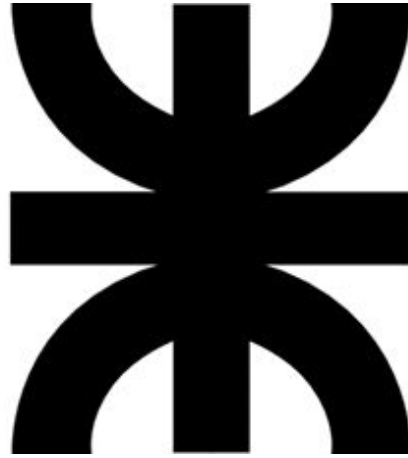


UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL RESISTENCIA



INGENIERÍA EN SISTEMAS DE INFORMACIÓN
Trabajo Práctico Integrador

Materia: Desarrollo de Aplicaciones Cliente - Servidor

Profesores:

- Jorge Eduardo Villaverde
- Fabricio Quevedo

Autores:

- Facundo Ramiro Cuzziol Boccioni
- Danilo Antonio Diez
- Alejandro Fabián Nadal
- Juan Cruz Soto
- Mariano Adrian Troncoso
- Nicolas Adrian Zini

Grupo: 2

Año: 2020

Escenario y abordaje del mismo	3
Escenario:	3
Pacientes	3
Centros hospitalarios	3
Ministerio de salud de la provincia	3
Contexto del sistema:	4
Parte que soluciona:	4
Se vincula con los siguientes subsistemas	4
Perfiles / roles de usuario del sistema.	4
Enlaces importantes	5
Caso/s de uso/s de integración con otro sistema.	5
Librerías / frameworks de frontend	5
React js	6
Librerías / frameworks de backend	6
Node js & Express	6
Mongoose	6
Serverless & Swagger	6
Base de Datos	6
MongoDB	6
Deployment en Cloud / servidor remoto:	7
Generación de Estadísticas automáticas	11
Docker	12
Postman	13
Arquitectura	14
Organización del trabajo	14
Swagger	17
Bibliografía (reseñas principales a páginas utilizadas para tutoriales, librerías, complementos utilizados).	19

Escenario y abordaje del mismo

Escenario:

En medio de la pandemia de COVID-19, el ministerio de salud de la provincia del Chaco ha solicitado a los alumnos de la cátedra Desarrollo de Aplicaciones Cliente-Servidor el diseño e implementación de un sistema que permita conectar a tres actores muy importantes en la lucha contra el virus: los pacientes, los centros hospitalarios y el ministerio de salud de la provincia.

Pacientes

Los pacientes podrán recibir información sobre el estado de la pandemia y recomendaciones por parte del ministerio de salud. Por otro lado, los pacientes pueden hacer consultas a los centros hospitalarios y reportar síntomas que les permita determinar si es necesario que se acerquen a los centros a realizarse test para detectar casos de COVID-19.

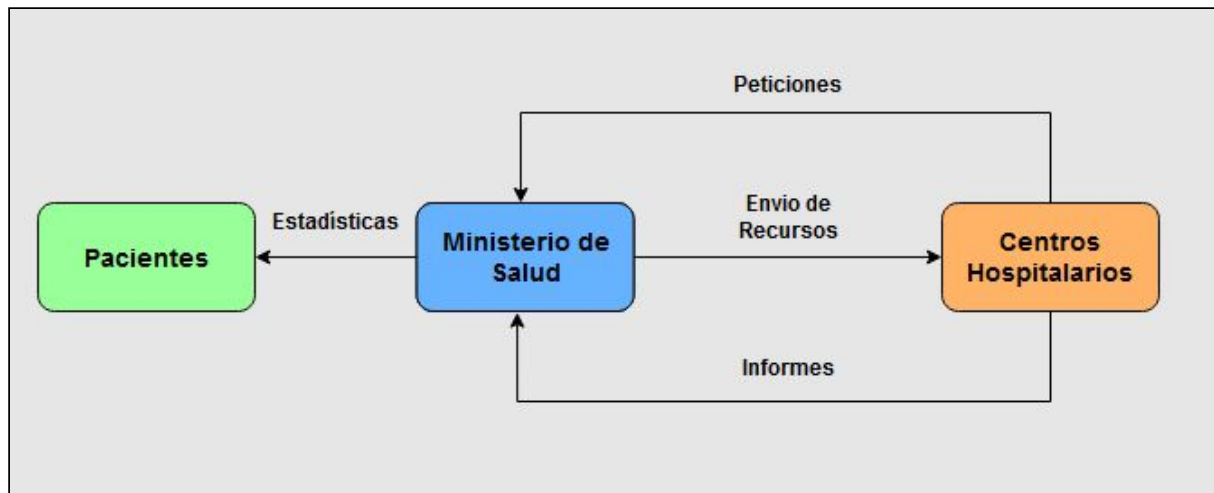
Centros hospitalarios

Los médicos de los centros hospitalarios pueden tomar las consultas realizadas por los pacientes y, en base a los síntomas y su experiencia médica determinar si es un caso sospechoso de COVID-19. Si el ministerio así lo terminase, un profesional puede ser asignado al seguimiento de los casos confirmados de COVID-19.

Ministerio de salud de la provincia

El ministerio recibe de todos los centros hospitalarios los reportes de casos sospechosos reportados por los pacientes y validados por los profesionales de la salud. En base a esta información, y la disponibilidad de recursos, el ministerio asigna recursos de los centros hospitalarios para el seguimiento y detección de casos de COVID-19. Por otro lado, genera reportes sobre el estado y evolución de casos positivos y sospechosos en forma diaria y en tiempo real.

Contexto del sistema:



Contexto de Sistema y relaciones con otros sistemas

El sistema se desarrolla en el contexto de la pandemia a causa del COVID-19, donde fue necesario el desarrollo de los subsistemas correspondientes a Pacientes, Centros Hospitalarios y Ministerio de Salud, los cuales están interrelacionados en cuanto a la información pertinente a la pandemia. Y con esto se busca mejorar la administración tanto de recursos como de tiempo en cuanto a la información que se procesa, logrando así un mayor alcance de las personas y a su vez mejorando la toma de decisiones por parte de la Salud Pública del país.

Parte que soluciona:

El sistema se encarga de la administración de los distintos recursos que el Ministerio de Salud de la provincia debe asignar a los Centros Hospitalarios, en base a las necesidades de los mismos y a los que tenga disponibles para entregar.

También se ocupa de generar estadísticas para el Módulo de Pacientes en base a los informes recibidos de cada centro hospitalario. Las estadísticas generadas se basan en la cantidad de personas contagiadas y las cuales no, esto discriminado por localidad y totales para la provincia de Chaco.

Se vincula con los siguientes subsistemas

Subsistema de Centros Hospitalarios (Grupo N° 3)

Subsistema de Pacientes (Grupo N° 1)

Perfiles / roles de usuario del sistema.

Administrador de recursos: es quien gestiona qué y cuántos recursos asignar a los centros hospitalarios. Esto lo hace en base a los que soliciten, lo que se les ha entregado en el pasado y si dispone de los mismos.

Enlaces importantes

Repositorio del proyecto: [Repositorio GitHub](#)

Link de la App: [Login Ministerio de Salud Publica - Chaco](#)

```
{  
  Usuario: user00,  
  Contraseña: 9hVUNzn  
}
```

Documentación en Swagger: [Documentación Swagger](#)

Invitación al Trello: [Invitación Trello](#)

Caso/s de uso/s de integración con otro sistema.

1. Un centro hospitalario solicita recursos:

Camino Principal:

- a. El Centro Hospitalario escribe y envía una petición al sistema
- b. El Administrador observa la nueva petición
- c. El administrador genera un nuevo envío a partir de la petición, cumpliendo con todos los recursos y médicos solicitados
- d. El Centro Hospitalario verifica si ha recibido nuevos envíos
- e. Fin de Caso de Uso.

Caminos Alternativos:

1. c. 1. Se rechaza la petición por una solicitud excesiva
 - 1.c.1.a. El administrador rechaza la petición por una solicitud excesiva
 - 1.c.1.b. El Centro Hospitalario revisa el estado de las peticiones y descubre que la suya ha sido rechazada.
 - 1.c.1.c. Fin Caso de Uso

2. El Módulo de pacientes solicita estadísticas referidas a la situación de la pandemia:

Camino Principal:

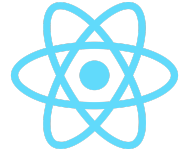
- a. El módulo de pacientes hace una solicitud de estadísticas y recibe las estadísticas.
- b. El módulo del ministerio de salud devuelve las estadísticas actualizadas al último informe.
- c. Fin de caso de uso.

Librerías / frameworks de frontend

Luego detallar las librerías principales que se utilizan y para qué sirven (definir en una o dos líneas para que sirve)

React js

Para el desarrollo del Frontend se utilizó principalmente React, esta librería de JavaScript se basa en la utilización de componentes, lo que nos permite trabajar cómodamente al hacerlo más modular. Específicamente utilizamos la “combinación” de dicha tecnología con el framework Bootstrap (el paquete npm se llama “react-bootstrap”), con esto logramos una combinación de las ventajas de ambas herramientas.



Librerías / frameworks de backend

Node js & Express

Estas dos librerías son las más esenciales para el funcionamiento del backend de la aplicación. Mediante ellas, podemos correr de manera local, el *server* o *backend*. Express resulta de gran importancia, para poder crear el servidor en donde correrá la parte de backend. Además de esto, Node js, junto con npm, permite instalar otras librerías de utilidad para el desarrollo del proyecto. Para esto, este framework incluye un archivo llamado *package.json*, en donde podemos indicarle distintos *scripts* a ejecutar (correr el backend, generar una build, correr tests, entre otros), así también como las dependencias necesarias para el proyecto.



Mongoose

Mongoose es una herramienta útil para modelar objetos de mongoDB con Node js. Para esto, nos permite crear modelos que representen las colecciones de Mongo, así también como diversos métodos para interactuar con la base, mediante los controladores.



Serverless & Swagger

Esta librería se basa en un archivo *serverless.yml*, detallando cada uno de los endpoints, con sus posibles respuestas y métodos correspondientes, para poder realizar un deploy efectivo del backend, así también como actualizar la documentación de cada uno de los endpoints en Swagger.



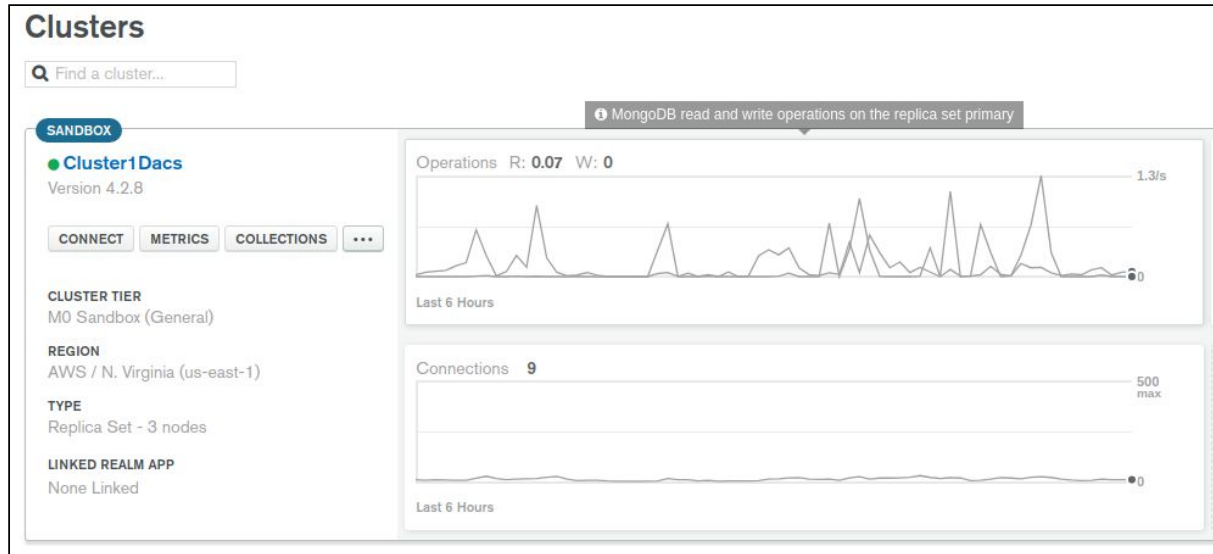
Base de Datos

MongoDB

Para guardar los datos que consideramos necesarios, utilizamos la base de datos no relacional MongoDB. Para esto, se utiliza un cluster gratuito de Mongo Atlas. De esta manera, mediante la creación de un usuario con una



determinada contraseña, cada uno de los integrantes del grupo puede conectarse para acceder, agregar, modificar o eliminar los datos existentes en cada una de las colecciones de la base de datos.



Cluster en Mongo Atlas

Además de esto, para realizar la conexión a la base de datos en el backend, debemos conectarnos mediante un comando *connect* provisto por *Mongoose*. Para esto, se dispone de un archivo llamado *keys.js* en donde se encuentra el *string de conexión* para dicha base. Sin embargo, la contraseña necesaria para la conexión, en base a dicho string de conexión, se obtiene de otro archivo que debe ser creado manualmente, llamado *secrets.js*. El motivo de la creación de este archivo, es para ocultar contenidos de este tipo, como contraseñas, a cualquier usuario que pueda acceder al código (como en un repositorio de GitHub). La sintaxis del archivo *secrets.js*, es la siguiente:

```
module.exports = {
  MONGO_PASSWORD : "grupo2"
}
```

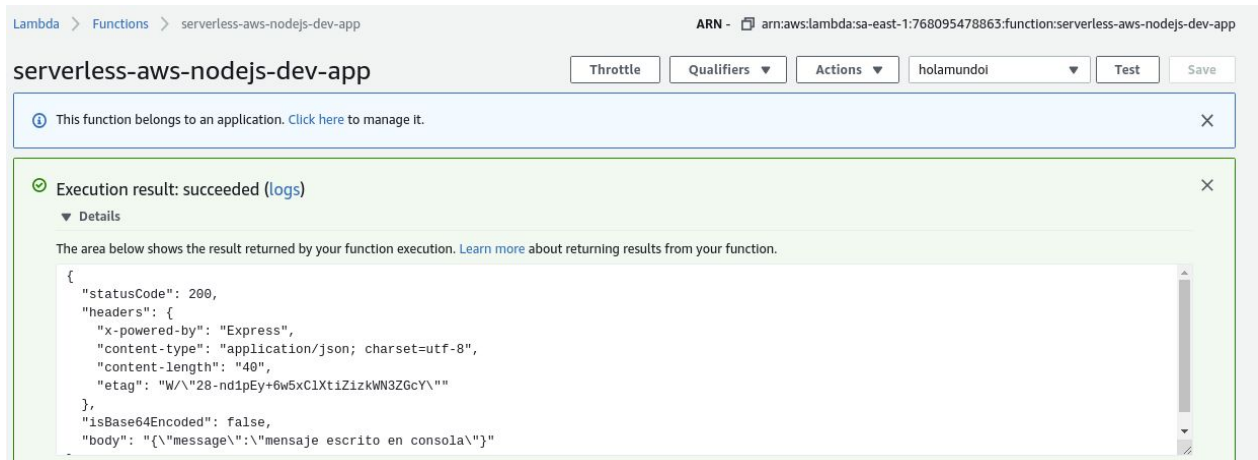
Deployment en Cloud / servidor remoto:

Condiciones iniciales (si es pago o no, que hay que crear)

El deployment en cloud se realiza en la plataforma AWS, cuya siglas significan Amazon Web Services. La misma ofrece una gran cantidad de recursos gratuitos durante su primer año de uso, y otros que son gratis para siempre, entrando en lo que se conoce como Free Tier.

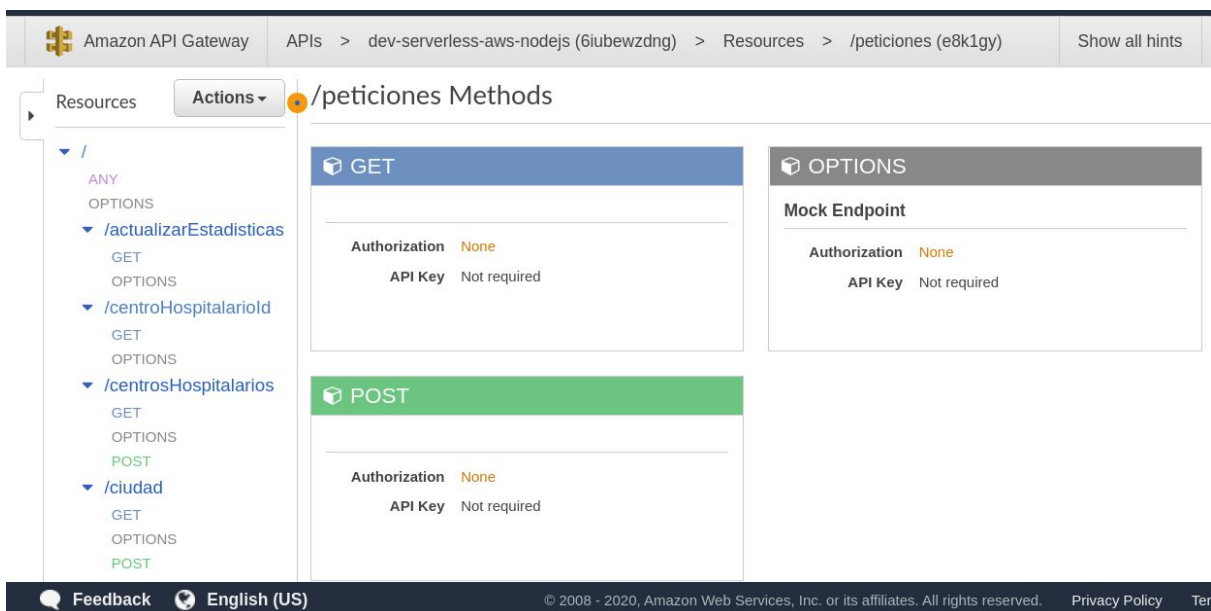
AWS posee numerosos servicios disponibles. Nosotros utilizamos los siguientes:

Amazon Lambda: permite ejecución remota de código. Está planteado para ejecuciones cortas y repetitivas, lo cual es perfecto para el funcionamiento de nuestro sistema. En el, se ejecuta el backend de nuestra aplicación.



AWS Lambda- prueba

API Gateway: Permite la creación de endpoints que permitan ejecutar los servicios puestos a disposición por Amazon Lambda. También genera la documentación para el Swagger.



CloudFormation: Es mediante este servicio que se construyen los recursos que toda la aplicación utiliza.

Delete Initiated for `arn:aws:cloudformation:sa-east-1:768095478863:stack/g2-sesiones-dev/6e809390-be51-11ea-814b-02270569e65c`

CloudFormation > Stacks > serverless-aws-nodejs-dev

Stacks (3)

Filter by stack name

Active ☐ View nested

g2-sesiones-dev
2020-07-04 20:52:34 UTC-0300
DELETE_IN_PROGRESS

serverless-aws-nodejs-dev
2020-05-05 15:05:11 UTC-0300
UPDATE_COMPLETE

sample-app-dev
2020-05-03 20:05:34 UTC-0300
UPDATE_COMPLETE

serverless-aws-nodejs-dev

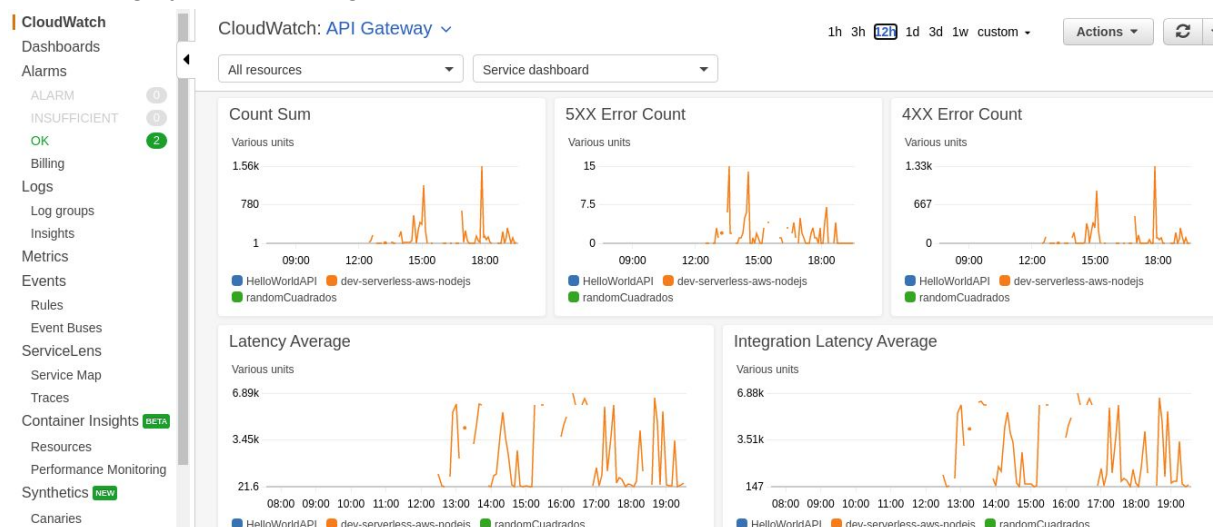
Delete Update Stack actions Create stack

Stack info Events Resources Outputs Parameters Template Change sets

Overview

Stack ID	Description
<code>arn:aws:cloudformation:sa-east-1:768095478863:stack/serverless-aws-nodejs-dev/f699e370-8efa-11ea-867e-023b3b5d2616</code>	The AWS CloudFormation template for this Serverless application
Status	Status reason
UPDATE_COMPLETE	-
Root stack	Parent stack
-	-

CloudWatch: Es este servicio el que nos permite inspeccionar el estado de la aplicación, ver sus logs y hacer un seguimiento del funcionamiento de la misma.



Serverless: es un framework para deployment. Mediante una descripción sumamente precisa y sistemática del funcionamiento de la aplicación, hace un deployment a AWS. Esta descripción se hace en un archivo llamado `serverless.yml`

AWS S3: Es el sistema de almacenamiento de AWS. Allí se almacenan todos los archivos dependientes de los demás servicios utilizados.

Welcome to Amazon S3. Create new buckets or select an existing bucket to view and configure properties. [Documentation](#)

We've temporarily re-enabled the previous version of the S3 console while we continue to improve the new S3 console experience. [Switch to the new console.](#)

S3 buckets [Discover the console](#)

Search for buckets All access types

+ Create bucket Edit public access settings Empty Delete

8 Buckets 1 Regions

Bucket name	Access	Region	Date created
<input type="checkbox"/> codebuild-sa-east-1-768095478863-input-bucket	Bucket and objects not public	South America (São Paulo)	Jun 27, 2020 2:41:01 PM GMT-0300
<input type="checkbox"/> codebuild-sa-east-1-768095478863-output-bucket	Bucket and objects not public	South America (São Paulo)	Jun 27, 2020 2:44:53 PM GMT-0300
<input type="checkbox"/> codepipeline-sa-east-1-930552209396	Objects can be public	South America (São Paulo)	May 31, 2020 10:18:34 AM GMT-0300
<input type="checkbox"/> dummyhealthministry	Objects can be public	South America (São Paulo)	May 3, 2020 1:03:36 PM GMT-0300
<input type="checkbox"/> fronthealthministry	Public	South America (São Paulo)	May 22, 2020 4:11:34 PM GMT-0300

Como se publica / actualiza

Se debe tener una cuenta de AWS. El proceso de registro es igual que el de cualquier servicio online.

Instalar serverless es similar a instalar cualquier paquete en javascript.

npm install serverless

Para actualizar la aplicación, se utiliza serverless. Es un framework que toma la documentación de la aplicación y la transforma en un stack mediante CloudFormation. Se instala con npm, y se deployea la aplicación con el siguiente comando:

serverless deploy

```
ale@debian:~/Documents/Universidad/DACS/2020-G2-TPI/API$ serverless deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service serverless-aws-nodejs.zip file to S3 (62.57 MB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.....
Serverless: Stack update finished...
Service Information
service: serverless-aws-nodejs
stage: dev
region: sa-east-1
stack: serverless-aws-nodejs-dev
resources: 90
api keys:
None
endpoints:
```

Para deployar el front, utilizamos la herramienta de consola de aws, tras hacer una build

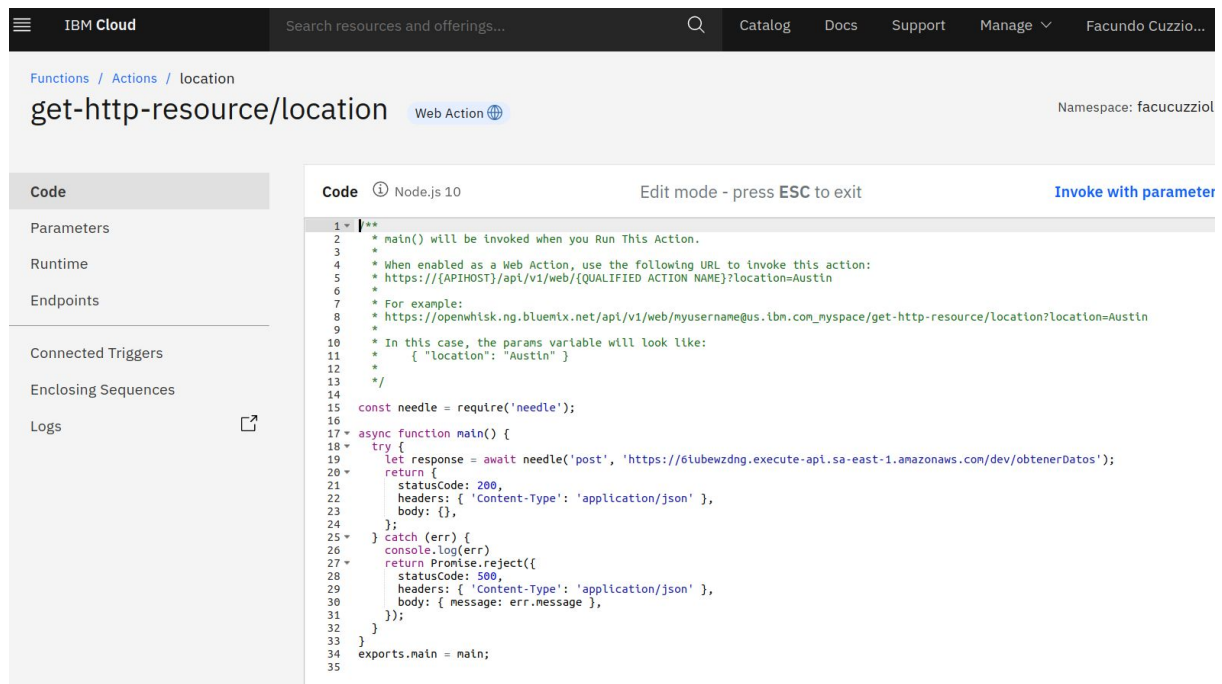
npm run build

aws s3 sync ./build s3://fronthealthministry

```
ale@debian:~/Documents/Universidad/DACS/2020-G2-TPI/front/minsalud$ aws s3 sync ./build s3://fronthealthministry
upload: build/asset-manifest.json to s3://fronthealthministry/asset-manifest.json
upload: build/precache-manifest.583700d6695efd6bdf46e521d2ed65b7.js to s3://fronthealthministry/precache-manifest.583700d6695efd6bdf46e521d2ed65b7.js
upload: build/logo192.png to s3://fronthealthministry/logo192.png
upload: build/logo512.png to s3://fronthealthministry/logo512.png
upload: build/robots.txt to s3://fronthealthministry/robots.txt
upload: build/manifest.json to s3://fronthealthministry/manifest.json
upload: build/service-worker.js to s3://fronthealthministry/service-worker.js
upload: build/index.html to s3://fronthealthministry/index.html
upload: build/static/js/2.34f8c0b3.chunk.js.LICENSE.txt to s3://fronthealthministry/static/js/2.34f8c0b3.chunk.js.LICENSE.txt
upload: build/static/js/runtime-main.51460274.js to s3://fronthealthministry/static/js/runtime-main.51460274.js
upload: build/static/js/runtime-main.51460274.js.map to s3://fronthealthministry/static/js/runtime-main.51460274.js.map
upload: build/static/js/main.adabeead.chunk.js to s3://fronthealthministry/static/js/main.adabeead.chunk.js
upload: build/static/media/ministerio-logo.7cb45ec9.png to s3://fronthealthministry/static/media/ministerio-logo.7cb45ec9.png
upload: build/static/js/2.34f8c0b3.chunk.js to s3://fronthealthministry/static/js/2.34f8c0b3.chunk.js
upload: build/static/js/main.adabeead.chunk.js.map to s3://fronthealthministry/static/js/main.adabeead.chunk.js.map
upload: build/static/css/main.cffcd0bd0.chunk.css to s3://fronthealthministry/static/css/main.cffcd0bd0.chunk.css
upload: build/static/js/2.34f8c0b3.chunk.js.map to s3://fronthealthministry/static/js/2.34f8c0b3.chunk.js.map
upload: build/static/css/main.cffcd0bd0.chunk.css.map to s3://fronthealthministry/static/css/main.cffcd0bd0.chunk.css.map
ale@debian:~/Documents/Universidad/DACS/2020-G2-TPI/front/minsalud$
```

Generación de Estadísticas automáticas

Existen ciertas tareas que deben ejecutarse automáticamente en el backend. Para ello, utilizamos IBM Cloud, el cual dispone de un sencillo sistema de cron para generar eventos HTTP en tiempos específicos. En concreto, utilizamos esta herramienta para generar el evento que actualiza las estadísticas, una vez al día.



The screenshot shows the IBM Cloud Functions console. The top navigation bar includes 'IBM Cloud', a search bar, and links for 'Catalog', 'Docs', 'Support', 'Manage', and the user 'Facundo Cuzzio...'. The main content area displays the details for a function named 'get-http-resource/location', which is a 'Web Action'. On the left sidebar, there are tabs for 'Code', 'Parameters', 'Runtime', 'Endpoints', 'Connected Triggers', 'Enclosing Sequences', and 'Logs'. The 'Code' tab is selected, showing a JavaScript code editor with the following code:

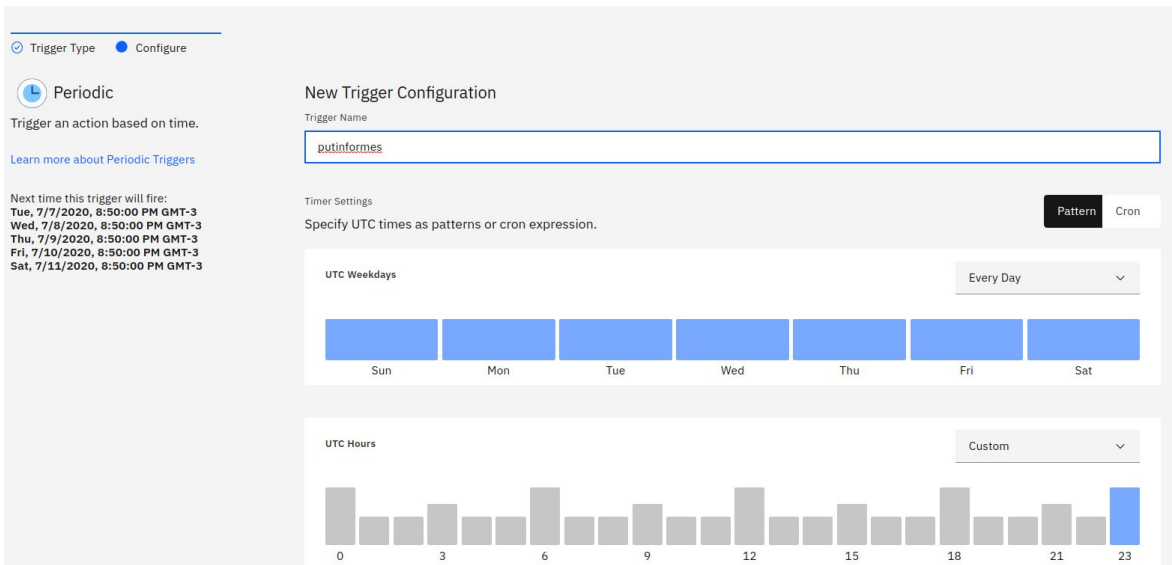
```
1  /**
2   * main() will be invoked when you Run This Action.
3   *
4   * When enabled as a Web Action, use the following URL to invoke this action:
5   * https://{APIHOST}/api/v1/web/{QUALIFIED ACTION NAME}?location=Austin
6   *
7   * For example:
8   * https://openwhisk.ng.bluemix.net/api/v1/web/myusername@us.ibm.com/myspace/get-http-resource/location?location=Austin
9   *
10  * In this case, the params variable will look like:
11  * { "location": "Austin" }
12  *
13  */
14
15  const needle = require('needle');
16
17  async function main() {
18    try {
19      let response = await needle('post', 'https://6tubewzdnq.execute-api.sa-east-1.amazonaws.com/dev/obtenerDatos');
20      return {
21        statusCode: 200,
22        headers: { 'Content-Type': 'application/json' },
23        body: {}
24      };
25    } catch (err) {
26      console.log(err);
27      return Promise.reject({
28        statusCode: 500,
29        headers: { 'Content-Type': 'application/json' },
30        body: { message: err.message },
31      });
32    }
33  }
34  exports.main = main;
```

Acción para generar los Informes desde IBM Cloud

Una vez que creamos la acción a ejecutar, debemos vincularla con un trigger, que se ejecute de manera periódica, eligiendo qué días y en qué momento queremos que se ejecute

Functions / Create / Trigger

Connect Trigger



Trigger Type Trigger Type Configure

Periodic
Trigger an action based on time.
[Learn more about Periodic Triggers](#)

Next time this trigger will fire:
Tue, 7/7/2020, 8:50:00 PM GMT-3
Wed, 7/8/2020, 8:50:00 PM GMT-3
Thu, 7/9/2020, 8:50:00 PM GMT-3
Fri, 7/10/2020, 8:50:00 PM GMT-3
Sat, 7/11/2020, 8:50:00 PM GMT-3

New Trigger Configuration

Trigger Name

Timer Settings
Specify UTC times as patterns or cron expression.

Pattern **Cron**

UTC Weekdays
Every Day

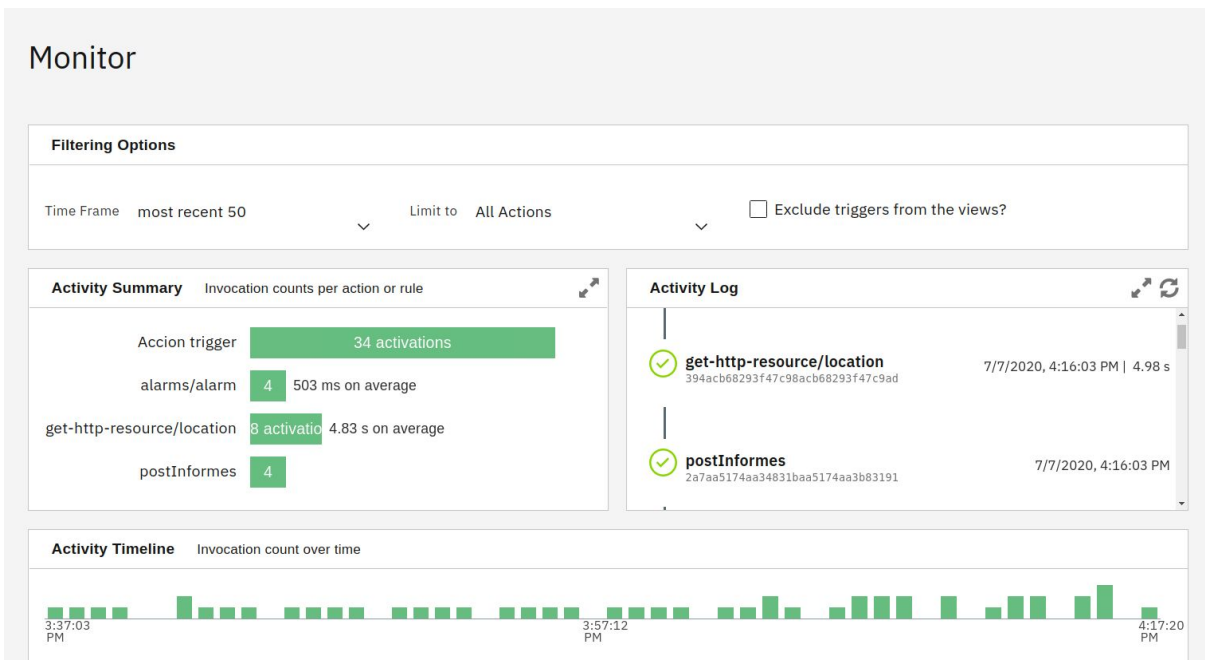
Sun Mon Tue Wed Thu Fri Sat

UTC Hours
Custom

0 3 6 9 12 15 18 21 23

Creación de un trigger para generar los informes una vez al día

Una vez que la acción se encuentra activa, podemos monitorear las acciones que se realizan, en la sección *Monitor* de IBM cloud:



Monitoreo de acciones ejecutadas por el trigger de IBM Cloud

Docker

También se cuenta con la opción de correr la aplicación de manera local mediante containers de Docker. Para esto, existe un archivo Dockerfile para el backend, así como un archivo Dockerfile separado para el frontend. Para conectar ambos containers y logran un correcto



funcionamiento de la aplicación, también se incluye, en la carpeta raíz del proyecto, un archivo *docker-compose* en formato de versión 3.

El archivo *docker-compose* tiene la finalidad de ejecutar ambos containers, donde el backed sería la parte de *server*, y el frontend, la parte del *cliente*. Estos conforman los servicios que se detallan en dicho archivo. Para cada uno de estos servicios, se utilizan variables de entorno, como el url de la API, y el puerto donde este corre, así también como los volúmenes y comandos necesarios para su ejecución.

Estos dos servicios se conectan, con la versión más reciente de frontend y backend disponibles en Cloud. De esta manera, ante cada deploy, tenemos la versión más reciente de la aplicación, al ejecutar *docker-compose up*.

Por otro lado, se accede a la base de datos mediante el cluster de MongoDB, para esto, docker recurre al archivo *secrets.js*, el cual se crea manualmente con anterioridad a la ejecución, y dentro de dicho archivo, se encuentra la contraseña utilizada para conectarse a la base de datos.

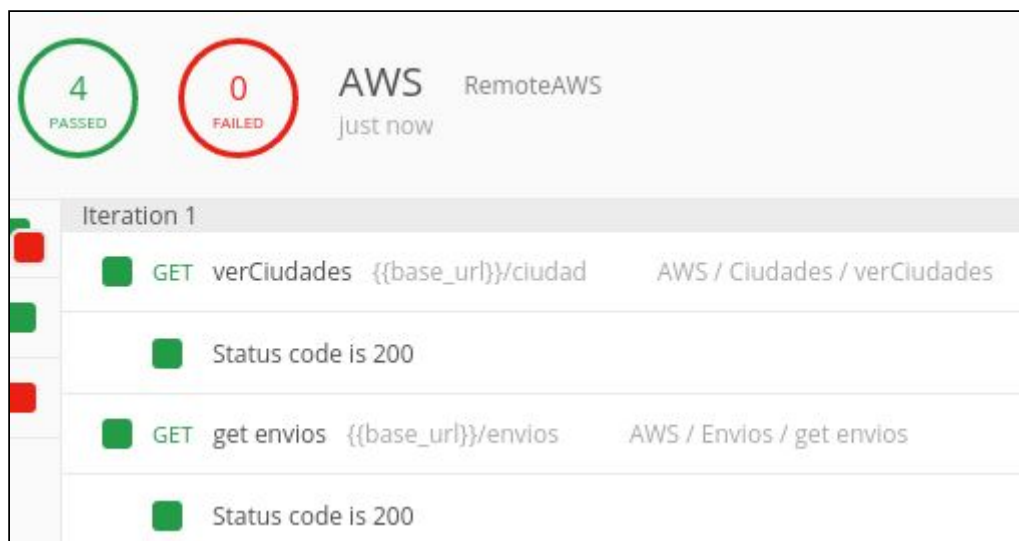
Postman

Postman es una herramienta de gran utilidad para probar los distintos endpoints que se crean, haciendo un GET o POST sobre cada uno y pasando contenido en formato JSON en la mayoría de los POST que así lo requieran.



Además de esto, también se realizan pruebas sobre cada ejecución en Postman, comprobando si el StatusCode que devuelta es igual a 200. En relación con esto, también creamos distintos entornos para correr los Endpoints. Por un lado, se cuenta con un entorno local, cuando corremos la aplicación de manera local. Asimismo, también se cuenta con un entorno de *RemoteAWS*, en donde se llama a los endpoints desde la instancia Cloud donde corre la aplicación.

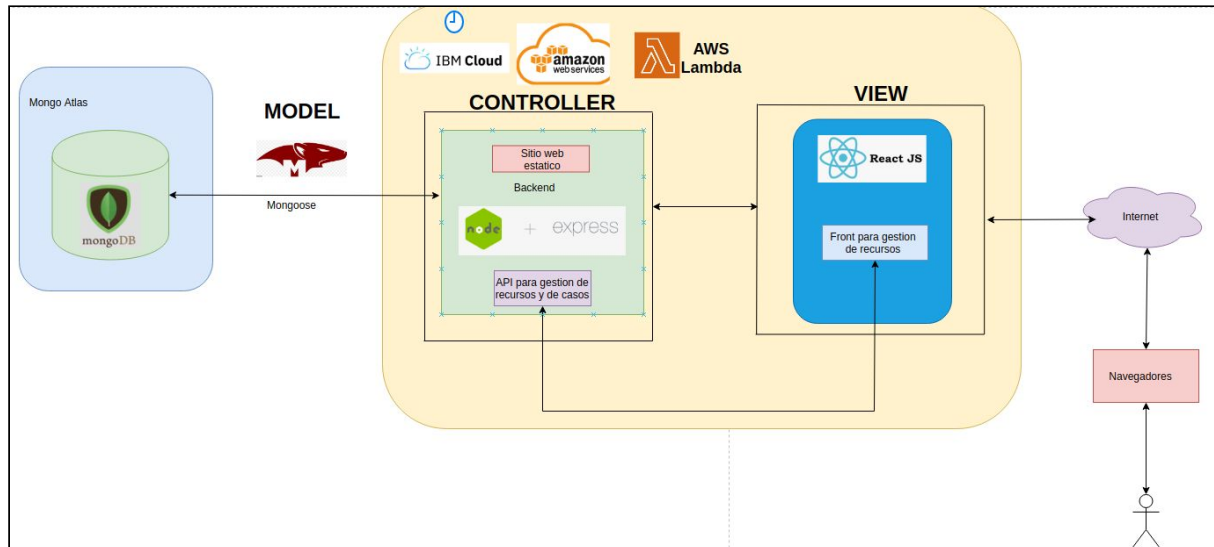
Finalmente, postman también cuenta con una herramienta llamada *Runner*, en donde podemos correr las pruebas mencionadas anteriormente, seleccionando individualmente cada uno de los métodos que se deseen probar, así también como el orden de ejecución de los mismos, obteniendo así un resumen de los resultados de los tests, como se muestra a continuación:



Resultados de ejecutar algunos tests con Postman Runner

Arquitectura

A continuación, se representa la arquitectura utilizada con modelo MVC, con las diferentes tecnologías utilizadas en cada una de las partes de la aplicación.



Arquitectura de la aplicación

Organización del trabajo


A fin de cumplir con las tareas asignadas el equipo de trabajo se organizó de la siguiente manera (tratando de aplicar lo aprendido en otras asignaturas, cómo Ingeniería de Software).

Principalmente nos inclinamos a la utilización de prácticas ágiles, con las herramientas y reuniones propuestas por las mismas.

Para cada funcionalidad del sistema, se definieron historias de usuario en intervalos de una semana, teniendo en cuenta los tiempos del equipo (que dependían de la carga de trabajo de otras materias) y la capacidad de realizar en este periodo de tiempo.

Luego de definir las historias de usuario, se asignaban tareas dividiendolos en subequipos de trabajo, así quedaban asignados dos o más personas por tarjeta. Se definen subequipos, donde una persona era más “experta” en la tarea (por ejemplo: más experiencia habiendo trabajado anteriormente en el mismo trabajo con consultas a la base de datos, generación de nuevos endpoints, creación de modelos, entre otros), con el fin de que todo el equipo de trabajo tenga contacto con cada parte del código. En cada sprint los subequipos iban rotando.

Al terminar la historia, adjuntamos los commits correspondientes, para lograr una mayor trazabilidad del trabajo:


A partir de los informes, generar Estadísticas
✕

en la lista [DONE Sprint 7](#)

MIEMBROS

AN

+

BACK

+

ETIQUETAS

SUGERENCIAS

Unirse

Comentarios

Descripción

Añadir una descripción más detallada...

Confirmaciones de GitHub

Quitar...

Confirmación 2be06b1 de FRe-DACS/2020-G2-TPI

Confirmación 333f2fb de FRe-DACS/2020-G2-TPI

Confirmación 16e8cca de FRe-DACS/2020-G2-TPI

It appears your browser's localStorage is disabled, or github.trello.services is not allowed access. To authenticate with GitHub please enable access to localStorage for github.trello.services

AÑADIR A LA TARJETA

Miembros

Etiquetas

Checklist

Vencimiento

Adjunto

Portada

POWER-UPS

GitHub

Conseguir más Power...

Tablero de Trello:

Tableros

DACS G2

Para asuntos privados

Privado

Invitar

ETIQUETAS

BACK

FRONT

TESTING

BUGS

DOMENTACION

BACKLOG

Script en Back para crear el secret.js

Determinar que vamos a hacer con los datos que NO enviamos a pacientes (tests), mostramos nosotros? nos hacemos los giles? o que? PRIORIDAD DE REALIZACIÓN DE US MUY BAJA

Cosas que otros grupos tienen que pasarnos

Acuerdo con otros grupos sobre los esquemas de bases de datos

Endpoint envios: grupo pacientes

Endpoint Informes: grupo pacientes

Lista / Informe de Medicos que hay un Hospital

Mas información en la UI de asignación de recursos

Para cada centro hospitalario, ver últimas peticiones hechas (mostrar fechas)

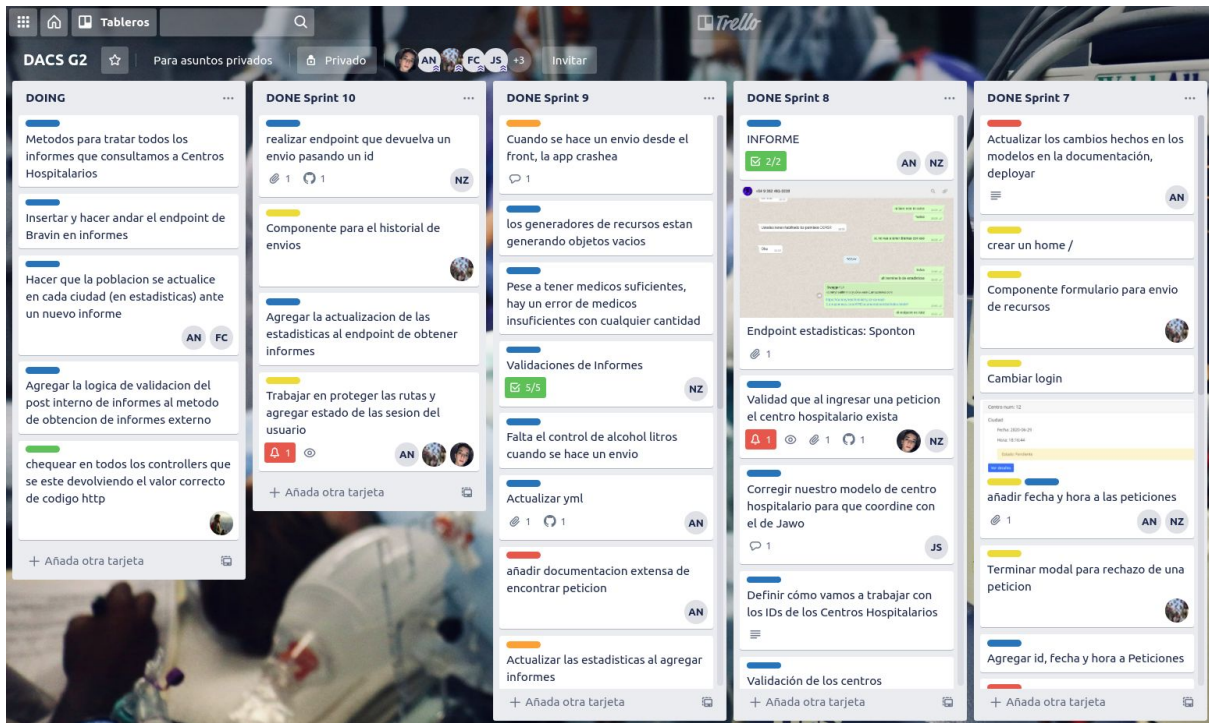
BUGS

Concatenacion de url en el front de envios y peticion amazon

TO DO

Aprender a generar eventos automaticos en AWS

Agregar información referida a la organización del trabajo en el informe



DACS G2 Para asuntos privados Privado Invitar

DOING

- Metodos para tratar todos los informes que consultamos a Centros Hospitalarios
- Insertar y hacer andar el endpoint de Bravin en informes
- Hacer que la poblacion se actualice en cada ciudad (en estadísticas) ante un nuevo informe
- Agregar la logica de validacion del post interno de informes al metodo de obtencion de informes externo
- chequear en todos los controllers que se este devolviendo el valor correcto de codigo http

+ Añada otra tarjeta

DONE Sprint 10

- realizar endpoint que devuelva un envio pasando un id
- Componente para el historial de envios
- Agregar la actualizacion de las estadísticas al endpoint de obtener informes
- Trabajar en proteger las rutas y agregar estado de la sesion del usuario

+ Añada otra tarjeta

DONE Sprint 9

- Cuando se hace un envio desde el front, la app crashea
- los generadores de recursos estan generando objetos vacios
- Pese a tener medicos suficientes, hay un error de medicos insuficientes con cualquier cantidad
- Validaciones de Informes
- Falta el control de alcohol litros cuando se hace un envio
- Actualizar yml
- añadir documentacion extensa de encontrar peticion
- Actualizar las estadísticas al agregar informes

+ Añada otra tarjeta

DONE Sprint 8

INFORME 2/2

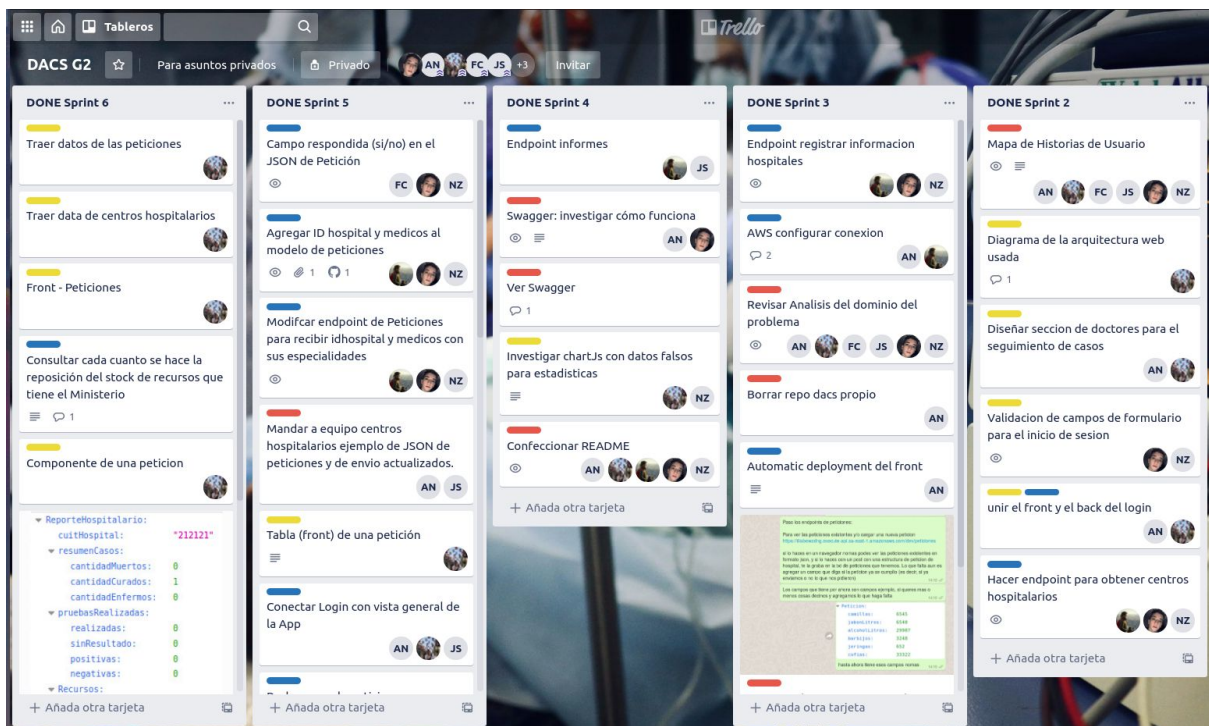
- Endpoint estadísticas: Sponton
- Validad que al ingresar una peticion el centro hospitalario exista
- Corregir nuestro modelo de centro hospitalario para que coordine con el de Jawo
- Definir cómo vamos a trabajar con los IDs de los Centros Hospitalarios
- Validación de los centros

+ Añada otra tarjeta

DONE Sprint 7

- Actualizar los cambios hechos en los modelos en la documentación, deployar
- crear un home /
- Componente formulario para envio de recursos
- Cambiar login
- añadir fecha y hora a las peticiones
- Terminar modal para rechazo de una peticion
- Agregar id, fecha y hora a Peticiones

+ Añada otra tarjeta



DACS G2 Para asuntos privados Privado Invitar

DONE Sprint 6

- Traer datos de las peticiones
- Traer data de centros hospitalarios
- Front - Peticiones
- Consultar cada cuanto se hace la reposición del stock de recursos que tiene el Ministerio
- Componente de una peticion

+ Añada otra tarjeta

DONE Sprint 5

- Campo respondida (si/no) en el JSON de Petición
- Agregar ID hospital y medicos al modelo de peticiones
- Modificar endpoint de Peticiones para recibir idhospital y medicos con sus especialidades
- Mandar a equipo centros hospitalarios ejemplo de JSON de peticiones y de envio actualizados.
- Tabla (front) de una petición
- Conectar Login con vista general de la App

+ Añada otra tarjeta

DONE Sprint 4

- Endpoint informes
- Swagger: investigar cómo funciona
- Ver Swagger
- Investigar chart.js con datos falsos para estadísticas
- Confeccionar README

+ Añada otra tarjeta

DONE Sprint 3

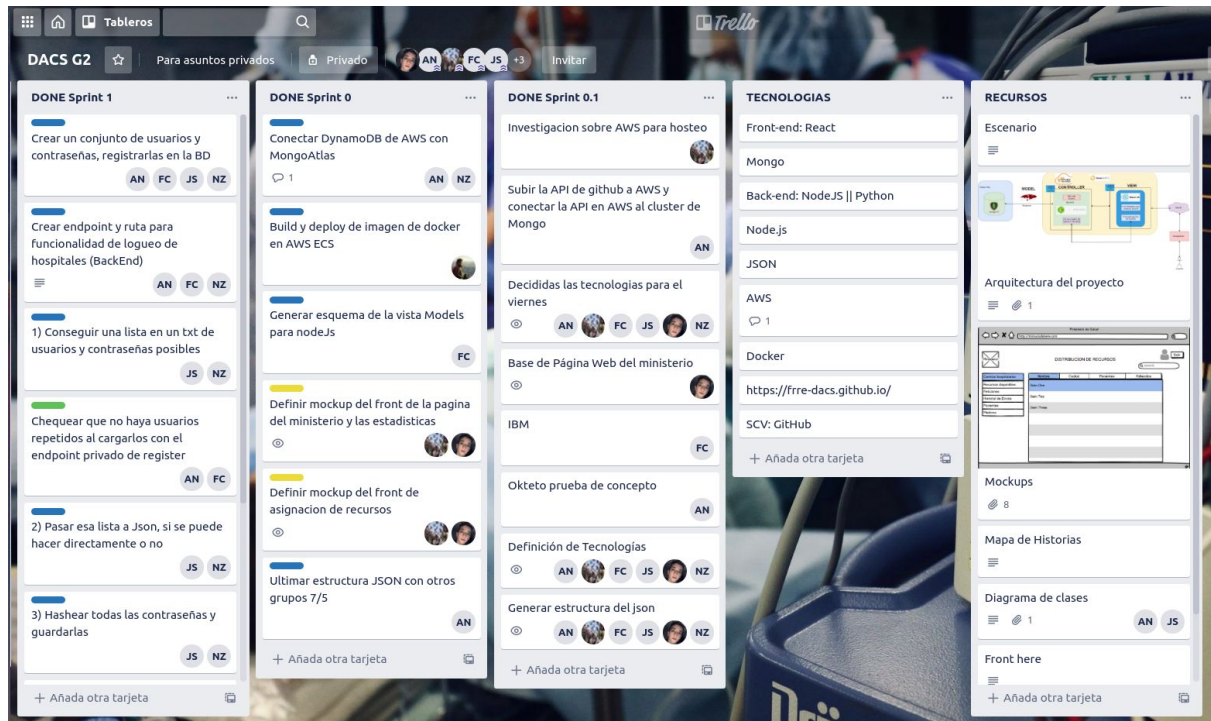
- Endpoint registrar informacion hospitales
- AWS configurar conexion
- Revisar Analisis del dominio del problema
- Borrar repo dacs propio
- Automatic deployment del front

+ Añada otra tarjeta

DONE Sprint 2

- Mapa de Historias de Usuario
- Diagrama de la arquitectura web usada
- Diseñar seccion de doctores para el seguimiento de casos
- Validacion de campos de formulario para el inicio de sesion
- unir el front y el back del login
- Hacer endpoint para obtener centros hospitalarios

+ Añada otra tarjeta



Swagger

[Documentación Swagger](#)

```
aws apigateway get-export --parameters extensions='apigateway'  
--rest-api-id 6iubewzdng --stage-name dev --export-type swagger  
latestswagger2.json --region sa-east-1
```

Utilizamos esta herramienta para la documentación referida a los endpoints utilizados, para que sean conocidos por los demás grupos, la función de los mismos, los parámetros necesarios y ejecutarlos para ver las posibles respuestas que se obtienen.

POST	/ciudad	Agrega una ciudad
OPTIONS	/ciudad	
GET	/encontrarPetición	Allows to find a petition by its mongo id
OPTIONS	/encontrarPetición	
GET	/encontrarPeticonesFecha	Gets a list of all the existing peticiones ordered by date
OPTIONS	/encontrarPeticonesFecha	
GET	/envios	Gets a list of all the existing Envios
POST	/envios	Adds a Envio to the list of existing Envios
OPTIONS	/envios	

Algunos endpoints en swagger

GET	/centroHospitalarioId	Gets a Centros Hospitalario
Parameters		
Name	Description	
idCentro string (query)	Id of Centro Hospitalario	idCentro - Id of Centro Hospitalario
Responses		
Code	Description	
200	200 response	Example Value Model
		<pre>{ "idCentro": 0, "ciudad": "string", "direccion": "string", "nombre": "string", "idCiudad": 0 }</pre>
403	403 response	

Obtención de centro hospitalario por idCentro

Bibliografía (reseñas principales a páginas utilizadas para tutoriales, librerías, complementos utilizados).

Se utilizaron las siguientes documentaciones oficiales (también se utilizaron otros enlaces a páginas de respaldo o tutoriales, pero resulta imposible en un sentido práctico colocar todos)

Github: <https://guides.github.com/>
NodeJS: <https://nodejs.org/api/all.html>
Express: <https://expressjs.com/>
Serverless: <https://www.serverless.com/framework/docs/>
React: <https://reactjs.org/docs/getting-started.html>
React-Bootstrap: <https://react-bootstrap.github.io/>
MongoDB: <https://docs.mongodb.com/>
Mongoose: <https://mongoosejs.com/docs/api.html>
Mongo Atlas: <https://docs.atlas.mongodb.com/>
AWS: <https://docs.aws.amazon.com/>
Docker:
<https://docs.docker.com/compose/compose-file/>
https://docs.docker.com/engine/reference/commandline/image_build/
Swagger: <https://swagger.io/>