

Documentación proyecto Diseño y Arquitectura de Software

En este documento se comentará sobre el contexto de la solución, los estilos y patrones arquitectónicos utilizados, las funcionalidades, tecnologías utilizadas para llegar a la solución y las responsabilidades éticas, empezando por el contexto:

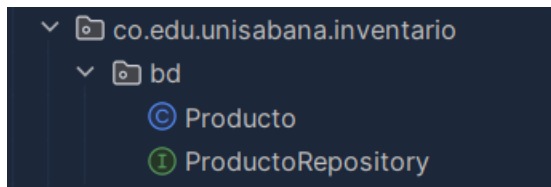
Contexto:

El aplicativo consiste en la gestión de inventario de cualquier tipo de tienda, la cual ofrece agregar, actualizar, eliminar y buscar productos con facilidad. Esta aplicación esta diseñada como una REST API, basándose en los principios de la arquitectura REST, donde además, se realizado un front para los usuarios para que sea de fácil interpretación y usabilidad además de implementar tecnologías dentro del backend para asegurar el correcto desarrollo del mismo.

Estilos arquitectónicos:

El aplicativo de gestión de inventario se considera una combinación entre Cliente-Servidor, siendo este el fuerte, y un ligero monolito. Por un lado, se considera cliente-servidor porque las capa cliente y la capa de servidor se encuentran desacopladas, esto teniendo en cuenta que por la capa de servidor encontramos el desarrollo de las APIs tipo REST generando una aplicación de tipo CRUD que genera cambios en la base de datos.

Las evidencias son las siguientes:



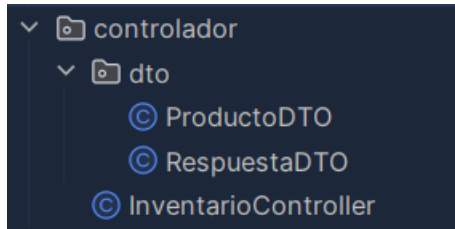
```
@Entity
@Table(name = "inv_rest")
@SQLDelete(sql = "UPDATE inv_rest SET borrado = true WHERE id=?")
@Where(clause = "borrado=false")
@Data
public class Producto {

    @Id
    @Column
    private int id;
    @Column
    private String nombre;
    @Column
    private String descripcion;
    @Column
    private int precio;
    @Column
    private int stock;
    @Column
    private String categoria;
    @Column(name = "fechacreacion")
    @CreatedDate
    private LocalDateTime fechaCreacion;
    @Column(name = "fechaact")
    @LastModifiedDate
    private LocalDateTime fechaAct;
    @Column
    private boolean borrado = Boolean.FALSE;
}
```

Lo que aquí tenemos son las entidades de la base de datos que se encuentran en el código, las cuales decidimos tener un mismo nombre en la mayoría para no tener la necesidad de añadir más

líneas de código Column, donde se tiene una anotación Entity que declara las variables como entidades.

Luego se tiene la interfaz del repositorio la cual se encarga del manejo de datos.



```
@Data
public class ProductoDTO {
    private int id;
    private String nombre;
    private String descripcion;
    private int precio;
    private int stock;
    private String categoria;

    9 usages  🇩🇪 Deutsch
    public ProductoDTO(int id, String nombre, String descripcion, int precio, int stock, String categoria) {
        this.id = id;
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.precio = precio;
        this.stock = stock;
        this.categoria = categoria;
    }

    3 usages  🇩🇪 Deutsch
    public ProductoDTO() {
    }
}
```

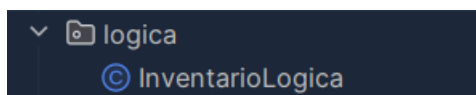
@Data

```
public class RespuestaDTO {  
    String mensaje;  
  
    8 usages  ⓘ Deutsch  
    public RespuestaDTO(String mensaje) { this.mensaje = mensaje; }  
  
    no usages  ⓘ Deutsch  
    public RespuestaDTO() {  
    }  
}
```

```
@ApiOperation(value = "Crear un producto", response = Producto.class)  
@PostMapping(path = "/producto/agregar")  
public RespuestaDTO agregarProducto(@RequestBody ProductoDTO productoDTO) {  
    try {  
        log.info("Agregando producto con ID "+productoDTO.getId()+" en la BD");  
        logica.agregarProducto(productoDTO);  
        return new RespuestaDTO( mensaje: "Producto guardado correctamente");  
    } catch (Exception e) {  
        log.info("Producto con ID "+productoDTO.getId()+" no se ha podido guardar");  
        return new RespuestaDTO( mensaje: "Se genero un error al guardar el producto");  
    }  
}
```

```
@ApiOperation(value = "Eliminar un producto por su ID", response = Producto.class)  
@DeleteMapping(path = "/producto/eliminar")  
public RespuestaDTO borrarProducto(@RequestParam int id) {  
    try {  
        log.info("Producto con ID "+id+" eliminado");  
        logica.eliminarProducto(id);  
        return new RespuestaDTO( mensaje: "Producto eliminado correctamente");  
    } catch (Exception e) {  
        log.info("Eliminación de producto con ID "+id+" fallida");  
        return new RespuestaDTO( mensaje: "El producto no se pudo eliminar");  
    }  
}
```

Teniendo en cuenta la carpeta del controlador nos encontramos con los DTO que son los objetos de transferencia de datos entre procesos, esto porque no se puede procesar las entidades de la base de datos comentadas anteriormente. Entre estas tenemos el DTO de respuesta el cual se encarga de transmitir un mensaje en los métodos trabajados en el controller y luego el DTO de los productos que se encargan de los procesos de transferencia de los diferentes métodos del controlador. Estos para comunicarse con la base de datos tienen unas anotaciones mapping que son las asignaciones entre solicitudes HTTP y los métodos del controlador, teniendo el de agregar, eliminar, actualizar y las búsquedas, además de unas anotaciones de `RequestBody` o `RequestParam` los cuales se encargan, en el orden escrito, para indicar que un parámetro debería estar vinculado al cuerpo de la solicitud HTTP (Mapping) que se usan usualmente para solicitudes POST y el otro se utiliza para circular los parámetros de una solicitud web a los parámetros de los métodos del controlador.



```
public Producto agregarProducto(ProductoDTO productoDTO) {  
    Producto producto = new Producto();  
    producto.setId(productoDTO.getId());  
    producto.setNombre(productoDTO.getNombre());  
    producto.setDescripcion(productoDTO.getDescripcion());  
    producto.setPrecio(productoDTO.getPrecio());  
    producto.setStock(productoDTO.getStock());  
    producto.setCategoria(productoDTO.getCategoria());  
    producto.setFechaCreacion(LocalDateTime.now());  
    producto.setFechaAct(LocalDateTime.now());  
    productoRepository.save(producto);  
    return producto;  
}
```

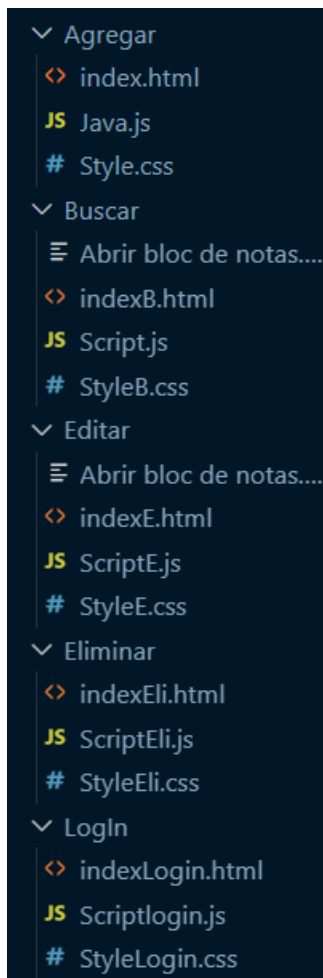
Ahora aquí tenemos la parte lógica del aplicativo, donde la entidad del producto se trabaja por medio de los procesos del DTO, ya sea añadiendo, actualizando, borrando y buscando los datos de la base de datos y es aquí donde el repository tiene efecto, por lo que luego de realizar los procesos

se encarga de realizar los cambios de la base de datos, por lo cual la lógica dentro del aplicativo se encarga de los procesos de los datos.

Además, dentro del aplicativo obtenemos diferentes funcionalidades como seguridad con una autenticación básica, un apartado de documentación por medio de Swagger y de configuración para la comunicación entre backend y frontend

Esto teniendo en cuenta todo lo que realiza el servidor para el cliente, abarcando sus funcionalidades de una manera general

Ahora por el apartado del usuario obtenemos un front de fácil interpretación el cual le ofrece a la persona que lo este utilizando la capacidad de simplemente realizar las peticiones de la gestión del inventario con simplemente añadiendo los datos necesarios



```

let boton = document.getElementById("btnGuardar");

boton.addEventListener("click", evento=>{
  | agregarProducto();
});

let agregarProducto = async()=>{

  let campos = {
    id : document.getElementById("id").value,
    nombre : document.getElementById("nombre").value,
    descripcion : document.getElementById("descripcion").value,
    precio : document.getElementById("precio").value,
    stock : document.getElementById("stock").value,
    categoria : document.getElementById("categoria").value
  };

  try {

    let username = 'admin';
    let password = 'clave1';
    let credentials = `${username}:${password}`;
    let encodedCredentials = btoa(credentials);

    const petition = await fetch("http://localhost:8080/producto/agregar", {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
        'Authorization': `Basic ${encodedCredentials}`,
      },
      body: JSON.stringify(campos),
    });
  }catch (error) {
    console.error('Error:', error);
  }
}

```

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="https://kit.fontawesome.com/eb496ab1a0.js" crossorigin="anonymous"></script>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css"
    integrity="sha512-iecdLmaskl7CVkqkXNQ/ZH/XLlvWZOjyj7Yy7tcenmpD1ypASozpmT/E0iPtmFIB46ZmdtAc9eNBvH0H/ZpibW=="
    crossorigin="anonymous" referrerpolicy="no-referrer" />
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/swiper@9/swiper-bundle.min.css">
  <link rel="stylesheet" href="Style.css">
</head>
<body>
  <ul>
    <li><a href="/Agregar/" ><i class="fa-solid fa-upload fa-2x"></i></a></li>
    <li><a href="/Editar/indexE.html" ><i class="fa-regular fa-pen-to-square fa-2x"></i></a></li>
    <li><a href="/Eliminar/indexEli.html" ><i class="fa-solid fa-trash fa-2x"></i></a></li>
    <li><a href="/Buscar/indexB.html" ><i class="fa-solid fa-magnifying-glass fa-2x"></i></a></li>
  </ul>

  <div>
    <h1 class="Titulo">Agregar</h1>
  </div>





```

```

  <script src="https://cdn.jsdelivr.net/npm/swiper@9/swiper-bundle.min.js"></script>
  <script src="/Agregar/Java.js"></script>
</body>
</html>

```

Estas evidencias son un ejemplo de lo que se tiene de la codificación del frontend, donde se especifican la URL del método del backend, con los elementos necesarios y la navegación entre las interfaces de los diferentes métodos.



Agregar

ID:

Nombre:

Descripción:

Precio:

Stock:

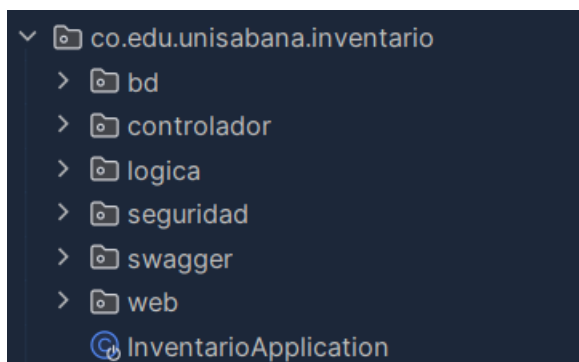
Categoría:

Guardar

Aquí mostrando la interfaz para agregar productos a la base de datos, sin la necesidad de entender que hay detrás de esta.

Esto finalizando la explicación de porque es un cliente servidor, mostrando código el cual se encarga del proceso el cual el usuario manda las peticiones que este requiere.

Patrón Arquitectónico



Como patrón arquitectónico se utilizó el patrón de capas, por lo cual el sistema está dividido por una serie de capas de las cuales cada una esta dividida con una responsabilidad especifica donde estas capas se encargan de la lógica y el acceso de datos, de almacenar los datos y de la seguridad del aplicativo.

Requerimientos funcionales

Registro de productos:

- Permite al usuario agregar productos nuevos

Actualización de productos:

- Permite al usuario la actualización de productos por medio de la indicación del ID, ya sea actualizando el nombre, el precio, el stock, la descripción o la categoría.

Eliminar productos:

- Permite al usuario eliminar productos de manera lógica, manteniendo la integridad de la base de datos

Buscar productos por el ID

- Permite al usuario buscar productos por medio del ID, mostrando las diferentes características que estos tienen

Buscar el stock de un producto por su ID

- Permite al usuario buscar el stock de un producto específico por medio del ID

Buscar la cantidad de productos por medio de la categoría

- Permite ver al usuario cuantos productos hay en una categoría específica, mostrando de todos los productos sus características

Requerimientos no funcionales

Mantenibilidad:

- El código se encuentra bien estructurado y con buenas practicas de programación, facilitando así el mantenimiento y evolución del aplicativo, haciendo uso de herramientas como SonarQube y Jacoco para garantizar la calidad del código

Disponibilidad

- El aplicativo es disponible y accesible para los usuarios, esto teniendo en cuenta que siendo la gestión de un inventario puede haber grandes cargas de información.

Seguridad:

- El código contiene una autenticación básica para el uso del aplicativo, además de tener un registro de logs que nos muestra que está haciendo el aplicativo.

Responsabilidad Ética:

1. ¿El sistema respeta la privacidad de los usuarios?

Sí, ya que la información que los usuarios utilizan es almacenada de forma segura evitando así que se pueda vulnerar de alguna forma

2. ¿El sistema tiene un impacto en el medio ambiente?

A gran escala el sistema no tiene un impacto significativo con el medio ambiente, sin embargo, por el uso del servidor si se llega a generar, aunque en una mínima cantidad un consumo de energía que podría llegar a ser un problema para el ambiente.

3. ¿Promueve la diversidad y la inclusión?

No, el sistema no promueve de forma explícita la diversidad y la inclusión, sin embargo es un sistema con el cual todo tipo de usuario puede utilizar sin mucho problema ya que su interfaz es sencilla e intuitiva

4. ¿Cómo afecta este sistema a la economía?

Afecta la economía ya que nuestro sistema de inventario permite la gestión y operación de datos asertivos de las diferentes industrias por lo que impulsa la eficiencia de estas. Por lo que juega un papel importante al momento de manejar dichos datos

5. ¿Daños Económicos?

En caso de haber fallas dentro del funcionamiento de la aplicación, el equipo se hace cargo de las pérdidas económicas