DESARROLLO DE SOFTWARE EN SISTEMAS DISTRIBUIDOS

Grupo 16 – Informe Final

Integrantes
Agustín Marcelo De Luca
Rodrigo Minaberrigaray
Rodrigo Pait

Introducción

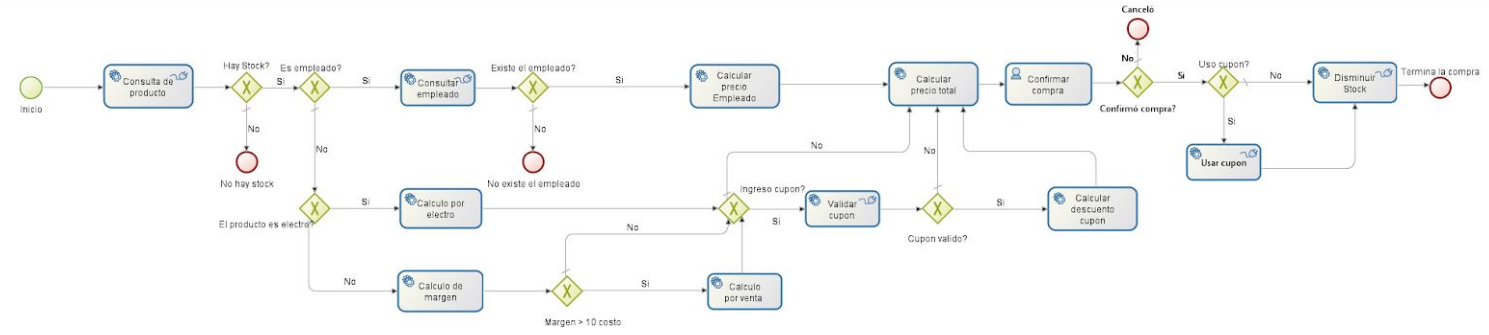
El objetivo primordial del trabajo se basó en la implementación de un sistema que, junto a 3 web services y un modelo de negocio diseñado e implementado a través del software Bonita Open Solutions, constituyen una solución distribuida para satisfacer la necesidad de un supermercado de facilitar a sus clientes la compra online.

Como punto de entrada, la página web provee un home con todos los productos disponibles para la compra, permitiendo a su vez al usuario la opción de iniciar sesión, donde en caso de seleccionar esta última brinda una pantalla de login para verificar que efectivamente se trate de un empleado del supermercado. El sitio permite seleccionar un producto para la compra, ingresar un cupón de descuento en caso de no encontrarse logueado como empleado, consultar el precio final con los descuentos correspondientes, y finalmente ejecutar la compra.

Por último, a nivel de arquitectura se posee:

- API de productos: Provee toda la información utilizada por el sistema para mostrar los diversos productos, así un conjunto de métodos que van a ser utilizados por el orquestador para obtener cierta información del producto que el usuario del sistema desea comprar y, caso de confirmarse la compra, disminuir el stock del mismo.
- API de RRHH: Almacena la información referente a los empleados del supermercado, y que se utiliza a fines de verificar la validez de los datos ingresados por un empleado a la hora de iniciar sesión en el sistema.
- API de cupones: Permite verificar la existencia o no de un cupón de descuento ingresado por el usuario del sistema, así como, en caso de existir, verificar su validez y de confirmarse la compra marcarlo como usado.
- Proceso diseñado con Bonita Open Solution: Actúa como orquestador de las 3 APIs previamente mencionadas para, en base a las reglas de negocio establecidas por la gerencia de la compañía, calcular el precio final de venta de un producto en cuestión.

Proceso para el cálculo del precio de venta



[Link a la imagen completa]

El proceso realizado con el modelador BPM Bonita Open Solution describe el proceso de compra de un producto orquestando con diferentes Web Services ofrecidos por el supermercado.

En principio se inicia una instancia (caso) del proceso desde el sistema web del supermercado, el cual envía como parámetro de entrada tres valores (producto_id, empleado_id y cupon), luego de esto el orquestador Bonita realiza un llamado a la API (Interfaz de programación de aplicaciones) de productos denominada “stock_api” a la cual se ejecuta un método GET “product/{producto_id}” para obtener la información completa del producto enviado como parámetro de entrada en la instanciación. En este punto se obtiene el precio de costo, precio de venta, tipo y stock del producto; en consecuente se realiza una validación de cantidad de stock y si dicha validación dictamina que se encuentra stock disponible para realizar la compra sigue su flujo, en caso contrario se termina dicha instancia.
El próximo paso a realizar por el orquestador es validar si parámetro de entrada “empleado_id” es distinto o igual a 0 (cero) en este momento se divide el flujo en dos:

- Si el parámetro ingresado es igual a cero se bifurca nuevamente gracias a una nueva validación pero esta vez con respecto al parámetro “producto_id”, más específicamente al tipo de producto:

    - En caso de que el producto sea de tipo “electro” se aplica la regla de negocio correspondiente establecida por el área comercial del supermercado.

    - En el caso de que el producto sea distinto del tipo “electro” se efectúa el cálculo correspondiente a la regla de negocio establecida por el área comercial.

  Luego de dicha bifurcación por el tipo de producto se realiza una nueva validación en la cual se comprueba el tercer parámetro de entrada “cupon”, dicho paso consta de la verificación de validez del cupon llamando a la API denominada “cupon_slim” almacenada en el servidor heroku mediante el método “coupon/{cupon}/isValid”, y en caso de que este sea válido se aplica la regla de negocio establecida por el área comercial del supermercado. Caso contrario no se aplica la regla de negocio.

- Si el parámetro ingresado es distinto de cero se realiza un llamado a la API denominada “rrhh_api” en el cual se realiza el método GET “employee/isValid?id={empleado_id}” para realizar la verificación de que dicho empleado ingresado es realmente un empleado válido. En caso satisfactorio se realiza el descuento correspondiente indicado por la regla de negocio del área comercial del supermercado, caso contrario de que el empleado no sea válido la instancia se termina.

En este momento se realiza una tarea manual en la cual gracias al orquestador y el sistema web del supermercado se discrimina el subtotal, descuento aplicado, y precio final para que el usuario final indique si desea realizar la compra. Para finalizar dicho proceso en caso de que el usuario final indique que desea efectuar la compra, se marca como utilizado al cupon mediante el método GET “coupon/{cupon}/use” de la API “cupon_slim” y se disminuye el stock del producto de mediante el método GET “product/{producto_id}/reduceQuantity” de la API “stock_api”.

Servicios web

Consideraciones generales respecto a los servicios web

Para los servicios web utilizamos dos tecnologías distintas:

- Por un lado, para los servicios de stock y empleados, se creó una API REST con Symfony 3.4. Ambos proyectos corren en localhost y tienen cada uno su propia base de datos MySQL.
- Por otro lado, la API REST de cupones se desarrolló con Slim 3 y está alojada en el servicio Cloud Heroku. Cuenta con una base de datos PostgreSQL.

API de stock de productos

Nombre del endpoint	Descripción	Parámetros
POST /product	Da de alta un producto y retorna la entidad creada.	- name: nombre del producto. Requerido. - cost_price: precio de costo del producto. Requerido. - sale_price: precio de venta del producto. Requerido. - product_type_id: ID del tipo de producto.
GET /product	Recupera todos los productos.	
GET /product/{id}	Recupera el producto con el id enviado en la URL.	
GET /product/{id}/reduceQuantity	Si el producto solicitado existe y tiene stock, disminuye en 1 dicha cantidad. Retorna el producto.	
POST /productType	Da de alta un tipo de producto y retorna la entidad creada.	- initials: iniciales del tipo de producto. Requerido. - description: descripción del tipo de producto. Requerido.
GET /productType	Recupera todos los tipos de producto.	
GET /productType/{id}	Recupera el tipo de producto con el id enviado en la URL.	

API de RRHH

Nombre del endpoint	Descripción	Parámetros
POST /employee	Da de alta un empleado y retorna la entidad creada.	- fistname: nombre del empleado. Requerido. - surname: apellido del empleado. Requerido. - email: email del empleado. Requerido. - password: contraseña del empleado. - employee_type_id: ID del tipo de empleado. Requerido.
GET /employee	Recupera todos los empleados.	
GET /employee/{id}	Recupera el empleado con el id enviado en la URL.	
GET /employee/find	Busca un empleado y lo devuelve si lo encuentra.	- email: email del empleado a buscar. Requerido.
GET /employee/isValid	Busca un empleado por id, devolviendo un booleano que indica si el mismo existe o no.	- id: ID del empleado. Requerido.
POST /employeeType	Da de alta un tipo de empleado y retorna la entidad creada.	- initials: iniciales del tipo de empleado. Requerido. - description: descripción del tipo de empleado. Requerido.
GET /employeeType	Recupera todos los tipos de empleado.	
GET /employeeType/{id}	Recupera el tipo de empleado con el id enviado en la URL.	

API de cupones

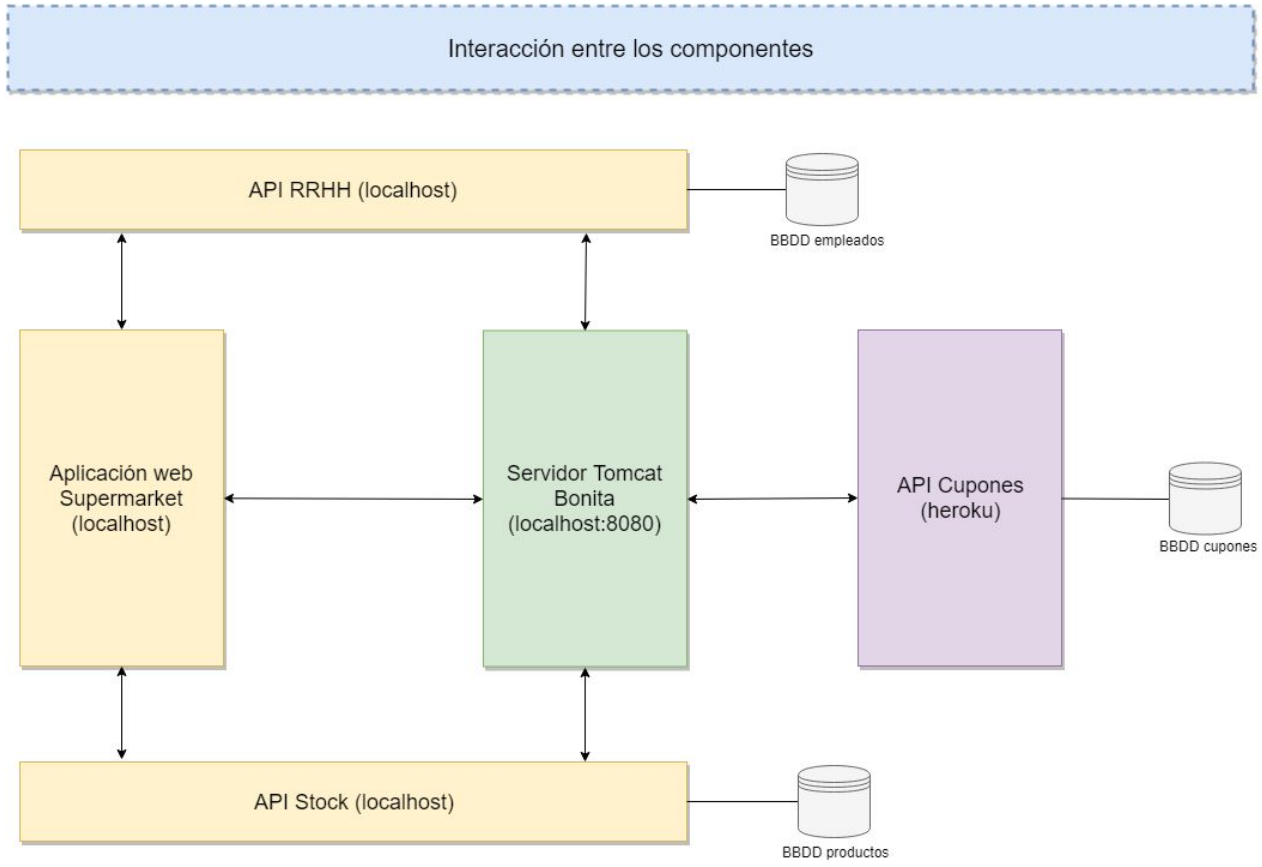
Nombre del endpoint	Descripción	Parámetros
GET /coupon	Recupera todos los cupones de descuento.	
GET /coupon/unuseds	Recupera todos los cupones de descuento que aún no fueron utilizados.	
GET /coupon/{cod}	Recupera el cupón con el código enviado en la URL.	
GET /coupon/{cod}/isValid	Busca el cupón con el código indicado, y devuelve si el mismo es válido o no.	
POST /coupon	Da de alta un cupón y retorna la entidad creada.	- initials: iniciales del tipo de producto. Requerido. - description: descripción del tipo de producto. Requerido.
POST /coupon/use	Marca un cupón como usado.	- cod: código del cupón
POST /coupon/unuse	Marca un cupón como no usado.	- cod: código del cupón.
POST /coupon/unuseAll	Marca todos los cupones como no usados.	
POST /coupon/delete/{cod}	Elimina el cupón con el código enviado en la URL.	
GET /coupon/{cod}/use	Marca el cupón indicado en la URL como usado.	
GET /coupon/{cod}/unuse	Marca el cupón indicado en la URL como no usado.	

Descripción de la solución

Para la aplicación de Supermarket se creó un proyecto Symfony 3.4. En rasgos generales, cuenta con 3 grandes funcionalidades:

1. Iniciar sesión en el sistema: Los empleados de Supermarket pueden utilizar sus credenciales para acceder al sistema y poder comprar los productos al precio de costo.
2. Listar los productos: En la página de inicio se muestra un listado de productos, mostrando su precio de lista y el stock disponible. Los productos que no tienen stock disponible no se muestran en el listado.
3. Realizar la compra de un producto: El usuario puede iniciar el proceso de compra de un producto de los que se muestran en el listado. Antes de iniciar el proceso de compra, se le da la opción al usuario de que ingrese un cupón de descuento, y luego se lo redirige una nueva página donde podrá ver el precio final del producto una vez aplicados los descuentos correspondientes. Allí podrá confirmar o cancelar su compra.

A continuación se presenta un diagrama que muestra cómo interactúan los distintos componentes del sistema.



Las interacciones que se muestran en el diagrama se dan en los siguientes contextos, relacionados con las funcionalidades descritas anteriormente:

1. Durante el proceso de login, la aplicación web Supermarket realiza una petición a la API de RRHH (GET /employee/find), enviándole el email ingresado por el usuario. Dicho método busca el empleado solicitado y lo devuelve a la aplicación si lo encuentra. Por motivos de simplificación, la API devuelve el empleado completo, incluida la contraseña para que la aplicación finalmente chequee las credenciales ingresadas en el formulario de login. Si los datos ingresados son correctos, se crea la sesión del usuario y se accede al sistema como empleado.
2. En la página de inicio, la aplicación web Supermarket realiza una petición a la API de Stock (GET /product) para obtener el listado de productos que se muestra en dicha página.
3. El proceso de la compra de un producto está orquestado con el BPM Bonita Open Solution. El proceso permite realizar la compra de un único producto y una única unidad al mismo tiempo. Para eso, se realizan múltiples llamados desde la aplicación web Supermarket a la API de Bonita:
    a. Se llama al método **POST /bonita/loginservice** para obtener un token de acceso. Dicho token se setea en una cookie y se utilizará para enviarlo también en los headers de cada uno de los requerimientos posteriores.
    b. Se llama al método **GET /bonita/API/bpm/process** para obtener el ID del proceso de Bonita.
    c. Se llama al método **POST /bonita/API/bpm/case** para crear una instancia del proceso e iniciarlo, enviándole el ID del producto a comprar, el ID del empleado logueado (o un 0 en caso de no estar logueado) y el cupón ingresado por el usuario. El proceso de Bonita durante su ejecución, invoca el método **GET /employee/isValid** de la API de RRHH para verificar si el empleado recibido es válido. También verifica que haya stock disponible para el producto solicitado, invocando al método **GET /product/{id}** de la API de Stock.
    d. Se llama al método **GET /bonita/API/bpm/case/{caseId}** para obtener el ID asignado al usuario que inició el proceso.
    e. Se llama repetidas veces al método **GET /bonita/API/bpm/caseVariable/{caseId}/precioTotal** hasta que dicha variable deja de ser 0. Esto significa que se terminó de calcular el precio final de la compra.
    f. Se llama al método **GET /bonita/API/bpm/caseVariable/{caseId}/cuponValido** para saber si el cupón que se ingresó fue correcto e informárselo al usuario. Para obtener esta información, el proceso de Bonita en algún punto invocó al método **GET /coupon/{cod}/isValid** de la API de cupones.

En este punto ya se tiene calculado el precio final y se redirige al usuario a la página de confirmación de compra, donde se le muestra la información correspondiente (si el cupón fue válido, el precio final, el descuento aplicado). Cuando el usuario confirme o cancele la compra se realizan además los siguientes llamados:

    g. Se llama nuevamente al método **POST /bonita/loginservice** para obtener un token de acceso. Esto se debe a que en el redireccionamiento de página se pierde una cookie correspondiente al token, y tuvimos que volver a llamar a este método para recuperarla.

h. Se llama al método **GET /bonita/API/bpm/task?c=10&p=0&f=processId={processId}&o=state** para obtener el ID de la tarea manual que frenó el proceso.
i. Si el usuario confirmó la compra, se llama al método **PUT /bonita/API/bpm/caseVariable/{caseId}/confirmacionCompra** para setear dicha variable en true, indicando que el se confirmó la compra. Al confirmarse la compra, el proceso de Bonita invoca dos métodos: por un lado realiza un llamado a **GET /coupon/{cod}/use** de la API de cupones (en caso de que se haya brindado un cupón válido) para marcar dicho cupón como usado; por otro lado, realiza un llamado a **GET /product/{id}/reduceQuantity** de la API de Stock para reducir en 1 el stock del producto comprado.
j. Tanto en caso de confirmación como de cancelación de la compra, se llama al método **PUT /bonita/API/bpm/userTask/{taskId}** para indicar que se completó la tarea manual.
k. Se llama repetidas veces al método **GET /bonita/API/bpm/archivedCase?o=id&f=sourceObjectId={caseId}** esperando que finalice el caso y así obtener el estado con el que finalizó.

Una vez finalizado este proceso, se redirige al usuario a la página de inicio (listado de productos).