCIÓN.....	4
1.1. Presentación del proyecto	
1.2. Motivación y justificación	
1.3. Objetivos generales y específicos	
1.4. Metodología de desarrollo y herramientas utilizadas	
2. PLANIFICACIÓN Y ANÁLISIS.....	6
2.1. Definición del problema	
2.2. Requisitos del sistema	
2.2.1. Requisitos funcionales	
2.2.2. Requisitos no funcionales	
2.3. Análisis de viabilidad	
2.3.1. Estudio de tecnologías	
2.3.2. Recursos necesarios	
2.3.3. Planificación temporal del desarrollo	
3. DISEÑO DE LA ARQUITECTURA DEL SISTEMA.....	7
3.1. Modelo de arquitectura (cliente-servidor)	
3.2. Diseño de la API REST	
3.2.1. Definición de endpoints	
3.2.2. Seguridad y autenticación (JWT, roles de usuario)	
3.3. Modelado de la base de datos	
3.3.1. Diagrama entidad-relación (ER)	
3.3.2. Estructura y normalización de tablas	
3.4. Diseño del frontend	
4. IMPLEMENTACIÓN DEL BACKEND.....	11
4.1. Configuración del entorno y dependencias (Spring Boot, Maven)	
4.2. Creación del modelo de datos en MariaDB	
4.3. Implementación de controladores y servicios	
4.4. Seguridad y autenticación con Spring Security y JWT	
4.5. Gestión de reservas y usuarios	
4.6. Envío de correos electrónicos con notificaciones	
4.7. Generación de informes y estadísticas	
5. IMPLEMENTACIÓN DEL FRONTEND.....	20
5.1. Configuración del entorno y dependencias (React)	
5.2. Desarrollo de la estructura de componentes	
5.3. Conexión con la API REST	
5.4. Gestión de sesiones y autenticación	

5.5. Diseño de las vistas y experiencia de usuario	
5.6. Implementación de funcionalidades principales	
5.6.1. Consulta de disponibilidad y reservas	
5.6.2. Sistema de fidelización (puntos y recompensas)	
5.6.3. Gestión de reservas y cancelaciones	
6. PRUEBAS Y DEPURACIÓN.....	30
6.1. Pruebas unitarias (JUnit)	
6.2. Pruebas de integración (Postman, Swagger)	
6.3. Pruebas de carga y rendimiento	
6.4. Pruebas de usabilidad y accesibilidad	
6.5. Estrategias de depuración y resolución de errores	
7. DESPLIEGUE Y MANTENIMIENTO.....	32
7.1. Contenedorización con Docker	
7.2. Despliegue del Front	
7.3. Estrategia de mantenimiento y actualizaciones	
8. RESULTADOS Y CONCLUSIONES.....	36
8.1. Comparación entre objetivos iniciales y resultados obtenidos	
8.2. Problemas encontrados y soluciones aplicadas	
8.3. Valoración del rendimiento del sistema	
8.4. Posibles mejoras y ampliaciones futuras	
9. BIBLIOGRAFÍA Y REFERENCIAS.....	41
10. ANEXOS.....	42
10.1. Código fuente relevante	
10.2. Documentación técnica de la API (Swagger)	
10.3. Manual de usuario y administrador	
10.4. Planificación temporal detallada	
10.5. Conclusiones finales	

## 1. INTRODUCCIÓN

### 1.1. Presentación del Proyecto

**SkyStay** es una plataforma web desarrollada para gestionar reservas en una cadena hotelera. El sistema se divide en dos partes:

- **Frontend:** Interfaz web intuitiva para que los usuarios puedan consultar disponibilidad, gestionar reservas y acceder a funcionalidades como el sistema de fidelización.
- **Backend:** Encargado de procesar la lógica del negocio, gestionar los datos y mantener la comunicación con la base de datos mediante una API REST desarrollada en Spring Boot.

El sistema contempla tanto la gestión de reservas por parte de los clientes como funcionalidades administrativas destinadas a los gestores de los hoteles.

---

### 1.2. Motivación y Justificación

En un entorno cada vez más digitalizado, las empresas hoteleras demandan soluciones modernas que agilicen la gestión de sus servicios y mejoren la experiencia del cliente.

SkyStay surge para cubrir esta necesidad ofreciendo:

- Una plataforma unificada y centralizada para cadenas hoteleras.
- Interfaz amigable para el cliente y funcionalidades avanzadas para administradores.
- Sistema de recompensas que mejora la fidelización.

El objetivo es crear un sistema eficaz, seguro y moderno que responda a las exigencias del sector hotelero actual.

---

### 1.3. Objetivos Generales y Específicos

**Objetivo General:** Desarrollar una plataforma completa de gestión de reservas hoteleras que integre frontend, backend y base de datos.

#### Objetivos Específicos:

1. Crear una API REST con Spring Boot para la gestión de reservas.
2. Implementar autenticación y autorización diferenciada para clientes y administradores.
3. Permitir reservas en tiempo real y gestión de recompensas.
4. Desarrollar un frontend responsive e intuitivo en React.
5. Incluir funcionalidades de cancelación y modificación de reservas.
6. Incorporar notificaciones por correo para eventos clave.

7. Asegurar la protección de los datos personales y transacciones.
  8. Generar estadísticas de ocupación para los administradores.
- 

#### 1.4. Metodología de Desarrollo y Herramientas Utilizadas

##### **Metodología:**

Se empleará una metodología ágil basada en **iteraciones semanales** con entregables funcionales, revisiones constantes y pruebas continuas.

##### **Herramientas de Desarrollo:**

- **Backend:** Spring Boot 3.x
  - **Frontend:** React
  - **Base de Datos:** MariaDB
  - **Control de versiones:** Git + GitHub
  - **Pruebas:** JUnit (pruebas unitarias) + Postman (pruebas de API)
  - **Documentación API:** Swagger
  - **Despliegue:** Docker
  - **IDE:** IntelliJ IDEA y Visual Studio Code
- 

## 2. PLANIFICACIÓN Y ANÁLISIS

### 2.1. Definición del Problema

Las cadenas hoteleras que operan en múltiples localizaciones enfrentan dificultades al gestionar reservas de forma centralizada. Los sistemas tradicionales suelen carecer de integración, personalización o escalabilidad. SkyStay pretende solventar estas deficiencias proporcionando una solución web moderna y eficiente.

---

### 2.2. Requisitos del Sistema

#### 2.2.1. Requisitos Funcionales

- Registro y autenticación de usuarios (clientes y administradores).
- Visualización de disponibilidad de habitaciones.
- Realización, cancelación y modificación de reservas.
- Canjeo de recompensas.
- Gestión de habitaciones, servicios y reservas por parte de los administradores.

- Envío automático de correos de confirmación y recordatorios.
- Generación de informes estadísticos para administradores.

### 2.2.2. Requisitos No Funcionales

- Sistema seguro con cifrado de datos sensibles.
  - Accesibilidad desde distintos dispositivos (responsive design).
  - Alta disponibilidad y rendimiento.
  - Trazabilidad y registro de actividades.
  - Despliegue en contenedores para facilitar la escalabilidad.
- 

## 2.3. Análisis de Viabilidad

### 2.3.1. Estudio de Tecnologías

#### **Software:**

- Backend: Spring Boot
- Frontend: React
- Base de datos: MariaDB
- Despliegue: Docker
- Pruebas: JUnit, Postman
- Control de versiones: Git + GitHub
- Documentación: Swagger

#### **Hardware:**

- CPU: 4 núcleos o superior
- RAM: mínimo 8 GB
- Disco: mínimo 50 GB SSD
- Sistema operativo servidor: Windows

### 2.3.2. Recursos Necesarios

- Ordenadores de desarrollo con capacidad adecuada para ejecución local.
- Acceso a servidores para pruebas de despliegue.
- Licencias y entornos de desarrollo (IDE, herramientas de documentación, etc.).

### 2.3.3. Planificación Temporal del Desarrollo

#### **Semana Tarea**

- 1 Análisis de requisitos y diseño de arquitectura
- 2-4 Desarrollo del backend (Spring Boot)

## Semana Tarea

- 5-6 Desarrollo del frontend (Angular)
- 7-8 Integración del backend con el frontend
- 9-10 Pruebas funcionales y corrección de errores
- 11 Despliegue en entorno real (Docker)
- 12 Documentación técnica y preparación de presentación final

## 3. DISEÑO DE LA ARQUITECTURA DEL SISTEMA

### 3.1. Modelo de Arquitectura (Cliente-Servidor)

El sistema SkyStay adopta un modelo de arquitectura **cliente-servidor**, en el cual la lógica de presentación (frontend) y la lógica de negocio (backend) se encuentran desacopladas, permitiendo una comunicación eficiente y escalable mediante peticiones HTTP.

- **Cliente (Frontend):** Aplicación Angular accesible desde navegadores. Se encarga de la interacción con el usuario y el envío de solicitudes a la API.
- **Servidor (Backend):** API REST desarrollada en Spring Boot que gestiona la lógica de negocio, operaciones sobre la base de datos y aplica medidas de seguridad.
- **Base de Datos:** Servidor MariaDB que almacena de forma estructurada toda la información relacionada con usuarios, reservas, habitaciones, etc.

El flujo de información es bidireccional: el cliente realiza peticiones y el servidor responde con datos procesados en formato JSON.

---

### 3.2. Diseño de la API REST

La API REST de SkyStay sigue los principios RESTful para garantizar interoperabilidad, simplicidad y eficiencia. Todas las operaciones están organizadas en endpoints lógicos y cumplen con los verbos HTTP adecuados.

### 3.2.1. Definición de Endpoints (ejemplos)

Recurso	Método Endpoint	Descripción
Usuarios	POST /api/auth/register	Registro de nuevos usuarios
Usuarios	POST /api/auth/login	Autenticación (login)
Habitaciones	GET /api/habitaciones	Obtener lista de habitaciones disponibles
Reservas	POST /api/reservas	Crear una nueva reserva
Reservas	GET /api/reservas/{id}	Consultar detalles de una reserva
Recompensas	PUT /api/recompensas/canjear	Canjear puntos de fidelización
Admin/Informes	GET /api/admin/informes	Obtener estadísticas e informes de ocupación

Todos los endpoints devuelven respuestas en JSON y validan los datos recibidos mediante DTOs y validadores; también habrá otros métodos de las funcionalidades que se incluyan posteriormente en el proyecto, como el saldo del cliente o los puntos.

---

### 3.2.2. Seguridad y Autenticación (JWT, Roles de Usuario)

Para garantizar la seguridad, SkyStay implementa un sistema de autenticación y autorización basado en **JWT (JSON Web Tokens)**:

- Al iniciar sesión correctamente, el servidor genera un **token JWT** firmado que el cliente debe incluir en cada petición (Authorization: Bearer).
- La API verifica este token en cada solicitud protegida.
- Se definen **roles de usuario**:
  - ROLE\_CLIENTE: Puede consultar habitaciones, crear y modificar sus reservas, canjear puntos.
  - ROLE\_ADMIN: Puede acceder a paneles de gestión, informes, modificar disponibilidad y gestionar usuarios.

Se protege cada endpoint con anotaciones como @PreAuthorize y filtros personalizados en Spring Security.

También las contraseñas estarán encriptadas en la base de datos con Spring Security para así evitar filtraciones de datos importantes.

El cliente para confirmar su cuenta tiene que verificar su correo electrónico con el link que se le proporcionará. Esto se realizará mediante una API de correo que tiene diferentes métodos para automatizar el envío de correos.

### 3.3. Modelado de la Base de Datos

El diseño de la base de datos se realiza respetando los principios de normalización para evitar redundancias y garantizar integridad referencial. Se emplea MariaDB como sistema gestor.

#### 3.3.1. Diagrama Entidad-Relación (ER)

El sistema contiene, entre otras, las siguientes entidades principales:

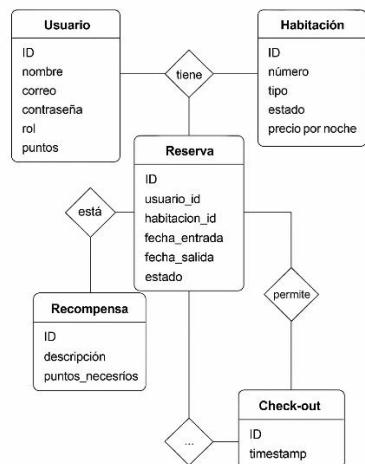
- **Usuario:** ID, nombre, correo, contraseña (hasheada), rol, puntos.
- **Habitación:** ID, número, tipo, estado, precio por noche.
- **Reserva:** ID, usuario\_id, habitacion\_id, fecha\_entrada, fecha\_salida, estado.
- **Recompensa:** ID, descripción, puntos\_necesarios.
- **Check-in / Check-out:** ID, reserva\_id, timestamps asociados.

Las relaciones incluyen:

- Un usuario puede tener muchas reservas.
- Una habitación puede estar asociada a varias reservas a lo largo del tiempo.
- Una reserva puede permitir el canje de una o más recompensas.

El diagrama ER puede ser generado en herramientas como MySQL Workbench o dbdiagram.io.

#### 3.3.1. Diagrama Entidad-Relación (ER)



### 3.3.2. Estructura y Normalización de Tablas

Las tablas están diseñadas siguiendo **hasta la 3<sup>a</sup> Forma Normal (3NF)**:

- Cada tabla tiene una clave primaria única.
  - Las claves foráneas aseguran relaciones entre entidades.
  - No se permite la redundancia de datos.
  - Se utilizan índices para optimizar consultas frecuentes (habitaciones disponibles, historial de reservas, etc.).
- 

## 3.4. Diseño del Frontend

El frontend será una SPA (Single Page Application) desarrollada en React. La comunicación con el backend se realiza mediante HTTP y JSON.

### 3.4.1. Wireframes y Prototipos

Los diseños iniciales se elaboran como wireframes para visualizar la experiencia de usuario.

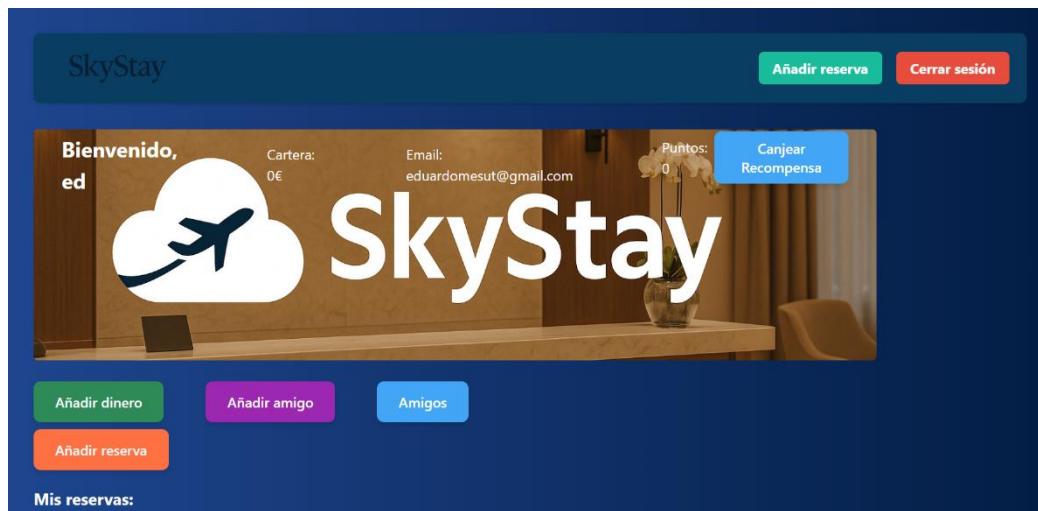
Se incluyen las siguientes pantallas:

- **Página de inicio:** acceso a login y registro.
  - **Página de registro:** Registro completo de cliente o admin.
  - **Panel de cliente:** reservas actuales, historial, sistema de puntos.
  - **Panel de administrador:** gestión de habitaciones, usuarios y reportes.
  - **Formulario de reserva:** selección de fechas y servicios extra.
  - **Confirmaciones y notificaciones:** mediante modales y alertas.
- 

### 3.4.2. UX/UI: Principios Aplicados

Se siguen principios de **Diseño Centrado en el Usuario (UCD)** y buenas prácticas de UI:

- **Consistencia:** estilos y componentes uniformes en toda la plataforma.
- **Feedback inmediato:** mensajes de éxito/error tras cada acción del usuario.
- **Accesibilidad:** contraste adecuado, navegación por teclado y etiquetas ARIA.
- **Responsive Design:** adaptable a móviles, tablets y escritorio.
- **Minimalismo funcional:** interfaz limpia, sin elementos innecesarios.



## 4. IMPLEMENTACIÓN DEL BACKEND

### 4.1. Configuración del entorno y dependencias (Spring Boot, Maven)

Para comenzar con el back, lo primero a realizar es tener un proyecto básico con todas las dependencias necesarias, en este caso en Maven debido a que vamos a utilizar dependencias en Spring y es mas sencillo de esta manera.

En este proyecto se ha implementado mediante el SpringInitialicer que te permite elegir las dependencias que quieras que contenga tu proyecto (Spring Security, Spring Web, Mariadb, DevTools...), con esto ya tenemos la base del proyecto que vamos a realizar.

Para la configuración general del proyecto tenemos el archivo de application.properties que nos encontramos en resources dentro del src, aquí podremos configurar la base de datos que vamos a utilizar y diferentes utilidades que nos da Spring para probar y configurar un proyecto.

### 4.2 Creación del modelo de datos en MariaDB

Una parte muy relevante es la base de datos que vamos a necesitar, en este caso la seleccionada es MariaDB, debido a que necesitamos una base de datos relacional, porque queremos crear tablas diferentes con relaciones entre estas.

En nuestra base de datos tendremos diferentes tablas: Clientes, Administradores, Objetos, Reservas, Hoteles, Habitaciones, Recompensas... Cada una tiene su ID y sus claves foráneas que hace que se relacionen entre si. Esto es relevante porque mediante a nuestro Back que es una API Rest podremos consultar, modificar, crear y eliminar diferentes entradas en a base de datos.(CRUD)

## 4.3 Implementación de controladores, recursos y entidades

Para este proyecto el Back en Spring se ha distribuido de la siguiente manera:

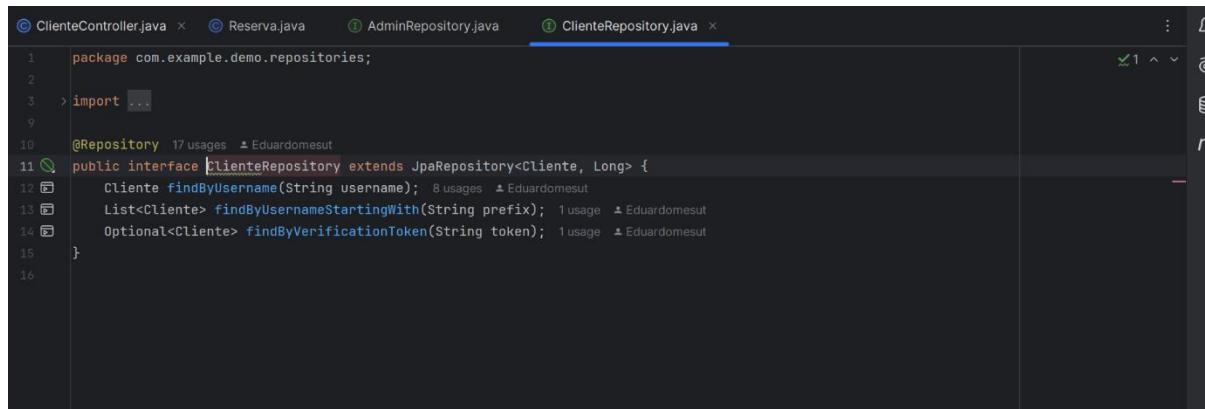
-Primero tenemos las **entidades** que son básicamente los objetos de nuestro proyecto que queremos que Spring los convierta y los trate como tablas de nuestra base de datos con sus diferentes columnas. Aquí hay un ejemplo con el objeto reserva:

```
1 package com.example.demo.entities.objects;
2
3 > import ...
4
5 @Entity
6 @Table(name = "reservas")
7 public class Reserva {
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    private Long id;
11
12    @ManyToOne 4 usages
13    @JoinColumn(name = "cliente_id")
14    @JsonIgnore
15    private Cliente cliente;
16
17    @ManyToOne 10 usages
18    @JoinColumn(name = "habitacion_id")
19    @JsonIgnore
20    private Habitacion habitacion;
21
22    @JsonProperty("numeroHabitacion") // Esto hará que aparezca en el JSON
23    public Integer getNumeroHabitacion() { return habitacion != null ? habitacion.getNumber() : null; }
24
25    @JsonProperty("nombreHotel") // Esto hará que aparezca en el JSON
26    public String getNombreHotel() { return habitacion != null ? habitacion.getHotel().getName() : null; }
27
28
29
30
31
32
33
34
35
```

```
15 public class Reserva {  
38  
39     @  
40         private LocalDate entryDate; 12 usages  
41     @  
42         private LocalDate exitDate; 10 usages  
43     @  
44         private Boolean isPayed; 6 usages  
45     @  
46         private String description; 6 usages  
47     @  
48         private Double price; 7 usages  
49  
50     public Reserva() { ▲ Eduardomesut  
51 }  
52     @  
53         public Reserva(Cliente cliente, Habitacion habitacion, LocalDate entryDate, LocalDate exitDate, Boolean isPayed, String description) {  
54             this.cliente = cliente;  
55             this.habitacion = habitacion;  
56             this.entryDate = entryDate;  
57             this.exitDate = exitDate;  
58             this.isPayed = isPayed;  
59             this.description = description;  
60             this.price = calcularPrecio();  
61     @  
62         public Reserva(Habitacion habitacion, LocalDate entryDate, LocalDate exitDate, Boolean isPayed, String description) { no usages ▲ Eduardomesut  
ng-rest-skystay > src > main > java > com > example > demo > entities > objects > Reserva 15:14 CRLF UTF-8 4 spaces
```

Como vemos para que Spring lo tome como una tabla en la base de datos lo tenemos que declarar como `@Entity`, después en `@Table(name)` se indica el nombre de la tabla. Luego con el `@Id` indicamos la clave primaria de la tabla y que se autoincrementa. También indicamos las claves foráneas con el `@ManyToOne` y el `@OneToMany`, después ponemos los constructores y uno en vacío, los getters y setters y en este caso un método que calcula el precio de la reserva dependiendo de la diferencia de días y del precio del establecimiento.

-Después tenemos los **repositorios** que básicamente contienen los métodos que vamos a utilizar para hacer consultas a la base de datos y que nos devuelva información en forma de Objetos, booleanos... Gracias a Spring se pueden hacer consultas complejas mencionando los atributos del objeto y Spring detecta el propio nombre sin necesidad de hacer ningún método extra. Aquí un ejemplo con el repositorio del Cliente:



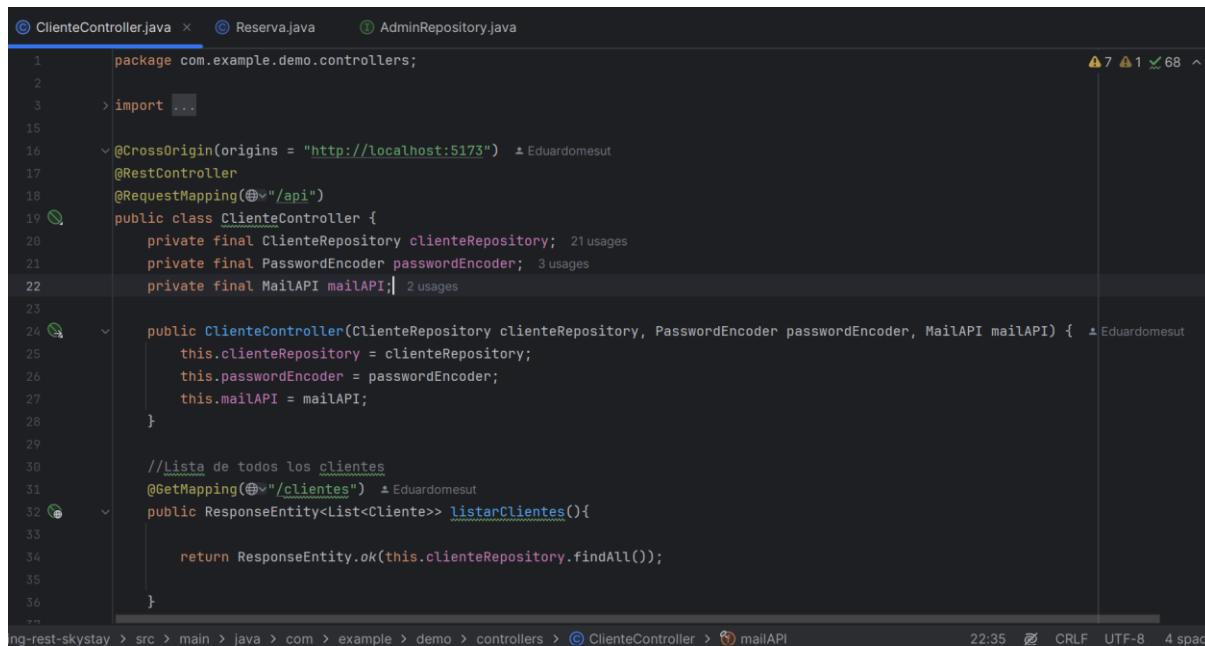
```

1 package com.example.demo.repositories;
2
3 > import ...
4
5
6 @Repository 17 usages ▲ Eduardomesut
7 public interface ClienteRepository extends JpaRepository<Cliente, Long> {
8     Cliente findByUsername(String username); 8 usages ▲ Eduardomesut
9     List<Cliente> findByUsernameStartingWith(String prefix); 1 usage ▲ Eduardomesut
10    Optional<Cliente> findByVerificationToken(String token); 1 usage ▲ Eduardomesut
11 }
12
13
14
15
16

```

Como vemos hay que indicar el **@Repository** para indicarselo a Spring y hacer un **extends** del **JpaRepository** con el Objeto y el tipo de Primary Key.

-Por último tenemos a los **controladores** que básicamente realizan las operaciones CRUD en la base de datos, es donde se conectará el front para hacer las diferentes peticiones para lanzar o recibir información a la API, sirve de intermediario entre el repositorio con la base de datos y el Front. Aquí un ejemplo del controlador del Cliente con las peticiones principales:



```

1 package com.example.demo.controllers;
2
3 > import ...
4
5
6 @CrossOrigin(origins = "http://localhost:5173") ▲ Eduardomesut
7 @RestController
8 @RequestMapping("/**/api")
9 public class ClienteController {
10     private final ClienteRepository clienteRepository; 21 usages
11     private final PasswordEncoder passwordEncoder; 3 usages
12     private final MailAPI mailAPI; 2 usages
13
14     public ClienteController(ClienteRepository clienteRepository, PasswordEncoder passwordEncoder, MailAPI mailAPI) { ▲ Eduardomesut
15         this.clienteRepository = clienteRepository;
16         this.passwordEncoder = passwordEncoder;
17         this.mailAPI = mailAPI;
18     }
19
20     //Lista de todos los clientes
21     @GetMapping("/**/clientes") ▲ Eduardomesut
22     public ResponseEntity<List<Cliente>> listarClientes(){
23
24         return ResponseEntity.ok(this.clienteRepository.findAll());
25
26     }
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
235
236
237
237
238
239
239
240
241
242
242
243
244
244
245
246
246
247
247
248
248
249
249
250
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
13
```

```

① ClienteController.java ② Reserva.java ③ AdminRepository.java

19  public class ClienteController {
37
38      //Info de cliente
39      @GetMapping(@"/clientes/{username}") ✎ Eduardomesut
40      public ResponseEntity<Cliente> infoCliente(@PathVariable String username){
41          Cliente cliente = clienteRepository.findByUsername(username);
42          return ResponseEntity.ok(cliente);
43      }
44      //reservas de cliente
45      @GetMapping(@"/clientes/{username}/reservas") ✎ Eduardomesut
46      public ResponseEntity<List<Reserva>> reservasCliente(@PathVariable String username){
47          Cliente cliente = clienteRepository.findByUsername(username);
48          return ResponseEntity.ok(cliente.getReservas());
49      }
50      //Creación de cliente
51      @PostMapping(@"/clientes") ✎ Eduardomesut
52      public ResponseEntity<Cliente> crearCliente(@RequestBody Cliente cliente){
53          cliente.setPassword(passwordEncoder.encode(cliente.getPassword()));
54          this.mailAPI.enviarCorreoVerificacion(cliente);
55          cliente.setVerified(false);
56          //this.clienteRepository.save(cliente);
57          return ResponseEntity.ok(cliente);
58
59
60      //CORREGIR LO DEL VERTETCADO

```

```

① ClienteController.java ② Reserva.java ③ AdminRepository.java

19  public class ClienteController {
64
65      @GetMapping(@"/api/verificar/{token}") ✎ Eduardomesut
66      public ResponseEntity<String> verificarCuenta(@PathVariable String token) {
67          Optional<Cliente> optional = clienteRepository.findByVerificationToken(token);
68          if (optional.isPresent()) {
69              Cliente cliente = optional.get();
70              cliente.setVerified(true);
71
72              cliente.setVerificationToken(null); // opcional: eliminar token
73              clienteRepository.save(cliente);
74              return ResponseEntity.ok( body: "Cuenta verificada correctamente");
75          } else {
76              return ResponseEntity.badRequest().body("Token inválido o expirado");
77          }
78      }
79
80      //Login del usuario
81      @PostMapping(@"/clientes/login") ✎ Eduardomesut
82      public ResponseEntity<Boolean> inicioSesion(@RequestBody LoginRequest loginRequest){
83          if (clienteRepository.findByUsername(loginRequest.getUser()) != null){
84              Cliente cliente = clienteRepository.findByUsername(loginRequest.getUser());
85              String passwordEncrypt = cliente.getPassword();
86              return ResponseEntity.ok(passwordEncoder.matches(loginRequest.getPassword(), passwordEncrypt));
87          }else{
88              return ResponseEntity.badRequest().body("Usuario no encontrado");
89          }
90      }

```

Angular-REST-Skystay > src > main > java > com > example > demo > controllers > ① ClienteController > ② mailAPI

22:35 CRLF UTF-8

Como se puede observar hay que declarar el `@RestController` para indicar que es un controlador de tipo Rest, y como vemos el resto son métodos cada uno con diferentes consultas, dependiendo del tipo (Get, Post, Delete, Update) se pondrá un tipo u otro de mapping, con la ruta de la petición http. Cada una de estas peticiones se conectará al repositorio necesario y realizará modificaciones en la base de datos u obtendrá información determinada para cada petición.

Con esta base podremos realizar la API Rest y conectarla, en este caso lo que se realizo para la comprobación del funcionamiento fue la utilización de Postman para realizar las peticiones a la API y probar su correcto funcionamiento.

#### 4.4 Seguridad y autenticación con Spring Security

Para asegurar los datos de los clientes y evitar las filtraciones de contraseñas, en este proyecto se ha utilizado Spring Security para encriptar y comparar las contraseñas, de tal manera que se filtra la base de datos con información de los usuarios las contraseñas aparezcan encriptadas. Para esto se han realizado los siguientes métodos:

```
//Creación de cliente
@PostMapping(value = "/clientes") ✎ Eduardomesut
public ResponseEntity<Cliente> crearCliente(@RequestBody Cliente cliente){
    cliente.setPassword(passwordEncoder.encode(cliente.getPassword()));
    this.mailAPI.enviarCorreoVerificacion(cliente);
    cliente.setVerified(false);
    //this.clienteRepository.save(cliente);
    return ResponseEntity.ok(cliente);
```

```
//Login del usuario
@PostMapping(value = "/clientes/login") ✎ Eduardomesut
public ResponseEntity<Boolean> inicioSesion(@RequestBody LoginRequest loginRequest){
    if (clienteRepository.findByUsername(loginRequest.getUser()) != null){
        Cliente cliente = clienteRepository.findByUsername(loginRequest.getUser());
        String passwordEncrypt = cliente.getPassword();
        return ResponseEntity.ok(passwordEncoder.matches(loginRequest.getPassword(), passwordEncrypt));
    }else{
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(false);
    }
}
```

Como se puede observar el método encode dentro del PasswordEncoder codifica la contraseña y el método matches es un booleano que indica si la contraseña coincide con la codificada lo que nos sirve perfectamente para un inicio de sesión del usuario.

## 4.5 Gestión de reservas y usuarios

Para la gestión de reservas se han realizado diferentes métodos para hacer que un usuario no pueda reservar una habitación que este ocupada en ese momento, para ello se ha realizado un método booleano de esta ocupado y para poder reservar una habitación una de las condiciones es que ese método te devuelva false. En dicho método hay que introducir la habitación, el día de entrada y el de salida, lo que se hace es recorrerse estos días comprobando que no hay ningún día de estos que coincida con una reserva, si no coincide ningún día el método retornará false y si coincide retornará true. Con esto podemos comprobar fácilmente en un rango de días si la habitación está o no ocupada.

```
//Hacer reserva
@PostMapping(value = "clientes/{clienteId}/{habitacionId}/reserva") ▾ Eduardomesut
public ResponseEntity<Reserva> crearReserva(@PathVariable Long clienteId,
                                              @PathVariable Long habitacionId,
                                              @RequestBody Reserva reserva) {
    Cliente cliente = clienteRepository.findById(clienteId)
        .orElseThrow(() -> new RuntimeException("Cliente no encontrado"));

    Habitacion habitacion = habitacionRepository.findById(habitacionId)
        .orElseThrow(() -> new RuntimeException("Habitación no encontrada"));

    //Restricción de habitación ocupada esos días
    List<Reserva> reservaList = reservaRepository.findAll();
    for (Reserva res:reservaList) {
        if (res.getHabitacion().getId() == habitacion.getId()){
            if (!res.getExitDate().isBefore(res.getEntryDate()) || res.getEntryDate().isAfter(res.getExitDate())) {
                return ResponseEntity.badRequest().build();
            }
        }
    }
}
```

## 4.6 Envío de correos electrónicos con notificaciones

Un añadido importante al proyecto es el de la API de correos electrónicos. Lo he realizado como una clase que implementa diferentes métodos: Un correo de confirmación al crear tu cuenta, un correo con el código de la recompensa canjeada, una confirmación al realizar una reserva de los datos de dicha reserva...

Esto se realiza gracias a Spring Mail con la que puedes automatizar los correos. Para esto necesitas una cuenta de correo electrónico con el permiso de aplicaciones.

```

① ClienteController.java ② ReservaController.java ③ MailAPI.java × ④ SecurityConfig.java ⑤ Reserva.java

10 import org.springframework.mail.javamail.JavaMailSender;
11 import org.springframework.stereotype.Service;
12
13 import java.util.UUID;
14
15 //Aqui hay que hacer la api con metodos de enviar correo al registrarse con verificación y otro metodo de codigo canjeado
16 //Tambien otro de el envio de la reserva con la informacion al correo al reservar en usuario
17 @Service 9 usages ▲ Eduardomesut
18 public class MailAPI {
19
20     private final ClienteRepository clienteRepository; 2 usages
21
22     public MailAPI(ClienteRepository clienteRepository) { this.clienteRepository = clienteRepository; }
23
24     @Autowired
25     private JavaMailSender mailSender;
26
27     private final String fromEmail = "tuemail@gmail.com"; 3 usages
28
29     // 1. Correo al registrarse con código de verificación
30     public void sendverificationEmail(String to, String verificationCode) { 1 usage ▲ Eduardomesut
31         SimpleMailMessage message = new SimpleMailMessage();
32         message.setFrom(fromEmail);
33         message.setTo(to);
34         message.setSubject("Verifica tu cuenta");
35
36     }

```

```

① ClienteController.java ② ReservaController.java ③ MailAPI.java × ④ SecurityConfig.java ⑤ Reserva.java

18 public class MailAPI {
19
20     // 2. Correo de confirmación al canjear código
21     public void sendCodeRedeemedEmail(String to, String code) { 1 usage ▲ Eduardomesut
22         SimpleMailMessage message = new SimpleMailMessage();
23         message.setFrom(fromEmail);
24         message.setTo(to);
25         message.setSubject("Código canjeado");
26         message.setText("Has canjeado correctamente el la recompensa! Aquí tiene su código para canjearla: " + code);
27         mailSender.send(message);
28     }
29
30     // 3. Correo de confirmación de reserva
31     public void sendReservationEmail(String to, String nombreUsuario, String detallesReserva) { 1 usage ▲ Eduardomesut
32         SimpleMailMessage message = new SimpleMailMessage();
33         message.setFrom(fromEmail);
34         message.setTo(to);
35         message.setSubject("Confirmación de reserva");
36         message.setText("Hola " + nombreUsuario + ",\n\nTu reserva ha sido confirmada.\n\nDetalles:\n" + detallesReserva);
37         mailSender.send(message);
38     }
39
40     @
41     public void enviarCorreoVerificacion(Cliente cliente) { 1 usage ▲ Eduardomesut
42         String token = UUID.randomUUID().toString();
43         cliente.setVerificationToken(token);
44         cliente.setVerified(false);
45     }

```

Como vemos hay métodos que realizan diferentes funciones en este caso tendremos 3. El primero será el mail de confirmación que llegará a la cuenta de correo utilizada para registrarse y que sirve para confirmar el correo de dicha cuenta.

## Verifica tu cuenta ➤ Recibidos x



skystayinfo@gmail.com  
para mí ▾

dom, 4 may, 12:18 (hace 20 horas)



Gracias por registrarte. Tu código de verificación es: Haz clic en el siguiente enlace para verificar tu cuenta:  
<http://localhost:5173/api/verificar/4aba65f7-d09c-4bc9-823c-f6326633c486>

Como vemos este sería un ejemplo del correo que se le enviará cuando se registra, en dicho correo hay un link para confirmar la cuenta en forma de token, cuando se hace click sobre el gracias a una petición del back se confirma la cuenta y se verifica. Gracias a este sistema se asegura que la cuenta que se ha utilizado al registrarse es la cuenta del cliente.

Aparte del anterior tenemos un correo que se envía automáticamente al cliente cada vez que realiza una reserva en uno de los hoteles, a continuación, puede ver un ejemplo donde aparte de la confirmación de la reserva incluye información del hotel reservado, el día de entrada y de salida.



3 de 23.019 < >

## Confirmación de reserva ➤ Recibidos x



skystayinfo@gmail.com  
para mí ▾

dom, 4 may, 12:45 (hace 20 horas)



Hola edu,

Tu reserva ha sido confirmada.

Detalles:

Hotel: Hilton, Día de entrada: 2025-05-09, Día de salida: 2025-05-10

Por último tendremos un correo que se enviará cuando se canjea una recompensa con los puntos del cliente acumulados, en este correo se enviará el código para canjear esta recompensa, se ha decidido realizar de esta manera para que sea más seguro y que solo en el correo del propietario se pueda ver el código de la recompensa.



A set of small, light-gray navigation icons located at the bottom of the screen. From left to right, they include: a back arrow, a download icon (square with a downward arrow), an exclamation mark icon (hexagon with an exclamation mark), a trash bin icon, a mail icon (envelope with a dot), a file icon (square with a right-pointing arrow), and three vertical dots.

2 de 23.019 < >

Código canjeado ➤ Recibidos x

1

[skystayinfo@gmail.com](mailto:skystayinfo@gmail.com)  
para mí ▾

dom, 4 may, 19:41 (hace 13 horas)

...

Has canjeado correctamente el la recompensa! Aquí tiene su código para canjearla: 5a4a4338

## 4.7 Generación de informes y estadísticas

De cara al administrador es importante que este pueda tener toda la información que pueda para poder administrar correctamente el hotel que este encargado, por lo que una de sus funciones es generar diferentes informes que este puede imprimir o guardar como pdf con información relativa a dicho hotel, diferentes clientes, reservas o capacidad y disponibilidad de diferentes habitaciones. Estos informes estarán disponibles en el apartado del administrador mediante un botón podrás generar este archivo que será enviado directamente al correo del administrador adjuntado con un PDF.

Por lo que este proceso lo realizará el back en Java mediante una Api que genere el documento pdf y la envíe por la api del correo adjuntando este archivo anteriormente creado. Esta implementación se incorporará en una actualización posterior.

## 5. IMPLEMENTACIÓN DEL FRONTEND

## 5.1. Configuración del entorno y dependencias (React)

Para el apartado visual se ha utilizado React debido a que hoy en día la web es un medio mucho más sencillo y multiplataforma que una aplicación nativa. Sobre todo para el cliente, se ha observado que muchas veces al reservar productos o servicios en vez de un ordenador se utiliza el teléfono móvil u otros dispositivos como las tablets, por lo que se ha decidido realizar este proyecto en web y conectarse a la base de datos y a las diferentes funcionalidades mediante la Api Rest en Spring. Como Framework se ha decidido utilizar React debido a que hoy en día es de los más utilizados y con más soporte. Y como IDE

Visual Studio Code, que es el mejor para desarrollo en JavaScript y en general todo el desarrollo web con sus diferentes Frameworks.

Para empezar en Visual Studio Code en su terminal nos movemos a la carpeta de nuestro proyecto y tenemos que tener instalado npm y Node y luego realizar los siguientes comandos para crear una plantilla de nuestro proyecto.

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS
powershell +

-a----  01/03/2025  17:37      15248 demo (3).zip
-a----  19/02/2025  16:54      44148887 Gameplay Cavern Rush Eduardo Merino.mp4
-a----  06/01/2025  13:02      10595 java-functional-advanced-main.zip
● -a----  12/12/2024  7:16      9905234432 ubuntu cliente.ova

PS C:\Users\Eduardo\Desktop\Principal\Proyectos> cd TFG
● PS C:\Users\Eduardo\Desktop\Principal\Proyectos\TFG> npm -v
11.0.0
● PS C:\Users\Eduardo\Desktop\Principal\Proyectos\TFG> node -v
v20.18.1
❖ PS C:\Users\Eduardo\Desktop\Principal\Proyectos\TFG> npx create-react-app react-skystay

You can find a list of up-to-date React frameworks on react.dev
For more info see:https://react.dev/link/cra

This error message will only be shown once per install.

Creating a new React app in C:\Users\Eduardo\Desktop\Principal\Proyectos\TFG\react-skystay.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
[...]
Compiled successfully!

You can now view react-skystay in the browser.

  Local:          http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

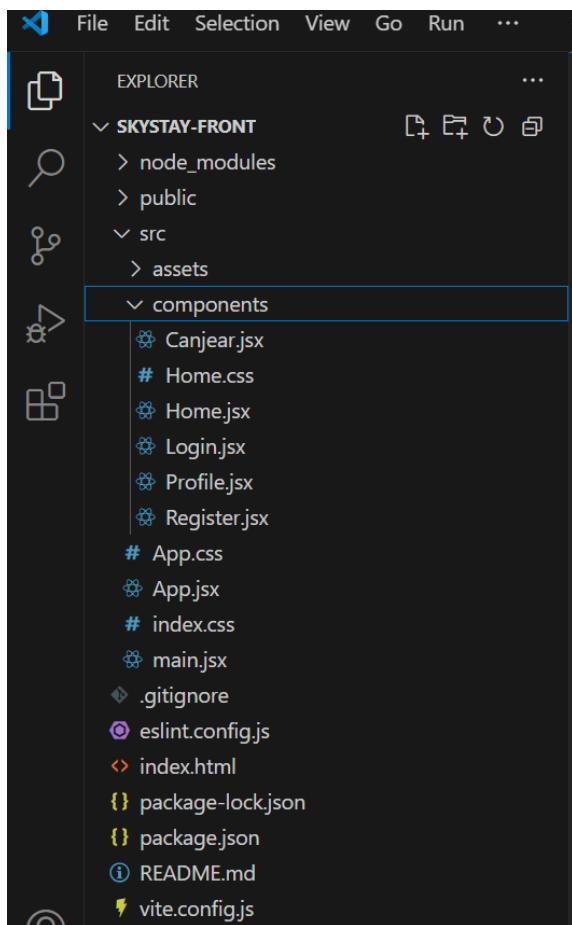
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
  
```

Cuando tengamos todo esto realizado podremos hacer un deploy de nuestro front para probar el funcionamiento en localhost:3030 y a partir de aquí tendremos nuestra estructura principal a la que más adelante se le añadirán dependencias como la de calendario para poder seleccionar las fechas de entrada y salida para el hotel, etc...

## 5.2. Desarrollo de la estructura de componentes

La estructura principal del Front es la siguiente:



Como se puede observar en la imagen anterior tenemos la estructura del proyecto:

### node\_modules

- Carpeta generada automáticamente por npm install.
- Contiene todas las dependencias del proyecto.

### public

- Archivos estáticos públicos accesibles directamente (por ejemplo, imágenes o favicon).
- index.html generalmente se encuentra aquí en un proyecto Vite o React.

### src

- Carpeta principal del código fuente.

Dentro de src hay:

### assets

- Contiene imágenes, fuentes o archivos estáticos que se usan dentro de los componentes.

## components

- Aquí es donde están los **componentes reutilizables o las diferentes vistas** del proyecto, donde dependiendo de la acción del usuario este llevará a una u otra vista.
- Los archivos .jsx son componentes de React (escritos en JSX).

Archivos dentro de components:

- Canjear.jsx: Una vista para el cliente donde este puede canjear puntos por diferentes recompensas.
- Home.jsx: Vista de la página principal o landing.
- Login.jsx: Vista para el inicio de sesión.
- Profile.jsx: Vista del perfil de usuario.
- Register.jsx: Vista para el registro de nuevos usuarios o administradores.

Aquí dependiendo de lo que necesitemos tendremos mas vistas para añadir opciones y funcionalidades a la web.

---

## src

- App.jsx: Componente principal de la aplicación, donde se definen las rutas y se estructura la navegación en este caso donde te redirige a las diferentes rutas que se encuentran en la carpeta components.
- index.css: Estilos globales, puede contener resets o configuraciones base.
- main.jsx: Punto de entrada de la aplicación React. Aquí se renderiza el <App /> dentro del root del HTML.

## 5.3. Conexión con la API REST

Para realizar las conexiones con la API se utiliza fetch para poder comunicarse con la API y realizar las diferentes peticiones que necesitemos (POST, GET, DELETE, UPDATE).

Como ejemplo para ver su funcionamiento he utilizado la vista del Login del Cliente:

### 1. Recogida de datos del formulario

El estado user y password se actualizan con los datos introducidos por el usuario mediante los inputs:

```
<form onSubmit={handleSubmit}>
  <h2>Iniciar Sesión</h2>
  <input
    placeholder="Usuario"
    value={user}
    onChange={(e) => setUsername(e.target.value)}
  />
  <input
    placeholder="Contraseña"
    type="password"
    value={password}
    onChange={(e) => setPassword(e.target.value)}
  />
  <button type="submit">Entrar</button>
</form>
</>
```

## 2. Envía las credenciales al servidor

Cuando se envía el formulario (onSubmit), se llama a la función handleSubmit, que hace lo siguiente:

```
const handleSubmit = async (e) => [
  e.preventDefault();
  const response = await fetch("http://localhost:8080/api/clientes/login", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ user, password })
  });
];
```

- Aquí se utiliza fetch para hacer una **petición POST** a la URL /api/clientes/login de tu API.
- En el body se envía un **JSON con el usuario y la contraseña**.
- El Content-Type: application/json indica que se están enviando datos en formato JSON.

## 3. Procesa la respuesta del servidor

```
const success = await response.json();
```

- Aquí se esta esperando la **respuesta del servidor**.
- Esta respuesta contiene algo que indique si el login fue exitoso, por ejemplo true o false.

#### 4. Si la autenticación es exitosa:

```

const success = await response.json();
if (success) {
  const userData = await fetch(`http://localhost:8080/api/clientes/${user}`);
  const userJson = await userData.json();
  setUser(userJson);
  navigate("/profile");
} else {
  alert("Credenciales incorrectas");
}
];

```

Si el login es correcto:

- Se hace otra petición GET para obtener la **información completa del usuario** (/api/clientes/{user}).
- Se guarda en el estado padre usando setUser(userJson) (esto se pasa como prop desde un componente superior).
- Se redirige al usuario a la ruta /profile usando navigate.

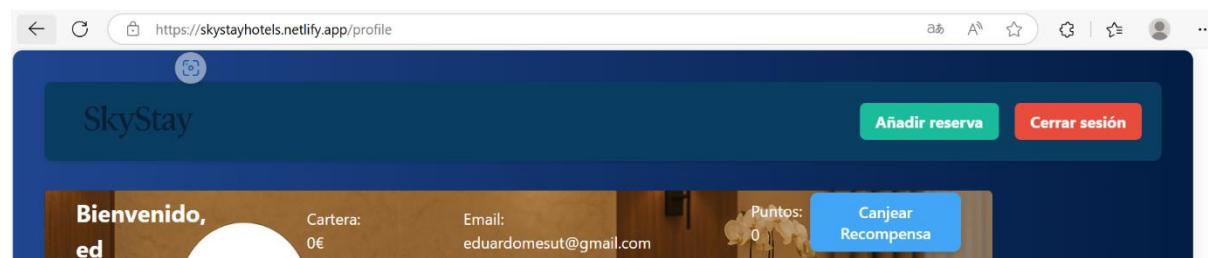
#### 5. Si el login falla:

- Muestra un mensaje de error si la autenticación no fue exitosa.

Y está es la manera que básicamente funcionan las peticiones en todas las vistas tanto para cambiar datos de la base de datos como para pedir cierta información y obtener los datos.

#### 5.4. Gestión de sesiones y autenticación

En el caso de las sesiones se ha utilizado un método más seguro que las propias rutas con el usuario, en este caso cuando inicias sesión entras en la ruta perfil y dependiendo de con que usuario hayas iniciado sesión la web será de una u otra manera con os datos de cada usuario. Con el sistema actual cada vez que el usuario sale de la web no se mantiene su cuenta y tiene que volver a iniciar sesión por seguridad.



Para la autenticación lo que hace el front es conectarse con la API en el inicio de sesión y realizar dicha petición y dependiendo de si es correcta o no que acceda al usuario o no le sea disponible y pruebe de nuevo, también por seguridad habrá un bloqueo de 30 min de la cuenta cuando se realicen mas de 5 intentos seguidos de inicio de sesión. Esto último será trabajo únicamente del Front que no realizará dicha petición hasta pasado ese tiempo.

[← Atrás](#)

### Iniciar Sesión

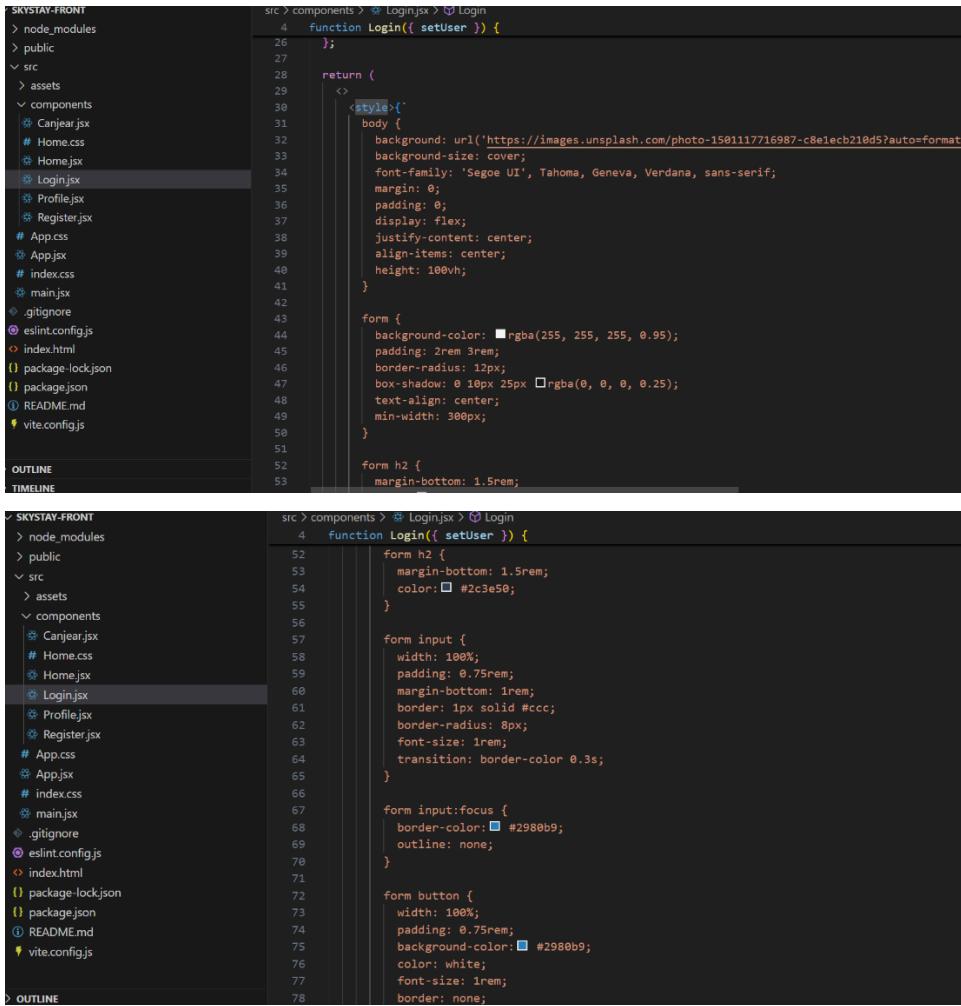
[Entrar como Cliente](#)

[Entrar como Admin](#)

## 5.5. Diseño de las vistas y experiencia de usuario

En las webs es importante el diseño visual tanto como la funcionalidad, por lo que es importante hacer unas vistas y una web que llame la atención, que sea legible, intuitiva y que sea acorde al tema y al sector que pertenece.

En este proyecto en cada vista justo encima del HTML esta la parte de style que es el código CSS que aportan estos detalles que ayudan al usuario a aportar este diseño que se busca.



```

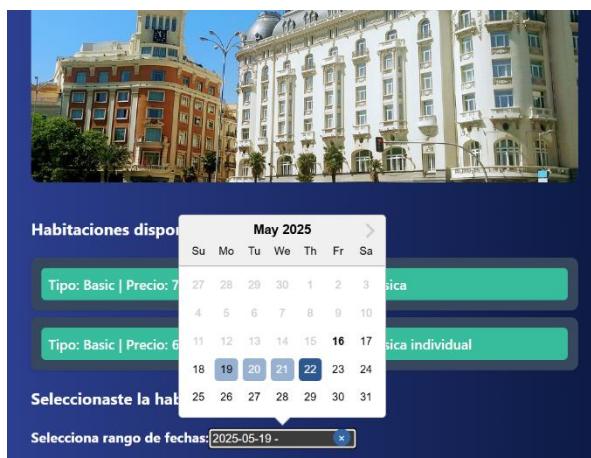
SKYSTAY-FRONT
src > components > Login.jsx > Login
  4  function Login({ setUser }) {
  5    ...
  6  }
  7
  8  return (
  9    <>
 10    <style>` 
 11      body {
 12        background: url('https://images.unsplash.com/photo-1501117716987-c8e1ecb210d5?auto=format&fit=crop&q=80');
 13        background-size: cover;
 14        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
 15        margin: 0;
 16        padding: 0;
 17        display: flex;
 18        justify-content: center;
 19        align-items: center;
 20        height: 100vh;
 21      }
 22
 23      form {
 24        background-color: #fff;
 25        padding: 2rem 3rem;
 26        border-radius: 12px;
 27        box-shadow: 0 10px 25px #0000000.25;
 28        text-align: center;
 29        min-width: 300px;
 30      }
 31
 32      form h2 {
 33        margin-bottom: 1.5rem;
 34      }
 35
 36      form input {
 37        width: 100%;
 38        padding: 0.75rem;
 39        margin-bottom: 1rem;
 40        border: 1px solid #ccc;
 41        border-radius: 8px;
 42        font-size: 1rem;
 43        transition: border-color 0.3s;
 44      }
 45
 46      form input:focus {
 47        border-color: #2980b9;
 48        outline: none;
 49      }
 50
 51      form button {
 52        width: 100%;
 53        padding: 0.75rem;
 54        background-color: #2980b9;
 55        color: white;
 56        font-size: 1rem;
 57        border: none;
 58      }
 59
 60    </style>
 61  )
 62}

SKYSTAY-FRONT
src > components > Login.jsx > Login
  4  function Login({ setUser }) {
  5    ...
  6
  7    form h2 {
  8      margin-bottom: 1.5rem;
  9      color: #2c3e50;
 10    }
 11
 12    form input {
 13      width: 100%;
 14      padding: 0.75rem;
 15      margin-bottom: 1rem;
 16      border: 1px solid #ccc;
 17      border-radius: 8px;
 18      font-size: 1rem;
 19      transition: border-color 0.3s;
 20    }
 21
 22    form input:focus {
 23      border-color: #2980b9;
 24      outline: none;
 25    }
 26
 27    form button {
 28      width: 100%;
 29      padding: 0.75rem;
 30      background-color: #2980b9;
 31      color: white;
 32      font-size: 1rem;
 33      border: none;
 34    }
 35
 36  }
 37}


```

Este es un ejemplo de la utilización del CSS en el archivo, en el return va tanto el HTML como el CSS junto.

Aparte del CSS básico en este proyecto se utilizan diferentes librerías para aportar diseños más personalizados y prediseños, el caso más claro es el calendario de reservas, donde el cliente puede seleccionar un rango de fechas que es visible en el calendario para que sea más sencillo el proceso de reserva de una habitación.



Aparte de esto se ha intentado con el uso de colores sin mucho contraste y con el tipo de navegación intuitiva hacer esta web accesible para toda clase de público, y que todos los procesos que puedes realizar queden claros y que sean sencillos de realizar.

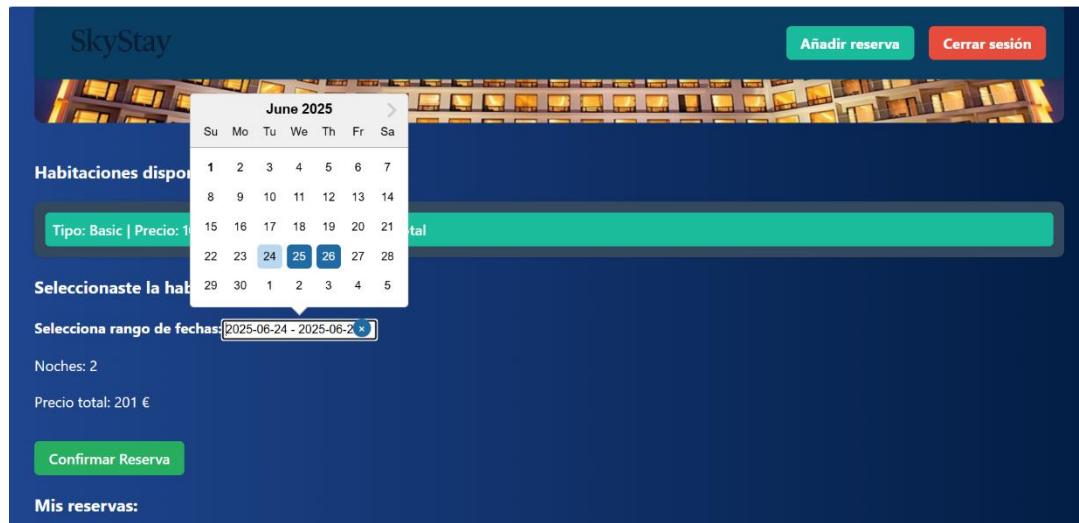
## 5.6. Implementación de funcionalidades principales

Algunas de las funcionalidades principales son las siguientes:

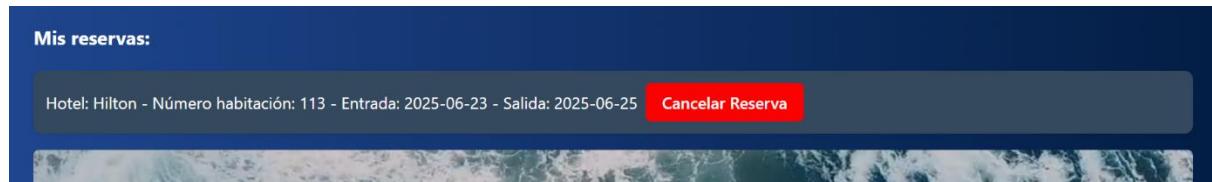
- Consulta de disponibilidad y reservas

Tanto el cliente como el administrador puede consultar y comprobar la disponibilidad de una habitación en un periodo de tiempo determinado gracias a los métodos que se conectan con la API.

También en el caso de las reservas, el propio cliente puede ver las reservas que ha realizado y en el caso del administrador puede ver todas las reservas realizadas en el hotel al que está a cargo.



The screenshot shows a dark-themed web application for 'SkyStay'. At the top right are 'Añadir reserva' and 'Cerrar sesión' buttons. Below the header is a banner featuring a building at night. The main area has a 'Habitaciones disponibles' section with a green button for 'Tipo: Basic | Precio: 1'. A date picker for 'June 2025' is open, showing days from 1 to 28. The days 24, 25, and 26 are highlighted in blue, indicating they are selected. Below the calendar, it says 'Seleccionaste la habitación' and 'Selección rango de fechas: 2025-06-24 - 2025-06-25'. It also shows 'Noches: 2' and 'Precio total: 201 €'. At the bottom are 'Confirmar Reserva' and 'Mis reservas:' buttons.



This screenshot shows the 'Mis reservas:' section. It lists a single reservation: 'Hotel: Hilton - Número habitación: 113 - Entrada: 2025-06-23 - Salida: 2025-06-25'. To the right of the reservation details is a red 'Cancelar Reserva' button. The background features a blurred image of a beach or sea.

- Sistema de fidelización (puntos y recompensas)

Un añadido importante para fortalecer la fidelización del cliente es el sistema de puntos y recompensas. Esto consiste en que el cliente cada vez que haga una reserva en la web se le acumularán puntos dependiendo del valor de la reserva realizada, estos puntos estarán vinculados a la cuenta y cuando acumulas una cantidad determinada puedes usarlos para canjear diferentes recompensas disponibles. Estas recompensas serán obtenidas por el cliente a través de un código que será enviado al correo del cliente en el momento que este canjee los puntos.

### Canjeo de Recompensas

Aquí podrás canjear tus puntos por recompensas.

Tus puntos: 3015

[Volver al perfil](#)

Viaje gratis - Descripción: Viaje muy bueno - Precio: 800 puntos

- Gestión de reservas y cancelaciones

El usuario puede cancelar su reserva y cuando haga esto la habitación ya estará libre para otra reserva en dicha habitación, aunque ahora al usuario no se le abonará de vuelta su dinero. Se hará una futura actualización de cancelación gratuita próximamente.

## 6. PRUEBAS Y DEPURACIÓN

### 6.1. Pruebas unitarias (JUnit)

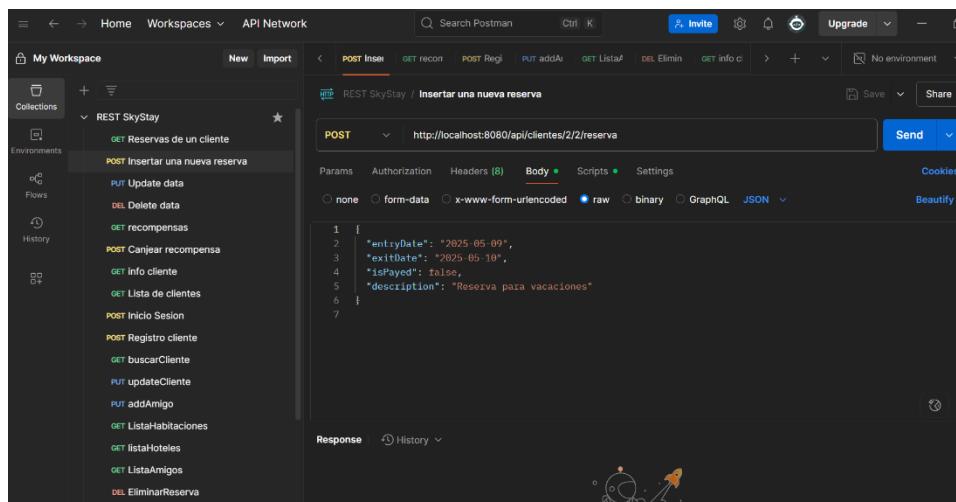
Las pruebas son una fase tan importante como el propio desarrollo ya que gracias a estas podemos comprobar el correcto funcionamiento del software antes de que lo haga el cliente final, y con esto poder detectar errores de diferentes tipos y corregirlos.

En este caso una de las pruebas mas relevantes son las pruebas unitarias,

Las pruebas unitarias se han centrado en los componentes del backend desarrollados en Spring Boot. Se han utilizado test con **JUnit 5**, orientados a verificar la lógica interna de los servicios, controladores y repositorios. Cada método crítico ha sido cubierto por casos de prueba que contemplan tanto comportamientos esperados como condiciones límite y errores esperados. El objetivo principal ha sido garantizar la fiabilidad del núcleo de la lógica de negocio.

## 6.2. Pruebas de integración (Postman, Swagger)

Para validar el correcto funcionamiento de la **API REST**, se realizaron pruebas de integración mediante **Postman**, simulando solicitudes típicas del flujo de uso: creación, modificación, cancelación y consulta de reservas, así como autenticación y gestión de recompensas. Las colecciones de Postman permitieron automatizar pruebas para diferentes escenarios.



The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'REST SkyStay' which includes endpoints such as 'GET Reservas de un cliente', 'POST Insertar una nueva reserva', 'PUT Update data', etc. The main panel shows a 'POST Insertar una nueva reserva' collection. It has a 'Body' tab selected with the following JSON payload:

```

1 {
2   "entryDate": "2025-05-09",
3   "exitDate": "2025-05-10",
4   "isStayed": false,
5   "description": "Reserva para vacaciones"
6 }

```

Además, **Swagger** se integró en el proyecto para documentar y explorar de forma interactiva los endpoints de la API, lo cual facilitó tanto el desarrollo como la detección temprana de inconsistencias entre los contratos de la API y su implementación.



The screenshot shows the Swagger UI interface for the 'admin-controller' API. It lists several endpoints:

- PUT /api/updatecliente/{id}**
- GET /api/administradores**
- POST /api/administradores**
- POST /api/administradores/login**
- GET /api/hoteles**
- GET /api/habitaciones/{id}**
- GET /api/buscarCliente/{palabra}**
- GET /api/administradores/{username}**

### 6.3. Pruebas de carga y rendimiento

Se efectuaron pruebas básicas de carga utilizando herramientas como **Apache JMeter**, con el fin de evaluar el comportamiento del sistema bajo múltiples peticiones simultáneas, simulando picos de usuarios accediendo al sistema para consultar disponibilidad o realizar reservas.

Estas pruebas se orientaron especialmente al backend, para asegurar que la API REST pudiera responder de forma eficiente sin provocar caídas ni tiempos de espera excesivos. Se identificaron cuellos de botella y se realizaron optimizaciones en las consultas a la base de datos y la configuración del entorno de ejecución.

### 6.4. Pruebas de usabilidad y accesibilidad

Desde el frontend, desarrollado en **React**, se realizaron pruebas de usabilidad con usuarios reales que simularon distintos perfiles (cliente y administrador). Se recopilaron observaciones sobre la claridad de la interfaz, la navegación, y la disposición de elementos clave como botones y formularios.

Asimismo, se validaron aspectos de **accesibilidad**, como el uso de etiquetas adecuadas, contraste de colores y navegación mediante teclado, siguiendo las pautas WCAG (Web Content Accessibility Guidelines). Estas pruebas contribuyeron a mejorar la experiencia de usuario final y garantizar la inclusión.

### 6.5. Estrategias de depuración y resolución de errores

Durante el desarrollo se aplicaron distintas estrategias de depuración tanto en frontend como en backend:

- En el backend, se utilizó el sistema de logs de Spring Boot con niveles diferenciados (INFO, DEBUG, ERROR), lo que permitió localizar problemas de forma precisa.
- En el frontend, se aprovechó la consola del navegador para inspeccionar errores de red y fallos de renderizado.
- Se recurrió a **debuggers integrados** en IntelliJ IDEA y Visual Studio Code para analizar el flujo de ejecución.
- Los errores detectados durante las pruebas fueron registrados en un sistema de control de incidencias, y se priorizó su corrección antes de cada entrega.

Estas estrategias permitieron mantener una alta calidad del software en cada iteración, favoreciendo una integración continua fluida y reduciendo el riesgo de errores en producción.

## 7. DESPLIEGUE Y MANTENIMIENTO

### 7.1. Contenedorización con Docker y Deploy con RESTED

Para la API Rest he querido utilizar Docker, que es una plataforma que permite empaquetar una aplicación con todas sus dependencias dentro de un contenedor. Este contenedor se puede ejecutar en cualquier máquina que tenga Docker instalado, asegurando que la aplicación funcione igual sin importar el entorno.

Esto es muy relevante porque en el caso de esta aplicación lo que se necesita es poder hacer un deploy fácilmente en cualquier tipo de entorno. Como en esta aplicación se necesitan diferentes servicios se ha optado por realizarlo con un Docker Compose que básicamente define todos los servicios en un archivo y los levanta en diferentes contenedores simultáneamente con el comando: “Docker-compose up”

```
👉 docker-compose.yml ✘
C: > Users > emerinf > Desktop > Proyecto > TFG > deploy > 👉 docker-compose.yml
1  version: '3.8'
2
3  services:
4    api:
5      build: .
6      container_name: contenedor-api
7      ports:
8        - "8080:8080"
9      depends_on:
10        - mariadb
11      environment:
12        spring.datasource.url: jdbc:mariadb://mariadb:3306/tfg
13        spring.datasource.username: root
14        spring.datasource.password:
15        spring.jpa.show-sql: true
16        spring.jpa.hibernate.ddl-auto: create
17        server.port: 8080
18        spring.sql.init.mode: always
19        spring.jpa.defer-datasource-initialization: true
20
21        spring.security.user.name: admin
22        spring.security.user.password: admin
```

```

24   |     spring.mail.host: smtp.gmail.com
25   |     spring.mail.port: 587
26   |     spring.mail.username: skystayinfo@gmail.com
27   |     spring.mail.password: ajnz brjc bfpm omxv
28   |     spring.mail.properties.mail.smtp.auth: true
29   |     spring.mail.properties.mail.smtp.starttls.enable: true
30
31
32   |     ▷ Run Service
33   |     mariadb:
34   |       image: mariadb:11
35   |       container_name: contenedor-mariadb
36   |       restart: always
37   |       environment:
38   |         MARIADB_ALLOW_EMPTY_ROOT_PASSWORD: "yes"
39   |         MARIADB_DATABASE: tfg
40   |       volumes:
41   |         - mariadb_data:/var/lib/mysql
42   |       ports:
43   |         - "3306:3306"
44
45   |     volumes:
46   |       mariadb_data:
47

```

Como se puede observar en las imágenes del Docker Compose, tenemos determinados diferentes servicios que va a ejecutar Docker en diferentes contenedores, en este caso el contenedor api y el contenedor mariadb donde se van a encontrar la API Rest en un .jar y la base de datos respectivamente.

A parte del Docker Compose necesitamos un archivo Dockerfile para que se cree una imagen y se copie el fichero jar del proyecto JAVA y que lo ejecute en Docker.

```

C: > Users > emerino > Desktop > Proyecto > TFG > deploy > Dockerfile > FROM
  1  FROM openjdk:21-jdk-slim
  2  COPY demo-0.0.1-SNAPSHOT.jar /app/mi-api.jar
  3  WORKDIR /app
  4  EXPOSE 8080
  5  ENTRYPOINT ["java", "-jar", "mi-api.jar"]
  6

```

Por último la API Rest se ha conseguido desplegar en un servicio llamado RESTED donde está en la nube siempre operativa y conectada con el Front en Netlify esto hace que independientemente del dispositivo y el lugar puedas acceder a la web y realizar las peticiones a la API en todo momento.

## 7.2. Despliegue del Front

Para el Front se ha utilizado una plataforma para poder subir a la web la aplicación y que se pueda acceder mediante cualquier dispositivo (Ordenadores, móviles, tablets...), en este caso se ha utilizado Netlify. Para esto se han seguido unos pasos:

1. *Preparar el proyecto React*

- Hay que asegurarse de que el proyecto **compila** con:

bash

Copiar Editar

npm run build

Esto generará una carpeta build/ (en React) con los archivos listos para producción.

## 2. Subir el proyecto a GitHub

- El proyecto tiene que estar en un repositorio online.

## 3. Entrar en Netlify

- Ir a: <https://app.netlify.com/>
- Iniciar sesión

## 4. Importar el repositorio

- Clica en “Add new site” → “Import an existing project”.
- Conecta tu cuenta de GitHub y elige el repo de tu frontend.

## 5. Configura el build

- **Build command:** npm run build
- **Publish directory:** build
- (Netlify detecta esto automáticamente si es un proyecto React)

## 6. Despliegue

- Darle a “Deploy site”.

Con esto se consiguió poner operativa la aplicación completa en el siguiente Link:

[SkyStayHotels](#)

## 7.3. Estrategia de mantenimiento y actualizaciones

Una parte muy relevante de un proyecto es el mantenimiento y las actualizaciones que se dan a lo largo de su tiempo de vida.

En cuanto al **mantenimiento** de la aplicación se harán diversas pruebas funcionales a lo largo del tiempo que aseguren el correcto funcionamiento de la aplicación. Así cuando se detecte un error o problema se pueda solucionar rápida y correctamente. Habrá diferentes backups tanto de la API Rest como del Front para si por cualquier problema una versión no funcione se pueda cambiar a otra anterior fácilmente.

También gracias a que Netlify esta vinculado con GitHub, cada vez que se hace una actualización del repositorio automáticamente el Front se actualiza y no se tiene que hacer

manualmente, lo que ayuda mucho a la rapidez y la facilidad para introducir nuevas actualizaciones.

En cuanto a las **actualizaciones**, sobre todo las posteriores, se centrarán sobre todo en mejorar la usabilidad por parte del usuario y en diferentes añadidos y mejoras que serán detallados a continuación:

- *Possibilidad de chat con amigos:*

Una de las actualizaciones más importantes que se quiere implementar en un futuro es la posibilidad de chatear con los amigos agregados. Esto será una tarea relativamente sencilla debido a que se utilizarán peticiones a la API con un nuevo objeto mensaje que enviará y recibirá información mediante GET y POST, y el usuario podrá enviar mensajes directos al amigo que desee.

- *Eventos especiales o temporales:*

Otra posible actualización será la de añadir diversos eventos dependiendo de la época del año, y estos eventos vendrán acompañados de diversos descuentos y sorteos entre los usuarios.

- *Reseñas:*

Un añadido bastante relevante que hacer en el futuro sería el de las reseñas, donde cada cliente puede votar de 1 a 5 estrellas el hotel donde se ha hospedado y dependiendo de esto los clientes al elegir hotel podrán ver su puntuación media para poder comprobar la satisfacción de clientes anteriores. Esto se haría de manera sencilla añadiendo la propiedad puntuación al objeto hotel y por medio de peticiones de los clientes que esta fuera variando.

## 8. RESULTADOS Y CONCLUSIONES

### 8.1. Comparación entre objetivos iniciales y resultados obtenidos

Inicialmente se pusieron unos objetivos, aunque realistas a la vez que algo optimistas, debido a todas las funcionalidades que quise introducir en un primer momento, como la seguridad, las recompensas canjeadas o los amigos; todo esto fue realizado correctamente pero al intentar abarcar tanto no se pulió al detalle cada apartado debidamente, por lo que me di cuenta que es mejor centrarse en ciertas acciones y pulirlas correctamente que intentar abarcar mucho y no lograr el objetivo al detalle.

Aun así tanto el objetivo general como los objetivos específicos se lograron correctamente:

**Objetivo General:** Desarrollar una plataforma completa de gestión de reservas hoteleras que integre frontend, backend y base de datos.

**Objetivos Específicos:**

1. Crear una API REST con Spring Boot para la gestión de reservas.
2. Implementar autenticación y autorización diferenciada para clientes y administradores.
3. Permitir reservas en tiempo real y gestión de recompensas.
4. Desarrollar un frontend responsive e intuitivo en React.
5. Incluir funcionalidades de cancelación y modificación de reservas.
6. Incorporar notificaciones por correo para eventos clave.
7. Asegurar la protección de los datos personales y transacciones.
8. Generar estadísticas de ocupación para los administradores.

Como se puede observar todos los objetivos han sido superados excepto el de estadísticas de ocupación que tiene acceso el administrador, que esto será uno de los añadidos posteriores en las actualizaciones.

Por lo que se puede concluir que los objetivos han sido en su mayoría superados satisfactoriamente, incluso realizando mas funciones de las previamente buscadas como todas las funciones que tiene la API de mail.

## 8.2. Problemas encontrados y soluciones aplicadas

Durante el desarrollo del proyecto se han encontrado diversos problemas de diferentes tipos, tanto a nivel de backend como de frontend. A continuación, se detallan algunos de los más relevantes junto con las soluciones aplicadas:

### 1. Ciclo de referencias entre entidades al devolver datos JSON

#### Problema:

Al realizar una petición que devolvía información de un cliente junto con sus reservas, se produjo el error:

HttpMessageNotWritableException: Could not write JSON: Document nesting depth (1001) exceeds the maximum allowed (1000)

Este error fue provocado por una relación de bucle entre las entidades Cliente y Reserva, donde cada entidad hacía referencia a la otra, generando una recursividad infinita durante la serialización a JSON.

#### Solución:

Se utilizaron las anotaciones `@JsonManagedReference` y `@JsonBackReference` para romper el ciclo de serialización, controlando qué parte de la relación se serializa y cuál no.

Alternativamente, en algunos casos, se empleó `@JsonIgnore` cuando no era necesario incluir la propiedad en la respuesta.

En otros el método que se utilizó fue cambiado por ejemplo para en vez de contener el objeto completo solo contenga String por ejemplo con el usuario del Cliente para que así no genere la recursividad.

---

### 2. Datos mostrados incorrectamente en React por error en el acceso a propiedades

#### Problema:

En el frontend, al intentar mostrar los amigos de un usuario, se accedía incorrectamente a `amigo.body` asumiendo que amigo era un objeto. Sin embargo, amigo era simplemente un string, lo que causaba que no se mostrara nada o apareciera `undefined`.

#### Solución:

Se corrigió el código eliminando el acceso erróneo a `.body`, y se utilizó directamente el valor del string amigo dentro del componente:

```
<div key={amigo}>{amigo}</div>
```

---

### 3. Persistencia de la autenticación al recargar la página

Problema:

Inicialmente, al recargar la aplicación React, el usuario autenticado perdía su sesión, ya que el token de autenticación no se almacenaba ni verificaba correctamente.

Solución:

Se implementó el almacenamiento del token JWT en localStorage y se añadió un mecanismo en el frontend para verificar su validez al iniciar la app. Además, se configuró un filtro en Spring Boot para autenticar las peticiones mediante el token almacenado.

---

*4. Errores de sincronización entre frontend y backend tras cambios en el modelo*

Problema:

Después de realizar modificaciones en las entidades del backend (por ejemplo, agregar nuevos campos a Reserva), las peticiones desde React empezaron a fallar debido a diferencias entre lo que el frontend enviaba y lo que el backend esperaba.

Solución:

Se actualizaron los DTOs, los controladores y los servicios en el backend para adaptarse a los nuevos modelos. Asimismo, se ajustaron los formularios y las peticiones del frontend para mantener la coherencia entre cliente y servidor.

---

Estos problemas y sus respectivas soluciones han sido fundamentales para consolidar el funcionamiento correcto y estable de la aplicación, permitiendo que tanto la lógica del negocio como la experiencia de usuario final sean robustas y coherentes.

### 8.3. Valoración del rendimiento del sistema

Una parte relevante del proyecto era el rendimiento del sistema, debido a que es una aplicación relativamente compleja que se tiene que conectar a diversos puertos y realizar diferentes conexiones entre el Front, el Back y la Base de datos.

Uno de los problemas que se encontraron fue a la hora de por ejemplo enviar los correos con la API, porque dependiendo de la conexión de donde se encuentre corriendo el Docker con la API, esta realiza las operaciones en un mayor o menor tiempo. Para esto la única solución que se ha encontrado es una actualización que se realizará en un futuro que es

correr el Docker de la API en la nube, para que este esté siempre desplegada y no haya problemas con el lugar donde esté conectada.

Por el resto, se puede observar que el rendimiento es bueno y fluido que es lo que se buscaba en un primer momento, para esto se realizaron pruebas de funcionamiento de la web para comprobar la fluidez tanto en versión móvil como escritorio. Gracias también a Spring vemos que las consultas y las conexiones con la Base de Datos se realizan de manera fluida y eficiente. Esto ayuda a que todo el programa funcione de una manera más rápida y sin fallos.

#### 8.4. Posibles mejoras y ampliaciones futuras

Como se puede observar y como se ha mencionado anteriormente este proyecto es la primera versión pero el propósito es que se vaya ampliando en el futuro con nuevas funcionalidades y características que se detallan a continuación:

A parte de las actualizaciones que se comentaron anteriormente hay diferentes ampliaciones futuras que serían añadidos bastante importantes de cara al futuro:

##### 1. Gestión de servicios adicionales

- Añadir la posibilidad de reservar servicios extra como desayuno, spa, parking o actividades dentro del hotel.
- Gestión dinámica de precios por temporada o demanda.

##### 2. Integración con pasarelas de pago

- Incorporar pagos online mediante plataformas como Stripe o PayPal, permitiendo realizar pagos seguros directamente desde la aplicación.

##### 3. Sistema de chat en tiempo real

- Comunicación directa entre clientes y recepción para resolver dudas, solicitudes especiales o cambios en la reserva.

##### 4. Valoraciones y comentarios

- Permitir que los clientes valoren su experiencia en el hotel y dejen reseñas visibles para otros usuarios.

##### 5. Gestión multi-idioma

- Implementar soporte multilingüe en la plataforma para llegar a un público internacional.

##### 6. Panel de estadísticas avanzado

- Incluir gráficas e indicadores visuales para la gestión hotelera: ocupación por meses, ingresos generados, reservas por canal, etc.

#### 7. Aplicación móvil

- Desarrollar una app nativa para Android/iOS que permita a los clientes realizar reservas desde sus dispositivos móviles.

#### 8. Sistema de alertas automatizadas

- Recordatorios push o vía email sobre fechas importantes: caducidad de recompensas, próximas reservas, promociones activas, etc.

#### 9. Check-in automático por QR

- Automatización del proceso de check-in mediante códigos QR para agilizar la llegada de los clientes al hotel.

#### 10. Gestión de promociones y descuentos

- Crear campañas promocionales con códigos de descuento y ofertas especiales para clientes frecuentes.

## 9. BIBLIOGRAFÍA Y REFERENCIAS

### **Spring Boot Documentation**

*Spring Framework Reference Documentation.*

Disponible en: <https://docs.spring.io/spring-boot/docs/current/reference/html/>

### **Spring Boot + Spring Data JPA**

*Guide to Spring Data JPA.*

Baeldung, 2023.

Disponible en: <https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa>

### **API REST con Spring Boot – Curso Práctico**

*Desarrollo de servicios REST con Spring Boot y Maven.*

Código Mentor, 2022.

Disponible en: <https://www.codigomentor.dev/posts/api-rest-con-spring-boot>

### **Documentación oficial de React**

*React – A JavaScript library for building user interfaces.*

Disponible en: <https://reactjs.org/docs/getting-started.html>

## JWT – Autenticación con tokens

*Introduction to JSON Web Tokens.*

Auth0, 2024.

Disponible en: <https://auth0.com/learn/json-web-tokens/>

## Spring Security y JWT

*Implementing JWT Authentication on Spring Boot APIs.*

Dev.to, 2023.

Disponible en: <https://dev.to/keyurparalkar/jwt-authentication-in-spring-boot-3fjj>

## Comunicación entre React y Spring Boot

*Consuming a RESTful Web Service with React.*

Spring.io Guides.

Disponible en: <https://spring.io/guides/tutorials/react-and-spring-data-rest/>

## Postman

*Herramienta para pruebas de API REST.*

Disponible en: <https://www.postman.com/>

# 10. ANEXOS

## 10.1. Código fuente relevante

Para revisar el código del proyecto detalladamente se recomienda ir directamente al repositorio público en GitHub: [Eduardomesut/TFG: SkyStay - Reservas de hoteles](https://github.com/Eduardomesut/TFG-SkyStay-Reservas-de-hoteles)

## 10.2. Documentación técnica de la API (Swagger)

admin-controller

PUT	/api/updatecliente/{id}	^
GET	/api/administradores	^
POST	/api/administradores	^
POST	/api/administradores/login	^ 
GET	/api/hoteles	^
GET	/api/habitaciones/{id}	^
GET	/api/buscarCliente/{palabra}	^
GET	/api/administradores/{username}	^

## ¿Por qué usar Swagger en tu proyecto?

- Documentación automática de tu API.
- Interfaz web para probar endpoints sin Postman.
- Ayuda a otros desarrolladores a entender cómo consumir tu API.
- Profesionaliza y mejora la presentación de tu proyecto.

---

## 10.3. Manual de usuario y administrador

### MANUAL DE USUARIO Y ADMINISTRADOR

---

#### 1. MANUAL DE USUARIO

##### 1.1. Acceso a la plataforma

1. Abre el navegador web.
2. Accede a la dirección: <https://skystayhotels.netlify.app/>
3. Introduce tu **usuario** y **contraseña** para iniciar sesión.
4. Haz clic en "Iniciar sesión".

Si no tienes cuenta, pulsa en "**Registrarse**" y completa el formulario.

---

##### 1.2. Funcionalidades principales

###### Visualizar información

- Una vez dentro, verás una **tabla con todos los clientes** disponibles (si eres usuario con permisos).
- Puedes usar el **buscador** para filtrar por nombre o email.

#### Realizar una reserva

1. Dirígete a la sección “Reservas”.
2. Selecciona el **hotel**, la **habitación** y las **fechas** deseadas.
3. Haz clic en "**Confirmar reserva**".
4. Aparecerá una notificación indicando si fue exitosa.

#### Añadir amigos

1. Ve a tu perfil.
2. Busca a otro usuario por nombre de usuario.
3. Pulsa "**Añadir como amigo**".

---

### 1.3. Cierre de sesión

- Haz clic en el botón "**Cerrar sesión**" situado en la parte superior derecha.

---

## 2. MANUAL DE ADMINISTRADOR

El administrador tiene acceso a funcionalidades exclusivas para la gestión completa del sistema.

---

### 2.1. Acceso de administrador

1. Accede como cualquier usuario, pero con una cuenta de tipo **admin**.
2. Al iniciar sesión, tendrás acceso al **panel de administración**.

---

### 2.2. Gestión de clientes

1. En el panel, haz clic en la sección "**Clientes**".
2. Podrás:
  - Ver la lista completa.
  - **Editar** información de un cliente.
  - **Eliminar** clientes (requiere confirmación).

---

### 2.3. Gestión de reservas

1. Accede a la sección “Reservas”.
2. Verás el historial de reservas de todos los usuarios.
3. Puedes:
  - Filtrar por usuario, fechas o estado.

- Cancelar reservas (por ejemplo, si hay errores o cambios).
- 

## 2.4. Gestión de usuarios

1. En la sección “Usuarios” puedes:
    - Ver el listado completo.
    - Cambiar roles (usuario ↔ administrador).
    - Eliminar cuentas inactivas o duplicadas.
- 

## 2.5. Visualización de logs o estadísticas (si aplica)

- Desde el panel puedes revisar estadísticas como número de registros, reservas totales, usuarios activos, etc.
- 

## 2.6. Recomendaciones

- Revisa regularmente la lista de usuarios y clientes.
  - Elimina cuentas obsoletas o fraudulentas.
  - Haz copia de seguridad periódica de la base de datos si la aplicación está en producción.
- 

## 3. CONTACTO Y SOPORTE

Para cualquier incidencia técnica, contactar con el equipo de desarrollo:

 Correo: [skystay@gmail.com](mailto:skystay@gmail.com)

 Teléfono: 123 456 789

 Documentación técnica (Swagger): <http://localhost:8080/swagger-ui/index.html>

## 10.4. Planificación temporal detallada

## Proceso Aplicación SkyStay

Gráfico de Gantt



[www.skystay.com](http://www.skystay.com)

### 10.5. Conclusiones finales

El desarrollo de SkyStay ha supuesto una experiencia muy enriquecedora tanto a nivel técnico como personal. A lo largo de este proyecto, he tenido la oportunidad de aplicar conocimientos adquiridos en clase de manera práctica, consolidando conceptos clave sobre el desarrollo web, arquitectura cliente-servidor y gestión de bases de datos.

Diseñar y construir una API REST desde cero con Spring Boot me ha permitido comprender en profundidad cómo se estructura y organiza la lógica del backend, cómo se gestionan las rutas y controladores, y cómo implementar medidas de seguridad como JWT y roles de usuario. Ha sido especialmente valioso aprender a establecer una separación clara entre las capas del sistema y mantener un código limpio y escalable.

Además, el trabajo con herramientas como Postman, Swagger, Docker y sistemas de control de versiones como Git/GitHub me ha proporcionado una visión más profesional del flujo de trabajo en entornos reales de desarrollo. También he podido mejorar mis habilidades en el diseño de bases de datos relacionales, cuidando aspectos como la integridad referencial, la normalización y la eficiencia de las consultas.

En cuanto al frontend, aunque el foco principal del proyecto ha estado en el backend, he aprendido a utilizar React y a conectar una API al Front lo que me ha ampliado mucho los conocimientos al tener que enfrentarme a tecnologías que no conocía.

Las clases han sido fundamentales como base teórica y guía práctica. Gracias a los contenidos y ejercicios vistos en el aula, he podido abordar este proyecto con conocimientos iniciales. Sobre todo en el ámbito de la lógica y la programación con Java que creo que ha sido clave, también muy relevante ha sido el conocimiento en bases de datos relacionales para poder estructurar correctamente el proyecto.

En resumen, *SkyStay* ha sido un proyecto desafiante pero muy gratificante, que me ha permitido integrar todo tipo de conocimientos y que me ha hecho mirar al desarrollo no solo desde un punto de vista de picar código sino desde el punto de vista del comienzo del proyecto, análisis de requisitos, desarrollo, deploy... En general me ha hecho ver todo el proceso complejo desde el inicio de la creación de un proyecto hasta el final o más bien el inicio del periodo de mantenimiento. Por lo que estoy muy contento con la realización de este proyecto y estoy ya pensando y buscando mejoras y actualizaciones para ampliarlo.

