Proyecto P#1 | Programación Avanzada.

Manuel Antonio Jiménez Victor.

Universidad Internacional San Isidro Labrador.

Curso:
Programación avanzada.
Código: ISB-32.

Profesor:
Estefania Boza Villalobos.

Nota del Autor:
El presente trabajo es requisito para optar por la aprobación del curso PROGRAMACIÓN AVANZADA, código del curso: ISB-32, del Bachillerato en Ingeniería de Sistemas, en la Universidad Internacional San Isidro Labrador. Para contactar al autor escriba por correo electrónico a: manuel.antonio5535@gmail.com

Introducción

En la actualidad, los sistemas de gestión de reservas son fundamentales para mejorar la organización y experiencia de los clientes en el sector hotelero. Una plataforma web que permita gestionar las habitaciones facilita el control interno de los empleados y ofrece a los clientes un acceso rápido y seguro a sus reservas.

En un entorno donde la digitalización de procesos es cada vez más necesaria, los hoteles requieren soluciones que no solo les permitan administrar sus habitaciones, sino también mejorar la experiencia del cliente. Un sistema web de reservas centralizado garantiza mayor eficiencia en la atención, reduce errores humanos y optimiza el tiempo de gestión de cada solicitud.

Además, el acceso a través de un portal en línea brinda comodidad a los clientes, quienes pueden consultar la disponibilidad de habitaciones y confirmar sus reservaciones sin necesidad de acudir físicamente al hotel. Por su parte, los empleados cuentan con una herramienta confiable para verificar, registrar y actualizar la información de los huéspedes en tiempo real, lo cual fortalece la organización interna y eleva la calidad del servicio.

Para el desarrollo de este sistema se utilizará Spring Boot como framework para la construcción del backend, MySQL como sistema gestor de base de datos relacional y React como librería para el frontend. Esta combinación permitirá construir una solución moderna, escalable, eficiente y con una interfaz amigable para el usuario.

Objetivo General

Desarrollar una aplicación web de gestión de reservas de habitaciones para un hotel, que permita el acceso tanto de clientes como de empleados mediante un sistema de login, ofreciendo funcionalidades de consulta y administración de reservaciones en una base de datos.

Objetivos Específicos

1. Implementar un backend en Spring Boot conectado a una base de datos MySQL que gestione las operaciones de reservas (crear, consultar, actualizar, eliminar).
2. Desarrollar un frontend en React que proporcione una interfaz intuitiva para clientes y empleados, con login diferenciado según el tipo de usuario.
3. Integrar mecanismos de seguridad y validación de datos, asegurando que la información de las reservas y usuarios esté protegida mediante autenticación y control de roles.

Justificación

La implementación de esta arquitectura basada en Spring Boot, MySQL y React garantiza un desarrollo ágil y la posibilidad de mantener el sistema a largo plazo con actualizaciones o nuevas funcionalidades. Cada una de estas tecnologías es ampliamente utilizada en la industria, lo que asegura una gran comunidad de soporte, documentación extensa y compatibilidad con servicios modernos de despliegue en la nube

La elección de Spring Boot + MySQL + React responde a la necesidad de contar con un sistema robusto, escalable y seguro. Spring Boot permite construir una API REST confiable, con soporte para integración con MySQL y manejo de seguridad mediante Spring Security. MySQL es un sistema gestor de base de datos ampliamente usado, que garantiza consistencia y eficiencia en la gestión de la información de las reservas.

React facilita el desarrollo de interfaces modernas, dinámicas y responsivas, brindando una mejor experiencia de usuario tanto en computadoras como en dispositivos móviles..

Adicionalmente, el sistema responde a la necesidad de seguridad y escalabilidad que demanda el sector hotelero. Mediante la integración de autenticación robusta y control de roles, se minimizan riesgos en el manejo de información sensible. Por otra parte, el uso de un frontend moderno en React facilita la ampliación futura hacia aplicaciones móviles o la integración con plataformas de terceros, como pasarelas de pago en línea o sistemas de marketing digital.

Alcances esperados

- Desarrollo de un sistema web funcional accesible desde el navegador.
- Login diferenciado para clientes y empleados.
- Posibilidad de visualizar, agregar y administrar reservas de habitaciones.
- Interfaz amigable para la interacción rápida con la base de datos.
- Sistema preparado para escalar con nuevas funcionalidades en el futuro (pagos en línea, reportes, notificaciones).

Requerimientos

Requerimientos funcionales:

- Registro y autenticación de usuarios (clientes y empleados).
- Creación, edición, consulta y eliminación de reservas.
- Validación de roles (empleado con permisos administrativos, cliente con permisos limitados).
- Visualización de reservas existentes en la base de datos.

Requerimientos no funcionales:

- Seguridad en el manejo de credenciales (Spring Security, JWT).
- Interfaz responsive (adaptable a dispositivos móviles).
- Conexión eficiente con base de datos MySQL.
- Escalabilidad y mantenibilidad del sistema.

Pre-Diseño Visual – Sistema de Reservas de Hotel

Estructura de pantallas y bocetos

El diseño preliminar del sistema web, dividido en sus principales secciones: login, panel principal, navegación lateral, formularios y reportes.

Pantalla de Login:

- Contiene los campos para ingresar usuario y contraseña.
- Muestra el nombre del hotel y un slogan representativo.
- Paleta basada en turquesa (#009688) y gris claro (#f2f2f2) para transmitir limpieza y modernidad.
- Iconografía: íconos de usuario y candado para los campos de entrada.

Panel Principal

- Presenta un encabezado con el nombre del sistema y un botón para cerrar sesión.
- Muestra tarjetas resumen con datos de reservas activas y habitaciones disponibles.
- Colores predominantes: fondo gris claro y acentos turquesa.

Formularios de registro y reportes

- Formularios simples con campos bien espaciados.
- Uso de tipografía legible (Segoe UI o Roboto).
- Íconos representativos junto a cada campo (por ejemplo, un calendario para fechas, una cama para habitaciones).

Paleta de colores

- Turquesa (#009688): transmite frescura, tranquilidad y confianza, ideal para entornos de hospedaje.

- Gris claro (#f2f2f2): fondo neutro que facilita la lectura y resalta los componentes principales.
- Blanco (#ffffff): aporta claridad y limpieza visual.
- Gris oscuro (#333333)**: utilizado en textos y botones secundarios.

Justificación: La combinación de turquesa y gris claro ofrece un diseño moderno y sobrio. Estos tonos facilitan la concentración en los elementos importantes sin sobrecargar la vista del usuario.

Tipografía

- Fuente principal: Segoe UI (alternativas: *Roboto* o *Arial*).

Justificación: Se seleccionó una tipografía sans-serif moderna, clara y altamente legible en pantallas, garantizando una experiencia de usuario agradable y profesional.

Imágenes e iconografía

- íconos sencillos y representativos para cada módulo del sistema (usuario, llave, calendario, cama, gráfico, etc.).
- Uso de librerías estándar: **Lucide React** y **FontAwesome**.

Justificación: El uso de iconografía uniforme mejora la comprensión de las funciones del sistema. Los íconos refuerzan la identidad visual y permiten una navegación más intuitiva.

Base de Datos

Tipo de base de datos y propósito

La base de datos utilizada para este sistema es una base de datos relacional. Su propósito principal es almacenar y gestionar la información esencial del sistema de reservas de habitaciones del hotel, incluyendo usuarios (clientes y empleados), habitaciones, reservas y estados de ocupación. Este tipo de base de datos garantiza integridad referencial, consistencia y facilidad para realizar consultas complejas como disponibilidad de cuartos o historial de reservas.

Motor de base de datos elegido

Se seleccionó MySQL, debido a su desempeño, amplia compatibilidad con Spring Boot, facilidad de uso y capacidad para manejar de manera eficiente sistemas transaccionales como el de reservas.

Tipos de datos

- INT: Identificadores numéricos (ID_Usuario, ID_Habitación, ID_Reserva).
- VARCHAR (N): Textos como nombres, correos, contraseñas encriptadas, roles.
- DECIMAL(10,2) : Precios de la habitación.
- DATE / DATETIME: Fechas de entrada, salida y registros.
- BOOLEAN / BIT: Estados (activo/inactivo, disponibilidad).

Normalización (hasta 3FN)

- Primera Forma Normal (1FN): No existen datos multivaluados ni campos repetidos.
- Segunda Forma Normal (2FN):Cada atributo depende completamente de la clave primaria.

- Tercera Forma Normal (3FN): No existen dependencias transitivas; por ejemplo, la información del usuario se almacena únicamente en la tabla Usuario.

Descripción del ERD

- Usuario: Almacena datos de empleados y clientes (nombre, correo, rol, contraseña).
- Habitación: Contiene número, tipo, descripción, precio y estado.
- Reserva: Registra la habitación reservada, el usuario que la realizó y las fechas.
- TipoHabitación: Clasifica los cuartos (simple, doble, suite).

Relaciones

- Un usuario puede realizar muchas reservas (1:N).
- Una habitación puede aparecer en muchas reservas (1:N).
- Una reserva pertenece a un usuario y a una habitación (N:1 y N:1)

Conexión con la Aplicación

El sistema utiliza Spring Boot + JPA + Hibernate para conectar la base de datos MySQL. Esta combinación permite simplificar la persistencia de datos mediante anotaciones y repositorios.

application.properties:
“spring.datasource.url=jdbc:mysql://localhost:3306/reservas_hotel
spring.datasource.username=root
spring.datasource.password=1234
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true”

Entidad de ejemplo (Reserva):
@Entity

```
public class Reserva {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@ManyToOne
private Usuario usuario;

@ManyToOne
private Habitacion habitacion;

private LocalDate fechaEntrada;
private LocalDate fechaSalida;
}
```

Back end

Arquitectura del back end: Arquitectura por Capas

El proyecto utiliza una arquitectura en capas combinada con el estilo arquitectónico REST, lo cual es adecuado para aplicaciones que requieren modularidad, mantenibilidad y escalabilidad. Esta arquitectura separa claramente la lógica del sistema en diferentes capas, permitiendo un desarrollo más organizado, reutilizable y fácil de mantener.

Capa de Controladores (Presentation Layer)

- Implementada con Spring Web (REST Controllers).
- Se encarga de recibir solicitudes del frontend vía HTTP.
- Valida datos de entrada (DTOs).
- Llama a los servicios que contienen la lógica de negocio.
- Devuelve respuestas en formato JSON.

Capa de Servicios (Business Logic Layer)

- Contiene la lógica de negocio.
- Coordina operaciones entre repositorios.
- Aplica validaciones avanzadas.
- Maneja transacciones cuando es necesario.

Capa de Persistencia (Data Access Layer)

- Implementada con Spring Data JPA.
- Interactúa con SQL Server.
- Contiene repositorios (JpaRepository) para cada entidad.
  Permite realizar operaciones CRUD sin escribir SQL manual.

Capa de Modelo (Domain Layer)

- Define las entidades JPA que mapean las tablas de la base de datos.
- Contiene relaciones entre entidades.

- Representan el “corazón” del modelo de datos.

Componentes del código del Backend

El backend está organizado siguiendo una arquitectura por capas que separa de forma clara los modelos, servicios, repositorios y controladores. Esto facilita el mantenimiento, la extensibilidad y la escalabilidad del sistema.

Modelos

Son clases Java anotadas con @Entity, que representan las tablas de la base de datos:

- Habitacion
- Cliente
- Empleado
- Reserva

Función:

- Mapear las tablas SQL a objetos Java.
- Definir relaciones (OneToMany, ManyToOne, etc.).
- Servir como base para la lógica del sistema.

Repositorios

Son interfaces que extienden JpaRepository, encargadas de acceder a la base de datos:

- HabitacionRepository
- ClienteRepository
- EmpleadoRepository
- ReservaRepository

Función:

- CRUD automático sin escribir SQL.
- Consultas personalizadas mediante Query Methods o @Query.

Servicios

Contienen la lógica de negocio, coordinan repositorios y aplican validaciones.

- HabitacionService
- ClienteService
- EmpleadoService
- ReservaService

Función:

- Procesar reglas del sistema.
- Manejar excepciones.
- Realizar transacciones.
- Orquestar el acceso a datos.

Controladores

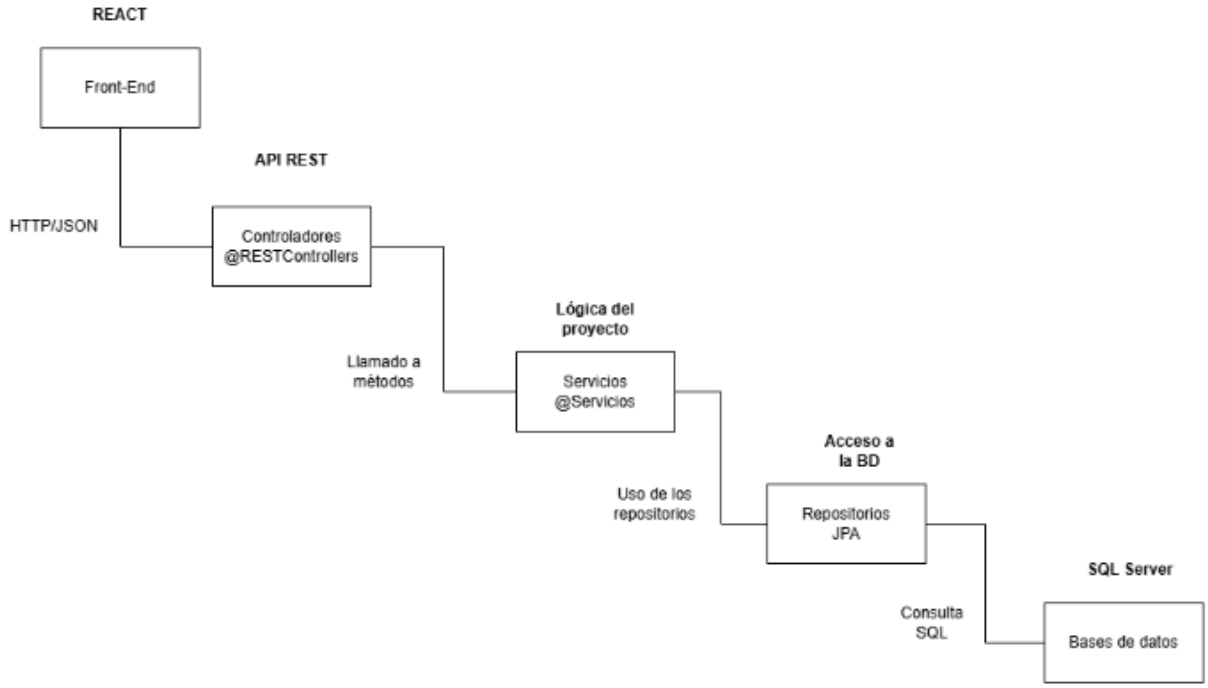
Exponen los endpoints REST:

- HabitacionController
- ClienteController
- EmpleadoController
- ReservaController

Función:

- Recibir solicitudes HTTP desde el frontend.
- Enviar datos JSON como respuesta.
- Invocar los servicios correspondientes.
- Realizar validaciones básicas usando DTOs.

Diagrama de flujo:



Pruebas funcionales

Se prueban endpoints completos, desde controlador a servicio a BD.

Ejemplo de prueba:

- GET /api/habitaciones/disponibles
- POST /api/reservas
- DELETE /api/clientes/{id}

Escalabilidad vertical

- Mejora de hardware del servidor.
- SQL Server permite aumentar RAM, CPU y almacenamiento sin reconfigurar la app.

Escalabilidad horizontal

- La API REST es stateless, por lo que puede ejecutarse en múltiples instancias.
- Se puede usar:
    - Docker
    - Kubernetes
    - AWS / Azure / GCP

Modularidad y Servicios independientes

Las capas permiten agregar nuevas funcionalidades sin afectar otras partes del sistema. Si el sistema crece, se puede evolucionar hacia microservicios más adelante sin rehacer todo.

Fragmentos de Código:

```java
import uisil.isb32.reservashotel.servicios.ClienteService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/clientes")
public class ClienteController {
    private final ClienteService service;
    public ClienteController(ClienteService service) { this.service = service; }

    @GetMapping
    public List<Cliente> findAll() { return service.findAll(); }

    @GetMapping("/{id}")
    public ResponseEntity<Cliente> findById(@PathVariable Integer id) {
        return service.findById(id).map(ResponseEntity::ok).orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<Cliente> create(@RequestBody Cliente c) {
        return ResponseEntity.ok(service.save(c));
    }

    @PutMapping("/{id}")
    public ResponseEntity<Cliente> update(@PathVariable Integer id, @RequestBody Cliente c) {
        return service.findById(id).map(existing -> {
            existing.setNombre(c.getNombre());
            existing.setTelefono(c.getTelefono());
            return ResponseEntity.ok(service.save(existing));
        }).orElse(ResponseEntity.notFound().build());
    }
}
```

Este es el controlador que nos permite recibir los request HTTP por medio del REST, para conectar con la aplicación front end..

```java
import jakarta.persistence.*;

@Entity
@Table(name = "Empleado")
public class Empleado {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer idEmpleado;

    @Column(length = 100, nullable = false)
    private String nombre;

    @Column(length = 20)
    private String telefono;

    // Getters y setters
    public Integer getIdEmpleado() { return idEmpleado; }
    public void setIdEmpleado(Integer idEmpleado) { this.idEmpleado = idEmpleado; }
    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }
    public String getTelefono() { return telefono; }
    public void setTelefono(String telefono) { this.telefono = telefono; }
}
```

Acá tenemos a la entidad Empleado que nos permite hacer la conversión por medio del JPA de las tablas a objetos para utilizarlo.

```java
import uisil.isb32.reservashotel.repositorios.HabitacionRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

@Service
public class HabitacionService {
    private final HabitacionRepository repo;
    public HabitacionService(HabitacionRepository repo) { this.repo = repo; }

    public List<Habitacion> findAll() { return repo.findAll(); }
    public List<Habitacion> findDisponibles() { return repo.findByOcupadoFalse(); }
    public Optional<Habitacion> findById(Integer id) { return repo.findById(id); }
    @Transactional
    public Habitacion save(Habitacion h) { return repo.save(h); }

    @Transactional
    public Habitacion marcarOcupada(Integer id, boolean ocupado) {
        Habitacion h = repo.findById(id).orElseThrow(() -> new RuntimeException("Habitación no encontrada"));
        h.setOcupado(ocupado);
        return repo.save(h);
    }
}
```

Este servicio es la lógica de la aplicación, por medio del repositorio que utiliza JPA, hace todas las consultas y el manejo de los objetos. Por medio de las listas se buscan los id necesarios o las flags necesarios como cuales están ocupadas. Está diseñado para poder hacer un set del trigger “isOccupied”.

Link del repositorio.

https://github.com/ManuelJ5535/reservashotel