

Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería en Sistemas

**Arquitectura de Software en la
Práctica**

Coupon Manager

Nicolas Damiani - 192489

Rafael Alonso - 201523

Sebastián Rodríguez - 192412

Indice

Introduccion	4
1.1. Proposito	4
1.2 Proposito del sistema	4
Objetivos de la arquitectura	5
2.1 Resumen de requerimientos funcionales	5
2.2 Resumen de requerimientos no funcionales	6
2.3 Restricciones	7
Documentación de la arquitectura	8
3.1 Vista de modulos	8
3.1.1 Vista de descomposición	8
3.1.1.1 Representacion primaria	8
3.1.1.2 Catalogo de elementos	9
3.1.1.3 Justificaciones de diseno	9
3.1.2 Vista de usos	10
3.1.2.1 Representacion primaria	10
3.1.2.2 Catalogo de elementos	10
3.1.2.3 Justificacion de diseno	10
3.2 Vista de componentes y conectores	11
3.2.1 Vista de componentes y conectores del monolito	11
3.2.1.1 Representacion primaria	11
3.2.1.2 Catalogo de elementos	12
3.2.1.3 Comportamiento	13
3.2.1.4 Justificaciones de diseno	13
3.3 Vista de asignación	15
3.3.1 Vista de despliegue	15
3.3.1.1 Representacion primaria	15
3.3.1.2 Catalogo de elementos	15
3.3.1.3 Justificacion de diseno	16
Justificaciones de Diseño para Requerimientos No Funcionales	18
Manual de uso	20
Descripción del proceso de deployment	21
6. Código fuente y estilos	21

Control de versiones	22
Pruebas de carga	23
Pruebas funcionales de controllers y modelos	25

1.Introduccion

1.1. Proposito

El siguiente documento tiene como propósito describir la arquitectura planificada para el servicio desarrollado Coupon Manager. Se identificarán requerimientos funcionales y no funcionales del sistema que servirán como conductores de la arquitectura y se especificará la arquitectura del sistema implementado siguiendo el modelo *Views&Beyond*.

1.2 Proposito del sistema

El sistema por desarrollar consiste en brindar un servicio que permita la creación, gestión y trazabilidad de cupones promocionales y descuentos, pudiendo ser gestionados desde un mismo lugar. Es decir, dado un cupón de una compañía ofrezca el sistema deberá validar si el cupón es válido y si además cumple con las reglas para que sea aplicado. A su vez, el sistema también deberá poder aplicar descuentos en tiempo real dadas ciertas condiciones en el proceso de compra de una plataforma online, como si la compra supera cierto monto, y cantidad de ítems entonces se aplica el descuento.

2. Objetivos de la arquitectura

2.1 Resumen de requerimientos funcionales

Id Requerimiento	Descripción	Actor
RF 1.1	Registro de usuario vía web	Usuario no registrado
RF 1.2	Registro de usuario vía link de invitación	Usuario no registrado
RF 2	Autenticación de usuario	Usuario administrador Usuario de organización
RF 3	Gestión de clave de aplicación	Usuario administrador
R 4	Gestión de promociones	Usuario administrador
R 5	Listado de promociones	Usuario administrador Usuario organizador
RF 6	Baja de promocion	Usuario administrador
RF 7	Reporte de uso	Usuario administrador Usuario de organización
RF 8	SDK Evaluación de promoción	Sistema externo
RF 9	Gestión de usuarios	Usuario administrador
RF 11	Limite de uso de cupon	Sistema
RF 12	Expiración de promociones	Sistema
RF 13	Agregar cupones a promocion	Usuario administrador
RF 14	Vencimiento de cupones	Sistema
RF 15	Reporte demográfico de usuarios	Usuario administrador Usuario de organización

2.2 Resumen de requerimientos no funcionales

Id Requerimiento	Atributo de calidad	Descripción
RNF 1	Performance	El tiempo de respuesta para todas las operaciones públicas del sistema en condición de funcionamiento normal deberá mantenerse por debajo de los 200ms para cargas de hasta 1200 req/m
RNF 2	Confiabilidad y disponibilidad	Se deberá proveer un endpoint HTTP de acceso público que informe el correcto funcionamiento del sistema (healthcheck)
RNF 3	Seguridad	Todo dato de configuración sensible que se maneje en el código fuente deberá poder especificarse en tiempo de ejecución mediante variable de entorno
RNF 4	Seguridad	Se deberá establecer un control de acceso basado en roles, distinguiendo entre usuarios administradores y usuarios de una organización. Se deberá restringir el acceso a las correspondiente funcionalidades.
RNF 5	Seguridad	El sistema deberá responder con código 40x a cualquier request mal formada o no reconocida. Toda comunicación entre clientes front end y componentes de back end deberán utilizar un protocolo de transporte seguro. Por otra parte, la comunicación entre componentes de back end deberá dentro de una red de alcance privado o deberan también utilizar un protocolo de transporte seguro autenticado por una clave de autenticación.
RNF 6	Mantenibilidad	El código fuente de la aplicación seguirá la guía oficial de Ruby, verificado por la herramienta analizadora de codigo estatico, Rubocop. Además se crea un archivo README.md, con todas las instrucciones necesarias para configurar un nuevo ambiente de desarrollo.
RNF 7	Testeabilidad	Se deberá mantener un script de generación de planes de prueba de carga utilizando la herramienta Apache jMeter, con el objetivo de ejercitar todo el sistema simulando la actividad de múltiples usuarios concurrentes. Además de las pruebas de carga, se deberá contar con pruebas funcionales automatizadas.

RNF 8	Seguridad y disponibilidad	Para facilitar la detección e identificación de fallas, se deben centralizar y retener los logs emitidos por la aplicación en producción por un periodo mínimo de 24hrs.
RNF 9	Escalabilidad y modificabilidad	Se requiere dividir la aplicación en un mínimo de tres microservicios.
RNF 10 (RNF 8 en letra 2)	Confiabilidad y disponibilidad	Para facilitar la detección e identificación de fallas, se deberá monitorear peticiones por minuto y tiempos de respuesta

2.3 Restricciones

- El sistema deberá ser desarrollado con el lenguaje Ruby utilizando el framework Ruby on Rails.
- El código y los comentarios deberán ser escritos en inglés siguiendo una guía de estilos de Ruby a determinar, chequeado con Rubocop.
- El control de versiones deberá llevarse a cabo en un repositorio Git y para el manejo de branches se deberá utilizar Git Flow.

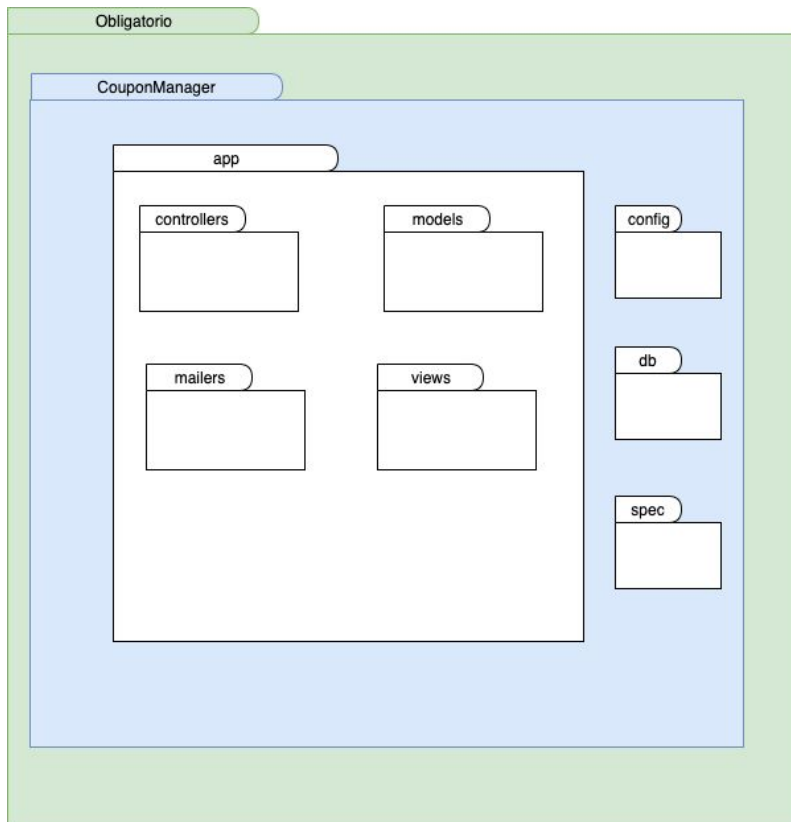
3. Documentación de la arquitectura

3.1 Vista de módulos

3.1.1 Vista de descomposición

3.1.1.1 Representacion primaria

La vista a continuación no incluye todos los módulos del sistema. Por temas de legibilidad y prolijidad se incluyeron aquellos más relevantes.



3.1.1.2 Catalogo de elementos

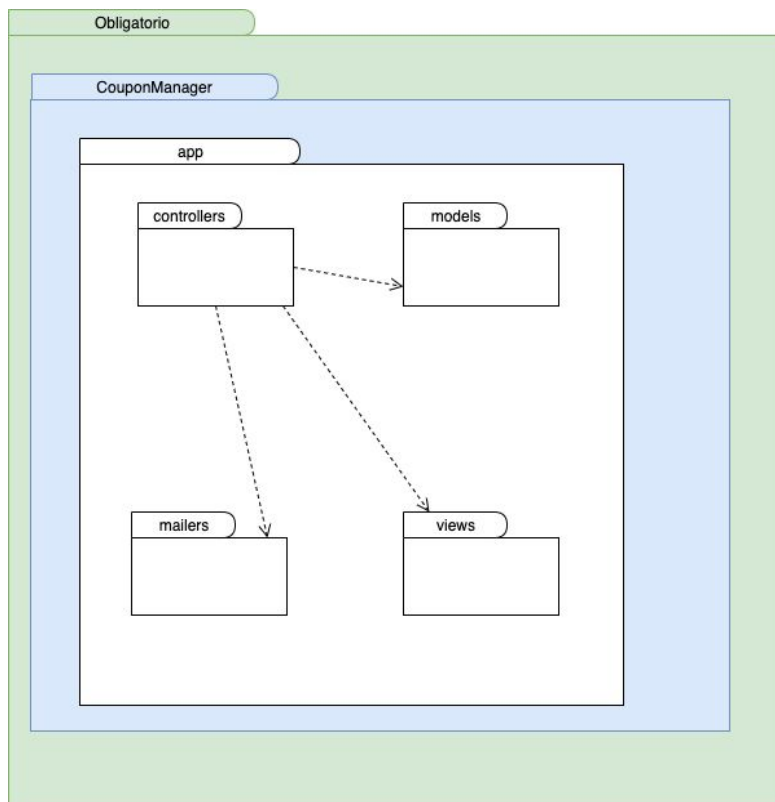
Elemento	Responsabilidad
Obligatorio	Es el módulo que contiene el proyecto.
CouponManager	Es el módulo que contiene todos los demás módulos del sistema, así como también archivos importantes como por ejemplo el Gemfile.
App	Es el módulo que contiene la lógica del sistema
Config	Es el módulo que contiene archivos de configuración del sistema.
Db	Es el módulo que contiene lo relacionado a la base de datos.
Controllers	Es el módulo que contiene los controladores del sistema que interactúan con los modelos y con las vistas.
Mails	Es el módulo que contiene la lógica para enviar mails
Models	Es el módulo que contiene los modelos del sistema

3.1.1.3 Justificaciones de diseño

En la vista anterior podemos ver la descomposición en módulos del sistema de manera simplificada ya que en ella no se encuentran todos los módulos del sistema. Esta estructura no fue creada por el equipo, sino que se autogeneran al crear el proyecto de Rails. Se tomó ventaja de esto para favorecer el reuso y la alta cohesión entre los módulos, logrando mejorar la mantenibilidad. Para lograr esto, los módulos se agrupan por funcionalidades comunes. Por ejemplo: mails contiene la lógica del envío de mails y está separado de db que contiene todo lo relacionado a la base de datos. De esta forma, en caso de modificarse la manera en que se envían los mails no afectará lo relacionado a la base de datos y viceversa. Así se logra bajar el acoplamiento y se podrá obtener un sistema el cual queda más mantenible.

3.1.2 Vista de usos

3.1.2.1 Representacion primaria



3.1.2.2 Catalogo de elementos

Los elementos presentes en el diagrama ya fueron especificados en el catálogo de elementos de la vista anterior.

3.1.2.3 Justificacion de diseno

En la vista anterior no se encuentran todos los módulos del sistema, sólo representamos aquellos los cuales consideramos más relevantes en cuanto al diseño. Lo más importante para

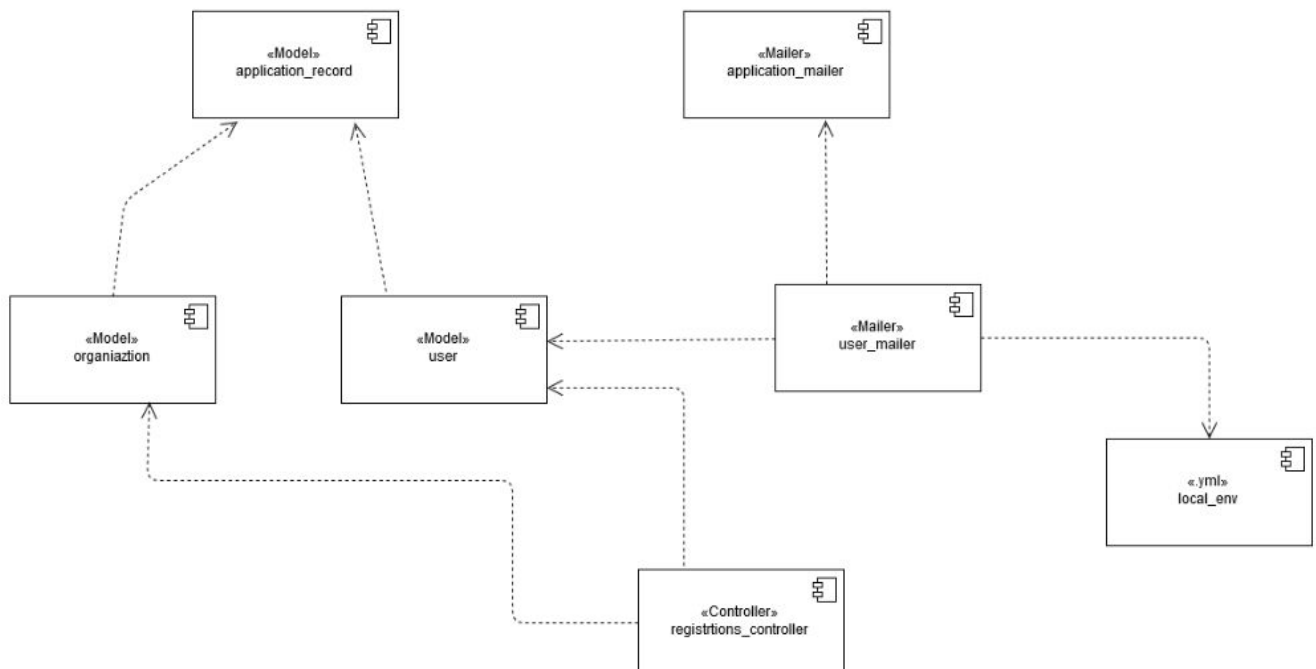
destacar es el uso del patrón MVC. Este se genera automáticamente por Rails ya que la arquitectura del framework se basa en este patrón. Este patrón consiste en separar la aplicación en tres componentes: modelos (información de los modelos de negocio), vistas (presentación de los datos y manejo de interfaces) y controladores (manejo de interfaz de usuario y de la aplicación, comunicándose con los dos anteriores).

Los motivos principales por los cuales se separa la aplicación en estos tres componentes es para favorecer la escalabilidad, la mantenibilidad y el reuso.

3.2 Vista de componentes y conectores

3.2.1 Vista de componentes y conectores del monolito

3.2.1.1 Representacion primaria

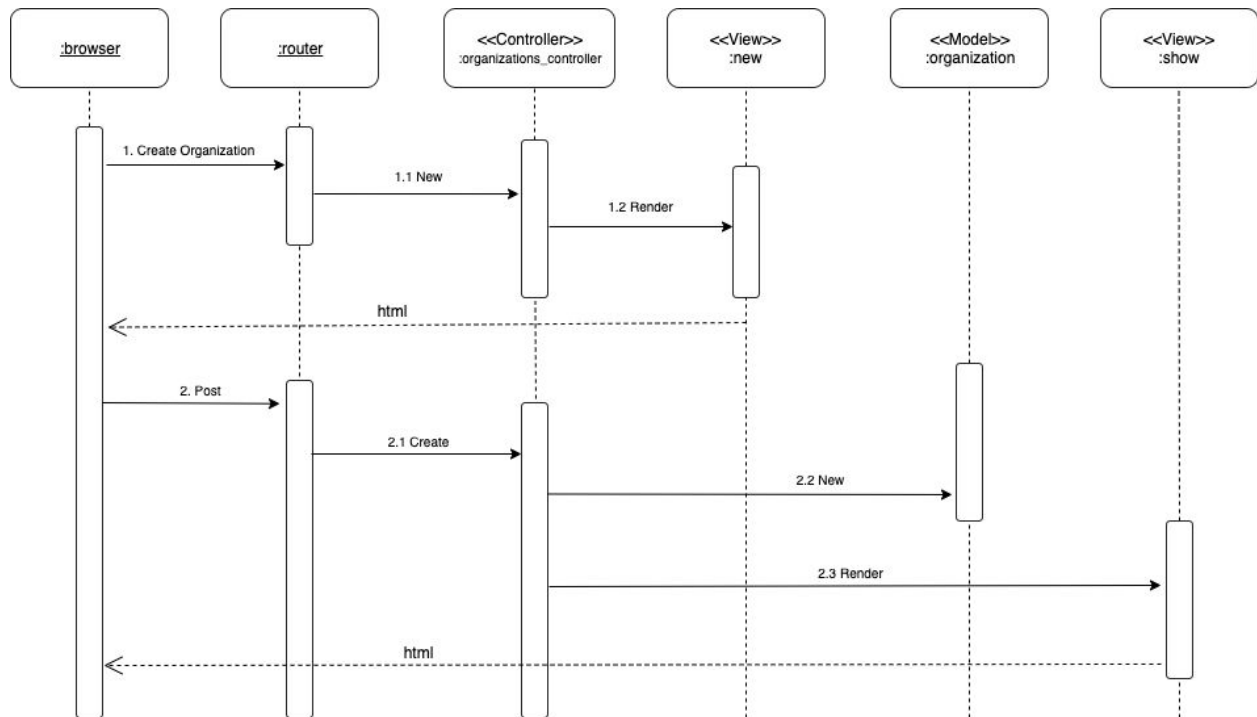


3.2.1.2 Catalogo de elementos

Componente	Tipo	Descripción
Application_record	Model	Posee la clase de Rails de la cual hereda Active Record
Organization	Model	Es el componente que contiene la información de las organizaciones.
User	Model	Es el componente que contiene la información de los usuarios
Application_mailer	Mailer	Componente que permite enviar mails desde la aplicación.
User_mailer	Mailer	Contiene la lógica para el envío de mails a los usuarios
Registration_controller	Controller	Componente que tiene la lógica para el manejo de los registros de usuarios
Local_env	.yml	Componente que tiene la definición de variables de entorno.

3.2.1.3 Comportamiento

Diagrama de secuencia para la creación de una organización.



3.2.1.4 Justificaciones de diseno

La vista anterior muestra una parte de los componentes y conectores del sistema CouponManager. Aquí podemos observar distintas cosas. En primer lugar, el controlador de nombre registrations_controller contiene la lógica para el manejo de los usuarios y el registro de estos. Para el manejo de los usuarios en la aplicación se utilizó la gema Devise la cual nos brinda de una forma sencilla la autenticación de usuarios.

Por otro lado, controlador healthcheck_controller contiene la lógica para poder verificar el estado del sistema. Se expone un endpoint para cumplir con el requerimiento. Se utilizó la táctica de ping/echo, donde un usuario puede acceder a dicho endpoint y corroborar el estado del sistema.

En cuanto a la seguridad en las comunicaciones entre componentes de front end y back end, Rails genera un token para prevenir ataques de csrf y con el cual valida que la request

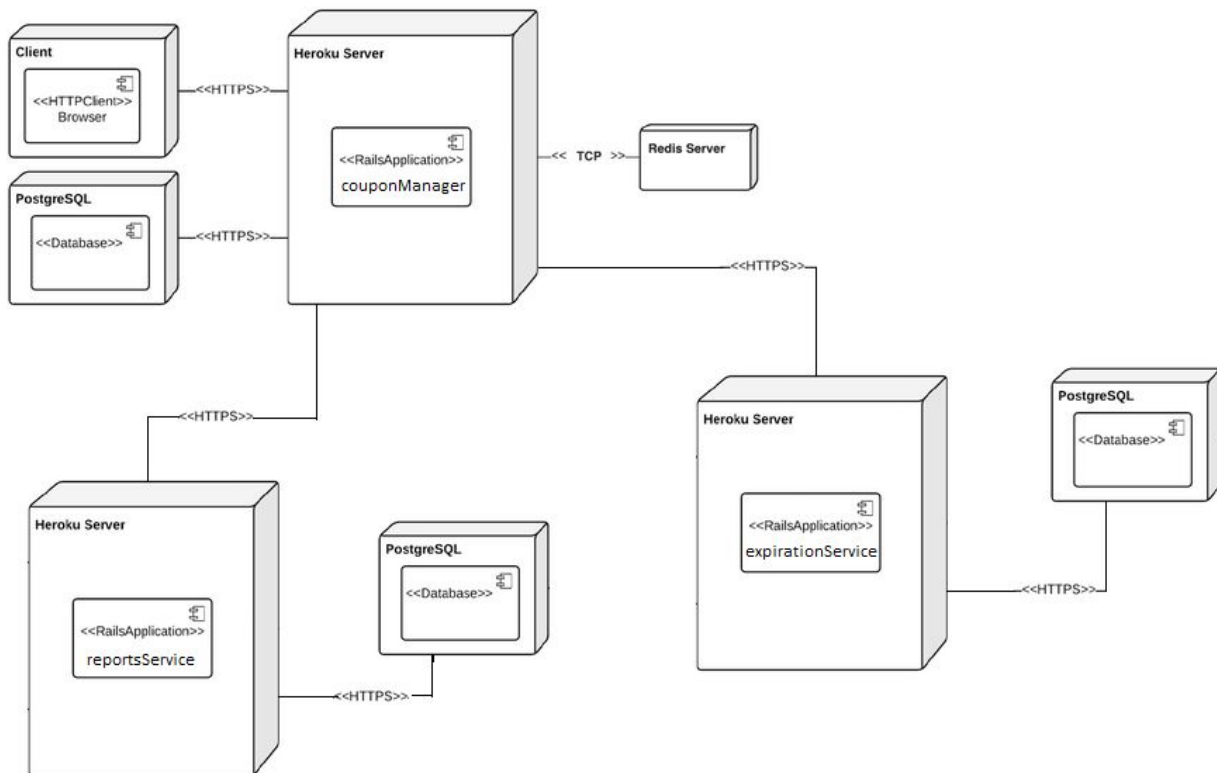
recibida provenga de una página válida. Este token se almacena en la página y en la session cookie y al recibir una request compara ambos tokens para verificar que sean iguales. Al utilizar esta medida de seguridad se está cumpliendo con el RQA.

Por último, se implementó un componente de nombre `local_env`. En este podemos encontrar las variables de entorno y de esta manera poder cumplir con uno de los requerimientos. El mismo es utilizado por el `user_mailer` para obtener las credenciales necesarias a la hora de enviar mails.

3.3 Vista de asignación

3.3.1 Vista de despliegue

3.3.1.1 Representacion primaria



3.3.1.2 Catalogo de elementos

Elemento	Responsabilidad
Heroku server	Es el ambiente de ejecución del sistema.
PostgreSQL	Es donde se encuentran las bases de datos de cada servicio

Client	Es el ambiente que se va a conectar con el sistema para utilizarlo
RailsApplication	El sistema en sí, deployado en el servidor heroku
Database	Es donde persiste la información del sistema flagger.
Redis server	Es donde se encuentra la memoria caché del sistema.
Browser	Es un cliente específico que va a usar el sistema. Se puso un browser a modo de ejemplo, pero tambien podria ser Postman u otro/
couponManager	El sistema que contiene la parte de usuarios y promociones/cupones
expirationService	El sistema que se encarga de enviar mails a los usuarios administradores cuando una promoción esta por vencer y cuando vencio
reportsService	El sistema que se encarga de manejar los reportes

3.3.1.3 Justificación de diseno

En la vista anterior se muestra la asignación física de los diferentes componentes y donde fue deployado cada uno de estos. Sobre la performance, debido a tener sistema desplegado en el mismo nodo, se logra hacer que la comunicación entre front end y back end sea mas rapida como resultado de estar en el mismo lugar.

Logs:

Se utilizó el servicio de Loggly relacionado al servicio nuestro para poder almacenar Logs, con errores, advertencias y demás en un periodo mayor a 24 horas.

Multi tenancy:

El sistema CouponManager implementa multi tenancy ya que tenemos misma aplicación la cual es usada por muchas organizaciones diferentes. La arquitectura que se eligió para la base de datos fue mantener una misma bd para todas las organizaciones. No fue necesario el uso de ninguna gema para implementarlo, sino que el manejo fue manualmente debido a que era una solución sencilla y eficiente que funciona correctamente. Lo que se hizo es que cada usuario solo puede interactuar con las cosas asociadas a su organización, escondiendo las cosas de las demás.

Ambientes:

En la vista de despliegue podemos ver lo relacionado al ambiente de producción. De todas formas, se utilizaron otros dos ambientes: development y testing. El de development es local y fue utilizado para desarrollar el sistema, mientras que el segundo, fue utilizado para el testing de la aplicación. Aquí se puede ver el uso de la técnica de testabilidad sandbox la cual es utilizada para probar el sistema de manera independiente del de producción y de esta forma no afectarse el uno al otro.

Arquitectura de microservicios:

Así como fue solicitado para esta entrega, el sistema fue dividido en diferentes microservicios. El servicio de couponmanager es el que posee toda la lógica sobre el manejo de las promociones y los usuarios. El servicio expiration es el encargado de enviar los mail a los diferentes usuarios administrativo el día antes de que una promoción se vense y también en el momento que ese vencimiento se hace efectivo. Finalmente el tercer microservicio trata sobre los reportes demográficos. En el mismo se envían los datos de las promociones que fueron evaluadas correctamente y también es el solicitado para traer toda la información de los reportes. Se utilizó Node.js para poder implementar este servidor.

Como observamos en el diagrama anterior, se utilizó el patrón “Database per service” al momento de tomar la decisión de que cada servicio tenga su propia base de datos. Con esto nos aseguramos un bajo acoplamiento entre los servicios y sus respectivas bases de datos y por otra parte que los cambios efectuados en una base de datos no impacten en las demás.

SDK:

Se implementó una gema en ruby con una interfaz que le permita al cliente, poder asignarle un id de promoción, las condiciones y el token del usuario. Esta sdk lo único que hace es pegarle al endpoint de evaluaciones, y fijarse el tiempo de respuesta para que en caso de ser mayor a 1000ms dar un mensaje de error

Monitoreabilidad:

Se utilizó la gema de ruby new relic para monitorear los tiempos de respuesta y peticiones por minuto del sistema. Esta nos permite ingresar a nuestro usuario de new relic y ver todos los detalles de peticiones al sistema.

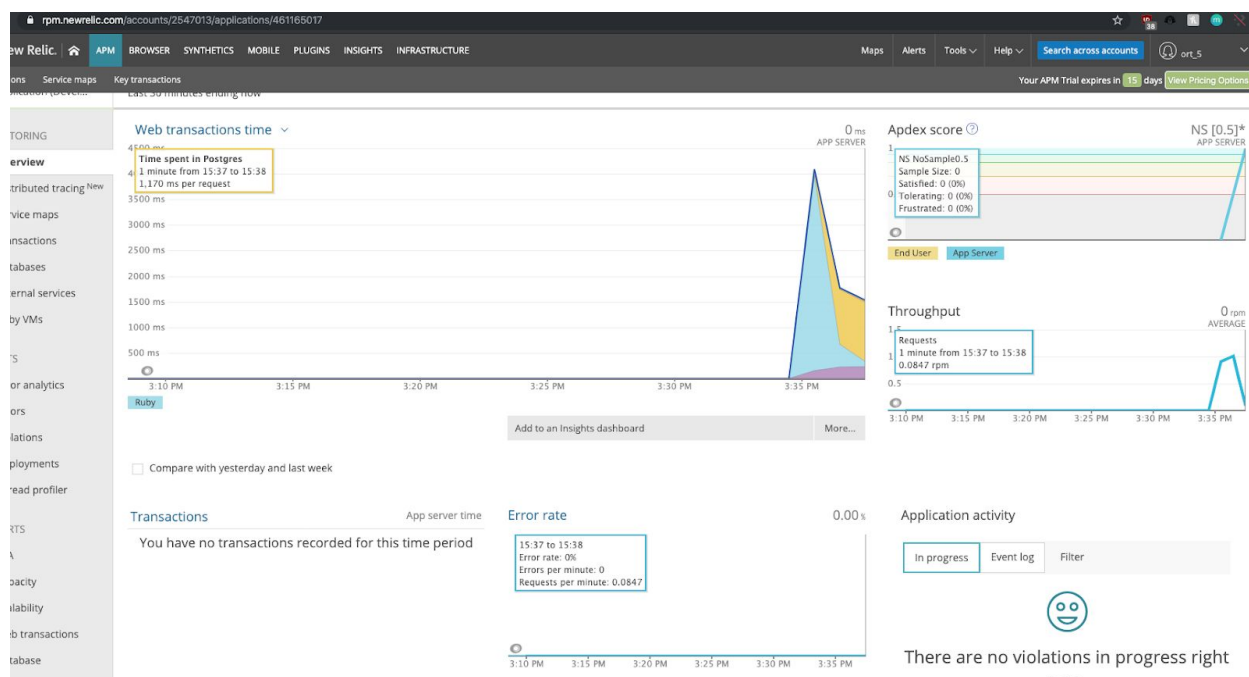
Justificaciones de Diseño para Requerimientos No Funcionales

A continuación se detalla una tabla, donde se ilustran las tácticas que fueron utilizadas para cumplir con cada requerimiento no funcional especificado.

Id Requerimiento	Atributo de calidad	Tácticas Utilizadas
RNF 1	Performance	Esto se logró utilizando un caché en redis (Múltiples copias de cómputos) y refactorizando el código hasta que sea lo más eficiente posible (Incrementar eficiencia de recursos)
RNF 2	Confiabilidad y disponibilidad	Se expone un endpoint healthcheck que permite saber si el servidor se encuentra activo (Ping/Echo)
RNF 3	Seguridad	Todo dato de configuración sensible que se maneje en el código fuente deberá poder especificarse en tiempo de ejecución mediante variable de entorno
RNF 4	Seguridad	Se deberá establecer un control de acceso basado en roles, distinguiendo entre usuarios administradores y usuarios de una organización. Se deberá restringir el acceso a las correspondiente funcionalidades.
RNF 5	Seguridad	Se limitó el acceso al sistema para requests inválidas con un código 40X (Limitar acceso y Limitar exposición) . Se usó un protocolo de transporte seguro (Encriptar datos) y se manejan roles y códigos de acceso para distintas partes del sistema (Autorizar actores y Autenticar actores)
RNF 6	Mantenibilidad	El código fuente de la aplicación sigue la guía oficial de Ruby, verificado por la herramienta analizadora de código estático, Rubocop. Además se creó un archivo README.md, con todas las instrucciones necesarias para configurar un nuevo ambiente de desarrollo.
RNF 7	Testeabilidad	Se deberá mantuvo un script de generación de planes de prueba de carga utilizando la herramienta Apache jMeter, con el objetivo de ejercitar todo el sistema simulando la actividad de múltiples usuarios concurrentes. Además de las pruebas de carga, cuenta con pruebas funcionales

		automatizadas para el RF 1 a nivel de sus controladores.
RNF 8	Seguridad y disponibilidad	Para facilitar la detección e identificación de fallas, se centralizaron y retuvieron los logs emitidos por la aplicación en producción por un periodo mínimo de 24hrs.
RNF 9	Mantenibilidad	El código se dividió en varios microservicios independientes (restringir dependencias, encapsular)
RNF 10 (RNF 8 en letra 2)	Confiabilidad y disponibilidad	Se utilizó new relic para monitorear los tiempos de respuesta y peticiones por minuto del sistema.

Captura de New Relic:



4. Manual de uso

Para poder hacer uso del sistema se debe de acceder al siguiente link:

En cuanto a los endpoints HTTP solicitados se crearon tres: uno para la evaluación de promociones otro para los reportes de las promociones y otro para informar el correcto funcionamiento del sistema. Los mismos se pueden acceder a través de los siguientes links:

Health check: <https://coupon-manager-arquitectura.herokuapp.com/healthcheck/check>

Evaluación de promociones:

<https://coupon-manager-arquitectura.herokuapp.com/promotions/evaluate?id=25>

Por body pasar los parámetros a evaluar. Ej.: {"total":41, "coupon_code":300, "user_id":45553213}

Reporte de promociones:

https://coupon-manager-arquitectura.herokuapp.com/promotions/report_rest?id=1

Ejemplo de uso de sdk

```
evaluatePromotion(1, {'total' => 41, 'coupon_code' => 300},  
"eyJhbGciOiJIub251In0.eyJwcm9tb3Rpb25zIjoiW1wiMVwiXSJ9.")
```

5. Descripción del proceso de deployment

Se deben seguir los siguientes pasos:

1. Iniciar sesión en Heroku mediante “heroku login”
2. Crear una aplicación en Heroku con “heroku create”
3. Hacer push a la rama master del proyecto al repositorio de heroku con “git push heroku master”
4. Realizar las migraciones necesarias para la aplicación en heroku con “heroku run rake db:migrate”
5. Asegurarse de tener un dyno corriendo en heroku con “heroku ps:scale web=1”
6. Finalmente abrir la página de la aplicación con “heroku open”.

6. Código fuente y estilos

Las restricciones de la letra incluyen que el sistema debe ser desarrollado en lenguaje Ruby utilizando el framework Ruby on Rails.

A su vez, se pedía seguir una guía de estilos la cual favorezca la mantenibilidad del proyecto y también debía ser chequeado el código utilizando la herramienta Rubocop. Para esto, se utilizó la gema rubocop-rails_config la cual permite el análisis del código para ver errores de estilos y así poder corregirlos. A continuación, la evidencia de correr dicha herramienta:

7. Control de versiones

En cuanto al control de versiones se creó un repositorio de Github privado en el cual se encuentran todos los archivos del proyecto. Se utilizó la herramienta Git tal como fue solicitado por en letra.

Para el flujo de trabajo se siguió GitFlow. Se creó una rama master, que es la principal y es en donde están las entregas del proyecto. Luego, una rama develop de la cual se fueron creando diferentes features para cada nueva funcionalidad que se iba desarrollando. Una vez finalizado el desarrollo de alguna funcionalidad, a la rama correspondiente se le hizo merge con la rama develop.

Los diferentes servicios se implementan en distintos repositorios de GitHub. Los nombres de estos son los siguientes:

Repositorio de obligatorio: <https://github.com/ArqSoftPractica/AlonsoDamianiRodriguez>

Microservicio principal: <https://github.com/nicolas-damiani/CouponManager>

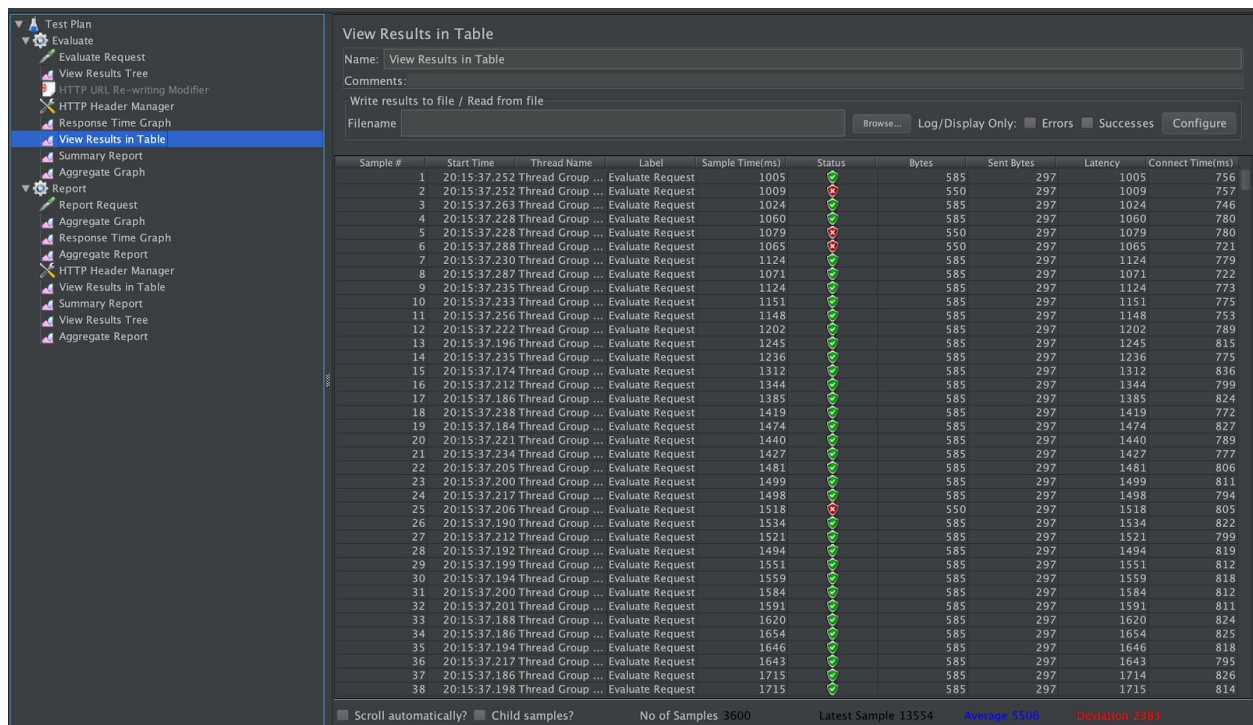
Microservicio reportes: <https://github.com/nicolas-damiani/ReportsService>

Microservicio expiraciones: <https://github.com/nicolas-damiani/ExpirationService>

8. Pruebas de carga

Para el RQA7 se pidieron pruebas de carga las cuales permiten simular el funcionamiento de la aplicación con muchos usuarios a la vez y de esta manera poder medir los tiempos de respuesta y analizar la performance. Para esto, se utilizó la herramienta Apache JMeter para crear pruebas de carga las cuales permitieran probar los endpoints públicos del sistema. A continuación, se deja evidencia de lo realizado:

Api Evaluate



The screenshot shows the Apache JMeter 'View Results in Table' window. The left sidebar contains a tree view of the test plan, with 'View Results in Table' selected. The main area displays a table of test results for the 'Api Evaluate' test. The table has columns for Sample #, Start Time, Thread Name, Label, Sample Time (ms), Status, Bytes, Sent Bytes, Latency, and Connect Time (ms). The results show 3600 samples, with the latest sample being 13554. The average latency is 1700 ms and the deviation is 2383 ms. The status column shows green checkmarks for successful requests and red X marks for failed requests.

Sample #	Start Time	Thread Name	Label	Sample Time (ms)	Status	Bytes	Sent Bytes	Latency	Connect Time (ms)
1	20:15:37.252	Thread Group ...	Evaluate Request	1005	✓	585	297	1005	756
2	20:15:37.252	Thread Group ...	Evaluate Request	1009	✓	550	297	1009	757
3	20:15:37.263	Thread Group ...	Evaluate Request	1024	✓	585	297	1024	746
4	20:15:37.228	Thread Group ...	Evaluate Request	1060	✓	585	297	1060	780
5	20:15:37.228	Thread Group ...	Evaluate Request	1079	✓	550	297	1079	780
6	20:15:37.288	Thread Group ...	Evaluate Request	1065	✓	550	297	1065	721
7	20:15:37.230	Thread Group ...	Evaluate Request	1124	✓	585	297	1124	779
8	20:15:37.287	Thread Group ...	Evaluate Request	1071	✓	585	297	1071	722
9	20:15:37.235	Thread Group ...	Evaluate Request	1124	✓	585	297	1124	773
10	20:15:37.233	Thread Group ...	Evaluate Request	1151	✓	585	297	1151	775
11	20:15:37.256	Thread Group ...	Evaluate Request	1148	✓	585	297	1148	753
12	20:15:37.222	Thread Group ...	Evaluate Request	1202	✓	585	297	1202	789
13	20:15:37.196	Thread Group ...	Evaluate Request	1245	✓	585	297	1245	815
14	20:15:37.235	Thread Group ...	Evaluate Request	1236	✓	585	297	1236	775
15	20:15:37.174	Thread Group ...	Evaluate Request	1312	✓	585	297	1312	836
16	20:15:37.212	Thread Group ...	Evaluate Request	1344	✓	585	297	1344	799
17	20:15:37.186	Thread Group ...	Evaluate Request	1385	✓	585	297	1385	824
18	20:15:37.238	Thread Group ...	Evaluate Request	1419	✓	585	297	1419	772
19	20:15:37.184	Thread Group ...	Evaluate Request	1474	✓	585	297	1474	827
20	20:15:37.221	Thread Group ...	Evaluate Request	1440	✓	585	297	1440	789
21	20:15:37.234	Thread Group ...	Evaluate Request	1427	✓	585	297	1427	777
22	20:15:37.205	Thread Group ...	Evaluate Request	1481	✓	585	297	1481	806
23	20:15:37.200	Thread Group ...	Evaluate Request	1499	✓	585	297	1499	811
24	20:15:37.217	Thread Group ...	Evaluate Request	1498	✓	585	297	1498	794
25	20:15:37.206	Thread Group ...	Evaluate Request	1518	✓	550	297	1518	805
26	20:15:37.190	Thread Group ...	Evaluate Request	1534	✓	585	297	1534	822
27	20:15:37.212	Thread Group ...	Evaluate Request	1521	✓	585	297	1521	799
28	20:15:37.192	Thread Group ...	Evaluate Request	1494	✓	585	297	1494	819
29	20:15:37.199	Thread Group ...	Evaluate Request	1551	✓	585	297	1551	812
30	20:15:37.194	Thread Group ...	Evaluate Request	1559	✓	585	297	1559	818
31	20:15:37.200	Thread Group ...	Evaluate Request	1584	✓	585	297	1584	812
32	20:15:37.201	Thread Group ...	Evaluate Request	1591	✓	585	297	1591	811
33	20:15:37.188	Thread Group ...	Evaluate Request	1620	✓	585	297	1620	824
34	20:15:37.186	Thread Group ...	Evaluate Request	1654	✓	585	297	1654	825
35	20:15:37.194	Thread Group ...	Evaluate Request	1646	✓	585	297	1646	813
36	20:15:37.217	Thread Group ...	Evaluate Request	1643	✓	585	297	1643	795
37	20:15:37.185	Thread Group ...	Evaluate Request	1715	✓	585	297	1714	826
38	20:15:37.198	Thread Group ...	Evaluate Request	1715	✓	585	297	1715	814

Summary statistics at the bottom: No of Samples 3600, Latest Sample 13554, Average 1700, Deviation 2383.

Numero de cargas: 1200

Tiempo promedio de respuesta: 5506 ms

Api Report

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	20:15:40.570	Thread Group ...	Report Request	5717	✓	658	248	5717	539
2	20:16:08.052	Thread Group ...	Report Request	7361	✓	658	248	7361	534
3	20:16:08.043	Thread Group ...	Report Request	7390	✓	658	248	7390	543
4	20:16:08.040	Thread Group ...	Report Request	7389	✓	658	248	7389	546
5	20:16:08.097	Thread Group ...	Report Request	7451	✓	658	248	7451	555
6	20:16:08.042	Thread Group ...	Report Request	7377	✓	658	248	7377	544
7	20:16:08.038	Thread Group ...	Report Request	7391	✓	658	248	7391	548
8	20:16:08.038	Thread Group ...	Report Request	7486	✓	658	248	7486	529
9	20:16:08.052	Thread Group ...	Report Request	7492	✓	658	248	7492	535
10	20:16:08.027	Thread Group ...	Report Request	7517	✓	658	248	7517	560
11	20:16:08.091	Thread Group ...	Report Request	7462	✓	658	248	7462	562
12	20:16:08.085	Thread Group ...	Report Request	7459	✓	658	248	7459	510
13	20:16:08.035	Thread Group ...	Report Request	7509	✓	658	248	7509	551
14	20:16:08.088	Thread Group ...	Report Request	7454	✓	658	248	7454	509
15	20:16:08.053	Thread Group ...	Report Request	7489	✓	658	248	7489	533
16	20:16:08.042	Thread Group ...	Report Request	7497	✓	658	248	7497	544
17	20:16:08.060	Thread Group ...	Report Request	7482	✓	658	248	7481	527
18	20:16:08.062	Thread Group ...	Report Request	7476	✓	658	248	7476	525
19	20:16:08.056	Thread Group ...	Report Request	7481	✓	658	248	7481	531
20	20:16:08.060	Thread Group ...	Report Request	7477	✓	658	248	7477	527
21	20:16:08.063	Thread Group ...	Report Request	7474	✓	658	248	7474	524
22	20:16:08.044	Thread Group ...	Report Request	7493	✓	658	248	7493	543
23	20:16:08.069	Thread Group ...	Report Request	7468	✓	658	248	7468	518
24	20:16:08.062	Thread Group ...	Report Request	7474	✓	658	248	7474	525
25	20:16:08.057	Thread Group ...	Report Request	7479	✗	550	248	7479	530
26	20:16:08.109	Thread Group ...	Report Request	7467	✓	658	248	7467	544
27	20:16:08.030	Thread Group ...	Report Request	7506	✓	658	248	7506	557
28	20:16:08.065	Thread Group ...	Report Request	7471	✓	658	248	7471	522
29	20:16:08.067	Thread Group ...	Report Request	7469	✓	658	248	7469	520
30	20:16:08.074	Thread Group ...	Report Request	7461	✓	658	248	7461	513
31	20:16:08.065	Thread Group ...	Report Request	7470	✓	658	248	7470	522
32	20:16:08.073	Thread Group ...	Report Request	7462	✓	658	248	7462	514
33	20:16:08.046	Thread Group ...	Report Request	7489	✓	658	248	7489	540
34	20:16:08.045	Thread Group ...	Report Request	7490	✓	658	248	7490	541
35	20:16:08.034	Thread Group ...	Report Request	7501	✓	658	248	7501	553
36	20:16:08.079	Thread Group ...	Report Request	7456	✓	658	248	7456	508
37	20:16:08.100	Thread Group ...	Report Request	7435	✓	658	248	7435	552
38	20:16:08.055	Thread Group ...	Report Request	7480	✓	658	248	7480	532

■ Scroll automatically? ■ Child samples? No of Samples 2401 Latest Sample 11741 Average 8539 Deviation 1880

Numero de cargas: 1200

Tiempo promedio de respuesta: 8254 ms

Es necesario aclarar que los tiempos de respuesta se vieron afectados severamente por la latencia del servidor y por la GUI de JMETER. Esto se puede validar viendo que los tiempos en la consola fueron mucho menores, alrededor de los 100 ms. De todas formas se incluyen las capturas de pantalla para evidenciar las pruebas de carga.

9. Pruebas funcionales de controllers y modelos

Como parte del RQA7 se solicita la realización de pruebas funcionales automatizadas para los distintos requerimientos funcionales. Para esto se realizó Minitest que es la librería por defecto de testing de Rails. Cada vez que se genera un proyecto nuevo, Rails crea un directorio de test en la aplicación.

Se realizaron pruebas sobre los modelos de Promotion, Organization y User y también se realizaron sobre el controlador de Promotions. En cuanto al controlador de User, no se realizó ningún tipo de prueba ya que para el mismo se utilizó Devise lo cual ya sabemos que funciona correctamente. A continuación presentamos algunas evidencias de los tests:

Modelos:

Users:

```
Finished in 0.380137s, 15.7838 runs/s, 15.7838 assertions/s.  
6 runs, 6 assertions, 0 failures, 0 errors, 0 skips
```

Promotions:

```
Finished in 0.309404s, 38.7842 runs/s, 38.7842 assertions/s.  
12 runs, 12 assertions, 0 failures, 0 errors, 0 skips
```

Organizations:

```
Finished in 0.252798s, 11.8672 runs/s, 11.8672 assertions/s.  
3 runs, 3 assertions, 0 failures, 0 errors, 0 skips
```

Controladores:

Promotions:

```
Finished in 1.886521s, 3.7105 runs/s, 3.7105 assertions/s.  
7 runs, 7 assertions, 0 failures, 0 errors, 0 skips
```