DOCUMENTACION FRONTED Y BACKEND

Equipo 2

Integrantes

Natalia Cancino Vega
Diego Cerqueda Herrera
Cesar David Gómez Núñez
Mario Antonio Moreno López

Sistemas Web

EE LTC Agosto 2023 - Enero 2024

Introducción

Esta documentación aborda lo realizado a lo largo del proyecto teniendo en cuenta el desarrollo front end el cual se desarrolló usando react y el back end el cual se hizo utilizando spark y MySQL.
El proyecto es acerca de una página web de reservación de habitaciones de 6 distintos hoteles a lo largo del país, cuenta con un Login para que solo los usuarios registrados sean los que puedan reservas las habitaciones.

Front end

LoginForm.CSS

Estilos Generales
Selector .wrapper
width: Define el ancho del contenedor a 450px.
background: Fondo transparente.
border: Borde sólido de 2px con color RGBA blanco y baja opacidad.
backdrop-filter: Aplica un desenfoque de 50px al fondo.
box-shadow: Sombra con un desplazamiento de 0, 0 y difuminado de 10px con color RGBA negro y baja opacidad.
color: Texto en color blanco.
border-radius: Bordes redondeados.
padding: Relleno interno de 50px arriba y abajo, 30px a los lados.
Selector .wrapper h1
font-size: Tamaño de fuente de 36px.
text-align: Texto centrado.
Selector .wrapper .input-box
position: Posición relativa.
width: Ancho del 100%.
height: Altura de 50px.
margin: Margen de 30px arriba y abajo, 0 a los lados.
Selector .input-box input
width y height: Ocupan el 100% del contenedor.
background: Fondo transparente.
border: Borde sólido de 2px con color RGBA blanco y baja opacidad.
outline: Borde de contorno al enfocar.
border-radius: Bordes redondeados.
font-size: Tamaño de fuente de 16px.
color: Texto en color blanco.
padding: Relleno interno de 20px arriba, 45px a la derecha, 20px abajo, 20px a la izquierda.
Selector .input-box input::placeholder
color: Color del texto de marcador de posición en blanco.

Selector .input-box .icon
position: Posición absoluta.
right: Alineación a la derecha.
top: Alineación en la mitad superior.
transform: Desplaza verticalmente en la mitad de la altura.
font-size: Tamaño de fuente de 16px.
Selector .wrapper .remember-forgot
display: Muestra como un flex-container.
justify-content: Espaciado uniforme entre elementos.
font-size: Tamaño de fuente de 14.5px.
margin: Margen de -15px arriba, 0 a los lados, 15px abajo.
Selector .remember-forgot label input
accent-color: Color de acento del campo de entrada.
Selector .remember-forgot a
color: Texto en color blanco.
text-decoration: Sin subrayado.
Selector .remember-forgot a:hover
text-decoration: Subrayado al pasar el mouse.
Selector .wrapper .boton1
display: Muestra como un elemento en línea.
position: Posición absoluta.
background-color: Ningún color de fondo.
top, left, width, height: Posición y dimensiones del botón.
padding: Sin relleno.
font-size: Tamaño de fuente de 14px.
Selector .wrapper button
width y height: Ocupan el 100% de ancho y 45px de altura.
background: Fondo blanco.
border: Sin borde.
outline: Sin contorno.
border-radius: Bordes redondeados.
box-shadow: Sombra con desplazamiento de 0, 0 y difuminado de 10px con color RGBA negro y baja opacidad.
cursor: Cambia a un puntero al pasar el mouse.
font-size: Tamaño de fuente de 16px.
color: Texto en color negro.
font-weight: Grosor de la fuente 700.
Selector .wrapper .register-link
font-size: Tamaño de fuente de 14.5px.
text-align: Texto centrado.
margin: Margen de 20px arriba, 0 a los lados, 15px abajo.
Selector .register-link p a
color: Texto en color blanco.

text-decoration: Sin subrayado.
font-weight: Grosor de la fuente 600.
Selector .register-link p a:hover
text-decoration: Subrayado al pasar el mouse.
Estilos Adicionales
Selector .cuerpo
width: Ancho del 160.1%.
background-repeat: No se repite el fondo.
height: Altura del 100%.
padding-left: Relleno izquierdo del 0%.
background-size: Tamaño del fondo del 100% al ancho y automático al alto.
margin: Sin márgenes.
font-family: Fuente de la familia 'Poppins' y tipografía sans-serif.
-webkit-font-smoothing y -moz-osx-font-smoothing: Suavizado de fuentes.
Selector .cuerpo
transform: Desplaza horizontalmente en -361px

loginForm.jsx

Importaciones
React
React: La biblioteca principal de React.
useState: Hook de React que permite agregar estado a los componentes de función.
Estilos y Íconos
'./LoginForm.css': Importación de un archivo de estilo CSS para aplicar estilos personalizados.
FaUser, FaLock, FaHome: Iconos de usuario, candado y hogar de la biblioteca de iconos react-icons/fa.
Material-UI Components
TextField, Button, Box: Componentes de Material-UI para campos de texto, botones y contenedores respectivamente.
Axios
axios: Cliente HTTP basado en promesas para realizar solicitudes HTTP.
React Router
useNavigate: Hook de React Router utilizado para la navegación programática entre las rutas.
Componentes Personalizados
PopupV y PopupIv: Componentes de ventana emergente para mostrar mensajes de inicio de sesión exitoso e incorrecto.
Función LoginForm
Estados
Cargando: Estado que indica si la solicitud de inicio de sesión está en progreso.
datosFormulario: Estado que almacena los datos del formulario (correo y contraseña).
nombre: Estado que almacena el nombre del usuario.
Funciones de Redirección
redirectToRecuperarContraseña: Redirige al usuario a la página de recuperación de contraseña.

redirectToHome: Redirige al usuario a la página principal del hotel.
Funciones de Alerta
mostrarAlertaLoginExitoso: Muestra una alerta de éxito con el nombre del usuario.
mostrarAlertaLoginFallido: Muestra una alerta de fallo de inicio de sesión por credenciales incorrectas.
mostrarAlertaLoginSinDatos: Muestra una alerta de fallo de inicio de sesión por datos faltantes.
Funciones de Conexión con el Backend
loginUsuario: Realiza una solicitud al backend para iniciar sesión con los datos del formulario.
Funciones de Manipulación de Popup
abrirPopupV y abrirPopupIv: Abre los popups de inicio de sesión exitoso e incorrecto respectivamente.
cambiosFormulario: Actualiza el estado de datosFormulario al cambiar los valores del formulario.
Renderización del Componente
El componente renderiza un formulario con campos de correo y contraseña, un botón de inicio de sesión y enlaces para recuperar contraseña y registrar una cuenta.
Utiliza iconos de Material-UI para mejorar la experiencia visual.
Muestra popups de éxito y error según la respuesta del backend.

RecuperarContra.jsx

Propiedades

props

Tipo: Object

Descripción: Objeto que contiene propiedades adicionales. En este caso, no se utilizan propiedades adicionales.

Estado

Cargando

Tipo: Boolean

Descripción: Estado que indica si la aplicación está en un estado de carga. Se utiliza para deshabilitar el botón "Entrar" durante operaciones asíncronas.

datosFormulario

Tipo: Object

Descripción: Estado que almacena los datos del formulario, incluyendo el correo electrónico y la contraseña.

mostrarContraseña

Tipo: Boolean

Descripción: Estado que indica si debe mostrarse el campo de contraseña en el formulario.

desactivarCorreo

Tipo: Boolean

Descripción: Estado que indica si el campo de correo electrónico debe estar desactivado.

cambio

Tipo: Number

Descripción: Estado que indica la fase actual del formulario (0 para la verificación del correo y 1 para el cambio de contraseña).

nombre

Tipo: String

Descripción: Estado que almacena el nombre.

mostrarPopupActualizarInv, mostrarPopupVeri, mostrarPopupV, mostrarPopupIv

Tipo: Boolean

Descripción: Estados que indican si los diferentes popups de la interfaz deben mostrarse o no.

Métodos

loginUsuario(evento)

Descripción: Maneja el evento de envío del formulario. Realiza una verificación del correo y, si es válido, permite al usuario cambiar la contraseña.

Uso: loginUsuario(evento)

mostrarAlertaCambioContraExitoso(), mostrarAlertaCambioContraFallido()

Descripción: Muestra una alerta (éxito o error) utilizando la biblioteca swal.

Uso: mostrarAlertaCambioContraExitoso(), mostrarAlertaCambioContraFallido()

cambiosFormulario(evento)

Descripción: Maneja los cambios en el formulario actualizando el estado datosFormulario.
Uso: cambiosFormulario(evento)
redirectToHome(), redirectToLogin()
Descripción: Redirige a diferentes rutas de la aplicación utilizando el objeto navigate de react-router-dom.
Uso: redirectToHome(), redirectToLogin()
abrirPopupActualizarInv(), abrirPopupVeri(), abrirPopupV(), abrirPopupIv()
Descripción: Abre los diferentes popups de la interfaz al cambiar los estados correspondientes.
Uso: abrirPopupActualizarInv(), abrirPopupVeri(), abrirPopupV(), abrirPopupIv()

Inicio.jsx

1. Importación de Librerías y Componentes
El código comienza importando las librerías y componentes necesarios para el funcionamiento de la página. Algunas de las importaciones clave incluyen:

React y sus ganchos (useEffect, useState): Para la creación de componentes y el manejo de efectos secundarios.
Material-UI: Se importan los componentes TextField, Button y Box de Material-UI para construir la interfaz de usuario.
Axios: Se utiliza para realizar solicitudes HTTP.
useNavigate de React Router DOM: Permite la navegación programática entre diferentes vistas de la aplicación.
Componentes personalizados PopupRegistro y PopupRegistroS: Utilizados para mostrar ventanas emergentes de registro.
2. Estado del Componente
El componente utiliza el estado de React para gestionar la información del formulario de registro (datosFormulario), el estado de carga (Cargando), y el nombre de usuario (nombreUsuario). Estos estados se actualizan en respuesta a diferentes interacciones del usuario y eventos en la aplicación.

3. Funciones Auxiliares
Se definen varias funciones auxiliares que se utilizan para mostrar alertas (mostrarAlertaLogOut, mostrarAlertaRegistroFallido, etc.), cerrar sesión (cerrarSesion), obtener el nombre de usuario (obtenerNombreUsuario), y realizar el registro de usuarios (registrarUsuario).

4. Efectos y Ciclo de Vida
Se utiliza el gancho useEffect para realizar llamadas a la API al cargar la página y obtener el nombre del usuario. Además, el componente está estructurado para que algunos efectos se ejecuten solo una vez al montar el componente.

5. Manipulación de la Navegación
Existen funciones que manejan la navegación a diferentes secciones de la página o redirigen a otras páginas de la aplicación (redirectToHotelPage1, redirectToReservaciones, etc.).

Main.jsx

1. Importación de Librerías y Componentes
React: La librería principal de React.
React Router DOM: Se importan las funciones y componentes necesarios para establecer la navegación mediante enrutamiento en la aplicación (BrowserRouter, Routes, Route).
React DOM Client (createRoot): Se utiliza para crear un punto de entrada para la renderización en el DOM.
2. Creación del Punto de Entrada y Renderización
Se utiliza createRoot para especificar el punto de entrada ('root') donde se renderizará la aplicación React. La aplicación se renderiza en modo estricto (React.StrictMode), lo que ayuda a detectar posibles problemas en la aplicación durante el desarrollo.
3. Configuración del Enrutamiento
Se utiliza Router para envolver las rutas de la aplicación. Dentro de Routes, se definen diferentes rutas usando el componente Route. Cada Route especifica un path (ruta URL) y el componente de React asociado a esa ruta (element prop).
4. Definición de Rutas y Componentes Asociados
Se definen rutas para diferentes secciones de la aplicación, como la página de inicio, páginas de hoteles, inicio de sesión, etc. Cada ruta está asociada a un componente de React específico que se renderizará cuando la URL coincida con la ruta especificada.

Main.jsx

1. Importación de Librerías y Componentes
React: La librería principal de React.
React Router DOM: Se importan las funciones y componentes necesarios para establecer la navegación mediante enrutamiento en la aplicación (BrowserRouter, Routes, Route).
React DOM Client (createRoot): Se utiliza para crear un punto de entrada para la renderización en el DOM.
2. Creación del Punto de Entrada y Renderización
Se utiliza createRoot para especificar el punto de entrada ('root') donde se renderizará la aplicación React. La aplicación se renderiza en modo estricto (React.StrictMode), lo que ayuda a detectar posibles problemas en la aplicación durante el desarrollo.
3. Configuración del Enrutamiento
Se utiliza Router para envolver las rutas de la aplicación. Dentro de Routes, se definen diferentes rutas usando el componente Route. Cada Route especifica un path (ruta URL) y el componente de React asociado a esa ruta (element prop).
4. Definición de Rutas y Componentes Asociados
Se definen rutas para diferentes secciones de la aplicación, como la página de inicio, páginas de hoteles, inicio de sesión, etc. Cada ruta está asociada a un componente de React específico que se renderizará cuando la URL coincida con la ruta especificada.

Uso y Flujo de la Aplicación

Cuando un usuario navega a una URL específica, el sistema de enrutamiento dirigirá la aplicación al componente correspondiente asociado a esa ruta.
Los componentes como Inicio, PaginaHotel1, etc., se renderizarán según la ruta especificada en el navegador.

paginaHotel1,2,3,4,5,6.jsx

1.Importación de Librerías y Componentes
React y React Hooks: Importa React y varios hooks de React para el manejo de estado y efectos laterales.
Material-UI Components: Utiliza componentes de Material-UI, como Button, Select, MenuItem, y TextField.
axios: Librería para realizar solicitudes HTTP.
react-datepicker: Componente para seleccionar fechas de forma interactiva.
react-icons/fa: Iconos de Font Awesome empaquetados como componentes de React.
useNavigate: Hook de react-router-dom para la navegación programática.
sweetalert: Librería para mostrar alertas personalizadas.
2. Estado del Componente
quantity: Estado para la cantidad de personas en la reservación.
checkInDate y checkOutDate: Estados para las fechas de check-in y check-out.
reservationData: Objeto que almacena la información de la reservación.
nombreUsuario: Estado para almacenar el nombre del usuario.
showReservationDetails: Estado para mostrar o no detalles de la reservación.
3. Funciones para Alertas y Redirecciones
Se definen varias funciones que utilizan la librería sweetalert para mostrar alertas de usuario y funciones para redirigir a diferentes páginas.
4. Funciones de Manipulación de Estado
Se definen funciones para manejar cambios en el estado, como incrementar y decrementar la cantidad de personas, cambiar las fechas de check-in y check-out, y cambiar la cantidad de personas.
5. Llamadas a Backend
Funciones para interactuar con el backend, como obtener el nombre de usuario, cerrar sesión y realizar una reservación.
6. Efecto Secundario al Montar el Componente
Utiliza el hook useEffect para obtener el nombre del usuario al cargar el componente.
7. Renderización del Componente
El código JSX renderiza la interfaz de usuario, mostrando detalles del hotel, formularios de reservación y otros elementos interactivos.

Universidad Veracruzana

BackEnd

App.java

Paquete: mx.uv
Dependencias:
spark - Framework web para Java
java.util, java.util.UUID - Utilidades de Java para manejar UUIDs y otras operaciones comunes.
Clase Principal: App
Atributos Estáticos
gson (tipo Gson): Instancia de la clase Gson para facilitar la conversión entre objetos Java y formato JSON.
usuarios (tipo HashMap<String, Usuario>): Mapa que almacena usuarios por su ID.
correoG, passwordG, nombreG, idG (tipo String): Variables globales para almacenar información del usuario actual.
Métodos Estáticos
main(String[] args): Método principal que inicia la aplicación. Configura el puerto, maneja CORS, y define diferentes endpoints para la API.

getHerokuAssignedPort(): Método que obtiene el puerto asignado por Heroku, o devuelve un puerto predeterminado si no está disponible.

Endpoints
GET /backend/verificar-conexion: Verifica la conexión al backend y devuelve un mensaje JSON.

POST /frontend/: Crea un nuevo usuario a partir de los datos proporcionados en el cuerpo de la solicitud. Almacena el usuario en la base de datos y devuelve un mensaje JSON con el ID del usuario.

POST /frontend/correoExiste: Verifica si un correo electrónico ya está registrado. Devuelve un JSON indicando si el correo existe.

POST /frontend/obtenerUsuario: Obtiene los datos del usuario actualmente autenticado y los devuelve en formato JSON.

POST /frontend/cerrarSesion: Cierra la sesión del usuario actual y devuelve los datos del usuario en formato JSON.

POST /frontend/login: Autentica a un usuario mediante la verificación de correo electrónico y contraseña. Devuelve un mensaje JSON indicando la validez de la autenticación.

POST /frontend/RecuperarContra: Verifica si un usuario con el correo proporcionado existe para recuperar la contraseña.

POST /frontend/ColocarContra2: Cambia la contraseña de un usuario.

POST /frontend/hacerReservacionHotel[1-6]: Realiza una reservación de hotel según los datos proporcionados en el cuerpo de la solicitud. Devuelve un mensaje JSON indicando el resultado de la operación.

POST /frontend/obtenerReservaciones: Obtiene las reservaciones del usuario actual y las devuelve en formato JSON.

POST /frontend/eliminarReservacion: Elimina una reservación según el ID proporcionado en el cuerpo de la solicitud. Devuelve un mensaje JSON indicando el resultado de la operación.

DAO.java

Clase DAO
La clase DAO (Data Access Object) proporciona métodos para realizar operaciones relacionadas con la base de datos, como obtener usuarios, crear usuarios, validar credenciales, entre otros.

Atributos
c (privado y estático): Una instancia de la clase Conexion que se utiliza para establecer la conexión a la base de datos.
Métodos
dameUsuarios(): List<Usuario>

Descripción: Este método obtiene una lista de todos los usuarios almacenados en la base de datos.
Parámetros: Ninguno.
Retorno: Lista de objetos Usuario.
crearUsuario(u: Usuario): String

Descripción: Este método crea un nuevo usuario en la base de datos.
Parámetros:
u (Tipo Usuario): Objeto que representa al usuario a ser creado.
Retorno: Mensaje indicando si la operación fue exitosa o no.
validarUsuario(correo: String, password: String): boolean

Descripción: Valida las credenciales de un usuario en la base de datos.
Parámetros:
correo (Tipo String): Correo electrónico del usuario.
password (Tipo String): Contraseña del usuario.
Retorno: true si las credenciales son válidas, false de lo contrario.
correoExistente(correo: String): boolean

Descripción: Verifica si ya existe un usuario con el correo proporcionado.
Parámetros:
correo (Tipo String): Correo electrónico a verificar.
Retorno: true si el correo ya existe, false de lo contrario.
existeUsuarioPorCorreo(correo: String): boolean

Descripción: Verifica si existe al menos un usuario con el correo proporcionado.
Parámetros:
correo (Tipo String): Correo electrónico a verificar.
Retorno: true si existe al menos un usuario con el correo, false de lo contrario.
cambiarContrasena(correo: String, nuevaContrasena: String): String

Descripción: Cambia la contraseña de un usuario en la base de datos.
Parámetros:
correo (Tipo String): Correo electrónico del usuario.
nuevaContrasena (Tipo String): Nueva contraseña a establecer.
Retorno: Mensaje indicando si la operación fue exitosa o no.
obtenerIdUsuario(correo: String, password: String): String

Descripción: Obtiene el ID de un usuario basado en las credenciales proporcionadas.
Parámetros:
correo (Tipo String): Correo electrónico del usuario.
password (Tipo String): Contraseña del usuario.
Retorno: ID del usuario si las credenciales son válidas, "falso" de lo contrario.
obtenerDatosUsuario(id: String): Usuario

Descripción: Obtiene los datos de un usuario dado su ID.
Parámetros:
id (Tipo String): ID del usuario.
Retorno: Objeto Usuario con los datos del usuario, o null si no se encuentra.
Métodos Relacionados con Reservaciones
hacerReservacion(r: Reservaciones): String

Descripción: Realiza una reservación en la base de datos.
Parámetros:
r (Tipo Reservaciones): Objeto que representa la reservación a realizar.
Retorno: Mensaje indicando si la operación fue exitosa o no.
obtenerNumeroReservaciones(idUsuario: String): int

Descripción: Obtiene el número de reservaciones realizadas por un usuario.
Parámetros:

idUsuario (Tipo String): ID del usuario.
Retorno: Número de reservaciones realizadas por el usuario.
dameReservacionesPorUsuario(idUsuario: String): List<Reservaciones>

Descripción: Obtiene la lista de reservaciones realizadas por un usuario.
Parámetros:
idUsuario (Tipo String): ID del usuario.
Retorno: Lista de objetos Reservaciones realizadas por el usuario.
eliminarReservacion(idReservacion: String): boolean

Descripción: Elimina una reservación en la base de datos.
Parámetros:
idReservacion (Tipo String): ID de la reservación a eliminar.
Retorno: true si la reservación se eliminó con éxito, false de lo contrario.

Conexión.java

Clase Conexion
Atributos
url (tipo String): URL de conexión a la base de datos MySQL.
driverName (tipo String): Nombre del controlador JDBC para MySQL.
username (tipo String): Nombre de usuario para autenticarse en la base de datos.
password (tipo String): Contraseña para autenticarse en la base de datos.
connection (tipo Connection): Objeto de conexión que representa la conexión a la base de datos.
Métodos
getConnection(): Método que establece una conexión a la base de datos y devuelve el objeto de conexión (Connection). Este método utiliza el patrón Singleton para garantizar una única instancia de conexión durante toda la ejecución de la aplicación.
Funcionamiento
Configuración de la Conexión:

URL (url): Especifica la ubicación y el puerto de la base de datos (jdbc:mysql://db4free.net:3306/mcndhoteles).
Driver (driverName): Especifica el controlador JDBC para MySQL (com.mysql.cj.jdbc.Driver).
Usuario (username): Especifica el nombre de usuario para la conexión a la base de datos (mcndhoteles).
Contraseña (password): Especifica la contraseña para la conexión a la base de datos (mcndhoteles).
Método getConnection():

Intenta cargar dinámicamente el controlador JDBC (Class.forName(driverName)).
Establece la conexión a la base de datos utilizando la información proporcionada (URL, usuario, contraseña).
Maneja las excepciones de SQLException y ClassNotFoundException imprimiendo mensajes de error en la consola.

Usuario.java

Atributos
id (tipo String): Identificador único del usuario.
correo (tipo String): Dirección de correo electrónico del usuario.
password (tipo String): Contraseña del usuario.
nombre (tipo String): Nombre del usuario.
Métodos
Getters y Setters
getId() y setId(String id):
Descripción: Obtienen y establecen el identificador único del usuario.
getCorreo() y setCorreo(String correo):
Descripción: Obtienen y establecen la dirección de correo electrónico del usuario.
getPassword() y setPassword(String password):
Descripción: Obtienen y establecen la contraseña del usuario.
getNombre() y setNombre(String nombre):
Descripción: Obtienen y establecen el nombre del usuario.
toString()
Descripción: Sobrescribe el método toString() para proporcionar una representación en cadena del objeto Usuario. La cadena resultante incluye los valores de los atributos (id, correo, password, nombre).
Constructores
Usuario():

Descripción: Constructor por defecto que no recibe parámetros. No realiza ninguna acción específica.
Usuario(String id, String correo, String password, String nombre):

Descripción: Constructor que recibe valores para inicializar los atributos id, correo, password y nombre al crear una instancia de la clase.
Funcionamiento

Reservaciones.java

Atributos
idR (tipo String): Identificador único de la reservación.
idU (tipo String): Identificador único del usuario asociado a la reservación.
idH (tipo String): Identificador único del hotel asociado a la reservación.
nombre (tipo String): Nombre relacionado con la reservación.
precio (tipo String): Precio asociado a la reservación.
checkIn (tipo String): Fecha de inicio de la reservación.
checkOut (tipo String): Fecha de finalización de la reservación.
personas (tipo String): Número de personas asociadas a la reservación.

Universidad Veracruzana

Métodos
Getters y Setters
getIdR() y setIdR(String idR):

Descripción: Obtienen y establecen el identificador único de la reservación.
getIdU() y setIdU(String idU):

Descripción: Obtienen y establecen el identificador único del usuario asociado a la reservación.
getIdH() y setIdH(String idH):

Descripción: Obtienen y establecen el identificador único del hotel asociado a la reservación.
getNombre() y setNombre(String nombre):

Descripción: Obtienen y establecen el nombre relacionado con la reservación.
getPrecio() y setPrecio(String precio):

Descripción: Obtienen y establecen el precio asociado a la reservación.
getCheckIn() y setCheckIn(String checkIn):

Descripción: Obtienen y establecen la fecha de inicio de la reservación.
getCheckOut() y setCheckOut(String checkOut):

Descripción: Obtienen y establecen la fecha de finalización de la reservación.
getPersonas() y setPersonas(String personas):

Descripción: Obtienen y establecen el número de personas asociadas a la reservación.
toString()
Descripción: Sobrescribe el método toString() para proporcionar una representación en cadena del objeto Reservaciones. La cadena resultante incluye los valores de los atributos (idR, nombre, precio, checkIn, checkOut, personas).
Constructores
Reservaciones():

Descripción: Constructor por defecto que no recibe parámetros. No realiza ninguna acción específica.
Reservaciones(String idR, String idU, String idH, String nombre, String precio, String checkIn, String checkOut, String personas):

Descripción: Constructor que recibe valores para inicializar los atributos al crear una instancia de la clase.
Funcionamiento
La clase Reservaciones representa un modelo simple para almacenar información relacionada con reservaciones. Los atributos privados se acceden a través de métodos getters y setters para garantizar el

encapsulamiento. La sobrescritura del método toString() facilita la obtención de una representación en cadena del objeto para fines de depuración y registro.