

Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio 2

Entregado como requisito para la aprobación de la materia

Arquitectura de software en la práctica

Matías Crizul – 186651

Santiago Marchisio – 187871

Docentes:
Tomas Piaggio

29 de Noviembre del 2018

Tabla de contenido

1. Descripción de la arquitectura	3
1.1. FlagsApp (Monolito)	4
1.1.1. Vista de módulos.....	4
1.1.2. Decisiones de diseño.....	4
1.1.3. Vista de componentes y conectores	4
1.2. Reportes (Micro Servicio)	7
1.2.1. Vista de módulos.....	7
1.2.2. Decisiones de diseño.....	7
1.2.3. Vista de componentes y conectores	7
1.3. Invitaciones (Micro Servicio).....	9
1.3.1. Vista de módulos.....	9
1.3.2. Decisiones de diseño.....	9
1.3.3. Vista de componentes y conectores	9
1.4. Vista de despliegue.....	11
2. Justificaciones de diseño	12
2.1. RF9. SDK para evaluación de “Flags”.....	12
2.2. RNF1. Performance.....	15
2.3. RNF2. Confiabilidad y disponibilidad	17
2.4. RNF3. Configuración y manejo de secretos.....	19
2.5. RNF4. Autenticación y autorización	20
2.6. RNF5. Seguridad	20
2.7. RNF6. Código fuente	22
2.8. RNF7. Evaluación de “flags” en formato JSON	25
2.9. RNF8. Pruebas	26
2.10. RNF9. Identificación de fallas.....	28
2.11. RNF10. Independencia de aplicaciones.....	29
3. Descripción del proceso de deployment	30

1. Descripción de la arquitectura

En este capítulo se presenta la arquitectura del sistema en base a los requerimientos funcionales y no funcionales especificados previamente.

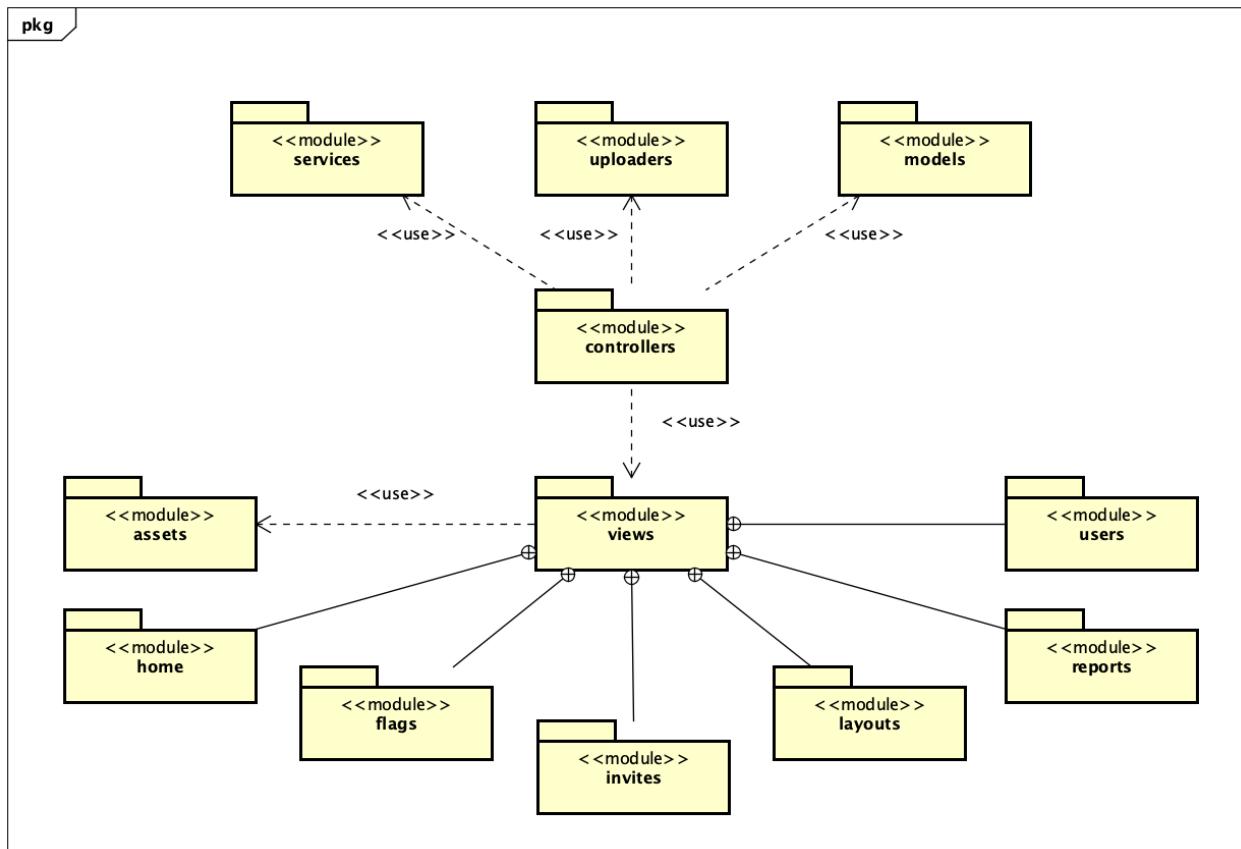
Para esto se muestra la arquitectura del sistema desde distintas vistas para mostrar el comportamiento desde distintos puntos; se realizará una descomposición del sistema, en las distintas aplicaciones requeridas.

Primero se irá a especificar la vista de módulos para mostrar la estructura lógica, luego la vista de componentes y conectores en la cual se verá la estructura del sistema en tiempo de ejecución, y por último la vista de asignación para ver como se distribuirá todo el sistema en un ambiente de ejecución.

Debido a que el sistema se encuentra separado en dos microservicios y un sistema monolítico heredado de la entrega anterior, se mostrará por separado los diagramas correspondientes a cada sistema, explicando las principales decisiones tomadas en cada uno. La vista de asignación será general de cómo se encuentran distribuidas las tres aplicaciones en un ambiente de ejecución, por lo cual esta será única y referirá a los tres sistemas.

1.1. FlagsApp (Monolito)

1.1.1. Vista de módulos



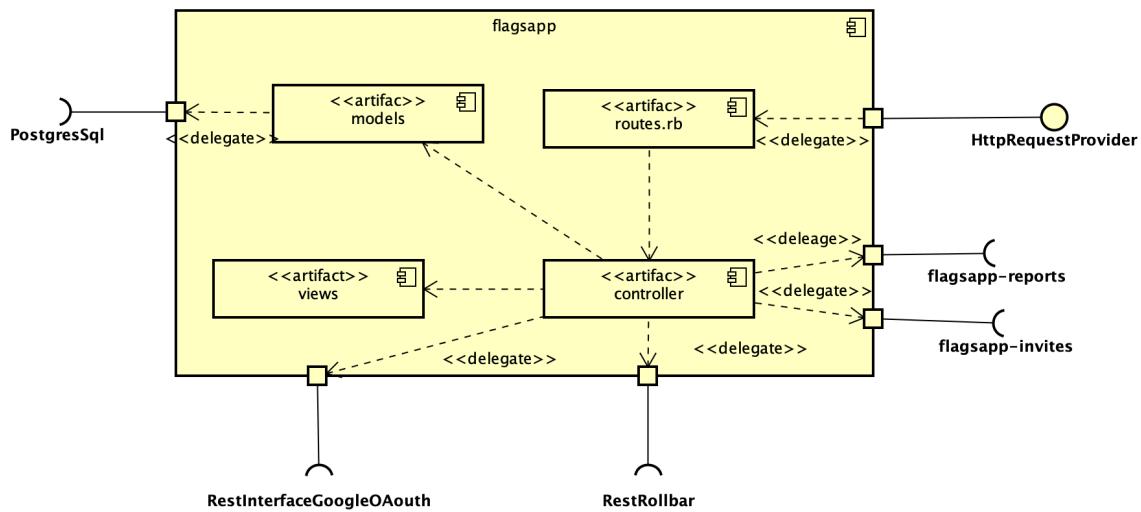
1.1.2. Decisiones de diseño

Ruby on Rails es una aplicación que utiliza el patrón MVC por lo cual no se pueden tomar grandes decisiones de diseño a nivel de módulos, por esto mismo la solución se agrupo en módulos, vista, controllers. El resto de los módulos observados en el diagrama son creados y gestionados internamente por Ruby on Rails

(No se presenta catalogo de elementos, debido a que no se posee un conocimiento claro de los elementos internos de Ruby auto generados, los cuales no son conocidos por el equipo de desarrollo)

1.1.3. Vista de componentes y conectores

A continuación se presentara una vista de componentes y conectores, estas misma reflejara como se procesan las request HTTP que ingresan al sistema.



Interfaces:

Elemento	Responsabilidades
RestRollbar	Interfaz que registra los errores que suceden en el sistema
RestInterfaceGoogleOauth	Se solicita la autenticación de los usuarios logueados con Google
PostgresSQL	Se le solicita el almacenamiento de los datos en la BD
FlagsApp-Reports	Encargado de guardar y gestionar la información de reportes, se le envía y solicita la información de los mismos.
FlagsApp-Invites	Encargado de los mecanismos de invitación externos, este enviará y validará la información de invitaciones realizadas mediante mecanismos alternativos.

Catálogo de elementos:

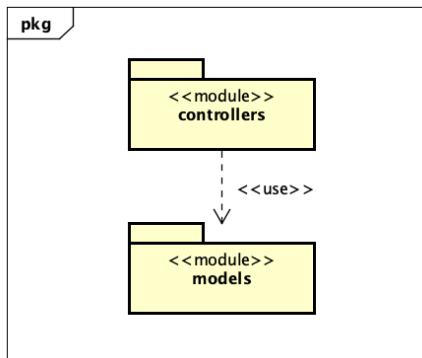
Elemento	Tipo	Responsabilidades
routes	.rb	Encargado de exponer las rutas para el cliente y redirigirlo a los métodos indicados
controller	.rb	Es el encargado de realizar la lógica del sistema, y realizar el render de la view indicada
models	.rb	Modela las entidades que se encuentran en la BD
views	.html.rb	Interfaz gráfica

Decisiones de diseño:

Se tomo la decisión de usar un login con Google debido a que la app que expone otorga un funcionamiento sencillo de configurar, esto se explicara mas adelante en los apartados de requerimientos no funcionales.

1.2. Reportes (Micro Servicio)

1.2.1. Vista de módulos

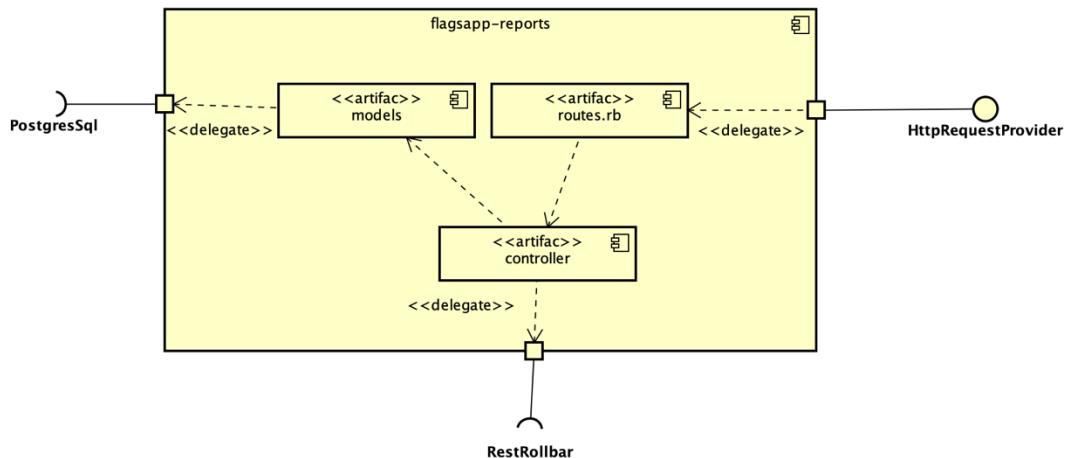


1.2.2. Decisiones de diseño

Las decisiones de diseño tomadas para esta aplicación son similares a las expresadas en la aplicación Flags, debido a que esta también es una aplicación Ruby on Rails, por lo cual no se puede realizar muchas variaciones dado que el desarrollo de la misma está fijado por el patrón MVC. Si podemos destacar que debido a que la misma fue implementada como una Web API, esta no renderiza vistas, solo retornará los datos en formato JSON que son solicitados al controlador.

Sin tomar en cuenta las vistas, el resto de los manejos internos tanto de modelos, controladores y validaciones se realizaron siguiendo los mismos lineamientos anteriormente planteados.

1.2.3. Vista de componentes y conectores



Interfaces:

Elemento	Responsabilidades
RestRollbar	Interfaz que registra los errores que suceden en el sistema
PostgresSQL	Se le solicita el almacenamiento de los datos en la BD

Catalogo de elementos:

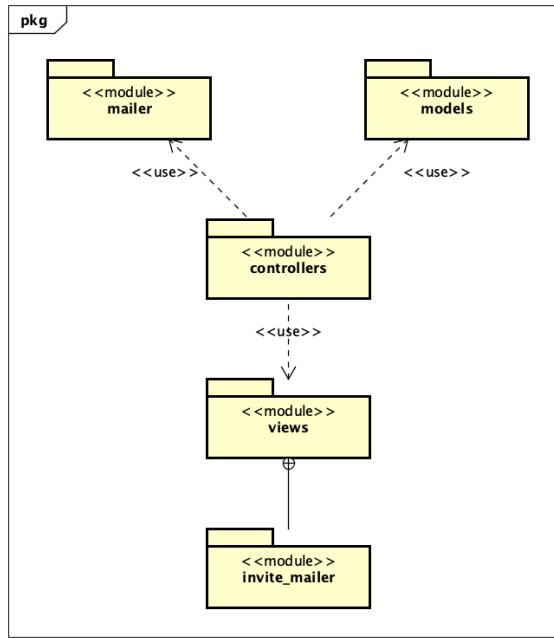
Elemento	Tipo	Responsabilidades
Routes	.rb	Encargado de exponer las rutas para quien lo consuma y redirigirlo a los métodos indicados
Controllers	.rb	Es el encargado de realizar la lógica del sistema, y enviar la información solicitada mediante una respuesta http.
Models	.rb	Modela las entidades que se encuentran en la BD

Decisiones de diseño:

Se optó por separar la funcionalidad de reportes a un micro servicio, debido a que esta no posee una gran relación lógica con las funcionalidades principales y críticas de FlagsApp, por lo cual se puede abstraer en otra aplicación diferente fácilmente, de esta forma logrando una mayor modificabilidad, y la posibilidad de desarrollar esta funcionalidad en cualquier otro lenguaje de programación.

1.3. Invitaciones (Micro Servicio)

1.3.1. Vista de módulos

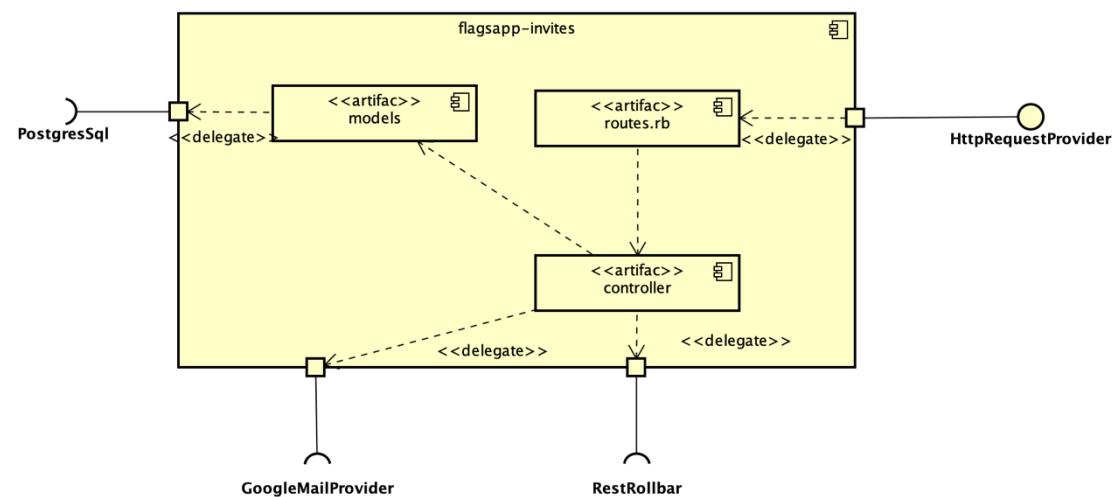


1.3.2. Decisiones de diseño

Las decisiones de diseño tomadas para esta aplicación son similares a las expresadas en la aplicación Flags, debido a que esta también es una aplicación Ruby on Rails, por lo cual no se puede realizar muchas variaciones dado que el desarrollo de la misma está fijado por el patrón MVC. Si podemos destacar que debido a que la misma fue implementada como una Web API, la misma no renderiza vistas, solo retornara los datos en formato JSON que son solicitados al controlador. A excepción de la vista “invite_mailer” que se utiliza para especificar el HTML que será enviado en los mails de invitaciones.

Sin tomar en cuenta las vistas, el resto de los manejos internos tanto de modelos, controladores y validaciones se realizaron siguiendo los mismos lineamientos anteriormente planteados.

1.3.3. Vista de componentes y conectores



Interfaces:

Elemento	Responsabilidades
RestRollbar	Interfaz que registra los errores que suceden en el sistema
PostgresSQL	Se le solicita el almacenamiento de los datos en la BD
GoogleMailProvider	Interfaz encargada de enviar mails, se invoca la misma para enviar los mails de invitación a los usuarios deseados.

Catalogo de elementos:

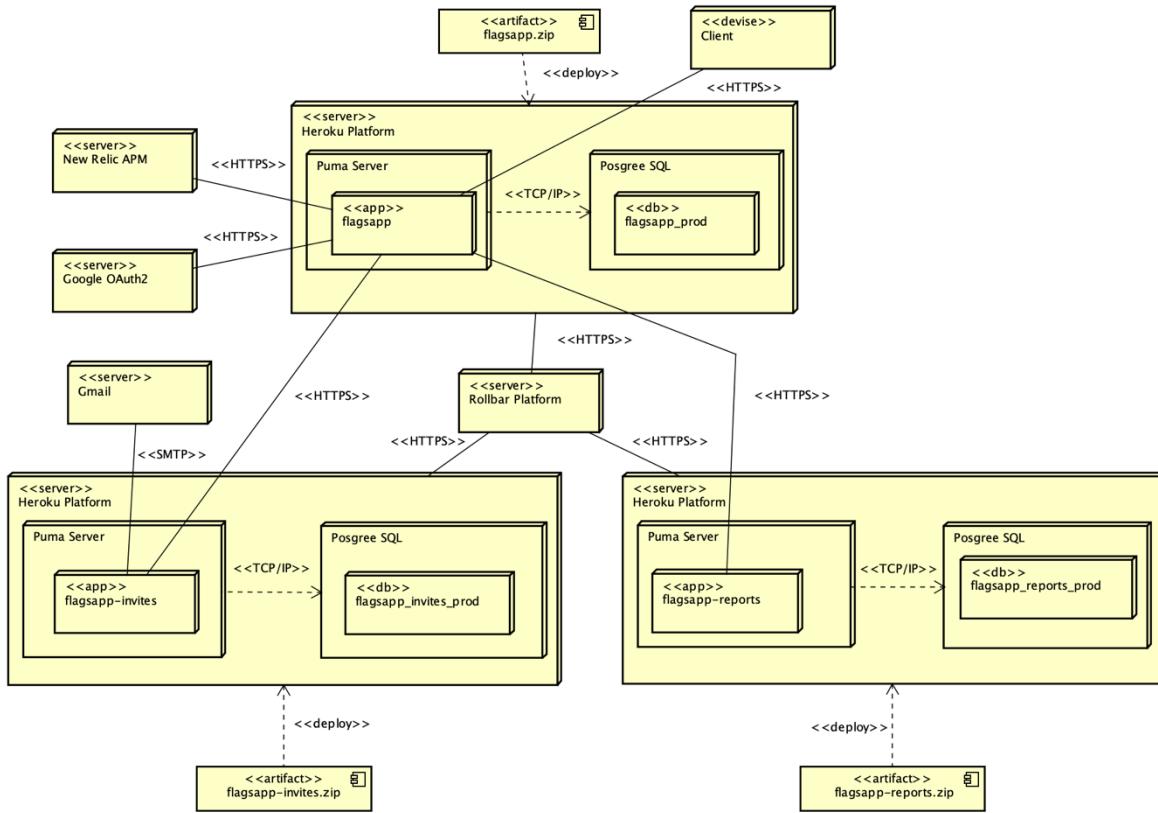
Elemento	Tipo	Responsabilidades
routes	.rb	Encargado de exponer las rutas para quien lo consuma y redirigirlo a los métodos indicados
controller	.rb	Es el encargado de realizar la lógica del sistema, y enviar la información solicitada mediante una respuesta http.
models	.rb	Modela las entidades que se encuentran en la BD

Se optó por separar la funcionalidad de invitar usuarios por razones similares a las anteriormente planteadas en reportes, en este caso también se tomó en cuenta la escalabilidad que podría llegar a tener los mecanismos de invitación de usuarios. En un futuro podrían requerirse múltiples sistemas de invitación diferentes o validaciones alternativas, las cuales cargarían al proyecto principal innecesariamente.

Además de la clara carga que sufre un servidor a la hora de tener que enviar múltiples mails y distintas decisiones futuras que pueden surgir como cambiar los servidores de POP3, SMTP, etc.

Tener estas funcionalidades separadas nos permite implementar nuevos mecanismos de invitación en cualquier momento sin afectar directamente el funcionamiento principal y crítico de FlgsApp, esto fue crucial a la hora de optar por separar este servicio de la aplicación principal.

1.4. Vista de despliegue



Los tres sistemas fueron desplegados en Heroku, en esta también se desplegó la base de datos PostgreSQL correspondiente de cada servicio. Los clientes tanto externos como internos del sistema accederán desde un navegador web realizando requests HTTP a nuestra aplicación.

FlagsApp se encargará de comunicarse con las aplicaciones externas que consume, Google OAuth para autenticar a los usuarios, NewRelicAPM con el fin de medir los tiempos de las request, RollBar para el control de errores, invitaciones app para enviar mails de registro, y reports para la obtención y actualización de los mismos.

Las conexiones se realizan mediante requests HTTP sin diferenciar si se trata de una aplicación externa a Heroku o una desplegada en él. Debido a que el deploy de las tres aplicaciones implementadas fue realizado en diferentes DYNOS de Heroku, por lo cual no es posible comunicarse entre ellas de forma local, dado que se encuentran en diferentes segmentos de red internos de Heroku.

2. Justificaciones de diseño

A continuación se pasara a detallar las decisiones de diseño tomadas para poder cumplir con cada uno de los RNFs (requerimientos no funcionales) solicitados.

2.1. RF9. SDK para evaluación de “Flags”

“Se deberá disponibilizar un SDK (lenguaje a elección) que permita consultar por la operación de evaluación de flag (RF7). El SDK debe conservar en caché por 2 minutos la respuesta. Además, en caso de que no se pueda conectar con el servicio de Flags o existan demoras, el SDK debe retornar “falso” al evaluar una flag en un tiempo menor a 1000 ms.”

Justificación:

Para la realización del RF9 se opto por implementar un proyecto en C#, de formato librería de clases el cual al compilarse se generara un archivo .dll. Este archivo puede ser importado en cualquier proyecto C#, permitiendo importar las funciones de evaluación para su uso.

Con el fin de cumplir con los requerimientos especificados se utilizo la librería provistas por C# para realizar requests HTTP

```
private string DoRequest(string requestUrl, string userId)
{
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(requestUrl);
    request.Headers["client_id"] = userId;
    request.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate;

    request.CachePolicy = policy;
    using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
    using (Stream stream = response.GetResponseStream())
    using (StreamReader reader = new StreamReader(stream))
    {
        return reader.ReadToEnd();
    }
}
```

Para el manejo del cache de forma personalizada se utilizo la librería `HttpRequestCachePolicy`, la cual permite personalizar el tiempo que se guarda el mismo. Este fue seteado por defecto al crear la clase evaluación con una duración de dos minutos como fue solicitado en el requerimiento.

```
0 references
public EvaluateFlag(string url)
{
    this.Url = url;
    this.policy = new HttpRequestCachePolicy(HttpCacheAgeControl.MaxAge,
        TimeSpan.FromMinutes(2));
    resultOfRequest = false;
}

0 references
public EvaluateFlag()
{
    this.Url = DEFAULT_URL;
    this.policy = new HttpRequestCachePolicy(HttpCacheAgeControl.MaxAge,
        TimeSpan.FromMinutes(2));
    resultOfRequest = false;
}
```

En la clase se proveen dos constructores, uno con parámetros y otro sin. En el constructor sin parámetros se setea la URL de `FlagsApp` por defecto hardcodeada en el mismo.

```
0 references
public string Url { get; set; }
private static readonly string DEFAULT_URL = "http://flagsapp-ort.herokuapp.com";
```

El segundo constructor permite indicar la URL a utilizar, esto se realizo para contemplar futuros cambios de URL que pueda poseer el servicio. Debido a que se trata de un SDK que se utilizara como un .dll ,no es posible dejar un archivo configuración para setear esta URL, debido a que no seria accesible por quien importe la librería.

Luego para cumplir con la especificación que ante cualquier falla el sistema debe retornar como respuesta false

```
0 references
public void RunEvaluate(string flagToken, string userId)
{
    bool returnValue = false;
    string requestUrl = this.Url + "/flags/" + flagToken + "/evaluate";
    try
    {
        string result = DoRequest(requestUrl, userId).Split(':')[1];
        string value = result.Remove(result.Length - 1);
        if (value == "true")
        {
            returnValue = true;
        }
    }
    catch (Exception)
    {
        returnValue = false;
    }

    lock (locker)
    {
        resultOfRequest = returnValue;
    }
}
```

Se controla cualquier posible error que suceda en el código y ante cualquier dificultad se retorna automáticamente una respuesta falsa.

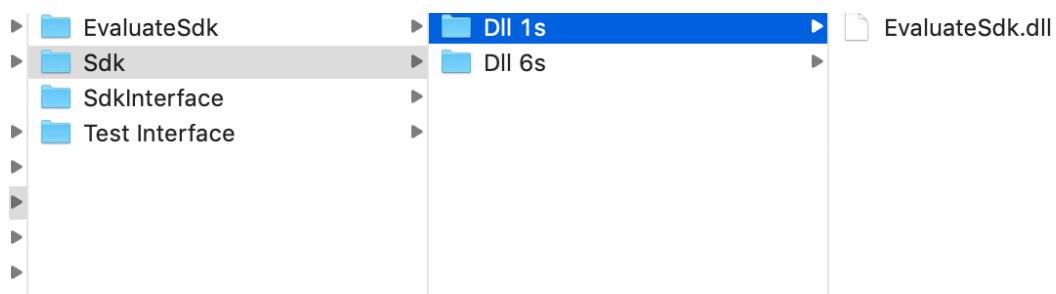
Como parte final del requerimiento se controlo que ante cualquier consulta que tarde mas de 1000ms se retorne false como respuesta, para esto se lanza un THREAD separado que realizara esta acción. Este es monitoreado y en caso de que este viva mas de 1000ms la operación es abortada y se retorna automáticamente false.

```
0 references
public bool Evaluate(string flagToken, string userId)
{
    Thread t = new Thread(() => RunEvaluate(flagToken, userId));
    t.Start();
    bool result = false;
    if (!t.Join(TimeSpan.FromMilliseconds(6000)))
    {
        t.Abort();
        result = false;
    }
    else
    {
        lock (locker)
        {
            result = resultOfRequest;
        }
    }
    return result;
}
```

Debido a que se sobrescriben variables globales dentro de diferentes THREAD de ejecución se implemento un sistema de LOCKs con el fin de evitar la generación de valores inconsistentes sobre esta misma. Respetando la correcta forma de controlar concurrencia y programación multithread en el lenguaje utilizado.

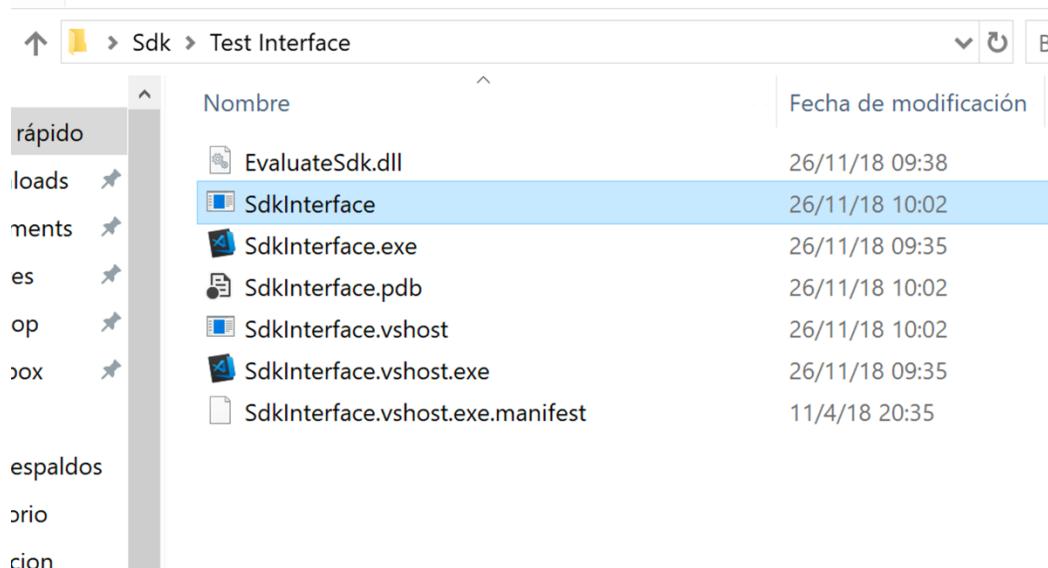
En el repositorio entregado se podrá encontrar dentro de la carpeta SDK dos carpetas, las cuales contienen en su interior una .dll, estas son EvaluateSdk.dll. Este archivo .dll será el SDK en cuestión, que se deberá importar en los proyectos que deseá ser utilizado.

Se proveen dos implementaciones debido a que una consta de una demora de 1000ms para realizar la request antes de abortar el envío y la otra de 6000ms. Se optó por entregar estas dos versiones por si se desea probar el sistema en conexiones de internet deficientes como las existentes en la Universidad ORT Montevideo, permitiendo obtener un mínimo de hándicap para que no falle la ejecución en todos los casos de prueba.

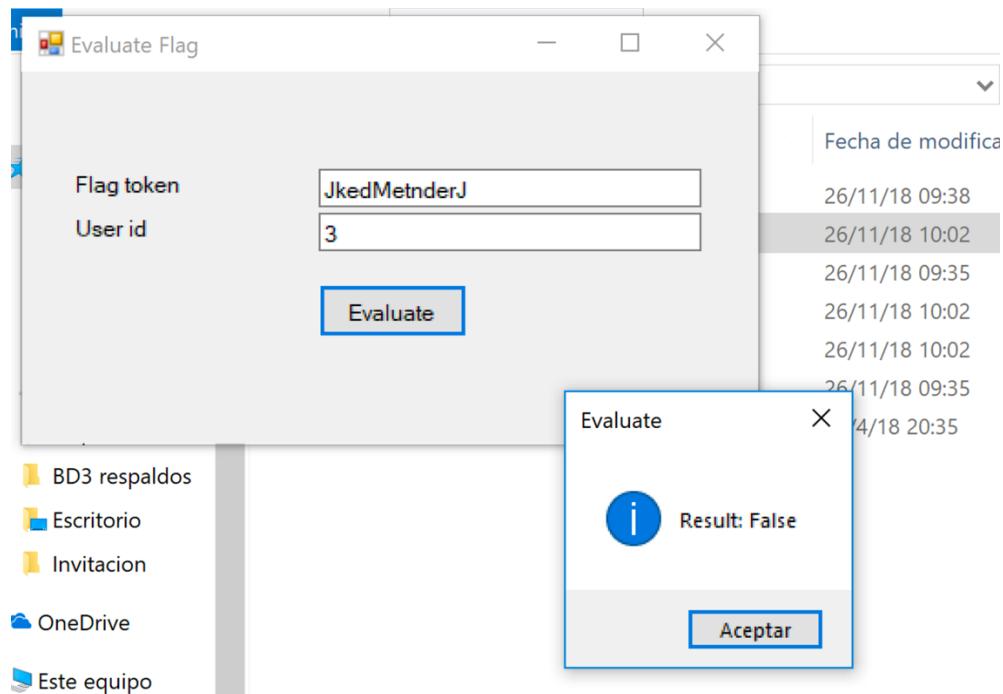


Por último también se provee un ejecutable de formato WindowsForm, en la carpeta TestInterface, el cual permite probar el correcto funcionamiento de la librería, en este caso se evaluara en la dirección saeteada por defecto.

Para probar la librería bastara con ejecutar en un equipo con sistema operativo Windows el ejecutable SdkInterface.exe



Este desplegará una ventana de escritorio que permitirá ingresar el token de la flags a evaluar y el id del usuario que desea probarlo, al presionar evaluar se desplegará la respuesta de la evaluación.



En el repositorio entregado se encuentra adjuntado el código fuente de las dos aplicaciones desarrolladas (Test interface y EvaluateSdk), así como los SDK creados en cuestión.

2.2. RNF1. Performance

"El tiempo de respuesta promedio para todas las operaciones públicas del sistema en condición de funcionamiento normal deberá mantenerse por debajo de los 100 ms. para cargas de hasta 1200 req/m."

Justificación:

Con el fin de lograr cumplir con los tiempos solicitados se busco modelar la solución de manera que las solicitudes a la base de datos sean mínimas, evitando la realización de joins innecesarios de tablas. Para lograr esto se priorizo en evitar separar en múltiples tablas la información referente a los usuarios y las flags, por mas que estas presenten optionales (caso usuario si es administrador y caso flag su tipo) se opto por dejar atributos optionales en ellos, con el fin de evitar realizar múltiples consultas a la hora de buscarlos.

	id [PK] bigint	name character varying	style_flag integer	active boolean	last_update timestamp without time zone	is_deleted boolean	percentage integer	auth_token character varying	organization_id bigint
1	2	test Boolean	1	true	2018-10-18 22:03:38.311883	false	[null]	nLMU5hnxkZW2RRB...	6
2	3	Test Percentage	2	true	2018-10-18 22:03:53.803582	false	50	SbMPQdnhdqL8gcU...	6
3	4	Test list	3	true	2018-10-18 22:04:08.661926	false	[null]	S6VqTisyfafGFHuEeQ...	6

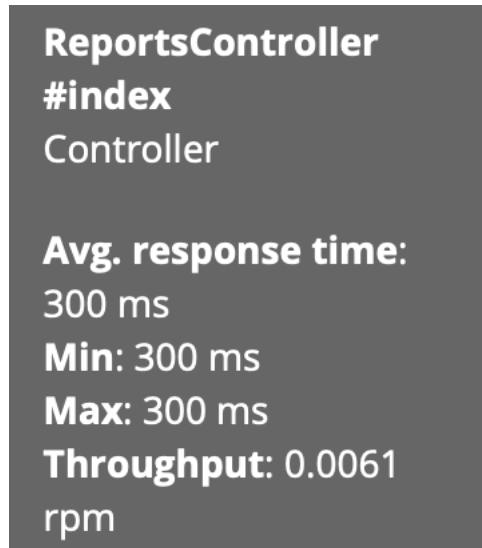
Como se puede observar en la siguiente tabla de flags, se presenta el atributo porcentaje como opcional para las flags que aplique.

También se opto por abstraer la funcionalidad de reportes a un micro servicio, separando los cálculos y procesos necesarios para el mismo a un sistema diferente, optimizando el funcionamiento de FlagsApp desligándolo de procesar los mismo.

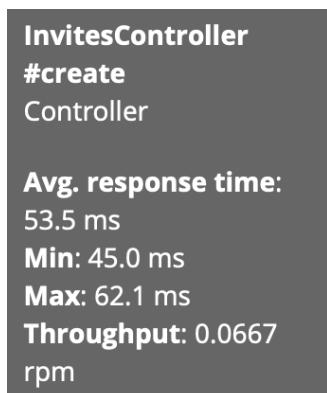
Dentro la nueva aplicación reportes se guardara en una única tabla todas las veces que se consulta por una flag a la hora de evaluarla, esto permite que al consultar sus estadísticas esta operación sean solo traer los datos únicos de la misma. Debido a que esta información se encuentra en un sistema aparte, el identificador que distinguirá a la flag será un token de identificación BASE64.

	id [PK] bigint	total_request integer	true_answer integer	false_answer integer	flag character varying
1	1	0	0	0	YXNKTkVSVU5WVUV...

Esto nos permitió balancear el tiempo que tardan las operaciones, disminuyendo la demora al evaluar una flag debido a que se abstrae el guardado de estos valores a otra aplicación. Logrando una mejora substancial en el funcionamiento general de la aplicación.



Al realizar pruebas con la herramienta NewRelic, los tiempos que toma realizar estas operaciones. En la primer imagen se puede observar lo explicado en el párrafo anterior al ver el tiempo que tomo realizar la consulta de los reportes, la cual tomo 300ms la cual es un tiempo mas que razonable teniendo en cuenta que para acceder a estos debemos realizar consultas al micro servicio de Reportes.



En la parte de invitaciones podemos ver que a pesar de tener que realizar consultas al micro servicio de Invitaciones se logro un tiempo de respuesta de 53.5ms, lo cual es un tiempo excelente considerando lo anteriormente descripto.

2.3. RNF2. Confiabilidad y disponibilidad

“Con el fin de poder monitorear la salud y disponibilidad del sistema, se deberá proveer un endpoint HTTP de acceso publico que informe el correcto funcionamiento del sistema (conectividad con bases de datos, colas de mensajes, disponibilidad para recibir requests, etc.). El mismo deberá responder a GET /healthcheck con status 200 OK, siendo cualquier otro código de respuesta considerando como una disrupción en el servicio. No se requieren formatos particulares para el cuerpo de la respuesta.”

Justificación:

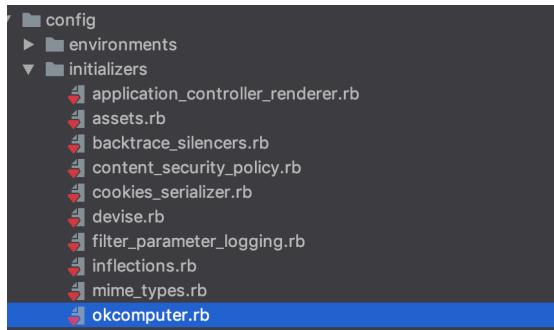
Para poder monitorear la salud y disponibilidad del sistema se optó por utilizar la gema “ok_computer” esta nos permite monitorear la conectividad con la base de datos, y la disponibilidad de recibir request.

Esta responde con un código 200 OK, y en el cuerpo del mensaje un mensaje descriptivo de si existe conectividad o no. La URI de este endpoint fue declarada de acuerdo al requerimiento.

```
Default Collection
  database: PASSED Schema version: 20181018153411 (0s)
  default: PASSED Application is running (0s)
```

La misma también permite configurar controles propios, los cuales no fueron utilizados para esta entrega debido a que el default ofrecido por la misma controla todas las estructuras utilizadas en la implementación. Pero si a futuro se implementan colas, caches u otros tipos de estructuras se podrá configurar la misma para que realice testeos sobre estos.

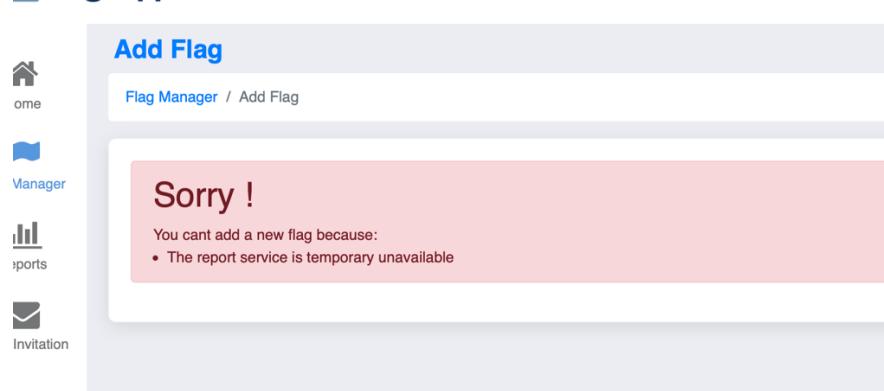
Con el fin de dejar preparado esto se generó un archivo de configuraciones propias para esta gema, el cual podrá ser útil a futuro.



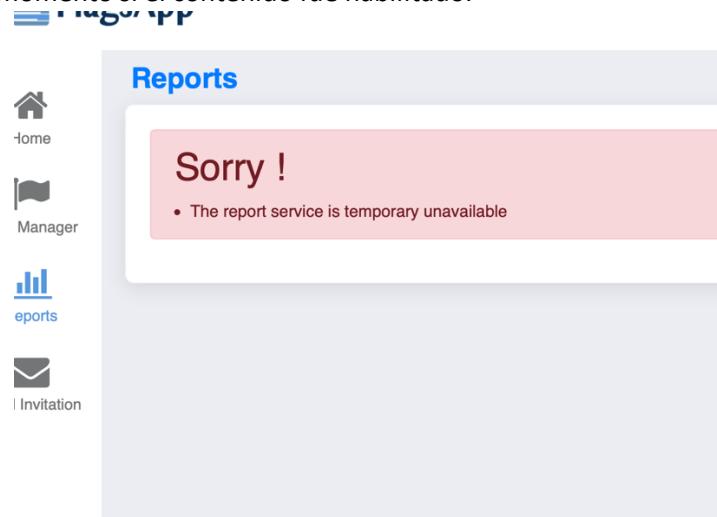
Esta fue instalada en las tres aplicaciones desarrolladas, permitiendo el monitoreo particular de cada una de las aplicaciones. La aplicación FlagsApp utiliza la ruta /healthcheck que posee cada micro servicio para verificar si una operación que requiera los mismos puede realizarse.

Como política de disponibilidad se optó por consultar mediante la ruta /healthcheck a la aplicación reportes cuando se crea una flag, se realiza una evaluación y se consultan los reportes:

- En el caso que se encuentre caída a la hora de crear una flag, no se permitirá al usuario crearla y se le mostrará un mensaje de error, esto se debe a que no sería correcto la creación de una flag que a futuro será imposible registrarle reportes, generaría una creación inconsistente en el sistema. El usuario podrá crear la misma en otro momento pero no crear flags inconsistentes.

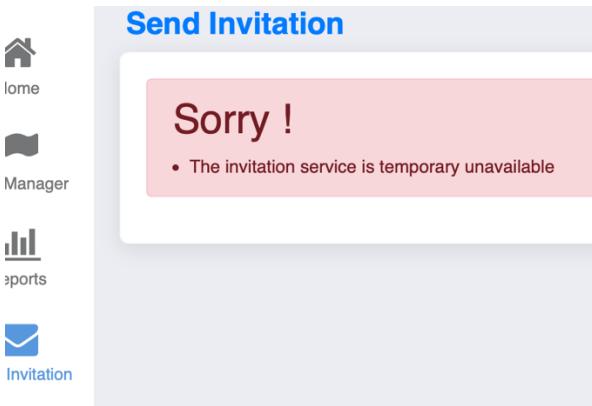


- Si el sistema se encuentra caído durante una evaluación, se responderá correctamente la evaluación ignorando el registro de esta acción, debido a que esta funcionalidad es crítica para usuarios externos al sistema, los cuales no deben verse afectados por fallas internas. El costo de no registrar esa evaluación será mínimo, debido a que en grandes cantidades de datos, las estadísticas tenderán a valores razonables sin evidenciar una diferencia al no registrarse un valor. La caída de este sistema no afecta a la correcta evaluación de una flag, por lo cual el resultado que obtendrá el usuario será correcto.
- De estar caído en el caso de solicitar los reportes se presentara un mensaje de error indicando que no es posible visualizar este contenido por el momento, se podrá consultar en cualquier momento si el contenido fue habilitado.

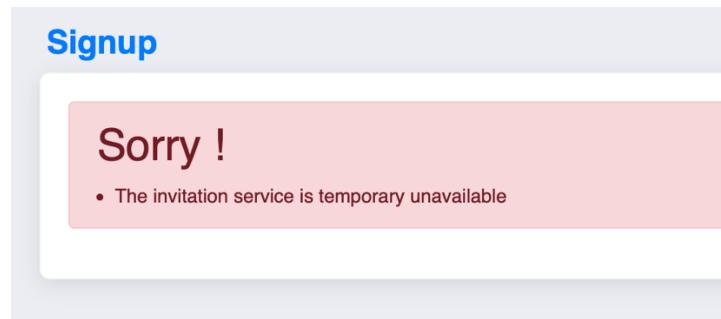


De igual forma que como se realiza con el microservicio de reportes, se decidió consultar a la ruta “/healthcheck” para verificar la disponibilidad del micro servicio invitaciones.

- En caso de que se encuentre caída a la hora de enviar una invitación, se muestra un mensaje de error indicando que no es posible enviar la misma por el momento, se podrá intentar en cualquier otro momento si la disponibilidad de este servicio se recupera.



- Si el micro servicio se encuentra caído a la hora de intentar registrarse por medio de una invitación muestra un mensaje de error, ya que no es posible verificar si la invitación se trata de una invitación autentica o no, además de no saber a que organización corresponde la invitación en cuestión.



2.4. RNF3. Configuración y manejo de secretos

“Relativo al desarrollo, todo dato de configuración sensible que se maneje en el código fuente deberá poder especificarse en tiempo de ejecución mediante variables de entorno, manteniendo todo valor fuera del repositorio. Esto incluye credenciales de acceso a APIs, URLs de servicios externos, y cualquier otra configuración específica del sistema.”

Justificación:

Para el cumplimiento de este requerimiento el equipo decidió utilizar la gema “Figaro” esta nos permite definir en un archivo llamada “application.yml” todo los valores de las variables de entorno que utilizamos en distintos ambientes (es decir desarrollo y producción).

```

development:
  INVITES_URL_SERVICE: "http://invites-flagsapp.herokuapp.com"
  INVITES_TOKEN_SERVICE: "SaMrV7mEkKhxAXZ7Sz"
  REPORTS_URL_SERVICE: "http://reports-flagsapp.herokuapp.com"
  REPORTS_TOKEN_SERVICE: "8Ve9taFh848EdyDFDb"
  GOOGLE_CLIENT_ID: "1002347430954-97h7o40ada37qh6gav2v810efkjvalkp.apps.googleusercontent.com"
  GOOGLE_CLIENT_SECRET: "IIaY-5h4VA2--t2AmB2Xfdy"
  ROLLBAR_ACCESS_TOKEN: "fc9ed72da9b7440bb137f1980f38949b"

```

Se intento que toda configuración sensible sea extraída fuera del código mediante este archivo de configuración.

Como por ejemplo la configuración del ActionMailer que se muestra a continuación:

```

ActionMailer::Base.smtp_settings = {
  :address           => "smtp.gmail.com",
  :port              => 587,
  :user_name         => ENV['GMAIL_USERNAME'],
  :password          => ENV['GMAIL_PASSWORD'],
  :authentication   => "plain",
  :enable_starttls_auto => true
}

```

La cloud que se decidió utilizar (Heroku) también nos permite sobrescribir estas variables mediante la consola de comandos. Esta política fue adoptada en las tres aplicaciones desarrolladas.

2.5. RNF4. Autenticación y autorización

“Se deberá establecer un control de acceso basado en roles, distinguiendo entre usuarios administradores y usuarios de una organización. Los permisos otorgados a los mismos deberán restringir el acceso a sus correspondientes funcionalidades, prohibiendo la interacción con cualquier otra operación pública del sistema.”

Justificación:

Se definió un protocolo de seguridad del tipo RBAC (Basado en roles), utilizando la gema “devise”, esta nos facilita el manejo de usuarios, como decisión de diseño se optó por manejar una única tabla de usuario donde se definió un atributo del tipo booleano que se especifica si es administrador; esta decisión fue principalmente por que no existía ningún tipo de atributo que diferencie a un administrador de un usuario común.

Para la restricción de los permisos otorgados se utilizaron distintas instrucciones que nos provee la gema para validar antes de renderizar cualquier acción de un controller que este sea un usuario valido.

```

class FlagsController < ApplicationController
  before_action :authenticate_user!, only: [:index, :show]
  before_action :authenticate_admin!, only: [:new, :create, :change, :destroy]

  def authenticate_admin!
    if current_user == nil
      redirect_to home_url
    else
      redirect_to home_url unless current_user.is_admin
    end
  end
end

```

2.6. RNF5. Seguridad

“El sistema deberá responder con código 40X a cualquier request mal formada o no reconocida por el mismo, no dejando expuesto ningún endpoint que no sea explícitamente requerido por alguna funcionalidad.”

Justificación:

Para lograr exponer únicamente las rutas de las funcionalidades que fueron desarrolladas se específico dentro de la definición de cada resource únicamente las rutas que se utilizaron, y dentro de las gemas utilizadas que se deben definir rutas se definieron únicamente las rutas que se irían a utilizar (como por ejemplo en la gema “devise” y “ok_computer”).

```

resources :reports, :only => [:index, :show]
resources :invites, :only => [:create, :new]

```

También se excluyeron ciertas rutas como por ejemplo las que no se utilizaban de la gema devise y se definieron explícitamente las rutas que si se utilizan de esta gema

```
devise_for :users,
  controllers: { sessions: 'users/sessions', registrations: 'users'
    skip: [:registrations]
  }
as :user do
  get 'users/sign_up', to: 'users/registrations#new', as: 'new_user_registration'
  post 'users', to: 'users/registrations#create', as: 'user_registration'
end
```

También se controla que los endpoint provistos para uso en apis externas validen los datos que reciben antes de realizar las operaciones requeridas:

```
def evaluate
  flag = Flag.where(auth_token: params[:id]).first
  return render json: { data: 'Error flag not found' }, status: 400 if flag.nil?
```

En la imagen se puede observar como, de no obtenerse información valida al realizar la evaluación se devuelve un 400.

Esta política fue adoptada en las tres aplicaciones desarrolladas, realizando principal hincapié en las validaciones referentes a los valores enviados a los micro servicios, debido a que la información que estos manejan ya no son claves foráneas de tablas que puedan validarse internamente, pasan a ser referencias a valores de otros sistemas. Esto requiere un mayor control de la información manejada en las aplicaciones.

```
def update_result
  token = params[:token]
  result = params[:result]
  return render json: ['The flag evaluate result is NULL'], status: :bad_request if result.nil?

  report = Report.find_by(flag: token)
  return render json: ['The report for flag ' + token + ' not found'], status: :bad_request if report.nil?

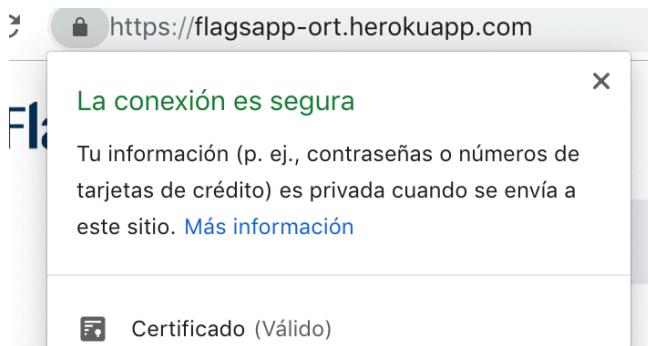
  report.total_request = report.total_request + 1
```

Lo expresado anteriormente se puede observar en la presente imagen tomada del micro servicio reportes, antes de realizar un update de los datos registrados se valida que la información enviada sea correcta.

“A su vez, 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 en lo posible realizarse dentro de una red de alcance privado; de lo contrario deberán también utilizar un protocolo de transporte seguro autenticados por una clave de autenticación.”

Justificación:

La comunicación entre los clientes del front end y los componentes del back end son realizados mediante un protocolo HTTPS que nos provee la Cloud en donde se encuentra desplegada la aplicación.



La comunicación entre componentes del back-end y front-end es realizada dentro de la misma red, ya que FlagsApp es una aplicación de tipo monolítica, a pesar de haberse extraído dos nuevos servicios de esta.

Las comunicaciones entre FlagsApp y los diferentes micro servicios se realizan utilizando el protocolo HTTPs descripto anteriormente, por lo cual estas también se realizan con los estándares de seguridad descriptos anteriormente.

Con el fin de mantener un control de las comunicaciones entre las tres aplicaciones implementadas, se optó por utilizar un token de verificación. Este es seteado en las variables de entorno de las tres aplicaciones. A la hora de comunicarse las aplicaciones envían este token para identificarse, y la aplicación que recibe la petición compara este con el que posee registrado internamente en sus variables de entorno.

```
INVITES_URL_SERVICE: "http://invites-flagsapp.herokuapp.com"
INVITES_TOKEN_SERVICE: "SaMrV7mEkKhxAZ7Sz"
REPORTS_URL_SERVICE: "http://reports-flagsapp.herokuapp.com"
REPORTS_TOKEN_SERVICE: "8Ve9taFh848EdYDFDb"
```

Una vez que la request llega al sistema Rails se encargará de manejar de forma segura la información, debido a que el resto de los manejos serán internos.

2.7. RNF6. Código fuente

“El sistema deberá ser desarrollado con el lenguaje Ruby, utilizando el framework Ruby on Rails para las funcionalidades de acceso web. Para mejorar la mantenibilidad del proyecto, todo el código y comentarios deberán ser escritos en inglés, siguiendo una guía de estilos de Ruby a determinar y chequeados con Rubocop.”

Justificación:

Con el fin de cumplir estos estándares la gema Rubocop fue instalada al comienzo del desarrollo de las tres aplicaciones, también fue configurado en el editor de texto utilizado para que nos alerte sobre estos errores a la hora de realizar la codificación.

```
28   def filter
29     @flags = evaluate_filter
30     render :index
31   end
32
33   def create
34     @flag = @flag_params
35     if @flag.is_deleted = false
36       @flag.last_update = DateTime.current
37       @flag_record = @flag.flag_records.new date_start: Date.current, active: @flag.active, date_end: Date.parse('1900-01-01')
38       @flag_record.save
39       if @flag.save
40         redirect_to flags_path
41       else
42         render :new
43       end
44     end
45   end
46 end
```

También se modificó el archivo config de esta gema para que muestre solo los errores de estilo

que realmente importaban, esto fue discutido con el profesor para llegar a un consenso de lo que realmente se evaluaría. Se excluyó también los archivos que no dependían del equipo de desarrollo como las configuraciones básicas del proyecto pre establecidas.

```
- 'app/controllers/users/*'

# Offense count: 10
Style/Documentation:
Exclude:
- 'app/controllers/*'
- 'app/uploaders/*'
- 'app/controllers/users/*'
- 'app/models/*'
- 'db/migrate/*'
- 'spec/**/*'
- 'test/**/*'
- 'app/helpers/invites_helper.rb'
- 'app/helpers/application_helper.rb'
- 'app/helpers/reports_helper.rb'
- 'app/mailers/invite_mailer.rb'
- 'app/helpers	flags_helper.rb'
- 'app/helpers/home_helper.rb'
- 'app/mailers/application_mailer.rb'
- 'config/application.rb'
```

Puede observarse en la imagen la configuración utilizada en este archivo, luego de todas las configuraciones mencionadas anteriormente se logró un total de 10 offenses en FlagsApp, las cuales no pudieron ser corregidas debido a que al poseer demasiados tipos de flags especificados en el sistema, al evaluar estas se deben utilizar un gran número de condicionales (if) que elevan la “Assignment Branch Condition”.

```
app/controllers/flags_controller.rb:10:27: E: Lint/Syntax: unexpected token tSYMBOL
(Using Ruby 2.5 parser; configure using TargetRubyVersion parameter, under AllCops)
    !healthcheck ? render :healthcheck_report_fail : @flag = Flag.new
                           ^^^^^^^^^^
app/controllers/reports_controller.rb:9:3: C: Metrics/AbcSize: Assignment Branch Condition size for index is too high. [17.52/15]
def index ...
^
app/controllers/reports_controller.rb:9:3: C: Metrics/MethodLength: Method has too many lines. [15/10]
def index ...
^
app/controllers/reports_controller.rb:27:3: C: Metrics/MethodLength: Method has too many lines. [12/10]
def show ...
^
app/controllers/reports_controller.rb:41:1: C: Layout/EmptyLinesAroundClassBody: Extra empty line detected at class body end.
app/controllers/users/registrations_controller.rb:28:5: C: Style/RedundantBegin: Redundant begin block detected.
begin
^
app/controllers/invites_controller.rb:17:7: W: Lint/UselessAssignment: Useless assignment to variable - invite.
  invite = create_invite(organization_id, sender_id, email)
^
87 files inspected, 10 offenses detected
MBPdeSantiago2:Arq santiagomarchisio$
```

La mayoría de estos errores suceden debido a este problema, los cuales no son puramente de estilo de código, son más de formato de solución.

Se utilizaron configuraciones similares a la de FlagsApp en los dos nuevos micro servicios implementados. Gracias a la correcta utilización de las reglas de estilo y el pequeño tamaño que ambos micro servicios poseen, se pudo obtener un resultado satisfactorio al realizar la prueba de estos proyectos con la herramienta planteada, obteniendo 1 solo offense en el microservicio de invitaciones y ninguno en el de reportes

```
Last login: Tue Nov 27 02:22:24 on ttys000
[MBPdeSantiago2:~ santiagomarchisio$ cd /Users/santiagomarchisio/Desktop/Invitacion
[MBPdeSantiago2:Invitacion santiagomarchisio$ rubocop
Inspecting 44 files
.....E.....  
Offenses:  
app/controllers/invites_controller.rb:9:83: E: Lint/Syntax: unexpected token tLABEL
(Using Ruby 2.5 parser; configure using TargetRubyVersion parameter, under AllCops)
    errors.empty? ? create_invitation(sender_id, organization_id, email) : render json: errors, status: :bad_request
                                         ^^^^^^  
44 files inspected, 1 offense detected
[MBPdeSantiago2:Invitacion santiagomarchisio$
```

Este no pudo ser eliminado debido a que incidía directamente con la funcionalidad deseada del sistema.

```
Last login: Tue Nov 27 02:30:22 on ttys000
[MBPdeSantiago2:~ santiagomarchisio$ cd /Users/santiagomarchisio/Desktop/Reportes
[MBPdeSantiago2:Reportes santiagomarchisio$ rubocop
Inspecting 41 files
.....  
41 files inspected, no offenses detected
[MBPdeSantiago2:Reportes santiagomarchisio$
```

“Por otra parte, el control de versiones del código se deberá llevar a cabo con repositorios Git debidamente documentados, que contengan en el archivo README.md una descripción con el propósito y alcance del proyecto, así como instrucciones para configurar un nuevo ambiente de desarrollo. Para el manejo de branches, se deberá utilizar Gitflow.”

Justificación:

El control de versiones del código fue llevado a cabo mediante 4 repositorio en Github, uno para la aplicación monolítica, otros dos para el servicio de invitaciones y reportes, y otro para el SDK de la aplicación. Todos estos se manejaron utilizando un esquema de desarrollo Gitflow. Se definieron 2 branches “develop”, y “master”

- [master]: Contiene el código en producción
- [develop]: Contiene el código en desarrollo
- [feature/nombre]: Contiene el código correspondiente a una feature de la aplicación.

Dentro de cada repositorio se encuentra un README en la branch master que especifica, pre requisitos, desplegado de la aplicación en distintos ambientes, gemas utilizadas, y un esquema de cómo se lleva a cabo el control de versionado.

FlagsApp 2.0

Este proyecto consta en el desarrollo de una app en Ruby on Rails, para poder cumplir con los requerimientos especificados por el cliente.

En esta segunda versión de la aplicación el cliente nos solicitó orientar la aplicación a una arquitectura de microservicios debido a los problemas de escalabilidad en el mismo, producto del gran éxito de la anterior versión.

Introducción

Estas instrucciones te van a permitir poder correr el proyecto tanto en un ambiente local para testing y develop, como en un ambiente de producción.

La documentación se encuentra en el siguiente link [click aquí](#)

Pre requisitos

Para poder ejecutar Flagsapp es necesario tener instalado las siguientes herramientas

- [Ruby 2.5.1P57](#)
- [Rails 5.2.0](#)
- [PostgreSQL 10.5](#)
- [ImageMagick 7.0.8](#)

Despliegado development y test

A continuación iremos a explicar paso por paso como hacer para poder ejecutar FlagsApp en un ambiente de desarrollo.

1- Configurando variables de entorno Primero que nada debemos configurar las variables de entorno en el archivo que se encuentra en [config/application.yml](#) dentro de la sección development las variables que debemos configurar son las siguientes:

```
development:
  INVITES_URL_SERVICE: "http://invites-flagsapp.herokuapp.com" # URL del microservicio de invitaciones
  INVITES_TOKEN_SERVICE: "oneToken" # Token de autenticación del microservicio de invitaciones
  REPORTS_URL_SERVICE: "http://reports-flagsapp.herokuapp.com" # URL del microservicio de reportes
  REPORTS_TOKEN_SERVICE: "oneToken" # Token de autenticación del microservicio de reportes
  GOOGLE_CLIENT_ID: "oneClientId" # Client ID de las credenciales para Google OAuth
  GOOGLE_CLIENT_SECRET: "oneSecretId" # Secret de las credenciales para Google OAuth
  ROLLBAR_ACCESS_TOKEN: "oneRollbarToken" # Token del servicio Rollbar
```

2- Instalando las Gemas utilizadas Lo segundo que debemos hacer es instalar las gemas que utilizamos corriendo el siguiente comando

2.8. RNF7. Evaluación de “flags” en formato JSON

“Con el fin de evaluar una “flag” dado un usuario, se deberá brindar una API RESTful mínima que permita consultar por una “flag” particular dado un usuario y retornar el resultado así como también un reporte de uso de la misma, todo en formato JSON.”

Justificación:

Para la evaluación y reporte de una flag se tienen disponibles los siguientes endpoints en FlagsApp:

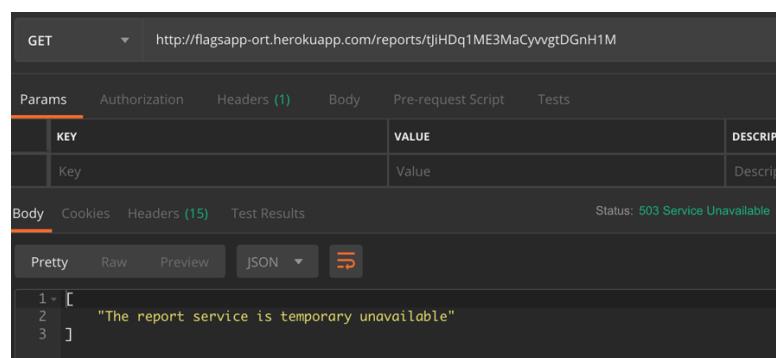
Verbo	Endpoint	Parametros	Retorno
GET	/flags/{token_flag}/evaluate	Header: client-id: ID del usuario	boolean
GET	/reports/{token_flag}	-	report object

La evaluación de una flag es un endpoint que nos retorna un booleano dependiendo de si esta evalua a true o false, y recibe por el header un parámetro que es el “client-id” el cual especifica el id del cliente que evalua la flag.

La generación de reporte de una flag retorna un objeto reporte que contiene diversos atributos como promedio tiempo de respuesta, total de request, total de request que evaluaron positivo, etc.

Como se explico anteriormente en cuando se discutió la seguridad, si alguno de los datos que reciben estos endpoints resulta ser invalido se responderá con un 400. También se explico en apartados anteriores la política de disponibilidad aplicada para estos endpoints, dependiendo del estado de los micro servicios requeridos.

Por ejemplo si el microservicio de reporte se encuentra caido este responderá con el siguiente mensaje en formato JSON



The screenshot shows a Postman request for a GET endpoint at <http://flagsapp-ort.herokuapp.com/reports/tJiHDq1ME3MaCyvvtDGnH1M>. The 'Params' tab is selected, showing a single parameter 'Key'. The 'Body' tab is selected, displaying a JSON response:

```
1  [
2    "The report service is temporary unavailable"
3  ]
```

2.9. RNF8. Pruebas

“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. De necesitar incluir claves o secretos, el script deberá recibir dichas configuraciones por variables de entorno.

Además de las pruebas de carga, se deberá contar con pruebas funcionales automatizadas para el RF 1 a nivel de sus controladores.”

Justificación:

Debido a los tiempos acotados no pudo realizarse pruebas exhaustivas de carga sobre el sistema, la complejidad y bajo servicio de la comunidad de la herramienta nos dificulto poder investigar y utilizar correctamente la misma.

De igual manera el equipo realizo numerosas pruebas funcionales y de integración sobre el sistema con el fin de garantizar el correcto funcionamiento. También fueron utilizadas herramientas alternas mas amigables para garantizar lo solicitado por el cliente como se planteo en el RNF-1

Estas pruebas fueron enfocadas en el correcto funcionamiento de los micro servicios por separado y luego tomando un enfoque de integración probando la aplicación directamente desde FlagsApp a través del sitio web.

▼ November 25

GET http://flagsapp-ort.hero
kuapp.com/flags/56Rqq
FZbqp1R2aATMjorAdp

▼ November 24

PUT https://reports-flagsapp.
herokuapp.com/report
s/JskCx0HPeDWrBdDq4

PUT https://reports-flagsapp.
herokuapp.com/report
s/Mpjm8RnU7BS35tq2p

GET https://reports-flagsapp.
herokuapp.com/report
s/Mpjm8RnU7BS35tq2p

GET http://localhost:3000/fla
gs/Mpjm8RnU7BS35tq2
pkyWWRKKG/evaluate

GET https://reports-flagsapp.
herokuapp.com/report
s/Mpjm8RnU7BS3dsdtq

PUT https://reports-flagsapp.
herokuapp.com/report
s/Mpjm8RnU7BS3dsdtq

GET https://reports-flagsapp.
herokuapp.com/flags/M
nim8RnlJ7BS35tn2nkvW

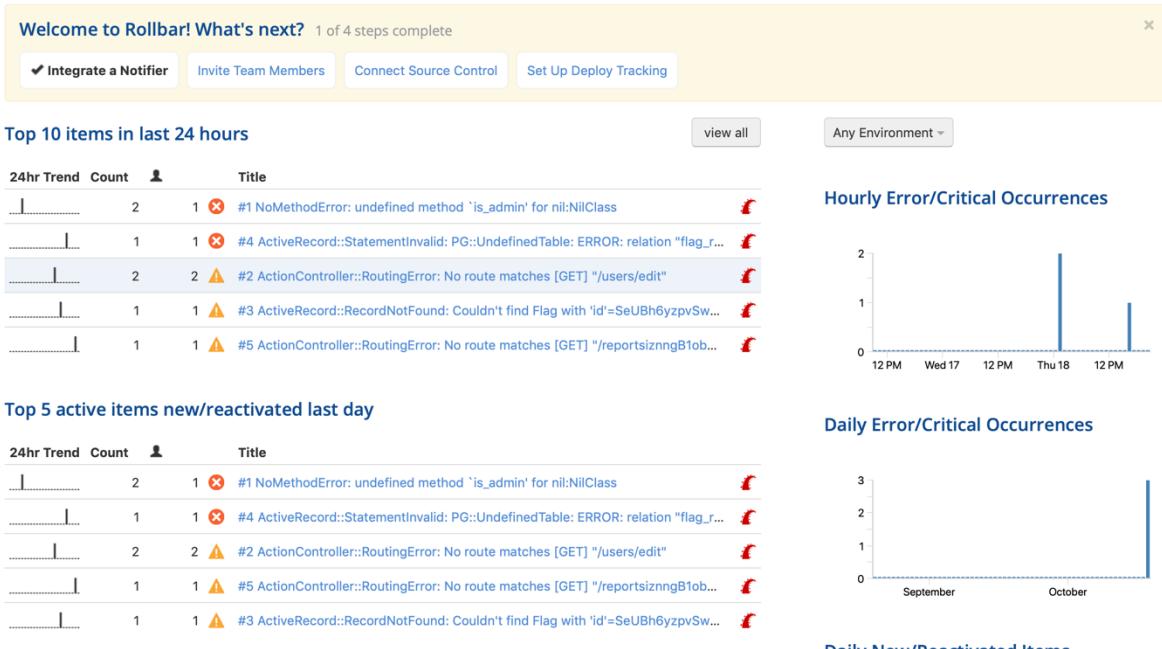
Se puede observar las pruebas realizadas con la herramienta postman sobre los endpoint.

2.10. RNF9. Identificación de fallas

“Para facilitar la detección e identificación de fallas, se deberán centralizar y retener los logs emitidos por la aplicación en producción por un periodo mínimo de 24 horas.”

Justificación:

Para el correcto control de fallas se utilizó la herramienta Rollbar, la cual nos permite obtener un registro de los errores que se presentaron en las distintas aplicaciones. Esta genera un log con todos los errores que suceden en el sitio, los almacena y ofrece una descripción detallada junto con un informe estadístico.



También envía una notificación vía mail al usuario encargado de la administración de errores, lo cual felicita el conocimiento en tiempo real de cualquier situación indeseada al equipo de desarrollo.

[flagsapp] production - New Error: #4 ActiveRecord::StatementInvalid: PG::UndefinedTable: ERROR: relation "flag_record" does not exist
LINE 2: FROM flags f, flag_records fr ^ : SELECT f.* FROM flags f, flag_records fr where fr.flag_id = f.id

Papelera X

Rollbar Notification <notifier@mail.rollbar.com> para mí ▾ 16:26 (hace 3 horas)

Traducir mensaje Desactivar p.

XXA inglés ▾ > español ▾ Project: flagsapp Environment: production Code Version: unspecified Host: f76717d5-4e79-4fba-b6ad-d3100032413e Timestamp: 2018-10-18 12:26 pm

A new item has occurred for the first time. View full details of the item at: <https://rollbar.com/marchisio.santiago/flagsapp/items/4/>. View the occurrence that triggered this notification at: <https://rollbar.com/marchisio.santiago/flagsapp/items/4/occurrences/56230683538/>

Message
ActiveRecord::StatementInvalid: PG::UndefinedTable: ERROR: relation "flag_records" does not exist LINE 2: FROM flags f, flag_records fr ^ : SELECT f.* FROM flags f, flag_records fr where fr.flag_id = f.id

URL
http://staging-flagsapp.herokuapp.com/filter?utf8=%E2%9C%93&style_filter=4&style_flag=1&name=&state=true&date=2018-10-19&state_date=true&commit=Filter

Traceback

Se optó por la instalación de esta herramienta en las tres aplicaciones desarrolladas, con el fin de llevar un desglose independiente de los errores que se generan, pudiendo analizarse y repararse de forma individual.

Debido a que FlagsApp consume los dos micro servicios, los errores que estos propaguen se verán registrados en FlagsApp también, pero el desglosé completo del mismo estará presente dentro del registro particular de cada micro servicio.

2.11. RNF10. Independencia de aplicaciones

Como fue expresado durante todo el documento se opto por separar la aplicación monolítica hereda de la primera entrega en tres aplicaciones diferentes, estas fueron FlagsApp, reports microservice y invites microservice.

FlagsApp continua siendo una aplicación monolítica, debido a que sigue poseyendo el front-end y back-end en ella. También se podría separar el modulo de logueo y control de usuarios que no deberían corresponder a la aplicación principal, con el fin de disminuir aun más el monolito implementado.

De igual forma se logro disminuir la carga dentro de FlagsApp, debido a las separaciones realizadas, como fue expresado a lo largo de todo el documento. Como mecanismo de comunicación entre las aplicaciones se utilizan request http, las cuales ofrecen una seguridad de encriptación propia del protocolo.

En un principio se intento utilizar WSO2 como gateway, con el fin de que las aplicaciones se comuniquen atreves de este. WSO2 permitía registrar las aplicaciones que requerían comunicarse, administrando las conexiones atreves de el, si un servicio requiere de otro este envía una request http a WSO2 solicitando por el mismo. WSO2 redirige la solicitud al servicio requerido, gestionando así las direcciones, también se encarga de verificar que la comunicación se realice entre aplicaciones validas colocando claves y verificándolas en las request.

La solución implementada con WSO2 tuvo que ser desestimada, debido a que el mismo ofrecía una versión de prueba por un tiempo limitado de 15 días, menor al mínimo requerido para que se realice la corrección del obligatorio.

Luego se opto por implementar nuestro propio servicio de gateway, para gestionar las distintas rutas de los micro servicios y la verificación de las comunicaciones entre estos. Se requeriría que los servicios posean la ruta de este gateway como variable de entorno, con el fin de suscribirse a este dinámicamente por si en algún momento cambia su ruta de acceso.

Este gateway particular debería gestionar las rutas que se le suscriban y validar mediante tokens de identificación que la comunicación fuese correcta. Luego de discutir esta solución en clase con el docente a cargo de la materia se llego a la conclusión de que seria la forma optima de implementarlo, pero se volvía demasiado aparatoso para los fines que se buscan evaluar en esta entrega, por lo cual también fue desestimada.

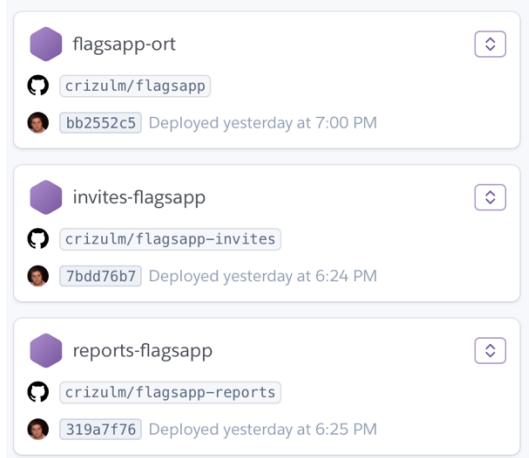
Como solución final se implemento como variable de entorno en todas las aplicaciones las rutas referentes al resto de los servicios, así como también los tokens de verificación con los que se comunicaran. Esta solución no ofrece una escalabilidad deseada ni permite desacoplar completamente los servicios, pero cumple de manera simple con lo deseado para esta entrega.

Estas variables podrán ser modificadas en run time de ser necesario debido a que pueden llegar a variar las direcciones o tokens, y no seria deseado tener que realizar cambios internos en la aplicación si sucede. También deberá quedar una persona encargada de actualizarlas si se presentan cambios en las rutas utilizadas.

3. Descripción del proceso de deployment

Para realizar el proceso de deployment el equipo decidió utilizar pipelines de Heroku, estos nos permiten tener desplegado el código que se encuentra en una branch especificada de nuestro repositorio.

El equipo decidió trabajar sobre 2 branches “develop” y “master”. La primera refiere al código que se encuentra en desarrollo este se despliega de forma local en la maquina de cada uno de los desarrolladores, y por ultimo la branch master contiene el código en producción.



Para cualquiera de los tres dynos debemos configurar lo que son las variables de entorno, y realizar las migraciones de la base de datos.

Para esto debemos realizar lo siguiente:

- 1- Irnos a “Settings” dentro del dyno.
- 2- Clickear “Reveal Config Vars”.
- 3- Ingresar cada una de las variables de entornos necesarias.

Nota: El equipo utilizo una gema llamada Figaro, que nos permite definir las variables de entorno en un archivo “yml” para cada uno de los ambientes (este archivo se encuentra en el directorio config/application.yml). En caso que no se especifiquen las variables de entorno en el dyno de Heroku, este tomara las variables ingresadas en este archivo

Luego de configurada de las variables de entorno debemos realizar las migraciones para poder crear la estructura de la base de datos necesaria para que nuestra app pueda correr.

Simplemente debemos dirigirnos a la consola de Heroku y correr el siguiente comando “rake db:migrate”.

La principal razón por la cual equipo opto por utilizar pipeline es la integración continua con un ambiente de desarrollo, es decir nuestro código es continuamente desplegado, y somos informado si nuestra app fue desplegada correctamente o no.



crizulm@gmail.com: Deployed [e24f8acd]

Yesterday at 4:46 PM · v40 · Roll back to here · [Compare diff](#)



crizulm@gmail.com: Build succeeded

Yesterday at 4:46 PM · [View build log](#)



crizulm@gmail.com: Build failed

Yesterday at 4:43 PM · [View build log](#)