|  |  |
| --- | --- |
| CAPTURA Y REPRESENTACIÓN DE DECISIONES DE DISEÑO  PRÁCTICA 1 |  |
|  |  |
| 22/11/2021  GRUPO 8  Hugo Coto G. ([h.coto.2018@alumnos.urjc.es](mailto:h.coto.2018@alumnos.urjc.es))|  Alberto García S. | Rodrigo Marqués B. |  Enrique Martín S. | Iván Sánchez L. |

[1. Roles 1](#_Toc88480645)

[2. Introducción 1](#_Toc88480646)

[3. Resultado Iteraciones 2](#_Toc88480647)

[Tarea 1: Semana 1 2](#_Toc88480648)

[Requisitos 2](#_Toc88480649)

[Tarea 2: Semana 2 3](#_Toc88480650)

[Arquitectura de microservicios 3](#_Toc88480651)

[Migración a nueva arquitectura 4](#_Toc88480652)

[Gestión de compras y acceso a plataforma de pagos 5](#_Toc88480653)

[Tarea 2: Semana 3 6](#_Toc88480654)

[Módulo de pedidos, compras y módulo de devoluciones 6](#_Toc88480655)

[Módulo de mensajería y seguridad 8](#_Toc88480656)

[Tarea 2: Semana 4 9](#_Toc88480657)

[Componentes de la lógica de negocio 9](#_Toc88480658)

[Componentes de la lógica de negocio 10](#_Toc88480659)

[Lógica de acceso de datos 12](#_Toc88480660)

[Middleware de mensajería e integración asíncrona de microservicios 14](#_Toc88480661)

[Creación de la nueva BBDD y bus lógico 15](#_Toc88480662)

[Determinar número de contenedores 16](#_Toc88480663)

[Vía de conexión, monitorización y comunicación de microservicios 17](#_Toc88480664)

[Componentes de presentación 18](#_Toc88480665)

[Tarea 3: Semana 5 19](#_Toc88480666)

[Conclusión en base a las decisiones tomadas 21](#_Toc88480667)

[Bibliografía 21](#_Toc88480668)

[Tabla de tiempos 21](#_Toc88480669)

# 1. Roles

Arquitectos Software Senior (ASS):

* Rodrigo Marqués Buil
* Enrique Martín Santorcaz

Arquitectos Software Junior (ASJ):

* Iván Sánchez Labrador

Arquitectos Software Cognitivos (ASC):

* Hugo Coto González
* Alberto García Sroda

# 2. Introducción

El objetivo de esta práctica es realizar el diseño de una tienda virtual basada en una arquitectura Web de tres capas y se desea migrar a una arquitectura de microservicios.

Semana a semana desarrollaremos los objetivos propuestos buscando una evolución incremental en nuestro trabajo, que nos de pie a poner en práctica de manera ordenada los puntos que se expondrán a continuación.

Se pondrán en práctica las nociones de diseño aprendidas durante el desarrollo de la asignatura, que darán como resultado un diseño capaz de dar respuesta a todos los requisitos funcionales capturados de las peticiones y descripción del cliente de lo que espera que sea el sistema que se va a diseñar.

Será en estos términos que tomaremos las decisiones de diseño en cuánto a patrones y arquitecturas que consideremos apropiadas dado el ámbito y las condiciones del proyecto. Dichas decisiones serán recogidas y documentadas en el debido formato de cara a una mejor legibilidad y estructuración.

Cada decisión tomada tendrá su reflejo en el correspondiente diagrama que conformará el marco general de nuestro diseño, y de este modo, a través de la interrelación entre las arquitecturas y los patrones seleccionamos obtendremos una visión clara de nuestra solución.

# 3. Resultado Iteraciones

## **Tarea 1: Semana 1**

A continuación, se dispone la tabla con los requisitos funcionales que han sido extraídos del enunciado de la práctica en la Tarea 1: Semana 1:

|  |  |  |
| --- | --- | --- |
| ID | Requisitos | Descripción |
| RF1 | Elección de arquitectura | Se indicará la elección de la arquitectura a tomar para el cliente. |
| RF1.1 | Arquitectura de microservicios | Se toma la Arquitectura basada en Microservicios como elección en el proyecto. |
| RF1.2 | Migración de la antigua arquitectura a la nueva | Se migrará todos los módulos de la aplicación para que estén basados en una arquitectura por microservicios. |
| RF1.3 | Mejorar la flexibilidad y escalabilidad | Realizar una optimización en las bases de datos y realizar buenas praxis en la utilización de la arquitectura. |
| RF2 | Componentes de presentación | Responsables de la parte cliente. |
| RF2.1 | Control de interfaz de usuario | Se indicará como se controla la interfaz de usuario. |
| RF2.2 | Consumo de servicios remotos | Se indicará como controlar el consumo de servicios remotos. |
| RF3 | Componentes de la lógica de negocio | Responsable de la funcionalidad de la aplicación. |
| RF3.1 | Módulo de pedidos y compras | Se indicará cómo realizar las compras y pedidos para usuario. |
| RF3.2 | Módulo detección de preferencias | Se indicará la información necesaria y trata correcta de la información de preferencias. |
| RF3.3 | Módulo de conexión a sistemas de pago | Se indicará de qué forma se realizarán los pagos y con que medios. |
| RF3.4 | Módulo de mensajería a dispositivos móviles | Se indicará la forma de la mensajería a dispositivos móviles. |
| RF3.5 | Módulo de seguridad en compras | Se indicará la seguridad en la que se basa la arquitectura. |
| RF3.6 | Módulo de devoluciones | Se indicará como se realizarán las devoluciones para clientes. |
| RF4 | Lógica de acceso a bases de datos | Son los componentes responsables para obtener acceso a las BBDD. |
| RF5 | Sistema de mensajería por middleware independiente | Implementación de middleware único para mensajería. |
| RF6 | Integración asíncrona de los microservicios | Se implementará para soportar un mayor número de compras y preferencias. |
| RF7 | Creación nueva base datos | Se creará para almacenar la localización de los microservicios. |
| RF8 | Integración bus eventos lógicos | Para una correcta coherencia de los datos entre las distintas BBDD. |
| RF9 | Determinar número de contenedores | Limitación de intentos de compra a 5 intentos. |
| RF10 | Conexión y microservicios | Definición de tipo de conexión a la aplicación y entre microservicios. |
| RF10.1 | Definir vía de conexión | Conexión a la aplicación. |
| RF10.2 | Definir comunicación y monitorización de los microservicios | Conexión y monitorización de los microservicios. |

*(Tabla de requisitos)*

## **Tarea 2: Semana 2**

En esta segunda semana, empezamos a tomar las decisiones de diseño en función a los requisitos funcionales que habíamos extraído la semana anterior. Primeramente, comenzamos por decidir qué tipo de arquitectura iba a ser la principal.

La arquitectura principal del nuevo sistema será una arquitectura de microservicios, ya que es la demanda que el cliente ha solicitado expresamente para mejorar la flexibilidad y escalabilidad del sistema que se quiere migrar desde una arquitectura Web de tres capas.

|  |  |
| --- | --- |
| *Título de la decisión* | Arquitectura de microservicios |
| *ID* | ADD-0001 |
| *Fecha* | 30-10-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Hugo Coto G. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF1: “Elección de arquitectura”  RF1.1: “Arquitectura de microservicios” |
| *Decisiones alternativas (opciones)* | Opción 1(única): Arquitectura de microservicios |
| *Decisión final tomada (opción seleccionada)* | Opción 1(única): Arquitectura de microservicios |
| *Consecuencias positivas* | * Es el que ha solicitado el cliente. * Mejoras en la flexibilidad y escalabilidad. * Facilidad a la hora de aplicar una implementación continua. |
| *Consecuencias negativas* | * Posible dificultad a la hora de migrar el sistema. * Tener verificado en la BBDD todos los microservicios del sistema para un mayor control. |
| *Documentación* | https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices |

*(Tabla ADD-0001)*

Aunque esta arquitectura también tiene sus inconvenientes, como pueden ser el gran consumo de recursos que implica, lidiar con la complejidad adicional de los sistemas distribuidos o implementar dependencias de un servicio a otro, la hemos elegido, en primer lugar, porque es la arquitectura que quiere el cliente y, además, facilitan el despliegue, el mantenimiento y el testeo de los microservicios.

Continuando con la toma de decisiones, propusimos comenzar a desarrollar el proyecto asentando la idea de revisar la escalabilidad y flexibilidad como un componente clave en futuras decisiones, siendo ésta la manera más sensata y segura de desarrollar estas características en el sistema.

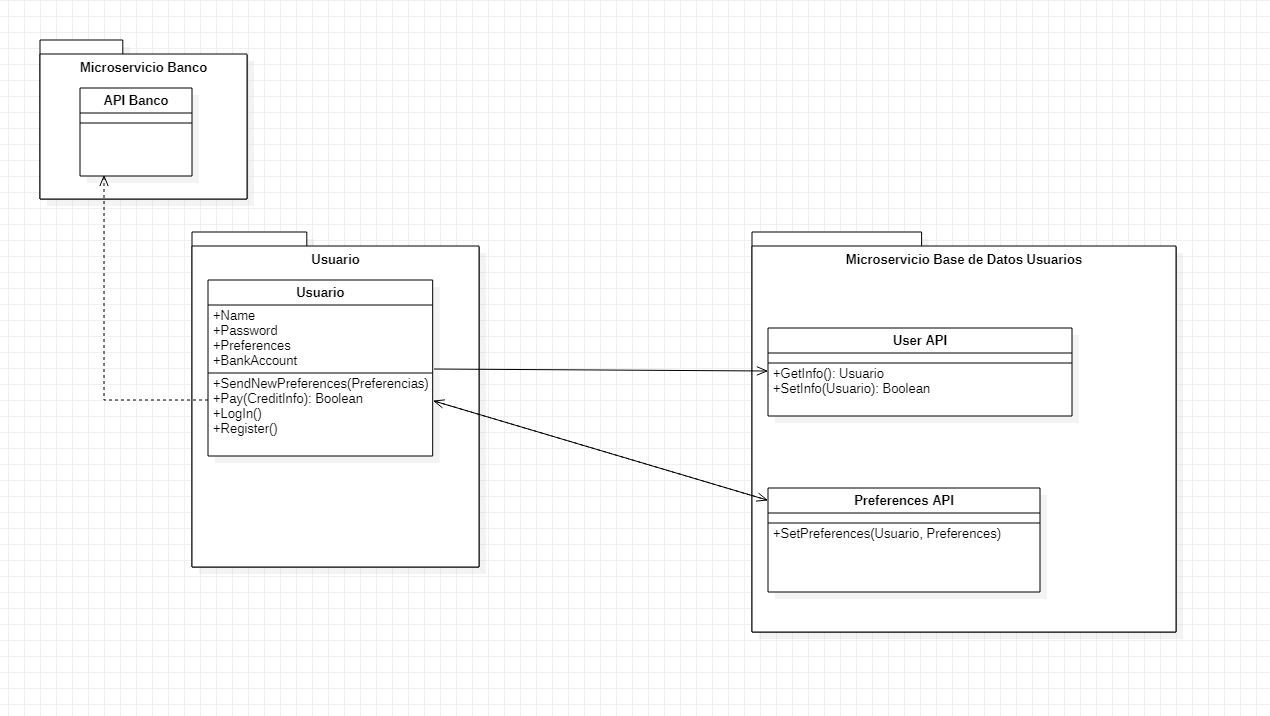
|  |  |
| --- | --- |
| *Título de la decisión* | Migración a nueva arquitectura |
| *ID* | ADD-0001 |
| *Fecha* | 30-10-2021 | 14-11-2021(revisión) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | *- No hay ASC en esta decisión -* |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF1.2: “Migración de la nueva arquitectura”  RF1.3: “Mejora de flexibilidad y escalabilidad” |
| *Decisiones alternativas (opciones)* | --- |
| *Decisión final tomada (opción seleccionada)* | --- |
| *Consecuencias positivas* | * Seguridad en implementación de escalabilidad y flexibilidad y revisión continua de las mismas. |
| *Consecuencias negativas* | * No se vislumbra una escalabilidad palpable desde el comienzo del desarrollo de la migración. * Posible pérdida del contacto con estos requisitos durante el desarrollo de la migración. |
| *Documentación* | --- |

*(Tabla ADD-0002)*

A continuación, dimos solución a la gestión de compras y acceso a las plataformas de pago. Para solventar el problema, utilizamos el patrón REST para comunicar los microservicios implementados.

|  |  |
| --- | --- |
| *Título de la decisión* | Gestión de compras y acceso a plataforma de pagos |
| *ID* | ADD-0004 |
| *Fecha* | 4-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Alberto García S. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF3.2: “Módulo de detección de preferencias”  RF3.3: “Módulo de sistema de pago” |
| *Decisiones alternativas (opciones)* | * Opción 1: la BBDD almacenará las preferencias de los usuarios en base a las búsquedas anteriormente realizadas en la aplicación, debe estar implementada mediante un solo microservicio con dos API REST desarrolladas en *JAVA*. * Opción 2: La lógica es la misma que en la opción 1, pero las API desarrolladas en *Phyton* puesto que es un lenguaje más potente e intuitivo. * Opción 3: La lógica es la misma que en la opción 1, pero las API desarrolladas en *Node* ya que es una opción perfectamente válida. |
| *Decisión final tomada (opción seleccionada)* | Opción 2 |
| *Consecuencias positivas* | * Comunes:   + El sistema de preferencias generado por búsquedas anteriores permite que el usuario encuentre productos parecidos a los que le interesan, y, por tanto, algo útil para el cliente.   + La BBDD NoSQL estaría correctamente conectada con los diferentes módulos del sistema, y recopilar la información de los diferentes usuarios no es una operación complicada de realizar ni implementar.   + Actualizar la BBDD cuando el usuario ha dejado de navegar por la aplicación con las nuevas preferencias generadas, permitirá que este sistema sea flexible y se adapte a los nuevos gustos de los usuarios. * Opción 1:   + JAVA está dispuesto para albergar este tipo de apps, lo que puede facilitar la implementación. * Opción 2:   + La potencia nativa de Python sumado con lo intuitivo que es el lenguaje, lo convierte en una opción más que factible. * Opción 3:   + Podemos expandir nuestro sistema añadiendo módulos de forma fácil gracias al NPM\* además de la alta escalabilidad que permite el uso de clusters. |
| *Consecuencias negativas* | * Separar en un sistema de dos APIs las implementaciones propuestas podría resultar confuso a la hora de conectarlas con los módulos de usuario e interfaz. * La base de datos NoSQL puede verse afectada por una sobrecarga en la lógica planteada. |
| *Documentación* | * <https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices> * https://microservices.io/patterns/index.html |

*(Tabla ADD-0004) \*****Node Package Manager***

**

*(UML ADD-0004)*

## **Tarea 2: Semana 3**

En este caso vamos a desarrollar los módulos de pedidos, compras y devoluciones. Para llevarlo a cabo, volveremos a utilizar el patrón REST para comunicar microservicios, mientras que la comunicación con las bases de datos se realizará mediante peticiones Query. Cabe destacar, que reestructuramos los módulos para una mejor división del sistema, que permita gestionarse de manera más escalable.

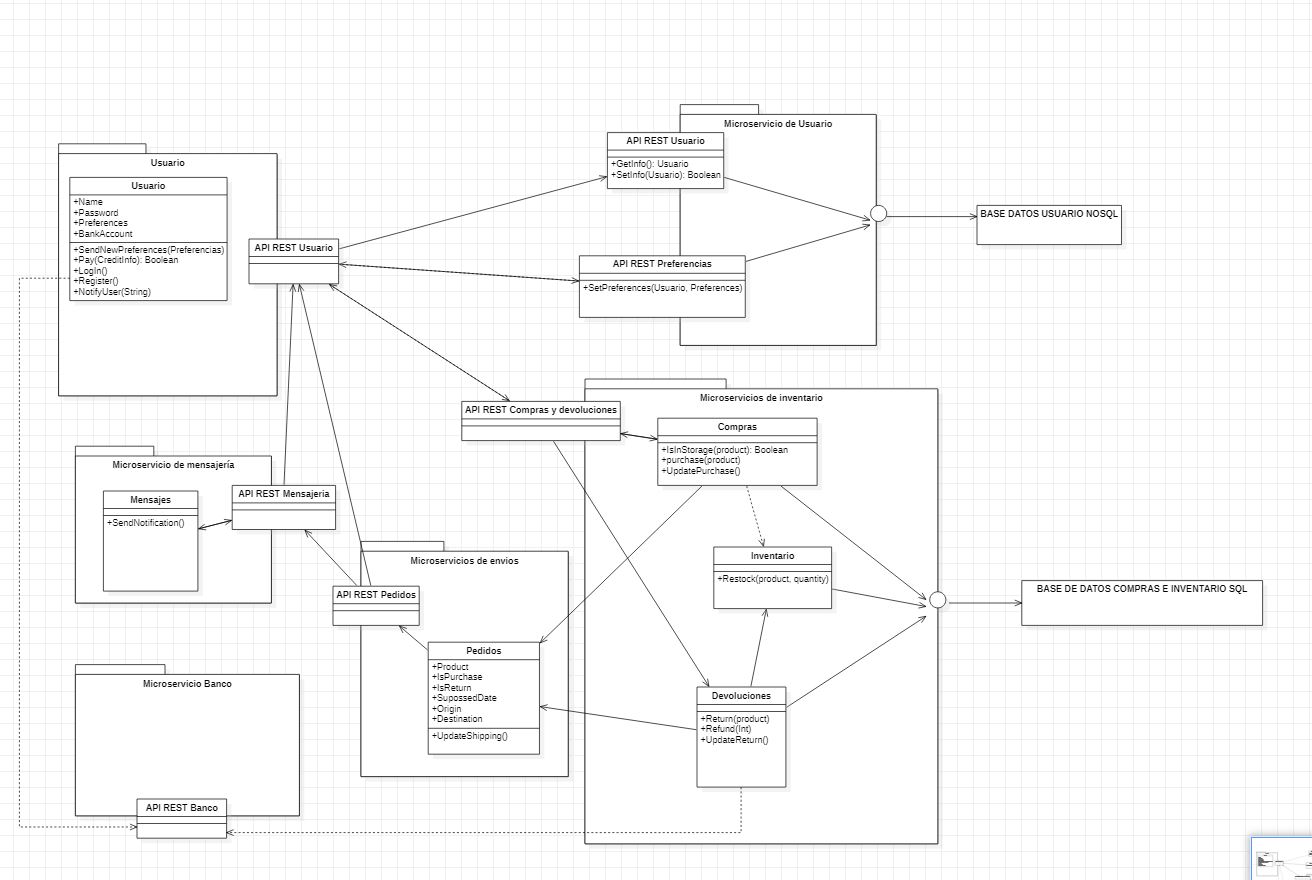
|  |  |
| --- | --- |
| *Título de la decisión* | Módulo de pedidos, compras y módulo de devoluciones |
| *ID* | ADD-0005 |
| *Fecha* | 4-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Hugo Coto G. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF3.1: “Módulo de pedidos y compras”  RF3.5: “Módulo de seguridad en compras”  RF3.6: “Módulo de devoluciones” |
| *Decisiones alternativas (opciones)* | * Opción 1: El módulo de seguridad se integrará dentro del de compras y devoluciones para asegurar una revisión continuada del mismo. Se dividirán el resto de módulos en dos microservicios: el de inventario gestionará pedidos, compras y devoluciones, además de relacionarse con la BBDD que almacena información sobre el propio inventario y las compras realizadas; y, por otro lado, el de envíos gestionará los pedidos. La relación para comprobar el estado del inventario se realizará mediante peticiones Query, mientras que los microservicios se comunicarán vía HTTP mediante APIs REST. * Opción 2: Separar la base de datos SQL en dos para diferenciar información de inventario y compras. * Opción 3: Separar en varias APIs la propuesta de la opción 1 para diferenciar usuarios móvil y Web. |
| *Decisión final tomada (opción seleccionada)* | Opción 1 |
| *Consecuencias positivas* | * Seguridad informática acorde a los datos que el sistema almacena al implementarse a lo largo del desarrollo. * Sistema de compras relacionado con el stock, asegurando un buen funcionamiento de este. * Se aseguran varios puntos exigidos por el cliente. |
| *Consecuencias negativas* | * BBDD no conectadas en este punto, pudiendo suponer un problema al desarrollar el sistema y enlazarlo. * Es posible que la BBDD sea muy compleja de manejar. |
| *Documentación* | * <https://docs.microsoft.com/es-es/azure/architecture/microservices/design/> * <https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices> |

*(Tabla ADD-0005)*

Finalmente, esta semana finalizó con el desarrollo del módulo de mensajería y seguridad. Al llevar a cabo el desarrollo de este, volvimos a utilizar el patrón REST para comunicar los diferentes microservicios involucrados en la comunicación.

|  |  |
| --- | --- |
| *Título de la decisión* | Módulo de mensajería y seguridad |
| *ID* | ADD-0006 |
| *Fecha* | 4-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Hugo Coto G. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF3.4: “Módulo de mensajería a dispositivos móviles” |
| *Decisiones alternativas (opciones)* | * Opción 1: Mediante API REST, el microservicio de mensajería se comunicará tanto con el módulo de usuario móvil como con el microservicio de pedidos. Esto se llevará a cabo mediante lenguaje de programación JAVA. * Opción 2: Misma lógica, pero en Python. * Opción 3: Misma lógica, pero en NodeJs. * Opción 4: Se añadirá a la opción 1 una BBDD que almacene los mensajes notificados. |
| *Decisión final tomada (opción seleccionada)* | Opción 2 |
| *Consecuencias positivas* | * Seguridad a la hora de implementar el sistema con una mayor escalabilidad y flexibilidad al ser un sistema más modularizado. * No es necesaria una nueva BBDD, la implementación de buenas relaciones es suficiente. * Mayor robustez del sistema al utilizar los datos de la BBDD SQL, minimizando los errores del sistema. |
| *Consecuencias negativas* | * Otro lenguaje de programación podría ser más interesante para estas características. (Inicialmente se propuso JAVA, pero tras la discusión ASC se cambió a Python). * Si no se implementa de manera correcta, al compartir la BBDD con otro microservicio podría dar lugar a fallos o colisiones. |
| *Documentación* | <https://microservices.io/patterns/microservices.html> |

*(Tabla ADD-0006)*

**

*(UML Componentes lógica de negocios)* ***\* Se aplica un solo UML debido a que es prácticamente parecido ya que una decisión dependía de la otra.***

## **Tarea 2: Semana 4**

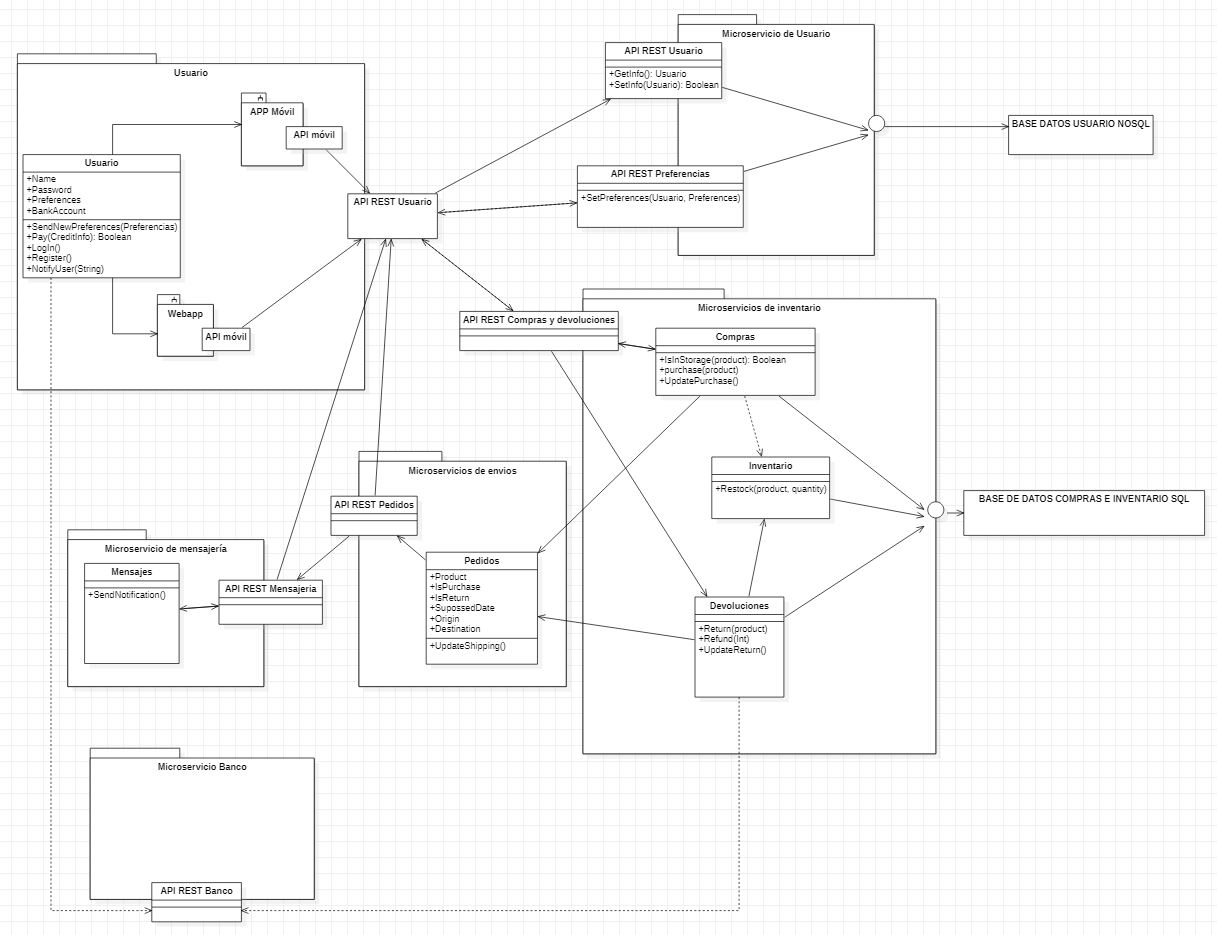
Al ser prácticamente la última semana, se llevaron a cabo las decisiones restantes y se consiguió una visión más global del proyecto en sí, forzando la revisión de algunas decisiones anteriores y haciendo que tuviéramos que retomar algunos debates que considerábamos cerrados.

|  |  |
| --- | --- |
| *Título de la decisión* | Componentes de la lógica de negocio |
| *ID* | ADD-0003 |
| *Fecha* | 12-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Alberto García S. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF3: “Componentes de la lógica de negocio” |
| *Decisiones alternativas (opciones)* | * Opción 1: es conveniente compaginar el número de requisitos con el número de microservicios a diseñar. Acabaría por tener uno que gestione la lógica de usuarios, otro para la gestión de compras y acceso a la plataforma de pago, otro para pedidos de compras y devoluciones, y otro para mensajería y seguridad. * Opción 2: Si orientáramos el diseño a una aplicación más enfocada a los microservicios, podemos entender que, además del de la lógica de usuario ya existente, podemos repartir la lógica de pedidos, compras y devoluciones en varios microservicios, resultando en uno para compras e inventario y otro para gestionar envíos. Por otro lado, un microservicio de mensajería parece necesario en cualquier opción. Además, incluiríamos otro para gestionar los pagos de las compras que realice el usuario, conectando la aplicación con el banco del mismo. * Opción 3: similar a la anterior, con la diferencia de que la lógica de compras, devoluciones e inventario, se gestiona de otra manera. Por un lado, el inventario se gestiona de manera independiente, mientras que las compras y devoluciones pueden mantenerse como un único microservicio. |
| *Decisión final tomada (opción seleccionada)* | Opción 2 |
| *Consecuencias positivas* | * Parece muy consistente con los requisitos pedidos. * Gracias a juntar compras, devoluciones e inventario en un solo requisito, podemos ahorrar en la implementación de las conexiones de BBDD. |
| *Consecuencias negativas* | * Implementar las clases con esta decisión dentro de los microservicios puede quedar algo difuso, especialmente en los microservicios de envío y compras y devoluciones. |
| *Documentación* | <https://microservices.io/patterns/microservices.html> |

*(Tabla ADD-0003)*

|  |  |
| --- | --- |
| *Título de la decisión* | Componentes de la lógica de negocio |
| *ID* | ADD-0008 |
| *Fecha* | 12-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Hugo Coto G. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF2.1: “Control de interfaz de usuario”  RF2.2: “Consumo de servicios remotos” |
| *Decisiones alternativas (opciones)* | * Opción 1: Se realizará una separación entre la versión móvil, la WebApp y las posibles aplicaciones a terceros, aplicando una arquitectura concreta de microservicios basada en Backend for frontends. Esto nos permitirá, en un futuro, diferenciar distintas API Gateways dependiendo de la necesidad y el entorno del servicio, pudiendo utilizar algunas API REST específicas para un entorno y facilitando que no se propaguen errores en posibles conexiones a API REST que no deberían ejecutarse en un entorno en concreto. * Opción 2: Se realizará un sistema de modelo vista controlador en la propia parte del cliente utilizando una única entrada por API Gateway ofreciendo la diferenciación de los componentes visuales de la aplicación. Luego, el propio cliente selecciona la API a la que tiene que conectarse. |
| *Decisión final tomada (opción seleccionada)* | Opción 1 |
| *Consecuencias positivas* | * Al tomar esta decisión tomaremos la entrada específica que es necesaria para cada APIREST o servicio. * Versión más consistente y escalable para cada entorno independientemente de los demás que se encuentren en el sistema. |
| *Consecuencias negativas* | * Mayor complejidad a la hora del desarrollo. |
| *Documentación* | <https://microservices.io/patterns/microservices.html> |

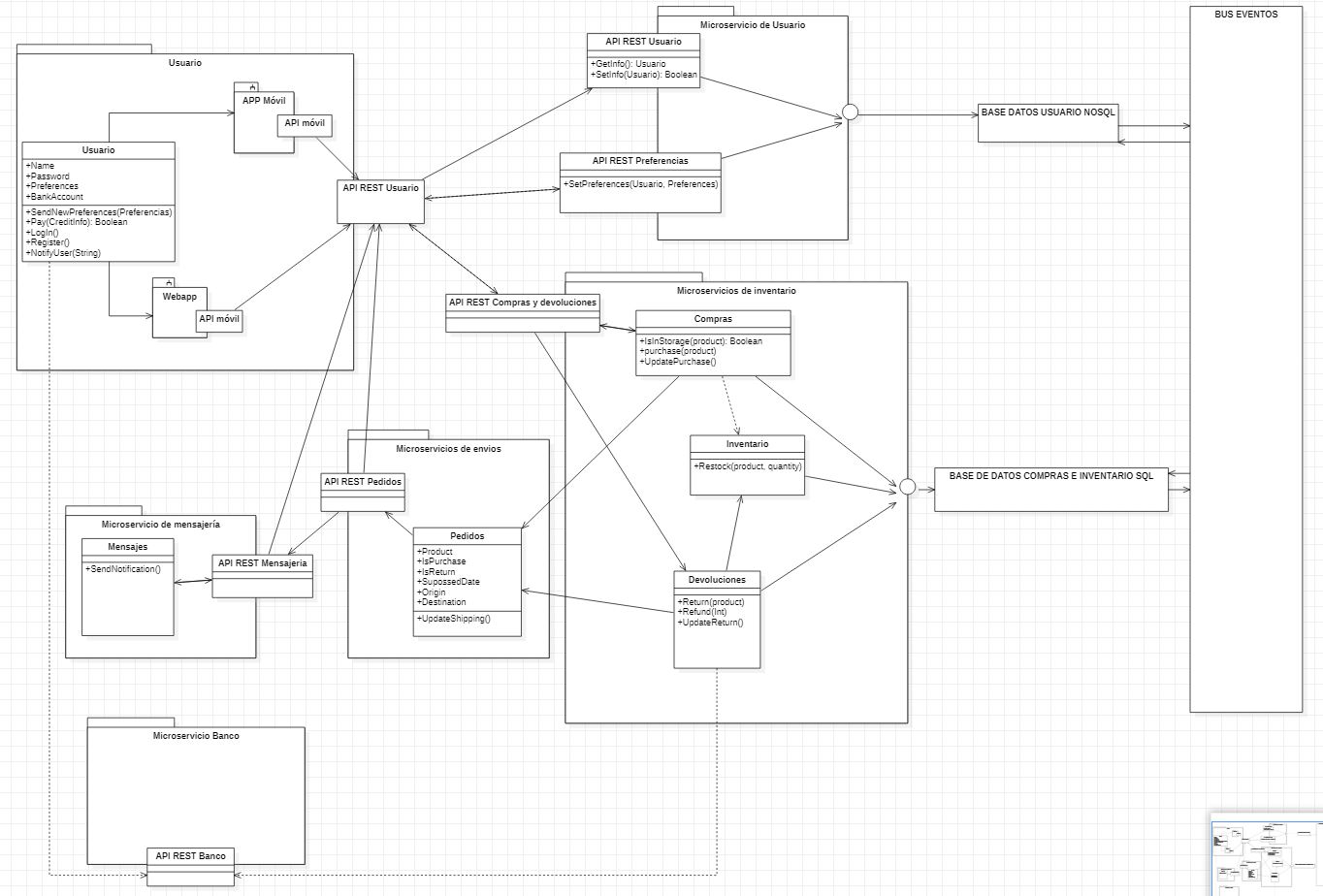
*(Tabla ADD-0008)*

**

*(UML ADD-0008)*

|  |  |
| --- | --- |
| *Título de la decisión* | Lógica de acceso de datos |
| *ID* | ADD-0009 |
| *Fecha* | 14-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Alberto García S. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF4: “Lógica de acceso a BBDD” |
| *Decisiones alternativas (opciones)* | * Opción 1: Considerar la posibilidad de expresar la dependencia mediante API, en lugar de una dependencia de BBDD directa. Eso permitiría cambiar el esquema de datos interno tanto como sea necesario y solo preocuparse por el diseño de la interfaz cuando se trata de dependencias. Esto facilita agregar API adicionales y depreciar lentamente las API antiguas en lugar de cambiar el diseño de una BBDD junto con todos los microservicios dependientes. En otras palabras, aún puede implementar nuevas versiones de Microservicios de forma independiente, siempre que las API antiguas aún sean compatibles. * Opción 2: Cuando no se puede evitar usar las mismas BBDD y está dividiendo los límites de su servicio de manera que varios equipos/servicios requieran las mismas entidades, introduce dos dependencias entre el equipo de Microservicios y el equipo responsable del esquema de datos: a) Formato de datos, b) Datos reales. Esto no es imposible de resolver, pero sí con cierta sobrecarga en la organización, y si introduce demasiadas, es probable que su organización se vea paralizada y ralentizada en el desarrollo. * Opción 3: Tendríamos dos BBDD, lo que permitiría separarlo totalmente de otros microservicios. Cuando se necesite, la coherencia entre las BBDD de los diferentes microservicios se logra mediante eventos de integración a nivel de aplicación, controlado por CQRS. Por esto, las restricciones de negocio deben adoptar la coherencia final entre los microservicios y BBDD. |
| *Decisión final tomada (opción seleccionada)* | Opción 3 |
| *Consecuencias positivas* | * Mayor seguridad en caso de corrupción, posible pérdida de datos, robo o filtración de seguridad. * Reducción de tamaño de cada una de las BBDD, lo que implica una mejora del uso de recursos. * Mejora de las velocidades de acceso y recuperación de datos. |
| *Consecuencias negativas* | * Mayor dificultad de implementación y relación de ambas BBDD, al no ser un único elemento independiente. * Aumento de la complejidad en tiempo al realizar un Backup completo. * Posible desactualización de metadatos al poder ser distintos en las BBDD. |
| *Documentación* | <https://microservices.io/patterns/microservices.html> |

*(Tabla ADD-0009) \* Command and Query Responsibility Segregation*



*(UML ADD-0009)*

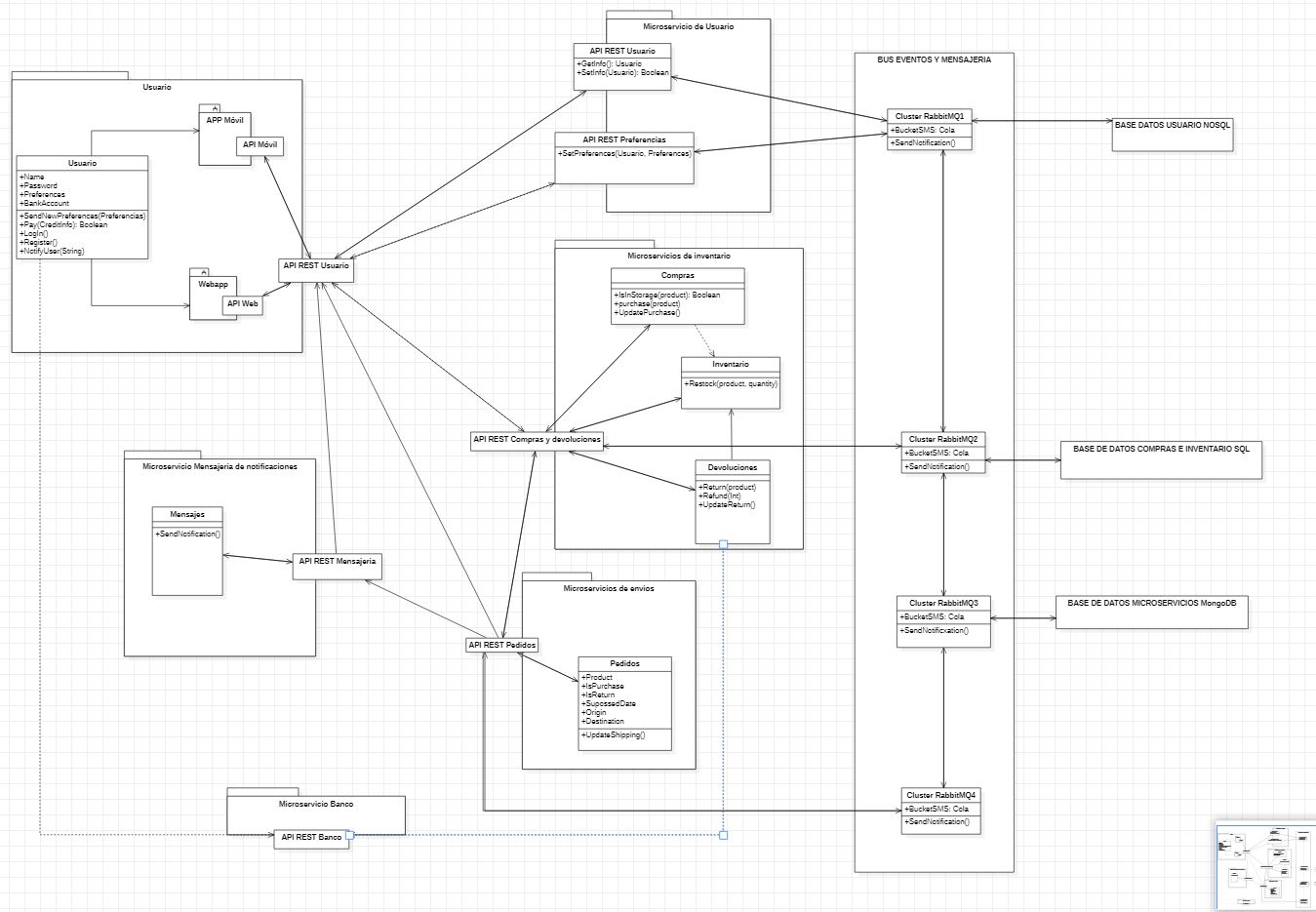
|  |  |
| --- | --- |
| *Título de la decisión* | Middleware de mensajería e integración asíncrona de microservicios |
| *ID* | ADD-0010 |
| *Fecha* | 14-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Hugo Coto G. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF5: “Sistema de mensajería por Middelware independiente”  RF6: “Integración asíncrona de los microservicios” |
| *Decisiones alternativas (opciones)* | * Opción 1: Se toma un middleware orientado a mensajes mediante un protocolo AMQP de punto a punto, es decir, los mensajes quedan almacenados hasta que el receptor, en este caso los microservicios que han solicitado la información den el ok a la recepción, esto permite la integración asíncrona mediante el bus de mensajes entre los distintos microservicios. * Opción 2: Se toma un middleware orientado a mensajes mediante un protocolo AMQP de Publish-Subscribe en los que aporta la información a los receptores de forma periódica siempre y cuando esté aceptado para recibir la información proporcionada por el publicador. * Opción 3: Se toma la comunicación entre los microservicios mediante HTTP *polling*, permitiendo realizar peticiones asíncronas HTTP entre los distintos microservicios. |
| *Decisión final tomada (opción seleccionada)* | Opción 1 |
| *Consecuencias positivas* | * Interoperabilidad y asincronidad de las peticiones y comunicaciones. * Persistencia de los datos y trata de ellos independientemente del formato de los datos y el destino de estos. |
| *Consecuencias negativas* | * Difícil implementación y puesta en marcha. |
| *Documentación* | <https://microservices.io/patterns/microservices.html> |

*(Tabla ADD-0010)*

*\* Los cambios de UML de esta decisión se reflejan el UML ADD-0011*

|  |  |
| --- | --- |
| *Título de la decisión* | Creación de la nueva BBDD y bus lógico |
| *ID* | ADD-0011 |
| *Fecha* | 14-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Hugo Coto G. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF5: “Sistema de mensajería por Middelware independiente”  RF6: “Integración asíncrona de los microservicios”  RF7: “Creación de nueva BBDD”  RF8: “Integración del bus de eventos lógicos” |
| *Decisiones alternativas (opciones)* | * Opción 1: Se creará un bus lógico basado en RabbitMQ con 3 cluster para las separaciones en las colas de mensajería. A su vez, se creará una BBDD conectada al bus en MongoDB para la localización de los microservicios. * Opción 2: igual a la opción 1, pero el bus lógico se creará en Azure Service Bus. * Opción 3: igual a la opción 1, pero el bus lógico se creará en MassTransit. |
| *Decisión final tomada (opción seleccionada)* | Opción 1 |
| *Consecuencias positivas* | * Interoperabilidad y asincronidad de las peticiones y comunicaciones. * Persistencia de los datos y trata de ellos independientemente del formato de los datos y el destino de estos. |
| *Consecuencias negativas* | * Difícil implementación y puesta en marcha. |
| *Documentación* | <https://microservices.io/patterns/microservices.html> |

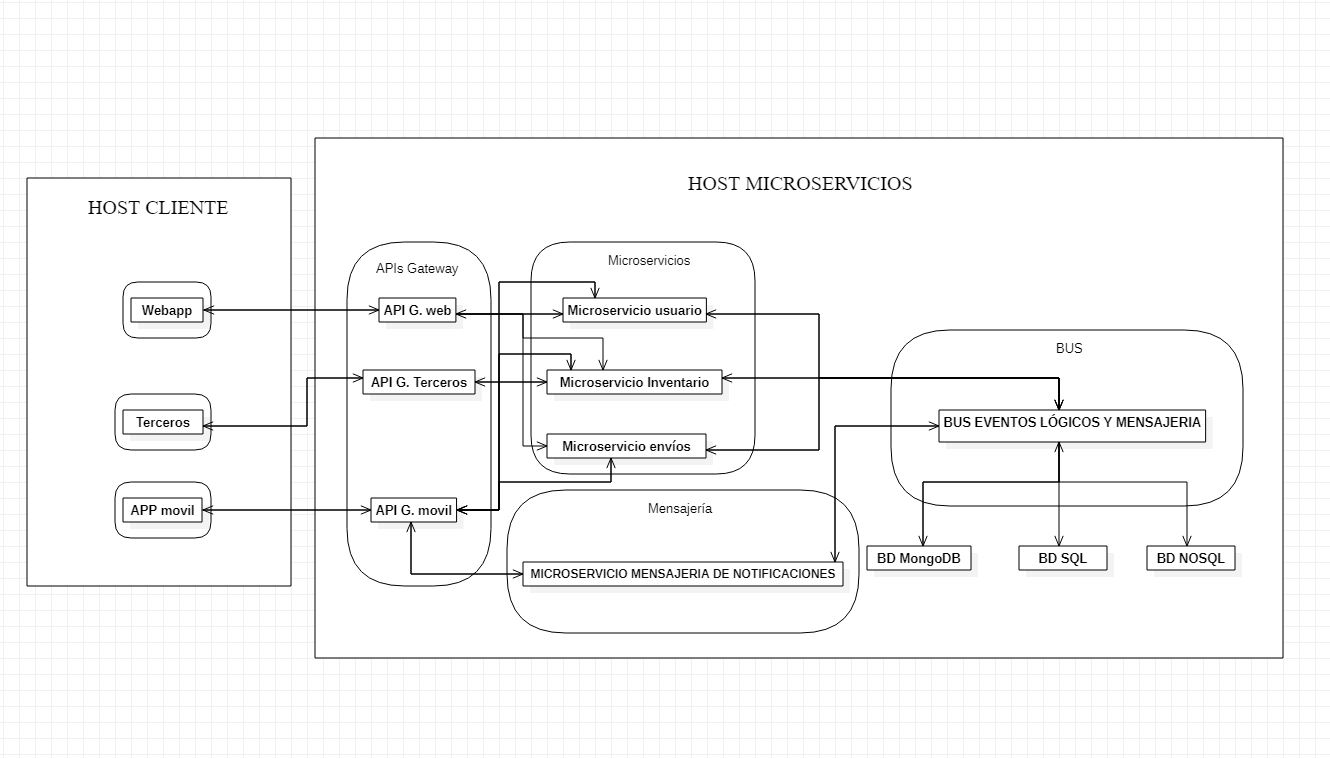
*(Tabla ADD-0011)*

**

*(UML ADD-0011)*

|  |  |
| --- | --- |
| *Título de la decisión* | Determinar número de contenedores |
| *ID* | ADD-0012 |
| *Fecha* | 14-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Hugo Coto G. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF9: “Determinar número de contenedores” |
| *Decisiones alternativas (opciones)* | * Opción 1: dos hosts de contenedores diferenciados, uno para la parte cliente y otra para el apartado de los microservicios. En el host cliente se crearán 3 contenedores que contendrán la WebApp, la versión móvil y las aplicaciones de terceros. En el caso del host de microservicios estará formado por 4 contendores formados por: API GATEWAYS, los microservicios de inventario y envíos, otro para la mensajería móvil y uno para el bus de eventos lógicos. * Opción 2 Lo mismo que en la opción 1, pero en el host de microservicios solamente habrá 3, unificando el 2 con el 3 de la opción anterior. * Opción 3: Lo mismo que en la opción 1, pero poniendo un contenedor con toda la info en cada uno de los hosts. * Opción 4: Similar al primer caso, pero en el host usuario unificar los 3 contenedores en uno solo. |
| *Decisión final tomada (opción seleccionada)* | Opción 1 |
| *Consecuencias positivas* | * Mayor optimización y rendimiento. * Control total de los distintos sistemas y los microservicios. |
| *Consecuencias negativas* | * Mayor complejidad de implementación. |
| *Documentación* | <https://microservices.io/patterns/microservices.html> |

*(Tabla ADD-0012)*



*(UML ADD-0012)*

|  |  |
| --- | --- |
| Título de la decisión | Vía de conexión, monitorización y comunicación de microservicios |
| *ID* | ADD-0013 |
| *Fecha* | 14-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Hugo Coto G. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF10: “Conexión y microservicios”  RF10.1: “Definir vía de conexión”  RF10.2: “Definir comunicación y monitorización de los microservicios” |
| *Decisiones alternativas (opciones)* | * Opción 1: Se realizará una comunicación HTTP a un único API Gateway para todos los microservicios con la separación pertinente a aparte de la gestión cliente del servidor. * Opción 2: Se realizará una comunicación HTTP a 3 API Gateway diferenciadas acorde a cada gestión de la app permitiendo diferenciar los distintos sistemas y monitorización de los microservicios. |
| *Decisión final tomada (opción seleccionada)* | Opción 2 |
| *Consecuencias positivas* | * Permite un mayor control de las API y su diferenciación. * Mayor rendimiento al tener separadas las distintas peticiones. |
| *Consecuencias negativas* | * Es más difícil de implementar y un poco más tedioso. |
| *Documentación* | <https://microservices.io/patterns/microservices.html> |

*(Tabla ADD-0013)*

*Debido a que este UML pertenece a la versión final se explicará en el apartado de la Semana 5*

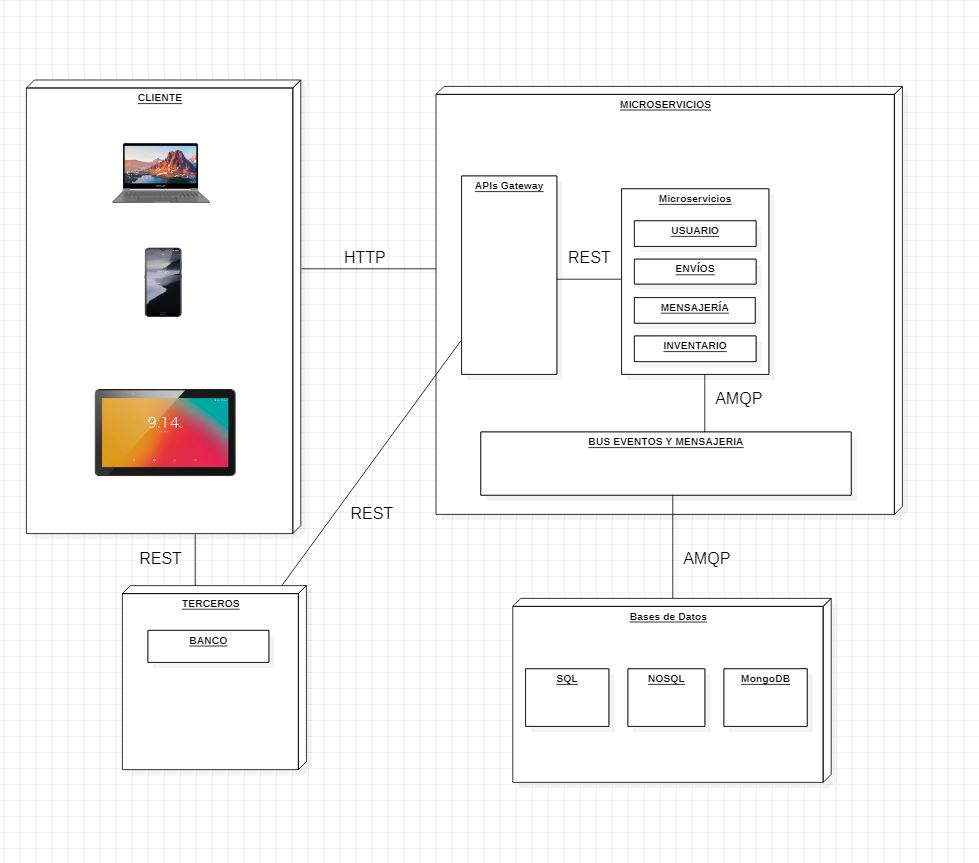
|  |  |
| --- | --- |
| *Título de la decisión* | Componentes de presentación |
| *ID* | ADD-0007 |
| *Fecha* | 17-11-2021 (última actualización) |
| *Creadores* | Enrique Martín S. & Rodrigo Marqués B. |
| *ASC* | Hugo Coto G. |
| *Estado* | Aceptada |
| *Requisitos (directrices de decisión)* | RF2: “Componentes de presentación” |
| *Decisiones alternativas (opciones)* | * Opción 1: El usuario debe poder registrarse e iniciar sesión en el sistema, además de cargar información con respecto a sus preferencias, compras actuales y compras concurrentes. Por otro lado, el sistema debe poder acceder a la plataforma de pago y recibir notificaciones que le envíe el sistema. Además, debe ser capaz de actualizar su información personal por si llega algún cambio. Estas son las funcionalidades que exige el cliente además de las que mantienen la coherencia, persistencia de datos, modularización y por tanto escalabilidad del sistema. Finalmente, estas implementaciones deben gestionarse desde la propia aplicación existente en los dispositivos del usuario, además de desarrollarse en *JAVA* para mantener ciertos paralelismos con otros módulos del sistema. * Opción 2: igual que la opción 1, pero implementadas en Python, ya que puede resultar más sencilla y directa su implementación. |
| *Decisión final tomada (opción seleccionada)* | Opción 1 |
| *Consecuencias positivas* | * JAVA permite mantener coherencia con otros módulos del sistema. * Varias decisiones ya están en el sistema en otras decisiones, luego será más sencillo de relacionar y/o implementar. * Mayor escalabilidad futura, gracias a múltiples opciones para presentar nueva información al usuario. |
| *Consecuencias negativas* | * Es posible que la implementación no sea tan directa como con otros lenguajes. |
| *Documentación* | <https://microservices.io/patterns/microservices.html> |

*(Tabla ADD-0007)*

*Debido a que este UML pertenece a la versión final se explicará en el apartado de la Semana 5*

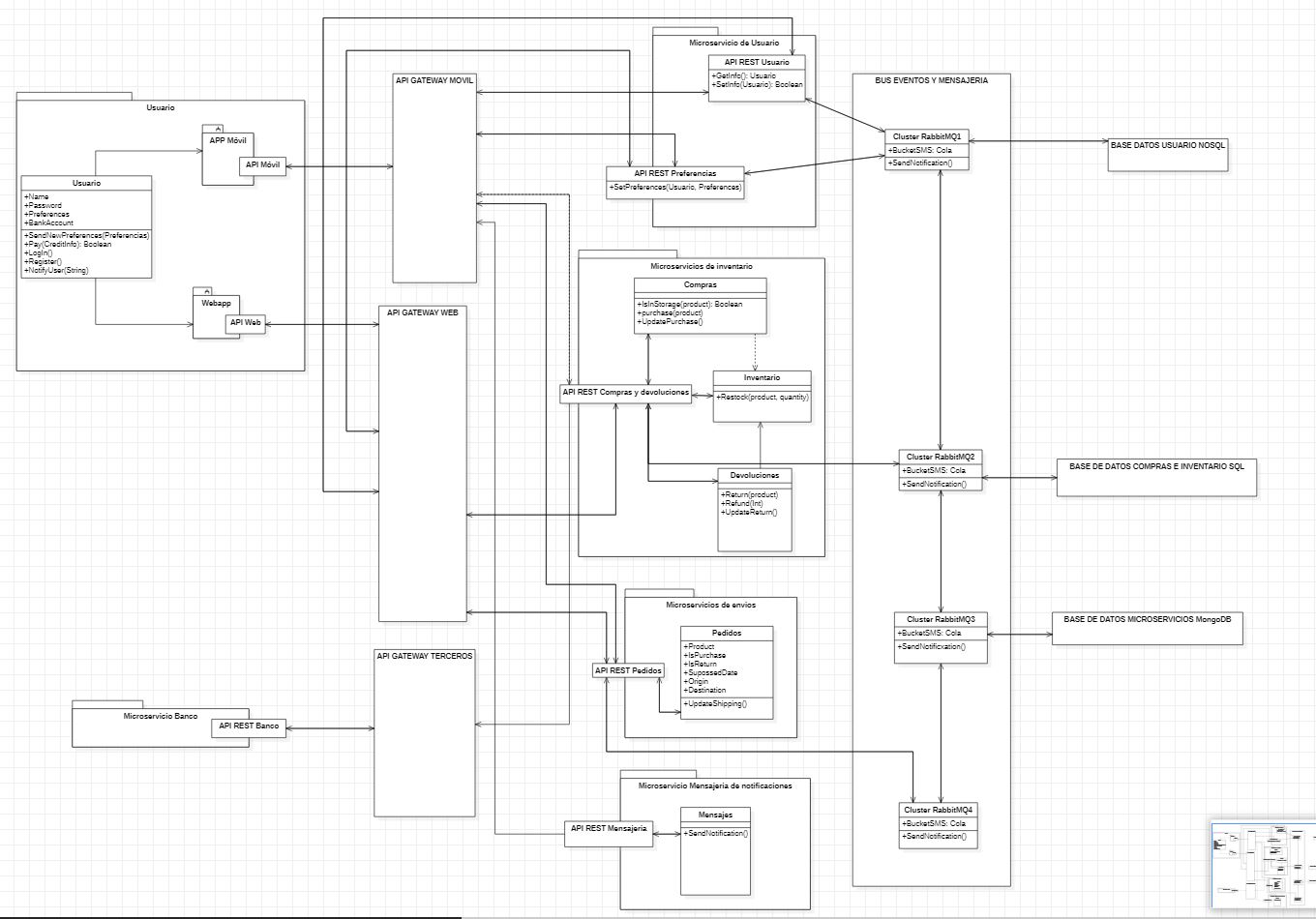
## **Tarea 3: Semana 5**

Tras las diferentes decisiones de diseños tomadas a lo largo de las iteraciones para la resolución de este ejercicio, el resultado final sería el conjunto formado por los siguientes diagramas:



*(Diagrama de despliegue)*

El diagrama de despliegue no ha sido posible realizarlo antes ya que varias decisiones esenciales no se tomaron hasta la semana 4, y antes que dar información a medias, hemos preferido guardarlo hasta este punto.



*(Diagrama de clases y paquetes final)*

Se puede apreciar en el diagrama final la unión de los microservicios mediante 3 API GATEWAY diferenciados facilitando la utilización de estos según la parte cliente desde la que se acceda. Para la persistencia de los datos, así como la comunicación asíncrona entre estos un bus de eventos y mensajería conectaos a todos estos y a las bases de datos.

# Conclusión en base a las decisiones tomadas

La experiencia que hemos obtenido del trabajo realizado nos aporta la importancia de analizar múltiples veces un problema antes de decidirnos por una solución, ya que las constantes revisiones ofrecen diferentes puntos de vista que nos ayuden a observar la imagen general, con los pros y los contras de cada punto, para abordar así las decisiones de diseño apropiadamente.

Además, nos hemos concienciado de la importancia de un buen diagrama de clases, ya que, si partimos de un diseño claro, la codificación pasa a ser un pequeño paso más para obtener el producto que desea el cliente.

Consideramos crucial el uso de iteraciones pues nos permiten detectar errores en etapas tempranas de diseño y corregirlos, sea sustituyéndolos por alternativas mejores o alterando aquellos puntos que hacen fallar la decisión actual.

De encontrarnos con un error de este calibre, nos supondría una gran pérdida de tiempo si utilizásemos un modelo en cascada, en el cual podríamos encontrarnos con que un pequeño problema puede llegar a convertirse en un error que inutilice todo el trabajo hecho hasta ahora, y en dicho caso, la única manera que tendríamos de enmendar el rumbo sería desechar todas las decisiones tomadas hasta aquella en la que se encuentre el error.

Finalmente estamos de acuerdo en que el desarrollo de la práctica nos ha servido no solo para poner en práctica los conocimientos teóricos aprendidos en la asignatura, sino que también para obtener un mayor enfoque en cuestiones de gran importancia para desempeñar nuestro futuro desarrollando software.

# Bibliografía

* Estilo de arquitectura de microservicios (05-10-2021) recuperado de Microsoft docs: <https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices>
* A pattern language for microservices, recuperado de: <https://microservices.io/patterns/index.html>
* Microservicios AWS, recuperado de: <https://aws.amazon.com/es/microservices/>
* Aprender acerca de la arquitectura de microservicios: https://docs.oracle.com/es/solutions/learn-architect-microservice/index.html#GUID-BDCEFE30-C883-45D5-B2E6-325C241388A5

# Tabla de tiempos

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Week | Iteration | Time in ADD  (ASS) | Reflection Time (ASS-ASC) | Time in refined ADD (ASS) | Design ADD Time (ASJ) |
| 1 | 1  (Requisitos) | 120 | 40 | 80 | 30 |
| 2 | 1 | 180 | 35 | 70 | 80 |
| 3 | 1 | 150 | 40 | 65 | 40 |
| 4 | 1 | 170 | 60 | 90 | 100 |