Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Gestión de calendarios para un servicio hospitalario

Autora: Carmen Cohen Calvo

Tutora: Dra. Isabel Román Martínez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo de Fin de Grado
Ingeniería de las tecnologías de Telecomunicación

Gestión de calendarios para un servicio hospitalario

Autor:
Carmen Cohen Calvo

Tutor:
Dra. Isabel Román Martínez
Profesora colaboradora

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Gestión de calendarios para un servicio hospitalario

Autora: Carmen Cohen Calvo

Tutora: Dra. Isabel Román Martínez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El secretario del Tribunal

A mi familia

A mis amigos

Agradecimientos

A mi familia y amigos por el apoyo recibido.

Resumen

El objetivo principal de este proyecto es diseñar e implementar un servicio de calendarios que permita publicar los turnos de los profesionales de un servicio hospitalario. Además, se generarán los calendarios individuales de los doctores involucrados, que les serán enviados por correo electrónico.

La solución ha sido diseñada en el entorno Spring y siguiendo el estándar iCalendar. Para ello se han utilizado las librerías ical4j y Javaxmail.

Este proyecto se basa en el trabajo realizado por Miguel Ángel González-Alorda Cantero titulado “Servicio para la gestión de actividades asistenciales complementarias” [1] y supone una versión de este, con las funciones de gestión de calendarios incorporadas.

Abstract

The main objective of this project is to design and implement a calendar service that allows publishing the shifts of the professionals of a hospital service. In addition, the individual calendars of the doctors involved will be generated and sent to them by email.

In addition, it has been simplifying the deployment method to facilitate continuous improvement of the project.

The solution has been designed in the Spring environment and following the iCalendar standard. For this, the ical4j and Javaxmail libraries have been used.

This project is based on the work conducted by Miguel Ángel González-Alorda Cantero entitled "Service for the management of complementary care activities" and is a version of this, with the functions of calendar management incorporated. [1]

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

FRAGMENTOS DE CÓDIGO

GLOSARIO

API	Application Programming Interfaces
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
MIME	Multipurpose Internet Mail Extensions
POP3	Post Office Protocol
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
URL	Uniform Resource Locator

1 INTRODUCCIÓN

El objetivo principal de este proyecto es implementar un servicio de calendarios que complemente el sistema de asignación de turnos creado en el trabajo *Servicio para la gestión de actividades asistenciales complementarias* [1]. En él se abarcaba el problema de la creación y manejo de los turnos y las actividades de los doctores del servicio de medicina interna del Hospital Universitario Virgen Macarena.

1.1 DESCRIPCIÓN DEL PROBLEMA

Programar y gestionar las actividades de los doctores tiene dos partes diferentes. La primera es programar los turnos. Esto es, asignar un conjunto de doctores a cada día y cada tipo de actividad del mes, cumpliendo con algunos requisitos como preferencia en tipo de turno, número de turnos de cada tipo y días festivos. La segunda parte sería, después de que se hayan programado y asignado los turnos, la gestión de estos. Esto es, notificar a los doctores de sus turnos, y facilitarles cambiarlos entre sí.

1.1.1 Problema de asignación de horarios

El problema de asignación de horarios consiste en asignar turnos a los doctores del hospital, pero cumpliendo con las restricciones impuestas.
En particular, en el problema que nos ocupa actualmente, cada día puede tener dos tipos diferentes de activades asistenciales complementarias:

- Jornadas complementarias: turnos que ocurren periódicamente, e independientemente de cualquier otra restricción. En particular, corresponden a las guardias del doctor que comienzan a las 20.00 de un día determinado, y finalizará a las 8.00 horas del día siguiente.

- Continuidades asistenciales: cualquier otro turno que no se produzca periódicamente, están sujetos a diferentes restricciones. En particular, estos turnos representan los que comienzan a las 15.00 de un día determinado, y finalizan a las 20.00 horas del mismo día. Se divide en dos subtipos, las consultas y las continuidades asistenciales[1].

La asignación y manejo de turnos y doctores es el problema que se ataja en el proyecto "*Servicio para la gestión de actividades asistenciales complementarias".*

1.1.2 Problema de gestión de horarios

El problema de gestión consta de dos partes diferentes. La primera es notificar a los doctores sus turnos, y la segunda es facilitar que los doctores cambien sus turnos entre sí. Ambos ocurrirán después de que se haya completado la programación de un mes determinado.
Notificar a los doctores sus turnos permite que los doctores siempre puedan saber, para cada día del mes, quién cubrirá los turnos.
Con respecto a la segunda parte del problema, si un doctor desea cambiar uno de sus turnos, tiene que seguir estos pasos:

1. El doctor dispuesto a cambiar un turno debe encontrar otro doctor que lo tome.
2. Después de que dos doctores han acordado cambiar un turno, tienen que hacer una solicitud para que el cambio sea revisado por el gerente.
3. El gerente puede aceptar o rechazar la solicitud.
4. En caso de que la solicitud sea aceptada, todos los doctores deben ser notificados del cambio. Especialmente los doctores involucrados en él.

[1] Los turnos no periódicos son en su mayoría continuidades asistenciales de ahí que compartan el nombre.

1.2 PUNTO DE PARTIDA

Actualmente se dispone de una solución para dar soporte a la primera tarea. Esta solución se basa en una aplicación web que le facilita al responsable de la asignación la automatización de esta tarea. Esta aplicación es cliente de un servicio REST, también desarrollado para el caso que nos ocupa. La aplicación web se comunica con el servicio REST a través de peticiones HTTP. El gestor interactúa con el sistema a través del navegador web.

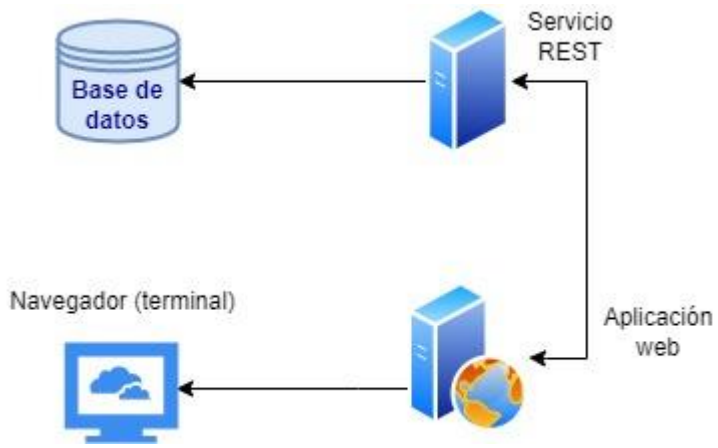


Ilustración 1: Estructura del servicio

A través de la web el gestor de turnos accede a un servicio web con su usuario y contraseña. Aquí puede añadir a los doctores, sus datos personales y sus preferencias en cuanto a turnos. Además, desde aquí se solicita a través del servicio REST la generación del calendario del mes.

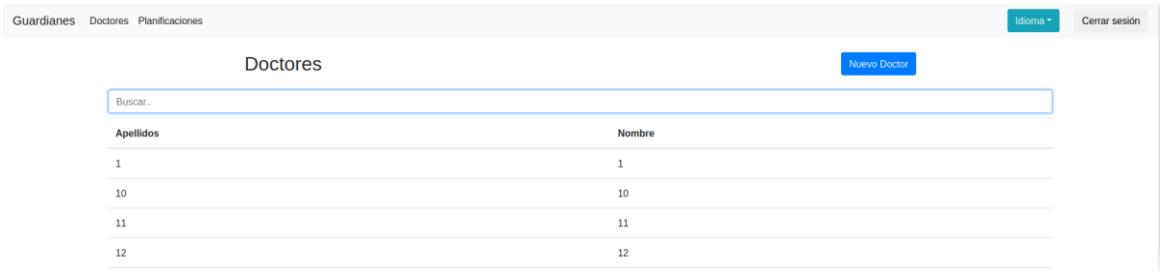


Ilustración 2: Listado de Doctores en cliente Web

Para generar el calendario del mes se selecciona generar planificación, se elige el mes deseado y se seleccionan los días festivos. Una vez le dé a aceptar el sistema genera el calendario de turnos según las preferencias de los doctores y las necesidades del servicio y sus restricciones.

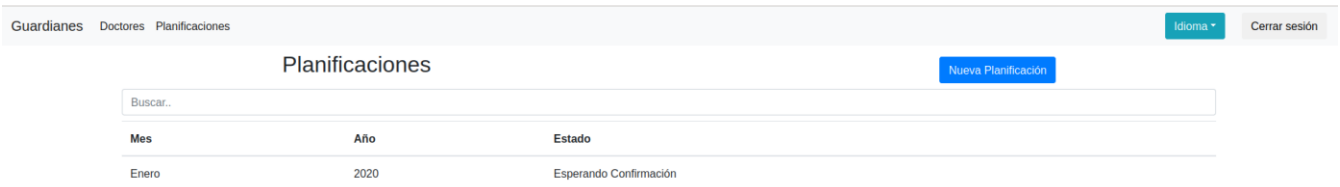


Ilustración 3: Listado de planificaciones en el cliente Web

El gestor debe revisar que el calendario sea correcto. Una vez confirmado puede descargar la planificación en formato Excel.

Ilustración 4: Ejemplo de planificación en el cliente Web

Para compartir la planificación el gestor tendría que enviarla manualmente por correo electrónico o colgarla en un tablón. Además, los cambios deberá gestionarlos el gestor y notificarlos a los doctores. En la actualidad los doctores no cuentan con la posibilidad de consultar el calendario en el servicio web, la única manera de consultarlo es que el gestor lo comparta con ellos por otra vía.

1	15	16				
2	17	18	17	18	20	13
3	19	20	19	20	9	13
4	2	1	2	17		
5	4	3	10	22	21	17
6	6	5	6	21		
7	7	8				
8	10	9				
9	12	11	11	22		
10	14	13	14	13		

Ilustración 5: Fragmento planificación Excel

1.3 ALCANCE DEL PROYECTO.

La solución propuesta en este trabajo es automatizar la notificación de los calendarios a los doctores. El gestor de los turnos no necesitará compartir los calendarios con los doctores una vez confirmado el calendario, se comunicarán automáticamente.

El sistema debe permitir que los cambios que haga el gestor en el calendario de actividades asistenciales se transmitan inmediatamente a los servicios de calendario personales de los doctores.

Además, los doctores deben disponer del calendario de actividades asistenciales oficial, actualizado, en su servicio de calendario personal

Los doctores dispondrán en su servicio de calendario personal de un calendario personalizado, siempre actualizado, con sus actividades asistenciales asignadas oficialmente.

Por otro lado, el proyecto abarca una optimización del despliegue del servicio REST para facilitar una entrega continua. Además, el servicio REST se extiende para dar soporte a un bot de Telegram que facilita el intercambio de turnos, aunque eso ya está fuera del alcance de este TFG.

2 ESTADO DE LA TECNOLOGÍA

En este apartado se explicarán las tecnologías involucradas para poder entender el desarrollo de este proyecto.

2.1 JAVA

Java es un lenguaje de programación de alto nivel orientado a objetos [2]. Para mantener la versión utilizada en el trabajo del que se parte, se usará Java 8, cuya documentación puede encontrarse en [3].

En el proyecto se utiliza la API JavaMail [4].

2.1.1 JavaMail

JavaMail es una API que facilita el envío de correo electrónico utilizando el lenguaje Java. La versión utilizada es la 1.6.2 cuya documentación se encuentra en [5].

Esta API soporta los distintos protocolos de servicio de correos electrónico: SMTP, IMAP, pop3, mbox, pop3remote.

```java
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SendEmail
{
 public static void main(String [] args){
      String to = "sonoojaiswal1988@gmail.com";//change accordingly
      String from = "sonoojaiswal1987@gmail.com";//change accordingly
      String host = "localhost";//or IP address

     //Get the session object
      Properties properties = System.getProperties();
      properties.setProperty("mail.smtp.host", host);
      Session session = Session.getDefaultInstance(properties);

     //compose the message
      try{
         MimeMessage message = new MimeMessage(session);
         message.setFrom(new InternetAddress(from));
         message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));
         message.setSubject("Ping");
         message.setText("Hello, this is example of sending email  ");

         // Send message
         Transport.send(message);
         System.out.println("message sent successfully....");

      }catch (MessagingException mex) {mex.printStackTrace();}
   }
```

Código 1: Envío de correo electrónico

En el ejemplo [6] se muestran las principales clases que se van a utilizar de esta librería:

- `public class MimeMessage`: Esta clase representa un mensaje de correo electrónico de estilo MIME. Para enviar un correo electrónico con "partes", por ejemplo, un archivo adjunto, se usa la clase MimeBodyPart, instanciando un objeto por cada parte. En el constructor de esta clase se le añade las propiedades a la cabecera del mensaje a través de la sesión que luego utilizará la clase *transport* para enviar el mensaje.
- `public final class Session`: Esta clase representa una sesión de correo. Es en esta clase donde se agregan los parámetros de autentificación como usuario y contraseña. Incorpora las propiedades y los valores predeterminados utilizados por las API de correo.
- `public abstract class Transport`: es una clase abstracta que modela el transporte de mensajes.
  El método send envía el mensaje a todas las direcciones de destinatario especificadas en el mensaje, utilizando el método de transporte adecuado para cada dirección.

2.2 SPRING

Spring [7] es un framework libre para desarrollar aplicaciones en Java que favorece una estructura modular. Una de las características es la posibilidad de implementar los distintos estereotipos de Java a través de anotaciones en la definición de las clases.

Gracias a esto la implementación del patrón modelo-vista-controlador es sencilla.

Como se explica en [8] 'El patrón MVC (Modelo, Vista, Controlador) [9] es un esquema de arquitectura por capas muy utilizado en el desarrollo de software basado en aplicaciones web. El modelo (M) controla todo lo relacionado con los datos, la vista (V) lo relacionado con las interfaces de usuario y el controlador (C) se encarga de la manipulación del M para mostrar información en la V'.

Ejemplos de anotaciones utilizadas en Spring son @Controller o @Entity.

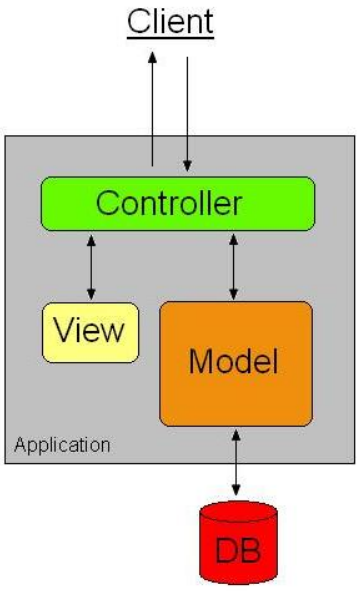


Ilustración 6: Patrón Modelo-Vista-Controlador

Además de este patrón se implementa la capa de servicio a través de la notación @Service. [10]

Como se define en [11] 'Una capa de servicio define el límite de una aplicación y el conjunto de operaciones disponibles desde la perspectiva de la interfaz de capas de cliente y coordina la respuesta de la aplicación a cada operación.' Es la capa que contiene la lógica del proceso.

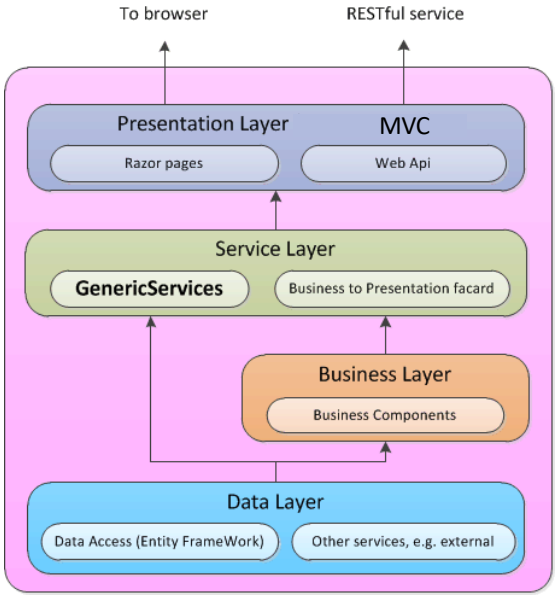


Ilustración 7: Capa de servicio

Las dependencias entre objetos se especifican en Spring con la anotación @Autowired que especifica las inyecciones de dependencias.

2.3 ICALENDAR

iCalendar es un formato de datos estándar creado por el IETF para la transferencia de calendarios entre sistemas informáticos, independientemente de cualquier servicio de calendario o protocolo particular.

Queda descrito en la RFC5545 [12]. Las extensiones más usadas en los archivos iCalendar suelen ser “. ical" o ".ics".

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//ZContent.net//Zap Calendar 1.0//EN
CALSCALE:GREGORIAN
METHOD:PUBLISH
        BEGIN:VEVENT
        SUMMARY:Abraham Lincoln
        UID:c7614cff-3549-4a00-9152-d25cc1fe077d
        SEQUENCE:0
        STATUS:CONFIRMED
        TRANSP:TRANSPARENT
        RRULE:FREQ=YEARLY;INTERVAL=1;BYMONTH=2;BYMONTHDAY=12
        DTSTART:20080212
        DTEND:20080213
        DTSTAMP:20150421T141403
        CATEGORIES:U.S. Presidents,Civil War People
        LOCATION:Hodgenville\, Kentucky
        GEO:37.5739497;-85.7399606
        DESCRIPTION:Born February 12\, 1809\nSixteenth President (1861-1865)\n\n\n
         \nhttp://AmericanHistoryCalendar.com
        URL:http://americanhistorycalendar.com/peoplecalendar/1,328-abraham-lincoln
        END:VEVENT
END:VCALENDAR
```

Código 2: Formato de archivo iCalendar

En el ejemplo [13] mostrado en el código se puede identificar lo siguiente:

- Tanto los eventos como el calendario están delimitados por BEGIN y END;
- Un calendario cuenta con propiedades como:
    - Versión de iCal (VERSION)
    - Un identificador (PRODID)
    - El tipo de calendario como gregoriano, hindú, islámico… (CALSCALE)
    - Método del calendario: add, cancel, publish, request…
- Un evento cuenta con las siguientes propiedades entre otras:
    - Resumen/nombre del evento (SUMMARY)
    - Id único de identificación del evento, no puede repetirse en el calendario (UID)
    - Versión del evento (SEQUENCE)
    - Estado: confirmado, pendiente, cancelado (STATUS)
    - Fecha de inicio/final (DTSTART/DTEND)
    - Coordenadas GPS (GEO)

Para trabajar con el estándar en Java se puede usar la librería iCal4j [14].

2.3.1 iCal4j

iCal4j es una librería de Java que ayuda a manejar y crear calendarios, eventos y listas de tareas, siguiendo las pautas de la RFC5545 anteriormente comentadas.

Un calendario está formado por componentes (como eventos o recordatorios), estos componentes contienen propiedades (como la fecha o los asistentes) y, por último, las propiedades tienen parámetros (como el nombre o email en el caso de un asistente a un evento).

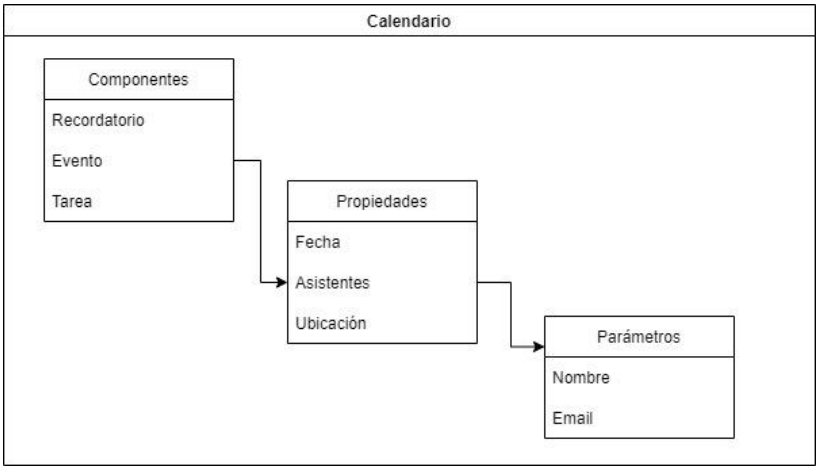


Ilustración 8: Estructura de calendario ical4j

```
Calendar calendar = new Calendar();
calendar.getProperties().add(new ProdId("-//Ben Fortuna//iCal4j 1.0//EN"));
calendar.getProperties().add(Version.VERSION_2_0);
calendar.getProperties().add(CalScale.GREGORIAN);

// Set date
java.util.Calendar startDate = new GregorianCalendar();
      startDate.add(java.util.Calendar.DAY_OF_MONTH, 25);
       startDate.set(java.util.Calendar.MONTH, java.util.Calendar.DECEMBER);

// initialise as an all-day event..
VEvent christmas = new VEvent(new Date(startDate.getTime()), "Christmas Day")
;

// Generate a UID for the event..
UidGenerator ug = new UidGenerator("1");
christmas.getProperties().add(ug.generateUid());

calendar.getComponents().add(christmas);
```

Código 3: Creación de calendario con un evento

En el ejemplo mostrado en el código 3 [15] vemos que para añadir propiedades al calendario que se acaba de instanciar es necesario obtener la lista de propiedades con el método getProperties () y posteriormente añadir la propiedad concatenando el método add(Property property).

En los eventos que duran todo el día no es necesario especificar la hora de finalización.

Se instancia un objeto de tipo VEvent cuyos parámetros iniciales son la fecha y el nombre. Se añade un identificador único en el calendario generado con la clase Uidgenerator. Por último, se añade el evento al calendario

2.4 CalDAV

CalDAV, especificado en la RFC 4791 [16], es un protocolo que permite manejar, acceder y compartir en internet los recursos de calendarios creados según el estándar iCalendar.

Para trabajar con CalDAV en Java se puede usar la librería CalDAV4j [17].

2.4.1 CalDAV4j

CalDAV4j es una librería de Java que implementa el protocolo CalDAV y se basa en la librería iCal4j para el procesado de la información del calendario. A través de peticiones HTTP permite la comunicación con el servidor que ofrece el servicio de calendario. Para ello se apoya en la clase HttpClient de Java. La documentación se encuentra en [18] .

2.4.2 DAViCal

DAViCal [19] es un servidor libre que implementa CalDAV y permite compartir y alojar calendarios.

Utiliza una base de datos PostgreSQL para almacenar la información de los calendarios.

2.5 Docker

Docker es un conjunto de herramientas que facilita la implantación de sistemas informáticos o aplicaciones a través de la virtualización basada en contenedores.

Es un sistema operativo para contenedores que virtualizan el sistema operativo de un servidor. Estos contenedores se gestionan a través de comandos sencillos que permiten iniciar, parar o reinstaurar dichos contenedores.

Los contenedores integran la aplicación y sus dependencias, pero comparten el kernel del sistema operativo con otros contenedores, que se ejecutan como procesos aislados en el sistema operativo anfitrión. En cambio, las máquinas virtuales integran la aplicación, las bibliotecas y/o los archivos binarios necesarios en un sistema operativo invitado completo. Esto lleva a que la virtualización basada en máquinas virtuales requiera más recursos que la inclusión en contenedores.

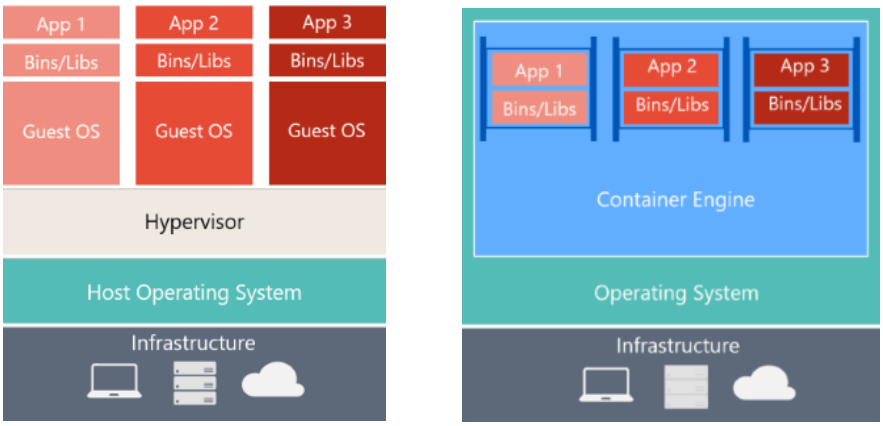


Ilustración 9: Diferencia entre máquina virtual y Docker [20]

El proceso para generar un contenedor se apoya en un DockerFile. Un DockerFile es un archivo de texto que contiene los comandos necesarios para crear una imagen.

Una imagen de Docker contiene la especificación de todo el sistema inicial en el que se va a basar un contenedor para su funcionamiento. Incluye: SO, sistema de ficheros, configuración, aplicaciones y sus dependencias, etc. Es una guía de instrucciones para construir un contenedor Docker.

Puede contener aplicaciones ejecutables, así como todos los recursos necesarios (volúmenes, configuración de red, dependencias…). Una vez ejecutamos esa imagen se crea una instancia del contenedor.

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
COPY target/*.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Código 4: Ejemplo DockerFile

En este mini Docker file [21] podemos ver los comandos principales:

- From: imagen base de la que se parte.
- Volume: crea un volumen que permite almacenar e intercambiar información con otros contendores.
- Copy: copia el archivo jar al volumen creado.
- Entrypoint: especifica el ejecutable que usará el contenedor al arrancar.

Además, existe docker-compose para desplegar aplicaciones multicontenedor a partir de archivos YAML. En él se relacionan contenedores entre sí y se pueden especificar propiedades de volúmenes, puertos u otros parámetros, al igual que con el DockerFile.

3 REQUISITOS

Este apartado incluye los requisitos del proyecto para el problema de la gestión de los calendarios.

3.1 ACTORES

Estos son los actores participantes en los casos de uso de interés en este proyecto.

A01	Doctor
Descripción	Este actor representa a los doctores miembros del servicio de medicina interna.

Tabla 1: Doctor

A02	Gestor de turnos
Descripción	Este actor representa al gestor de turnos.

Tabla 2: Gestor de turnos

3.2 CASOS DE USO

En el punto de partida contamos con un servidor REST, un servidor de aplicaciones web y el dispositivo de usuario. El servidor de aplicaciones web y el servidor REST se comunican entre sí a través de peticiones HTTP. Los componentes podrían desplegarse en una sola máquina o estar distribuidos. Durante el desarrollo se despliega todo en el mismo equipo para simplificar el trabajo. El dispositivo del usuario es el navegador web de un equipo.

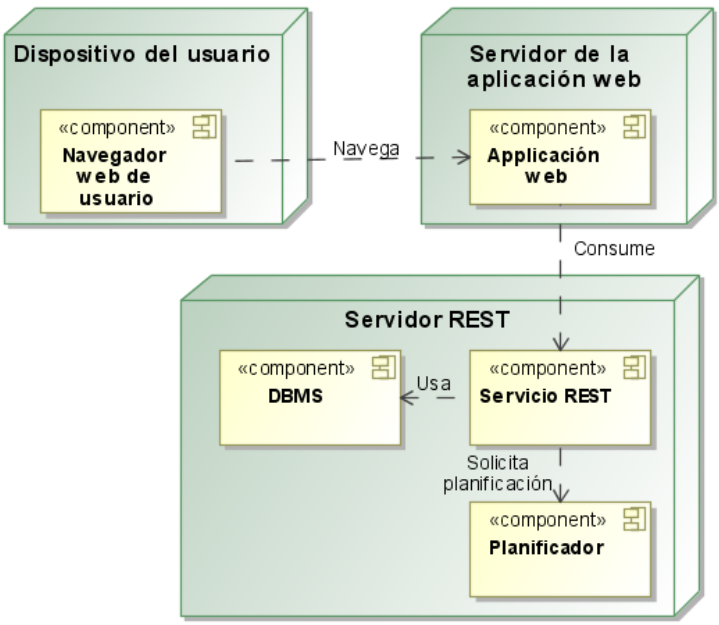


Ilustración 10: Diagrama de despliegue servicio actual

3.2.1 Creación/modificación del calendario y notificación al doctor

3.2.1.1 Sistema actual

En la actualidad [1] el gestor de turnos es el encargado de generar la planificación a través del cliente web conectado al servicio REST. Posteriormente tendrá que gestionar los cambios de turno entre doctores si fuera necesario. Tras ambas acciones debe notificarse a los doctores a través de un email.

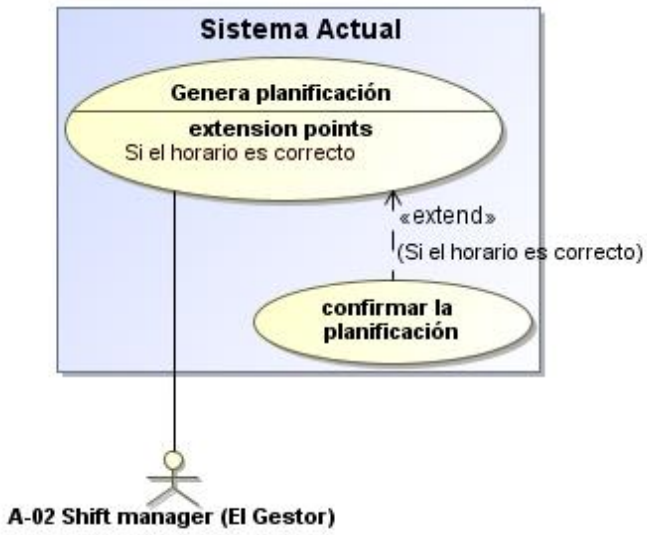


Ilustración 11: Diagrama de caso de uso actual

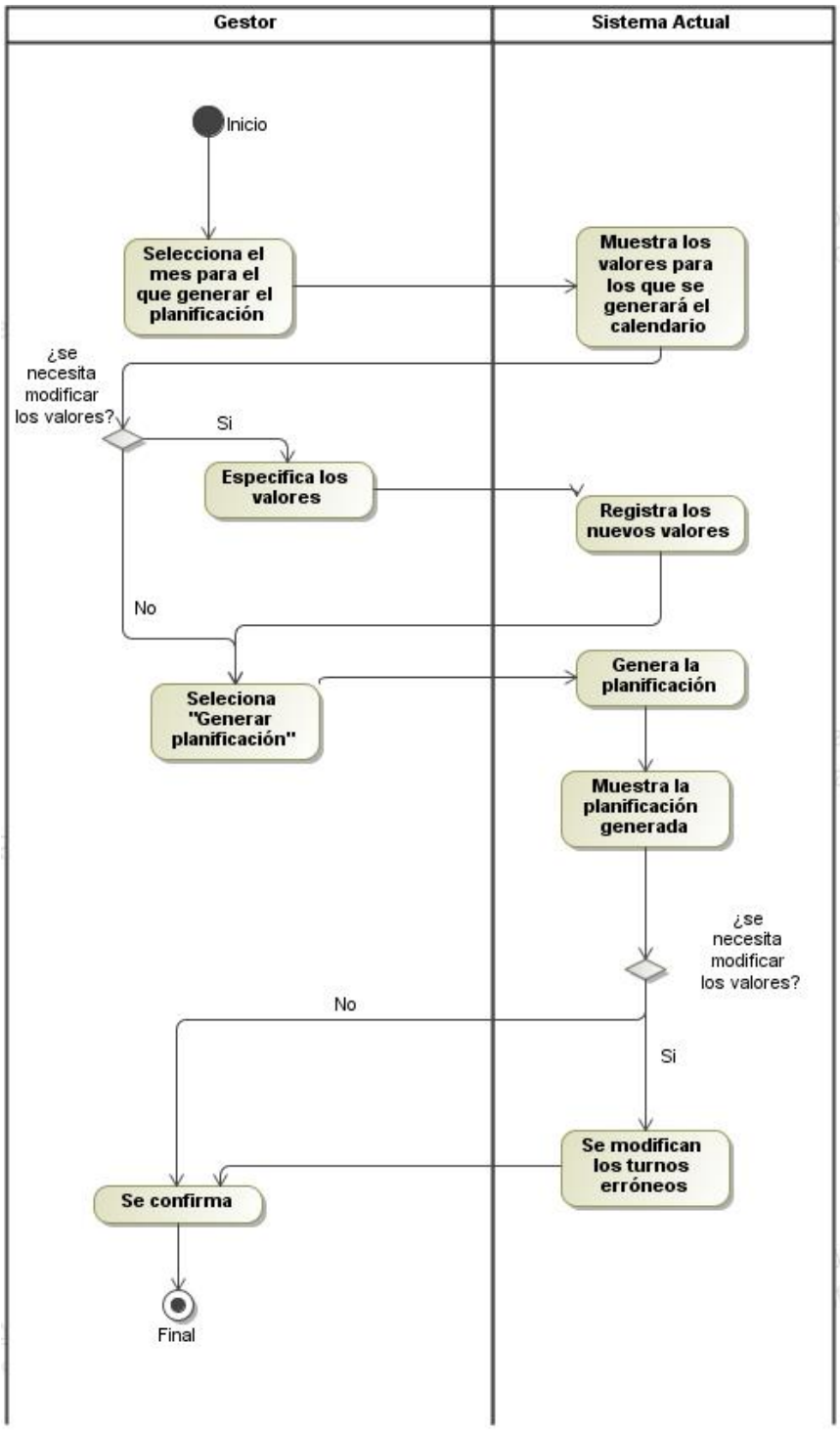


Ilustración 12: Diagrama de actividad - Generación de horarios actualmente

3.2.1.2 Sistema deseado

El resultado deseado es que una vez que el gestor de turnos confirme la planificación generada automáticamente el sistema cree los calendarios individuales de cada doctor y los notifique vía email. Esto ocurriría también cuando se hiciera un cambio en el calendario.

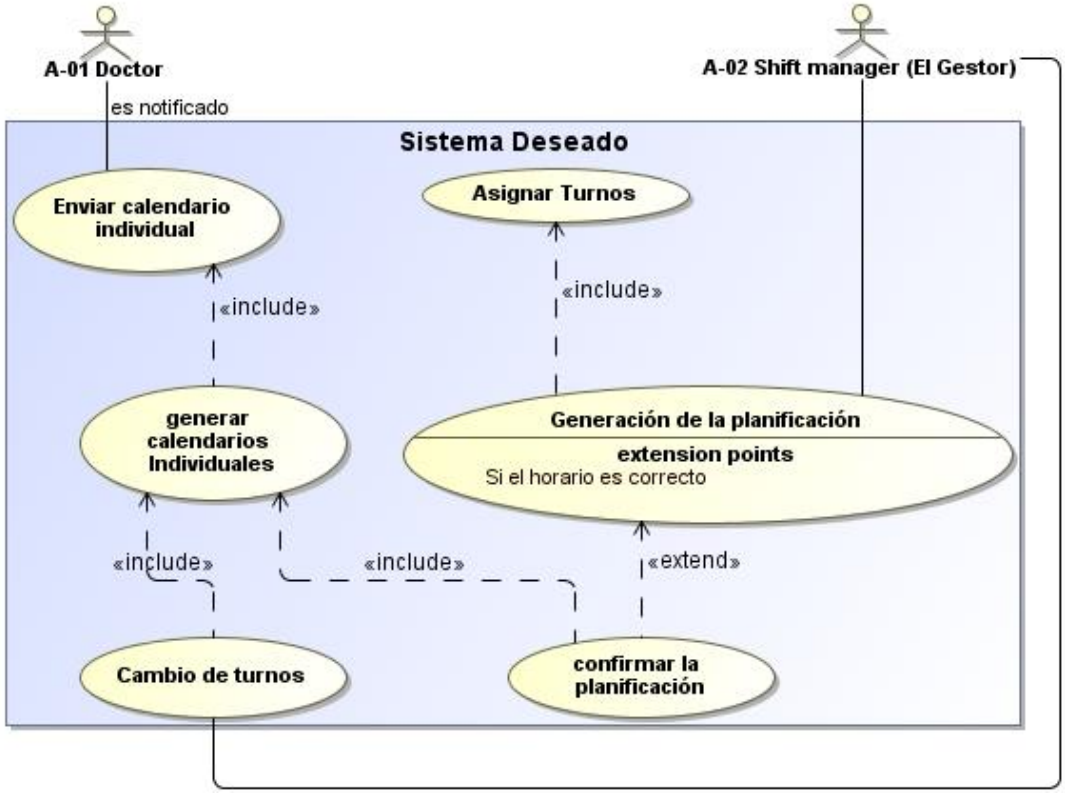


Ilustración 13: Caso de uso deseado

Generación del horario del mes:

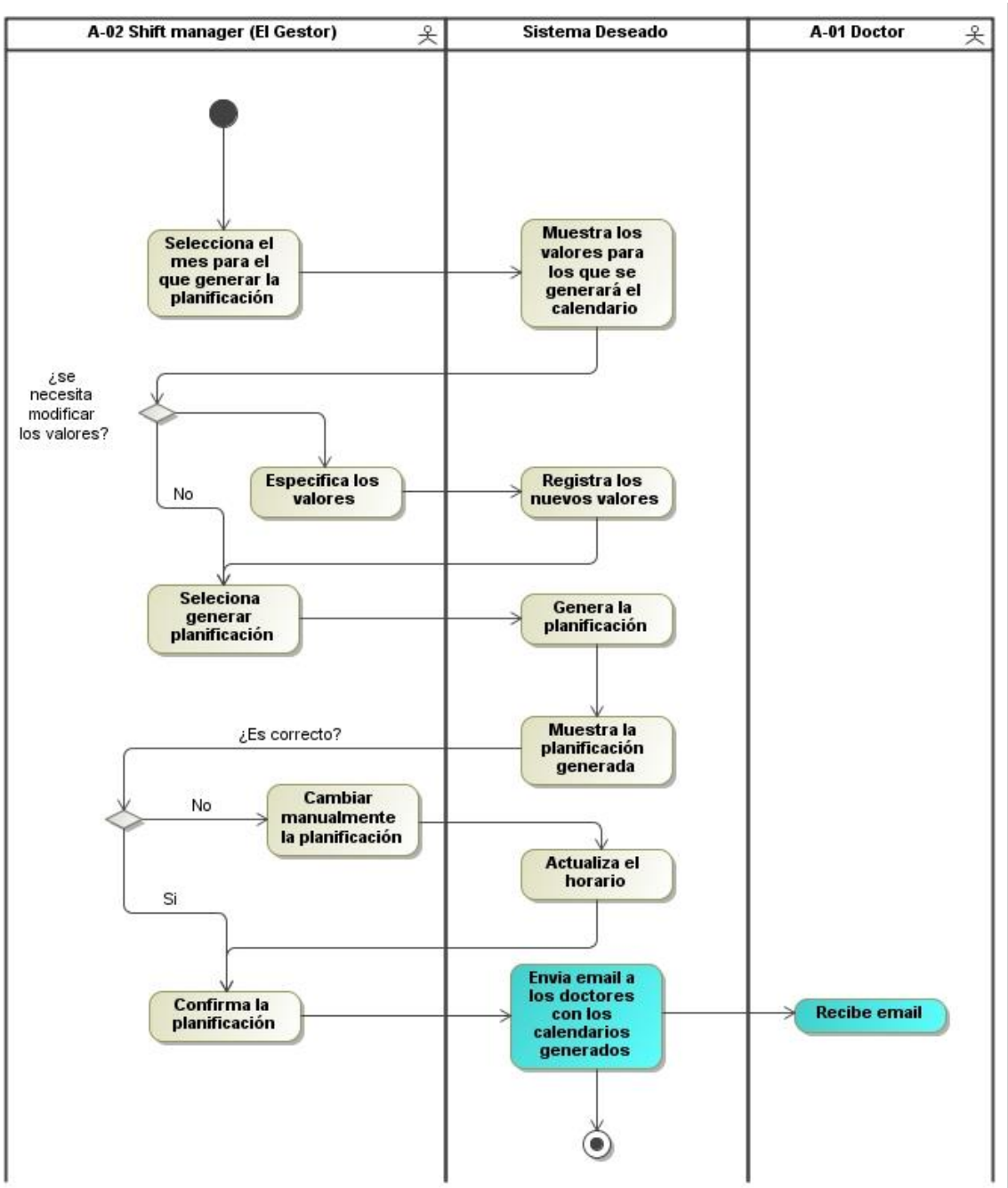


Ilustración 14: Diagrama de actividad – Comportamiento deseado

En el apartado 4 se desarrollan los casos de uso con más detalle.

3.3 REQUISITOS FUNCIONALES DEL SISTEMA

3.3.1 Requisitos de información

RI-01	Información del asistente
Descripción	El sistema debe almacenar la información del asistente (doctor) a un evento.
Datos específicos	El sistema tendrá que almacenar los siguientes datos del asistente. 1. Nombre 2. Email 3. Tipo de usuario de calendario.

Tabla 3: Información de asistente

RI-02	Información del evento
Descripción	El sistema debe almacenar la información de un evento (turno).
Dependencias	RI-01
Datos específicos	El sistema tendrá que almacenar los siguientes datos del asistente. 1. Fecha 2. Título 3. UID (identificador único) 4. Conjunto de asistentes

Tabla 4: Información de evento

RI-03	Información del calendario
Descripción	El sistema debe almacenar la información del calendario.
Dependencias	RI-02
Datos específicos	El sistema tendrá que almacenar los siguientes datos del asistente. 1. ProdId (identificador del productor que creó el objeto iCalendar) 2. Version (versión del intérprete de icalendar) 3. CalScale (define la escala del calendario) 4. TimeZoneRegistry 5. Conjunto de eventos

Tabla 5 : Información del calendario

3.3.2 Requisitos de conducta del sistema

RC-01	Disponibilidad del calendario
Descripción	Los doctores dispondrán del calendario de actividades asistenciales oficial, siempre actualizado, en su servicio de calendario personal.

Tabla 6: Disponibilidad del calendario

RC-02	Informe automático sobre el calendario
Descripción	Los cambios que se hagan al calendario de actividades asistenciales oficial se transmitirán inmediatamente a los servicios de calendario personales de los doctores.

Tabla 7: Informe automático sobre el calendario

RC-03	Calendario personalizado
Descripción	Los doctores podrán disponer en su servicio de calendario personal de un calendario personalizado, siempre actualizado, con sus actividades asistenciales asignadas oficialmente.

Tabla 8: Calendario personalizado

RC-04	Informe automático afectados directos
Descripción	Los cambios en el calendario de actividades asistenciales oficial serán informados inmediatamente a los interesados directamente afectados en el mismo.

Tabla 9: Informe automático afectados directos

3.3.3 Reglas de negocio

RN-01	Persistencia de turnos
Descripción	Los cambios de turno tienen que ser validados por el gestor antes de ser persistidos.

Tabla 10: Persistencia de turnos

4 SOLUCIÓN DISEÑADA

4.1 SERVIDOR REST DEL NUEVO SISTEMA

El sistema diseñado se va a dividir en dos apartados, el primero en el que se detallan pequeñas modificaciones que se le han hecho al proyecto previo y otro apartado en el que se explica el grueso de este proyecto.

4.1.1 Ampliación del sistema

En este apartado se van a comentar los recursos añadidos al proyecto original para que ofrezca funciones de soporte a un Bot de Telegram que permitirá cambios de turnos. Esta aplicación no entra dentro del alcance de este proyecto. Este caso de uso se detallará en puntos venideros.

Para permitir la posibilidad de gestionar los cambios de turno mediante un Bot de Telegram se ha incluido el atributo TelegramID en la entidad doctor.

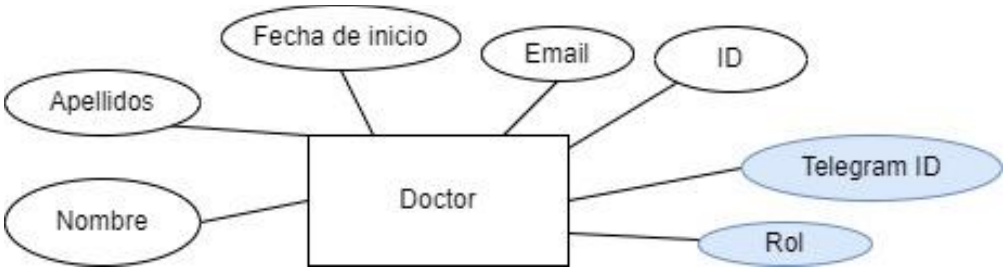


Ilustración 15: Entidad doctor

Para permitir la interacción del Bot con el servicio REST se han añadido nuevas recursos y acciones a las ya existentes. Estas peticiones deben incluir una cabecera de autentificación básica con las credenciales del servicio REST.

Recurso	Id Doctor
URI	/guardians/doctors/idDoctor
Nuevos recursos	• GET o Parámetros (uno de los dos): ▪ Email del doctor (email) ▪ Id de Telegram (idTel) o Respuestas: ▪ 200 OK: Devuelve el ID del doctor en formato Long ▪ 404 NOT FOUND: Si no existe un doctor con el correo electrónico dado.

Ejemplo	"url": "dominio/guardians/doctors/idDoctor?idTel=<Telegram id>", "method": "GET"

Tabla 11: Recurso id de doctor

Recurso	Telegram ID
URI	/guardians/doctors/telegramID

Nuevos recursos	• GET
	  ○ Parámetro:
	  ▪ ID único del doctor (doctorId)
	  ○ Respuestas:
	  ▪ 200 OK: Devuelve el ID de telegram del doctor en formato String
	  ▪ 404 NOT FOUND: Si no existe un doctor con el id dado o si el doctor no tiene asociado id de telegram
	• PUT:
	  ○ Parámetros: el id personal del doctor va en la url y el id de telegram va en el cuerpo. ("/telegramID/{doctorId}")
	  ○ Respuestas:
	  ▪ 200 OK: devuelve un mensaje de texto si ha sido exitoso la inserción
	  ▪ 404 NOT FOUND: si no existe un doctor con el id dado

Ejemplo	"url": "dominio/guardians/doctors/telegramID/{doctorId}", "method": "PUT", "headers": { "Content-Type": "text/plain" }, "data": "<Telegram ID>"

Tabla 12: Recurso ID de Telegram

Además, se añade un recurso rol para poder diferenciar los distintitos perfiles de los doctores. Estas peticiones deben incluir una cabecera de autentificación básica con las credenciales del servicio REST.

Recurso	Rol doctor
URI	/guardians/doctors/rol

Nuevos recursos	• GET
	  ○ Parámetro: email
	  ○ Respuestas:
	  ▪ 200 OK: Devuelve los roles del doctor en formato String
	  ▪ 404 NOT FOUND: Si no existe un doctor con el email dado
	• PUT:
	  ○ Parámetros: el email personal del doctor va en la url y el rol va en el cuerpo. ("/rol/{email}")
	  ○ Respuestas:
	  ▪ 200 OK: devuelve un mensaje de texto si ha sido exitoso la inserción
	  ▪ 404 NOT FOUND: si no existe un doctor con el email dado o si no existe el rol
	• DELETE
	  ○ Parámetro: el email personal del doctor va en la url y el rol va en el cuerpo. ("/rol/{email}")
	  ○ Respuesta:
	  ▪ 200 OK: devuelve un mensaje de texto si ha sido exitoso la inserción
	  ▪ 404 NOT FOUND: si no existe un doctor con el email dado, si no existe el rol o si el doctor no tiene ese rol asignado.

Ejemplo	`"url": "dominio/guardians/octors/rol/{email}",`
	`"method": "PUT",`
	`"headers": {`
	`"Content-Type": "text/plain" },`
	`"data": "Administrador"`

Tabla 13: Recurso rol doctor

Recurso	Entidad rol
URI	/guardians/rol

Nuevos recursos	• GET
	  ○ Respuestas:
	  ▪ 200 OK: Devuelve los roles en formato String
	• PUT:

	○ Parámetros: el rol a añadir. ("/rol/{rol}") ○ Respuestas: ▪ 200 OK: devuelve un texto si ha sido exitoso la inserción ▪ 404 NOT FOUND: si ya existe ese rol
Propiedades	• Rol

Ejemplo	`"url": "dominio/guardians/rol/{rol}",`
	`"method": "PUT"`
	
	`"url": "dominio/guardians/rol",`
	`"method": "GET"`

Tabla 14 : Recurso entidad rol

4.2 FUNCIONALIDADES AÑADIDAS AL SERVICIO REST

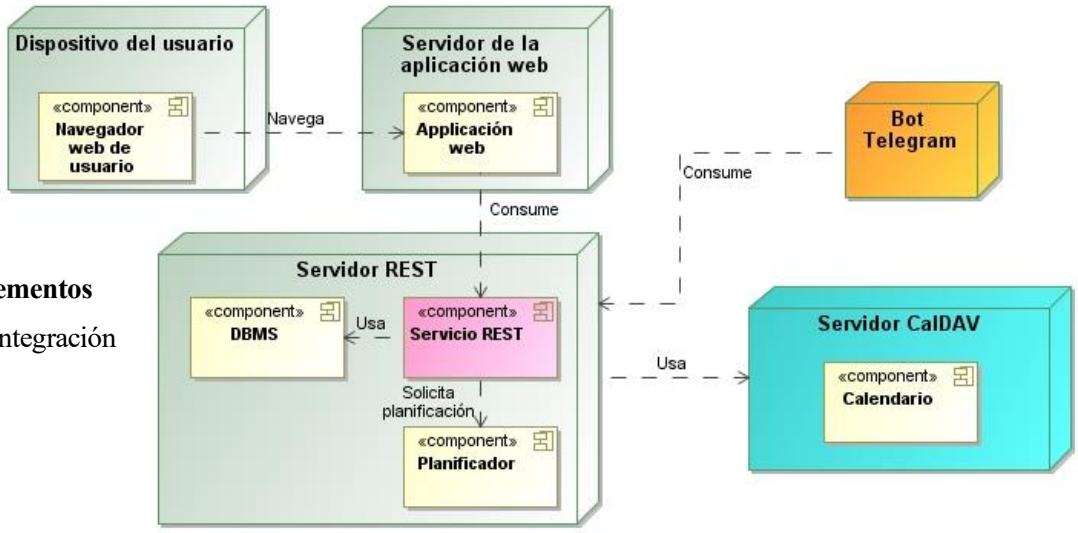
En este apartado se va a comentar al detalle la implementación del servicio de calendario incluyendo la notificación vía email a los doctores. La edición del calendario con el nuevo turno del mes se genera una vez que el gestor de turnos confirma el calendario o si un calendario ya generado se modifica.

Como servicio de calendario se usa un servidor CalDAV en concreto en DaviCal. El servidor está alojado en un contenedor de Docker. Esto hace que al sistema original se añada un componente. Este punto es el objetivo principal de este trabajo.

Además, como se ha comentado con anterioridad, se han añadido recursos y funcionalidades que darán soporte la modificación de turnos.

El servicio REST se encarga de las operaciones de crear, leer, actualizar y borrar información del sistema. A las funcionalidades ya existentes se han añadido las que permitirán estas acciones para el servicio de calendario.

El diagrama incluye el planificador, encargado de programar los turnos, y la aplicación web que otorga una interfaz al usuario para interactuar con el sistema.



Estado de los elementos

Naranja: Pendiente de integración

Rosa: Modificados

Azul: Nuevos

Ilustración 16: Diagrama de despliegue

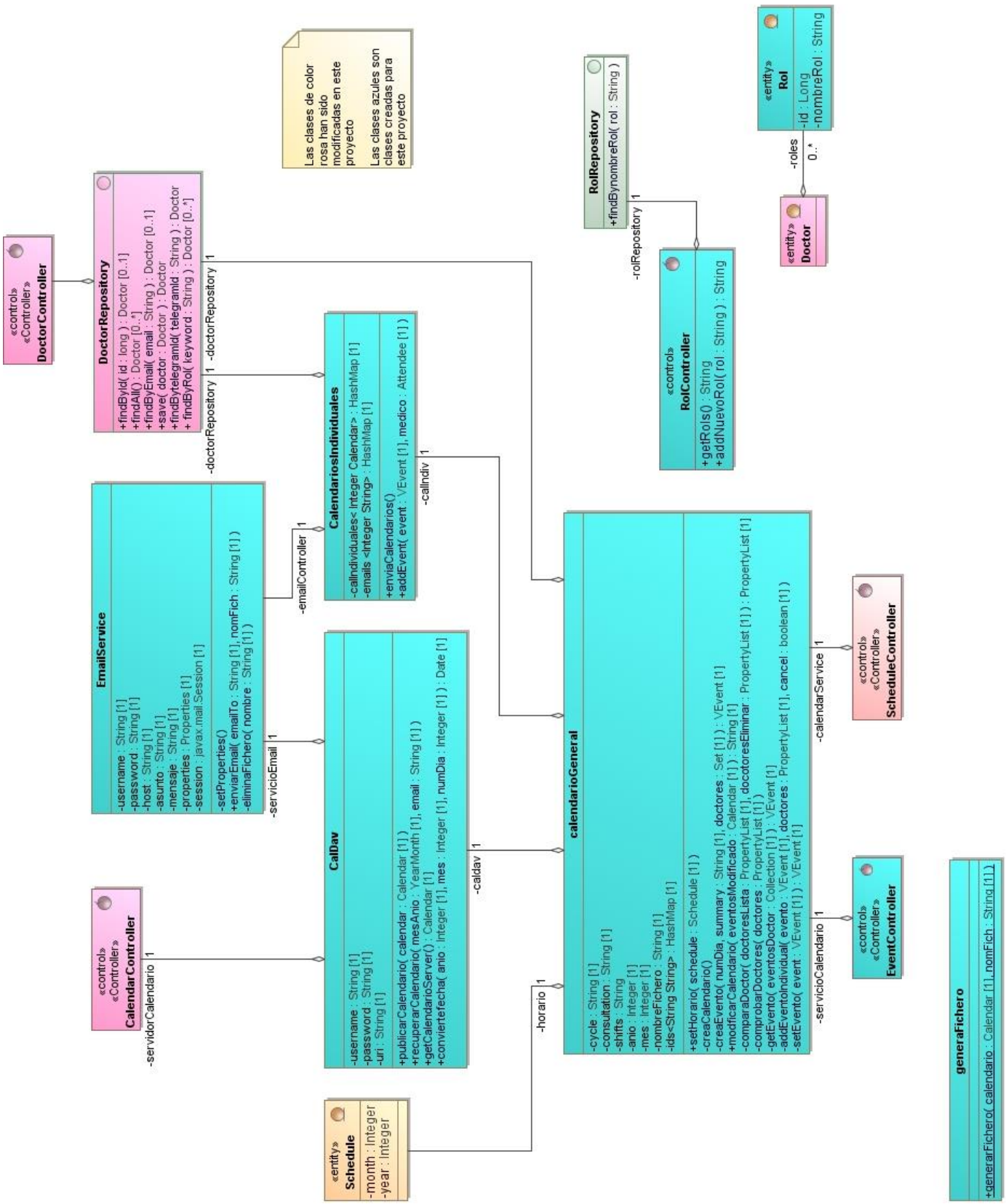


Ilustración 17: Diagrama de clases creadas

Las clases que se han creado para este este proyecto son las de color turquesa para diferenciarlas de las ya existentes. Estas son las siguientes:

- `generaFichero:` es la clase que se encarga de crear el fichero con el calendario individual de cada doctor. Este es fichero que será enviado como archivo adjunto por correo electrónico a cada doctor.
- La clase `calendarioGeneral` se encarga de manejar el calendario: Crea el calendario en formato iCalendar gracias a la librería ical4j a partir de la entidad Schedule y modifica el calendario generado si así se solicita.
- La clase `CalDAV` se encarga de la comunicación con el servidor. Se encarga de publicar el calendario general una vez se ha creado y de recuperar el calendario de un doctor para un mes concreto cuando lo solicite.
- `CalendarioIndividual` tiene una funcionalidad muy similar a la de `calendarioGeneral,` pero en este caso lo que se genera es un calendario individual con ical4j por cada doctor para que posteriormente puedan ser enviados a través de correo electrónico como archivo adjunto gracias a `generaFichero`.
- `EmailService` se encarga de enviar por correo electrónico, gracias a la API JavaMail, el archivo ics generado con el calendario personalizado de cada doctor.

4.2.1 Casos de uso

A continuación, se desarrollan los tres casos de uso principales del sistema. Con objeto de asimilar la explicación a la terminología de la API iCal4j en algunos casos se hablará de los turnos como eventos y de los doctores como asistentes.

4.2.1.1 Creación del calendario y notificación al doctor

La clase `SchedulerController` es la encargada de gestionar la petición de creación de una nueva planificación. Será esta clase la que llamará a nuestro servicio una vez el gestor confirma que la planificación es correcta y comienza el proceso para persistirla. Este proceso incluye crear el calendario correspondiente a dicha planificación. Para ello se solicita el calendario del servicio de medicina vía http a la URL definida en las propiedades de la aplicación (en un archivo llamado application.properties) y la respuesta se convierte a formato iCal4j. Si no existe calendario en la URL otorgada se crea uno directamente en formato ical4j para publicarse posteriormente.

En la clase `calendarioGeneral` se añade al calendario un evento por cada turno con los doctores que participan en ellos, además de enviar esos mismos eventos a la clase `calendariosIndividuales` donde, como su nombre indica, se crean los calendarios individuales de cada doctor. Cada evento tiene un UID único con el siguiente formato *yyymmd(d)xx* donde xx es una clave según el tipo de turno. Una vez todo ha sido creado la clase `CalDAV` publica el calendario general en el servidor y los calendarios individuales se pasan a un fichero que es enviado por correo electrónico a cada doctor gracias a la API JAVAX Mail.

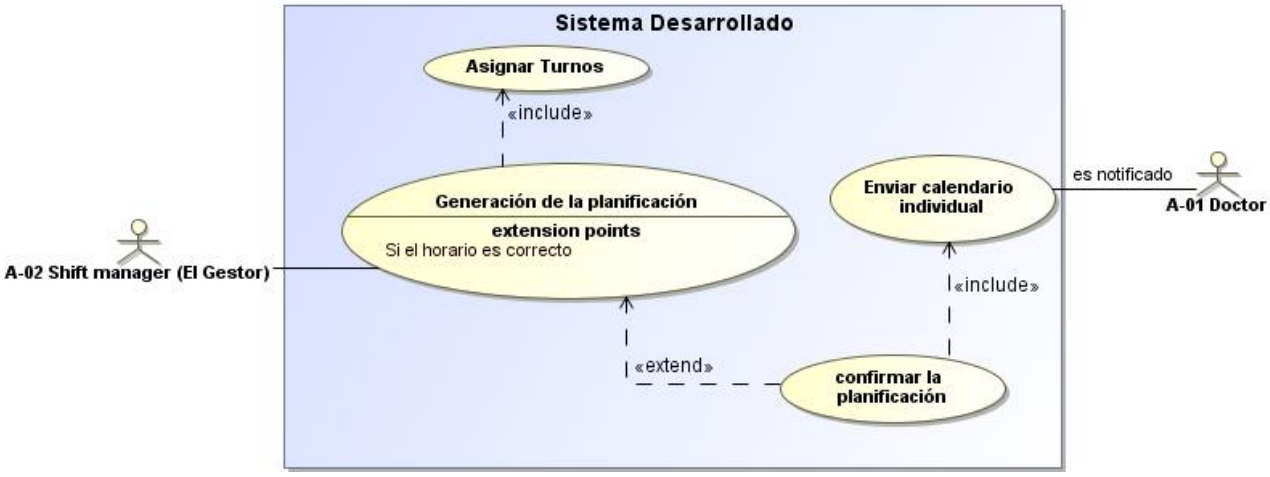


Ilustración 18: Caso de uso - Crear calendario

Este caso de uso se desarrolla según el siguiente procedimiento:

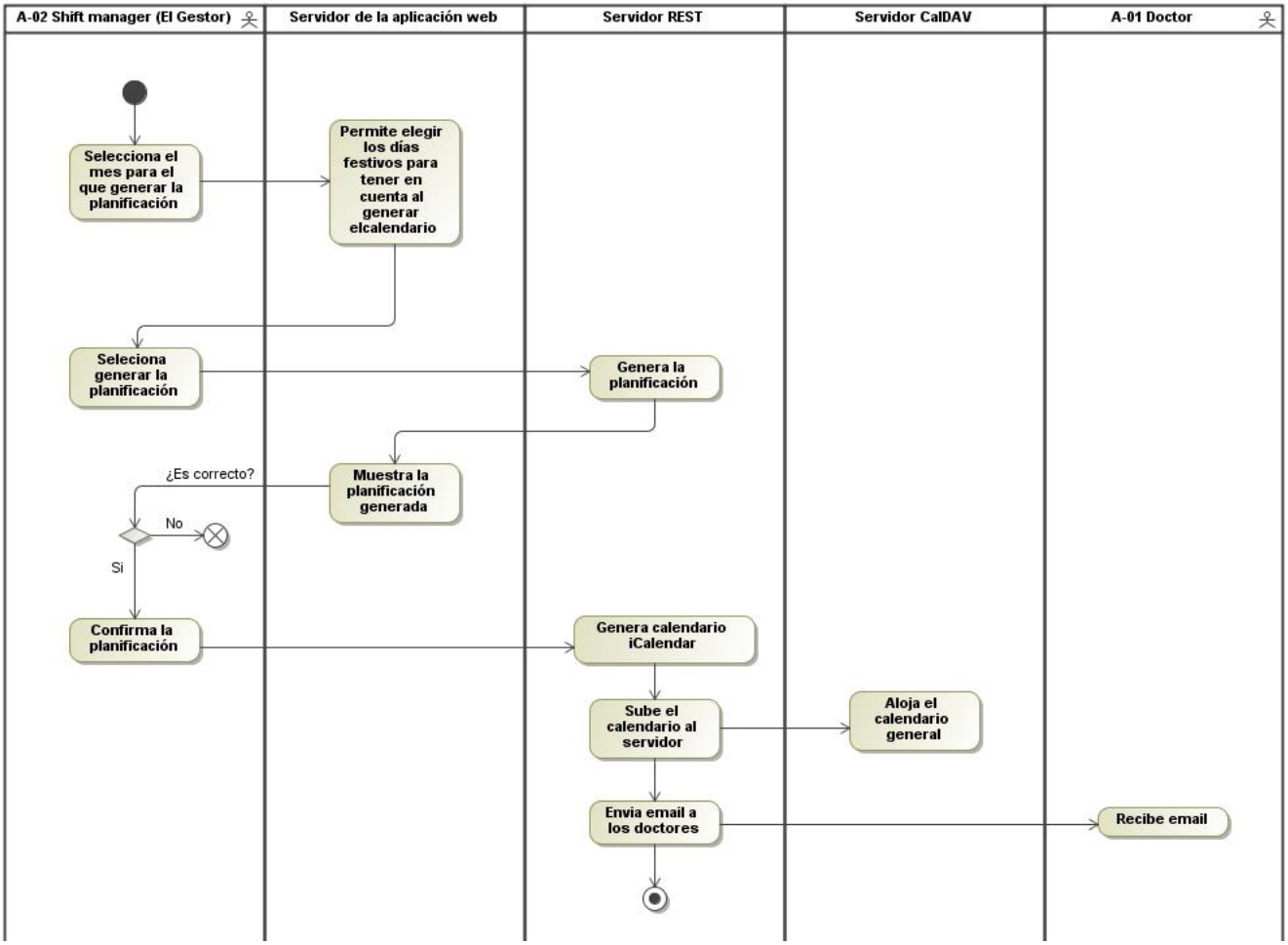


Ilustración 19: Diagrama de actividad - Crear calendario

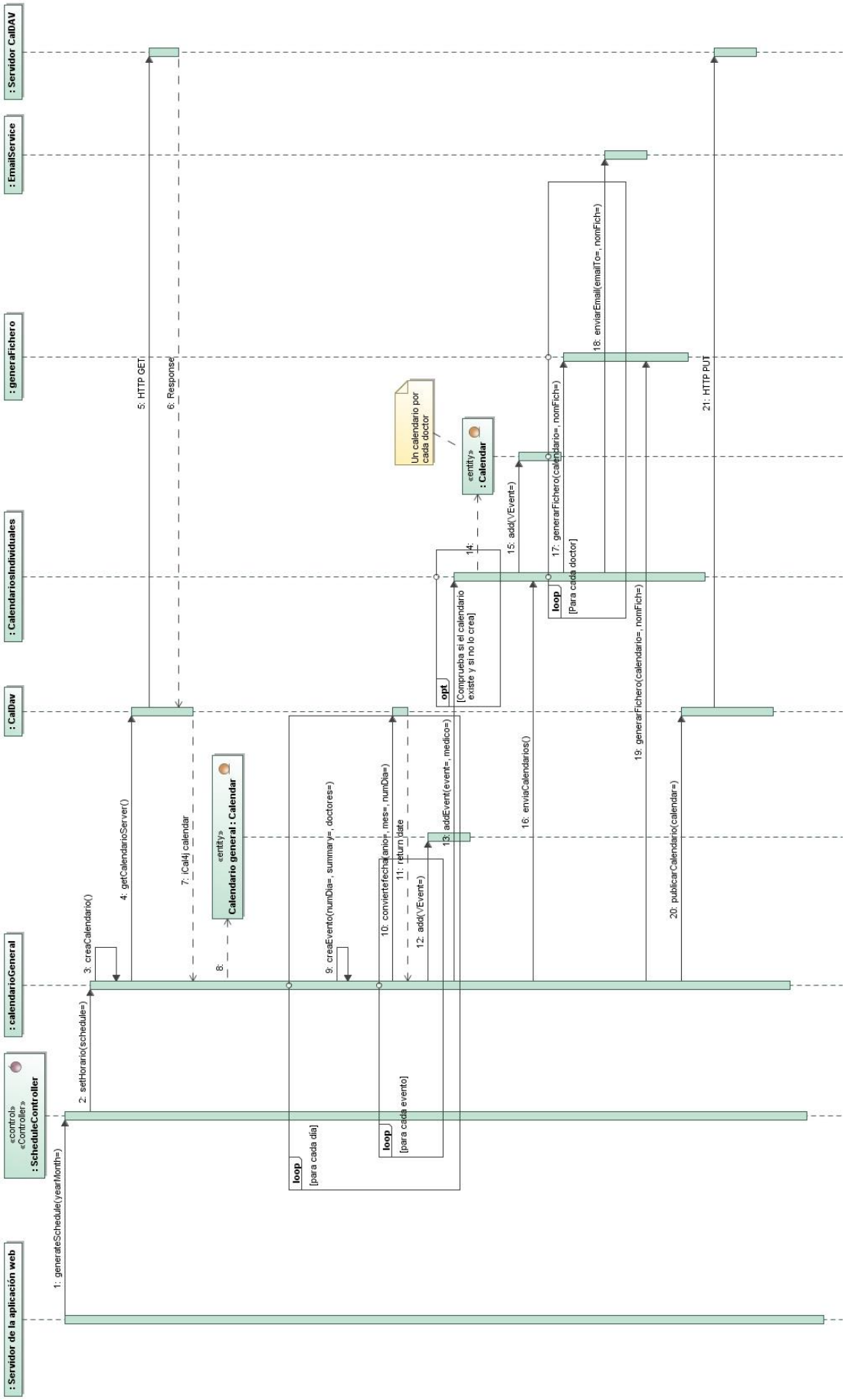


Ilustración 20: Diagrama de secuencia - Crear calendario

4.2.1.2 Recuperar calendario individual

Cuando un doctor desea recuperar su calendario para un mes concreto. Se realiza una petición al servicio REST en la que se le pasa como parámetros el mes y año además del correo electrónico del doctor. El sistema solicita el calendario del servicio de medicina al servidor y se convierte a formato iCal4j.

Una vez obtenido se utiliza un filtro para buscar todos los eventos en el mes del año seleccionado. A esta selección se le pasa un segundo filtro buscado eventos cuyos asistentes (doctores) contengan el email de la petición. Con los eventos que quedan tras la selección se genera un fichero con el calendario y se envía al correo electrónico proporcionado. Si no se encontrara ningún evento con las características deseadas se mostraría un mensaje de error.

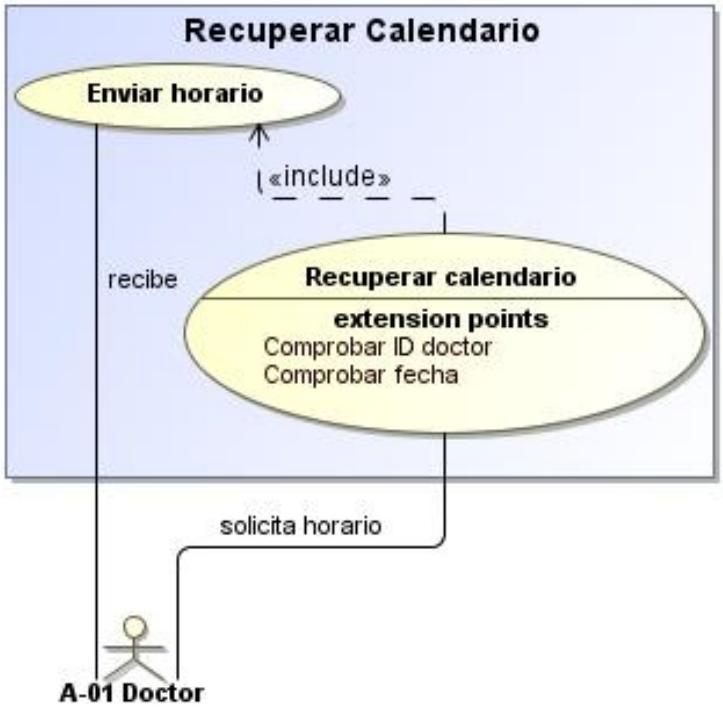


Ilustración 21: Caso de uso - Recuperar calendario individual

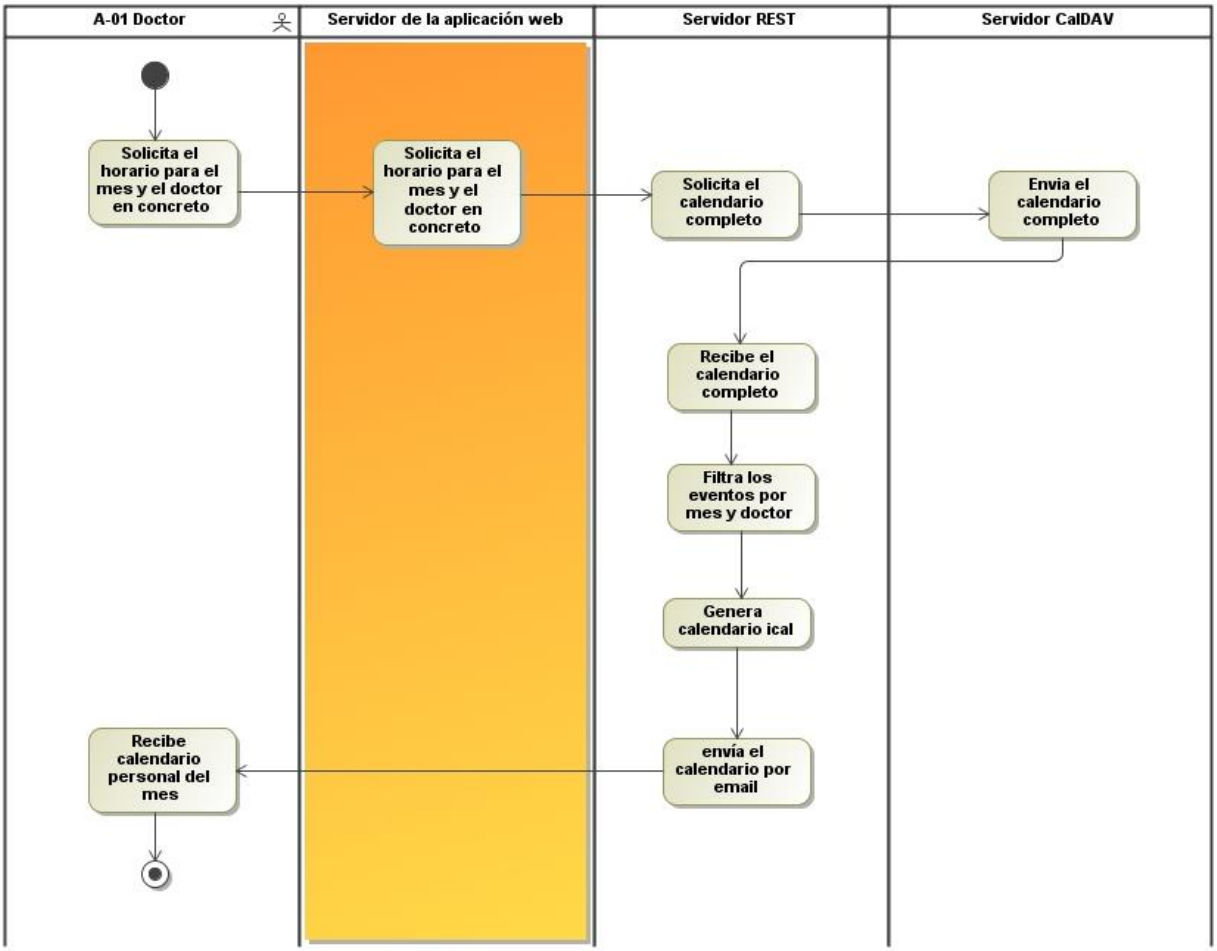


Ilustración 22: Diagrama de activad - Recuperar calendario individual

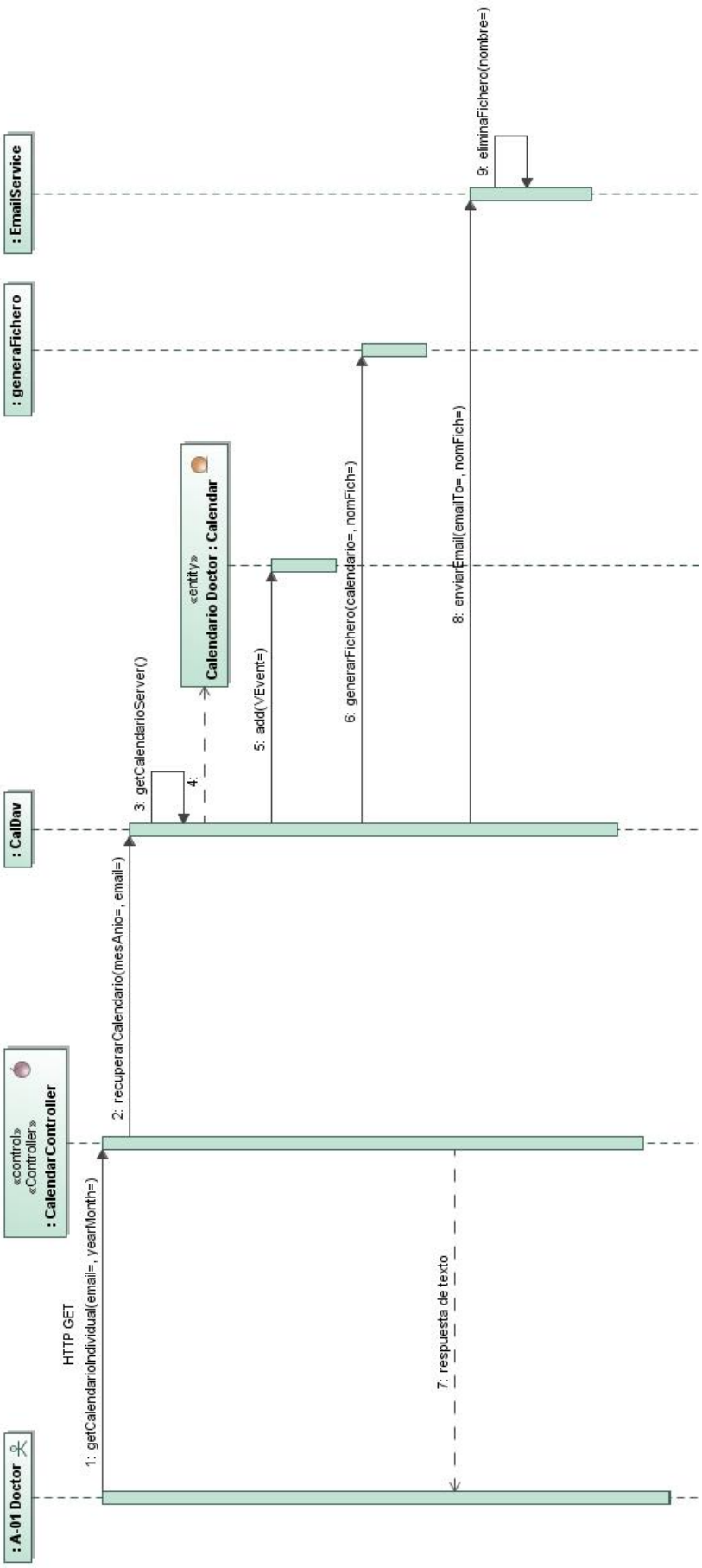


Ilustración 23: Diagrama de secuencia - Recuperar calendario individual

4.2.1.3 Modificación del calendario

`EventController` es el controlador REST encargado de manejar las peticiones relacionadas con la modificación del horario que ya se había creado en el caso de uso *Crear calendario*. En el cuerpo de la petición HTTP deben incluirse los eventos a modificar, contenidos en un calendario en formato ical.

Recurso	Evento
URI	/guardians/event
Nuevos recursos	• POST: o URI: /guardians/event /update o Parámetros: el evento a modificar va en el cuerpo en formato String según el estándar ical o Respuestas: ▪ 200 OK: devuelve un mensaje de texto si ha sido exitoso la actualización ▪ 404 NOT FOUND: si no existe un evento con el id y fecha dada o, si los doctores asistentes al nuevo evento no existen en la base de datos
Ejemplo	`"url": "http://localhost:8080/guardians/api/event/update", "method": "POST", "headers": { "Content-Type": "text/plain" }, "data": "BEGIN:VCALENDAR\nPRODID:-//Mozilla.org/NONSGML Mozilla Calendar V1.1//EN\nVERSION:2.0\nBEGIN:VEVENT\nDTSTAMP:20220207T194654Z\nUID:132022jc\nSUMMARY:Jornadas Complementarias\nATTENDEE:mailto:carcohcal@alum.us.es\nATTENDEE:mailto:20@guardians.com\nDTSTART;VALUE=DATE:20220301\nEND:VEVENT\nEND:VCALENDAR\n"`

Tabla 15: Recurso evento

En la clase `calendarioGeneral` se solicita el calendario general del servicio de medicina al servidor y se convierte a formato iCal4j. Se comprueba que el UID de cada evento modificado proporcionado en la petición es igual a un UID de un evento ya existente en el calendario general. Una vez esto ha sido verificado, se comprueba que los doctores a los que involucra ese cambio existen en la base de datos. Una vez verificado que la información es correcta se modifica al calendario general y se generan los calendarios individuales de los doctores afectados por ese cambio. Cuando todo ha sido creado se publica el calendario general ya modificado en el servidor y se envían los calendarios individuales a través de correo electrónico. Como el UID de los eventos persiste, el doctor puede actualizar su calendario personal con el archivo enviado. Se le añadirán o eliminarán los eventos según proceda.

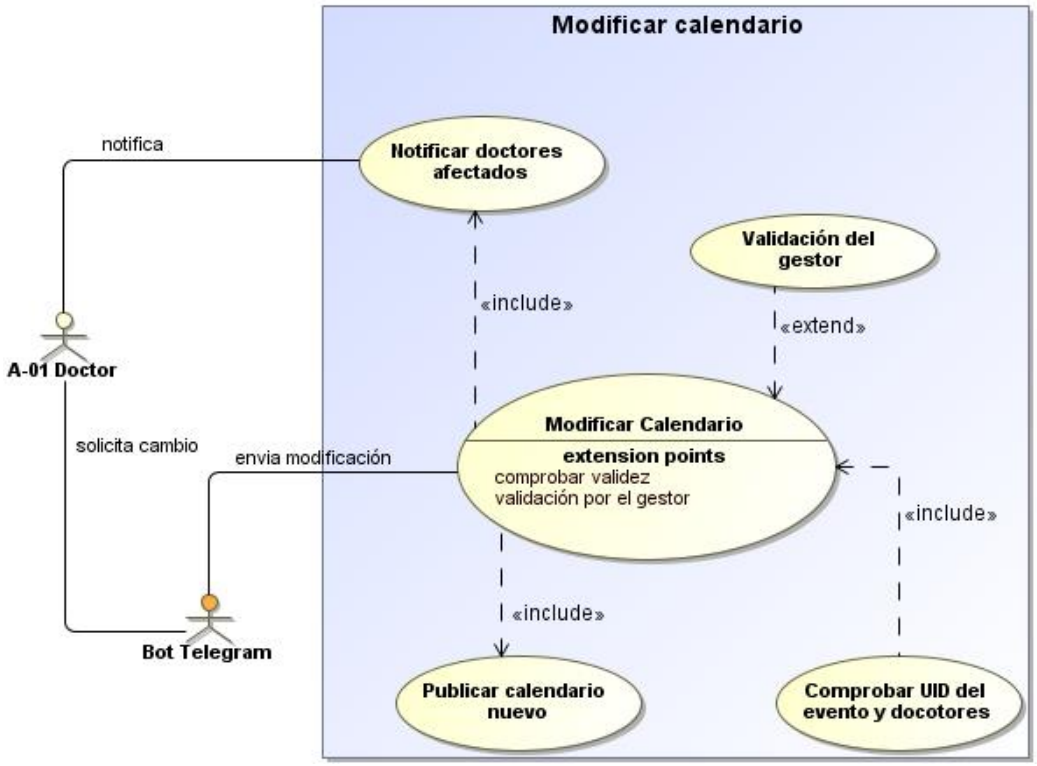


Ilustración 24: Caso de uso - Modificar calendario

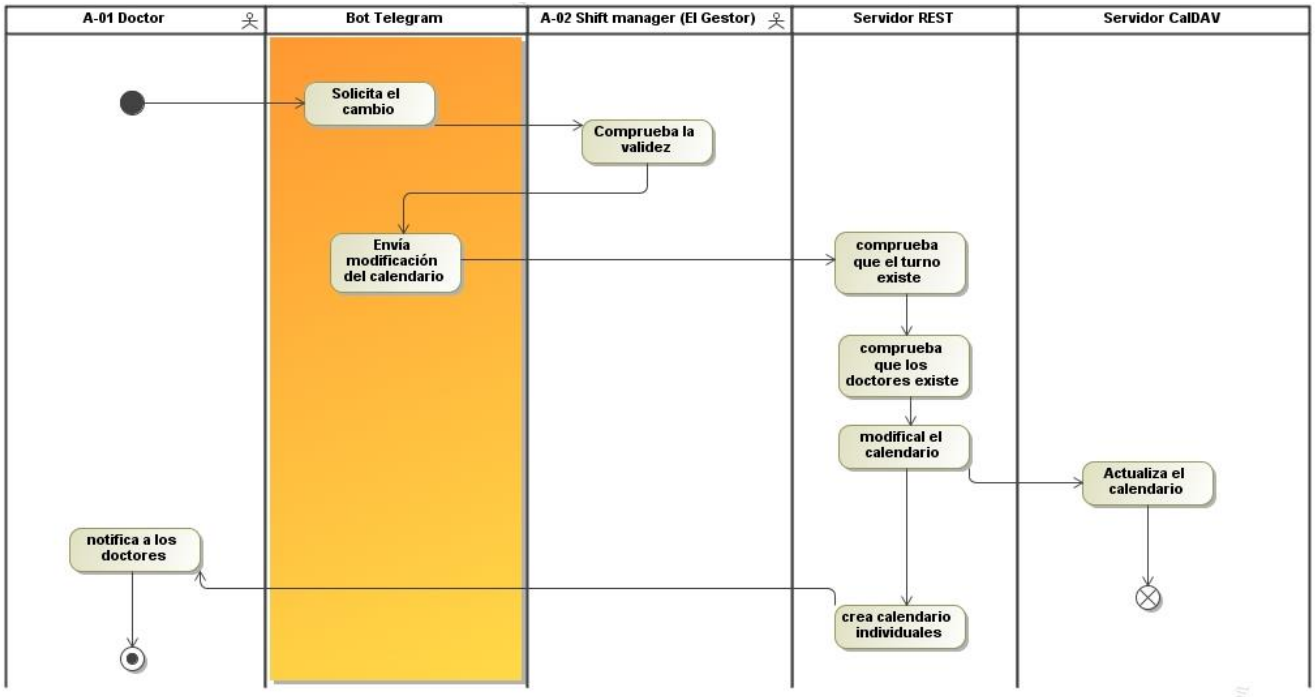


Ilustración 25: Diagrama de actividad - Modificar calendario

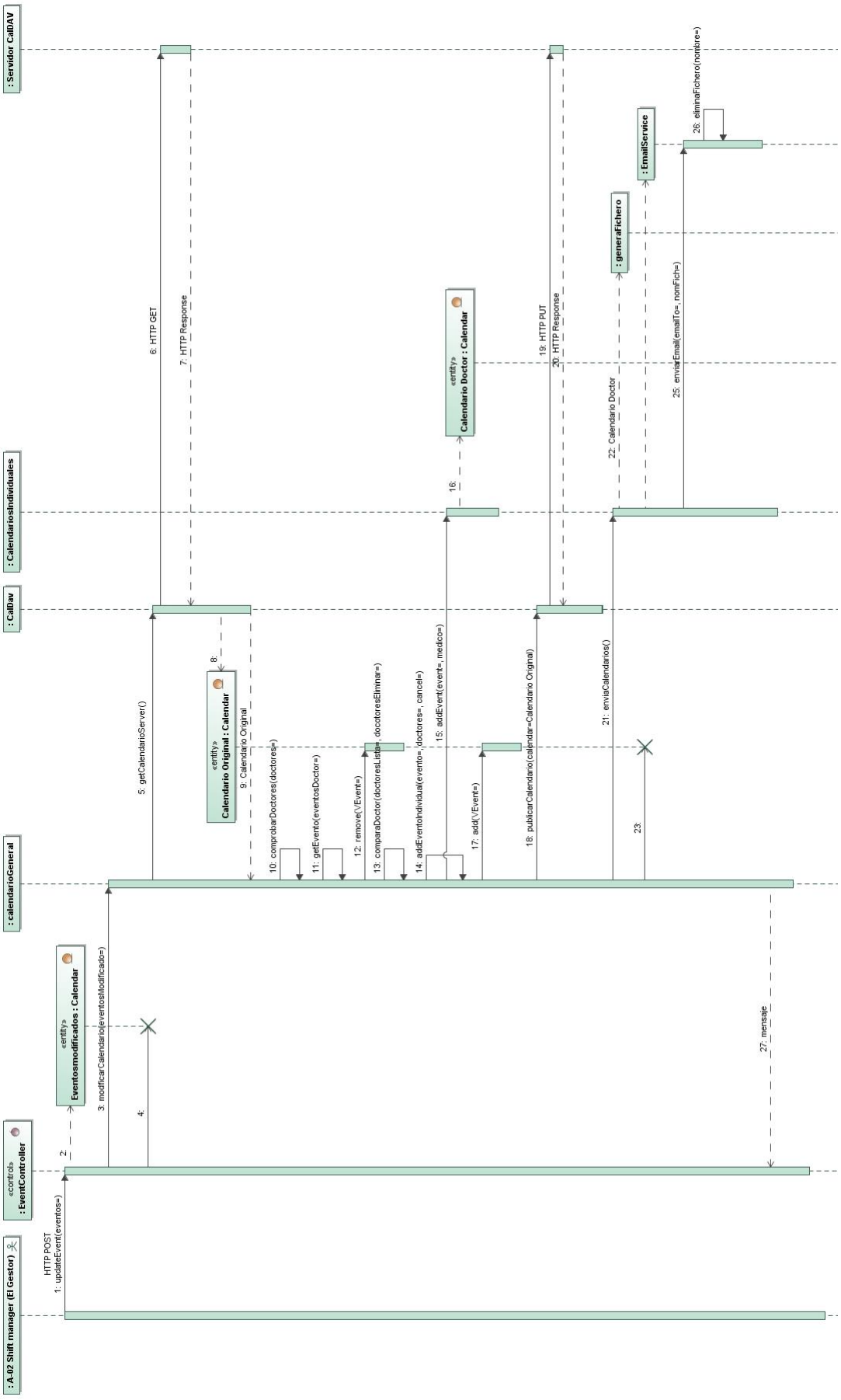


Ilustración 26: Diagrama de secuencia - Modificar calendario

4.2.2 Despliegue

Para facilitar el despliegue del sistema se ha optado por utilizar Docker por lo que es necesario su instalación. La imagen se creará en base a otra ya preconfigurada con los requisitos y dependencias del sistema; como Java8, Python y Git. Esto hace que el usuario solo tenga que configurar los parámetros propios de su caso.

Para el despliegue es necesario clonar la rama *despliegue* del repositorio https://github.com/tfg-projects-dit-us/ServicioDeCalendarios de GitHub.

```
git clone -b despliegue https://github.com/tfg-projects-dit-us/ServicioDeCalendarios
```

Además, hará falta crear un token de autentificación de github con permisos para leer paquetes (Guía).

A continuación, se detallan los pasos para el despliegue, que incluye modificar ciertos parámetros de distintos archivos:

1) En *docker-compose-calendario.yml* se define el servicio de calendario. Para esto hay que configurar la base de datos que persistirá los calendarios generados, creando la contraseña de administrador (POSTGRES_PASSWORD). Para el servicio de calendario en sí los parámetros a definir son: la contraseña de la base de datos (PGSQL_ROOT_PASS), configurada anteriormente, y las contraseñas de usuario(PASSDAVDB) y administrador (ADMINDAVICALPASS) del servidor además del hostname.

```yaml
db:
  image: postgres
  volumes:
    - pgsql_data:/var/lib/postgresql/data
  restart: always
  environment:
    POSTGRES_PASSWORD: "BotGuardianes2021"
calendar:
  image: tuxnvape/davical-standalone
  volumes:
    - davical_config:/config
  restart: always
  ports:
    - 80:80
  environment:
    HOST_NAME: "guardianes"
    PGSQL_ROOT_PASS: "BotGuardianes2021"
    PASSDAVDB: "BotGuardianes2021"
    DBHOST: "db"
    ADMINDAVICALPASS: "BotGuardianes2021"
    LANG: "en_US.UTF-8"
    LC_ALL: "en_US.UTF-8"
    DAVICAL_LANG: "en_US"
```

Código 5: Docker-compose servidor calendario

2) Una vez configurado iniciamos el servicio con el siguiente comando:

   ```
   Docker compose -f /path/docker-compose-calendario.yml up -d
   ```

3) Una vez levantado el contenedor nos conectamos a http://localhost e iniciamos sesión con la cuenta de administrador (usuario admin y la contraseña creada) para crear al menos un usuario con permisos de escritura y lectura para el servidor aparte del administrador.

4) Los siguientes parámetros del archivo *resource/application.properties* deben ser modificados:

   a) Parámetros correspondientes al servicio de calendario utilizado por el usuario. En este caso al utilizar el servicio de calendario en el contenedor primero se deberá crear el usuario en el servidor a través de la interfaz web. El usuario y contraseña son los generados en el paso 4. El nombre del calendario es a elegir.

      i) calendario.user = USUARIO DEL CALENDARIO

ii) calendario.psw = CONTRASEÑA DEL CALENDARIO

iii) calendario.uri = http://calendar/caldav.php/<Usuario>/calendar/<Nombre del calendario>

b) Parámetros correspondientes al servicio de email utilizado por el usuario

i) email.host = HOST SMTP DEL SERVICIO DE EMAIL

ii) email.loggin = USUARIO DEL EMAIL

iii) email.password = CONTRASEÑA EMAIL

c) Parámetros de la base de datos myql.

i) spring.datasource.username= USUARIO DE LA BASE DE DATOS

ii) spring.datasource.password= CONTRASEÑA DE LA BASE DE DATOS

5) En el fichero *Dockerfile* deberá sustituir <*Authorization token*> con el token generado con anterioridad.

6) Crea la imagen de Docker (que debe estar instalado en el sistema) con el comando: `docker build -t yourusername/repository-name:tag .`

7) En el fichero docker-compose-rest.yml se definen las variables de configuración del servicio REST y la base de datos mysql del servicio REST.

a) En la base de datos mysql los parámetros a configurar son el nombre de base datos (MYSQL_DATABASE), el usuario y la contraseña de esa base de datos (MYSQL_USER & MYSQL_PASSWORD respectivamente). Estos deben coincidir con los del archivo *application.properties.* Las tablas populate_sql se deberán modificar con los datos correspondientes para cada caso de uso.

```yaml
myapp-mysql:
  image: mysql:8.0
  environment:
    - MYSQL_ROOT_PASSWORD=root
    - MYSQL_DATABASE=db_guardians
    - MYSQL_USER=springuser
    - MYSQL_PASSWORD=CatChairShowShoe
  ports:
    - 3306:3306
  volumes:
    - ./sql-scripts/createTables.sql:/docker-entrypoint-initdb.d/createTables.sql
    - ./sql-scripts/populateDoctors.sql:/docker-entrypoint-initdb.d/populateDoctors.sql
    - ./sql-scripts/populateShiftConfigs.sql:/docker-entrypoint-initdb.d/populateShiftConfigs.sql
    - ./sql-scripts/populateAllowedShifts.sql:/docker-entrypoint-initdb.d/populateAllowedShifts.sql
```

Código 6: Docker-compose Base de datos Mysql

b) El servicio REST los parámetros a configurar son los definidos con anterioridad en la base de datos mysql, esto hará que ambos contenedores se conecten entre sí.

```yaml
myapp-main:
  image: andrewem/serviciodecalendarios:prueba
  restart: on-failure
  depends_on:
    - myapp-mysql
  ports:
    - 8080:8080
  environment:
    - DATABASE_HOST=myapp-mysql
    - DATABASE_USER=springuser
    - DATABASE_PASSWORD=CatChairShowShoe
    - DATABASE_NAME=db_guardians
    - DATABASE_PORT=3306
```

Código 7: Docker-compose servicio REST

8) Modifica el archivo docker-compose con el nombre de la imagen creada en el paso 6.

```
myapp-main:
   image: yourusername/repository-name:tag
```

9) Ejecutar la aplicación con `Docker compose -f /path/docker-compose-rest.yml up -d`

Tras este paso el servicio REST ya estaría en funcionamiento en la URL http://localhost:8080/api/guardians

5 CONCLUSIONES Y FUTURAS MEJORAS

5.1 FUTURAS MEJORAS

El sistema actual es una ampliación del original, pero sigue habiendo requisitos que aún no se han cumplido. En este apartado se describirán posibles líneas de trabajo para mejorar y completar esta necesidad.

5.1.1 Alojar calendarios individuales en el servicio de calendario

Actualmente el calendario individual es enviado al doctor como un archivo adjunto con extensión ics para que posteriormente dicho doctor lo importe a su servicio de calendario personal (Google Calendar, iCalendar etc.)

Si un doctor quiere recuperar dicho calendario el sistema parte del calendario general del servicio y filtra los eventos asociados para dicho doctor y reenvia el archivo para que los agregue a su calendario personal.

La mejora sería utilizar el servicio de calendario para alojar los calendarios individuales a la vez que el general y que los doctores pudieran suscribirse a su calendario personal desde cualquier servicio de calendario.

Para ello se podría crear un calendario identificado por el email de cada doctor.

5.1.2 Añadir funcionalidades a la aplicación web

Queda pendiente la actualización de la interfaz de usuario, en nuestro caso aplicación web, para integrar las nuevas funcionalidades que se han incluido en este proyecto que actualmente solo están disponibles a través de peticiones al servicio REST. Entre ellos añadir: Habilitar la gestión de nuevos atributos de doctor como telegram id o rol, facilitar un apartado para poder gestionar los tipos de roles existentes en el sistema y por último crear un apartado para permitir a los usuarios para consultar sus calendarios individuales.

5.1.3 Crear perfiles de permisos.

Actualmente el administrador de turnos es el único que puede acceder a la interfaz web. Por ello sería necesario crear distintos perfiles de permisos para que los doctores pudieran acceder a la interfaz web con permisos restringidos, como por ejemplo poder solicitar la recuperación de su calendario o visualizarlo una vez esté alojado en el servidor o poder editar campos de su información de contacto.

5.2 CONCLUSIONES

Este proyecto se enfoca en ampliar el proyecto *Servicio para la gestión de actividades asistenciales complementarias* [1] y prepararlo para aplicar un proceso de integración y mejora continua.

Para ello se ha desarrollado un servicio de calendario basado en el estándar iCalendar [12] que plasma la planificación mensual generada por el sistema en un calendario accesible a través de un servidor DaviCal [19].

Este calendario se genera una vez la planificación ha sido confirmada por el gestor de turnos. Además, a la vez que este calendario ha sido generado, se obtienen calendarios individuales por cada doctor que son enviados a los mismos por email gracias a la API JavaMail [5].

Para crear la funcionalidad de modificar los turnos se han añadido recursos como rol e id de Telegram para poder integrar en el futuro un bot de telegram que permitiera el cambio a través de él. Una vez el bot notificara ese cambio al sistema procedería a modificar el calendario general y notificar a los doctores implicados con los cambios, enviándoles una actualización de esos eventos vía email.

Por último, para facilitar el despliegue en producción se ha integrado el sistema con Docker. Existe una imagen base con todas las dependencias del sistema, y a partir de ella se construye el sistema. Se proporciona el dockerfile para crear la imagen del sistema y un docker-compose que integra el sistema con una imagen mysql, lo que facilita el despliegue de la base de datos. Para la puesta en producción solo haría falta modificar los parámetros individuales del sistema, como contraseñas o servicios de email y calendarios específicos.

REFERENCIAS

[1] M. González-Alorda Cantero, Shift scheduling and management service, Sevilla: Universidad de Sevilla, 2020.

[2] P. P. Garrido Abenza, «Comenzando a programar con JAVA,» Universidad Miguel Hernández, 2015.

[3] Oracle, «Java™ Platform, Standard Edition 8,» [En línea]. Available: https://docs.oracle.com/javase/8/docs/api/. [Último acceso: 09 08 2021].

[4] Oracle, «JavaMail Reference Implementation,» [En línea]. Available: https://javaee.github.io/javamail/. [Último acceso: 09 08 2021].

[5] «JavaMail Documentation,» [En línea]. Available: https://www.javadoc.io/doc/javax.mail/javax.mail-api/latest/javax/mail/package-summary.html. [Último acceso: 09 08 2021].

[6] J. T. POINT, «Sending Email in Java,» [En línea]. Available: https://www.javatpoint.com/example-of-sending-email-using-java-mail-api. [Último acceso: 11 08 2021].

[7] VMware, «Spring Framework,» [En línea]. Available: https://spring.io/projects/spring-framework. [Último acceso: 11 08 2021].

[8] C. Ávila Garzón, «Modelo Vista Controlador,» Facultad de Matemáticas e Ingenierías, 11 12 2019. [En línea]. Available: https://repositorio.konradlorenz.edu.co/handle/001/1528. [Último acceso: 09 2021].

[9] [En línea]. Available: https://commons.wikimedia.org/wiki/File:MVC_Diagram.jpg. [Último acceso: 09 2021].

[10] J. P. Smith, «Architecture of Business Layer working with Entity Framework,» [En línea]. Available: https://www.thereformedprogrammer.net/architecture-of-business-layer-working-with-entity-framework/. [Último acceso: 09 2021].

[11] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley , 2012.

[12] I. E. T. Force, «Internet Calendaring and Scheduling Core Object Specification,» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc5545. [Último acceso: 08 2021].

[13] «iCalendar File Format,» [En línea]. Available: https://icalendar.org/. [Último acceso: 08 2021].

[14] B. Fortuna, «iCal4j User Guide,» [En línea]. Available: https://ical4j.github.io/ical4j-user-guide/. [Último acceso: 08 2021].

[15] B. Fortuna, «iCal4j Examples,» [En línea]. Available: https://ical4j.github.io/ical4j-user-guide/examples/. [Último acceso: 08 2021].

[16] «Calendaring Extensions to WebDAV (CalDAV),» [En línea]. Available:

https://datatracker.ietf.org/doc/html/rfc4791. [Último acceso: 08 2021].

[17] A. Mishra, «CalDAV4j Wiki,» [En línea]. Available: https://github.com/caldav4j/caldav4j/wiki. [Último acceso: 08 2021].

[18] «CalDAV4j Documentación,» [En línea]. Available: https://javadoc.io/doc/com.github.caldav4j/caldav4j/latest/index.html. [Último acceso: 08 2021].

[19] A. McMillan, «About DAViCal,» [En línea]. Available: https://www.davical.org/index.php. [Último acceso: 08 2021].

[20] Microsoft , «¿Qué es Docker?,» Microsoft , 2022. [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/architecture/microservices/container-docker-introduction/docker-defined.

[21] Spring. [En línea]. Available: https://spring.io/guides/topicals/spring-boot-docker/. [Último acceso: 2022].

[22] IETF. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc822. [Último acceso: 09 2021].

[23] J. Klensin, «Simple Mail Transfer Protocol,» 11 07 2008. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc5321. [Último acceso: 10 2021].

[24] I. C. Education, «Dcoker,» IBM Inc., [En línea]. Available: https://www.ibm.com/cloud/learn/docker. [Último acceso: 06 2022].

[25] J.-P. Gouigoux, Docker : primeros pasos y puesta en práctica de una arquitectura basada en micro-servicios, Cornellá de Llobregat, Barcelona: Ediciones Eni, 2018.

[26] Amazon, «¿Qué es Docker?,» [En línea]. Available: https://aws.amazon.com/es/docker/. [Último acceso: 06 2022].