UNIVERSIDAD ORT URUGUAY

Facultad de Ingeniería

Envíos Ya

Obligatorio 2
Arquitectura de software en la práctica

María Inés Fernández - 186503
Martín González - 146507

2017

Índice

1. Descripción general del trabajo

El sistema fue desarrollado para la empresa Envios Ya, que quiere revolucionar el mercado de los envíos de paquetes, debido a que hoy dia es un proceso que consume mucho tiempo, el costoso y tedioso. La idea fue aplicar un modelo de economía colaborativa, ofreciendo un servicio fácil de usar a través de un sistema web moderno.

En esta segunda entrega nos nos focalizamos en dos grandes aspectos. Por un lado, mejorar el sistema ya entregado. Hicimos un proceso de refactoreo donde se logró mejorar la mantenibilidad del código fuente. Por otra parte, dividimos nuestro sistema en microservicios como es pedido para hacerlo mas escalable. Se separó en 3 aplicaciones diferentes:

- Aplicación principal
- Microservicio de costos
- Microservicio de log

Repositorios

Envios- ya: *https://github.com/martin27ago/envisoya.git*

CostsService*: https://github.com/mariainesfgp/CostsService.git*

LoggerService*: https://github.com/mariainesfgp/LoggerService.git*

Urls bluemix

Envios-ya *: http://enviosya-arq-gf-app.mybluemix.net*

CostsService: *http://apicostos-arq-GF-app.mybluemix.net*

LoggerService: *http://logger-arq-GF.mybluemix.net*

2. Descripción de la arquitectura

2.1. Division en microservicios

Se decidió dividir el sistema en 3 proyectos diferentes. El proyecto original y luego los microservicios de costos y logs.

Decidimos extraer los costos a un microservicio, ya que utiliza una api externa, también utilizada una base de datos con cierta complejidad y lllamadas a la api cada 10 minutos. Al abstraerlo a un microservicio podemos desacoplar la lógica de calcular el costo con las funcionalidades principales y también ganar extensibilidad, el dia de mañana si se quiere modificar la forma en que se calculan los costos no es necesario modificar la aplicación principal.

El segundo microservicio es muy simple y solamente se encarga de logear. Para ello tiene un controlador con un método que es llamado mediante http por las apis externas. La funcionalidad de este método es persistir la información que llega en la base de datos.

2.2. Framework

Para el desarrollo de la arquitectura se utilizo Ruby on Rails, este framework de por sí crea una arquitectura propia que es la misma para todas las aplicaciones, basada en el modelo: Modelo, Controladores y Vistas.

A continuacion se describen las diferentes aplicaciones:

2.3. Envios Ya

2.4. Modelos

A continuación se lista modelos del sistema y su utilidad.

User: Representa a los usuarios del sistema, en este caso los que realizan los pedidos. Guarda información básica de la persona, incluyendo una foto de perfil. Nuestra aplicación se integra con Facebook para realizar tanto el registro como el iniciar sesión de un usuario si es deseado. Para subir archivos(images/pdf) utilizamos la gema Paperclips.

Dentro de los usuarios se agrego un usuario adminisitrador que se encarga de activar a los cadetes que se registran en la aplicación.

Delivery: Representa a los cadetes del sistema, los que se encargan de entregar los pedidos. Además de la información básica de la persona, se guarda la licencia de conducir de los mismos y la documentación del vehículo en un pdf. También es necesario guardar la ubicación geográfica del mismo.

Shipping: El pedido guarda información de dirección de dirección de origen y destino, usuario que lo emite, cadete responsable y peso . También tiene costo del mismo que se calcula mediante una API externa que provee información de costo por kilo y costo entre

zonas. Para calcular el mismo se guarda en la base de datos información de la API externa, la información se actualiza cada 10 minutos.

Discount: Guarda información de un descuento, esto incluye para quien es, si fue enviado por otra persona, el porcentaje, y booleanos que indican si esta activo y usado. Fue necesario crear estas últimas variables para manejar la lógica de generación de descuentos que se plantea en el RF5. Donde un usuario genera un 50% de descuento si le envía un pedido al otro que no esta registrado en el sistema, y el mismo se registra y realiza un pedido teniendo también un 50% de descuento. Para hacer un sistema mas extensible, ya que luego seguramente se puedan generar descuentos de otra forma, se realizo una entidad separada.

2.5. Controladores

UsersController: Manejo de usuarios, tanto darlos de alta como modificación.

DeliveriesController: Manejo de cadetes, alta y modificación. También se encarga de activar un cadete, esta acción solo la puede realizar el administrador.

ShippingsController: Se encarga de crear pedidos, finalizarlos y calcular el costo de los mismos.

SessionsController: Manejo de las sesiones del sistema. Se encarga de crear sesiones cuando un usuario ingresa tanto por Facebook o con usuario y contraseña y destruirlas cuando el mismo cierra sesión.

ApplicationContoller: Se encarga de exponer el endpoint de healthcheck que pide, y devolver los cadetes en formato de json. También tiene 2 métodos muy útiles que se utilizan en los otros controlados que devuelven que usuario esta logeado al sistema mediante el session.

2.6. Jobs

Para mejorar la performance del sistema, algunas tareas se realizan de forma paralela en un worker independiente. Se usaron las tecnologías de Redis y Sidekiq para lograr esto.

SendMailJob: Cada 7 minutos, recorre la lista de pedidos finalizados y actualiza los costos que de aquellos que no estan actualizados y le envia un email de recepcion al usuario que emitio el pedido.

2.7. Microservicio Costos

2.8. Modelos

Zone: Guarda el nombre de todas las zonas de Montevideo, cada una con su id, y polígono correspondiente. Esta información se obtiene de la API provista.

Cost: Se encarga de guardar la información de el costo por kilo de un paquete y actualizar esta información.

CostZones: Guarda una matriz con todas las combinaciones entre las distintas zonas y los 3 últimos costos de cada combinación.

2.9. Controladores

CostsController: Se encarga de las funcionalidades de costos. Este controlador es llamado por la aplicación enviosya. Tiene un método que calcula el costo, y lo devuelve también otorgando la información de si el costo esta actualizado o es un estimativo. Luego tiene el método isUpdated que simplemente devuelve si la base de datos está actualizada o no.

2.10. Jobs

UpdateCostJob: Cada 10 minutos realiza una llamada a la API para actualizar los costos por zona y por peso de los pedidos.

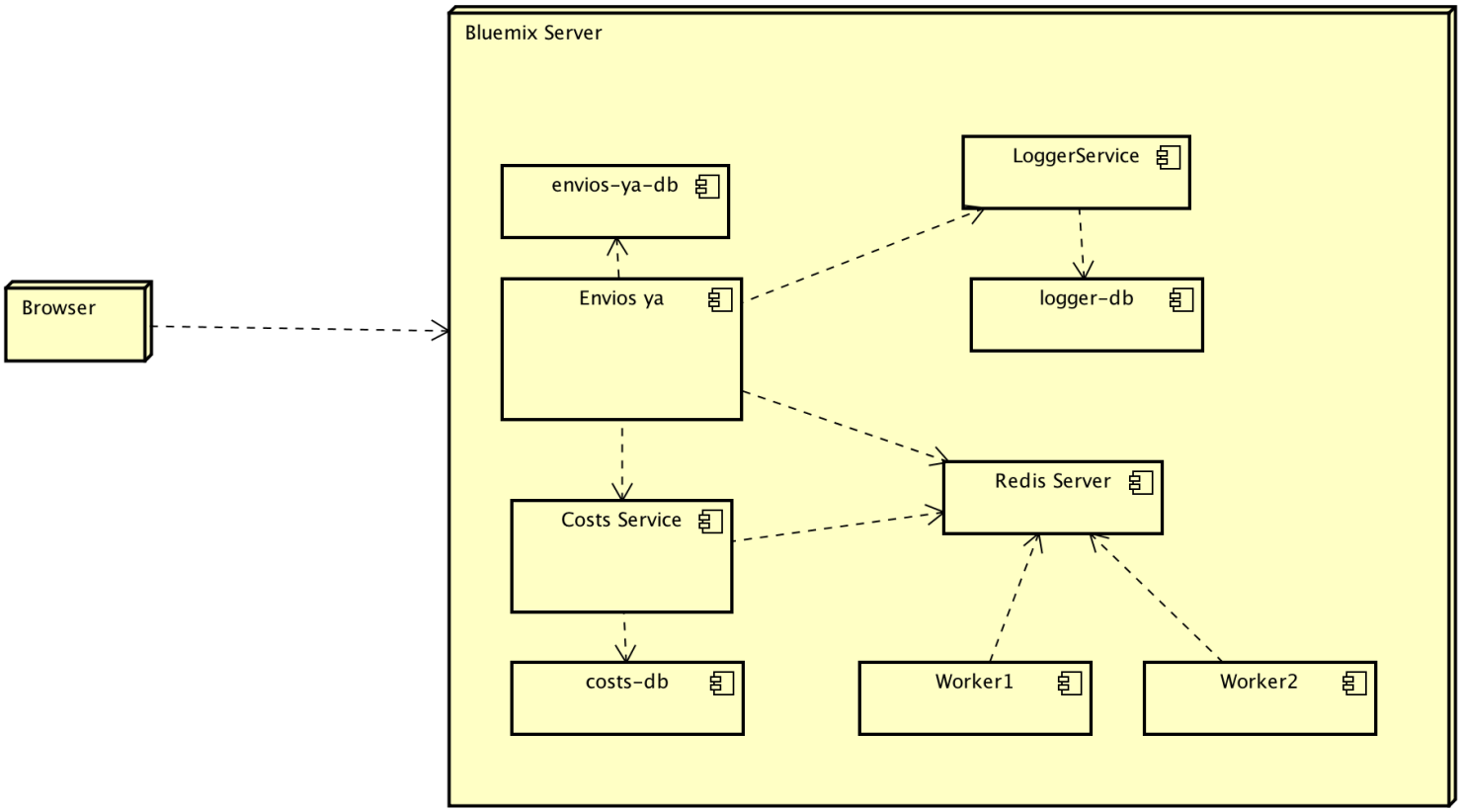
Microservicio Log

Modelos

LogHelper: Se encarga del manejo del logeo del sistema. Esta funcionalidad es llamada por la aplicación envíos ya y se encarga de guardar en la base de datos la información del log: fecha, tipo y descripción.

2.11. Diagrama de deploy

Para deployar la aplicación utilizamos el servidor bluemix. Por un lado se deployó la aplicación Envios ya con su base de datos. Luego se deployaron por separado los dos microservicios con sus respectivas bases de datos. También fue necesario tener un servidor redis con los workers para poder realizar los Jobs cada determinado tiempo.



3. Datos de prueba

Para poder utilizar la aplicación en producción se generaron datos de prueba haciendo un seed en la base de datos con la siguiente información:

Usuarios:

Name	Surname	Email	Password	Document
Admin	Admin	admin@admin.com	admin	123456789
Maria Ines	Fernandez	mariainesfgp@gmail.com	123	47477733
Martin	Gonzalez	mg27ago@gmail.com	123	43934600

Cadetes:

Name	Surname	Email	Password	Document
Maria	Cadete	mariainesfgp@gmail.com	123	47477733
Martin	Cadete	mg27ago@gmail.com	123	17332020

4. Justificacion del diseño

Para cumplir con los requerimientos no funcionales se tomaron ciertas decisiones de diseño que se detallan a continuación.

4.1. Performance

Para que los tiempos de respuesta sean rápidos, se realizan algunas acciones de forma asincrónica. Por ejemplo, cuando se confirma un pedido, se debe enviar un mail de confirmación. Todos los emails los enviamos de forma asincrónica, utilizando 'deliver_later' de Rails, que envía los emails a una cola para luego consumirlos. De esta manera, no se ve afectado el tiempo de respuesta.

4.2. Confiabilidad y disponibilidad

Se creo un endpoint donde se verifica la disponibilidad del sistema. Dentro del mismo lo que verifica es si la base de datos está activa. También se llama a la api de costos para verificar que esté levantada. Para lograr esto se realiza dentro del método del controlador una llamada simple y poco costosa a la base de datos para verificar que la misma esté levantada. Éste endpoint contesta 200 OK si es servicio esta funcionando como es esperado.

Endpoint: '/application/heathcheck'

4.3. Configuración y manejo de secretos

En esta segunda entrega, pudimos mejorar aspectos que antes habían quedado sin terminar. Por un lado, se encriptaron las contraseñas en la base de datos para lograr una mayor seguridad. También se extrajeron todas las variables configurables, como credenciales, uris a apis, etc. Para lograr esto, se creo en la carpeta initializers, una clase *attributes_env.rb* donde

se guardan todas las variables de entorno. Luego se referencias de otras partes del código utilizando ENV:

4.4. Autenticación y autorización

Para la autenticación, tanto los usuarios como cadetes pueden utilizar Facebook para registrarse y loggease. Utilizamos la gema OmniAuth Facebook para hacer la integración. El usar una API externa de un sistema tan confiable como Facebook hace que la aplicación sea segura,

La autorización de un usuario del sistema a cierta funcionalidad se logró mediante sesiones y métodos de verificación antes de cada acción. Cuando un usuario ingresa al sistema se guarda una sesión con su id. En los controladores de usuario, cadete y envió, antes de realizar cualquier acción se verifica con la sesión si el usuario tiene los permisos necesarios para realizar la misma(si es cadete, usuario o administrador). Esto se realizo mediante un método de Rails 'before_action'.

4.5. Seguridad

La comunicación entre el front-end y back-end de la aplicación se realiza de manera segura ya que estamos utilizando el framework de Ruby on Rails que soluciona ese problema.

Para que el sistema sea seguro, se verifica antes de cada llamada que el usuario logeado tenga permisos para acceder a la acción que quiere. De no ser así se redirige a su pagina principal. Realizando esto se logra no dejar ningún end-point del sistema expuesto.

4.6. Código fuente

El sistema fue desarrollado con el lenguaje Ruby utilizando el framework Ruby on Rails. Se siguieron estándares de clean code para lograr un código mantenible. Debido a la falta de tiempo y el hecho de que estamos aprendiendo un lenguaje nuevo de programación, todavía el código tiene aspectos a mejorar y refactorear. Esto se va a poder mejorar esto para la segunda entrega.

Para el manejo de versiones se utilizó un repositorio en Github y Gitflow para facilitar el manejo de branches.

4.7. Obtener información de los cadetes en formato JSON

Para cumplir con este requerimiento se creo un endpoint en nuestra webApi que devuelve todos los cadetes en formato json.

Endpoint : /application/deliveriesjson

4.8. Pruebas

Para esta segunda entrega realizamos pruebas con la herramienta JMeter. Las distintos escenarios probados fueron:

- Página principal
- Login
- Formulario de envío(llenar los datos antes de crear el envío)
- Crear envío

- Ver mis envíos

Resultados obtenidos:

Prueba	Promedio	Min	Max
Página principal	956	76	3632
Sign in	636	177	954
Formulario envío	536	55	824
Crear envío	487	153	764
Ver mis envíos	376	131	600

Esta tabla muestra un resumen de las pruebas realizadas, donde para cada una, muestra el promedio, el tiempo máximo y mínimo en milisegundos. El promedio de todas las pruebas es de 598 ms. Si bien, algunos tiempos se pueden mejorar, está dentro de lo razonable. Se puede observar, que el llamar a otro microservicio no afecta en cantidad el tiempo de respuesta, ya que en el formulario de envío, se llama al servicio de costos para realizar el costo. Sin embargo, el promedio de ésta funcionalidad es de 536 ms.

El archivo .jmx con las pruebas de jMeter esta en el repositorio, con el nombre *EnviosYaPruebas.jmx*

4.9. Identificación de fallas

Para facilitar la identificación de fallas se mantiene un log que guarda información relevante. Se utilizo la gema 'Logging', que genera un archivo app.log dentro de la estructura del proyecto donde se regista por un lado información de cuando se registra un usuario, cadete se realiza un envío y se finaliza el mismo. Por otro lado se guardan también las fallas del sistema, por ejemplo cuando la base de datos no se pudo actualizar, un usuario no se pudo crear, etc.

Como el log tiene que ser de las últimas 24 horas, creamos un 'job' que mediante sidekiq corre todos los días a las 3am borrando el archivo txt. De esta manera el log se mantiene actualizado automáticamente y esto se hace de forma simple y eficiente.

5. Manual de instalación

1. Bajarse de los 3 repositorios el código fuente.
2. Abrir el proyecto envios ya, configurar el databse.yml con las credenciales.
3. Crear la base de datos y correr el seed (rake db:setup, rake db:seed), para cargar datos de prueba.

4. Abrir el proyecto de costos, configurar el database.yml y cambiar el puerto a 1500. Crear la base de datos y correr los seeds.
5. Abrir el proyecto de logs, database.yml y cambiar el puerto a 1200. Crear la base de datos.
6. Prender un servidor sidekiq para la aplicación envíos ya otro para el servicio de costos. Para ello es necesario abrir una consola, pararse en el proyecto y correr el comando sidekiq.
7. Prender un servidor redis mediante el comando redis-server.
8. Prender los servidores rails para cada aplicación.
9. La aplicación esta lista para ser usada!

Mejoras

Para esta segunda entrega realizamos una serie de mejoras dentro del código, que se detallan a continuación.

Rubocop

Se instaló la gema de rubocop para seguir los estándares de Ruby on Rails a la hora de programar y de esta manera poder lograr un código fuente mas mantenible.

Encriptación de contraseñas

Debido a la falta de tiempo, en la entrega anterior no encriptamos las contraseñas. Pudimos mejorar este aspecto del sistema encriptando todas las contraseñas de los usuarios y cadetes en la base de datos. Para ello utilizamos la gema Bcrypt.

Extracción de variables

Para mejorar la calidad de nuestro código abstrajimos todas las credenciales y variables configurables del sistema a un mismo lugar. De esta manera, es mas fácil configurar el sistema y cambiar valores. Utilizamos variables de entorno, que están todas encapsuladas en una sola clase y luego se llaman en los modelos y controladores.

Pruebas

Para esta segunda entrega pudimos realizar pruebas utilizando Jmeter. Como se detalló anteriormente en el documento, se hicieron algunas pruebas para poder testear el sistema y sus tiempos de respuesta.