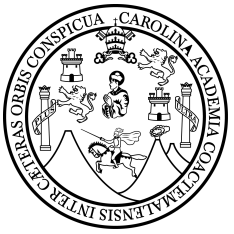
Teoría de Sistemas 1
Ingeniería en Ciencias y Sistemas
División de Ciencias de la Ingeniería
Centro Universitario de Occidente
Universidad de San Carlos de Guatemala



SISTEMA HOTELERÍA

Enlace Repositorio

https://github.com/Isaac03483/Hoteru

Manual técnico.

11/21/2,023

Terminologías:

Aplicación Cliente Servidor:

Una aplicación cliente servidor es un tipo de aplicación informática que se basa en el modelo de comunicación cliente-servidor. En este modelo, las tareas se reparten entre dos tipos de programas:

- **Los clientes:** Son los programas que realizan peticiones a los servidores.
- **Los servidores:** Son los programas que proporcionan recursos o servicios a los clientes.

La comunicación entre clientes y servidores se realiza a través de una red, generalmente Internet.

Un ejemplo de aplicación cliente servidor es un navegador web. El navegador web es el cliente que realiza peticiones al servidor web para obtener las páginas web. El servidor web es el servidor que proporciona las páginas web a los clientes.

Otro ejemplo de aplicación cliente servidor es un correo electrónico. El cliente de correo electrónico es el cliente que envía y recibe correos electrónicos. El servidor de correo electrónico es el servidor que almacena los correos electrónicos y los entrega a los clientes.

Las aplicaciones cliente servidor tienen una serie de ventajas, entre las que se incluyen:

- **Eficiencia:** La carga de trabajo se reparte entre los clientes y los servidores, lo que mejora la eficiencia del sistema.
- **Seguridad:** Los datos se pueden almacenar de forma centralizada en el servidor, lo que facilita su protección.
- **Escalabilidad:** Las aplicaciones cliente servidor se pueden escalar fácilmente para adaptarse a un mayor número de usuarios.

Las aplicaciones cliente servidor son muy comunes en Internet. La mayoría de las aplicaciones web, como las redes sociales, las tiendas online y los servicios de streaming, se basan en el modelo cliente-servidor.

Patrón Modelo Vista Controlador:

El patrón MVC (Modelo Vista Controlador) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos:

- **Modelo:** Es el responsable de gestionar los datos de la aplicación.
- **Vista:** Es la encargada de mostrar la información a los usuarios.
- **Controlador:** Es el responsable de recibir las peticiones de los usuarios y coordinar la interacción entre el modelo y la vista.

El patrón MVC se utiliza comúnmente en aplicaciones cliente servidor, ya que permite separar las responsabilidades de la aplicación entre el cliente y el servidor.

En el lado del cliente, el patrón MVC se utiliza para representar la interfaz de usuario. La vista se encarga de mostrar la información al usuario, y el controlador se encarga de recibir las peticiones del usuario y actualizar la vista.

En el lado del servidor, el patrón MVC se utiliza para gestionar los datos de la aplicación. El modelo se encarga de acceder a los datos almacenados en el servidor, y el controlador se encarga de procesar las peticiones del cliente y actualizar el modelo.

API REST:

Una API REST es una interfaz de programación de aplicaciones (API) que se basa en el estilo de arquitectura de transferencia de estado representacional (REST). REST es un conjunto de principios arquitectónicos que definen cómo se deben diseñar y utilizar los servicios web.

Una API REST se utiliza para exponer los recursos de un sistema a otros sistemas o aplicaciones. Estos recursos pueden ser datos, funciones o servicios.

Las API REST se comunican a través del protocolo HTTP. Los verbos HTTP se utilizan para definir las operaciones que se pueden realizar sobre los recursos. Los verbos HTTP más utilizados son:

- **GET:** Obtener un recurso.
- **POST:** Crear un recurso.
- **PUT:** Actualizar un recurso.
- **DELETE:** Eliminar un recurso.

Servidor:

Modelos:

La aplicación del lado del servidor cuenta con los siguientes modelos:

1. **EmployeeType** Este modelo hace referencia a los tipos de empleados que se pueden agregar al sistema. Los campos con los que cuenta son:
    - **Id:** Tipo de dato numérico, llave primaria y auto incrementable.
    - **type:** Tipo de dato varchar, hace referencia al nombre del tipo.
2. **RoomType:** Este modelo hace referencia a los tipos de habitaciones que se pueden agregar al sistema. Los campos con los que cuenta son:
    - **Id:** Tipo de dato numérico, llave primaria y auto incrementable.
    - **Type**: Tipo de dato varchar, hace referencia al nombre del tipo.
    - **CostPerDay:** Tipo de dato decimal, hace referencia al precio por día.
3. **Employee:** Este modelo hace referencia a todos los empleados registrados en el sistema. Los campos con los que cuenta son:

    - **Id:** Tipo de dato numérico, llave primaria y auto incrementable.
    - **Username:** Tipo de dato varchar, hace referencia al nombre de usuario con el que ingresará al sistema. Este campo es único.

- **Name:** Tipo de dato varchar, hace referencia al nombre del empleado.
- **Password:** Tipo de dato varchar, hace referencia a la contraseña con la que el usuario ingresará al sistema.
- **EmployeeTypeId:** Tipo de dato numérico, hace referencia a la llave primaria de la tabla EmployeeType.

4. **Room:** Este modelo hace referencia a todas las habitaciones registradas en el sistema. Los campos con los que cuenta son:
    - **Id:** Tipo de dato numérico, llave primaria y auto incrementable.
    - **RoomTypeId:** Tipo de dato numérico, hace referencia a la llave primaria de la tabla RoomType.
    - **State:** Tipo de dato varchar, nos dice el estado en el que se encuentra la habitación (disponible u ocupada).
5. **Task:** Este modelo hace referencia a todas las tareas registradas en el sistema. Los campos con los que cuenta son:
    - **Id:** Tipo de dato numérico, llave primaria y auto incrementable.
    - **Name:** Tipo de dato varchar, hace referencia al nombre de la tarea.
    - **Description:** Tipo de dato varchar, hace referencia a la descripción de la tarea. Esto para ser más detallados.
    - **Date:** Tipo de dato date. Hace referencia a la fecha en la que se registró la tarea.
    - **EmployeeId:** Tipo de dato numérico, hace referencia a la llave primaria del empleado que se “asignó” la tarea.
    - **State:** Tipo de dato varchar, hace referencia al estado en el que se encuentra la tarea (pendiente, asignada o finalizada).
6. **Client:** Este modelo hace referencia a todos los clientes registrados en el sistema. Los campos con los que cuenta son:
    - **Nit:** Tipo de dato varchar, hace referencia al nit del cliente. Es la llave primaria.
    - **Name:** Tipo de dato varchar, hace referencia al nombre del cliente.
7. **Reservation:** Este modelo hace referencia a todas las reservaciones registradas en el sistema. Los campos con los que cuenta son:
    - **Id:** Tipo de dato numérico, llave primaria y auto incrementable.
    - **Nit:** Tipo de dato varchar, hace referencia a la llave primaria de la tabla Client.
    - **RoomId:** Tipo de dato numérico, hace referencia a la llave primaria de la tabla Room.
    - **Date:** Tipo de dato date, hace referencia a la fecha en la que se registró la reservación.
    - **InitDate:** Tipo de dato date, hace referencia a la fecha de inicio de la reservación:
    - **EndDate:** Tipo de dato date, hace referencia a la fecha de finalización de la reservación.
    - **Total:** Tipo de dato numérico, hace referencia al total a pagar por la reservación.

Rutas:

Rutas para la autenticación:

1. **auth/{username}/{password}:** Esta petición es de tipo GET llamando al método del controlador AuthController: show.

Rutas para los empleados:

1. **employees:** Esta petición es de tipo GET llamando al método del controlador EmployeeController: index.
2. **employees/{id}:** Esta petición es de tipo GET llamando al método del controlador EmployeeController: show.
3. **employees:** Esta petición es de tipo POST llamando al método del controlador EmployeeController: store.
4. **employees:** Esta petición es de tipo PUT llamando al método del controlador EmployeeController:update.
5. **employees/{id}:** Esta petición es de tipo DELETE llamando al método del controlador EmployeeController: destroy.

Rutas para los tipos de empleados:

1. **employee-types:** Esta petición es de tipo GET llamando al método del controlador EmployeeTypeController: index.
2. **employee-types/{id}:** Esta petición es de tipo GET llamando al método del controlador EmployeeTypeController: show.
3. **employee-types:** Esta petición es de tipo POST llamando al método del controlador EmployeeTypeController: store.
4. **employee-types/{id}:** Esta petición es de tipo DELETE llamando al método del controlador EmployeeTypeController: destroy.

Rutas para los tipos de habitaciones:

1. **room-types:** Esta petición es de tipo GET llamando al método del controlador RoomTypeController: index.
2. **room-types:** Esta petición es de tipo POST llamando al método del controlador RoomTypeController: store.
3. **room-types/{id}:** Esta petición es de tipo DELETE llamando al método del controlador RoomTypeController: destroy.

Rutas para las habitaciones:

1. **rooms:** Esta petición es de tipo GET llamando al método del controlador RoomController: index.
2. **rooms:** Esta petición es de tipo POST llamando al método del controlador RoomController: store.

3. **rooms/{id}:** Esta petición es de tipo DELETE llamando al método del controlador RoomController: destroy.

Rutas para las tareas:

1. **tasks:** Esta petición es de tipo GET llamando al método del controlador TaskController: index.
2. **tasks:** Esta petición es de tipo POST llamando al método del controlador TaskController: store.
3. **tasks:** Esta petición es de tipo PUT llamando al método del controlador TaskController: update.
4. **tasks/employees:** Esta petición es de tipo PUT llamando al método del controlador TaskController: updateTaskState.
5. **tasks/undone/{type}:** Esta petición es de tipo GET llamando al método del controlador TaskController: findEmployeeTypeTasks.
6. **tasks/my-tasks/{id}:** Esta petición es de tipo GET llamando al método del controlador TaskController: findEmployeeTasks.
7. **tasks/{id}:** Esta petición es de tipo DELETE llamando al método del controlador TaskController: destroy.

Rutas para los clientes:

1. **clients/find/{nit}:** Esta petición es de tipo GET llamando al método del controlador ClientController: show.

Rutas para las reservaciones:

1. **reservations:** Esta petición es de tipo GET llamando al método del controlador ReservationController: index.
2. **reservations:** Esta petición es de tipo POST llamando al método del controlador ReservationController: store.
3. **reservations/today:** Esta petición es de tipo GET llamando al método del controlador ReservationController: todayReservations.

Rutas para los reportes:

1. **reports/today-tasks:** Esta petición es de tipo GET llamando al método del controlador ReportController: countTodayTasks.
2. **reports/earnings/{init}/{end}:** Esta petición es de tipo GET llamando al método del controlador ReportController: earnings.
3. **reports/best-room-types:** Esta petición es de tipo GET llamando al método del controlador ReportController: bestRoomTypes.
4. **reports/best-clients:** Esta petición es de tipo GET llamando al método del controlador ReportController: bestClients.

Controladores:

1. **AuthController:** En este controlador el único método implementado es:
    - **show:** Este método es el encargado de retornar un empleado siempre y cuando las credenciales sean las correctas.
2. **ClientController:** En este controlador el único método implementado es:
    - **show:** Este método retorna el cliente en base a un parámetro recibido que es: **nit** de tipo string.
3. **EmployeeController:** Este controlador posee los siguientes métodos:
    - **index:** Este método devuelve todos los empleados registrados en el sistema.
    - **store:** Este método almacena un nuevo empleado al sistema.
    - **show:** Este método muestra a un empleado en particular en base a un parámetro: **id** de tipo numérico.
    - **update:** Este método actualiza la información de un empleado previamente registrado al sistema.
    - **destroy:** Este método elimina permanentemente a un empleado del sistema.
4. **EmployeeTypeController:** Este controlador posee los siguientes métodos:
    - **index:** Este método devuelve todos los tipos de empleados registrados en el sistema.
    - **store:** Este método almacena un nuevo tipo de empleado al sistema.
    - **show:** Este método muestra un tipo de empleado en particular en base a un parámetro: **id**.
    - **destroy:** Este método elimina permanentemente a un tipo de empleado del sistema.
5. **ReportController:** Este controlador posee los siguientes métodos:
    - **countTodayTasks:** Este método retorna el número de tareas completadas, pendientes y asignadas durante el día.
    - **earnings:** Este método retorna las ganancias en un intervalo de tiempo. Los parámetros son el inicio y el fin del intervalo.
    - **bestRoomTypes:** Este método retorna los tipos de habitaciones más solicitados por los clientes.
    - **bestClients:** Este método retorna a los clientes que han realizado más reservaciones en el sistema. Además devuelve el total gastado en el hotel.
6. **ReservationController:** Este controlador posee los siguientes métodos:
    - **index:** Este método devuelve todas las reservaciones registradas en el sistema.
    - **store:** Este método almacena una nueva reservación en el sistema. Verificando que la habitación solicitada se encuentre disponible en esas fechas.
    - **todayReservations:** Este método devuelve todas las reservaciones registradas que inician en la fecha actual.
7. **RoomController:** Este controlador posee los siguientes métodos:
    - **index:** Este método devuelve todas las habitaciones registradas en el sistema.
    - **store:** Este método almacena una nueva habitación en el sistema.
    - **destroy:** Este método elimina permanentemente una habitación del sistema.

    - **updateRoomState:** Este método actualiza el estado de una habitación en base a la fecha actual.
8. **RoomTypeController:** Este controlador posee los siguientes métodos:
    - **index:** Este método devuelve todos los tipos de habitaciones registradas en el sistema.
    - **store:** Este método almacena un nuevo tipo de habitación en el sistema.
    - **destroy:** Este método elimina permanentemente un tipo de habitación del sistema.
9. **TaskController:** Este controlador posee los siguientes métodos:
    - **index:** Este método devuelve todas las tareas registradas en el sistema.
    - **store:** Este método almacena una nueva tarea en el sistema.
    - **update:** Este método actualiza la información de una tarea previamente ingresada al sistema.
    - **updateTaskState:** Este método actualiza el estado de una tarea.
    - **findEmployeeTypeTasks:** Este método devuelve todas las tareas en base a un tipo de empleado. Recibe como parámetro el id del tipo de empleado.
    - **findEmployeeTasks:** Este método devuelve todas las tareas asignadas de un empleado. Recibe como parámetro el id del empleado.
    - **destroy:** Este método elimina permanentemente una tarea del sistema.

Cliente:

Módulos:

La aplicación cuenta con los siguientes módulos:

- **admin:** Este módulo posee todas las funcionalidades del administrador.
- **auth:** Este módulo corresponde a la autenticación de los usuarios en la aplicación.
- **employee:** Este módulo posee todas las funcionalidades de los empleados en general.
- **recep:** Este módulo posee todas las funcionalidades de los recepcionistas.

Servicios:

La aplicación cuenta con los siguientes servicios. Tomar en cuenta que estos son los encargados de comunicar la aplicación del cliente con la aplicación del servidor.

- **auth**
- **client**
- **employee**
- **report**
- **reservation**
- **room**
- **task**