

**DISEÑO DE UNA ARQUITECTURA DE SOFTWARE EN EL PROCESO DE EVALUACIÓN Y HABILITACIÓN POR PARTE
DE LA SECRETARÍA DE SALUD DE LA GOBERNACIÓN DEL PUTUMAYO A LOS PRESTADORES DE SERVICIO
DE SALUD DEL DEPARTAMENTO DEL PUTUMAYO**

AUTOR:

EDWARD SAMIR ALEGRIA SALAZAR

UNIVERSIDAD DE PAMPLONA

INGENIERIA DE SISTEMAS

DEPARTAMENTO DE INGENIERIA ELECTRICA, ELECTRONICA SISTEMAS Y COMUNICACIONES

FACULTAD DE INGENIERIA Y ARQUITECTURA

PAMPLONA NORTE DE SANTANDER

TESIS DE GRADO

2022

**DISEÑO DE UNA ARQUITECTURA DE SOFTWARE EN EL PROCESO DE EVALUACIÓN Y HABILITACIÓN POR PARTE
DE LA SECRETARÍA DE SALUD DE LA GOBERNACIÓN DEL PUTUMAYO A LOS PRESTADORES DE SERVICIO
DE SALUD DEL DEPARTAMENTO DEL PUTUMAYO**

AUTOR:

EDWARD SAMIR ALEGRIA SALAZAR

TRABAJO DE GRADO PARA OPTAR POR EL TITULO DE:

INGENIERO DE SISTEMAS

DIRECTOR:

JORGE OMAR PORTILLA JAIMES

UNIVERSIDAD DE PAMPLONA

INGENIERIA DE SISTEMAS

DEPARTAMENTO DE INGENIERIA ELECTRICA, ELECTRONICA SISTEMAS Y COMUNICACIONES

FACULTAD DE INGENIERIA Y ARQUITECTURA

PAMPLONA NORTE DE SANTANDER

TESIS DE GRADO

2022

DEDICATORIA.

Dedicado a toda mi familia en especial a mis padres por haberme forjado como la persona que soy en la actualidad; muchos de mis logros se los debo a ustedes entre los que se incluye este, me enseñaron la

importancia de ser una persona responsable, dedicado, disciplinado, constante y un excelente ser humano para así formarme como un ciudadano de bien. Al cuerpo de docentes y compañeros que siempre me apoyaron y creyeron en mi capacidad durante este largo proceso, y que fueron mi motivación e inspiración para seguir adelante.

Edward Samir Alegría Salazar

AGRADECIMIENTOS.

Quiero agradecer a toda mi familia que, en todo momento, me apoyaron para no rendirme, a mi Madre Aida Lucia Salazar, que siempre estuvo apoyándome en las diferentes circunstancias de la vida durante

este proceso formativo y personal, a mi padre, Luis Eduardo Alegría Flórez, por su apoyo incondicional, a mis compañeros por todo el trabajo en equipo.

Agradecer de manera especial a mi Director de trabajo de grado Jorge Omar Portilla, por su excelente profesionalismo al compartirme su sabiduría, sus consejos para la construcción personal, por la paciencia y ánimos que me dio a lo largo de este trabajo de grado, por la calidad de ser humano que es y por la motivación constante en cualquier eventualidad.

Finalmente, a la Universidad de Pamplona, por permitirme ser parte de esta familia educativa, que, mediante un amplio grupo de docentes, me formaron como profesional líder para la construcción de nuevos proyectos y tener la capacidad de resolver múltiples problemas para la sociedad, siempre de la manera más ética y respetuosa.

A todos muchas gracias.

TABLA DE CONTENIDO

1	INTRODUCCIÓN	16
2	CAPITULO I.....	18
2.1	GENERALIDADES DE LA ENTIDAD.....	18
2.1.1	Datos generales:	18
2.1.2	Ubicación de la entidad pública.....	18
2.1.3	Misión y visión de la entidad	18
2.1.3.1	Misión	18
2.1.3.2	Visión.....	19
2.1.4	Relación de la entidad pública con la sociedad	19
2.2	PLANTEAMIENTO DEL PROBLEMA.....	19
2.2.1	Descripción del problema:	20
2.2.2	Formulación del problema	20
2.3	OBJETIVOS	21
2.3.1	Objetivo general.....	21
2.3.2	Objetivos Específicos:	21
2.4	JUSTIFICACIÓN	21
2.5	DELIMITACIÓN	22
2.5.1	Espacio:.....	22
2.5.2	Tiempo:	22
2.5.3	Contenido:.....	23
2.5.4	Alcance:	23
3	CAPITULO II	23
3.1	MARCO TEORICO.....	23
3.1.1	ANTECEDENTES:	23
3.1.1.1	Estado del arte:.....	23
3.1.2	ARQUITECTURA DE SOFTWARE.....	25
3.1.2.1	Historia.....	25

3.1.2.2	Definición de arquitectura de software	26
3.1.2.3	Importancia de la arquitectura de software	27
3.1.3	TIPOS DE ARQUITECTURA DE SOFTWARE	28
3.1.3.1	Estilos de arquitectura.....	28
3.1.3.2	Cliente/Servidor	29
3.1.3.3	Estructura de un cliente servidor.....	31
3.1.3.4	Arquitectura en capas.....	33
3.1.3.4.1	Estructura de una arquitectura en capas.....	34
3.1.3.5	Arquitectura monolítica	39
3.1.3.5.1	Estructura de una arquitectura monolítica	40
3.1.3.6	Arquitectura de Micokernel	41
3.1.3.6.1	Estructura de una arquitectura microkernel	42
3.1.3.7	Arquitectura orientada a servicios (SOA).....	44
3.1.3.7.1	Estructura de una arquitectura orientada a servicios (SOA).....	45
3.1.3.8	Arquitectura de microservicios	46
3.1.3.8.1	Estructura de una arquitectura microservicios	47
3.1.3.9	Arquitectura impulsada por eventos (EDA)	49
3.1.3.10	Transferencia de estado representacional (REST).....	52
3.1.4	PATRONES DE DISEÑO	55
3.1.4.1	Clasificación de patrones de diseño	56
3.1.4.1.1	Orientado a Subsistemas de componentes	57
3.1.4.1.2	Orientado a clases y objetos.....	62
3.2	ANÁLISIS COMPARATIVO DE TRES TIPOS DE ARQUITECTURA	68
3.2.1	Arquitectura de microservicios	68
3.2.1.1	Ventajas y Desventajas	70
3.2.1.2	Uso del patrón de microservicios.....	71
3.2.2	Arquitectura basada en capas.....	72
3.2.2.1	Ventajas y Desventajas	73

3.2.2.2	Uso del patrón basado en capas	74
3.2.3	Arquitectura de Microkernel.....	75
3.2.3.1	Ventajas y Desventajas	75
3.2.3.2	Uso de la Arquitectura Microkernel	76
3.3	USO DE LA ARQUITECTURA BASADA EN CAPAS (USO DE FRAMEWORK NESTJS) 76	
3.3.1	Controladores.....	79
3.3.2	Capa de Servicio	79
3.3.3	Capa de acceso a los datos	79
4	CAPITULO III.....	83
4.1	ARQUITECTURA EN CAPAS	83
4.1.1	¿Por qué la arquitectura en capas?	83
4.2	PATRÓN DE DISEÑO.....	85
4.2.1	Patrón de Diseño Data Access Object (DAO)	85
4.2.2	Patrón de Diseño Data Transfer Object (DTO)	85
4.2.2.1	¿Por qué el uso del patrón de diseño DTO?.....	86
4.3	DESARROLLO DEL PROYECTO	86
4.4	SOLUCIÓN OBJETIVO ESPECIFICO UNO	86
4.5	SOLUCIÓN OBJETIVO ESPECIFICO DOS.....	88
4.6	SOLUCIÓN OBJETIVO ESPECIFICO TRES	91
4.6.1	Capa de presentación Secretaría de Salud de la Gobernación del Putumayo	94
4.6.2	Capa de Servicio (lógica de negocio) Secretaría de Salud de la Gobernación del Putumayo	104
4.6.3	Capa de Base de Datos (Persistencia de Datos).....	113
4.7	VALIDEZ DE LA ARQUITECTURA EN CAPAS	114
4.1	CONCLUSIONES	123
4.1	REFERENCIAS BIBLIOGRAFICAS.....	123

TABLA DE ILUSTRACIONES

Ilustración 1: Arquitectura Cliente - Servidor	29
Ilustración 2: Múltiples servidores por cliente.....	31
Ilustración 3: Estructura de una aplicación cliente - servidor	32
Ilustración 4: Arquitectura en capas	33
Ilustración 5: Comunicación entre capa presentación y lógica.....	35
Ilustración 6: Comunicación entre capa lógica y acceso a datos	35
Ilustración 7:Flujo capa lógica y acceso a datos	36
Ilustración 8: Respuesta lógica de negocio a capa presentación.....	37
Ilustración 9: Resultado de la consulta	37
Ilustración 10: Salto indebido entre capas	39
Ilustración 11: Arquitectura Monolítica.....	40
Ilustración 12: Proceso de compilación	41
Ilustración 13: Arquitectura Microkernel	42
Ilustración 14: API del microkernel.....	43
Ilustración 15: Implementando un plug-in.....	44
Ilustración 16: Consumo de servicios	45
Ilustración 17: Arquitectura Monolítica.....	47
Ilustración 18: Arquitectura de Microservicios	48
Ilustración 19: Comunicación entre Microservicios	49
Ilustración 20: Arquitectura EDA	50
Ilustración 21: Llamada directa a una operación asíncrona.....	52
Ilustración 22: Procesamiento del lado del cliente	54
Ilustración 23: Modelo de Patrón de Diseño.....	56
Ilustración 24: Patrones de Diseño	57
Ilustración 25: Diagrama Arquitectura de Microservicios.....	68
Ilustración 26: Diagrama Topología API REST	69
Ilustración 27: Acceso de peticiones con capas cerradas.....	72

Ilustración 28: Acceso de peticiones con capas abiertas	73
Ilustración 29: Diagrama Arquitectura Microkernel.....	75
Ilustración 30: Capas NestJS	78
Ilustración 31: Flujo entre capas	80
Ilustración 32: Comunicación entre capas	82
Ilustración 33: Implementación modelo en capas.....	91
Ilustración 34: Capa de presentación	92
Ilustración 35: Views	93
Ilustración 36: View-Home y View-Menu	96
Ilustración 37: View-control-usuario y View-prestador	97
Ilustración 38: View-indicadores.....	97
Ilustración 39: View-login y View-password.....	98
Ilustración 40: View-Rol-Pamec.....	99
Ilustración 41: View-Rol-Sic	100
Ilustración 42:View-Rol-SP.....	101
Ilustración 43: View-Rol-Reso	102
Ilustración 44: Servicios	103
Ilustración 45:Guards.....	104
Ilustración 46:Componente Autenticación	106
Ilustración 47: Componente Guard.....	107
Ilustración 48:Componente Rol	108
Ilustración 49: Componente SIC.....	109
Ilustración 50: Componente Pamec	110
Ilustración 51: Componente SP	110
Ilustración 52:Componente Usuario	111
Ilustración 53: Componente Prestador.....	112
Ilustración 54: Componente Municipio	113
Ilustración 55: Capa de Presentación - Home.....	115

Ilustración 56: Login.....	116
Ilustración 57: Vista Admin - Logueado	117
Ilustración 58: Vista Admin - Prestador	118
Ilustración 59: Vista Admin - Sic	119
Ilustración 60: Vista Admin - Acta Sic.....	120
Ilustración 61: Vista Admin - Evaluación Sic	121

RESUMEN

La presente tesis realiza el diseño de una arquitectura de software en el proceso de evaluación y habilitación por parte de la secretaría de Salud de la Gobernación del Putumayo a los prestadores de servicio de salud del Departamento del Putumayo,

Para determinar el diseño de la arquitectura de software a implementar, fueron comparadas la arquitectura de microservicios, la arquitectura basada en capas y la arquitectura de microkernel con el propósito de demostrar la relación y diferencia, ventajas y desventajas que permiten evaluar su desempeño y seleccionar la opción más adecuada.

En el primer capítulo se describe las Generalidades de la entidad a la cual se le hará el diseño de la arquitectura del software, se exponen los datos de la empresa, se cita la misión y visión de la entidad y se hace énfasis en la necesidad a satisfacer. Se plantea la problemática, se establece el objetivo general y los objetivos específicos de la presente tesis, así como la justificación, los alcances y los límites de la misma.

El segundo capítulo refiere al marco teórico y la importancia que tiene el diseño de una arquitectura de software, los tipos y los patrones de diseño, se lleva a cabo un análisis comparativo de tres arquitecturas de software como son la arquitectura de microservicios, la arquitectura basada en capas y la arquitectura de microkernel, el cual permite seleccionar la opción mas viable para el diseño que se requiere realizar en el presente trabajo, se concluye el capítulo dos haciendo referencia al uso de la arquitectura basada en capas basándose en las funcionalidades que esta desarrolla.

En el tercer capítulo se justifica las razones del porque se escogió la Arquitectura en capas para llevar a cabo este proyecto en la secretaria de Salud de la Gobernación del Putumayo, se indica los patrones de diseño que existen y se justifica por qué se determinó como opción el patrón de diseño DTO. Por otra parte, se destaca el desarrollo del proyecto, las soluciones a cada uno de los objetivos planteados y se

lleva a cabo la validez de la arquitectura en capas, se presentan las conclusiones a las que se llegaron con el diseño de una arquitectura de software. Se responde el objetivo general y específicos de la tesis; y se enuncian las recomendaciones a la entidad.

PALABRA CLAVES: Arquitectura de software, Patrón de diseño, arquitectura de microservicios, arquitectura basada en capas y la arquitectura de microkernel

SUMMARY

This thesis carries out the design of a software architecture in the process of evaluation and authorization by the Secretary of Health of the Government of Putumayo to the health service providers of the Department of Putumayo,

To determine the design of the software architecture to be implemented, the microservices architecture, the layer-based architecture and the microkernel architecture were compared with the purpose of demonstrating the relationship and difference, advantages and disadvantages that allow evaluating their performance and selecting the most suitable option.

The first chapter describes the Generalities of the entity to which the design of the software architecture will be made, the company data is exposed, the mission and vision of the entity are cited and emphasis is placed on the need to satisfy The problem is raised, the general objective and the specific objectives of this thesis are established, as well as the justification, scope and limits of it.

The second chapter refers to the theoretical framework and the importance of the design of a software architecture, the types and patterns of design, a comparative analysis of three software architectures is carried out, such as the microservices architecture, the based on layers and the microkernel architecture, which allows selecting the most viable option for the design that is required to be carried out in the present work, chapter two concludes by referring to the use of the architecture based on extensive layers in the functionalities that this develops .

In the third chapter, the reasons why Layered Architecture was chosen to carry out this project in the Putumayo Governor's Health Secretary are justified, the design patterns that exist are indicated and it is justified why it is perfected as an option. the DTO design pattern. On the other hand, the development of the project is highlighted, the solutions to each one of the proposed objectives and the validity of the architecture in layers is carried out, the conclusions reached with the design of a software architecture

are presented. . The general and specific objective of the thesis is answered; and the recommendations to the entity are enunciated.

KEYWORDS: Software architecture, Design pattern, microservices architecture, layer-based architecture, and microkernel architecture

INTRODUCCIÓN

El presente trabajo tiene como objetivo principal diseñar una arquitectura de software para el desarrollo de una aplicación web para la secretaria de salud de la Gobernación del Putumayo.

Determinar como primera medida generar eficiencia en todos los procesos que lleva a cabo la secretaria de salud de la gobernación del putumayo en el área de sistema obligatorio de la garantía de la calidad.

Igualmente, la arquitectura de software ayudará a tener eficiencia en el procedimiento establecido que se lleva a cabo para evaluar a un prestador de servicio de salud y estos se proyecten de una forma clara y concisa, a esto se incluye el proceso de administración de usuarios que se encargan de realizar las evaluaciones pertinentes a cada prestador de servicio de salud.

La importancia de establecer un buen diseño en la arquitectura de software es que esta sea fácil de escalar en futuras modificaciones a dicha aplicación, así mismo facilita la evaluación por parte de los funcionarios al momento de llevar a cabo la evaluación a los prestadores de servicio de salud, ya que actualmente se utiliza mecanismos no óptimos para dichas evaluaciones que se realizan a los prestadores de servicio de salud.

En este sentido, se espera que, con la implementación de una arquitectura de software en la secretaria de salud de la Gobernación del Putumayo, facilite el desarrollo de una aplicación web para llevar a cabo el control de evaluaciones a los prestadores de servicio de salud.

CAPITULO I

En el desarrollo del presente trabajo de grado se establece una serie de capítulos donde como primera medida, el capítulo I, determinar las generalidades de la entidad para así conocer su misión, visión, etc., posteriormente, se realiza el desarrollo del planteamiento del problema, seguido de los objetivos tanto general como específicos, la justificación y por último la delimitación que hace indicación a abarcar el alcance, tiempo, espacio y contenido de la entidad

GENERALIDADES DE LA ENTIDAD

Datos generales:

El área de sistema obligatorio de la garantía de calidad de la secretaria de salud de la Gobernación del Putumayo es la encargada de hacer inspección, vigilancia, control, seguimiento y habilitación a los prestadores de servicio de salud en el departamento del Putumayo para que estos presten un buen servicio, cumpliendo con las normas y requisitos establecidos por el Ministerio de la Protección Social.

Ubicación de la entidad pública

La secretaria de salud de la Gobernación del Putumayo está ubicada en el Departamento del Putumayo en su capital Mocoa con su respectiva dirección Cra. 4 #8-26 en el Barrio José María Hernández

Teléfono (57+8) 4206600

Correo Electrónico: salud@putumayo.gov.co

Misión y visión de la entidad

Misión

Una entidad pública de la orden departamental encargada de dirigir el Sistema Territorial de Seguridad Social en Salud, mediante procesos de planificación, investigación, coordinación, inspección, vigilancia, control y asistencia técnica; de conformidad con los principios y normas que rigen el sistema;

respetando e incorporando las prácticas y saberes ancestrales de nuestra región; para contribuir a mejorar el nivel de vida de la población putumayense.

Visión

Como institución rectora del Sistema Territorial de Seguridad Social en Salud del Putumayo seremos destacados en el país en procesos de planificación, coordinación, vigilancia y control, con una estructura organizacional integrada, ágil, moderna y con capacidad de respuesta frente a los cambios; utilizando sistemas e información, tecnología adecuada y el mejor talento humano, que nos permita cumplir los objetivos propuestos, evaluando periódicamente nuestro desempeño mediante índices de gestión previamente definidos.

Relación de la entidad pública con la sociedad

El propósito del Gobierno Nacional en la prestación de los servicios de salud es garantizar el acceso, optimizar el uso de los recursos y mejorar la calidad de los servicios que se prestan a la población, por lo tanto, para poder ofrecer servicios de calidad a los usuarios, el Ministerio de la Protección Social ha implementado políticas en el sistema general de Seguridad Social en Salud que genere expectativas positivas a los usuarios en el acceso al portafolio de servicios que ofrecen los prestadores de servicios de salud.

Se puede afirmar que un complemento que le permite al Ministerio de Protección Social a través de las secretarías de salud departamentales y municipales para velar por el cumplimiento de estas normas es el control y vigilancia que tiene estas dependencias para garantizar de una manera integrada la salud de la población por medio de acciones de salubridad dirigidas tanto de manera individual como colectiva.

PLANTEAMIENTO DEL PROBLEMA

Descripción del problema:

La arquitectura de software define la estructura del sistema, inicialmente se desarrolla en las primeras etapas del desarrollo de software a gran profundidad representa un diseño de alto nivel del sistema el cual tiene como propósito satisfacer la calidad, seguridad, desempeño y modificabilidad que ayudará en el desarrollo de una aplicación web

La falta de una aplicación web en la secretaria de salud de la gobernación del putumayo en el área de sistema obligatorio de la garantía de la calidad, que defina los procedimientos y las condiciones de inscripción y habilitación de los prestadores de servicio de salud, así como el sistema de información de la calidad, seguridad del paciente, programa de auditoría para el mejoramiento de la atención en salud, limitan el procesamiento de entradas de información, el almacenamiento de datos relacionados con la entidad y los prestadores de servicios de salud así como la producción de reportes oportunos que permitan la identificación de hallazgos e incumplimiento de la norma.

El presente proyecto surge ante la necesidad de la secretaria de salud de la Gobernación de Putumayo, teniendo en cuenta que el Ministerio de salud y Protección social, les exige a los diferentes actores del sistema de salud colombiano, entre ellos las secretarías de salud de las diferentes regiones del país, homologar el modo de verificar los estándares y criterios para la habilitación de los servicios de salud a través de una aplicación web.

Formulación del problema

En el presente proyecto como primera medida se realizó la extracción de todas las características técnicas que se lleva a cabo para dar cumplimiento al área de sistema obligatorio de garantía de la calidad de la Secretaría de Salud de la Gobernación del Putumayo, para luego poder analizar todo lo requerido y así establecer de una forma más detallada como se estructurará el diseño de arquitectura y concluir con el diseño de una de arquitectura de software basada en capas que servirá como prototipo para llevar a cabo el desarrollo de una aplicación web.

OBJETIVOS

Objetivo general

1. Diseñar una arquitectura de software para el desarrollo de una aplicación web en el área de sistema obligatorio de la garantía de la calidad en la secretaria de salud de la gobernación del putumayo

Objetivos Específicos:

2. Extraer los datos requeridos que maneja el área de sistema obligatorio de la garantía de la calidad para el proceso de diseñar la arquitectura de software
3. Analizar los datos extraídos y establecer de una forma detallada como será su estructura para el diseño de arquitectura que tendrá la aplicación web
4. Diseñar el tipo de arquitectura de software que definirá las funcionalidades que se llevarán a cabo de acuerdo al análisis de los datos extraídos

JUSTIFICACIÓN

La implementación de una arquitectura de software para el desarrollo de una aplicación web para los departamentos en Colombia les permiten estar unificados en todo el territorio nacional en lo relacionado con el modo de verificación de las condiciones técnico administrativas, financiera, tecnológica y científica definidas en el sistema único de habilitación nacional, lo que les permite a los departamentos estar articulados entre sí para el cumplimiento de la normatividad vigente.

La secretaria de salud del Putumayo en el área de sistema obligatorio de la garantía de la calidad viene cumpliendo parcialmente con estos criterios establecidos en la resolución 3100 de 2019, por lo tanto, se requiere la implementación de una aplicación web que contenga los requerimientos establecidos conforme a la normatividad exigida por el gobierno nacional, este proceso de implementación se lleva a cabo con los conocimientos adquiridos en mi proceso de formación en Ingeniería de Sistemas y que va a facilitar la total implementación de esta resolución en esta entidad, proporcionando así las herramientas necesarias para que se cumplan con las condiciones establecidas en el Sistema Único de Habilitación, contribuyendo con ello, a brindar estándares de calidad y criterios definidos para el funcionamiento de los servicios de salud.

El propósito de este proyecto es que la secretaria de salud de la Gobernación del Putumayo en el área de sistema obligatorio de la garantía de la calidad quede con una arquitectura de software bien diseñada para así facilitar el desarrollo de una aplicación web que le permita dar cumplimiento a lo previsto en la resolución 3100 de 2019.

DELIMITACIÓN

Espacio:

El espacio donde se llevará a cabo el proyecto de diseño de una arquitectura de software en la secretaria de salud de la gobernación del putumayo específicamente en el área de sistema obligatorio de la garantía de la calidad

Tiempo:

El tiempo que se necesita para la realización del proyecto se define por un cronograma de actividades, así mismo los tiempos establecidos por la universidad de pamplona Norte de Santander, Colombia, estableciendo una revisión y finalmente la sustentación del proyecto de grado.

Contenido:

El presente proyecto se orienta en conocer como son las normas que se estipulan por el Ministerio de salud y Protección social para llevar a cabo los requisitos que necesita un prestador de servicio de salud con el fin de evaluarlo y así habilitarlo o no en la prestación del servicio

Alcance:

El diseño de una arquitectura de software una vez elaborado le permitirá a la secretaria de salud de la Gobernación del Putumayo tener un prototipo que le permita desarrollar un software o una aplicación que brinde de una manera más detallada información relacionada con los procedimientos y condiciones de los prestadores de servicio de salud y habilitación de los mismos, facilitando de esta manera el cumplimiento de los requisitos a los prestadores de servicio de salud y a los funcionarios les permita hacer seguimiento en los reportes que se generen en las visitas de vigilancia y control.

CAPITULO II

En el desarrollo del capítulo II, se abarca el marco teórico de los temas relacionados con la arquitectura del software dando como inicio una breve descripción de la historia de la arquitectura y la importancia de esta en la construcción de un software, de la misma manera se hace una recopilación sobre la definición de la arquitectura y la importancia que tiene en estructurar de una manera correcta la elaboración de un software.

MARCO TEORICO.

ANTECEDENTES:

Se contempla el estado del arte de estudios relacionados con el diseño de una Arquitectura de Software basados en diferentes autores, indicando la importancia en el desarrollo de esta.

Estado del arte:

A continuación, se exponen diferentes investigaciones que aportan un enfoque nacional e internacional dando así relación con el problema, en este caso el diseño de una arquitectura de software.

1. Nacionales:

Torres, P. (2017), Bogotá. D.C, en el presente trabajo de grado titulado: Definición de una Arquitectura de Referencia para Sistemas de Información en Redes de Sensores Inalámbricos Ubicuos basados en calidad de servicio, de la Universidad Nacional de Colombia, la investigación recoge una mirada a las principales propuestas realizadas en diferentes ámbitos por instituciones referentes - en especial en lo definido por la Unión Internacional de Telecomunicaciones – UIT/ITU, que han intentado estandarizar una arquitectura de referencia para este tipo de redes con el fin de reducir las complejidades de integración, el uso de múltiples tecnologías, la heterogeneidad de los diferentes protocolos existentes y facilitar: la interconexión, la comunicación y futuros desarrollos en este campo. Adicionalmente tiene como objetivo el desarrollo de una arquitectura de referencia aplicable a un caso de estudio de orden nacional e internacional, tomando el área de los Sistemas Inteligentes de Transporte SIT/ITS, como un sector fundamental en el desarrollo de las naciones y del cual se tiene pleno conocimiento por parte de los involucrados en la investigación.

Ruiz, H. (2018) Cartagena de Indias, desarrolla una tesis de grado titulada: Arquitectura de Software para la gestión y proyección financiera de planes de negocio, de la Universidad de Cartagena, expone el diseño de una arquitectura para un sistema que permita gestionar la información generada por los proyectos de planes de negocio que realizan los estudiantes en los Programas de Administración en la Fundación Universitaria Tecnológico Comfenalco.

2. Internacionales:

Matla, C. (2014), Ciudad de México, México., el presente proyecto de la Universidad Nacional Autónoma de México la cual lleva como título: Desarrollo de Software Guiado por la Norma ISO/IEC 29110 y

SCRUM: SIDEV V.2.0, plantea analizar los fallos que presenta Sidev v.1.0 y proponer una solución a éstos, ya sea a través de una actualización del sistema o bien mediante la creación de un sistema nuevo y para ello se basan principalmente en SCRUM y la Norma ISO/IEC 29110, generando así el proceso de un desarrollo de software que involucre prácticas pertinentes a diferentes metodologías y así generar una solución que atienda las necesidades establecidas.

Crisóstomo, P. (2020), Huancayo, Perú., el proyecto de grado: “Modelo Emergente para el desarrollo de sistemas informáticos eficientes, utilizando una Arquitectura de Microservicios en Sunat – División de Desarrollo de Software Aduanero” presenta como objetivo determinar el efecto de la utilización del modelo Emergente de Microservicios para el desarrollo de sistemas informáticos en SUNAT-DDSA con respecto al modelo monolítico, el cual plantea evaluar el tiempo de respuesta de la utilización del modelo emergente de microservicios para el desarrollo de sistemas informáticos en SUNAT-DDS, con respecto tipo de arquitectura monolítica, así mismo espera medir la disponibilidad en la utilización del modelo emergente de microservicios para el desarrollo de sistemas informáticos en SUNAT- DDSA, con respecto al modelo monolítico, ya por último, evalúa el rendimiento en la utilización del modelo emergente de microservicios para el desarrollo de sistemas informáticos en SUNAT- DDSA, con respecto al modelo monolítico.

ARQUITECTURA DE SOFTWARE

Historia

La historia de la arquitectura del software remonta a la época de los años 60 se empieza a utilizar el término cuando Edsger Dijkstra la Universidad Tecnológica de Eindhoven en Holanda en 1968 propuso que se establezca una estructura correcta de los sistemas de software antes de que se inicie la programación como tal, escribiendo código de cualquier manera. En 1972, Parnas publicó un ensayo en el que discutía la forma en que el modularidad en el diseño de sistemas podía mejorar la flexibilidad y el

control conceptual del sistema, introduciendo el concepto de Ocultamiento de información, La herencia de este concepto en la ingeniería y la arquitectura ulterior es inmensa, y se confunde estrechamente con la idea de abstracción. un año después de la sesión en que se fundara la ingeniería de software, P. I. Sharp formuló que la ingeniería era diferente a la “arquitectura”. Fred Brooks Jr y Ken Iverson llamaban arquitectura a la estructura conceptual de un sistema en la perspectiva del programador Brooks, diseñador del sistema operativo OS/360 y Premio Turing 2000, utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario”.

Definición de arquitectura de software

Como plantea (Oscar Javier Blancarte, 2020), en su libro introducción a la arquitectura de software un enfoque práctico nos dice que no existe un consenso para definir con exactitud que es la arquitectura de software, por lo contrario, existe diversas publicaciones donde se dan diferentes definiciones, lo que hace complicado decir con exactitud que es la arquitectura de software, basado en lo anterior se pueden desglosar algunas definiciones de diferentes autores como las siguientes:

“La arquitectura es un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema “

— "An introduction to Software Architecture" de David Garlan y Mary Shaw

“La Arquitectura de Software se refiere a las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos. “

— Software Engineering Institute (SEI)

“El conjunto de estructuras necesarias para razonar sobre el sistema, que comprende elementos de software, relaciones entre ellos, y las propiedades de ambos. “

— Documenting Software Architectures: Views and Beyond (2nd Edition), Clements et al,
AddisonWesley, 2010

“La arquitectura se define como la organización fundamental de un sistema, encarnada en sus componentes, sus relaciones entre sí y con el entorno, y los principios que rigen su diseño y evolución “

— ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software
Intensive Systems

“Una arquitectura es el conjunto de decisiones importantes sobre la organización de un sistema de software, la selección de los elementos estructurales y sus interfaces mediante las cuales se compone el sistema “

— Rational Unified Process, 1999

“Una arquitectura es el conjunto de decisiones importantes sobre la organización de un sistema de software, la selección de los elementos estructurales y sus interfaces mediante las cuales se compone el sistema “

— Rational Unified Process, 1999

Importancia de la arquitectura de software

La arquitectura de un sistema de software nace con los sistemas de mediana o gran envergadura, se propone como una solución para un problema determinado.

Conforme los sistemas de software van creciendo en su complejidad, bien sea por número de requerimientos o por el impacto de los mismo, se hace necesario establecer medios para el manejo de esta complejidad (Hofmeister et al, 1996).

En general, la técnica es descomponer el sistema en piezas que agrupan aspectos específicos del mismo, producto de un proceso de abstracción y que al organizarse de cierta manera constituyen la base de la solución de un problema en particular. De esta manera, la arquitectura de software puede ser vista como la estructura del sistema en función de la definición de los componentes y sus interacciones (Bass et al., 1998).

La práctica ha demostrado que resulta importante extender el concepto considerando los requerimientos y restricciones del sistema, junto a un argumento que justifique que la estructura definida satisface los requerimientos, dándole un sentido más amplio a la definición del término. (Boehm et al., 1995; Lane, 1990)

TIPOS DE ARQUITECTURA DE SOFTWARE

Estilos de arquitectura

Como señala (Oscar Javier Blancarte, 2020), comprender el estilo arquitectónico es necesario regresar a la arquitectura tradicional (construcción), donde se define que es un método específico de construcción, caracterizado por las características que lo hacen notable y se distingue por las características que hacen que un edificio u otra estructura sea notable o históricamente identificable. Por otra parte, en el software aplica exactamente igual, pues un estilo arquitectónico determina las características que debe tener un componente que utilice ese estilo, lo cual hace que sea fácilmente reconocible. De la misma forma que podemos determinar a qué periodo de la historia pertenece una construcción al observar sus

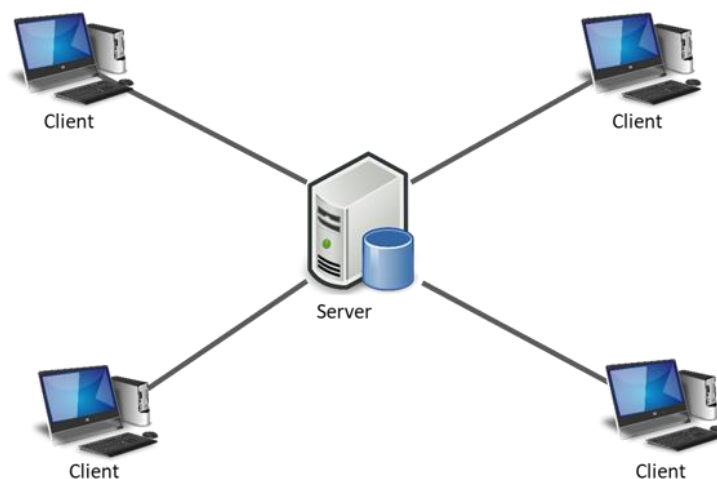
características físicas, materiales o método de construcción, en el software podemos determinar que estilo de arquitectura sigue un componente al observar sus características.

Cliente/Servidor

Según (Oscar Javier Blancarte, 2020), el estilo Cliente-Servidor es uno de los estilos arquitectónicos distribuidos más conocidos, el cual está compuesto por dos componentes, el proveedor y el consumidor.

El proveedor es un servidor que brinda una serie de servicios o recursos los cuales son consumido por el Cliente. Así mismo nos afirma que en una arquitectura Cliente-Servidor existe un servidor y múltiples clientes que se conectan al servidor para recuperar todos los recursos necesarios para funcionar, en este sentido, el cliente solo es una capa para representar los datos y se detonan acciones para modificar el estado del servidor, mientras que el servidor es el que hace todo el trabajo pesado.

Ilustración 1: Arquitectura Cliente - Servidor



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

En esta arquitectura, el servidor deberá exponer un mecanismo que permite a los clientes conectarse, que por lo general es TCP/IP, esta comunicación permitirá una comunicación continua y bidireccional, de

tal forma que el cliente puede enviar y recibir datos del servidor y viceversa. (Oscar Javier Blancarte, 2020).

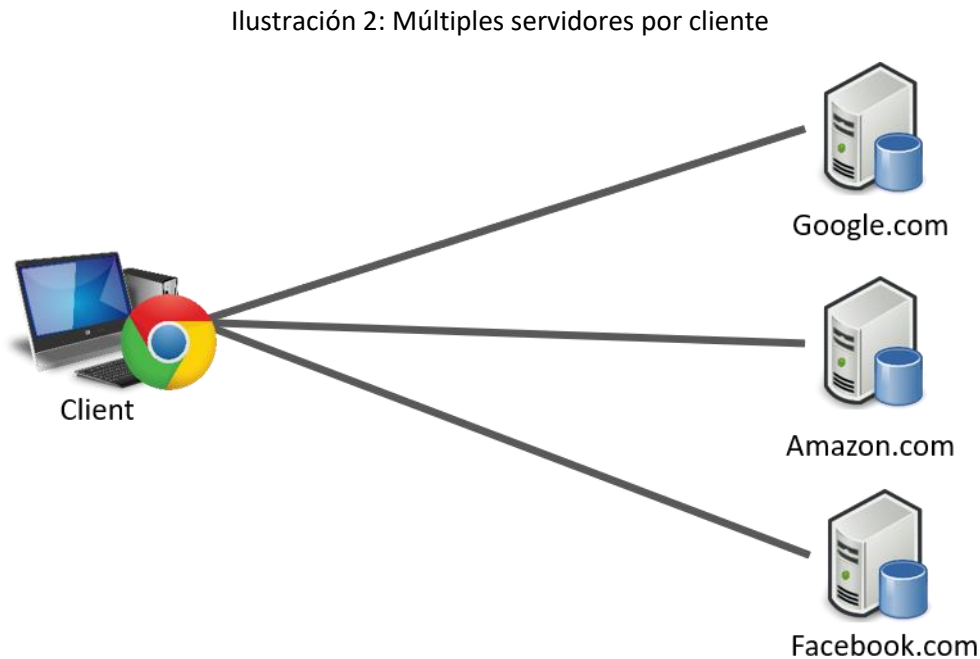
Este estilo es considerado una arquitectura distribuida debido a que el servidor y el cliente se encuentran distribuidos en diferentes equipos (aunque podrían estar en la misma máquina) y se comunican únicamente por medio de la RED o Internet. (Oscar Javier Blancarte, 2020).

En esta arquitectura, el Cliente y el Servidor son desarrollados como dos aplicaciones diferentes, de tal forma que cada una puede ser desarrollada de forma independiente, dando como resultado dos aplicaciones separadas, las cuales pueden ser construidas en tecnologías diferentes, pero siempre respetando el mismo protocolo de comunicación para establecer comunicación. (Oscar Javier Blancarte, 2020).

(Oscar Javier Blancarte, 2020), afirma que la idea central de separar al cliente del servidor radica en la idea de centralizar la información y la separación de responsabilidades, por una parte, el servidor será la única entidad que tendrá acceso a los datos y los servirá solo a los clientes del cual el confía, y de esta forma, protegemos la información y la lógica detrás del procesamiento de los datos, además, el servidor puede atender simultáneamente a varios clientes, por lo que suele ser instalado en un equipo con muchos recursos. Por otro lado, el cliente suele ser instalado en computadoras con bajos recursos, pues desde allí no se procesa nada, simplemente actúa como un visor de los datos y delega las operaciones pesadas al servidor.

Además (Oscar Javier Blancarte, 2020), determina que uno de los puntos más característicos de la arquitectura Cliente-Servidor es la centralización de los datos, pues el server recibe, procesa y almacena todos los datos provenientes de todos los clientes. Si bien los clientes por lo general solo se conectan a un solo servidor, existen variantes donde hay clientes que se conectan a múltiples servidores para

funcionar, tal es el caso de los navegadores, los cuales, para consultar cada página establece una conexión a un servidor diferentes, pero al final es Cliente-Servidor:



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

Como se puede observar en la imagen, el navegador actúa como un cliente, pero en lugar de conectarse a un solo servidor, puede conectarse a múltiples servidores, el navegador no sirve para absolutamente nada si no podemos acceder a un servidor. (Oscar Javier Blancarte, 2020).

Estructura de un cliente servidor

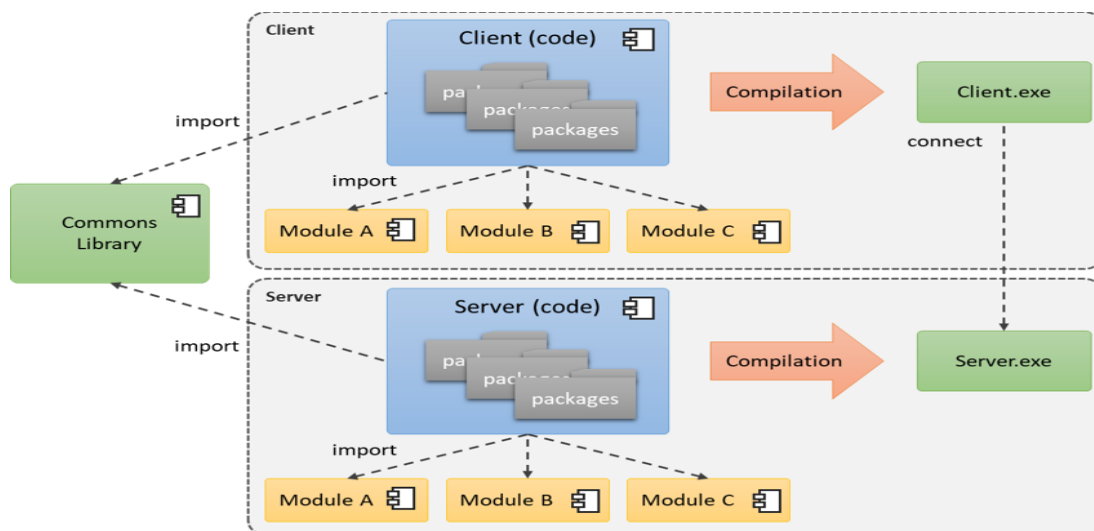
Según (Oscar Javier Blancarte, 2020), el cliente y el servidor son aplicaciones diferentes, por lo que pueden tener un ciclo de desarrollo diferente, así como usar tecnologías y un equipo diferente entre sí. Sin embargo, en la mayoría de los casos, el equipo que desarrolla el servidor también desarrolla el

cliente, por lo que es normal ver que el cliente y el servidor están contruidos con las mismas tecnologías.

Este estilo arquitectónico es construido en lo general como Monolíticos, donde cada desarrollo crea su propio ejecutable único y funciona sobre un solo equipo, con la diferencia de que estas aplicaciones no son autosuficientes, ya que existe una dependencia simbiótica entre los dos. (Oscar Javier Blancarte, 2020).

En este sentido, es normal tener 3 artefactos, el Cliente, el Servidor y una tercera librería que contiene Objetos comunes entre el servidor y el cliente, esta librería tiene por lo general los Objetos de Entidad, DTO, interfaces y clases base que se usan para compartir la información, es decir, objetos que se utilizan en las dos aplicaciones y se separan para no repetir código (Principio DRY – Don't repeat yourself), sin embargo, este tercer componente no es obligatorio que exista, sobre todo si el cliente y el servidor utilizan tecnologías diferentes o son implementados con diferentes proveedores. (Oscar Javier Blancarte, 2020).

Ilustración 3: Estructura de una aplicación cliente - servidor



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

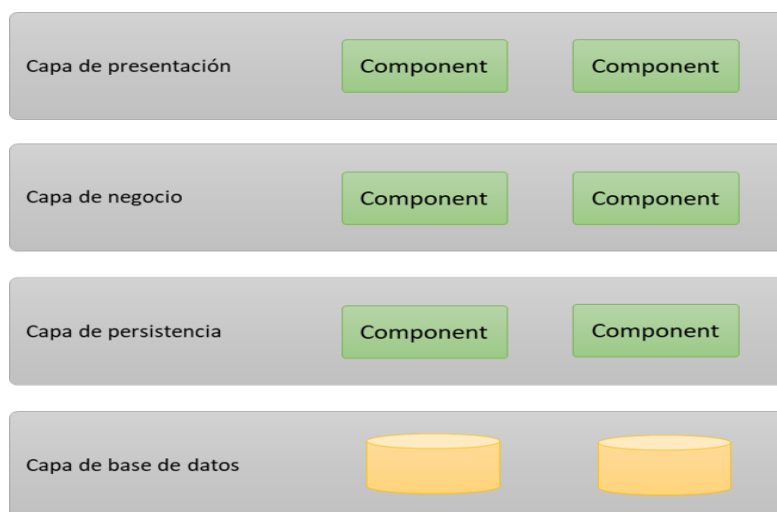
Elaborado por: Oscar Javier Blancarte, 2020

Arquitectura en capas

La arquitectura basada en capas se enfoca principalmente en el agrupamiento de funcionalidad relacionada dentro de una aplicación en distintas capas que son colocadas verticalmente una encima de otra, la funcionalidad dentro de cada capa se relaciona con un rol o responsabilidad específica. (Miguel Muñoz Serafín, 2018).

Por otro lado (Oscar Javier Blancarte, 2020), la arquitectura en capas consta en dividir la aplicación en capas, con la intención de que cada capa tenga un rol muy definido, como podría ser, una capa de presentación (UI), una capa de reglas de negocio (servicios) y una capa de acceso a datos (DAO), sin embargo, este estilo arquitectónico no define cuantas capas debe de tener la aplicación, sino más bien, se centra en la separación de la aplicación en capas (Aplica el principio Separación de preocupaciones (SoC)).

Ilustración 4: Arquitectura en capas



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

Al momento de establecer el desarrollo, la mayoría de las veces este estilo arquitectónico es implementado en 4 capas, presentación, negocio, persistencia y base de datos, sin embargo, es habitual ver que la capa de negocio y persistencia se combinan en una sola capa, sobre todo cuando la lógica de persistencia está incrustada dentro de la capa de negocio. (Oscar Javier Blancarte, 2020).

Estructura de una arquitectura en capas

En una arquitectura en capas, todas las capas se colocan de forma horizontal, de tal forma que cada capa solo puede comunicarse con la capa que está inmediatamente por debajo, por lo que, si una capa quiere comunicarse con otras que están mucho más abajo, tendrán que hacerlo mediante la capa que está inmediatamente por debajo. Por ejemplo, si la capa de presentación requiere consultar la base de datos, tendrá que solicitar la información a la capa de negocio, la cual, a su vez, la solicitará a la capa de persistencia, la que, a su vez, la consultará a la base de datos, finalmente, la respuesta retornará en el sentido inverso hasta llegar la capa de presentación. (Oscar Javier Blancarte, 2020).

Como expresa (Miguel Muñoz Serafín, 2018), la arquitectura en capas define una comunicación entre ellas la cual se define de la siguiente manera

1. Según (Miguel Muñoz Serafín, 2018), la capa de presentación solicita el ID del producto que el usuario desea comprar y se lo proporciona a la capa de lógica de negocio.

Ilustración 5: Comunicación entre capa presentación y lógica



Fuente: Adoptado de (Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft)

Elaborado por: Miguel Muñoz Serafín, 2018

2. Con base en lo dicho por (Miguel Muñoz Serafín, 2018), la capa de lógica de negocio debe verificar si aún hay existencias del producto requerido por el usuario y para esto solicita los datos del producto a la capa de acceso a datos pasándole el ID del producto a buscar.

Ilustración 6: Comunicación entre capa lógica y acceso a datos

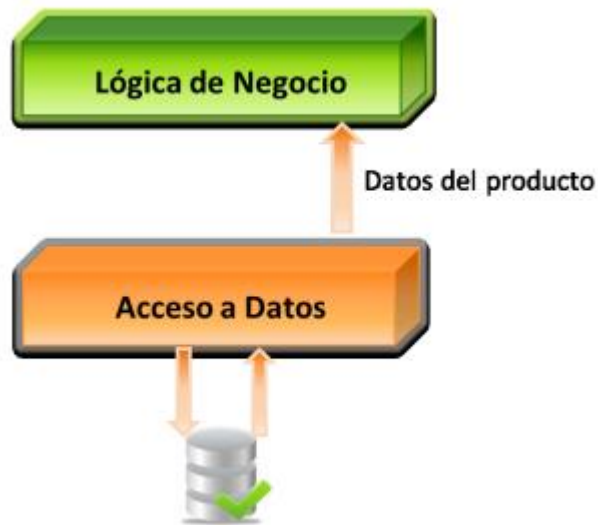


Fuente: Adoptado de (Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft)

Elaborado por: Miguel Muñoz Serafín, 2018

3. Como señala (Miguel Muñoz Serafín, 2018), la capa de acceso a datos busca la información del producto en la fuente de datos y lo devuelve a la capa de lógica de negocio.

Ilustración 7: Flujo capa lógica y acceso a datos



Fuente: Adoptado de (Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft)

Elaborado por: Miguel Muñoz Serafín, 2018

4. De acuerdo con (Miguel Muñoz Serafín, 2018), con los datos recibidos, la capa de lógica de negocio determina si el producto puede ser vendido o no y proporciona la respuesta de la consulta a la capa de presentación.

Ilustración 8: Respuesta lógica de negocio a capa presentación



Fuente: Adoptado de (Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft)

Elaborado por: Miguel Muñoz Serafín, 2018

5. Como analiza (Miguel Muñoz Serafín, 2018), la capa de presentación muestra al usuario la respuesta que la capa de lógica le ha proporcionado.

Ilustración 9: Resultado de la consulta



Fuente: Adoptado de (Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft)

Elaborado por: Miguel Muñoz Serafín, 2018

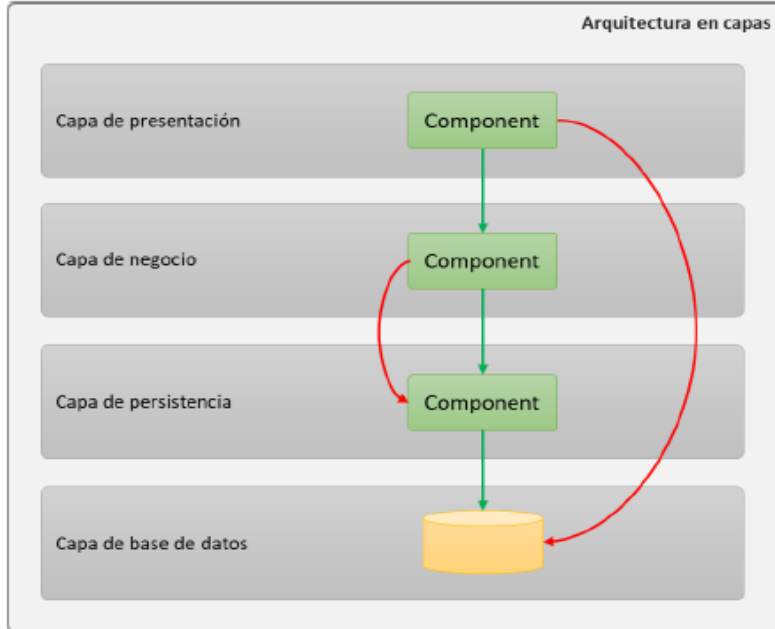
Teniendo en cuenta a (Oscar Javier Blancarte, 2020), la separación de la aplicación en capas busca cumplir con el principio de separación de preocupaciones, de tal forma que cada capa se encargue una tarea muy definida, por ejemplo, la capa de presentación solo se preocupa por presentar la información de forma agradable al usuario, pero no le interesa de donde viene la información ni la lógica de negocio

que hay detrás, en su lugar, solo sabe que existe una capa de negocio que le proporcionará la información.

Por otra parte, la capa de negocio solo se encarga de aplicar todas las reglas de negocio y validaciones, pero no le interesa como recuperar los datos, guardarlos o borrarlos, ya que para eso tiene una capa de persistencia que se encarga de eso. Por otro lado, la capa de persistencia es la encargada de comunicarse a la base de datos, crear las instrucciones SQL para consultar, insertar, actualizar o borrar registros y retornarlos en un formato independiente a la base de datos. De esta forma, cada capa se preocupa por una cosa y no le interesa como le haga la capa de abajo para servirle los datos que requiere. (Oscar Javier Blancarte, 2020).

Importante en esta arquitectura es el orden de las capas, ya que saltarse una capa para irnos sobre una más abajo suele ser un grave error, ya que bien es posible hacerlo, empezamos a crear un desorden sobre el flujo de comunicación, lo que nos puede llevar al típico anti patrón conocido como código espagueti, el cual consiste en que empezamos a realizar llamadas desde cualquier capa a otra capa, haciendo que el mantenimiento se vuelva un verdadero infierno. (Oscar Javier Blancarte, 2020).

Ilustración 10: Salto indebido entre capas



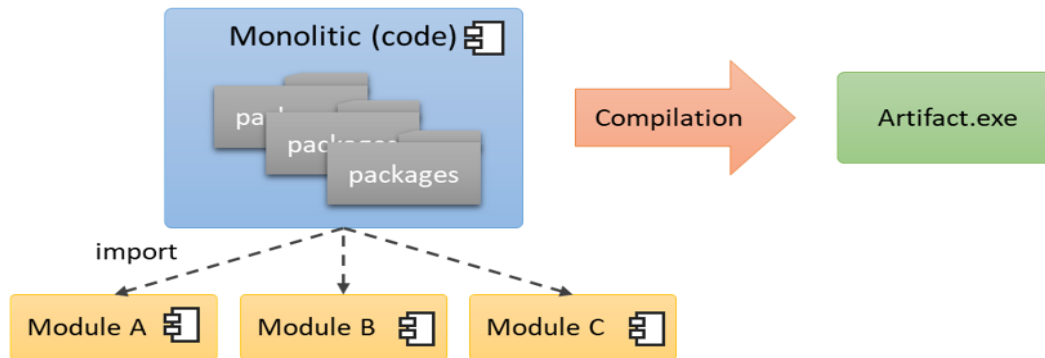
Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

Arquitectura monolítica

De acuerdo con (Oscar Javier Blancarte, 2020), el estilo arquitectónico monolítico consiste en crear una aplicación autosuficiente que contenga absolutamente toda la funcionalidad necesaria para realizar la tarea para la cual fue diseñada, sin contar con dependencias externas que complementen su funcionalidad. En este sentido, sus componentes trabajan juntos, compartiendo los mismos recursos y memoria. En pocas palabras, una aplicación monolítica es una unidad cohesiva de código.

Ilustración 11: Arquitectura Monolítica



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

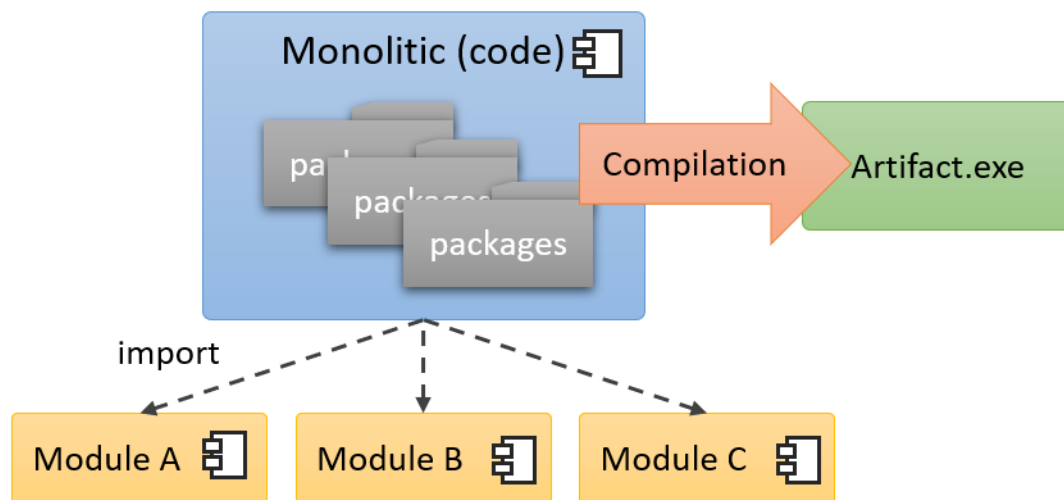
(Oscar Javier Blancarte, 2020), muestra en la ilustración 11 que se puede apreciar cómo funciona el proceso de compilación de una aplicación Monolítica, el cual todos los paquetes junto con sus dependencias son compilados y da como resultado un solo artefacto, el cual incluye todo el código junto con las dependencias. En este ejemplo se observa cómo se ha creado un EXE, pero se pudo haber creado un Jar en el caso de Java o un JS en el caso de JavaScript, dependiendo la tecnología utilizada tendremos un artefacto diferente, pero al final, todos contendrán todo el código con sus dependencias.

Estructura de una arquitectura monolítica

Como explica (Oscar Javier Blancarte, 2020), la estructura Monolítica puede tener una serie de paquetes bien organizados y un código muy claro, donde cada paquete puede tener cierta parte de la funcionalidad y están desacoplados uno de otro (recordemos que eso es independiente del estilo arquitectónico). Sin embargo, al momento de compilarse el código, todo se empaqueta como un solo software. Cabe mencionar que cualquier librería que sea requerida, será exportada como parte del archivo compilado de salida.

En la opinión de (Blancarte, 2020), menciona que algo a tomar en cuenta es que cuando los módulos son compilados por separado y son instalados como complementos ya no se trata de un Monolítico y pasar a ser un estilo arquitectónico de Microkernel, el cual se expone más adelante.

Ilustración 12: Proceso de compilación



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

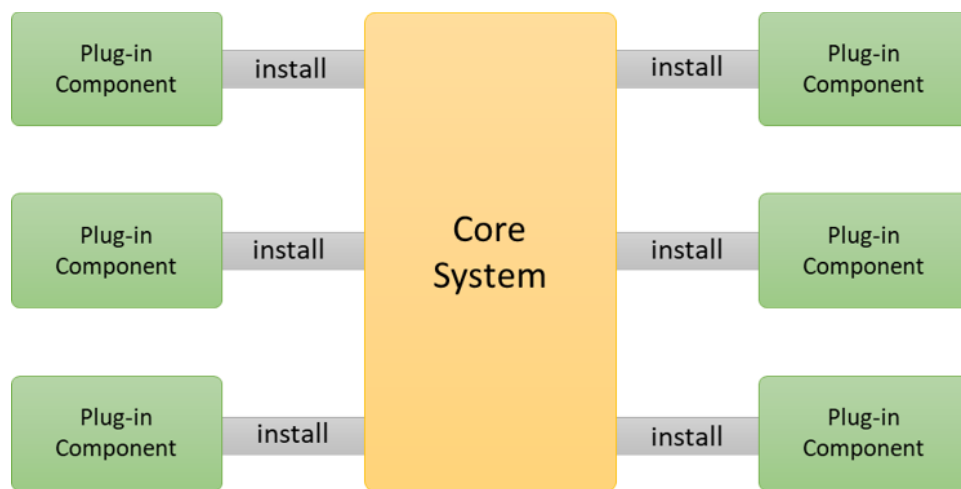
(Blancarte, 2020), expone, cómo funciona el proceso de compilación de una aplicación Monolítica, el cual todos los paquetes junto con sus dependencias son compilados y da como resultado un solo artefacto, el cual incluye todo el código junto con las dependencias. En este ejemplo se puede notar que se crea un EXE, pero se pudo haber creado un Jar en el caso de Java o un JS en el caso de JavaScript, dependiendo la tecnología utilizada se tendrá un artefacto diferente, pero al final, todos contendrán todo el código con sus dependencias.

Arquitectura de Micokernel

El estilo arquitectónico de Microkernel o también conocido como arquitectura de Plug-in, permite crear aplicaciones extensibles, mediante la cual es posible agregar nueva funcionalidad mediante la adición de pequeños plugin que extienden la funcionalidad inicial del sistema. (Oscar Javier Blancarte, 2020).

Continuando con lo mencionado por (Oscar Javier Blancarte, 2020), en esta arquitectura las aplicaciones se dividen en dos tipos de componentes, en sistema Core (o sistema central) y los plugins (o módulos), el sistema Core contiene los elementos mínimos para hacer que la aplicación funcione y cumpla el propósito para el cual fue diseñada, por otra parte, los módulos o plugins con componentes periféricos que se añaden o instalan al componente Core para extender su funcionalidad. En este sentido, solo puede haber un componente Core y muchos Plugins.

Ilustración 13: Arquitectura Microkernel



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

La idea central de este estilo arquitectónico es permitir la extensión de su funcionalidad o personalización, pero respetando el principio Open-Closed, es decir, está abierto para extender la funcionalidad, pero cerrado para modificar su funcionalidad principal. De esta forma, se logra que los desarrolladores puedan crear plugins para agregar nueva funcionalidad o extender la existente, pero sin alterar la funcionalidad Core del sistema. (Oscar Javier Blancarte, 2020).

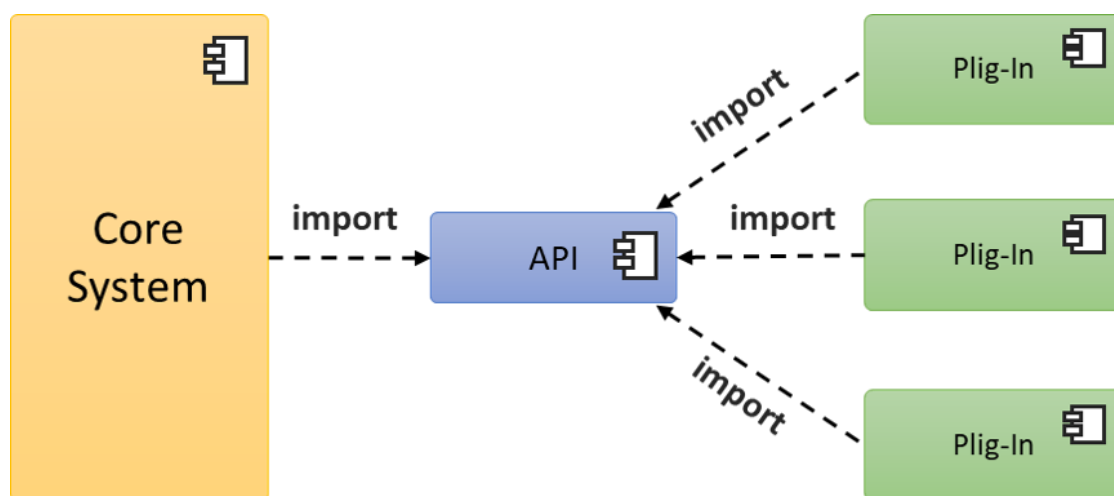
Estructura de una arquitectura microkernel

Los sistemas que utilizan una arquitectura de Microkernel no son fáciles de desarrollar, pues necesitamos crear aplicaciones que son capaces de agrandar dinámicamente su funcionalidad a medida que nuevos plugins son instalados, al mismo tiempo que debemos de tener mucho cuidado de que los plugins no modifiquen o alteren la esencia de la aplicación. (Oscar Javier Blancarte, 2020).

Una arquitectura monolítica es fácil de construir porque todas las piezas están juntas. Sin embargo, debe probarse como una sola unidad, lo que hace que la depuración y las actualizaciones sean potencialmente dolorosas. Los microservicios son lo contrario. Son fáciles de depurar, aislar e implementar, pero hay muchas partes móviles que administrar. El seguimiento de las interacciones entre los componentes puede resultar complicado. Tomado de sitio web (kryptonsolid)

Por su parte, (Blancarte, 2020), afirma que para permitir la construcción de plugins, el sistema Core debe de proporcionar un API o una definición la cual el plugin debe de implementar, de esta forma tenemos el sistema Core y el API que provee para que los desarrolladores creen los plugins. El API es una serie de clases o interfaces que deben ser implementadas por el Plug-in, las cuales serán analizadas por el sistema Core al momento de la instalación.

Ilustración 14: API del microkernel



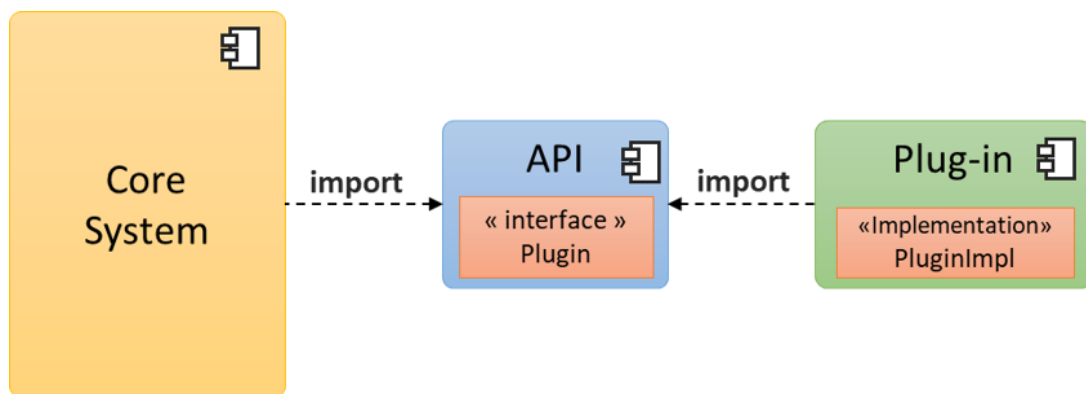
Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

En la imagen anterior que corresponde a la ilustración 10 se puede observar el rol que tiene el API en la arquitectura. El API en realidad es un componente independiente del sistema Core, el cual contiene solamente las clases e interfaces que deberán conocer tanto el sistema Core como los Plugins. (Blancarte, 2020).

Por lo general hay una interfaz o clase abstracta llamada Plugin la cual debe de implementar una clase del componente Plugin, la cual servirá como punto de entrada para el sistema Core, la cual le dará toda la información del Plugin al sistema Core. (Blancarte, 2020),

Ilustración 15: Implementando un plug-in



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

Arquitectura orientada a servicios (SOA)

La Arquitectura de desarrollo de software Orientada a Servicios viene de una evolución de orientación a objetos, luego a componentes para convertirse en un nuevo paradigma que propone convertir los recursos de software en servicios disponibles que puedan ser reutilizables y fáciles de integrar en grandes y complejas aplicaciones, facilitando la interoperabilidad entre las aplicaciones y el web service.

Así se integran diferentes ciclos de vida e implementaciones de soluciones con múltiples tecnologías.

Tomado de sitio web (revistas.utp.edu.co)

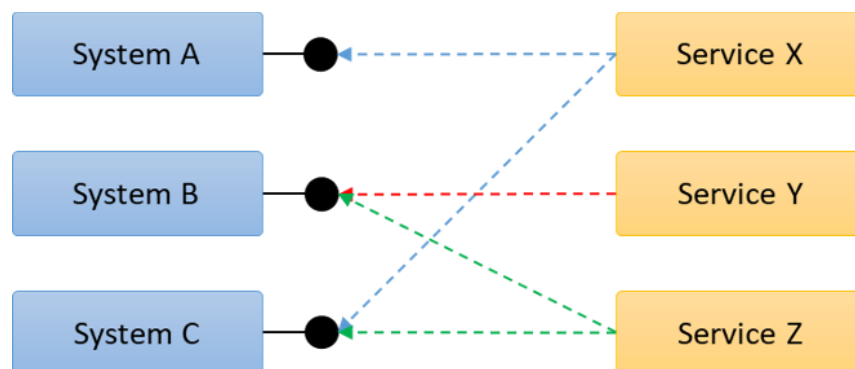
SOA solo promueve la creación de servicios para lograr la integración de aplicaciones, por lo que se podría decir que CORBA es el precursor de SOA, aunque al momento de crear CORBA no existía el término de SOA. Por su parte, el nacimiento de SOA como arquitectura no solo trae el concepto de crear servicios sino un nuevo paradigma que revoluciono la forma en que construimos software de hoy en día. (Blancarte, 2020),

En otras palabras, la SOA integra los elementos del software que se implementan y se mantienen por separado, y permite que se comuniquen entre sí y trabajen en conjunto para formar aplicaciones de software en distintos sistemas.

Estructura de una arquitectura orientada a servicios (SOA)

El autor (Blancarte, 2020), menciona que lo primero que tenemos que entender al momento de implementar una arquitectura SOA es que, todas las aplicaciones que construimos están diseñadas para ser integradas con otras aplicaciones, por lo que deben de exponer como servicios todas las operaciones necesarias para que otras aplicaciones puedan interactuar con ella.

Ilustración 16: Consumo de servicios



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

En la imagen anterior podemos ver como las aplicaciones del lado izquierdo (A, B, C) exponen una serie de servicios que estarán disponibles por la red, los cuales son consumidos por las aplicaciones del lado derecho (X, Y, Z), aun que podrían ser consumidos por cualquier otra aplicación que este dentro de la misma red, o en su defecto, por Internet. (Blancarte, 2020).

La idea central de construir servicios va más allá de solventar la problemática de la integración con aplicaciones heterogéneas, si no que brinda una serie de ventajas que ayudan a la reutilización de los servicios, encapsulamiento de las aplicaciones, seguridad, una larga lista de características. (Blancarte, 2020).

Arquitectura de microservicios

Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Cada uno de servicio es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado. Un contexto delimitado es una división natural de una empresa y proporciona un límite explícito dentro del cual existe un modelo de dominio. Tomado de sitio web (learn.microsoft.com)

Un Microservicios es un pequeño programa que se especializa en realizar una pequeña tarea y se enfoca únicamente en eso, por ello, decimos que los Microservicios son Altamente Cohesivos, pues toda las operaciones o funcionalidad que tiene dentro está extremadamente relacionadas para resolver un único problema. (Blancarte, 2020).

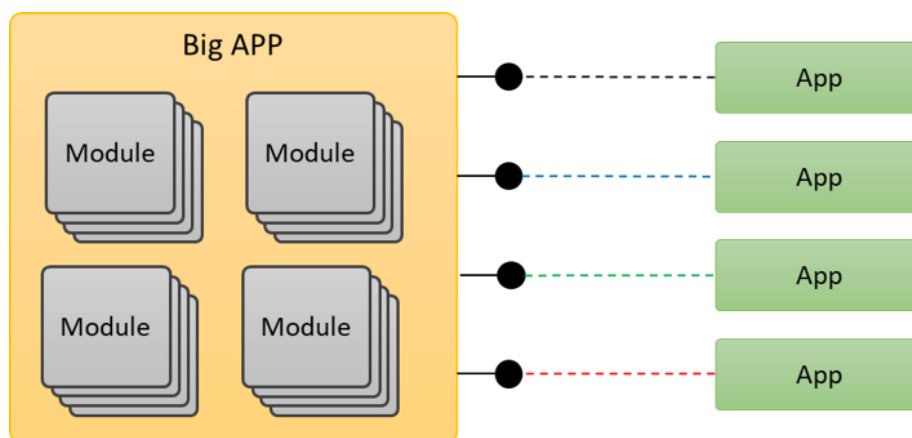
En este sentido (Blancarte, 2020), dice que los Microservicios son todo lo contrario a las aplicaciones Monolíticas, pues en una arquitectura de Microservicios se busca desmenuzar una gran aplicación en

muchos y pequeños componentes que realizar de forma independiente una pequeña tarea de la problemática general.

Estructura de una arquitectura microservicios

Comprender los microservicios es necesario regresar a la arquitectura Monolítica, donde una sola aplicación tiene toda la funcionalidad para realizar una tarea de punta a punta, además, una arquitectura Monolítica puede exponer servicios, tal como lo vemos en la siguiente imagen: (Blancarte, 2020).

Ilustración 17: Arquitectura Monolítica



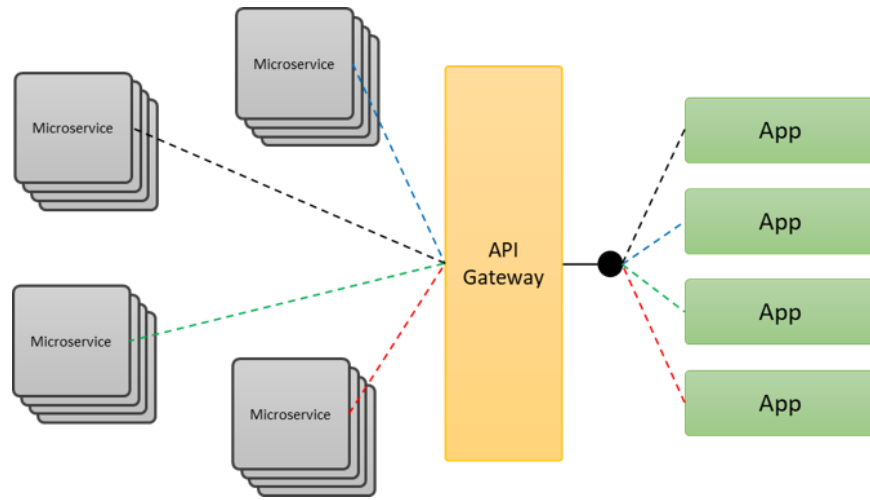
Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

Una aplicación Monolítica tiene todos los módulos y funcionalidad necesario dentro de ella, lo que lo hace una aplicación muy grande y pesada, además, en una arquitectura Monolítica, es Todo o Nada, es decir, si la aplicación está arriba, tenemos toda la funcionalidad disponible, pero si está abajo, toda la funcionalidad está inoperable. (Blancarte, 2020).

La arquitectura de Microservicios busca exactamente lo contrario, dividiendo toda la funcionalidad en pequeños componentes autosuficientes e independientes del resto de componentes, tal y como lo puedes ver en la imagen anterior. (Blancarte, 2020).

Ilustración 18: Arquitectura de Microservicios

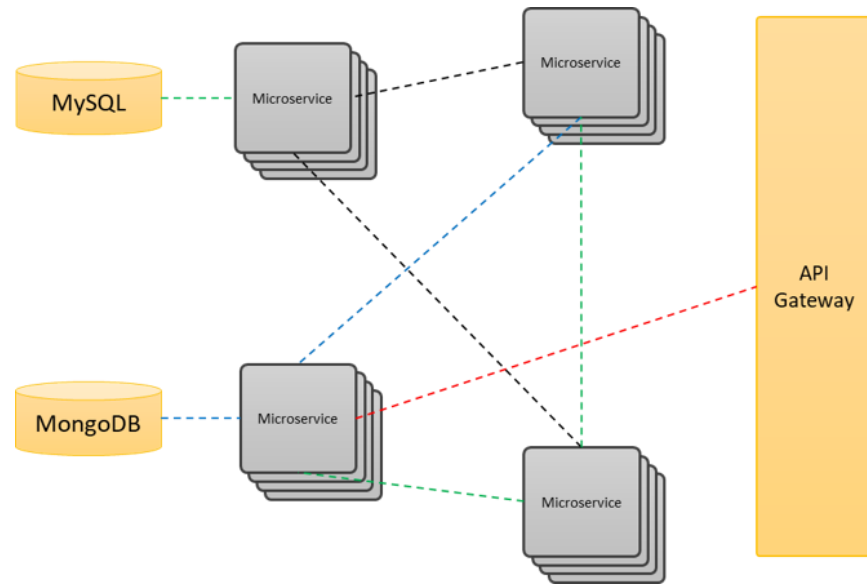


Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

Para que el funcionamiento de la arquitectura de microservicios sea similar al de una aplicación funcional de la nube, los servicios deben solicitar datos a los demás servicios constantemente a través de la mensajería. Al desarrollar una malla de servicios en una aplicación, se simplifica la comunicación entre ellos. Sin embargo, es posible que también deba integrar la arquitectura de microservicios con sus aplicaciones heredadas y demás fuentes de datos. Tomado de sitio web (redhat.com)

Ilustración 19: Comunicación entre Microservicios



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

Algo a tomar en cuenta es que los Microservicios trabajan en una arquitectura distribuida, lo que significa todos los Microservicios son desplegados de forma independiente y están desacoplados entre sí. Además, deben ser accedidos por medio de protocolos de acceso remoto, como colas de mensajes, SOAP, REST, RPC, etc. Esto permite que cada Microservicio funcione o pueda ser desplegado independientemente de si los demás están funcionando. (Blancarte, 2020).

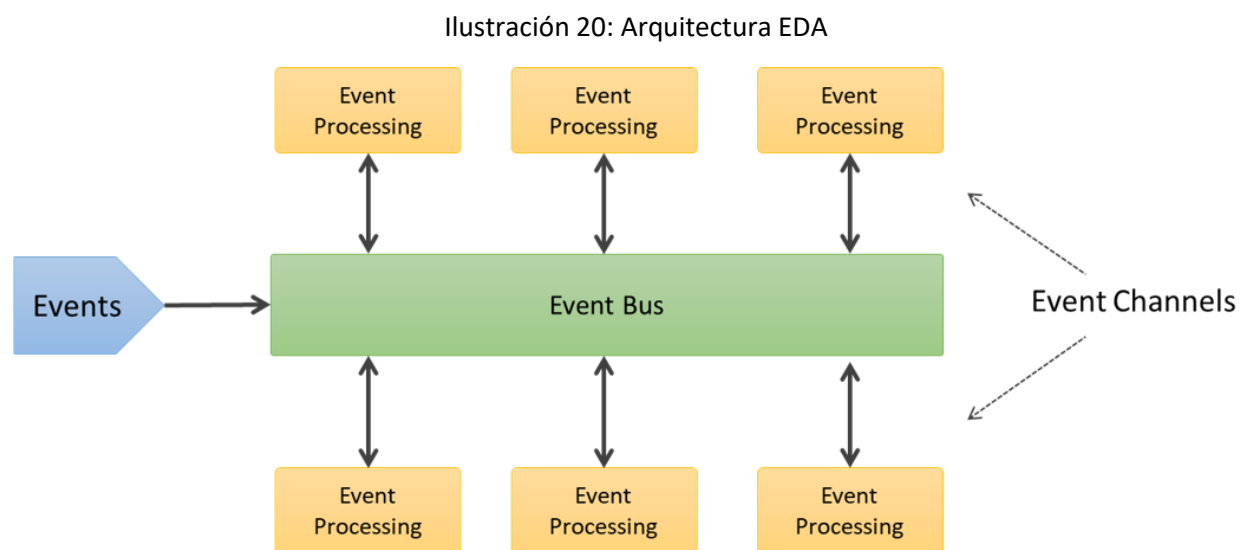
Arquitectura impulsada por eventos (EDA)

La arquitectura basada en eventos es un modelo y una arquitectura de software que sirve para diseñar aplicaciones. En un sistema como este, la captura, la comunicación, el procesamiento y la permanencia de los eventos son la estructura central de la solución. Esto difiere del modelo tradicional basado en solicitudes. Tomado de sitio web (redhat.com)

Esta arquitectura está compuesta por productores y consumidores de eventos. El primero detecta los eventos y los representa como mensajes. No conoce al consumidor del evento ni el resultado que generará este último. Tomado de sitio web (redhat.com)

Una vez que se detecta un evento, este se transmite del productor a los consumidores a través de canales de eventos, donde se procesa de manera asíncrona con una plataforma para este fin. Cuando se produce un evento, se debe informar a los consumidores, quienes podrían procesarlo o simplemente recibirlo. Tomado de sitio web (redhat.com)

(Blancarte, 2020), escribe que, en una arquitectura EDA los componentes no se comunican de forma tradicional, en la cual se establece comunicación de forma síncrona, se obtiene una respuesta y se procede con el siguiente paso. En esta arquitectura, se espera que las aplicaciones lancen diversos “eventos” para que otros componentes puedan reaccionar a ellos, procesarlos y posiblemente generar nuevos eventos para que otros componentes continúen con el trabajo.



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

Como podemos ver en la imagen anterior, todo inicia con la llegada de un nuevo evento a algo que llamaremos por ahora Event Bus, el cual es un componente que se encarga de administrar todos los eventos de la arquitectura. El Event Bus se encargará de recibir el evento y colocarlo en los Event Channels, las cuales son básicamente una serie de Colas (Queues) o Temas (Topics) sobre los cuales habrá ciertos componentes a la escucha de nuevos mensajes. (Blancarte, 2020).

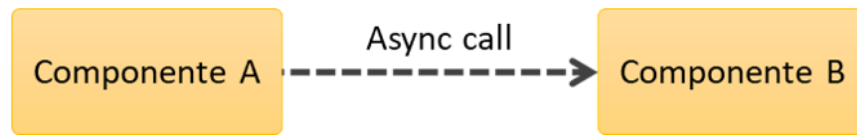
Por otra parte, los Event Processing son componentes de negocio que está a la escucha de nuevos eventos depositados en los Event Channels. Los Event Processing son los encargados de procesar los eventos, es decir, realizar acciones concretas sobre el sistema, como crear, actualizar o borrar algún registro, finalmente, el proceso puede terminar de forma silenciosa, o puede generar un nuevo evento para que otro componente lo tome y continúe con el procesamiento. (Blancarte, 2020).

Para complementar se tiene en cuenta algo importante y es que en todos lo Event Processing estos se comportan como componentes distribuidos, dando así que estos estén absolutamente desacoplados del resto y así el funcionamiento de uno solo no va a afectar a alterar los demás componentes.

En este sentido (Blancarte, 2020), menciona que en esta arquitectura cada Event Processing solo tiene conocimiento de la tarea que el realiza y no tiene conocimiento de la existencia de otros Event Processing, lo cual permite que trabajen de forma autónoma.

EDA es una arquitectura asíncrona, por ende, no se puede permitir ejecutar una operación y esperar una respuesta, por lo que se debe dejar la solicitud y esperar hasta que alguien la atienda, durante este proceso, se puede llamar directamente a la operación asíncrona del otro componente, sin embargo, esto puede ser una mala idea. (Blancarte, 2020).

Ilustración 21: Llamada directa a una operación asíncrona.



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

En la imagen anterior podemos ver claramente un fuerte acoplamiento entre los dos componentes, lo cual va en contra de la arquitectura EDA, la cual busca un bajo acoplamiento entre los componentes, además, de que la operatividad de un componente no debe depender de la operatividad del otro. En este escenario, el componente A depende de que B esté funcionando al momento de realizar la llamada, ya que, si no, podrían suceder dos cosas, que el proceso completo falle porque no pudo realizar la llamada a B o la segunda, que el proceso continúe ante la falla, pero perdamos el mensaje enviado a B. Sea cual sea el caso, no es el esperado, porque al final tenemos un fallo total o parcial, lo que nos deja en un estado inconsistente. (Blancarte, 2020).

Para (Blancarte, 2020), determinar la solución a este tipo de problemas, es hacer uso de un intermediario que tomará las solicitudes, las almacenará y las entregará al consumidor adecuando cuando este esté disponible, de esta forma, el productor del mensaje podrá dejar el mensaje en el Intermediario y continuar con el proceso, pues el Mediador garantizará que el mensaje será entregado al consumidor. Por otra parte, el consumidor (Componente B) estará a la escucha de nuevos mensajes en el Intermediario para procesarlos. Si el consumidor está fuera de servicio cuando el productor (Componente A) deja el mensaje, este no se perderá, pues el Intermediario lo guardará y lo entregará cuando el Consumidor esté nuevamente operativo.

Transferencia de estado representacional (REST)

“REST proporciona un conjunto de restricciones arquitectónicas que, cuando se aplican como un todo, enfatizan la escalabilidad de las interacciones de los componentes, la generalidad de las interfaces, la implementación independiente de los componentes y los componentes intermedios para reducir la latencia de interacción, reforzar la seguridad y encapsular los sistemas heredados.”

— Roy Fielding

Algo muy importante a tener en cuenta, y es que REST ignora los detalles de implementación del componente y la sintaxis del protocolo para enfocarse en los roles de los componentes, las restricciones sobre su interacción con otros componentes y la representación de los datos. Por otro lado, abarca las restricciones fundamentales sobre los componentes, conectores y datos que definen la base de la arquitectura web, por lo tanto, podemos decir que la esencia de REST es comportarse como una aplicación basada en la red. (Blancarte, 2020).

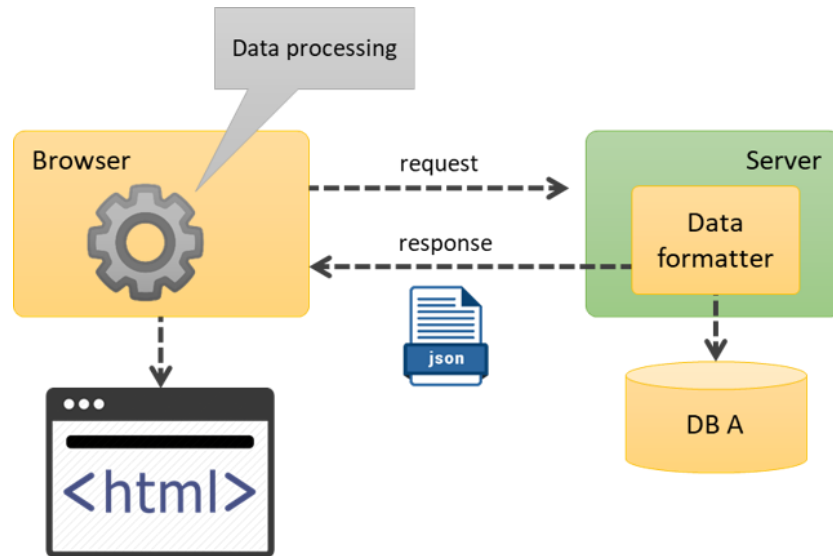
(Blancarte, 2020), representa la arquitectura en REST por 3 conceptos clave los cuales son: Datos, conectores y componentes

1. **Datos:**

En cualquier arquitectura distribuida, son los componentes de negocio quien procesa la información y nos responde la información que se produjo de tal procesamiento, por ejemplo, una aplicación web tradicional, donde la vista es construida en el backend y nos responde con la página web ya creada, con los elementos HTML que la componente y los datos incrustados. En este sentido, la aplicación web proceso los datos en el backend, y como resultado nos arroja el resultado de dicho procesamiento, sin embargo, REST lo que propone es que los datos sean enviados en bruto para que sea el interesado de los datos el que los procese para generar la página (o cualquier otra cosa que tenga que generar), en este sentido, lo que REST propone no es generar la página web, si no que mandar los datos en bruto

para que el consumidor sea el responsable de generar la página a través de los datos que responde REST. (Blancarte, 2020).

Ilustración 22: Procesamiento del lado del cliente



Fuente: Adoptado de (Introducción a la arquitectura de software un enfoque práctico)

Elaborado por: Oscar Javier Blancarte, 2020

2. Conectores

Los conectores como los componentes que encapsulan la actividad de acceso a los recursos y transferencia, de esta forma, REST describe los conectores como interfaces abstractas que permite la comunicación entre compontes, mejorando la simplicidad al proporcionar una separación clara de las preocupaciones y ocultando la implementación subyacente de los recursos y los mecanismos de comunicación. (Blancarte, 2020).

REST describe que todas las solicitudes a los conectores sean sin estado, es decir, que deberán contener toda la información necesaria para que el conector comprenda la solicitud sin importar las llamadas que se hallan echo en el pasado, de esta forma, ante una misma solicitud deberíamos de obtener la misma respuesta. (Blancarte, 2020).

3. Componentes

Finalmente, los componentes con software concretos que utilizan los conectores para consultar los recursos o servirlos, ya sea una aplicación cliente como lo son los navegadores (Chrome, Firefox, etc.) o Web Servers como Apache, IIS, etc. Pero también existen los componentes intermedios, que actúan como cliente y servidor al mismo tiempo, como es el caso de un proxy o túnel de red, los cuales actúan como servidor al aceptar solicitudes, pero también como cliente a redireccionar las peticiones a otro servidor. (Blancarte, 2020).

Los componentes se pueden confundir con los conectores, sin embargo, un conector es una interface abstracta que describe la forma en que se deben de comunicar los componentes, mientras que un componente es una pieza de software concreta que utiliza los conectores para establecer la comunicación. (Blancarte, 2020).

PATRONES DE DISEÑO

Los patrones de diseño son soluciones para problemas típicos y recurrentes que nos podemos encontrar a la hora de desarrollar una aplicación. Podemos agrupar estos patrones de diseño en según su propósito y tenemos que hay 3 tipos de patrones de diseño: patrones creacionales, patrones de comportamiento y patrones estructurales. En el artículo explicamos en detalles que son y cómo influyen en el desarrollo de aplicaciones. Tomado de sitio web (dialnet.unirioja.es)

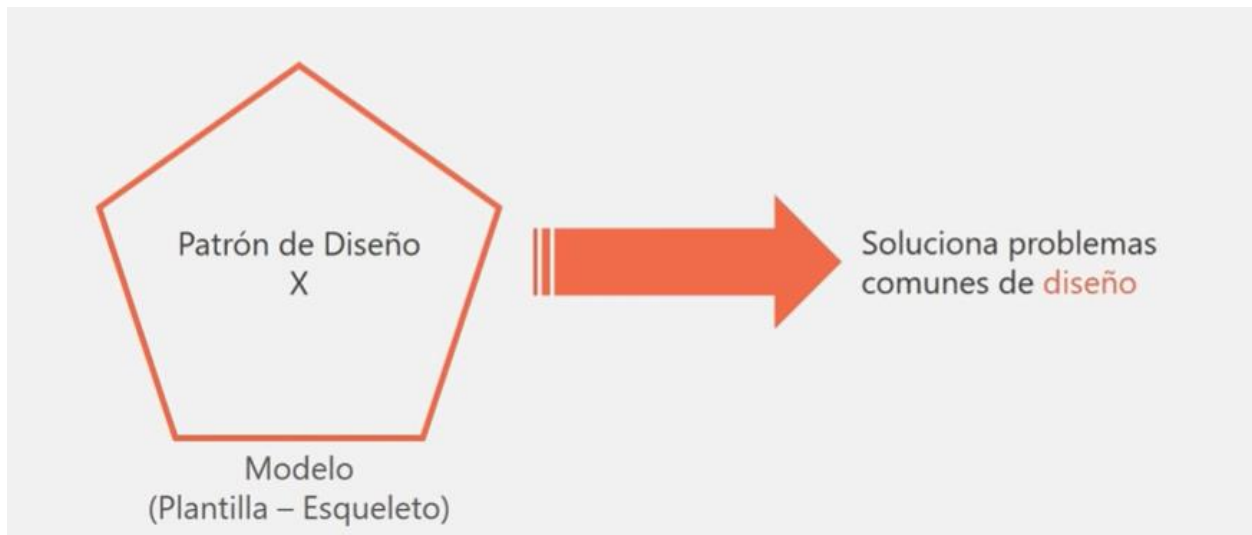
Los patrones de diseño, estos resuelven problemas más localizados, tienen menos impacto en la base del código, afectan una sección específica de la base del código, como, por ejemplo:

4. Cómo instanciar un objeto cuando solo sabemos qué tipo necesita instanciarse en tiempo de ejecución.
5. Cómo hacer que un objeto se comporte de manera diferente según su estado.

Tomado de sitio web (rjcodeadvance.com)

La particularidad de los patrones de diseño es que describen una estructura interna y la relación entre algunas clases o módulos internos para así obtener una solución en la estructura o componente de un sistema de software, es decir un patrón de diseño solo afecta en uno o varios componentes internos de un sistema de software o aplicación. Tomado de sitio web (rjcodeadvance.com)

Ilustración 23: Modelo de Patrón de Diseño



Fuente: Adoptado de (<https://www.youtube.com/watch?v=9NjtWyjjLKQ>)

Elaborado por: RJ Code Advanced, 2018

El patrón de arquitectura representa la relación entre los componentes de todo el sistema y cada uno de ellos está compuesto por pequeños módulos. Altera a la estructura global del sistema, ya que estos plasman un esquema organizativo estructural para sistemas de software, por lo tanto, los patrones de diseño ayudan a comprender esa arquitectura o estructura que se haya utilizado. Tomado de sitio web (rjcodeadvance.com)

Clasificación de patrones de diseño

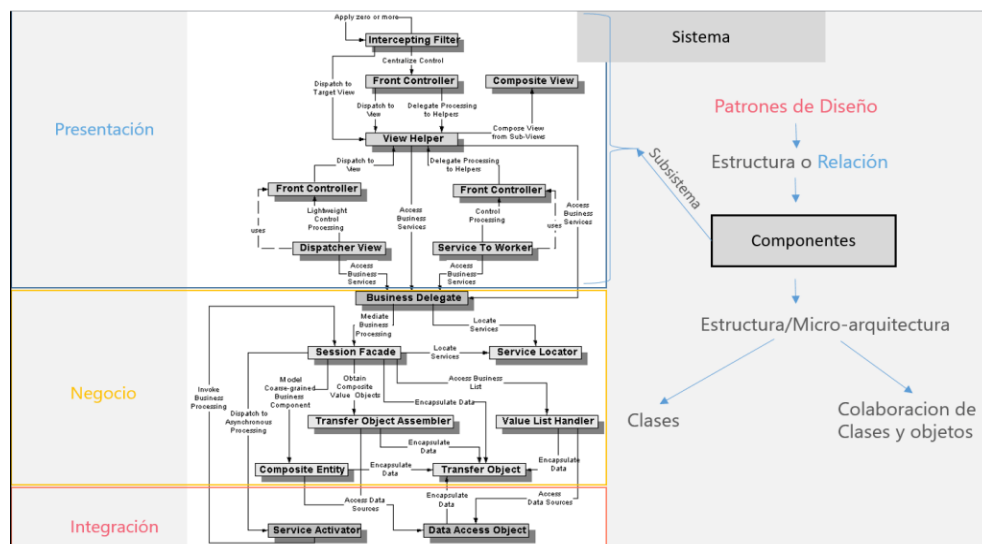
Orientado a Subsistemas de componentes

Basado en el libro (J2EE PATTERNS Best Practices and Design Strategies – Java), indica que existen 5 capas en la arquitectura J2EE las cuales son las siguientes:

1. Cliente
2. Presentación
3. Negocios
4. Integración
5. Recurso

A esto corresponden 15 patrones de diseño los cuales se encuentran divididos en 3 de las capas de presentación, negocios e integración.

Ilustración 24: Patrones de Diseño



Fuente: Adoptado de (<https://www.youtube.com/watch?v=9NjtWyjjLKQ>)

Elaborado por: RJ Code Advanced, 2018

1. **Presentación**

Como primera medida tenemos la presentación y es el encargado de la interfaz gráfica y de la interacción con el usuario los cuales son los siguientes:

1. **Intercepting Filter:**

Se usa cuando se quiere hacer algún preprocesamiento/postprocesamiento con solicitud o respuesta de la aplicación. Los filtros se definen y aplican en la solicitud antes de pasar la solicitud a la aplicación de destino real. Los filtros pueden realizar la autenticación/autorización/registro o seguimiento de la solicitud y luego pasar las solicitudes a los controladores correspondientes. Tomado de sitio web (tutorialspoint.com).

2. **Front Controller:**

Se utiliza para proporcionar un mecanismo de manejo de solicitudes centralizado, de modo que todas las solicitudes sean manejadas por un único controlador. Este controlador puede realizar la autenticación/autorización/registro o seguimiento de la solicitud y luego pasar las solicitudes a los controladores correspondientes. Tomado de sitio web (tutorialspoint.com).

3. **View Helper:**

El patrón View Helper describe las formas en que esto se puede hacer utilizando etiquetas personalizadas y JavaBeans para adaptar el modelo a la presentación dentro de la vista. Tomado de sitio web (link.springer.com).

4. **Composite View:**

El patrón Composite View, es responsable de crear vistas compuestas a partir de más de una subvista. Permite que el contenido y el diseño se conecten al marco. Esto simplifica en gran medida el desarrollo de la aplicación. Tomado de sitio web (dineshonjava.com).

5. Service To Worker:

El patrón Service to Worker es el resultado de combinar el patrón View Helper con un despachador en coordinación con el patrón Front Controller. Tomado de sitio web (corej2eepatterns.com).

6. Dispatcher View:

El patrón Dispatcher View sugiere diferir la recuperación de contenido hasta el momento del procesamiento de la vista. Además, el despachador juega un papel más limitado en la gestión de vistas, ya que la elección de la vista normalmente ya está incluida en la solicitud. Tomado de sitio web (corej2eepatterns.com).

2. Negocio

Estos patrones se encargan de las reglas de negocio, validaciones de negocio y transferencia de datos, corresponden a los siguientes

1. Business Delegate:

Se utiliza para desacoplar el nivel de presentación y el nivel empresarial. Básicamente se utiliza para reducir la comunicación o la funcionalidad de búsqueda remota al código de

nivel empresarial en el código de nivel de presentación. Tomado de sitio web (tutorialspoint.com).

2. Session Facade:

Abstrae las interacciones de los objetos de negocio y proporciona una capa de servicio que expone sólo los interfaces requeridos. Así, oculta a la vista de los clientes las interacciones complejas entre los participantes. Tomado de sitio web (juntadeandalucia.es).

3. Service Locator:

El objetivo de este patrón es mejorar el modularidad de su aplicación eliminando la dependencia entre el cliente y la implementación de una interfaz. Tomado de sitio web (stackify.com).

4. Value Object Assembler:

Es muy útil en un modelo Cliente-Servidor, donde las llamadas remotas del lado del cliente al Servidor deben minimizarse para reducir la sobrecarga de la red. Tomado de sitio web (ciol.com).

5. Value List Handler:

Proporciona una función de almacenamiento en caché en el bean de sesión del controlador de lista de valores para mantener el conjunto de resultados obtenido de la ejecución de una consulta. El conjunto de resultados es una colección de objetos de transferencia que se pueden serializar si es necesario. Tomado de sitio web (corej2eepatterns.com).

6. Composit Entity:

Utiliza varios componentes para la implementación. Cada uno de los componentes tiene ciertas tareas que realizar y ciertos problemas que atender. Tomado de sitio web (dineshonjava.com).

7. Data Transfer Object (DTO):

Tiene como finalidad de crear un objeto plano (POJO) con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarlas en una única clase simple. Tomado de sitio web (oscarblancarteblog.com).

1. Integración

Estos patrones se encargan de explotar los datos de cualquier fuente ya sea archivos o bases de datos los cuales son los siguientes:

1. Service Activator:

Se implementa como un servicio de delegación y escucha JMS que puede escuchar y recibir mensajes JMS. En una aplicación usa componentes EJB y su contenedor EJB implementa EJB versión 2.0 o posterior, puede usar un bean controlado por mensajes (MDB) para recibir solicitudes asíncronas. Tomado de sitio web (javaskool.com)

2. Data Access Object (DAO):

Pretende principalmente independizar la aplicación de la forma de acceder a la base de datos, o cualquier otro tipo de repositorio de datos. Para ello se centraliza el código relativo al acceso al repositorio de datos en las clases llamadas DAO. Fuera de las clases DAO no debe haber ningún tipo de código que acceda al repositorio de datos. Tomado de sitio web (cursohibernate.es).

Orientado a clases y objetos

Según lo contenido en el libro (Design Pattern – GOF), se presentan veintitrés patrones de diseño los cuales se dividen en tres categorías:

3. **Creacionales**

Estos se encargan de la inicialización y configuración de objetos, en esta se encuentra cinco patrones de diseño los cuales son:

1. **Abstract Factory:**

Defina una interfaz o clase abstracta para crear familias de objetos relacionados (o dependientes) pero sin especificar sus subclases concretas. Eso significa que Abstract Factory permite que una clase devuelva una fábrica de clases. Entonces, esta es la razón por la que Abstract Factory Pattern es un nivel más alto que Factory Pattern. Tomado de sitio web (jvatpoint.com).

2. **Builder:**

Tiene como objetivo "Separar la construcción de un objeto complejo de su representación para que el mismo proceso de construcción pueda crear diferentes representaciones". Se utiliza para construir un objeto complejo paso a paso y el paso final devolverá el objeto. El proceso de construcción de un objeto debe ser genérico

para que pueda usarse para crear diferentes representaciones del mismo objeto.
Tomado de sitio web (geeksforgeeks.org).

3. Factory Method:

Define una interfaz o clase abstracta para crear un objeto, pero deje que las subclases decidan qué clase instanciar. En otras palabras, las subclases son responsables de crear la instancia de la clase. Tomado de sitio web (javatpoint.com).

4. Prototype:

Es una parte de los patrones de diseño creacional. Su objetivo principal es reducir el número de clases para una aplicación. Brinda la posibilidad de crear una copia del objeto existente independientemente de la implementación real de sus clases, en palabras simples, el objeto recién creado requiere más recursos de los que queremos usar o tener disponibles, por lo que creamos la copia exacta del objeto disponible. Tomado de sitio web (javatpoint.com).

5. Singleton:

"Define una clase que tiene solo una instancia y proporciona un punto de acceso global a ella".

En otras palabras, una clase debe garantizar que solo se cree una única instancia y que todas las demás clases puedan utilizar un único objeto. Tomado de sitio web (javatpoint.com).

6. Estructurales

Estos separan la interfaz de la implementación se ocupan de como las clases y objetos se agrupan para formar estructuras más grandes, aquí se encuentran seis patrones de diseño que son los siguientes:

1. Adapter (de Objetos):

“Convierte la interfaz de una clase en otra interfaz que un cliente quiere”.

En otras palabras, proporcionar la interfaz de acuerdo con los requisitos del cliente mientras se utilizan los servicios de una clase con una interfaz diferente.

El patrón de adaptador también se conoce como envoltura. Tomado de sitio web (javatpoint.com).

2. Bridge:

Simplemente separa la abstracción funcional de la implementación para que los dos puedan variar de forma independiente. El patrón de puente también se conoce como manija o cuerpo. Tomado de sitio web (javatpoint.com).

3. Composite:

Permite que los clientes operen de manera genérica en objetos que pueden o no representar una jerarquía de objetos. Tomado de sitio web (javatpoint.com).

4. Facade:

Proporciona una interfaz unificada y simplificada para un conjunto de interfaces en un subsistema, por lo tanto, oculta las complejidades del subsistema del cliente. En otras palabras, Facade Pattern describe una interfaz de nivel superior que hace que el subsistema sea más fácil de usar. Tomado de sitio web (javatpoint.com).

5. Flyweight:

Un patrón Flyweight dice que solo "para reutilizar tipos de objetos similares ya existentes almacenándolos y creando nuevos objetos cuando no se encuentra ningún objeto coincidente" Tomado de sitio web (javatpoint.com).

6. Proxy:

Un patrón de proxy "proporciona el control para acceder al objeto original".

Por lo tanto, podemos realizar muchas operaciones, como ocultar la información del objeto original, cargar a pedido, etc. El patrón de proxy también se conoce como sustituto o marcador de posición. Tomado de sitio web (javatpoint.com).

7. Comportamiento

Se encarga en describir objetos o clases y la comunicación que hay entre ellos aquí tenemos los siguientes:

1. Interpreter:

Un patrón de intérprete dice que "definir una representación de la gramática de un idioma dado, junto con un intérprete que usa esta representación para interpretar oraciones en el idioma".

Básicamente, el patrón Intérprete tiene un área limitada donde se puede aplicar. Podemos discutir el patrón Intérprete solo en términos de gramáticas formales, pero en esta área hay mejores soluciones, por eso no se usa con frecuencia. Tomado de sitio web (javatpoint.com).

2. Template Method:

Un Template Method afirma que "Simplemente define el esqueleto de una función en una operación, aplazando algunos pasos a sus subclases". Tomado de sitio web (javatpoint.com).

3. Chain of Responsibility

Chain of Responsibility afirma que simplemente "evita vincular al remitente de una solicitud con su receptor dando a múltiples objetos la oportunidad de manejar la solicitud". Tomado de sitio web (javatpoint.com).

4. Command:

Encapsula una solicitud bajo un objeto como un comando y pásela al objeto invocador. El objeto invocador busca el objeto apropiado que puede manejar este comando y pasa el comando al objeto correspondiente y ese objeto ejecuta el comando. Tomado de sitio web (javatpoint.com).

5. Iterator:

Iterator Pattern se usa "para acceder a los elementos de un objeto agregado secuencialmente sin exponer su implementación subyacente". Tomado de sitio web (javatpoint.com).

6. Mediator:

Se usa para reducir la complejidad de la comunicación entre varios objetos o clases. Este patrón proporciona una clase de mediador que normalmente maneja todas las comunicaciones entre diferentes clases y admite la facilidad de mantenimiento del código mediante un acoplamiento flexible. Tomado de sitio web (javatpoint.com).

7. Memento:

Un Patrón Memento afirma que "restaurar el estado de un objeto a su estado anterior".

Pero debe hacer esto sin violar la Encapsulación. Tomado de sitio web (jvatpoint.com).

8. Observer:

Se define como "una dependencia uno a uno para que cuando un objeto cambie de estado, todos sus dependientes sean notificados y actualizados automáticamente".

Tomado de sitio web (jvatpoint.com).

9. State:

Se define como "el comportamiento de la clase cambia en función de su estado". En State Pattern, se crea objetos que representan varios estados y un objeto de contexto cuyo comportamiento varía a medida que cambia su objeto de estado. Tomado de sitio web (jvatpoint.com).

10. Strategy:

"Define una familia de funcionalidades, encapsula cada una y las hace intercambiables".

El patrón de estrategia también se conoce como política. Tomado de sitio web (jvatpoint.com).

11. Visitor:

Maneja algoritmos, relaciones y responsabilidades entre objetos. Se utiliza para realizar una operación en un grupo de objetos similares en tiempo de ejecución. Tomado de sitio web (jvatpoint.com).

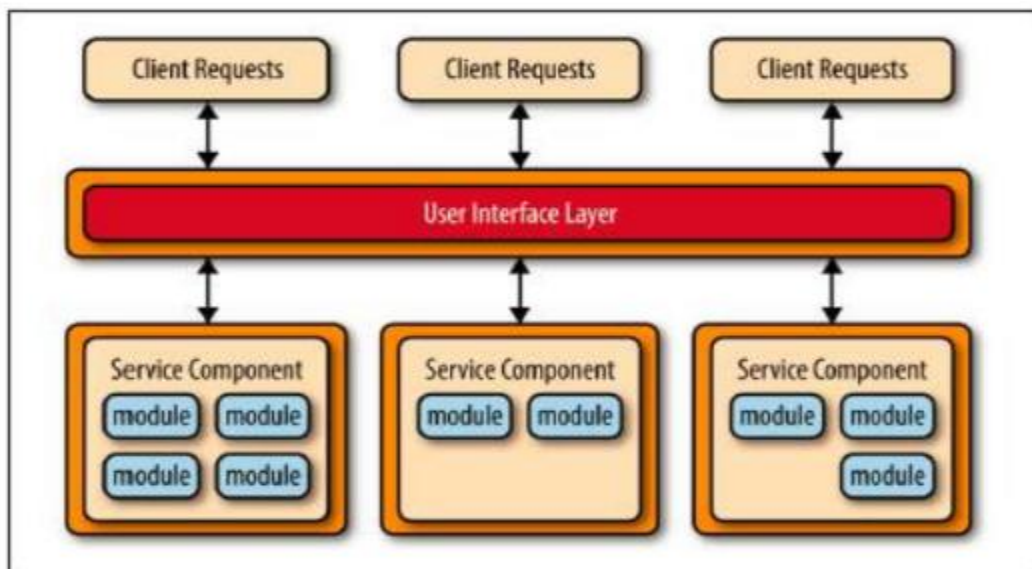
ANÁLISIS COMPARATIVO DE TRES TIPOS DE ARQUITECTURA

Arquitectura de microservicios

Descripción

Como ya se había mencionado en páginas anteriores la arquitectura de microservicios es un modelo tradicional de un programa de software que se compila como una unidad unificada y que es autónoma e independiente de otras aplicaciones. Esta se ha presentado como una alternativa viable a las aplicaciones monolíticas y arquitecturas orientada a servicios. Sin embargo, debido a que es relativamente nueva, hay mucha confusión en la industria sobre de qué se trata este tipo de arquitectura y como se implementa.

Ilustración 25: Diagrama Arquitectura de Microservicios



Fuente: Adoptado de (<https://emanuelpeg.blogspot.com/2017/07/>)

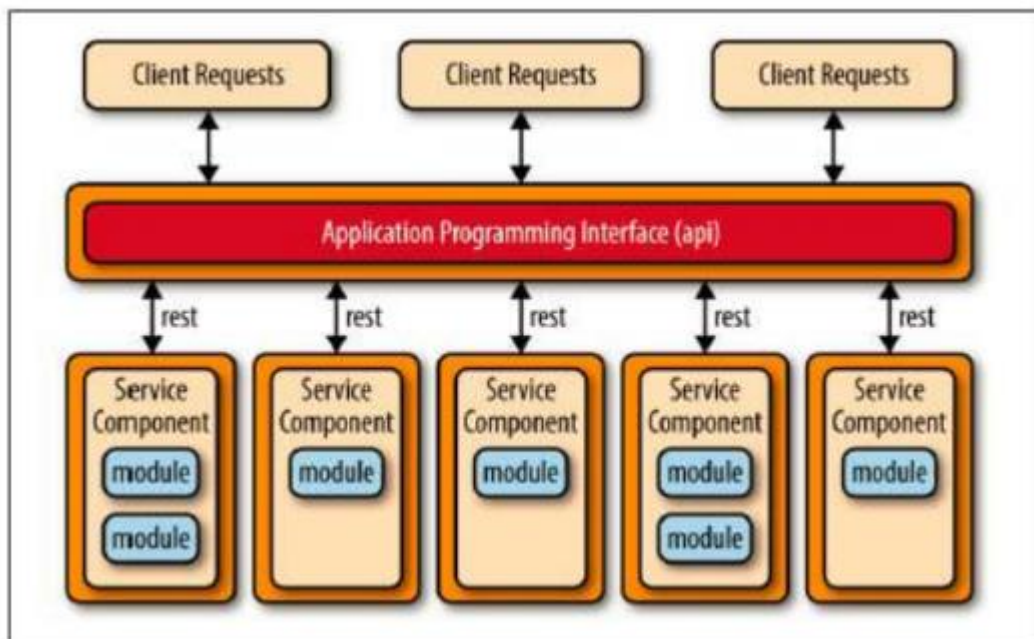
Elaborado por: Autor del proyecto, 2017

El estilo de arquitectura de microservicios naturalmente evolucionó a partir de dos fuentes principales: aplicaciones monolíticas desarrolladas utilizando el patrón de arquitectura en capas y aplicaciones distribuidas desarrolladas a través del patrón de arquitectura orientada a servicios.

12. Topología basada en API REST

Esta topología es muy útil para sitios que exponen servicios individuales pequeños e independientes a través de algún tipo de API (interfaz de programación de aplicaciones). Entre los ejemplos de esta topología se encuentran algunos servicios web RESTful basados en la nube y de propósito único comunes como los es Yahoo!, Google y Amazon. Tomado de sitio web (emanuelpeg.blogspot.com)

Ilustración 26: Diagrama Topología API REST



Fuente: Adoptado de (<https://emanuelpeg.blogspot.com/2017/07/>)

Elaborado por: Autor del proyecto, 2017

13. Topología basada en REST

Se diferencia de la anterior topología en que las solicitudes de los clientes se reciben a través de pantallas de aplicaciones comerciales tradicionales basadas en la web o cliente pesado en lugar de a través de una simple capa de API. Los componentes de servicio en esta topología difieren de los de la topología basada en API-REST en que estos componentes de servicio tienden a ser más grandes, con menor granularidad y representan una pequeña porción de la aplicación empresarial general en lugar de una solución con mayor granularidad, con servicios de única acción. Tomado de sitio web (emanuelpeg.blogspot.com)

14. **Topología de mensajería centralizada**

Esta es similar a la topología basada en REST de la aplicación anterior, excepto que en lugar de utilizar REST para el acceso remoto, esta topología usa un intermediario de mensajes centralizado liviano (por ejemplo, ActiveMQ, HornetQ, etc.). El intermediario de mensajes livianos de esta topología no realiza ninguna orquestación, transformación o enrutamiento complejo; más bien, es solo un transporte liviano para acceder a los componentes del servicio remoto. La topología de mensajería centralizada generalmente se encuentra en grandes aplicaciones empresariales o aplicaciones que requieren un control más sofisticado sobre la capa de transporte entre la interfaz de usuario y los componentes del servicio. Tomado de sitio web (emanuelpeg.blogspot.com)

Ventajas y Desventajas

15. **Ventajas**

- Puede soportar más fácilmente la entrega continua
- Resuelve muchos de los problemas comunes que se encuentran tanto en aplicaciones monolíticas como en arquitecturas orientadas a servicios.

- Proporciona la capacidad de realizar implementaciones de producción en tiempo real, lo que reduce significativamente la necesidad de las implementaciones tradicionales de producción de mensuales o semanales.
- Debido a que la funcionalidad está aislada en componentes de servicios separados y distintos, el desarrollo se vuelve más fácil debido al alcance más pequeño y aislado.

16. **Desventajas**

- Produce redundancia al repetir porciones pequeñas entre los componentes de servicio con el fin de hacerlos independientes y separar su despliegue.
- Debido a la naturaleza distribuida de este patrón, es muy difícil mantener una única unidad transaccional de trabajo a través de (y entre) los componentes del servicio.
- Comparte algunos de los mismos problemas complejos que se encuentran en el patrón ADE, incluida la creación de contratos, el mantenimiento y el gobierno, la disponibilidad remota del sistema y la autenticación y autorización de acceso remoto.
- Este patrón no se presta naturalmente a aplicaciones de alto rendimiento debido a la naturaleza distribuida del patrón de arquitectura de microservicios.

Uso del patrón de microservicios

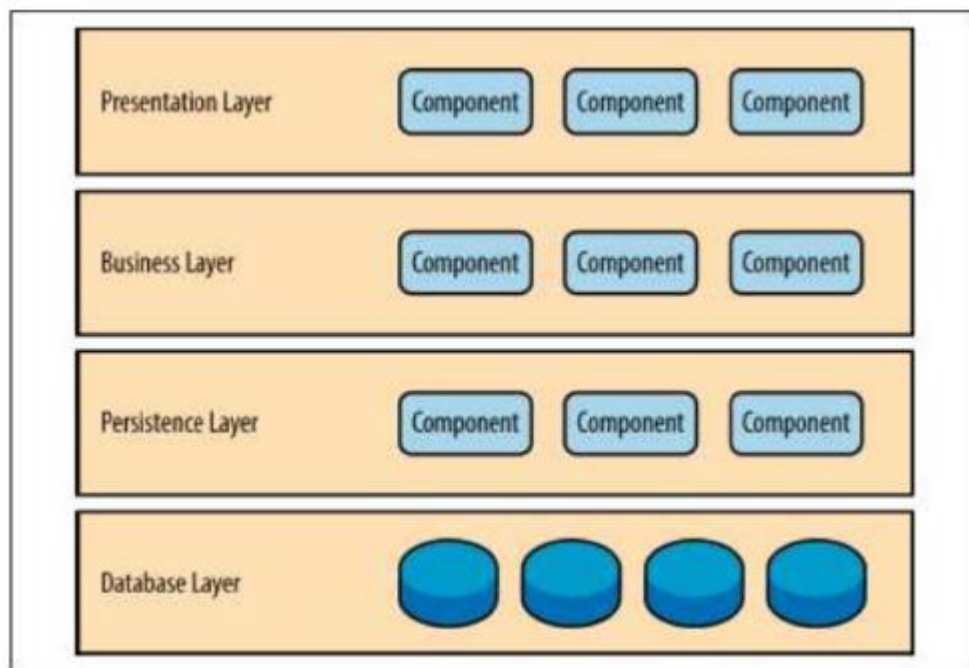
Los microservicios pueden ofrecer mejores ventajas de agilidad y escalabilidad que cualquier otro paradigma, de esta forma se puede determinar en una posibilidad de poder adoptar aplicaciones que tengan requisitos extremos en estas áreas

Arquitectura basada en capas

Descripción

Anteriormente ya se había mencionado esta arquitectura con el fin de dar una mayor descripción los componentes de esta arquitectura están organizados en capas horizontales, cada capa desempeña un papel específico dentro de la aplicación. Generalmente dichas capas son: presentación, lógica de negocio, persistencia y base de datos, aunque este patrón no restringe el número de capas a utilizar. Además, que usualmente la capa de lógica de negocio y persistencia se combinan formando una gran capa. Cada capa en la arquitectura forma una abstracción alrededor el trabajo que debe hacerse para satisfacer una solicitud particular del negocio. De igual forma, una de las poderosas características del patrón de arquitectura en capas es la separación de obligaciones entre los componentes, los componentes dentro de una capa específica solo tratan con la lógica que pertenece a esa capa.

Ilustración 27: Acceso de peticiones con capas cerradas.

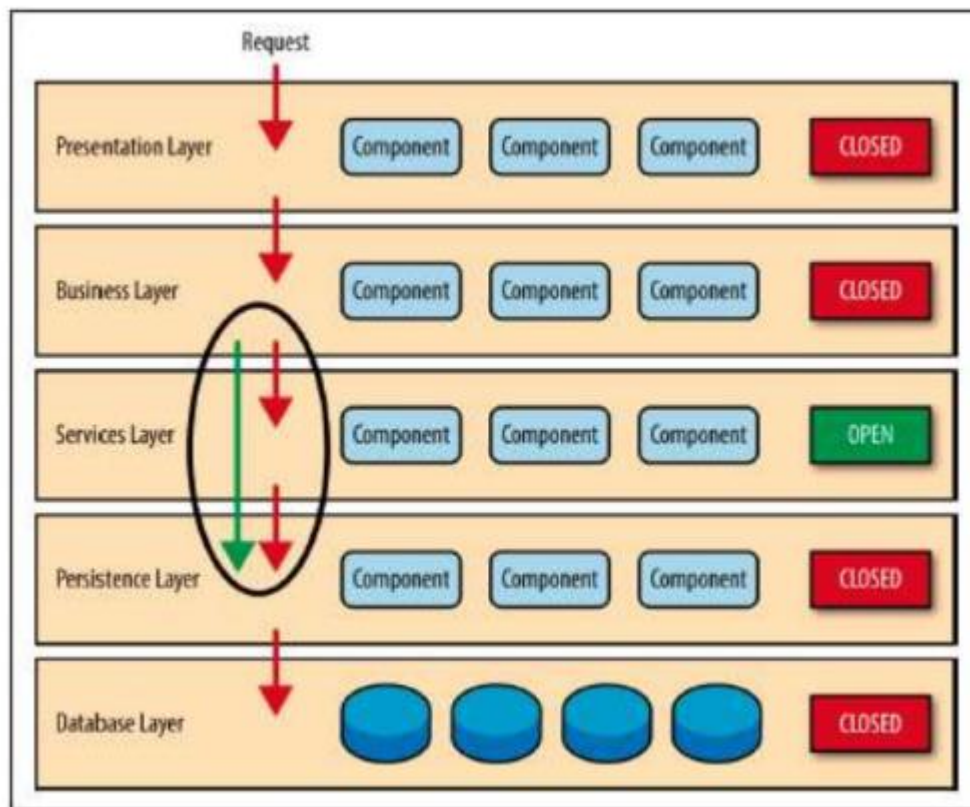


Fuente: Adoptado de (<https://programmerclick.com/article/338572330/>)

Elaborado por: programmerclick.com, 2020

Algo importante de esta arquitectura es el concepto de aislamiento el cual significa que los cambios realizados en una capa de la arquitectura generalmente no afectan o afectan componentes en otras capas, el cambio se aísla a los componentes dentro de esa capa y posiblemente a otra capa asociada (como una capa de persistencia que contiene SQL)

Ilustración 28: Acceso de peticiones con capas abiertas



Fuente: Adoptado de (<https://programmerclick.com/article/338572330/>)

Elaborado por: programmerclick.com, 2020

Ventajas y Desventajas

17. **Ventajas**

Entre las ventajas se puede mencionar lo siguiente:

- Define roles específicos a cada capa, lo que desacopla el desarrollo y a su vez lo facilita.
- Es un patrón sólido de propósito general, por lo que es un buen punto de partida para la mayoría de las aplicaciones, particularmente cuando no está seguro de qué patrón de arquitectura es el más adecuado para su aplicación.
- Debido a la isolación, los componentes de cada capa son fácilmente testeables.
- Es fácil de desarrollar porque la mayoría de las empresas desarrollan aplicaciones separando conjuntos de habilidades por capas (presentación, negocios, base de datos),

18. **Desventajas**

Así mismo se tiene desventajas las cuales son:

- Tiende a prestarse a aplicaciones monolíticas, incluso si divide la capa de presentación y las capas de negocios en unidades desplegables separadas.
- Poco ágil debido a su naturaleza monolítica.
- Su facilidad de despliegue se ve mermado cuando es utilizado para aplicaciones muy robustas.
- No está orientada a desempeñar un alto rendimiento debido a la necesidad de atravesar múltiples capas.

Uso del patrón basado en capas

Hay algo particular y es que cuando no se tiene claridad el patrón que se requiera utilizar, el estilo arquitectónico basado en capas es el más adecuado para el desarrollo de una aplicación

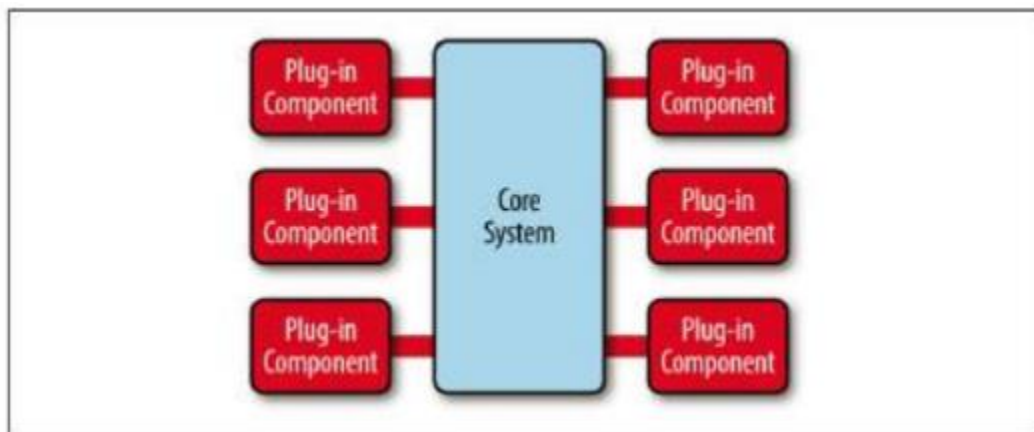
Arquitectura de Microkernel

Descripción

Este estilo arquitectónico como se mencionó anteriormente,

Para (Blancarte, 2020), permite agregar funciones adicionales de la aplicación como complementos a la aplicación principal, proporcionando extensibilidad, separación y aislamiento de características, la lógica se describe en la división de tres módulos de plug-in independientes del sistema core básico, proporcionando extensibilidad, flexibilidad y aislamiento de características de la aplicación y lógica de procesamiento personalizado.

Ilustración 29: Diagrama Arquitectura Microkernel



Fuente: Adoptado de (<https://programmerclick.com/article/96461809084/>)

Elaborado por: programmerclick.com, 2020

Como se puede observar en la ilustración 25 los plug-ins son componentes autónomos e independientes que contiene procesamiento especializado, funciones adicionales y código personalizado que está destinado a mejorar o ampliar el sistema central para producir capacidades de negocio adicionales.

Ventajas y Desventajas

19. **Ventajas**

- Puede estar embebido o usarse como parte de otro patrón de arquitectura.

- Agilidad en el desarrollo. Los cambios se pueden aislar e implementar en gran medida de forma rápida a través de módulos plug-in acoplados libremente.
- Dependiendo de la implementación del proyecto, los plug-ins pueden agregarse al core en tiempo de ejecución, minimizando el tiempo para despliegue.
- Los plug-ins se pueden probar de forma aislada sin afectar la funcionalidad del core.

20. **Desventajas**

- Debido a que la mayoría de las implementaciones de arquitectura de microkernel están basadas en productos y generalmente son de menor tamaño, se implementan como unidades únicas y, por lo tanto, no son altamente escalables.
- La arquitectura de microkernel requiere un diseño cuidadoso y una gobernanza contractual, por lo que es bastante complejo implementar

Uso de la Arquitectura Microkernel

Para determinar el uso de este estilo arquitectónico, particularmente se define para aquellas aplicaciones en las que liberará funciones adicionales a lo largo del tiempo y necesitará controlar qué usuarios obtiene dichas funciones.

USO DE LA ARQUITECTURA BASADA EN CAPAS (USO DE FRAMEWORK NESTJS)

NestJs es un framework de back-end la cual tiene una arquitectura sólida que facilita el desarrollo de aplicaciones escalables y mantenibles.

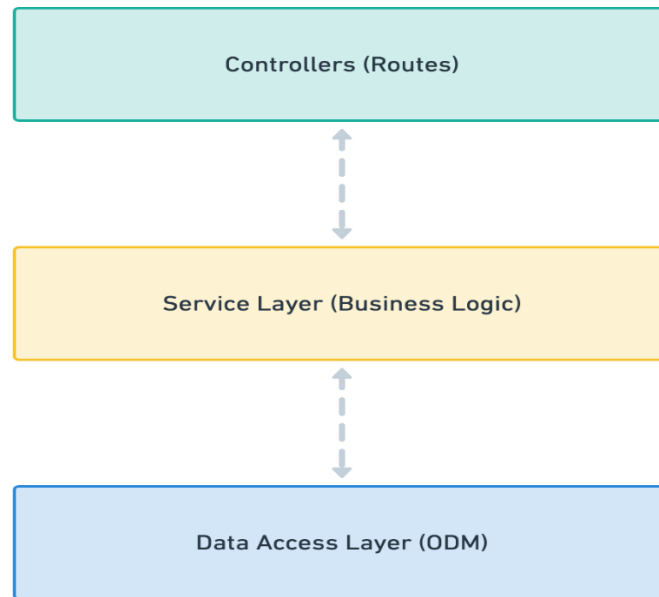
Esta se implementa o se basa en la arquitectura basada en capas, donde lo más importante son los módulos, el cual tienen una organización jerárquica, donde existe un módulo principal. Dentro de cualquier módulo se puede encontrar artefactos, que son siempre clases de programación orientada a objetos, entre las capas que se incluye son:

21. **Módulos:** módulos independientes
22. **Controladores:** se encargan de implementar los casos de uso de la aplicación, característicamente responden a las rutas de las aplicaciones web, es decir a las URL o endpoints de una API
23. **Providers:** aquí se encuentra muchas clases, de diversos tipos, entre las que tenemos los pipes, los servicios, repositorios, etc. Todas estas clases tienen una característica en común, que son inyectables en otras clases.

Algo interesante que se puede mencionar en el uso de este framework es que este tipo de arquitectura esta inspirada en el framework de Front-end (Angular).

Su desarrollo es similar al framework mencionado anteriormente la cual se establece de la siguiente manera:

Ilustración 30: Capas NestJS



Fuente: Adoptado de (<https://codigoencasa.com/nest-js-patron-arquitectonico-controladores-proveedores-y-modulos/>)

Elaborado por: Mariana Rolfo, 2022

Como se puede observar en la imagen anterior se puede determinar que los controladores y la capa de servicio trabajan juntos para llevar a cabo una lógica, pero son cosas completamente diferentes.

Teniendo en cuenta lo anterior como primera medida la capa donde se encuentran los controladores se ocupa esencialmente de las rutas de la aplicación web, un controlador puede tener varias rutas diferentes y todo depende del mecanismo de enrutamiento para controlar qué controlador recibe qué solicitud.

Ahora bien, si se implementa la lógica dentro de los controladores, como por ejemplo registrar un usuario si el usuario aún no está registrado o de lo contrario llevar a cabo controles de validación sólo dentro de los controladores.

Como ya se ha mencionado anteriormente esto no tendría mucho sentido, ya que probablemente no haría mucha diferencia si sólo se tiene una pequeña aplicación, el problema radica cuando dicha aplicación crece y se tiene que registrar mucho más controladores y rutas, así mismo escribir más lógica, ahí es donde las cosas probablemente empiezan a salirse de control y ciertamente no son mantenibles.

Controladores

El único propósito de un controlador es recibir peticiones de la aplicación y ocuparse de todas las rutas establecidas

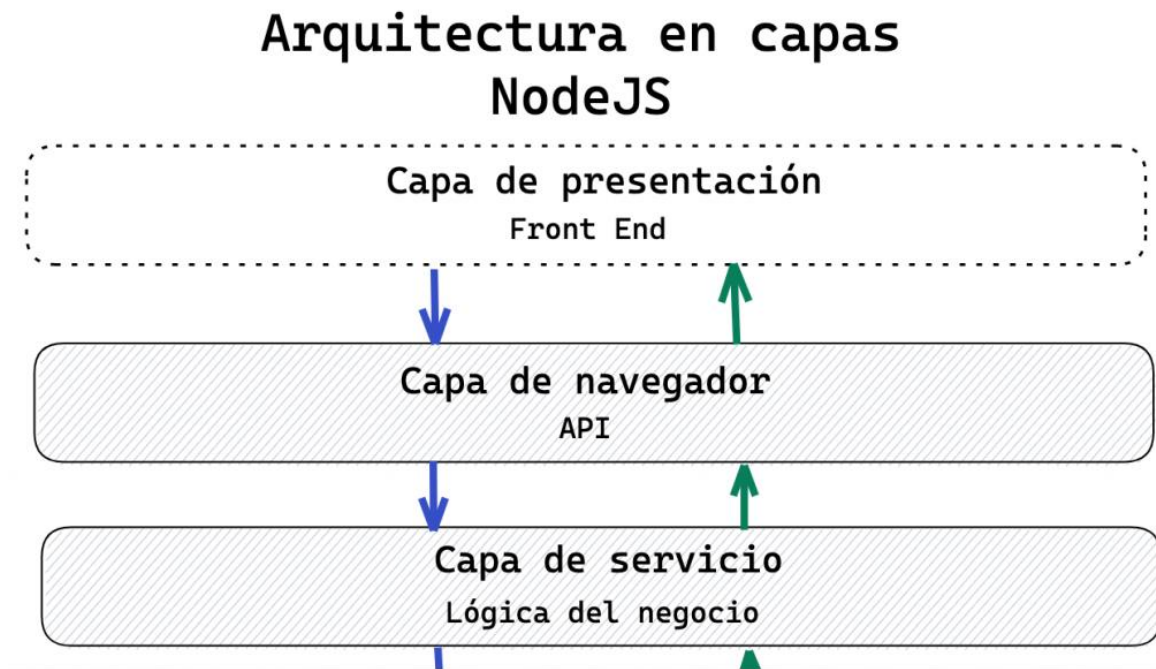
Capa de Servicio

En esta capa se incluye únicamente la lógica de negocio, un claro ejemplo de ello son todas las operaciones CRUD y los métodos que determinan cómo se pueden crear, almacenar y actualizar los datos.

Capa de acceso a los datos

Esta capa tiene un propósito importante y es la que se encarga de la lógica con el objetivo de acceder a los datos almacenados en un almacenamiento persistente y es aquí donde se da el uso de MySQL, MongoDB, entre otros.

Ilustración 31: Flujo entre capas



Fuente: Adoptado de (<https://ctrly.blog/es/arquitectura-capas/>)

Elaborado por: Yari Antonieta, 2021

Se puede observar en la ilustración 31 la cual representa la arquitectura en capas ya mencionado dando a conocer como fluye entre sus capas. Para desglosar mejor el comportamiento de este flujo se determina lo siguiente:

1. **Movimiento entre capas**

Todo empieza en la capa de presentación y es aquí donde el usuario llama una solicitud de datos la cual es enviada hacia el API que representa la capa de rutas, seguido de ello esta llama a la capa de servicio, y es aquí donde la capa del navegador espera respuesta de la capa de servicio para así retornar lo

solicitado por el usuario. Para llevar a cabo lo anterior los datos solicitados son calculados u operados por la capa de servicio justo antes de enviarlos al servidor, una vez son transformados, la capa de servicio procede a hacer el llamado al componente inyectado de la capa de base de datos y aquí suministra los datos. Por último, en la capa de base de datos se lleva a cabo la solicitud de datos al servidor, esta capa se estructura como una solicitud de una promesa. La promesa soluciona la respuesta del servidor.

Una vez se genera solución con la respuesta del servidor esta dará un retorno a la capa de servicio, para así mismo generar una respuesta a la capa superior llamada capa de navegador que es donde se encuentran todas las rutas.

Finalmente, esta respuesta llega al usuario por medio de la primera capa que corresponde a la capa de presentación, con ello se da claridad de que ninguna capa saltará a otra capa.

2. Implementación de arquitectura en capas

El uso de esta arquitectura se lleva a cabo reglas importantes las cuales van a determinar cómo se debe comportar la comunicación entre las capas.

Como primera regla, se debe tener una responsabilidad única por cada capa, para tener claridad de ello las capas deben encontrarse perfectamente delimitadas, dando, así como ejemplo la capa de **presentación** que es la encargada de atender al cliente y así mismo de representar la información para el mismo.

Como segunda regla se tiene que una vez creada dichas capas estas se deben representar en una estructura jerárquica, ya que su comunicación debe ser debajo de dicha capa y no al contrario. Adicional

a ello no debe haber ningún salto entre capas ya que esto violaría la regla y podría generar lo conocido como código espagueti. Tomado de sitio web (sospnt.com)

Ilustración 32: Comunicación entre capas



Fuente: Adoptado de (<https://ctrly.blog/es/arquitectura-capas/>)

Elaborado por: Martin Bertucelli, 2019

CAPITULO III

En este capítulo se hace una justificación del porque se escogió la arquitectura en capas para llevar a cabo este proyecto y la implementación del mismo, indicando la arquitectura a desarrollar y el porqué de su uso, así mismo se muestra dos patrones de diseño con el objetivo de hacer uso de uno para así definir el porqué de su uso, posteriormente se establece el desarrollo del proyecto dando como inicio la solución del primer objetivo específico para llegar hasta el último objetivo específico que sería el tercero, por último se desarrolló las conclusiones y las recomendaciones, junto a ello dando así la finalidad del proyecto con las referencias bibliográficas que ayudaron a la construcción del trabajo de grado para determinar el éxito del mismo.

ARQUITECTURA EN CAPAS

¿Por qué la arquitectura en capas?

Definir la arquitectura más adecuada para el desarrollo del presente proyecto ha requerido un análisis profundo de diferentes arquitecturas, para ello se ha estudiado la arquitectura microservicios, la arquitectura microkernel y la arquitectura en capas. A continuación, se explicarán 3 razones fundamentales por las cuales se ha decidido que la opción más viable ha sido la arquitectura en capas

Primero, la arquitectura en capas hace que reduzca el riesgo y el impacto de los cambios tecnológicos en comparación con arquitectura microservicios y la arquitectura microkernel. Uno de los riesgos que puede presentar la arquitectura microservicios es que genera mayor consumo de recursos, dado que cada microservicio tiene su propio sistema. Por otro lado, la arquitectura microkernel presenta un nivel de riesgo en el mantenimiento ya que puede resultar difícil asegurar la compatibilidad en todos los plugins, si en algún momento se realizan cambios en el sistema. Por el contrario, la arquitectura en capas disminuye el riesgo y el impacto de los cambios tecnológicos al momento de realizar cambios en el

mecanismo de persistencia, pues no será necesario realizar cambios en otras capas, además esto aumenta el rendimiento ya que la carga de trabajo no está determinada por una sola capa y esto hace que aumente la escalabilidad y así mismo la tolerancia a fallos.

Segundo, el rendimiento es otra de las ventajas de la arquitectura en capas frente a la arquitectura microservicios y la arquitectura microkernel. La arquitectura microservicios ha sido analizada en cuanto a su rendimiento, se ha encontrado que si los microservicios no están bien diseñados esto hace que atender las solicitudes del usuario obligue a pasar por muchos microservicios y esto hace que procesar las solicitudes sea más lento, mientras que la arquitectura en capas muestra una ventaja en este aspecto debido a que la responsabilidad de las solicitudes del usuario están establecidas de manera independiente dando así una mejor respuesta a dichas peticiones por el usuario. La arquitectura microkernel se caracteriza por su rendimiento en que puede personalizar y racionalizar las aplicaciones para incluir sólo las características que necesita.

Sin embargo, la arquitectura en capas, en cuanto a su agilidad para el presente proyecto se ha seleccionado, considerando que no se requiere que dicha aplicación sea competitiva con relación a otras entidades, y no se hace necesario una proyección de alto rendimiento en la persistencia de los datos, ya que estos se encuentran estandarizados por políticas del Gobierno Nacional establecidas en la resolución 3100 del 2019 y la resolución 256 de 2016.

Tercero, la arquitectura en capas facilita llevar a cabo la testeabilidad para detectar fallos, lo cual es más complejo en la arquitectura microservicios y la arquitectura microkernel. Para detectar fallos en la arquitectura microservicios se requiere como primera medida una prueba funcional que es donde se aplica la lógica empresarial y el comportamiento del servicio, esto es complicado ya que los microservicios no tienen una interfaz de usuario para facilitar de alguna manera el examen que se

requiere para detectar el fallo, por otro lado, si no se diseña correctamente dicha arquitectura puede causar choques debido al alto nivel de tráfico. Por otro lado, el proceso de detección de fallos en la arquitectura microkernel es más compleja por el testeo de los plugins ya que depende según las características y el funcionamiento del mismo, dando así que sean muy dependientes de otros plugins. A diferencia de las dos arquitecturas mencionadas, en la arquitectura en capas permite probar cada una de las partes de manera aislada sin necesidad de mantener todo el sistema corriendo, y esto ha hecho que esta arquitectura sea la más utilizada para el desarrollo de aplicaciones.

En conclusión, la arquitectura en capas ha sido la opción más viable para implementarla en el presente proyecto enfocado a la Secretaría de Salud del Putumayo, debido a que reduce en gran medida el riesgo y el impacto a los cambios tecnológicos, permite mayor agilidad en los procesos, y facilita considerablemente el testeo de fallas permitiendo realizar este proceso de manera aislada.

PATRÓN DE DISEÑO

Para determinar el uso de un patrón de diseño se tuvo en cuenta dos de ellos los cuales corresponden al patrón de diseño Data Access Object (DAO) y el patrón de diseño Data Transfer Object (DTO).

Patrón de Diseño Data Access Object (DAO)

Por medio de este patrón de diseño se propone separar por completo la lógica de negocio de la lógica para acceder a los datos, de esta manera, el DAO brindará todos los métodos necesarios del CRUD, en este sentido, este es utilizado en la capa de negocio dando así el uso de este patrón de diseño (DAO) con el único objetivo de poder interactuar con los datos.

Patrón de Diseño Data Transfer Object (DTO)

Particularmente en ese patrón de diseño se propone la creación de clases en la cual se compone por una serie de atributos que pueden ser enviados o recuperados del servidor en una sola invocación, dando así que en un patrón de diseño DTO pueda contener toda la información pertinente ya sea de múltiples fuentes o tablas, para así concentrar todo en una simple clase.

Por medio de una clase se podrá controlar los datos enviados, además, no solo enviar si no también cambiar estos datos si así se requiere, algo particular es que este patrón de diseño no tiene un impacto en la capa de servicios, ya que solo se encargan de transmitir una respuesta.

¿Por qué el uso del patrón de diseño DTO?

Como ya se mencionó anteriormente este patrón de diseño es muy efectivo para transmitir información entre un cliente y un servidor, para ello se necesita establecer dicha transmisión de datos en la arquitectura en capas, dando así la posibilidad de crear estructuras de datos independientes del modelo obtenido en la entidad – relación, así mismo dará facilidad de controlar el formato, los nombres y los tipos de datos que se utiliza para dar uso a un determinado requerimiento.

Como última medida para dar claridad a este patrón es que, si se llega a cambiar el modelo de los datos en una base de datos, este no se va a alterar.

DESARROLLO DEL PROYECTO

Para el respectivo desarrollo del proyecto, se estableció el uso de los objetivos específicos dando así una guía para determinar el desarrollo que se define en el objetivo general.

SOLUCIÓN OBJETIVO ESPECIFICO UNO

El desarrollo de este objetivo dará a conocer toda la información requerida por parte del área de sistema obligatorio de garantía de la calidad.

Para ello se logra obtener las características técnicas que se tendrá en cuenta para el uso del estilo arquitectónico a implementar

3. Usuarios:

Por parte de la información extraída, los usuarios obtendrán roles establecidos por un usuario administrador, donde este determinará los permisos que tendrá cada usuario en el módulo correspondiente, dando así el funcionamiento al administrador el poder de crear, editar, actualizar y eliminar usuarios que así lo requiera.

4. Seguridad:

Posteriormente, se llevará acabo el uso de un proceso de autenticación, para así llevar acabo un control de auditoría de cada una de las acciones realizadas por el usuario, teniendo en cuenta los parámetros de autenticación que se requiera, así mismo el administrador llevará el proceso de control de usuarios

5. Prestadores de servicio de salud:

Los prestadores de servicio de salud son entes que prestan un servicio y para ello el administrador llevará acabo la información pertinente de dichos prestadores para así determinar el estado de estos los cuales se encuentran establecidos en la plataforma del REPS que brinda el Ministerio de Salud y Protección Social, dando así el uso de un CRUD correspondiente a los prestadores de servicio de salud.

6. Indicadores de servicios

Para determinar los requisitos que deberá tener en cuenta un prestador de servicio de salud, son los indicadores establecidos por una resolución la cual se debe permitir adicionar, editar y eliminar diferentes indicadores que se encuentran ya estipulados por el Ministerio de Salud y Protección Social

7. Evaluación a prestadores de servicio de salud

El área de sistema obligatorio de garantía de la calidad deberá llevar a cabo el proceso de evaluación que se le realiza a un prestador de servicio de salud, teniendo en cuenta todos los requerimientos establecidos, para así dar cumplimiento con lo por lo estipulado por el Gobierno Nacional.

8. Reportes

Por medio del proceso de evaluación a los prestadores de servicio de salud, se generará reportes necesarios para así dar cumplimiento con un informe el cuál determinará el cumplimiento de un prestador de servicio de salud.

SOLUCIÓN OBJETIVO ESPECIFICO DOS

Se realizó un análisis de la información extraída para así dar claridad de lo requerido por la Secretaría de Salud de la Gobernación del Putumayo dirigido al área de sistema obligatorio de garantía de la calidad, dando como resultado lo siguiente:

1. Usuarios:

Como primera medida se detalla con claridad los roles que deberá tener en cuenta para el acceso de cada usuario en sí.

Con base en lo anterior se pudo analizar que los roles necesarios son los siguientes:

1. Administrador:

Permitirá tener control de los usuarios con el objetivo de determinar detalladamente que acciones a realizados un usuario en específico

2. SIC (Sistema de información de calidad):

Llevará a cabo el proceso de evaluación teniendo en cuenta los indicadores que se encuentran en la resolución 256 del 2016 establecida por el Gobierno Nacional.

3. PAMEC:

Obtendrá el control de auditoría a los prestadores de servicio de salud teniendo en cuenta unos formatos estipulados por el área de sistema obligatorio de garantía de la calidad

4. Seguridad del Paciente:

El área de garantía de la calidad, brinda unos formatos para así llevar a cabo el proceso de evaluación a las EPS e IPS que servirán para llevar el control en la prestación del servicio de salud.

5. Resolución 3100 de 2019:

El Ministerio de Salud y Protección Social establece una nueva resolución, la cual establece unos requisitos para dar cumplimiento a los prestadores de servicio de salud, dando así los cumplimientos que utilizará el usuario para definir la evaluación que se le realizará al prestador de servicio de salud

1. Seguridad:

Adicionalmente a lo anterior, urge la necesidad de establecer un sistema de autenticación para dar acceso a los usuarios y así habilitar las funcionalidades necesarias por cada Rol, con esto se da garantía de que un rol a excepción del Administrador tenga acceso a otros roles.

2. Prestadores de servicio de salud:

Siguiendo el análisis de la información extraída, es necesario establecer la información de los prestadores de servicio de salud brinda por el Ministerio de Salud y Protección Social, determinando el estado de estos según la información brindada.

3. Indicadores de Servicio

Se estudia la resolución 256 de 2016, con el objetivo de extraer todo los dominios, indicadores y criterios que estipula la misma, para así tener los lineamientos necesarios para dar uso el sistema de evaluación que se le hace a un prestador de servicio de salud. Para ello es necesario establecerlo en el modelo.

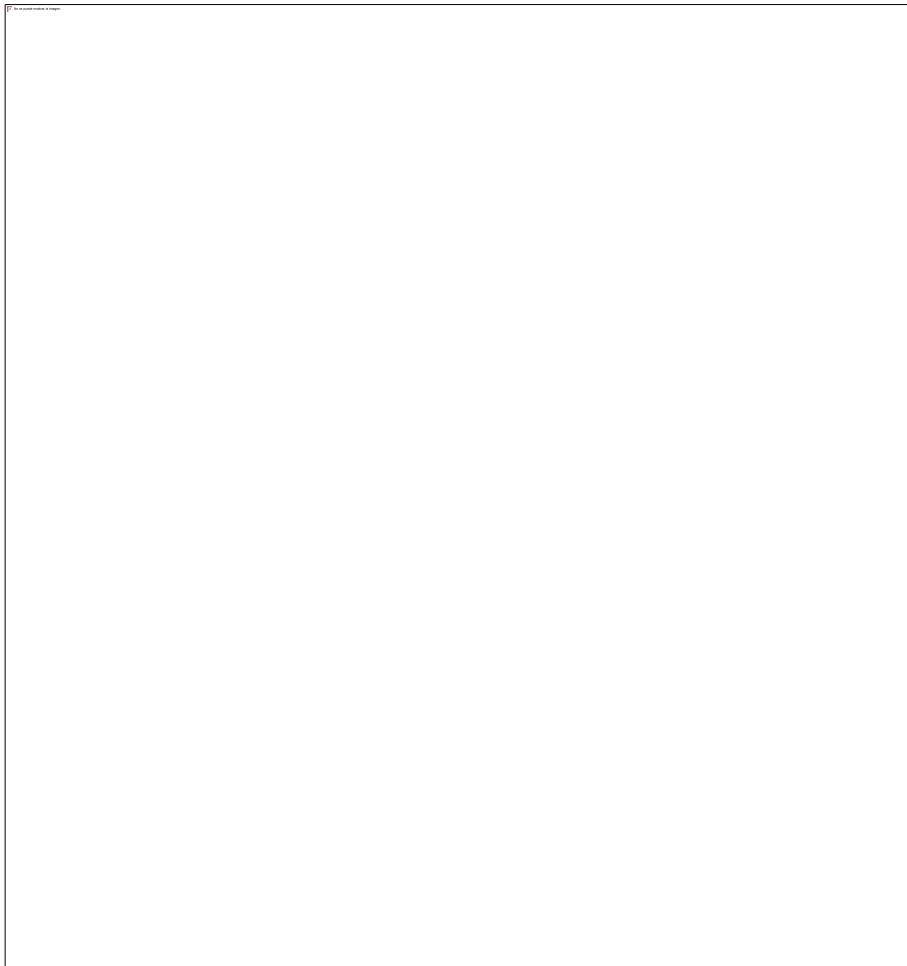
4. Evaluación a prestadores

Después de ver lo obtenido se procede a dar un diseño de cómo se realizará el proceso de evaluación con el fin de dar cumplimiento a en la prestación del servicio de salud establecido por el Gobierno Nacional, es aquí donde se realiza las pautas necesarias para poder ingresar, modificar y eliminar todos los criterios necesarios para dicha evaluación.

SOLUCIÓN OBJETIVO ESPECIFICO TRES

Para el desarrollo de este objetivo se realizó diferentes procesos para implementar la arquitectura en capas, como primera medida la implementación general del modelo en capas

Ilustración 33: Implementación modelo en capas



Elaborado por: Autoría Propia

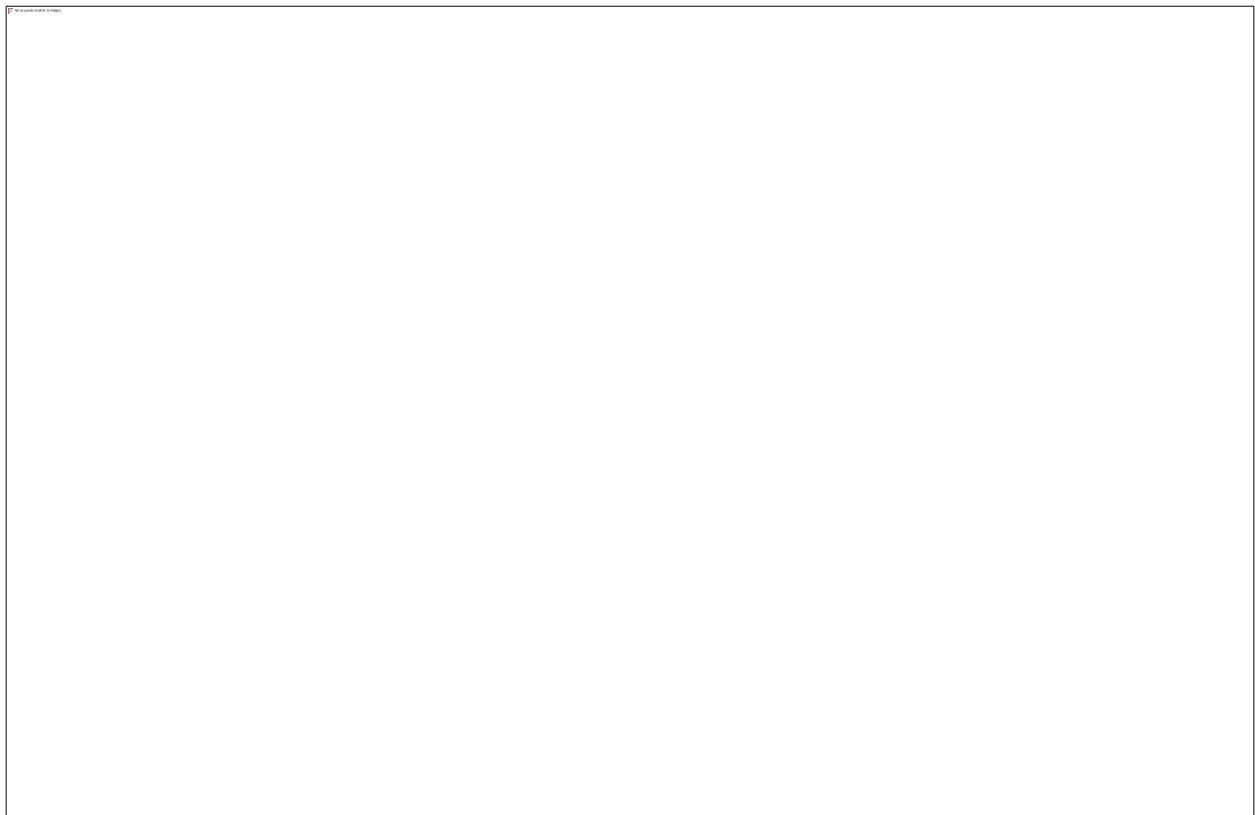
En la ilustración 33 se estructura el modelo a seguir dando así el uso de 4 capas las cuales son:

1. La capa 2: La que se va a encargar de la presentación (vista)

2. La capa 1: Brindará toda la lógica de negocio
3. La capa 0: Se encargará del modelo de sistema de base de datos

Posteriormente se procede a implementar el modelo de las capas, como primera medida se lleva a cabo el proceso de la capa de presentación, la cual se desglosará para dar claridad a lo requerido en dicha capa.

Ilustración 34: Capa de presentación



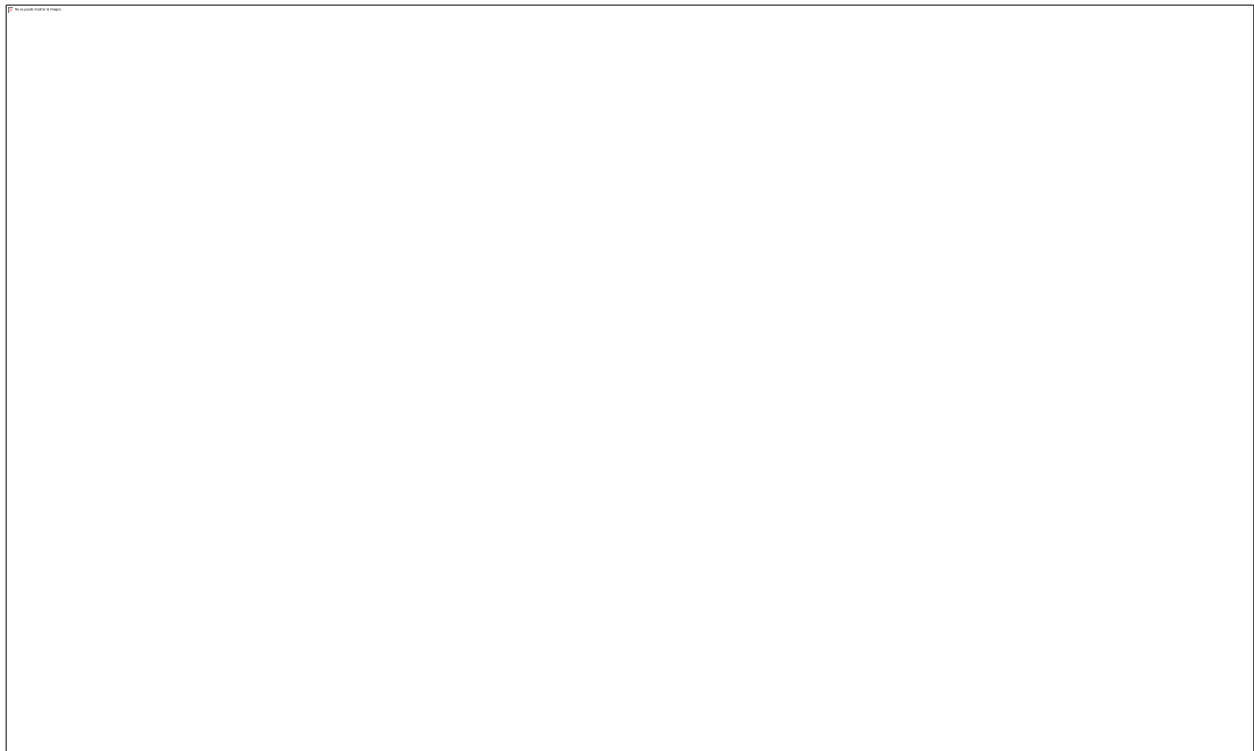
Elaborado por: Autoría Propia

En la ilustración 34, se puede observar cómo será la arquitectura en la capa de presentación, el funcionamiento de esta se representa por la interacción del usuario por medio de la plantilla, seguido de la vista la cual transmitirá esas peticiones al componente, en el modelo de vista el componente almacena los atributos públicos para que la vista pueda acceder a ellos sin ningún problema,

adicionalmente almacena los métodos para que la vista pueda dar uso de ellos y así cumplir con ciertas funcionalidades ya sea para dirigirse a la capa de negocio o en si a la lógica de interfaz de usuario. Toda petición que se hace a la capa de negocio es transmitirá por el servicio de forma asincrónica con la ayuda de del patrón DTO que servirá para establecer la comunicación.

Ya dando claridad del uso de las vistas, el componente y los servicios, se realiza el modelo de capas con las view necesarias, para dar ejemplo de ello se tiene lo siguiente:

Ilustración 35: Views



Elaborado por: Autoría Propia

En la ilustración 35, se crean las vistas necesarias para llevar a cabo la responsabilidad de la capa de presentación, dando a conocer los componentes y servicios, junto a un core que servirá para que en

algún momento dado se necesite en otra aplicación, este se de uso de ese paquete y así sea fácil su reutilización.

Capa de presentación Secretaría de Salud de la Gobernación del Putumayo

Ya aclarado como se define la arquitectura en la capa de presentación se procede a realizar el modelo con los requerimientos solicitados por la Secretaría de Salud de la Gobernación del Putumayo, los cuales fueron extraídos y analizados por los dos objetivos específicos anteriores.

Se definen las vistas para la capa de presentación las cuales se determina que son las siguientes:

1. VIEW-HOME
2. VIEW-MENU
3. VIEW-FOOTER
4. SERVICIOS
5. GUARDS
6. INTERCEPTOR
7. VIEW-AUTENTICACIÓN
 1. View-login
 2. View-change-password
8. VIEW-USUARIO-ADMIN
 1. View-control-usuarios
 2. View-indicadores
 3. View-prestadores

9. VIEW-ROLES

1. View-rol-pamec

1. View-rol-pamec-home

2. View-rol-pamec-acta

3. View-rol-pamec-evaluación

4. View-rol-pamec-evaluaciones

1. View-rol-sic

1. View-rol-sic-home

2. View-rol-sic-acta

3. View-rol-sic-evaluación

4. View-rol-sic-evaluaciones

5. View-rol-sp

6. View-rol-sp-home

7. View-rol-sic-acta_ips

8. View-rol-sic-acta_pro

9. View-rol-evaluación-ips

10. View-rol-evaluación-pro

11. View-rol-evaluaciones-ips

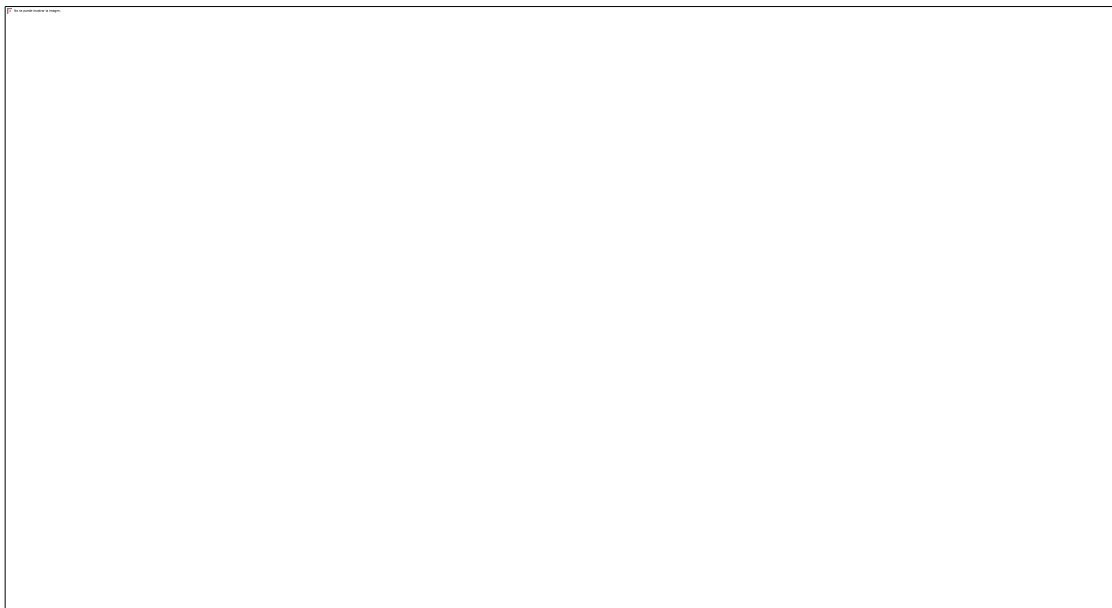
12. View-rol-evaluaciones-pro

13. View-rol-reso

1. View-rol-reso-home
2. View-Requerimiento

Con base en lo anterior, una vez definido las vistas, servicios, guard e interceptor se proceden a realizar el diseño gráfico de la arquitectura esperada

Ilustración 36: View-Home y View-Menu



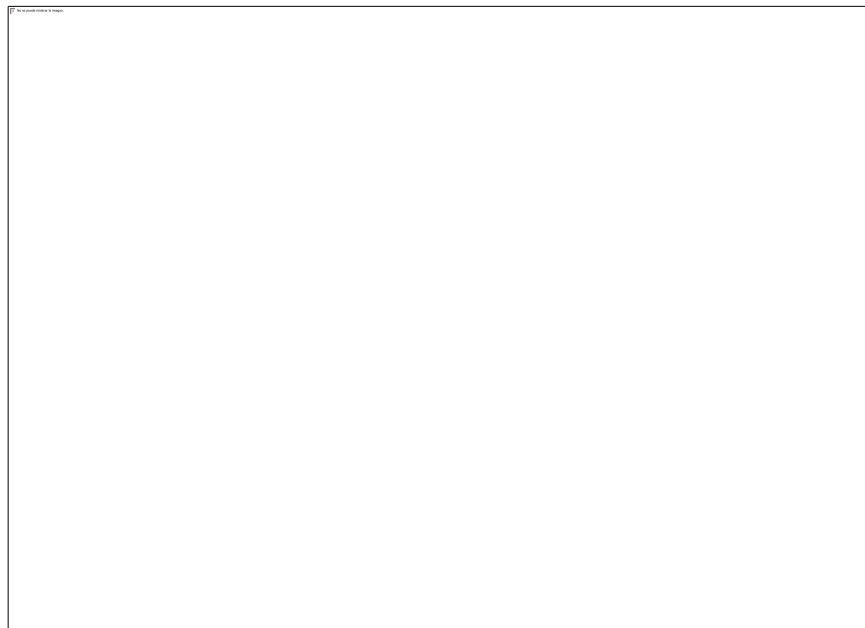
Elaborado por: Autoría Propia

Ilustración 37: View-control-usuario y View-prestador



Elaborado por: Autoría Propia

Ilustración 38: View-indicadores



Elaborado por: Autoría Propia

Ilustración 39: View-login y View-password



Elaborado por: Autoría Propia

Ilustración 40: View-Rol-Pamec



Elaborado por: Autoría Propia

Ilustración 41: View-Rol-Sic



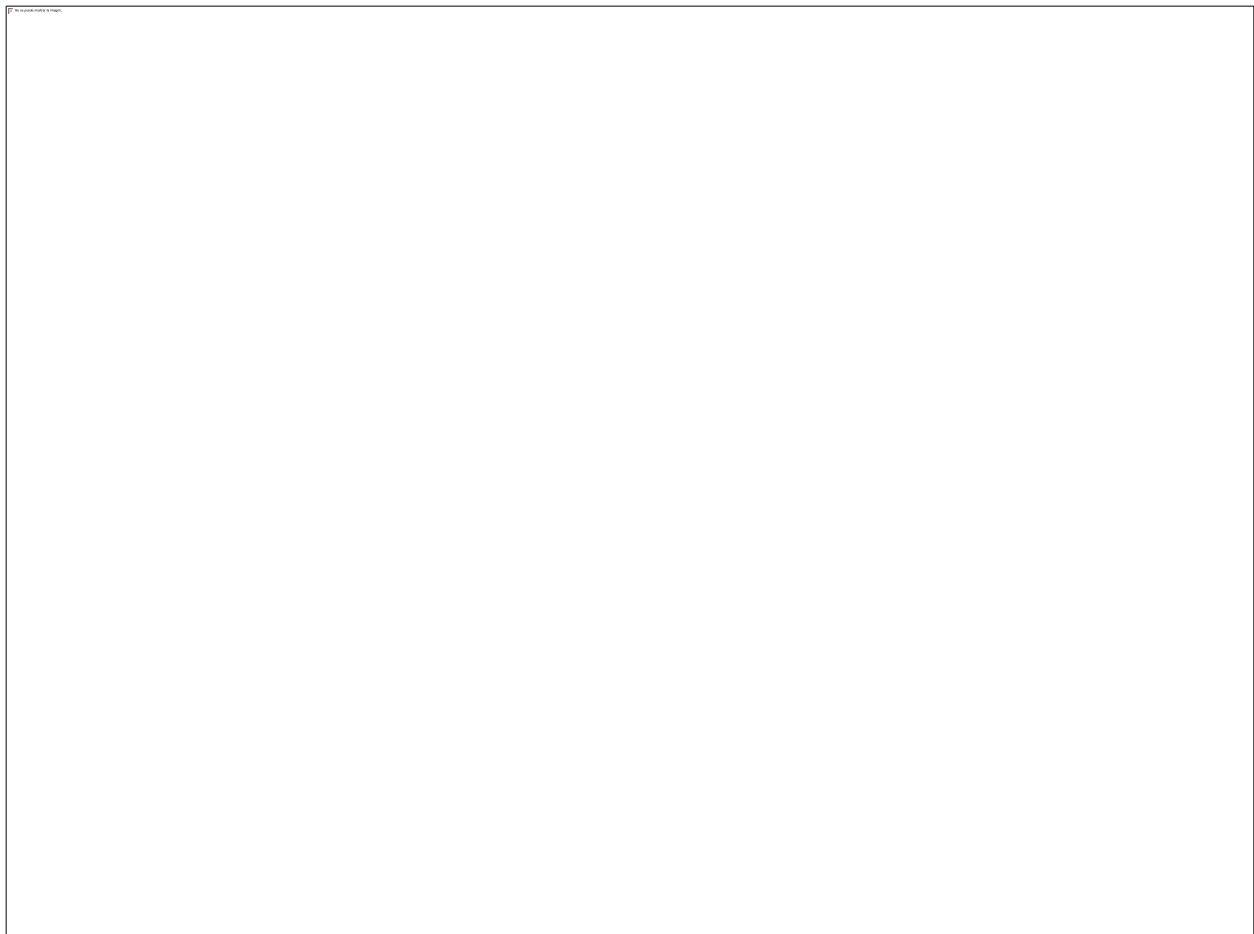
Elaborado por: Autoría Propia

Ilustración 42:View-Rol-SP



Elaborado por: Autoría Propia

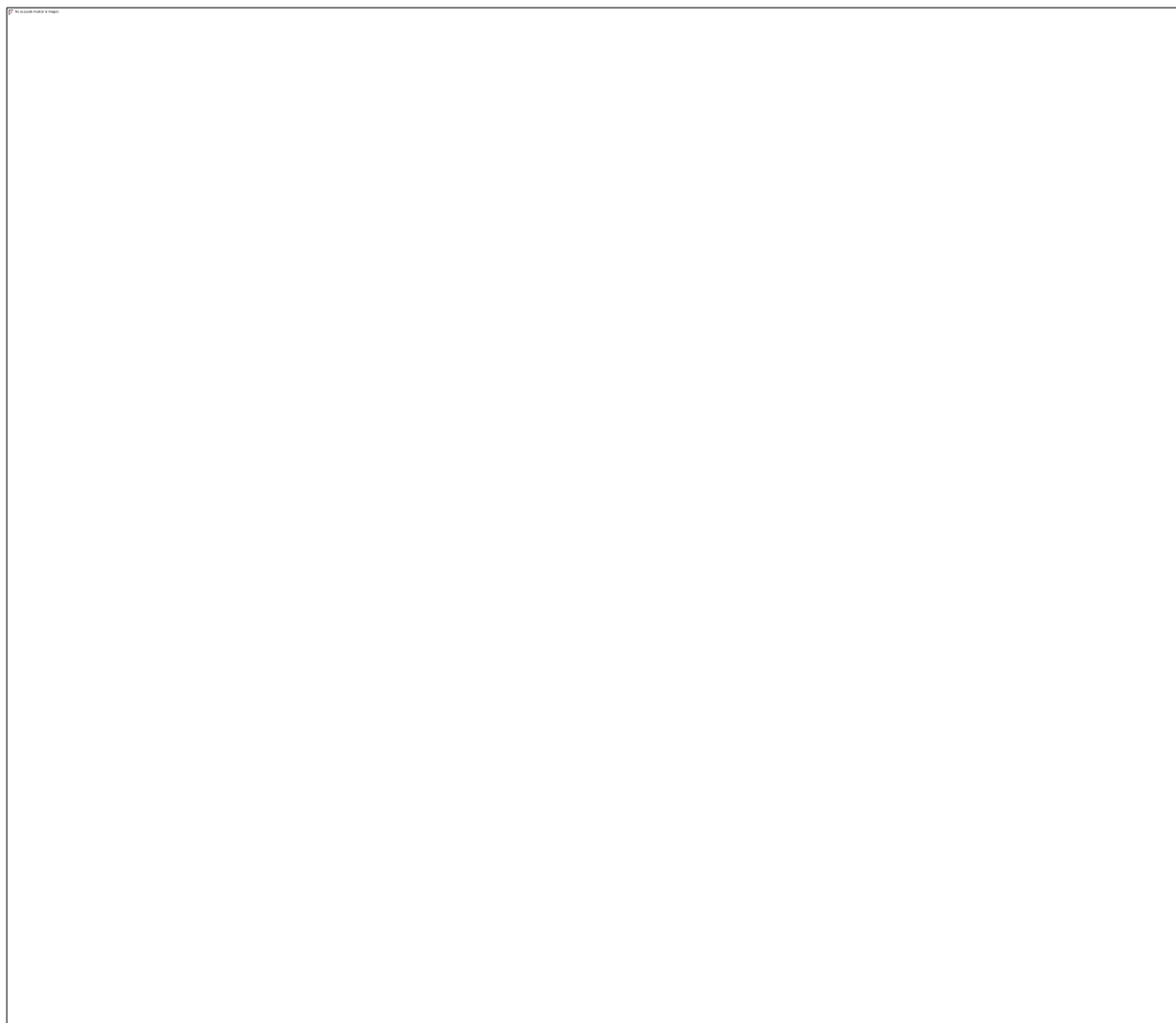
Ilustración 43: View-Rol-Reso



Elaborado por: Autoría Propia

A continuación, se presenta en la misma capa de presentación la implementación de los servicios junto a los guards que funcionará para dar un orden específico a las rutas dependiendo el rol del usuario que haya iniciado sesión.

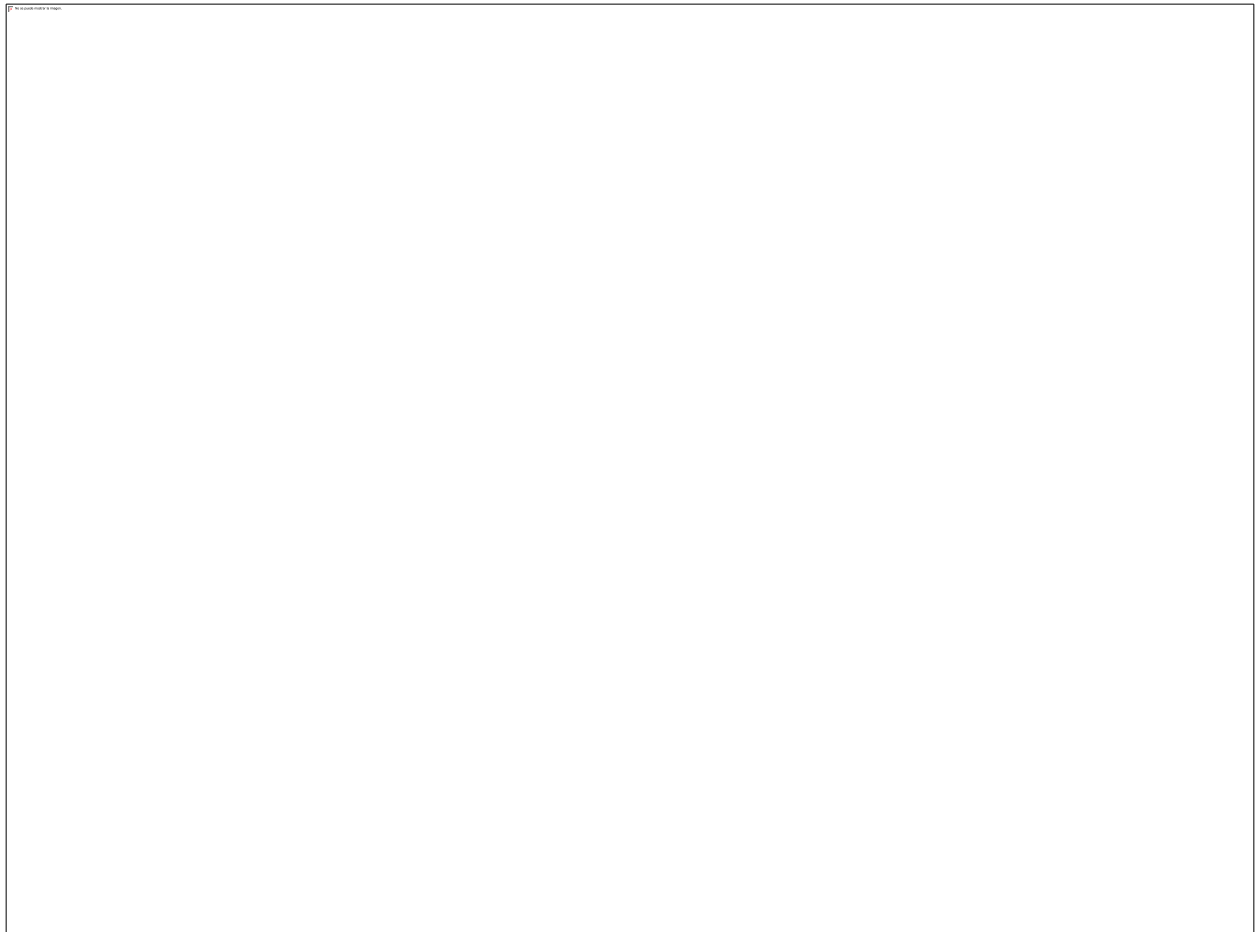
Ilustración 44: Servicios



Elaborado por: Autoría Propia

Posteriormente se presenta el modelo de Guards implementado en la capa de presentación

Ilustración 45: Guards



Elaborado por: Autoría Propia

Capa de Servicio (lógica de negocio) Secretaría de Salud de la Gobernación del Putumayo

En el proceso de desarrollo de la capa de servicio (lógica de negocio), se establece el conjunto de algoritmos que van a resolver el trabajo que el usuario desea o necesita ejecutar.

En esta capa se estableció el uso de un Framework de Back-end conocido como NestJs, el cual tiene una estructura muy similar a la del Framework de Front-end (Angular), dando así el uso de componentes y servicios, adicionalmente tiene controladores que permiten el uso de los métodos GET, POST, DELETE y PUT para recibir las peticiones del usuario por parte de la capa de presentación, así mismo también

contiene módulos que permiten agrupar elementos con el propósito de contener varios módulos con clases altamente relacionadas entre sí.

Para la construcción de esta capa se separó por componentes para así tener un control ordenado de las solicitudes del usuario.

1. AUTENTICACIÓN

2. GUARDS

3. ROL

4. PRESTADOR

1. CLASE

2. CLASIFICACION

3. TIPO

1. USUARIO

2. SIC

3. SP

4. PAMEC

5. RESO

6. MUNICIPIO

7. APPMODULE

Ilustración 46:Componente Autenticación



Elaborado por: Autoría Propia

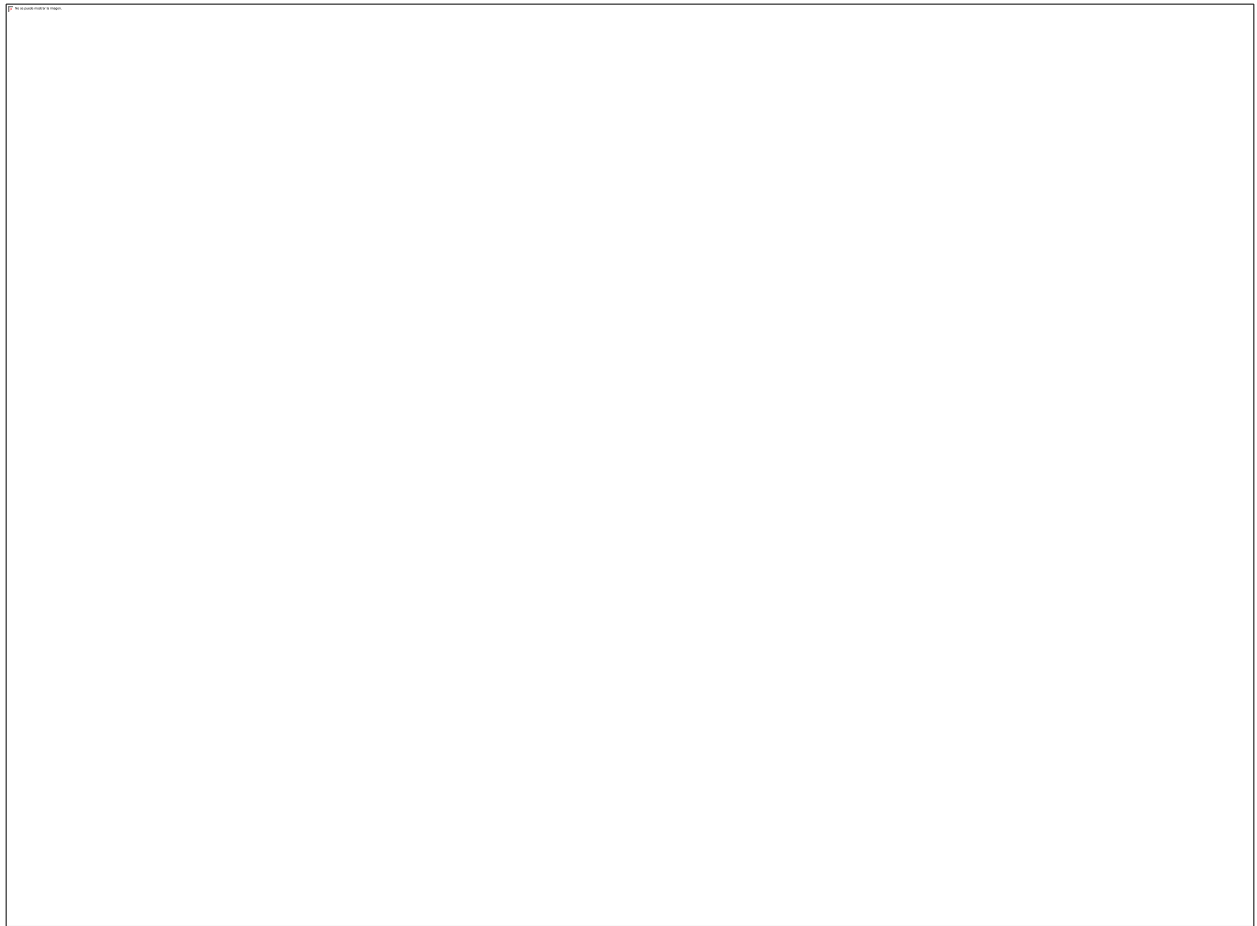
En la ilustración 46, en el componente de autenticación se define los servicios y controladores que darán uso a los DTO. Este componente de autenticación se almacena en el APP-MODULE que es el que determina la conexión hacia la capa de datos para así dar respuesta a lo requerido por el componente de autenticación.

Ilustración 47: Componente Guard



Elaborado por: Autoría Propia

Ilustración 48:Componente Rol

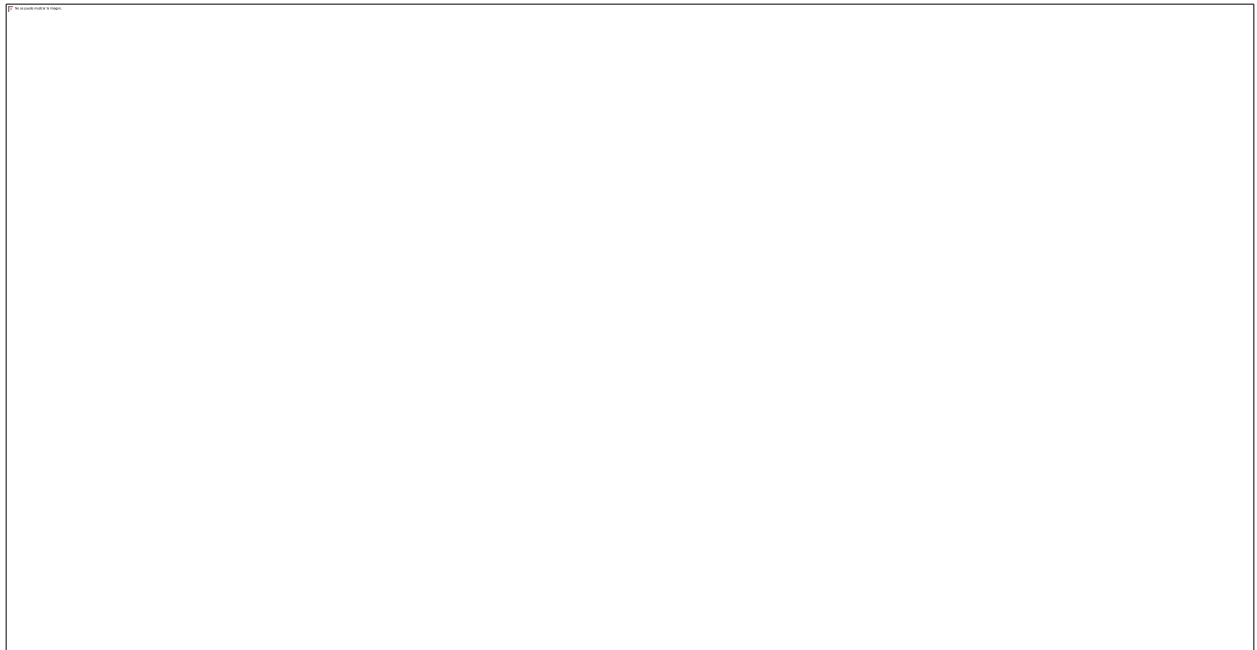


Elaborado por: Autoría Propia

En la ilustración 47, se definen los servicios, controladores y los DTO.

Posteriormente el componente es almacenado en el APP-MODULE.

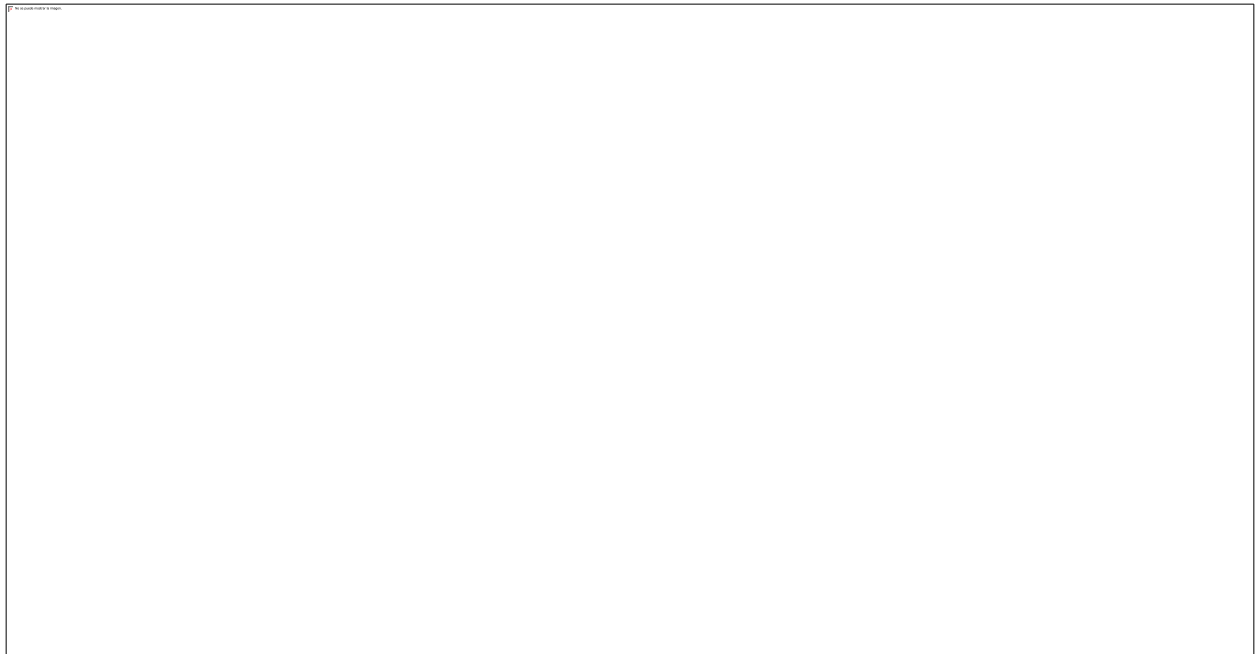
Ilustración 49: Componente SIC



Elaborado por: Autoría Propia

En el componente SIC haciendo uso de un ORM (Object-Relational Mapper) el cual es un modelo que permite mapear las estructuras de una base de datos, se define las entidades que necesita el componente SIC.

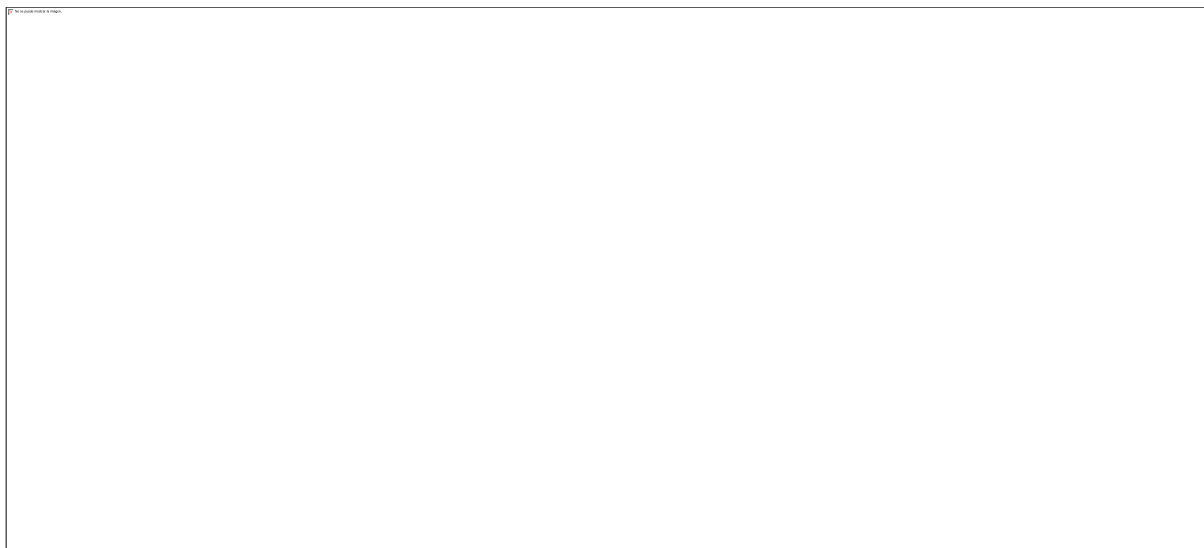
Ilustración 50: Componente Pamec



Elaborado por: Autoría Propia

En la ilustración 49, se define las entidades que necesita el componente PAMEC.

Ilustración 51: Componente SP



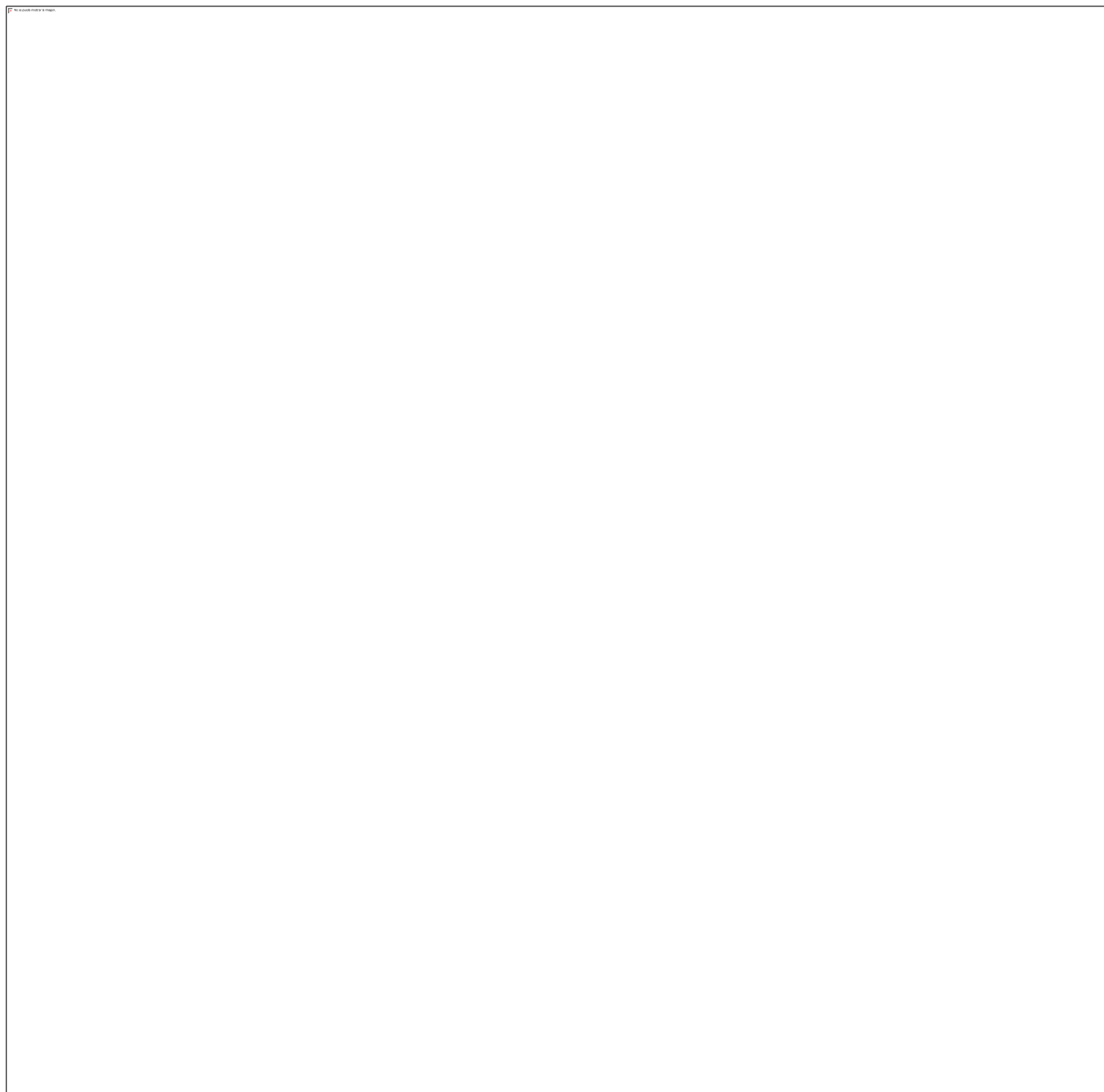
Elaborado por: Autoría Propia

Ilustración 52:Componente Usuario



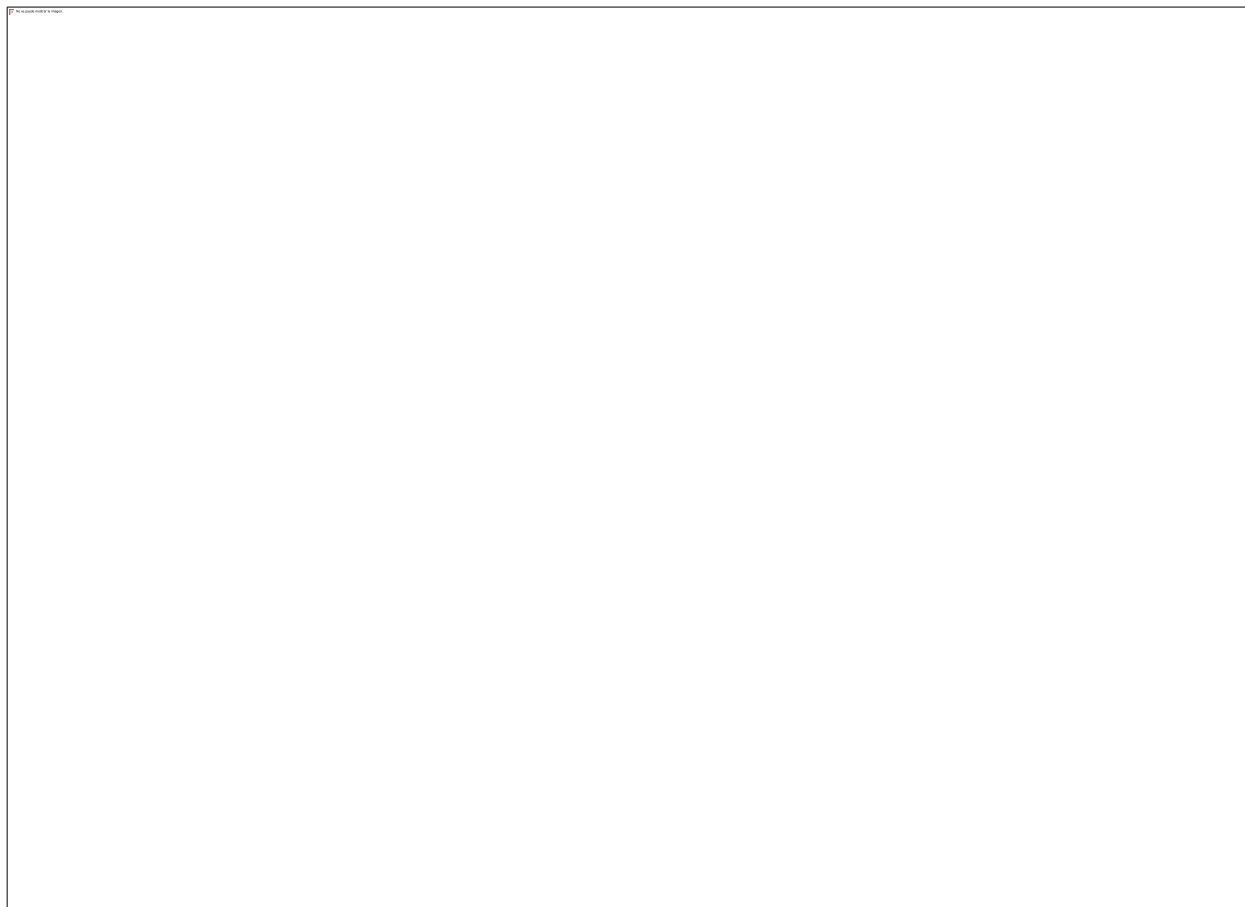
Elaborado por: Autoría Propia

Ilustración 53: Componente Prestador



Elaborado por: Autoría Propia

Ilustración 54: Componente Municipio



Elaborado por: Autoría Propia

Capa de Base de Datos (Persistencia de Datos)

En el proceso de desarrollo de esta capa fue necesario solo establecer la creación de una base de datos con el objetivo de almacenar todas las entidades que fueron creadas por el modelo ORM, dando así el uso de una base de datos relacional (SQL Server, Oracle, MySQL, etc.)

VALIDEZ DE LA ARQUITECTURA EN CAPAS

Para dar viabilidad al prototipo de esta arquitectura por medio del desarrollo que aún se encuentra en proceso se obtiene lo siguiente:

Capa de Presentación:

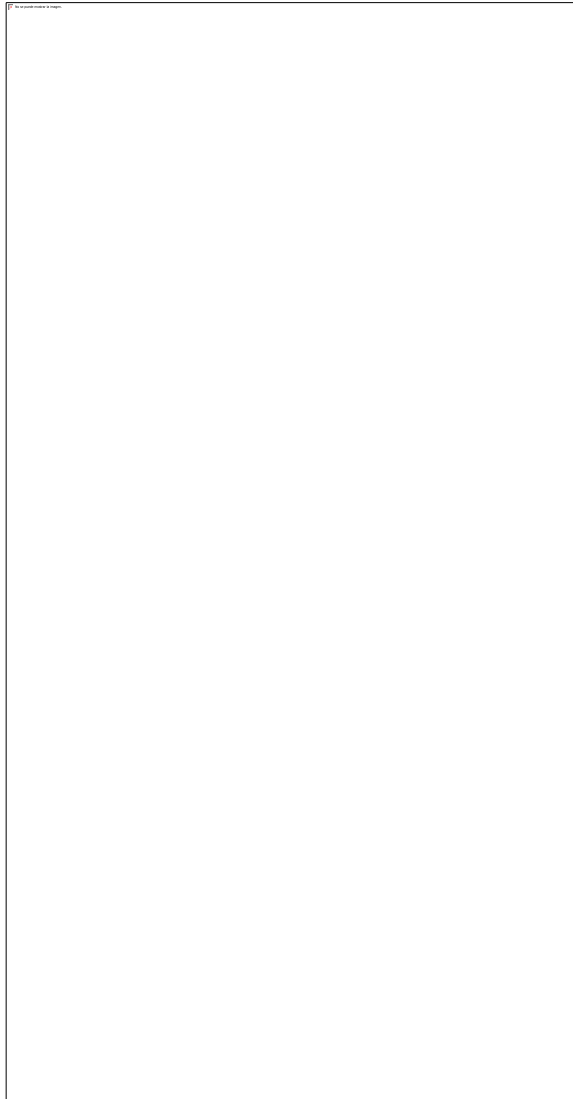
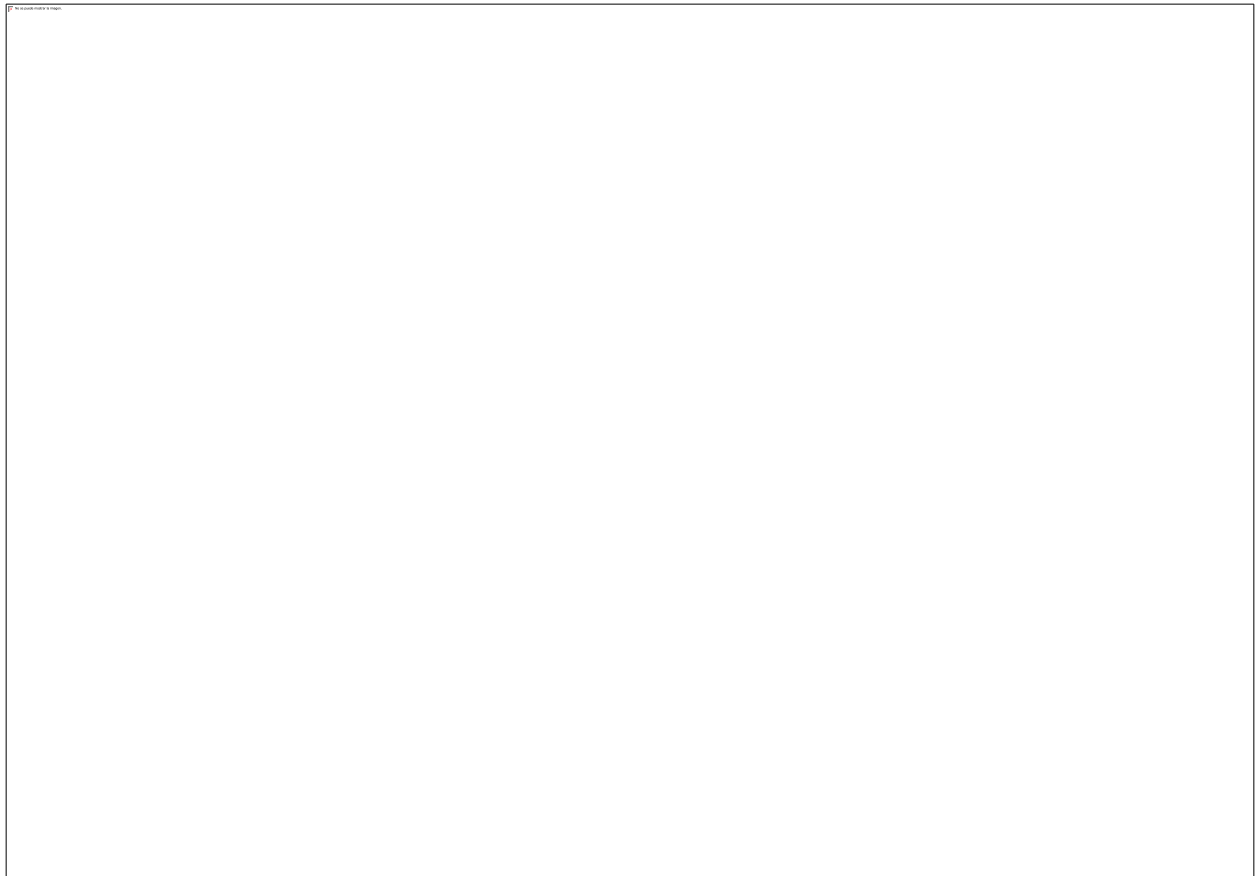


Ilustración 55: Capa de Presentación - Home



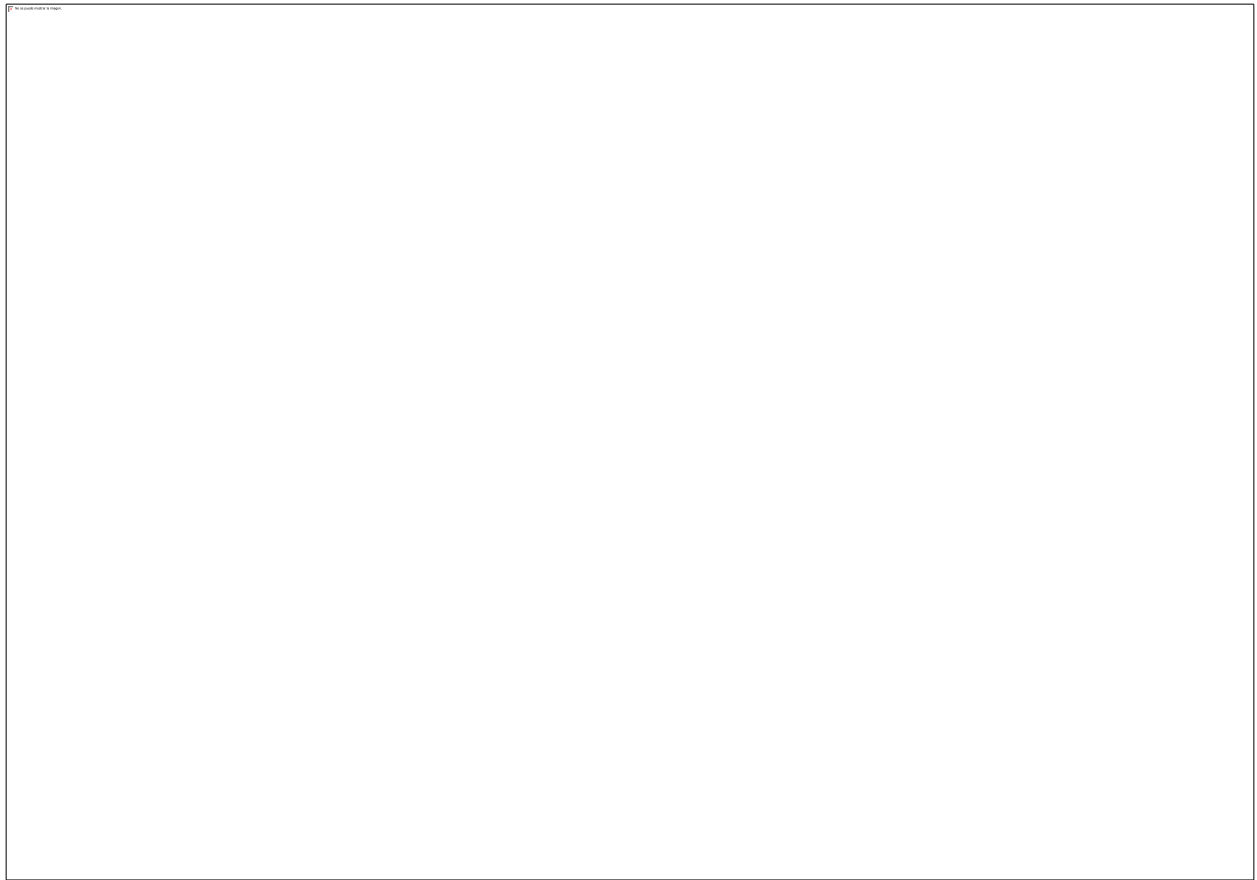
Elaborado por: Autoría Propia

En la ilustración 55 se observa el Home de la capa de presentación

Ilustración 56: Login

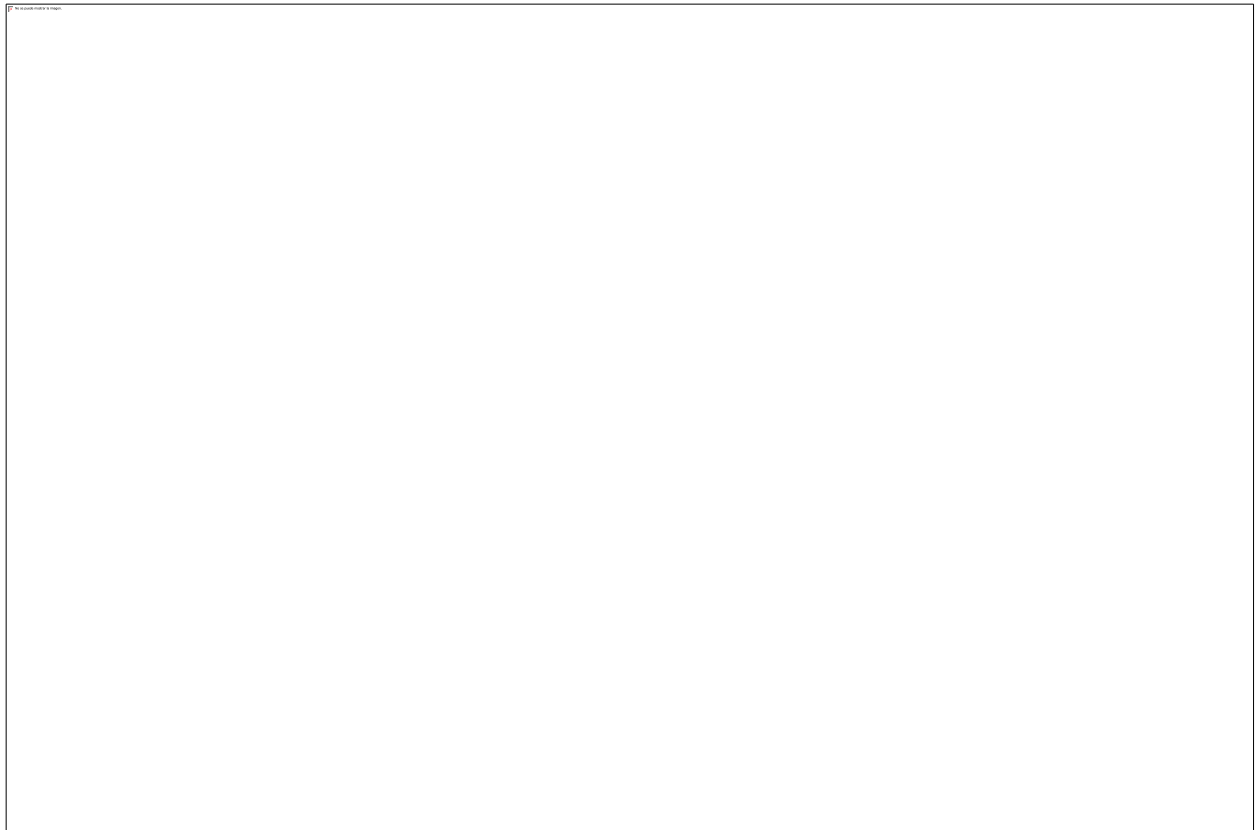
Elaborado por: Autoría Propia

Ilustración 57: Vista Admin - Logueado



Elaborado por: Autoría Propia

Ilustración 58: Vista Admin - Prestador



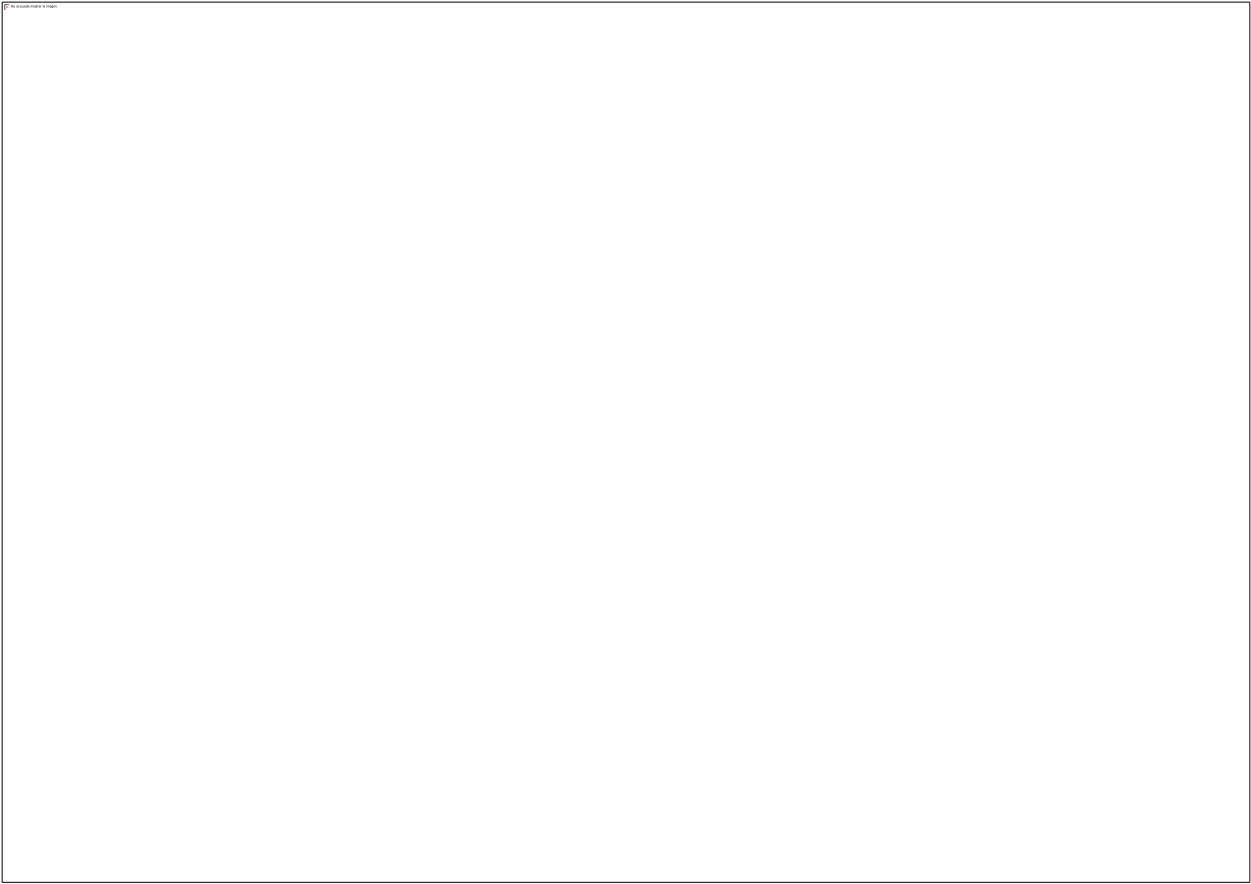
Elaborado por: Autoría Propia

Ilustración 59: Vista Admin - Sic



Elaborado por: Autoría Propia

Ilustración 60: Vista Admin - Acta Sic



Elaborado por: Autoría Propia

Ilustración 61: Vista Admin - Evaluación Sic

SEHAB

localhost:4200/sic/evaluacion

SEHAB Inicio SIC SP PAMEC RESOLUCIÓN 3100 USUARIOS PRESTADORES Samirjr

Sistema de Información para la Calidad

INDICADORES PARA EL MONITOREO DE LA CALIDAD/PRESTADORES DE SERVICIOS ACTA DE VERIFICACION Y SEGUIMIENTO

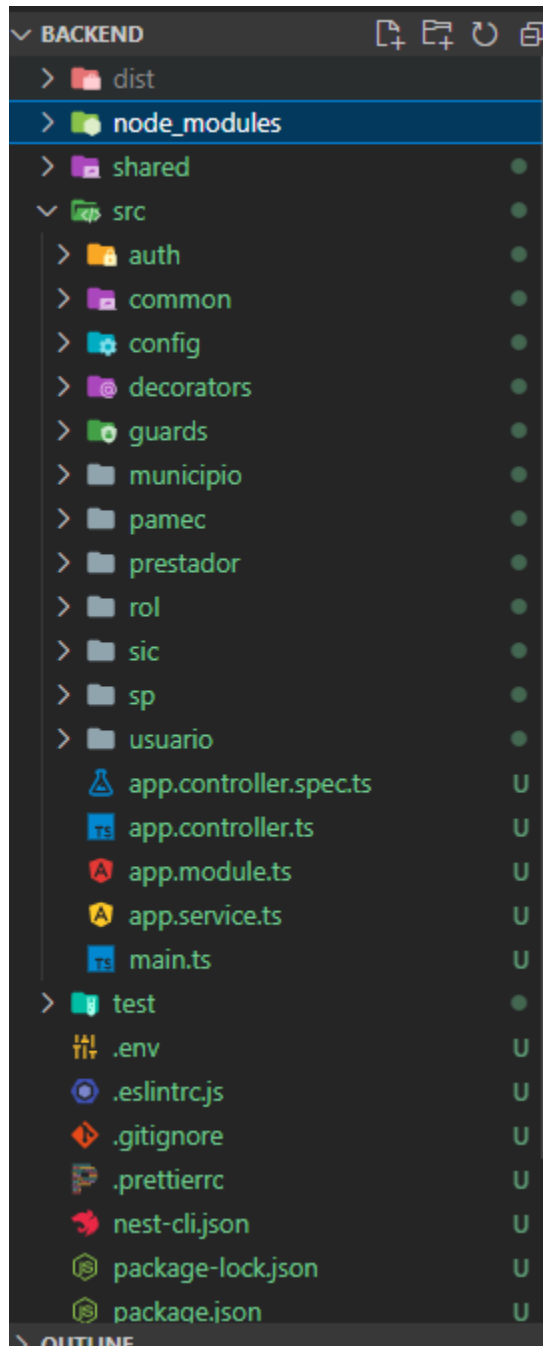
Nombre del Prestador a Evaluar

Nro.	Criterios	Cumplimiento	Observaciones
1	¿Existe en la entidad un funcionario o funcionarios responsables de la coordinación de los procesos del Sistema de Información para la calidad, con funciones claramente definidas?	Selecciona...	
2	¿Se ha hecho socialización de las normas que reglamentan el Sistema de Información para la calidad (Resolución 256 del 05 de febrero de 2016)	Selecciona...	
3	¿La IPS ha definido un equipo de trabajo y un espacio para el análisis de los indicadores del Sistema?	Selecciona...	
4	¿Consolidan la información de los indicadores como lo exige la resolución 256 de 2016?	Selecciona...	

Elaborado por: Autoría Propia

En las ilustraciones anteriores se puede observar fragmentos del desarrollo de la capa de presentación que demuestran el avance del proyecto.

Capa de Servicios



OUTLINE

1. CONCLUSIONES

El objetivo principal del trabajo de grado es diseñar una arquitectura de software en el proceso de evaluación y habilitación por parte de la secretaría de salud de la gobernación del putumayo a los prestadores de servicio de salud del departamento del putumayo, esta, contiene todos los lineamientos necesarios para la construcción de una aplicación totalmente funcional como lo es la arquitectura de software en capas.

8. La arquitectura de software planteada puede servir como un punto inicial para la elaboración e implementación de una aplicación web que les permita dar solución a la habilitación, evaluación y seguimiento de los prestadores de servicio de salud en el departamento del Putumayo
9. Con la construcción del prototipo se pone en funcionamiento algunas funcionalidades expuestas en la arquitectura de software en capas, este resultado permite comprobar que dicha arquitectura esta orientada a resolver de una manera rápida y eficiente las solicitudes requeridas por los usuarios.
10. Se concluye que al establecer este prototipo ayudará al desarrollador a integrar todas las funcionalidades establecidas por la Secretaría de Salud de la Gobernación del Putumayo de una manera menos compleja.

1. REFERENCIAS BIBLIOGRAFICAS

Blancarte Iturralde, O. (2020). Introducción a la arquitectura de software un enfoque práctico,
Primera Edición, Ciudad de México.

Martin, Bertucelli. (25 de noviembre 2019). Arquitectura de capas. Obtenido de
<https://sospnt.com/blog/118-arquitectura-de-capas>

RJ Code Advance. (2020). Patrones de diseño. Obtenido de
<https://rjcodeadvance.com/patrones-de-software-que-es-patron-de-diseno-parte-2/>

Mark Richards. Universidad Pedagógica y Tecnológica de Colombia. (2018). Comparación
Patrones De Arquitectura De Software. Obtenido de
<https://www.studocu.com/co/document/universidad-pedagogica-y-tecnologica-de-colombia/ingenieria-de-software-ii/comparacion-patrones-de-arquitectura-de-software/5107437>

Mariana Rolfo. (3 de agosto de 2022). NestJS Patrón. Obtenido de
<https://codigoencasa.com/nest-js-patron-arquitectonico-controladores-proveedores-y-modulos/>

Hmong. Arquitectura de software. Obtenido de
https://hmong.es/wiki/Software_architecture

Milena C, Ligia S, Jeronimo R. (2009). Integración de procesos utilizando la arquitectura
orientada a servicios. Obtenido de
<https://revistas.utp.edu.co/index.php/revistaciencia/article/view/3093>

Microsoft. Estilos de Arquitectura. Obtenido de
<https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices>

Miguel M. (2018). Manual del Estudiante Introducción al desarrollo de aplicaciones N-Capas

con tecnologías Microsoft. Primera Edición.