Brew Tattoos

Requisitos de hardware

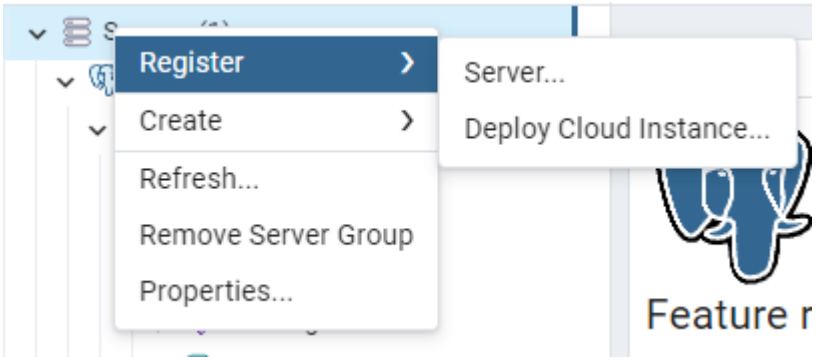
Para la parte de hardware se necesitará únicamente un ordenador (sobremesa o portatil) con 6gb de memoria RAM. Aunque funcione con unos 5gb lo recomendado serían 6gb, puesto que Docker ocupa bastante.

Requisitos de Software

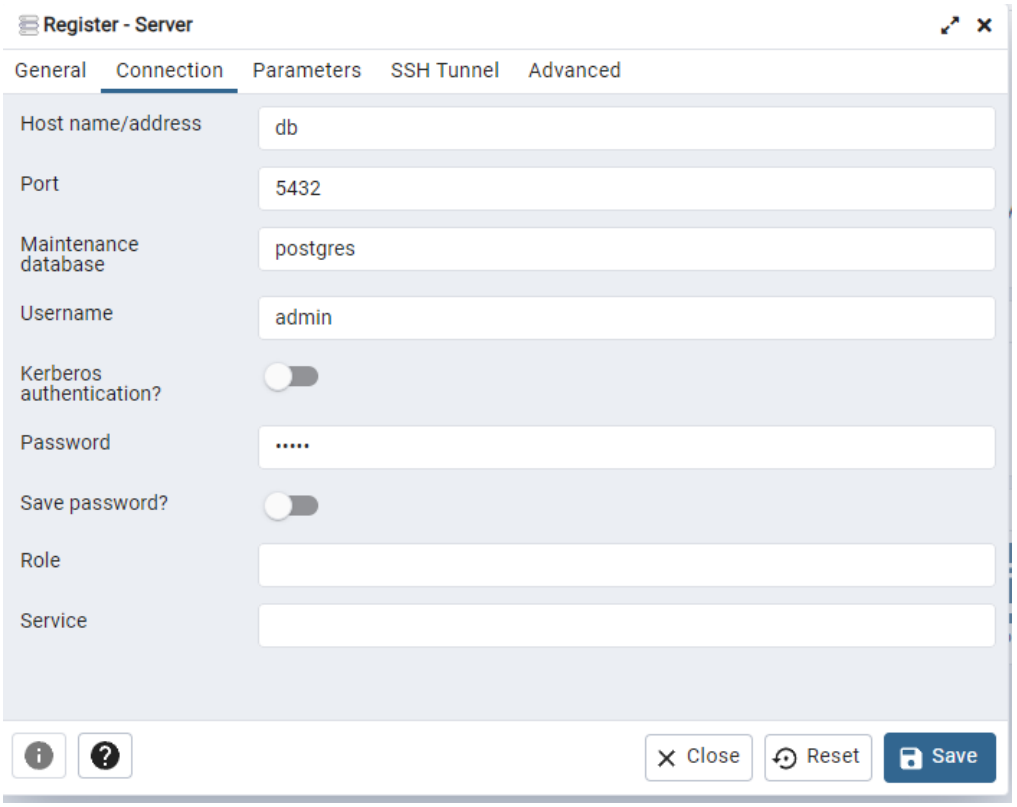
- JDK 11.0.11 como mínimo, ya que es la utilizada para el desarrollo del proyecto.
- Tener Java actualizado.
- Spring Tools Suite o IntelliJ IDEA como IDEs necesarios para la ejecución del proyecto.
- Librería Lombok para Java, puede ser descargada en el siguiente enlace gratuitamente: https://projectlombok.org/download
- Docker Desktop o en su defecto Docker por consola, aunque este proyecto ha sido desarrollado con el primero.

Guía de instalación del entorno y puesta en marcha

- Primero tendremos que tener instalados tanto Java como el JDK.
- Más tarde tras descargar el Spring Tools (El cual es portable, por lo que no hace falta instalación), tendremos que ejecutar la librería de Lombok, el cual nos pedirá que indiquemos la ubicación de nuestro IDE.
- Tras esto tendremos que instalar Docker Desktop y dejarlo abierto.
- Una vez hecho todo lo anterior tendremos que dirigirnos a nuestro IDE y abrir el proyecto. Dentro de este hay un archivo **application.properties** en la siguiente ruta: src/main/resources. En este archivo tendremos que asegurarnos tanto de que la línea de **PROD** esté activa como de que la línea de **DEV** esté comentada.
- Ahora desde la carpeta raíz del repositorio de git (donde se encuentra el archivo docker-compose.yml) tendremos que realizar el siguiente comando en consola: **docker-compose up -d**. Esto creará los contenedores que utilizaremos para la base de datos.
- Una vez arrancados los contenedores iremos a la ruta en Docker Desktop del contenedor pgadmin y nos aparecerá un login. Tendremos que introducir las credenciales admin@admin.com **/ root**.
- Lo único que nos quedaría sería registrar el servidor en PostGreSQL de la siguiente manera:



Una vez pulsemos en **Server…** nos aparecerá una ventana que nos indica que tenemos que introducir un nombre de servidor, aquí podemos poner lo que sea, la pestaña importante es la de **Connection**, la cual se encuentra justo encima del diálogo. Tendremos que rellenarla de la siguiente forma:



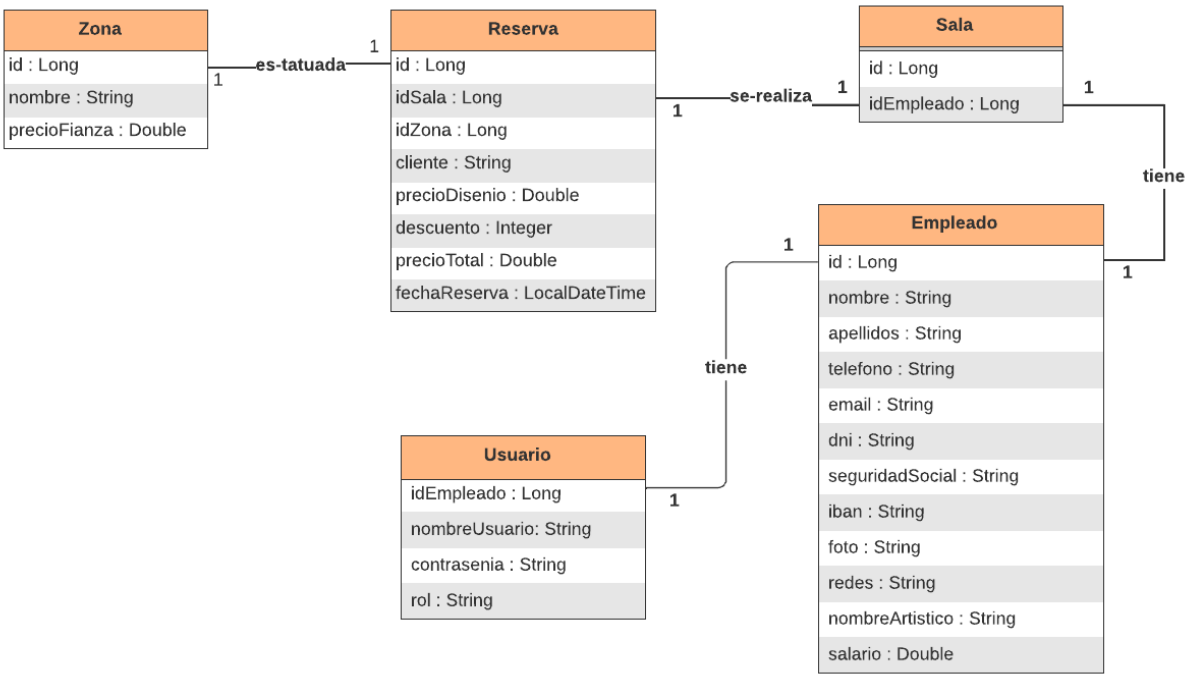
El usuario y contraseña son **admin / admin**. Cuando esté todo relleno pulsamos en **Save** y ya estaría nuestra BBDD creada, si queremos podemos comprobarlo desplegando los menús de la izquierda.

- El último paso que nos quedaría sería pulsar en el proyecto y después en el **botón verde de Play** justo encima. La primera vez que lo iniciemos nos pedirá cómo queremos ejecutarlo, tendremos que seleccionar **Spring Boot Application** (Normalmente está la última o penúltima opción).
- Ahora lo único que quedaría sería acceder a localhost:9000 para visualizar la página. Podemos acceder a la parte del administrador pulsando en Iniciar sesión arriba a la derecha y utilizando las credenciales que encontraremos en el **import.sql** en la siguiente ruta: **src/main/resources**. El usuario y contraseña que nos permite visualizar todo es: **admin** / **admin**.

Explicación del funcionamiento de la aplicación

Las funcionalidades de la página se pueden dividir en dos partes, la primera es la que ve el usuario (aquel que no esté logueado), y la segunda es la del administrador.

Diagrama de clases



Usuario

La vista de usuario se caracteriza por un menú navegable en el cual podremos explorar las diferentes zonas de nuestra página principal (Landing).

Podemos encontrar los apartados estudio, artistas, dónde estamos y pide cita.

El apartado de artistas se pinta con un foreach con thymeleaf directamente con la tabla empleado de la base de datos.

En este último apartado encontramos un formulario el cual podemos rellenar y una vez pulsado el botón de **Enviar** se nos notificará con un alert por JavaScript que nuestra petición ha sido enviada.

En este formulario hay un input de tipo hidden, el cual cambia el value a través de JavaScript para recoger todos los datos proporcionados y son enviados a nuestro controlador en Java como un String. Este String junto con un mensaje de “Nueva cita” más la fecha actual son enviados al correo ejemplotrabajofinal@gmail.com, al cual se puede acceder con la contraseña: **iesmajuelo**. A este llegará la información mencionada anteriormente.

Además de lo ya explicado, en cada formulario de esta aplicación se ha utilizado un archivo JavaScript **personalizado** para la revisión de que todos los campos son rellenados correctamente.

También arriba a la derecha podemos pulsar en **Iniciar Sesión**, lo que nos llevará a la pantalla del Login.

En la parte del footer se encuentran algunos datos adicionales: el e-mail del estudio, las redes sociales y el horario. Además también dispondremos de enlaces que nos conducirán a distintas partes de la página.

Y por último se ha utilizado la librería de JavaScript AOS para las animaciones cada vez que hacemos scroll por la página (https://michalsnik.github.io/aos/).

Administrador

Por otra parte, la vista de administrador nos permitirá gestionar las citas de los clientes y asignarlas a los artistas de nuestro estudio, además de poder modificar tanto los precios de las zonas de los tatuajes, como los tatuadores que se encuentran asignados a cada sala. El admin podrá editar cada uno de ellos, el empleado solo podrá editar el suyo mismo.

A estas gestiones podremos acceder a través del navbar estático o con los cuatro botones que se muestran en el main de la zona de administración.

Como podemos observar existen cuatro tipos de operaciones en la gestión del estudio: gestión de zonas del cuerpo, gestión de las citas (o reservas), gestión de salas y gestión de empleados.

Además de todo esto podremos acceder a la landing del usuario si realizamos click en la opción de “Zona usuario”. Gracias a la autenticación que hemos realizado en nuestro login en el navbar de la vista de usuario aparecerán tanto una opción que puede mandarnos de vuelta a la zona admin, como un desplegable con el nombre de usuario y una opción para cerrar sesión.
Al pulsar en este botón de cerrar sesión nos devolverá a la página principal de la vista de usuario, donde podremos volver a loguearnos si queremos proseguir con nuestras gestiones.

Volviendo a nuestro menú principal de la vista de administrador podremos ir visualizando los distintos datos que nos ayudarán a realizar las citas o reservas, las cuales son la finalidad de esta página.

- Empezando con el botón “Mostrar zonas”. Dentro de este podremos encontrar una lista de zonas del cuerpo, las cuales tienen además un precio de fianza, además de dos botones (editar y eliminar).

  También como podemos comprobar encima de nuestra tabla de datos existe un buscador, en el cual podemos filtrar la tabla por precio y un botón de añadir una nueva zona, en caso de que nos sea necesario. Al usar el editar o el añadir se nos abrirá un formulario, en el que podemos cumplimentar la información necesaria para que nuestra zona quede registrada en nuestra base de datos.

  Aquellas que guardemos nos permitirán más tarde calcular el precio total de nuestra sesión de tatuaje.

  Tras enviar el formulario se nos devolverá a la parte de la tabla, donde podremos comprobar si nuestra zona aparece en dicha tabla o utilizar cualquier botón del navbar para realizar cualquier otra gestión.

- Ahora pasemos al botón de mostrar salas. Una vez pulsado nos dirigirá a una nueva tabla donde podremos comprobar que existen 4 salas (las mismas que en el propio estudio).

  Cada sala está asignada a uno de nuestros tatuadores. En esta parte de nuestra página también dispondremos de los botones añadir, editar y borrar, además del buscador en el cual podemos filtrar por nombre del empleado. Como nuestro estudio solamente dispone de 4 salas únicamente tendremos en él 4 tatuadores, por lo que ahora mismo nuestras únicas opciones serían borrar a uno de ellos antes de agregar uno nuevo o editar uno ya existente.

  Si accedemos a uno de los formularios podremos ver que solamente podremos rellenar el id del tatuador en ellos.

  Al darle a enviar como anteriormente se nos redirigirá a la página de la tabla con todos los tatuadores en ella.

- Si pulsamos el botón de mostrar empleados podremos observar una lista de 5 empleados, si nos logueamos como admin, o un único perfil, si accedes como tatuador.

  Lo único que podemos hacer es con esta tabla es editar, puesto que siempre habrá 4 tatuadores registrados en el estudio. Se podrá modificar toda la información personal del empleado (además de la foto), pero este no podrá ser añadido o eliminado.

Tras realizar modificaciones (si lo deseamos) podemos pulsar en Enviar y esto nos llevará a la zona de la tabla, la cual estará actualizada con la información que hemos indicado anteriormente.

- Por último haremos una visita a la parte de gestión de citas pulsando en “Mostrar citas”, que es donde estas zonas, salas y empleados convergerán para registrar una reserva.

  Como vemos esta es una tabla más amplia en la que están registrados todos los datos de nuestra cita: El nombre del cliente, la fecha de la cita, el tatuador que va a realizar el tatuaje (que viene dado por la sala que tiene asignada dicho tatuador), la zona en la que el cliente se lo va a realizar, el precio de fianza de dicha zona (que viene dado por la tabla Zona), el precio del diseño, el descuento que le hemos ofrecido y el precio total que se suma automáticamente al realizar la reserva.

  También volvemos a tener nuestros botones como en las anteriores páginas que nos permitirán añadir, modificar y eliminar las citas.

  Pasemos al formulario de la reserva:

  Aquí podremos introducir los datos pertinentes. La zona del tatuaje y la sala serán asignadas por debajo gracias a la recogida de los IDs, aunque al realizar la reserva aparezca en los textos la información pertinente de cada tabla. Además también se realizan los cálculos necesarios para conseguir el precio total de la cita al darle al botón de “Enviar”.

  Una vez despachado el formulario se nos devolverá a la misma tabla de citas mostrada anteriormente, en la que podremos proseguir con nuestras gestiones, volver al menú principal o cerrar sesión si lo vemos necesario.

Lógica de negocio

● **Mayor de 18:** Una de las lógicas de negocio del estudio será que el cliente deberá ser mayor de 18 años para ser tatuado. Ésta es aplicada en el formulario de la zona de usuario, ya que si no eres mayor de 18 no será posible enviar la información para solicitar la cita.

● **Suma total:** Cuando se han introducido todos los datos del tatuaje, el precio de la fianza del objeto Zona será setteado y se realizará el cálculo del precio total con el precio del diseño y su descuento.

● **Descuento del 10%:** Cuando el precio del diseño del tatuaje sea mayor de 200 € se le asignará automáticamente un descuento del 10% (en caso de que no exista un descuento mayor o igual a este).

● **Máximo 2 tatuajes al día:** Como máximo un tatuador podrá realizar dos sesiones al día, ya que el primero es realizado en la mañana y el segundo en la tarde. En caso de que intentemos asignar una tercera cita se nos redirigirá a un mensaje de error que nos lo notificará.

Explicación del código o procesos más complejos o interesantes.

Principalmente los procesos más complejos han sido los del ReservaService, que es donde se han creado los métodos para que las funciones principales funcionen. Algunos de ellos son:

- **void settearSala(Reserva a, List <Sala> lista):**

  Aunque este método sólo establece el empleado de una sala en función de una lista de salas, su descubrimiento hizo el desarrollo mucho más liviano, puesto que hasta ese momento no me dí cuenta de que para setear una Sala dentro de una Reserva debía hacerlo directamente con un seteo en el **save** de la entidad a cambiar. También he utilizado otro prácticamente igual con Zona.

  Los parámetros utilizados son:

  - **a**: La reserva a la cual se le establecerá el empleado de la sala.
  - **lista**: La lista de salas disponibles.

  Se crea un iterator que recorre la lista de salas y busca una sala que tenga el mismo id que la sala asociada a la reserva "a". Cuando encuentra la sala correspondiente, establece el empleado de la sala en la reserva.

- **comprobarFecha(Reserva a, List <Reserva> lista, List <Sala> listaSala):**

  Este método comprueba si una reserva cumple el requisito de dos citas como máximo en un día.

Los parámetros utilizados son los siguientes:

- **a**: La reserva a comprobar.
- **lista**: La lista de reservas en la cual se realizará la comprobación.
- **listaSala**: La lista de salas asociadas a las reservas.

Primero se llama al método "settearSala" para establecer la sala asociada a la reserva "a" en función de la lista de salas. Luego se utiliza un stream para filtrar las reservas de la lista que tienen el mismo empleado y la misma fecha que la reserva "a". Por último, se compara el tamaño de la lista filtrada con la constante int "numCitas", la cual equivale a 2, y devuelve true si es mayor o igual, o false en caso contrario.

Este valor se recogerá más tarde para que se muestre en la página un mensaje de error en caso de que no se haya cumplido la condición de la fecha.

- **settearPrecioTotal(Reserva a):**

  Este método calcula y establece el precio total de una reserva en función del precio del diseño, la zona y el descuento.

  Parámetro:

  - **a**: La reserva a la cual se le calculará y establecerá el precio total.

  Primero se comprueba si el precio del diseño de la reserva es mayor o igual que la constante "numDisenio", cuyo valor es 200, y si el descuento actual de la reserva es menor que la constante "numDescuento", la cual vale 10. Si se cumplen ambas condiciones, establece el descuento de la reserva como el valor de la constante. Luego se calcula el precio total restando al precio del diseño más la fianza de la zona, el porcentaje de descuento aplicado.

  Esto forma parte de la lógica de negocio mencionada anteriormente, si un diseño cuesta más de 200€ se le aplica un descuento del 10% en caso de que no tenga aplicado un descuento mayor.

- **findAceptadas(List <Reserva> lista, Optional <Usuario> user) y findPendientes(List <Reserva> lista, Optional <Usuario> user)**

  Estos métodos encuentran las reservas que han sido aceptadas y las que están como pendientes, además las filtra con el usuario que está logueado actualmente.

Existen otras versiones iguales de estos métodos, pero sin el usuario. Estas son utilizadas para la vista del admin.

Como parámetros tenemos:

- **lista**: Es la lista de reservas existentes.
- **user**: El usuario logueado actualmente.

Se utiliza un stream para filtrar las reservas de la lista que no están aceptadas o que si lo están, dependiendo del método, y que tienen el mismo id de empleado que el usuario proporcionado, ya que comparten id. Más tarde se devuelve una lista con las reservas filtradas.

Esto fue desarrollado para la vista de las reservas en la tabla de citas, ya que hay dos apartados diferentes en los que se diferencian estas dos listas.

- **buscarTatuadores(String nombre, List <Reserva> lista)**

  Busca tatuadores en la lista de reservas por su nombre. El cual es cogido desde el formulario de búsqueda de HTML.

  Los parámetros son:
  - **nombre**: El nombre del tatuador a buscar.
  - **lista**: La lista de reservas en la cual se realizará la búsqueda.

  Recorre la lista de reservas con un for each y compara el nombre del empleado asociado a cada reserva con el nombre proporcionado. Si hay coincidencia, agrega la reserva a una lista final que se devuelve como resultado para mostrarlo posteriormente. Esto más tarde se recoge en el controlador de reservas y se muestra en la tabla.

  Existe otro método igual para buscar por nombre de cliente en caso de que el logueo haya sido con una cuenta de empleado.

Por último este método me llevo bastante tiempo desarrollarlo y tuve que buscar bastante información, es un método simple pero que cumple una función muy grande. Se tuvo que crear un servicio aparte solo para esta funcionalidad, la de enviar un email a una dirección en concreto.

- **void sendEmail(String toEmail, String subject, String body)**

  Este método envía un correo electrónico utilizando la clase JavaMailSender.

  Los parámetros utilizados son los siguientes:
  - **toEmail**: La dirección de correo electrónico del destinatario.
  - **subject**: El asunto del correo electrónico.
  - **body**: El cuerpo del correo electrónico.

  Primero se crea un objeto SimpleMailMessage y establece los atributos del mensaje, como la dirección de correo electrónico del remitente, el destinatario, el texto y el asunto. Luego se utiliza el JavaMailSender (que se ha inyectado anteriormente utilizando @Autowired) para enviar el mensaje de correo electrónico con el método **send**.

  Esto hace posible el envío del formulario de usuario al correo electrónico de ejemplo mostrado anteriormente. Junto con el archivo de JavaScript que reúne todos los value de los diferentes inputs hace posible esta funcionalidad.

Mayores dificultades encontradas.

- Principalmente el cambio de IDE y de tipo de proyecto (Maven) han supuesto una dificultad bastante grande, aunque una vez le cogí el gusto he sabido desenvolverme bien.

  Spring tiene una cosa que es bastante potente como incluir en el archivo pom.xml todas las dependencias que necesites y el archivo application.properties que te permite modificar las propiedades del proyecto a tu gusto, pero a la vez hacen el salto de un proyecto de Java normal a uno de Spring bastante grande.

- Algo que me supuso también un gran reto fue el formulario de enviar emails al pulsar el botón de enviar. Aunque en las prácticas vi bastante JavaScript, el mezclar este con Java me costó bastante. Una vez pude unir la información de varios tutoriales me quité un gran peso de encima, puesto que esta es la funcionalidad más importante de la zona de usuario.
- También el correcto funcionamiento de Docker me llevó bastante tiempo, puesto que había que implementar varios cambios en Spring, como por ejemplo la incorporación de dependencias (que en ese momento no tenía mucha idea de como hacerla) y la creación tanto de perfiles de Spring como del docker-compose.

Además los perfiles me fueron bastante útiles puesto que podía desarrollar en una base de datos ficticia y podía realizar bastantes cambios de golpe sin tener que ponerme a borrar contenedores de Docker cada vez.

- Por último también me tomé como mejora de disciplina a la hora de organizar el tema de la distribución de paquetes del proyecto. Me costó bastante saber dónde iba cada clase/interfaz, aunque al final del proyecto me ayudó bastante para poder acceder rápidamente a las diferentes zonas del trabajo.

Trabajo realizado por Pedro Puertas Rodríguez