Universidad Rey Juan Carlos

PRÁCTICA1: CAPTURA Y REPRESENTACIÓN DE DECISIONES DE DISEÑO

Diseño y Arquitectura Del Software

Grupo 5-Test

Adrián Muñoz Mora
Alberto Pacho Bernardos
Sergio Martín Muñoz
Yeray Cornejo Cantalapiedra
Álvaro Cano García
Israel Pozuelo Martín
Correo: i.pozuelo.2018@alumnos.urjc.es

PRÁCTICA1: CAPTURA Y REPRESENTACIÓN DE DECISIONES DE DISEÑO

Índice

1. Nombre de los integrantes para cada rol

- **Arquitectos Senior (ASS)**
  - Alberto Pacho Bernardos
  - Sergio Martín Muñoz
- **Arquitectos Cognitivos (ASC)**
  - Álvaro Cano García
  - Adrián Muñoz Mora
- **Arquitectos Junior (ASJ)**
  - Yeray Cornejo Cantalapiedra
  - Israel Pozuelo Martín

2. Informe sobre las capturas con ADMentor

ADMentor es una herramienta que da soporte para el lenguaje del modelado orientado a las decisiones de arquitectura. Te da la opción de recoger los posibles problemas de diseño en tu arquitectura y poder exportarlo para usar en otros proyectos. A partir de unos problemas de diseño se puede modelizar a través de ADMentor.

Esta herramienta ha sido ejecutada sobre Sparx Enterprise Architect, por lo que nos hemos tenido que informar de cómo utilizar esta herramienta mediante tutoriales y documentos facilitados por el profesor. El formado MADR nos ha permitido describir el estado y la evolución de las decisiones. A continuación, mostraremos una serie de pasos necesarios para el uso de esta herramienta:

- Primero, arriba a la izquierda hay un botón con el logo del software descargado donde haciendo ‘click’ se abrirá un panel con diferentes opciones, si seleccionamos “New Project” nos permitirá crear un proyecto nuevo. Además, este panel permite guardar, abrir, recargar o imprimir un proyecto dado.

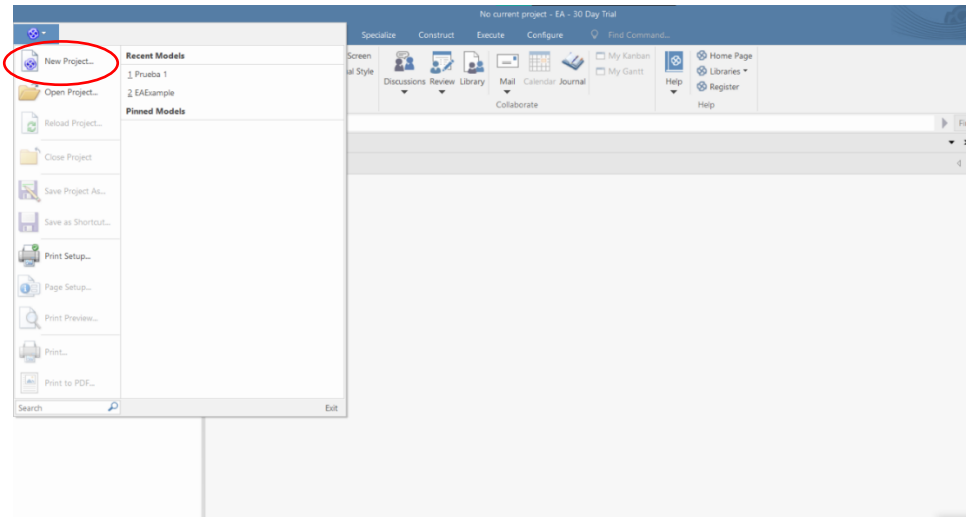

Ilustración 1 Captura Uso ADMentor 1

- Segundo, cuando ya tengamos nuestro proyecto abierto, tendremos que pinchar con el botón derecho del ratón donde pone “Model”, y seleccionar “Add a Model using Wizard”. En el panel que se nos abre tendremos una lista llamada ADMentor donde aparecen 3 opciones, nosotros nos centraremos en los dos primeros, si los seleccionamos nos abrirán los diagramas donde representar los casos de uso.

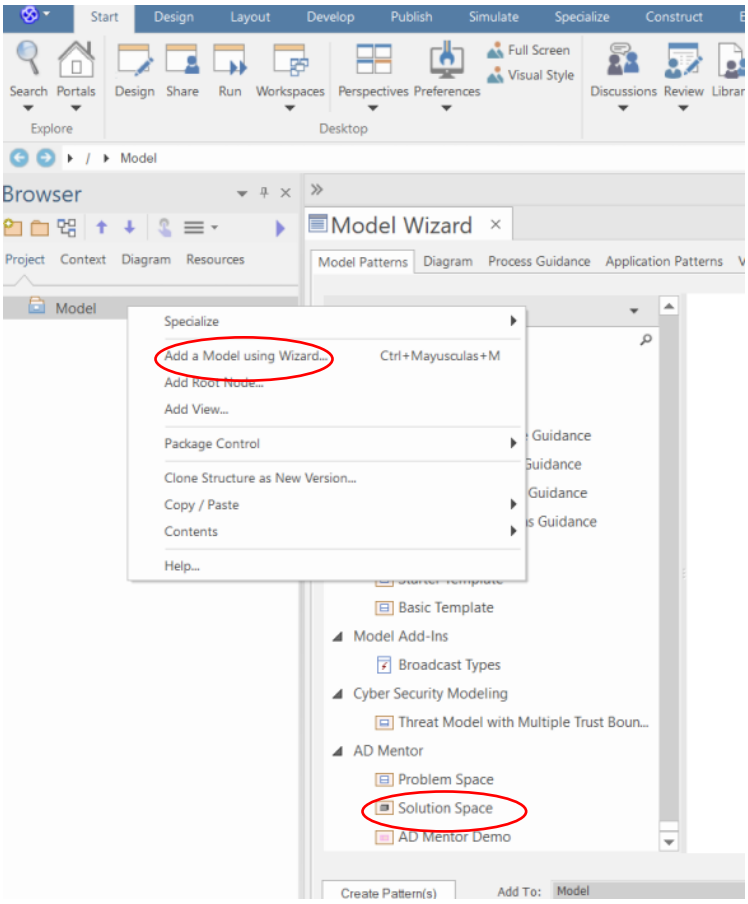

Ilustración 2 Captura Uso ADMentor 2

- Por último, se explicará los elementos que componen un diagrama, como se relacionan y sus atributos.
  En primer lugar, podremos modificar las propiedades de los elementos haciendo 'click' 2 veces con el botón izquierdo del ratón.
  Los diagramas están compuestos por 2 elementos principales relacionados entre sí, el problema de diseño y la opción para resolverlo. Estos dos componentes se encuentran en la Toolbox a la izquierda de la aplicación.

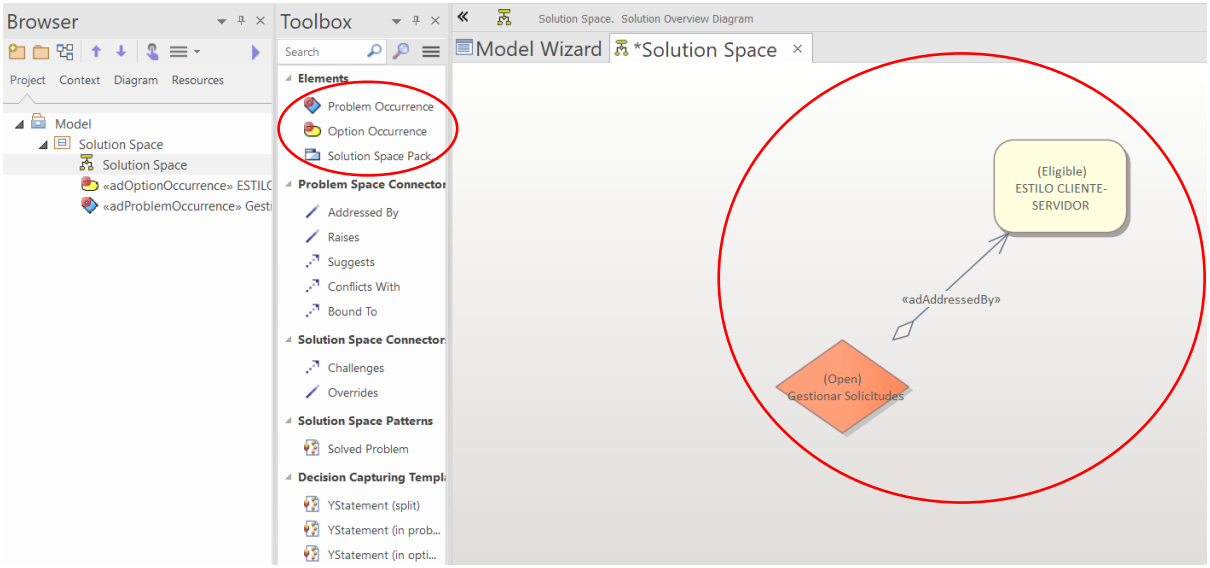

Ilustración 3 Captura Uso ADMentor 3

3. Descripción de los resultados

Semana 1

En esta primera semana nos reunimos para capturar los requisitos de la aplicación a la vez que los ASJ se inician en el uso del formato MADR y el uso de ADMentor descargando el software pertinente. Se pueden observar los requisitos en el anexo 8.

Semana 2.

Decisiones de estilo

Tras analizar el problema en profundidad los arquitectos senior (ASS) han tomado la decisión de basarse en un estilo de Cliente-Servidor. Ya que el cliente indica que quiere que los usuarios que utilicen la aplicación tengan una interfaz propia, una lógica de negocio y lógica de acceso de base de datos. Ya que la arquitectura Cliente-Servidor se divide en 3 capas. Siendo estos los puntos 3.2, 3.3 y 3.4 propios de la tabla de requisitos realizada la semana 1.

Documento de riesgo

Tras una reunión entre los ASS y los ASC, se ha discutido si sería mejor implementar una arquitectura por Capas. Pero finalmente hemos decidido la arquitectura Cliente-Servidor ya que aporta la división por capas.

Por otro lado, hemos descartado las siguientes arquitecturas:

El modelo vista controlador ya que es un estilo iterativo y no se va a adecuar lo suficiente a nuestro problema, aunque en un primer momento hemos dudado sobre si sería el adecuado.

El modelo de tuberías y filtros ya que este realiza una función completamente distinta a lo que nuestro cliente necesita por eso lo hemos descartado instantáneamente.

El de eventos ya que nuestro problema no valora en ningún momento el uso de eventos ni la interacción con ellos.

Decisiones de diseño:

En la reunión del día 18 de noviembre, los arquitectos senior (ASS) tomaron la decisión de implementar los siguientes patrones de diseño:

- Facade provee de una interfaz única, simple para acceder a una interfaz o grupo de interfaces de un sistema. Este patrón de diseño hace referencia al requisito RF-3.2 y al RF-2 ya que será necesaria una interfaz de usuarios y el consumo de servicios en remoto.

Semana 3

Decisiones de diseño:

En la reunión del día 24 de noviembre, los arquitectos senior (ASS) tomaron la decisión de implementar los siguientes patrones de diseño:

Proxy ya que proporciona un intermediario de un objeto para controlar su uso. En este caso, este patrón hace referencia al requisito RF-3.4 porque se necesita acceder a una base de datos y las páginas más frecuentemente visitadas.

- Singleton garantiza la existencia de una única instancia para una clase. Este patrón se puede aplicar al requisito RF-5.3 y RF-5.4 ya que se necesitará una cesta de la compra y un solo sistema de identificación de usuarios.

- Parallel split es un punto de flujo donde un thread individual se divide en varios threads que pueden ejecutarse en paralelo permitiendo la ejecución simultanea de varias actividades. En nuestro caso, para comprobar si hay stock de un producto a la hora de realizar un pago. Este patrón de diseño hace referencia al requisito RF-5.2 donde se procesan los diferentes pedidos para no comprar un producto agotado.

Documento De Riesgo

En la reunión del día 25 de noviembre los ASC reflexionaron con los ASS sobre los problemas que estas decisiones podían ocasionar, en un principio los ASS al ver los patrones de diseño aprendidos pensaron en un patrón builder pero rápidamente los ASC rechazaron esta propuesto debido a que el builder no tenía sentido ahí y se propuso utilizar el singleton. Las demás decisiones fueran aprobadas sin ningún problema por los ASC

Semana 4

Decisiones de diseño:

En la reunión del día 1 de diciembre, los arquitectos senior (ASS) tomaron la decisión de implementar los siguientes patrones de diseño:

- Access Token: Permite el control de acceso a la aplicación, detecta a los usuarios que intentan acceder a los recursos de nuestra aplicación, este patrón hace referencia al requisito 5.3
- API Gateway: Nos permite enrutar las peticiones desde el exterior hacia cada microservicio, este patrón hace referencia al requisito 6 y 6.1.
- Observer: Hace referencia al requisito 6.2, ya que una actualización produce un cambio de estado y este ha de notificarse.

Documento De Riesgo

Tras tomar las decisiones los ASS los ASC se reunieron para discutir sobre las decisiones tomadas y se dieron cuenta que faltaba algo que hiciese referencia al requisito 6, más en concreto al requisito 6.2, ya que necesitamos un bus de eventos y para ello necesitamos un estilo por eventos, que se incluye dentro del estilo principal de nuestra arquitectura, que es cliente servidor. Después de plantear estos problemas a los ASS, estos volvieron a reunirse y tomaron la decisión de que debía de estar implementado el estilo por eventos. Por otra parte, los patrones anteriores no generaron problema y fueron rápidamente aceptados. De esta forma añadimos una nueva decisión de diseño:

- Estilo por eventos: Los sistemas EDA monitorizan los datos del entorno y analizan los eventos para proporcionar la respuesta más adecuada. Una arquitectura EDA tiene emisores de eventos, consumidores y canales para transmitir los eventos. Todo esto hace referencia al requisito 6.2 para tener un bus de eventos y el 3.1 ya que el estilo por eventos necesita que los eventos se transmitan entre sistema poco acoplados mediante mensajes.

Semana 5

Documento de riesgo:

Después de la semana 4 los ASC se han reunido para terminar de extraer las conclusiones y han llegado a la conclusión de que existen ciertos errores de diseño que antes no se dieron cuenta. Estos son el uso del singleton y el pararell Split. El primero ha sido finalmente rechazado debido a que se consideró que no eran una opción viable al tener cada usuario una cesta de la compra. El segundo fue simplemente porque la funcionalidad propuesta tras estudiarla exhaustivamente no era la adecuada. Esto fue transmitido a los ASS y tras una reflexión fue cambiado y los ASJ lo rechazaron el MADR.

Diagrama De Clases

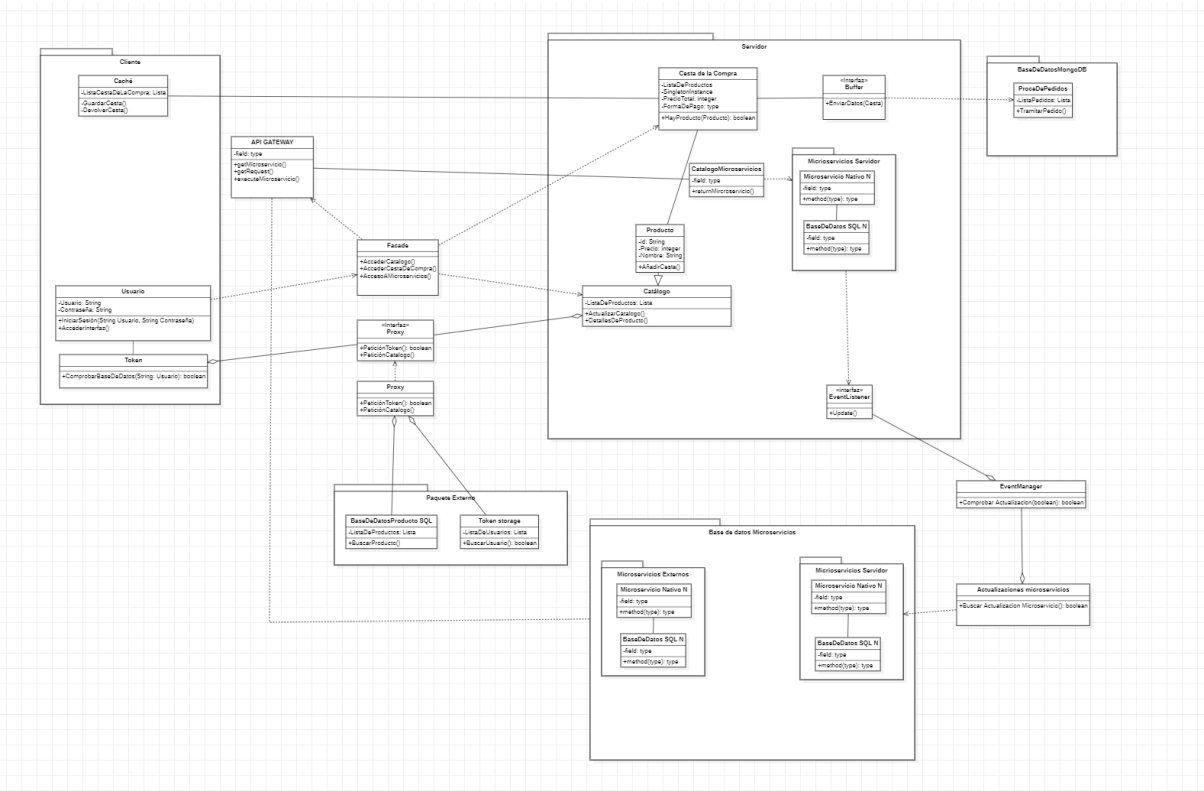

Ilustración 4 Diagrama De Clases

El UML está dividido en dos paquetes principales, el paquete cliente y el paquete Servidor, y tres paquetes de bases de datos diferentes.

En el paquete cliente se puede encontrar el usuario y la caché del propio usuario. El usuario tiene comunicación con el proxy para verificar la identificación de este mediante tokens. Además, el usuario tiene comunicación con la clase Facade ya que es una interfaz única y da acceso a las diferentes funcionalidades de la aplicación web.

En el paquete cliente se encuentra el catálogo con todos los productos disponibles en la tienda y un catálogo con los diferentes microservicios. Además, se puede encontrar la cesta

de la compra para poder ir añadiendo productos y poder tramitar el pedido mediante la clase buffer que está conectada a la base de datos MongoDB.

En el paquete externo, se encuentran la base de datos privada del propio servidor, en ella se encuentran los productos y los usuarios.

En el paquete de Base de Datos de Microservicios se encuentran todos los microservicios, tanto internos de la aplicación como externos. Este paquete está conectado a una serie de clases que detectan cuando un microservicio se ha actualizado. Las actualizaciones se detectan mediante el patrón observer.

Diagrama De Despliegue

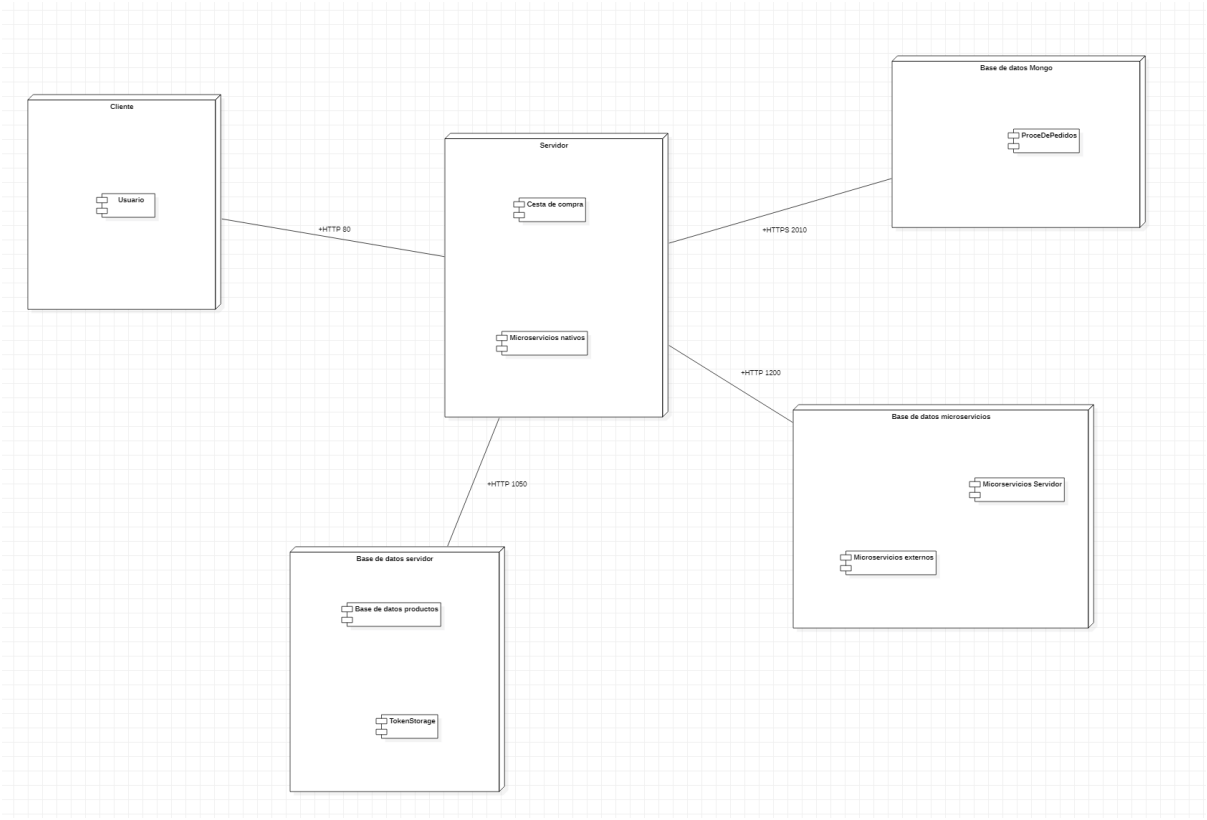

Ilustración 5 Diagrama De Despliegue

Este tipo de diagrama sirve para poder entender la arquitectura física del sistema. Contamos con un nodo para el cliente, otro para el servidor y otros para las diferentes bases de datos.

4. Decisiones tomadas en cada iteración

Los ASJ han modelado las decisiones de diseño con la herramienta ADMentor y el formato MADR.

```
# [Decisión 001]: Estilo Arquitectónico

 * Estado [aceptado]
 * Decisores: [Alberto Pacho-Sergio Martín]
 * Fecha: [2020-11- 18]

## Contexto y declaración del problema

El cliente necesita cambiar la arquitectura de su página web. Además, de solicitar diversos componentes de la aplicación, como serían el canal de mensajería, componentes de presentación, lógica de dominio y lógica de base de datos.

## Impulsores de decisión

RF1-RF3-RF6-RF5

## Opciones consideradas

 * Estilo por Capas.
 * Estilo Cliente-Servidor.
 * Estilo Rest.

## Resultado de la decisión

Opción elegida: Cliente-Servidor, porque tras analizar fríamente el problema propuesto, hemos llegado a la conclusión del que el cliente necesita basarse en una arquitectura Cliente-Servidor ya que el estilo se puede dividir por diferentes capas donde se pueden aplicar las necesidades. En este caso la lógica de negocio, la lógica de base de datos y el canal de mensajería.

### Consecuencias positivas
 * Facilidad de integración en páginas web.
 * Permite acceder a diferentes bases de datos de una forma dinámica.

### Consecuencias negativas
 No se han encontrado consecuencias negativas.
```

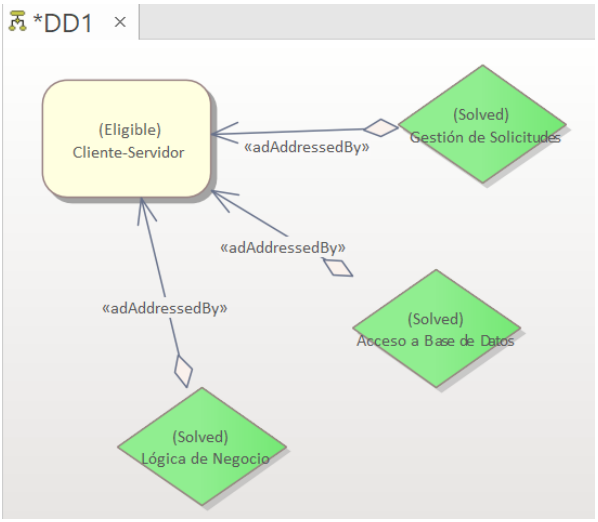

Ilustración 6 Captura DD1

[Decisión 002]: Interfaz única

 * Estado [aceptado]
 * Decisores: [Alberto Pacho-Sergio Martín]
 * Fecha: [2020-11- 18]

Contexto y declaración del problema

Se requiere una interfaz única tanto para los usuarios web como los móviles. Además de que esta interfaz comunique con los distintos microservicios existentes.

Impulsores de decisión

RF3.2 RF2

Opciones consideradas

 * Patrón Facade.

Resultado de la decisión

Opción elegida: “FACADE”, ya que provee de una interfaz única simple para un sistema complejo.

Consecuencias positivas
 * Independencia, portabilidad y reutilización.
 * Reducción de dependencias entre subsistemas y los clientes.
 * A la hora de modificar las clases de los subsistemas basta con realizar cambios en la interfaz externa para que los clientes puedan quedar aislados.

Consecuencias negativas
* Si el acceso de clientes es masivo, podían acabar utilizando solo una parte de la interfaz externa.

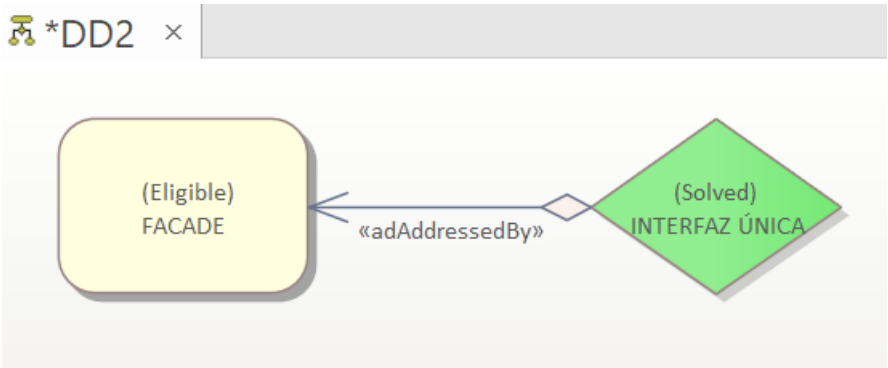

Ilustración 7 Captura DD2

[Decisión 003]: Una Única Instancia

 * Estado [rechazado]
 * Decisores: [Alberto Pacho-Sergio Martín]
 * Fecha: [2020-11- 18]

Contexto y declaración del problema

Se tiene la necesidad de crear una única cesta, dado que solo puede existir una compartida para todos los productos, para que el usuario pueda ir añadiéndolos. Además, se requiere que solo pueda haber un único usuario en la sesión iniciada.

Impulsores de decisión

RF-5.4 RF-5.3

Opciones consideradas

 * Patrón Builder.
 * Patrón Sigleton.

Resultado de la decisión
Opción elegida: “SIGLETON”, este patrón garantiza la creación de una única instancia para una clase.

Consecuencias positivas

Al utilizar este patrón, nos aseguramos de que no haya instancias duplicadas.

Consecuencias negativas

Dificultad para realizar testing en el programa.

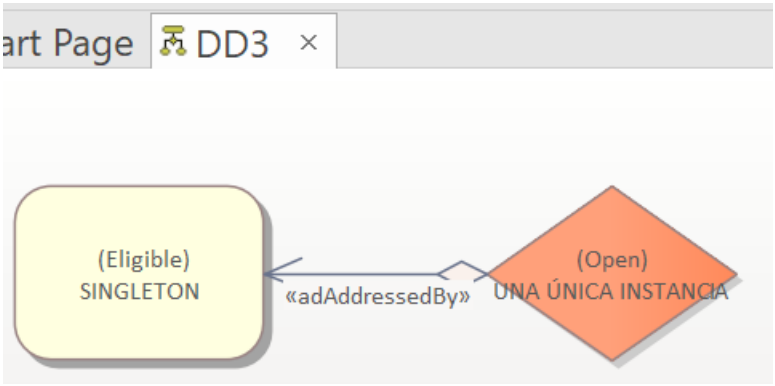

Ilustración 8 Captura DD3

[Decisión 004]: Proxy
 * Estado [aceptado]
 * Decisores: [Alberto Pacho-Sergio Martín]
 * Fecha: [2020-11- 18]

Contexto y declaración del problema

Se requiere cierto control de uso y comunicación con la base de datos específica del servidor.

Impulsores de decisión

RF-3.4

Opciones consideradas

* Proxy

Resultado de la decisión

Opción elegida: El patrón "PROXY". Este patrón proporciona un intermediario de un objeto para controlar su uso. Y de esta forma una clase funciona como una interfaz de otro elemento.

Consecuencias positivas

Puedes controlar el objeto de servicio sin que los clientes lo sepan.
Puedes gestionar el ciclo de vida del objeto de servicio cuando a los clientes no les importa.

Consecuencias negativas

El código puede complicarse ya que debes introducir gran cantidad de clases nuevas.
La respuesta del servicio puede retrasarse.

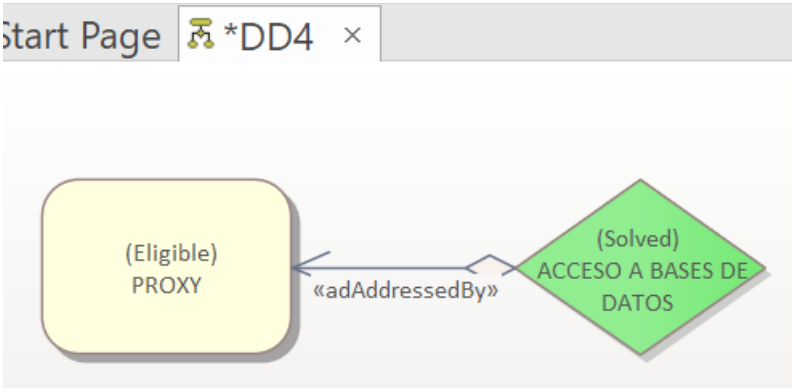

Ilustración 9 Captura DD4

[Decisión 005]: Parallel Split
 * Estado [rechazado]
 * Decisores: [Alberto Pacho-Sergio Martín]
 * Fecha: [2020-11- 18]

Contexto y declaración del problema
Se necesita saber si hay stock disponible de un producto a la hora de realizar un pedido.

Impulsores de decisión

 RF-5.2

Opciones consideradas

 * Parallel Split

Resultado de la decisión

Opción elegida: patrón “PARALLEL SPLIT”. Este patrón se divide en varios threads que pueden ejecutarse en paralelo permitiendo la ejecución simultanea de varias actividades.

Consecuencias positivas
Permite realizar comprobaciones en tiempo real para asegurarse de que haya stock de un producto a la hora de realizar un pedido.

Consecuencias negativas

Limita el número de hilos del procesador, evitando que otras tareas puedan utilizarlos.

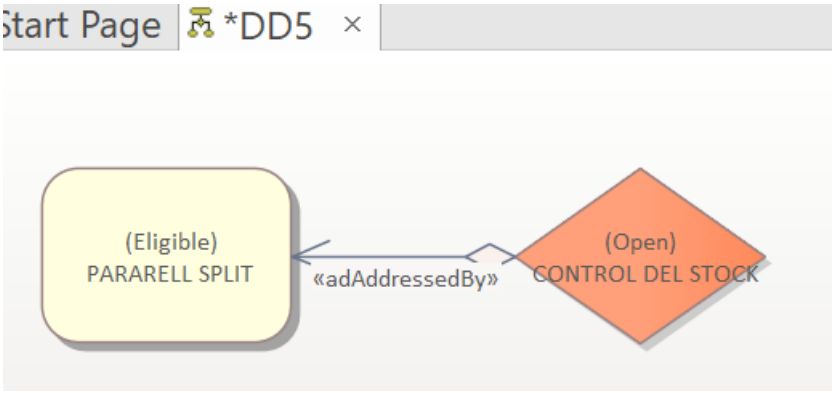

Ilustración 10 Captura DD5

[Decisión 006]: Eventos
 * Estado [aceptado]
 * Decisores: [Alberto Pacho-Sergio Martín]
 * Fecha: [2020-12- 01]
Contexto y declaración del problema

Se requiere implementar una arquitectura basada en diferentes aspectos o cambios de la aplicación, la cual reaccionará de una forma u otra dependiendo de los eventos entrantes.
Impulsores de decisión

 RF-6.1 - RF.6.2

Opciones consideradas

 * Estilo por Eventos

 * Estilo MVC

Resultado de la decisión
Opción elegida: estilo "EVENTOS". Debido principalmente a dos características que posee el estilo por eventos:
* Ya que un evento representa un cambio significativo en el estado de un
Sistema.
* Los eventos se transmiten entre sistema poco acoplados mediante mensajes.

Consecuencias positivas
* Simplicidad.
* Modularidad: una sola modalidad para eventos diversos.

* Puede mejorar la eficiencia.

Consecuencias negativas

* Posibilidad de que se desborde el sistema.
* Potencial imprevisión de escalabilidad.
* No hay mucho soporte de recuperación en caso de que haya un error.

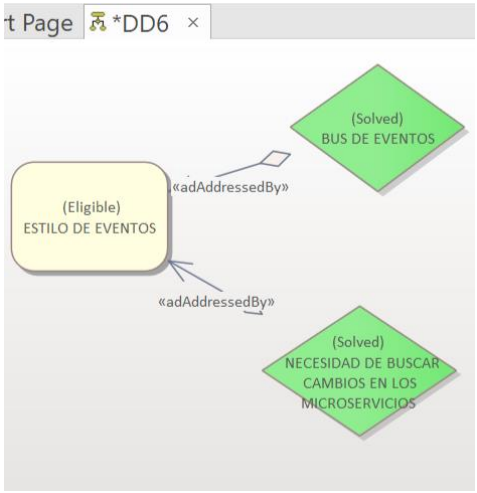

Ilustración 11 Captura DD6

[Decisión 007]: Comunicación entre microservicios y cliente

* Estado [aceptado]
* Decisores: [Alberto Pacho-Sergio Martín]
* Fecha: [2020-12- 02]

Contexto y declaración del problema

Comunicación de cliente a microservicio de HTTP a través de un Gateway que contiene diversas APIs. Tanto para los microservicios nativos como las externas.

Impulsores de decisión

RF-6.1 - RF-4 - RF-5.1 - RF-7

Opciones consideradas

* Api Gateway

Resultado de la decisión

Opción elegida: patrón "API GATEWAY". Este patrón es un sistema intermediario que proporciona una interfaz API REST para hacer de enrutador desde un único punto de entrada, el API Gateway, hacia un grupo de microservicios. Interactúa como puerta de enlace "Gateway".

Consecuencias positivas

Cuenta con diversos puntos positivos. Seguridad, análisis y supervisión, monitorización de uso, administración de ciclo de vida, y traducción de protocolos.

Consecuencias negativas

Limita el número de hilos del procesador, evitando que otras tareas puedan utilizarlos.

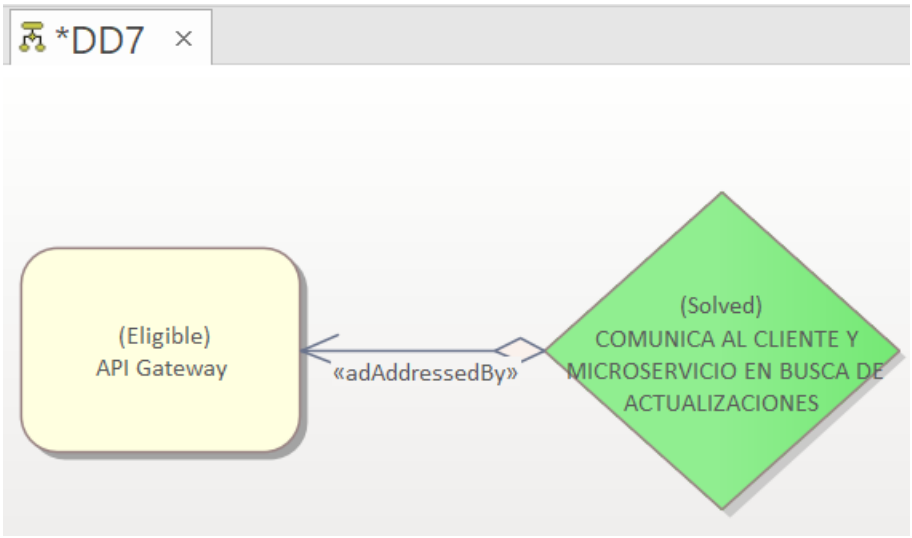

Ilustración 12 Captura DD7

[Decisión 008]: Identificación de Usuarios
 * Estado [aceptado]
 * Decisores: [Alberto Pacho-Sergio Martín]
 * Fecha: [2020-12-02]
Contexto y declaración del problema

Se tiene la necesidad de identificar al usuario que quiere iniciar sesión en la tienda

Impulsores de decisión

 RF-5.3

Opciones consideradas

 * Access Token

Resultado de la decisión

Opción elegida: Patrón "Access Token", que nos permitirá identificarnos con nuestra cuenta de usuario en el servicio web, resolviendo así nuestra necesidad descrita anteriormente

Consecuencias positivas

Tenemos control de acceso de usuarios en nuestra tienda web, así como aportar una mayor seguridad en el acceso y registro de los usuarios de nuestra tienda mediante el uso de tokens

Consecuencias negativas

El usar tokens en lugar de enviar el usuario y contraseña en cada petición del servidor puede ser algo más complejo, pero como hemos comentado antes nos aporta mucha más seguridad

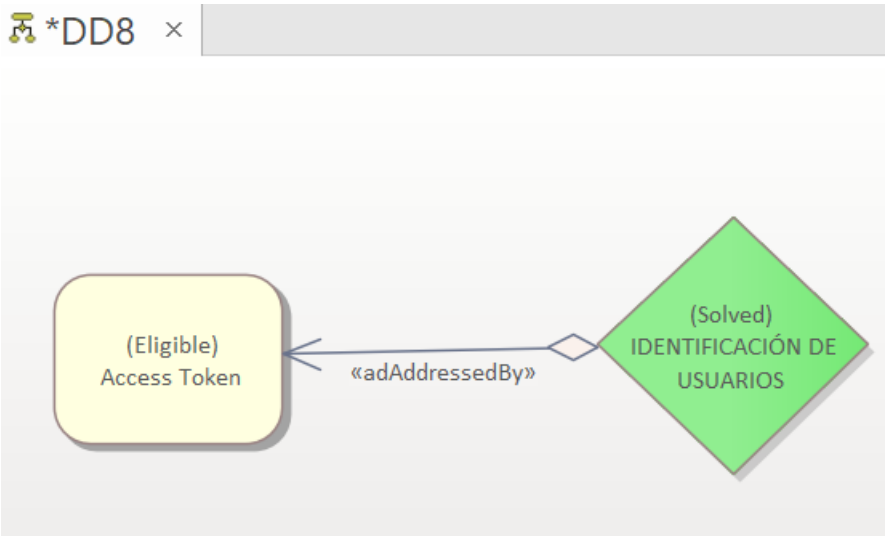

Ilustración 13 Captura DD8

[Decisión 009]: Notificación y cambios de actualizaciones

* Estado [aceptado]
* Decisores: [Alberto Pacho-Sergio Martín]
* Fecha: [2020-12-02]

Contexto y declaración del problema

Se requiere comunicación asincrónica basada en eventos y propaga las actualizaciones entre microservicios nativos del sistema.

Impulsores de decisión

RF-6.2

Opciones consideradas

* Observer

Resultado de la decisión

* Opción elegida: Patrón "OBSERVER". Este patrón nos permite definir una dependencia entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes y aplicara dicho cambio.

Consecuencias positivas

* Puedes establecer relaciones entre objetos durante el tiempo de ejecución.

Consecuencias negativas

* Los suscriptores son notificados en un orden aleatorio.

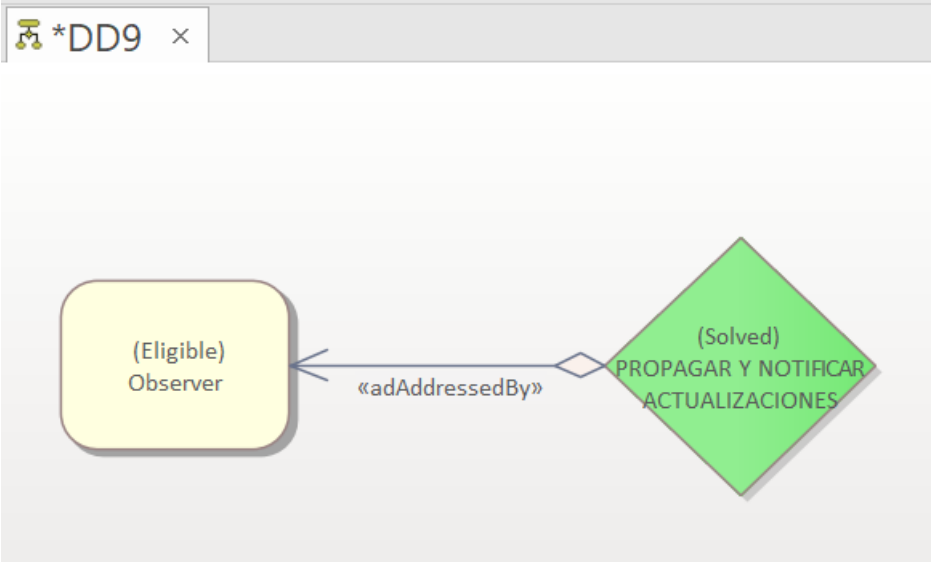

Ilustración 14 Captura DD9

5. Conclusiones

Después de valorar individualmente el trabajo realizado, hemos llegado a la conclusión conjunta del esfuerzo que nos ha supuesto esta práctica, dados los escasos conocimientos que teníamos sobre esta.
Una vez puestos en contexto hemos conseguido alcanzar los objetivos a través de dedicación y búsqueda de información para entender en profundidad el contenido.
A pesar de ciertos problemas a la hora de elegir el diseño de arquitectura hemos podido observar la importancia de estructurar un proyecto según su arquitectura.

6. Bibliografía

Además de los diferentes apuntes en PDF que se nos han facilitado en esta asignatura, también nos hemos documentado en diferentes páginas web:

- https://refactoring.guru/es/design-patterns/catalog
- https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/diagrama-de-despliegue/
- https://microservices.io/patterns/apigateway.html
- https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/api-gateway
- https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/access-token
- https://reactiveprogramming.io/blog/es/patrones-de-diseno/facade

7. Anexo de tiempos

SEMANA	ITERACION	TIEMPO ASS	REFLEXION ASS-ASC	REFELXION ASS	TIEMPO ASJ
1	1	50	20	15	35
1	2	35	15	15	30
2	1	45	20	10	35
2	2	40	15	15	25
3	1	50	20	35	40
3	2	20	10	15	30
4	1	40	20	25	20
4	2	25	30	20	20
5	1	15	15	15	30
5	2	30	10	20	40

Tabla 1 Tiempos DD

8. Anexo de requisitos

Requisitos ID	Nombre	Descripción
RF-01	Gestionar Solicitudes	Gestionar las solicitudes mediante una lógica de negocio y base de datos.
RF-1.1	Peticiones de Solicitudes	Las peticiones se devuelven en HTML y JSON.
RF-02	Ajuste de Pantallas	La aplicación se deberá ajustar automáticamente en los diferentes dispositivos.
RF-03	Componentes de la aplicación	La aplicación necesita tener cuatro componentes básicos para su correcto funcionamiento.
RF-3.1	Catálogo de productos	La aplicación requiere un catálogo que se actualice con los diversos productos de la tienda.
RF-3.2	Componentes de Presentación	Permiten el control de la interfaz de usuario y el consumo de servicios remotos.
RF-3.3	Lógica de dominio	Se trata de la lógica de dominio de la aplicación.
RF-3.4	Acceso a Base De Datos	Permite el acceso a distintos tipos de BBDD, como SQL y NoSQL.
RF-04	Enlazar Microservicios Externos	Se enlazarán microservicios externos de forma asíncrona para reforzar la fiabilidad de los microservicios.
RF-05	Funcionalidades en microservicios independientes	Se desarrolla e implementa cada microservicio de forma independiente sin que afecte a otros subsistemas. Se soportarán en bases de datos SQL distintas, a excepción de la cesta que utiliza una base de datos NoSQL. La coherencia entre las bases de datos de los diferentes microservicios se logra mediante eventos de integración de nivel de aplicación.

RF-5.1	Catálogo de Microservicios	Es una lista de los diferentes microservicios disponibles que se pueden usar.
RF-5.2	Procesado de Pedidos	La aplicación debe poder procesar los pedidos que realice el usuario en la tienda.
RF-5.3	Identificación de Usuarios	La aplicación es capaz de identificar el usuario que ha iniciado sesión con su cuenta de la tienda.
RF-5.4	Cesta de Compra	Se utilizará una caché para almacenar la cesta de compra con los productos que desea comprar y la localización de microservicios de terceros, en una base de datos MongoDB.
RF-5.4.1	Caché de la Cesta de Compra	Se requiere guardar en caché la cesta de la compra de cada cliente.
RF-06	Comunicación entre los clientes y los microservicios	El cliente desea comunicarse con el microservicio y lo puede hacer a través de la puerta de enlace de una API o también con el microservicio MVC, que se comunica con otros microservicios mediante la puerta de enlace de API. Se podrá usar contenedores de microservicios dentro de un host o mediante un clúster. Se utilizarán dos tipos de comunicación.
RF-6.1	Consultar Actualizaciones	Comunicación de cliente a microservicio de HTTP a través de un Gateway que contiene diversas APIs Se debe poder consultar si es necesario que la aplicación se actualice.
RF-6.2	Bus de eventos	Comunicación asincrónica basada en eventos. Propaga las actualizaciones entre microservicios y se puede implementar agentes de mensajería como RabbitMQ, Service Bus (Azure Service Bus, NServiceBus, MassTransit...)

RF-7	Microservicios nativos	Se requiere una serie de microservicios nativos que nuestro servidor ya cuente con ellos. De esta forma dotara de más escalabilidad a nuestro sistema.

Tabla 2 Requisitos