ISSN: 2711-3760
https://doi.org/10.47185/27113760.v1n1.7

Utilización de Arquitecturas Limpias para Trabajo con Buenas Prácticas en la Construcción de Aplicaciones Java.

Martínez Z, Julio César [1(*)]; Henao, César [2]; Henao, Federico [2] & Zapata, Esteban [1]

[1]Politécnico Colombiano Jaime Isaza Cadavid, Facultad Ingenierías, Medellín, Colombia
[2]Corporación Universitaria Americana, Facultad Ingenierías, Medellín, Colombia

RESUMEN

En este artículo se propone la creación de aplicaciones Java a nivel empresarial aplicando buenas prácticas con el uso de arquitecturas limpias de software. Se hace una exploración de arquitecturas limpias y sus antecedentes, la forma como están conformadas, finalmente se propone en este trabajo una estructura de cómo organizar aplicaciones en módulos gradle para un desacoplamiento fuertemente entre ellos, dentro del proyecto de manera que facilita trabajo al equipo de desarrolladores que estén realizando un proyecto dentro de una organización, convirtiéndolo en un producto altamente escalable, mantenible y testeable, y garantizando demás atributos de calidad en el software. Se probará el proyecto usando tecnologías como: Spring Boot y Gradle para probar un Back-end que preste un servicio con un API REST.

Palabras clave: Arquitectura de software; Programación; Buenas prácticas; gestión en desarrollo de software

Use of Clean Architecture to work with best practices in the construction of Java applications.

ABSTRACT

This paper proposes the creation of Enterprise Java applications applying best practices using software clean architecture. An overview about clean architecture and their antecedents, the way they are made, finally a structure is proposed in this work on how to organize applications in Gradle modules for a strong decoupling between then into the project to facilitate work to developers team who are carrying out a project within an organization, turning it into a highly scalable, maintainable and testable product, and guaranteeing other quality attributes in the software. The project will be tested using technologies such as: Spring Boot and Gradle to test a Back-end that provides a service with a REST API.

Keywords: Software Architecture; Programming; Best practices; Software development management

Recibido: 08/10/2020 Aceptado: 25/11/2020
Correspondencia: (*) julio_martinez54172@elpoli.edu.co

ISSN: 2711-3760
https://doi.org/10.47185/27113760.v1n1.7

1. INTRODUCCIÓN

En las empresas todos quieren aplicar buenas prácticas en sus proyectos software, pero tal vez no sepan como iniciar con la estructura de un proyecto. En este trabajo se propone una estructura de módulos con buenas prácticas, basado en arquitecturas limpias, para que las organizaciones puedan seguir un estándar y mejorar tiempos de producción en el desarrollo de sus aplicaciones software en los equipos de trabajo y por supuesto, el proceso. La ventaja es que el proyecto será mantenible para todos los implicados y para futuros integrantes que lleguen nuevos.

Las arquitecturas limpias traen consigo ventajas. Por ejemplo, según Sánchez (2016), los objetos encargados de la coordinación entre objetos de la capa de dominio y datos son los Casos de Uso, también denominados Interactors. Esta definición está más cercana la jerga funcional y se entiende mejor. La principal diferencia es que, en Arquitecturas Limpias, los objetos que coordinan (Casos de uso) se encuentran ubicados dentro de la Capa de Dominio y aquí si tienen una definición más concreta y clara. Por lo tanto, un caso de uso se necesita cuando se tiene que coordinar una acción requerida por el usuario.

Una Arquitectura limpia se puede aplicar o implementar en muchas plataformas, en este artículo, se trata para Aplicaciones empresariales Java con el Framework Spring boot y el gestor Gradle. Se propone una estructura en donde se organicen los proyectos empresariales con Java, de manera que sean escalables, mantenibles, fácil de realizar pruebas, entre otros atributos de calidad.

A continuación, se mencionan algunos trabajos relacionados en los que se ha trabajado con el uso de Arquitecturas limpias. Por ejemplo, el caso de Bui (2017), realiza en su trabajo de grado de la Universidad Metropolia de Helsinki en el que propone el trabajo "Programación reactiva y arquitectura limpia en el desarrollo de Android", el cual busca encontrar una buena arquitectura que pueda ser usada en proyectos posteriores para un equipo de trabajo, también hace refactorización de una aplicación existente usando arquitectura limpia, con uso de inyección de dependencias y programación reactiva. Por su parte, Solanes (2018), en su trabajo de investigación, desarrolla una aplicación Android para un restaurante aplicando principios de Arquitectura limpia y su implementación. Bourkhary (2019), el cual propone probar y evaluar un modelo arquitectónico y una técnica de gestión de estados para el framework flutter, un framework para el desarrollo de aplicaciones móvil bastante robusto utilizado comúnmente para desarrollo de aplicaciones móviles para iOS y Android. Es el caso del trabajo de Rodriguez (2019), quien propone en su trabajo de fin de máster llamado "Un caso práctico de Mobile-D", como su nombre lo dice, un caso práctico de mobile-D, como metodología ágil, en el desarrollo de aplicaciones móviles y utiliza Arquitectura limpia para la aplicación desarrollada; utiliza Django como tecnología. Es el caso de Knill (2019), propone en su trabajo de investigación sobre Refactoring con Arquitectura limpia, en donde demuestra cómo puede refactorizar una aplicación existente con Arquitectura Limpia sin tener que iniciar el proyecto desde cero. La mayoría de los proyectos mencionados anteriormente son para uso en aplicaciones móviles, en el caso del presente artículo será para aplicaciones web.

2. MARCO TEÓRICO

Para un mejor entendimiento del presente artículo, se presentan algunos conceptos relacionados.

2.1 Arquitectura de Software

La arquitectura de software es una forma de pensar sobre los sistemas informáticos, por ejemplo, su configuración y diseño. Por sistemas informáticos, nos referimos al hardware, el software y los componentes de comunicación (Herzog, 2015). Un conjunto de componentes reunidos no nos proporciona una solución de problema (Bass y Clements 2012). Debemos imponer una topología para la interacción y comunicación sobre ellos y garantizar los componentes tanto integrar (comunicarse físicamente) así como interoperar (lógicamente comunicarse) (Kaisler 2005).

2.2 Actividades de la Arquitectura de Software

La arquitectura de software se compone de una serie de actividades de arquitectura (que cubren todo el ciclo de vida arquitectónico) y una serie de actividades generales de arquitectura (que apoyan las actividades) (Yang. 2016).

En las siguientes secciones, proporcionamos una breve descripción sobre actividades y procesos de arquitectura de software.

Las actividades específicas de arquitectura de software se componen de cinco ítems:

- El análisis arquitectónico (AA) define los problemas de una arquitectura debe resolver. El resultado de esta actividad es un conjunto de arquitecturas requisitos significativos (ASR) (Hofmeister, 2007)

- Architectural Synthesis (AS) propone una arquitectura candidata soluciones para abordar los ASR recopilados en AA, por lo que esta actividad pasa del problema al espacio de la solución (Hofmeister, 2007)

- La evaluación arquitectónica (AE) asegura que la arquitectura las decisiones de diseño que se toman son las correctas, y el candidato Las soluciones arquitectónicas propuestas en AS se miden contra el ASR recogidos en AA (Hofmeister, 2007.

- La implementación arquitectónica (AI) realiza la arquitectura mediante creando un diseño detallado (Tang A, 2010).

- El Mantenimiento y Evolución de la Arquitectura (AME) es cambiar una arquitectura con el propósito de arreglar fallas y La evolución es responder a los nuevos requisitos a nivel arquitectónico (Postma A, 2004).

2.3 Arquitectura Hexagonal

Esta arquitectura se suele representar con forma de hexágono, pero la forma (hexagonal) o cantidad de lados no interesa. También llamada: Patrón puertos y adaptadores. Tiene como principal motivación separar nuestra aplicación en distintas capas o regiones con su propia responsabilidad. De esta manera consigue desacoplar capas de nuestra aplicación permitiendo que evolucionen de manera aislada. Además, tener el sistema separado por responsabilidades nos facilitará la reutilización (Salguero, 2018).

2.4 Arquitectura Limpia: Definición

La arquitectura limpia es una filosofía de diseño de software que separa los elementos de un diseño en niveles de anillo. La regla principal de la arquitectura limpia es que las dependencias de código solo pueden provenir de los niveles externos hacia adentro. El código en las capas internas no puede tener conocimiento de las funciones en las capas externas. Las variables, funciones y clases (cualquier entidad) que existen en las capas externas no pueden mencionarse en los niveles más internos. Se recomienda que los formatos de datos también permanezcan separados entre niveles.

La Arquitectura Limpia se refiere a organizar el proyecto para que sea fácil de entender y cambiar a medida que el proyecto crece. Esto no sucede por casualidad. Se necesita planificación intencional (Barrios, 2020).

La arquitectura limpia, gira entorno a los principios SOLID, de los que se hablará más adelante.

2.5 Git

Git es un sistema de control de versiones distribuido gratuito y de código abierto diseñado para manejar todo, desde proyectos pequeños a muy grandes, con velocidad y eficiencia.

Git es fácil de aprender y ocupa poco espacio con un rendimiento increíblemente rápido. Supera a las herramientas SCM como Subversion, CVS, Perforce

y ClearCase con características como bifurcaciones locales económicas, áreas de preparación convenientes y múltiples flujos de trabajo (git, 2020).

2.6 Gradle

(Gradle, 2020) define Gradle como una herramienta de automatización de compilación de código abierto centrada en la flexibilidad y el rendimiento. Los scripts de compilación de Gradle se escriben utilizando Groovy o Kotlin DSL:

- Altamente personalizable: Gradle se modela de una manera que es personalizable y extensible de las formas más fundamentales.

- Rápido: Gradle completa las tareas rápidamente reutilizando los resultados de ejecuciones anteriores, procesando solo las entradas que cambiaron y ejecutando tareas en paralelo.

- Potente: Gradle es la herramienta de compilación oficial para Android y viene con soporte para muchos idiomas y tecnologías populares.

En aplicaciones Java Gradle cumple algunas características como: Compilación incremental para Java, evitación de compilación para Java, soporte Groovy integrado, soporte Scala integrado, soporte integrado para herramientas de calidad de código JVM, empaquetado y distribución para JAR, WAR y EAR, publicación en repositorios Maven, publicación en Repositorios Ivy, integración de Ant.

2.7 SOLID

SOLID es el acrónimo de cinco (5) principios para combatir arquitecturas de software difíciles de mantener y mejorar el desarrollo de software, y facilitar su lectura, fueron propuestos por Robert C. Martin:

- S: Single Responsibility Principle
- O: Open-Closed Principle
- L: Liskov Substitution Principle
- I: Interface Segregation Principle
- D: Dependency Inversion Principle

En Chebanyuk (2016), se explican estos cinco (5) principios de manera detallada.

3. METODOLOGÍA

Para la elaboración del presente trabajo, se puede resumir en los siguientes métodos o procedimientos:

3.1 Revisión

Se realizó una revisión de trabajos relacionados con anterioridad sobre arquitecturas limpias, en donde la mayoría están hechos para desarrollo de aplicaciones en empresas para aplicaciones móviles con Java/Android, y algunos de ellos son un refactoring total de las arquitecturas, obteniendo al final productos exitosos en la fase de desarrollo. También se exploraron conceptos y terminologías principales referentes al proyecto como: Arquitectura de software, Arquitectura limpia, Arquitectura hexagonal, la cual es un tipo de arquitectura limpia; también sobre los principios SOLID, que están íntimamente ligados con una Arquitectura limpia, y herramientas como GIT y Gradle para la administración del plugin para la gestión de proyectos Java que se obtiene en este trabajo.

Se establece que la Arquitectura limpia surge de mejora de otras arquitecturas, de las que ella tiene antecedentes. Se inicia en el año 2005, con la Arquitectura Hexagonal, también llamada Patrón Puertos y Adaptadores, la cual menciona implícitamente dos (2) capas: la capa de lógica de negocios y la de mecanismos de entrega e infraestructura; más adelante en el año 2008 surge una mejora o evolución, llamada Arquitectura Onion, la cual plantea dos capas adicionales: capa de servicios de aplicación, servicios de dominios y modelos de dominios. Finalmente se propone Arquitectura limpia, realizando una propuesta más capas.

3.2 Arquitectura Limpia

Lo que se busca con Arquitectura limpia es que la complejidad disminuya a medida que evoluciona el desarrollo de un producto software.

ISSN: 2711-3760
https://doi.org/10.47185/27113760.v1n1.7

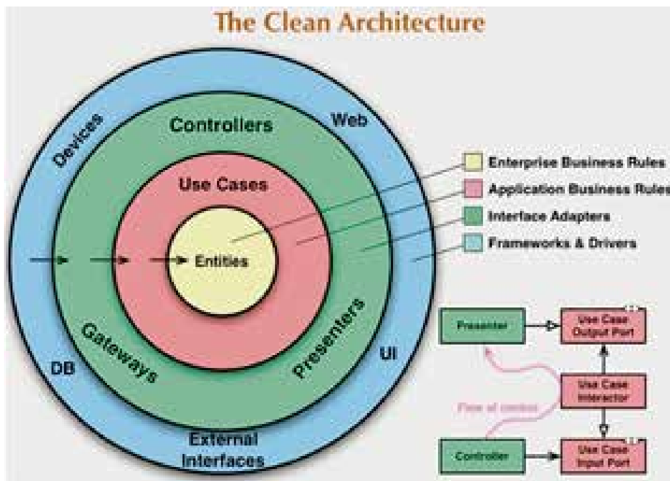
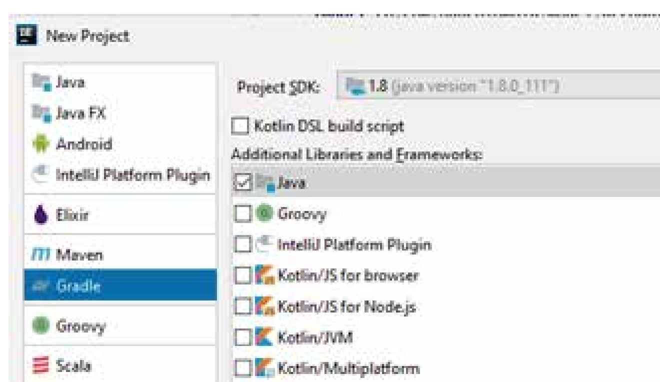


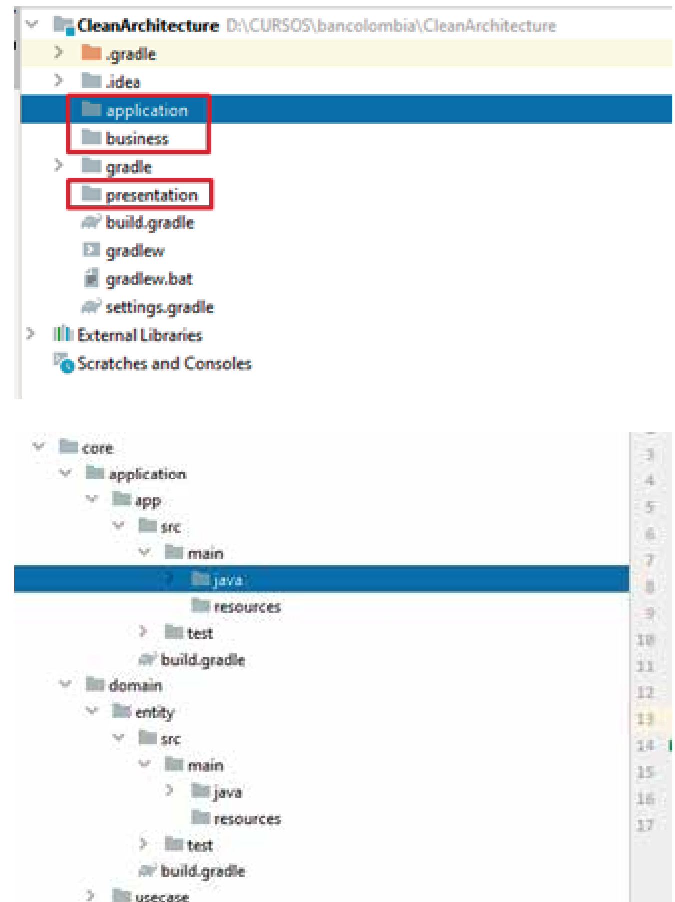
Figura 1. *Diagrama Arquitectura limpia. Tomado de: https://robertogarrido.com/arquitectura-limpia-en-ios/*

La figura anterior (Figura 1.) muestra un diagrama de Arquitecturas limpias (Clean Architecture). En donde su autor, Robert. C. Martin, presenta un modelo conceptual para lograr la separación de responsabilidades mediante capas correctamente definidas. En donde la capa más interna es la de Entities, pero el flujo de dependencias va de afuera hacia dicha capa más interna, no hay flujo de manera contraria; una capa interior e inferior no deberá conocer ni depender de una capa exterior.

La arquitectura que se plantea en este artículo es para el desarrollo con aplicaciones JavaTM Empresariales. La arquitectura se puede separar a nivel de paquetes o a nivel de módulos, para esta propuesta se hace a nivel de módulos.

Se hace la propuesta inicial de organizar el proyecto de la siguiente manera:





Finalmente podríamos plantear como sigue en el siguiente item.

3.3. Estructuración del proyecto.

3.3 Estructuración del proyecto en módulos

La estructuración final después de la serie de pasos analizados anteriores quedaría de la siguiente forma:

Se tiene como estructura dos directorios principales: core y presentation. En el core están los módulos de: application, que es donde se ensamblan distintos módulos y se resuelven depencias, y está clase main, y el domain que contiene los entities y casos de uso; en presentation está lo principal del proyecto, y están los módulos: de base de datos y api rest.

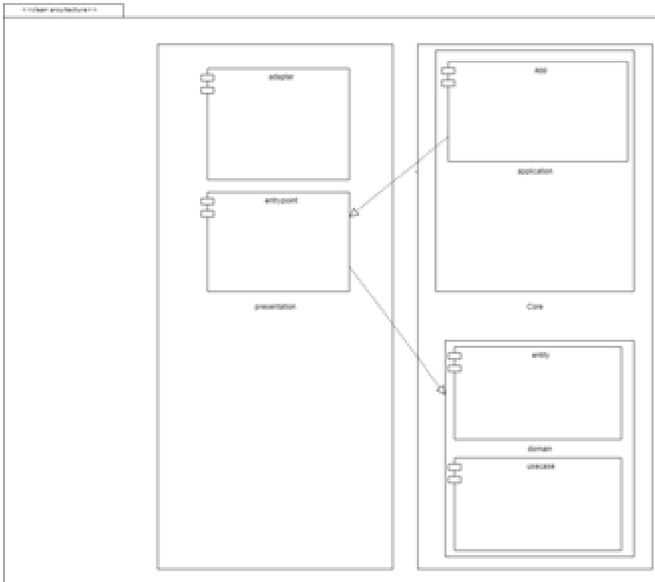
La razón de que se determina esta estructura final es mejorarla de la manera posible que se entienda y no organice el proyecto en donde tienda a amontonar clases en un solo paquete o muchos paquetes en un solo módulo y sea de la mejor forma

ISSN: 2711-3760
https://doi.org/10.47185/27113760.v1n1.7

legible y entendible por el equipo de desarrollo que se enfrente al proyecto.

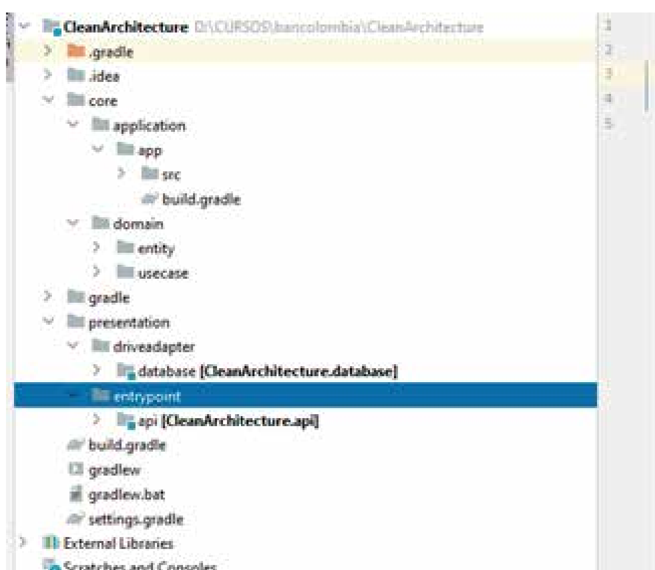
4. RESULTADOS Y/O DISCUSIÓN

La arquitectura de alto nivel queda:



El anterior diagrama de componentes se muestra para la comprensión de la organización del proyecto de forma modular de manera ilustrativa a grandes rasgos. Lo que se ve reflejado en el proyecto que se está desarrollando.

Organizado en el IDE (Integrated Development Enviroment) de desarrollo se vería algo como muestra la siguiente figura:



En la sección 3.3 se explica la estructura del proyecto de cada uno de los módulos creados y su estructura u organización interna.

Se tiene una estructura definida y configuración final de un proyecto que cumple con algunas ventajas como la agilización en la creación de clases Java y estructuras en equipos de desarrollo, por la generación automática de código y el orden con las buenas prácticas de una arquitectura limpia.

El presente artículo se enfatiza en implementación de Clean Architecture para aplicaciones o proyectos con Java usando el framework Spring boot; pero estas son independientes a frameworks, lo cual es uno de sus beneficios, además es independiente también a UI, bases de datos o algún otro agente externo.

5. CONCLUSIONES

Se pudo ver las características de Arquitecturas Limpias (Clean Architecture), las ventajas que tienen su uso al momento de querer implementarlas en proyectos a gran escala a nivel de organizaciones.

Es importante adoptar este tipo de herramientas en el momento de querer realizar nuevos proyectos de desarrollo de software o incluso se pueden hacer migraciones de proyectos existentes que facilitarían la mantenibilidad y garantizarían un sinnúmero de atributos de calidad en el software y beneficios.

La organización del proyecto es importante y queda acorde al seguimiento de Clean Architecture. Se mostró una diagramación de la arquitectura a alto nivel que ilustra la organización para mejor entendimiento, posteriormente se mostró un proyecto real en un IDE en donde se podrá apreciar aún más concretamente y trabajar con esa estructura en los proyectos empresariales deseados. Se puede afirmar que no tiene mucho sentido utilizar este tipo de arquitectura para proyectos de muy baja envergadura.

Como se pudo notar, las arquitecturas limpias son la evolución a anteriores arquitecturas propuestas,

y tienen notables mejoras a dichas antecesoras para mejorar falencias que tal vez tienen las otras en comparación con lo actual y es posible que puedan evolucionar con el tiempo a medida que es ensayada e implementada.

Como trabajo futuro se pretende incorporar un plugin de Gradle que genere automáticamente la estructura ya propuesta en paquetes con sus respectivas clases del proyecto que se desee realizar con aplicaciones comunes o reactivas con Java.

ISSN: 2711-3760
https://doi.org/10.47185/27113760.v1n1.7

REFERENCIAS

Barrios, B. Arquitectura Limpia para el resto de nosotros. Obtenido de: https://medium.com/@BryanSBarrios/arquitectura-limpia-para-el-resto-de-nosotros-440a4fba4d92 (enero, 2020).

Bass L, Clements P, Kazman R (2012) Software architecture in practice. Addison-Wesley, USA.

Bourkhary, S. (2019) (https://drive.google.com/file/d/1E3CWd83CogRirIFbobJ8LEmexlIbQjOU/view).

Bui, T. (2017). Reactive programming and clean architecture in Android development.
Chebanyuk, E. (2016). An Approach to Class Diagrams Verification According to SOLID Design Principles.

Git. Git. Obtenido de: https://git-scm.com (septiembre, 2020).

Gradle. Gradle. Obtenido de: https://docs.gradle.org/current/userguide/userguide.html (septiembre, 2020).

Herzog J (2015) Software architecture in practice third edition written by len bass, paul clements, rick kazman. ACM SIGSOFT Software Engineering Notes 40: 51-52.

Hofmeister C, Kruchten P, Nord RL, Obbink H, Ran A, et al. (2007) A general model of software architecture de-sign derived from five industrial approaches. J Syst Software 80: 106-126.

Knill, M. Refactoring to clean architecture. Obtenido de: http://courses.cecs.anu.edu.au/courses/CSPROJECTS/19S2/finalTalks/u6052043.pdf (septiembre, 2019).

Kaisler SH (2005) Software paradigms. John Wiley & Sons, USA.

Rodriguez, F. (2019). A Clean Approach to Flutter Development through the Flutter Clean Architecture Package.

Salguero, E. Arquitectura Hexagonal. Obtenido de: https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f (junio, 2018).

Sanchez, J. ¿Por qué utilizo clean architecture?. Obtenido de: http://xurxodev.com/por-que-utilizo-clean-architecture-en-mis-proyectos/ (julio, 2016).

Systems and software engineering - architecture description. ISO/IEC/IEEE 42010. 2011.- Software engineering – software life cycle processes – maintenance. ISO/IEC 14764:2006. - Postma A, America P, Wijnstra JG (2004) Component replacement in a longliving architecture: the 3RBDA approach. Proceedings. Fourth Working IEEE/IFIP Conference. pp: 89-98.].

Tang A, Avgeriou P, Jansen A, Capilla R, Babar MA (2010) A comparative study of architecture knowledge management tools. J Syst Software 83: 352-370.

Solanes, M. (2018). MyEMenu. Implementant Clean Architecture.

Tang A, Avgeriou P, Jansen A, Capilla R, Babar MA (2010) A comparative study of architecture knowledge management tools. J Syst Software 83: 352-370.

Yang C, Liang P, Avgeriou P (2016) A systematic mapping study on the combination of software architecture and agile development. J Syst Software 111: 157-184.

Wagener, E. (2019). A guide to building clean architectures for the web.