# SEMANA 2. ITERACION 1

**DIA 1**

## Decisiones de diseño

Tras analizar el problema en profundidad los arquitectos senior (ASS) han tomado la decisión de basarse en una arquitectura 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 y

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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SEMANA | ITERACION | TIEMPO ASS | REFLEXION ASS-ASC | REFELXION ASS | TIEMPO ASJ |
| 1 | 1 |  |  |  |  |
| 1 | 2 |  |  |  |  |
| 2 | 1 |  |  |  |  |
| 2 | 2 |  |  |  |  |

DÍA 2

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:

* 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.
* 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.
* Singleton garantiza la existencia de una única instancia para una clase. Este patrón se puede aplicar al requisito RF-3.1 ya que se necesitará un canal de mensajería.
* 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-3.4 donde se utiliza una cesta de la compra.