**Documentación del proyecto Java**

**Documentación de los controladores:**

Documentación del Controlador ClienteController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad Cliente, así como algunas consultas adicionales relacionadas con la llegada y salida de los clientes en un hotel.

Rutas Base:

Todas las rutas se basan en "/clientes".

1. Crear un nuevo cliente

- Método: POST

- Ruta: "/"

- Descripción: Crea un nuevo cliente en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa al cliente a ser creado.

2. Actualizar un cliente por su ID

- Método: PUT

- Ruta: "/{id}"

- Descripción: Actualiza un cliente existente por su ID.

- Parámetros: El ID del cliente se toma de la URL, y se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados del cliente.

3. Borrar un cliente por su ID

- Método: DELETE

- Ruta: "/{id}"

- Descripción: Borra un cliente existente por su ID.

- Parámetros: El ID del cliente se toma de la URL.

4. Consultar todos los clientes

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todos los clientes en la base de datos.

5. Consultar un cliente por su ID

- Método: GET

- Ruta: "/{id}"

- Descripción: Recupera un cliente específico por su ID.

- Parámetros: El ID del cliente se toma de la URL.

6. Consultar la llegada de un cliente al hotel por fecha de entrada

- Método: GET

- Ruta: "/llegada"

- Descripción: Recupera la lista de clientes que llegaron al hotel en una fecha de entrada específica.

- Parámetros: Se espera un parámetro de consulta "fechaEntrada" que representa la fecha de entrada en formato de texto.

7. Consultar la salida de un cliente al hotel por fecha de salida

- Método: GET

- Ruta: "/salida"

- Descripción: Recupera la lista de clientes que salieron del hotel en una fecha de salida específica.

- Parámetros: Se espera un parámetro de consulta "fechaSalida" que representa la fecha de salida en formato de texto.

Notas Adicionales:

- Este controlador utiliza el repositorio "ClienteRepository" para acceder a la base de datos y realizar las operaciones CRUD.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre el cliente con el ID especificado.

Documentación del Controlador ConsumoController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad Consumo, que tiene una clave primaria compuesta. También proporciona una consulta adicional para buscar consumos por parte del cliente y registro de consumo.

Rutas Base:

Todas las rutas se basan en "/consumos".

1. Crear un nuevo consumo

- Método: POST

- Ruta: "/"

- Descripción: Crea un nuevo consumo en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa el consumo a ser creado.

2. Actualizar un consumo por su clave primaria compuesta

- Método: PUT

- Ruta: "/"

- Descripción: Actualiza un consumo existente por su clave primaria compuesta.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados del consumo.

3. Borrar un consumo por su clave primaria compuesta

- Método: DELETE

- Ruta: "/"

- Descripción: Borra un consumo existente por su clave primaria compuesta.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene la clave primaria compuesta del consumo a ser eliminado.

4. Consultar todos los consumos

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todos los consumos en la base de datos.

5. Consultar un consumo por su clave primaria compuesta

- Método: GET

- Ruta: "/"

- Descripción: Recupera un consumo específico por su clave primaria compuesta.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene la clave primaria compuesta del consumo a ser consultado.

6. Consultar consumos por parte del cliente

- Método: GET

- Ruta: "/consultarPorCliente"

- Descripción: Recupera una colección de consumos por parte del cliente y registro de consumo.

- Parámetros: Se espera un parámetro de consulta "nombreCliente" que representa el nombre del cliente y "registroConsumo" que representa el registro de consumo.

Notas Adicionales:

- Este controlador utiliza el repositorio "ConsumoRepository" para acceder a la base de datos y realizar las operaciones CRUD y consultas.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre el consumo con la clave primaria compuesta o no se cumplan las condiciones de las consultas.

Documentación del Controlador HabitacionController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad Habitación, así como una consulta adicional para buscar habitaciones por tipo.

Rutas Base:

Todas las rutas se basan en "/habitaciones".

1. Crear una nueva habitación

- Método: POST

- Ruta: "/"

- Descripción: Crea una nueva habitación en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa la habitación a ser creada.

2. Actualizar una habitación por su ID

- Método: PUT

- Ruta: "/{id}"

- Descripción: Actualiza una habitación existente por su ID.

- Parámetros: El ID de la habitación se toma de la URL, y se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados de la habitación.

3. Borrar una habitación por su ID

- Método: DELETE

- Ruta: "/{id}"

- Descripción: Borra una habitación existente por su ID.

- Parámetros: El ID de la habitación se toma de la URL.

4. Consultar todas las habitaciones

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todas las habitaciones en la base de datos.

5. Consultar una habitación por su ID

- Método: GET

- Ruta: "/{id}"

- Descripción: Recupera una habitación específica por su ID.

- Parámetros: El ID de la habitación se toma de la URL.

6. Consultar habitaciones por tipo

- Método: GET

- Ruta: "/consultarPorTipo"

- Descripción: Recupera una colección de habitaciones por tipo.

- Parámetros: Se espera un parámetro de consulta "tipo" que representa el tipo de habitación.

Notas Adicionales:

- Este controlador utiliza el repositorio "HabitacionRepository" para acceder a la base de datos y realizar las operaciones CRUD y consultas.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre la habitación con el ID especificado o no se cumplan las condiciones de las consultas.

Documentación del Controlador HotelController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad Hotel.

Rutas Base:

Todas las rutas se basan en "/hoteles".

1. Crear un nuevo hotel

- Método: POST

- Ruta: "/"

- Descripción: Crea un nuevo hotel en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa el hotel a ser creado.

2. Actualizar un hotel por su ID

- Método: PUT

- Ruta: "/{id}"

- Descripción: Actualiza un hotel existente por su ID.

- Parámetros: El ID del hotel se toma de la URL, y se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados del hotel.

3. Borrar un hotel por su ID

- Método: DELETE

- Ruta: "/{id}"

- Descripción: Borra un hotel existente por su ID.

- Parámetros: El ID del hotel se toma de la URL.

4. Consultar todos los hoteles

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todos los hoteles en la base de datos.

5. Consultar un hotel por su ID

- Método: GET

- Ruta: "/{id}"

- Descripción: Recupera un hotel específico por su ID.

- Parámetros: El ID del hotel se toma de la URL.

Notas Adicionales:

- Este controlador utiliza el repositorio "HotelRepository" para acceder a la base de datos y realizar las operaciones CRUD.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre el hotel con el ID especificado.

Documentación del Controlador PlanDeConsumoController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad PlanDeConsumo.

Rutas Base:

Todas las rutas se basan en "/planesDeConsumo".

1. Crear un nuevo PlanDeConsumo

- Método: POST

- Ruta: "/"

- Descripción: Crea un nuevo PlanDeConsumo en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa el PlanDeConsumo a ser creado.

2. Actualizar un PlanDeConsumo por su ID

- Método: PUT

- Ruta: "/{id}"

- Descripción: Actualiza un PlanDeConsumo existente por su ID.

- Parámetros: El ID del PlanDeConsumo se toma de la URL, y se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados del PlanDeConsumo.

3. Borrar un PlanDeConsumo por su ID

- Método: DELETE

- Ruta: "/{id}"

- Descripción: Borra un PlanDeConsumo existente por su ID.

- Parámetros: El ID del PlanDeConsumo se toma de la URL.

4. Consultar todos los PlanesDeConsumo

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todos los PlanesDeConsumo en la base de datos.

5. Consultar un PlanDeConsumo por su ID

- Método: GET

- Ruta: "/{id}"

- Descripción: Recupera un PlanDeConsumo específico por su ID.

- Parámetros: El ID del PlanDeConsumo se toma de la URL.

Notas Adicionales:

- Este controlador utiliza el repositorio "PlanDeConsumoRepository" para acceder a la base de datos y realizar las operaciones CRUD.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre el PlanDeConsumo con el ID especificado.

Documentación del Controlador ProducClienteController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad ProducCliente. La entidad ProducCliente tiene una clave primaria compuesta.

Rutas Base:

Todas las rutas se basan en "/producclientes".

1. Crear un nuevo ProducCliente

- Método: POST

- Ruta: "/"

- Descripción: Crea un nuevo ProducCliente en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa el ProducCliente a ser creado.

2. Actualizar un ProducCliente por su clave primaria compuesta

- Método: PUT

- Ruta: "/"

- Descripción: Actualiza un ProducCliente existente por su clave primaria compuesta.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados del ProducCliente. La clave primaria compuesta se extrae de los datos proporcionados.

3. Borrar un ProducCliente por su clave primaria compuesta

- Método: DELETE

- Ruta: "/"

- Descripción: Borra un ProducCliente existente por su clave primaria compuesta.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene la clave primaria compuesta del ProducCliente a ser borrado.

4. Consultar todos los ProducClientes

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todos los ProducClientes en la base de datos.

5. Consultar un ProducCliente por su clave primaria compuesta

- Método: GET

- Ruta: "/"

- Descripción: Recupera un ProducCliente específico por su clave primaria compuesta.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene la clave primaria compuesta del ProducCliente a ser consultado.

Notas Adicionales:

- Este controlador utiliza el repositorio "ProducClienteRepository" para acceder a la base de datos y realizar las operaciones CRUD.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre el ProducCliente con la clave primaria compuesta especificada.

Documentación del Controlador ProducServicioController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad ProducServicio. La entidad ProducServicio tiene una clave primaria compuesta.

Rutas Base:

Todas las rutas se basan en "/producservicios".

1. Crear un nuevo ProducServicio

- Método: POST

- Ruta: "/"

- Descripción: Crea un nuevo ProducServicio en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa el ProducServicio a ser creado.

2. Actualizar un ProducServicio por su clave primaria compuesta

- Método: PUT

- Ruta: "/"

- Descripción: Actualiza un ProducServicio existente por su clave primaria compuesta.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados del ProducServicio. La clave primaria compuesta se extrae de los datos proporcionados.

3. Borrar un ProducServicio por su clave primaria compuesta

- Método: DELETE

- Ruta: "/"

- Descripción: Borra un ProducServicio existente por su clave primaria compuesta.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene la clave primaria compuesta del ProducServicio a ser borrado.

4. Consultar todos los ProducServicios

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todos los ProducServicios en la base de datos.

5. Consultar un ProducServicio por su clave primaria compuesta

- Método: GET

- Ruta: "/"

- Descripción: Recupera un ProducServicio específico por su clave primaria compuesta.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene la clave primaria compuesta del ProducServicio a ser consultado.

Notas Adicionales:

- Este controlador utiliza el repositorio "ProducServicioRepository" para acceder a la base de datos y realizar las operaciones CRUD.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre el ProducServicio con la clave primaria compuesta especificada.

Documentación del Controlador ProductoController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad Producto.

Rutas Base:

Todas las rutas se basan en "/productos".

1. Crear un nuevo Producto

- Método: POST

- Ruta: "/"

- Descripción: Crea un nuevo Producto en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa el Producto a ser creado.

2. Actualizar un Producto por su ID

- Método: PUT

- Ruta: "/{id}"

- Descripción: Actualiza un Producto existente por su ID.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados del Producto. El ID del Producto se especifica en la ruta.

3. Borrar un Producto por su ID

- Método: DELETE

- Ruta: "/{id}"

- Descripción: Borra un Producto existente por su ID.

- Parámetros: El ID del Producto se especifica en la ruta.

4. Consultar todos los Productos

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todos los Productos en la base de datos.

5. Consultar un Producto por su ID

- Método: GET

- Ruta: "/{id}"

- Descripción: Recupera un Producto específico por su ID.

- Parámetros: El ID del Producto se especifica en la ruta.

Notas Adicionales:

- Este controlador utiliza el repositorio "ProductoRepository" para acceder a la base de datos y realizar las operaciones CRUD.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre el Producto con el ID especificado.

Documentación del Controlador ReservaController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad Reserva.

Rutas Base:

Todas las rutas se basan en "/reservas".

1. Crear una nueva Reserva

- Método: POST

- Ruta: "/"

- Descripción: Crea una nueva Reserva en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa la Reserva a ser creada.

2. Actualizar una Reserva por su ID

- Método: PUT

- Ruta: "/{id}"

- Descripción: Actualiza una Reserva existente por su ID.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados de la Reserva. El ID de la Reserva se especifica en la ruta.

3. Borrar una Reserva por su ID

- Método: DELETE

- Ruta: "/{id}"

- Descripción: Borra una Reserva existente por su ID.

- Parámetros: El ID de la Reserva se especifica en la ruta.

4. Consultar todas las Reservas

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todas las Reservas en la base de datos.

5. Consultar una Reserva por su ID

- Método: GET

- Ruta: "/{id}"

- Descripción: Recupera una Reserva específica por su ID.

- Parámetros: El ID de la Reserva se especifica en la ruta.

Notas Adicionales:

- Este controlador utiliza el repositorio "ReservaRepository" para acceder a la base de datos y realizar las operaciones CRUD.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre la Reserva con el ID especificado.

Documentación del Controlador ServicioController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad Servicio. Además, proporciona una operación personalizada para consultar la reserva de un servicio por parte de un cliente.

Rutas Base:

Todas las rutas se basan en "/servicios".

1. Crear un nuevo Servicio

- Método: POST

- Ruta: "/"

- Descripción: Crea un nuevo Servicio en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa el Servicio a ser creado.

2. Actualizar un Servicio por su ID

- Método: PUT

- Ruta: "/{id}"

- Descripción: Actualiza un Servicio existente por su ID.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados del Servicio. El ID del Servicio se especifica en la ruta.

3. Borrar un Servicio por su ID

- Método: DELETE

- Ruta: "/{id}"

- Descripción: Borra un Servicio existente por su ID.

- Parámetros: El ID del Servicio se especifica en la ruta.

4. Consultar todos los Servicios

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todos los Servicios en la base de datos.

5. Consultar un Servicio por su ID

- Método: GET

- Ruta: "/{id}"

- Descripción: Recupera un Servicio específico por su ID.

- Parámetros: El ID del Servicio se especifica en la ruta.

6. Consultar la reserva de un servicio por parte de un cliente

- Método: GET

- Ruta: "/consultarReservaPorCliente"

- Descripción: Recupera la lista de Servicios reservados por un cliente específico para un servicio en particular.

- Parámetros: Se esperan dos parámetros de consulta: "nombreCliente" (nombre del cliente) y "nombreServicio" (nombre del servicio).

Notas Adicionales:

- Este controlador utiliza el repositorio "ServicioRepository" para acceder a la base de datos y realizar las operaciones CRUD.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre el Servicio con el ID especificado.

Documentación del Controlador UsuarioController

Descripción:

Este controlador maneja las operaciones CRUD (Crear, Leer, Actualizar, Borrar) relacionadas con la entidad Usuario. Además, proporciona una operación personalizada para consultar usuarios por tipo.

Rutas Base:

Todas las rutas se basan en "/usuarios".

1. Crear un nuevo Usuario

- Método: POST

- Ruta: "/"

- Descripción: Crea un nuevo Usuario en la base de datos.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que representa el Usuario a ser creado.

2. Actualizar un Usuario por su ID

- Método: PUT

- Ruta: "/{id}"

- Descripción: Actualiza un Usuario existente por su ID.

- Parámetros: Se espera un objeto JSON en el cuerpo de la solicitud que contiene los datos actualizados del Usuario. El ID del Usuario se especifica en la ruta.

3. Borrar un Usuario por su ID

- Método: DELETE

- Ruta: "/{id}"

- Descripción: Borra un Usuario existente por su ID.

- Parámetros: El ID del Usuario se especifica en la ruta.

4. Consultar todos los Usuarios

- Método: GET

- Ruta: "/"

- Descripción: Recupera la lista de todos los Usuarios en la base de datos.

5. Consultar un Usuario por su ID

- Método: GET

- Ruta: "/{id}"

- Descripción: Recupera un Usuario específico por su ID.

- Parámetros: El ID del Usuario se especifica en la ruta.

6. Consultar usuarios por tipo

- Método: GET

- Ruta: "/consultarPorTipo"

- Descripción: Recupera una colección de Usuarios que coinciden con un tipo de usuario específico.

- Parámetros: Se espera un parámetro de consulta "tipo" que indica el tipo de usuario a buscar.

Notas Adicionales:

- Este controlador utiliza el repositorio "UsuarioRepository" para acceder a la base de datos y realizar las operaciones CRUD.

- Las respuestas a las solicitudes pueden ser respuestas exitosas (HTTP 200 OK) o respuestas de error (HTTP 404 Not Found) en caso de que no se encuentre el Usuario con el ID especificado o no se encuentren Usuarios del tipo especificado.

**Documentación de las entidades:**

Documentación de la Entidad Cliente

Descripción:

La entidad Cliente representa a los clientes de un hotel. Cada instancia de Cliente contiene información como el nombre del cliente, la cantidad de reservas realizadas, la cantidad de consumos y el tipo de plan al que está asociado.

Atributos:

- id\_clientes: Identificador único del cliente (clave primaria).

- nombre: El nombre del cliente.

- reservas: La cantidad de reservas realizadas por el cliente.

- comsumos: La cantidad de consumos realizados por el cliente.

- tipoDePlan: El tipo de plan al que está asociado el cliente.

Constructores:

1. Constructor por defecto: Inicializa un objeto Cliente sin parámetros.

2. Constructor con parámetros: Permite crear un objeto Cliente especificando todos los atributos, excepto el ID que se genera automáticamente.

Métodos Getters y Setters:

- Para cada atributo, se proporcionan métodos getter para acceder a su valor y métodos setter para establecer su valor.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y está mapeada a una tabla en la base de datos.

- La anotación `@Table` se utiliza para especificar el nombre de la tabla en la base de datos a la que está mapeada la entidad.

- El atributo `id\_clientes` está marcado con `@Id` y `@GeneratedValue`, lo que indica que es la clave primaria y su valor se generará automáticamente.

- La entidad Cliente se utiliza para representar a los clientes del hotel en la base de datos y puede estar relacionada con otras entidades, como Reserva o Consumo, dependiendo de la lógica de tu aplicación.

Documentación de la Entidad Consumo

Descripción:

La entidad Consumo representa un registro de consumo en un hotel. Cada instancia de Consumo contiene información sobre el consumo de un producto en un plan de consumo específico. La clave primaria de esta entidad es una clave primaria compuesta que incluye una referencia al plan de consumo, al producto, un registro y una fecha.

Atributos:

- pk: Objeto de tipo ConsumoPK que representa la clave primaria compuesta de la entidad. Contiene referencias al plan de consumo, al producto, un registro y una fecha.

Constructores:

1. Constructor por defecto: Inicializa un objeto Consumo sin parámetros.

2. Constructor con parámetros: Permite crear un objeto Consumo especificando el plan de consumo, el producto, el registro y la fecha.

Métodos Getters y Setters:

- El método `getPk()` permite acceder al objeto ConsumoPK que representa la clave primaria compuesta.

- El método `setPk()` permite establecer el valor del objeto ConsumoPK.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y está mapeada a una tabla en la base de datos.

- La anotación `@Table` se utiliza para especificar el nombre de la tabla en la base de datos a la que está mapeada la entidad.

- La entidad Consumo utiliza una clave primaria compuesta representada por el objeto ConsumoPK.

Documentación de la Clase ConsumoPK

Descripción:

La clase ConsumoPK representa la clave primaria compuesta de la entidad Consumo. Contiene referencias al plan de consumo, al producto, un registro y una fecha. Esta clave primaria compuesta se utiliza para identificar de manera única cada registro de consumo en la base de datos.

Atributos:

- planDeConsumo\_id\_plan: Objeto de tipo PlanDeConsumo que representa la referencia al plan de consumo al que pertenece el consumo.

- productos\_id\_productos: Objeto de tipo Producto que representa la referencia al producto consumido.

- registro: Cadena de texto que representa un registro o identificador único para el consumo.

- fecha: Objeto de tipo Date que representa la fecha en que se registró el consumo.

Constructores:

1. Constructor por defecto: Inicializa un objeto ConsumoPK sin parámetros.

2. Constructor con parámetros: Permite crear un objeto ConsumoPK especificando el plan de consumo, el producto, el registro y la fecha.

Métodos Getters y Setters:

- Los métodos getters permiten acceder a los valores de los atributos de la clave primaria compuesta.

- Los métodos setters permiten establecer los valores de los atributos de la clave primaria compuesta.

Notas Adicionales:

- La anotación `@Embeddable` se utiliza para indicar que esta clase es una clave primaria incrustada y se utilizará como parte de la clave primaria de otra entidad.

- La clase Consumo utiliza un objeto ConsumoPK como su clave primaria compuesta para identificar de manera única cada registro de consumo en la base de datos.

Documentación de la Clase Habitacion

Descripción:

La clase Habitacion representa una entidad que almacena información sobre las habitaciones en un hotel. Cada objeto de esta clase corresponde a una habitación en el hotel y contiene atributos como el tipo de habitación, la capacidad, la dotación, el costo y la cuenta asociada.

Atributos:

- id: Identificador único de la habitación.

- tipo: Cadena de texto que indica el tipo de habitación (por ejemplo, individual, doble, suite).

- capacidad: Entero que representa la capacidad máxima de ocupantes en la habitación.

- dotación: Entero que indica la dotación o comodidades incluidas en la habitación.

- costo: Entero que representa el costo de la habitación por noche.

- cuenta: Entero que identifica la cuenta asociada a la habitación.

Constructores:

1. Constructor por defecto: Inicializa un objeto Habitacion sin parámetros.

2. Constructor con parámetros: Permite crear un objeto Habitacion especificando el tipo, la capacidad, la dotación, el costo y la cuenta.

Métodos Getters y Setters:

- Los métodos getters permiten acceder a los valores de los atributos de la habitación.

- Los métodos setters permiten establecer los valores de los atributos de la habitación.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "bares" según la anotación `@Table`.

- El atributo "id" se utiliza como clave primaria de la entidad y se genera automáticamente utilizando la estrategia `GenerationType.AUTO`.

Documentación de la Clase Hotel

Descripción:

La clase Hotel representa una entidad que almacena información sobre hoteles. Cada objeto de esta clase corresponde a un hotel y contiene atributos como el nombre, la cadena hotelera a la que pertenece, la dirección y la categoría.

Atributos:

- id: Identificador único del hotel.

- nombre: Cadena de texto que indica el nombre del hotel.

- cadenaHotelera: Cadena de texto que especifica la cadena hotelera a la que pertenece el hotel.

- direccion: Cadena de texto que representa la dirección física del hotel.

- categoria: Cadena de texto que describe la categoría del hotel (por ejemplo, cinco estrellas, cuatro estrellas, etc.).

Constructores:

1. Constructor por defecto: Inicializa un objeto Hotel sin parámetros.

2. Constructor con parámetros: Permite crear un objeto Hotel especificando el nombre, la cadena hotelera, la dirección y la categoría.

Métodos Getters y Setters:

- Los métodos getters permiten acceder a los valores de los atributos del hotel.

- Los métodos setters permiten establecer los valores de los atributos del hotel.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "Hoteles" según la anotación `@Table`.

- El atributo "id" se utiliza como clave primaria de la entidad y se genera automáticamente utilizando la estrategia `GenerationType.AUTO`.

Documentación de la Clase PlanDeConsumo

Descripción:

La clase PlanDeConsumo representa una entidad que almacena información sobre planes de consumo en un hotel. Cada objeto de esta clase corresponde a un plan de consumo y contiene atributos como el nombre del plan, una descripción, un descuento aplicable y una lista de inclusiones.

Atributos:

- id\_plan: Identificador único del plan de consumo.

- nombre: Cadena de texto que indica el nombre del plan de consumo.

- descripcion: Cadena de texto que proporciona una descripción detallada del plan de consumo.

- descuento: Valor numérico que representa el descuento aplicable a este plan.

- inclusiones: Cadena de texto que describe las inclusiones o servicios que se ofrecen como parte de este plan.

Constructores:

1. Constructor por defecto: Inicializa un objeto PlanDeConsumo sin parámetros.

2. Constructor con parámetros: Permite crear un objeto PlanDeConsumo especificando el nombre, la descripción, el descuento y las inclusiones del plan.

Métodos Getters y Setters:

- Los métodos getters permiten acceder a los valores de los atributos del plan de consumo.

- Los métodos setters permiten establecer los valores de los atributos del plan de consumo.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "PlanesDeConsumo" según la anotación `@Table`.

- El atributo "id\_plan" se utiliza como clave primaria de la entidad y se genera automáticamente utilizando la estrategia `GenerationType.AUTO`.

Documentación de la Clase ProducCliente

Descripción:

La clase ProducCliente representa una entidad que almacena información sobre la relación entre clientes y productos. Cada objeto de esta clase corresponde a un cliente que ha adquirido un producto específico, y se utiliza para gestionar la relación entre clientes y productos en una base de datos.

Atributos:

- pk: Objeto de la clase ProducClientePK que almacena las claves primarias compuestas que identifican la relación entre un cliente y un producto.

Constructores:

1. Constructor por defecto: Inicializa un objeto ProducCliente sin parámetros.

2. Constructor con parámetros: Permite crear un objeto ProducCliente especificando el cliente y el producto relacionados.

Métodos Getters y Setters:

- El método getter `getPk` permite acceder al objeto ProducClientePK que contiene las claves primarias compuestas de la relación entre cliente y producto.

- El método setter `setPk` permite establecer el objeto ProducClientePK que representa la relación entre cliente y producto.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "ProductosDeClientes" según la anotación `@Table`.

- La relación entre clientes y productos se representa mediante la clave primaria compuesta almacenada en el atributo "pk" de esta clase.

Documento de Clase ProducCliente

Descripción:

La clase ProducCliente representa una entidad que almacena información sobre la relación entre clientes y productos. Cada objeto de esta clase corresponde a un cliente que ha adquirido un producto específico y se utiliza para gestionar la relación entre clientes y productos en una base de datos.

Atributos:

- pk: Objeto de la clase ProducClientePK que almacena las claves primarias compuestas que identifican la relación entre un cliente y un producto.

Constructores:

1. Constructor por defecto: Inicializa un objeto ProducCliente sin parámetros.

2. Constructor con parámetros: Permite crear un objeto ProducCliente especificando el cliente y el producto relacionados.

Métodos Getters y Setters:

- El método getter `getPk` permite acceder al objeto ProducClientePK que contiene las claves primarias compuestas de la relación entre cliente y producto.

- El método setter `setPk` permite establecer el objeto ProducClientePK que representa la relación entre cliente y producto.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "ProductosDeClientes" según la anotación `@Table`.

- La relación entre clientes y productos se representa mediante la clave primaria compuesta almacenada en el atributo "pk" de esta clase.

Documento de Clase ProducServicio

Descripción:

La clase ProducServicio es una entidad que almacena información sobre la relación entre servicios y productos. Cada objeto de esta clase representa un servicio que ha utilizado un producto específico, y se utiliza para gestionar la relación entre servicios y productos en una base de datos.

Atributos:

- pk: Objeto de la clase ProducServicioPK que almacena las claves primarias compuestas que identifican la relación entre un servicio y un producto.

Constructores:

1. Constructor por defecto: Inicializa un objeto ProducServicio sin parámetros.

2. Constructor con parámetros: Permite crear un objeto ProducServicio especificando el servicio y el producto relacionados.

Métodos Getters y Setters:

- El método getter `getPk` permite acceder al objeto ProducServicioPK que contiene las claves primarias compuestas de la relación entre servicio y producto.

- El método setter `setPk` permite establecer el objeto ProducServicioPK que representa la relación entre servicio y producto.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "ProductosDeServicios" según la anotación `@Table`.

- La relación entre servicios y productos se representa mediante la clave primaria compuesta almacenada en el atributo "pk" de esta clase.

Documento de Clase ProducServicioPK

Descripción:

La clase ProducServicioPK es una clase embebida (Embeddable) que almacena las claves primarias compuestas que identifican la relación entre servicios y productos. Cada objeto de esta clase representa una clave primaria compuesta que incluye un servicio y un producto, y se utiliza para gestionar la relación entre servicios y productos en una base de datos.

Atributos:

- servicio\_id\_servicio: Objeto de la clase Servicio que representa el servicio relacionado con la relación servicio-producto.

- productos\_id\_productos: Objeto de la clase Producto que representa el producto relacionado con la relación servicio-producto.

Constructores:

1. Constructor por defecto: Inicializa un objeto ProducServicioPK sin parámetros.

2. Constructor con parámetros: Permite crear un objeto ProducServicioPK especificando el servicio y el producto relacionados.

Métodos Getters y Setters:

- El método getter `getServicio\_id\_servicio` permite acceder al objeto Servicio que representa el servicio relacionado con la relación servicio-producto.

- El método setter `setServicio\_id\_servicio` permite establecer el objeto Servicio que representa el servicio relacionado con la relación servicio-producto.

- El método getter `getProductos\_id\_productos` permite acceder al objeto Producto que representa el producto relacionado con la relación servicio-producto.

- El método setter `setProductos\_id\_productos` permite establecer el objeto Producto que representa el producto relacionado con la relación servicio-producto.

Notas Adicionales:

- La clase ProducServicioPK se utiliza para definir la clave primaria compuesta de la entidad ProducServicio.

- La anotación `@Embeddable` indica que esta clase es embebida y se utiliza como parte de la clave primaria compuesta.

- La relación entre servicios y productos se representa mediante las anotaciones `@ManyToOne` y `@JoinColumn`, que especifican cómo se relacionan los objetos de esta clase con las entidades Servicio y Producto.

Documento de Clase Producto

Descripción:

La clase Producto es una entidad que almacena información sobre los productos disponibles en un sistema. Cada objeto de esta clase representa un producto con un identificador único, un nombre y un costo asociado. Se utiliza para gestionar la información de los productos en una base de datos.

Atributos:

- id\_productos: Identificador único del producto.

- nombre: Nombre del producto.

- costo: Costo del producto.

Constructores:

1. Constructor por defecto: Inicializa un objeto Producto sin parámetros.

2. Constructor con parámetros: Permite crear un objeto Producto especificando el nombre y el costo del producto.

Métodos Getters y Setters:

- El método getter `getId` permite acceder al identificador único del producto.

- El método setter `setId` permite establecer el identificador único del producto.

- El método getter `getNombre` permite acceder al nombre del producto.

- El método setter `setNombre` permite establecer el nombre del producto.

- El método getter `getCosto` permite acceder al costo del producto.

- El método setter `setCosto` permite establecer el costo del producto.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "Productos" según la anotación `@Table`.

- La anotación `@Id` indica que el atributo `id\_productos` es la clave primaria de la entidad Producto.

- La anotación `@GeneratedValue` especifica que el valor del identificador único se generará automáticamente.

Documento de Clase Reserva

Descripción:

La clase Reserva es una entidad que almacena información sobre las reservas realizadas en un sistema. Cada objeto de esta clase representa una reserva con un identificador único, una fecha de entrada, una fecha de salida y el número de personas asociadas a la reserva. Se utiliza para gestionar la información de las reservas en una base de datos.

Atributos:

- id: Identificador único de la reserva.

- fechaEntrada: Fecha de entrada de la reserva.

- fechaSalida: Fecha de salida de la reserva.

- numPersonas: Número de personas asociadas a la reserva.

Constructores:

1. Constructor por defecto: Inicializa un objeto Reserva sin parámetros.

2. Constructor con parámetros: Permite crear un objeto Reserva especificando la fecha de entrada, la fecha de salida y el número de personas.

Métodos Getters y Setters:

- El método getter `getId` permite acceder al identificador único de la reserva.

- El método setter `setId` permite establecer el identificador único de la reserva.

- El método getter `getFechaEntrada` permite acceder a la fecha de entrada de la reserva.

- El método setter `setFechaEntrada` permite establecer la fecha de entrada de la reserva.

- El método getter `getFechaSalida` permite acceder a la fecha de salida de la reserva.

- El método setter `setFechaSalida` permite establecer la fecha de salida de la reserva.

- El método getter `getNumPersonas` permite acceder al número de personas asociadas a la reserva.

- El método setter `setNumPersonas` permite establecer el número de personas asociadas a la reserva.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "Reservas" según la anotación `@Table`.

- La anotación `@Id` indica que el atributo `id` es la clave primaria de la entidad Reserva.

- La anotación `@GeneratedValue` especifica que el valor del identificador único se generará automáticamente.

Documento de Clase Servicio

Descripción:

La clase Servicio es una entidad que almacena información sobre los servicios ofrecidos en un sistema. Cada objeto de esta clase representa un servicio con un identificador único, un nombre, una descripción, un costo adicional, un horario y su disponibilidad. Se utiliza para gestionar la información de los servicios en una base de datos.

Atributos:

- id\_servicio: Identificador único del servicio.

- nombre: Nombre del servicio.

- descripcion: Descripción del servicio.

- costoAdicional: Costo adicional asociado al servicio.

- horario: Horario del servicio.

- disponibilidad: Disponibilidad del servicio.

Constructores:

1. Constructor por defecto: Inicializa un objeto Servicio sin parámetros.

2. Constructor con parámetros: Permite crear un objeto Servicio especificando el nombre, la descripción, el costo adicional, el horario y la disponibilidad del servicio.

Métodos Getters y Setters:

- El método getter `getId` permite acceder al identificador único del servicio.

- El método setter `setId` permite establecer el identificador único del servicio.

- El método getter `getNombre` permite acceder al nombre del servicio.

- El método setter `setNombre` permite establecer el nombre del servicio.

- El método getter `getDescripcion` permite acceder a la descripción del servicio.

- El método setter `setDescripcion` permite establecer la descripción del servicio.

- El método getter `getCostoAdicional` permite acceder al costo adicional del servicio.

- El método setter `setCostoAdicional` permite establecer el costo adicional del servicio.

- El método getter `getHorario` permite acceder al horario del servicio.

- El método setter `setHorario` permite establecer el horario del servicio.

- El método getter `getDisponibilidad` permite acceder a la disponibilidad del servicio.

- El método setter `setDisponibilidad` permite establecer la disponibilidad del servicio.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "Servicios" según la anotación `@Table`.

- La anotación `@Id` indica que el atributo `id\_servicio` es la clave primaria de la entidad Servicio.

- La anotación `@GeneratedValue` especifica que el valor del identificador único se generará automáticamente.

Documento de Clase Usuario

Descripción:

La clase Usuario es una entidad que almacena información sobre los usuarios en un sistema. Cada objeto de esta clase representa un usuario con un identificador único y un tipo. El tipo puede indicar el rol o categoría del usuario en el sistema. Se utiliza para gestionar la información de los usuarios en una base de datos.

Atributos:

- id: Identificador único del usuario.

- tipo: Tipo de usuario que puede representar su rol o categoría en el sistema.

Constructores:

1. Constructor por defecto: Inicializa un objeto Usuario sin parámetros.

2. Constructor con parámetros: Permite crear un objeto Usuario especificando el tipo del usuario.

Métodos Getters y Setters:

- El método getter `getId` permite acceder al identificador único del usuario.

- El método setter `setId` permite establecer el identificador único del usuario.

- El método getter `getTipo` permite acceder al tipo del usuario.

- El método setter `setTipo` permite establecer el tipo del usuario.

Notas Adicionales:

- La anotación `@Entity` indica que esta clase es una entidad JPA y se mapeará a una tabla en la base de datos con el nombre "Usuarios" según la anotación `@Table`.

- La anotación `@Id` indica que el atributo `id` es la clave primaria de la entidad Usuario.

- La anotación `@GeneratedValue` especifica que el valor del identificador único se generará automáticamente.

**Documentación de los repositorios:**

Interfaz ClienteRepository

Descripción:

La interfaz ClienteRepository define los métodos para acceder y manipular la información de los clientes en una base de datos.

Métodos:

1. `crearCliente`: Permite crear un nuevo cliente en la base de datos.

- Parámetros:

- `nombre` (String): El nombre del cliente.

- `reservas` (Long): La cantidad de reservas del cliente.

- `consumos` (Long): La cantidad de consumos del cliente.

- `tipoDePlan` (String): El tipo de plan del cliente.

2. `actualizarCliente`: Permite actualizar un cliente existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del cliente a actualizar.

- `nombre` (String): El nuevo nombre del cliente.

- `reservas` (Long): La nueva cantidad de reservas del cliente.

- `consumos` (Long): La nueva cantidad de consumos del cliente.

- `tipoDePlan` (String): El nuevo tipo de plan del cliente.

3. `borrarCliente`: Permite borrar un cliente de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del cliente a borrar.

4. `darClientes`: Consulta y devuelve una colección de todos los clientes en la base de datos.

5. `darCliente`: Consulta y devuelve un cliente por su ID.

- Parámetros:

- `id` (long): El ID del cliente a consultar.

6. `consultarLlegadaClienteAlHotel`: Consulta y devuelve una lista de clientes que han llegado al hotel en una fecha específica.

- Parámetros:

- `fechaEntrada` (String): La fecha de entrada al hotel en formato de cadena.

7. `consultarSalidaClienteAlHotel`: Consulta y devuelve una lista de clientes que han salido del hotel en una fecha específica.

- Parámetros:

- `fechaSalida` (String): La fecha de salida del hotel en formato de cadena.

Interfaz ConsumoRepository

Descripción:

La interfaz ConsumoRepository define los métodos para acceder y manipular la información de los consumos en una base de datos. Incluye operaciones para crear, actualizar, borrar y consultar consumos, así como también para consultar consumos por parte de un cliente.

Métodos:

1. `crearConsumo`: Permite crear un nuevo consumo en la base de datos.

- Parámetros:

- `planId` (Long): El ID del plan de consumo asociado al consumo.

- `productoId` (Long): El ID del producto asociado al consumo.

- `registro` (String): El registro del consumo.

- `fecha` (Date): La fecha del consumo.

2. `actualizarConsumo`: Permite actualizar un consumo existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del consumo a actualizar.

- `planId` (Long): El nuevo ID del plan de consumo asociado al consumo.

- `productoId` (Long): El nuevo ID del producto asociado al consumo.

- `registro` (String): El nuevo registro del consumo.

- `fecha` (Date): La nueva fecha del consumo.

3. `borrarConsumo`: Permite borrar un consumo de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del consumo a borrar.

4. `darConsumos`: Consulta y devuelve una colección de todos los consumos en la base de datos.

5. `darConsumo`: Consulta y devuelve un consumo por su ID.

- Parámetros:

- `id` (long): El ID del consumo a consultar.

6. `consultarConsumoPorCliente`: Consulta y devuelve una lista de consumos realizados por un cliente específico.

- Parámetros:

- `nombreCliente` (String): El nombre del cliente.

- `registroConsumo` (String): El registro del consumo.

7. `findById`: Permite buscar un consumo por su clave primaria compuesta (ConsumoPK).

Interfaz HabitacionRepository

Descripción:

La interfaz HabitacionRepository define los métodos para acceder y manipular la información de las habitaciones en una base de datos.

Incluye operaciones para crear, actualizar, borrar y consultar habitaciones, así como también para consultar habitaciones por tipo.

Métodos:

1. `crearHabitacion`: Permite crear una nueva habitación en la base de datos.

- Parámetros:

- `tipo` (String): El tipo de la habitación.

- `capacidad` (Long): La capacidad máxima de la habitación.

- `costo` (Long): El costo de la habitación.

- `cuenta` (Long): El número de cuenta de la habitación.

- `idReserva` (Long): El ID de la reserva asociada a la habitación.

- `hotelId` (Long): El ID del hotel al que pertenece la habitación.

2. `actualizarHabitacion`: Permite actualizar una habitación existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID de la habitación a actualizar.

- `tipo` (String): El nuevo tipo de la habitación.

- `capacidad` (Long): La nueva capacidad máxima de la habitación.

- `costo` (Long): El nuevo costo de la habitación.

- `cuenta` (Long): El nuevo número de cuenta de la habitación.

- `idReserva` (Long): El nuevo ID de la reserva asociada a la habitación.

- `hotelId` (Long): El nuevo ID del hotel al que pertenece la habitación.

3. `borrarHabitacion`: Permite borrar una habitación de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID de la habitación a borrar.

4. `consultarPorTipo`: Consulta y devuelve una colección de habitaciones que coinciden con un tipo específico.

- Parámetros:

- `tipo` (String): El tipo de habitación a consultar.

5. `darHabitaciones`: Consulta y devuelve una colección de todas las habitaciones en la base de datos.

6. `darHabitacion`: Consulta y devuelve una habitación por su ID.

- Parámetros:

- `id` (long): El ID de la habitación a consultar.

Interfaz HotelRepository

Descripción:

La interfaz HotelRepository define los métodos para acceder y manipular la información de los hoteles en una base de datos.

Incluye operaciones para crear, actualizar, borrar y consultar hoteles.

Métodos:

1. `crearHotel`: Permite crear un nuevo hotel en la base de datos.

- Parámetros:

- `nombre` (String): El nombre del hotel.

- `cadenaHotelera` (String): La cadena hotelera a la que pertenece el hotel.

- `direccion` (String): La dirección del hotel.

- `categorIa` (String): La categoría o clasificación del hotel.

2. `actualizarHotel`: Permite actualizar un hotel existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del hotel a actualizar.

- `nombre` (String): El nuevo nombre del hotel.

- `cadenaHotelera` (String): La nueva cadena hotelera a la que pertenece el hotel.

- `direccion` (String): La nueva dirección del hotel.

- `categorIa` (String): La nueva categoría o clasificación del hotel.

3. `borrarHotel`: Permite borrar un hotel de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del hotel a borrar.

4. `darHoteles`: Consulta y devuelve una colección de todos los hoteles en la base de datos.

5. `darHotel`: Consulta y devuelve un hotel por su ID.

- Parámetros:

- `id` (long): El ID del hotel a consultar.

Interfaz PlanDeConsumoRepository

Descripción:

La interfaz PlanDeConsumoRepository define los métodos para acceder y manipular la información de los planes de consumo en una base de datos.

Incluye operaciones para crear, actualizar, borrar y consultar planes de consumo.

Métodos:

1. `crearPlanDeConsumo`: Permite crear un nuevo plan de consumo en la base de datos.

- Parámetros:

- `idPlan` (Long): El ID del plan de consumo.

- `nombre` (String): El nombre del plan de consumo.

- `descripcion` (String): La descripción del plan de consumo.

- `descuento` (Double): El descuento aplicado en el plan de consumo.

- `inclusiones` (String): Las inclusiones o servicios incluidos en el plan de consumo.

- `idReserva` (Long): El ID de la reserva asociada al plan de consumo.

- `clientesId` (Long): El ID del cliente asociado al plan de consumo.

- `hotelId` (Long): El ID del hotel al que pertenece el plan de consumo.

- `servicioId` (Long): El ID del servicio asociado al plan de consumo.

2. `actualizarPlanDeConsumo`: Permite actualizar un plan de consumo existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del plan de consumo a actualizar.

- `nombre` (String): El nuevo nombre del plan de consumo.

- `descripcion` (String): La nueva descripción del plan de consumo.

- `descuento` (Double): El nuevo descuento aplicado en el plan de consumo.

- `inclusiones` (String): Las nuevas inclusiones o servicios incluidos en el plan de consumo.

- `idReserva` (Long): El nuevo ID de la reserva asociada al plan de consumo.

- `clientesId` (Long): El nuevo ID del cliente asociado al plan de consumo.

- `hotelId` (Long): El nuevo ID del hotel al que pertenece el plan de consumo.

- `servicioId` (Long): El nuevo ID del servicio asociado al plan de consumo.

3. `borrarPlanDeConsumo`: Permite borrar un plan de consumo de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del plan de consumo a borrar.

4. `darPlanesDeConsumo`: Consulta y devuelve una colección de todos los planes de consumo en la base de datos.

5. `darPlanDeConsumo`: Consulta y devuelve un plan de consumo por su ID.

- Parámetros:

- `id` (Long): El ID del plan de consumo a consultar.

Interfaz ProducClienteRepository

Descripción:

La interfaz ProducClienteRepository define los métodos para acceder y manipular la información de la relación entre clientes y productos en una base de datos.

Incluye operaciones para crear, actualizar, borrar y consultar la relación entre clientes y productos.

Métodos:

1. `crearProducCliente`: Permite crear una nueva relación entre cliente y producto en la base de datos.

- Parámetros:

- `clientesId` (Long): El ID del cliente relacionado.

- `productosId` (Long): El ID del producto relacionado.

2. `actualizarProducCliente`: Permite actualizar una relación entre cliente y producto existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID de la relación entre cliente y producto a actualizar.

- `clientesId` (Long): El nuevo ID del cliente relacionado.

- `productosId` (Long): El nuevo ID del producto relacionado.

3. `borrarProducCliente`: Permite borrar una relación entre cliente y producto de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID de la relación entre cliente y producto a borrar.

4. `darProducClientes`: Consulta y devuelve una colección de todas las relaciones entre clientes y productos en la base de datos.

5. `darProducCliente`: Consulta y devuelve una relación entre cliente y producto por su ID.

- Parámetros:

- `id` (Long): El ID de la relación entre cliente y producto a consultar.

6. `findById`: Permite buscar una relación entre cliente y producto por su clave primaria compuesta (ProducClientePK).

Interfaz ProducServicioRepository

Descripción:

La interfaz ProducServicioRepository define los métodos para acceder y manipular la información de la relación entre servicios y productos en una base de datos.

Incluye operaciones para crear, actualizar, borrar y consultar la relación entre servicios y productos.

Métodos:

1. `crearProducServicio`: Permite crear una nueva relación entre servicio y producto en la base de datos.

- Parámetros:

- `servicioId` (Long): El ID del servicio relacionado.

- `productosId` (Long): El ID del producto relacionado.

2. `actualizarProducServicio`: Permite actualizar una relación entre servicio y producto existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID de la relación entre servicio y producto a actualizar.

- `servicioId` (Long): El nuevo ID del servicio relacionado.

- `productosId` (Long): El nuevo ID del producto relacionado.

3. `borrarProducServicio`: Permite borrar una relación entre servicio y producto de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID de la relación entre servicio y producto a borrar.

4. `darProducServicios`: Consulta y devuelve una colección de todas las relaciones entre servicios y productos en la base de datos.

5. `darProducServicio`: Consulta y devuelve una relación entre servicio y producto por su ID.

- Parámetros:

- `id` (Long): El ID de la relación entre servicio y producto a consultar.

6. `findById`: Permite buscar una relación entre servicio y producto por su clave primaria compuesta (ProducServicioPK).

Interfaz ProductoRepository

Descripción:

La interfaz ProductoRepository define los métodos para acceder y manipular la información de productos en una base de datos.

Incluye operaciones para crear, actualizar, borrar y consultar productos.

Métodos:

1. `crearProducto`: Permite crear un nuevo producto en la base de datos.

- Parámetros:

- `nombre` (String): El nombre del producto.

- `costo` (Long): El costo del producto.

2. `actualizarProducto`: Permite actualizar un producto existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del producto a actualizar.

- `nombre` (String): El nuevo nombre del producto.

- `costo` (Long): El nuevo costo del producto.

3. `borrarProducto`: Permite borrar un producto de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del producto a borrar.

4. `darProductos`: Consulta y devuelve una colección de todos los productos en la base de datos.

5. `darProducto`: Consulta y devuelve un producto por su ID.

- Parámetros:

- `id` (Long): El ID del producto a consultar.

Interfaz ReservaRepository

Descripción:

La interfaz ReservaRepository define los métodos para acceder y manipular la información de reservas en una base de datos.

Incluye operaciones para crear, actualizar, borrar y consultar reservas.

Métodos:

1. `crearReserva`: Permite crear una nueva reserva en la base de datos.

- Parámetros:

- `fechaEntrada` (String): La fecha de entrada de la reserva.

- `fechaSalida` (String): La fecha de salida de la reserva.

- `numPersonas` (Long): El número de personas en la reserva.

- `habitacionId` (Long): El ID de la habitación asociada a la reserva.

- `planDeConsumoId` (Long): El ID del plan de consumo asociado a la reserva.

- `clientesId` (Long): El ID del cliente asociado a la reserva.

- `hotelId` (Long): El ID del hotel asociado a la reserva.

2. `actualizarReserva`: Permite actualizar una reserva existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID de la reserva a actualizar.

- `fechaEntrada` (String): La nueva fecha de entrada de la reserva.

- `fechaSalida` (String): La nueva fecha de salida de la reserva.

- `numPersonas` (Long): El nuevo número de personas en la reserva.

- `habitacionId` (Long): El nuevo ID de la habitación asociada a la reserva.

- `planDeConsumoId` (Long): El nuevo ID del plan de consumo asociado a la reserva.

- `clientesId` (Long): El nuevo ID del cliente asociado a la reserva.

- `hotelId` (Long): El nuevo ID del hotel asociado a la reserva.

3. `borrarReserva`: Permite borrar una reserva de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID de la reserva a borrar.

4. `darReservas`: Consulta y devuelve una colección de todas las reservas en la base de datos.

5. `darReserva`: Consulta y devuelve una reserva por su ID.

- Parámetros:

- `id` (Long): El ID de la reserva a consultar.

Interfaz ServicioRepository

Descripción:

La interfaz ServicioRepository define los métodos para acceder y manipular la información de servicios en una base de datos.

Incluye operaciones para crear, actualizar, borrar y consultar servicios.

Métodos:

1. `crearServicio`: Permite crear un nuevo servicio en la base de datos.

- Parámetros:

- `nombre` (String): El nombre del servicio.

- `descripcion` (String): La descripción del servicio.

- `costoAdicional` (Long): El costo adicional del servicio.

- `horario` (Long): El horario del servicio.

- `disponibilidad` (Long): La disponibilidad del servicio.

- `hotelId` (Long): El ID del hotel asociado al servicio.

2. `actualizarServicio`: Permite actualizar un servicio existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del servicio a actualizar.

- `nombre` (String): El nuevo nombre del servicio.

- `descripcion` (String): La nueva descripción del servicio.

- `costoAdicional` (Long): El nuevo costo adicional del servicio.

- `horario` (Long): El nuevo horario del servicio.

- `disponibilidad` (Long): La nueva disponibilidad del servicio.

- `hotelId` (Long): El nuevo ID del hotel asociado al servicio.

3. `borrarServicio`: Permite borrar un servicio de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del servicio a borrar.

4. `darServicios`: Consulta y devuelve una colección de todos los servicios en la base de datos.

5. `darServicio`: Consulta y devuelve un servicio por su ID.

- Parámetros:

- `id` (Long): El ID del servicio a consultar.

6. `consultarReservaDeServicioPorCliente`: Consulta y devuelve una lista de servicios reservados por un cliente específico.

- Parámetros:

- `nombreCliente` (String): El nombre del cliente.

- `nombreServicio` (String): El nombre del servicio a consultar en las reservas del cliente.

Interfaz UsuarioRepository

Descripción:

La interfaz UsuarioRepository define los métodos para acceder y manipular la información de usuarios en una base de datos.

Incluye operaciones para crear, actualizar, borrar y consultar usuarios.

Métodos:

1. `crearUsuario`: Permite crear un nuevo usuario en la base de datos.

- Parámetros:

- `tipoDocumento` (String): El tipo de documento del usuario.

- `tipo` (String): El tipo de usuario.

- `hotelId` (Long): El ID del hotel al que está asociado el usuario.

- `numeroDocumento` (Long): El número de documento del usuario.

- `nombre` (String): El nombre del usuario.

- `correoElectronico` (String): El correo electrónico del usuario.

2. `actualizarUsuario`: Permite actualizar un usuario existente en la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del usuario a actualizar.

- `tipoDocumento` (String): El nuevo tipo de documento del usuario.

- `tipo` (String): El nuevo tipo de usuario.

- `hotelId` (Long): El nuevo ID del hotel al que está asociado el usuario.

- `numeroDocumento` (Long): El nuevo número de documento del usuario.

- `nombre` (String): El nuevo nombre del usuario.

- `correoElectronico` (String): El nuevo correo electrónico del usuario.

3. `borrarUsuario`: Permite borrar un usuario de la base de datos por su ID.

- Parámetros:

- `id` (Long): El ID del usuario a borrar.

4. `consultarUsuariosPorTipo`: Consulta y devuelve una colección de usuarios que tienen un tipo específico.

- Parámetros:

- `tipo` (String): El tipo de usuario a consultar.

5. `darUsuarios`: Consulta y devuelve una colección de todos los usuarios en la base de datos.

6. `darUsuario`: Consulta y devuelve un usuario por su ID.

- Parámetros:

- `id` (Long): El ID del usuario a consultar.