

Universidad ORT Uruguay

Facultad de Ingeniería

Ing. Bernard Wand-Polak

Centinela

Arquitectura de Software en la Práctica

Obligatorio II

Manuel Larrosa - 175136

Sebastian Zawrzykraj - 180110

Matias Settimo - 152946

Profesores

Leticia Esperon

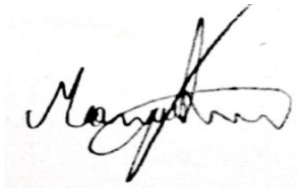
Alejandro Tocar

2020

Declaración de autoría

Nosotros, Manuel Larrosa, Matias Settimo y Sebastián Zawrzykraj, declaramos que el trabajo que se presenta en la presente obra es de nuestra propia mano. Pudiendo asegurar que:

- La obra fue producida en su totalidad mientras realizamos la materia Arquitectura de Software en la Práctica.
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad.
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra.
- En la obra, hemos acusado recibo de las ayudas recibidas.
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y que fue contribuido por nosotros.
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



Manuel Larrosa



Matias Settimo



Sebastian Zawrzykraj

Indice

1. Introducción	5
1.1. Antecedentes	5
1.2. Propósito del documento	5
1.3. Propósito de la arquitectura	5
2. Análisis	6
2.1. Descripción de requerimientos	6
2.2. Desafíos y decisiones de diseño	7
2.3. Bosquejo de solución	8
2.3.1. Actores	8
2.3.2. Descripción general del diseño de la solución	9
3. Documentación de la arquitectura	11
3.1. Vista de módulos	11
3.1.1. Modelo de Datos	11
3.1.1.1. Microservicio de Bugs	11
3.1.1.1.1. Representación Primaria	11
3.1.1.1.2. Catálogo de Elementos	12
3.1.1.2. Microservicio de Usuarios	13
3.1.1.2.1. Representación Primaria	13
3.1.1.2.2. Catálogo de Elementos	14
3.1.1.3. Microservicio de Organizaciones	14
3.1.1.3.1. Representación Primaria	14
3.1.1.3.2. Catálogo de Elementos	14
3.1.1.4. Microservicio de Notificaciones	15
3.1.1.4.1. Representación Primaria	15
3.1.1.4.2. Catálogo de Elementos	15
3.1.1.5. Microservicio de Reportes	16
3.1.1.5.1. Representación Primaria	16
3.1.1.5.2. Catálogo de Elementos	16
3.1.1.6. Microservicio de Visualizador de Costos	17
3.1.1.6.1. Representación Primaria	17
3.1.1.6.2. Catálogo de Elementos	17
3.2. Vista de componentes y conectores	18
3.2.1. Vista de componentes y conectores del sistema	18
3.2.1.1. Representación primaria	18
3.2.1.2. Catálogo de elementos	18
3.2.1.3. Comportamiento	21
3.2.1.3.1. Creación de un usuario con su organización	21

3.2.1.3.2. Creación de un bug	22
3.2.1.3.3. Creación de una invitación y registro	23
3.2.1.3.4. Solicitud de errores por el frontend al autenticar	23
3.3. Vista de Asignación	25
3.3.1. Vista de Despliegue	25
3.3.1.1. Representación Primaria	25
3.3.1.2. Catálogo de Elementos	25
4. Justificaciones de diseño	28
4.1. Uso de cola de mensajería y tópicos	29
4.2. Descomposición en microservicios	30
4.3. Performance y testing	31
4.4. Confiabilidad y disponibilidad	36
4.5. Configuración y manejo de secretos	39
4.6. Autenticación y autorización	39
4.7. Envío de Notificaciones.	40
4.8. Seguridad y Centralización de Logs	40
4.9. Código fuente	42
4.10. Pruebas	43
4.11. Identificación de fallas	43
4.12. Uso de técnicas de caching	44
4.13. Tenancy	44
4.14. Microservice Chassis	45
5. Proceso de deployment y DevOps	46
5.1. Creación de imágenes docker y contenedores	46
5.2. Compose de Servicios y Despliegue.	49
5.3 Deployment a Producción.	50
6. Resultados y conclusiones	59
6.1. Proceso de trabajo y requerimientos funcionales	59
6.2. Sobre los nuevos requerimientos funcionales	60
6.3. Requerimientos no funcionales	63
6.4. Mejoras y faltantes sobre los requerimientos solicitados	66
6.5. Oportunidades de mejoras detectadas sobre el diseño	67
7. Anexos	69
7.1. Requerimientos y restricciones	69
7.1.1. Requerimientos Funcionales	69
7.1.2. Requerimientos No Funcionales	72
7.1.3. Restricciones	75
7.2. Sprints realizados	76
7.2.1. Sprint 4	76

7.2.2. Sprint 5	76
7.3. Api rest	77
7.4. URLs de la aplicación	78
7.5. Test basados en escenarios	79

1. Introducción

1.1. Antecedentes

Dado el éxito del sistema Centinela, se decide exponer el mismo como una solución por fuera del sistema EnviosYa implementando una arquitectura basada en microservicios.

Dicho sistema proveerá las siguientes funcionalidades como registro de errores y bugs desde los sistemas de la empresa (con identificación de prioridad y ambiente que lo reporta), administración de usuarios (administradores y desarrolladores), administración de bugs (edición, asignación a usuarios, resolución, etc), notificaciones por correo y distintos reportes sobre los errores generados.

1.2. Propósito del documento

Este documento no es una especificación de requerimientos, sino un análisis que se ampara en los mismos.

El propósito tampoco es brindar un diseño de bajo nivel, sino proveer una especificación completa del diseño arquitectónico de Centinela, buscando detallar distintos aspectos de su implementación, decisiones de diseño, y tácticas de despliegue.

También se brindarán algunos detalles de la gestión del trabajo, herramientas utilizadas y otras decisiones tomadas por el equipo.

1.3. Propósito de la arquitectura

La arquitectura del sistema, que será la base para su elaboración, está basada en el análisis de sus requerimientos.

Toda decisión está amparada en los mismos, buscando favorecer los atributos de calidad solicitados por el cliente y deseados por el equipo de trabajo, a efectos de ofrecer una solución que resuelva de la mejor forma el problema, teniendo en cuenta ventajas y problemas que pueda introducir (*trade-offs*).

2. Análisis

2.1. Descripción de requerimientos

La especificación detallada de los requerimientos se encuentra en el apartado anexo [7.1 Requerimientos y restricciones](#).

Se pasará a una breve descripción de los mismos, a efectos de poder explicar algunas decisiones tomadas para el diseño de la solución.

1. Requerimientos funcionales

- a. Registro web de usuarios administradores y desarrolladores (los usuarios administradores podrán crear usuarios y editar bugs; los usuarios desarrolladores solo podrán cerrar bugs).
- b. Autenticación de usuarios.
- c. Gestión de claves para que las distintas aplicaciones puedan crear errores.
- d. Gestión web de los errores para los usuarios (listado, visualización, edición)
- e. Reportes estadísticos de errores creados (web y rest).
- f. Reportes de errores asignados (web).
- g. Reporte de consumo y facturación para el mes actual o anteriores.
- h. Proveer un SDK para que los clientes utilicen a la hora de crear bugs
- i. Gestionar alertas de bugs por correo de manera que los usuarios puedan configurar distintas preferencias para recibir las mismas.

2. Principales requerimientos no funcionales y restricciones

- a. Performance con tiempos de respuesta promedio debajo de 350 ms. para cargas hasta 1200 req/m.
- b. Informe del funcionamiento del sistema mediante un endpoint HTTP para poder verificar la disponibilidad del sistema.
- c. Configuración de variables de entorno para los distintos ambientes para mejorar la seguridad y modificabilidad del sistema.
- d. Seguridad de acceso a la información de acuerdo a las distintas responsabilidades de tipo de usuario, aplicaciones, etc.
- e. Interfaz web, endpoints REST para la comunicación con el backend, utilización de un repositorio en Github con archivos README.md descriptivos.
- f. Centralizado de logs emitidos por todos los microservicios.
- g. Pruebas de carga con Apache JMeter.
- h. La aplicación debe ser desarrollada como una arquitectura basada en microservicios con al menos uno de estos desarrollado en un lenguaje de programación distinto al de los demás microservicios..

- i. Pruebas automatizadas para al menos tres requerimientos funcionales de los anteriores donde será necesario ejecutar las pruebas automatizadas del código cuando un nuevo commit se integre a la rama principal.
- j. No ocasionar downtime cuando se despliega una nueva versión de algún microservicio.
- k. Monitorear las peticiones por minuto y el tiempo de respuesta de los distintos endpoints.

2.2. Desafíos y decisiones de diseño

La solución requerida nos presentó varios desafíos, tanto tecnológicos como de diseño, en los cuales el equipo tenía nula experiencia previa.

En cuanto a los desafíos de diseño de la solución:

1. Diseñar sistemas distribuidos en la nube, con sus distintas particularidades y estrategias, como las recomendaciones *twelve factors*.
2. Refactorización y separación del monolito anterior a una arquitectura basada en microservicios con todo lo que esto conlleva, en especial la comunicación entre estos.

En cuanto a requerimientos tecnológicos:

Aprender distintas estrategias de despliegue en AWS (*Amazon Web Services*), entender su funcionamiento y cuál es la que más nos conviene, teniendo como restricción la cuenta *Educate* de AWS proporcionada por la Universidad Ort Uruguay, a efectos de realizar pruebas.

Tomar decisiones en cuanto a frameworks a utilizar para frontend y backend, herramientas de monitoreo, y herramientas gráficas para visualización de estadísticas.

Selección de que cola de mensajería usamos para comunicar los microservicios.

Lenguaje utilizado para desarrollar uno de los microservicios extraídos de manera de satisfacer uno de los requerimientos no funcionales.

Esto nos llevó a elegir algunas herramientas que ya conocíamos y confiábamos, tanto por su productividad como por ser orientadas a la web: NodeJs, y Angular para frontend, entendiendo que con las mismas se nos facilitaría la implementación de la solución, pudiendo enfocarnos en las decisiones de nivel arquitectónico.

En cuanto al lenguaje de programación seleccionado para desarrollar uno de los microservicios, decidimos utilizar C# con el framework .NET Core ya que este es conocido por todo el equipo de trabajo de forma que esto agiliza el desarrollo de este microservicio.

También tomamos la decisión de investigar la posibilidad de incorporar algún template de frontend que nos permita enfocarnos más en la usabilidad del mismo y no en la estética, dado que tampoco contábamos con un diseñador de interfaces experimentado en el equipo.

Por tal motivo, luego de una investigación previa, se tomó la decisión de experimentar con el template Nebular, de la empresa Akveo (<https://akveo.github.io/nebular/>). El mismo ofrece muchos componentes gratuitos desarrollados en angular, que simplificaron algunas tareas, como la generación de un menú, visualización de estadísticas, generación de estilos css homogéneos y atractivos, y algunos componentes customizados, como una tabla que nos ofrecía una manipulación de los eventos y comportamiento hacia el usuario bastante amigable, para la vista de bugs.

También se fueron tomando varias decisiones importantes a nivel del deploy, que dadas algunas restricciones que tenemos por la cuenta educativa, favorecieron la simplicidad del mismo, tal vez en detrimento de algún requisito de calidad, pero que entendemos podremos mejorar a futuro. Esto se detalla más adelante, en el apartado [5. Proceso de deployment y dev-ops](#).

2.3. Bosquejo de solución

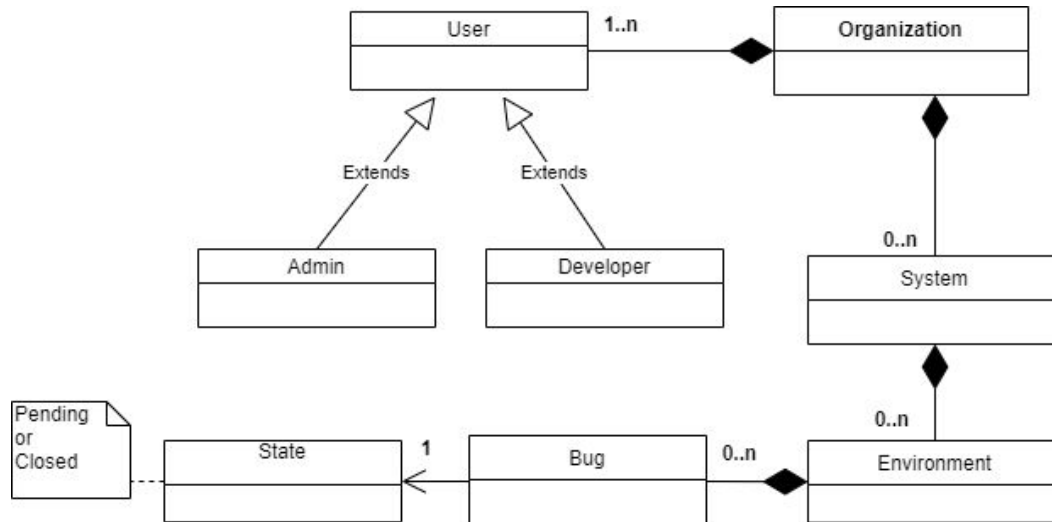
2.3.1. Actores

Para poder comprender cuál es la hoja de ruta hacia donde vamos, entendemos necesario un bosquejo rápido de la solución, a efectos de identificar actores externos del sistema, cómo se relacionan, y algunas decisiones de diseño tomadas para cumplir con las especificaciones.

En primer lugar, tenemos identificados los siguientes actores:

- **Organizaciones**, que quieren utilizar nuestro sistema para reportar **Bugs** de sus distintos **Sistemas**, y los **Ambientes** de esos sistemas que son los que pueden generar un bug (un ambiente de un sistema, podría ser por ejemplo el ambiente de Desarrollo, o de Producción).
- Usuarios **Desarrollador** y **Administrador**.

De esta manera presentamos el siguiente diagrama de dominio reducido, que busca representar qué es lo que necesita cada organización que nos contrata:



Dado el requerimiento **RF1**, donde una organización es creada cuando el primer usuario administrador se registra en el sistema, tomamos como supuesto que cada usuario se pasará a identificar por su correo, y que este será único para todo el sistema, no pudiendo registrarse en más de una organización con el mismo correo. A su vez, cada organización para existir, requerirá de al menos un usuario.

2.3.2. Descripción general del diseño de la solución

Observando los requerimientos funcionales, pasamos a mostrar cómo se han resuelto los más relevantes.

Para esto se recomienda en primer lugar observar el [diagrama de componentes](#) general a todo el sistema, el cual está detallado más adelante y muestra cuáles son los microservicios que interactúan en la solución, la cual se compone de:

- Microservicio de Organizaciones
- Microservicio de Usuarios
- Microservicio de Bugs
- Microservicio de Envío de mails
- Microservicio Visualizador de Costos
- Microservicio de Reportes
- Microservicio de Notificaciones
- Una aplicación **SPA** (*Single Page Application*), representada como CentinelaCli que se ofrece como frontend.

La aplicación frontend se encargará de interactuar con los usuarios, donde podrán crear su organización e ingresar al sistema, ver listados y administrar bugs, administrar otros usuarios, administrar sus preferencias, ver estadísticas, reportes y consumos (**RF1, RF2, RF3, RF4, RF5, RF6,**

RF7, RF8, RF11, RF12, RF14, RF15). La misma se comunicará con el backend de los distintos microservicios que consuma a través de un cliente Rest de Angular, cumpliendo con la restricción **RES2**, de entablar comunicaciones REST.

Los diferentes microservicios tendrán a su cargo atender distintas request que les genera el frontend, responder a las solicitudes de otros microservicios y también atender las request REST ofrecidas en sus endpoints (**RF9 y RF10**).

Por otro lado, algunos microservicios también estarán escuchando eventos que ocurran en el sistema para procesarlos.

Toda la comunicación entre microservicios se realiza a través de colas de mensajerías salvo aquellos que se comunican con el Microservicio de Usuarios para autenticarse, esto lo hacen mediante RPI. Al tener la mayoría de los microservicios comunicados mediante colas de mensajería, evitamos la mayoría de las request síncronas de forma tal que podamos cumplir los requerimientos **RNF1**.

La separación por microservicios es la siguiente:

- **Microservicio de Bugs:** encargado de atender todos los request de los bugs (creación, actualización, obtención). Este ejecuta dos workers:
 - **bugProcessor** : éste se encarga de atender los request REST de creación de bugs que llegan.
 - **queueProcessor** : éste se encarga de atender los eventos de creación de usuario y ambientes para almacenarlos en su base local.
- **Microservicio de Usuarios:** encargado de la creación de los usuarios y autenticación de los mismos. Emite el evento de creación y actualización de usuarios a la cola.
- **Microservicio de Organizaciones:** encargado de la creación de organizaciones, sistemas y ambientes. Emite los eventos de creación de organización, sistema y ambiente a la cola.
- **Microservicio de Envío de mails:** encargado de todo lo referente al envío de mails del sistema, ya sean los mails de invitación a nuevos usuarios así como también los mails de notificación de bugs de las preferencias de cada usuario.
- **Microservicio de Notificación:** encargado de guardar las preferencias de cada usuario sobre el envío de alertas de bugs y enviar a la cola de mensajería los eventos de envío de mail cuando sean pertinentes.
- **Microservicio de Reportes:** encargado de devolver los diferentes reportes solicitados por el usuario.
 - **queueProcessor** : éste worker está encargado de conectarse con la cola de mensajería y escuchar a los eventos de creación de usuario, creación de bugs y actualización de bugs, para insertar en la base de datos local del microservicio y así tener los datos para armar los reportes.
- **Microservicio de Visualizador de Costos:** encargado de calcular y devolver el el costo de los bugs y usuarios para la organización en un mes y año dados. Este microservicio está escuchando los eventos de creación de usuarios y bugs para insertar en su base local. Para satisfacer el **RNF9** fue codificado en C# con el framework .NET core.

Para ver ejemplos de cómo interactúan los componentes del sistema, puede entrar a los apartados [creación de un bug](#) y [creación de una invitación y registro](#), que se encuentran más adelante.

3. Documentación de la arquitectura

3.1. Vista de módulos

Este tipo de vistas incorporan decisiones sobre cómo se estructura el sistema en un conjunto de unidades de código o datos.

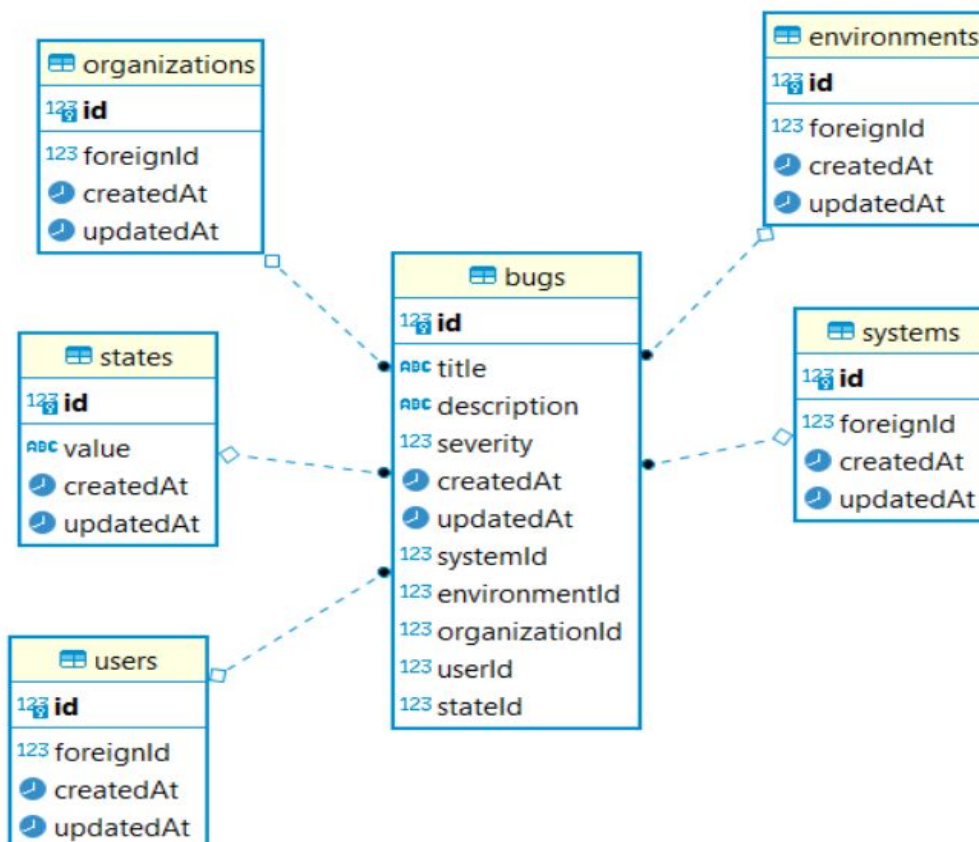
Decidimos únicamente agregar el modelo de datos el cual encontramos que esclarece mucho la estructura de tablas de cada microservicio. Encontramos innecesario ahondar en detallar la estructura interna de cada uno de estos utilizando vistas como la de descomposición o la vista de uso.

3.1.1. Modelo de Datos

A continuación pasaremos a mostrar y describir los Modelos entidad relación de las bases de datos de los microservicios del sistema.

3.1.1.1. Microservicio de Bugs

3.1.1.1.1. Representación Primaria

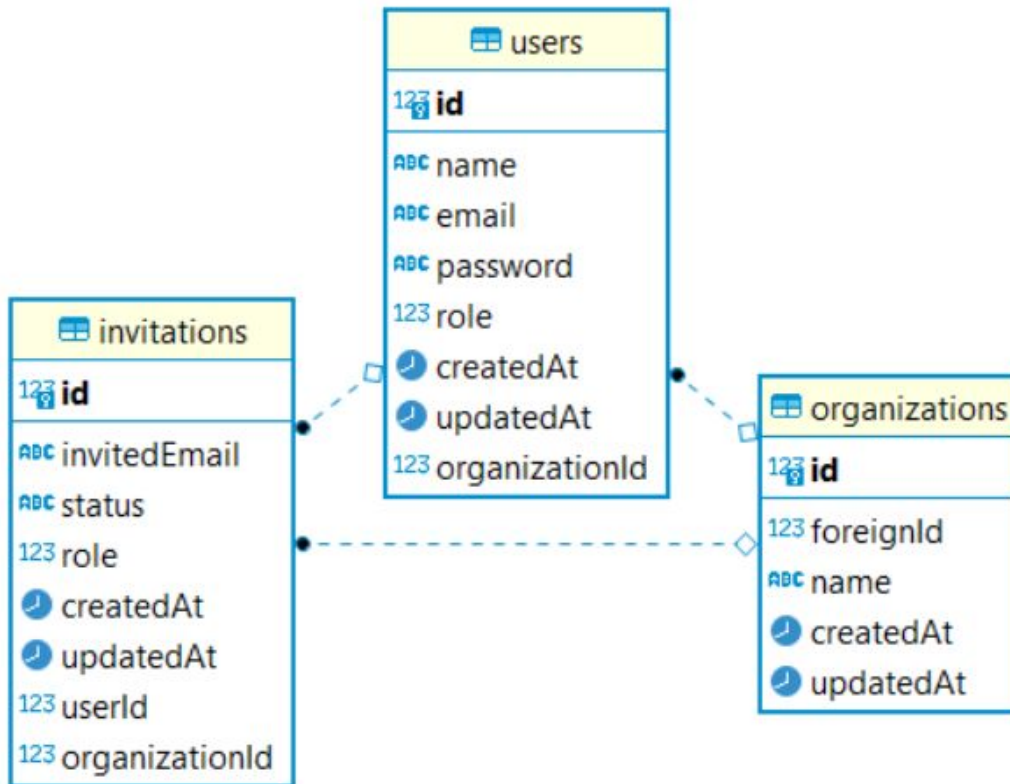


3.1.1.1.2. Catálogo de Elementos

Elemento	Responsabilidad
bugs	Tabla principal de bugs, en esta guardamos todo lo relacionado a los bugs del sistema y este microservicio
organizations	Tabla auxiliar de organizaciones para el microservicio de bugs, se guarda una foreignId que hace referencia al id de la organización real del sistema creada en el microservicio de organizaciones
systems	Tabla auxiliar de sistemas para el microservicio de bugs, se guarda una foreignId que hace referencia al id del sistema real creado en el microservicio de organizaciones
environments	Tabla auxiliar de ambientes para el microservicio de bugs, se guarda una foreignId que hace referencia al id del ambiente real creado en el microservicio de organizaciones
states	Contiene los estados en los cuales los bugs se podrán encontrar.
users	Tabla auxiliar de usuarios para el microservicio de bugs, se guarda una foreignId que hace referencia al id del usuario real creado en el microservicio de usuarios

3.1.1.2. Microservicio de Usuarios

3.1.1.2.1. Representación Primaria

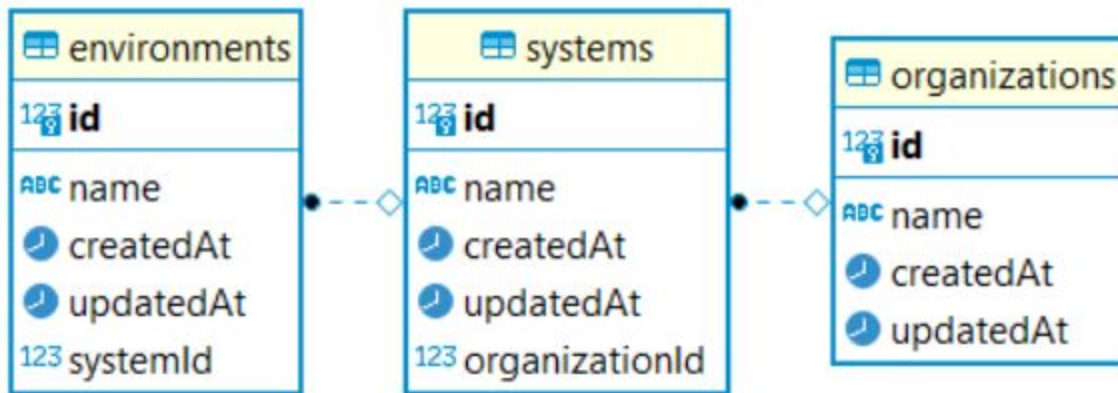


3.1.1.2.2. Catálogo de Elementos

Elemento	Responsabilidad
users	Tabla principal de usuarios, en esta guardamos todo lo relacionado a los usuarios del sistema y este microservicio
organizations	Tabla auxiliar de organizaciones para el microservicio de usuarios, se guarda una foreignId que hace referencia al id de la organización real del sistema creada en el microservicio de organizaciones
invitations	Contiene todas las invitaciones enviadas a los usuarios así como el estado actual de la misma (pendiente, cancelada, aceptada). Tiene dos foreign key, una con el usuario al cual fue enviada y otra con la organización para la cual pertenece esta invitación.

3.1.1.3. Microservicio de Organizaciones

3.1.1.3.1. Representación Primaria



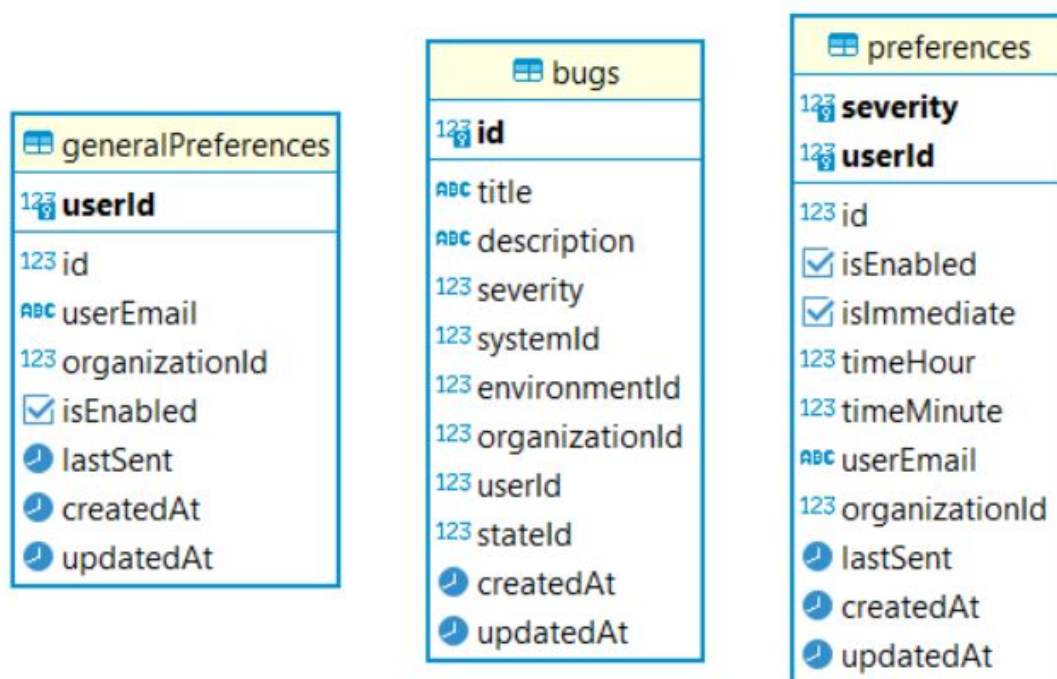
3.1.1.3.2. Catálogo de Elementos

Elemento	Responsabilidad
organizations	Tabla principal de organizaciones, en esta guardamos todo lo relacionado a las organizaciones del sistema y este microservicio.
systems	Tabla principal de sistemas, en esta guardamos todo lo relacionado a los sistemas de este microservicio. Posee

	una foreign key con organización para saber a cual pertenece.
environments	Tabla principal de ambientes, en esta guardamos todo lo relacionado a los ambientes del sistema y de este microservicio. Posee una foreign key con sistema para saber a cual pertenece.

3.1.1.4. Microservicio de Notificaciones

3.1.1.4.1. Representación Primaria



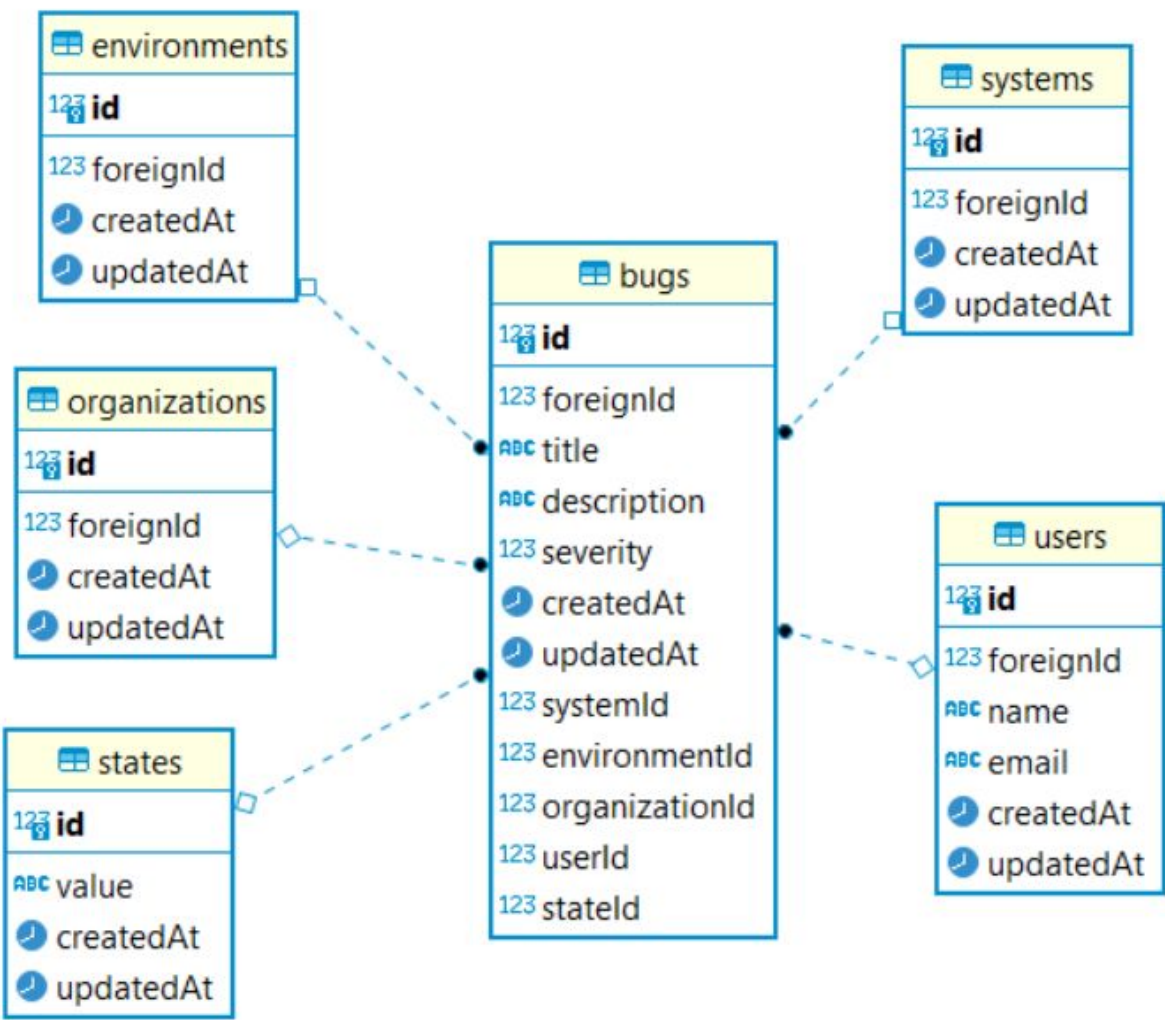
3.1.1.4.2. Catálogo de Elementos

Elemento	Responsabilidad
preferences	Contiene las configuraciones de las preferencias de cada usuario. Contiene un <code>userId</code> y <code>organizationId</code> pertenecientes al id del usuario y la organización reales existentes en el microservicio usuario y microservicio de organización.
generalPreferences	En esta tabla almacenamos las si el usuario quiere o no recibir mail por los alertas de bugs.

bugs	Contiene una réplica de la información de los bugs del microservicio de bugs para ser notificados al usuario según sus preferencias.
------	--------------------------------------------------------------------------------------------------------------------------------------

3.1.1.5. Microservicio de Reportes

3.1.1.5.1. Representación Primaria



3.1.1.5.2. Catálogo de Elementos

Elemento	Responsabilidad
bugs	Réplica de los datos de bugs del Microservicio de Bugs, con una foreignId al id real del bug.

organizations	Réplica de los datos de bugs del Microservicio de Organizaciones, con una foreignId al id real de la organización.
systems	Réplica de los datos de bugs del Microservicio de Organizaciones, con una foreignId al id real del sistema.
environments	Réplica de los datos de bugs del Microservicio de Organizaciones, con una foreignId al id real del ambiente.
states	Contiene los estados en los cuales los bugs se podrán encontrar.
users	Réplica de los datos de bugs del Microservicio de Usuarios, con una foreignId al id real del usuario.

3.1.1.6. Microservicio de Visualizador de Costos

3.1.1.6.1. Representación Primaria



3.1.1.6.2. Catálogo de Elementos

Elemento	Responsabilidad
CostsHistory	Esta tabla almacena un histórico de los costos por mes y año tanto para bugs como usuarios existentes por cada organización. El organizationId corresponde con el id de la organización creado en el Microservicios de Organizaciones.
OrganizationsCosts	Contiene el total de bugs y usuarios creados por cada organización para cada mes y año.
_EFMigrationsHistory	Tabla donde se almacena todo lo referente a las migraciones.

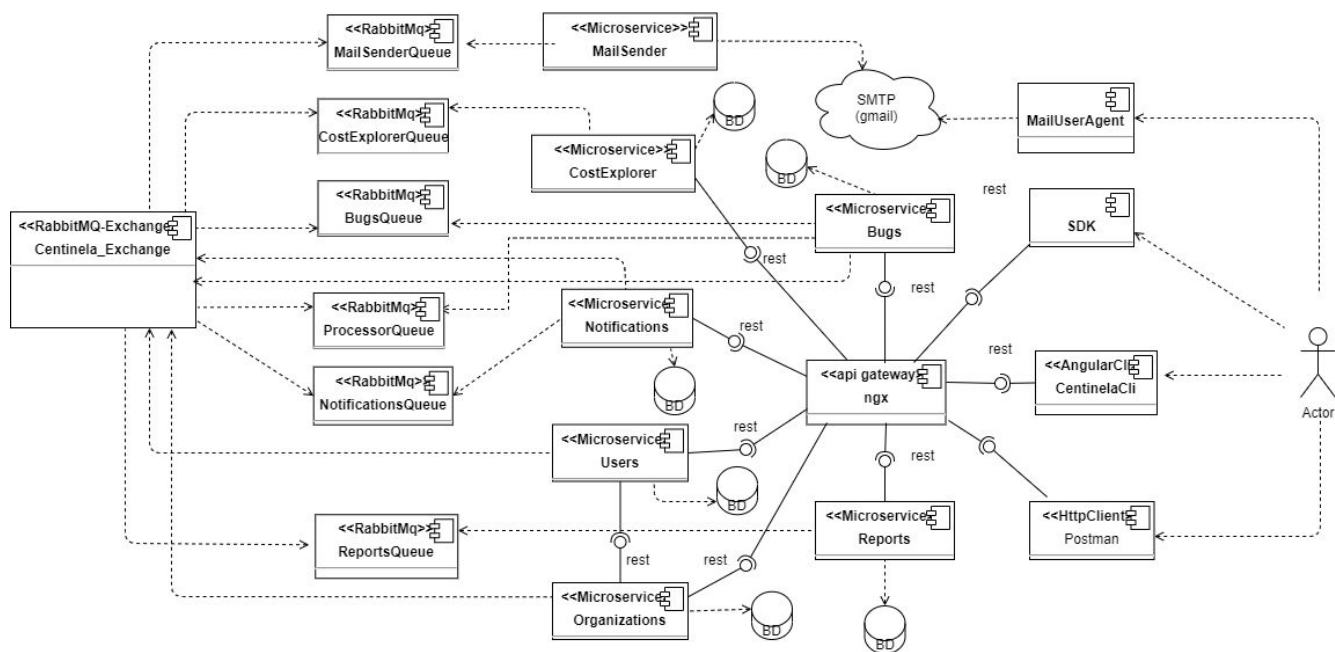
3.2. Vista de componentes y conectores

Esta vista tiene como objetivo mostrar en tiempo de ejecución cómo se relacionan los distintos componentes del sistema a través de sus conectores, básicamente mostrando qué componentes exponen interfaces (puertos) para que puedan ser consumidas por otros componentes.

3.2.1. Vista de componentes y conectores del sistema

Esta vista contiene todos los componentes del sistema y los conectores que permiten su comunicación. No tiene en cuenta la conexión REST que tienen los microservicios de CostExplorer, Notifications, Organizations y Bugs con el microservicio de Users, a efectos de realizar el chequeo de autorización de usuarios.

3.2.1.1. Representación primaria



3.2.1.2. Catálogo de elementos

Componente/ conector	Responsabilidad
CentinelaCli	Es la aplicación SPA del frontend realizada en Angula.
Postman	Cliente postman.
SDK	SDK realizado en NodeJs para realizar request de

	creación de bugs.
MailUserAgent	Casilla de correo de un usuario.
nginx	Contenedor que aloja nuestro proxy reverso. Este se encargará de direccionar los distintos request REST al endpoint del microservicio correspondiente.
Microservicio Bugs	<p>Este es el microservicio de bugs, se encarga de gestionar todos los request referente a los bugs (creaciones, actualizaciones y obtención de los mismos). Cada vez que un bug es creado o actualizado, se envía al exchange este evento para que lo envíe a quien sea pertinente. Posee dos workers:</p> <ul style="list-style-type: none"> • Uno llamado bugProcessor el cual se encarga de procesar los request de los bugs de creación leyendo de la cola de rabbit BugsQueue y persistiendolos en la base de datos. • El otro queueProcessor se encarga de leer de la cola ProcessorQueue la cual está escuchando por los eventos de creación de usuarios y creación de ambientes para persistir ambos en su base local.
Microservicio Reports	<p>Microservicio encargado de atender los request de los clientes solicitando los distintos reportes. Posee un worker:</p> <ul style="list-style-type: none"> • El worker queueProcessor se encarga de consumir la cola ReportsQueue escuchando los tópicos de creación y actualización de bugs así como también el de creación de usuario para persistirlos en su base y así utilizarlos a la hora de responder a los request de reportes.
Microservicio MailSender	Este microservicio es el encargado de escuchar en la cola MailSenderQueue y procesar estos mensajes, enviando los mail pertinentes a los destinatarios de dichos mensajes.
Microservicio CostExplorer	Este microservicio consume los mensajes que llegan a la cola CostExplorerQueue cuyo tópico será el de usuario y bug creado, para luego persistir estos datos en su base de datos y así generar el reporte de costos.
Microservicio Notifications	<p>Este microservicio es el encargado de la creación y actualización de las preferencias del usuario. El mismo consume de la cola NotificationsQueue el tópico de bugs actualizados para así persistir en su base la información de los bugs de los usuarios y de esta forma poder generar el evento de envío de mail al usuario en caso de que satisfaga alguna de las preferencias de estos. Estos envíos de mail son ejecutados cada cierto tiempo por procesos programados con este fin.</p>

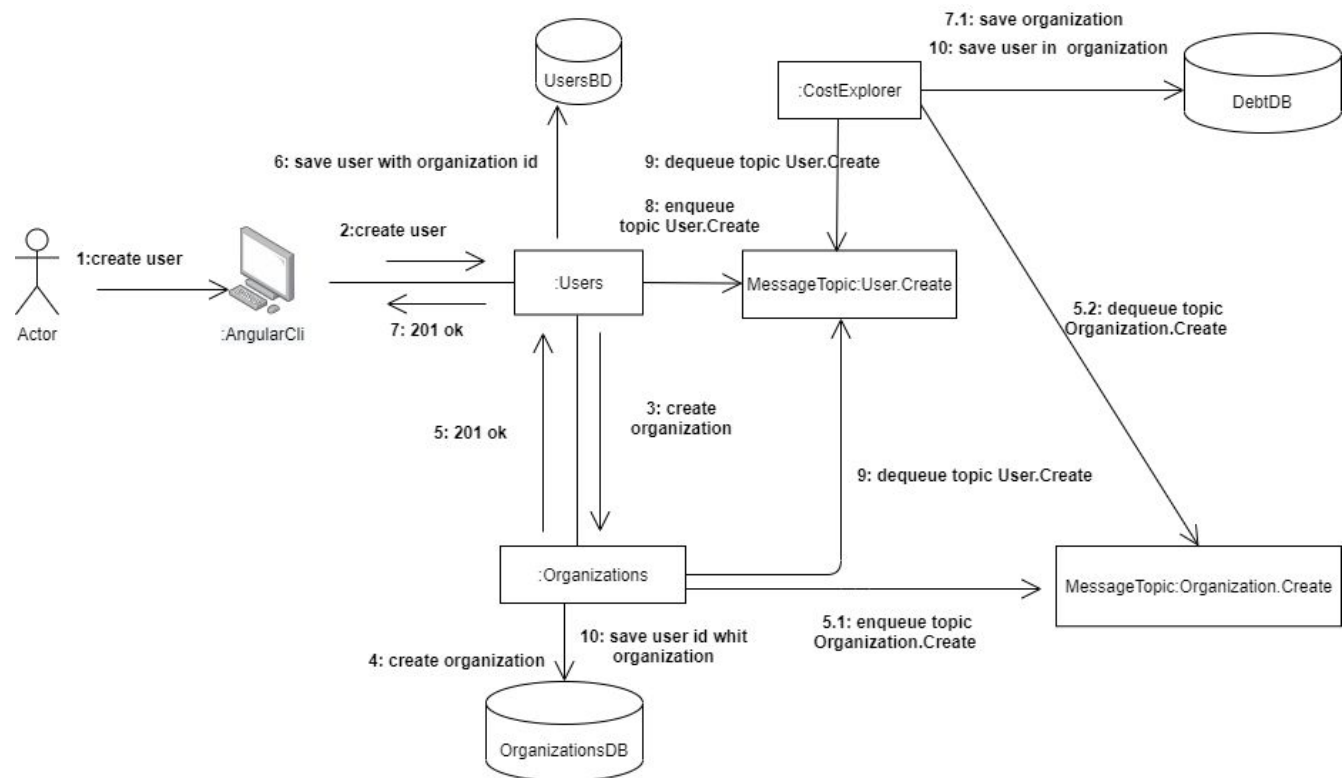
Microservicio Users	Este microservicio es el encargado de procesar los request de creación de usuarios. Cada vez que un usuario es creado en el sistema lo almacena en su base de datos y envía al exchange el evento de creación de usuario. Este servicio también es el encargado de autenticar los token de usuarios haciendo también de servicio de identidad federada.
Microservicio Organizations	Este microservicio es el encargado de gestionar la creación de organizaciones, sistemas y ambientes. Cada vez que uno de estos es creado le envía al exchange el evento para que él mismo lo gestione y envíe a quien corresponda. Este microservicio es el encargado de generar la <i>keyConnection</i> a la hora de crear un ambiente.
RabbitMQ Queues	Estas son las colas de rabbitMQ las cuales consumen algunos de los microservicios
BD	Las distintas bases de datos de los microservicios, se decidió como norma que todas fueran postgres
Centinela_Exchange	Topic exchange de rabbitMQ el cual será el encargado de dirigir los diferentes eventos a las colas pertinentes que los están escuchando

3.2.1.3. Comportamiento

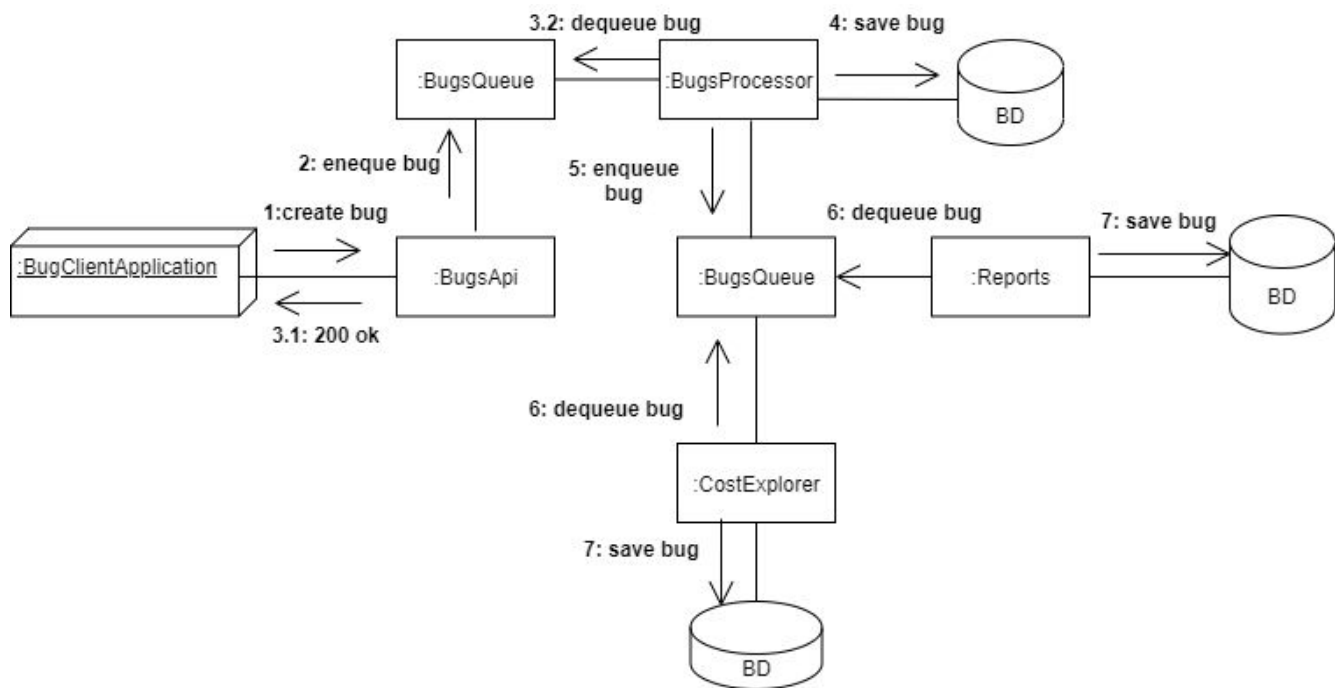
Pasaremos ahora a mostrar cómo interactúan los distintos componentes para los casos de uso más importantes del sistema. También se pueden ver algunos diagramas menos formales pero más interactivos y animados en el **readme.md** del repositorio de la siguiente dirección:

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Main>

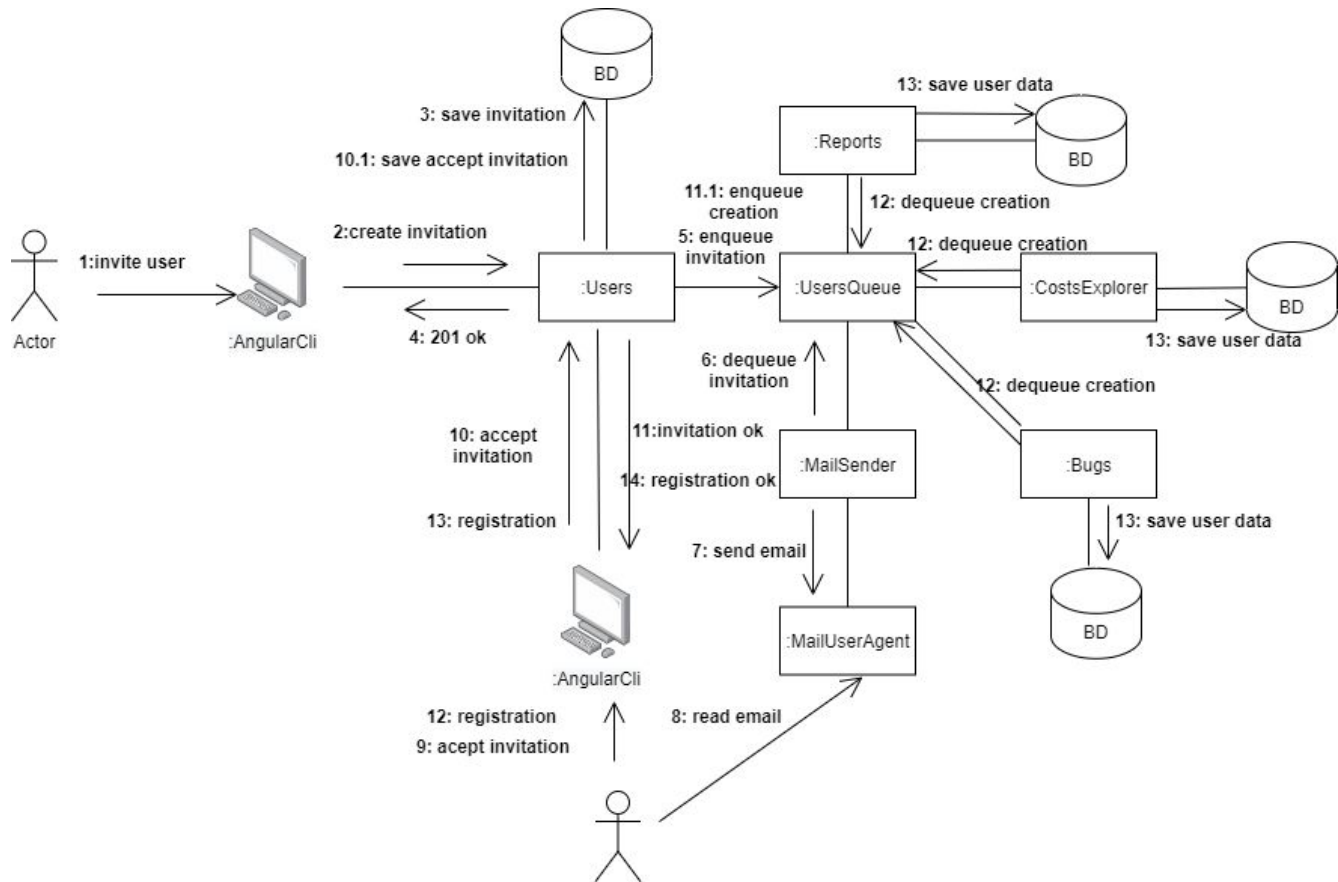
3.2.1.3.1. Creación de un usuario con su organización



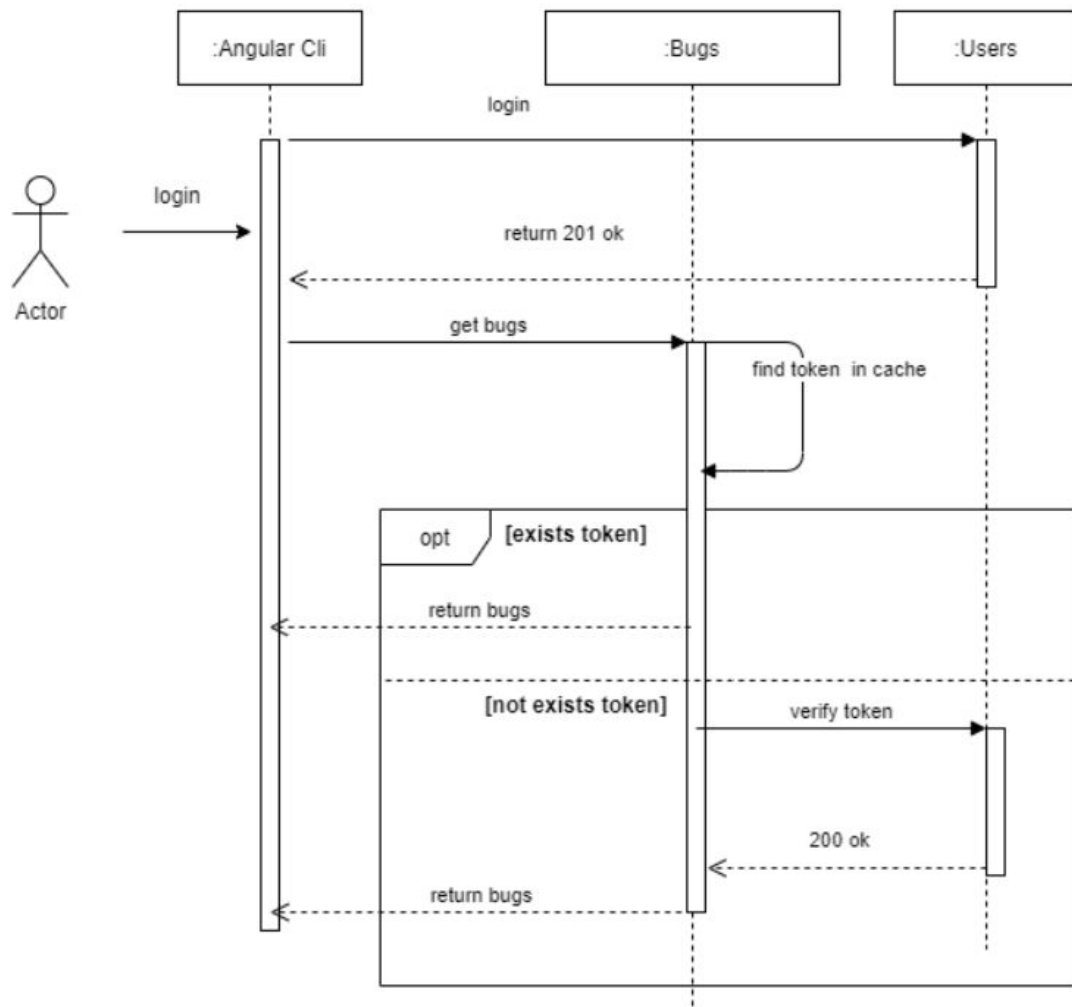
3.2.1.3.2. Creación de un bug



3.2.1.3.3. Creación de una invitación y registro



3.2.1.3.4. Solicitud de errores por el frontend al autenticar

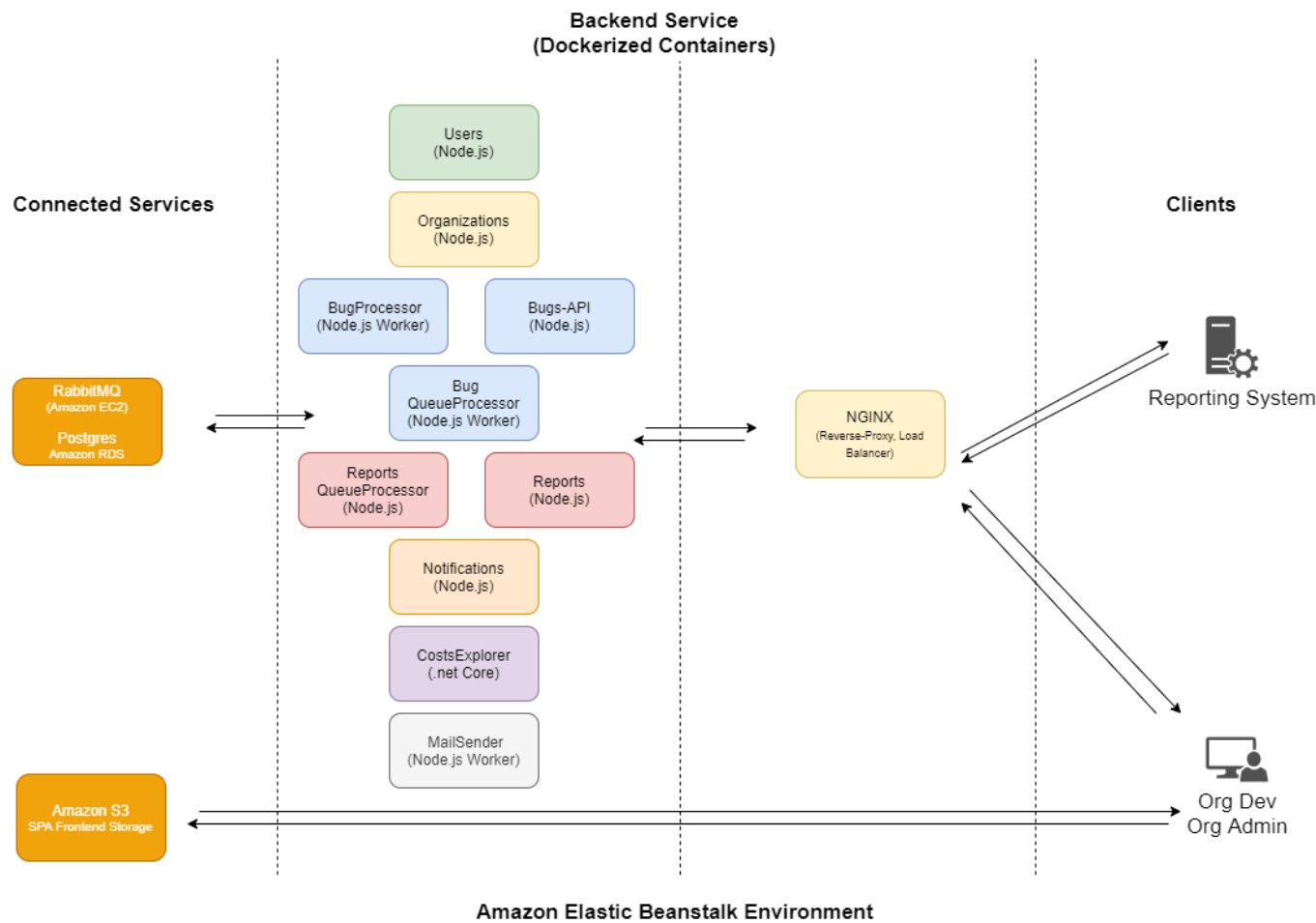


3.3. Vista de Asignación

En este tipo de vista, se incorporan las decisiones sobre cómo se relaciona el sistema con las estructuras que no son software en su ambiente (CPU, file systems, redes, equipos de desarrollo, etc). Muestran la relación entre los elementos del software y los elementos del entorno externo en el que se crea y ejecuta el software.

3.3.1. Vista de Despliegue

3.3.1.1. Representación Primaria



3.3.1.2. Catálogo de Elementos

Elemento	Responsabilidad
RabbitMQ (Amazon EC2)	Aquí tenemos dos servicios conectados. Por un lado RabbitMQ que estará alojado en Amazon EC2 dentro de

Postgres Amazon RDS	una máquina virtual. Esto es debido a las limitaciones de la cuenta educate para utilizar un servicio de cola de mensajes SQS. Por otro lado tenemos un servicio de base datos Postgres alojado en Amazon RDS.
Amazon S3 SPA Frontend Storage	Utilizando Amazon S3 alojaremos el frontend de nuestra aplicación.
Users	Este contenedor aloja el servicio de usuarios. Está desarrollado en NodeJS. Se encarga de mantener la lista de usuarios del sistema y hace de servicio de identidad federada para los otros microservicios del ecosistema.
Organizations	Este contenedor aloja el servicios de usuarios. Está desarrollado en NodeJS. Se encarga de mantener los datos de las organizaciones, sistemas y ambientes que tiene todo el ecosistema. A su vez es quien genera las claves de conexión de ambiente <i>keyConnection</i> . Necesaria para publicar Bugs desde la SDK o hacia el endpoint rest que ofrece el microservicio de usuarios.
Bugs-API	Este contenedor, parte del microservicio de Bugs que se encarga de atender y responder todas las requests sobre los bugs que se publiquen. Ofrece un endpoint para que se reporten bugs y otro para que los mismos se puedan actualizar. Está construido en NodeJS. Planificó para que tenga pocos I/Os con la base datos ya que en el caso de bugs publicados, este solo los encola en una cola de mensajes.
BugProcessor	Este contenedor, también parte del microservicio de Bugs, es quien procesa los mensajes publicados por la Api de bugs. Este proceso worker lee los mensajes de la cola de mensajería y persiste los bugs en la base de datos. Construido en NodeJs.
BugQueueProcessor	Este contenedor, también parte del microservicio de Bugs, es quien procesa los mensajes publicados por otros microservicios. Este proceso worker lee los mensajes de la cola de mensajería con notificaciones de creación de usuarios y/o organizaciones y persiste estos datos en la base de datos. Construido en NodeJs.
Reports	Este contenedor es parte del microservicio de Reportes. Se encarga de atender las solicitudes de reportes que llegan desde el cliente. Hace consultas hacia su base de datos. También fue construido en NodeJS.
ReportsQueueProcessor	Este contenedor es parte del microservicio de Reportes. Se encarga de ingerir los mensajes que llegan por cola de mensajería y los inserta en la base de datos de reportes.

Notifications	Este contenedor es parte del microservicio de Notificaciones. Está atendiendo solicitudes de creación y actualización de preferencias de notificaciones. Además ejecuta procesos programados a ejecutarse cada cierto tiempo para generar notificaciones hacia los usuarios. Este servicio además consume datos publicados en una cola de mensajería y los inserta en su base de datos. A su vez, consume los datos de su base de datos para enviar detalles de las notificaciones a los clientes. También está construido en NodeJS.
CostExplorer	Este contenedor es para alojar al microservicio de Costos. Está construido en C# con el framework .net Core. Se encarga de consumir mensajes publicados en su cola de mensajería y almacenar la cantidad de bugs y usuarios que se crean en su base de datos. Además ofrece una API para que los usuarios puedan consultar los costos que han incurrido en el mes corriente y meses pasados.
MailServer	El contenedor de MailSender ejecuta el código del microservicio de mailSender. Su única función es la de procesar los mensajes que le sean publicados en su cola de mensajería y así despachar correos a los usuarios. No utiliza base de datos.
NGINX	Contenedor que aloja nuestro proxy reverso. Este se encargará de direccionar los distintos request REST a los endpoints correspondientes.
Reporting Systems	Estos serán todo los sistema externos de los clientes que utilizan centinela para reportar BUGS mediante la sdk provista.
Org Dev/ Org Admin	Estos son los equipos o dispositivos de las organizaciones de los clientes, ya sean con el rol de developer o admin que van a interactuar con nuestro frontend directamente o mediante REST.

4. Justificaciones de diseño

El proyecto tecnológico a desarrollar debe usar tecnologías web y se deberá ajustar a las características de una aplicación cloud native. El equipo podrá optar por usar mayoritariamente la tecnología de “Plataforma como servicio” PaaS.

En nuestro caso, para la etapa de desarrollo y para el despliegue, los componentes principales de nuestro sistema fueron construidos para ser ejecutados como contenedores independientes. Esta decisión fue tomada con el fin de favorecer la portabilidad y escalabilidad de nuestro sistema, a lo que destacamos:

Los contenedores tienen la capacidad de ser agnósticos a la infraestructura en la cual están desplegados, permitiendo a nosotros hacer el despliegue de los mismos en cualquier plataforma que ofrezca el servicio de manejo de contenedores. Ya sea en nuestro pc de desarrollo como luego en producción.

A su vez, estos nuestros servicios contenerizados no manejan estado (Stateless) lo que habilita a que nuestro sistema tenga la capacidad de escalar de forma horizontal (en número de instancias de estos contenedores) como también en forma vertical (mejorando la capacidad de la infraestructura que los aloja).

El uso de contenedores también favoreció a que el equipo de desarrollo pueda enfocarse por completo al diseño y codificación. Evitando así, que cada uno de nosotros tenga que realizar complicadas configuraciones en sus equipos para soportar la construcción del sistema.

Para el alojamiento de contenido estático, en nuestro caso el frontend como una single page application (SPA) en Angular. Utilizamos la plataforma como servicio de AWS llamada Amazon S3.

Si bien la tecnología de programación que utilizamos ofrece librerías para la publicación de contenido estático, el equipo tomó la decisión de alojar la SPA en una PaaS como lo es Amazon S3.

S3 nos da la ventaja de eliminar el consumo de recursos que pudiera causar la descarga de este contenido en el caso que fuera alojado en conjunto con nuestra capa de negocio. Permite además un desacoplamiento completo de lo que puedan ser las actualizaciones o cambios que pueda tener el frontend del backend.

Al mismo tiempo, Amazon S3 está preparado para soportar la transferencia de grandes volúmenes de datos sin la necesidad de tener que definir criterios de escalado o capacidad ofreciendo una tasa de disponibilidad superior 99.9% de uptime

La practicidad de su uso fue otra de las razones por la cual tomamos la decisión de utilizar S3, todas las actualizaciones del frontend se pudieron realizar simplemente con una acción de Drag n’ Drop de nuestro código compilado hacia el storage de S3.

Nuestro servicio depende también de servicios conectados o backing services como lo son nuestra base de datos relacional y el servicio de colas de mensajería. Si bien ambos servicios son esenciales para el funcionamiento de nuestro sistema desarrollado, no son parte de la implementación del mismo, sino que solo son servicios que éste consume. Ambos backing services fueron pensados para ser consumidos desde servicios PaaS como los que ofrece Amazon RDS para alojar nuestra base datos relacional y Amazon MQ para el manejo de colas de mensajería.

Aquí queremos notar que debido a las restricciones de nuestra cuenta Educativa en AWS, Amazon MQ no era una opción permitida a utilizar, con lo que utilizamos un servicio IaaS como lo es Amazon EC2 para desplegar allí una máquina virtual con Ubuntu donde instalamos dentro un servicio de cola

de mensajería de RabbitMQ y así poder ofrecerlo como PaaS a nuestra aplicación.

La decisión principal para determinar la arquitectura de esta manera fué que una base de datos relacional era adecuada para el modelo de datos que se necesitaba manejar. Los requerimientos de performance se podían alcanzar y además nos permite mantener la integridad relacional de forma continua de los datos allí almacenados. Además utilizarlo como servicio conectado nos da la flexibilidad de que mañana pueda ser otro proveedor y/o que se pueda escalar verticalmente el motor de BD. A su vez, no hay necesidad de gestionar los backups y/o réplicas de los datos ya que están ofrecidos dentro del paquete PaaS.

La decisión sobre el uso de una cola de mensajería dentro de la arquitectura de nuestro sistema se basó mayormente en que queríamos incrementar la performance usando tácticas para permitir un gran throughput en la publicación de bugs. A la vez desacoplar lo más posible la interacción síncrona de los microservicios y así mejorar los tiempos de respuesta del sistema hacia el cliente.

Debido a que los bugs publicados, finalmente se persisten en una base de datos relacional, los tiempos de acceso de esta podrían llegar a limitar la capacidad de respuesta de nuestro servicio a la creación de bugs. Se implementó entonces la táctica de “Limitar respuesta a eventos” para la cual el sistema al recibir la publicación de un nuevo bug encola un trabajo en una cola de mensajes y responde al cliente. Luego este trabajo es realizado por un proceso worker que desencola el trabajo y lo almacena en la BD. Cualquier retraso en las operaciones de I/O a la base de datos son “amortiguadas” por nuestro servicio de colas de mensajería y habilitan a que la aplicación pueda aceptar una mayor cantidad de req/s.

Otro uso principal de las cola de mensajería es la de la generación de “eventos” con datos. Estos eventos son consumidos por otros microservicios que requieran esos datos publicados en los “eventos”.

4.1. Uso de cola de mensajería y tópicos

Globalmente para toda la arquitectura, utilizamos un servicio de cola de mensajería con el uso de tópicos. La premisa fue que cada microservicio que realice una acción de creación, actualización o eliminación de elementos importantes para el sistema en general, emita un “evento” en un tópico con los datos que fueron afectados. Los demás microservicios interesados en esta acción pueden suscribirse a este tópico y así recibir el evento y los datos afectados y realizar las acciones que encuentren pertinentes para ese caso. Esta **coreografía** entre microservicios habilita que cada microservicio se construye solamente actuando sobre lo que debe hacer con los mensajes que recibe.

Como ejemplo podemos destacar eventos como la publicación desde el microservicio de usuarios el objeto usuario recientemente creado en el tópico ``bug.create``. Estos microservicios como el de BUGS, REPORTES y COSTEXPLORER consumen este mensaje y realizan las acciones pertinentes.

- BUGS guarda una copia de los datos del usuario y su id para futuras asignaciones de este usuario a algun bug.
- REPORTES lleva una copia de los nombre de los usuarios y sus id para en consecuencia poder generar reportes con los datos de los usuarios.

- COSTEXPLORER escucha este tópico a fin de llevar cuenta de cuántos usuarios se crean en el sistema y así poder cobrar acorde a la cantidad de usuarios que posee cada organización. Sucede lo mismo cuando ocurre la creación o actualización de un bug. El microservicio de bugs emite el bug en los tópicos *bug.create* y *bug.update* respectivamente. Así otros servicios pueden suscribirse a estos tópicos y actuar en consecuencia.

La creación de bugs (*bug.create*) es escuchada por:

- El microservicio de BUGS, para almacenar los bugs en la base de datos.
- El microservicio de COSTOS, para llevar la cuenta de cuantos bugs fueron creados en la organización y así cobrar por el uso del sistema.
- El microservicio de REPORTES, para tener una copia de los bugs que existen en el sistema a fin de generar los reportes solicitados.

La actualización de un bug (*bug.update*) es escuchada por:

- El microservicio de REPORTES, para llevar la cuenta de cómo se han modificado los bugs a fin de crear reportes actualizados para los usuarios.
- El microservicio de NOTIFICACIONES, con el fin de almacenar o despachar notificaciones inmediatas por correo a aquellos usuarios que lo hayan solicitado.

También se utilizó el servicio de colas de mensajería para el envío de correos electrónicos. Cada vez que sea necesario generar una notificación por correo, ya sea para realizar una invitación o enviar notificaciones de bugs asignados, se genera un evento *notification.email*. Que es consumido por el microservicio MAILSENDER que despacha los emails.

Los servicios de Centinela encolan estos trabajos de envío de correo electrónico y el worker se encarga de desencolados y enviar los correos.

Como nota final de esta sección, recomendamos al lector ver [oportunidades de mejora](#) en el apartado reintento exponencial de trabajos para ver cómo se podría mejorar el manejo de la cola de mensajería.

4.2. Descomposición en microservicios

Una de las tareas principales de esta entrega era la correcta descomposición de nuestro monolito existente hacia una arquitectura basada en microservicios.

Fueron varias las horas de planificación y discusión para poder acordar cómo y en cuantos microservicios deberíamos separar la arquitectura. Tal como fue mencionado anteriormente, la premisa fue intentar desacoplar lo máximo posible la interacción síncrona entre servicios.

Una vez resuelta esta arquitectura procedimos a pensar cómo descomponer nuestro monolito.

Utilizamos como base la descomposición por subdominios. Uno de los criterios que primaron para la división fue la independencia de dominios. Separar los microservicios de tal forma que cada uno tenga la responsabilidad de manejar una o más entidades del dominio para todo el sistema. Se puede apreciar por ejemplo que el microservicio de Usuarios que es quien gestiona todo lo relacionado con los datos de los usuarios. Este es el único con la potestad de mantener a los usuarios, todos los demás microservicios que utilicen datos de los usuarios, o bien harán consultas a este o recibirán los eventos que este microservicio genere. Si bien otros microservicios pueden llegar a mantener

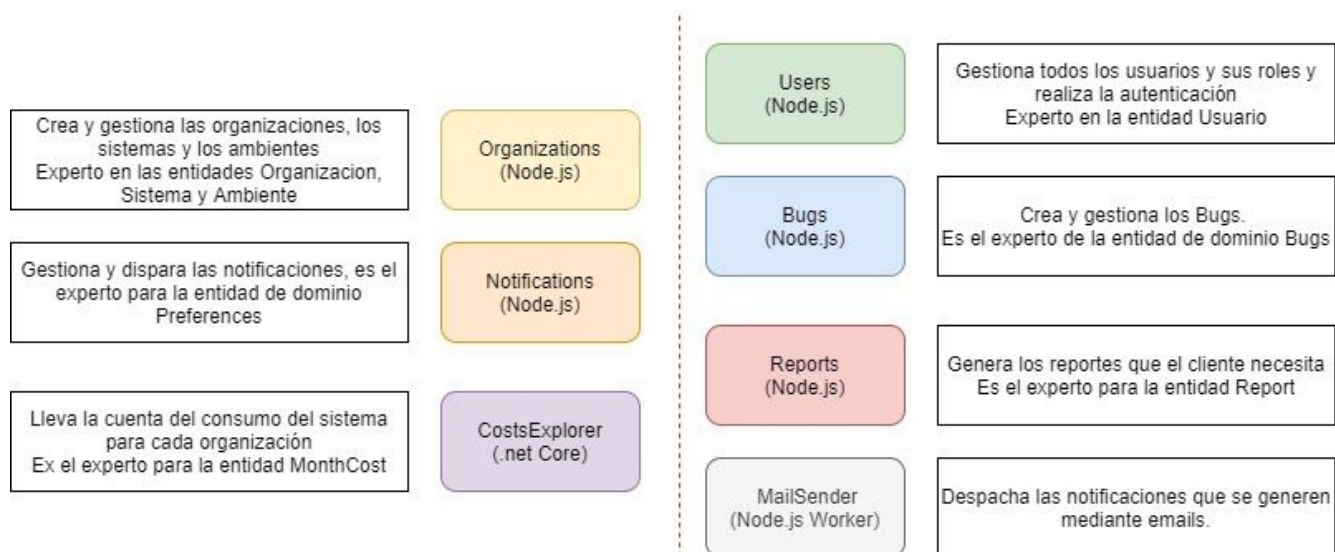
algunos datos de los usuarios, esto son solo los necesarios para poder realizar su tarea y estos datos son mantenidos por los mensajes que pueda publicar USUARIOS.

Un caso destacable es la autenticación. Donde todos los endpoints del ecosistema que requieran autenticación verificarán el token hacia el microservicio de usuarios. El único con la potestad de determinar si un usuario/token es válido.

Sucede lo mismo con el microservicio de Organizaciones, este es el microservicio maestro para la gestión de Organizaciones, Sistemas y Ambientes. Como el caso anterior, otros servicios que requieran datos de estas entidades del dominio consultarán o recibirán eventos generados por este microservicio. Ver el ejemplo de creación de un usuario administrador con una organización, para apreciar como el microservicio de usuarios solicita la información el microservicio de organizaciones para poder crear el usuario en una organización.

BUGS es el microservicio con la potestad de manejar la entidad Bugs de dominio. Toda acción relacionada con bugs es generada y/o notificada por este. Los demás microservicios que requieran datos de bugs lo obtendrán a partir de este.

La descomposición por microservicios y la división por subdominios a su vez nos permitió que podamos codificar microservicios en distintos lenguajes. La única necesidad en este caso es poder comunicarse con el mismo protocolo. Mensajería para recibir/publicar eventos útiles para otros microservicios y HTTP REST para las consultas sincronicas.



4.3. Performance y testing

Para lograr satisfacer que las respuestas a todas las operaciones públicas del sistema se encuentren por debajo de los 350 ms para una carga de hasta 1200 req/m, utilizamos colas de mensajería.

Aquellas acciones que pudieran llegar a tomar tiempo en procesarse son encoladas en una “base de datos en memoria”. De esta forma encolar el trabajo para que pueda realizarse luego y poder así contestar al usuario lo antes posible. Esta “base de datos en memoria” que nos permite cachear la información, tiene muy bajos tiempos de latencia y permite además que se puedan escalar

indistintamente las apis que contestan a los usuarios de aquellos procesos que procesan y realizan los trabajos que requieren mayor tiempo de procesamiento.

En la siguiente imagen de de un test de carga generado con Apache JMeter, en el mismo fue generado con **200** threads simultáneos más **10 threads** consultando el estado de cada uno de los microservicios.

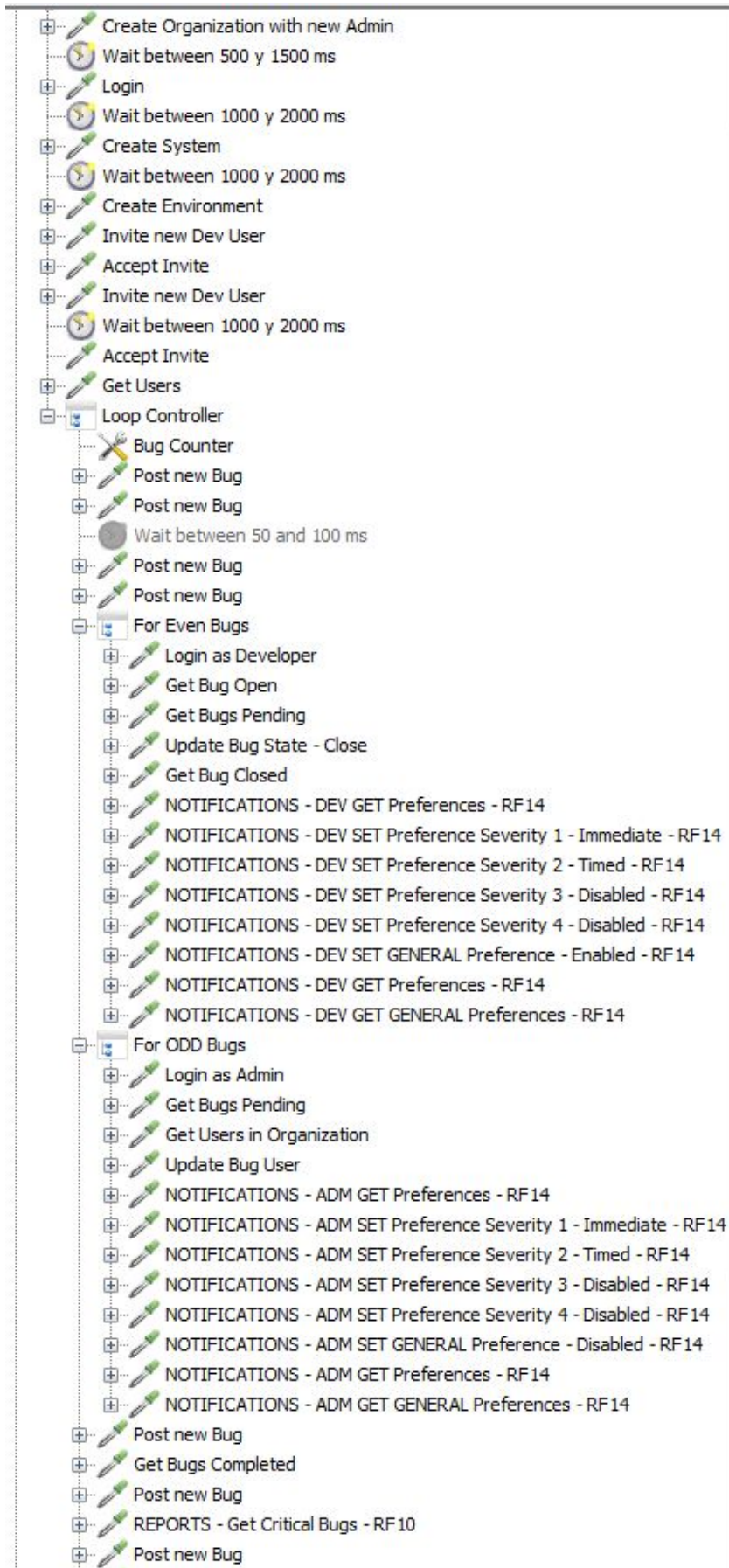
Label	# Samples	Average	Error %	Throughput
PING Bugs microservice ...	1787	49	0.00%	1.3/sec
PING Organizations micr...	1788	31	0.00%	1.3/sec
PING Users microservice...	1786	50	0.00%	1.3/sec
Create Organization wit...	193	112	0.00%	3.2/sec
PING Notifications micro...	1787	31	0.00%	1.3/sec
PING Reports microservi...	1785	29	0.00%	1.3/sec
Login	200	82	0.00%	3.1/sec
PING CostsExplorer micr...	1782	8	0.00%	1.3/sec
Create System	200	29	0.00%	3.0/sec
Create Environment	200	16	0.00%	3.0/sec
Invite new Dev User	400	18	0.00%	5.0/sec
Accept Invite	400	107	0.00%	5.1/sec
Get Users	200	10	0.00%	2.8/sec
Post new Bug	20770	8	0.00%	15.4/sec
Login as Admin	2020	84	0.00%	2.1/sec
Get Bugs Pending	1985	42	0.00%	1.5/sec
Get Users in Organization	1000	8	0.00%	49.1/min
Update Bug User	1000	33	0.00%	48.9/min
NOTIFICATIONS - ADM ...	2000	16	0.00%	1.6/sec
NOTIFICATIONS - ADM ...	1000	13	0.00%	48.8/min
NOTIFICATIONS - ADM ...	1000	13	0.00%	48.9/min
NOTIFICATIONS - ADM ...	1000	13	0.00%	48.7/min
NOTIFICATIONS - ADM ...	1000	13	0.00%	48.6/min
NOTIFICATIONS - ADM ...	1000	12	0.00%	48.6/min
NOTIFICATIONS - ADM ...	1000	6	0.00%	48.6/min
Get Bugs Completed	1857	36	0.00%	1.4/sec
REPORTS - Get Critical B...	1838	18	0.00%	1.4/sec
REPORTS - Get Bug stat...	1816	47	0.00%	1.4/sec
COSTS - Get Costs - RF12	1812	9	0.00%	1.4/sec
REPORTS - Get Top 10 ...	1805	18	0.00%	1.4/sec
REPORTS - Get >2 days...	1801	15	0.00%	1.5/sec
Login as Developer	991	84	0.00%	49.5/min
Get Bug Open	986	50	0.00%	49.7/min
Update Bug State - Close	979	34	0.00%	49.5/min
Get Bug Closed	968	13	0.00%	49.1/min
NOTIFICATIONS - DEV ...	1849	18	0.00%	1.6/sec
NOTIFICATIONS - DEV ...	949	14	0.00%	48.6/min
NOTIFICATIONS - DEV ...	934	13	0.00%	48.0/min
NOTIFICATIONS - DEV ...	923	13	0.00%	47.5/min
NOTIFICATIONS - DEV ...	909	13	0.00%	46.9/min
NOTIFICATIONS - DEV ...	899	13	0.00%	46.6/min
NOTIFICATIONS - DEV ...	876	7	0.00%	45.8/min
TOTAL	70281	23	0.00%	51.0/sec

Como podemos apreciar el “throughput” (el cual se calcula como **número de request/unidad de tiempo**, siendo el tiempo calculado desde el inicio del primer sample hasta el final del último) nos da **51.0 req/sec** lo cual se traduce a **3060 req/min** con un promedio de **23 ms** por request, superando **ampliamente** los 350 ms deseados en este requerimiento.

Queremos destacar un tiempo de respuesta que consideramos sumamente importante que es el de la creación de bugs. En este caso, el promedio de respuesta para la creación de bugs fue de **8 ms**. Basado en un test con la creación de **20770 BUGS**.

Para el testing que se realizó se desarrolló un flujo completo de la actividad que puede tener el sistema de forma diaria. A continuación se muestra una captura de la interfaz de JMeter y una explicación de las tareas que se realizaron.

Las ejecución de pruebas define 2 thread groups, un thread group (*Default behaviour*) que representará el comportamiento de los usuarios y los sistemas que envían bugs a la aplicación y otro thread group (*Ping*) que estará monitoreando el estado general de la aplicación.



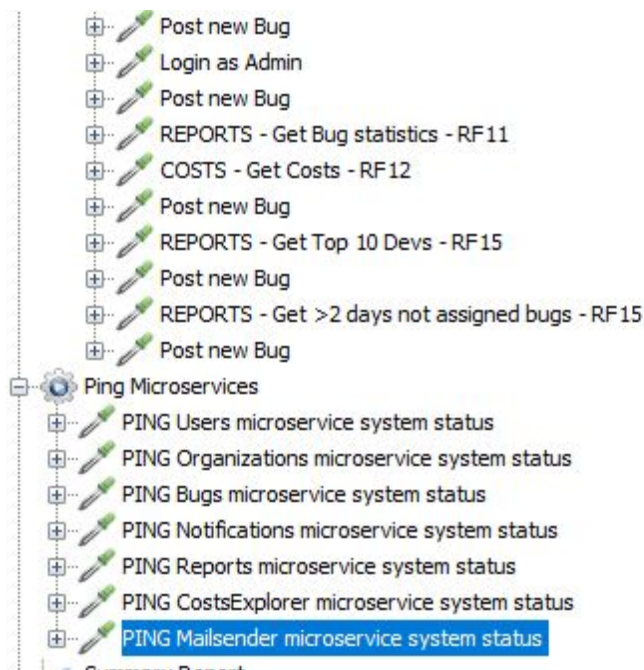
El primer thread group establece la creación de una organización con un nuevo usuario Administrador. Luego se simula el tiempo que le toma a este loguearse en el sistema. Se espera nuevamente un tiempo de entre 2 o 3 segundos para que el nuevo administrador logueado pueda crear un sistema y un ambiente.

Luego que el sistema y el primer ambiente son creados. Se hace la invitación a dos nuevos desarrolladores, los cuales a continuación aceptan la invitación. Finalmente se hace una solicitud para obtener todos los usuarios de la nueva organización.

A continuación de la etapa inicial de creación de la organización, sistema y ambiente y de invitar a los desarrolladores que usarán el sistema. Se definió en JMeter un Loop que se ejecuta por **220** ciclos. Para simular la carga de 220 bugs y el comportamiento de 220 solicitudes hacia nuestro servicio.

En el caso de los ciclos **pares**, el proceso continúa con el login al sistema como un usuario desarrollador. Luego se simula que el desarrollador obtenga la lista de bugs pendientes. De estos bugs pendientes, JMeter luego obtiene los detalles de uno de ellos. A el cual cambia su estado a cerrado. Una vez cerrado el bug, se lista un reporte de los bugs en estado cerrado.

Para esta entrega además se agregó que el desarrollador



postean 11 bugs por iteración.

El otro thread group (*Ping Microservices*) solamente se realiza la tarea de lanzar el sanity check para corroborar el estado de cada uno de los microservicios, este se repite cada 1 segundo y con 10 threads concurrentes haciendo las consultas.

consulte sus preferencias y realice cambios en las mismas.

Para el caso de los ciclos **impares**, JMeter realiza un login como un usuario administrador, obtiene los bugs pendientes, luego obtiene los usuarios que existen en la organización y luego asigna un bug a un desarrollador, luego hace pruebas de cambiar sus notificaciones por severidad.

Al cierre de este loop, para los ciclos pares e impares, Jmeter solicita una lista de los bugs completados y luego el reporte de los 5 errores críticos del sistema.

Además se hace la solicitud de consumo y costos y la prueba de los otros reportes solicitados

Durante todas estas acciones descritas se simula el posteo de bugs. En total se

4.4. Confiabilidad y disponibilidad

Para cumplir con este requerimiento, se creó un endpoint dentro de cada microservicio con el cual se se puede verificar el estado de los diferentes componentes de cada sistema retornando un json con el estado de cada uno de estos.

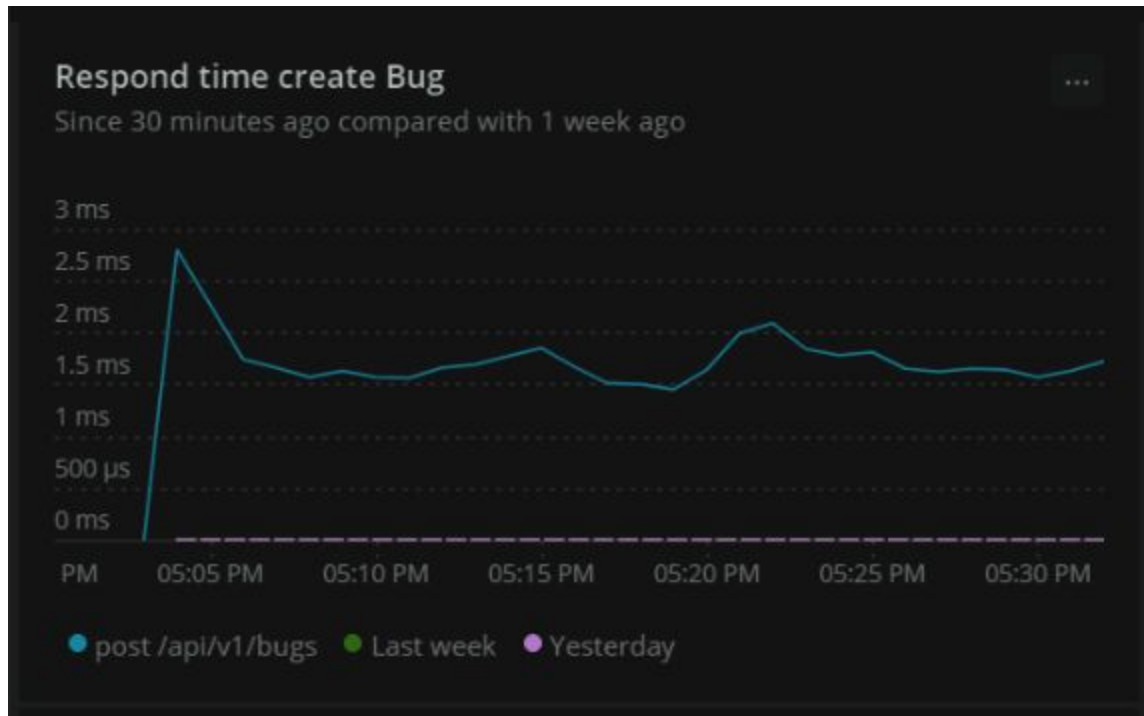
A continuación dejamos las capturas de su funcionamiento

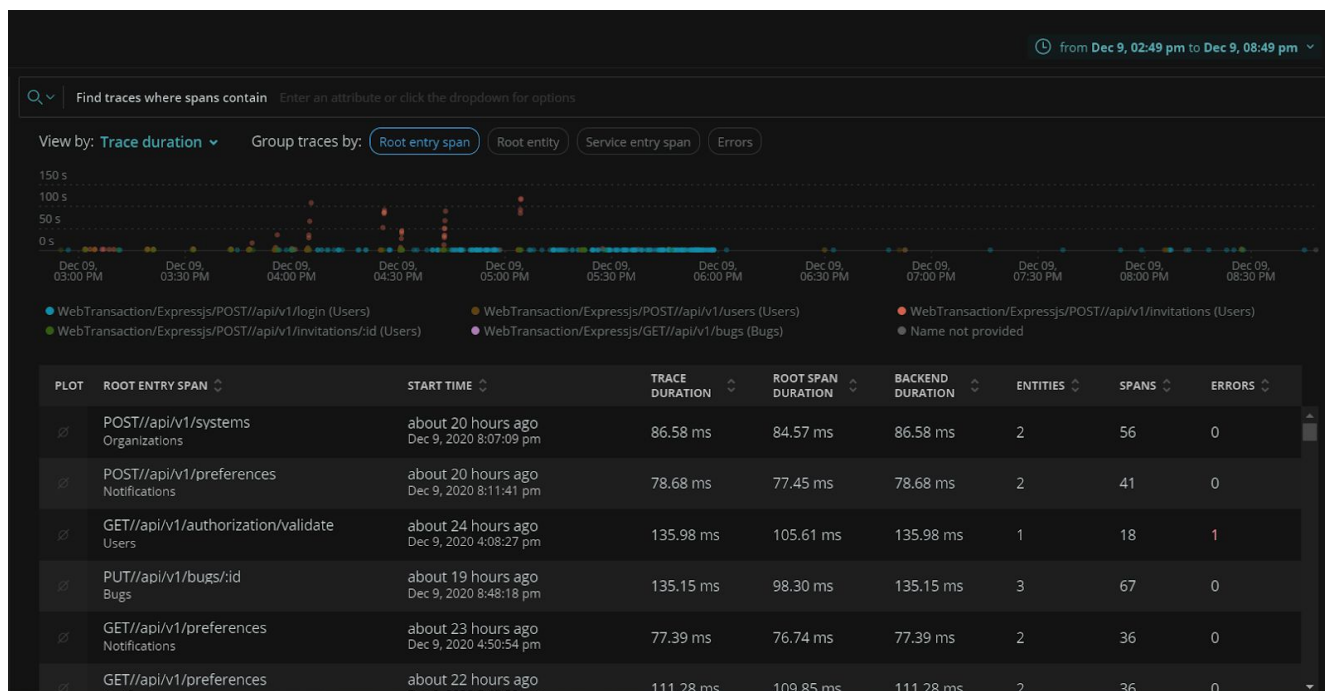
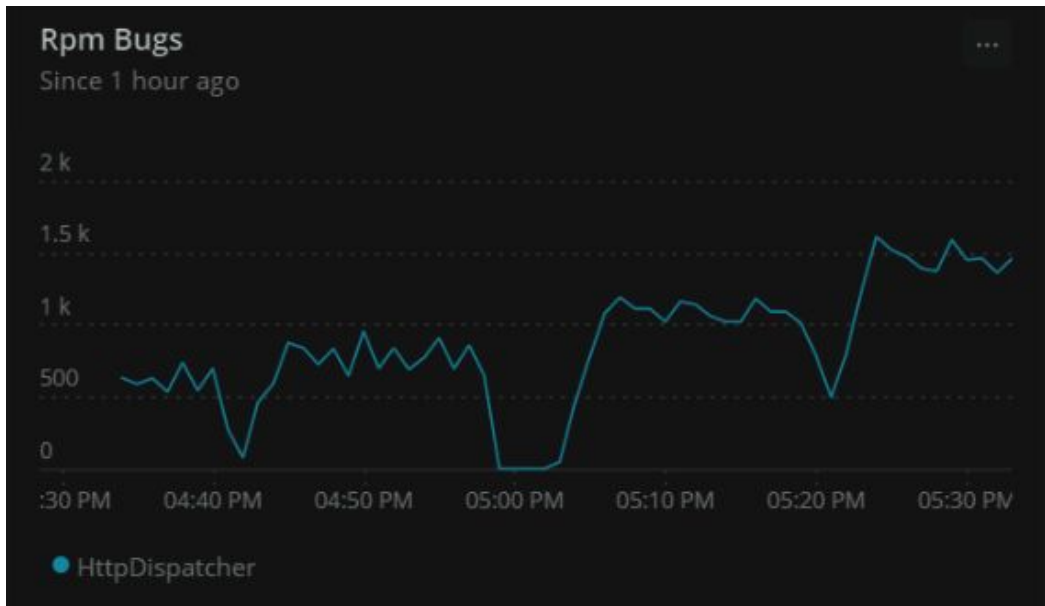
Todo OK	Problemas con BD y MSQ
<pre>{ "status": "OK", "data": { "postgresBd": "pong", "queueStatistics": { "connection_closed": 0, "channel_closed": 0, "consumer_deleted": 0, "exchange_deleted": 0, "queue_deleted": 0, "vhost_deleted": 0, "node_node_deleted": 0, "channel_consumer_deleted": } } }</pre>	<pre>{ "status": "Alerted", "data": { "postgresBd": "Postgres is dead", "queueStatistics": "Could not get queue statistics" } }</pre>

Como se puede apreciar se devuelve el estado de cada parte del sistema, siendo **OK** respuesta exitosa y en caso contrario un mensaje indicando que no hubo respuesta.

También podemos ver el estado de las colas indicando que tan ocupadas se encuentran y los estados de los mensajes que tienen dentro.

A la vez nuestros microservicios reportan su estado a la herramienta de telemetría cloud New Relic® Aquí creamos consolas donde podemos monitorear en tiempo real los tiempos de respuesta de cada endpoint, la cantidad de requests por minuto y otros datos importantes para la correcta monitoreabilidad de nuestro software.





Para el caso de mantener la disponibilidad y poder cumplir con el 99% de uptime. La construcción de los sistemas se dockerizó a fin de poder iniciar múltiples instancias del mismo contenedor. Cada uno de nuestros servicios se encuentra alojado detrás de un reverse proxy que da la posibilidad de balancear la carga entre varias instancias de cada servicio.

Hoy el reverse proxy es un contenedor más dentro de nuestro ambiente y se encarga de distribuir los request a los contenedores correctos. En un ambiente cloud. La arquitectura contempla el uso de varias instancias del contenedor de reverse-proxy o la implementación de **Amazon API Gateway** en entornos de máxima demanda.

4.5. Configuración y manejo de secretos

Toda información de configuración no sensible y la cual no era necesario cambiarla en tiempo de ejecución se especificó en archivos de configuración internos a la aplicación en la carpeta config.

Por otro lado, toda la información de configuración sensible se maneja a nivel de variables de entorno, las cuales fueron cargadas en AWS y leídas desde ahí, no estando cargadas en el código fuente de nuestro entorno.

Para mejor comprensión del DevOps se dejó en cada proyecto un archivo **.env-example** con un ejemplo de las variables de entorno requeridas por microservicio. Allí las contraseñas están ocultas. A su vez el equipo de desarrollo tuvo especial cuidado de no publicar claves o información sensible en los repositorios.

Algo importante que queremos destacar.

El repositorio MASTER en el cual se encuentran los scripts de deployment mediante el uso de docker compose y otras configuraciones SI contiene secretos de uso de deployment en el ambiente de desarrollo.

Queremos notarlo debido a que el equipo conoce que no es ideal, pero para este fin educativo era importante que todos manejamos el mismo ambiente de deployment para las pruebas del sistema. Al mismo tiempo que los docentes tengan acceso a estos scripts para conocer cómo hicimos el despliegue.

4.6. Autenticación y autorización

Para el uso de autenticación y autorización se utilizó manejo de tokens. La arquitectura elegida es la autenticación mediante uso de tokens de JSON Web Tokens en las request que requieran autorización. Esta permite que la autenticación de un actor ocurra sin la necesidad de enviar contraseñas en cada solicitud y habilita el manejo sesiones y la transferencia de datos dentro de estos tokens.

Cuando un cliente se registra, su contraseña es almacenada en la base de datos de usuarios forma encriptada. Esto ofrece la ventaja de que en caso de comprometerse la seguridad de la base de datos de usuarios, las contraseñas de nuestros clientes no quedan comprometidas. El proceso de login ocurre comparando las versiones encriptadas de la contraseña, y en los casos que este login sea satisfactorio. El sistema retorna un web token que el cliente utilizará para cada uno de los siguientes requests que realice. Estos web tokens tienen una vigencia de 1hr (configurable) los cuales deben ser renovados para mantener la sesión activa. Además estos tokens llevan información útil del usuario y su organización. Esta información es utilizada por nuestro sistema de backend y frontend y que en caso de ser alterada su integridad provocaría un funcionamiento incorrecto del sistema. JSON Web Tokens permite validar que esta información que viaja en el token no sea alterada, ya que en caso que lo fuera, el token dejaría de ser válido.

El manejo de la autorización en nuestro sistema se basó en el uso del control de acceso basado en roles. Elegimos esta táctica para simplificar y a su vez asegurar el correcto control de acceso de los usuarios a los recursos. Cada uno de los endpoints publicados tiene definido cuales son los roles que

pueden utilizarlo y es mediante el uso de los tokens antes mencionados que se puede identificar y obtener el rol de un usuario.

Cada intento de login y cada verificación de token queda registrada en el log de la aplicación. Con el fin de poder determinar quienes accedieron al servicio y además poder hacer un diagnóstico en caso de error.

Como se mencionó anteriormente. Todos los microservicios delegan la validación del token al microservicio de usuarios. Este actúa como Identidad federada para el sistema y en base a su respuesta satisfactoria o insatisfactoria, los demás microservicios obtienen las propiedades que se embeben en el token y dan acceso a sus endpoints.

Tomamos la decisión de utilizar el mismo mecanismo para asignar las claves de aplicación. Cada vez que se crea un “entorno”, el sistema genera un JSON WebToken (esta vez sin caducidad) que debe ser adjuntado en las request que los sistemas externos realizan al publicar sus bugs. Este token contiene además toda la información relevante del ambiente al cual debe ser publicado el bug con el fin de que nuestro sistema pueda rápidamente identificar la organización, del sistema y el entorno al cual pertenece el nuevo bug reportado.

4.7. Envío de Notificaciones.

Un desafío que nos surgió fue de cómo solucionar el envío de notificaciones cuando escalan la cantidad de instancias de este servicio.

El microservicio de notificaciones corre trabajos agendados cada 10 minutos (configurable) y obtiene cuales son los usuarios que tienen preferencias que cumplen con el criterio de procesarse.

En el caso que múltiples instancias funcionen a la vez, todas correrían las tareas de notificación al mismo momento.

Para evitar que múltiples instancias envíen la misma notificación, una vez procesada la misma, se marca en la base de datos como enviada. Con un timestamp en el atributo lastsent. Otras instancias que quieran obtener las preferencias de notificación pendientes no obtendrán esta notificación ya que ya fue marcada como enviada ese día.

A su vez, el proceso de generación de notificaciones no es un proceso que tome tiempo de realizarse ya que no es este el microservicio que finalmente despache los correos y que deba esperar la respuesta del servidor de correo. Sino que, este solamente encola los mensajes generados para que el microservicio de MailSender los consuma a su ritmo y envíe los correos.

También, la frecuencia de procesamiento de la tarea agendada se configura mediante variables de entorno y es posible además que cada instancia tenga configuraciones diferentes.

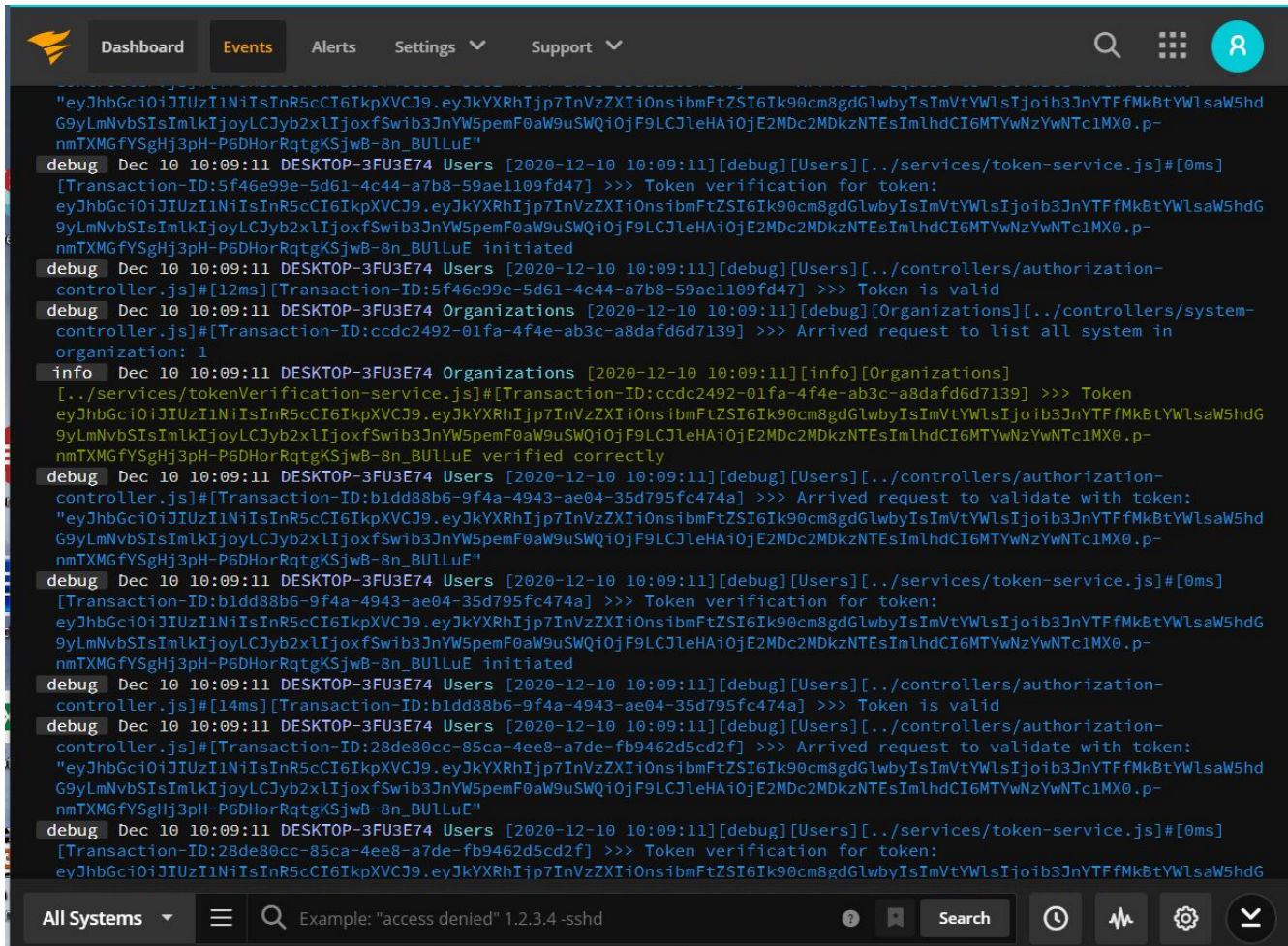
4.8. Seguridad y Centralización de Logs

Para favorecer la seguridad, todos nuestros endpoints fueron diseñados para responder códigos de error acordes a la situación. Esto permite que, ya sea nuestro frontend como otros clientes que utilicen nuestro SaaS sepan interpretar y actuar acorde el tipo de error.

Para el control, toda actividad del sistema es logueada y centralizada dentro de un syslog server.

Para ello implementamos una librería de logger, centinela-logger, que permite configurar un servicio

de syslog que centralice todos los logs de aplicación. -debido a las restricciones que tuvimos para utilizar amazon para esta entrega utilizamos el servicio PaperTrail de SolarWinds - que escucha la publicación de logs por un stream tcp/udp y permite hacer búsquedas del histórico por hasta 7 días.



The screenshot shows a web-based log management interface. At the top, there is a navigation bar with tabs for 'Dashboard', 'Events' (which is active), 'Alerts', 'Settings', and 'Support'. To the right of the navigation bar are search and filter icons. Below the navigation bar, the main area displays a list of log entries. Each entry is a JSON object representing a log record, including fields like 'timestamp', 'source', 'level', and 'message'. The logs show various system events, including token verification, token validation, and system initialization. At the bottom of the interface, there is a search bar with the text 'Example: "access denied" 1.2.3.4 -sshd' and several icons for filtering and sorting the logs.

```
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7InVzZXI0nsibmFtZSI6Ikp9cm8gdGlwbyIsImVtYWlsIjoib3JnYTFfMkBTYWlsaW5hdG9yLmNvbSIsImkIjoyLCJyb2x1IjoxfSwib3JnYW5pemF0aW9uSWQiojF9LCJleHAiOjE2MDc2MDkzNTesImhhdCI6MTYwNzYwNTc1MX0.p-nmTXMGfYSGHj3pH-P6DHorRqtgKSjwB-8n_BULLuE"
```

```
debug Dec 10 10:09:11 DESKTOP-3FU3E74 Users [2020-12-10 10:09:11][debug][Users][../services/token-service.js]#[0ms][Transaction-ID:5f46e99e-5d61-4c44-a7b8-59ae1109fd47] >>> Token verification for token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7InVzZXI0nsibmFtZSI6Ikp9cm8gdGlwbyIsImVtYWlsIjoib3JnYTFfMkBTYWlsaW5hdG9yLmNvbSIsImkIjoyLCJyb2x1IjoxfSwib3JnYW5pemF0aW9uSWQiojF9LCJleHAiOjE2MDc2MDkzNTesImhhdCI6MTYwNzYwNTc1MX0.p-nmTXMGfYSGHj3pH-P6DHorRqtgKSjwB-8n_BULLuE initiated
```

```
debug Dec 10 10:09:11 DESKTOP-3FU3E74 Users [2020-12-10 10:09:11][debug][Users][../controllers/authorization-controller.js]#[12ms][Transaction-ID:5f46e99e-5d61-4c44-a7b8-59ae1109fd47] >>> Token is valid
```

```
debug Dec 10 10:09:11 DESKTOP-3FU3E74 Organizations [2020-12-10 10:09:11][debug][Organizations][../controllers/system-controller.js]#[Transaction-ID:ccdc2492-01fa-4f4e-ab3c-a8dafd6d7139] >>> Arrived request to list all system in organization: 1
```

```
info Dec 10 10:09:11 DESKTOP-3FU3E74 Organizations [2020-12-10 10:09:11][info][Organizations][../services/tokenVerification-service.js]#[Transaction-ID:ccdc2492-01fa-4f4e-ab3c-a8dafd6d7139] >>> TokeneyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7InVzZXI0nsibmFtZSI6Ikp9cm8gdGlwbyIsImVtYWlsIjoib3JnYTFfMkBTYWlsaW5hdG9yLmNvbSIsImkIjoyLCJyb2x1IjoxfSwib3JnYW5pemF0aW9uSWQiojF9LCJleHAiOjE2MDc2MDkzNTesImhhdCI6MTYwNzYwNTc1MX0.p-nmTXMGfYSGHj3pH-P6DHorRqtgKSjwB-8n_BULLuE verified correctly
```

```
debug Dec 10 10:09:11 DESKTOP-3FU3E74 Users [2020-12-10 10:09:11][debug][Users][../controllers/authorization-controller.js]#[Transaction-ID:b1dd88b6-9f4a-4943-ae04-35d795fc474a] >>> Arrived request to validate with token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7InVzZXI0nsibmFtZSI6Ikp9cm8gdGlwbyIsImVtYWlsIjoib3JnYTFfMkBTYWlsaW5hdG9yLmNvbSIsImkIjoyLCJyb2x1IjoxfSwib3JnYW5pemF0aW9uSWQiojF9LCJleHAiOjE2MDc2MDkzNTesImhhdCI6MTYwNzYwNTc1MX0.p-nmTXMGfYSGHj3pH-P6DHorRqtgKSjwB-8n_BULLuE"
```

```
debug Dec 10 10:09:11 DESKTOP-3FU3E74 Users [2020-12-10 10:09:11][debug][Users][../services/token-service.js]#[0ms][Transaction-ID:b1dd88b6-9f4a-4943-ae04-35d795fc474a] >>> Token verification for token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7InVzZXI0nsibmFtZSI6Ikp9cm8gdGlwbyIsImVtYWlsIjoib3JnYTFfMkBTYWlsaW5hdG9yLmNvbSIsImkIjoyLCJyb2x1IjoxfSwib3JnYW5pemF0aW9uSWQiojF9LCJleHAiOjE2MDc2MDkzNTesImhhdCI6MTYwNzYwNTc1MX0.p-nmTXMGfYSGHj3pH-P6DHorRqtgKSjwB-8n_BULLuE initiated
```

```
debug Dec 10 10:09:11 DESKTOP-3FU3E74 Users [2020-12-10 10:09:11][debug][Users][../controllers/authorization-controller.js]#[14ms][Transaction-ID:b1dd88b6-9f4a-4943-ae04-35d795fc474a] >>> Token is valid
```

```
debug Dec 10 10:09:11 DESKTOP-3FU3E74 Users [2020-12-10 10:09:11][debug][Users][../controllers/authorization-controller.js]#[Transaction-ID:28de80cc-85ca-4ee8-a7de-fb9462d5cd2f] >>> Arrived request to validate with token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7InVzZXI0nsibmFtZSI6Ikp9cm8gdGlwbyIsImVtYWlsIjoib3JnYTFfMkBTYWlsaW5hdG9yLmNvbSIsImkIjoyLCJyb2x1IjoxfSwib3JnYW5pemF0aW9uSWQiojF9LCJleHAiOjE2MDc2MDkzNTesImhhdCI6MTYwNzYwNTc1MX0.p-nmTXMGfYSGHj3pH-P6DHorRqtgKSjwB-8n_BULLuE"
```

```
debug Dec 10 10:09:11 DESKTOP-3FU3E74 Users [2020-12-10 10:09:11][debug][Users][../services/token-service.js]#[0ms][Transaction-ID:28de80cc-85ca-4ee8-a7de-fb9462d5cd2f] >>> Token verification for token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7InVzZXI0nsibmFtZSI6Ikp9cm8gdGlwbyIsImVtYWlsIjoib3JnYTFfMkBTYWlsaW5hdG9yLmNvbSIsImkIjoyLCJyb2x1IjoxfSwib3JnYW5pemF0aW9uSWQiojF9LCJleHAiOjE2MDc2MDkzNTesImhhdCI6MTYwNzYwNTc1MX0.p-nmTXMGfYSGHj3pH-P6DHorRqtgKSjwB-8n_BULLuE"
```

Además de esto, al utilizar todos servicios en el mismo proveedor cloud pudimos aprovechar y realizar la comunicación con todos los servicios conectados dentro de una red interna. Estos servicios fueron configurados con contraseñas de acceso. Y estas contraseñas son solo configurables en variables del entorno y no están presentes en el código fuente.

Para la comunicación de el frontend hacia el backend debemos aclarar que no utilizamos protocolos seguros. Si bien la aplicación y sus librerías están preparadas para soportarlo, al ser ésta desarrollada sólo con fines académicos y debido a que existe un costo de contratación de los certificados SSL para permitir la comunicación segura, optamos por no implementarlo pero sí desarrollar todo para poder soportarlo en el futuro.

4.9. Código fuente













Los lenguajes seleccionados para la codificación del sistema fueron **NodeJs** y **C# dotnet core** para el backend y **Angular** para el frontend. A su vez nos apoyamos en un template como explicamos en la sección 2.2.Desafíos y decisiones para agilizar el desarrollo del frontend.

En cuanto a los estándares de codificación, nos basamos en las convenciones de codificación de NodeJS (<https://docs.npmjs.com/misc/coding-style>) para lo cual configuramos un formateador de texto que nos ayudará con dicha tarea.

Para el manejo de los branches utilizamos Gitflow como el requerimiento lo solicita, de forma comenzamos creamos dos branches, master y develop. Para el desarrollo de cada nueva funcionalidad se sale de develop llamando a la nueva rama “feature/nombre_funcionalidad” mientras que los fix serán “fix/nombre_fix”.

Una vez finalizado el trabajo en esa feature se hace merge a develop.

Como cada microservicio debe ser considerado como un producto individual, el equipo optó por crear repositorios independientes para cada microservicio. Estos se encuentran dentro de la organización de github ArqSoftPractica - <https://github.com/ArqSoftPractica>

12 repositories in the Larrosa-Settimo-Zawrzykraj team		
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-bugreporter-node	 Private updated 5 days ago	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Bugs	 Private updated yesterday	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Chassis	 Private template updated 2 days ago	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-CostExplorer	 Private updated 22 hours ago	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Frontend	 Private updated 26 days ago	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Logger	 Private updated 4 days ago	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-MailSender	 Private updated yesterday	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Main	 Private updated 3 hours ago	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Notifications	 Private updated yesterday	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Organizations	 Private updated yesterday	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Reports	 Private updated yesterday	Admin ▾
ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Users	 Private updated yesterday	Admin ▾

4.10. Pruebas

Para satisfacer con este requerimiento, se generaron los archivos correspondientes tanto las colecciones de postman como los scripts de JMeter. Estos se encuentran ubicados dentro del repositorio Main dentro de la carpeta **tests**, dentro de esta carpeta están identificadas con nombre las carpetas correspondientes a los archivos de las pruebas.

4.11. Identificación de fallas

Como ya nos referimos en la sección 4.5. Seguridad, los logs son guardados **SolarWinds** mediante el stream de los mismos hacia Paper Trail donde los estamos almacenando por más de 7 días. Para crear las diferentes entradas de los registros de logs utilizamos la librería centinela-logger la cual usa la librería winston para realizar las acciones de logueo.

En logger definimos diferentes niveles para las entradas del log según lo que se quisiera indicar, estos niveles son: info, debug, warning, error y fatal; con ellos pudimos manejar los diferentes tipos de mensajes que queríamos loguear y así saber gracias a los niveles que había pasado.

A su vez cada entrada del log fue armada de la siguiente forma:

[Fecha y hora en formato YYYY-MM-DD HH:MM:SS] [Nivel de información] [Archivo donde se está realizando la entrada del log] >>> Mensaje de la entrada del log.

A la hora de armar el mensaje de la entrada para el log, también se pueden mandar los tiempos de inicio y fin para de esta forma llevar un seguimiento de cuánto demoran las diferentes acciones en el sistema.

De la misma forma que logueamos los tiempos de inicio y fin de las operaciones, también se registra un Transaction-ID que permite hacer el tracking de una transacción por todos los microservicios que participaron.

4.12. Uso de técnicas de caching

En el caso de las acciones del usuario donde se necesitaba obtener datos de más de un microservicio, se optó por cachear datos en el browser del usuario, esto a efectos de mejorar la performance para no hacer más de un request en cada acción.

En algunos casos se tomó la decisión de guardar datos en el Session Storage (por ejemplo algunos datos básicos de facturas, para poder ver el detalle luego si el usuario seleccionaba una factura en concreto (estos datos se pierden al cerrar la ventana. En otros casos se optó por guardar en Local Storage, para que no se pierdan inmediatamente (datos básicos como datos del usuario, el token de sesión, y nombres de sistemas y ambientes). Esto último permite que el usuario por ejemplo pueda hacer click en un link a un bug desde su cuenta de correo y pueda abrirlo en el navegador, sin tener que volver a loguearse.

4.13. Tenancy

Dado que en un principio se estima una cantidad de usuarios pequeña, y se pretende desarrollar un sistema de funcionalidad común a muchos usuarios, se optó por desarrollar una arquitectura Multi-tenant, esto es, una arquitectura donde la misma instancia de software podrá atender a varios clientes, con una base de datos común a todos los usuarios. Esto permite tener una mejor economía de escala y mantenimiento, optimizando recursos y distribuyendo costos entre todos los clientes, y por lo tanto, brindando un producto más económico a los mismos.

4.14. Microservice Chassis

Para la presente entrega el equipo tomó la decisión de utilizar el patrón Microservice Chassis, la cual fue de gran ayuda para establecer un estándar entre los distintos micro servicios a desarrollar.

Se pasó a confeccionar un repositorio exclusivo para tal fin

(<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Chassisykraj-Chassis>), del cual se realizó un fork al resto de los repositorios cuyo proyecto se realizó en el lenguaje NodeJs. Para el repositorio de C# se tomó la decisión de no realizar un chassis, dado que era uno solo.

Para este chassis se resolvieron varias cosas comunes a todos los proyectos:

- Servicio de chequeo de token contra el microservicio de Usuarios
- Endpoint de Healthcheck
- Logger
- Conexión con Postgres
- Conexión con RabbitMq
- Servicio de Swagger
- Agente de New Relic
- Middleware para tracing (generación y recuperación de transaction ids en las colas de mensajería, en los requests de usuario y hacia otros microservicios).

5. Proceso de deployment y DevOps

5.1. Creación de imágenes docker y contenedores

Parte de este proyecto educativo es el de la familiarización y utilización de plataformas cloud para el despliegue del sistema desarrollado.

La base de la construcción es el uso de contenedores con el fin de:

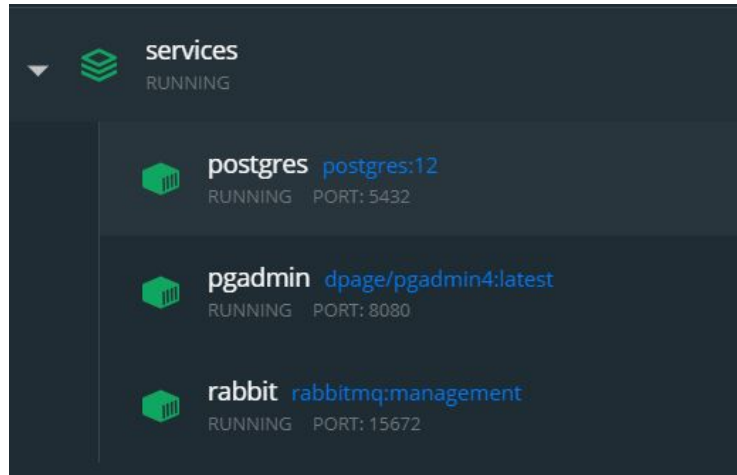
- Mejorar la eficiencia de la utilización de los recursos disponibles
- Permitir que existan ciclos de desarrollo y deployment mucho más rápidos.
- Garantizar la portabilidad, ya que son independientes a la infraestructura en la cual se despliegan.

Construcción:

Para la construcción de nuestro software utilizamos la tecnología de contenedores docker con las herramientas docker run y docker-compose.

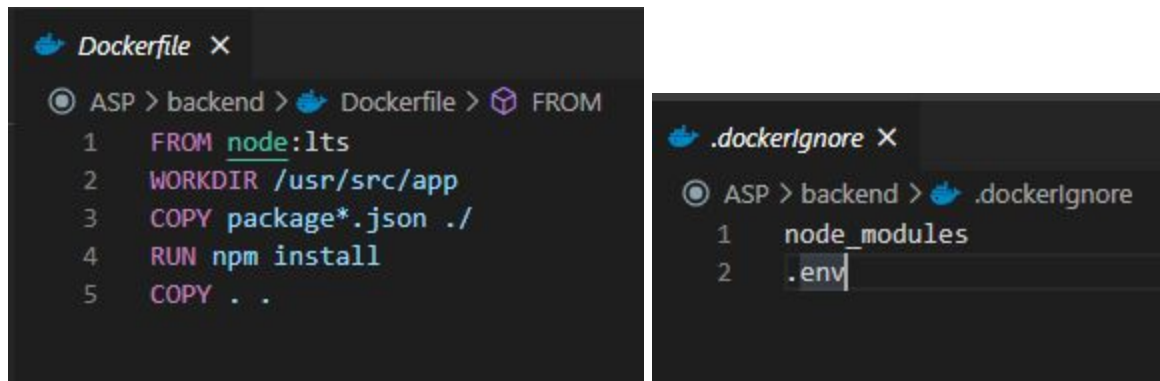
Para el proceso de desarrollo utilizamos Docker desktop para alojar los contenedores. En nuestro ambiente tenemos alojados los siguientes contenedores para ofrecer los servicios de los cuales la aplicación se conecta.

- Postgres - Contenedor que aloja una base de datos relacional.
- RabbitMQ - contenedor que ofrece el servicio de colas de mensajería.
- PgAdmin - Contenedor que aloja un WebService para la gestión de base de datos Postgres.



Para la construcción del ambiente de cloud, utilizamos docker-compose con el fin de orquestar la construcción de las imágenes de todos los contenedores necesarios y Dockerfile para la definir las configuraciones de cada una de nuestras imágenes.

- Cada microservicio de nuestro sistema contiene un Dockerfile que determina cómo se construye su contenedor. Para los contenedores de NodeJS, la construcción de la imagen, es la siguiente.



```
Dockerfile X
ASP > backend > Dockerfile > FROM
1 FROM node:lts
2 WORKDIR /usr/src/app
3 COPY package*.json ./
4 RUN npm install
5 COPY . .

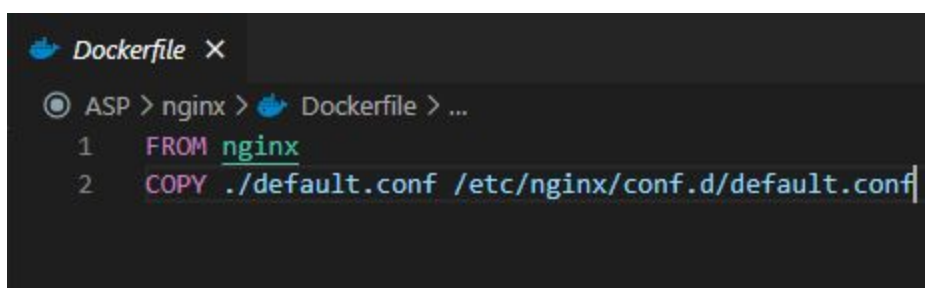
.dockerignore X
ASP > backend > .dockerignore
1 node_modules
2 .env
```

El archivo Dockerfile da las instrucciones de obtener la imagen de NodeJS long term support. Luego da la instrucción copiar el archivo *package.json* para continuar con la instalación de las dependencias requeridas para nuestro código.

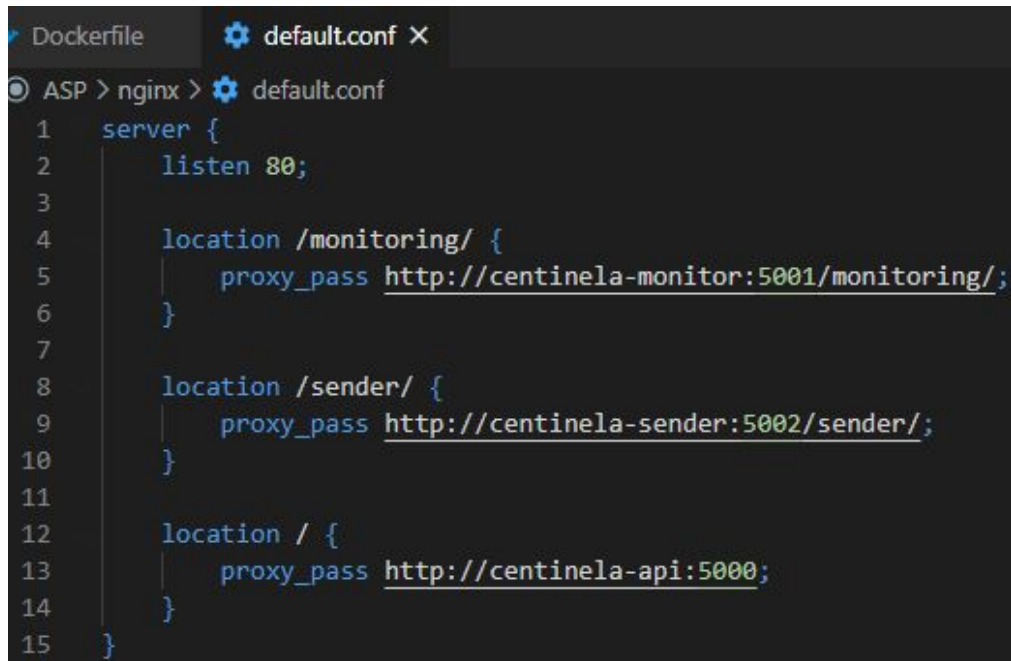
Al final copia todos los otros archivos del sistema, codebase y variables de entorno, en conjunto con el archivo *.dockerignore* para así evitar copiar las dependencias y archivos no necesarios en la imagen.

- Otra imagen que se construye durante el deployment es la del balanceador de carga NGINX.

En este caso, el *dockerfile* indica que se debe descargar la última versión de la imagen pública de NGINX y que luego se copie el archivo de configuración *default.conf* construido para que funcione en nuestro entorno.

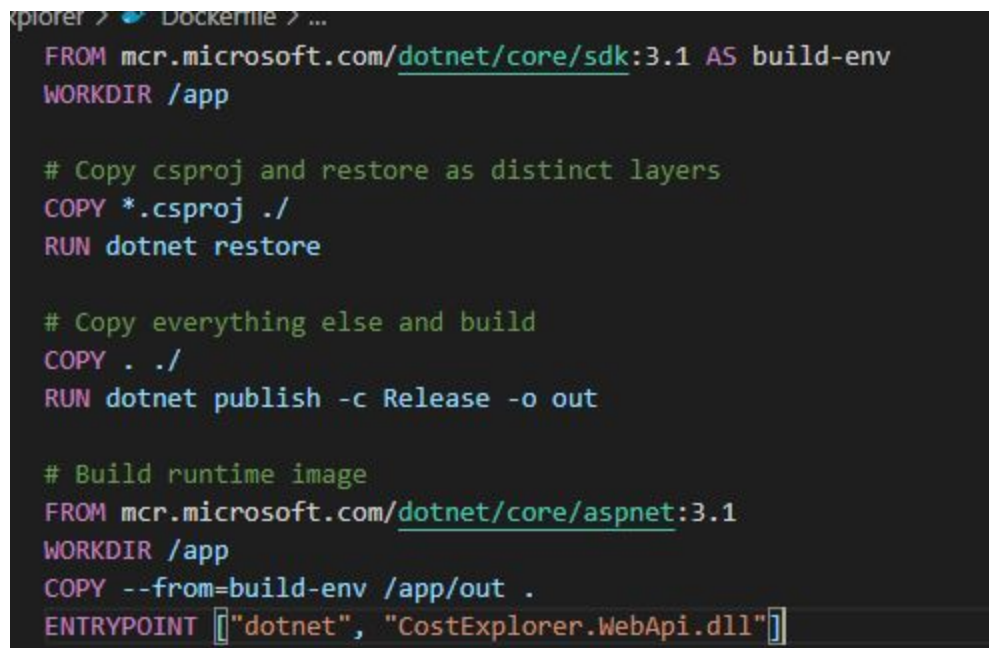


```
Dockerfile X
ASP > nginx > Dockerfile > ...
1 FROM nginx
2 COPY ./default.conf /etc/nginx/conf.d/default.conf
```

```
Dockerfile  default.conf X
ASP > nginx > default.conf
1  server {
2      listen 80;
3
4      location /monitoring/ {
5          proxy_pass http://centinela-monitor:5001/monitoring/;
6      }
7
8      location /sender/ {
9          proxy_pass http://centinela-sender:5002/sender/;
10     }
11
12     location / {
13         proxy_pass http://centinela-api:5000;
14     }
15 }
```

- Otra imagen destacable es la del contenedor de CostsExplorer el cual se construye en base a un contenedor de net core 3.1. Realiza la compilación de nuestro código y ejecuta el software de las Web API.



```
CostsExplorer > Dockerfile > ...
FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build-env
WORKDIR /app

# Copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out

# Build runtime image
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "CostExplorer.WebApi.dll"]
```

5.2. Compose de Servicios y Despliegue.

El orquestador de estas imágenes y el encargado de construir los contenedores en el ambiente cloud es el archivo *docker-compose-cloud.yml*.

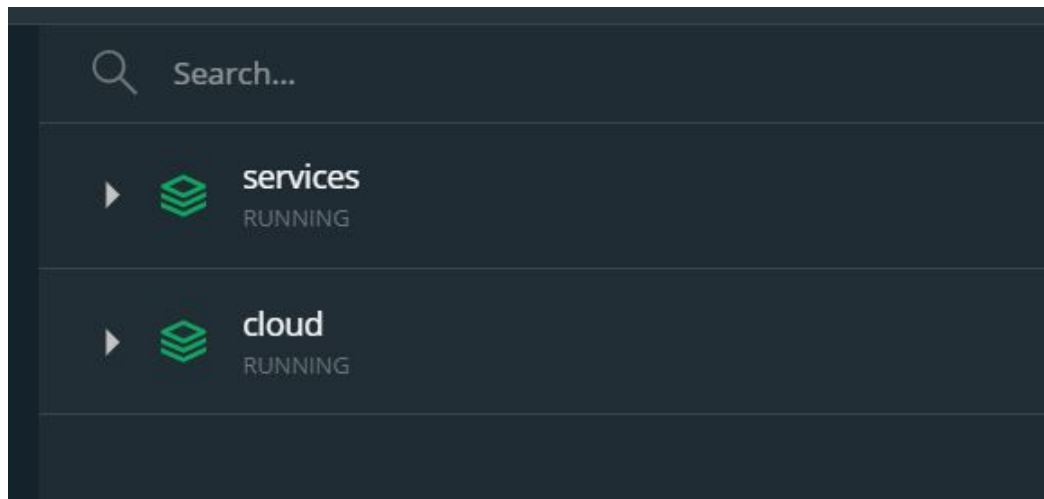
Este crea cada uno de los contenedores que se utilizarán en el ambiente cloud utilizando cada una de las imágenes previamente construidas por el DockerFile. Se agrega captura de una porción del *docker-compose-cloud.yml*.

```
version: "3.7"
services:
  nginx:
    build: ./nginx
    container_name: nginx
    ports:
      - "80:80"
    depends_on: ...
    networks:
      - centinela-net
    logging:
      driver: syslog
      options:
        syslog-address: "udp://logs.papertrailapp.com:39799"
        tag: "{{.Name}}/{{.ID}}"
  users:
    build: ./users
    container_name: users
    command: ["node", "index.js"]
    env_file:
      - production.env
    ports:
      - "5000:5000"
    networks:
      - centinela-net
```

Para iniciar el ambiente cloud en doce desktop basta con ejecutar el script CloudStart.bat. Este realizará la ejecución de docker-compose up para el archivo *docker-compose-cloud.yml*. Una vez iniciados todos los servicios se puede observar el estado del compose.

```
Creating notifications ... done
Creating bugs-api ... done
Creating reports ... done
Creating mailsender ... done
Creating bugs-processor ... done
Creating organizations ... done
Creating bugs-queue-processor ... done
Creating users ... done
Creating reports-queue-processor ... done
Creating costs ... done
Creating nginx ... done
C:\Users\mlset\source\repos\ASP>
```

Una vez los backing services y los microservicios se encuentran iniciados se puede apreciar el siguiente estado en docker desktop.



5.3 Deployment a Produccion.

Nota importante. Para esta instancia no nos fue posible realmente probar el deployment a producción debido a que nos fueron bloqueadas las cuentas educativas de AWS. Igualmente detallaremos cuál sería teóricamente el deployment a AWS.

Deployment del backend.

Para el despliegue a producción utilizamos el servicio de Amazon Elastic Beanstalk.

Para ello subimos el código en un archivo zip.

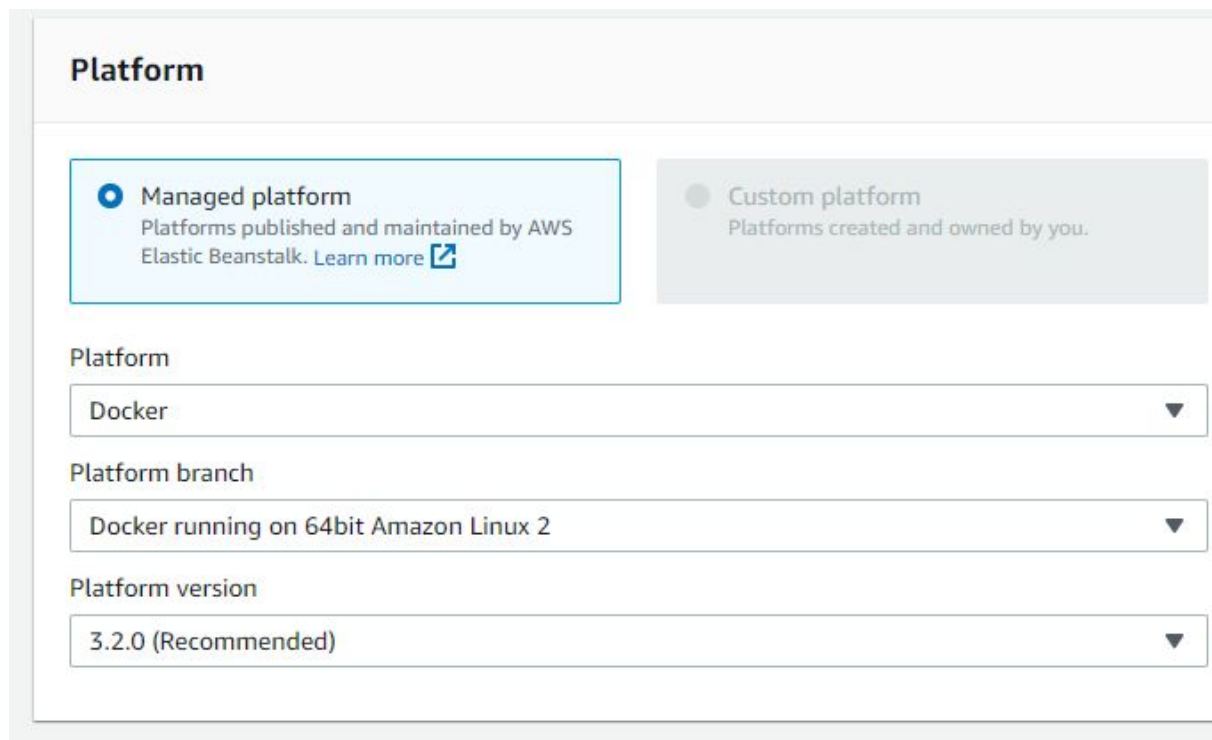
La estructura del archivo comprimido es la siguiente:

Cada carpeta contendrá todo el código fuente del microservicio. La compilación y la instalación de dependencias se realiza al momento de la creación del contenedor. Los archivos .env y production.env son una manera sencilla de pasar las variables de entorno. Pero pueden ser sobrescritas por configuraciones hechas en el ambiente de producción. Como Amazon EBS

```
— deploy.zip
  |— bugs/
  |   |— src/
  |   |— dockerfile
  |   |— .dockerIgnore
  |   |— package.json
  |   |— .env
  |   |— costsexplorer/
  |   |   |— src/
  |   |   |— dockerfile
  |   |   |— .dockerIgnore
  |   |   |— package.json
  |   |   |— .env
  |   |— mailsender/
  |   |   |— src/
  |   |   |— dockerfile
  |   |   |— .dockerIgnore
  |   |   |— package.json
  |   |   |— .env
  |   |— notifications/
  |   |   |— src/
  |   |   |— dockerfile
  |   |   |— .dockerIgnore
  |   |   |— package.json
  |   |   |— .env
  |   |— organizations/
  |   |   |— src/
  |   |   |— dockerfile
  |   |   |— .dockerIgnore
  |   |   |— package.json
  |   |   |— .env
  |   |— reports/
  |   |   |— src/
  |   |   |— dockerfile
  |   |   |— .dockerIgnore
  |   |   |— package.json
  |   |   |— .env
  |   |— users/
  |   |   |— src/
  |   |   |— dockerfile
  |   |   |— .dockerIgnore
  |   |   |— package.json
  |   |   |— .env
  |   |— nginx/
  |   |   |— dockerfile
  |   |   |— default.json
  |   |— docker-compose.yml
  |   |— production.env
```

Cada carpeta contiene lo referido a la imagen y su dockerfile para las instrucciones de construcción.

Este archivo zip se carga en Amazon Elastic Beanstalk y se configura un ambiente para su uso como host de contenedores Docker en una plataforma Amazon Linux 2 que soporta ambientes multicontenedor.



The screenshot shows the 'Platform' configuration section of the AWS Elastic Beanstalk console. It features two radio buttons: 'Managed platform' (selected) and 'Custom platform'. Below these are three dropdown menus for 'Platform' (set to 'Docker'), 'Platform branch' (set to 'Docker running on 64bit Amazon Linux 2'), and 'Platform version' (set to '3.2.0 (Recommended)').

Platform

☒ **Managed platform**
Platforms published and maintained by AWS Elastic Beanstalk. [Learn more](#)

☐ **Custom platform**
Platforms created and owned by you.

Platform
Docker ▼

Platform branch
Docker running on 64bit Amazon Linux 2 ▼

Platform version
3.2.0 (Recommended) ▼

La carga del archivo *zip*, que incluye los binarios y la configuración, crea un nuevo ambiente el cual lanza la construcción e inicio de todos los contenedores declarados en el archivo *docker-compose.yml* como fue explicado anteriormente.

Una vez que el ambiente se encuentra listo se puede apreciar su estado de Health OK.

Centinela-env

[Centinela-env.eba-zhfdmg23.us-east-1.elasticbeanstalk.com](#)

(e-3dh2hrx4ht)

Application name: **Centinela**

Refresh

Actions ▼

Health

Ok

Causes

Running version

centinela-v3-external-redis-1

Upload and deploy

Platform

Docker running on 64bit
Amazon Linux 2/3.2.0

Change

Elastic beanstalk deja a la vista la URL del ambiente por el cual nuestro servicio atenderá las request a través del contenedor NGINX.

Los contenedores requieren ciertas configuraciones que se especifican en un archivo de variables de entorno llamado *production.env*.

Este archivo es utilizado por ElasticBeanstalk para crear todas la variables de entorno y ofrecerlas a los contenedores que las necesiten.

Un ejemplo de estas variables es la definición de los parámetros para la conexión con el servicio RDS ofrecido por AWS para el hosting de nuestra base de datos relacional.

```
#DATABASE SETTINGS
DATABASE_USER=CentinelaUsr
DATABASE_PASSWORD=
DATABASE_NAME=CentinelaProd
DATABASE_HOST=centinelafree.cfsmohvyfeaq.us-east-1.rds.amazonaws.com
DATABASE_PORT=5432
```

Y para la conexión con el servicio de Amazon MQ o Amazon SQS.

```
8
9 #Queuing
10 HOST_RABBITMQ=4fd93SgAMQ.ec2.eastea1.aws.com
11
12
```

Sin embargo, tal como se explicó más arriba, no nos fue posible utilizar el servicio de Mensajería debido a una restricción de la cuenta educativa.

Por tal motivo creamos una instancia de EC2 de una máquina virtual en Ubuntu 18 LTS en la cual instalamos el servicio de RabbitMQ y lo configuramos para nuestra necesidad. Esta máquina virtual actúa como un backing service cloud para dar soporte a nuestro sistema.

Instance summary for i-05d723f17161c9471 (Redis-Ubuntu) Info		
Updated less than a minute ago		
Instance ID	Public IPv4 address	Private IPv4 addresses
i-05d723f17161c9471 (Redis-Ubuntu)	34.226.122.127 open address	172.31.81.255
Instance state	Public IPv4 DNS	Private IPv4 DNS
Running	ec2-34-226-122-127.compute-1.amazonaws.com open address	ip-172-31-81-255.ec2.internal
Instance type	Elastic IP addresses	VPC ID
t2.micro	-	vpc-fa44bb87
IAM Role	Subnet ID	
-	subnet-164ae337	

```
ubuntu@ip-172-31-81-255: ~
ubuntu@ip-172-31-81-255:~$ service redis status
● redis-server.service - Advanced key-value store
   Loaded: loaded (/lib/systemd/system/redis-server.service; enabled; vendor preset: enable)
   Active: active (running) since Wed 2020-10-21 21:22:41 UTC; 21h ago
     Docs: http://redis.io/documentation,
           man:redis-server(1)
  Process: 900 ExecStart=/usr/bin/redis-server /etc/redis/redis.conf (code=exited, status=0/SUCCESS)
 Main PID: 951 (redis-server)
    Tasks: 4 (limit: 1140)
   CGroup: /system.slice/redis-server.service
           └─951 /usr/bin/redis-server 0.0.0.0:6379

Oct 21 21:22:41 ip-172-31-81-255 systemd[1]: Starting Advanced key-value store...
Oct 21 21:22:41 ip-172-31-81-255 systemd[1]: redis-server.service: Can't open PID file /var/run/redis.pid: Permission denied
Oct 21 21:22:41 ip-172-31-81-255 systemd[1]: Started Advanced key-value store.
lines 1-14/14 (END)
```


Monitoreo y Actualizaciones.

Para monitorear el estado del sistema, Elastic Beanstalk hace PINGs constantes hacia la URL del sistema. En este caso, si el contenedor NGINX contesta, el estado de salud del sistema será considerado como saludable.

```
▼ 2020-10-22T16:30:03.723-03:00 nginx | 172.31.42.2 - - [22/Oct/2020...
nginx | 172.31.42.2 - -
[22/Oct/2020:19:29:57 +0000] "GET / HTTP/1.1" 200 20 "-" "ELB-
HealthChecker/2.0" "-"
```

Copy

Para tener un mejor control de todas las partes del sistema, adicionado al monitoreo de Elastic Beanstalk, utilizamos dos tácticas que ya fueron explicadas arriba.

Una es la táctica de por la cual ofrecemos un **Sanity Check** que comprueba el estatus de cada componente del sistema.

Este sanity check se invoca utilizando la táctica de PING hacia el endpoint /monitoring/v1/ping. Previo a dar una respuesta, el sanity check verifica que todos los componentes del sistema estén activos y responde de la siguiente manera:

```
localhost/api/v1/bugs/ping
{
  "status": "OK",
  "data": {
    "postgresBd": "pong",
    "queueStatistics": {
      "connection_closed": 0,
      "channel_closed": 0,
      "consumer_deleted": 0,
      "exchange_deleted": 0,
      "queue_deleted": 0,
      "vhost_deleted": 0,
      "node_node_deleted": 0,
      "channel_consumer_deleted": 0
    }
  }
}
```

Gestión de Logs en Producción

Para la gestión de Logs, configuramos nuestro ambiente para hacer uso del servicio CloudWatch de Amazon, en el cual se envían todos los logs generados por todos los procesos del sistema hacia este repositorio. Allí podemos observar en tiempo real todo lo que sucede en el ambiente desplegado en elastic beanstalk.

Aquí un ejemplo:

▶	2020-10-21T19:45:08.984-03:00	centinela-api		[2020-10-21 22:45:08][info][../controllers/invitation-con...
▶	2020-10-21T19:45:09.734-03:00	nginx		172.31.42.2 - - [21/Oct/2020:22:45:08 +0000] "POST /api/v1/invita...
▶	2020-10-21T20:15:33.772-03:00	nginx		172.31.42.2 - - [21/Oct/2020:23:15:33 +0000] "OPTIONS /api/v1/inv...
▶	2020-10-21T20:15:33.772-03:00	centinela-api		[2020-10-21 23:15:33][debug][../controllers/invitation-co...
▶	2020-10-21T20:15:34.023-03:00	nginx		172.31.42.2 - - [21/Oct/2020:23:15:33 +0000] "GET /api/v1/invitat...
▶	2020-10-21T20:15:50.290-03:00	nginx		172.31.42.2 - - [21/Oct/2020:23:15:50 +0000] "OPTIONS /api/v1/inv...
▶	2020-10-21T20:15:50.290-03:00	centinela-api		[2020-10-21 23:15:50][debug][../services/invitation-servi...
▶	2020-10-21T20:15:50.540-03:00	centinela-api		[2020-10-21 23:15:50][debug][../services/invitation-servi...
▶	2020-10-21T20:15:50.540-03:00	centinela-api		[2020-10-21 23:15:50][debug][../services/invitation-servi...
▼	2020-10-21T20:15:50.540-03:00	centinela-api		[2020-10-21 23:15:50][debug][../services/invitation-servi...
		centinela-api		[2020-10-21 23:15:50][debug][../services/invitation-service.js] >>> This
		invitation		
		{		
		"id": 114,		
		"invitedEmail": "msettimo@adinet.com.uy",		
		"status": "Pending",		
		"role": 2,		
		"userId": 1,		
		"organizationId": 1		
		}		
		is being set as: Confirmed		
▶	2020-10-21T20:15:53.043-03:00	nginx		172.31.42.2 - - [21/Oct/2020:23:15:50 +0000] "POST /api/v1/invita...
▶	2020-10-21T20:18:02.918-03:00	nginx		172.31.42.2 - - [21/Oct/2020:23:18:02 +0000] "OPTIONS /api/v1/inv...

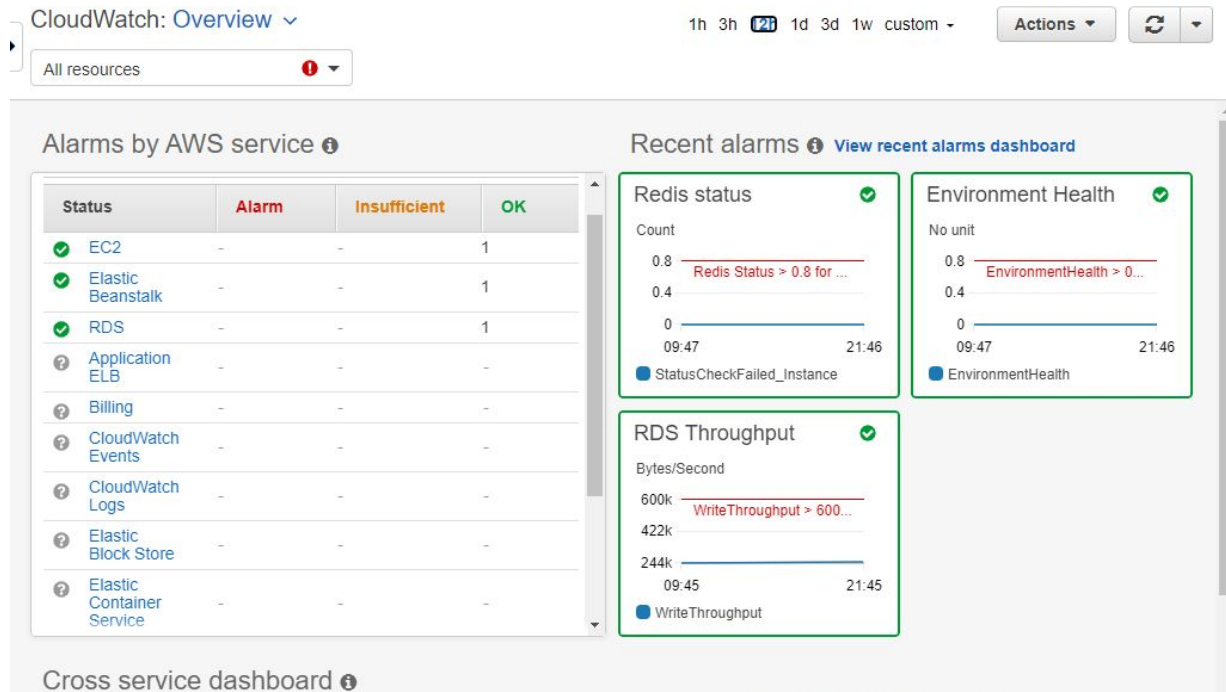
Copy

Según cómo se configuren las variables de entorno referidas al logging en cada microservicio. Podemos además enviar los logs a un Syslog server. Como PaperTrail (que se explicó más arriba en este documento).. Las variables de entorno que se requiere la librería `centinela-logger` (que construimos) permiten definir el comportamiento de los logs.

```
#LOGGING
# Si el LOG_ENVIRONMENT es local, crea una carpeta para los logs en un archivo, y en consola
# Si el LOG_ENVIRONMENT es cloud, envia los logs utilizando syslog a un servicio cloud
# LOG_CLOUD_LOGGING_HOST y LOG_CLOUD_LOGGING_PORT y a consola
# Si el LOG_ENVIRONMENT es both, genera los logs en consola, localmente y en la nube
# Si el LOG_ENVIRONMENT es console, solo genera los logs en consola.
# Log levels ( error | warn | info | verbose | debug )
LOG_LEVEL=info
LOG_ENVIRONMENT=local
LOG_LOCAL_FOLDER=./logs/
LOG_LOCAL_FILE_NAME=logs.log
LOG_LOCAL_EXCEPTION_FILE_NAME=exceptions.log
LOG_CLOUD_LOGGING_HOST=logs.papertrailapp.com
LOG_CLOUD_LOGGING_PORT=39799
```

Cloudwatch además, permite configurar alertas para múltiples métricas en las cuales podemos ser notificados de cambios en el estado del sistema.

En nuestro caso tenemos definidas 3 alertas para el monitoreo y control de la instancia de EC2, de la instancia de Elastic Beanstalk y el estado general de RDS.



Desplegabilidad.

Para la despleabilidad a producción de nuevas versiones de los microservicios y evitar downtime del sistema. El equipo desarrolló las APIS de forma versionada. Todas las apis actualmente comienzan con su versión **/api/v1/**. El despliegue de nuevas versiones de los microservicios agregarían una nueva versión del endpoint pero mantendrían la versión anterior operativa.

Luego, mediante el uso de las herramientas de monitoreo como **NewRelic**, y los **Logs** del sistema se puede observar el uso que tiene cada uno de estos endpoints y así monitorear qué actores siguen consumiendo las versiones anteriores de la API.

En cuanto a la creación de nuevas instancias con versiones actualizadas de los microservicios. Elastic Beanstalk ofrece la herramienta de rollout update, la cual podemos utilizar y así evitar downtime.

En el caso que existan múltiples instancias del microservicio en funcionamiento, Elastic Beanstalk permite hacer rollout updates: EBs detiene una porción (configurable) de instancias de la aplicación, mientras mantiene las otras instancias operativas atendiendo las solicitudes de los clientes.

Luego lanza las versiones actualizadas de las instancias con la nueva configuración. Si estas nuevas instancias se mantienen activas, EBs continua con otra porción de instancias para actualizar.

Este proceso se repite las cantidad de veces que sea requerida para actualizar todas las instancias del ambiente.

Si una de las porciones en proceso de actualización no pasa a tener un estado saludable durante un tiempo de espera (el valor predeterminado es de 30 minutos), el resto del proceso de actualización se cancela.

A su vez, se realiza el rollout inverso para volver a la versión anterior que se encuentra en funcionamiento.

La configuración de rollout update de EBs determina las propiedades de cómo será la porción de instancias que se actualizarán a la vez y cuál será el criterio para considerar una actualización exitosa. Como también cuantas instancias queremos que siempre estén disponibles para atender a los clientes.

Configuration updates

Changes to virtual machine settings and VPC configuration trigger rolling updates to replace the instances in your environment without downtime.
[Learn more](#)

Rolling update type

Rolling based on Health

Batch size

1

The maximum number of instances to replace in each phase of the update.

Minimum capacity

1

The minimum number of instances to keep in service at all times.

Pause time

hh:mm:ss

Pause the update for up to an hour between each batch.

6. Resultados y conclusiones

Pasamos ahora a comunicar los resultados que hemos obtenido, y como hemos trabajado para llegar a los mismos.

6.1. Proceso de trabajo y requerimientos funcionales

En cuanto al proceso de trabajo, el equipo decidió tomar a Scrum como metodología de trabajo. Se dividió el trabajo en 2 Sprints, en los cuales progresivamente se fue mejorando la dinámica y los resultados, como se puede ver en el apartado sprint realizados.

El resultado estado final del trabajo, con sus tareas pendientes, se encuentra en el último sprint:

<https://tree.taiga.io/project/zebasetas-bug-tracking/backlog>

De acuerdo a los resultados, se entiende que se llegó prácticamente al 96% de los requerimientos funcionales solicitados, tanto de la primera como de la segunda entrega, quedando un remanente de algunas mejoras menores que quedarán pendientes para futuros sprints (puede verse último sprint <https://tree.taiga.io/project/zebasetas-bug-tracking/taskboard/sprint-5-3052?kanban-status=1922560>).



Se realizaron también test similares a los tests de escenarios, para verificar el cumplimiento de los requerimientos: [7.5. Test basados en escenarios](#).

Algunas actividades del proceso que nos fueron de gran ayuda:

1. Se trabajó de utilizando Gitflow, se intentó tener una nomenclatura de branches que se puede ver en nuestra wiki (que aún resta pulir):

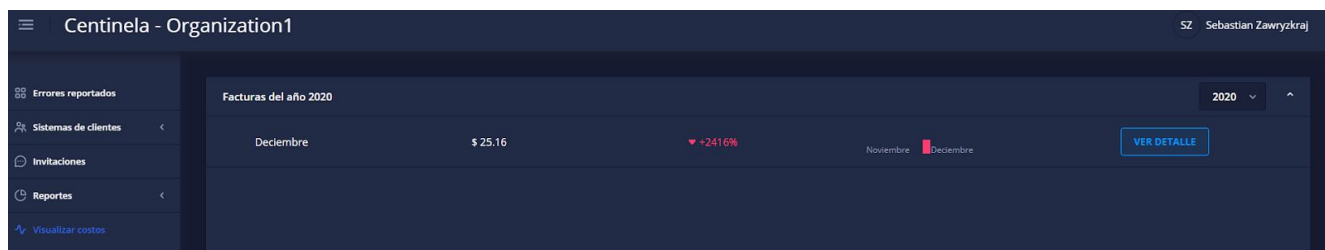
<https://tree.taiga.io/project/zebasetas-bug-tracking/wiki/2-propuesta-de-gestion-de-repositorios>

2. Se realizaron inspecciones de código cruzadas.
3. Se estableció que una tarea no estaba terminada hasta tener el test de JMeter, y a la vez que un compañero no le diera el visto bueno.
4. Se realizaron tests de integración luego de cada cierre de Sprint
5. Se intentó acordar estas pautas en el definition of done

<https://tree.taiga.io/project/zebasetas-bug-tracking/wiki/1-definition-of-done>

6.2. Sobre los nuevos requerimientos funcionales

- a. Requerimientos de la primer entrega: **RF1, RF2, RF3, RF4, RF5, RF6, RF7, RF8, RF9, RF10, RF11**: Se pasan a cumplir de la misma forma que en la entrega anterior. Para la misma se generó una separación del dominio en varios microservicios, dónde en algunos casos tienen que combinarse para enviar las respuestas al usuario, o es necesario el cacheo de datos en el front para la mejora de performance (buscando no hacer varios requests para poder mostrar los datos en pantalla).
- b. Requerimientos sobre segunda entrega:
 - I. **RF12**: se generó un microservicio exclusivo (CostExplorer) que atiende este requerimiento, y un link específico en el menú del usuario desarrollador para poder acceder al mismo.



Puede realizarse tanto la visualización anual de todos los costos resumidos, o ingresar a ver detalle y hasta descargar un reporte basico en pdf.

Factura Diciembre de 2020

Item	Cantidad	Unidad	Total
Errores reportados	16	U\$s 0.01	U\$s 0.16
Usuarios activos	5	U\$s 5	U\$s 25
Costo final			U\$s 25.16

DESCARGAR

December_2020.pdf 1 / 1

Centinela - Tu Bugtracker favorito!

Factura Diciembre de 2020

Item	Cantidad	Unidad	Total
Errores reportados	23	U\$s0.01	U\$s0.23
Usuarios activos	0	U\$s5	U\$s0
	Total		U\$s0.23

- II. **RF13:** se generó un sdk que puede descargarse de la siguiente url:
<https://www.npmjs.com/package/centinela-sdk>.
 Existe un repositorio exclusivo para el mismo:
<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-bugreporter-node>,
 donde también puede descargarse un proyecto para probarlo:
<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Main/tree/master/SDKTestApp>
- III. **RF14:** se generó un un microservicio exclusivo (Notifications) que presenta un conjunto de crones que atienden la configuración de los usuarios. Esta configuración puede accederse desde la opción de configuración en el frontend.



Donde se accede a una pantalla en la cual figura tanto la posibilidad de configurar distintas opciones para cada tipo de severidad.

ALERTAS POR SEVERIDAD

OTRAS ALERTAS

Severidad 1

☐ No enviar correo

☒ Envío inmediato

☐ Envío a hora definida por usuario

Hora: 01 Minutos: 30

Severidad 2

☐ No enviar correo

☐ Envío inmediato

☒ Envío a hora definida por usuario

Hora: 03 Minutos: 30

Severidad 3

☐ No enviar correo

☐ Envío inmediato

☒ Envío a hora definida por usuario

Hora: 03 Minutos: 30

Severidad 4

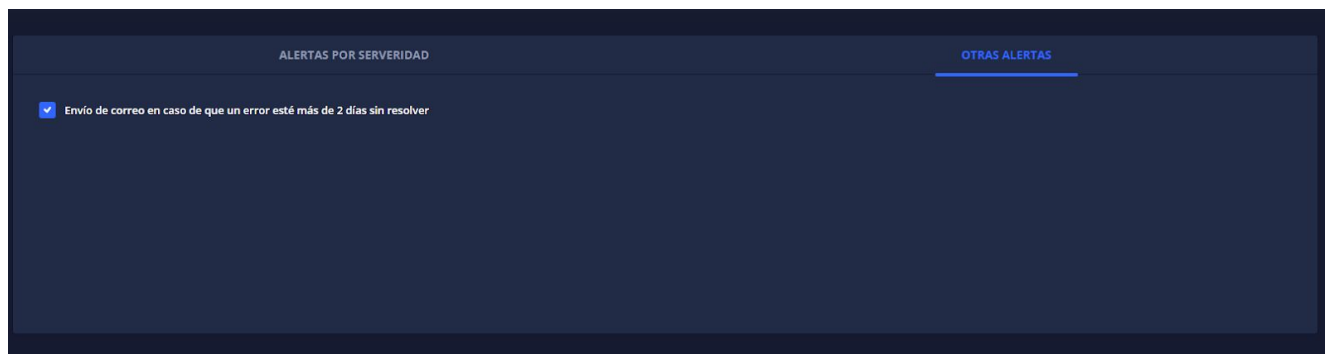
☐ No enviar correo

☐ Envío inmediato

☒ Envío a hora definida por usuario

Hora: 10 Minutos: 00


Como también puede marcar para que le lleguen notificaciones por errores asignados con más de 2 días sin resolver.



Por defecto todas las alertas se encuentran apagadas.

Luego, el sistema se encarga del envío de los correos correspondientes, de acuerdo a la configuración.

Cuando se solicita que se envíen correos de severidad 4 a determinada hora (sólo envía los de las últimas 24 horas:

 **Centinela App** <centinelaappnotification@gmail.com>
para mí ▼

Reporte diario de bugs asignados con severidad 4

Id	Título	Descripción	Link
238	Steuber, Rau and Osinski...	Non et veritatis quia ut. Debitis fugiat assumenda autem similique exercitationem nisi quos ut. Ex cum nulla.	Ir
239	Botsford - Lubowitz	Minus quis sapiente. Quia illo eos maxime cumque ratione excepturi et. Quo velit vel quis reiciendis ab fugit iure. Ipsa incidunt ad nihil dolore cupiditate.	Ir

Cuando se solicita que se envíen correos sobre bugs asignados con 2 días sin resolver.

Reporte diario de bugs que tienes asignados con más de 2 días sin resolver

Id	Título	Descripción	Link
49	Murphy, Jones and Abbott...	Delectus quae autem. Quidem voluptas beatae expedita qui veritatis. Distinctio nesciunt consequatur molestias quo hic qui. Repellendus est placeat asperiores enim corrupti quis. Quam cum harum amet.	Ir

Correo generado automáticamente por Centinela - Tu Bug Tracker favorito



Se te ha asignado un Bug

Título: Graham, Moen and Zboncak...

Descripción: Consequuntur porro omnis dolorem error. Dolorum quis illum ad numquam numquam reiciendis ea iure aut.

Link [lr](#)



Cuando se solicita que se envíe un bug con severidad X inmediatamente.

En todos los correos se envía el link, con el cuál usuario podrá acceder a la aplicación y visualizar el error.

- IV. **RF15:** se generó un microservicio específico (Reportes) para tal fin. No solo pueden visualizarse los reportes de la entrega anterior, si no los nuevos.

A screenshot of a web application dashboard titled 'Centinela - Organization1'. The top right corner shows the user 'SZ Sebastian Zawrykraj'. The left sidebar contains a menu with items: 'Errores reportados', 'Sistemas de clientes', 'Invitaciones', 'Reportes' (highlighted), 'Estadísticas de bugs', 'Top 10 desarrolladores', and 'Bugs con 2 días sin asignar'. The main content area displays a table titled 'Top 10 desarrolladores que resolvieron más bugs asignados en los últimos 30 días'. The table has three columns: a developer ID, name, email, and a count. The first row shows 'SZ Sebastian Zawrykraj' with email 'sebastianzawrykraj@gmail.com' and a count of '3'.

Top 10 desarrolladores que resolvieron más bugs asignados en los últimos 30 días			
SZ	Sebastian Zawrykraj	sebastianzawrykraj@gmail.com	3

Obs: puede acceder a un manual de usuario para visualizar los flujos de uso, en un documento (Manual_de_usuario_v2.pdf) adjunto a este documento.

6.3. Requerimientos no funcionales

En cuanto a los requerimientos no funcionales se detalla su cumplimiento

1. Se cumplió con los requerimientos de performance (**RNF1**) documentados en el apartado [performance](#).
2. Se cumplió con el requerimiento de confiabilidad y disponibilidad (**RNF2**), como se indica en el apartado de [confiabilidad](#).

3. Se cumple con la configuración y manejo de secretos (**RNF3**), como se indica en el apartado [configuración y manejo de secretos](#).
4. Se cumple con el requerimiento de autenticación (**RNF4**), como se indica [autenticación y autorización](#).
5. Se cumple con el apartado de seguridad (**RNF5**), indicado en [seguridad](#)
6. Se cumple con el apartado de código fuente (**RNF6**). Se deja un archivo Readme en el repositorio, indicando cómo configurar un ambiente de desarrollo. También se puede acceder a esa información en nuestra wiki <https://tree.taiga.io/project/zebasetas-bug-tracking/wiki/3-configuracion-y-ambientes>
También se puede ver el apartado rest [api rest](#), donde puede ver cómo generamos nuestra api (o visitar directamente el cliente swagger configurado (<http://centinela-env.eba-zhfdmg23.us-east-1.elasticbeanstalk.com/api/v1/docs/>))
7. Se cumple con el requerimiento de identificación de fallas (**RNF8**), como se indica en [identificación de fallas](#)
8. Se cumplió en el requerimiento (**RNF9**) de estilo arquitectónico solicitado, realizando un microservicio por concepto de dominio como se indica más arriba. Todos los microservicios son codificados en lenguaje NodeJs, con excepción del microservicio CostExplorer, el cual fue codificado en C# con el framework .NET Core.
9. Se cumple parcialmente con el requerimiento de integración continua (**RNF10**) dado que se configura y se realizan tests automáticos para todos los microservicios, mediante GitHub Actions, pero no se tiene un cubrimiento completo de todos los casos de prueba solicitados.

The screenshot shows a GitHub repository page for 'ArqSoftPractica / Larrosa-Settimo-Zawrzykraj-Bugs'. The 'Actions' tab is selected, showing a workflow run for 'Merge pull request #2 from ArqSoftPractica/develop Develop Node.js CI #6'. The workflow is titled 'build (12.x)' and succeeded yesterday in 15s. A list of jobs is shown on the left, including 'build (12.x)' and 'build (14.x)'. The main panel displays the details of the 'build (12.x)' job, showing a list of steps: 'Set up job', 'Run actions/checkout@v2', 'Use Node.js 12.x', 'Run npm ci', 'Run npm run build --if-present', 'Run npm test', 'Post Run actions/checkout@v2', and 'Complete job'. Each step has a duration listed on the right.

El script de integración continua de GitHub Actions que posee cada repositorio de los microservicios se ejecuta con cada pull request a master o con cada push a master.

```

4   name: Node.js CI
5
6   on:
7     push:
8       branches: [ master ]
9     pull_request:
10      branches: [ master ]
11
12  jobs:
13    build:
14
15      runs-on: ubuntu-latest
16
17      strategy:
18        matrix:
19          node-version: [12.x, 14.x]
20
21      steps:
22      - uses: actions/checkout@v2
23      - name: Use Node.js ${ matrix.node-version }
24        uses: actions/setup-node@v1
25        with:
26          node-version: ${ matrix.node-version }
27      - run: npm ci
28      - run: npm run build --if-present
29      - run: npm test

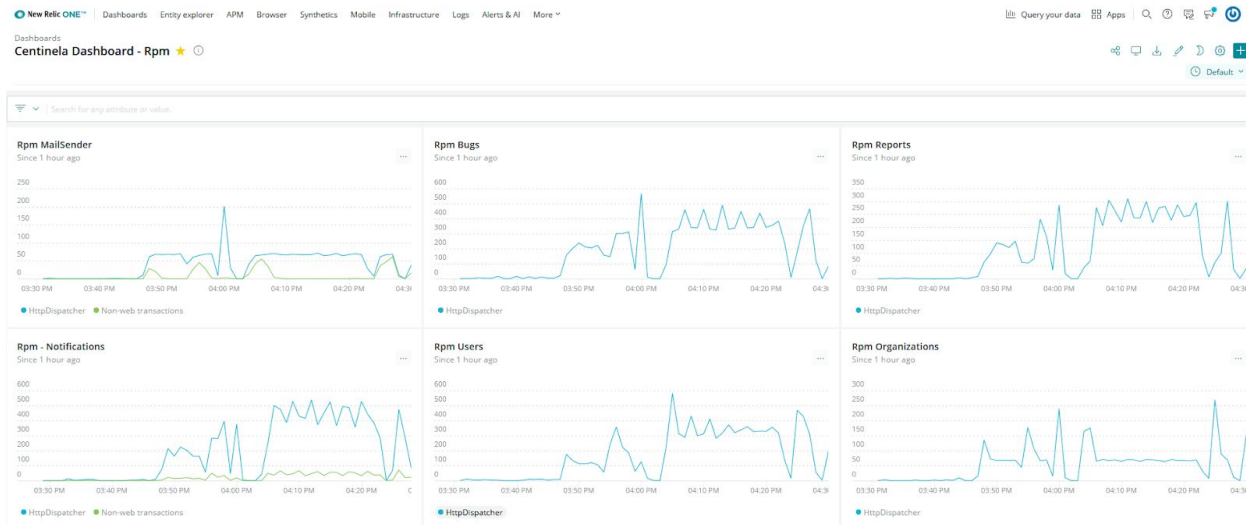
```

10. Se cumple con el requerimiento de identificación de fallas (**RFN11**), los logs quedan tanto centralizados como retenidos por más de 7 días en la página <https://papertrailapp.com>.

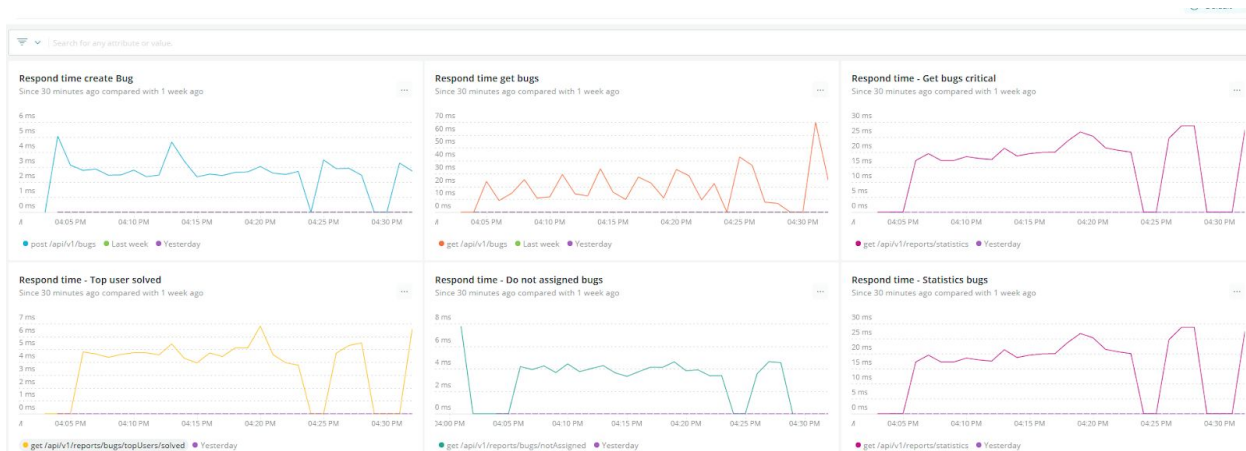


11. Con respecto a la Desplegabilidad (**RNF12**), se generó un contenedor dockerizado que permite su despliegue independiente en AWS. Basta con desplegar este nuevo servicio y configurar el ngynx para que el tráfico se dirccione al mismo, como se detalla más arriba.
12. En referencia a la Monitoreabilidad (**RNF13**), se generó un monitoreo desde New Relic para todos los microservicios, menos el de CostExplorer. Donde cada uno tiene una entrada de datos independientes para su análisis. También se generaron dashboards que tienen en cuenta las métricas específicas que se solicitaron.

Dashboard de requets por minuto:



Dashboard de tiempos de respuesta:



6.4. Mejoras y faltantes sobre los requerimientos solicitados

Dentro de las mejoras a realizar, los apartados que nos quedaron pendientes más relevantes:

1. Mejoras en los mensajes de error: el equipo dejó como pendiente mejorar los distintos mensajes que entrega la aplicación en los casos de posibles errores que puedan darse en la aplicación. Por ejemplo, para validaciones no cumplidas o requests malformados, los mensajes no son 100% informativos a efectos de que sea más fácil para el frontend tomar algunas decisiones.
2. Monitoreo del CostExplorer: se realiza un monitoreo con New Relic en todos los microservicios menos en este.
3. Dado que no se pudo acceder a una cuenta de AWS educativa, se realiza un deploy local con Docker Compose. Nos quedan pendientes algunas mejoras para poder trabajar en el deploy de los microservicios. Para futuras versiones intentaremos aplicar tareas de codedeploy ofrecidos por AWS para desplegar los servicios de forma totalmente independiente.
4. Si bien la codificación del sistema fue totalmente en inglés, al ser un servicio pensado para utilizar en Uruguay, el frontend debería mostrar su interfaz de login y registro en idioma español. El resto del frontend está desarrollado en español. Idealmente se podría construir agnóstico al idioma o al menos en los dos idiomas más comunes de nuestros clientes.
5. Monitoreo central: si bien no era un requerimiento, el equipo se había planteado tener un endpoint independiente, provisto por un microservicio separado, que comunique el estado del sistema en general, como se había realizado para la primera entrega. De todos modos, para la presente entrega se cumple el healthcheck en cada microservicio, el cual comunica el estado de la base de datos y de la cola de mensajería.
6. Fecha del error: queda pendiente mostrar la fecha de generación del error en el frontend.
7. Tests automatizados: el equipo entiende que cumple parcialmente con este requerimiento, dado que no se llegó a un cubrimiento completo de los casos de uso solicitados, pero sí se prepararon todos los repositorios para el trabajo con Integración Continua mediante GitHub Actions.

6.5. Oportunidades de mejoras detectadas sobre el diseño

1. **Reintento exponencial de trabajos:** para la primera entrega utilizamos bull y redis como sistema de procesamiento de trabajos, esto nos permitía configurar reintentos exponenciales. Es decir que si un trabajo fallaba, el trabajo se volvía a encolar como trabajo a reintentar y este se reintentaba luego de un cierto tiempo de espera. Si el trabajo volvía a fallar, se volvía a reintentar pero ahora con un tiempo más extenso, calculado de forma exponencial. Esta situación podría ocurrir en 3 oportunidades, luego de lo cual el trabajo era marcado como fallido y descartado.

En esta entrega, cambiamos la tecnología y utilizamos rabbitmq. Si bien es posible conseguir el mismo comportamiento descrito arriba, no fue codificado de esa forma. Si bien **SI** implementamos una política de reintentos, estos reintentos son inmediatos. Si una tarea falla, esta es reintentada 4 veces y luego de la 4ta falla se descarta. La no implementación de esta funcionalidad fue debido a la complejidad requerida para llevarla a cabo y la necesidad del equipo de enfocarse en otras tareas.

Si bien es una vez implementado, es un cambio fácilmente aplicable ya que el manejo de cola

de mensajería está sumamente desacoplado y sería solamente que la nueva versión respete la interfaz existente.

La idea concepto de esta mejora se basaría en este post :

<https://www.alphasights.com/news/exponential-backoff-with-rabbitmq?locale=en>

2. Uso de base de datos no relacionales para los reportes:
Si bien cumplimos plenamente con los requerimientos de performance que tenía el proyecto. El equipo hubiera deseado contar con el tiempo adecuado para mejorar la versión del microservicio de reportes para que utilice una base de datos no relacional. El equipo entiende para un uso excesivo del sistema de reportes. Una base de datos no relacional del tipo documental como MongoDB puede ser más performante en estas situaciones.
3. Uso de cache centralizado para tokens.
Hoy cada microservicio mantiene un caché local de los tokens que fueron validados contra el microservicio de usuarios. Se podría aumentar la performance y a su vez reducir la carga de trabajo de verificar los tokens de usuario por parte del microservicio de Usuarios si todos los microservicios compartieran un “shared memory cache” como REDIS. De esta forma cualquier microservicio podría consultar el caché por tokens previamente validados y no tener que verificarlo nuevamente contra el endpoint de verificación que expone Usuarios.
4. Uso de técnicas de respuestas dinámicas dependiendo del pedido del cliente. La implementación de GraphQL en nuestra API Gateway que centralice y “arme” dinámicamente las respuestas que solicita el cliente que hoy en día se realizan con más de un request. A su vez aprovechar para hacer allí caching de datos que son solicitados con mayor frecuencia.

7. Anexos

7.1. Requerimientos y restricciones

7.1.1. Requerimientos Funcionales

ID Requerimiento	Descripción	Actor
RF1 - Registro de usuario administrador	Los usuarios administradores podrán registrarse vía web como administrador de su organización. Para ello, deberá ingresar nombre, email, nombre de organización (único en el sistema) y contraseña. Al registrarse de esa manera, el administrador habrá creado una organización nueva. Luego, podrá invitar otros usuarios administrador y usuarios desarrolladores (RF2).	<ul style="list-style-type: none">• Usuario Administrador
RF2 - Registro de usuarios mediante invitación	Un usuario administrador deberá poder invitar a otros usuarios administradores y desarrolladores a la plataforma, enviándole un correo electrónico que contenga un link al registro anterior (RF1), quitando el campo de “nombre de organización” y con el agregado de un mensaje que indique a qué organización se estaría uniendo.	<ul style="list-style-type: none">• Usuario Administrador• Usuario Desarrollador
RF3 - Autenticación de usuario	Dado un usuario no autenticado cuando está presente en la página principal del sistema y hace click en “Ingresar” entonces se despliega un formulario de ingreso que solicita su correo electrónico y una contraseña de acceso. Una vez autenticado el usuario el sistema debe redireccionarlo a la pantalla de listado de errores.	<ul style="list-style-type: none">• Usuario Administrador• Usuario Desarrollador
RF4 - Gestión de clave de aplicación	El sistema debe permitir a los usuarios administradores crear claves de acceso de aplicación. Estas claves únicas de acceso son las que el cliente debe utilizar para invocar los servicios REST provistos por el sistema. Ver RNF4. De las claves de acceso simplemente se solicita un nombre, o “entorno” (staging, production) , para identificarlas.	<ul style="list-style-type: none">• Usuario Administrador
RF5 - Gestión de errores	El sistema debe permitir a los usuarios administradores editar errores mediante un formulario en la web. De los mismos se puede editar el título, descripción (opcional), severidad (opcional, número entre 1 y 4) y desarrollador asignado (opcional). Los errores son creados mediante el	<ul style="list-style-type: none">• Usuario Administrador

	endpoint REST mencionado en RF9.	
RF6 - Listado de errores	<p>El sistema debe permitir a todos los usuarios ver un listado web de los errores que existen para su organización. La vista por defecto deben ser los errores no resueltos , ordenados por más crítico primero (es decir, severidad 1 antes que severidad 2, etc.). También se debe poder tener la opción de mostrar errores ya resueltos.</p> <p>En el listado se tiene que poder ver el nombre, severidad, estado (resuelto o no resuelto) y desarrollador asignado.</p>	<ul style="list-style-type: none"> • Usuario Administrador • Usuario Desarrollador
RF7 - Detalles de error	El sistema debe permitir a todos los usuarios clicar en un error del listado (RF6), y esto debe dirigirlos a una vista en la que se muestra la información del error (título, descripción, severidad, estado, desarrollador asignado), y el botón que se especifica en RF8.	<ul style="list-style-type: none"> • Usuario Administrador • Usuario Desarrollador
RF8 - Resolución de error	El sistema debe permitir a todos los usuarios clicar un botón en la vista de RF7 que marque el error como resuelto.	<ul style="list-style-type: none"> • Usuario Administrador • Usuario Desarrollador
RF9 - Creación de error (REST)	<p>Se deberá disponibilizar un endpoint REST que permita crear errores. Este endpoint será utilizado en los sistemas de los clientes de nuestro sistema, para que cuando ocurre una excepción en su código, se reporte automáticamente en nuestro producto. Los atributos que se pueden enviar para crear el error, son los mismos que se mencionan en RF5 , excepto “desarrollador asignado”.</p> <p>Importante: Cuando un error es creado mediante este endpoint, se debe enviar un email a todos los miembros de la organización con información básica del mismo, para que estén al tanto de las fallas que ocurren en sus sistemas.</p>	<ul style="list-style-type: none"> • Sistema Externo
RF10 - Errores críticos (REST)	<p>Se deberá disponibilizar un endpoint REST que permita obtener los 5 errores no resueltos con mayor severidad para la organización. Este endpoint será integrado y utilizado por sistemas de los clientes, como dashboards de monitoreo y en canales de comunicación, como por ejemplo Slack.</p> <p>Este endpoint debe responder en hasta 200 ms para cargas de hasta 1200 req/m ya que es crítico obtener con rapidez los errores más graves.</p>	<ul style="list-style-type: none"> • Sistema Externo

RF11 - Reporte de estadísticas de errores	El sistema debe permitir a los usuarios administradores obtener un reporte web que muestre cuántos errores ocurrieron en un rango de fechas dado, y cuántos de ellos fueron resueltos. También debe mostrar cuántos errores hubo por cada tipo de severidad.	<ul style="list-style-type: none"> • Usuario Administrador
RF12 - Visualizar consumo	Los usuarios administradores deben poder acceder a una página en la que pueden ver la factura del mes actual, o de meses anteriores. La facturación de Centinela, al ser un SaaS, se basa en un modelo de cobros mensuales y por uso. Se cobrará de la siguiente manera: <ul style="list-style-type: none"> • 0.01 USD por error reportado • 5 USD por usuario de la organización por mes No es necesario que el sistema realice ningún cobro. Se pide simplemente mostrar el monto que sería facturado para el mes corriente, detallando el cálculo.	<ul style="list-style-type: none"> • Usuario Administrador
RF13 - SDK para creación de error	Se deberá disponibilizar un SDK (lenguaje a elección) que permita hacer uso de la operación de crear error (RF9 del anterior obligatorio). En caso de que no se pueda conectar con el servicio de Centinela o el tiempo de respuesta supere los 500 ms el SDK debe dar timeout y retornar error.	<ul style="list-style-type: none"> • Sistema Externo
RF14 - Recepción de alertas por correo	Los usuarios desarrolladores deben poder configurar sus preferencias en cuanto a qué correos electrónicos recibir. El usuario debe poder activar correos electrónicos para los siguientes eventos: <ul style="list-style-type: none"> • Al asignarse un nuevo error: <ul style="list-style-type: none"> ○ El usuario debe seleccionar si quiere recibirlo inmediatamente. ○ Optar por recibir una alerta en cierto momento del día (al horario que indique) con la cantidad total de errores que ocurrieron durante ese día. Se debe tener en cuenta que tanto el inicio del día como hora de envío de alerta deben pertenecer al mismo huso horario que el usuario desarrollador. En ambos casos lo hace filtrando por severidad para decidir si recibe las alertas de errores o no. <ul style="list-style-type: none"> • Cuando tiene errores asignados de más de 2 días sin resolver, indistintamente su severidad. Por defecto, las alertas por correo deberán estar apagadas.	<ul style="list-style-type: none"> • Usuario Administrador • Usuario Desarrollador
RF15 -	Además del reporte existente de la primera iteración se	<ul style="list-style-type: none"> • Usuario

Reporte de asignación de errores	<p>pide:</p> <ul style="list-style-type: none"> • Reporte de top 10 desarrolladores que resolvieron más incidencias asignadas en los últimos 30 días. • Reporte de incidencias que llevan más de 2 días sin estar asignadas 	Administrador
----------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------

7.1.2. Requerimientos No Funcionales

ID Requerimiento	Descripción	Atributo de calidad
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 350 ms. para cargas de hasta 1200 req/m.	Performance
RNF2 - Confiabilidad y disponibilidad	Con el fin de poder monitorear la salud y disponibilidad del sistema, se deberá proveer un endpoint HTTP de acceso público que informe el correcto funcionamiento del sistema (conectividad con bases de datos, colas de mensajes, disponibilidad para recibir requests, etc.).	Disponibilidad
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.	Modificabilidad
RNF4 - Autenticación y autorización	Se deberá establecer un control de acceso basado en roles, distinguiendo entre usuarios administradores y usuarios desarrolladores. Los permisos otorgados a los mismos deberán restringir el acceso a sus correspondientes funcionalidades, prohibiendo la interacción con cualquier otra operación pública del sistema. Los usuarios de una organización no	Seguridad

	<p>pueden bajo ningún concepto poder acceder a datos de otra organización.</p> <p>Las claves de aplicación deben ser generadas como JSON Web Tokens y el sistema debe validar que el token sea válido y tenga los permisos necesarios para acceder a los endpoints especificados en RF9 y RF10.</p>	
RNF5 - Seguridad	<p>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.</p> <p>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.</p>	Seguridad
RNF6 - Código fuente	<p>El sistema deberá ser desarrollado con un lenguaje, framework y/o librerías elegidas por el equipo, que sean tecnologías web que permitan cumplir con los requerimientos que necesitan una interfaz web, como también los que necesitan ser endpoints REST.</p> <p>Por otra parte, el control de versiones del código se deberá llevar a cabo con repositorios Git debidamente documentados, que contienen 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.</p>	Modificabilidad
RNF7 - Pruebas	<p>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</p>	Testeabilidad

	<p>usuarios concurrentes. De necesitar incluir claves o secretos, el script deberá recibir dichas configuraciones por variables de entorno.</p> <p>Además de las pruebas de carga, se deberá contar con pruebas funcionales automatizadas para el RF9 y RF10.</p>	
RNF8 - 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 período mínimo de 24 horas.	Disponibilidad
RNF9 - Estilo de arquitectura	<p>Se deberá llevar el estilo de arquitectura hacia uno basado en microservicios, especificando fronteras bien definidas entre los mismos, así como las interfaces mediante las cuales se comunicarán.</p> <p>Cada uno de estos servicios deberán verse desarrollados y desplegados en aplicaciones diferentes, comunicándose exclusivamente mediante colas de mensaje o llamadas HTTP en tiempo de ejecución. Al menos uno de los microservicios debe estar en un lenguaje de programación diferente a los demás.</p>	Interoperabilidad
RNF10 - Integración continua	<p>Además de las pruebas de carga, se deberá contar con pruebas automatizadas para al menos tres requerimientos funcionales de su elección en al menos uno de los repositorios.</p> <p>El o los repositorios involucrados deben correr las pruebas automatizadas del código cuando un nuevo commit se integra a la rama principal.</p>	Testeabilidad
RNF11 - Identificación de fallas	Para facilitar la detección e identificación de fallas, se deberán centralizar y retener los logs emitidos por todo el sistema en producción por un período mínimo de 24 horas.	Disponibilidad
RNF12 - Desplegabilidad	Se deberá poder desplegar una nueva versión de cualquier microservicio, sin ocasionar <i>downtime</i> .	Disponibilidad

RNF13 - Monitoriabilidad	Para facilitar el diagnóstico ante eventuales fallas, se deberá monitorear las siguientes métricas de los distintos servicios: <ul style="list-style-type: none"> • Peticiones por minuto • Tiempos de respuesta de distintos endpoints 	Disponibilidad
-----------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------

7.1.3. Restricciones

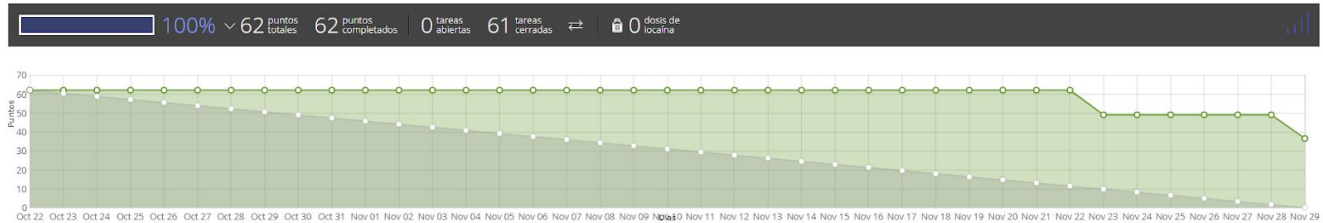
ID Reestricción	Descripción
RES1 - Arquitectura	El estilo arquitectónico del sistema debería de estar basado en microservicios.
RES2 - Servicios	Los servicios expuestos deberán ser Rest
RES3 - Repositorio	El código fuente, documentación y artefactos deberán ser gestionados en Git.
RES4 - Pruebas	Deberemos utilizar la herramienta Apache JMeter para realizar las pruebas de carga.
RES5 - Flujo de trabajo	Para el manejo de las ramas en el repositorio se deberá utilizar GitFlow.
RES6 - Lenguaje de codificación	Al menos uno de los microservicios deberá de estar desarrollado en un lenguaje de programación diferente a los demás.

7.2. Sprints realizados

7.2.1. Sprint 4

Link de taiga: <https://tree.taiga.io/project/zebasetas-bug-tracking/taskboard/sprint-4-4693>

SPRINT 4 CENTINELA 22 OCT. 2020-29 NOV. 2020

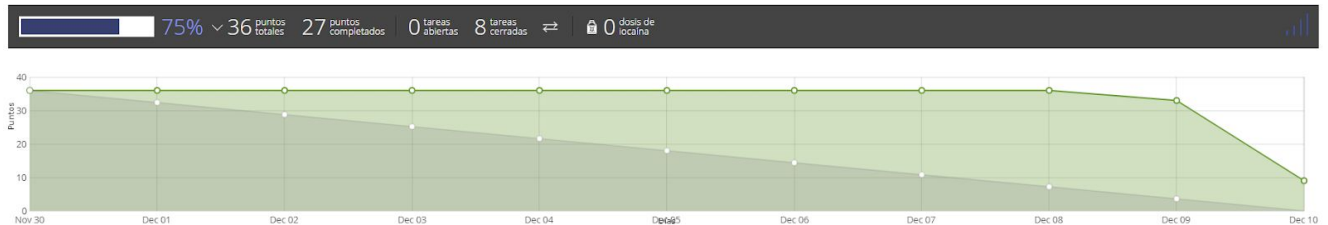


7.2.2. Sprint 5

Link de taiga: <https://tree.taiga.io/project/zebasetas-bug-tracking/taskboard/sprint-5-3052>

Resultado:

SPRINT 5 CENTINELA 30 NOV. 2020-10 DIC. 2020



7.3. Api rest

Users		▼
POST	/users	Create new Organization with Admin User
GET	/users	Get Users In Org
POST	/login	Login
POST	/invitations	Create Invite
GET	/invitations/1	Get Invitation Details
POST	/invitations/1	Accept Invitation
Organization		▼
POST	/organizations	Create Organization
GET	/organizations	Get Organizations
GET	/organizations/12	Get Organization by ID
POST	/systems	Create System
GET	/systems	Get Systems
POST	/systems/2/environments	Create Environment
GET	/systems/2/environments	Get Environments
Bugs		▼
POST	/bugs	Create Bug
GET	/bugs	Get Bugs
PUT	/bugs/2	Update Bug

Notifications		▼
GET	/notifications/ping	Ping
POST	/preferences	Add/Update Preference
GET	/preferences	Get Preferences
POST	/preferences/general	Add/Update General Preference
GET	/preferences/general	Get General Preferences
Reports		▼
GET	/reports/statistics	Statistics Report
GET	/reports/critical	Critical
GET	/reports/bugs/topUsers/solved	Solved
GET	/reports/bugs/notAssigned	Not Assigned
CostsExplorer		▼
GET	/costs	Get Costs

7.4. URLs de la aplicación

URL de Swagger con la descripción general de la API:

http://{{production_url}}/api/v1/docs/

URL del sistema de gestión del proyecto:

<https://tree.taiga.io/project/zebasetas-bug-tracking/backlog>

URL equipo donde se encuentran todos los repositorios de GitHub:

<https://github.com/orgs/ArqSoftPractica/teams/larrosa-settimo-zawrzykraj/repositories>

Repositorio BugReporter

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-bugreporter-node>

Repositorio Bugs

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Bugs>

Repositorio Chassis

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Chassis>

Repositorio CostsExplorer

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-CostExplorer>

Repositorio Frontend

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Frontend>

Repositorio Logger

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Logger>

Repositorio MailSender

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-MailSender>

Repositorio Main

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Main>

Repositorio Notifications

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Notifications>

Repositorio Organizations

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Organizations>

Repositorio Reports

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Reports>

Repositorio Reports

<https://github.com/ArqSoftPractica/Larrosa-Settimo-Zawrzykraj-Users>

7.5. Test basados en escenarios

Caso de uso	Registro de usuario administrador		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras
Registro de usuario admin con correo que ya existe	Mensaje de que ya existe el usuario	Ok	No se crear la organización, pero no se notifica al usuario del error
Registro de usuario admin con nombre de organizacion que ya existe	Mensaje de que ya existe el organización	Ok	No se crear la organización, pero no se notifica al usuario del error
Registro de usuario admin con un mail en formato inválido	Mensaje mail inválido	Ok	

Registro de usuario admin con password con menos de 4 caracteres	Mensaje password inválido	Ok	
Registro de un usuario sin todos los campos completados	Mensaje de error	Ok	
Registro de usuario con organización correcta	Ir al login	Ok	
Caso de uso	Login		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras
Login con correo o password incorrecto	Mensaje de credenciales inválidas	Ok	
Login de usuario admin correcto	Dirigirse a pantalla de bugs, donde se ven los bugs de toda su organización. Se puede acceder a un menú con todas las opciones: Errores reportados, Sistemas, Invitaciones, Reportes, Visualizar Costos, Preferencias de correo (en la configuración de usuario)	OK	
Login de usuario desarrollador correcto	Dirigirse a pantalla de bugs, donde se ven los bugs de toda su organización. Solo puede acceder al menú de Errores reportados y Preferencias de correo (en la configuración de usuario).	Ok	
Caso de uso	Invitación de usuario		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras

Invitación a nuevo usuario administrador con correo repetido, o de otra organización	Mensaje de correo repetido o que está en otra organización	OK	No se avisa que es un correo que ya existe
Invitación a nuevo usuario desarrollador con correo repetido, o de otra organización	Mensaje de correo repetido o que está en otra organización	OK	No se avisa que es un correo que ya existe
Invitación a nuevo usuario administrador	Le debe llegar un correo al usuario invitado con un link para que pueda aceptar la invitación y registrarse	OK	
Invitación a nuevo usuario desarrollador	Le debe llegar un correo al usuario invitado con un link para que pueda aceptar la invitación y registrarse	OK	
Caso de uso	Aceptar invitación		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras
Aceptar invitación de un usuario admin	Recepción de correo con link. Acceso al registro con el link y posterior reenvío a la pantalla de login una vez acabado el proceso	Ok	
Aceptar invitación de un usuario desarrollador	Recepción de correo con link. Acceso al registro con el link y posterior reenvío a la pantalla de login una vez acabado el proceso	OK	
Caso de uso	Registro de sistema		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras
Registro de sistema con nombre repetido dentro de la misma organización	Mensaje de error, de que ya existe la organización	Ok	

Registro de un sistema con nombre repetido en otra organización	Mensaje de éxito - Visualización del sistema en el listado de sistemas	Ok	
Registro de un sistema sin nombre repetido	Mensaje de éxito - Visualización del sistema en el listado de sistemas	Ok	
Caso de uso	Registro de ambiente		
Escenario	Resultado esperado	Ok (Si/NO)	Comenatrios/mejoras
Registro de un ambiente con nombre repetido dentro de un sistema	Error, ya existe ambiente	OK	
Registro de un ambiente con nombre repetido, pero dentro de otro sistema en la misma organización	Se registra y se recibe el token	Ok	
Registro de un sistema con un nuevo nombre	Se registra y se recibe el token	OK	
Caso de uso	Creación de error		
Escenario	Resultado esperado	Ok (Si/NO)	Comenatrios/mejoras
Creaciónd de un error de severidad 1	(201) Ok /// Ver en pantalla de bug el error, con su severidad	OK	
Creaciónd de un error de severidad 2	(201) Ok /// Ver en pantalla de bug el error, con su severidad	Ok	
Creaciónd de un error de severidad 3	(201) Ok /// Ver en pantalla de bug el error, con su severidad	OK	
Creaciónd de un error de severidad 4	(201) Ok /// Ver en pantalla de bug el error, con su severidad	OK	
Creación de un error de severidad 5	Error al crear un bug	OK	
Creación de un error sin titulo	Error al crear un bug	OK	

Creación de un error sin descripción	(201) Ok /// Ver en pantalla de bug el error, con su severidad	OK	
Caso de uso	Ver un error		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras
Ver un error usuario admin, accediendo desde la pantalla de listado de bugs	Se accede a un error mediante la pantalla de bugs por parte del admin, con los campos: título, descripción, desarrollador y estado y severidad, disponibles para editar	Ok	Falta mostrar la fecha
Ver un error usuario desarrollador, accediendo desde la pantalla de listado de bugs	Se accede a un error mediante la pantalla de bugs por parte del desarrollador, con el todos los datos del bug a la vista, pero solo con el campo Estado disponible para editar	Ok	Falta mostrar la fecha
Ver un error con un id inexistente (ir a localhost:4200/bug/1000)	Mensaje de que no existe la página		
Caso de uso	Editar un error		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras
Edición de un bug por parte de un usuario admin: Edición de cualquier campo: título, descripción, desarrollador y estado y severidad	Ver el bug con su nuevo estado en el listado de bugs	Ok	Falta mostrar la fecha (aunque no estaba claro en el requerimiento)
Cierre de un bug por parte de un usuario desarrollador	Ver el bug con su nuevo estado en el listado de bugs	Ok	Falta mostrar la fecha (aunque no estaba claro en el requerimiento)
Caso de uso	Reporte de bugs críticos		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras

Generación de reporte sin un keyConnection valido	Se responde error	Ok	
Generación de reporte con keyConnection valido	Se retornan los errores no resueltos Deben estar ordenados por mayor severidad Se deben mostrar un máximo de 5 errores	Ok	
Caso de uso	Reporte de estadísticas de errores		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras
Usuario admin quiere acceder al reporte de errores	Permite mostrar un reporte de cantidad de errores por rango de fechas Permite visualizar cantidad de errores por severidad Permite visualizar cantidad de errores por resolución	Ok	
usuario desarrollador quiere acceder al reporte de errores	No se le presenta la opción en el menú, tampoco puede acceder mediante la url	Ok	
Caso de uso	Reporte de top 10 desarrolladores que resolvieron más incidencias asignadas en los últimos 30 días.		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras
Usuario admin quiere acceder al reporte de errores	Se visualiza reporte	OK	
usuario desarrollador quiere acceder al asinación de errores	No se le presenta la opción en el menú, tampoco puede acceder mediante la url	Ok	
Caso de uso	Reporte de incidencias que no están asignadas por más de 2 días		

Escenario	Resultado esperado	Ok (Si/NO)	Comenatrios/mejoras
Usuario admin quiere acceder al reporte de errores	Se visualiza reporte	Ok	
usuario desarrollador quiere acceder al asinación de errores	No se le presenta la opción en el menú, tampoco puede acceder mediante la url	Ok	
Caso de uso	Configuración y recepción de alertas		
Escenario	Resultado esperado	Ok (Si/NO)	Comenatrios/mejoras
Usuario admin se loguea en la aplicación	Alertas apagadas, no recibir correos	Ok	
Usuario developer se loguea en la aplicación	Alertas apagadas,no recibe correos	Ok	
Usuario solicita recibir correos con bugs asignados y sin resolver hace más de 2 días	Recibe el correo correctamente	OK	
Usuario solicita recibir correos de bugs severidad 1 inmediatamente	Recibe el correo inmediatamente	OK	
Usuario solicita recibir correos de bugs severidad 2 inmediatamente	Recibe el correo inmediatamente	OK	
Usuario solicita recibir correos de bugs severidad 3 inmediatamente	Recibe el correo inmediatamente	OK	
Usuario solicita recibir correos de bugs severidad 4 inmediatamente	Recibe el correo inmediatamente	OK	
Usuario solicita recibir correos de bugs severidad 1 inmediatamente, y se le asigna un bug que previamente era de severidad 2	Recibe el correo inmediatamente	OK	
Usuario solicita recibir correos de bugs severidad 2 inmediatamente, , y se le asigna un bug que estaba asignado a un usuario	Recibe el correo inmediatamente	OK	

Usuario solicita recibir correos de bugs severidad 1 a las 20:00	Recibe el correo inmediatamente	OK	
Usuario solicita recibir correos de bugs severidad 2 a las 20:00	Recibe el correo inmediatamente	OK	
Usuario solicita recibir correos de bugs severidad 3 a las 20:00	Recibe el correo inmediatamente	OK	
Usuario solicita recibir correos de bugs severidad 4 a las 20:00	Recibe el correo inmediatamente	OK	
Caso de uso	Visualización de costos		
Escenario	Resultado esperado	Ok (Si/NO)	Comentarios/mejoras
Usuario desarrollador quiere ver costos	No le aparece la opción del menú, ni puede acceder desde la url	Ok	
Usuario admin quiere ver costos	Puede acceder a los costos filtrados por año, puede ver el de 1 mes en particular, puede ver ese mes en pdf	Ok	