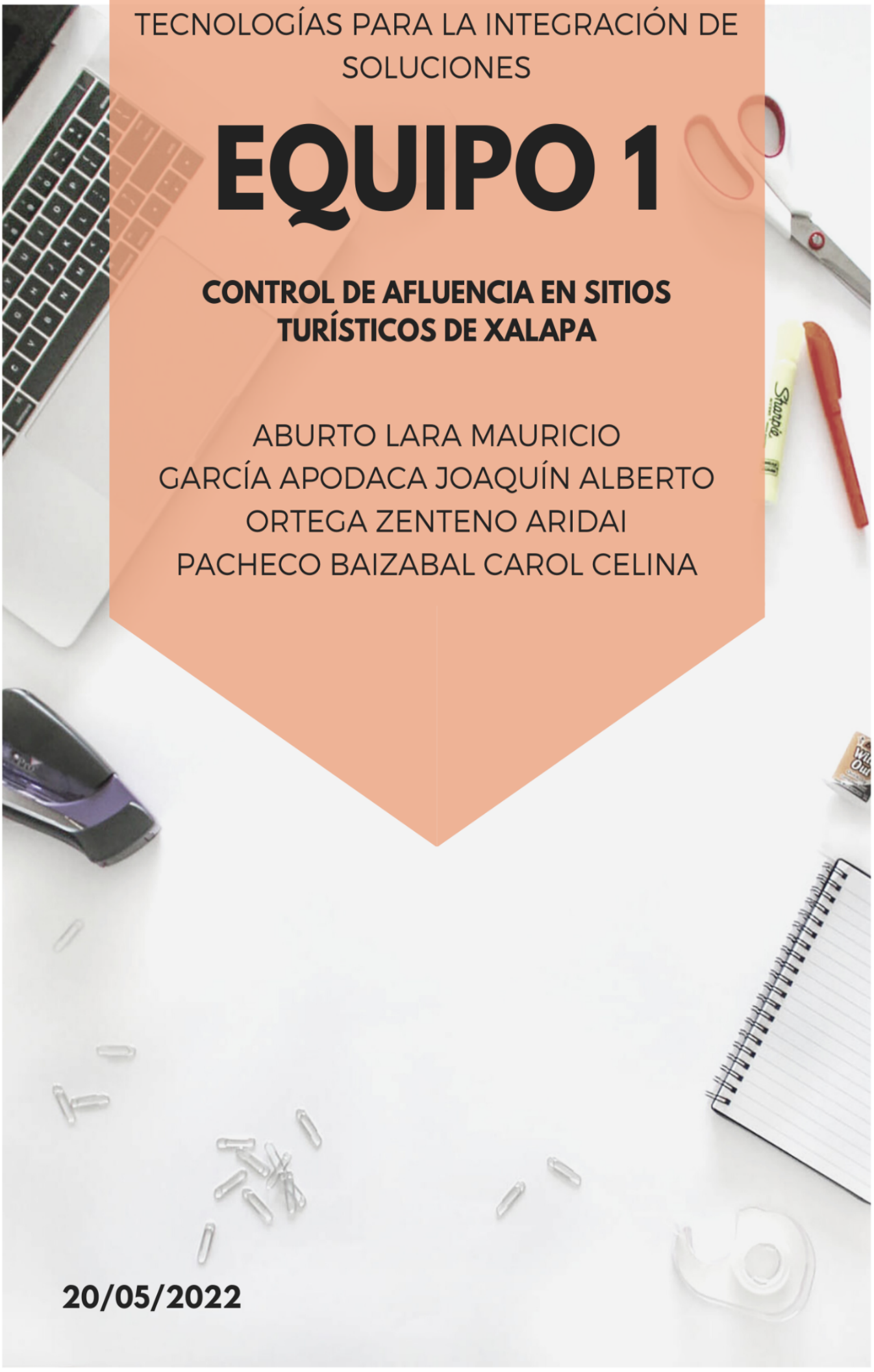
TECNOLOGÍAS PARA LA INTEGRACIÓN DE SOLUCIONES

EQUIPO 1

CONTROL DE AFLUENCIA EN SITIOS TURÍSTICOS DE XALAPA

ABURTO LARA MAURICIO
GARCÍA APODACA JOAQUÍN ALBERTO
ORTEGA ZENTENO ARIDAI
PACHECO BAIZABAL CAROL CELINA



20/05/2022

ÍNDICE

TECNOLOGÍAS PARA LA INTEGRACIÓN DE SOLUCIONES

PROYECTO

Pinacoteca Diego Rivera, Biblioteca Carlos Fuentes, Ágora de la Ciudad de Xalapa

CONTROL DE AFLUENCIA EN SITIOS TURÍSTICOS DE XALAPA

INTRODUCCIÓN

En la actualidad los sistemas web son indispensables para la administración de locales, negocios, empresas, museos, etcétera, así mismo la implementación de bases de datos permiten a estos llevar el control adecuado de la información.

Se identificó una situación donde se puede aplicar un servicio web el cual beneficiaría a un entorno que recurramos, facilitando a los usuarios el manejo de información, aplicando los conocimientos vistos durante el curso como lo son, SOAP, REST, el uso y manejo de Docker.

MOTIVACIÓN

Promover la cultura realizando un servicio que permita tener reunidos los servicios web en un solo lugar para su fácil acceso y que agilice las búsquedas en los sitios web de estos lugares emblemáticos de la ciudad de Xalapa.

PROBLEMÁTICA

Se ubicaron tres lugares cercanos territorialmente y emblemáticos de la ciudad de Xalapa de los que se desea registrar el número de visitantes para basar esta información en la divulgación de eventos que tomen lugar en estos lugares.

SOLUCIÓN

Pinacoteca Diego Rivera

- Registrar los visitantes
    - ID
    - Nombre de persona
    - Fecha
    - Hora
    - Num de personas
- Registrar eventos por temporadas

- Registrar eventos
- Modificar
- Buscar
    - ID
    - Nombre del evento
    - Fecha
    - Hora
    - Costo
- Registrar artistas / Historial de los artistas

Biblioteca Carlos Fuentes

- Registrar lo visitantes
- Registrar el uso de la biblioteca o los servicios informáticos
- Solicitar espacio para conferencia o evento

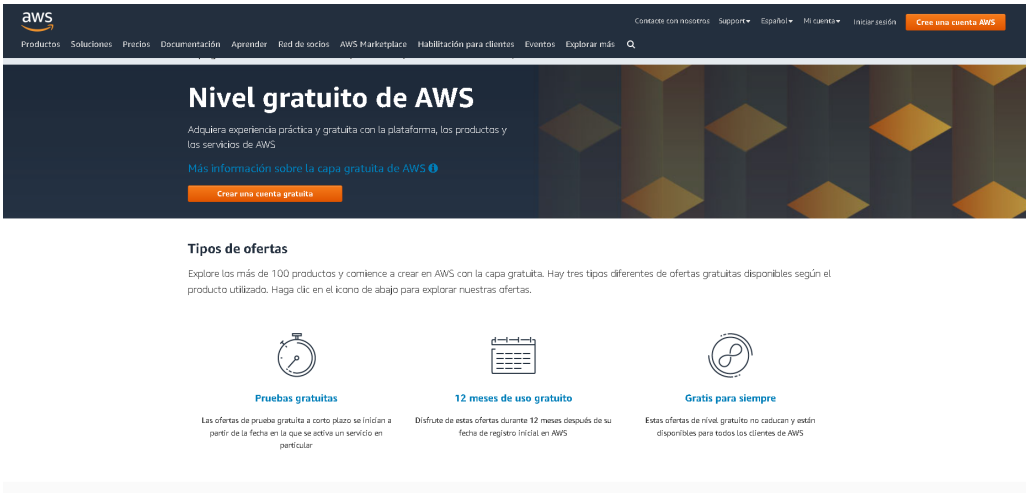
Ágora de la ciudad de Xalapa

- Registrar los visitantes
- Registrar costos de los eventos
- Modificar
- Buscar
    - ID
    - Nombre del evento
    - Fecha
    - Hora
    - Costo
- Registrar las películas que se proyectan

HOSTING

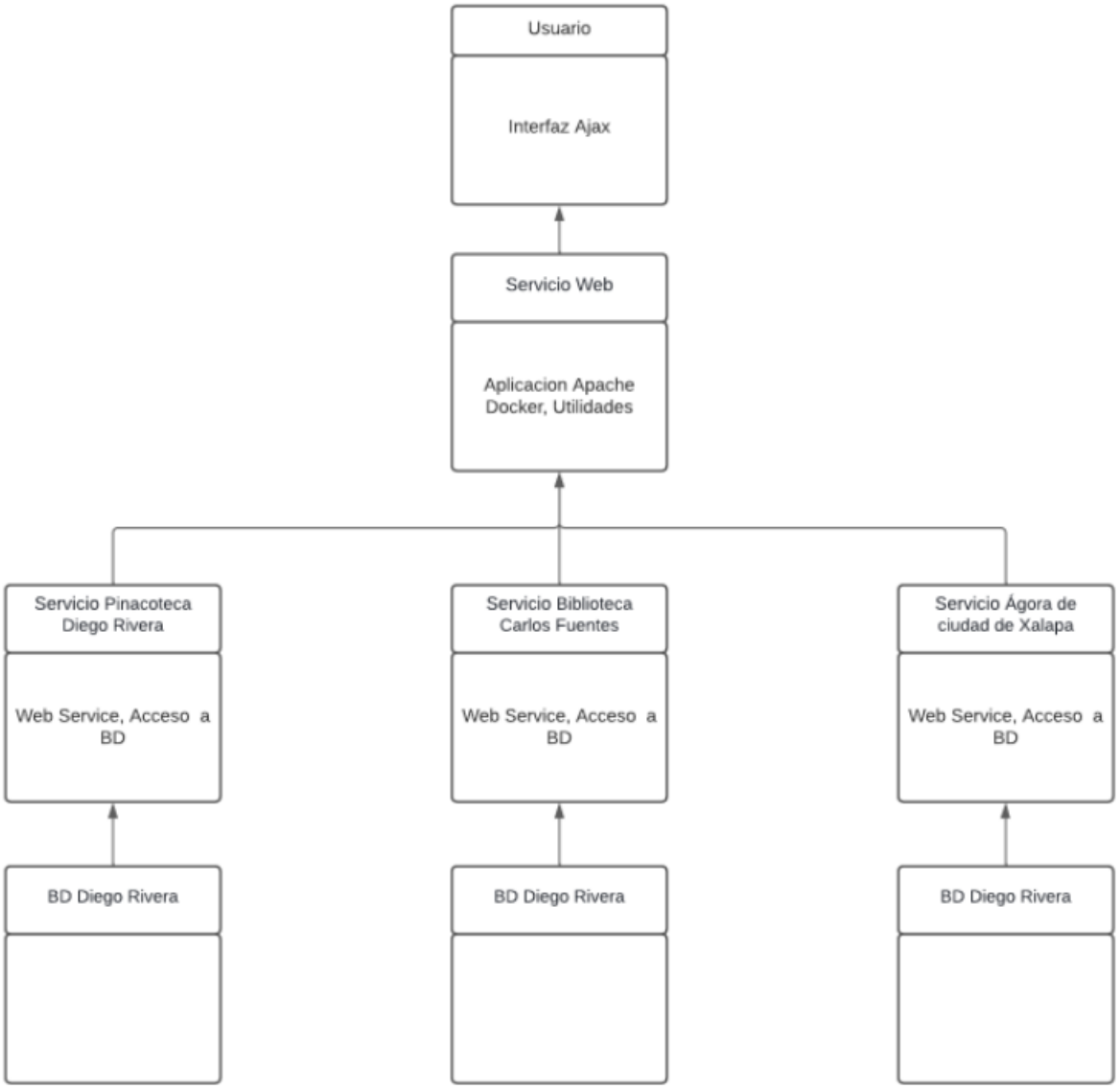
Buscamos servicios de hosting que preferiblemente tengan un plan de servicios gratuitos donde las opciones que vimos fueron:

Amazon Web Services (**AWS** abreviado) es una colección de servicios de computación en la nube pública (también llamados servicios web) que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de Internet por Amazon.com. Es usado en aplicaciones populares como Dropbox, Foursquare, HootSuite. Es una de las ofertas internacionales más importantes de la computación en la nube y compite directamente con servicios como Microsoft Azure, Google Cloud Platform y IBM Cloud. Es considerado como un pionero en este campo.



https://aws.amazon.com/es/free/?trk=6e90e8fa-6bd8-4a6f-be4b-3bc9e717eb2e&sc_channel=ps&sc_campaign=acquisition&sc_medium=ACQ-P|PS-GO|Brand|Desktop|SU|AWS|Core|LATAMO|ES|Text&ef_id=Cj0KCQjw1N2TBhCOARIsAGVHQc571QEEjuxk0x4CmYwDjT5cv_vUbZm3sZK4VSeWPYBTghhx-ziNqtgaAgCyEALw_wcB:G:s&s_kwcid=AL!4422!3!561348326849!e!!g!!amazon%20aws&ef_id=Cj0KCQjw1N2TBhCOARIsAGVHQc571QEEjuxk0x4CmYwDjT5cv_vUbZm3sZK4VSeWPYBTghhx-ziNqtgaAgCyEALw_wcB:G:s&s_kwcid=AL!4422!3!561348326849!e!!g!!amazon%20aws&all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all

Modelo de Despliegue


Usuario
Interfaz Ajax
Servicio Web
Aplicacion Apache Docker, Utilidades
Servicio Pinacoteca Diego Rivera
Web Service, Acceso a BD
Servicio Biblioteca Carlos Fuentes
Web Service, Acceso a BD
Servicio Ágora de ciudad de Xalapa
Web Service, Acceso a BD
BD Diego Rivera
BD Diego Rivera
BD Diego Rivera

FUNCIONAMIENTO DEL SISTEMA SOAP BIBLIOTECA “CARLOS FUENTES”

El primer sistema funcional es la Biblioteca “Carlos Fuentes” la cual se trabajó en su totalidad con SOAP y se montó en una imagen en Heroku, la cual nos permite hacer uso directo del sistema.

CONTIENE

Reservaciones:

Esto consiste en administrar y tener un registro de la información referente a las reservaciones de la biblioteca. En la solución se contempló:

- Registrar Reservaciones
- Modificar Reservaciones
- Buscar Reservaciones
- Eliminar Reservaciones

Consideraciones completas que ayudan a llevar un orden y una administración adecuada de las reservaciones, permitiendo a los usuarios tener más confiabilidad y credibilidad en que serán respetadas y respaldadas sus reservaciones registradas.

Servicios

La biblioteca cuenta con diversos servicios y por ende, requieren de un orden y registro correcto, para ello se implementó como solución:

- Listar Servicios
- Agregar Servicios
- Eliminar Servicios

Esto le permite a los administradores poder conocer y manejar cada uno de los servicios funcionales dentro de la biblioteca.

Registro

Actualmente la biblioteca recibe constantemente a nuevos visitantes recurrentes, para poder tener un registro de la cantidad de visitantes que ingresan a la biblioteca cada determinado tiempo se planteó la solución utilizando el manejó de:

- Registrar Visitantes

Que ayuda a los administradores a saber cuántos visitantes y en qué fechas son más comunes.

ENDPOINTS

RESERVACIONES

RegistrarReservacionesRequest

```
public RegistrarReservacionesResponse reservar(@RequestPayload
RegistrarReservacionesRequest nombre){
        RegistrarReservacionesResponse respuesta = new
RegistrarReservacionesResponse();
        respuesta.setRespuesta("Hola  " + nombre.getNombre());
        Reservacion s = new Reservacion();
        s.setNombre(nombre.getNombre());
            s.setConcepto(nombre.getConcepto());
            s.setFecha(nombre.getFecha());
            s.setHoraInicio(nombre.getHoraInicio());
            s.setHoraFin(nombre.getHoraFin());
            s.setTiempo(nombre.getTiempo());
        Ireservaciones.save(s);
        return respuesta;
    }
```

Recibe los parámetros de concepto, Fecha, Hora de inicio y fin y tiempo total para la reservación del lugar.

BuscarReservacionesRequest

```
public BuscarReservacionesResponse buscarReservaciones(){
        BuscarReservacionesResponse respuesta = new
BuscarReservacionesResponse();

        List<Reservacion> listreservaciones = (List<Reservacion>)
Ireservaciones.findAll();
        BuscarReservacionesResponse.Reservacion s = new
BuscarReservacionesResponse.Reservacion();
        for(Reservacion r : listreservaciones){
            s = new BuscarReservacionesResponse.Reservacion();
            s.setId(r.getId());
            s.setNombre(r.getNombre());
            s.setConcepto(r.getConcepto());
            s.setFecha(r.getFecha());
            s.setHoraInicio(r.getHoraInicio());
```

```
            s.setHoraFin(r.getHoraFin());
            s.setTiempo(r.getTiempo());
            respuesta.getReservacion().add(s);
        }
        return respuesta;
    }
```

Se muestran las reservaciones realizadas.

BorrarReservacionesRequest

```
@PayloadRoot(namespace = "https://Biblioteca.mx/Biblioteca",
localPart = "BorrarReservacionesRequest")
    @ResponsePayload
    public BorrarReservacionesResponse
borrarReservaciones(@RequestPayload BorrarReservacionesRequest
nombre){
        BorrarReservacionesResponse respuesta = new
BorrarReservacionesResponse();
        Ireservaciones.deleteById(nombre.getId());
        respuesta.setRespuesta("Elemento Eliminado");
        return respuesta;
    }
```

Se introduce el ID de la reservación que se desea eliminar

Registro de Visitantes

RegistrarVisitantesRequest

```
public RegistrarVisitantesResponse
registrarVisitantes(@RequestPayload RegistrarVisitantesRequest
nombre){
        RegistrarVisitantesResponse respuesta = new
RegistrarVisitantesResponse();
        Registro s = new Registro();
        s.setNombre(nombre.getNombre());
        s.setFecha(nombre.getFechaDdMmAa());
        s.setParejas(nombre.getParejas());
        Iregistro.save(s);
        return respuesta;
```

```
    }
```

Se recibe el nombre, fecha y las personas que acompañan al representante.

SERVICIOS

AgregarServicioRequest

```
public AgregarServicioResponse agregarActividad(@RequestPayload
AgregarServicioRequest agregar){
        AgregarServicioResponse respuesta = new
AgregarServicioResponse();
        respuesta.setRespuesta("SERVICIO ANOTADO EXITOSAMENTE");
        Servicios servicio = new Servicios();
        servicio.setNombre(agregar.getNombre());
        servicio.setMotivo(agregar.getMotivo());
        servicio.setTiempo(agregar.getTiempo());
        servicio.setFecha(agregar.getFecha());
        iservicios.save(servicio);
        return respuesta;
       }
```

El registro de servicios recibe un nombre, motivo (que es la razón para la que está siendo utilizado ese servicio), el tiempo que va a ser utilizado y la fecha.

ListarServicioRequest

```
public ListarServicioResponse ListarTareas(){
       ListarServicioResponse respuesta = new
ListarServicioResponse();
        Iterable<Servicios> lista = iservicios.findAll();

        for (Servicios servicio : lista) {
           ListarServicioResponse.Servicio a = new
ListarServicioResponse.Servicio();
            a.setNombre(servicio.getNombre());
            a.setId(servicio.getId());
            a.setMotivo(servicio.getMotivo());
            a.setTiempo(servicio.getTiempo());
            a.setFecha(servicio.getFecha());
```

```
            respuesta.getServicio().add(a);
        }
        return respuesta;
    }
```

Aquí se pueden consultar todos los registros de uso de servicios que ha habido.

EliminarServicioRequest

```
public EliminarServicioResponse eliminarActividad(@RequestPayload
EliminarServicioRequest eliminar){
         EliminarServicioResponse respuesta = new
EliminarServicioResponse();
         iservicios.deleteById(eliminar.getId());
         respuesta.setRespuesta("ACTIVIDAD ELIMINADA EXITOSAMENTE");
         return respuesta;
     }
```

Aquí se pueden eliminar todos los registros de uso de servicios que ha habido.

Forma de ejecución de los contenedores

Contamos con un contenedor en docker, el cual contiene el servicio de biblioteca contenerizado.
docker pull aridaioza/biblioteca
docker run -it --rm -p 8080:8080 aridaioza/biblioteca

+ /ws/biblioteca.wsdl

DOCKER FILE - CONTENEDOR EN DOCKER

En este dockerfile hacemos uso de la imagen del profesor para partir de ahí y mandar la instrucción con el documento scrip.sh de comenzar a correr la base de datos y el jar previamente introducidos en la carpeta /app de la imagen, la base dee datos hace uso del documento script.sql, instrucciones:

from ubuntu:20.04
workdir /app
expose 8080
cmd ["/app/script.sh"]
add ServiciosXLP/Biblioteca-0.0.1-SNAPSHOT.jar /app
run apt-get update

```
run apt-get install -y openjdk-8-jdk
RUN apt-get install -y mariadb-server
add script.sql /app/script.sql
RUN chmod 755 /app/script.sql
add script.sh /app/script.sh
run chmod 755 /app/script.sh
RUN /etc/init.d/mysql start; mysql < /app/script.sql
```

DOCKER FILE HEROKU

```
from rrojano/jdk8
workdir /app

cmd ["/app/script.sh"]
add Biblioteca-0.0.1-SNAPSHOT.jar /app/Biblioteca-0.0.1-SNAPSHOT.jar
run apt-get update
RUN apt-get install -y mariadb-server
add Script.sql /app/script.sql
RUN chmod 755 /app/script.sql
add Script.sh /app/script.sh
run chmod 755 /app/script.sh
RUN /etc/init.d/mysql start; mysql < /app/script.sql
```

FUNCIONAMIENTO DEL SISTEMA SOAP PINACOTECA

El segundo servicio que se implementó fue la pinacoteca “Diego Rivera” este sistema funcional que se implementó totalmente en soap y se montó en una imagen en heroku para hacer uso de esta imagen en el momento que lo necesitemos

Contiene

EVENTOS

Para tener un control idóneo sobre los eventos en la pinacoteca se contemplaron 2 servicios que serán utilizados para el correcto funcionamiento los cuales son:

- Agregar Eventos
- Listar Eventos

Donde se espera que con el buen manejo de estos servicios no se tenga problemas al momento de tener un control sobre los eventos que se van a realizar en la pinacoteca

ARTISTAS

Se contempló únicamente un servicio en la pinacoteca relacionado con los artistas, ya que se espera que se agreguen las obras a este lugar.

- Registrar Artistas

Con esto se espera tener un conocimiento de los artistas y las obras que se encuentran en este lugar

Visitantes

Se tiene planeado únicamente tener un historial de las personas que han visitado el lugar con el único objetivo de tener un control de cuantas personas van y quienes han accedido, por lo que se planeó el servicio de:

- Registrar Visitantes

Donde en caso de ser necesario se tenga un historial de las personas que entraron a este lugar

ENDPOINTS

EVENTOS

AgregarEventoRequest

```
public AgregarEventoResponse agregarActividad(@RequestPayload
AgregarEventoRequest agregar){
       AgregarEventoResponse respuesta = new AgregarEventoResponse();
        respuesta.setRespuesta("EVENTO ANOTADO EXITOSAMENTE");
        Evento evento = new Evento();
        evento.setNombre(agregar.getNombre());
        evento.setDescripcion(agregar.getDescripcion());
        evento.setFecha(agregar.getFecha());
        evento.setHora(agregar.getHora());
        evento.setCosto(agregar.getCosto());
        ieventos.save(evento);
        return respuesta;
    }
```

Este método recibe el nombre, una descripción (sobre qué o para qué es el evento), fecha, hora y costo de la entrada al evento. Devuelve una liga de éxito si se ha agregado exitosamente.

ListarEventoRequest

```
public ListarEventoResponse ListarTareas(){
       ListarEventoResponse respuesta = new ListarEventoResponse();
        Iterable<Evento> lista = ieventos.findAll();

        for (Evento evento : lista) {
          ListarEventoResponse.Evento a = new ListarEventoResponse.Evento();
           a.setNombre(evento.getNombre());
           a.setId(evento.getId());
           a.setDescripcion(evento.getDescripcion());
           a.setHora(evento.getHora());
           a.setCosto(evento.getCosto());
           a.setFecha(evento.getFecha());
           respuesta.getEvento().add(a);
        }
        return respuesta;
    }
```

Este método devuelve la lista de los eventos que se llevarán a cabo en la Pinacoteca, no necesita ningún parámetro de entrada.

ARTISTAS

RegistrarArtistaRequest

```
public RegistrarArtistaResponse agregarActividad(@RequestPayload
RegistrarArtistaRequest agregar){
      RegistrarArtistaResponse respuesta = new RegistrarArtistaResponse();
       respuesta.setRespuesta("ARTISTA ANOTADO EXITOSAMENTE");
       Artista artista = new Artista();
       artista.setNombre(agregar.getNombre());
       artista.setApellidos(agregar.getApellidos());
       iartistas.save(artista);
       return respuesta;
   }
```

Este método recibe el nombre y apellidos del artista que se presentará en la Pinacoteca. Devuelve una liga de éxito si se ha agregado el artista exitosamente.

VISITANTES

RegistrarVisitantesRequest

```
public RegistrarVisitantesResponse agregarActividad(@RequestPayload
RegistrarVisitantesRequest agregar){
      RegistrarVisitantesResponse respuesta = new
RegistrarVisitantesResponse();
       respuesta.setRespuesta("VISITANTE ANOTADO EXITOSAMENTE");
       Visitante visitante = new Visitante();
       visitante.setNombre(agregar.getNombre());
       visitante.setFecha(agregar.getFecha());
       visitante.setAcompañantes(agregar.getAcompañantes());
       ivisitantes.save(visitante);
       return respuesta;
   }
```

Este método recibe el nombre, fecha y número de acompañantes del visitante que ha visitado la Pinacoteca. Devuelve una liga de éxito si se ha agregado el visitante exitosamente.

DOCKERFILE HEROKU

En este dockerfile hacemos uso de la imagen del profesor para partir de ahí y mandar la instrucción con el documento scrip.sh de comenzar a correr la base de datos y el jar previamente introducidos en la carpeta /app de la imagen, la base dee datos hace uso del documento script.sql, instrucciones:

```
from rrojano/jdk8
workdir /app
cmd ["/app/script.sh"]
add Pinacoteca-0.0.1-SNAPSHOT.jar /app/Pinacoteca-0.0.1-SNAPSHOT.jar
run apt-get update
RUN apt-get install -y mariadb-server
add Script.sql /app/script.sql
RUN chmod 755 /app/script.sql
add Script.sh /app/script.sh
run chmod 755 /app/script.sh
RUN /etc/init.d/mysql start; mysql < /app/script.sql
```

Nota: los documentos script.sql y scrip.sh se podrán encontrar dentro de la carpeta de “Desplegar Pinacoteca” en el repositorio de github.

FUNCIONAMIENTO DEL SISTEMA SOAP ÁGORA DE LA CIUDAD

Él último sistema planteado en la solución, fue el Ágora de la ciudad de Xalapa. Este sistema se basó en su totalidad con SOAP y al igual que los servicios anteriores, se montó en una imagen de Heroku. Con este sistema se estaría completando la solución propuesta para la problemática planteada en un inicio.

CONTIENE
Evento
En esta sección el administrador puede, con ayuda del sistema, monitorear y manejar la información referente a los eventos que se realizan en el sitio, para ello necesita:

- Agregar evento
- Listar evento
- Modificar evento

Cada uno recaba la información requerida para poder llevar un control adecuado del servicio que se brinda.
Visitantes
El Ágora debe llevar un registro y control de los visitantes que ingresan para ello ocupa lo siguiente:

Registrar visitantes
Listar visitantes

Los cuales permiten que el administrador pueda visualizar la cantidad de visitantes que llegan cada determinado tiempo.
Películas
Finalmente el Ágora ofrece un servicio de películas, es por eso que se implementó:

Registrar películas

El cual ayuda al registro de las películas a proyectar.

ENDPOINTS

EVENTOS

AgregarEventoRequest

```
public AgregarEventoResponse agregarActividad(@RequestPayload
AgregarEventoRequest agregar){
      AgregarEventoResponse respuesta = new AgregarEventoResponse();
       respuesta.setRespuesta("EVENTO ANOTADO EXITOSAMENTE");
       Evento evento = new Evento();
       evento.setNombre(agregar.getNombre());
       evento.setDescripcion(agregar.getDescripcion());
       evento.setFecha(agregar.getFecha());
       evento.setHora(agregar.getHora());
       evento.setCosto(agregar.getCosto());
       ieventos.save(evento);
       return respuesta;
   }
```

Este método recibe el nombre, una descripción (sobre qué o para qué es el evento), fecha, hora y costo de la entrada al evento. Devuelve una liga de éxito si se ha agregado exitosamente.

ListarEventoRequest

```
public ListarEventoResponse ListarTareas(){
      ListarEventoResponse respuesta = new ListarEventoResponse();
       Iterable<Evento> lista = ieventos.findAll();

       for (Evento evento : lista) {
          ListarEventoResponse.Evento a = new ListarEventoResponse.Evento();
           a.setNombre(evento.getNombre());
           a.setId(evento.getId());
           a.setDescripcion(evento.getDescripcion());
           a.setHora(evento.getHora());
           a.setCosto(evento.getCosto());
           a.setFecha(evento.getFecha());
```

```
            respuesta.getEvento().add(a);
        }
        return respuesta;
    }
```
Este método devuelve la lista de los eventos que se llevarán a cabo en la Pinacoteca, no necesita ningún parámetro de entrada.

PELÍCULAS

RegistrarPeliculaRequest

```
public RegistrarPeliculaResponse agregarActividad(@RequestPayload
RegistrarPeliculaRequest agregar){
        RegistrarPeliculaResponse respuesta = new RegistrarPeliculaResponse();
        respuesta.setRespuesta("PLEÍCULA ANOTADA EXITOSAMENTE");
        Pelicula pelicula = new Pelicula();
        pelicula.setNombre(agregar.getNombre());
        pelicula.setFecha(agregar.getFecha());
        pelicula.setHora(agregar.getHora());
        ipeliculas.save(pelicula);
        return respuesta;
    }
```
Este método recibe el nombre, fecha y hora de proyección de la película que se presentará en el Ágora. Devuelve una liga de éxito si se ha agregado la película se ha registrado exitosamente.

VISITANTES

RegistrarVisitantesRequest

```
public RegistrarVisitantesResponse agregarActividad(@RequestPayload
RegistrarVisitantesRequest agregar){
        RegistrarVisitantesResponse respuesta = new
RegistrarVisitantesResponse();
        respuesta.setRespuesta("VISITANTE ANOTADO EXITOSAMENTE");
        Visitante visitante = new Visitante();
        visitante.setNombre(agregar.getNombre());
        visitante.setFecha(agregar.getFecha());
        visitante.setAcompañantes(agregar.getAcompañantes());
        ivisitantes.save(visitante);
        return respuesta;
    }
```
Este método recibe el nombre, fecha y número de acompañantes del visitante que ha visitado el Ágora. Devuelve una liga de éxito si se ha agregado el visitante exitosamente.

ListarVisitantesRequest

```
public ListarVisitantesResponse ListarVisitas(){
      ListarVisitantesResponse respuesta = new ListarVisitantesResponse();
       Iterable<Visitante> lista = ivisitantes.findAll();

       for (Visitante visitante : lista) {
         ListarVisitantesResponse.Visitante a = new
ListarVisitantesResponse.Visitante();
          a.setId(visitante.getId());
          a.setNombre(visitante.getNombre());
          a.setFecha(visitante.getFecha());
          a.setAcompañantes(visitante.getAcompañantes());
          respuesta.getVisitante().add(a);
       }
       return respuesta;
   }
```

Este método no recibe ningún parámetro, simplemente debe haber visitantes registrados del Ágora para generar una lista de resultados a partir de ellos.

DOCKERFILE HEROKU

En este dockerfile hacemos uso de la imagen del profesor para partir de ahí y mandar la instrucción con el documento scrip.sh de comenzar a correr la base de datos y el jar previamente introducidos en la carpeta /app de la imagen, la base dee datos hace uso del documento script.sql, instrucciones:

```
from rrojano/jdk8
workdir /app

cmd ["/app/script.sh"]
add Agora-0.0.1-SNAPSHOT.jar /app/Agora-0.0.1-SNAPSHOT.jar
run apt-get update
RUN apt-get install -y mariadb-server
add Script.sql /app/script.sql
RUN chmod 755 /app/script.sql
add Script.sh /app/script.sh
run chmod 755 /app/script.sh
RUN /etc/init.d/mysql start; mysql < /app/script.sql
```

Nota: los documentos script.sql y scrip.sh se podrán encontrar dentro de la carpeta de “Desplegar Agora” en el repositorio de github.

FUNCIONAMIENTO DEL SISTEMA REST “Control de visitantes: General”

ENDPOINTS

VISITANTES

@GetMapping("/registrarVisitante")

public String saludarM(@RequestParam(name="nombre", defaultValue = "Sin nombre!!") String nombre,@RequestParam(name="team") String team) {
if(nombre!=null){
Visitante s = new Visitante(nombre, team);
ivistantes.save(s);
saludos.add(s);
return "Hola "+nombre + " - Nombre Registrado";
}else{
return "No se registro ningun Nombre";
}
 	
}
 	
Este método sí recibe parámetros, debemos introducir el nombre del visitante y el número de personas que lo acompañan para posteriormente, poder obtener un resgitro del visitante.

@GetMapping("/buscarVisitante")

public Iterable<Visitante> buscarSaludo() {
Iterable<Visitante> lista = ivistantes.findAll();
return lista;
}
 	
No recibe ningún parámetro, simplemente debe registrarse un visitante antes para poder consultar la lista..

@GetMapping("/eliminarVisitante")

public String elimarSaludo(@RequestParam(name="nombre", defaultValue = "Sin nombre!!") String nombre) {
if(nombre!=null){
ivistantes.deleteById(nombre);
return "Nommbre = "+nombre + " - No existe";

```
    }else{
        return "No se elimino ningun Nombre";
    }
  }
```

Para eliminar un visitante es necesario introducir como parámetro el nombre que se desea eliminar, esto para mantener un control con los registros.

```
@GetMapping("/modificarVisitante")
```

```
public String modificarSaludo(@RequestParam(name="id", defaultValue = "Sin nombre!!") String id, @RequestParam(name="nombre",defaultValue = "Sin nombre!!") String nombre, @RequestParam(name="team", defaultValue = "0") String team) {
    if(id!=null){
        if(nombre!=null){
            if(team!=null){
                Optional<Visitante> v = ivistantes.findById(id);
                Visitante s = v.get();
                s.setNombre(nombre);
                s.setTeam(team);
                ivistantes.save(s);
                return "Todo Cambiado";
            }else{
                    Optional<Visitante> v = ivistantes.findById(id);
                    Visitante s = v.get();
                    s.setNombre(nombre);
                    ivistantes.save(s);
                    return "Nombre Cambiado";
            }
        }else{
            if(team!=null){
                Optional<Visitante> v = ivistantes.findById(id);
                Visitante s = v.get();
                s.setTeam(team);
                ivistantes.save(s);
                return "Team Cambiado";
            }else{
                return "Nada cambiado";
            }
        }
    }else{
        return "No se modifico ningun registro";
    }

  }
```

Este método modifica los elementos que introduzcas como parámetros de

@GetMapping("/modificarVisitante")
acuerdo a su ID, recibe el id, nombre y número de acompañantes y los edita en caso de que hubiera una equivocación en el registro.

DOCKERFILE HEROKU

En este dockerfile hacemos uso de la imagen del profesor para partir de ahí y mandar la instrucción con el documento scrip.sh de comenzar a correr la base de datos y el jar previamente introducidos en la carpeta /app de la imagen, la base dee datos hace uso del documento script.sql, instrucciones:

```
from rrojano/jdk8
workdir /app

cmd ["/app/script.sh"]
add RegistroVisitantes-0.0.1-SNAPSHOT.jar
/app/RegistroVisitantes-0.0.1-SNAPSHOT.jar
run apt-get update
RUN apt-get install -y mariadb-server
add Script.sql /app/script.sql
RUN chmod 755 /app/script.sql
add Script.sh /app/script.sh
run chmod 755 /app/script.sh
RUN /etc/init.d/mysql start; mysql < /app/script.sql
```

Nota: los documentos script.sql y scrip.sh se podrán encontrar dentro de la carpeta de “Desplegar Registro” en el repositorio de github.

LINK DE HEROKU

Servicio de Biblioteca

https://biblioteca-tis.herokuapp.com/ws/biblioteca.wsdl

Servicio de Pinacoteca

https://pinacoteca-tis.herokuapp.com/ws/pinacoteca.wsdl

Servicio de Ágora

https://agora-tis.herokuapp.com/ws/agora.wsdl

Servicio de Registro de Visitantes

https://registrovisitantes-tis.herokuapp.com/registrarVisitante?nombre=Joaquin&team=

LINK DE GITHUB

https://github.com/JoaquinGA01/ServiciosXLP