

Proyecto

de CFGS

2º DAM



Tabla de contenidos

i. Introducción.....	3
ii. Pruebas.....	3
iii. Despliegue en la nube: Netlify y Heroku.....	9
a. Opciones de deployment.....	9
b. Ventajas de estas dos plataformas.....	9
iv. Documentación de usuarios básica.....	13
a. Para empleados que trabajen con productos (stock).....	13
b. Para los clientes que hacen pedidos.....	16
c. Para consumidores de información del negocio.....	17
d. Información funcional para los técnicos y desarrolladores, recurso multimedia.....	19
v. Anexo.....	19
Explicación del funcionamiento conceptual del super-sistema. Tridimensionalidad, asincronía y concurrencia.....	19
vi. Repositorios o fuentes de información.....	21
vii. Referencias para esta etapa.....	22

Introducción

En este tratado se van a diferenciar conceptualmente los dos conceptos finales según el entendimiento tradicional de elaboración del software: testing (pruebas) y deployment (explotación). No obstante, si se ha seguido el hilo de esta saga, se han paralelizado tareas a medida que se ha ido desarrollando el proyecto.

Pruebas

Las pruebas son un recurso fundamental para dar calidad al software. Estas pretenden hacer mostrar los errores lo más pronto posible, pues se sabe que siempre están. La diferencia está en saber encontrarlos. Es por ello que se debe seguir una estrategia que permita abarcar el mayor número de casos relevantes.

En este proyecto se ha pretendido romper la aplicación a medida que se ha ido desarrollando para o bien solucionar un problema, o bien saber de las vulnerabilidades que se cuentan desde un primer momento.

Tests realizados:

- Pruebas unitarias con Junit5 y assertJ

Tomando como ejemplo la clase repositorio de producto, se ha tratado de seguir la metodología BDD.

- Concepto de prueba usando Mocks¹ con Mockito

Usando la clase del servicio de producto se ha pretendido usar mocks y utilizar diferentes técnicas para asegurarse que se hacen las llamadas a los métodos pertinentes y que se pudiera cubrir la mayor parte de código posible.

¹ No exitoso. Para mayor aclaración consultar ambas clases en el código, ya que están comentadas.

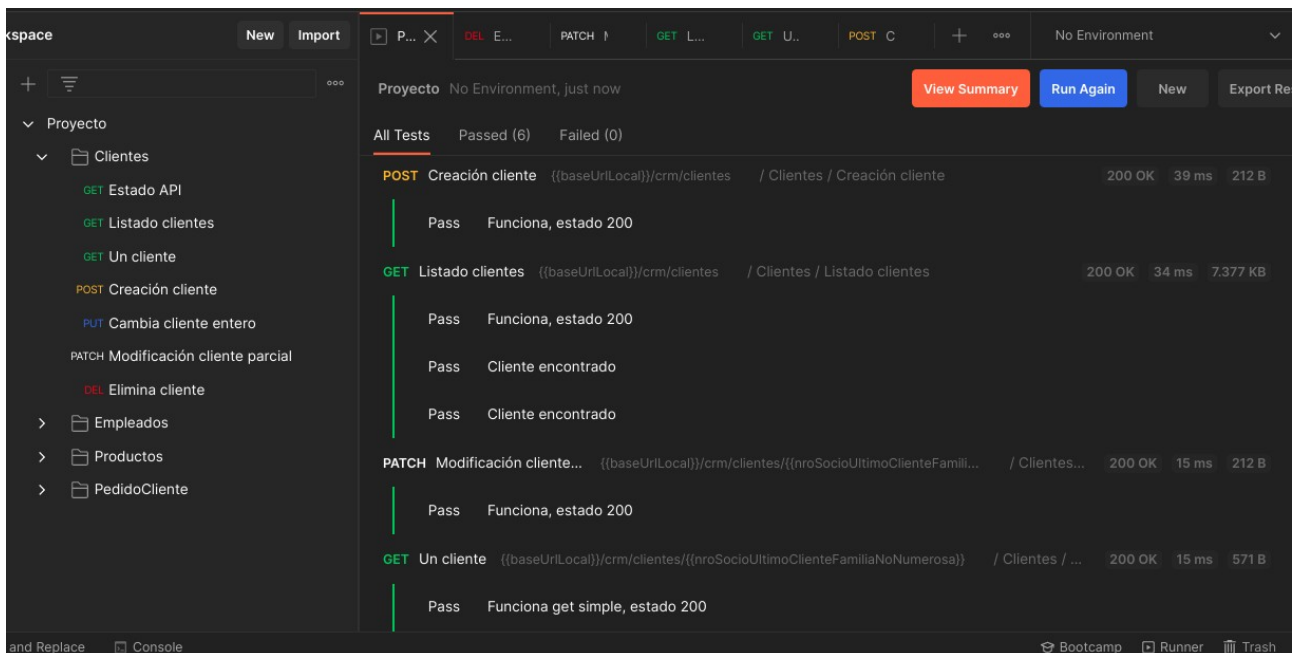
- Prueba de contenedores

Tanto el .jar como la base de datos por separado de manera independiente (la integración con docker-compose/kubernetes no se hizo en este proyecto).

- Prueba de la API

En este caso se utilizó la herramienta Postman, además se trato de sugerir un camino hacia la automatización de los tests con el Runner.

Nota: también es cierto que sólo se hizo una colección y de los tests expuestos, porque los otros deberían haberse modificado para que devolvieran un estado concreto al que pudiera reaccionar apropiadamente el aplicativo.



Concepto de automatización de pruebas usando Postman

Fuente: elaboración propia

Ejemplo de script del código usando la librería de aserciones Chai

```
// comprueba el estado de la respuesta

pm.test("Funciona, estado 200", () => {
  pm.response.to.have.status(200);
});

// toma el objeto respuesta para trabajar con él
const response = pm.response.json();

// recoge una lista filtrada
const clientesNoFamiliaNumerosa = response.filter(
  (cliente) => cliente.familiaNumerosa === false
);

// toma la inserción más reciente del subconjunto filtrado
const cliente = clientesNoFamiliaNumerosa[clientesNoFamiliaNumerosa.length - 1];

// crea una variable global para automatizar el proceso en la colección (la variable será accesible para los
// otros servicios)
if(cliente){
  pm.globals.set("nroSocioUltimoClienteFamiliaNoNumerosa", cliente.nroSocio);
}

// realiza algunas aserciones con Chai
pm.test("Cliente encontrado", () => {
  pm.expect(cliente).to.be.an('object');
  pm.expect(cliente.familiaNumerosa).to.be.false;
});

// finalmente imprime el número de socio y su nombre
console.log(cliente.nroSocio);
console.log(cliente.nombre);
```

- Prueba del frontend en navegador
- Prueba de integración backend con capa de persistencia²
- Prueba integración frontend-backend en local³
- Prueba con base de datos en memoria⁴.
- Prueba del deployment backend en Heroku
- Prueba del deployment backend con Postgres en Heroku
- Prueba de comunicación frontend local con backend completo en Heroku
- Prueba de frontend en Netlify con backend local

2 En este caso se usó Docker para correr una imagen del servidor PostgreSQL con el que se comunicaba la aplicación.

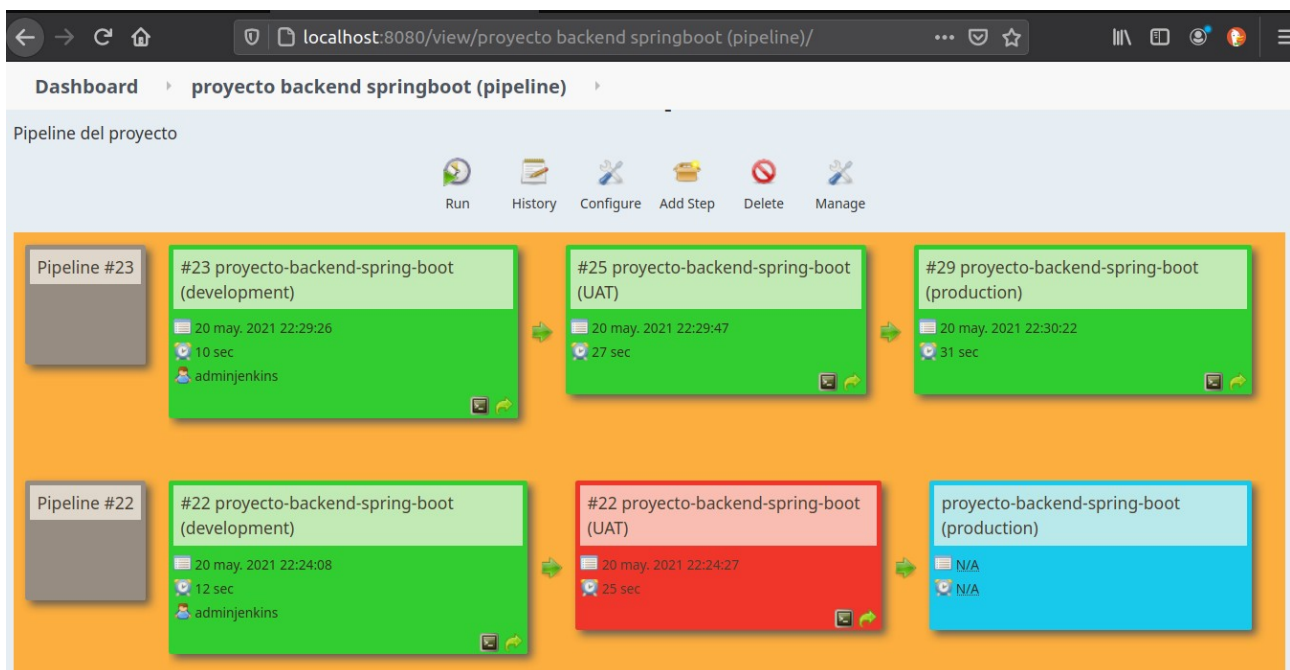
3 Las imágenes fueron mostradas en la anterior entrega.

4 Para los tests unitarios también se usó H2.

- Prueba de frontend en Netlify con backend en Heroku
- Prueba del sistema accediendo desde dispositivo móvil
- Prueba de validación con 3 clientes accediendo con móvil, tablet, PC y agregado datos con Postman.
- Prueba de carga con los tres dispositivos mencionados concurrentemente, sirve de estrés⁵.
- Prueba de integración continua con Jenkins en local⁶

Nota: en la imagen se pueden ver 3 procesos uno de construcción, otro de tests y otro de producción que simbolizan las fases de compilación, tests e instalación (recuérdese que se empaqueta previamente).

En la primera pipeline ocurrió un error al no haber lanzado el contenedor Docker con la base de datos.



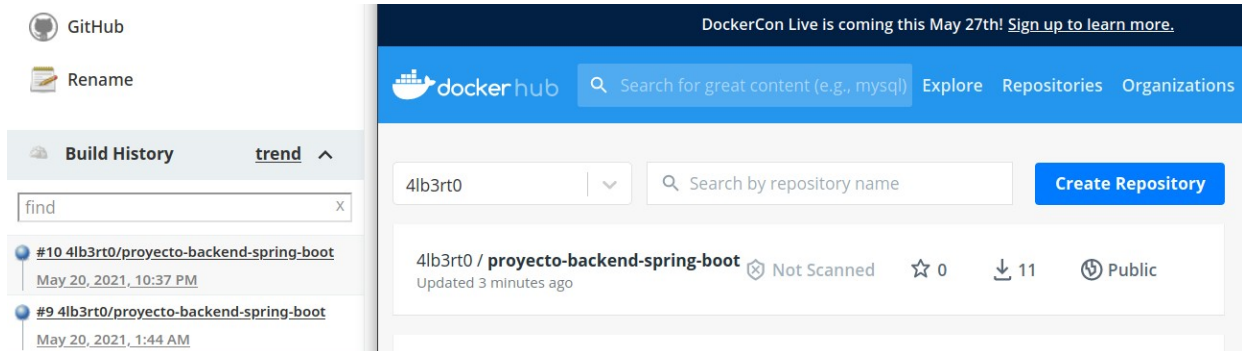
Prueba de CI con Jenkins en local

Fuente: elaboración propia

5 No sako resultados concluyentes pues no estoy seguro cuánto depende de los servicios gratuitos en los que se hizo el despliegue. No obstante, debido a la naturaleza asíncrona de la aplicación, no se observaron inconsistencias en los datos, sí en la respuesta. Habría que implementar un aviso informativo de que la transacción no fue exitosa.

6 Las dos tareas de Jenkins se activan al hacer push en el repositorio de mi cuenta en GitHub

- Pipeline con despliegue de una imagen Docker en Dockerhub⁷



Prueba de CI con Jenkins y deployment en Dockerhub

Fuente: elaboración propia

Nota: a la izquierda se ve el build #9 (hecho ayer al hacer push), y el #10 ahora. A la derecha se comprueba que se acaba de crear la imagen en el repositorio.

Despliegue en Netlify y Heroku

Como al final ha resultado que se ha hecho el despliegue en dos plataformas que no se habían considerado previamente, comentaré brevemente su razón.

Opciones de deployment

¿Una aplicación cerrada o crearla en la nube? Estas eran las dos posibilidades que había estado barajando, ambas tenían sus pros y contras. Por una parte me llamaba la atención crear una imagen que contuviera todo lo necesario como para poder ser ejecutada en local (si se hubiese instalado el software correspondiente) o en remoto si se hubiese hecho un deployment externo. Al final, por esa misma necesidad de instalación previa se optó por la nube.

⁷ La idea inicial era hacer el despliegue en AWS con EC2 y fargate, pero al solicitar tarjeta de crédito se buscó una alternativa.

AWS era el proveedor que había comentado y del me había informado. Puesto que posee una gran cantidad de servicios ofrecidos independientemente del lenguaje, su popularidad, etc. Lo veía como una alternativa interesante; pero el hecho de haber tenido que dar la tarjeta de crédito en el registro me hizo pensarlo de nuevo.

Ahí es donde me topé con Netlify y Heroku.

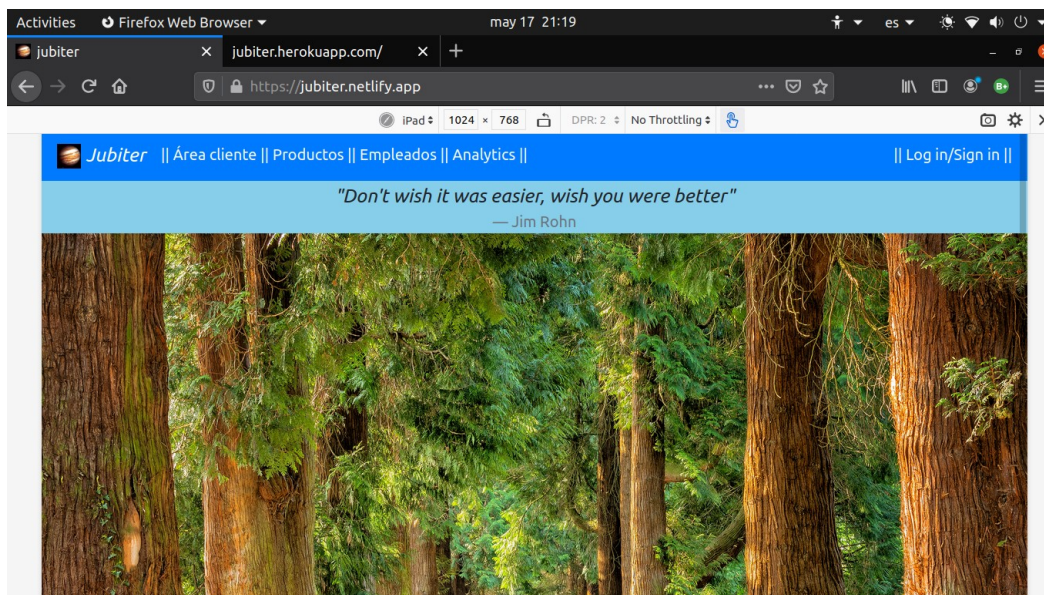
Ventajas de estas dos plataformas

Lo cierto es que son bastante conocidas en el mundo de los desarrolladores y la CI. Ambas plataformas permiten crear pipelines tipo DevOps y ofrecen adhesión con servicios de terceros. De hecho, lo tengo integrado con los repositorios de GitHub y cada vez que se hace un push en la rama main se compila automáticamente.

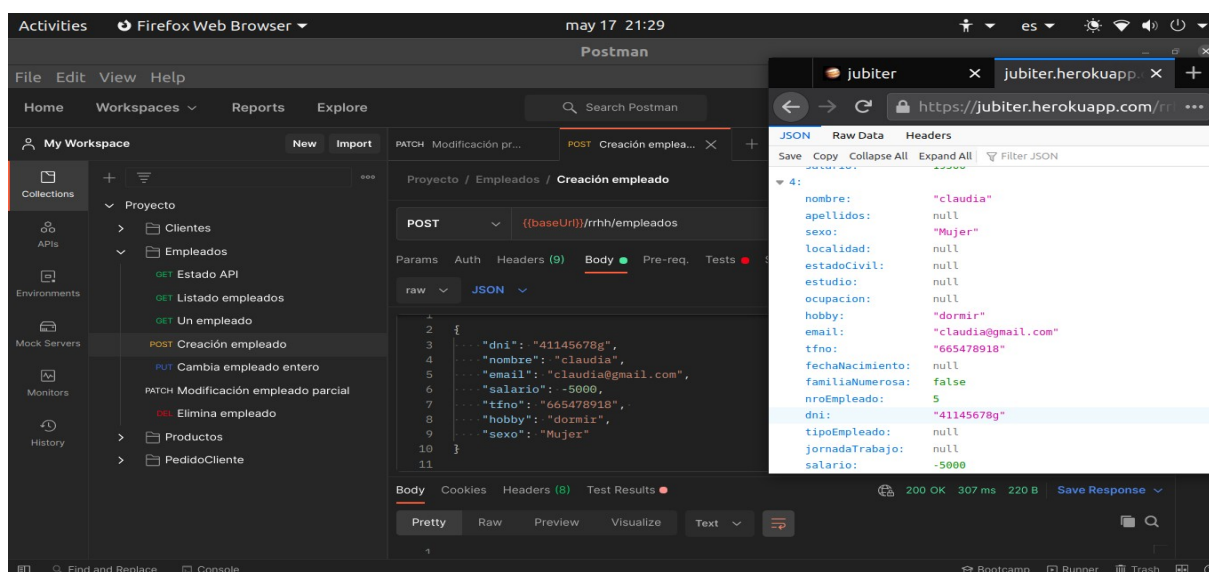
Netlify está más relacionado con el frontend y React, Heroku por su parte aloja proyectos más ‘pesados’, dispone de su CLI y está muy horientado a la virtualización (con sus dynos).

La experiencia de encontrar estas plataformas ha sido enriquecedora, pues se ha podido comprobar lo sencillo que es adaptarse a las tecnologías cuando se tiene un conocimiento básico de los conceptos informáticos actuales.

Como punto negativo estaría el hecho de que me siento más cómodo trabajando y controlando lo que pasa en cada momento. Es verdad que con estas empresas se puede aumentar el grado de control sobre la aplicación, pero su punto fuerte es el hecho de proveer ese servicio con ajustes mínimos.



**Comprobación del despliegue en la nube por parte de
ambos proveedores**
Fuente: elaboración propia



Comprobación del API con Postman
Fuente: elaboración propia

Documentación usuarios básica

Se ha pretendido buscar una funcionalidad mínima en donde se dé cabida a muy poca confusión.

Además se acompaña con ayudas informativas textuales.

Lo que sí es cierto es que debería mejorarse la reacción informativa ante incorrecciones en los datos de entrada en la IU, cosa que se trató de poner énfasis a lo largo del curso; pero que aquí decidí invertir el tiempo en conseguir otras funcionalidades primero.

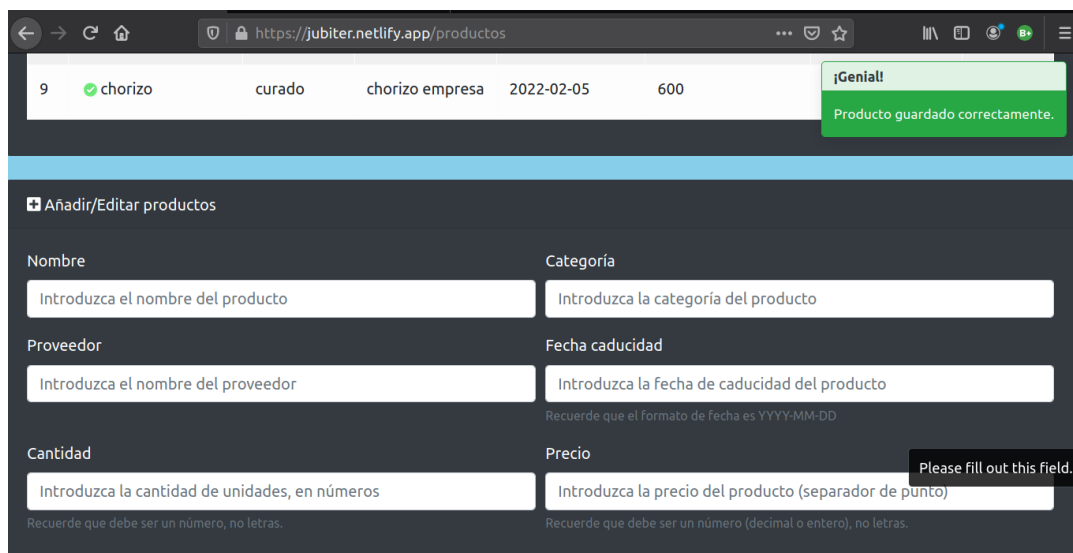
En cualquier caso, una funcionalidad común a todos sería que para entrar deben autenticarse con su usuario y contraseña (de ser posible se buscaría un doble factor).

Documentación para empleados que trabajen con productos (stock)

En esta ocasión se debería rellenar el formulario completamente siguiendo los consejos proporcionados.

Tras ello se debe pulsar el botón de guardar o la tecla ‘enter’.

Automáticamente aparecerá en la lista superior.



The screenshot shows a web browser at the URL <https://jubiter.netlify.app/productos>. At the top, there is a table with product data. Below this, a green notification box says "¡Genial! Producto guardado correctamente." The main section is titled "Añadir/Editar productos" and contains a form with the following fields:

- Nombre:** Introduce el nombre del producto
- Categoría:** Introduce la categoría del producto
- Proveedor:** Introduce el nombre del proveedor
- Fecha caducidad:** Introduce la fecha de caducidad del producto. A note below says: "Recuerde que el formato de fecha es YYYY-MM-DD".
- Cantidad:** Introduce la cantidad de unidades, en números. A note below says: "Recuerde que debe ser un número, no letras."
- Precio:** Introduce la precio del producto (separador de punto). A note below says: "Recuerde que debe ser un número (decimal o entero), no letras." There is also a tooltip that says "Please fill out this field."

Inserción de un producto en BBDD

Fuente: elaboración propia

La única opción que tendría el empleado sería eliminar un producto de la lista, para ello debería simplemente pulsar el botón rojo de eliminar.



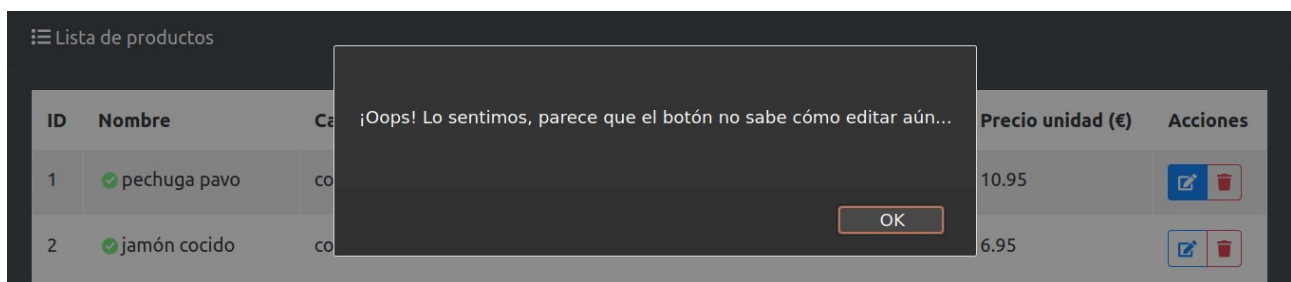
The screenshot shows a web browser at the URL <https://jubiter.netlify.app/productos>. A red notification box at the top right says "¡Genial! Producto eliminado correctamente." Below this is a table titled "Lista de productos".

ID	Nombre	Categoría	Proveedor	Fecha caducidad	Cantidad producto	Precio unidad (€)	Acciones
1	✓ pechuga pavo	cocido	Frial	2021-08-01	100	10.95	 
2	✓ jamón cocido	cocido	Campofrío	2021-07-10	100	6.95	 
4	✓ jamón serrano DO	curado	Montesierra	2021-11-01	120	35	 

Eliminación de un producto de la BBDD

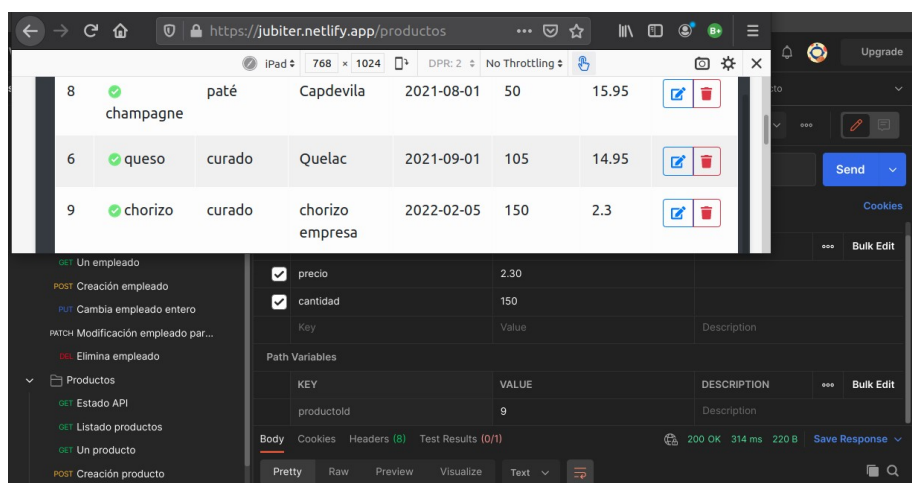
Fuente: elaboración propia

Para el botón de edición, simplemente se manda un mensaje indicando que no está implementada esa funcionalidad. Pero sabemos que es accesible desde un cliente HTTP como Postman o cURL.



Acción de edición de un producto de la BBDD

Fuente: elaboración propia



Patch desde Postman
Fuente: elaboración propia

Documentación para los clientes que hacen pedidos

En esta ocasión, tras registrarse se iría a parar a la pantalla personal de cada uno. Donde se recibiría un mensaje de bienvenida.

Debajo de este se encuentra una lista filtrada de productos, en donde se podrá obtener una visión general del estado de cada producto. Así, podrá simplemente insertar en el formulario el ID del producto que desea y la cantidad.

Tras ello, se pulsaría guardar y el producto se restaría automáticamente.

The image shows a web application interface. At the top, there is a table with product information: ID (6), name (queso), quantity (-15), and price (14.95). Below the table is a section titled 'Pedir producto'. It contains a form with the following fields: 'Número de socio' (with value 7), 'Email' (with value cervantes@hotmail.com), 'ID pedido cliente' (with value b34d2d45-4aaa-44ec-a506-ba9863b0fa70), 'ID producto:' (with a dropdown menu), and 'Cantidad' (with a text input field). There are also buttons for 'Introduzca la cantidad de unidad' and 'Recuerde que debe ser un número, no letras.'

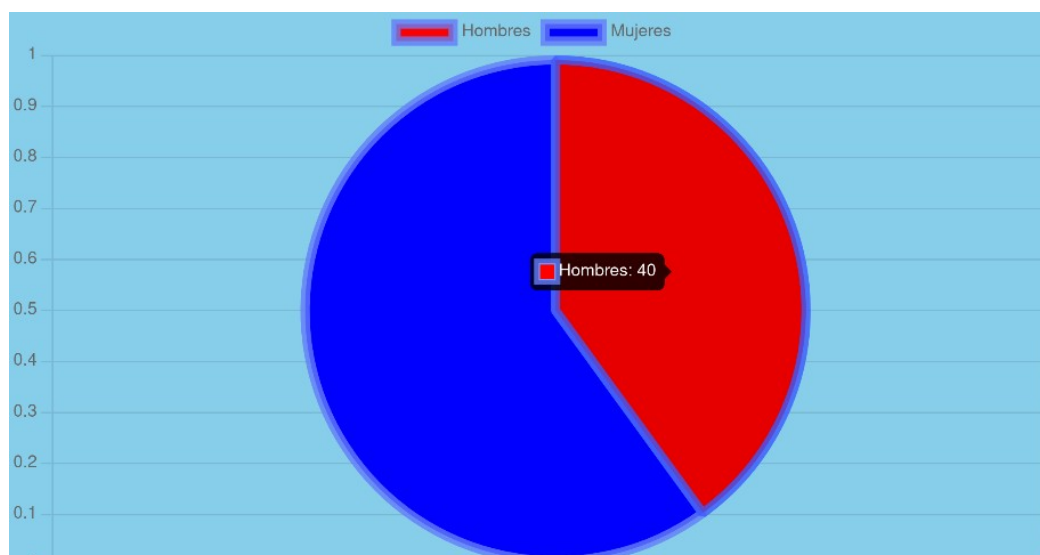
Sólo dos campos que permiten inputs
Fuente: elaboración propia

Nota aparte: como mejora quedaría que la inserción de un número negativo no sea posible (puesto que suma) y que en vez de introducir el ID se usara una lista desplegable con el nombre del producto.

Documentación para consumidores de información del negocio

En este caso tan sólo deben entrar para visualizar el cuadro de mando en near-real-time.

La acción que podría realizar es modificar el número de variables pinchando sobre la leyenda, es decir, si se pincha desaparecería esa información. Si se vuelve a clicar, aparecería.



Queda corregido el bug hombre/mujer de la anterior entrega

Fuente: elaboración propia

Otra característica de estos gráficos de la librería Chart.js es que se proporciona información al situarse encima de ellos (efecto hover), lo cual proporciona información inclusiva.

En cuanto a los gráficos de barras, es interesante poder contar con una visión general que haga uso de un rango negativo.

Especialmente útil puede ser aumentar/reducir el número de variables para ir comparándolas y sacar conclusiones más enriquecedoras.

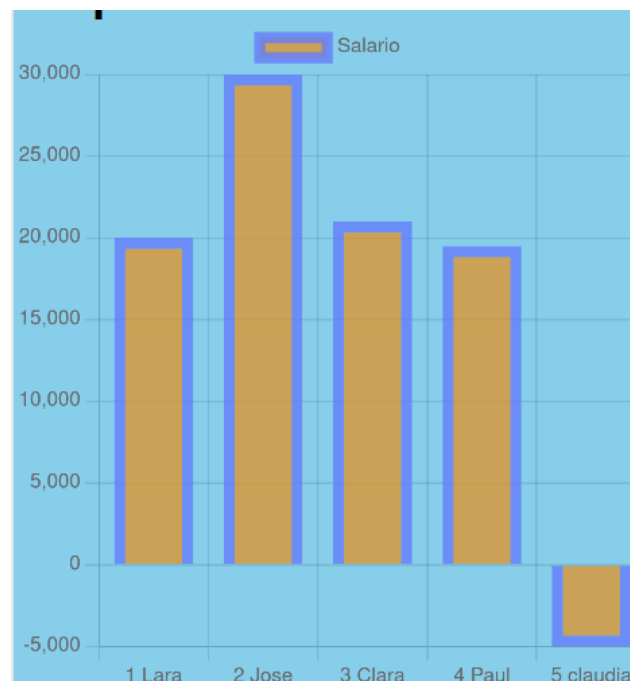


Gráfico de barras actualizado con inserción empleado

Fuente: elaboración propia

Información funcional para los técnicos y desarrolladores, recurso multimedia:

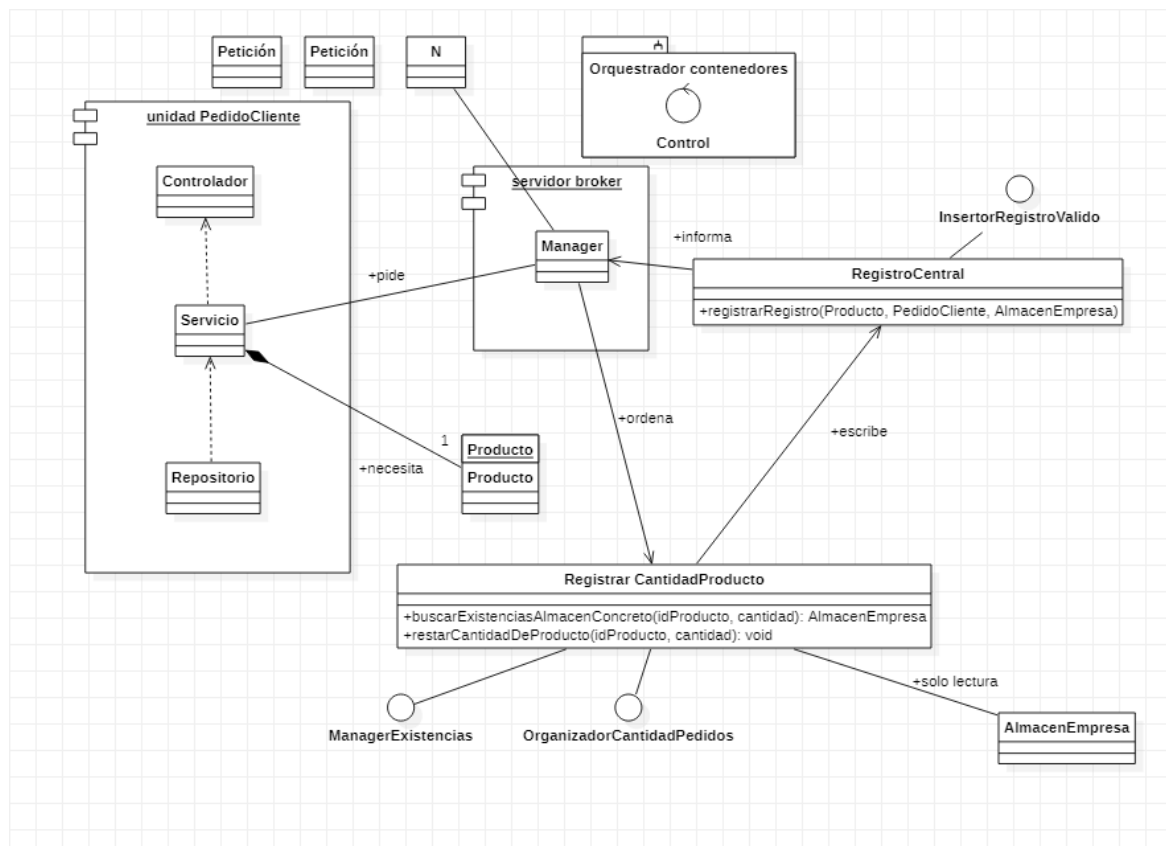
Este apartado que fue utilizado para mostrar la funcionalidad en la fase de desarrollo, sirve de ejemplo para mostrar la funcionalidad del aplicativo y entender su trayectoria⁸.

<https://www.youtube.com/watch?v=CZgwAQIV6dc>

⁸ Se comprueba que no se ha ampliado la funcionalidad de la aplicación y que sólo se han añadido las clases de tests del backend. El resto sólo hacía falta el tiempo para documentarlo brevemente.

Anexo

Explicación del funcionamiento conceptual del super-sistema. Tridimensionalidad, asincronía y concurrencia.



Explicación funcional de relaciones entre componente mediante diagrama UML

Fuente: elaboración propia

Este esquema modelado pretende explicitar tanto la estructura como el dinamismo en el sistema.

El mismo podría interpretarse de la siguiente manera:

Dado un conjunto de peticiones emitidas asincrónicamente, existirá un componente organizador del paso de mensajes (broker tipo RabbitMQ) que enrutará la petición con su destino particular.

Además llevará cuenta de cada tarea que se esté realizando.

Como puede verse sobrepasado por la gran cantidad de mensajes, existe una figura controladora del número de instancias de estos contenedores. Por eso se ha tratado de señalar como un nodo.

El componente de la izquierda va a representar cada servicio de esta arquitectura. Concretamente se ejemplifica el servicio implementado en esta aplicación: el conjunto de PedidoCliente. Mediante asociaciones se pretende mostrar qué enlaces existen entre ellos.

El nombre en las relaciones acompaña a la navegabilidad. Sin embargo, se ha pretendido dejar lo suficientemente flexible como para poder ser modificado según se necesite en cada apartado.

Por ejemplo, se ha presentado una composición del servicio PedidoCliente con Producto por haberse implementado así en código, pero lo ideal sería ir más en la línea del paso de mensajes.

Se ha obviado la base de datos, pero podría ser una para el subconjunto dado, o que se relacione con un apartado de una base de datos centralizada,

Se puede ver, a su vez, el poder de absorción que tiene la relación ternaria en esta parte del sistema (Producto – PedidoCliente – AlmacenEmpresa), llegando incluso a ser lógico plantearse la concurrencia como un hecho natural.

Repositorios o fuentes de información:

- GitHub:

Backend → <https://github.com/NR4l3rt0/proyecto-backend-spring-boot>

Frontend → <https://github.com/NR4l3rt0/proyecto-frontend-react>

Históricos de commit, y scripts (sql y shell) → <https://github.com/NR4l3rt0/proyecto>

- DockerHub:

<https://hub.docker.com/repository/docker/4lb3rt0/proyecto-backend-spring-boot>

- Youtube (muestra funcional técnica):

<https://www.youtube.com/watch?v=CZgwAQIV6dc>

- Enlace a web frontend en Netlify:

<https://jubiter.netlify.app/>

- Enlace a web backend en Heroku:

<https://jubiter.herokuapp.com/>

Referencias para esta etapa

Canales de youtube:

Amigoscode.

Java Techie.

Sobre testing y otros conceptos en la elaboración de software:

<https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-005-software-construction-spring-2016/readings/>