Proyecto

de CFGS

2º DAM

Tabla de contenidos

Introducción

El presente trabajo pretende mostrar una idea clara acerca de las elecciones que se siguen tomando para la realización del proyecto. Asimismo, se hace hincapié en la palabra 'elección' puesto que ya desde un primer momento se hace caprichoso desarrollar la habilidad de resolver problemas y adaptarse a los cambios.

"Walking on water and developing software from a specification are easy if both are frozen."

Edward V. Berard[1]

Por tanto, matizaría que este proyecto se puede dividir en 2 ideas. Por una parte estaría el concepto del *super proyecto*, entendido como una idea más cercana a lo que se consideraría real; y por otra, la pequeña adaptación que se pretende realizar. Es decir, sería una implantación a escala; pero los conceptos tratados pretenden ser los mismos.

Diseño inicial de la interfaz

La interfaz gráfica se orienta hacia los tres tipos principales de dispositivos[2], lo que conlleva que se debe adaptar a las particularidades de cada uno. Como se pretende realizar un diseño *responsive* lo más inclusivo e intuitivo posible, se realizará un diseño horizontal por secciones; ya que cada vez más el usuario está acostumbrado a ello.

1 Citado en https://www.softwarequotes.com/showquotes.aspx?id=613&name=Edward%20Berard .
2 Refiriéndose al tamaño PC, mediano y móviles.

Página principal.

Sección 1 → imagen corporativa representativa, a la derecha un botón de registro y login.

Sección 2 → justo debajo de lo anterior.

Sección 3 → […]

En cada sección se pretende dar una idea acerca del producto, quién es la empresa y qué ofrece. Se puede valer de elementos multimedia y enlaces a las secciones particulares que se quiera ampliar la información.

En la parte inferior estará la información de contacto, ubicación y notas legales.

Figura 1. Ejemplo de página web relacionada (portada).

Fuente: https://www.sap.com/index.html

Página de registro o login.

Consiste en un formulario simplemente, en él se muestra la información mínima: texto informativo, campos a rellenar y botón de registro o cancelación del proceso. Si es una transacción de registro, informará del éxito o fracaso de la acción del mismo. Si es login (autenticación), llevará a la pantalla de su perfil con sus pedidos (si es cliente), área de trabajo (si es trabajador) o cuadro de mando (si es del perfil business).

De esta forma queda separada la funcionalidad por roles, lo que mejorará el tratamiento de la información a la vez que permite que la ayuda contextual sea la relevante para ellos. Es decir, a un cliente le puede interesar saber más sobre productos o su persona de contacto, mientras que a los gerentes les resultará más interesante saber la información evolutiva del negocio.

Área de trabajo o cliente.

En la parte superior una barra de navegación donde se reflejan los accesos directos más relevantes y a la derecha tendrá la opción de acceder a su perfil y salir del sistema.

Figura 2. GUI en PC y tablets

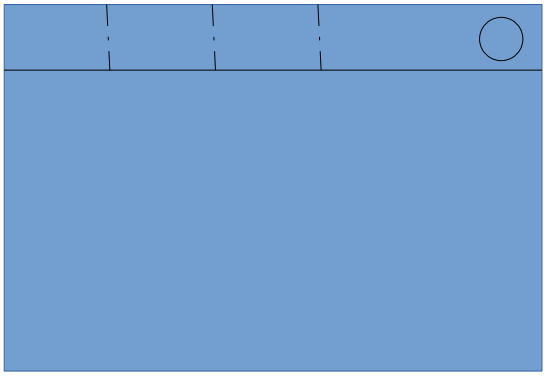
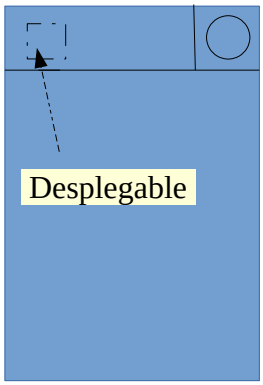


Figura 3. GUI en disp. móviles

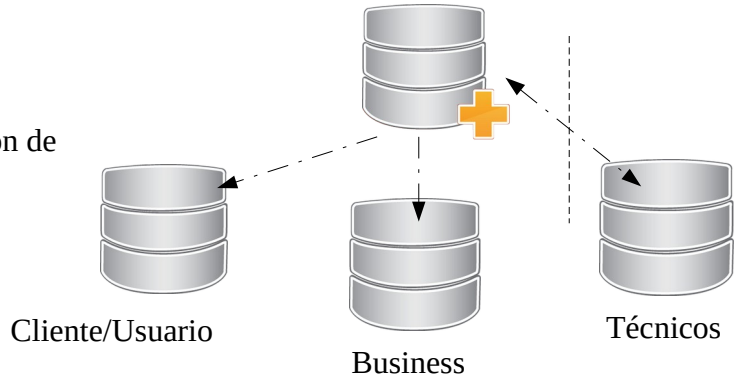


Nota: se pretende conseguir un diseño funcional y minimalista que sea cercano a las tareas que el usuario pretende realizar. Además, se busca que le sea fácilmente reconocible independientemente del dispositivo por el que accede.

Diseño del almacenamiento de datos

Realmente es este apartado el que va a dar coherencia a todo el proyecto, pues los datos es lo que proporciona sentido al negocio. Tratando de conseguir que exista confidencialidad, integridad y disponibilidad se ha optado por realizar una organización de la siguiente manera: En el *super proyecto* se va a tomar una "única fuente de verdad" centralizada, retomando la idea de los ERP. Se realizará una fragmentación horizontal en donde se tendrán réplicas de acceso a lectura. Para escribir, sí se realiza en un nodo central y se informará al resto de las réplicas con patrón *publisher-subscriber*. Además, se divide por roles, lo que ayudará a mejorar el acceso a la información y su disponibilidad. Existen 3[3] roles, de los cuales el cliente hará pedidos pero está supervisado por un vendedor, es éste el que deberá notificar realizar la transacción en un primer momento. El perfil *business* es independiente.

Figura 4.
Lógica de distribución de datos.



3 No se tendrán en cuenta en esta explicación los perfiles técnicos (desarrolladores y administradores).

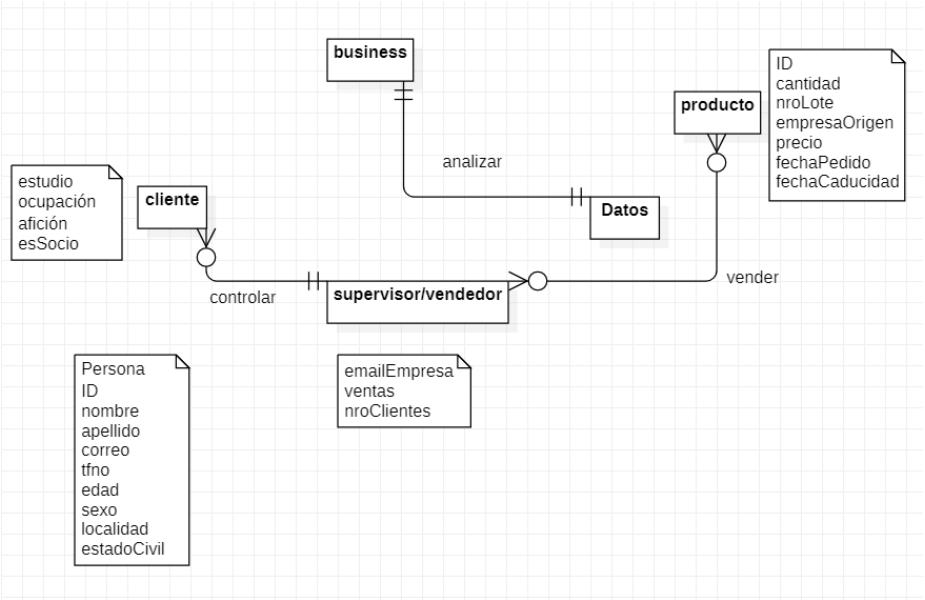
Como se aprecia en la imagen, aunque hay un nodo central; podría tratarse de una estructura recursiva, en donde dependiendo de la cantidad de peticiones pudiera ampliarse ese nodo o el conjunto entero. Además el nodo central seguramente sea una copia del total y estará en una ubicación distinta.

A modo informativo, el equipo técnico llevará su propio esquema basado en local/desarrollo/testing/producción, que será implementado según la política y estrategia que se quiera desarrollar. Por eso dan y reciben datos.

¿Por qué esta separación?

Uno de los principales motivos es por el alcance del proyecto. En un primer momento se pretende crear una dupla ERP-CRM, en donde el CRM se centraría en los clientes, los "supervisores" serían vendedores que harían de nexo de unión con el equipo interno (ERP) y el economista necesitaría datos generales; pero estaría más en el módulo BI.

Figura 5. Modelo ER inicial.



Este modelo viene a ser una representación de los conceptos más esenciales que pretende abarcar la empresa. Se correspondería con la siguiente lógica:

- Muchos clientes está supervisado por un vendedor.
- Un cliente está interesado en comprar productos.
- Estos productos son vendidos por muchos vendedores.
- De la relación vender surge un conjunto de datos que será analizado por un conjunto de personas (o procesos).
- Tanto el cliente como el supervisor heredarán de persona.
- Se omite profundizar en el equipo business y en la relación que tenga el producto con otros aspectos del proceso compra/venta y demás regulaciones.
- De los clientes interesan hábitos y perfiles.
- De los vendedores interesa saber posible *matches* con clientes.
- Del producto interesa su ratio de venta, empresas relacionadas, periodo temporal activo, cantidad y precio[4].

Diseño de la estructura de software

En el software existe una relación directamente proporcional entre la complejidad, tiempo en ejecución con la fragilidad del mismo. Es por esto, que se han venido desarrollando desde hace tiempo estrategias (patrones de diseño), para mitigar estos efectos. Han aparecido nuevos paradigmas de programación orientado a componentes, aspectos,… Sin embargo, la

4 Se puede pensar que es un pilar en sí mismo del que brotarán más ramas.

evolución que parece más factible es desacoplar funcionalidades totalmente. Y para ello, una de las mejores estrategias es la que se enfoca en desarrollar servicios. Es cierto que existe mucha disparidad conceptual en este sector, pero como apunta Martin Fowler en la conferencia que dió en GOTO acerca de los microservicios. Lo importante no es la definición en sí, sino el conjunto de prácticas lo que puede ser determinante a la hora de desarrollar buenos servicios.

Es aquí donde pretende situarse este proyecto, por tanto se va a seguir una arquitectura de servicios que se comunicarán entre sí. Dentro de los mismos, se sigue un patrón MVC en su mayoría y se pretende hacer uso de interfaces y composiciones entre clases[5] (por encima de la herencia).

La estructura interna de directorios es y será la estandarizada tanto por Spring/Springboot como Maven.

En cuanto al frontend, ReactJS obliga también a seguir sus propias reglas, por lo que trataré de seguir los estándares lo máximo posible.

Comunicación entre servicios.

Se pretende desarrollar una comunicación Restful lo más completa posible, haciendo uso de HTTP, APIs y de instrumentos como HATEOAS, que favorece la interconexión de los recursos. Springboot posee muchas opciones de filtrado y comunicación como Netflix Zuul, Eureka, etc. Buscaré la que mejor resuelva la problemática.

De esta manera, se conseguirá un sistema extensible gracias a esta arquitectura.

5 Lo que en Spring parece ser nombrado como inyecciones.

Así, se pueden entender estos servicios como componentes individuales que pueden ser organizados para ser tratados en conjunto. Para ello nos valdremos de Docker que nos proporcionará una mayor independencia por su flexibilidad.

Diseño de la estructura de hardware

Una vez que se tengan las imágenes creadas, se pueden trabajar con ellas desde repositorio local o remoto (como Dockerhub).

Por el diseño que se ha desarrollado, se pueden elegir diferentes opciones para el *deployment* orquestrado: DockerSwarm o Kubernetes principalmente[6]. Pretendo usar Kubernetes por ser una tecnología open source con gran capacidad de control y monitorización.

Con ello se puede formar un cluster con todos los servicios necesarios e ir regulando según se requiera. Esto es una solución prácticamente independiente de los sistemas operativos y requisitos hardware/software. Ya que al trabajar en capas de software, se puede ajustar más fácilmente qué imagen de base será la que corra el sistema. Se pueden determinar qué recursos son estrictamente necesarios y cuales no.

Funciones que el sistema pretende realizar

- Acceso al sistema con autenticación.
- Operaciones CRUD sobre productos.
- Desarrollo de pequeño cuadro de mando.

6 A nivel local trabajaré con minikube y el programa kubectl.

- Tarea programada matinal de emisión de informe.
- Utilización de *message broker* que reaccione a algún evento concreto.

Lista de referencias más relevantes

https://spring.io/guides/tutorials/react-and-spring-data-rest/

https://spring.io/guides/tutorials/rest/

https://spring.io/guides/gs/scheduling-tasks/

https://spring.io/guides/gs/rest-service/

https://spring.io/guides/gs/consuming-rest/

https://spring.io/guides/gs/relational-data-access/

https://spring.io/guides/gs/authenticating-ldap/

https://spring.io/guides/gs/messaging-rabbitmq/

https://spring.io/guides/gs/validating-form-input/

https://spring.io/guides/gs/handling-form-submission/

https://spring.io/guides/gs/accessing-data-rest/

https://spring.io/guides/gs/spring-boot-docker/

https://spring.io/guides/gs/spring-boot-kubernetes/

https://tanzu.vmware.com/developer/blog/understanding-the-differences-between-rabbitmq-vs-kafka/

https://tanzu.vmware.com/developer/guides/messaging-and-integration/rabbitmq-gs/

https://www.tutorialspoint.com/sap_abap/index.htm

https://www.youtube.com/watch?v=0NaNShUTwYY&t=3054s

https://www.youtube.com/watch?v=0EXfb3HzBTs (DBMS)

https://www.youtube.com/watch?v=RreRD41qlpw

https://www.youtube.com/watch?v=GZvSYJDk-us

https://www.youtube.com/watch?v=42J4KMHUJjE (API Rest tutorials point)

https://www.youtube.com/watch?v=wgdBVIX9ifA

https://martinfowler.com/articles/microservices.html

https://www.coursera.org/learn/ibm-containers-docker-kubernetes-openshift

https://www.freecodecamp.org/learn

https://spring.io/guides/gs/routing-and-filtering/

https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html