SkyStay

**ÍNDICE (Provisional)**

**1. INTRODUCCIÓN**

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**

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**

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  
3.4.1. Wireframes y prototipos  
3.4.2. UX/UI: principios aplicados

**4. IMPLEMENTACIÓN DEL BACKEND**

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**

5.1. Configuración del entorno y dependencias (Angular)  
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**

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**

7.1. Contenedorización con Docker  
7.2. Despliegue en servidores y configuración  
7.3. Monitorización y gestión de logs  
7.4. Estrategia de mantenimiento y actualizaciones

**8. RESULTADOS Y CONCLUSIONES**

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**

**10. ANEXOS**

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

Aquí voy a realizar la memoria del proyecto,

**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 Angular.
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: Angular
* 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) |
| 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.

HACER DIAGRAMA

**3.3.2. Estructura y Normalización de Tablas**

Las tablas están diseñadas siguiendo **hasta la 3ª 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 Angular. 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.

Herramientas utilizadas para diseño: **Figma** y **Adobe XD** (opcional).

**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 SpringInizialicer que te permite elegir las dependencias que quieres 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:

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

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 contructores 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 establecmiento.

-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:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Como vemos hay que indicar el @Repositorio 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:

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

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, dependiento 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 se filtra la base de datos con información de los usuarios las contraseñas aparezcan encriptadas. Para esto se han realizado los siguientes métodos:

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

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

-------Añadir mas cosas de seguridad aquí-----------------------------------

**4.5 Gestión de reservas y usuarios**