Presentación proyecto de grado

Sustentación arquitectónica PILAE

Juan C. Salazar

Facultad de ingeniería, Pregrado ingeniería de Sistemas, Universidad Católica de Oriente

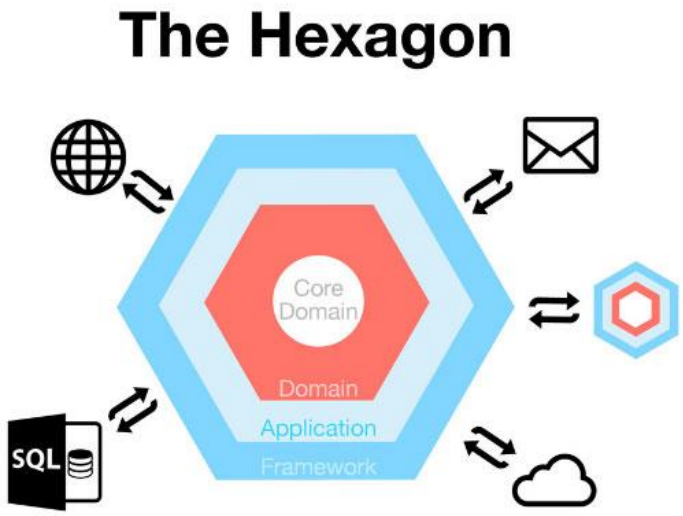
16 de noviembre de 2021

Contenido

Arquitectura propuesta

Para el proyecto de PILAE, se propuso realizar una arquitectura hexagonal

¿Qué es una arquitectura hexagonal?



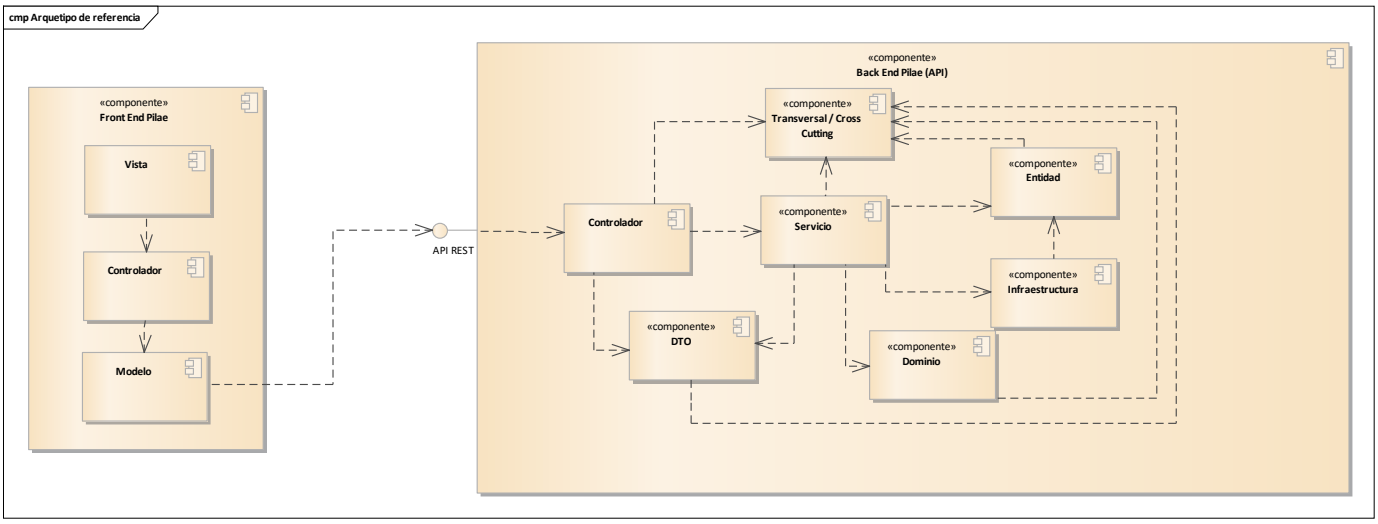
(Ferrer, 2016)

"La arquitectura hexagonal es un estilo de arquitectura de software que mueve el foco de un programador desde un plano más conceptual hacia la distinción entre el interior y el exterior del software. La parte interior son los casos prácticos y el modelo domain está construido sobre ello. La parte exterior es UI, base de datos, mensajería, etc. La conexión entre el interior y el exterior es mediante puertos, y su implementación equivalente se conocen como adaptadores. Por esta razón, este estilo de arquitectura hexagonal se conoce habitualmente como **Puertos y Adaptadores.**" (Novoseltseva, 2018)

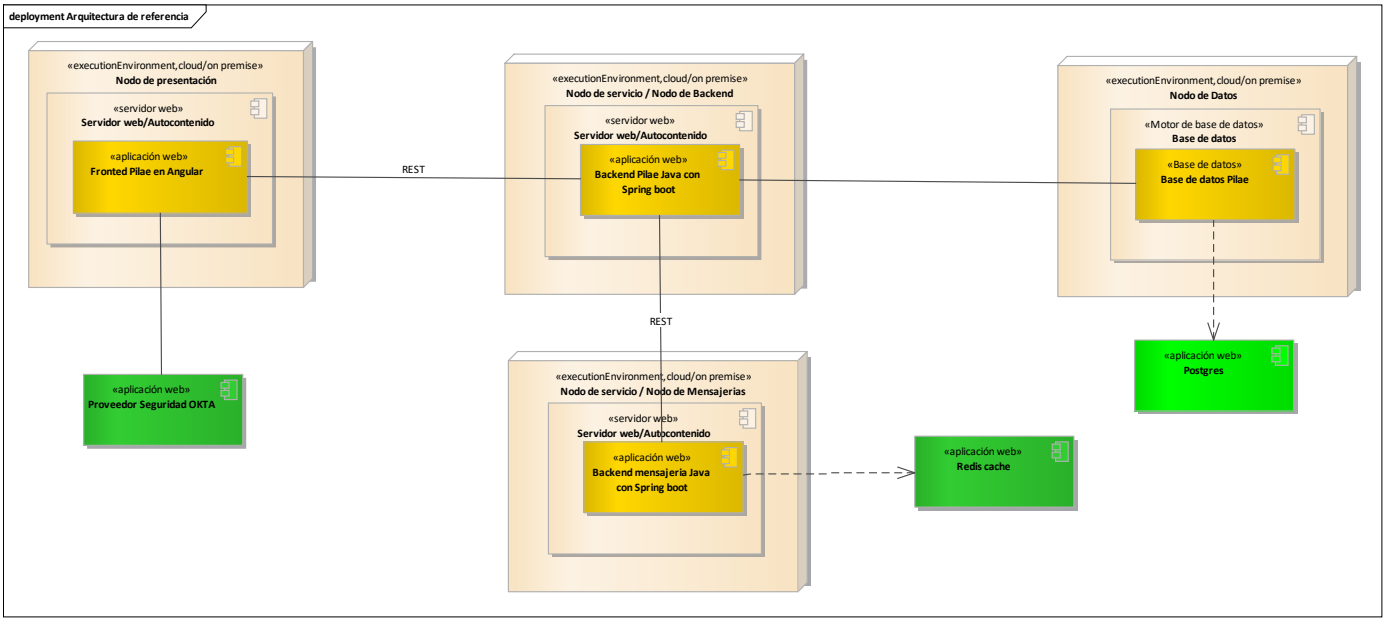
Como ya se pudo observar en la definición es una arquitectura que nace de la necesidad de que podamos separar de una manera limpia, nuestras aplicaciones, separándola así por medio de capas que definan su propia responsabilidad, así mismo podemos desacoplar las responsabilidades de nuestra aplicación y podamos escalarla y evolucionarla de una manera más fácil.

Arquetipo y arquitectura de referencia

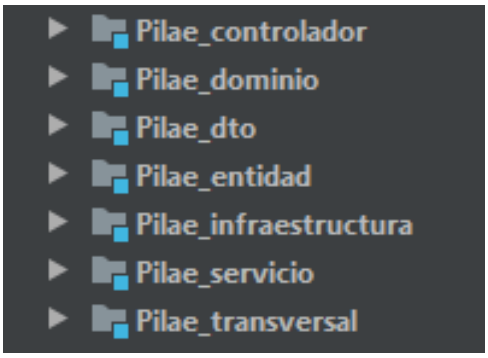
La arquitectura hexagonal en el proyecto PILAE está dividida por las capas de infraestructura, servicio, controlador y dominio. El arquetipo de referencia para el proyecto PILAE se puede observar en la siguiente imagen:



La arquitectura de referencia y los componentes a los que se conecta, que son la base de datos, el front y el API de mensajería se puede observar en la siguiente imagen:



Dentro de la aplicación de PILAE separamos las capas en modulos, cómo podemos ver en la siguiente imagen.



Cada modulo se encarga de segmentar las responsabilidades dentro de la aplicación, tenemos nuestras 6 capas de la aplicación:

-Controlador

-Dominio

-Dto

-Entidad

-Infraestructura

-Servicio

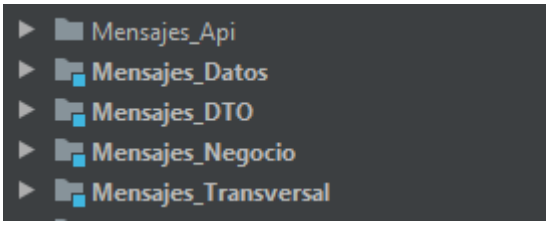
Tenemos la capa transversal, que se encarga de las operaciones que son transversales a todas las capas de la aplicación, como lo son las excepciones.

API de mensajería

Dentro de las buenas practicas de programación se encuentra el manejo correcto de los mensajes que retorna la aplicación (sea al usuario u otra aplicación).

Para el proyecto de Pilae también se creo una API de mensajería que se encarga de retornar los mensajes técnicos y para los usuarios, por medio de un catalogo definido en la aplicación, este catalogo contiene tanto los mensajes cuando una acción se realiza de forma exitosa, como cuando hay algún error o algún fallo dentro de la aplicación.

La API de mensajeria se divide en los siguientes componentes



Motivación de la arquitectura

Para muchas de las aplicaciones hoy en día, es un poco engorroso y resulta en una gran problemática la infiltración de toda la lógica de negocio en gran parte de la interfaz de usuario, esto puede acarrear varios problemas cómo:

-Gran acoplamiento dentro de las aplicaciones.

-Dificultad para la automatización de las pruebas.

-Dificultad en la integración con otras aplicaciones.

-Imposibilidad para pasar de un uso del sistema impulsado por humanos a un sistema de ejecución por lotes.

Sabemos que las arquitecturas limpias nos ayudan a solventar varios de los problemas que se viven hoy dentro del desarrollo de software, algunas de las características positivas es que las arquitecturas limpias son:

-Independientes del framework.

-Testables.

-Independientes de la UI

-Independientes de la base de datos.

-Independiente de agentes externos

-Más tolerantes al cambio.

-Reutilizables.

-Mantenibles. (Salguero, 2018)

Por ello la principal motivación para usar la arquitectura hexagonal dentro de la plataforma de PILAE, cómo ya lo hemos hablado es poder separar la aplicación en las diferentes capas para poder segmentar y desacoplar las distintas responsabilidades de la aplicación.

Gracias a que podemos desacoplar las diferentes responsabilidades y que es una arquitectura basada en la inyección de dependencias, podemos testear de manera fácil cada capa de la aplicación, realizando mocks a las distintas capas y a las respuestas de estas, en la siguiente imagen podemos ver un poco esto dentro de la aplicación de PILAE:

Se realiza un Mock de la capa de infraestructura

```
@Mock
DeporteRepositorioJpa deporteRepositorio;
```

Y se realiza una prueba de alguna de las funcionalidades de manera sencilla simplemente realizando un mockeo al llamado de la capa de infraestructura

```
@Test
public void debeRetornarDeporteDominio(){

    List<DeporteEntidad> deporteEntidadList = new ArrayList<>();
    deporteEntidadList.add(deporteEntidad);

    Mockito.when(deporteRepositorio.findAll()).thenReturn(deporteEntidadList);

    List<DeporteDominio> deporteDominios = deporteServicio.obtenerTodos();
    Mockito.verify(deporteRepositorio,Mockito.times( wantedNumberOfInvocations: 1)).findAll();
    Assert.assertFalse(deporteDominios.isEmpty());
}
```

De esta manera vemos lo sencillo que es realizar una prueba automatizada de la capa de servicio, debido a que está bien segmentadas las responsabilidades de la aplicación.

Otras de las motivaciones para aplicar la arquitectura hexagonal es poder realizar código limpio, esto se ve un poco en lo fácil que puede resultar aplicar varios de los principios de buenas prácticas de programación, cómo lo pueden ser los principios SOLID, KISS y DRY, esto debido al desacoplamiento que tenemos.

Principalmente y fundamentado en el hecho del desacoplamiento, tenemos beneficios directamente en forma de:

-Alta testabilidad (Aplicación del Principio de Inversión de Dependencias (DIP) de SOLID para la interacción del dominio con el resto de elementos)

-Alta tolerancia al cambio (Principio de Abierto/Cerrado (OCP) de SOLID derivado de la aplicación del DIP)

-Alta reutilización de código debido a la división estricta de responsabilidades a nivel de Application Services y Domain Services (Principio de Responsabilidad Única (SRP) de SOLID). (Ferrer, 2016).

Otro de las grandes motivadores a la hora de usar la arquitectura hexagonal es sin duda que podemos separar nuestra lógica de negocio con nuestros métodos de entrega de información, esto gracias a que podemos usar los adaptadores, de esta manera los datos de negocio y los métodos orientados al negocio queda totalmente segregado, pudiendo así centrarnos y crear las reglas de negocio de una manera más limpia y transparente.

Conclusión

Dentro de la arquitectura de software a lo largo de los años se ha visto la importancia de realizar código limpio que sea mantenible y escalable, esto ha llevado a crear e implementar arquitecturas limpias dentro de las aplicaciones, para la aplicación de PILAE, esto no es una excepción, queremos llevar esta plataforma a una expansión natural y que permita gestionar cada vez más tipos de deportes, queremos que se pueda mantener y pueda soportar el mayor número de usuarios posibles, para lograr este objetivo queremos implementar las mejores practicas de programación y las mejores tecnologías disponibles, así mismo las mejores y más limpias arquitecturas.

Bibliografía

Cockburn, A. (20 de Febrero de 2018). *alistair.cockburn.u*. Obtenido de https://alistair.cockburn.us/hexagonal-architecture/

Ferrer, J. (12 de Mayo de 2016). *Codely*. Obtenido de https://codely.tv/blog/screencasts/arquitectura-hexagonal-ddd/

Novoseltseva, E. (20 de Abril de 2018). *Apiumhub*. Obtenido de https://apiumhub.com/es/tech-blog-barcelona/arquitectura-hexagonal/

Salguero, E. (22 de Enero de 2018). *Medium*. Obtenido de https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f