|  |  |
| --- | --- |
| **Icono  Descripción generada automáticamente** | **Universidad Peruana Los Andes**  Facultad de Ingeniería  **Escuela Profesional de Sistemas y Computación** |

**Examen Parcial**

Asignatura: **Arquitectura de Software**

Código: P00923J Apellidos y Nombres: PEREZ MESCUA YANQUEIR

Fecha: 25.10.24

**Proyecto: Sistema de Gestión para Restaurante**

**1. Enunciado del proyecto**

**Desarrollar un sistema de gestión para un restaurante que permita manejar la administración de clientes, menús, mesas, reservas y restaurantes. Este sistema está diseñado para simplificar la administración del restaurante y mejorar la experiencia de servicio, con un enfoque en facilitar la creación, visualización y administración de reservas y menús para los clientes.**

**2. Identificación del problema del proyecto**

**El restaurante enfrenta desafíos en la organización eficiente de las reservas, la administración de clientes, y la actualización de menús y mesas disponibles. Sin un sistema centralizado, la gestión de estas tareas puede resultar en errores, tiempos de espera prolongados para los clientes, y dificultades para el personal en el manejo de datos. La ausencia de un sistema también afecta la visibilidad en tiempo real de la disponibilidad de mesas y reservas, limitando la capacidad de respuesta del restaurante y afectando la satisfacción del cliente.**

**3. Objetivo**

**Desarrollar una plataforma web que permita al restaurante gestionar clientes, menús, mesas y reservas de manera integrada, automatizada y en tiempo real. El sistema buscará simplificar las tareas del personal, reducir los errores de administración y mejorar la experiencia de servicio al cliente, al proporcionar una interfaz fácil de usar donde todos los datos sean accesibles y editables en una única plataforma.**

**4. Identificación de requerimientos**

**Requerimiento 1: Gestión de Clientes**

**- Nombre R1: Gestión de Clientes**

**- Resumen: Registrar, listar y gestionar la información de los clientes, permitiendo su acceso y edición desde la plataforma.**

**- Entradas: Datos del cliente (nombre, teléfono, correo, etc.).**

**- Resultados: Listado de clientes accesible para gestionar información y realizar reservas asociadas.**

**Requerimiento 2: Administración de Menús**

**- Nombre: R2: Administración de Menús**

**- Resumen: Crear, actualizar y listar menús del restaurante, mostrando detalles de cada platillo, su descripción y precio.**

**- Entradas: Datos del menú (nombre, descripción, precio, restaurante asociado).**

**- Resultados: Vista de menús disponibles y actualizados para consulta y edición.**

**Requerimiento 3: Control de Mesas**

**- Nombre: R3: Control de Mesas**

**- Resumen: Administrar las mesas disponibles en el restaurante, especificando su capacidad y disponibilidad para realizar reservas.**

**- Entradas: Datos de las mesas (número de mesa, capacidad, restaurante asociado).**

**- Resultados: Vista y actualización de disponibilidad de mesas para gestionar las reservas.**

**Requerimiento 4: Gestión de Reservas**

**- Nombre: R4: Gestión de Reservas**

**- Resumen: Permitir la creación, edición y visualización de reservas, vinculadas con clientes y mesas específicas.**

**- Entradas: Datos de la reserva (cliente, mesa, fecha y hora).**

**- Resultados: Listado de reservas con detalles y posibilidad de modificación para reflejar cambios o cancelaciones.**

**Requerimiento 5: Administración de Restaurantes**

**- Nombre: R5: Administración de Restaurantes**

**- Resumen: Gestionar información y detalles de cada restaurante, proporcionando contexto para la disponibilidad de menús y mesas.**

**-Entradas: Datos del restaurante (nombre, ubicación, capacidad, etc.).**

**- Resultados: Listado de restaurantes asociados y accesibles para la creación de menús y asignación de mesas y reservas.**

**Estos requerimientos establecen los módulos necesarios para gestionar cada área clave del restaurante en un sistema único e integrado.**

1. **Enlace del proyecto en GitHub**

[**https://github.com/Yanqueir/ProyectoArqui.git**](https://github.com/Yanqueir/ProyectoArqui.git)

1. **Diseño de Base de Datos (Script) la base de datos debe estar alojada en un servidor de la nube (**puede utilizar Microsoft Azure, Google Firebase, u otro)

-- Crear la base de datos

CREATE DATABASE restaurante\_reservas;

-- Usar la base de datos

USE restaurante\_reservas;

CREATE TABLE Usuarios (

id BIGINT NOT NULL AUTO\_INCREMENT,

nombre\_usuario VARCHAR(50) NOT NULL,

password VARCHAR(255) NOT NULL,

PRIMARY KEY (id),

UNIQUE KEY nombre\_usuario\_unique (nombre\_usuario)

);

-- Tabla Restaurante

CREATE TABLE Restaurante (

id INT AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100) NOT NULL,

direccion VARCHAR(255) NOT NULL,

telefono VARCHAR(20),

email VARCHAR(100),

descripcion TEXT

);

-- Tabla Mesa

CREATE TABLE Mesa (

id INT AUTO\_INCREMENT PRIMARY KEY,

numero\_mesa INT NOT NULL,

capacidad INT NOT NULL,

restaurante\_id INT,

FOREIGN KEY (restaurante\_id) REFERENCES Restaurante(id) ON DELETE SET NULL

);

-- Tabla Cliente

CREATE TABLE Cliente (

id INT AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100) NOT NULL,

apellido VARCHAR(100) NOT NULL,

telefono VARCHAR(20),

email VARCHAR(100)

);

-- Tabla Reserva

CREATE TABLE Reserva (

id INT AUTO\_INCREMENT PRIMARY KEY,

fecha\_hora DATETIME NOT NULL,

numero\_personas INT NOT NULL,

cliente\_id INT,

mesa\_id INT,

estado VARCHAR(20) NOT NULL,

FOREIGN KEY (cliente\_id) REFERENCES Cliente(id) ON DELETE SET NULL,

FOREIGN KEY (mesa\_id) REFERENCES Mesa(id) ON DELETE SET NULL

);

-- Tabla Menu

CREATE TABLE Menu (

id INT AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100) NOT NULL,

descripcion TEXT,

precio DECIMAL(10, 2) NOT NULL,

restaurante\_id INT,

FOREIGN KEY (restaurante\_id) REFERENCES Restaurante(id) ON DELETE SET NULL

);

-- Insertar datos en Restaurante

INSERT INTO Restaurante (nombre, direccion, telefono, email, descripcion) VALUES

('La Taberna de Paco', 'Calle Falsa 123, Madrid', '912345678', 'info@tabernapaco.com', 'Restaurante tradicional español'),

('El Gourmet Italiano', 'Avenida del Gourmet 45, Barcelona', '934567890', 'contacto@gourmetitaliano.com', 'Especialidades italianas y pizzas'),

('Sushi Express', 'Calle del Sushi 78, Valencia', '963258741', 'reservas@sushiexpress.com', 'Sushi y comida japonesa rápida'),

('La Bodega de Ana', 'Plaza Mayor 1, Sevilla', '954321987', 'info@bodegadeana.com', 'Comida típica andaluza y vinos'),

('Café del Arte', 'Calle de los Pintores 9, Bilbao', '944332211', 'contacto@cafedelar.com', 'Café y pastelería artesanal'),

('El Asador de Juan', 'Calle del Asador 22, Zaragoza', '976543210', 'reservas@asadorjuan.com', 'Carnes a la parrilla y tapas'),

('La Pizzería de Luca', 'Calle de la Pizza 56, Granada', '958765432', 'info@pizzerialuca.com', 'Pizzas y pastas caseras'),

('Comida Mexicana 24/7', 'Avenida de México 99, Murcia', '968745612', 'contacto@comidamexicana247.com', 'Comida mexicana auténtica'),

('El Rincón Vegetariano', 'Calle Verde 11, Alicante', '965478963', 'reservas@rinconvegetariano.com', 'Comida vegetariana y vegana'),

('El Marisco Dorado', 'Calle del Marisco 33, Palma', '971234567', 'info@mariscodorado.com', 'Mariscos frescos y cocina mediterránea');

-- Insertar datos en Mesa

INSERT INTO Mesa (numero\_mesa, capacidad, restaurante\_id) VALUES

(1, 4, 1),

(2, 2, 1),

(3, 6, 2),

(4, 4, 2),

(5, 2, 3),

(6, 8, 4),

(7, 4, 4),

(8, 4, 5),

(9, 2, 6),

(10, 6, 7);

-- Insertar datos en Cliente

INSERT INTO Cliente (nombre, apellido, telefono, email) VALUES

('Carlos', 'Pérez', '600123456', 'carlos.perez@example.com'),

('Ana', 'Gómez', '600234567', 'ana.gomez@example.com'),

('Luis', 'Fernández', '600345678', 'luis.fernandez@example.com'),

('Marta', 'López', '600456789', 'marta.lopez@example.com'),

('Jorge', 'Martínez', '600567890', 'jorge.martinez@example.com'),

('Isabel', 'Hernández', '600678901', 'isabel.hernandez@example.com'),

('Pedro', 'García', '600789012', 'pedro.garcia@example.com'),

('Lucía', 'Moreno', '600890123', 'lucia.moreno@example.com'),

('Antonio', 'Vázquez', '600901234', 'antonio.vazquez@example.com'),

('Sonia', 'Alonso', '600012345', 'sonia.alonso@example.com');

-- Insertar datos en Reserva

INSERT INTO Reserva (fecha\_hora, numero\_personas, cliente\_id, mesa\_id, estado) VALUES

('2024-07-20 19:00:00', 4, 1, 1, 'confirmada'),

('2024-07-21 13:00:00', 2, 2, 2, 'pendiente'),

('2024-07-22 20:00:00', 6, 3, 3, 'confirmada'),

('2024-07-23 21:00:00', 4, 4, 4, 'confirmada'),

('2024-07-24 14:00:00', 2, 5, 5, 'cancelada'),

('2024-07-25 12:00:00', 8, 6, 6, 'confirmada'),

('2024-07-26 18:00:00', 4, 7, 7, 'pendiente'),

('2024-07-27 16:00:00', 4, 8, 8, 'confirmada'),

('2024-07-28 17:00:00', 2, 9, 9, 'confirmada'),

('2024-07-29 15:00:00', 6, 10, 10, 'cancelada');

-- Insertar datos en Menu

INSERT INTO Menu (nombre, descripcion, precio, restaurante\_id) VALUES

('Menú del Día', 'Entrada, plato principal y postre', 15.00, 1),

('Menú Italiano', 'Pasta, pizza y postre', 20.00, 2),

('Menú Sushi', 'Sushi variado', 25.00, 3),

('Menú Andaluz', 'Tapas y plato principal', 22.00, 4),

('Menú Café y Pastelería', 'Café y pasteles', 10.00, 5),

('Menú Parrilla', 'Carnes a la parrilla y guarniciones', 30.00, 6),

('Menú Pizza y Pasta', 'Pizza y pasta al gusto', 18.00, 7),

('Menú Mexicano', 'Tacos, burritos y guacamole', 20.00, 8),

('Menú Vegetariano', 'Comida vegetariana variada', 17.00, 9),

('Menú Mariscos', 'Mariscos frescos y ensalada', 28.00, 10);

1. **Código explicado del proyecto**

package com.example.Restaurante.Controller;

import com.example.Restaurante.Model.Cliente;

import com.example.Restaurante.Model.ClienteRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller

@RequestMapping("/clientes")

public class ClienteController {

@Autowired

private ClienteRepository clienteRepository;

@GetMapping("/listar")

public String listarClientes(Model model) {

model.addAttribute("clientes", clienteRepository.findAll());

return "cliente";

}

@GetMapping("/formulario")

public String mostrarFormularioCliente(@RequestParam(value = "id", required = false) Long id, Model model) {

Cliente cliente = id != null ? clienteRepository.findById(id).orElseThrow(() -> new IllegalArgumentException("ID inválido: " + id)) : new Cliente();

model.addAttribute("cliente", cliente);

return "formulario-cliente";

}

@PostMapping("/guardar")

public String guardarCliente(Cliente cliente, RedirectAttributes redirectAttributes) {

clienteRepository.save(cliente);

String mensaje = cliente.getId() != null ? "Cliente actualizado exitosamente." : "Cliente agregado exitosamente.";

redirectAttributes.addFlashAttribute("mensaje", mensaje);

return "redirect:/clientes/listar";

}

@GetMapping("/eliminar/{id}")

public String eliminarCliente(@PathVariable("id") Long id, RedirectAttributes redirectAttributes) {

clienteRepository.deleteById(id);

redirectAttributes.addFlashAttribute("mensaje", "Cliente eliminado exitosamente.");

return "redirect:/clientes/listar";

}

}  
  
Este código es un controlador de Spring Boot para manejar clientes en una aplicación de restaurante.

1. \*\*Inyección de Dependencia\*\*: `ClienteRepository` se inyecta para realizar operaciones CRUD en la base de datos de clientes.

2. \*\*Listar Clientes\*\* (`/listar`): Método `listarClientes` que usa `clienteRepository.findAll()` para obtener todos los clientes y enviarlos a la vista `cliente`.

3. \*\*Formulario de Cliente\*\* (`/formulario`): Método `mostrarFormularioCliente` que muestra un formulario para agregar o editar un cliente, dependiendo si se recibe un `id` de cliente.

4. \*\*Guardar Cliente\*\* (`/guardar`): Método `guardarCliente` que guarda o actualiza el cliente y redirige a la lista de clientes con un mensaje de éxito.

5. \*\*Eliminar Cliente\*\* (`/eliminar/{id}`): Método `eliminarCliente` que elimina el cliente por `id` y redirige a la lista con un mensaje confirmando la eliminación.

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package com.example.Restaurante.Model;

import jakarta.persistence.\*;

@Entity

@Table(name = "Cliente")

public class Cliente {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "nombre", nullable = false)

private String nombre;

@Column(nullable = false)

private String apellido;

private String telefono;

private String email;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public String getApellido() {

return apellido;

}

public void setApellido(String apellido) {

this.apellido = apellido;

}

public String getTelefono() {

return telefono;

}

public void setTelefono(String telefono) {

this.telefono = telefono;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}  
Este código define la clase `Cliente`, que representa la entidad "Cliente" en la base de datos para una aplicación de restaurante.

1. \*\*Anotaciones de Entidad\*\*:

- `@Entity` indica que esta clase es una entidad JPA que se mapeará a una tabla en la base de datos.

- `@Table(name = "Cliente")` especifica el nombre de la tabla en la base de datos.

2. \*\*Atributos\*\*:

- `id`: clave primaria, con generación automática usando `@GeneratedValue(strategy = GenerationType.IDENTITY)`.

- `nombre`, `apellido`: atributos obligatorios, mapeados a columnas con `@Column(nullable = false)`.

- `telefono`, `email`: atributos opcionales.

3. \*\*Getters y Setters\*\*: Métodos para acceder y modificar los valores de los atributos de la entidad.  
/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package com.example.Restaurante.Model;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface ClienteRepository extends JpaRepository<Cliente, Long> {

// Puedes agregar métodos de consulta personalizados aquí si es necesario

}  
Este código define la interfaz `ClienteRepository`, que permite realizar operaciones CRUD en la entidad `Cliente` a través de Spring Data JPA.

1. \*\*Extensión de JpaRepository\*\*: `ClienteRepository` extiende `JpaRepository<Cliente, Long>`, lo que proporciona métodos CRUD y de paginación para la entidad `Cliente`. El parámetro `Long` indica el tipo de la clave primaria de `Cliente`.

2. \*\*Anotación @Repository\*\*: Marca esta interfaz como un componente de repositorio, permitiendo que Spring la detecte y gestione automáticamente.

1. \*\*Métodos Personalizados\*\*: Al extender `JpaRepository`, se pueden agregar métodos de consulta personalizados si es necesario.

<!DOCTYPE html>

<html lang="es" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Lista de Clientes</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"

integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"

crossorigin="anonymous">

<style>

body {

background-color: #f9f3e0;

}

.navbar {

background-color: #ffb347;

}

.navbar-brand, .navbar-nav .nav-link {

color: #fff;

}

.container {

background-color: #fff;

border-radius: 10px;

padding: 20px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

margin-top: 20px;

}

h1 {

color: #ff9c00;

}

</style>

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark">

<a class="navbar-brand" href="#">Sistema de Gestión</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"

aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item">

<a class="nav-link" th:href="@{/clientes/listar}">Clientes</a>

</li>

<li class="nav-item">

<a class="nav-link" th:href="@{/menus/listar}">Menús</a>

</li>

<li class="nav-item">

<a class="nav-link" th:href="@{/mesas/listar}">Mesas</a>

</li>

<li class="nav-item">

<a class="nav-link" th:href="@{/reservas/listar}">Reservas</a>

</li>

<li class="nav-item">

<a class="nav-link" th:href="@{/restaurantes/listar}">Restaurantes</a>

</li>

</ul>

</div>

</nav>

<div class="container mt-4">

<h1>Lista de Clientes</h1>

<a th:href="@{/clientes/formulario}" class="btn btn-success mb-3">Agregar Cliente</a>

<table class="table">

<thead>

<tr>

<th>ID</th>

<th>Nombre</th>

<th>Apellido</th>

<th>Teléfono</th>

<th>Email</th>

<th>Acciones</th>

</tr>

</thead>

<tbody>

<tr th:each="cliente : ${clientes}">

<td th:text="${cliente.id}"></td>

<td th:text="${cliente.nombre}"></td>

<td th:text="${cliente.apellido}"></td>

<td th:text="${cliente.telefono}"></td>

<td th:text="${cliente.email}"></td>

<td>

<a th:href="@{/clientes/formulario(id=${cliente.id})}" class="btn btn-primary btn-sm">Editar</a>

<a th:href="@{/clientes/eliminar/{id}(id=${cliente.id})}" class="btn btn-danger btn-sm"

onclick="return confirm('¿Estás seguro de que quieres eliminar este cliente?');">Eliminar</a>

</td>

</tr>

</tbody>

</table>

</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"

integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"

crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"

integrity="sha384-iG9kWpXR+pDcyEj3Wm8zcMOi0hMHHMIIdR3wsq8ofdr0tvcJbCI+5LhktPhckKp2"

crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"

integrity="sha384-b4gt1jrGC7Jh4AgTPSdUtOBvfO8sh/jFc0iqcpL5bnJ7I4E3lt6p4U5CkUnwK+qI"

crossorigin="anonymous"></script>

</body>

</html>  
  
Este archivo HTML utiliza \*\*Thymeleaf\*\* y \*\*Bootstrap\*\* para generar una interfaz web que lista los clientes y permite realizar operaciones CRUD en ellos dentro del sistema de gestión de un restaurante. Aquí un resumen de cada parte:

1. \*\*Cabecera\*\*: Define el idioma (`lang="es"`) y la codificación (`UTF-8`). Incluye Bootstrap para el diseño y estilos personalizados para la interfaz.

2. \*\*Barra de Navegación\*\*: Usa `navbar` de Bootstrap con enlaces de navegación (`Clientes`, `Menús`, `Mesas`, `Reservas`, `Restaurantes`) que redirigen a las respectivas listas mediante `th:href`.

3. \*\*Contenido Principal\*\*:

- \*\*Encabezado y Botón\*\*: Muestra un título y un botón "Agregar Cliente" que enlaza al formulario para añadir o editar clientes.

- \*\*Tabla\*\*: Muestra los clientes en filas, generadas dinámicamente con `th:each`. Los datos de cada cliente (`id`, `nombre`, `apellido`, `telefono`, `email`) se muestran usando `th:text`.

- \*\*Acciones\*\*: Cada cliente tiene botones para editar y eliminar:

- `Editar` redirige al formulario de edición.

- `Eliminar` confirma antes de eliminar al cliente.

4. \*\*Scripts\*\*: Incluye jQuery, Popper y Bootstrap para el soporte de elementos interactivos.