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,

Resumen: El proyecto va a consistir en una web de una compañía de hoteles qu epueda conectarse por medio de una API REST a un Back y que este realice todas las peticiones necesarias a la base de datos por medio de peticiones HTTP de tipo GET, POST, UPDATE, DELETE…

Funciones Cliente:

-Registro

-Inicio de sesión

-Página principal

-Consulta de sus reservas

-Consulta de sus datos

-Consulta de recompensas

-búsqueda de habitaciones y precios por fechas

-Cambio de datos del propio usuario como la contraseña

-Borrado de cuenta

-Insertar códigos de descuento

-Agregar métodos de pago

-Cancelación de reservas

POSIBLES SOLO SI HAY TIEMPO

Agregar amigos con sus usuario y poder enviar mensajes y contratar juntos habitaciones

Funciones Administrador:

-Listar clientes

-Añadir habitaciones y editarlas y eliminarlas

-Gestionar reservas de los clientes de ese hotel que administran

-Enviar correo informativo de un evento a los usuarios del hotel

-Gestionar la información de los clientes

-Añadir y modificar reservas

-Modificar recompensas

POSIBLES SOLO SI HAY TIEMPO

ATRIBUTOS DE PERFIL QUE FALTAN

Cliente: Métodos de pago como lista

Cliente: Lista de amigos agregados a su red

Cliente: Suma total de puntos obtenidos en su cuenta

Cliente: boleano de estaverificado

API CORREO

-Habrá un sistema por el que cada vez que un usuario se registre recibirá un correo de verificación para poder validar su cuenta.

-También recibirá un correo cada vez que reserve una habitación y en este correo se le adjuntaran los datos de la reserva

-Por último al canjear una recompensa se le enviará al cliente un correo con el código de la recompensa que puede canjear.

Intentar realizar esta api con Spring y sino hacerla manual de otra manera con la que tengo de ejemplo del otro proyecto. En principio se puede utilizar spring mail

API TIEMPO - MENOS IMPORTANTE

Una pequeña API en la que al ver tus reservas próximas como usuario al dar a ver detalles puedas observar el tiempo que va a hacer en esa cuidad los días que te vas a alojar.

Explicaión ChatGPT de hacer la API correo

Para implementar un servicio de registro con verificación de cuenta por correo en una aplicación **Java con Spring Boot**, puedes seguir estos pasos:

**1. Dependencias necesarias**

Asegúrate de tener en tu pom.xml las siguientes dependencias para enviar correos:

xml

CopiarEditar

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-mail</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.11.5</version>

</dependency>

También puedes necesitar una base de datos, como PostgreSQL o MySQL.

**2. Configurar el servicio de correo**

En el archivo application.properties o application.yml, configura los datos de tu proveedor SMTP (por ejemplo, Gmail):

properties

CopiarEditar

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=tu-correo@gmail.com

spring.mail.password=tu-contraseña

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

⚠ **IMPORTANTE**: Si usas Gmail, activa **"Acceso de aplicaciones menos seguras"** o usa una **contraseña de aplicación**.

**3. Crear el modelo de usuario**

Define una entidad User en tu aplicación:

java

CopiarEditar

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String email;

private String password;

private String verificationToken;

private boolean enabled = false;

// Getters y Setters

}

**4. Crear el repositorio de usuario**

java

CopiarEditar

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByEmail(String email);

Optional<User> findByVerificationToken(String token);

}

**5. Servicio de correo para enviar el link de verificación**

java

CopiarEditar

@Service

public class EmailService {

@Autowired

private JavaMailSender mailSender;

public void sendVerificationEmail(String recipientEmail, String token) {

String subject = "Verifica tu cuenta";

String verificationLink = "http://localhost:8080/api/auth/verify?token=" + token;

String message = "Haz clic en el siguiente enlace para activar tu cuenta: " + verificationLink;

SimpleMailMessage email = new SimpleMailMessage();

email.setTo(recipientEmail);

email.setSubject(subject);

email.setText(message);

mailSender.send(email);

}

}

**6. Servicio de registro**

java

CopiarEditar

@Service

public class AuthService {

@Autowired

private UserRepository userRepository;

@Autowired

private EmailService emailService;

public void registerUser(String email, String password) {

if (userRepository.findByEmail(email).isPresent()) {

throw new RuntimeException("El correo ya está registrado");

}

User user = new User();

user.setEmail(email);

user.setPassword(password); // ⚠ Guardar encriptado con BCrypt

user.setVerificationToken(UUID.randomUUID().toString());

userRepository.save(user);

// Enviar correo de verificación

emailService.sendVerificationEmail(user.getEmail(), user.getVerificationToken());

}

public boolean verifyAccount(String token) {

Optional<User> userOpt = userRepository.findByVerificationToken(token);

if (userOpt.isPresent()) {

User user = userOpt.get();

user.setEnabled(true);

user.setVerificationToken(null);

userRepository.save(user);

return true;

}

return false;

}

}

**7. Controlador REST para manejar el registro y la verificación**

java

CopiarEditar

@RestController

@RequestMapping("/api/auth")

public class AuthController {

@Autowired

private AuthService authService;

@PostMapping("/register")

public ResponseEntity<String> register(@RequestParam String email, @RequestParam String password) {

authService.registerUser(email, password);

return ResponseEntity.ok("Registro exitoso. Revisa tu correo para verificar tu cuenta.");

}

@GetMapping("/verify")

public ResponseEntity<String> verifyAccount(@RequestParam String token) {

boolean verified = authService.verifyAccount(token);

if (verified) {

return ResponseEntity.ok("Cuenta verificada correctamente.");

}

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("Token inválido o expirado.");

}

}

**8. Explicación rápida**

1. **El usuario se registra** enviando su correo y contraseña.
2. **Se genera un token único** y se guarda en la base de datos.
3. **Se envía un correo** con el enlace de verificación (/api/auth/verify?token=XYZ).
4. **Cuando el usuario hace clic**, el servicio verifica el token y activa la cuenta.

**Extras: Mejoras**

* **Encriptar la contraseña** antes de guardarla (BCryptPasswordEncoder).
* **Agregar expiración al token de verificación**.
* **Enviar correos HTML en lugar de texto plano**.
* **Implementar seguridad con JWT** después de la verificación.

# ÍNDICE

### INTRODUCCIÓN Y OBJETIVOS

### ANTECEDENTES

#### 2.1 PUNTO DE PARTIDA

#### 2.2 DECISIÓN DE DESARROLLO DEL PROYECTO

### OBJETIVOS

#### 3.1 FUNCIONALIDADES DE SKYSTAY

### DESARROLLOPRÁCTICO

#### 4.1 PUNTO DE PARTIDA

#### 4.2 DECISIÓN DE DESARROLLO DEL PROYECTO

### ANTECEDENTES

#### 4.1 PUNTO DE PARTIDA

### ANTECEDENTES

#### 4.1 PUNTO DE PARTIDA

### ANTECEDENTES

#### 4.1 PUNTO DE PARTIDA

### ANTECEDENTES

#### 4.1 PUNTO DE PARTIDA

### ANTECEDENTES

#### 4.1 PUNTO DE PARTIDA

### ANTECEDENTES

#### 4.1 PUNTO DE PARTIDA

### 

### 