**UNIVERSIDAD PRIVADA DE TACNA**

**FACULTAD DE INGENIERIA**

**Escuela Profesional de Ingeniería de Sistemas**



**ANEXO – 02**

**“PATRONES DE DISEÑO ESTRUCTURALES,**

**DE COMPORTAMIENTO, CREACIONALES”**

Curso: Patrones de software

Docente: Mag. Patrick Cuadros Quiroga

**Integrantes:**

**Romero Roque, Angelica Beatriz      (2019063327)**

**Chambilla Maquera, Aroceli Noemi  (2018060897)**

**Arhuata Turpo, Alex                              (2019063634)**

**Rojas Bedregal ,Brian Erik** **(2018060904)**

**Vilca Condori Erlang Fernando (2019064024)**

**Lima Ccosi, Jeackory Geilly               (2019063326)**

**Tacna – Perú**

***2023***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CONTROL DE VERSIONES | | | | | |
| Versión | Hecha por | Revisada por | Aprobada por | Fecha | Motivo |
| 1.0 | ACHM  brb | ACHM | PCQ | 07/06/2023  11/07/2023 | Versión Original |

Contenido

[1. INTRODUCCION 3](#_Toc139936311)

[2. ¿Qué son los patrones de diseño de software? 3](#_Toc139936312)

[3. ¿Por qué usar patrones de diseño? 4](#_Toc139936313)

[4. ¿Son estos los únicos patrones que existen? 4](#_Toc139936314)

[5. TIPOS DE PATRONES DE DISEÑO DE SOFTWARE 4](#_Toc139936315)

[6. Patrones Creacionales 4](#_Toc139936316)

[7. Patrones estructurales 5](#_Toc139936317)

[8. Patrones de comportamiento 5](#_Toc139936318)

[9. IMPLEMENTACION 5](#_Toc139936319)

[10. PATRON REPOSITORY 5](#_Toc139936320)

[11. PATRON DATA TRANFER OBJECT 6](#_Toc139936321)

[12. **PATRON SERVICE** 7](#_Toc139936322)

[13. PATRON DE FABRICA 8](#_Toc139936323)

# INTRODUCCION

La historia de los patrones de diseño comienza en 1977, pero no en el ámbito de software, sino en el de la ingeniería civil. En este año Christopher Alexander publica el libro "A Pattern Language Towns, Buildings, Construction".

En este libro en lugar de hablar sobre los detalles de la construcción, Alexander se centra en los problemas comunes que enfrenta un trabajo de arquitectura. Estos problemas los explica de la forma más general posible, capturando solo su esencia. Con esto, logró que el libro hable de problemas generales y atemporales. En este libro también habla de las relaciones entre estos problemas.   
   
¿Qué tiene que ver esto con el software? Pues bien, en 1994 un grupo de 4 sujetos: Erich Gamma, Richard Helm, Ralph Johnson, y John Vlissides tomaron esta idea de patrones para poder aplicarla al mundo del desarrollo de software y publicaron uno de los libros más influyentes en la historia de la programación: "Design Patterns: Elements of Reusable Object-Oriented Software". Cariñosamente nos referimos a este grupo de autores con el nombre de “La banda de los 4” (Gang of Four, en inglés) o simplemente GoF.

De acuerdo con los autores, todos los patrones de diseño se basan en los siguientes principios del diseño orientado a objetos:

* Programar hacia una interface, no a una implementación.
* Favorecer la composición de objetos frente a la herencia de clases.
* Determinar qué parte del problema es común y qué parte es variable.
* Permitir el reemplazo de la parte variable mediante la interfaz común.

Hay que recordar que estos autores no tomaron solo su experiencia para escribir estos patrones. Sino que recopilaron la experiencia de los problemas comunes que observaron en la comunidad y los concentraron en una sola fuente de información. 

# ¿Qué son los patrones de diseño de software?

Si eres programador o estás despegando en el desarrollo informático y la programación, tarde o temprano te toparás con el término (patrones de diseño de software). El desarrollo de software es un proceso complejo que requiere un enfoque estructurado y eficiente para lograr resultados óptimos.

En este contexto, los patrones de diseño de software se han convertido en una herramienta invaluable para los desarrolladores, brindando soluciones probadas y comprobadas para problemas comunes de diseño.

Al comprender y aplicar estos patrones, los profesionales del desarrollo pueden mejorar significativamente la eficiencia y mantenibilidad del código, evitando errores costosos y optimizando el tiempo de desarrollo.

# ¿Por qué usar patrones de diseño?

El gran crecimiento del sector de las tecnologías de la información ha hecho que las prácticas de desarrollo de software evolucionen. Antes se requería completar todo el software antes de realizar pruebas, lo que suponía encontrarse con problemas. Para ahorrar tiempo y evitar volver a la etapa de desarrollo una vez que este ha finalizado, se introdujo una práctica de prueba durante la fase de desarrollo.

Esta práctica se usa para identificar condiciones de error y problemas en el código que pueden no ser evidentes en ese momento. En definitiva, los patrones de diseño te ayudan a estar seguro de la validez de tu código, ya que son soluciones que funcionan y han sido probados por muchísimos desarrolladores siendo menos propensos a errores.

# ¿Son estos los únicos patrones que existen?

No, cada tecnología, lenguaje de programación, o incluso framework, puede definir sus propios patrones de diseño. En esta serie veremos los patrones más comunes y, dependiendo de la respuesta y necesidades, veré si es necesario que entremos en patrones más específicos.

También es importante entender que esos patrones de diseño se usan en aplicaciones que usan el paradigma orientado a objetos. Diferentes paradigmas tienen una serie diferente de patrones.

# TIPOS DE PATRONES DE DISEÑO DE SOFTWARE

 Los patrones de diseño más utilizados se clasifican en tres categorías principales, cada patrón de diseño individual conforma un total de 23 patrones de diseño. Las cuatro categorías principales son:

1. *Patrones creacionales*
2. *Patrones estructurales*
3. *Patrones de comportamiento*

*Mapa conceptual N°01*



*Tipos de patrones*

# Patrones Creacionales

Los patrones de creación proporcionan diversos mecanismos de creación de objetos, que aumentan la flexibilidad y la reutilización del código existente de una manera adecuada a la situación. Esto le da al programa más flexibilidad para decidir qué objetos deben crearse para un caso de uso dado.

Estos son los patrones creacionales:

# Patrones estructurales

Facilitan soluciones y estándares eficientes con respecto a las composiciones de clase y las estructuras de objetos. El concepto de herencia se utiliza para componer interfaces y definir formas de componer objetos para obtener nuevas funcionalidades.

# Patrones de comportamiento

El patrón de comportamiento se ocupa de la comunicación entre objetos de clase. Se utilizan para detectar la presencia de patrones de comunicación ya presentes y pueden manipular estos patrones.

Estos patrones de diseño están específicamente relacionados con la comunicación entre objetos.

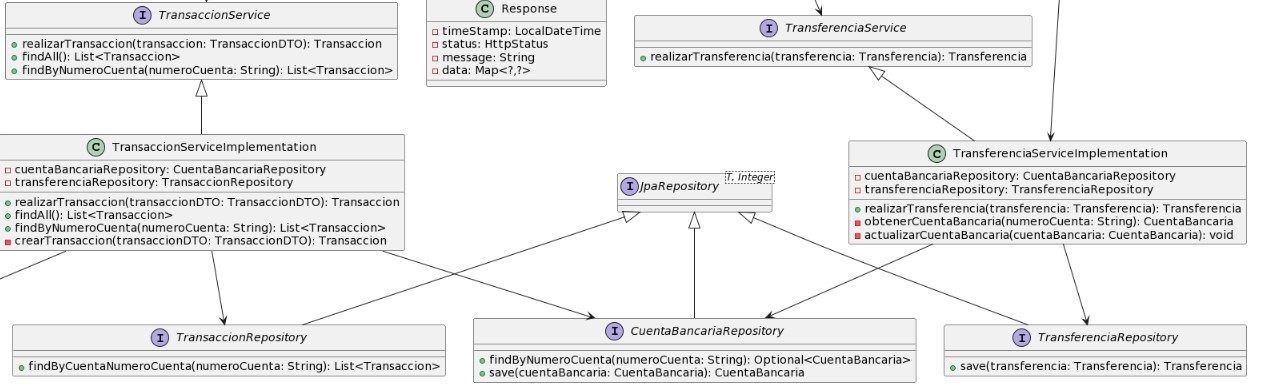
# IMPLEMENTACION

Explicaremos en cómo se llevó a cabo la implementación de los patrones de diseño estructural, creacional y de comportamiento en el proyecto detallando el porqué de su uso.

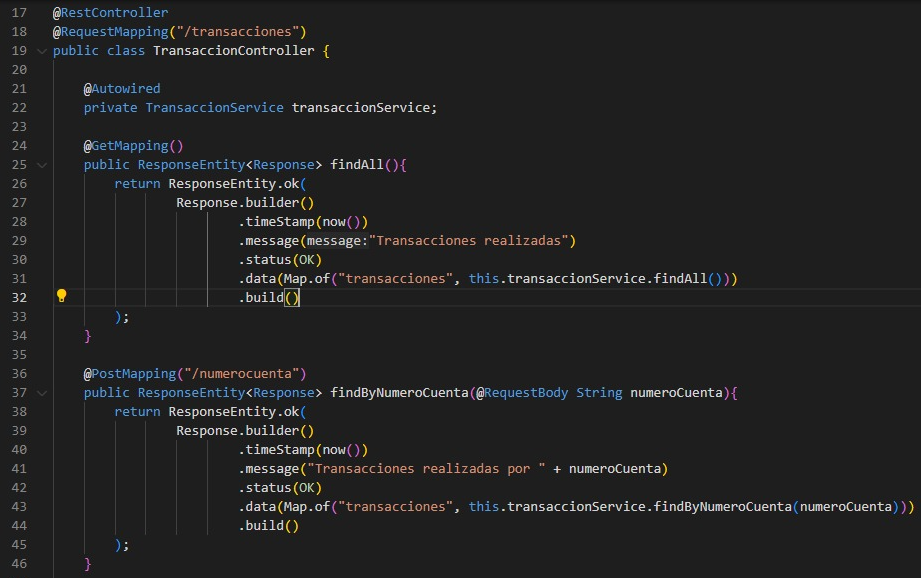
# PATRON REPOSITORY

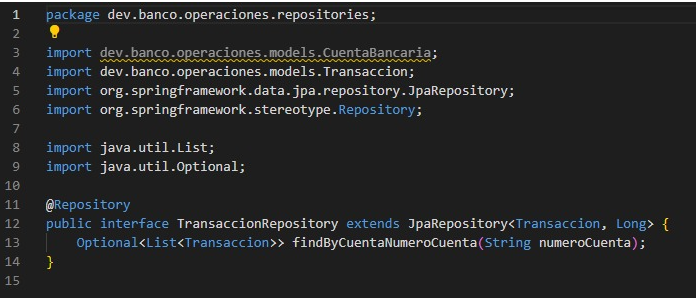
El patron Repository es uno de los patrones más clásicos en Enterprise Design Pattern. Este patrón hace referencia a como persistir un objeto en una base de datos. Por el ejemplo si disponemos de la clase transacciones como podemos persistir esta de forma sencilla. Para ello el patrón de diseño se encarga de agrupar todas las operaciones en una única clase que las gestiona.

*Mapa conceptual N°2: Patron Repository*



Este código es bastante funcional y nos aborda todas las necesidades que nosotros tenemos a la hora de gestionar la persistencia elemental de una transacción.

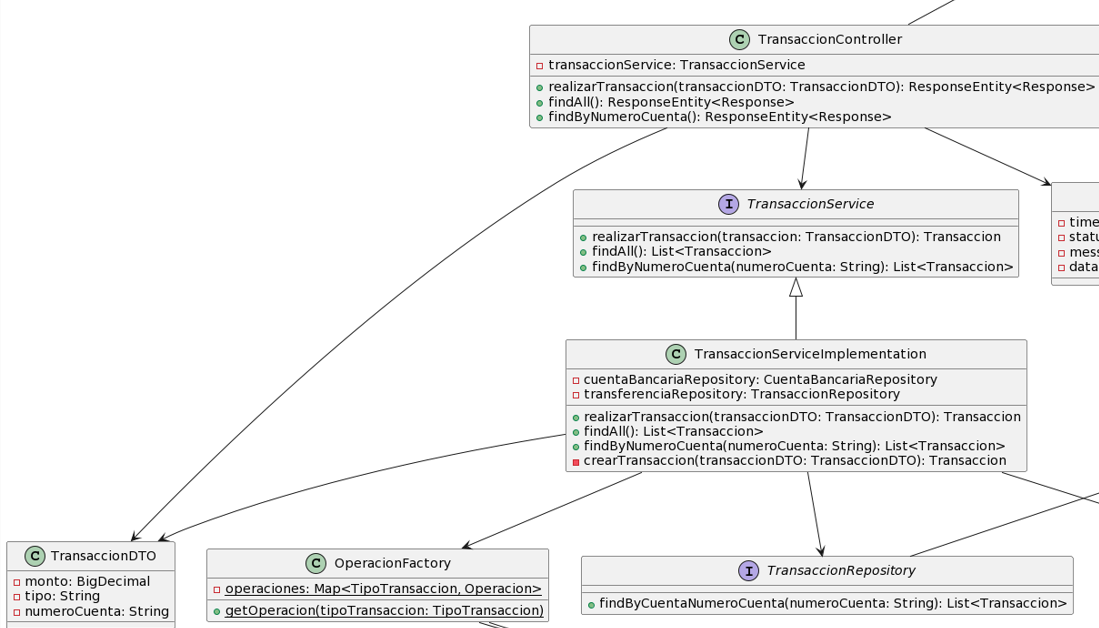




# PATRON DATA TRANFER OBJECT

El uso de DTO o Data Transfer Object es uno de los conceptos más habituales a nivel de Arquitectura cuando devolvemos en nuestros servicios estructuras de datos. Muchos servicios devuelven objetos de negocio o gráfos con objetos de negocio relacionados. Es decir, cuando nosotros tenemos un método que nos devuelve información sobre Transacciones lo más sencillo es devolver por ejemplo una lista de Personas desde un servicio.

*Mapa conceptual N°3: Patrón DTO*



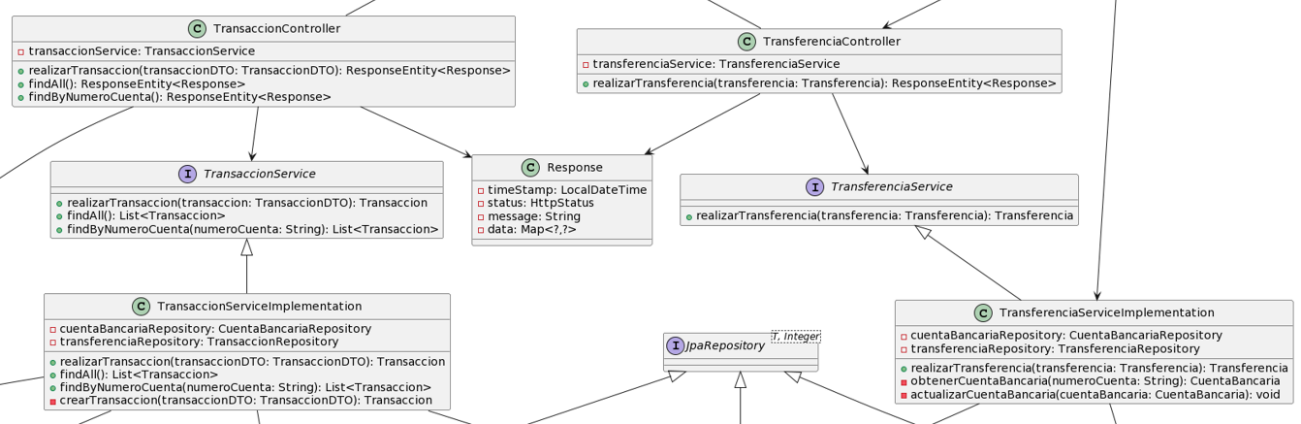
Creamos un método ValidarTipoOperacion para que este servicio valido Transaccion\_DTO.

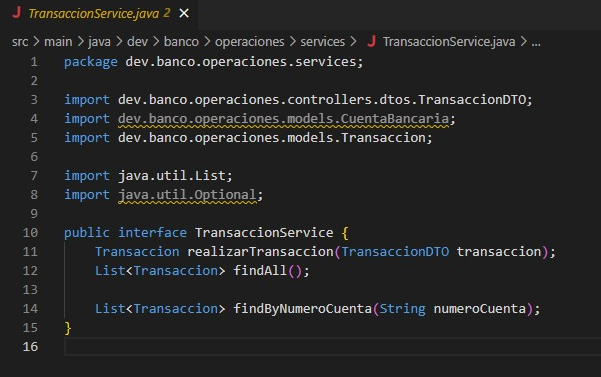


# **PATRON SERVICE**

Es una de las anotaciones más habituales de Spring Framework. Se usa para construir una clase

*Mapa conceptual N°4: Patrón Service*

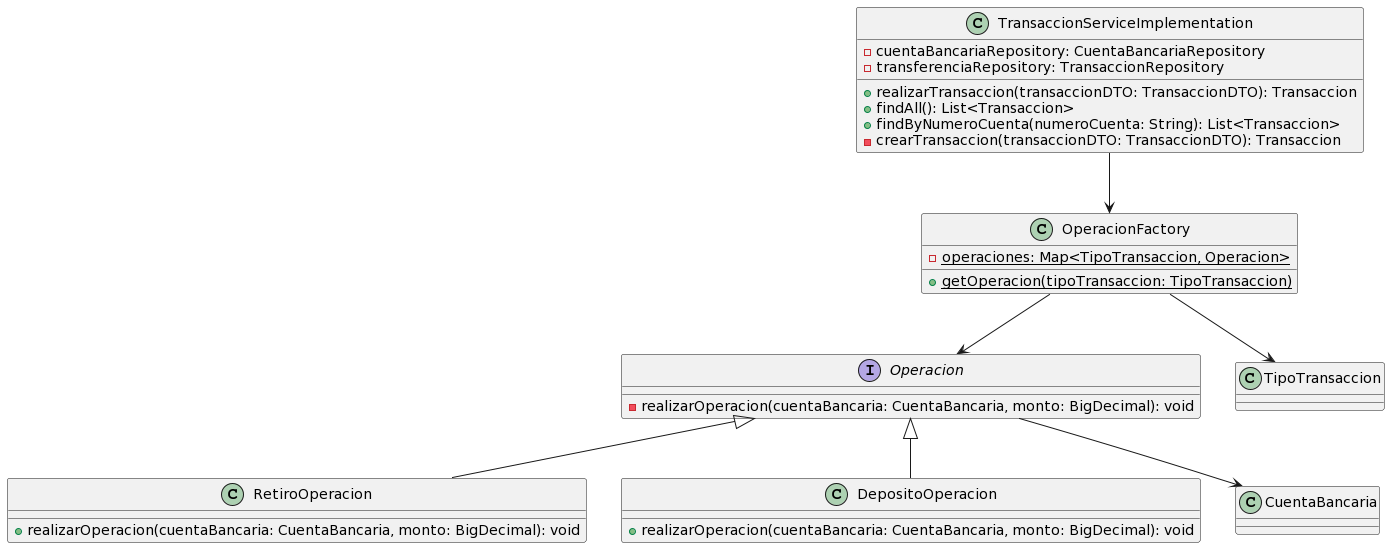




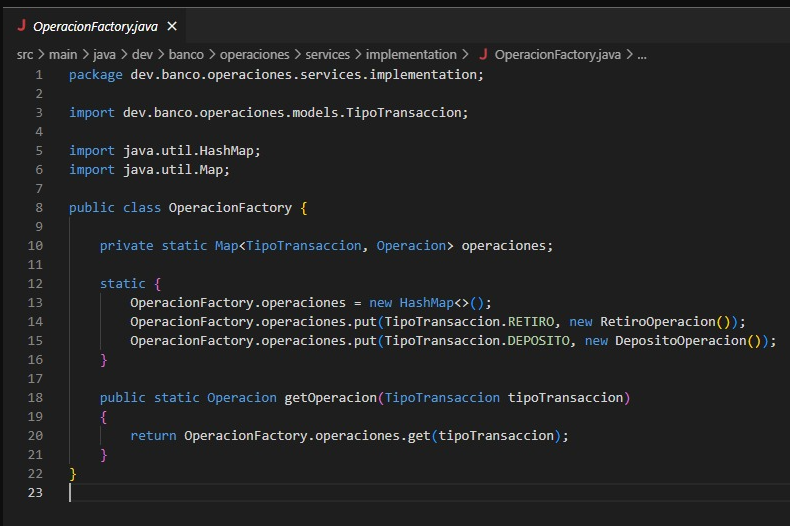
# PATRON DE FABRICA

El patrón Factory es uno de los patrones fundamentales a nivel de diseño orientado a objeto. Este patrón pertenece al grupo de patrones creacionales y nos simplifica la construcción de una jerarquía de clases.

Sin embargo, a veces a la gente le cuesta ver cómo usar este patrón en su código. Vamos a utilizar un ejemplo sencillo en el que tendremos una jerarquía de clases Transacción como se muestra a continuación.

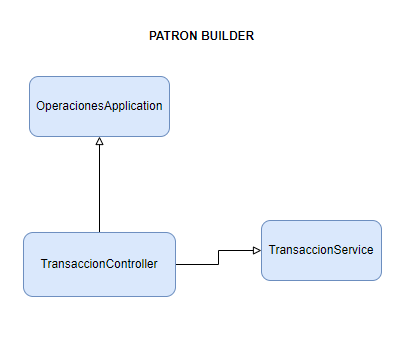


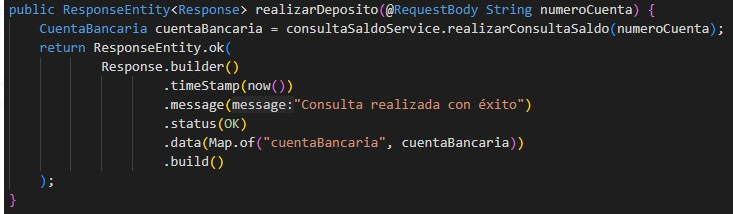
Vamos a ver la parte del código:



# PATRON BUILDER

Builder es un patrón de diseño creacional con ámbitos de objeto. Nos ayuda a crear objetos complejos de una forma sencilla usando un procedimiento de "paso por paso". Esto facilita el trabajo enormemente cuando tenemos objetos con muchos atributos, pero no necesitamos establecer de inicio todos sus valores, o no siempre establecemos los mismos. De esta forma nos guía en la construcción de estos obejtos. También ayuda en el ensamblado de objetos complejos que están formados por otros objetos.

   
  
Podemos apreciar la parte del código



# Conclusión

En conclusión, el principio SOLID es una guía valiosa para el diseño de software orientado a objetos, ya que promueve la creación de sistemas flexibles, mantenibles y escalables. Al aplicar estos principios, se obtiene un código más limpio, modular y reutilizable, lo que facilita la evolución del software a lo largo del tiempo y mejora la calidad general del sistema.

# Referencia Bibliográfica:

* Vista de Creación de una guía para la utilización de la arquitectura “clean architecture” en aplicaciones Android con lenguaje de programación Java, utilizando como modelo la implementación de un prototipo de monitoreo deportivo para un equipo de ultimate. (s/f). Utp.ac.pa. Recuperado de https://revistas.utp.ac.pa/index.php/ric/article/view/1750/html
* de Aplicações Web Utilizando Os Princípios Solid, M. V. R. D. (2022). UNIVERSIDADE DO SUL DE SANTA CATARINA. Com.br. Recuperado de https://repositorio.animaeducacao.com.br/bitstream/ANIMA/22977/1/2022-tcc\_Miguel-Ramos.pdf
* Benito, M., & Felipe, J. (2021). Metodologías ágiles de desarrollo aplicadas a la enseñanza de la programación. Recuperado de https://rua.ua.es/dspace/handle/10045/116132#vpreview
* DESARROLLO DEL MÓDULO DE CONTABILIDAD EN LENGUAJE NET CORE CUMPLIENDO CON LOS PRINCIPIOS DE DESARROLLO SOLID Y DE ARQUITECTURA LIMPIA USANDO LOS FRAMEWORK ENTITY, First, C., & DEVEXPRESS Y BOOTSTRAP 4 EN UN REPOSITORIO SQL SERVER CON CONTROL DE VERSIONES GIT BAJO EL MARCO DE TRABAJO SCRUM EN MICROSHIF. (s/f). PROGRAMA DE INGENIERÍA DE SISTEMAS. Edu.co. Recuperado de http://repositoriodspace.unipamplona.edu.co/jspui/bitstream/20.500.12744/5636/1/CUADRADO\_2019\_TG.pdf
* Vista de Creación de una guía para la utilización de la arquitectura “clean architecture” en aplicaciones Android con lenguaje de programación Java, utilizando como modelo la implementación de un prototipo de monitoreo deportivo para un equipo de ultimate. (s/f-b). Utp.ac.pa. Recuperado de https://revistas.utp.ac.pa/index.php/ric/article/view/1750/2491

Link del Repositorio del Proyecto en Github:

https://github.com/UPT-FAING-EPIS/proyecto-unidad-iii-pds-api\_operacion