---

1. Resumen del Sistema

Se construirá un sistema para gestionar hoteles y sus habitaciones, cumpliendo con validaciones específicas de acomodación según el tipo de habitación, respetando las restricciones funcionales y técnicas establecidas.

---

2. Arquitectura General

- **Backend:** PHP (Laravel)
- **Frontend:** React (SPA desacoplada del backend)
- **Base de Datos:** PostgreSQL
- **API:** RESTful
- **Patrones**

1. Patrón MVC (Modelo – Vista – Controlador) – *Laravel Backend*

Justificación:
Laravel está basado en el patrón **MVC**, lo que permite una separación clara de responsabilidades:

- **Modelo (Model)**: Representa la lógica de datos (Habitacion, Hotel, etc.).
- **Vista (View)**: No aplica directamente en APIs, pero los datos se devuelven en formato JSON.
- **Controlador (Controller)**: Orquesta la lógica del negocio y responde a las peticiones HTTP.

Ventajas:

- Facilita mantenimiento y pruebas.
- Permite escalabilidad.
- Organiza claramente el código.

2. Patrón Repository/Service (en backend extendido) – *Opcional pero recomendado en Laravel*

Justificación:
Si el proyecto crece, se puede separar la lógica de negocio del controlador usando **Services o Repositories**.

- El **Service** encapsula reglas de negocio (ej. validar combinaciones válidas).
- El **Repository** se encarga de consultas complejas a la base de datos.

Ventajas:

- Reutilización de código.
- Testeabilidad aislada.
- Limpieza en los controladores.

3. Patrón Custom Hooks (React.js Frontend)

Justificación:
Se encapsuló la lógica relacionada a tipos de habitaciones, acomodaciones, y habitaciones en un solo **Hook personalizado** (useHoteles), siguiendo el principio de reutilización y separación de lógica.

Ventajas:

- Reutilización fácil en múltiples componentes.
- Código más limpio y modular.
- Separa lógica del UI.

4. Patrón Service Layer (React.js)

Justificación:
El archivo habitacionesService.js y hatelesService.js actúam como una **capa de servicios**, centralizando todas las peticiones HTTP (API calls). Esto desacopla la lógica de red del componente.

Ventajas:

- Cambios en la API no afectan directamente el componente.
- Mayor organización.
- Mejora pruebas y mocking.

---

5. Patrón Presentational vs Container Components (React)

Justificación:
Componentes como <Habitaciones /> se comportan como componentes de presentación (solo muestran datos), mientras que el hook y el componente que lo consume se comportan como **container components** que manejan lógica.

Ventajas:

- Separación entre lógica y presentación.
- Facilita pruebas unitarias.
- Componentes más reutilizables.

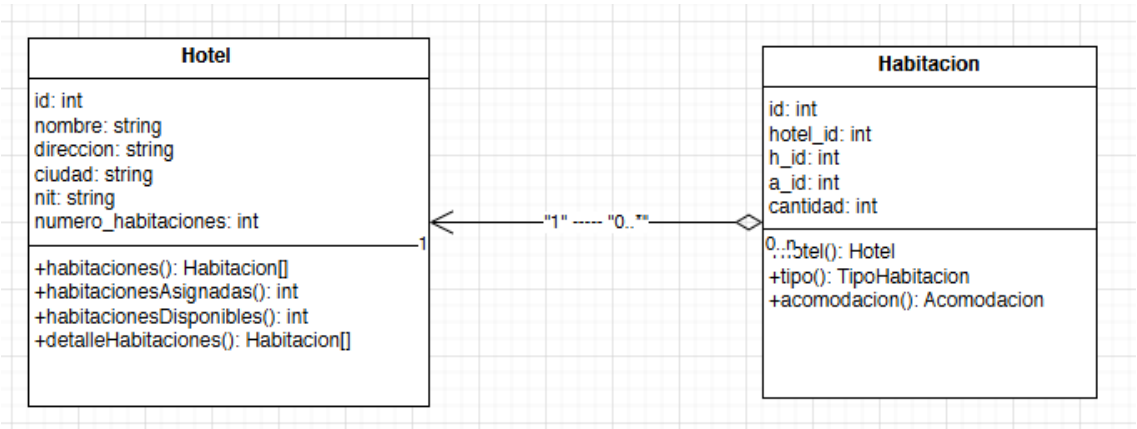
CONCLUSIÓN

Este proyecto usa una combinación de patrones:

Capa	Patrón clave aplicado	Justificación principal
Backend	MVC + Service Layer (opcional)	Organización, escalabilidad y separación de lógica.
Frontend	Custom Hooks + Service Layer + Presentational	Reutilización, separación de lógica y claridad.
Base de datos	Relacional normalizada + FK bien definidas	Integridad referencial y consultas eficientes.

---

3. Modelo de Datos (UML Simplificado)

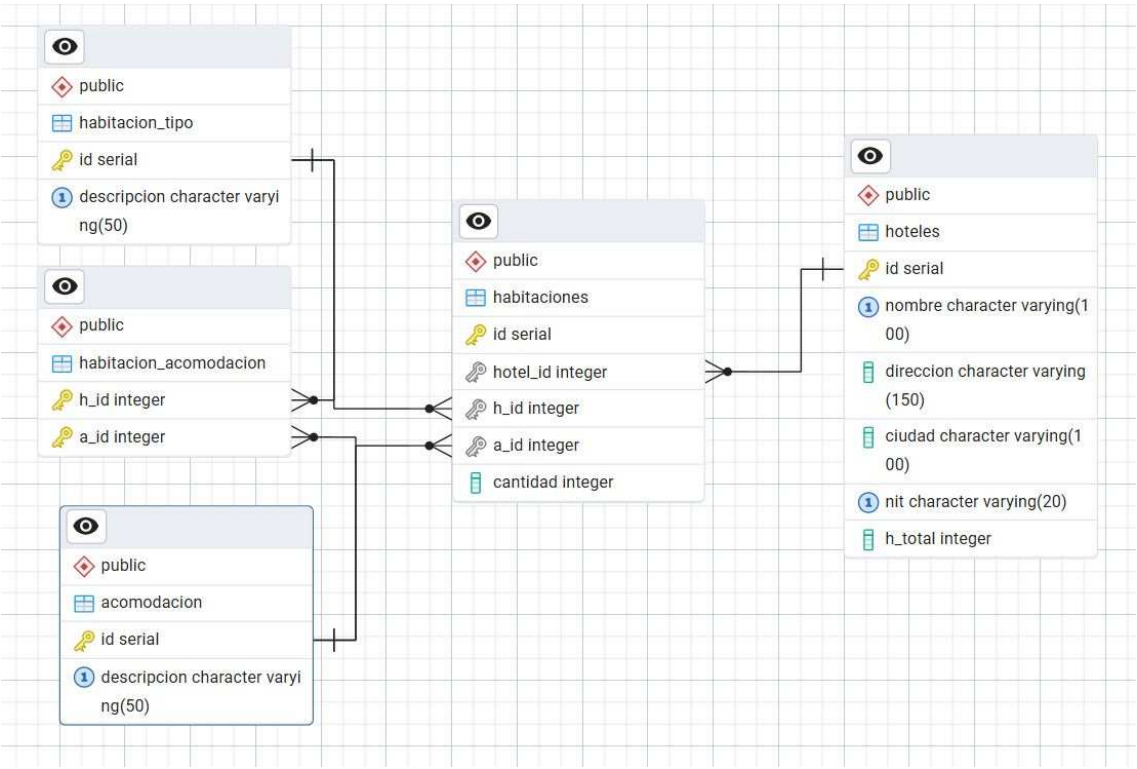


Restricciones:

- La suma de habitaciones por hotel no debe exceder numero_habitaciones
- Validación cruzada tipo ↔ acomodación según tabla:
  - Estándar: Sencilla, Doble
  - Junior: Triple, Cuádruple
  - Suite: Sencilla, Doble, Triple

---

4. Modelo ER



4. Estructura de API RESTful (Ejemplos)

Hoteles

- POST /api/hoteles → Crear hotel
- GET /api/hoteles → Listar hoteles
- GET /api/hoteles/{id} → Ver detalles de un hotel
- DELETE /api/hoteles/{id} → Eliminar hotel
- GET /api/hoteles/habitaciones/detalle → Detalle de habitaciones por hotel

Habitaciones

- POST /api/habitaciones → Agregar tipo de habitación y acomodación
- GET /api/hoteles/{id} /habitaciones/total→ Habitaciones asiganas por hotel
- DELETE /api/ habitaciones /{id} → Eliminar hotel

API RESTful Test
// el back esta publicado en https://itbf-backend.onrender.com/api
1. GET /hoteles (Listar todos los hoteles)
Método: GET

URL: **https://itbf-backend.onrender.com/api/**hoteles/

Qué esperar: Un JSON con todos los hoteles


```
{
  "status": "success",
  "codigo": 200,
  "error": null,
  "mensaje": "Listado de hoteles obtenido correctamente",
  "datos": [
    {
      "id": 6,
      "nombre": "DECAMERON MEDELLIN",
      "direccion": "CALLE 23 58-25",
      "ciudad": "Medellín",
      "nit": "56765432-1",
      "h_total": 10
    },
    {
      "id": 3,
      "nombre": "DECAMERON SANTA MARTA",
      "direccion": "CARRERA 23 58-25",
      "ciudad": "SANTA MARTA",
      "nit": "54675678-9",
      "h_total": 25
    }
  ]
}
```


2. GET /hoteles (Datos de un hotel)
Método: GET

URL: **https://itbf-backend.onrender.com/api/**hoteles/{id}

Qué esperar: Un JSON un hotel


```
{
  "status": "success",
  "codigo": 200,
  "error": null,
  "mensaje": "Hotel encontrado correctamente",
  "datos": {
    "id": 6,
    "nombre": "DECAMERON MEDELLIN",
    "direccion": "CALLE 23 58-25",
    "ciudad": "Medellín",
    "nit": "56765432-1",
    "h_total": 10
  }
```

```
}
```

3. GET /hoteles (Datos vacíos por ejemplo un id que no exista)
Método: GET

URL: **https://itbf-backend.onrender.com/api/**hoteles/{id}

Qué esperar: Un JSON con el menaje de error

```
{
  "status": "error",
  "codigo": 404,
  "error": "Hotel no encontrado",
  "mensaje": "No se encontró un hotel con el ID proporcionado",
  "datos": null
}
```

4. POST /hoteles (crear un hotel)
Método: POST
Body

```
  {
     "nombre": "SANTA HELENA OCAÑA",
     "direccion": "CALLE 23 58-25",
     "ciudad": "OCAÑA",
     "nit": "1223445540-9",
     "h_total": 40
   }
```

URL: **https://itbf-backend.onrender.com/api/**hoteles/

Qué esperar: Un JSON la confirmación de la creación o el error en caso que los datos no cumplan con la validación

```
{
  "status": "success",
  "codigo": 201,
  "error": null,
  "mensaje": "Hotel creado correctamente",
  "datos": {
    "nombre": "SANTA HELENA OCAÑA",
    "direccion": "CALLE 23 58-25",
    "ciudad": "OCAÑA",
    "nit": "1223445540-9",
    "h_total": 40,
    "id": 10
  }
}
```

5. PUT /hoteles (Actualizar un hotel)
Método: PUT
Body

```
{
   "nombre": "SANTA HELENA OCAÑA",
   "direccion": "CALLE 23 58-25",
   "ciudad": "OCAÑA",
   "nit": "1223445540-9",
   "h_total": 30
 }
```

URL: **https://itbf-backend.onrender.com/api/**hoteles/{id}

Qué esperar: Un JSON la confirmación de la actualizacion o el error en caso que los datos no cumplan con la validación

```
{
 "status": "success",
 "codigo": 200,
 "error": null,
 "mensaje": "Hotel actualizado correctamente",
 "datos": {
  "id": 10,
  "nombre": "SANTA HELENA OCAÑA",
  "direccion": "CALLE 23 58-25",
  "ciudad": "OCAÑA",
  "nit": "1223445540-9",
  "h_total": 30,
  "habitaciones": []
 }
}
```

6. DELETE /hoteles (Borrar)
Método: DELETE

URL: **https://itbf-backend.onrender.com/api/**hoteles/

Qué esperar: Un JSON la confirmación del borrado o el error en caso que los datos no cumplan con la validación

```
{
 "status": "success",
 "codigo": 200,
 "error": null,
 "mensaje": "Hotel eliminado satisfactoriamente",
 "datos": null
}
```

7. GET /hoteles (Listar todos los hoteles con el detalle de su habitaciones)
Método: GET

URL: **https://itbf-backend.onrender.com/api/**hoteles/habitaciones/detalle

Qué esperar: Un JSON con todos los hoteles con el detalle completo de habitaciones y acomodación


```
{
  "status": "success",
  "codigo": 200,
  "error": null,
  "mensaje": "Listado de hoteles con detalle de habitaciones",
  "datos": [
    {
      "id": 6,
      "nombre": "DECAMERON MEDELLIN",
      "ciudad": "Medellín",
      "direccion": "CALLE 23 58-25",
      "nit": "56765432-1",
      "h_total": 10,
      "habitaciones_asignadas": 4,
      "habitaciones_disponibles": 6,
      "detalle_habitaciones": [
        {
          "id": 20,
          "tipo_habitacion": "ESTANDAR",
          "acomodacion": "SENCILLA",
          "cantidad": 4
        }
      ]
    }, ....

      ]
    }
  ]
}
```


8. POST /Habitaciones (Agregar una distribución por habitación)
Método: POST
Body


```
{
  "hotel_id": 6,
  "h_id": 3,
  "a_id": 2,
  "cantidad": 1
}
```


URL: **https://itbf-backend.onrender.com/api/**habitaciones/

Qué esperar: Un JSON con la confirmación o el erro en caso de la validación no se pueda crear

success


```
{
  "status": "success",
  "codigo": 201,
  "error": null,
  "mensaje": "Habitación creada correctamente",
  "datos": {
    "hotel_id": 6,
    "h_id": 3,
    "a_id": 2,
    "cantidad": 1,
    "id": 24
  }
}
```


error


```
{
  "status": "error",
  "codigo": 409,
  "error": {
    "habitacion": [
      "Ya existe una habitación con esta combinación de hotel, tipo y acomodación."
    ]
  },
  "mensaje": "Registro duplicado",
  "datos": null
}
```


9. DELETE /Habitaciones (Borrar una distribución de habitación)
Método: GET

URL: **https://itbf-backend.onrender.com/api** /habitaciones/{id}

Qué esperar: Un JSON con la confirmación o el erro en caso de la validación no se pueda borrar

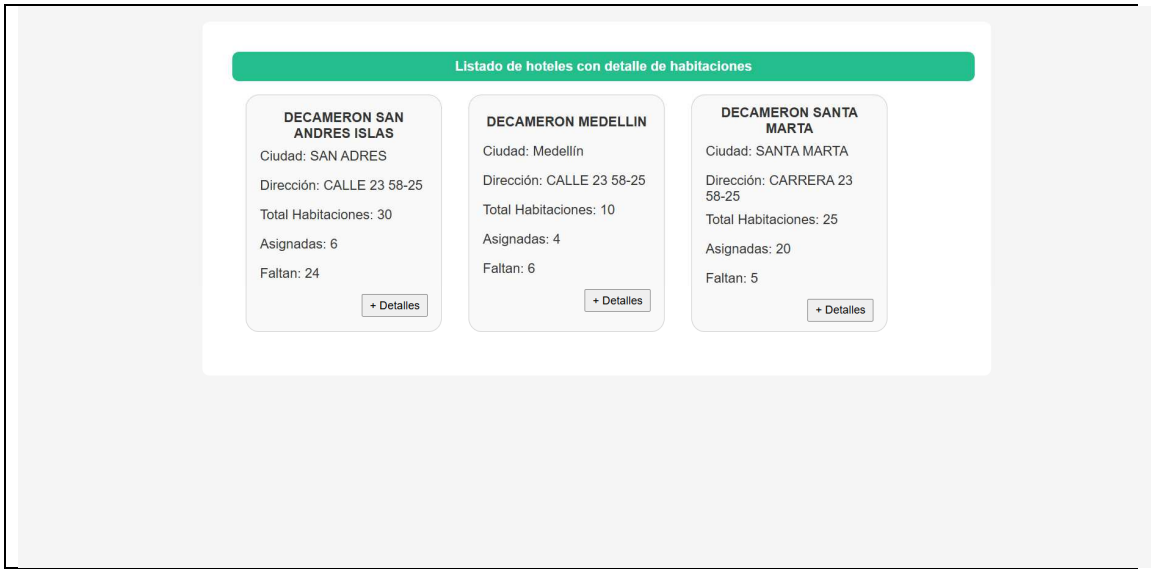

```
{
  "status": "success",
  "codigo": 200,
  "error": null,
  "mensaje": "Habitación eliminada satisfactoriamente",
  "datos": null
}
```

5. Validaciones Clave

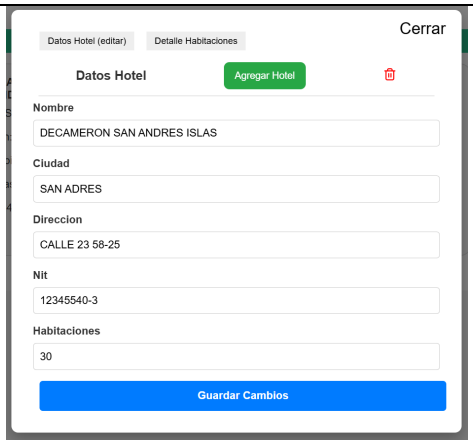
- El NIT debe ser único .
- No puede haber un hotel con el mismo nombre.
- No puede repetirse un tipo + acomodación en el mismo hotel.
- La suma de habitaciones no debe superar el total definido para el hotel.
- Validación de tipo de habitación vs acomodación según reglas.

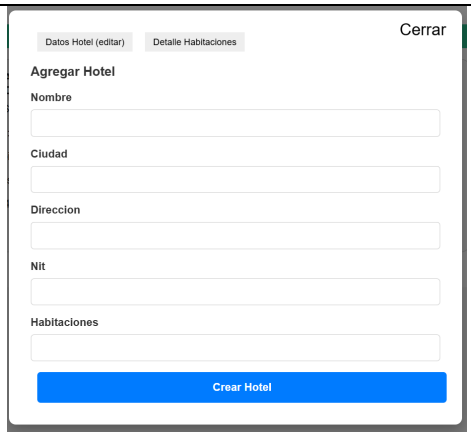
6. Frontend React.js
// el Front esta publicado en https://itbf-front.onrender.com/

- UI responsive
- Validación en cliente y servidor
- Tablas dinámicas para ver habitaciones por hotel
- Formularios con mensajes de error descriptivos



El manejo de la app, se presenta de forma sencilla, e intuitiva, por estar orientada a administradores de información, busca una experiencia de usuario minimalista, página de inicio es un dashboard que muestra en la parte superior el mensaje de respuesta de la api y organiza los hoteles en cards con la información básica, a medida que se crean organiza la card adaptándose a cualquier pantalla





El botón de **+Detalles** despliega un modal, en la parte superior navegamos entre **Datos Hotel (Editar)** donde podemos agregar un hotel, borrarlo, o editar los datos,

Y sección **Detalle habitación** donde vemos la disponibilidad y distribución de las habitaciones, acá podemos borrar y agregar una habitación, para agregar una habitación se selecciona el tipo de habitación y la acomodación presenta solo las posibilidades según el modelo de negocio planteado.

En ambos casos para eliminar, presenta un alerta para confirmar o no

Se planteó una validación desde el backend, para todas la restricciones del modelo de negocio, desde el dashboard se carga los datos, y de allí al modal, por eso todas trazabilidad de datos parte del dashboard, para el caso de las habitaciones no se creó la edición, al cargar el modal el usuario tiene un panorama de cómo esta las distribuciones y acomodaciones, en caso de error la borra y agrega de forma más fácil y como ya lo mencione la validación de la acomodación según el tipo de habitación

No permite agregar más habitaciones que las disponibles y en caso de editar el hotel no permite menos que las ya asignadas.