Explicación PRACTICA “ENTORN SERVIDOR”

Andrés Carrasco Dols
2r CFGS Desenvolupament d’Aplicacions Web

INDEX

OBJECTIVOS

Este trabajo parte de la primera práctica e incorpora los siguientes requisitos:

1. Las imágenes se guardan automáticamente en el servidor cada vez que se agrega o elimina un objeto, o periódicamente , yo he elegido la segunda opción , a la hora de crear el dibujo se crea una imagen y una versión apretando el botón y en el modify se crea una nueva versión al darle enviar o se guarda automáticamente a los treinta segundos.

2. Cada vez que se guarda una imagen, se almacena también la versión anterior, creando un historial de cambios. Estas versiones son consultables al ver la imagen, y cada imagen siempre tiene una versión "actual", que es la última añadida.

3. El usuario puede recuperar cualquier versión anterior de la imagen, cada una con un marcador de tiempo (timestamp) asociado.EL marcador será fecha de creación de cada versión.

4. Al eliminar una imagen, se borran también todas sus versiones anteriores.

5. Cuando un usuario elimina una imagen, esta no se elimina de la base de datos, sino que se traslada a la papelera del usuario. El usuario puede optar por eliminar definitivamente la imagen también de la papelera.

6. La información en el servidor se almacena en una base de datos MySQL para evitar pérdida de datos al reiniciar la aplicación en el servidor.

7. Las imágenes pueden compartirse en modo "lectura" y "escritura". Un usuario solo puede ver imágenes "públicas" o directamente compartidas con él.

8. Se implementa un sistema de permisos (lectura/escritura) para cada imagen compartida. Usuarios con permisos de escritura pueden borrar y modificar la imagen.

9. Solo los "propietarios" de las imágenes pueden compartirlas. Un usuario no puede compartir con otros usuarios una imagen que no sea suya.

10. Un usuario puede realizar copias de imágenes (sin incluir el historial) propias o compartidas.

ESTRUCTURA DEL PROGRAMA

Este proyecto estará dividido en dos partes, el **backend** que maneja el procesamiento de datos y la lógica del servidor, es decir , realiza las operaciones que no son visibles para el usuario y el **frontend** se encarga de la interfaz y la interacción con el usuario, en resumen los dos trabajan juntos para crear una aplicación completa.

En la parte **backend** contamos con:

-Controller:En esta carpeta, se encuentran los Servlets(controladores),que se encargan de manejar y coordinar todas las solicitudes que llegan a la aplicación, asegurándose de que todo funcione como debería. Está dividido en dos:

-**Get**: Se utiliza para obtener datos del servidor y enviarlos al cliente.

-**Post**: Se utiliza para enviar datos desde el cliente al servidor.

-Services: Esta carpeta sirve como intermediario entre los controladores y el Repo. Ayuda a mantener una separación clara entre la lógica del servlet y la interacción con la base de datos a través del Repo.

-Entities: Aquí se definen las clases que representan los objetos de la aplicación.

-Repos: Se encarga de definir los métodos que el servicio debe implementar , también de interactuar con la base de datos y realizar operaciones de lectura y escritura.

-DTO: Actúa como un contenedor que agrupa información proveniente de diversas clases en una única estructura de datos. Facilita la transferencia de datos.

-Filter: Contiene clases relacionadas con la gestión de filtros en la aplicación web. Estas clases realizan la autenticación del usuario antes de que la solicitud llegue a los controladores.

-Utils: Sirve como un repositorio para funciones que son comunes en el código y pueden ser utilizadas a lo largo de la aplicación.

En la parte **frontend** contamos con los archivos **:**
-WEB-INF/html: Esta carpeta alberga las páginas html.
-Js: Contiene archivos JavaScript para la interactividad del frontend.
-Css : En esta carpeta se definirá los estilos de nuestras páginas html.

TECNOLOGIAS ASSOCIADAS

Maven
Maven es una herramienta de construcción y gestión de proyectos Java. Simplifica la gestión de dependencias, define un ciclo de vida para el desarrollo y organiza la estructura del proyecto.

Tomcat
Tomcat es un servidor de aplicaciones web de código abierto especializado en ejecutar servlets y JavaServer Pages (JSP). Proporciona un entorno ligero y fácil de usar para el desarrollo y despliegue de aplicaciones web Java.

Docker
Docker es una plataforma de contenerización que permite empaquetar aplicaciones y sus dependencias en contenedores, facilita la implementación y ejecución de aplicaciones en entornos aislados y distribuidos.

Spring
Spring Web es un módulo de Spring Framework diseñado para facilitar el desarrollo de aplicaciones web en Java, proporcionando abstracciones y patrones de diseño para la creación de controladores, manejo de solicitudes y vistas.

JdbcTemplate
JdbcTemplate es una clase de Spring que simplifica el acceso a bases de datos relacionales mediante JDBC en aplicaciones Java.

MySQL
MySQL es un sistema de gestión de bases de datos que facilita el almacenamiento, organización y recuperación de datos. Utilizado en aplicaciones web, permite la creación de tablas para estructurar la información y ofrece un lenguaje (SQL) para realizar consultas y manipular datos de manera fácil.

En este proyecto utilizamos un archivo de configuración llamado **pom.xml** para definir la estructura y las dependencias del proyecto. En este archivo, se especifican las dependencias necesarias para compilar y ejecutar el proyecto, así como otras configuraciones esenciales.

Anotaciones Spring

@Controller

La anotación @Controller en Spring se utiliza para marcar una clase como un controlador, indicando que gestiona solicitudes HTTP y define métodos para manejar esas solicitudes. Un controlador se encarga de procesar las solicitudes del cliente y devolver la respuesta adecuada.

@Autowired

La anotación @Autowired facilita la gestión de dependencias al permitir que Spring inyecte automáticamente las instancias necesarias en las variables de clase marcadas con esta anotación. Esto elimina la necesidad de crear instancias manualmente y simplifica la integración entre componentes en una aplicación Spring.

@Component

Indica que una clase es un componente gestionado por el contenedor de Spring, utilizado comúnmente para clases no específicas. Estos componentes son candidatos para la detección automática por parte de Spring, simplificando la configuración y gestión

@Configuration

Indica que esta clase es una clase de configuración de Spring.

@Services

Es una anotación específica de Spring que se utiliza para marcar clases como servicios, facilitando la gestión de componentes, la detección automática y la inyección de dependencias en aplicaciones basadas en Spring.

@Repository

Es una anotación específica para capas de acceso a datos en una aplicación Spring y juega un papel importante en la gestión de excepciones y en la integración de operaciones de base de datos con el contenedor de Spring.

EXPLICACIÓN DEL CÓDIGO (BACKEND)

CONTROLLER

RegisterController

El controller **RegisterController** maneja solicitudes relacionadas con el registro de usuarios en la aplicación web.

Creamos una instancia de la clase **UserService** que será el intermediario entre los servlets y el Repo, nos permitirá incluir métodos para validar el usuario.

Luego tenemos **@GetMapping("/register")** que maneja las solicitudes GET dirigidas a "/register". En este caso no hace nada más ya que no enviamos información del servidor a la página html.

Por último tenemos el **@PostMapping("/register")** que se encarga de recibir información del formulario de registro, en este caso el login , name y el password de tu cuenta. Luego utilizamos estos datos para registrar un usuario mediante **userService.registerUser()**. Si hay un problema, muestra un mensaje de error en la página de registro. Si el registro es exitoso, redirige al usuario a la página de inicio de sesión ("/login").

```
@Controller
public class RegisterController {
    1 usage
    @Autowired
    UserService userService;
    no usages  Andres Carrasco Dols
    @GetMapping("/register")
    public String Register() { return "register"; }

    no usages  Andres Carrasco Dols *
    @PostMapping("/register")
    public String PostRegister(Model model, @RequestParam String login,
                               @RequestParam String name,
                               @RequestParam String password) {
        //Metodo para registrar el usuario.
        String register = userService.registerUser(login, name, password);
        //Si no cumple con los requisitos te muestra un mensaje de error.
        if (register != null) {
            model.addAttribute( attributeName: "error", register);
            return "/register";
        } else { //Si se ha guardado correctamente te redirige a login.
            return "redirect:/login";
        }
    }
```

LoginController

El controller **LoginController** maneja solicitudes relacionadas con el inicio de sesión de los usuarios en nuestra aplicación web , responderá a las solicitudes que lleguen a esa URL específica login.

Creamos un **HttpSession session** que utilizará para almacenar información del usuario después de que ha iniciado sesión.

También creamos una instancia de la clase **UserService** que será el intermediario entre los servlets y el Repo, nos permitirá incluir métodos para validar el usuario.

El método **login** maneja las solicitudes GET dirigidas a **"/**login**" ,**en este caso, simplemente redirige la solicitud a la página html sin realizar ninguna acción.

El método **Postlogin** obtiene los parámetros del formulario de inicio de sesión, en este caso (login, password) para verificar si el usuario existe mediante **userService.userExists()**. Si es correcto, establece la sesión del usuario y redirige a la página "/CanvasDraw". Si no es correcto, muestra un mensaje de error en la vista de inicio de sesión.

```java
@Controller
public class LoginController {
    1 usage
    @Autowired
    HttpSession session;
    2 usages
    @Autowired
    UserService userService;
    no usages  Andres Carrasco Dols
    @GetMapping("/login")
    public String Login() { return "login"; }

    no usages  Andres Carrasco Dols
    @PostMapping("/login")
    public String PostLogin(Model model,@RequestParam String login, @RequestParam String password) {
        //Metodo para iniciar sesion, si se ha iniciado correctamente.
        if (userService.userExists(login, password)) {
            session.setAttribute( s: "user", userService.user(login));
            return "redirect:/CanvasDraw";
        }
        //Si no se ha iniciado correctamente sale el error.
        model.addAttribute( attributeName: "error",  attributeValue: "El usuario o la contraseña incorrectos.");
        return "login";
    }
}
```

LogoutController

El controller LogoutController se encarga de las solicitudes relacionadas con la finalización de sesión.Obtenemos la sesión actual con **HttpSession session** que hemos definido antes en el login.

El método **Logout** indica que el método Logout responderá a las solicitudes dirigidas a la URL "/Logout". En este caso, maneja solicitudes de cierre de sesión.

Si la “**session”** no es nula significa que hay una session activa , si es así, el método invalida la sesión existente utilizando el método invalidate() de la interfaz HttpSession. Luego, redirige al usuario a la página de inicio de sesión ("/login").

```java
@Controller
public class LogoutController {
    2 usages
    @Autowired
    HttpSession session;

    no usages  Andres Carrasco Dols
    @GetMapping("/Logout")
    public String Logout() {
        // Invalidar la sesion existente
        if (session != null) {
            session.invalidate();
        }
        // Redirirgir a login
        return "redirect:/login";
    }
}
```

CanvasController

El controller CanvasController maneja las interacciones relacionadas con la creación de dibujos .Los métodos GET y POST realizan operaciones específicas para mostrar la página y procesar la creación de dibujos, también obtenemos la sesión actual con **HttpSession session** que hemos definido antes en el login.

El método **CanvasDraw** ,maneja las solicitudes dirigidas a "/CanvasDraw". En este caso obtiene el usuario actual de la sesión y agrega su nombre de usuario al modelo. Esto lo hacemos para mostrar un mensaje de bienvenida al usuario.

```
@Controller
public class CanvasController {
    2 usages
    @Autowired
    HttpSession session;
    1 usage
    @Autowired
    DrawService drawService;

    no usages  Andres Carrasco Dols
    @GetMapping("/CanvasDraw")
    public String CanvasDraw(Model model) {
        //Obtenemos el usuario actual
        User user = (User) session.getAttribute( s: "user");
        model.addAttribute( attributeName: "user", user.getLogin());
        return "CanvasDraw";
    }
```

El método **PostMapping("/CanvasDraw")** , este método obtiene el usuario actual y los parámetros del formulario (figures, NomImage y visibilidad). Utilizamos la función **saveDrawAndVersion** para guardar el dibujo y su versión asociada, pasando los parámetros necesarios. Si el guardado se realiza correctamente, se devolverá la página "/CanvasDraw". Si ocurre un error, se agregara un mensaje de error a la plantilla y se mostrará nuevamente la interfaz de CanvasDraw.

```
@PostMapping("/CanvasDraw")
public String PostCanvasDraw(Model model, @RequestParam String figures, @RequestParam String NomImage,
                             @RequestParam String visibility) {
    //Obtenemos el usuario actual
    User user = (User) session.getAttribute( s: "user");
    //Metdodo para guardar el dibujo y la version.
    String saveDrawAndVersion = drawService.saveDrawAndVersion(user, model, NomImage, visibility, figures);
    //Si no se ha guardado correctamente, mensaje de error.
    if (saveDrawAndVersion != null) {
        model.addAttribute( attributeName: "error", saveDrawAndVersion);
        return "CanvasDraw";
    } else {
        return "redirect:/CanvasDraw";
    }
}
```

AllDrawController

El controller AllDrawController se encarga de mostrar una lista de dibujos del usuario y procesar acciones relacionadas con esos dibujos, como enviar un dibujo a la papelera , ver el el dibujo, compartirlo y modificarlo.

El método **AllDraw** maneja las solicitudes dirigidas a "/AllDraw". Obtenemos el usuario actual de la sesión, luego utilizamos el servicio drawService que contendrá un método **getDraws** para obtener una lista de objetos **DrawWithVersionDTO** que representan los dibujos del usuario. Esta lista se envía a la pagina gracias al objeto **model** que se encarga de pasar datos desde el controlador a la vista.

```java
@Controller
public class AllDrawController {
    2 usages
    @Autowired
    HttpSession session;
    2 usages
    @Autowired
    DrawService drawService;

    no usages  Andres Carrasco Dols
    @GetMapping("/AllDraw")
    public String AllDraw(Model model) {
        //La sesion del usuario actual
        User user = (User) session.getAttribute( s: "user");

        // Crear una lista para almacenar información sobre el dibujo y su versión
        List<DrawWithVersionDTO> drawWithVersionList = drawService.getDraws(user.getId());

        // Agregar la lista de DTOs al modelo
        model.addAttribute( attributeName: "allDraws", drawWithVersionList);
        model.addAttribute( attributeName: "current_id",user.getId());
        return "AllDraw";
    }
```

El método **PostAllDraw** maneja las solicitudes POST dirigidas a "/AllDraw". Obtenemos el usuario actual de la sesión y el parámetro id que representa el id del dibujo seleccionado. Luego, utilizamos el servicio drawService para comprobar si el usuario tiene permitido enviar el dibujo a la basura , esto lo hacemos con **processAllDraw** que comprueba si es el creador de la imagen o si tiene permisos para realizar esta acción.

```java
    @PostMapping("/AllDraw")
    public String PostAllDraw(@RequestParam int id){
        //La sesion del usuario actual
        User user = (User) session.getAttribute( s: "user");
        boolean success = drawService.processAllDraw(id, user);
        if (success){ //Si no hay problema , redirige a la basura
            return "redirect:/TrashDraw";
        } //Si no te redirige a AllDraw
        return "redirect:/AllDraw";
    }
}
```

TrashDrawController

El controller TrashDrawController maneja la visualización y las operaciones sobre los dibujos que se encuentran en la papelera. El método GET muestra los dibujos en la papelera, y el método POST realiza acciones de borrar o restaurar.

El método **TrashDraw** maneja las solicitudes dirigidas a "/TrashDraw". Este método realiza la misma acción que el método get explicado del AllDraw, obtenemos la session actual y con la instancia drawServiece obtenemos una lista de objetos DrawWithVersionDTO que contendrán los dibujos que están puestos en la papelera.

```java
@Controller
public class TrashDrawController {
    2 usages
    @Autowired
    HttpSession session;

    2 usages
    @Autowired
    DrawService drawService;

    no usages  Andres Carrasco Dols
    @GetMapping("/TrashDraw")
    public String TrashDrawController(Model model) {
        //Obtenemos el usuario actual
        User user = (User) session.getAttribute( s: "user");

        // Crear una lista para almacenar información sobre el dibujo y su versión
        List<DrawWithVersionDTO> drawWithVersionList = drawService.getDrawsTrash(user.getId());

        // Agregar la lista de DTOs al modelo
        model.addAttribute( attributeName: "allDraws", drawWithVersionList);
        return "TrashDraw";
    }
```

El método **PostTrashDraw** maneja las solicitudes POST dirigidas a "/TrashDraw". Obtenemos el usuario actual de la sesión y los parámetros id que representa la id del dibujo y action que devuelve “delete” o “restore”. Luego, utiliza el método **deletOrRestoreTrashDraw** del servicio drawService para realizar operaciones correspondientes a la papelera de dibujos, como restaurar o eliminar un dibujo, también comprobará si tiene permiso para poder borrar este dibujo. Después de realizar la acción, redirige a la página "/TrashDraw".

```java
no usages  Andres Carrasco Dols
@PostMapping("/TrashDraw")
public String PostTrashDrawController(Model model, @RequestParam int id, @RequestParam String action) {
    //Obtenemos el usuario actual
    User user = (User) session.getAttribute( s: "user");

    // Llamamos al servicio para realizar las operaciones correspondientes
    drawService.deleteOrRestoreTrashDraw(id, action, user);
    return "redirect:/TrashDraw";
}
```

ViewDrawController

El controller ViewDrawController maneja la visualización de un dibujo específico y la acción de copiar el dibujo.

El método **ViewDraw** maneja las solicitudes GET dirigidas a "/ViewDraw". Obtenemos el usuario actual de la sesión y comprueba si el usuario tiene permisos para ver el dibujo identificado por drawId. Si tiene permisos, obtiene todas las versiones del dibujo utilizando el método **getAllVersionById** de la instancia **versionService** y agrega algunos atributos al modelo drawName que es nombre del dibujo que se muestra, allVersionsOfTheDraw para poder mostrarla/ver si elige otra versión y el id del dibujo que es drawId.

```
no usages  Andres Carrasco Dols
@GetMapping("/ViewDraw")
public String ViewDraw(Model model, @RequestParam int drawId , @RequestParam String drawName)  {
    //Obtenemos el usuario actual
    User user = (User) session.getAttribute( s: "user");
    //Comprobar si eres el propietrio y no esta en la basura.
    if (!drawService.canUserViewDraw(drawId, user)) {
        return "redirect:/AllDraw";
    }

    //Obtener todas las versiones.
    List<Version> allVersionsOfTheDraw = versionService.getAllVersionById(drawId);

    // Estos atributos se enviarán a la página JSP asociada para poder mostrarlos.
    model.addAttribute( attributeName: "drawName", drawName);
    model.addAttribute( attributeName: "allVersionsOfTheDraw",allVersionsOfTheDraw);
    model.addAttribute( attributeName: "drawId",drawId);
    // Mostrar la vista si tiene permisos
    return "ViewDraw";
```

El método **PostViewDraw** maneja las solicitudes POST dirigidas a "/ViewDraw". Al igual que en el método GET, comprueba si el usuario y el dibujo cumplen con las condiciones . Si cumplen, llama al servicio drawService que tiene el método **copiaDrawAndVersion** para copiar el dibujo y su versión asociada. Si la copia es exitosa, redirige a la vista "ViewDraw", de lo contrario, redirige a la página principal de dibujos ("/AllDraw").

```
@PostMapping("/ViewDraw")
public String PostViewDraw(Model model, @RequestParam String jsonData, @RequestParam int draw_Id) {
    //Obtenemos el usuario actual
    User user = (User) session.getAttribute( s: "user");
    //Comprobar si eres el usuario y no esta en la basura.
    if (!drawService.canUserViewDraw(draw_Id, user)) {
        System.out.println("Hola");
        return "redirect:/AllDraw";
    }

    //Metodo para guardar el dibujo y la version.
    String copiaDrawAndVersion = drawService.copiaDrawAndVersion(user, jsonData);
    if (copiaDrawAndVersion != null) {
        return "ViewDraw";
    }
    return "redirect:/AllDraw";
}
```

ModifyCanvasController

El controller ModifyCanvasController maneja la modificación de un dibujo en el lienzo.Muestra la última versión del dibujo y el usuario puede modificarla.

El método **ModifyCanvas"** maneja las solicitudes GET dirigidas a "/ModifyCanvas". Obtenemos el usuario actual, luego con in if validamos si tiene permisos para modificar el dibujo (comprobando si es propietario o si tiene los permisos necesarios), si no tiene permisos lo redirijo a AllDraw, luego recopilamos información sobre el dibujo seleccionado, como la id del dibujo, su visibilidad que puede ser pública o privada y sus figuras para poder modificarlas. Luego establecemos los atributos en el modelo y devuelve la vista "ModifyCanvas".

```
@GetMapping("/ModifyCanvas")
public String ModifyCanvas(Model model, @RequestParam String drawName,
                           @RequestParam int drawId) {
    //La sesion del usuario actual
    User user = (User) session.getAttribute( s: "user");
    //Comprobar si el usuario tiene permisos de modificar
    if (!drawService.validateDrawModifyAndTrash(drawId, user)) {
        return "redirect:/AllDraw";
    }
    // Obtener el dibujo por su ID
    Version selectedDraw = versionService.getVersionById(drawId);
    //Metodo para obtener la visibilidad
    boolean visibility = drawService.getVisibility(drawId);
    // Convertir la cadena de figuras a una cadena JSON
    String selectedFiguresJson = selectedDraw.getFigures();
    // Establecer los atributos en la solicitud.
    model.addAttribute( attributeName: "drawName", drawName);
    model.addAttribute( attributeName: "drawId", drawId);
    model.addAttribute( attributeName: "selectedFiguresJson", selectedFiguresJson);
    model.addAttribute( attributeName: "visibility", visibility);
    model.addAttribute( attributeName: "OwnerPropietary", drawService.propietaryDraw(drawId, user.getId()));
    return "ModifyCanvas";
}
```

El método **PostModifyCanvas** maneja las solicitudes POST dirigidas a "/ModifyCanvas". Al igual que en el método GET, valida los permisos del usuario y luego llamamos al servicio drawService para guardar la una nueva versión. Si hay un error, agrega un mensaje de error a la pantalla y vuelve a "ModifyCanvas". Si la modificación es exitosa, redirige a la página principal de dibujos ("/AllDraw").

```
no usages   Andres Carrasco Dols *
@PostMapping("/ModifyCanvas")
public String PostModifyCanvas(Model model,
                               @RequestParam int drawId,
                               @RequestParam String figures,
                                @RequestParam String visibility,
                                @RequestParam String drawName) {
    //Obtenemos el usuario actual
    User user = (User) session.getAttribute( s: "user");
    //Comprobar si el usuario tiene permisos de modificar el dibujo correspondiente.
    if (!drawService.validateDrawModifyAndTrash(drawId, user)) {
        return "redirect:/AllDraw";
    }
    // Llamamos al servicio para realizar las operaciones correspondientes
    String modifyCanvas = drawService.processUpdateDrawAndCreatVersion(model,
            drawName, drawId, figures, visibility, user);
    //Si no es un , es que hay un error y muestra ese mensaje.
    if (modifyCanvas != null){
        model.addAttribute( attributeName: "error", modifyCanvas);
        return "ModifyCanvas";
    }
    return "redirect:/AllDraw";
```

ShareDraw

El controller ShareDraw maneja las solicitudes de compartir y eliminar permisos (escritura/lectura) a los diferentes usuarios.

El método **ShareDraw** maneja las solicitudes GET dirigidas a "/ShareDraw" , Obtenemos la sesión actual del usuario, luego verificamos si el usuario es el propietario del dibujo y si el dibujo no está en la basura. Despues obtenemos una lista de todos los usuarios y agregamos esta lista junto con el ID del dibujo al modelo.

```
no usages  Andres Carrasco Dols *
@GetMapping("/ShareDraw")
public String ShareDraw(Model model, @RequestParam int drawId) {
    //La sesion del usuario actual
    User user = (User) session.getAttribute( s: "user");
    //Comprobar si eres el propietrio y no esta en la basura.
    if (!drawService.canUserShareDraw(drawId, user)) {
        return "redirect:/AllDraw";
    }
    //Obtenemos una lista de los usuarios.
    List<User> users = userService.allUsers(user.getId());
    // Agregar los usuarios y el ID del dibujo.
    model.addAttribute( attributeName: "users",users);
    model.addAttribute( attributeName: "drawId",drawId);
    return "ShareDraw";
}
```

El método **PostShareDraw** maneja las solicitudes POST dirigidas a "/ShareDraw". Después obtenemos el usuario actual,luego llamamos al servicio drawService para implementar el método de compartir dibujo (**ShareDraw**), y redirige a la página principal de dibujos después de compartir el dibujo.

```
no usages  Andres Carrasco Dols *
@PostMapping("/ShareDraw")
public String PostShareDraw(Model model,@RequestParam int drawId, @RequestParam int userId,
                            @RequestParam String permission){
    //La sesion del usuario actual
    User user = (User) session.getAttribute( s: "user");
    // Aqui implementamos el metodo de compartir dibujo
    drawService.ShareDraw(drawId, userId, permission, user);
    return "redirect:/AllDraw";
}
```

El método **PostDeletePermissions** maneja las solicitudes POST dirigidas a "/DeletePermissions". Otra vez obtenemos el usuario actual, verificamos si el usuario es el propietario del dibujo y si el dibujo no está en la basura. Luego llamamos al servicio **permissionService** para eliminar los permisos de un usuario sobre un dibujo. Por último redirige a la página principal “/AllDraw”.

```java
@PostMapping("/DeletePermissions")
public String PostDeletePermissions(Model model, @RequestParam int drawId, @RequestParam int userId) {
    //La sesion del usuario actual
    User user = (User) session.getAttribute( s: "user");
    //Comprobar si eres el propietrio y no esta en la basura.
    if (!drawService.canUserShareDraw(drawId, user)) {
        return "redirect:/AllDraw";
    }
    //Eliminar los permisos.
    permissionService.deletePermissionsUser(drawId, userId);
    return "redirect:/AllDraw";
}
```

Utils

Sirve como un repositorio para funciones que son comunes en el código y pueden ser utilizadas a lo largo de la aplicación.

```java
@Controller
public class ObjectCounter {
    3 usages  Andres Carrasco Dols *
    public int countFiguresInJson(String figures) {
        try {
            // Crear un objeto ObjectMapper
            ObjectMapper objectMapper = new ObjectMapper();
            // Convertir la cadena JSON en un árbol JSON.
            JsonNode jsonNode = objectMapper.readTree(figures);
            // Verificar si el nodo raíz del árbol es un array JSON.
            if (jsonNode.isArray()) {
                // Si es un array JSON, devolver la cantidad de elementos
                return jsonNode.size();
            } else {
                // Si no es un array JSON, devolver 0
                return 0;
            }
            // Si hay un error al procesar la cadena JSON
        } catch (JsonProcessingException e) {
            e.printStackTrace();
            return 0;
        }
    }
}
```

Creamos este método que utilizaremos a lo largo del código para contar el número de elementos en un array JSON representado como una cadena de texto. Retorna 0 si la cadena no es un array JSON o si hay algún error al procesar, sino devuelve el número de elementos del array.

Filter

Contiene clases relacionadas con la gestión de filtros en la aplicación web. Estas clases realizan la autenticación del usuario antes de que la solicitud llegue a los controladores.

SessionInterceptor verifica si el usuario está autenticado (si hay una sesión de usuario activa) y redirige a la página de inicio de sesión si no lo está. Además utiliza la anotación @Component permitiendo que Spring la detecte automáticamente y la gestione.

```java
@Component
public class SessionInterceptor implements HandlerInterceptor {
    1 usage
    @Autowired
    HttpSession session;
    no usages  Andres Carrasco Dols *
    @Override
    public boolean preHandle(
                            HttpServletRequest req,
                            HttpServletResponse resp,
                            Object handler) throws Exception{
        // Obtenemos el usuario actual
        User user = (User) session.getAttribute( s: "user");
        // Agregar el usuario al atributo.
        req.setAttribute( s: "user", user);

        // Verificar si el usuario está autenticado.
        if (user == null) {
            // Redirigir a la página de inicio de sesión si no está autenticado.
            resp.sendRedirect( s: "/login");
            return false;
        }
        // Continuar con la ejecución del controlador actual
        return true;
    }
```

Configuration configura el interceptor (SessionInterceptor) para aplicarse a rutas específicas en una aplicación Spring MVC. El interceptor realiza lógica de manejo de sesiones antes de que se procesen las solicitudes dirigidas a las rutas especificadas, asegurando que ciertos recursos solo están disponibles para usuarios autenticados.

```java
@org.springframework.context.annotation.Configuration
public class Configuration implements WebMvcConfigurer {
    1 usage
    @Autowired
    SessionInterceptor sessionInterceptor;
    no usages  Andres Carrasco Dols *
    @Override
    public void addInterceptors(InterceptorRegistry registry){
        registry.addInterceptor(sessionInterceptor)
                .addPathPatterns("/CanvasDraw","/AllDraw",
                        "/ViewDraw","/ModifyCanvas","/TrashDraw","/ShareDraw");
    }
}
```

Entities

Aquí se definen las clases que representan los objetos de la aplicación. Las entidades presentan información sobre dibujos, permisos, usuarios y versiones en la base de datos. Cada entidad tiene atributos específicos que capturan detalles relevantes para su funcionalidad en la aplicación. Estas clases reflejan la estructura de la base de datos y la lógica subyacente en la aplicación.

Class User

Guarda información del usuario, como su nombre de usuario, nombre y contraseña.

```
public class User {
    2 usages
    int id;
    3 usages
    String login;
    3 usages
    String name;
    3 usages
    String password;
```

Class Draw

Representa y almacena información sobre dibujos, incluyendo su nombre, propietario, fecha de creación, estado de visualización y si está en la papelera.

```
public class Draw {
    2 usages
    private int id;
    4 usages
    private String nameDraw;
    4 usages
    private int owner_id;
    3 usages
    private String creationDate;

    4 usages
    private boolean visualization;

    3 usages
    private boolean inTheTrash;
```

Class Version

Almacena distintas versiones de un dibujo que uniremos las dos tablas con el id_draw, registrando las figuras, el número de figuras, la fecha de modificación y el usuario que realizó la modificación.

```
public class Version {
    3 usages
    private int id;
    4 usages
    private int id_draw;
    4 usages
    private String figures;
    4 usages
    private int numFigures;
    3 usages
    private String modificationDate;
    4 usages
    private int id_user;
```

Class Permissions

Gestiona los permisos asociados a un dibujo para un usuario específico, indicando si el dibujo está en la papelera del usuario.

```
public class Permissions {
    3 usages
    private int id_draw;
    3 usages
    private int id_users;
    3 usages
    private String permissions;
    2 usages
    private String in_your_trash;
```

DTO

La clase **DrawWithVersionDTO** combina información de las clases Draw, Version, y Permissions. Es útil cuando necesitamos transmitir datos como entre el servicio y la capa de presentación.

```
public class DrawWithVersionDTO {
    2 usages
    private int id;
    2 usages
    private String nameDraw;
    2 usages
    private int owner_id;
    2 usages
    private String creationDate;
    2 usages
    private boolean visualization;
    2 usages
    private boolean inTheTrash;
    2 usages
    private String figures;
    2 usages
    private int numFigures;
    2 usages
    private String modificationDate;
    2 usages
    private String permissions;
```

Services

UserService

Este servicio actúa como intermediario para la lógica de negocio compartida entre el registro e inicio de sesión. Incluye métodos como la verificación de existencia de usuarios, la autenticación y el registro de ellos. Nos ayuda en la separación entre la lógica del servlet y la interacción con la base de datos a través del Repo.

Creamos una instancia **UserRepo** que la utilizaremos para interactuar con la base de datos que utilizamos para guardar los usuarios.

```java
@Service
public class UserService {
    5 usages
    @Autowired
    UserRepo userRepo;

    //Metodo para registrarse
    1 usage   Andres Carrasco Dols
    public String registerUser(String login, String name, String password) {
        if (existsLogin(login)) {
            return "El usuario ya existe.";
        } else if (password.length() < 5) {
            return "La contraseña debe tener al menos 5 caracteres.";
        } else if (!validUsername(login)) {
            return "El login solo permite letras y números.";
        } else if (!noSpaces(password)) {
            return "La contraseña no debe contener espacios.";
        } else {
            register(login, name, password);
            return null;
        }
    }
```

Este método implementa la funcionalidad de registro , lo utilizamos para validar las condiciones de registro , para empezar creamos un booleano **existLogin** para verificar si el usuario existe, utilizamos login porque es único.

Después, mediante condicionales (if), se comprueba si el usuario ya existe , que no se permiten caracteres especiales en el login y si la contraseña tiene al menos de 5 caracteres o si hay espacios en blanco . Si hay algún error, se muestra un mensaje, y si no hay errores, se registra el usuario y se redirige a la página de inicio de sesión.

- El método **register** se encarga de registrar un nuevo usuario. Antes de hacerlo, cifra la contraseña utilizando el método xifratMD5.

- El método **existLogin** comprueba si ya existe un usuario con el mismo nombre de usuario (login) , utilizó **equalsIgnoreCase** comparación de cadenas sin importar las diferencias de mayúsculas y minúsculas.

```
// Método que verifica si existe un usuario con el nombre de usuario proporcionado.
1 usage  Andres Carrasco Dols
public boolean existsLogin(String login) { return userRepo.existUser(login); }

// Método para registrar un nuevo usuario.
1 usage  Andres Carrasco Dols
public void register(String login, String name, String password) {
    User user = new User(login,name,password);
    user.setPassword(xifratMD5(user.getPassword()));
    userRepo.register(user);
}
```

Estos son los métodos que validan si el login no tiene caracteres especiales y si la contraseña tiene espacios.

```
// Métodos para validar login y contraseñas
1 usage  Andres Carrasco Dols
private boolean validUsername(String word) { return word.matches( regex: "[a-zA-Z0-9]+"); }
1 usage  Andres Carrasco Dols
private boolean noSpaces(String password) { return !password.contains(" "); }
```

- El método **userExists** comprueba si en BBDD contiene el nombre de usuario y contraseña que proporcionamos existe. La contraseña se cifra antes de realizar la verificación utilizando el método xifratMD5.

-Este método **xifratMD5** cifra la contraseña utilizando el algoritmo de cifrado MD5.

-El método **user** lo utilizó para obtener toda la información del usuario actual.

-El método para listar todos los usuarios excepto el usuario actual , lo utilizo para dar permisos a los usuarios sobre el dibujo.

```
// Método que comprueba la existencia de un usuario.
1 usage
public boolean userExists(String login, String password) { return userRepo.login(login,xifratMD5(password));

// Función que cifra una contraseña utilizando el algoritmo de hash MD5
2 usages
public String xifratMD5(String password) { return DigestUtils.md5Hex(password).toUpperCase(); }

//Metodo para obtener toda la informacion del usuario actual.
public User user(String login) { return userRepo.user(login);}

//Metodo para listar todos los usuarios
1 usage
public List<User> allUsers(int id_user) { return userRepo.allUsers(id_user);}
```

DrawService

Este código sirve para gestionar operaciones relacionadas con dibujos (Draw), sus versiones y permisos. A continuación, proporcionaré una explicación de las funciones presentes en el código, aparte el DrawService nos ayuda en la separación entre la lógica del controller y la interacción con la base de datos a través del Repo.

Aquí hay un resumen de los principales métodos y sus propósitos:

```
//Actualizar el dibujo a la papelera                                    ⚠ 7  ✓ 251
1 usage  Andres Carrasco Dols
public void updateTrash(int id, int id_user) { drawRepo.updateDraw(id, id_user); }
1 usage  Andres Carrasco Dols
public void updateYourTrash(int id, int id_user) {drawRepo.uodateYourTrash(id,id_user);}

//Obtener una lista de los dibujos en la papelera
1 usage  Andres Carrasco Dols
public List<DrawWithVersionDTO> getDrawsTrash(int id) { return drawRepo.getDrawsTrash(id); }

//Metodo para ver si el usuario tiene permisos lectura o escitura del dibujo
1 usage  Andres Carrasco Dols
public boolean hasPermissionsWriting(int id_draw, int id_user) {
    return drawRepo.hasPermissionsWriting(id_draw, id_user);
}
//Metodo para comprobar si tiene permisos.
3 usages  Andres Carrasco Dols
private boolean hasPermissions(int drawId, int id_user) { return drawRepo.hasPermissions(drawId, id_user)
//Metodo borrar el dibujo con sus versiones
1 usage  Andres Carrasco Dols
public void deleteDraw(int id_draw) { drawRepo.deleteDraw(id_draw); }

//Metodo restaurar el dibujo
1 usage  Andres Carrasco Dols
public void restoreDraw(int id_draw) { drawRepo.restoreDraw(id_draw); }
```

1. **updateTrash** y **updateYourTrash**: Actualizan el estado de un dibujo a la papelera.

2. **getDrawsTrash**: Obtiene una lista de dibujos en la papelera.

3. **hasPermissionsWriting**: Verifica si un usuario tiene permisos de escritura para un dibujo específico.

4. **hasPermissions** : comprueba si tiene permisos de lectura o escritura.

5. **deleteDraw** y **restoreDraw**: Eliminan o restauran un dibujo.

Mas metodos que utilizo en el drawService :

```
//Para obtener el propietario del dibujo                                    7  251
7 usages  Andres Carrasco Dols
public boolean propietaryDraw(int drawId, int id_user) { return drawRepo.propietaryDraw(drawId, id_user); }
//Metodo para obtener la visibilidad
2 usages  Andres Carrasco Dols
public boolean getVisibility(int drawId) { return drawRepo.getVisibility(drawId); }
//Metodo para cambiar la visibilidad
1 usage  Andres Carrasco Dols
public void updateVisibility(String newName, int drawId, String visibility) {
    drawRepo.updateVisibility(newName, drawId, convertToBoolean(visibility));
}
//Borrar permisos del usuario sobre el dibujo.
1 usage  new *
public void deletePermissionUser(int id, int id_user) {drawRepo.deletPermissionUser(id,id_user);}
//Pasar de string a booleano.
2 usages  new *
public static boolean convertToBoolean(String visibility) {return "public".equalsIgnoreCase(visibility);
//Metodo para comprobar si esta en la basura
5 usages  new *
public boolean trashDraw(int drawId) { return drawRepo.trashDraw(drawId); }

//Metodo para comporbar si esta en la basura tuya.
3 usages  Andres Carrasco Dols
public boolean in_your_trash(int drawId) {
    return drawRepo.in_your_trash(drawId);
}
```

6. **propietaryDraw**: Verifica si el usuario es el propietario de un dibujo.

7. **getVisibility** y **updateVisibility**: Obtiene y actualiza la visibilidad de un dibujo.

8. **deletePermissionUser**: Elimina los permisos de un usuario para un dibujo.

9. **convertToBoolean**: Convierte una cadena ("public") a un valor booleano, si no nos pasan public devolver false.

10. **trashDraw** y **in_your_trash**: Verifican si un dibujo está en la papelera general o en la papelera del usuario. Estos métodos los hago para cuando pase esto de que un usuario comparta el dibujo a otro usuario y ese usuario lo mande a la papelera, al usuario principal no se le quite el dibujo compartido sino que vaya a la papelera del usuario

Este es el método que utilizamos para guardar un dibujo y su versión. A éste método le pasamos toda la información necesaria, y una vez hecho comprobamos si el String figures está vacío , esto lo sabemos con el método **countFiguresJason** si es igual a cero es que no ha hecho ningún dibujo y retorna un mensaje de error.

Luego comprobamos si el nombre está vacío o no , si está vacío generamos uno aleatorio con el método **generateRandomName** , luego guardamos el dibujo con esta información (drawName, el id del usuario y la visualización si es pública o privada) una vez guardado obtenemos el id que se ha generado automáticamente en la base de datos , obtenemos esta id para incluirlo cuando guardamos las versiones , lo hacemos para identificar cuáles versiones tiene el dibujo. Si todo ha salido bien retorna null.

```
//Guradar el dibujo y su version
1 usage  Andres Carrasco Dols *
public String saveDrawAndVersion(User user, String nomImage, String visibility, String figures) {
    //Comprobar si las figuras están vacías
    if (objectCounter.countFiguresInJson(figures) == 0) {
        return "No se han dibujado figuras. Debes dibujar al menos una figura.";
    }

    //Si el nombre está vacío, genera uno aleatorio
    String drawName = nomImage.isEmpty() ? generateRandomName() : nomImage;

    // Guardar el dibujo
    Draw savedDraw = saveDraw(drawName, user.getId(), visibility);
    // Obtener la ID del dibujo recién creado
    int drawId = savedDraw.getId();

    // Guardar la versión
    versionService.saveVersion(drawId, figures, user.getId());

    // Devolver null indica que la operación se realizó con éxito, sin errores.
    return null;
}
```

Este es el método para generar un nombre aleatorio y guardar el dibujo:

```
// Método que genera un nombre aleatorio para una imagen.
3 usages  new *
public String generateRandomName() { return "image_" + UUID.randomUUID().toString();}

//Metodo para guardar el dibujo
2 usages  Andres Carrasco Dols
public Draw saveDraw(String newName, int owner_id, String visibility) {
    Draw draw = new Draw(newName, owner_id, convertToBoolean(visibility));
    return drawRepo.saveDraw(draw);
}
```

Estos métodos los utilizamos en el AllDrawController.

```
//Obtener una lista de los dibujos
1 usage  new *
public List<DrawWithVersionDTO> getDraws(int id_user) {return drawRepo.getDraws(id_user);}

//Metodo para comprobar a que papelera mandarlo
1 usage  Andres Carrasco Dols *
public boolean processAllDraw(int id, User user) {
    //Comprobar si eres el propietario y sis tienes permisos.
    boolean ownerProprietary = propietaryDraw(id, user.getId());
    boolean userPermission = hasPermissions(id, user.getId());
    //Si es el propietario o tiene permisos
    if (ownerProprietary || userPermission) {
        // Actualizar la imagen a Papelera si se cumple alguna de las condiciones.
        if (ownerProprietary) {
            updateTrash(id, user.getId());
        } else if (userPermission) {
            updateYourTrash(id, user.getId());
        }
        return true; // Indica que la actualización fue exitosa.
    } else {
        return false; // Indica que el usuario no es propietario ni tiene permisos.
    }
}
```

El método **getDraws** retorna una lista de los dibujos que puede ver cada usuario , si es el propietario los puede ver todos , si le comparten los dibujos, si es privado o publico , si está en la basura.

El método **processAllDraw**, determina si un usuario tiene permisos para mover un dibujo a la papelera. Comprueba si el usuario es el propietario del dibujo o si tiene permisos sobre él. Si es así, actualiza el estado del dibujo a "Papelera" y retorna true indicando que la operación fue exitosa. En caso contrario, retorna false, indicando que el usuario no tiene los permisos necesarios para llevar a cabo la operación.

El siguiente método **canUserViewDraw** lo utilizamos para comprobar si el usuario puede ver el dibujo, lo puede ver si cumple con estas condiciones, que sea el propietario y que su dibujo no esté en la basura , si tiene permisos tanto escritura o lectura y no esté en su basura o sea publico el dibujo.

```
//SI tiene permisos para ver el dibujo
2 usages  Andres Carrasco Dols
public boolean canUserViewDraw(int drawId, User user) {
    // Comprobar si eres el propietario del dibujo.
    boolean isTheOwner = propietaryDraw(drawId, user.getId());

    // Comprobamos que no esté en la basura.
    boolean trashDrawGeneral = trashDraw(drawId);

    //Comprobamos que tiene permisos.
    boolean userPermission = hasPermissions(drawId, user.getId());

    // Método para comprobar si está en la papelera del usuario.
    boolean inYourTrash = in_your_trash(drawId);

    boolean getVisibility = getVisibility(drawId);

    return isTheOwner && trashDrawGeneral || userPermission && inYourTrash || getVisibility;
    }
```

Este método, **copiaDrawAndVersion**, realiza una copia de un dibujo y su versión asociada. Genera un nuevo nombre para la copia, guarda el dibujo con ese nombre, y luego guarda una nueva versión del dibujo con los datos proporcionados , la visibilidad de la copia se establece como "false"

```
//Copiar el dibujo y su version
1 usage   Andres Carrasco Dols
public String copiaDrawAndVersion(User user, String jsonData) {
    String drawName = "Copia " + generateRandomName();
    boolean visibility = false;
    // Guardar el dibujo
    Draw savedDraw = saveDraw(drawName, user.getId(), String.valueOf(visibility));

    // Obtener la ID del dibujo recién creado
    int drawId = savedDraw.getId();

    // Guardar la versión
    versionService.saveVersion(drawId, jsonData, user.getId());
    return null;
}
```

Este método, **deleteOrRestoreTrashDraw**, maneja la acción de eliminar o restaurar un dibujo que se encuentra en la papelera. Se evalúan varias condiciones.

```
//Borrar la imagen o restaurarla
public void deleteOrRestoreTrashDraw(int id, String action, User user) {
    // Método para comprobar si eres el propietario del dibujo.
    boolean ownerPropietary = propietaryDraw(id, user.getId());
    // Método para comprobar si tienes permisos.
    boolean userPermission = hasPermissions(id, user.getId());
    // Método para comprobar que no esté en la basura general.
    boolean trashDraw = trashDraw(id);
    // Método para comprobar si está en la papelera del usuario.
    boolean inYourTrash = in_your_trash(id);
    if ("delete".equals(action)) {
        if (ownerPropietary) {
            if (!trashDraw) {
                    deleteDraw(id);
            }
        } else if (userPermission) {
            if (!inYourTrash) {
                    deletePermissionUser(id, user.getId());
                }
            }
    } else if ("restore".equals(action)) {
            if (ownerPropietary) {
                    restoreDraw(id);
        } else if (userPermission) {
                    permissionService.updatePermissionTrash(id);
        }
    }
}
```

Comprobamos si es el propietario y el dibujo está en la basura lo elimina, si no es el propietario pero tiene permisos de escritura, borra sus permisos sobre este dibujo , por último tenemos la versión de restaura que restaura el dibujo a AllDraw.

El método **validateDrawModifyAndTrash** comprueba si un usuario tiene acceso para modificar un dibujo. Evalúa si el usuario es el propietario y el dibujo está en la basura general, o si el usuario tiene permisos de escritura y el dibujo está en su papelera. Devuelve true si se cumplen estas condiciones, indicando acceso válido; de lo contrario, devuelve false, indicando que el usuario no tiene acceso y debe dirigirse a la página principal de dibujos (/AllDraw).

```java
//Validar que puede modificar.
2 usages  Andres Carrasco Dols
public boolean validateDrawModifyAndTrash(int drawId, User user) {
    // Método para comprobar si eres el propietario del dibujo.
    boolean ownerPropietary = propietaryDraw(drawId, user.getId());

    // Método para comprobar si tienes permisos de escritura.
    boolean userPermissionWriting = hasPermissionsWriting(drawId, user.getId());

    // Método para comprobar que no esté en la basura general.
    boolean trashDraw = trashDraw(drawId);

    // Método para comprobar si está en la papelera del usuario.
    boolean inYourTrash = in_your_trash(drawId);

    // Si eres el propietario y el dibujo no está en la basura general,
    // o tienes permisos de escritura y el dibujo no está en tu papelera,
    // redirige a la página principal de dibujos.
    if ((ownerPropietary && trashDraw) || (userPermissionWriting && inYourTrash)) {
        return true; // Acceso válido
    } else {
        return false; // No tienes acceso, redirige a /AllDraw
    }
}
```

Este método lo utilizamos en **shareDraw** , solo puede compartir una imagen si es el propietario y el dibujo no está en la papelera.

```java
//Comprobar si el usuario puede compartir.
2 usages  Andres Carrasco Dols
public boolean canUserShareDraw(int drawId, User user) {
    // Comprobar si eres el propietario del dibujo.
    boolean isTheOwner = propietaryDraw(drawId, user.getId());

    // Comprobamos que no esté en la basura.
    boolean trashDrawGeneral = trashDraw(drawId);

    return isTheOwner && trashDrawGeneral;
}
```

Este método **processUpdateDrawAndCreatVersion** lo utilizamos en modifyDrawController , en vez de subir un dibujo simplemente añadimos una nueva versión , comprobando que no esté vacío que no sea igual a la última versión hecha. Si ha hecho algún cambio en el nombre o visibilidad del dibujo lo actualiza.

```java
//Subir la version modificada
1 usage  Andres Carrasco Dols
public String processUpdateDrawAndCreatVersion(String drawName, int drawId, String figures, String visibili
    // Obtener la última versión
    List<Version> allVersionsOfTheDraw = versionService.getAllVersionById(drawId);
    Version lastVersion =allVersionsOfTheDraw.get(0);

    // Comprobar si el dibujo está vacío o es igual al anterior
    if (objectCounter.countFiguresInJson(figures) == 0 || lastVersion.getFigures().equals(figures)) {
        return "No se han dibujado figuras. Debes dibujar al menos una figura.";
    }

    // Si el nombre está vacío, genera uno aleatorio
    String newName = drawName.isEmpty() ? generateRandomName() : drawName;

    // Actualizar el dibujo
    updateVisibility(newName, drawId, visibility);

    // Guardar la nueva versión
    versionService.saveVersion(drawId, figures, user.getId());
    return null;
}
```

Este método lo utilizamos en el post de **ShareDrawController** para comprobar si es el usuario y si no está en basura otra vez , lo hago como medida de seguridad.
Si el usuario al que le vamos a dar permisos ya tiene pues lo actualiza y si no tiene los añade.

```java
//Metodo para compartir el dibujo
1 usage  Andres Carrasco Dols
public void ShareDraw(int drawId, int userId, String permission, User user) {
    // Comprobamos si el usuario es el propietario
    boolean ownerPropietary = propietaryDraw(drawId, user.getId());

    // Método para comprobar que no esté en la basura.
    boolean trashDraw = trashDraw(drawId);

    // Si el usuario es el propietario y el dibujo no está en la basura,
    // procedemos a compartir el dibujo.
    if (ownerPropietary && trashDraw) {
        // Si ya tiene permisos, los actualiza, sino los añade.
        boolean existPermission = permissionService.existpermissionUser(drawId, userId);

        if (permission.equals("R") || permission.equals("RW")) {
            if (existPermission) {
                permissionService.updatePermission(drawId, userId, permission);
            } else {
                permissionService.permissionUser(drawId, userId, permission);
            }
        }
    }
}
```

VersionService

La clase VersionService es un servicio de Spring (@Service) que gestiona operaciones relacionadas con las versiones de los dibujos. Sus funciones principales son:

```java
@Service
public class VersionService {
    3 usages
    @Autowired
    VersionRepo versionRepo;
    1 usage
    @Autowired
    ObjectCounter objectCounter;

    //Metodo para guardar el dibujo
    3 usages  Andres Carrasco Dols
    public void saveVersion(int drawId, String figures, int owner_id) {
        Version version = new Version(drawId,figures,objectCounter.countFiguresInJson(figures),owner_id);
        versionRepo.saveVersion(version);
    }

    //Metodo para obtener las figuras de la version del dibujo
    1 usage  Andres Carrasco Dols
    public Version getVersionById(int drawId) { return versionRepo.getVersionById(drawId); }

    //Metodo para obtener una lista de las versiones por su id.
    2 usages  Andres Carrasco Dols
    public List<Version> getAllVersionById(int drawId) { return versionRepo.getAllVersionById(drawId); }
}
```

-saveVersion: Guarda una nueva versión del dibujo, tomando como parámetros el ID del dibujo, las figuras en formato JSON, el ID del propietario y la cantidad de figuras. Utiliza un objeto Version y el repositorio versionRepo para realizar la operación de guardado.

-**getVersionById**: Obtiene una versión específica del dibujo según su ID. Utiliza el repositorio versionRepo para acceder a la base de datos y recuperar la información de la versión.

-**getAllVersionById**: Obtiene una lista de todas las versiones asociadas a un dibujo específico, identificado por su ID. Utiliza el repositorio versionRepo para acceder a la base de datos y recuperar la lista de versiones.

PermissionService

La clase PermissionService es un servicio de Spring (@Service) que gestiona operaciones relacionadas con los permisos de usuarios en dibujos. Sus funciones principales son:

```java
@Service
public class PermissionService {
    @Autowired
    PermissionRepo permissionRepo;
    //Metodo para crear permisos
    public void permissionUser(int drawId, int userId, String permission) {
        Permissions permissions = new Permissions(drawId,userId,permission);
        permissionRepo.permissionUser(permissions);
    }
    //Metodo para crear comprobar si tiene.
    public boolean existpermissionUser(int drawId, int userId) {
        return permissionRepo.ExistpermissionUser(drawId,userId);
    }
    //Metodo para actualizar permisos
    public void updatePermission(int drawId, int userId, String permission) {
        Permissions permissions = new Permissions(drawId,userId,permission);
        permissionRepo.updatePermission(permissions);
    }
    //Metodo para borrar permisos
    public void deletePermissionsUser(int drawId, int userId) {
        permissionRepo.deletePermissionsUser(drawId,userId);
    }

    //Metodo para actualizar la papelera de permisos.
    public void updatePermissionTrash(int id) {
        permissionRepo.updatePermissionTrash(id);
    }
}
```

-**permissionUser**: Crea y guarda nuevos permisos para un usuario para un dibujo determinado. Utiliza un objeto Permissions y el repositorio permissionRepo para realizar la operación de guardado.

-**existpermissionUser**: Comprueba si ya existen permisos para un usuario en un dibujo determinado. Utiliza el repositorio permissionRepo para verificar la existencia de los permisos.

-**updatePermission**: Actualiza los permisos de un usuario en un dibujo. Utiliza un objeto Permissions y el repositorio permissionRepo para realizar la operación de actualización.

-**deletePermissionsUser**: Elimina los permisos de un usuario en un dibujo específico. Utiliza el repositorio permissionRepo para realizar la operación de eliminación.

-**updatePermissionTrash**: Actualiza la papelera de permisos, probablemente para reflejar cambios en la base de datos relacionados con la papelera.

Repo/RepoImpl

UserRepo/UserRepoImpl

UserRepo actúa como intermediario entre UserService y la base de datos e implementa los métodos que interactúan con ella. En resumen , UserService se encarga de las operaciones relacionadas con los usuarios y UserRepo es responsable de las operaciones relacionadas con la base de datos.

```
3 usages  1 implementation  Andres Carrasco Dols
public interface UserRepo {
    1 usage  1 implementation  Andres Carrasco Dols
    boolean existUser(String login);

    1 usage  1 implementation  Andres Carrasco Dols
    void register(User user);

    1 usage  1 implementation  Andres Carrasco Dols
    boolean login(String login, String s);

    1 implementation  Andres Carrasco Dols
    User user(String login);

    1 usage  1 implementation  Andres Carrasco Dols
    List<User> allUsers(int idUser);
}
```

En esta parte del código explicaremos **UserRepoImpl** que implementa la interfaz UserRepo.

Realizaremos consultas a la base de datos por lo tanto utilizaremos **JdbcTemplate jdbcTemplate** que es un componente de Spring utilizado para simplificar el código de acceso a la base de datos mediante JDBC .

```
@Repository
public class UserRepoImpl implements UserRepo{
    @Autowired
    JdbcTemplate jdbcTemplate;

    //Metodo por si existe el usuario.
    @Override
    public boolean existUser(String login) {
        String sql = "SELECT COUNT(*) FROM user WHERE login = ?";
        int count = jdbcTemplate.queryForObject(sql, Integer.class, login);
        return count > 0;    }
    //Metodo para registrar usuario.
    @Override
    public void register(User user) {
        jdbcTemplate.update("INSERT INTO user (login,name,password) VALUES (?,?,?)",
                user.getLogin(),user.getName(),user.getPassword());
    }
    //Metodo para iniciar sesion.
    @Override
    public boolean login(String login, String password) {
        String sql = "SELECT COUNT(*) FROM user WHERE login = ? AND password = ?";
        int count = jdbcTemplate.queryForObject(sql, Integer.class, login, password);
        return count > 0;
    }
```

```java
//Metodo pra obtener informacion del usuario.
Andres Carrasco Dols
@Override
public User user(String login) {
    String sql = "SELECT * FROM user WHERE login = ?";
    return jdbcTemplate.queryForObject(sql, new BeanPropertyRowMapper<>(User.class), login);
}
//Metodo pra obtener una lista de usuarios.
1 usage  Andres Carrasco Dols
@Override
public List<User> allUsers(int idUser) {
    String sql = "SELECT * FROM user WHERE id <> ?";
    return jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(User.class), idUser);
}
```

-El método **existUser**: Verifica si un usuario con el nombre de usuario (login) proporcionado ya existe en la base de datos.

-El método **register**: Registra un nuevo usuario en la base de datos con el nombre de usuario, nombre y contraseña proporcionados.

-El método **login**: Verifica si un usuario con el nombre de usuario y contraseña proporcionados existe en la base de datos, devolviendo true si la combinación es válida.

-El método **user**: Recupera un usuario de la base de datos basado en el nombre de usuario.

-El método **allUsers**: Recupera todos los usuarios de la base de datos, excluyendo al usuario con el ID especificado.

BeanPropertyRowMapper: Se utiliza para mapear automáticamente las filas de resultados de la consulta a objetos Java. En este caso, se utiliza para mapear las filas de la tabla user a objetos de la clase Use

DrawRepo/DrawRepoImpl

DrawRepo actúa como intermediario entre DrawService y la base de datos e implementa los métodos que interactúan con ella. En resumen , DrawService se encarga de las operaciones relacionadas con los dibujos y DrawRepo es responsable de las operaciones relacionadas con la base de datos.

```java
public interface DrawRepo {
    Draw saveDraw(Draw draw);
    List<DrawWithVersionDTO> getDraws(int id_user);
    void updateDraw(int id, int id_user);
    List<DrawWithVersionDTO> getDrawsTrash(int id);
    boolean hasPermissionsWriting(int id_draw, int id_user);
    void deleteDraw(int id_draw);
    void restoreDraw(int id_draw);
    boolean propietaryDraw(int drawId, int id_user);
    boolean getVisibility(int drawId);
    void updateVisibility(String newName,int drawId, boolean visibility);
    boolean userCanSee(int drawId, int idUser);
    void uodateYourTrash(int id, int id_user);
    void deletPermissionUser(int id, int idUser);
    boolean trashDraw(int drawId);
    boolean in_your_trash(int drawId);
    boolean hasPermissions(int drawId, int idUser);
}
```

En esta parte del código explicaremos DrawRepoImpl que implementa la interfaz DrawRepo.

```java
//Metodo para guardar el dibujo(draw) y obtener la id generada en la BBDD.
@Override
public Draw saveDraw(Draw draw) {
    KeyHolder keyHolder = new GeneratedKeyHolder();

    jdbcTemplate.update(connection -> {
        PreparedStatement ps = connection.prepareStatement(
                "INSERT INTO draw (nameDraw, owner_id, creationDate,
                visualization) VALUES (?, ?, NOW(), ?)",
                Statement.RETURN_GENERATED_KEYS
        );

        ps.setString(1, draw.getNameDraw());
        ps.setInt(2, draw.getOwner_id());
        ps.setBoolean(3, draw.isVisualization());
        return ps;
    }, keyHolder);

    // Obtener el ID generado automáticamente
    Number key = keyHolder.getKey();

    if (key != null) {
        draw.setId(key.intValue());
    }
    return draw;
}
```

El método **saveDraw**: Guarda un dibujo en la base de datos, generando y recuperando su ID asignado automáticamente. Utiliza un KeyHolder para obtener el ID después de la inserción.

```java
//Metodo para mostrar las imagenes
1 usage  Andres Carrasco Dols *
@Override
public List<DrawWithVersionDTO> getDraws(int id_user) {
    String sql = "SELECT draw.*, version.numFigures AS numFigures, version.figures AS figures, " +
            "version.modificationDate AS modificationDate, permissions.permissions AS permissions " +
            "FROM draw JOIN version ON draw.id = version.id_draw LEFT JOIN permissions ON draw.id = " +
            "permissions.id_draw AND permissions.id_user = ? WHERE (draw.visualization = 1 OR draw.owner_id = ? " +
            "OR (permissions.permissions IN ('R', 'RW') AND permissions.id_user = ?)) \n" +
            "  AND draw.inTheTrash = 0 AND ((permissions.id_user IS NULL OR permissions.id_user <> ?) OR " +
            "(permissions.id_user = ? AND in_your_trash = false)) AND version.modificationDate = (SELECT " +
            "MAX(modificationDate) FROM version WHERE id_draw = draw.id) ORDER BY version.modificationDate " +
            "DESC;";

    List<DrawWithVersionDTO> allDrawWhithVersion = jdbcTemplate.query(sql,
            new BeanPropertyRowMapper<>(DrawWithVersionDTO.class),id_user, id_user, id_user, id_user, id_user);
    return allDrawWhithVersion;
}
```

El Método **getDraws**: Recupera una lista de dibujos con información relacionada con sus versiones y permisos basada en el ID de usuario proporcionado.Las condiciones son:

Que los dibujos que son visibles, pertenecen al usuario o tienen permisos específicos, que no estén en la papelera general,tampoco en la papelera del usuario y muestra solo la última versión de cada dibujo. La versión se ordena por la fecha de modificación de la versión en orden descendente.

```java
// Método para actualizar el campo inTheTrash en la tabla draw
1 usage  Andres Carrasco Dols
@Override
public void updateDraw(int id, int id_user) {
    // Aquí verificamos que el id_user es el propietario de la imagen antes de realizar la actualización
    String sql = "UPDATE draw SET inTheTrash = 1 WHERE id = ? AND owner_id = ?";
    jdbcTemplate.update(sql, id, id_user);
}

//Metodo para actualizar tu papelera
1 usage  Andres Carrasco Dols
@Override
public void updateYourTrash(int id, int id_user) {
//Aquí verificamos que el id_user es el propietario de la imagen antes de realizar la actualización
    String sql = "UPDATE permissions SET in_your_trash = 1 WHERE id_draw = ? AND id_user = ?";
    jdbcTemplate.update(sql, id, id_user);
}
```

Los métodos **updateDraw** y **updateYourTrash** actualizan el estado del campo inTheTrash en la tabla draw o in_your_trash en la tabla permissions, respectivamente.

La consulta **getDrawsTrash** busca dibujos con detalles sobre sus versiones y permisos asociados, asegurándose de incluir aquellos que están en la papelera general o en la papelera del usuario, según las condiciones especificadas en la consulta, en resumen recupera una lista de dibujos que se encuentran en la papelera.

```java
//Metodo para obetener lista de dibujos
1 usage  Andres Carrasco Dols *
@Override
public List<DrawWithVersionDTO> getDrawsTrash(int id) {
    String sql = "SELECT draw.*, MAX(version.figures) AS figures, " +
            "MAX(version.numFigures) AS numFigures, " +
            "MAX(version.modificationDate) AS modificationDate, " +
            "permissions.permissions " +
            "FROM draw " +
            "JOIN version ON draw.id = version.id_draw " +
            "LEFT JOIN permissions ON draw.id = permissions.id_draw AND permissions.id_user = ? " +
            "WHERE (draw.inTheTrash = true AND draw.owner_id = ?) OR (permissions.id_user = ? AND " +
            "permissions.in_your_trash = true) GROUP BY draw.id;";
    List<DrawWithVersionDTO> allDrawWhithVersion = jdbcTemplate.query(sql,
            new BeanPropertyRowMapper<>(DrawWithVersionDTO.class),id,id,id);
    return allDrawWhithVersion;
}
```

Estos métodos **hasPermissions** y **hasPermissionsWriting** verifica si un usuario tiene ciertos permisos (lectura o escritura) para un dibujo específico.

```java
//Si tiene permisos
1 usage  Andres Carrasco Dols *
@Override
public boolean hasPermissions(int drawId, int idUser) {
    String checkPermissionsSql = "SELECT COUNT(*) FROM permissions WHERE id_draw = ? AND id_user = ? " +
            "AND (permissions = 'RW' OR permissions = 'R')";
    int count = jdbcTemplate.queryForObject(checkPermissionsSql, Integer.class, drawId, idUser);
    return count > 0;
}
//SI tiene permisos de escritura
1 usage  Andres Carrasco Dols
@Override
public boolean hasPermissionsWriting(int id_draw, int id_user) {
    String checkPermissionsSql = "SELECT COUNT(*) FROM permissions WHERE id_draw = ? " +
            "AND id_user = ? AND permissions = 'RW'";
    int count = jdbcTemplate.queryForObject(checkPermissionsSql, Integer.class, id_draw, id_user);
    return count > 0;
}
```

El método **deleteDraw** elimina un dibujo y sus registros relacionados en las tablas permissions y version.

```java
//Borrar el dibujo.
1 usage  Andres Carrasco Dols
@Override
public void deleteDraw(int id_draw) {
    // Eliminar de la tabla permissions
    String deletePermissionsSql = "DELETE FROM permissions WHERE id_draw = ?";
    jdbcTemplate.update(deletePermissionsSql, id_draw);

    // Eliminar de la tabla version
    String deleteVersionSql = "DELETE FROM version WHERE id_draw = ?";
    jdbcTemplate.update(deleteVersionSql, id_draw);

    // Eliminar de la tabla draw
    String deleteDrawSql = "DELETE FROM draw WHERE id = ?";
    jdbcTemplate.update(deleteDrawSql, id_draw);
}
```

El método **deletPermissionUser** elimina los permisos de un usuario para un dibujo específico.

Los métodos **rashDraw** y **in_your_trash** verifica si un dibujo está en la basura general o en la basura del usuario, respectivamente.

```java
//Eliminar permisos usuario
@Override
public void deletPermissionUser(int id, int idUser) {
    // Eliminar de la tabla permissions
    String deletePermissionsSql = "DELETE FROM permissions WHERE id_draw = ? AND id_user = ?";
    jdbcTemplate.update(deletePermissionsSql, id, idUser);
}

//Verificar que esta en la basura general
@Override
public boolean trashDraw(int drawId) {
    // Consulta SQL para verificar si el dibujo está en la papelera en ambas tablas
    String sql = "SELECT COUNT(*) FROM draw WHERE id = ? AND inTheTrash = false";
    int count = jdbcTemplate.queryForObject(sql, Integer.class, drawId);
    return count > 0;
}

//Verificar sis esta en tu basura
@Override
public boolean in_your_trash(int drawId) {
    String sql = "SELECT COUNT(*) FROM permissions WHERE id_draw = ? AND in_your_trash = false";
    int count = jdbcTemplate.queryForObject(sql, Integer.class, drawId);
    return count > 0;
}
```

El método **restoreDraw**: Restaura un dibujo de la papelera.

El método **propietaryDraw**: Verifica si un usuario es el propietario del dibujo.

Los método **getVisibility** y **updateVisibility** obtiene y actualiza la visibilidad de un dibujo.

```java
//Restaurar el dibujo
@Override
public void restoreDraw(int id_draw) {
    String sql = "UPDATE draw SET inTheTrash = 0 WHERE id = ?";
    jdbcTemplate.update(sql, id_draw);
}

//Comprobar el propietario del dibujo
@Override
public boolean propietaryDraw(int drawId, int idUser) {
    String checkUserDraw = "SELECT COUNT(*) FROM draw WHERE id = ? AND owner_id = ?";
    int count = jdbcTemplate.queryForObject(checkUserDraw, Integer.class, drawId, idUser);
    return count > 0;
}

//Obtener la visibilidad
@Override
public boolean getVisibility(int drawId) {
    String checkVisibilityDraw = "SELECT visualization FROM draw WHERE id = ?";
    return jdbcTemplate.queryForObject(checkVisibilityDraw, boolean.class, drawId);
     }
//Actualizar la visibilidad
@Override
public void updateVisibility(String newName,int drawId, boolean visibility) {
    String sql = "UPDATE draw SET nameDraw = ?, visualization = ? WHERE id = ?";
    jdbcTemplate.update(sql, newName, visibility, drawId);
}
```

PermissionRepo/PermissionRepoImpl

PermissionRepo actúa como intermediario entre PermissionService y la base de datos e implementa los métodos que interactúan con ella. En resumen , PermissionService se encarga de las operaciones relacionadas con los permisos de los usuarios y PermissionRepo es responsable de las operaciones relacionadas con la base de datos.

```java
3 usages  1 implementation  Andres Carrasco Dols
public interface PermissionRepo {
    1 usage  1 implementation  Andres Carrasco Dols
    void permissionUser(Permissions permissions);

    1 usage  1 implementation  Andres Carrasco Dols
    void updatePermission(Permissions permissions);

    1 usage  1 implementation  Andres Carrasco Dols
    boolean ExistpermissionUser(int drawId, int userId);

    1 usage  1 implementation  Andres Carrasco Dols
    void deletePermissionsUser(int drawId, int userId);

    1 usage  1 implementation  Andres Carrasco Dols
    void updatePermissionTrash(int id);
}
```

En esta parte del código explicaremos PermissionRepoImpl que implementa la interfaz PermissionRepo.

Realizaremos consultas a la base de datos por lo tanto utilizaremos **JdbcTemplate jdbcTemplate** que es un componente de Spring utilizado para simplificar el código de acceso a la base de datos mediante JDBC .

```java
@Repository
public class PermissionRepoImpl implements PermissionRepo{
    @Autowired
    JdbcTemplate jdbcTemplate;

    //Metodo para dar permisos
    @Override
    public void permissionUser(Permissions permissions) {
        jdbcTemplate.update("INSERT INTO permissions (id_draw,id_user,permissions) VALUES (?,?,?)",
                permissions.getId_draw(),permissions.getId_users(),permissions.getPermissions());
    }
    //Metodo para actualizar permisos.
    @Override
    public void updatePermission(Permissions permissions) {
        String updateSql = "UPDATE permissions SET permissions = ? WHERE id_draw = ? AND id_user = ?";
        jdbcTemplate.update(updateSql, permissions.getPermissions(), permissions.getId_draw(), permissions.getId_users());
    }
    //Metod para comprobar si tiene permisos
    @Override
    public boolean ExistpermissionUser(int drawId, int userId) {
        String selectSql = "SELECT COUNT(*) FROM permissions WHERE id_draw = ? AND id_user = ?";
        int count = jdbcTemplate.queryForObject(selectSql, Integer.class, drawId, userId);
        return count > 0;    }
```

El método **permissionUser** guarda nuevos permisos en la base de datos. Toma un objeto Permissions que contiene información sobre el dibujo (id_draw), el usuario (id_user), y los permisos asignados. Realiza una inserción en la tabla permissions.

El método **updatePermission**: Actualiza los permisos existentes en la base de datos. Toma un objeto Permissions con la misma información que permissionUser y actualiza los permisos en la tabla permissions donde el id_draw y id_user coinciden con los proporcionados.

El método **ExistpermissionUser**: Verifica si existen permisos para un dibujo (id_draw) y un usuario (id_user) específicos. Devuelve true si hay al menos un registro que coincide con esas condiciones en la tabla permissions, indicando que existen permisos.

```java
    //Metodo para borrar permisos
    @Override
    public void deletePermissionsUser(int drawId, int userId) {
        String deleteSql = "DELETE FROM permissions WHERE id_draw = ? AND id_user = ?";
        jdbcTemplate.update(deleteSql, drawId, userId);
    }
    //Metodo para actalizar papelera
    @Override
    public void updatePermissionTrash(int id) {
        String updateTrash = "UPDATE permissions SET in_your_trash = 0 WHERE id_draw = ?";
        jdbcTemplate.update(updateTrash,id);
    }
}
```

El método **deletePermissionsUser**: Elimina los permisos de un usuario específico (id_user) para un dibujo específico (id_draw). Realiza un DELETE en la tabla permissions basado en las condiciones proporcionadas.

El método **updatePermissionTrash**: Actualiza la columna in_your_trash en la tabla permissions. Cuando se llama con un id_draw, establece in_your_trash en 0 (false) para ese dibujo en la tabla permissions, indicando que el dibujo ya no está en la papelera del usuario.

VersionRepo/VersionRepoImpl

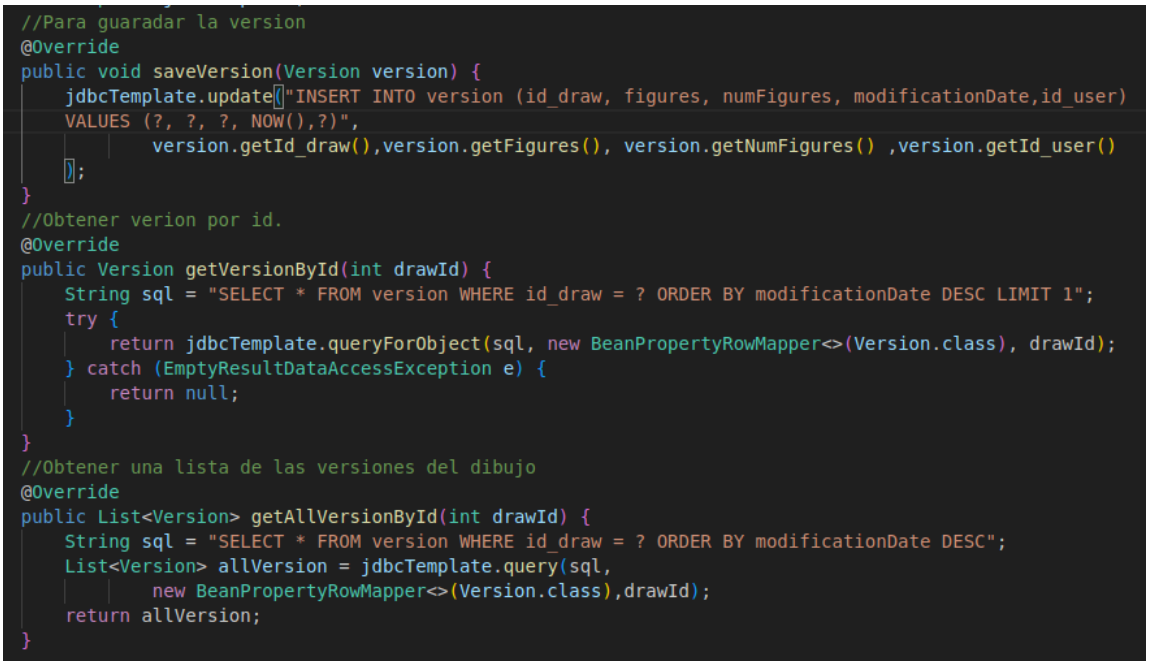
VersionRepo actúa como intermediario entre VersionService y la base de datos e implementa los métodos que interactúan con ella. En resumen , VersionService se encarga de las operaciones relacionadas con las versiones del dibujo y VersionRepo es responsable de las operaciones relacionadas con la base de datos.

```
3 usages  1 implementation  Andres Carrasco Dols
public interface VersionRepo {
    1 usage  1 implementation  Andres Carrasco Dols
    void saveVersion(Version version);

    1 usage  1 implementation  Andres Carrasco Dols
    Version getVersionById(int drawId);

    1 usage  1 implementation  Andres Carrasco Dols
    List<Version> getAllVersionById(int drawId);
}
```

En esta parte del código explicaremos VersionRepoImpl que implementa la interfaz VersionRepo.



```
//Para guaradar la version
@Override
public void saveVersion(Version version) {
    jdbcTemplate.update("INSERT INTO version (id_draw, figures, numFigures, modificationDate,id_user)
    VALUES (?, ?, ?, NOW(),?)",
            version.getId_draw(),version.getFigures(), version.getNumFigures() ,version.getId_user()
    );
}
//Obtener verion por id.
@Override
public Version getVersionById(int drawId) {
    String sql = "SELECT * FROM version WHERE id_draw = ? ORDER BY modificationDate DESC LIMIT 1";
    try {
        return jdbcTemplate.queryForObject(sql, new BeanPropertyRowMapper<>(Version.class), drawId);
    } catch (EmptyResultDataAccessException e) {
        return null;
    }
}
//Obtener una lista de las versiones del dibujo
@Override
public List<Version> getAllVersionById(int drawId) {
    String sql = "SELECT * FROM version WHERE id_draw = ? ORDER BY modificationDate DESC";
    List<Version> allVersion = jdbcTemplate.query(sql,
            new BeanPropertyRowMapper<>(Version.class),drawId);
    return allVersion;
}
```

El método **saveVersion** guarda una nueva versión en la base de datos. Toma un objeto Version que contiene información sobre el dibujo (id_draw), las figuras (figures), la cantidad de figuras (numFigures), la fecha de modificación (modificationDate), y el usuario (id_user). Realiza una inserción en la tabla version.

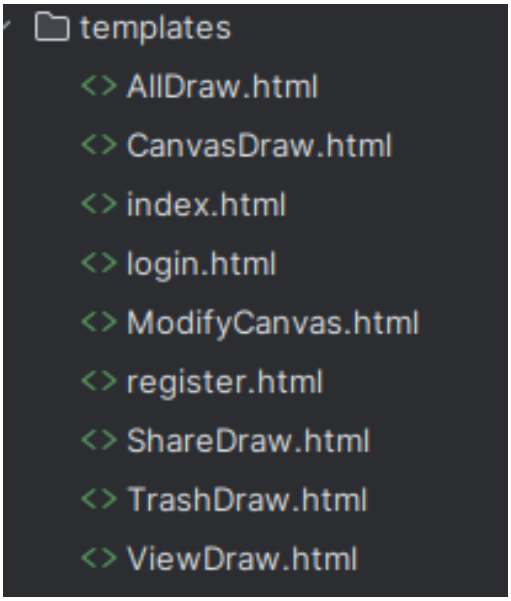
El método **getVersionById** obtiene la última versión de un dibujo específico (id_draw). Realiza una consulta SQL que selecciona todas las columnas de la tabla version donde el id_draw coincide y ordena los resultados por modificationDate de manera descendente. Devuelve el resultado utilizando BeanPropertyRowMapper para mapear las filas de la base de datos a objetos Version. Si no hay resultados, devuelve null.

El método **getAllVersionById** obtiene una lista de todas las versiones de un dibujo específico (id_draw). Realiza una consulta SQL similar a getVersionById, pero sin la limitación de una sola fila. Devuelve una lista de objetos Version ordenados por modificationDate de manera descendente.

EXPLICACIÓN DEL CÓDIGO (FRONTEND)

Html

Dentro de la carpeta Template nos encontramos con las páginas html con las que interactúa el cliente. Estas son las páginas de nuestra aplicación web :



Css

En esta carpeta se definirá los estilos de nuestras páginas html.Tengo cuatro ficheros html :

-**styleForm.css** : Aquí definir los estilos del formulario de las páginas Register y Login.

-**DrawTable.css** : Aquí definir los estilos de las tablas de las páginas del UserDraw y AllDraw.

-**CanvasDrawStyles.css** : Aquí defino el estilo de la página principal del programa.

-**ViewDraw.css** : Aquí definir los estilos de la página View.

-**TrahsDraw.css** : Aquí definir los estilos de la página web de basura.

JAVASCRIPT

Como he dicho anteriormente necesitamos código JavaScript y es lo que explicaremos ahora. He declarado los scrips que sean type module por que es una forma de declarar que un archivo JavaScript se comporta como un módulo, aprovechando así las características de modularidad y encapsulación proporcionadas por el sistema de módulos de JavaScript. Esto permitirá hacer imports y exports desde diferentes códigos javaScrip

Explicaré las cosas que he añadido a partir del código anterior de la primera parte. código

He creado una archivo js llamado draw al que he definido las variables canvas, ctx y draw , he hecho esto para exportar esta parte e importarlo donde la necesite , así organizar mejor el codigo código.

-Canvas(canvas y ctx) : Representa el área visual donde se llevará a cabo el dibujo, el ctx es el contexto 2D que permite manipular la representación gráfica en el lienzo.

-La función **draw** se encarga de dibujar una figura en el canvas(ctx) según la información proporcionada en el objeto figure, es decir , toma un objeto figure y utiliza las propiedades de ese objeto para dibujar la figura correspondiente en el canvas.

Con esta line importo esto : import { canvas, ctx ,draw } from './draw.js';

También he modificado la función render realizaba que se encarga de limpiar, mostrar y dibujar figuras en un lienzo, incluyendo la capacidad de dibujar líneas en tiempo real si el usuario está actualmente trazando una línea.

El cambio que he hecho es cambiar el onclick que tenía , cuando pones que sea type module no te deja poner onclick por lo tanto lo he cambiado a la imagen de abajo.

```
figures.forEach((figure, i) => {
    logs.innerHTML += `<li>Tipo: ${figure.type} - Color: ${figure.color}
    <button id="${i}" onclick="removeFigure(${i})" class="Delete-Button">Eliminar</button>
    </li>`;
    draw(figure);
});
```

```
figures.forEach((figure, i) => {
    const listItem = document.createElement("li");
    listItem.innerHTML = `Tipo: ${figure.type} - Color: ${figure.color}
    <button data-index="${i}" class="Delete-Button">Eliminar</button>
    <button data-index="${i}" class="Modify-Button">Modify</button>`;

    const deleteButton = listItem.querySelector(".Delete-Button");
    deleteButton.addEventListener("click", () => {
        removeFigure(i);
    });

    const modifyButton = listItem.querySelector(".Modify-Button");
    modifyButton.addEventListener("click", () => {
        modifyFigure(i);
    });
    logs.appendChild(listItem);
    draw(figure);
});
```

También he añadido , como también la imagen se puede poner pública o privada he añadido este código para detectar si cambia la visibilidad.

```
// Declaración de la variable visibility
let visibility = "private";

// Event listener para el cambio en los elementos radio
document.querySelectorAll('.visibility').forEach(function (radio) {
   radio.addEventListener('change', function () {
   setVisibility(this.value);
     });
});

// Función para establecer la visibilidad
function setVisibility(value) {
    visibility = value;
    localStorage.setItem('visibility', value);
}
```

Como uno de los requisitos he añadido que se pueda modificar la figura en tiempo real, si apreta el botón modify puede modificar la figura.

```
// Event listener para el cambio en los campos del formulario
colorInput.addEventListener('input', () => modifyFigure(figures.length - 1));
sizeInput.addEventListener('input', () => modifyFigure(figures.length - 1));
fillFigureCheckbox.addEventListener('input', () => modifyFigure(figures.length - 1));

// Función para actualizar la vista de una figura específica
const modifyFigure = (index) => {
    const figure = figures[index];

    // Obtén los elementos del formulario
    const colorInput = document.getElementById("color");
    const sizeInput = document.getElementById("size");
    const fillCheckbox = document.getElementById("fillFigure");

    // Actualiza los valores de la figura con los nuevos valores del formulario
    figure.color = colorInput.value;
    figure.size = sizeInput.value;
    figure.filled = fillCheckbox.checked;

    // Borra el lienzo y vuelve a dibujar todas las figuras
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    figures.forEach(draw);

    // Dibuja la figura modificada
    draw(figure);
};
```

Otro cambio es como pedía en la práctica he añadido un localstorage que significa que guarde la última configuración , la última figura , su color, tamaño y si está relleno o no.

```
// Agregamos event listeners para detectar cambios y guardar en localStorage
figureSelect.addEventListener('change', function () {
    localStorage.setItem('figure', figureSelect.value);
});

colorInput.addEventListener('input', function () {
    localStorage.setItem('color', colorInput.value);
});

sizeInput.addEventListener('input', function () {
    localStorage.setItem('size', sizeInput.value);
});

fillFigureCheckbox.addEventListener('change', function () {
    localStorage.setItem('fillFigure', fillFigureCheckbox.checked);
});
```

Al cargar la página se muestran los cambios ,

```
//Al cargar la pagina
window.addEventListener('load', () => {
    // Recuperamos valores si existen
    const storedFigure = localStorage.getItem('figure');
    const storedColor = localStorage.getItem('color');
    const storedSize = localStorage.getItem('size');
    const storedFillFigure = localStorage.getItem('fillFigure');
    const storedVisibility = localStorage.getItem('visibility');

    // Establece los valores recuperados en los elementos del formulario
    if (storedFigure) {
        figureSelect.value = storedFigure;
    }
    if (storedColor) {
        colorInput.value = storedColor;
    }
    if (storedSize) {
        sizeInput.value = storedSize;
    }
    if (storedFillFigure) {
        fillFigureCheckbox.checked = storedFillFigure === 'true';
    }
});
```

Como otro requisito era utilizar **fetch** que es es una herramienta poderosa y flexible para trabajar con solicitudes y respuestas HTTP en JavaScript. Para mandarlo directamente a Post sin necesidad de cogerlo con los inputs en el html.

```
// Función para guardar figuras
async function saveFigures() {
    try {
        // Convierte el array "figures" a una cadena JSON
        var figuresJSON = JSON.stringify(figures);
        const formData = new FormData();

        //Obtener el id del dibujo
        const drawId = document.getElementById("drawId").value;
        formData.append("drawId", drawId);

        // Adjuntar las figuras en formato JSON
        formData.append("figures", figuresJSON);

        // Obtener el valor de visibilidad seleccionado
        const selectedVisibility = document.querySelector('input[name="type_visibility"]:checked').value;
        // Actualizar el valor del campo oculto de visibilidad
        document.getElementById("visibility").value = selectedVisibility;
        formData.append("visibility", selectedVisibility);

        // Obtener el valor del nombre
        const imageName = document.getElementById("drawName").value;
        formData.append("drawName", imageName);
```

Creamos un objeto FormData para facilitar el envío de datos al servidor, en ese objeto añadimos las figuras , la visibilidad y el nombre del dibujo.

Luego utilizamos la función fetch para realizar una solicitud POST a la ruta '/ModifyCanvas'. Envíamos el objeto FormData que contiene todos los datos que queremos enviar. También manejamos los posibles errores.

```
    // Enviar las figuras y la imagen al servidor
    const response = await fetch('/ModifyCanvas', {
        method: 'POST',
        body: formData,
    });
    // Manejar la respuesta del servidor
    if (response.ok) {
        alert("¡Se ha modificado correctamente!");
    } else {
        alert("Error al modificar. Tiene que haber una figura y no se puede subir la misma version");
    }
} catch (error) {
        alert("Error de red. Por favor, inténtalo de nuevo más tarde.");
}
```

Esta puesto que se envíe cuando aprete el boton de enviar , pero aparte he añadido un contador que si pasan treinta segundo si pintar lo envíe directamente, es como un guardado automático.

```
// Función para enviar automáticamente las figuras al servidor después de 30 segundos de inactividad
function submitModify() {
    if (shouldSubmitAutomatically) {
        saveFigures();
        shouldSubmitAutomatically = false;
    }
}
```

Para compartir permisos he creado una tabla para elegir qué permisos le quiere dar , por eso he implementado este código que observa el cambio en un elemento de selección HTML ("permisos") y actualiza un campo de entrada oculto ("permission") en consecuencia.

```
document.addEventListener('DOMContentLoaded', function() {
    // Se intenta obtener el elemento con el id "permisos"
    var select = document.getElementById('permisos');
    //Se obtiene el valor oculto.
    var hiddenInput = document.getElementById('permission');
    //Comprueba si no es null.
    if (select) {
    //Cuando cambia se actualiza el valor del input oculto.
    select.addEventListener('change', function() {
        hiddenInput.value = select.value;
    });
    } else {
    //Si no ha encontrado.
        console.error('El elemento con id "permisos" no fue encontrado.');
  }
});
```

También para borrar un dibujo he implementado un modal para confirmar la eliminación del dibujo, Espera a que el contenido del documento HTML (DOM) esté completamente cargado.

Asocia funciones a eventos en elementos HTML. Los botones de eliminación en la tabla y los botones en un modal activan acciones como mostrar el modo de confirmación, cerrar el modal o eliminar un dibujo.
Define funciones para mostrar el modo de confirmación, cerrar el modal y realizar la eliminación del dibujo. La eliminación se realiza al enviar un formulario de eliminación, y luego se cierra el modal.

```
document.addEventListener('DOMContentLoaded', function () {
    var deleteButtons = document.querySelectorAll('.delete');
    var modalDeleteButton = document.querySelector('.modal-delete');
    var modalCancelButton = document.querySelector('.modal-cancel');
    var deleteForm = document.getElementById('deleteForm');
    var deleteModal = document.getElementById('deleteModal');

    // Agregar un listener a cada botón de eliminar en la tabla
    deleteButtons.forEach(function (deleteButton) {
        deleteButton.addEventListener('click', function () {
            confirmDelete();
        });
    });

    // Agregar un listener al botón de eliminar en el modal
    modalDeleteButton.addEventListener('click', function () {
        deleteDraw();
    });

    // Agregar un listener al botón de cancelar en el modal
    modalCancelButton.addEventListener('click', function () {
        closeModal();
    });

    // Función para mostrar el modal de confirmación
    function confirmDelete() {
        deleteModal.style.display = 'block';
    }
```

```
    // Agregar un listener al botón de cancelar en el modal
    modalCancelButton.addEventListener('click', function () {
        closeModal();
    });

    // Función para mostrar el modal de confirmación
    function confirmDelete() {
        deleteModal.style.display = 'block';
    }

    // Función para cerrar el modal
    function closeModal() {
        deleteModal.style.display = 'none';
    }

    // Función para realizar la eliminación
    function deleteDraw() {
        deleteForm.submit();
        closeModal();
    }
```

DIFICULTADES Y SOLUCIONES

A lo largo del proyecto me he encontrado con diferentes problemas que no podía resolver, algunos los he podido resolver mirándolo con tiempo, también he encontrado problemas que por mí solo no he podido resolver y he pedido ayuda a mis compañeros que me han dado consejos y me han ayudado a resolverlos, también buscando por internet para encontrar posibles soluciones e intentar adaptarlo a mi código.

Por último también he pedido cuando he visto que no podía resolver por mí ayuda al profesor para que me diga consejos y me indique el camino para seguir adelante.