

Documentación de escenarios de prueba

En este documento se explicará el desarrollo de los escenarios de prueba para verificar el correcto funcionamiento de la aplicación. Para ello, fue necesario separar los escenarios en 3 partes principales:

1. CRUD de los RFs (interfaz gráfica).
2. Inserción incorrecta a entidades.
3. RFCs.

1. CRUD de los requerimientos funcionales:

Para probar las operaciones de CRUD de los RFs, primero fue necesario crear la interfaz gráfica de la aplicación, lo cual es explicado en la documentación del proyecto de SW. Ya con esto, una vez se ejecuta se puede probar cada operación. La aplicación como tal es muy intuitiva al indicar claramente qué botón realiza qué operación:

Create	Añadir...
Read	(Lista de documentos)
Update	Editar
Delete	Eliminar

Para mayor claridad, se mostrará un ejemplo de las operaciones CRUD con un tipo de habitación a continuación.

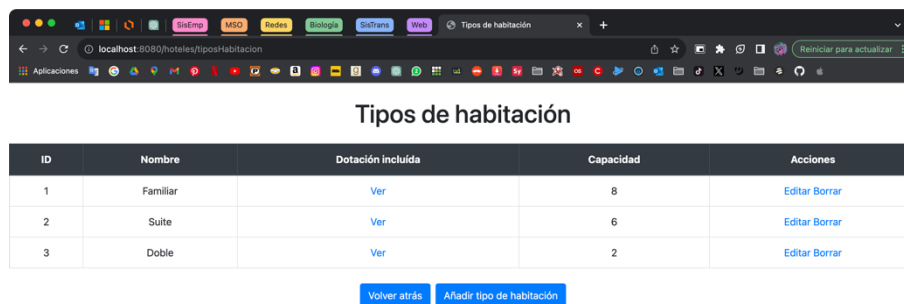
Read:

De momento, la colección tipos_habitacion cuenta con los siguientes documentos:

<pre>_id: 1 nombre: "Familiar" capacidad: 8 ▸ dotacion_incluida: Array (2)</pre>
<pre>_id: 2 nombre: "Suite" capacidad: 6 ▸ dotacion_incluida: Array (3)</pre>
<pre>_id: 3 nombre: "Doble" capacidad: 2 ▸ dotacion_incluida: Array (2)</pre>

Fig. 1: Colección tipos_habitacion

Que en la aplicación se ve así:

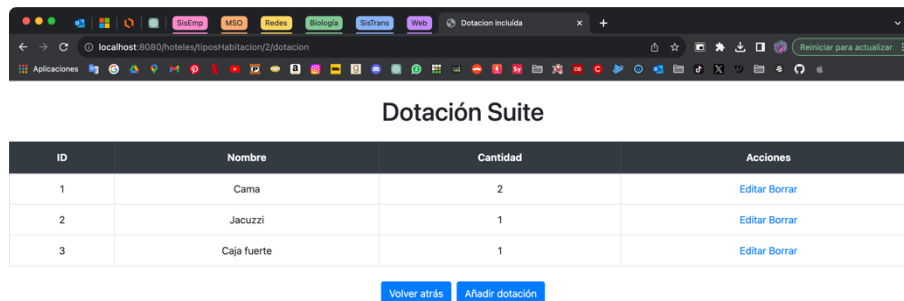


ID	Nombre	Dotación incluida	Capacidad	Acciones
1	Familiar	Ver	8	Editar Borrar
2	Suite	Ver	6	Editar Borrar
3	Doble	Ver	2	Editar Borrar

[Volver atrás](#) [Añadir tipo de habitación](#)

Fig. 2: Read tipos_habitacion

La dotación incluida, al ser un documento embebido, cuenta con una vista aparte:




ID	Nombre	Cantidad	Acciones
1	Cama	2	Editar Borrar
2	Jacuzzi	1	Editar Borrar
3	Caja fuerte	1	Editar Borrar

[Volver atrás](#) [Añadir dotación](#)

Fig. 3: Read tipos_habitacion.dotacion_incluida

Create:

Si se quiere crear un tipo de habitación, la vista de crear muestra lo siguiente:



Crear tipo de habitación

Nombre
Suite doble

Capacidad
7

Cancelar Guardar

Fig. 4: Create tipos_habitacion

Una vez se guarda, el tipo de habitación creado sale en la lista:



Tipos de habitación

ID	Nombre	Dotación incluida	Capacidad	Acciones
1	Familiar	Ver	8	Editar Borrar
2	Suite	Ver	6	Editar Borrar
3	Doble	Ver	2	Editar Borrar
4	Suite doble	Ver	7	Editar Borrar

Volver atrás Añadir tipo de habitación

Fig. 5: Read tipos_habitacion actualizado

No solo eso, sino que la colección en la base de datos también se actualiza:

<pre> _id: 1 nombre: "Familiar" capacidad: 8 ▸ dotacion_incluida: Array (2) </pre>
<pre> _id: 2 nombre: "Suite" capacidad: 6 ▸ dotacion_incluida: Array (3) </pre>
<pre> _id: 3 nombre: "Doble" capacidad: 2 ▸ dotacion_incluida: Array (2) </pre>
<pre> _id: 4 nombre: "Suite doble" capacidad: 7 _class: "uniandes.edu.co.hoteles.modelo.Tipo_habitacion" </pre>

Fig. 6: Colección tipos_habitacion actualizada

Como se puede ver, el tipo de habitación creado cuenta con un campo `_class` que indica que la creación se hizo a través del proyecto de Java y no directamente desde la base de datos.

Update:

Si se quiere editar un tipo de habitación, la vista de editar, con los campos ya editados, muestra lo siguiente:

Editar tipo de habitación

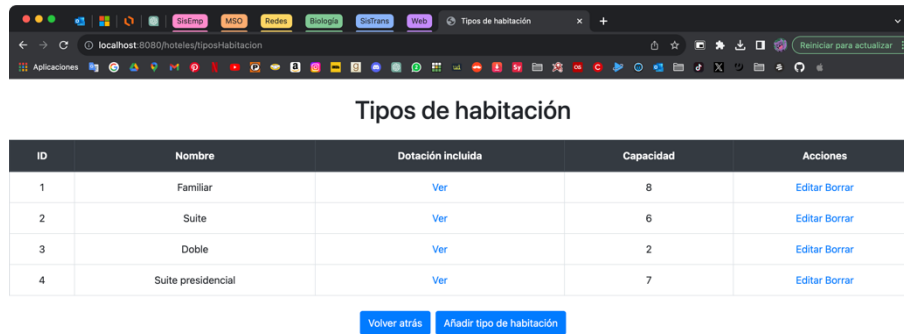
Nombre
Suite presidencial

Capacidad
7

Cancelar Guardar

Fig. 7: Update tipos_habitacion

Una vez se guarda, el tipo de habitación editado muestra los cambios en la lista:



ID	Nombre	Dotación incluida	Capacidad	Acciones
1	Familiar	Ver	8	Editar Borrar
2	Suite	Ver	6	Editar Borrar
3	Doble	Ver	2	Editar Borrar
4	Suite presidencial	Ver	7	Editar Borrar

Volver atrás Añadir tipo de habitación

Fig. 8: Read tipos_habitacion actualizado

No solo eso, sino que la colección en la base de datos también se actualiza:



```

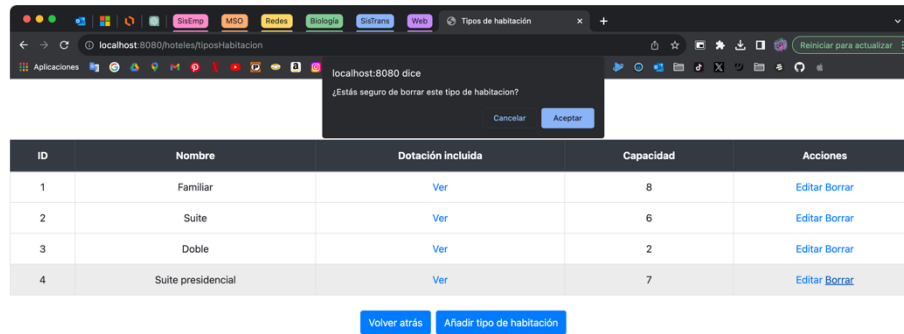
[
  {
    "_id": 1,
    "nombre": "Familiar",
    "capacidad": 8,
    "dotacion_incluida": Array (2)
  },
  {
    "_id": 2,
    "nombre": "Suite",
    "capacidad": 6,
    "dotacion_incluida": Array (3)
  },
  {
    "_id": 3,
    "nombre": "Doble",
    "capacidad": 2,
    "dotacion_incluida": Array (2)
  },
  {
    "_id": 4,
    "nombre": "Suite presidencial",
    "capacidad": 7,
    "_class": "uniandes.edu.co.hoteles.modelo.Tipo_habitacion"
  }
]

```

Fig. 9: Colección tipos_habitacion actualizada

Delete:

Si se quiere borrar un tipo de habitación, se debe confirmar el siguiente mensaje:



localhost:8080/hoteles/tiposHabitacion/4/delete

Fig. 10: Delete tipos_habitacion

Una vez se confirma, el tipo de habitación eliminado ya no sale en la lista:

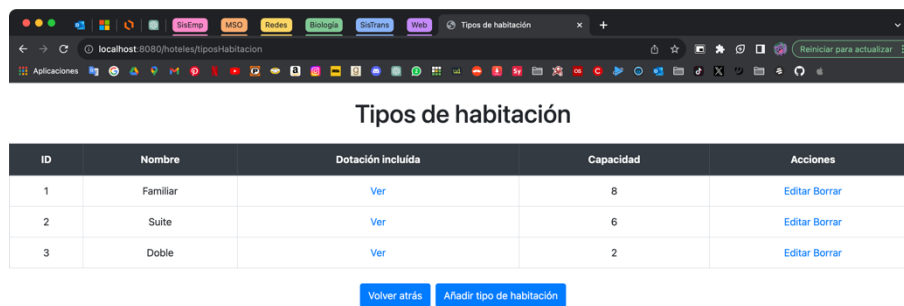


Fig. 11: Read tipos_habitacion actualizado

No solo eso, sino que la colección en la base de datos también se actualiza:

<pre> _id: 1 nombre: "Familiar" capacidad: 8 ▸ dotacion_incluida: Array (2) </pre>
<pre> _id: 2 nombre: "Suite" capacidad: 6 ▸ dotacion_incluida: Array (3) </pre>
<pre> _id: 3 nombre: "Doble" capacidad: 2 ▸ dotacion_incluida: Array (2) </pre>

Fig. 12: Colección tipos_habitacion actualizado

Nota: el CRUD de dotación_incluida y el resto de las entidades funciona igual.

2. Inserción incorrecta a entidades:

Para probar la inserción incorrecta a las colecciones de las entidades, primero fue necesario crear las validaciones para cada una, las cuales pueden ser vistas en docs/archivos-json/validaciones.

Colección habitaciones:

Por parte de la colección habitaciones, se sabe que el campo tipo debe ser un entero, pues referencia el id de la colección tipos_habitacion. Por eso, el siguiente script muestra un error:

```

> db.habitaciones.insertOne({_id: 401, "costo_noche": 500000, tipo: "Familiar"})
✖ ▶ MongoServerError: Document failed validation

```

Fig. 13: Inserción inválida a habitaciones

Por parte del arreglo consumos en la colección habitaciones, se sabe que el campo fecha es un string, a pesar de representar, como indica su nombre, una fecha. Por eso, el siguiente script muestra un error:

```

> db.habitaciones.insertOne({_id: 401, "costo_noche": 500000, tipo: 1,
  consumos: [{_id: 6, servicio: 1, cliente: "1006", fecha: new Date(), hora: "10:00"}]});
✖ ▶ MongoServerError: Document failed validation

```

Fig. 14: Inserción inválida a habitaciones.consumos

Por parte del arreglo reservas en la colección habitaciones, se sabe que el campo estado es una enumeración que indica que el valor debe ser “Por entrar”, “Entro”, “Salio con deuda” y “Salio sin deuda”. Por eso, el siguiente script muestra un error:

```
> db.habitaciones.insertOne({_id: 401, "costo_noche": 500000, tipo: 1,
  reservas: [{codigo: 9, fecha_entrada: "2023-12-25", fecha_salida: "2023-12-31", estado: "Ninguno", num_huespedes: 3}],
  cliente: {num_documento: "1006", tipo_documento: "CC", nombre: "Laura", correo: "l.restrepop@uniandes.edu.co"}}});
MongoServerError: Document failed validation
```

Fig. 15: Inserción inválida a habitaciones.reservas

Por parte del objeto cliente en el arreglo reservas en la colección habitación, se sabe que el campo num_documento es obligatorio. Por eso, el siguiente script muestra un error:

```
> db.habitaciones.insertOne({_id: 401, "costo_noche": 500000, tipo: 1,
  reservas: [{codigo: 9, fecha_entrada: "2023-12-25", fecha_salida: "2023-12-31", estado: "Por entrar", num_huespedes: 3}],
  cliente: {tipo_documento: "CC", nombre: "Laura", correo: "l.restrepop@uniandes.edu.co"}}});
MongoServerError: Document failed validation
```

Fig. 16: Inserción inválida a habitaciones.reservas.cliente

Colección tipos_habitacion:

Por parte de la colección tipos_habitacion, se sabe que el campo capacidad debe ser un entero. Por eso, el siguiente script muestra un error:

```
> db.tipos_habitacion.insertOne({_id: 4, nombre: "Suite doble", capacidad: 7.5});
MongoServerError: Document failed validation
```

Fig. 17: Inserción inválida a tipos_habitacion

Por parte del arreglo dotacion_incluida en la colección tipos_habitacion, se sabe que el campo cantidad es obligatorio. Por eso, el siguiente script muestra un error:

```
> db.tipos_habitacion.insertOne({_id: 4, nombre: "Suite doble", capacidad: 7,
  dotacion_incluida: [{nombre: "Cama"}]});
MongoServerError: Document failed validation
```

Fig. 18: Inserción inválida a tipos_habitacion.dotacion_incluida

Colección servicios:

Por último, por parte de la colección servicios, se sabe que el campo tipo_cobro es una enumeración que indica que el valor debe ser “Habitacion”, “Por día”, “Alojamiento” y “Gratuito”. Por eso, el siguiente script muestra un error:

```
> db.servicios.insertOne({_id: 16, nombre: "Servicio", tipo_cobro: "Ninguno", precio: 10000, capacidad: 5});
MongoServerError: Document failed validation
```

Fig. 19: Inserción inválida a servicios

Por parte del arreglo productos en la colección tipos_habitacion, se sabe que el campo precio es obligatorio. Por eso, el siguiente script muestra un error:


```
> db.servicios.insertOne({_id: 16, nombre: "Servicio", tipo_cobro: "Por día", precio: 10000, capacidad: 5, productos: [{nombre: "Pepsi 250ml"}]});
MongoServerError: Document failed validation
```

Fig. 20: Inserción inválida a servicios.productos

Nota: todos estos scripts pueden ser encontrados en docs/archivos-json/escenarios-prueba.

3. Requerimientos funcionales de consulta:

Para probar los requerimientos funcionales de consulta, primero fue necesario poblar las colecciones. Los scripts de población pueden ser vistos en docs/archivos-json/poblaciones. Por otro lado, para ahorrar espacio en este documento, los scripts también incluyen los resultados esperados debajo del texto “Esto debería devolver” al final de cada consulta. Siendo así, a continuación solo se encontrará la interpretación de los resultados.

RFC1:

Este resultado puede ser comprobado al ver la colección consumos, ver qué servicios fueron consumidos desde el 1ro de enero de 2023, ver su costo y sumarlo por habitación. La base de datos en este momento cuenta con 5 consumos, de los cuales 4 han sido consumidos este año.

Para la habitación 101 solo hubo un consumo con costo de \$1.000, que es exactamente el valor que aparece en el resultado. Para la habitación 202 hubo 2 consumos con costo de \$50.000 cada uno, dando \$100.000 en total, que es exactamente el valor que aparece en el resultado. Para la habitación 301 hubo un consumo, pero de un servicio si ningún costo, por lo que en el resultado el valor para esta habitación es 0. El resto de las habitaciones tuvieron consumos antes a este año o no tuvieron consumos en lo absoluto, por lo que su valor es 0.

RFC2:

Este resultado puede ser comprobado al contar la cantidad de días entre la fecha de entrada y la fecha de salida de cada reserva y sumarlo por habitación, teniendo en cuenta los casos en que ambas fechas están entre el último corrido o alguno o ambos valores se salen del rango. Luego, hay que dividir este valor por la cantidad de días que han pasado en el año actual y multiplicarlo por 100.

Teniendo en cuenta que la fecha de hoy es 1ro de diciembre de 2023 (334 días), las únicas habitaciones que están ocupadas dentro del año corrido son la 101, la 102, la 202 y la 303. Para la habitación 101 solo hubo una reserva que lleva 12 días, por lo que su porcentaje es $(12/334)*100 = 3.59\%$. Para la habitación 102 solo hubo una reserva, pero esta duró todo el año corrido, por lo que su valor es 100%. La habitación 202 tiene una reserva con los mismos días que la habitación 101, por lo que el valor es el mismo. Para la habitación 303 hubo dos reservas, una de 12 días y la otra de 7, por lo que su porcentaje es $(19/334)*100 = 5.69\%$. El resto de las habitaciones no están ocupadas dentro del año corrido, por lo que su valor es 0.

RFC3:

Este resultado puede ser comprobado al ver la colección consumos y buscar los consumos con el cliente indicado dentro de las fechas indicadas.

El cliente con número de documento 1007 tuvo 3 consumos, 2 de los cuales ocurrieron entre el 1ro de enero de 2023 y el 1ro de diciembre de 2023. Siendo así, la consulta devuelve correctamente los consumos que cumplen con las condiciones.

RFC4:

Este resultado puede ser comprobado al ver la colección consumos y buscar quién ha consumido el servicio de Gimnasio, que tiene id 1, 2 veces.

El cliente con número de documento 1007 es el único que ha consumido el servicio de Gimnasio, y lo ha hecho 3 veces. De estas 3 veces, 2 ocurrieron el mismo día, por lo que la consulta retorna correctamente su información. Esta información puede ser verificada buscando una reserva con el cliente 1007 y dando click en este valor.

Nota: los resultados están agrupados por fecha, como en la entrega anterior del proyecto.