

1º Práctica Desarrollo web en entorno Servidor 2022/23
2º CFGS Desarrollo de aplicaciones web
Dámaso Simal Castro

Polygon

Objetivo de la práctica

Realizar una aplicación que permita que los usuarios puedan dibujar y almacenar figuras poligonales de diverso grado de complejidad. También será posible recuperar los polígonos propios y ajenos y también podrá borrar los que haya generado el propio usuario propietario del mismo.

Para llevar a cabo este objetivo, usaremos una aplicación Maven multipágina escrita en Java, usando servlets en un servidor Tomcat montado con Docker. Las vistas de la página se generarán en archivos JSP, los cuales son HTML con capacidades de interpretar JSTL.

Para llevar a cabo la tarea usaremos el patrón de diseño MVC el cual permitirá realizar un desarrollo en tres capas, separando la capa de presentación, la de negocio y la de datos.

Descripción de las herramientas que usaremos:

MVC

El patrón MVC (Modelo Vista Controlador) es uno de los patrones de diseño usados para realizar una encapsulación y aislamiento, separando los datos del usuario para garantizar el aislamiento y la seguridad, solo permitiendo acceder a los datos mediante los métodos autorizados por el controlador.

En este caso los controladores, implementando servlets son la capa de presentación, ya que muestra las vistas montadas en los JSP mediante los RequestDispatcher, los cuales montan la página en la dirección especificada en el mismo para obtener información del usuario a través de los formularios que conforman el archivo JSP.

La capa de negocio se realiza mediante los servicios, clases que implementan las funciones que transmiten la información recibida en los controladores a la base de datos, para su almacenado y las funciones para recuperar de la base de datos la información solicitada por el controlador.

Finalmente la capa de datos es manejada por los DAO (Data Access Object) las cuales son interfaces que contienen los métodos de almacenamiento, recuperación y eliminación de datos en la base de datos. Estos son solicitados por las funciones en los servicios ya que el objetivo de este patrón de diseño es que el usuario nunca tenga acceso a los datos directamente desde el controlador

Docker

Docker es un proyecto de código abierto que permite un despliegue automatizado en contenedores de software generando una capa de abstracción y automatización para desplegar aplicaciones independientemente del sistema operativo

Un contenedor Docker simula los servicios y procesos necesario de un sistema operativo para ejecutar las distintas aplicaciones montadas en el mismo, reduciendo la necesidad y el consumo de una máquina virtual, permitiendo un número mayor de contenedores sin tener tanto consumo como el sistema completo.

Tomcat

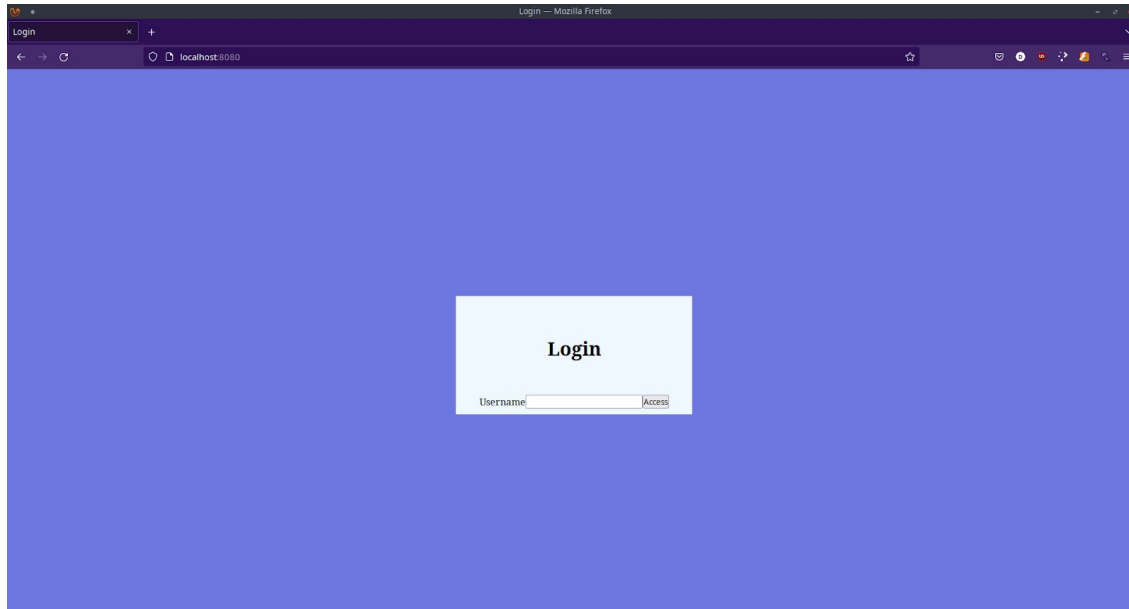
Contenedor de servlets desarrollado por Apache, implementa especificaciones de servlets y JSP (JavaServer Pages). Funciona como servidor web para entornos de desarrollo con pocos requisitos de velocidad

Maven

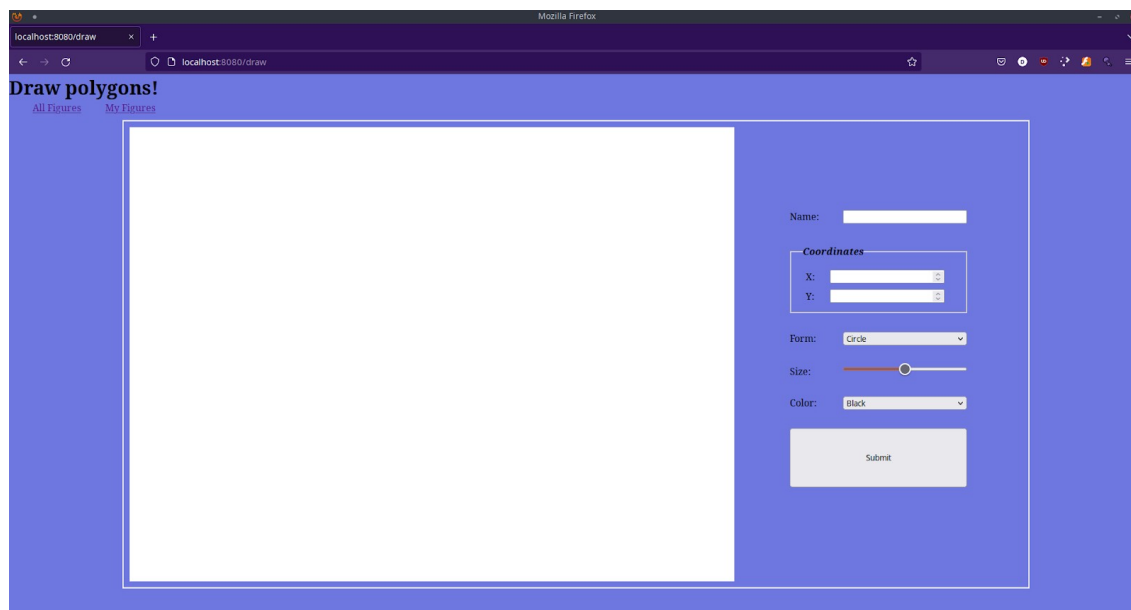
Es una herramienta de gestión y construcción de proyectos Java usando un POM (Project Object Model) escrito en XML para describir el proyecto, sus dependencias, módulos y componentes externos, así como su orden de construcción. También prepara la compilación y el empaquetado del proyecto.

Descripción de la aplicación:

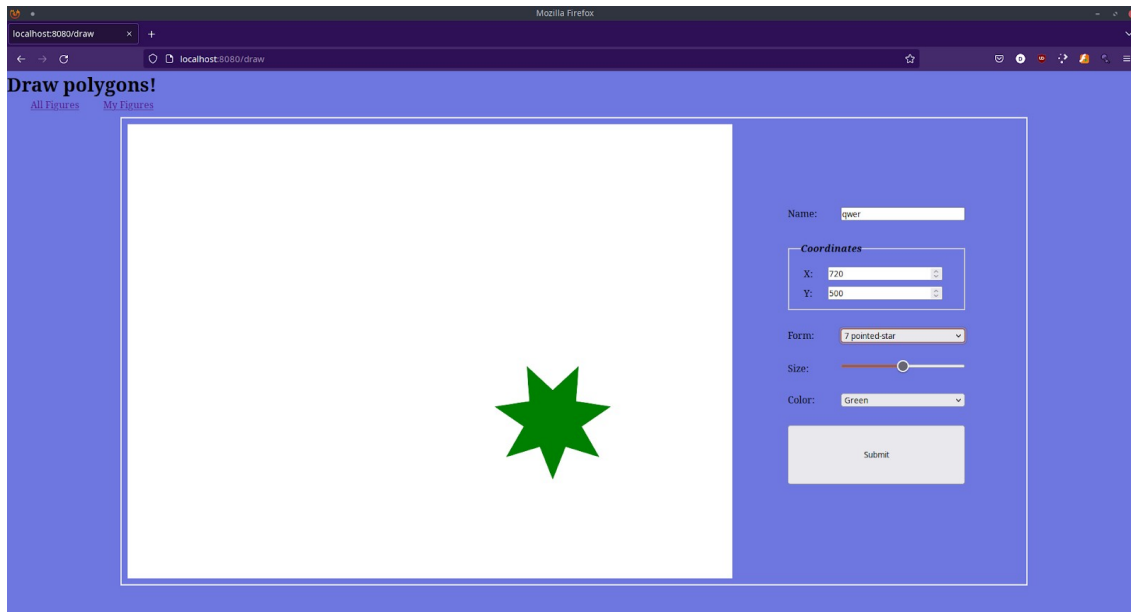
El proyecto tendrá una página principal, el Login, en el cual se solicitará al usuario un nombre para identificarle dentro del servidor:



Tras insertar un nombre, le mandará a la página de dibujo, donde se encontrará el elemento canvas y un formulario que le solicitará los datos de la figura a representar. Los valores son requeridos, siendo el nombre el único omitible, ya que la aplicación asignará un nombre único.

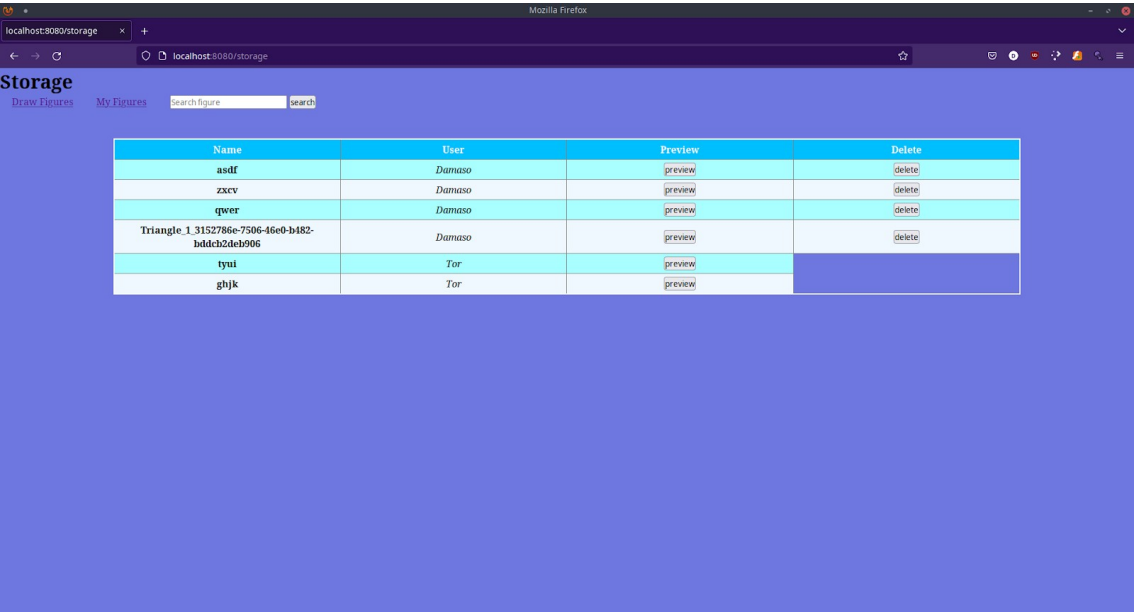


Cada cambio en el formulario actualizará la presentación del canvas para mostrar el estado actual de la figura a crear. Una vez encontrado un diseño satisfactorio, el usuario podrá usar el botón de Submit para mandar la figura a guardar.



Una vez guardado, el formulario se reinicia para inserción de más figuras en caso de querer. Podremos entonces acceder mediante los enlaces en la esquina superior, permitiendo entrar en la página donde están todas las figuras (All figures) o donde solo se ven las del usuario (My figures).

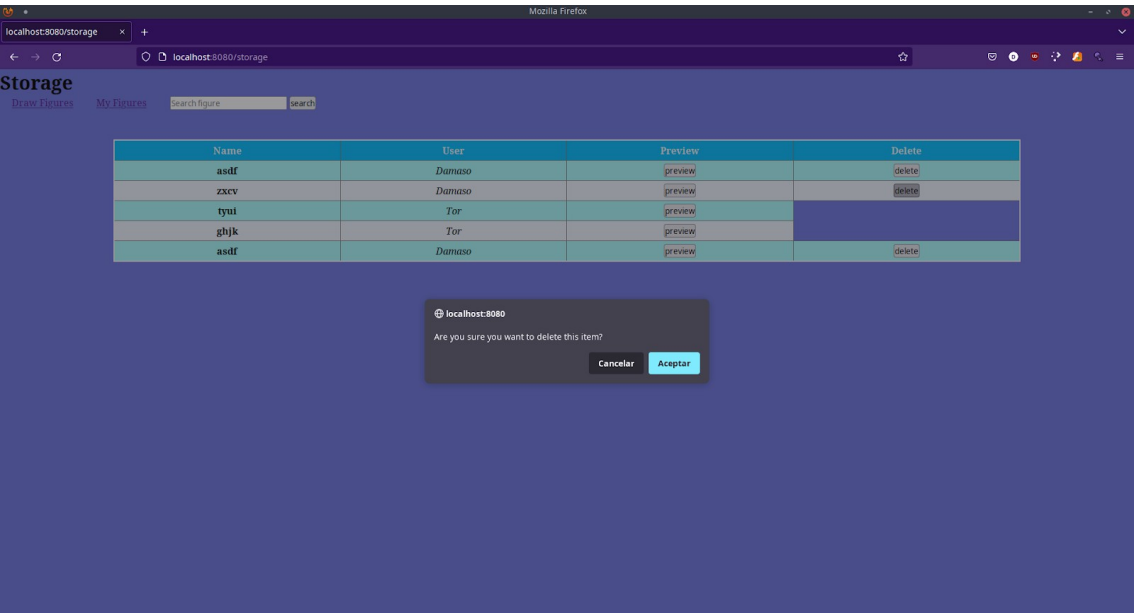
En la página de todas las figuras se pueden ver todas las figuras ordenadas en una tabla, donde se ve en la primera columna el nombre asignado a la figura, en la segunda el nombre del usuario propietario de la misma, en la tercera el botón que permite la previsualización de la figura y en la cuarta, en caso de ser propietario de la figura, podrá borrarla



The screenshot shows a web browser window at localhost:8080/storage. The page has a dark blue header with the title 'Storage' and navigation links 'Draw Figures' and 'My Figures'. A search bar is present. Below the header is a table with four columns: Name, User, Preview, and Delete. The table contains six rows of data. The first three rows are owned by 'Damaso' and the last three by 'Tor'. Each row has a 'preview' button and a 'delete' button (only visible for the 'Damaso' entries).

Name	User	Preview	Delete
asdf	Damaso	preview	delete
zxcv	Damaso	preview	delete
qwer	Damaso	preview	delete
Triangle_1_3152786e-7306-46e0-b182-bd1d32deh906	Damaso	preview	delete
tyui	Tor	preview	
ghjk	Tor	preview	

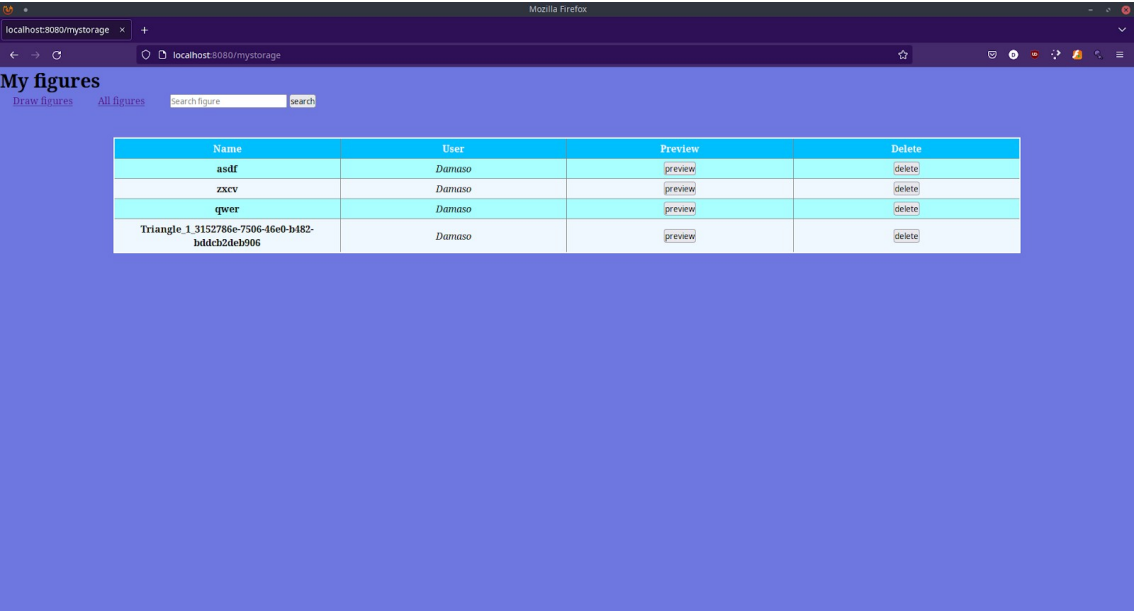
En caso de intentar borrar una figura primero ejecutará un prompt en el que se solicitará confirmación para el borrado antes de realizarlo



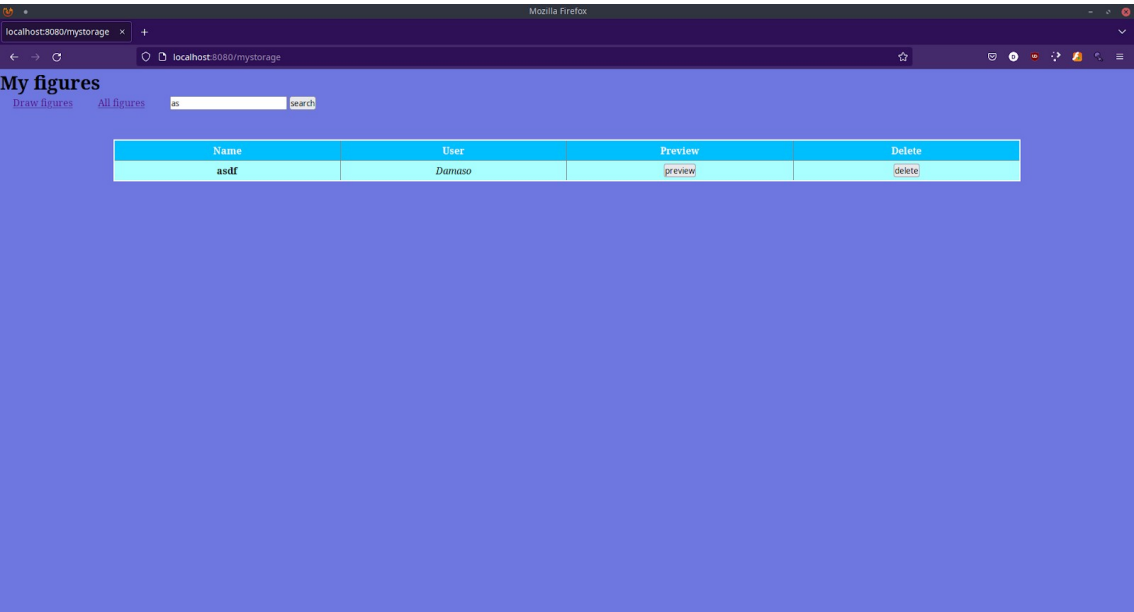
This screenshot shows the same 'Storage' application interface as the previous one, but with a confirmation dialog box open in the foreground. The dialog box is titled 'localhost:8080' and asks 'Are you sure you want to delete this item?'. It has two buttons: 'Cancelar' (Cancel) and 'Aceptar' (Accept). The table in the background is partially visible and shows the same data as before.

Name	User	Preview	Delete
asdf	Damaso	preview	delete
zxcv	Damaso	preview	delete
tyui	Tor	preview	
ghjk	Tor	preview	
asdf	Damaso	preview	delete

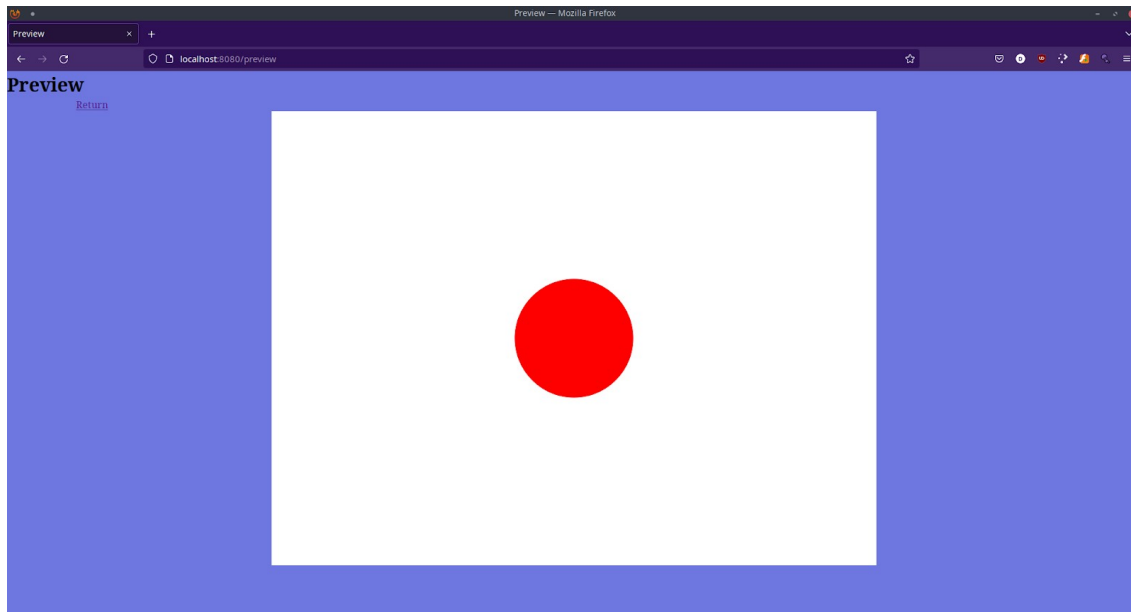
En la página de las figuras propias, la lista mostrada sólo mostrará las figuras creadas por el propio usuario.



Junto a los enlaces, tenemos un renglón para introducir una búsqueda, para localizar figuras por patrón de texto



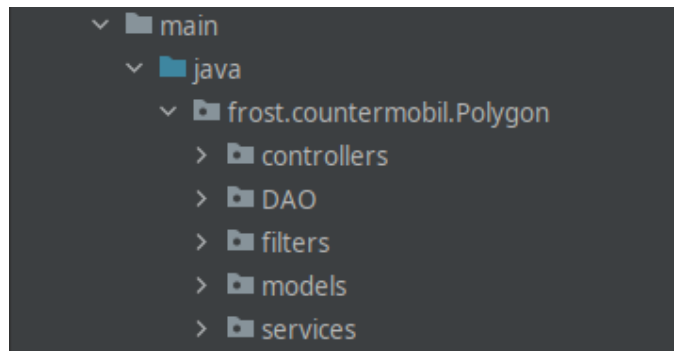
En último lugar, al pulsar preview permitirá entrar en otra página que tendrá un canvas con la figura almacenada mostrada. En esta página la única acción disponible será regresar a la página del almacén general



Funcionamiento del algoritmo

Este algoritmo se divide en clases Java y vistas JSP escritas en HTML y con funciones JSTL para implementar funciones Java en estos.

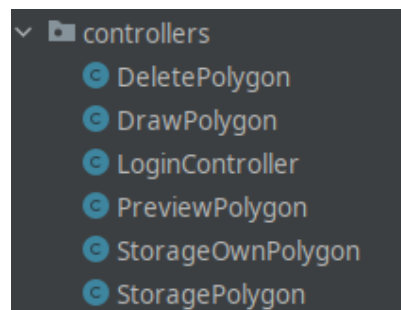
Las clases están separadas en paquetes según su propósito.



Paquete controller:

Controller es el módulo que controla la entrada y salida de información y contiene las clases que implementan HttpServlet para montar el servlet en la dirección especificada, utilizan sobreescripciones de métodos doGet y doPost para recibir la información al entrar en la página y al completar los formularios, respectivamente. Mediante el uso de un RequestDispatcher este programa puede mostrar los JSP en el explorador.

Encontramos los siguientes controladores



Por orden de uso, el primero es el LoginController:

LoginController

LoginController recoge la información insertada en el renglón de “User Name” y la prepara para y la almacena en la sesión del navegador como “userName” para que el filtro de usuarios pueda recuperarlo:

```
String userName = req.getParameter("userName");  
HttpSession session = req.getSession();
```

Tras esto, redirige al servlet “/draw”, el cual está construido en DrawPolygon.

DrawPolygon

DrawPolygon contiene su método doGet para montar la página web desde el jsp correspondiente y un método doPost para recuperar la información insertada en la configuración de la figura, para poder pasar toda esta información al servicio de figuras junto con los datos del usuario:

```
HttpSession session = req.getSession();  
String owner = (String) session.getAttribute("userName");  
int ownerId = (int) (session.getAttribute("userId"));  
String figureName = req.getParameter("figureName");  
int xCoord = Integer.parseInt(req.getParameter("figureX"));  
int yCoord = Integer.parseInt(req.getParameter("figureY"));  
String figureForm = req.getParameter("figureForm");  
int figureSize = Integer.parseInt(req.getParameter("figureSize"));  
String figureColor = req.getParameter("figureColor");  
figureService.newFigure(figureName, owner, ownerId, figureForm,  
figureColor, xCoord, yCoord, figureSize);
```

Los valores se recuperan mediante req.getParameter, el cual nos devuelve un String con el nombre del valor name del elemento que recogió la información en el formulario HTML. De esta manera, cada vez que el usuario pulsa submit, la información del formulario se mandará mediante método Post y el método doPost de este servlet la recuperará y mandará al FigureService para su gestión.

StoragePolygon/StorageOwnPolygon

En la clase StoragePolygon también posee un método doGet para montar la página mediante un dispatcher al cargar el sitio y además recuperará mediante el servicio de figuras el total de las figuras guardadas en el servidor y pasarlas como atributo para que el JSP pueda generar la tabla que los almacena con todos los elementos.

También guarda el userId para, a la hora de borrar una figura, muestre el botón o no.

```
List<Figure> figures = figureService.getAllFigures();  
req.setAttribute("userId", session.getAttribute("userId"));  
req.setAttribute("figures", figures);
```

El método doPost únicamente es usado en caso de que el usuario utilice el renglón de búsqueda, ya que cogerá el patrón introducido para usarlo como parámetro de la función de búsqueda y pueda filtrar la lista de figuras por ese patrón mediante el servicio de figuras.

```
String searchFigure = req.getParameter("figureSearch");  
figures = figureService.findFiguresByName(figures, searchFigure);  
req.setAttribute("figures", figures);
```

La única diferencia entre el controlador del almacén general y el del almacén propio es que el propio usa un método distinto de la clase de FigureService para que solo devuelva la lista de figuras que ha creado el usuario actual.

DeletePolygon

El controlador DeletePolygon únicamente consta de método doPost, ya que no es una página que se muestre, realiza el proceso de borrado de la figura escogida y devuelve al almacén general. El método requiere tanto el id del usuario como el de la figura, para realizar en su lógica interna la comprobación adicional de autoridad de borrado.

PreviewPolygon

El controlador PreviewPolygon posee un método doPost ya que el acceso al mismo solo es mediante una solicitud en las páginas de almacenamiento, que se mandan por formulario HTML con método Post. Se recuperan todos los atributos de la figura creada para poder pasarlos al javascript para que se reconstruya en el canvas.

```
Figure figure = figureService.previewFigure(figureId);  
int xCoord = figure.getXCoordinate();  
int yCoord = figure.getYCoordinate();  
String figureForm = figure.getForm();  
int figureSize = figure.getSize();  
String figureColor = figure.getColor();
```

```
req.setAttribute("figureX", xCoord);  
req.setAttribute("figureY", yCoord);  
req.setAttribute("figureForm", figureForm);  
req.setAttribute("figureSize", figureSize);  
req.setAttribute("figureColor", figureColor);
```

Paquete services

Tras esto tenemos los servicios. Los servicios son las clases que se encargan de enviar la información recuperada por los controladores para que el DAO las almacene, por tanto en su mayor parte se componen de métodos “passthrough”, ya que su función es pasar la información recibida a otro método, en este caso los de gestión de datos en la base de datos.

UserService

Este caso es la integridad del UserService, el cual solo pasa los datos para instanciar o recuperar usuarios a UserDao.

FigureService

En la clase FigureService podemos observar tres métodos adicionales: newFigure, findFiguresByName y createName.

El primero es newFigure, al cual pasamos como parámetro todos los datos necesarios para instanciar el objeto figura dentro de nuestra base de datos, y el objeto resultante es mandado a FigureDAO para almacenarlo en la base.

```
figure.setName(name);  
figure.setColor(color);  
figure.setXCoordinate(xCoord);  
figure.setYCoordinate(yCoord);  
figure.setSize(size);  
figure.setOwnerId(ownerId);  
figure.setOwner(owner);
```

Si el usuario no ha ingresado un nombre para la figura el programa detecta esto y antes de almacenarla usa el método createName para darle un nombre.

```
if (figure.getName().equals("")) {  
    figure.setName(createName(figure));  
}
```

findFiguresByName devuelve la lista de figuras que cumplan el patrón que corresponda al nombre de la figura. Genera la lista a devolver y va realizando un bucle forEach para comparar los nombres de cada figura con el patrón proporcionado y en caso de encajar, las almacena en esta lista nueva que es la que se devuelve

```

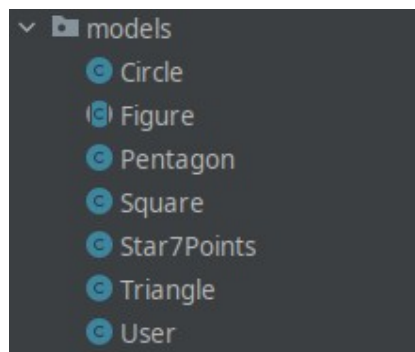
public List<Figure> findFiguresByName(List<Figure> figures, String
figureName) {
    List<Figure> foundFigures = new ArrayList<>();
    for (Figure figure : figures) {
        if (figure.getName().contains(figureName)) {
            foundFigures.add(figure);
        }
    }
    System.out.println(foundFigures);
    return foundFigures;
}

```

El método privado createName tiene como propósito asignar un nombre único a una figura sin nombre asignado durante su creación, para ello usamos la clase UUID (Universal Unique Identifier) para asignar como nombre la forma del objeto, el id del usuario y finalmente el UUID para prevenir que se generen dos objetos con el mismo nombre.

Paquete models

Las clases Model son aquellas que nos sirven para determinar los objetos que poblarán nuestras bases de datos.



Los principales siendo User, para cada uno de los usuarios, de los cuales nos interesa saber su nombre de usuario y su ID y Figure, la cual es una clase abstracta de la cual las otras figuras (Circle, Square, Triangle, Pentagon y Star7Points) son herederas.

Ambas clases únicamente tienen como métodos los setter y los getter ya que el único trabajo que nos interesa realizar con ellas es asignar y recuperar sus atributos, ya que como bien indica el nombre de su paquete, son solo modelos a partir de los cuales ordenar la información introducida.

Paquete DAO

Este paquete contiene 2 interfaces, una para gestionar usuarios y otra para gestionar figuras.

UserDAO

En la interfaz de UserDAO podemos encontrar los siguientes métodos:

```
void registerUser(User user);
```

Este primer método sirve para añadir un objeto user en el almacenamiento.

```
User recoverUser(String userName);
```

Este segundo método recupera un objeto user usando su nombre de usuario para localizarlo en el almacenamiento.

FigureDAO

De la interfaz FigureDAO podemos encontrar los siguientes métodos:

```
void save(Figure f);
```

Este método guarda un objeto figure dentro del almacenamiento.

```
List<Figure> getAllFigures();
```

Este método recupera en una lista todos los objetos figure almacenados.

```
List<Figure> getFiguresByUser(int userId);
```

Este método recupera en una lista los objetos figure creados por un usuario concreto.

```
void removeFigure(String figureId, int ownerId);
```

Este método retira un objeto figure concreto del almacenamiento. El id del propietario es usado como parámetro para realizar comprobaciones en el código.

```
Figure recoverFigure(String figureId);
```

Este método recupera un objeto figure concreto del cual se recupera la id. Usado por el controlador PreviewPolygon exclusivamente.

Paquete Local:

Dentro del paquete Local nos encontramos las clases que implementan estas interfaces para realizar el guardado dentro del servidor Tomcat.

UserDAOLocal

Guarda los usuarios dentro de una lista de users y almacena una id. Ambos atributos son estáticos para que solo exista una instancia de los mismos.

El método registerUser es un método sincronizado para que solo se pueda realizar una vez por hilo para evitar duplicidades. Coge el usuario y le asigna la id actual e incrementa el valor, para que siguientes usuarios creados no se les asigne la misma id.

El método recoverUser compara en un forEach cada usuario registrado en la lista y compara sus nombres con el nombre pasado por parámetro a la función y devuelve el usuario coincidente

FigureDAOLocal

El método save coge asigna una id al objeto figura que está guardando y tras eso lo guarda en la lista. Después incrementa en uno el valor del id para evitar repeticiones, tal y como hace el método registerUser.

El método getAllFigures recupera la lista completa de usuarios y la devuelve.

El método getFiguresByUser a partir de la id del usuario recupera todas las figuras que sean suyas al compararlas en un bucle forEach.

El método removeFigure a partir de una id de figura identifica una figura en la lista mediante el forEach que recorre la lista, pero utiliza el id de usuario para compararlo con el id del dueño de la figura. Si no coinciden, manda un mensaje a la consola indicando que el usuario que ha solicitado el borrado no es el usuario propietario de la figura. Si coinciden, la figura es borrada de la lista.

El método recoverFigure a partir de la id de la figura recupera una figura concreta de la lista de figuras.

Paquete Filter

UserFilter

El paquete filter contiene una clase cuyo único propósito es controlar que se cumpla la condición en su método doFilter en las páginas especificada en el urlPatterns de @webFilter, las cuales son un array de Strings

Dentro de esta se realiza una comprobación de si el valor "userName" ha sido instanciado en la sesión. En caso negativo, redirige al login para la identificación.

Si existe el userName comprobará el id de ese nombre de usuario, en caso de haber hecho login anteriormente, recuperará el id correspondiente para ese nombre de usuario y lo almacenará también en la sesión. En caso de no haber realizado nunca login, llamará al método registerUser del servicio de usuarios para asignarle un id y guardarlo.