**Proyecto 1: Property Management System (PMS)**

**Jhoan Sebastian Sánchez Suarez**

**Manuela Rivas Gomez**

**Nicolás Ruíz Pérez**

1. **Contexto del problema**

Se está desarrollando un programa para el manejo de distintas funcionalidades dentro del sistema de un hotel. Para este programa se tienen 3 entes principales que podrán interactuar con el sistema que son, administrados, empleado (recepcionista o empleado regular) y huésped. Con esto en mente, cada ente puede llevar a cabo una serie de acciones y solicitudes en el sistema, siendo el huésped el ente con más restricciones de interacción y el administrador con conocimiento y control total de las diferentes funcionalidades del sistema. Es por esto por lo que el flujo y funcionamiento del sistema se puede entender con base en la figura 1. Esta figura ilustra las interacciones entre los diferentes entes con el sistema de una forma totalmente genérica, sin embargo, le permite al lector del documento entender el flujo del sistema solicitado.

Figura 1. Diagrama de flujo de funcionalidades del sistema. Este es un diagrama de flujo genérico de las interacciones entre entes que pueden interactuar con el sistema solicitado y las interacciones que pueden tener con el mismo.
Figura 1. Diagrama de flujo del funcionamiento e interacciones en el sistema propuesto. Diagramade flujo genérico de las interacciones entre entes que pueden interactuar con el sistema solicitado y las interacciones que pueden tener con el mismo.

Se proponen inicialmente tres tipos de interfaces administradoras con las cuales únicamente empleados y/o el administrador pueden interactuar, es importante recalcar que el huésped no tiene acceso directo a ninguno de estos administradores de funcionalidades, por lo cual es el ente con mayor cantidad de restricciones prácticas dentro del sistema solicitado. Cabe resaltar que el administrador es el único que tiene la potestad de interactuar con el administrador de inventario de habitaciones, ya que debe ser el único habilitado para modificar tarifas de habitaciones, ingresar nuevas habitaciones y modificar los atributos de las mismas y comparte con los empleados el acceso al administrador de servicios y reservas ya que, el administrador puede también modificar tarifas e ingresar servicios y el empleado es el encargado de ingresar el consumo de un servicio por parte de los huéspedes, de forma que interactúa también con el administrador de reservas y huéspedes indirectamente. Cabe resaltar que aunque el administrador puede interactuar con él, los empleados son los que principalmente interactúan con el administrados de reservas, puesto que son los entes encargados de registrar las reservas, salidas, entradas y cuentas de los huéspedes en el sistema (diferenciando más adelante las labores entre los empelados normales y los recepcionistas). El huésped no interactúa directamente con ningún administrador del sistema, solo puede realizarle solicitudes al empleado para poder interactuar con el mismo.

**2. Planteamiento de la solución**

**2.1. Componentes candidatos y estereotipos**

1. Al reconocer la necesidad de administrar las habitaciones y poder controlar toda la información referente a estas, es evidente la necesidad de implementar un “Controlador de habitaciones” encargado de almacenar consigo la información referente al inventario de las habitaciones, controlar las tarifas de estas y permitir la creación de nuevas habitaciones. Este administrador cumple con los estereotipos *Information Holder* y *Controller* también para no solo almacenar la información sino también poder modificar ciertas cosas e interactuar con las partes de este.

2. En respuesta a la necesidad funcional del ingreso, factura y control de servicios adicionales que puede ofrecer el hotel se requiere otro componente encargado de interactuar con estos servicios de forma que se debe implementar un “Controlador de servicios” encargado de modificar y controlar los cambios en la información correspondiente a los servicios por lo cual este adopta el estereotipo de *Controller*.

3. Por otro lado, en respuesta a la necesidad del manejo de reservas, diferente a un huésped individual optamos por implementar un “Controlador de Reservas”. Este cumple con las funciones relacionadas directamente con reservas e interactúa con el “Controlador de huéspedes” de forma que de la mano sea posible cancelar y generar reservas, facturar productos a una reserva, tomar en cuenta grupos, entre otras cosas. Es por esto por lo que este Controlador, como su nombre lo indica, adopta el estereotipo de *Controller*.

4. En cuanto a la necesidad funcional solicitada para la ejecución del programa, fue evidente que era necesario generar un “Controlador de huéspedes”. Este está encargado de mediar las funcionalidades relacionadas directamente con la interacción con los huéspedes con sus correspondientes reservas y el resto de las partes del hotel como ingresar servicios adicionales a la cuenta final de estos, entre otros. Es por esto por lo que este Controlador cumple con el estereotipo de *Controller*.

5. Por último, en relación a la necesidad funcional de poder realizar pagos en diferentes plataformas con diferentes tarjetas de crédito fue evidente la necesidad de implementar, además, un “Controlador de pagos”. Este fue generado con el fin de manipular y almacenar toda la información de las tarjetas de crédito y medios de pago para poder realizar pagos digitalmente por diferentes plataformas por lo cual este controlador cumple con el estereotipo de *Information Holder*.

**Cambios en la clase Habitación:**

Para los cambios en la clase Habitación se planea agregar atributos a la clase para contener la información adicional que se requiere. Por otro lado, también se cambiará la interfaz de usuario para que los administradores puedan suministrar toda esta información para poder crear una habitación en el sistema.

Por lo que la clase Habitación en el UML de la aplicación tendría los siguientes cambios:

Texto, Tabla, Excel

Descripción generada automáticamente

*1. Habitación antigua en el UML*

Imagen que contiene Tabla

Descripción generada automáticamente

*2 Nuevo UML de la Habitación implementando los cambios*

Y el diseño de la interfaz de usuario tendría los siguientes cambios:

Texto

Descripción generada automáticamente

*3. Antiguo diseño del requerimiento Crear Habitación*

Texto, Carta

Descripción generada automáticamente

*4 Nuevo diseño del requerimiento Crear Habitación*

**2.2 Responsabilidades y Requerimientos funcionales**

**Requerimientos funcionales administrador:**

* Cargar un archivo con la información de las habitaciones en el inventario.
* Crear una habitación en el inventario.
* Cargar tarifa de habitación por tipo de cuarto, en un rango de fechas y en ciertos días de la semana.
* Cambiarla tarifa de un servicio del hotel.
* Cargar menús del restaurante.
* Crear un producto de restaurante.
* Crear un nuevo servicio

**Requerimientos funcionales recepcionista:**

* Crear reserva a nombre de uno o varios huéspedes.
* Generar factura para una reserva.
* Consultar inventario de habitaciones.
* Cancelar una reserva.

**Requerimientos funcionales empleado:**

* Registrar el consumo de un servicio a nombre de uno o más huéspedes.

**Requerimientos funcionales generales:**

* Cargar los datos de los usuarios.
* Generar archivo de log con el historial de reservas de un grupo de huéspedes.
* Avisar si no hay una tarifa definida en los próximos 365 días.
* Permitir pagos por diferentes plataformas con tarjetas de crédito
* Almacenar información de las tarjetas de crédito

**Responsabilidades:**

|  |  |  |
| --- | --- | --- |
| **#** | **Responsabilidad** | **Componente** |
| **1** | Cargar información de habitaciones en el inventario | Controlador Habitaciones |
| **2** | Crear habitación |
| **3** | Cargar tarifa de habitación por tipo de cuarto, rango de fechas y días de semana |
| **4** | Consultar inventario de habitaciones |
| **5** | Avisar si no hay tarifa definida en los últimos 365 días |
| **6** | Crear reserva a nombre de un huésped o un grupo | Controlador Reservas |
| **7** | Generar factura para una reserva |
| **8** | Cancelar una reserva |
| **9** | Crear servicio | Controlador Servicios |
| **10** | Crear producto menú |
| **11** | Cambiar tarifa servicio o producto |
| **12** | Cargar menú y servicios |
| **13** | Registrar un consumo de un servicio por una reserva |
| **14** | Generar archivo de log con el historial de reservas de un grupo | Controlador Huéspedes |
| **15** | Cargar información de usuarios | Hotel |
| **16** | Administrar información de tarjetas de crédito | Controlador Pagos |
| **17** | Manejar plataformas para pagos con tarjetas |
| **18** | Debitar de una tarjeta un monto indicado para un pago |
| **19** | Guardar un movimiento de tarjeta |

Se tomó decisión de otorgarle a cada componente las responsabilidades planteadas previamente puesto que se separaron los requerimientos funcionales dependiendo de los entes con los que interactuaban, y de ahí, el ente que más influencia y trabajo tendría en realizar el requerimiento funcional se determinó como el encargado de la responsabilidad. Posteriormente Se mostrarán las diferentes interacciones y colaboraciones entre los componentes para llevar a cabo tareas, sin embargo, siempre hay uno que cuenta con mayor parte de la responsabilidad y de las operaciones que se deben llevar a cabo para cumplir con el requerimiento por lo que se le otorgó esa responsabilidad a este componente indicado.

**2.3 Colaboraciones**

Hay ciertas responsabilidades que requieren de la interacción entre uno o mas controladores para poder llevar a cabo su ejecución. Algunas de estas son:

1. **Registrar el consumo de un servicio a una reserva:** 
   * 1. Controlador de servicios entrega la información del servicio que se consumió
     2. Controlador de reservas le solicita esta información al controlador de servicios
     3. Controlador de reservas lo registra en la reserva indicada
2. **Crear reserva:**
   * 1. Controlador de huéspedes recibe la información de cada huésped en la reserva
     2. Controlador de habitaciones revisa disponibilidad para la habitación que desean y los registra en esta dado el caso que esté libre
     3. Controlador de reservas le solicita esta información al de huéspedes y habitaciones y los registra juntos en una reserva dado el caso que sean varias personas o genera la reserva para solo la única persona
3. **Pagar con tarjeta de crédito:** 
   * 1. Controlador de Reservas recibe la solicitud de emitir una factura para pagar una reserva
     2. Controlador de pagos recibe la solicitud de realizar un pago
     3. Formas de pago inicia el proceso de realizar un pago con la información de controlador de pagos
     4. Se guarda el pago con tarjeta en forma de pago y controlador de pagos
     5. Se ve reflejado el pago en controlador de reservas

Estos son algunos de los ejemplos de colaboraciones que existen entre los diferentes entes controladores de la aplicación.

**3. Diagramas de la aplicación**

**4. Diseño de interfaces gráficas**

**4.1. Diseño interfaz gráfica de usuario de la aplicación para administrar hoteles:**

Como es claro, la interfaz pasó de ser interacción con el usuario por consola a una interfaz gráfica. Esta interfaz se compone por un núcleo principal compuesto por tres subnúcleos grandes a partir de los cuales se ramifican el resto de las clases que contienen los objetos creados para la funcionalidad adecuada de la aplicación.

El núcleo principal se llama *IniciarInterfaz* y es el main. Al ejecutase se inicia la aplicación de tal forma que se solicita el login y este mismo determina cual es la siguiente interfaz que ejecutará. De esta forma, iniciar interfaz determina que tipo de rol es el que tiene cada usuario que intenta ingresar a la aplicación y también determina si el login existe.

Posteriormente, entran en juego los tres subnúcleos siguientes, *InterfazAdministrador*, *InterfazRecepcionista* e *InterfazEmpleado*. *IniciarInterfaz* determina cual de los tres subnúcleos ejecuta dependiendo del rol del usuario que ingrese al ejecutarse.

Para el caso de la *InterfazAdministrador*, esta se ejecuta cuando el rol del usuario es de administrador. En este caso, al ejecutarse, se podrán observar una serie de botones, cada uno encargado de llevar a cabo un requerimiento funcional diferente de los que se le atribuyeron al administrador previamente. Cada uno de estos botones cuenta con una clase asociada de forma que se logre recopilar la información necesaria por parte del usuario y se ejecute adecuadamente la lógica para realizar la función que tiene estipulada. Las clases asociadas a *InterfazAdministrador* son: *crearHabitacion*, *cargarTarifaHabitacion*, *ModificarTarifaServicio*, *crearServicio*, *crearProductoRestaurante*, *consultarTarifas*, *consultarOcupacion*, *cambiarUsuario*. Como se puede apreciar, el administrador es el rol que más funcionalidades puede realizar por lo cual *InterfazAdministrador* es la clase con más interacciones con otras clases.

Por otro lado, cuando la clase *IniciarInterfaz* detecta que el rol del usuario es recepcionista se ejecuta *InterfazRecepcionista*. Esta interfaz se compone por las clases: *CrearResarva*, *GenerarFecturaReserva*, *VerInventario*, *CancelarReserva*, *GenerarLog*, *cambiarUsuario*. En el caso del recepcionista es posible apreciar que su función principal es manipular todo lo relacionado con las reservas directamente y la primera interacción con el huésped.

Por último, si *IniciarInterfaz* detecta el rol de usuario empleado se ejecutará *InterfazEmpleado*. Esta interfaz es la más limitada y básica puesto que se compone de dos clases únicamente y puede realzar dos operaciones, estas son: *regisrarConsumo* y *cambiarUsuario*. Como podemos notar, empleado es el ente con menos requerimientos funcionales puesto que únicamente puede interactuar con la aplicación para registrar el consumo de un producto o servicio por parte de una reserva.

De esta forma es que se compone principalmente la interfaz gráfica. Se tomó la decisión de separarla por los tres tipos de roles que existen en el universo del problema, de tal forma que cada uno pudiera realizar sus funciones y estas no se mezclaran. A partir de ahí, se creó una clase por cada requerimiento funcional que se debía llevar a cabo para cada uno de los componentes, y, con base en eso, se realizaron las clases extra o auxiliares necesarias para poder completar el diseño exitosamente.

**4.2. Diseño interfaz gráfica de usuario de la aplicación para huéspedes de hoteles:**

La interfaz gráfica para este usuario fue mucho mas sencilla de diseñar que la de la administración de los hoteles. En este caso únicamente hubo un núcleo central, *InterfazHuésped*, esto debido a que solo iba a haber un tipo de usuario específico por lo cual solo hay un tipo de funcionalidades que puede realizar el usuario.

Inicialmente, una vez se corre la aplicación se inicializa un panel en el cual se presentan dos opciones únicamente, crear un nuevo usuario o verificar una ya creado. El único método nuevo de lógica en esta aplicación fue esta funcionalidad de crear un nuevo usuario, implementado por la clase *CrearUsuario*. Esta trabaja en conjunto con Hotel para añadir el usuario al txt de usuarios y agregarlo a los usuarios propios del hotel para poder realizar la lectura adecuada de los mismos tomando en cuenta el nuevo usuario creado.

Una vez creado un usuario se procede a realizar el inicio de sesión para un login y contraseña válidos se va a abrir el panel principal, *InterfazHuesped,* que se compone de 5 botones cada uno con una función. Las funcionalidades que se le permite realizar al usuario son, Observar el diagrama de ocupación anual del hotel, revisar la disponibilidad de una habitación en unas fechas específicas, realizar una reserva, pagar una reserva y cambiar de usuario. De esta forma, las calses que componen esta interfaz son *DispHabitacion, CrearReserva, ConsultarOcupacion* y las subclases que componen clase de las nombradas previamente.

Cabe resaltar que *CrearReserva* y *ConsultarOcupacion* junto con todas sus subclases son clases que se implementaron de la interfaz gráfica de administración de hotel para no repetir código de ninguna forma y que el desarrollo y desempeño de la aplicación sea óptimo.

Esta fue la lógica principal con la cual se desarrolló la interfaz gráfica para la aplicación de huéspedes, en la cual se reusaron muchas funcionalidades y lógica implementados para la aplicación de administración de

**5. Descripción de los archivos de datos:**

**5.1. Habitaciones**

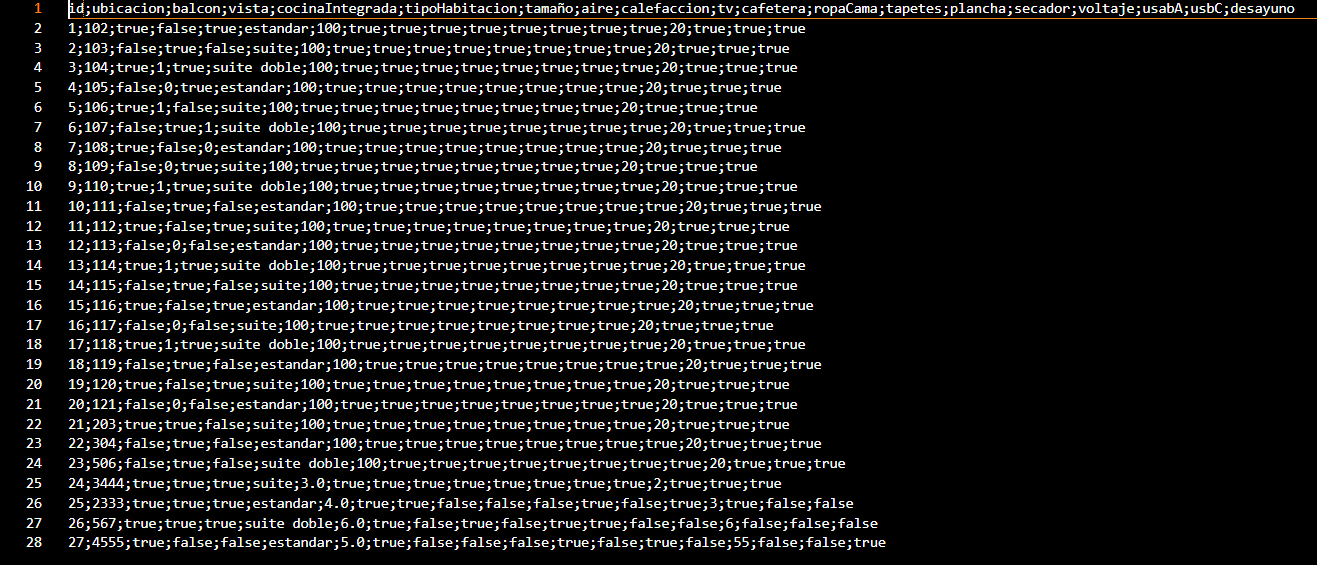


Figura Archivo .txt de habitaciones

Como se puede apreciar en la figura 1, el archivo con la información de las habitaciones contiene seis columnas, las cuales representan lo siguiente:

**Id**: El identificador único que cada habitación.

**Ubicación**: El número de la habitación respecto a la distribución del hotel.

**Balcón**: Un booleano que determina si la habitación tiene balcón o no.

**Vista**: Un booleano que determina si la habitación tiene vista hacía el exterior o no.

**Cocina** **Integrada**: Un booleano que determina si la habitación posee una cocina integrada o no.

**Tipo** **Habitación**: Determina el tipo de habitación, los tipos posibles son “estándar”, “suite” y “suite doble”.

**Tamaño:** Indica el tamaño del cuarto

**Aire:** Un booleano que determina si la habitación tiene aire acondicionado o no.

**Calefacción:** Un booleano que determina si la habitación tiene calefacción o no.

**TV:** Un booleano que determina si la habitación tiene TV o no.

**Cafetera:** Un booleano que determina si la habitación tiene cafetera o no.

**Ropa Cama:** Un booleano que determina si la habitación tiene ropa de cama hipolergénica o no.

**Tapetes:** Un booleano que determina si la habitación tiene tapetes hipoalergénicos o no.

**Plancha:** Un booleano que determina si la habitación tiene plancha o no.

**Secador:** Un booleano que determina si la habitación tiene secador o no.

**Voltaje:** Un entero que indica el valor del voltaje de los enchufes

**USBa:** Un booleano que determina si la habitación tiene USB tipo A o no.

**USBc:** Un booleano que determina si la habitación tiene USB tipo C o no.

**Desayuno:** Un booleano que determina si la habitación incluye desayuno o no.

**5.2. Cama**

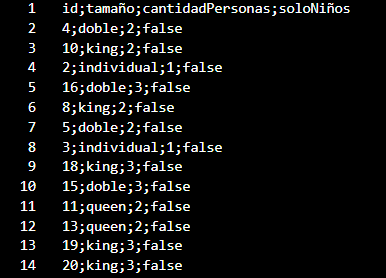


Figura Fragmento del archivo .txt con la información de las camas

Como se puede apreciar, el archivo tiene cuatro columnas que representan lo siguiente:

**Id:** El identificador único de la habitación a la que pertenecen.

**Tamaño**: Representa el tamaño de la cama, los tamaños posibles son “individual”, “doble”, “queen” y “King”.

**Cantidad** **Personas**: Representa el número de personas máximo que la cama puede alojar.

**Solo** **Niños**: Booleano que determina si una cama es solo apta para niños o no, las únicas camas que pueden ser aptas para niños son las individuales.

**5.3. Servicios**

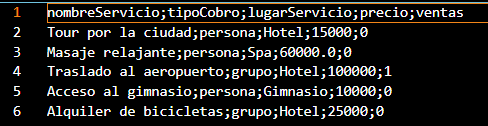


Figura Archivo .txt con la información de los servicios

Como se puede apreciar, el archivo de los servicios tiene 4 columnas, las cuales representan lo siguiente:

**Nombre** **Servicio**: Representa el nombre del servicio que presta el hotel.

**Tipo** **cobro**: Representa si el cobro es por persona o por grupo de personas.

**Lugar** **Servicio**: Representa el lugar en el que se presta el servicio, que puede o no ser dentro de las instalaciones del hotel.

**Precio**: Representa el valor en pesos que tiene el servicio, en caso de ser grupal el precio representa el valor que debe de pagar cada persona que usó el servicio.

**5.4. Productos del Restaurante**

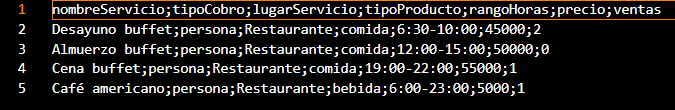


Figura Archivo .txt con el menú del restaurante

Como se puede apreciar, el archivo con el menú del restaurante tiene seis columnas, las cuales representan:

**Nombre** **Servicio**: El nombre del plato o bebida ofrecido por el hotel.

**Tipo** **Cobro**: Tipo de cobro, puede ser por persona o grupal.

**Lugar** **servicio**: Lugar en el que se presta el servicio.

**Tipo de** **Producto**: El tipo de producto, puede ser “comida” o “bebida”.

**Rango** **Horas**: El rango de horas del día en el que está disponible el producto del restaurante, está en formato “HH:mm-HH:mm”.

**Precio**: Precio en pesos colombianos del producto del restaurante.

**5.5. Usuarios**

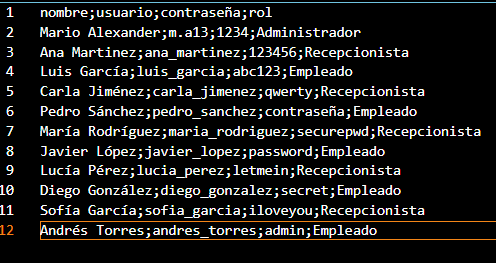


Figura Archivo .txt con la información de los usuarios

Como se puede apreciar, el archivo tiene cuatro columnas que representan lo siguiente:

**Nombre**: El nombre real del empleado o huésped del hotel.

**Usuario**: Un username con el que el empleado inicia sesión en el programa.

**Contraseña**: La contraseña asociada al usuario.

**Rol**: El rol del usuario, esto determina las opciones disponibles dentro del programa. Los roles posibles son “administrador”, “recepcionista”, “empleado” o “huésped”.

**5.6. Tarifas**

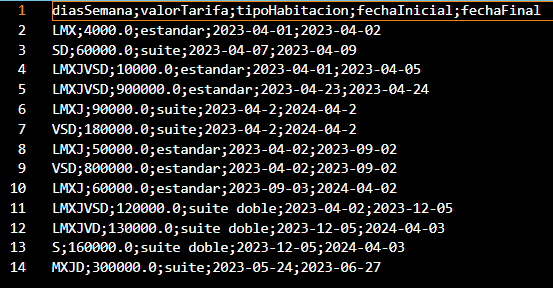


Figura 6 Archivo .txt con la información de las tarifas del hotel

Como se aprecia en la imagen, el archivo cuenta con 5 columnas posibles, cada una con información correspondiente a diferentes características de las tarifas:

**Días Semana:** Los días de la semana a os que se le implementan la tarifa

**Valor tarifa:** Valor en pesos de la tarifa

**Tipo Habitación:** Tipo de habitación a la cual se le aplica la tarifa

**Fechas Inicial:** Fecha inicial de la tarifa

**Fechas Final:** Fecha final de la tarifa

**5.7. Reservas**

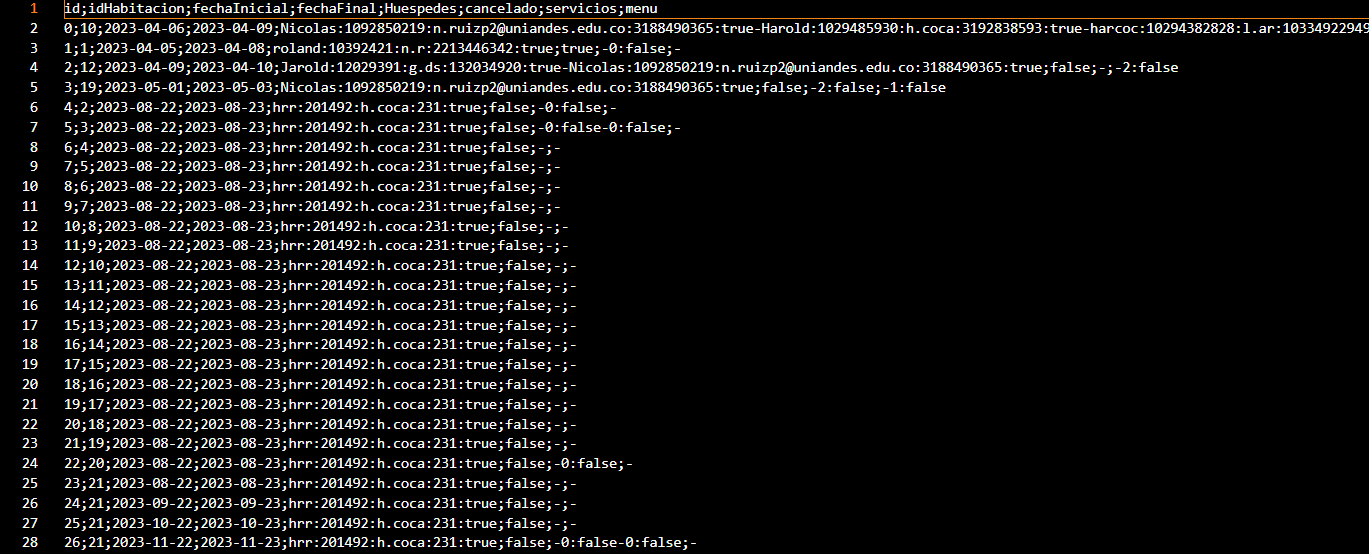


Figura 7 Archivo .txt con la información de las reservas del hotel

**Id:** Id de la reserva

**Id Habitación:** Id de la habitación de la reserva

**Fecha Inicial:** Fecha inicial de la reserva

**Fechas Final:** Fecha final de la reserva

**Huéspedes:** Huéspedes en al reserva

**Cancelado:** Productos cancelados de la reserva

**Servicios:** Servicios consumidos por la reserva

**Menú:** Productos del restaurante consumidos por la reserva

**5.8. Métodos de pago activos**

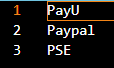


Figura 8 Archivo .txt con la información de los métodos de pago digitales activos del hotel

Únicamente cuenta con una columna y esta corresponde al nombre del método de pago digital con tarjeta activo.

**5.9. Tarjetas**



Figura 9 Archivo .txt con la información de las tarjetas de crédito disponibles para pagos

**Columna 1:** Número de la tarjeta de crédito

**Columna 2:** Fecha de caducidad de la tarjeta

**Columna 3:** Monto disponible

**5.10. Transacciones con PayU**

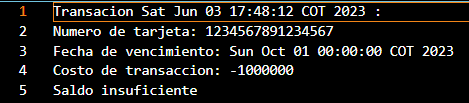


Figura 10 Archivo .txt con la información transacciones realizadas con PayU

**Fila 1:** Fecha de la transacción

**Fila 2:** Número de la tarjeta con la que se realizó una transacción

**Fila 3:** Fecha de vencimiento de la tarjeta

**Fila 4:** Monto por el cual se realizó la transacción

**Fila 5:** Resultado de la transacción

(Se repiten en el mismo orden los mismos parámetros para el resto de las filas)

**5.11. Transacciones con PayPal**

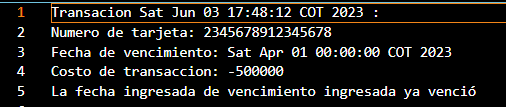


Figura 11 Archivo .txt con la información transacciones realizadas con PayPal

**Fila 1:** Fecha de la transacción

**Fila 2:** Número de la tarjeta con la que se realizó una transacción

**Fila 3:** Fecha de vencimiento de la tarjeta

**Fila 4:** Monto por el cual se realizó la transacción

**Fila 5:** Resultado de la transacción

(Se repiten en el mismo orden los mismos parámetros para el resto de las filas)

**5.12. Transacciones con PSE**

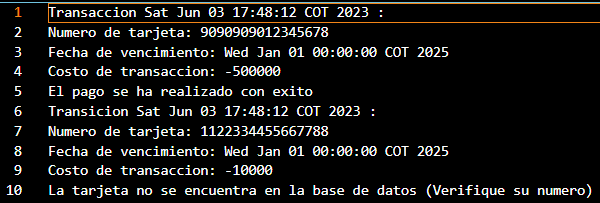


Figura 12 Archivo .txt con la información transacciones realizadas con PSE

**Fila 1:** Fecha de la transacción

**Fila 2:** Número de la tarjeta con la que se realizó una transacción

**Fila 3:** Fecha de vencimiento de la tarjeta

**Fila 4:** Monto por el cual se realizó la transacción

**Fila 5:** Resultado de la transacción

(Se repiten en el mismo orden los mismos parámetros para el resto de las filas)

**5.13. Huéspedes**

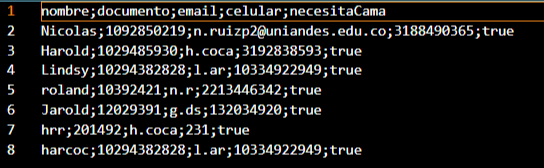


Figura 13 Archivo .txt con la información de los huéspedes que realizan las reservas del hotel

**Nombre:** Nombre de la persona que realizó la reserva

**Documento:** Documento del huésped

**Email:** Correo electrónico del huésped

**Celular:** Teléfono celular del huésped

**Necesita Cama:** Booleano que determina si necesita cama extra el huésped

**Cambios en la clase Habitación:**

Para la implementación se hicieron algunos cambios en el código, sobre todo en cuanto a la carga, creación y construcción de habitaciones. También se cambió el CSV de la siguiente forma:

**Texto

Descripción generada automáticamente**

*5 antiguo .txt Habitaciones*

**Texto

Descripción generada automáticamente**

*6. nuevo .txt Habitaciones*

Como se puede apreciar, se agregaron más variables al archivo para poder contener la nueva información que ahora contienen las habitaciones.

**Gráficas**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

La aplicación permite la visualización de 5 gráficas que representan información dentro de la aplicación:

Grafica ventas servicios: Muestra un histograma en el cual se muestran las ventas de cada producto/producto restaurante en el hotel.

Gráfica valor facturas: Muestra un gráfico de línea que gráfica el valor de las facturas de las reservas del hotel en función de la fecha en la que termina cada reserva.

Gráfica relación consumo/tarifa: Grafica un gráfico de línea que muestra la relación entre el consumo de las reservas en función de la cantidad pagada por la habitación.

Grafica 5 habitaciones más solicitadas: Muestra un histograma que representa las 5 habitaciones con más reservas registradas en el sistema, junto al número de reservas que tienen.

Graficar huéspedes con más reservas: Muestra un histograma que representa los 10 huéspedes con más reservas registradas en el sistema, junto al número de reservas.

Para la implementación de las gráficas solo se hizo un cambio en la lógica del modelo, pues se agregó a la clase Producto un contador de las veces que ha sido consumido, el cual aumenta cuando un empleado registra un consumo en el sistema.

**6. Uso de las interfaces**

**6.1 Uso de la interfaz de administración de hoteles**

Para poder comprobar el funcionamiento adecuado de la consola se sugieren tres usuarios diferentes, uno para administrador, otro para recepcionista y otro para empleado.

1. Administrador – Usuario: m.a13, Contraseña:1234
2. Recepcionista – Usuario: ana\_martinez, Contraseña:123456
3. Empleado – Usuario: luis\_garcia, Contraseña:abc123

Inicialmente haga login con el usuario de administrador. Una vez ingrese va a observar una ventana rectangular con 8 botones. Pruebe cada uno de ellos, al hacer click sobre cualquiera se abrirá una ventana emergente que le indicará lo que tiene que seleccionar para poder ejecutar la operación que desea. Puede comprobar que las funciones para crear nuevas habitaciones o servicios funcionaron revisando que haya un cambio en los .txt originales. Una vez termine de realizar una operación al hacer click en los botones para confirmarla la operación se ejecutará, podrá observar un panel que le muestra que su operación fue exitosa y se cerrará la ventana emergente permitiéndole elegir una nueva opción. Una vez haya probado todas las opciones posibles en administrador haga click sobre cambiar usuario e ingrese el usuario de recepcionista.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Ilustración Interfaz de administrador

Cuando ingrese al menú de recepcionista realice el mismo proceso que en el caso del administrador. Pruebe cada uno de las opciones que hay para seleccionar y llene lo que se solicita. Una vez más puede comprobar si crear reserva sirvió adecuadamente revisando el txt de reservas. Cuando realice todas las pruebas necesarias sobre recepcionista, una vez más haga click sobre cambar usuario e ingrese el usuario del empleado.

Interfaz de usuario gráfica

Descripción generada automáticamente

Ilustración Interfaz de recepcionista

Por último, cuando ingrese al menú del empleado, solo va a encontrar una posibilidad diferente a cambiar usuario y esta es registrar el consumo de un servicio. En esta va a observar que al hacer click sobre el botón se abrirá una ventana con los identificadores de la reserva a la que le quiere agregar el consumo, puede seleccionar si quiere agregar el consumo de un producto o servicio y luego el correspondiente producto o servicio.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ilustración Interfaz de empleado

Luego podrá simplemente cerrar la ventana con la X en la esquina superior y habrá terminad la prueba de la aplicación.

**6.2. Uso de la interfaz de la aplicación de huéspedes**

Una vez se corra la aplicación se debe iniciar creando un usuario de huésped ya que no hay usuarios creados con esta funcionalidad y compruebe el funcionamiento adecuado revisando el txt correspondiente a usuarios. Una vez creado el usuario, inicie sesión con este mismo.

Posteriormente pruebe la funcionalidad correcta para cada uno de las opciones que le permite este menú, de tal forma que observe que es posible crear una reserva y note el cambio efectuado en el txt correspondiente a reservas. Intente realizar un pago con las opciones de tarjeta que hay y observe el resultado de la transacción. Luego intente ingresar una habitación y un rango de fechas para determinar su disponibilidad. Por último ingrese a el de ocupación y revise que se muestra correctamente el gráfico indicado.

Por último, simplemente cierre la ventana con la X y ya habrá acabado de correr el programa.

**Pruebas unitarias:**

En el src hay dos archivos de pruebas, uno de pruebas de integración sobre las reservas y otro de pruebas unitarias sobre las cargas de datos.

Por parte de las pruebas de integración, se tiene la clase `ReservaIntegrationTest` que realiza pruebas de integración para la clase `Reserva` del paquete `Modelo`. Está utilizando el framework de pruebas JUnit 5. En el método `setup()`, se configuran los datos necesarios para la ejecución de las pruebas. Se crea una instancia de `Reserva` con un rango de fechas, una habitación, una lista de huéspedes y otros parámetros. A continuación, se definen una serie de métodos de prueba que verifican el comportamiento y los resultados esperados de varios métodos de la clase `Reserva`. Cada método de prueba está anotado con `@Test`, lo que indica que es un método de prueba.

Por ejemplo, el método `testGetHuespedes()` prueba el método `getHuespedes()` de la clase `Reserva`. Compara el resultado devuelto por el método con la lista de huéspedes configurada en la instancia de `Reserva` y verifica que sean iguales utilizando `Assertions.assertEquals()`. De manera similar, se realizan pruebas para otros métodos como `getRangoFecha()`, `getHabitacion()`, `getIdReserva()`, `getProductoMenuConsumido()`, `getServiciosConsumidos()`, `isCancelado()`, `setCancelado()`, `addProductoRestaurante()` y `addServicio()`. Cada prueba verifica un aspecto específico del comportamiento de la clase `Reserva`. Estas pruebas de integración ayudan a garantizar que la clase `Reserva` funcione correctamente y cumpla con los requisitos esperados en un escenario integrado.

Por parte de las pruebas unitarias, la clase `cargaDatosTest` es una clase de prueba que realiza pruebas para la carga de datos en diferentes controladores del sistema. Utiliza el framework de pruebas JUnit 5.

En el método `SetUp()`, se inicializan los controladores `ControladorHabitaciones`, `ControladorHuespedes`, `ControladorPagos`, `ControladorServicios` y `ControladorReservas`.

A continuación, se definen una serie de métodos de prueba que verifican el correcto funcionamiento de la carga de datos en cada controlador.

* El método `testCargaHabitaciones()` prueba la carga de datos de habitaciones y camas desde archivos. Verifica que se haya cargado correctamente una habitación específica y compara sus atributos con los valores esperados utilizando los métodos de aserción de JUnit.
* El método `testCargarTarifas()` prueba la carga de tarifas para las habitaciones. Verifica que se haya cargado correctamente el precio por día de una habitación en una fecha específica y lo compara con el valor esperado.
* El método `testCargarHuespedes()` prueba la carga de datos de huéspedes desde un archivo. Verifica que se haya cargado correctamente un huésped específico y compara sus atributos con los valores esperados.
* El método `testCargarMediosPago()` prueba la carga de formas de pago desde un archivo. Verifica que se hayan cargado correctamente las formas de pago y comprueba que no sean nulas.
* El método `testCargarTarjetas()` prueba la carga de información de tarjetas de crédito desde un archivo. Verifica que se hayan cargado correctamente las tarjetas y comprueba la presencia o ausencia de tarjetas específicas en el mapa de información de tarjetas.
* El método `testCargarServiciosYMenu()` prueba la carga de servicios y el menú de restaurante desde archivos. Verifica que se hayan cargado correctamente un servicio y un producto de restaurante específicos y compara sus atributos con los valores esperados.
* El método `testCargarReservas()` prueba la carga de reservas utilizando los datos cargados previamente en otros controladores. Verifica que se haya cargado correctamente una reserva específica y compara la habitación asociada con el ID de habitación esperado.

Estas pruebas aseguran que la carga de datos en los controladores se realice correctamente y que los datos se almacenen de manera adecuada para su posterior uso en el sistema.

**UML:** Los UML se la aplicación son los siguientes:

Diagrama, Esquemático

Descripción generada automáticamente

Este uml comprende la lógica, interfaz de usuario e interfaz de huepedes del proyecto.