Universidad Rey Juan Carlos – Diseño y Arquitectura Software (DAS)

**Práctica1: Captura y Representación de Decisiones de Diseño**

Curso 2020-2021

GRUPO 4 (GRUPO DE TEST)

Azahara Andújar Muñoz-Quirós

Carlota Menéndez del Campo

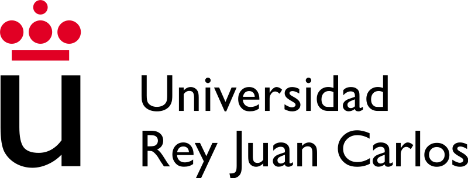
Javier Méndez García-Brioles

Alberto Jiménez Gómez

David Mestanza Rubia

Manuel Martín Aláez

**Contacto**: [a.andujar.2017@alumnos.urjc.es](mailto:a.andujar.2017@alumnos.urjc.es) (Representante)



## **ÍNDICE**

Contenido

[**ÍNDICE** 2](#_Toc58427688)

[**ROLES** 3](#_Toc58427689)

[**INFORME SOBRE EL USO DE ADMENTOR** 4](#_Toc58427690)

[5](#_Toc58427691)

[**1- ANÁLISIS DE REQUISITOS:** 6](#_Toc58427692)

[**2- RESULTADOS** 9](#_Toc58427693)

[Iteración 1 9](#_Toc58427694)

[Iteración 2 10](#_Toc58427695)

[Iteración 3 11](#_Toc58427696)

[Iteración 4 12](#_Toc58427697)

[Iteración 5 13](#_Toc58427698)

[Iteración 6 14](#_Toc58427699)

[**3 - CONCLUSIONES** 15](#_Toc58427700)

[**4 - BIBLIOGRAFÍA** 16](#_Toc58427701)

[**5- ANEXO: TABLA DE TIEMPOS DE TRABAJO** 17](#_Toc58427702)

## **ROLES**

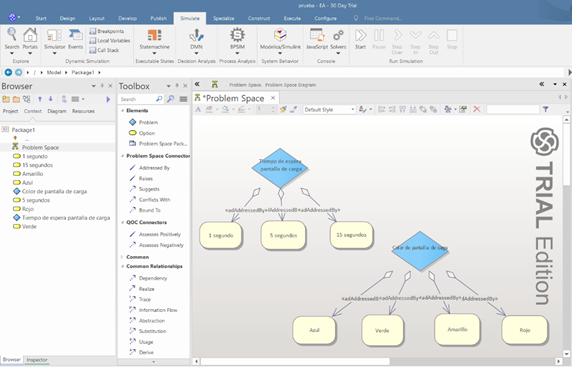
Tabla 1: Roles de los participantes

|  |  |
| --- | --- |
| **Arquitectos Software Senior (ASS)** | Alberto Jiménez Gómez y Manuel Martín Aláez |
| **Arquitectos Software Junior (ASJ)** | Azahara Andújar Muñoz-Quirós y Carlota Menéndez del Campo |
| **Arquitectos Software Cognitivos (ASC)** | David Mestanza Rubia y Javier Méndez García-Brioles |

## **INFORME SOBRE EL USO DE ADMENTOR**

Hemos planteado unos pequeños problemas sobre la pantalla de carga de cierta aplicación. Primero, hemos propuesto el tiempo que se debería mostrar la pantalla de carga, y después, el color que esta debería tener.

En la **Figura 1.1** hemos introducido nuestros problemas en el Espacio de Problemas de ADMentor para después tomar una decisión.

****

*Figura 1.1: Espacio de Problemas*

En la **Figura 1.2** aparece nuestro Espacio de Soluciones, donde se ve que hemos decidido lo siguiente:

* El tiempo que durará la pantalla de carga será de 5 segundos ya que no es ni demasiado corto ni demasiado largo, respecto a las otras opciones.
* El color de la pantalla será el azul, ya que nos resulta más amigable y cómodo para el usuario. Las otras opciones serían menos apropiadas para la vista del usuario.

## 

*Figura 1.2: Espacio de la Solución*

## **1- ANÁLISIS DE REQUISITOS:**

|  |  |  |
| --- | --- | --- |
| **Identificador** | **Nombre** | **Descripción** |
| **RF-1** | Adaptación del producto al cliente | La presentación de las páginas web y los productos deberán adaptarse automáticamente al cliente desde PC, Tablet o móvil. De esta forma habrá que tener en cuenta que hay que hacer la migración pensada para la universalidad de clientes. |
| **RF-2** | Componentes de presentación | Son los responsables del control de la interfaz de usuario y el consumo de servicios remotos. Este componente está directamente relacionado con el RF-1 ya que será el que controlará la forma en la que se adaptan las páginas webs y los productos a los distintos clientes. |
| **RF-3** | Lógica de dominio o de negocios | La aplicación deberá tener una forma definidas unas reglas para determinar como la información que proviene del mundo real puede ser creada, almacenada y cambiada. |
| **RF-4** | Lógica de acceso a base de datos | Son los componentes de acceso a datos responsables de obtener acceso a las bases de datos (SQL o NoSQL). |
| **RF-5** | Lógica de integración de aplicaciones | Habrá que desarrollar un canal de mensajería entre aplicaciones para que todos sepan que hacen el resto siempre que deban saberlo. Principalmente en agentes de mensajería. |
| **RF-6** | Integrar microservicios asíncronos | La aplicación deberá poder integrar microservicios de forma asíncrona para reforzar la fiabilidad de los microservicios nativos de la tienda virtual. Estos microservicios no deberán afectar a otros subsistemas |
| **RF-7** | Integrar aplicaciones externas asíncronas | La aplicación deberá poder integrar aplicaciones externas de forma asíncrona para reforzar la fiabilidad de los microservicios nativos de la tienda virtual. |
| **RF-8** | Soporte de tareas | El sistema deberá soportar las tareas de identificación de usuarios (que necesita un nombre de usuario y una contraseña), catálogo de microservicios y procesado de pedidos (que consta de varias opciones de forma de pago junto con el número del pedido), que utilizarán una base de datos SQL. |

|  |  |  |
| --- | --- | --- |
| **Identificador** | **Nombre** | **Descripción** |
| **RF-9** | Comunicación de microservicios | Los microservicios independientes se comunicarán a través del protocolo REST y de forma asíncrona. |
| **RF-10** | Cesta de compra | Habrá una función cesta de compra que usará una caché para almacenar su información dinámica (como los productos y la cantidad de dichos productos) y accederá a la localización de microservicios de terceros a través de una base de datos MongoDB. |
| **RF-11** | Bases de datos de microservicios | Cada microservicio tiene su propia base de datos, lo que permite separarlo totalmente de otros microservicios. |
| **RF-11.1** | Coherencia entre Bases de Datos | La coherencia entre las bases de datos de los diferentes microservicios se logra mediante eventos de integración de nivel de aplicación (a través de un bus de eventos lógicos), como por ejemplo mediante Command and Query Responsibility Segregation (CQRS). |
| **RF-12** | Comunicación entre clientes y microservicios | Los clientes móviles y de páginas Web se comunican con los puntos de conexión de puerta de enlace de API única enrutando las solicitudes de los clientes a los microservicios y, a continuación, se comunican con los microservicios. |
| **RF-12.1** | Comunicación entre clientes web y microservicio | Los clientes web tradicionales se comunican con el microservicio MVC. Mas adelante, El microservicio MVC se comunica con otros microservicios mediante una puerta de enlace API. |
| **RF-13** | Utilización de contenedores | El sistema constará de contenedores, dentro de los cuales se implementarán los diferentes microservicios. Los contenedores podrán ir implementados dentro de un mismo y único host o bien se podrá contar con un clúster para poder orquestar los contenedores, donde cada contenedor se ejecuta en una host diferente. En este caso cada nodo podría ejecutar un cierto número de contenedores. |
| **RF-14** | Comunicación de cliente a través de HTTP | Los clientes podrán comunicarse con el microservicio de HTTP a través de un Gateway que contiene diversas APIs con el fin de poder consultar las actualizaciones desde las aplicaciones cliente. |

|  |  |  |
| --- | --- | --- |
| **Identificador** | **Nombre** | **Descripción** |
| **RF-15** | Comunicación asíncrona basada en eventos. | Se podrá utilizar una comunicación asíncrona realizada a través de un bus de eventos. Se implementará bien a través de un agente de mensajería (como RabbitMQ) o de un Service Bus (como Azure Service Bus). |
| **RF-15.1** | Propagación de actualizaciones entre los microservicios. | La comunicación asíncrona basada en eventos servirá para propagar actualizaciones entre los microservicios. |
| **RF-15.2** | Integración con aplicaciones externas. | La comunicación asíncrona basada en eventos servirá para integrar aplicaciones externas. |
| **RF-16** | Comunicación con microservicios. | Las aplicaciones cliente pueden comunicarse con esos microservicios. La comunicación se hará por contenedores a través de las direcciones URL públicas publicadas por las puertas de enlace de API. |

***Nota****: El requisito funcional RF-15.2 es un requisito duplicado del RF-7.*

## **2- RESULTADOS**

## Iteración 1

Tuvimos diversos problemas a la hora de iniciar el proyecto, pues para todos era algo nuevo. Tuvimos problemas con el formato MADR, sobre todo a la hora de crear los archivos .md, pues no teníamos ni idea de cómo se utilizaban. Tras mucho ensayo y error, y preguntarle muchas dudas al profesor, conseguimos por fin arrancar. Además, a la hora de proponer Arquitecturas también hubo dificultades.

En primer lugar, el formato MADR se nos hizo costoso entenderlo debido a la cantidad de información que requería el mismo, pero supimos sacarlo adelante. Por otra parte, empezamos proponiendo 3 Arquitecturas distintas, siendo una de ellas descartadas rápidamente; aun así, mantuvimos una discusión acalorada los equipos ACS y ACC sobre si era mejor aplicar SOA o Microservicios, siendo esta última la resultante y la primera piedra para continuar el proyecto.

Diagrama

Descripción generada automáticamente

*Figura 2.1: Diagrama de la Iteración 1*

ASJ:

La primera iteración fue como toma de contacto con el programa StarUML. Al no ser unas decisiones muy complejas y al empezar el diagrama de cero, no tuvimos grandes dificultades.

## Iteración 2

Esta iteración es la primera que hicimos en la semana 3 y es la primera en la que nos pusimos por completo a tomar decisiones basadas en los requisitos. En esta iteración nos dimos cuenta de la utilidad del patrón Mediator, el cual nos ha ayudado a lo largo de toda la practica a establecer conexiones independientes. Este patrón lo utilizamos en las decisiones **ADD-6 y ADD-8** en esta iteración. También utilizamos el Patrón CQRS para **ADD-7** y para la decisión **ADD-9** utilizamosMongoDB.

En esta iteración debido a que todavía no estábamos seguros de cómo funcionaba el formato MADR cambiamos nomenclatura en las decisiones tomadas, lo que nos llevó a cometer el error de modificar el formato de las decisiones tomadas en la iteración anterior. Esto nos llevó que la fecha de modificación no concuerde con la iteración 1, pero eso fue un error puntual y no nos pasará en el resto de iteraciones.

Diagrama

Descripción generada automáticamente

*Figura 2.2: Diagrama de la Iteración 2*

ASJ:

En la segunda iteración nos costó entender cómo introducir el Patrón Mediator en nuestra arquitectura. Como aún nos faltaban muchas decisiones importantes no sabíamos muy bien cómo modelar los microservicios ya que aún no estaban diferenciados. Además, decidimos cambiar los microservicios a componentes en vez de utilizar paquetes, ya que nos parecía más adecuado de esta manera, porque un microservicio quedaría raro dentro de un paquete.

## Iteración 3

Aun en la semana 3 continuamos con nuestra toma de decisiones, abordando algunos requisitos que nos habíamos dejado atrás. Tuvimos problemas para entender lo que nos pedía un requisito **(RF-8)** y pedimos ayuda al profesor para terminar de comprenderlo. Finalmente propusimos dos decisiones **(ADD-11 y ADD-12)** de las cuales solo nos quedamos con una. Sobre la decisión **ADD-10**, estuvimos mucho tiempo buscando un patrón de diseño que pudiese encajar con el requisito **RF-6**. Tras varias propuestas que no encajaban y mucho tiempo invertido en búsqueda encontramos el patrón de diseño **BUS**. Muy completo, muy concreto, y aparentemente, creado por y para el problema que teníamos.

Diagrama

Descripción generada automáticamente

*Figura 2.3: Diagrama de la Iteración 3*

ASJ:

En esta tercera iteración empezamos a entender cómo debíamos modelar las decisiones utilizando los diferentes patrones, y además supimos diferenciar los microservicios en los cuatro que tenemos en la **Figura 2.3** (Cesta de Compra, Identificación de Usuarios, Catálogo de Microservicios y Procesado de Pedidos), y la arquitectura empezó a tener más sentido. Además, nos informamos bastante sobre las arquitecturas de microservicios y vimos bastantes ejemplos, por lo que empezamos a saber cómo funcionaba este tipo de arquitecturas.

## Iteración 4

En esta iteración, continuamos tomando decisiones entre las que cabe destacar el patrón federación (**ADD-13**), ya que nos costó un poco encontrar un patrón con el que consiguiésemos integrar aplicaciones externas, hasta que encontramos este patrón. Además, estuvimos un rato discutiendo porque algunos creíamos que se ajustaba más al problema que otros, pero como acabo siendo nuestra única opción factible, la terminamos aceptando. Aparte de esto, las demás decisiones de esta iteración no nos costaron demasiado encontrarlas y ni ponernos de acuerdo.

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

*Figura 2.4: Diagrama de la Iteración 4*

ASJ:

En la cuarta iteración introducimos cada base de datos dentro de su microservicio al llegarnos la decisión de que cada microservicio debería de estar en un contenedor **(ADD-15)**.

## Iteración 5

En esta iteración volvimos a escoger el patrón Mediator para una de nuestras decisiones, ya que se parecía a algunas de las anteriores decisiones en las que escogimos este patrón como para poder ser resuelto por el mismo, pero era lo suficientemente diferente como para necesitar otra decisión. Además, para la comunicación de puerta de enlace API(**ADD-17**), que inicialmente íbamos a usar el patrón de puerta de enlace API, encontramos el patrón BFF, que es como el patrón anterior, solo que un poco más complejo, pero resolvía los problemas que podía tener si no todos los clientes utilizasen los mismos métodos de nuestra API.

Diagrama

Descripción generada automáticamente

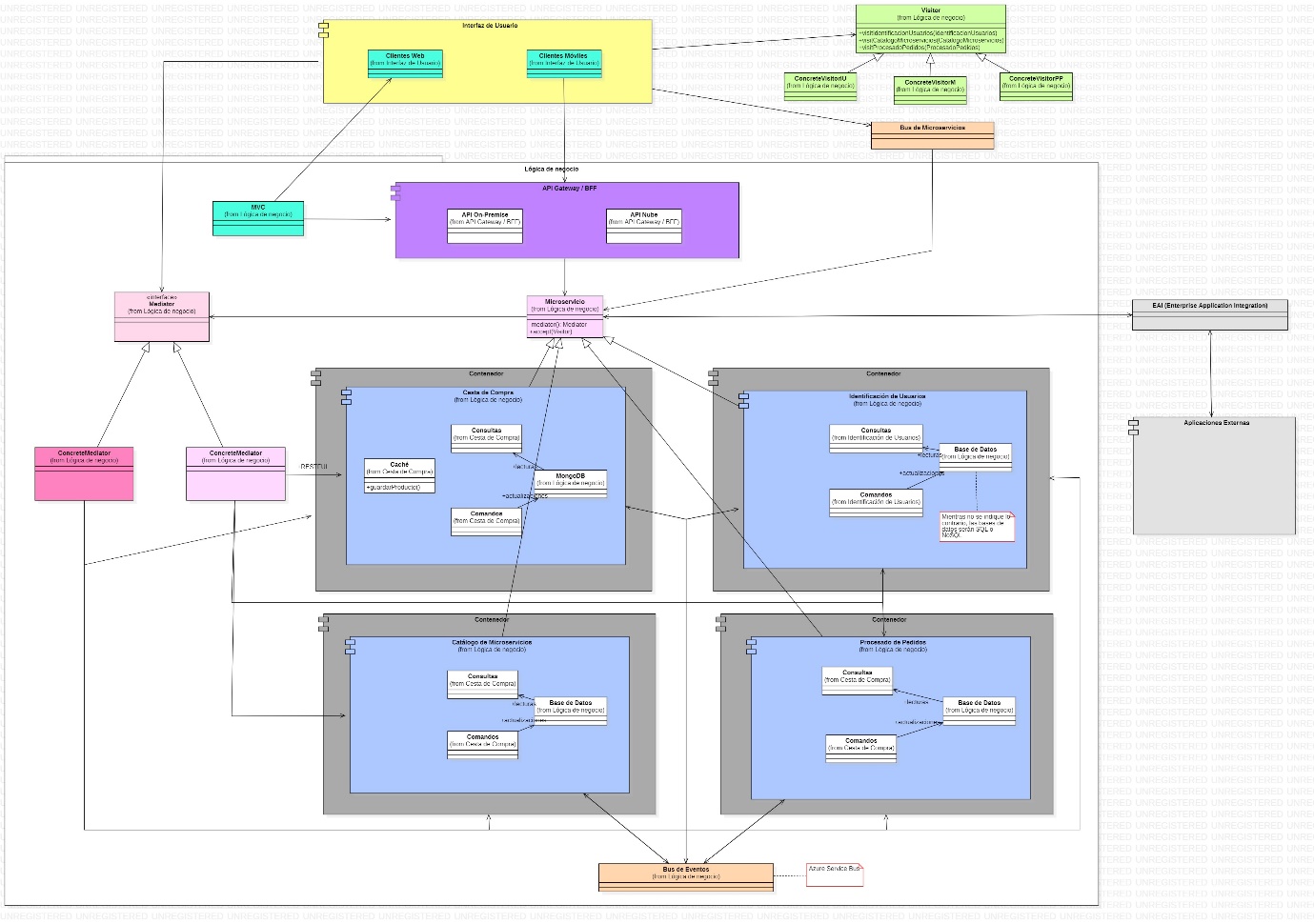
*Figura 2.5: Diagrama de la Iteración 5*

ASJ:

En la quinta iteración no tuvimos grandes problemas ya que algunos de los patrones utilizados anteriormente volvieron a aparecer.

## Iteración 6

Nos dimos cuenta de que teníamos un requisito duplicado, solo nos quedaba por hacer el **RF-15** y sus subrequisitos **(RF-15.1 y RF-15.2)**. Tras una nueva charla con el profesor nos dimos cuenta de que el podíamos obviar el RF-15.2 porque era exactamente igual que nuestro **RF-7**, así que eso mismo hicimos. Englobamos los requisitos **(RF-15 y RF-15.1)** en un mismo pack y propusimos dos patrones para solucionarlos. Nos planteamos el mediator, ya famoso en nuestro grupo y motivo de chiste fácil, y el patrón BUS. Al final nos decidimos por el segundo ya que este patrón lo habíamos escogido antes para un problema similar **(RF-6)** aunque no igual, y su versatilidad nos gustaba bastante.



*Figura 2.6: Diagrama de la Iteración 6*

ASJ:

En la sexta y última iteración solo tuvimos una decisión que modelar, y como ya sabíamos que aspecto final iba a tener nuestro diagrama, fue bastante sencillo.

## **3 - CONCLUSIONES**

Como conclusión principal destacaríamos los roles del equipo. Creemos que, en cualquier proyecto, sea del tipo que sea, debe haber roles como los que hemos representado los ASS y los ASC. Que haya un grupo que decida ideas es algo básico y necesario para cualquier proyecto, pero que haya otro que revise todo lo que decide ese primer grupo, le pone pegas, y plantea objeciones o nuevas ideas que en primera instancia no se habrían ocurrido nos parece una genialidad. Dicho esto, creemos que los errores y dudas que hemos tenido a lo largo del trabajo nos han enseñado a adaptarnos a situaciones adversas que no teníamos planeadas. Además, para reforzar lo dicho al principio hemos aprendido no solo a tomar mejores decisiones, sino a hablar y discutir sobre ellas para acabar tomando la mejor decisión posible de la forma más completa.

Como ASJs hemos aprendido a interpretar las decisiones de diseño y a modelarlas con patrones de diseño a través de un programa que no habíamos utilizado antes (StarUML). Además, hemos aprendido cómo modelar una arquitectura de microservicios desde cero, conociendo todas las partes de la misma y entendiendo cómo funcionan y se comunican entre sí. Esto nos parece interesante ya que actualmente se utilizan bastante las arquitecturas de este tipo en las empresas, y conocer cómo funcionan nos parece bastante útil y que facilita mucho el trabajo a la hora de ponerse a trabajar en este campo.

## **4 - BIBLIOGRAFÍA**

* Aplicación de patrones CQRS y DDD simplificados en un microservicio (8 de octubre de 2018). Recuperado de <https://docs.microsoft.com/es-es/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/apply-simplified-microservice-cqrs-ddd-patterns>
* Diseño de una aplicación orientada a microservicios (2 de octubre de 2018). Recuperado de <https://docs.microsoft.com/es-es/dotnet/architecture/microservices/multi-container-microservice-net-applications/microservice-application-design>
* Microservicios: más que a suma de sus partes (2 de marzo 2020). Recuperado de <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/los-microservicios-en-el-desarrollo-de-aplicaciones/>
* Patrón de Federación. Recuperado de <https://sites.google.com/site/toptecnologicosdelainfo/unidad-4/patrones-de-integracion>
* Rest y RestFul APIs (28 de diciembre de 2019). Recuperado de: <https://www.youtube.com/watch?v=JD6VNRdGl98&ab_channel=LeonardoKuffo>
* Patrón de bus. Recuperado de <https://andreshevia.com/2014/05/25/patrones-de-diseno-de-integracion-empresarial-a-aplicar/>
* Patrón de API Getway. Recuperado de <https://www.paradigmadigital.com/dev/api-gateway-patrones/>
* Patrón BFF (7 de agosto de 2019). Recuperado de: <https://rlbisbe.net/2019/08/07/el-patron-bff-backend-for-frontend/>
* StarUML. Recuperado de: https://staruml.io/

## **5- ANEXO: TABLA DE TIEMPOS DE TRABAJO**

Tabla 5.1: Tabla de tiempos de trabajo

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Semana** | **Iteración** | **Tiempo en ADD (ASS)** | **Tiempo de reflexión**  **(ASS y ASC)** | **Tiempo en refinar ADD**  **(ASS)** | **Tiempo de diseño ADD**  **(ASJ)** |
| 2 | 1 | 90 | 85 | 35 | 45 |
| 3 | 2 | 55 | 45 | 20 | 80 |
| 3 | 3 | 40 | 45 | 30 | 30 |
| 4 | 4 | 35 | 45 | 20 | 35 |
| 4 | 5 | 40 | 50 | 25 | 20 |
| 5 | 6 | 25 | 70 | 30 | 15 |