# CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

## Proyecto Final de Ciclo

# CALENDARIZACIÓN Y REGISTRO DE HORARIOS DEL HOSPITAL VETERINARIO PENINSULAR

**Autor:** Javier Humberto García Gómez.
**Tutor:** Jorge Juan Delgado Durán.
**Fecha de entrega:** 4 de noviembre de 2020.
**Convocatoria:** 1S 2020 2021.

## Contenido

## Tabla de ilustraciones

# 1. Introducción

**NOTA:** Si bien mi idea original era enviar todo el proyecto terminado, debido al mensaje que publicó ayer en cuanto no enviar el código y por espeto a su tiempo, no lo envío. Sin embargo, si desea consultarlo lo puede descargar de este enlace: https://github.com/JavierGarciaGomez/HVPManagement. De igual forma, tiene un video explicativo en este playlist: https://www.youtube.com/watch?v=hbOJdMtLvus&list=PLDKYjVmIS0jg69A3Zcff0DvqF-sOHZTDg

Por otro lado, estoy consciente del límite de 60 páginas del trabajo, y procure ajustarme, pero no sé exactamente que pueda prescindir (de hecho omití algunos aspectos).

Agradecería cualquier observación que me pudiera realizar para poder tener una buena nota.

Saludos.

## 1.1 Motivación

Este proyecto se elabora como parte del ciclo de formación profesional de Desarrollo de Aplicaciones Multiplataforma y consiste en una aplicación para escritorio que apoye en la gestión de distintas funciones administrativas al Hospital Veterinario Peninsular.

*Antecedentes de la empresa*

El Hospital Veterinario Peninsular es una empresa mexicana, con sede en la ciudad de Mérida, Yucatán, que comenzó a funcionar en 1990. A lo largo de estos treinta años de historia ha había formado de forma independiente, hasta que, en 2016, por medio de un acuerdo comercial con la cadena norteamericana para mascotas Petco, se estableció como una sección de las sucursales de estas tiendas en la ciudad de Mérida, dedicado exclusivamente a la prestación del servicio veterinario y venta de medicamentos con receta, en tanto que la tienda es la única que puede comercializar con alimentos, productos y juguetes para mascota.

Desde la referida alianza en 2017, se han abierto dos sucursales nuevas, una en 2018 y la otra en 2019, y se cerró la sucursal que se tenía independientemente antes de la

mencionada alianza. En este sentido, y esto es una razón fundamental del proyecto, hasta antes de 2017 solo existía una sucursal y ahora se cuentan con tres sucursales diferentes.

*Software actual de la empresa*

La empresa desde hace varios años ha trabajado con el software QVET, que es definido por la misma empresa que lo desarrolla y distribuye como: [1]

> Es el programa de gestión de clínicas y hospitales veterinarios más completo del mercado en lengua española y que más veterinarios utilizan en sus negocios.
>
> Los más de 20 años de experiencia y el extenso uso de QVET por más de 6.000 clínicas en 30 países hacen que nuestros ingenieros hayan desarrollado un sistema muy completo en el que todas las necesidades del centro veterinario estén resueltas de forma segura, estable y fácil de utilizar.

Sin embargo, la administración ante la imposibilidad de realizar determinadas funciones en el referido software o no realizarlas tal como lo necesita, ha incorporado otros softwares más concretos e incluso diseñado e implementado numerosas hojas de cálculo que, con el apoyo del lenguaje VBA, han logrado optimizar varias de estas funciones.

Por cuestiones propias de la empresa, el software contratado con QVET funciona solo de forma local, por lo que se contrataron dos licencias para las sucursales grandes y la pequeña funciona sin software, por lo que la información relacionada con los datos del personal, clientes, servicios y expedientes no se sincronizan y ni siquiera se comparten.

*Necesidad del proyecto*

En este orden de ideas lo que se pretende es, en una primera etapa, modernizar los registros que se hacen actualmente en hojas de cálculo y con VBA así como implementar funcionalidades que nos otorgan las bases de datos y el lenguaje Java.

1

En esta primera etapa el proyecto contendrá principalmente los módulos de calendarización de horarios, registros de entradas y salidas, agenda de citas y gestión de la información de los colaboradores.

## 1.2. Abstract

*En español*

El objetivo principal de este proyecto es diseñar e implementar una aplicación que apoye a la administración y a los colaboradores en la gestión del Hospital Veterinario Peninsular, en las actividades de registro de nuevos colaboradores, la programación de calendarios de trabajos, el registro de entradas y salidas y la agenda de citas.

La motivación para seleccionar este tema fue que el equipo desarrollador ha colaborado previamente con la empresa en desarrollos similares, pero con otras tecnologías y por el conocimiento empírico del equipo en lo que se refiere a las funciones y actividades de la empresa.

Para el desarrollo de este proyecto se utilizó Java como lenguaje de programación, una base de datos de MySQL en un servidor de Google Cloud Platform para el manejo de los datos, así como las herramientas Maven, Hibernate y JavaFX. Por su intuitividad fue seleccionado IntelliJ Idea Ultimate como entorno de desarrollo.

En el proyecto se desarrollaron los siguientes módulos para la administración del Hospital Veterinario Peninsular:

a) Módulo de inicio de sesión o login, para la autenticación de los usuarios y determinar los roles para el acceso al resto de módulos.
b) Módulo de administración de colaboradores, para registrar y editar la información de los colaboradores de la empresa, incluyendo además de sus datos personales, sus datos como usuario de la aplicación y sus condiciones de trabajo.
c) Módulo de calendario laboral, que permite a la administración calendarizar los turnos de trabajo del personal y contar con mecanismos de validación útiles.
d) Módulo de registro de entradas y salidas, que tiene mucha utilidad para el cálculo de nómina y el otorgamiento de bonos de puntualidad.
e) Módulo de agenda, para calendarizar los servicios que presta la empresa.

f) Módulo de incidentes, con el fin de que los usuarios puedan presentar incidentes y estos puedan ser atendidos.

*En inglés*

The main objective of this project is to design and implement an application that supports the administration and collaborators in the management of the Hospital Veterinario Peninsular, in the activities of registration of new collaborators, the programming of work calendars, the record of entries and exits, and to schedule appointments.

The motivation for selecting this topic was that the development team had previously collaborated with the company in similar projects, but with other technologies and empirical knowledge of the team regarding the functions and activities of the company.

For the development of this project, Java was used as the programming language, a MySQL database on a Google Cloud Platform server was utilized to manage the data, as well as the software tools and technologies: Maven, Hibernate, and JavaFX. Due to its intuitiveness, IntelliJ Idea Ultimate was selected as the integrated development environment.

The project developed the following modules for the administration of the Hospital Veterinario Peninsular:

a) Login module, for user authentication and determining the roles for access to the rest of the modules.
b) Employee administration module, to register and edit the information of the company's employees, including, in addition to their personal data, their data as a user of the application and their working conditions.
c) Work calendar module, which allows the administration to schedule the staff's work shifts and have useful validation mechanisms.
d) Entry and exit registration module, which is very useful for calculating payroll and granting punctuality bonuses.
e) Appointment module, to schedule the services provided by the company.
f) Incident module, so that users can present incidents, and the administration solve them.

## 1.3. Objetivos propuestos (generales y específicos)

*Objetivo general*

Diseñar e implementar una aplicación que apoye a la administración y a los colaboradores en la gestión del Hospital Veterinario Peninsular, en las actividades de registro de nuevos colaboradores, la programación de calendarios de trabajos, el registro de entradas y salidas y la agenda de citas.

Objetivos específicos:

a) Realizar la autenticación de los usuarios y permitir que puedan acceder a distintos módulos y vistas específicas dependiendo de si se han autenticado y de su rol.
b) Registrar y editar la información de los colaboradores de la empresa, incluyendo además de sus datos personales, sus datos como usuario de la aplicación y sus condiciones de trabajo.
c) Permitir a la administración calendarizar los turnos de trabajo del personal y contar con mecanismos de validación útiles.
d) Registrar las entradas y salidas en las diversas sucursales de la empresa, así como llevar un control sobre sus retardos y bonos de puntualidad.
e) Agendar citas de los servicios que presta la empresa en cada una de sus sucursales.
f) Ingresar y gestionar incidentes relacionados con la calendarización de jornadas de trabajo y registro de horarios, así como cualquier situación relacionada con la aplicación.
g) Agregar y modificar aspectos relacionados con la empresa, accesorios para el resto de los módulos como son las sucursales, los horarios de apertura, los puestos de trabajo y los tipos de jornadas de trabajo.

## 2. Metodología utilizada

Para el análisis, diseño e implementación de este proyecto se utilizaron tanto los modelos iterativo-incremental como el modelo en cascada con retroalimentación. Se utilizó el modelo iterativo-incremental en virtud de que uno de los módulos debía entregarse primero: el de la agenda de citas; pero en su mayoría se utilizó el modelo en cascada.

El modelo iterativo-incremental nos ofrece como ventajas que no se necesitan conocer todos los requisitos, que se permiten hacer entregas tempranas operativas al cliente y que las entregas facilitan la retroalimentación de los próximos entregables.[2]

Por su parte el modelo en cascada nos ofrece las ventajas de que es fácil de comprender, planificar y seguir, que la calidad del producto resultante es alta y que permite trabajar con personal poco cualificado. [3]

Como se ha mencionado, si bien se utilizó el modelo iterativo-incremental para un módulo en específico, en realidad la metodología predominante es el modelo en cascada con retroalimentación, el cual cuenta con el siguiente ciclo de vida:

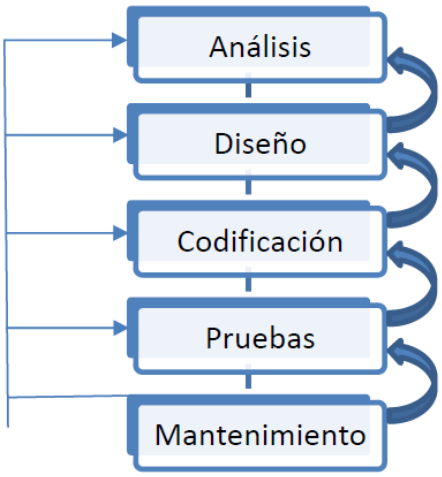


*Ilustración 1. Modelo en cascada. Fuente: Ilerna, Material Didáctico entornos de desarrollo.*

[2] Material entornos de desarrollo. P 19 y ss
[3] Material entornos de desarrollo. P 19 y ss

*Análisis*

Durante esta etapa se pretende determinar cuáles son los requisitos y las posibles alternativas y soluciones. En este sentido, cabe destacar que el equipo desarrollador está familiarizado con las funciones de la empresa y sus requerimientos, adicionalmente, en algunos módulos específicos ya se habían realizado implementaciones similares con VBA, por lo que las entrevistas y reuniones con el cliente fueron escasas, y esta etapa más que nada fue de lluvia de ideas y la definición de requisitos funcionales y no funcionales.

*Diseño*

En relación con el diseño, se realizo con base en una estructura orientada a objetos, por lo que se reflexionó primeramente que tipo de objetos se utilizarían y como se relacionarían entre ellos, y se llegó a la siguiente conclusión

- Entidades de las bases de datos
- Objetos de acceso a datos (DAO)
- Clase modelo para concentrar las variables comunes.
- Clase utilidades para concentrar los métodos comunes.
- Controladores de las vistas.

Además de definir las clases se proyectaron diagramas de flujo y pseudocódigos para la correcta comprensión y elaboración de cada módulo.

*Codificación*

Esta etapa fue sin dudas la más compleja, pues durante la codificación de cada módulo fueron surgiendo nuevas necesidades y oportunidades que hicieron que el equipo de trabajo se replanteara nuevas clases y métodos.

Para la codificación, con base en el diseño referido, se procuró mantener un código limpio y ordenado que facilitará la reutilización de código y un mantenimiento más optimizado cuando llegara el caso.

De igual forma se respetaron las convenciones sobre la escritura del código relacionadas con los comentarios, las sentencias y la identación.

*Pruebas*

Durante esta etapa, se hizo un análisis minucioso sobre que aspectos de los módulos eran sujetos a error y se llevaron al límite las opciones, de tal forma que se pudiera comprobar si el funcionamiento era el esperado.

Si bien, el equipo no es tan extenso para evitar que el programador no sea el mismo que realice las pruebas, en algunos módulos si se hicieron pruebas específicas no solo por el cliente real, sino en otra zona horaria, que fue uno de los temas más retadores en el desarrollo del proyecto.

Las pruebas realizadas fueron tanto de caja negra como de caja blanca.

*Mantenimiento*

El proyecto, al momento de la entrega de este documento, aun no se ha implementado en su totalidad, por lo que el mantenimiento como tal no ha podido realizarse; sin embargo, en los módulos que se han ya implementado: la agenda de citas y el inicio de sesión, se ha realizado un mantenimiento constante, aunque esporádico pues se ha cumplido con lo esperado por el cliente.

# 3. Tecnologías y herramientas utilizadas en el proyecto

## 3.1 Lenguaje de programación

El lenguaje de programación elegido para el desarrollo de la aplicación es Java, el cual cuenta con las siguientes ventajas: 4

a) Es el lenguaje que se utilizó mayoritariamente a lo largo de todo el ciclo formativo.
b) Es simple, en el sentido de que es directo para redactar, compilar, depurar y aprender.
c) Es orientado a objetos lo que otorga facilidades para la reutilización de código y las demás características propias de este paradigma: herencia, cohesión, abstracción, polimorfismo y encapsulación.
d) Permite la distribución computacional, en el sentido de que varios equipos de una red pueden trabajar simultáneamente.
e) Es seguro, pues cuenta con una administración de seguridad que permite definir el acceso de cada clase.
f) Asegura un buen funcionamiento de la memoria con la división que realiza entre el heap y el stack.
g) Es multihilo, pues permite que los métodos del código puedan ejecutarse secuencial o simultáneamente.

## 3.2 IDE utilizado

El IDE utilizado fue IntelliJ Idea 2020.1.3 (Ulitmate Edition), las razones de esto son variadas, pero la más destacada es su intuitividad. Si bien se podría hacer una lista detallada de los motivos de esta elección consideramos que es más claro transmitir esta idea como elemento central.

## 3.3 Herramientas y librerías utilizadas

Entre las herramientas y librerías utilizadas destacamos las siguientes:

[4] https://data-flair.training/blogs/pros-and-cons-of-java/

a) *Maven:* es una herramienta de administración de proyectos basada en un POM (modelo de objetos del proyecto) usado para la construcción de proyectos y generar sus dependencias y documentación. [5]

b) *Hibernate:* es una herramienta de mapeo de objetos relacionales para Java que asegura los servicios de consulta y persistencia de los objetos, logrando al mismo tiempo un alto rendimiento[6]. Esta herramienta fue seleccionada por sus grandes ventajas en lo que se refiere a las consultas a la base de datos.

c) *JavaFX:* es un conjunto de paquetes de gráficos y contenidos multimedia que permite a los desarrolladores diseñar, crear, probar, depurar, y despegar aplicaciones de cliente enriquecidas a través de diversas plataformas[7]. Fue utilizada en virtud de que, actualmente es la herramienta más utilizada dentro del entorno Java para desarrollar interfaces.

[5] https://www.geeksforgeeks.org/introduction-apache-maven-build-automation-tool-java-projects/#:~:text=Maven%20provides%20project%20information%20(log,etc%20without%20doing%20any%20scripting.
[6] https://www.mindfiresolutions.com/mindfire/Java_Hibernate_JDBC.pdf
[7] https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

## 4. Estimación de recursos y planificación

Para la elaboración de este proyecto se utilizó el siguiente diagrama:

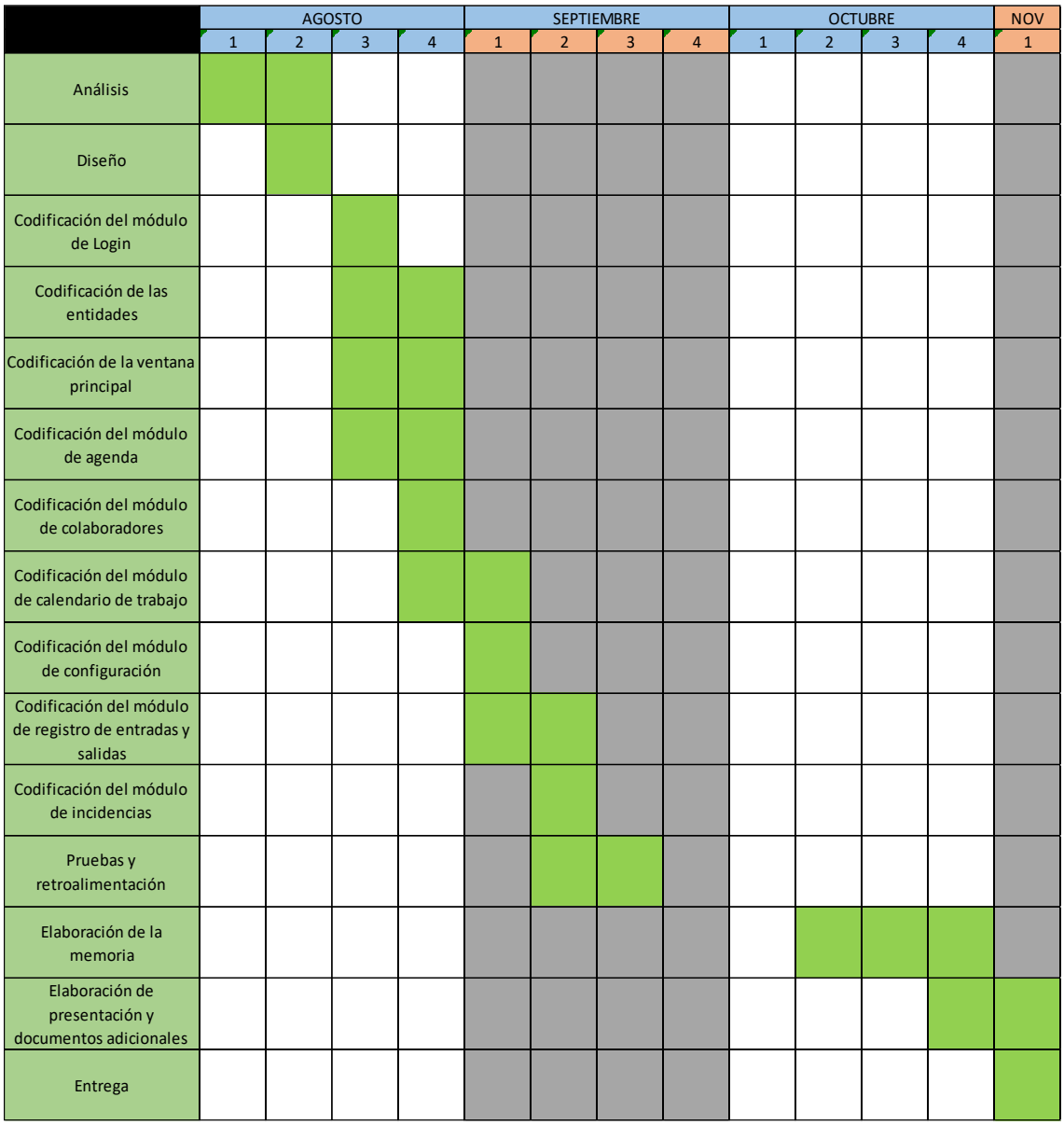


*Ilustración 2. Diagrama de Gantt. Fuente: Elaboración propia.*

# 5. Desarrollo del proyecto

## 5.1 Inicio

### 5.1.1 Finalidad

El proyecto, de acuerdo con los objetivos generales y específicos planteados, deberá ser capaz de registrar nuevos colaboradores; permitir calendarizar los horarios; posibilitar la agenda de citas; proporcionar un mecanismo para el registro de entradas y salidas; y posibilitar a los usuarios a levantar incidencias, agregar y configurar aspectos adicionales como las sucursales, los tipos de jornadas de trabajo, los horarios de las sucursales y los puestos de trabajo.

La finalidad es tener en una sola aplicación mecanismos suficientes para que la administración de la empresa pueda llevar un registro adecuado de las jornadas laborales del personal, y con este mecanismo poder hacer un cálculo adecuado de la nómina.

### 5.1.2 Necesidades

Las necesidades del proyecto son muy variadas y se analizarán con base en los módulos con que cuenta el proyecto, a saber: Main Window, Login, Manage Users, Work Schedule, Schedule, Attendance control, Configurations, así como las clases de apoyo: Model y Utilities.

*Ventana principal (Main Window)*

Las necesidades de este módulo son las siguientes:

a) Cuando se inicie la aplicación deberá primero lanzar una ventana inicial que anuncie al usuario que la aplicación se está cargando.
b) En lo que se carga la ventana principal, se realizará la conexión por medio de Hibernate, en virtud de que las funcionalidades importantes de la aplicación necesitan una conexión a una base de datos.
c) La ventana principal permitirá el acceso al resto de los módulos.
d) La ventana principal mostrará al usuario logueado, en caso de haber uno.
e) La ventana principal restringirá el acceso a determinados módulos, dependiendo del rol que tengan los usuarios o de si están logueados o no.

*Módulo de inicio de sesión (Login)*

Las necesidades de este módulo son las siguientes:

a) Dependiendo de si se abre login de la ventana principal o no, regresará a esta o a la ventana de origen.
b) Validará si el usuario existe y si la contraseña coincide
c) En la base de datos se registrará cada usuario que se loguee así mismo se registrará la salida del último usuario.
d) Se permitirá al usuario cambiar su contraseña desde esta sección.
e) Para cambiar la contraseña, el usuario deberá ingresar su contraseña anterior y confirmar su nueva contraseña.

*Módulo de colaboradores (Manage Users)*

a) Apenas se inicie, para ahorrar tiempos de carga, cargar en memoria las tablas con la información de los colaboradores y el catálogo de puestos.
b) Que la información que sea algo confusa cuente con tooltips para dar claridad.
c) Que permita diferentes visualizaciones dependiendo de la acción: solo visualizar, editar o agregar nuevo.
d) Que dependiendo de la visualización se activen o desactiven botones.
e) Que en caso de que el usuario no sea administrador o gerente, no permita editar o agregar nuevos colaboradores.
f) Que cuente con combobox de usuarios, puestos de trabajo, formas de pago y roles.
g) Que permita guardar nuevos usuarios o cambios.
h) Que al momento de editar o grabar, realice validaciones sobre si el código alfabético y el código numérico ya existen o si los nombres tienen un mínimo de caracteres.
i) Que los valores que dependen de otros llenados manualmente se calculen automáticamente antes de insertar o actualizar.
j) Que haya un botón de refrescar para que los valores automáticos se actualicen con base en los llenados manualmente.
k) Que se haga una propuesta automática del código alfabético de los usuarios con base en lo llenado en nombre y apellido.

*Módulo de calendario laboral (WorkSchedule)*

a) Que cuente con la posibilidad de cambiar de vista para organizar por colaborador o por sucursales.
b) Cambiar las opciones dependiendo de si se trata de un usuario o de un gerente o administrador.
c) Cargar, apenas iniciando o cada que vez que se recargue:
    1. La lista de jornadas de trabajo por semana, en caso de existir;
    2. Los colaboradores activos;
    3. Las sucursales;
    4. Los horarios de apertura;
    5. Los puestos de trabajo;
    6. Los tipos de jornadas de trabajo;
d) Crear una lista de apertura y cierre de cada sucursal por día (21 horarios).
e) En caso de que la vista sea por sucursal:
    1. Dividir el panel principal en tres secciones por cada una de las sucursales.
    2. Cargar el número de filas por cada sucursal, dependiendo del número de registros de la base de datos.
    3. Permitir agregar o eliminar filas por cada sucursal.
    4. Que cada bloque de día y colaborador permita establecer el código alfabético del colaborador, el horario de entrada y el horario de salida.
    5. Que la selección de combobox sea con base en la lista de colaboradores activos.
    6. Que se genere automáticamente la lista de colaboradores que descansan en cada día.
f) En caso de que la vista sea por colaborador:
    1. Que genere una fila por cada uno de los colaboradores.
    2. Que cada bloque de día y colaborador permita establecer el tipo de jornada de trabajo, la hora de entrada y de salida y la sucursal.
    3. Que la información del tipo de jornada de trabajo y de la sucursal sea a través de combobox.
    4. Que dependiendo de la selección del tipo de jornada de trabajo o de la sucursal se actualicen en resto de información en consecuencia.
g) Cuestiones comunes para ambas vistas:
    1. Que se actualicen las fechas de acuerdo con la semana seleccionada.

h) Que, cuente con selector de fecha, de la semana siguiente, de la anterior o de esta semana, para actualizar la información que se está cargando o que se va a registrar.
i) Que por medio de un botón se permita volver a cargar la información tal como está registrada en la base de datos.
j) Que por medio de un botón se permita hacer un refresco de los datos aun no grabados en la base de datos, por ejemplo, los colaboradores que descansarán ese día.
k) Que, por medio de un botón, se puedan validar los siguientes datos:
    1. Que las horas trabajadas por un colaborador sean congruentes con la jornada de trabajo para la que fue contratado.
    2. Que la jornada de trabajo esté dentro del horario de apertura de la sucursal correspondiente.
    3. Que, dependiendo del tipo de jornada de trabajo seleccionada, cuente o no con una sucursal.
    4. Que, dependiendo del tipo de jornada de trabajo seleccionada, cuente o no con una jornada de trabajo.
    5. Que la hora de entrada no sea posterior a la hora de salida.
    6. Que haya un recepcionista a la hora de cierre.
    7. Que haya un médico o un asistente a en cada hora en que estén abiertas las sucursales de Urban y de Harbor.
    8. Que ningún colaborador cuente con dos jornadas de trabajo en el mismo día, independientemente de si es en una o más sucursales.
    9. Que se notifiquen los cambios que habían antes con los colaboradores en comparación con los nuevos.
l) Que, con base en la validación, se reporte una tabla al usuario para conocer las posibles inconsistencias.
m) Que los errores sean advertencias o errores, en el primer caso, permitir cambios, en el segundo, prohibir guardar en la base de datos.
n) Que, por medio de un botón, se haga una visualización de cuántas horas de personal médico hay en cada día en cada sucursal.
o) Que, por medio de un botón, se pueda visualizar un apartado gráfico del horario, el cual tendrá las siguientes necesidades:
    1. Que, por cada sucursal, determine cual es el horario mínimo de entrada y el horario máximo de salida, para determinar el número de celdas.
    2. Que, establezca por cada sucursal una fila por cada hora disponible.

3. Que, con base en la semana seleccionada, actualice las fechas de cada día.
4. Que rellene la tabla con el código de cada colaborador, dependiendo de sus jornadas de trabajo.
5. Que pinte el fondo de cada celda, dependiendo de la posición de cada colaborador.
6. Que por cada día se generen tantas columnas como colaboradores irán ese día a la sucursal a laborar.
7. Que, en caso de ser una sucursal exprés, solo establezca dos filas, una para la hora de inicio y otra para la hora que esté en la mitad.

p) Que, por medio de un botón, se permita la inserción o actualización en la base de datos; sin embargo, antes de hacer esto, deberá realizar la validación referida anteriormente.

q) Que, una vez determinado que se va a grabar en la base de datos, hacer una lista para establecer qué datos se van a insertar y cuáles actualizar para hacer más ágil el proceso de guardado.

r) Que, por medio de un botón, se permita hacer una captura de pantalla del horario para compartir con los colaboradores.

s) Que, por medio de un botón, se otorgue la posibilidad de copiar la calendarización de una semana a otras semanas, apartado que contará con las siguientes necesidades:

1. Que pregunte al usuario de que semana quiere copiar los datos, a que semana quiere copiarlos y el número de semanas después de esta que desee sea insertada o reemplazada también la información de la semana original.
2. Que, en caso de que la semana a copiar no cuente con información, impedir la acción de copiado.
3. Que confirme con el usuario si desea reemplazar la información en las semanas que ya contaban con información previa registrada.
4. Que realice el copiado y pegado e informe al usuario del éxito de la acción.

t) Que permita al usuario eliminar todos los datos registrados de una semana en concreto.

u) Que permita al usuario establecer incidentes sobre el calendario semanal.

*Módulo de agenda (Schedule)*

a) Que cuente con un calendario para poder agregar citas en cada una de las sucursales.
b) Que, por medio de selectores de fecha, se pueda mover a la semana anterior, a la siguiente, a la actual o a cualquier otra fecha.
c) Que solo los usuarios que se hayan logueado tengan la posibilidad de agregar nuevas citas.
d) Que las citas se distingan cromáticamente de acuerdo con la sucursal.
e) Que tanto las fechas como las horas se ajusten a lo seleccionado por el usuario, así como a los horarios de apertura.
f) Que por cada cita se pida al usuario el registro de los datos correspondientes.
g) Que las citas se muestren en el calendario solo el nombre del servicio y de la mascota.
h) Que se puedan registrar varias citas a la misma hora y el mismo día.
i) Que se permita al usuario las sucursales y los médicos que desea visualizar.
j) Que sea posible editar las citas.
k) Que se advierta en caso de que se trate de registrar una cita para una fecha anterior.

*Módulo de registro de entradas y salidas (Attendance control)*

a) Que la vista inicial corresponda a los datos del usuario registrado, para que pueda registrar inmediatamente su entrada o salida.
b) Que, apenas abierto, se puedan observar, al menos los siguientes datos: código del colaborador, el último registro realizado, su próxima entrada o salida, el estado actual en comparación con su próxima entrada o salida, la sucursal y la acción a registrar.
c) Que la vista de gerente o administrador cuente con botones adicionales para editar los registros.
d) Que cuando el usuario desee registrar, se validen algunos datos, de tal forma que se muestren advertencias o errores.
    1. Entre las advertencias están registrar en una sucursal diferente de la que le corresponde de acuerdo con el calendario laboral o que trate de registrar dos salidas o dos entradas seguidas, aun siendo días diferentes, o que no tenga un horario calendarizado para ese día.
    2. Entre los errores se encuentran registrar dos veces la misma acción en el mismo día o bien que no se haya llenado correctamente.

e) Que se permita cambiar el usuario desde este mismo módulo.
f) Que, por medio de un botón, se otorgue la posibilidad de revisar los registros realizados, para lo cual se tomarán en cuenta las siguientes necesidades:
    1. Que la vista principal refleje los datos del registro: colaborador, sucursal, fecha y hora, acción, el estado (en tiempo o no) y los minutos de diferencia.
    2. Que se permita avanzar a la siguiente o anterior quincena.
    3. Que, por medio de un botón, se puedan visualizar los registros de entrada y salida, que debieron ocurrir y que no ocurrieron.
    4. Que, por medio de un botón, el usuario pueda registrar nuevos incidentes.
    5. Que se pueda visualizar de forma instantánea el número de retardos, la situación con respecto al bono de puntualidad y el número de registros faltantes.
g) Que, en caso de usuarios con los roles de gerente o de administrador, puedan entrar a una vista especial para editar los registros.

*Módulo de incidentes (Incidents)*

a) Que el usuario pueda crear incidentes.
b) Que el usuario pueda conocer el estado de los incidentes realizados en determinadas fechas.
c) Que los administradores puedan conocer incidentes y darlos por resueltos.
d) Que los administradores, para resolver un incidente, puedan pasar directamente a editar el calendario o el registro de horarios.

*Módulo de configuraciones (Configurations)*

a) Que permita realizar diferentes configuraciones menores como son las sucursales los tipos de jornada de trabajo y los puestos de trabajo
b) Que el usuario pueda en cada apartado agregar, editar o eliminar.

*Aspectos communes (Model y utilitiesl)*

a) Que, desde esta misma clase se pueda acceder a diferentes objetos y variables que sean comunes para toda la aplicación.
b) Que cuente con métodos que sean comunes para toda la aplicación.

*Entidades (Entities)*

a) Que cuente con entidades de bases de datos sobre: citas, registro de horarios, sucursal, colaborador, información detallada del colaborador, incidente, puesto de trabajo, log, horarios de apertura, usuario, tipo de jornada de trabajo y jornada de trabajo.

## 5.2. Análisis

### 5.2.1 Diagramas entidad-relación

En virtud de que la aplicación cuenta con trece entidades, consideramos oportuno hacer diversos diagramas. Para efectos de simplificar los diagramas, solo se pondrán los cinco atributos más importantes, en caso de que la entidad en cuestión tenga un número mayor.

### 5.2.1.1 Diagramas relacionados con la entidad colaborador

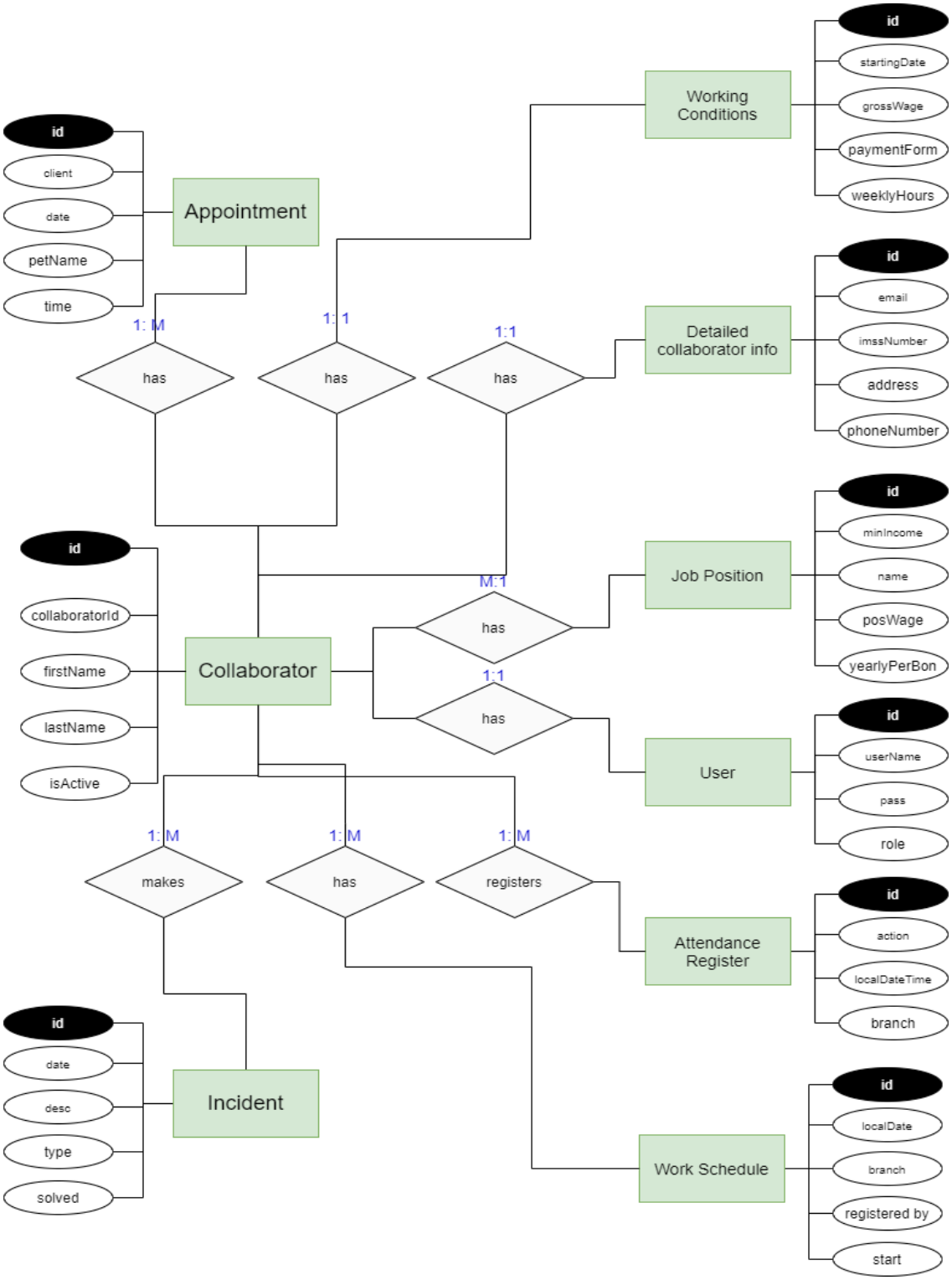


*Ilustración 3. Diagrama entidad-relación. Fuente: Elaboración propia.*

### 5.2.1.2 Diagramas relacionados con la entidad sucursal

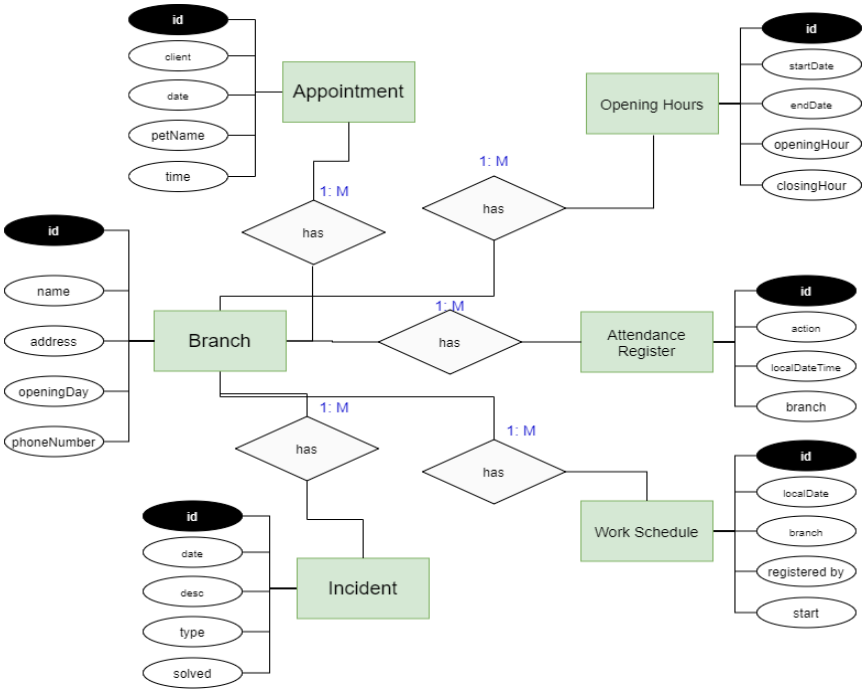


*Ilustración 4. Diagrama entidad-relación. Fuente: elaboración propia.*

### 5.2.1.2 Otros

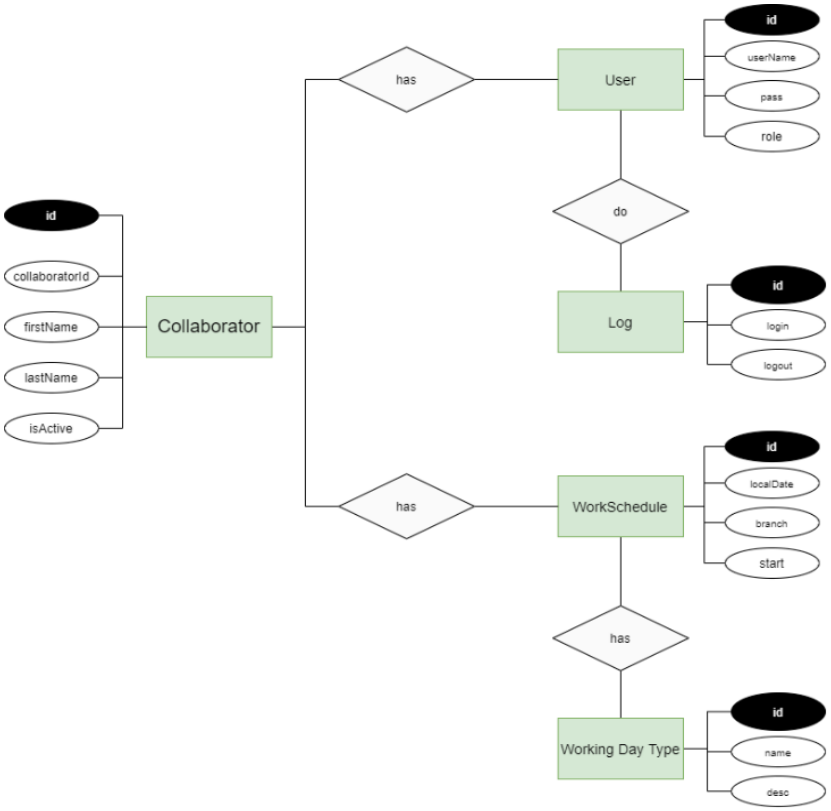


*Ilustración 5. Diagrama entidad-relación. Fuente: Elaboración propia.*

### 5.2.2 Diagramas de clases

Realmente poder hacer un diagrama de clases que sea útil para representar lo que se espera que represente un diagrama de este tipo es ocioso, pues el proyecto cuenta con un gran número de clases que hacen que sus relaciones sean difíciles de proyectar e imposibles plasmar en un documento de texto.

Es por ello que, como hemos hecho anteriormente, subdividiremos la presentación en módulos.

Por otro lado, en el diagrama de clases con sus relaciones no se presentarán ni los métodos ni los campos, sin embargo la representación gráfica de algunas clases con estos elementos se hará, en algunos casos, en el apartado 5.3 de este documento.

Finalmente, la última prevención es que gran parte de las vinculaciones entre clases ocurren mediante archivos de configuración xml (JavaFx), los cuales no están representados, por lo que diversas relaciones no se presentarán a pesar de estar presentes.

#### 5.2.2.1 Main

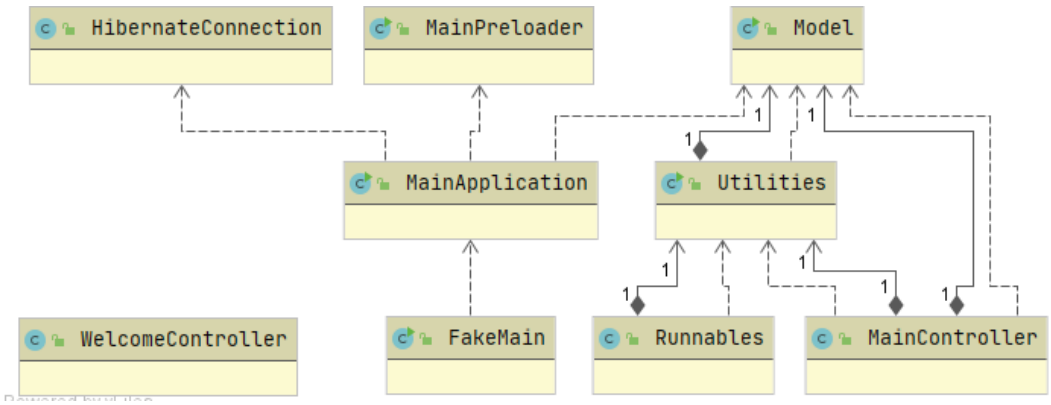


*Ilustración 6. Diagrama de clases. Fuente: Elaboración propia.*

### 5.2.2.2 Login

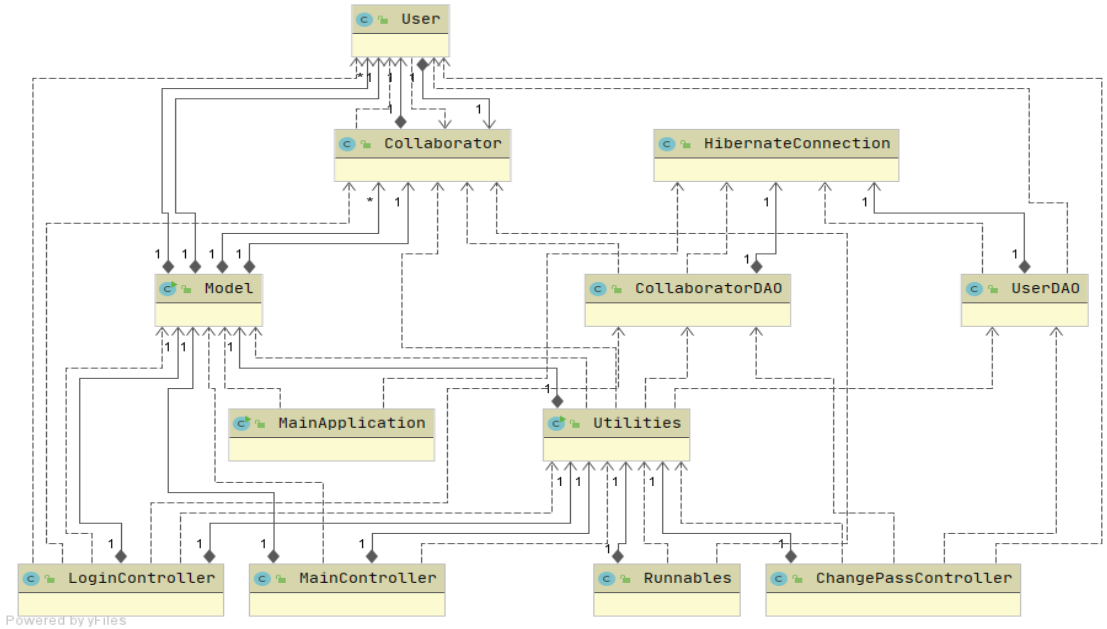


*Ilustración 7. Diagrama de clases. Fuente: Elaboración propia.*

### 5.2.2.3 Manage users

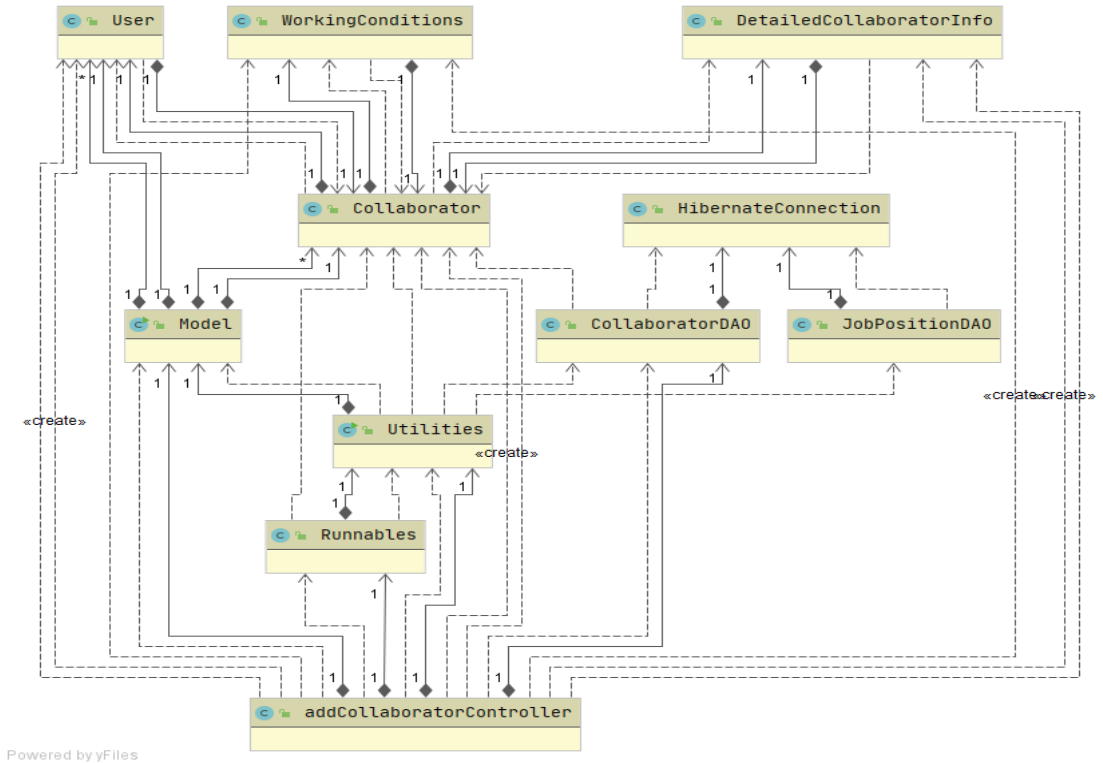


*Ilustración 8. Diagrama de clases. Fuente: Elaboración propia.*

### 5.2.2.4 WorkSchedule

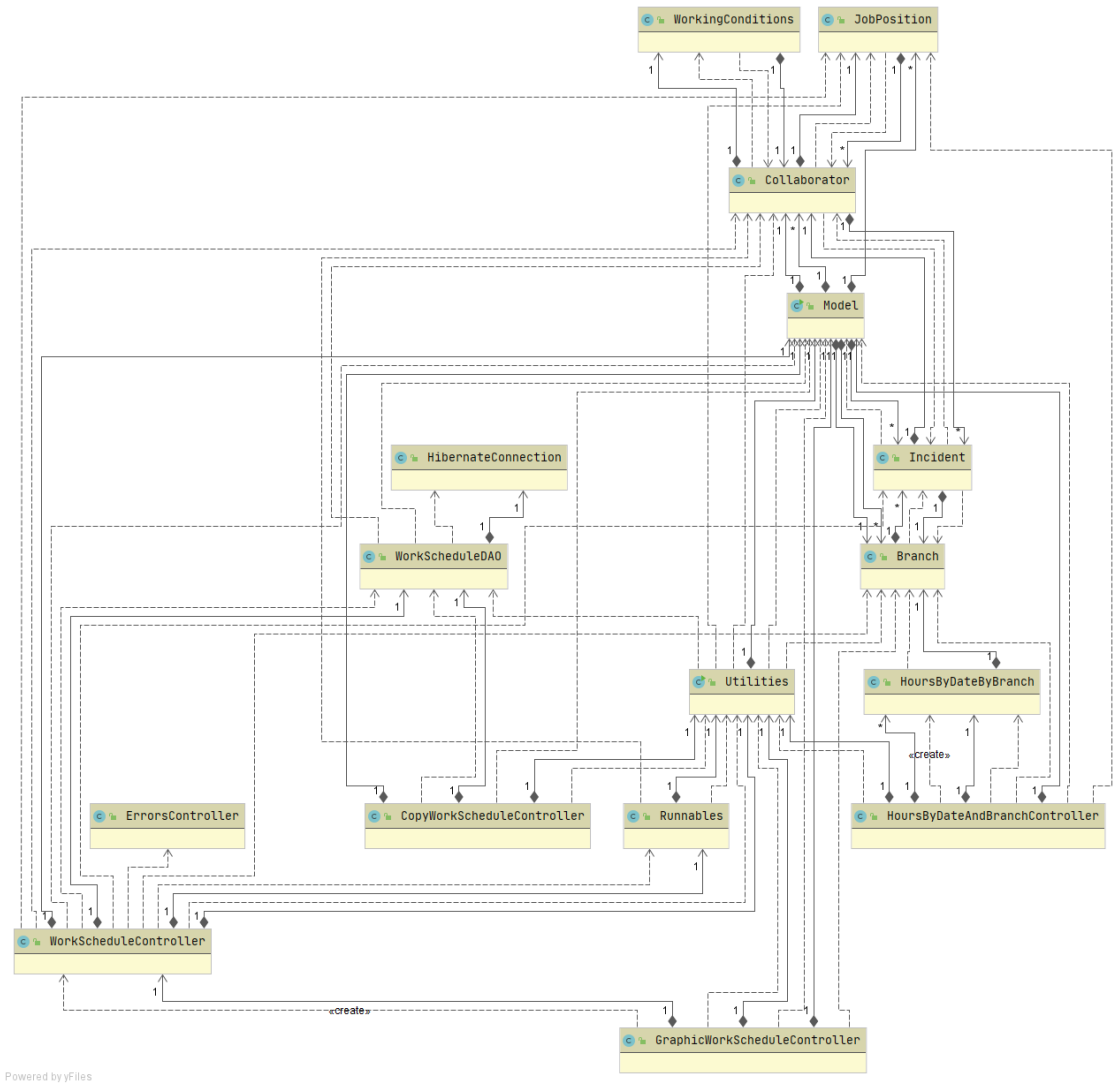


*Ilustración 9. Diagrama de clases. Fuente: Elaboración propia.*

5.2.2.5 Schedule

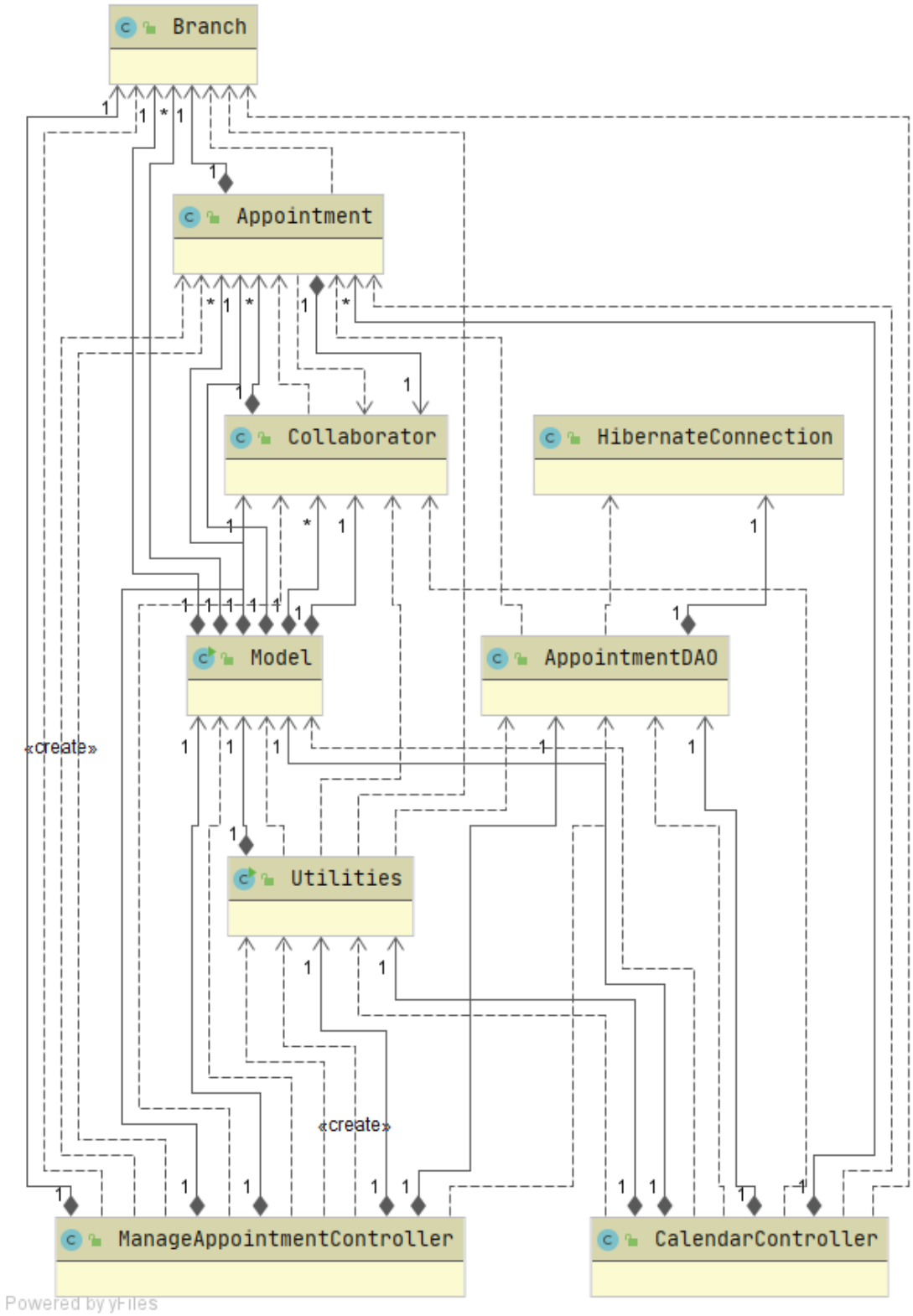


*Ilustración 10. Diagrama de clases. Fuente: Elaboración propia.*

### 5.2.2.6 Attendance control

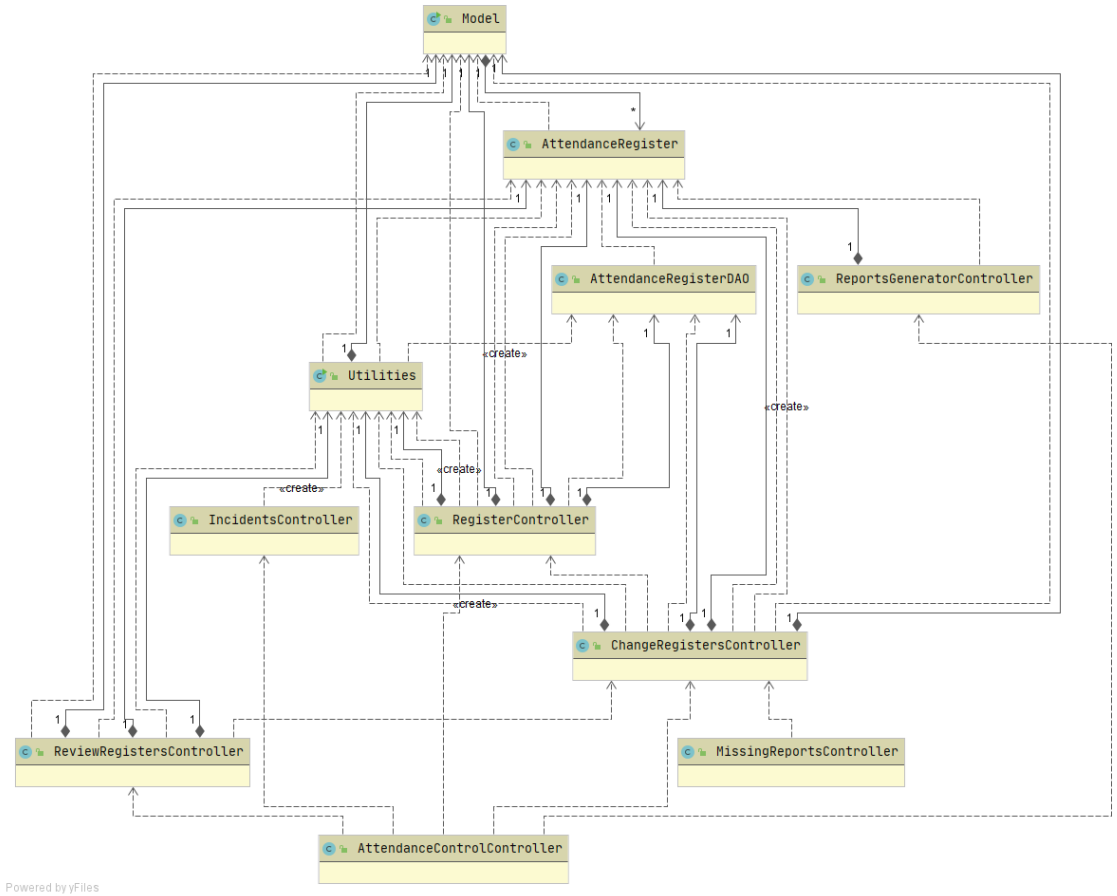


*Ilustración 11. Diagrama de clases. Fuente: Elaboración propia.*

### 5.2.2.7 Incidents

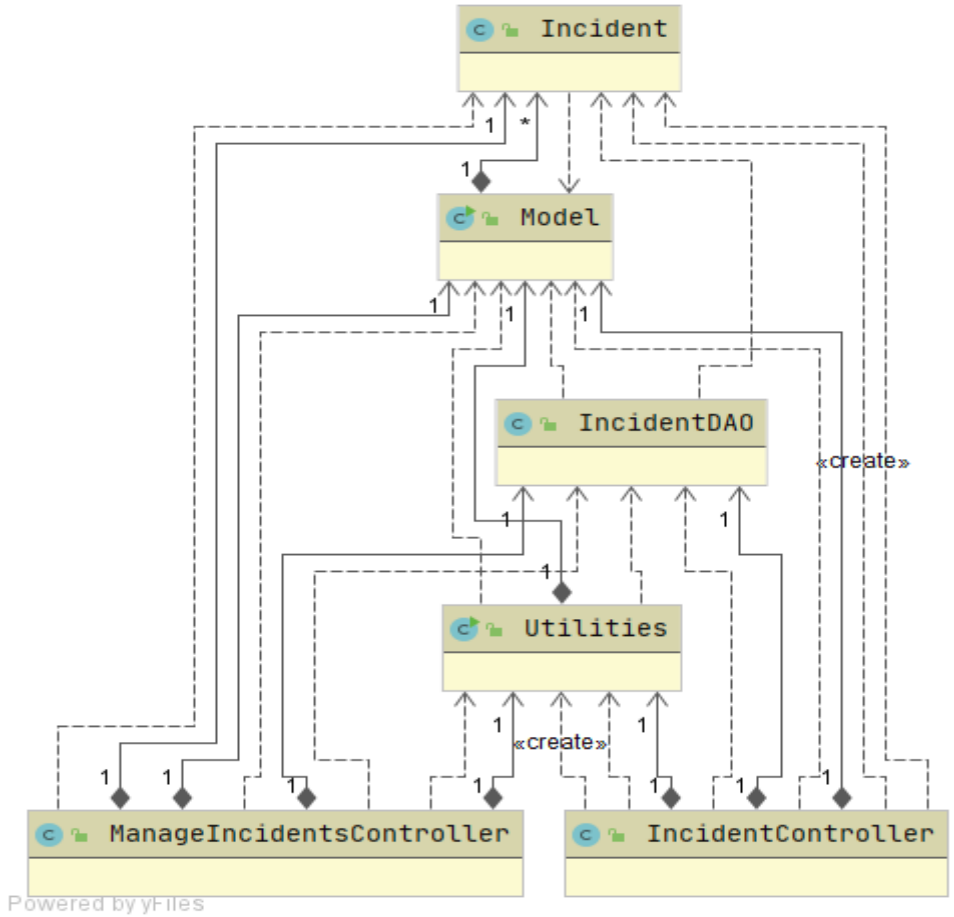


*Ilustración 12. Diagrama de clases. Fuente: Elaboración propia.*

### 5.2.2.8 Configurations

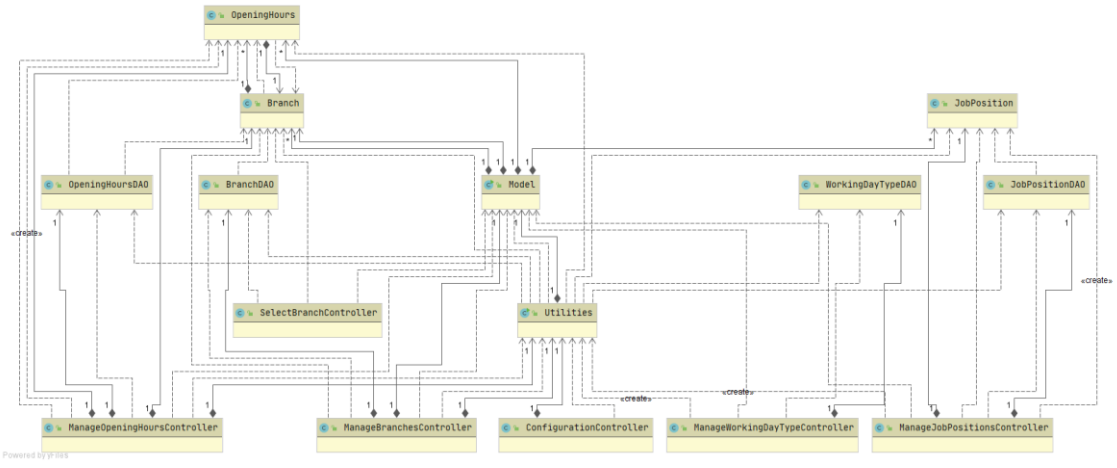


*Ilustración 13. Diagrama de clases. Fuente: Elaboración propia.*

### 5.2.3 Diagramas de casos de uso

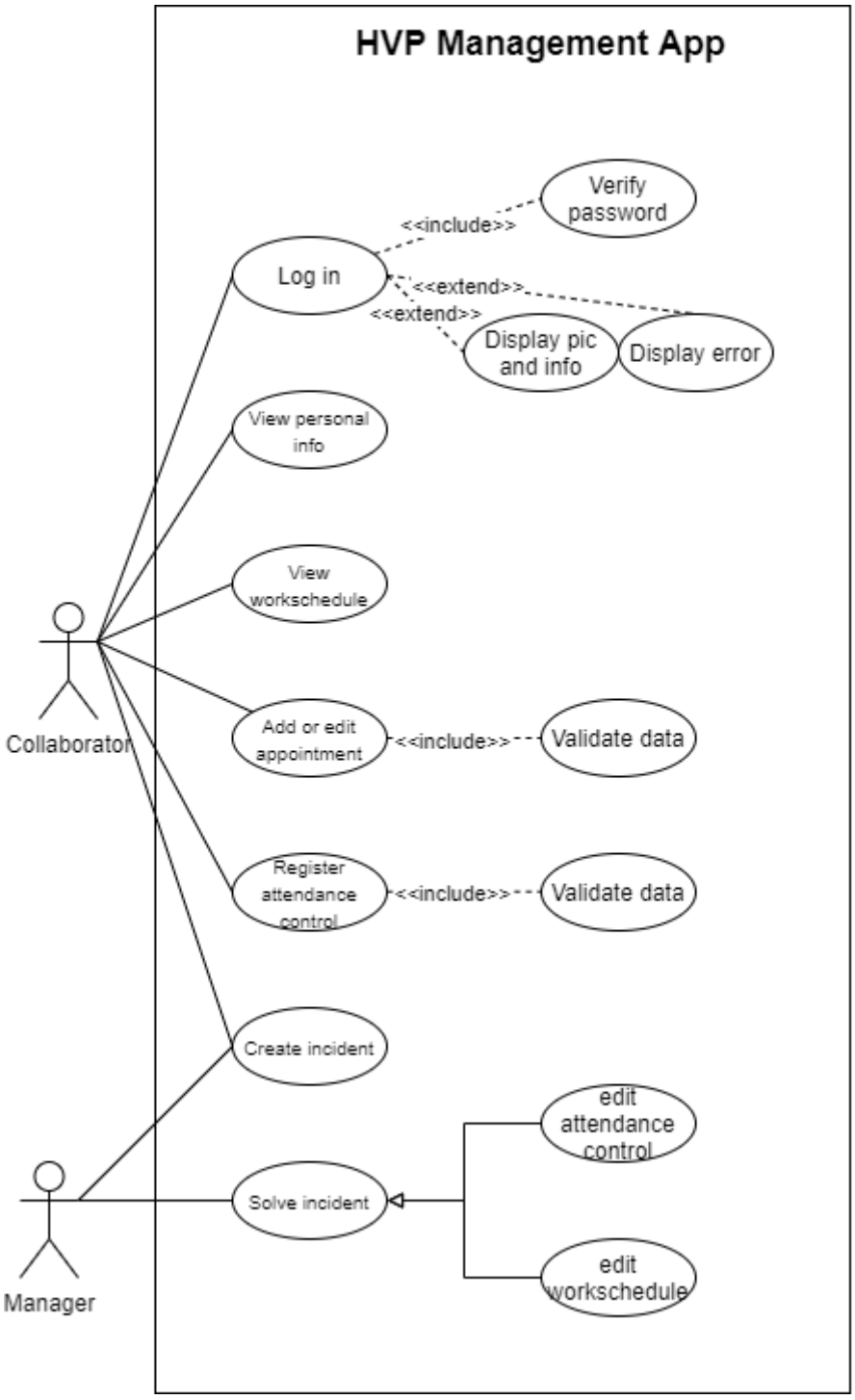


*Ilustración 14. Diagrama casos de uso. Fuente: elaboración propia.*

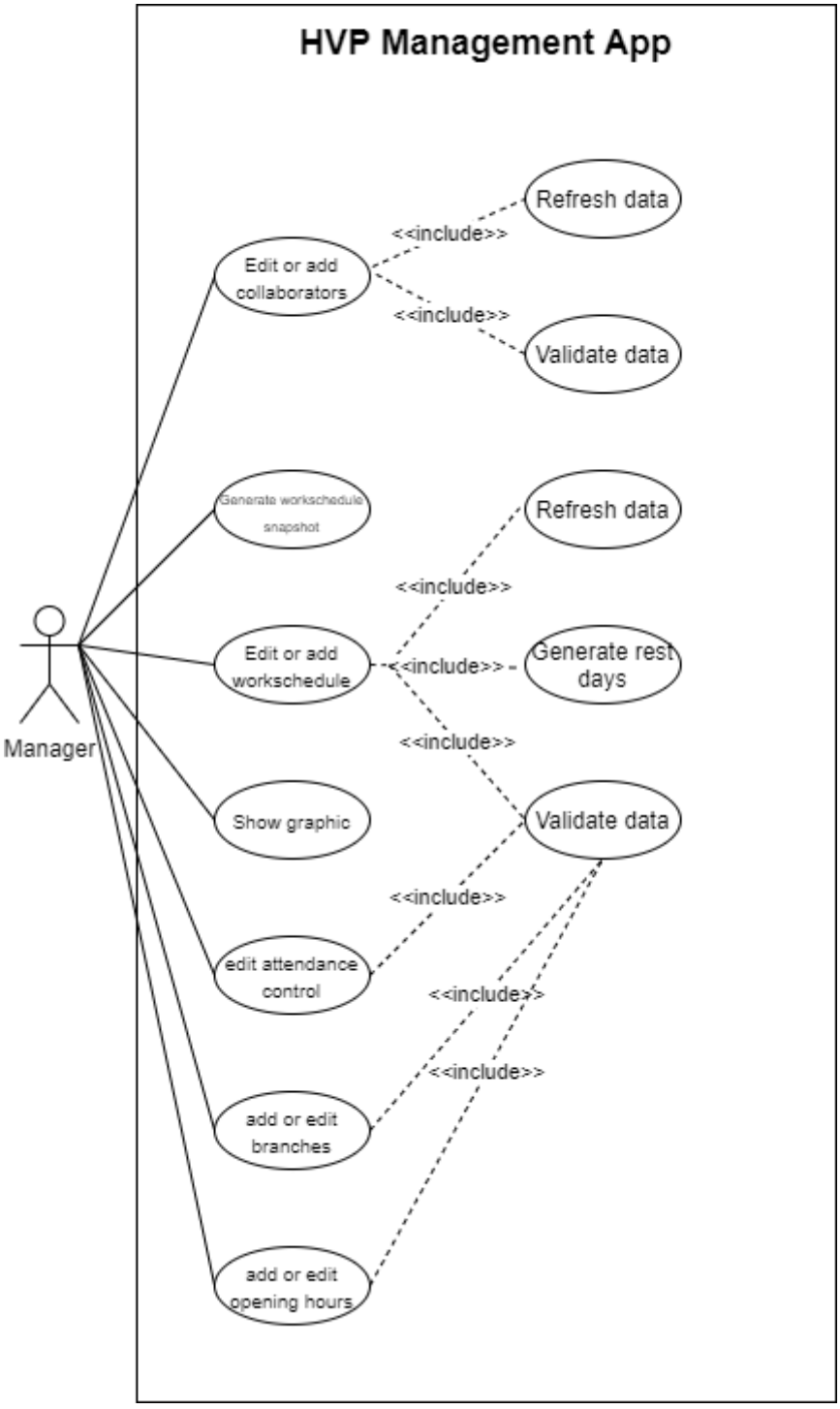


*Ilustración 15. Diagrama casos de uso. Fuente: elaboración propia.*

### 5.2.4. Tablas de requisitos funcionales y no funcionales

Los requisitos funcionales se presentan en la siguiente tabla:

| Id | Requerimiento funcional |
|---|---|
| RF01 | Los usuarios deben haber iniciado sesión para incorporar nueva información a la base de datos. |
| RF02 | Los usuarios que no sean administradores no podrán editar datos de otros colaboradores. |
| RF03 | Todos los inicios de sesión serán registrados. |
| RF04 | Los usuarios no podrán visualizar información sensible o confidencial que no sea propia. |
| RF05 | Ni los usuarios ni el gerente podrán cambiar los id de los registros de las bases de datos. |
| RF06 | Los administradores podrán insertar o editar colaboradores, sucursales, registros de entrada y de salida y calendario de horarios. |
| RF07 | Los usuarios podrán registrar sus entradas y salidas. |
| RF08 | Los usuarios podrán agendar y editar citas. |
| RF09 | Los usuarios podrán generar incidentes para que sean atendidos por los administradores. |
| RF10 | Los administradores podrán resolver incidentes. |
| RF11 | El sistema validará todos los datos antes de registrarlos. |
| RF12 | El sistema realizará todos los cálculos automáticos previos necesarios antes de registrar la información en la base de datos. |

Los requisitos no funcionales se presentan en la siguiente tabla:

| Id | Requerimiento no funcional |
|---|---|
| NF01 | El usuario deberá contar con la plataforma Java instalada en su ordenador, en su versión 8. |
| NF02 | El sistema operativo deberá ser Windows. |
| NF03 | El equipo del usuario deberá contar con una memoria RAM de 1GB. |
| NF04 | El equipo del usuario deberá contar con una capacidad de almacenamiento de al menos 100 MB. |
| NF05 | Los datos de almacenamiento de la base de datos serán en la nube, a través de la plataforma Google Platform. |
| NF06 | En los diferentes módulos del sistema, antes de realizar modificaciones a la base de datos, se validarán los posibles errores y advertencias, que serán indicados al usuario. |
| NF07 | La aplicación podrá funcionar en diferentes sucursales simultáneamente. |
| | La aplicación deberá representar correctamente la fecha en que se registren los eventos, a pesar de que sea consultado en una región con zona horaria diferente. |
| NF08 | Para agilizar la manipulación de la información, en cada módulo se cargarán en memoria todos los objetos que se relacionen con los parámetros de entrada, por ejemplo la fecha. |

## 5.3. Diseño

Este apartado se explicará en relación con la estructura del proyecto, en este sentido este esta esquematizado de la siguiente manera:

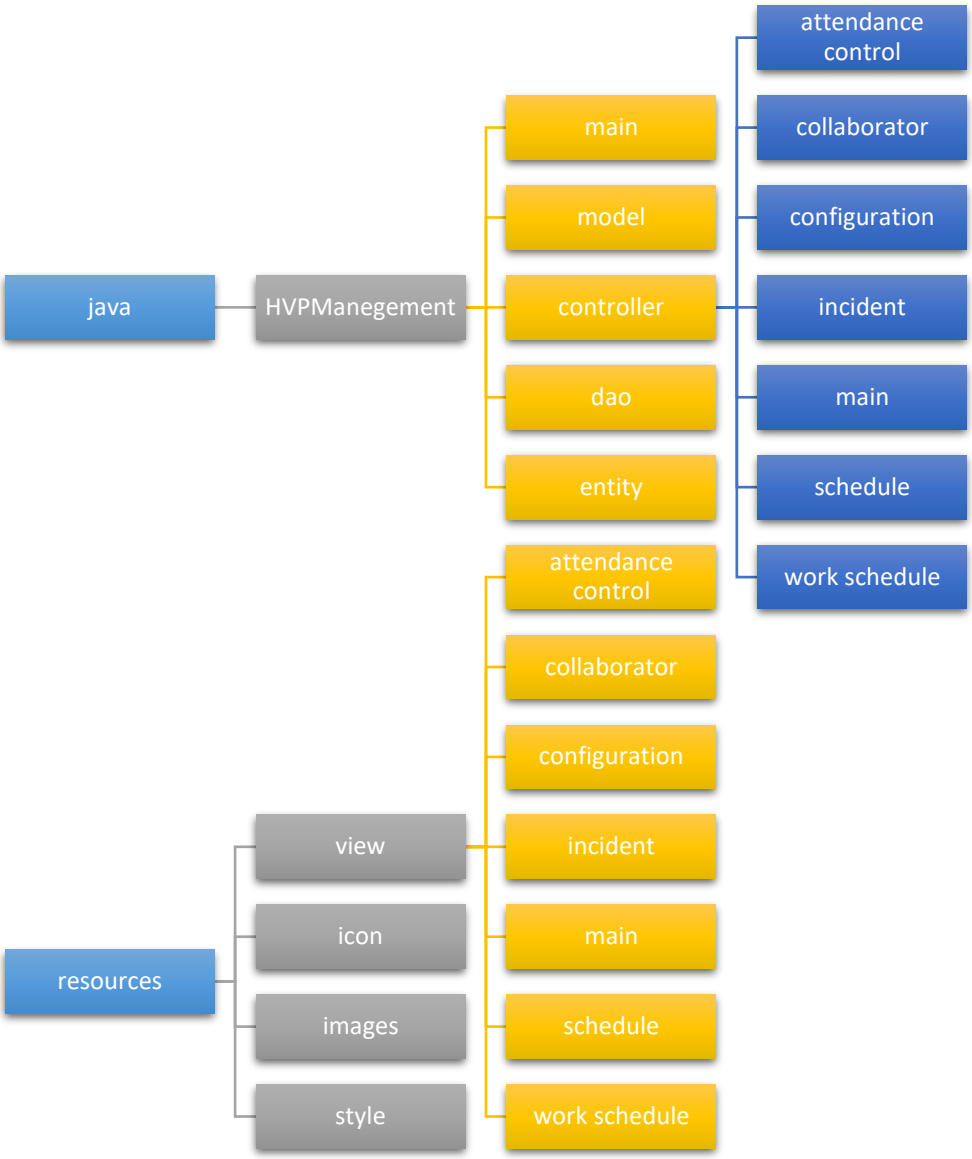


*Ilustración 16. Esquema de contenido. Fuente: Elaboración propia*

### 5.3.1 Java

En este apartado están las clases del proyecto y se esquematizaron en un principio en controladores, objetos e acceso a datos, entidades, main y el modelo.

#### 5.3.1.1 Main

En este directorio están las clases iniciales de la aplicación, es decir, desde las cuales se inicia su ejecución, y su contenido es el siguiente:

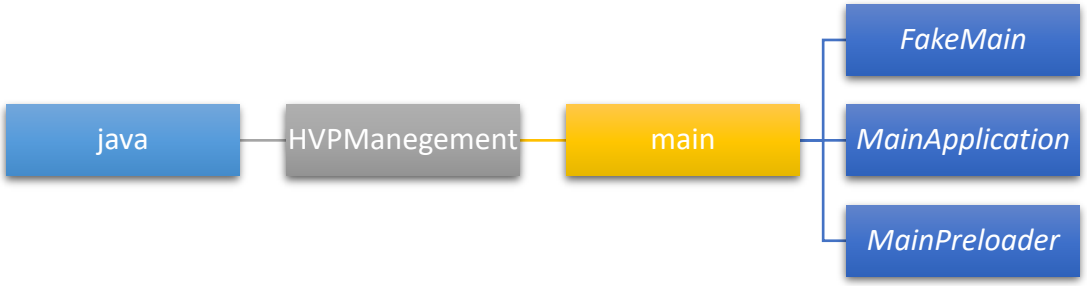


*Ilustración 17. Esquema de contenido. Fuente: Elaboración propia*

*FakeMain*

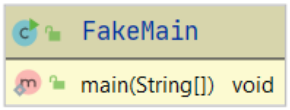


*Ilustración 18. Diagrama de clase. Fuente: Elaboración propia*

Esta clase es totalmente auxiliar y surge por la imposibilidad de que el artefacto o la aplicación pueda ejecutarse directamente, por lo que lo único que contiene esta clase es un método main que invoca la clase *MainApplication.*

*MainApplication*

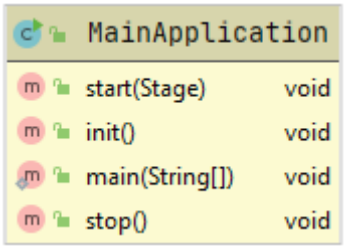


*Ilustración 19. Diagrama de clase. Fuente: Elaboración propia*

Los métodos de esta clase son:

a) Main: Por medio del método launchApplication establece el preloader.
b) Start: Llama a la vista main (la interfaz principal de usuario), poniendo los atributos por defecto.
c) Init: Crea la conexión por medio de Hibernate, esto en realizad lo hace antes de llamar a *start,* de tal forma que se visualice el preloader en lo que se carga la ventana principal.
d) Stop: Cuando se cierra la aplicación registra un log, por medio de la clase LogDAO, de salida de sesión del usuario.

*MainPreloader*

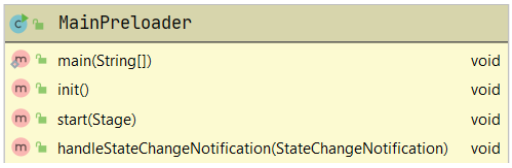


*Ilustración 20. Diagrama de clase. Fuente: Elaboración propia*

Los métodos más importantes de esta clase son:

a) Start: Por medio de este método se llama a una ventana previa que sirva para informar al usuario que la aplicación se está conectando con la base de datos.
b) HandleStateChangeNotification: Este método se sobreescribe para que una vez terminados de cargar los métodos *init* y *start* de la clase MainApplication, se cierre la ventana del preloader.

#### 5.3.1.2 Model

En este directorio están las clases comunes de la aplicación, que no se ajusten a otra clasificación, las principales clases son model y utilities:

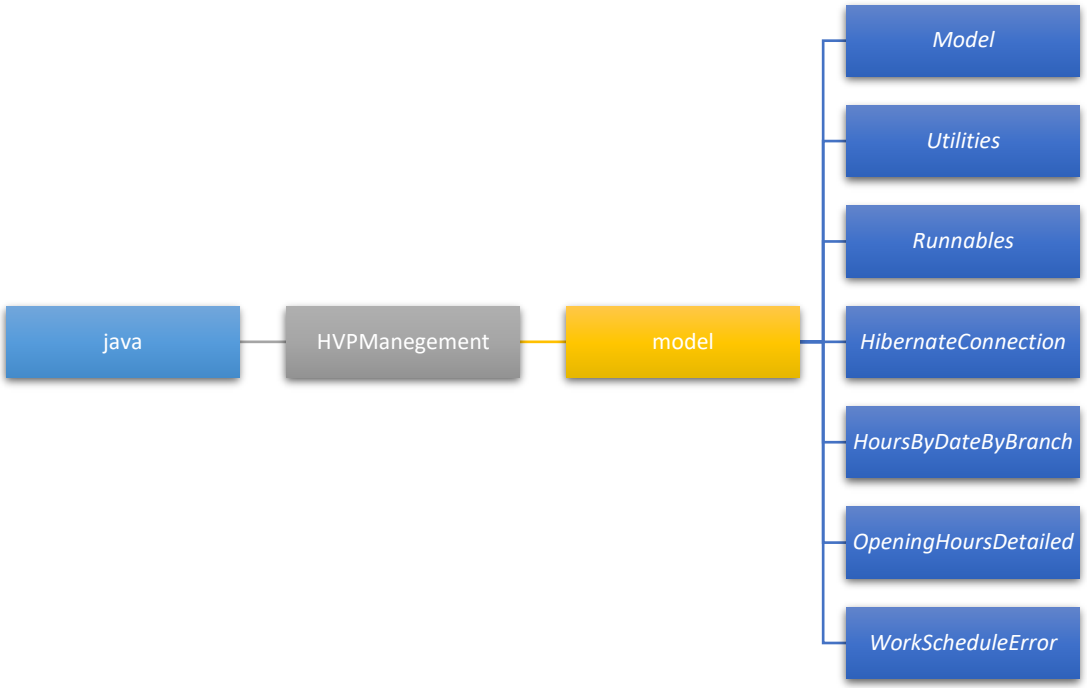


*Ilustración 21. Esquema de contenido. Fuente: Elaboración propia*

*Model*

Esta clase es un singleton, por lo que cuenta con una sola instancia. Como se puede observar contiene mayoritariamente instancias de objetos que son utilizados por uno o más módulos de la aplicación, resulta ocioso detallar cada uno de estos, por lo que explicaremos los más importantes:

a) Formateadores: Que sirven para dar formato a fechas, dependiendo de los resultados esperados en la interfaz o los esperados para que sean capturados por el usuario.
b) Listas de objetos: Son listas provisionales de objetos que en la mayoría de los casos, hacen que el rendimiento de los módulos sean más eficaces, entre estos encontramos: listas de citas, de registros de entradas y salidas, sucursales, colaboradores, horarios de apertura y calendarios de trabajo.
c) Valores individuales temporales: son valores individuales que se cargan en memoria para una actividad en concreta pero recurrente en los distintos módulos como la lista de errores, el usuario que ha iniciado sesión, el rol de este, entre otros.
d) Constantes: son los constantes que prevalecen en los módulos como los días de la semana.

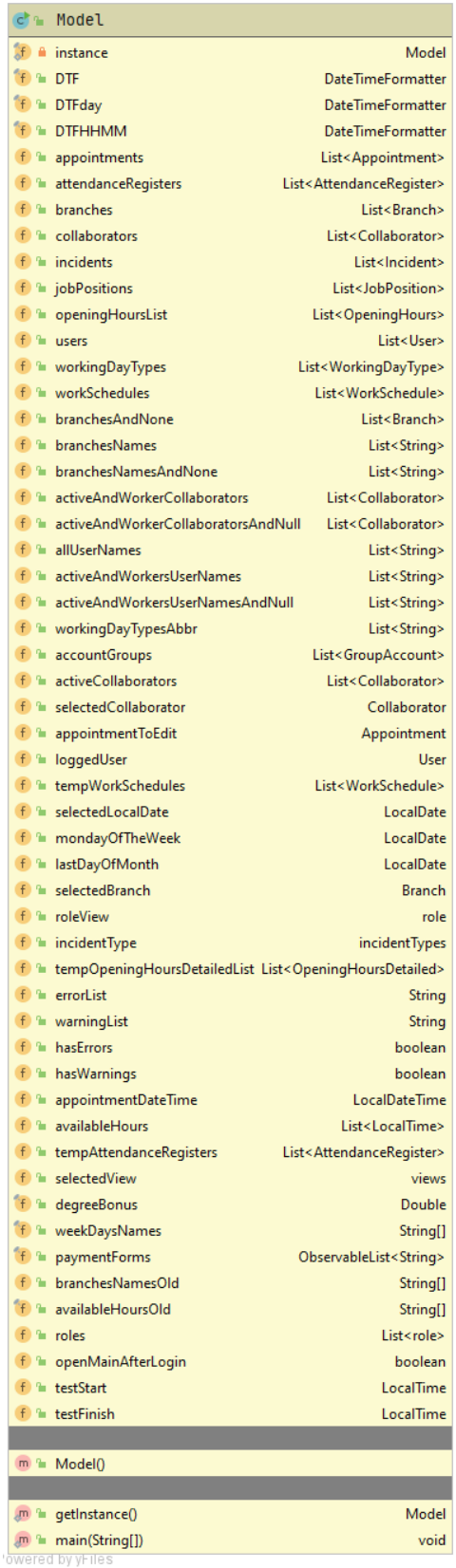


*Ilustración 22. Diagrama de clase. Fuente: Elaboración propia*

*Utilities*

Esta clase es un singleton, por lo que cuenta con una sola instancia. Como se puede observar contiene mayoritariamente métodos de apoyo que son utilizados por uno o más módulos de la aplicación, resulta ocioso detallar cada uno de estos, por lo que explicaremos los más importantes:

a) Lanzadores de alertas o de ventanas: Su función es, con base en determinados parámetros cambiar de vista o lanzar una alerta.
b) Ajustadores de hora y de fecha: Como la aplicación cuenta con interacción mayoritariamente mexicana, realiza ajustes para que la visualización y edición de la información sea compatible.
c) Buscador de valores en una lista: Su función es que, con base en algunos parámetros, devuelva un valor único o una lista de valores u objetos dentro de una lista más amplia.

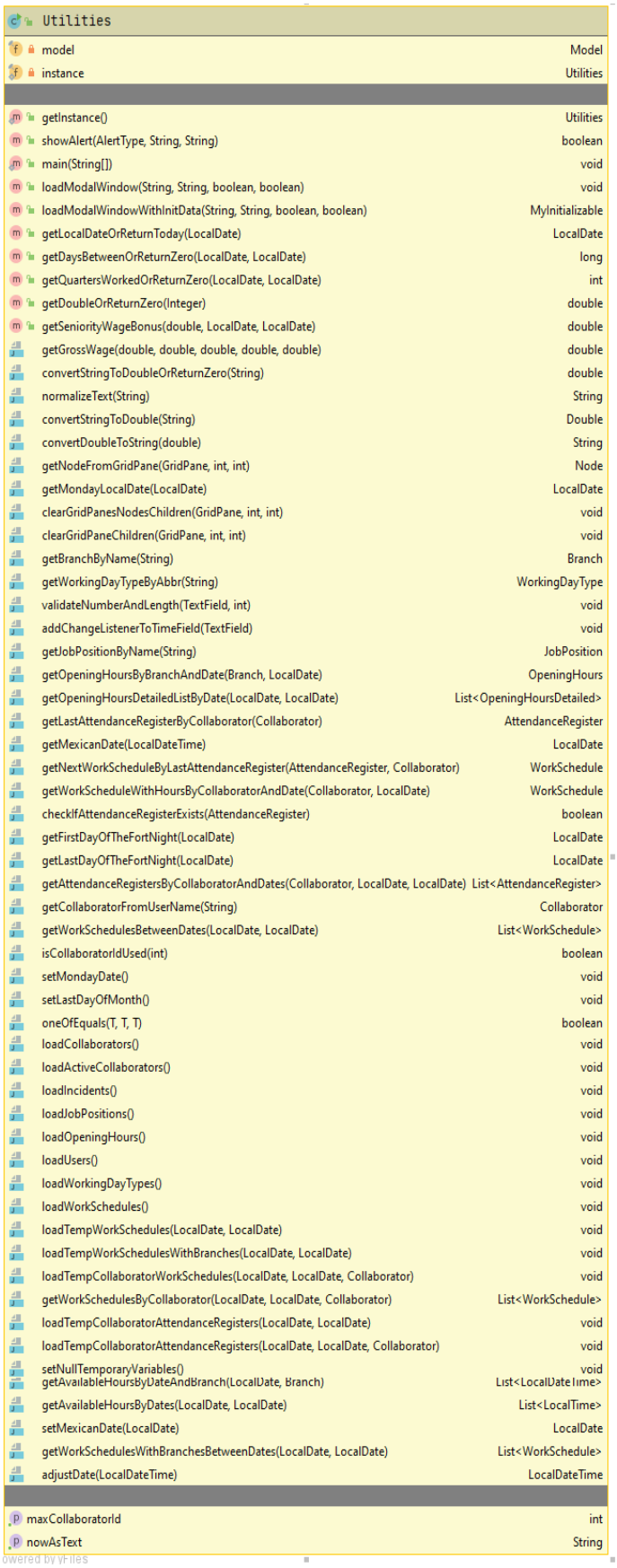


*Ilustración 23. Diagrama de clase. Fuente: Elaboración propia*

d) Cálculos específicos: Tiene por objeto, con base en determinados parámetros, realizar un cálculo interno, por ejemplo obtener el salario bruto del colaborador.
e) Detectar componentes: Su función es buscar, agregar o eliminar elementos de un componente, particularmente de *gridpanes*.
f) Accesorios de componentes: Su función es agregar funcionalidades a diferentes componentes, como por ejemplo eventos para detectar que los datos sean correctamente introducidos.
g) Cargar información de la base de datos: Son diversos métodos que sirven para cargar información compleja de la base de datos en una instancia de objeto para que puedan visualizarse o manipularse correctamente.

*Runnables*

Esta clase es un singleton, por lo que cuenta con una sola instancia. Como se puede observar contiene mayoritariamente métodos de apoyo que son utilizados para generar hilos que carguen en memoria información de la base de datos, para que puedan impulsarse de forma simultánea en los diferentes módulos.

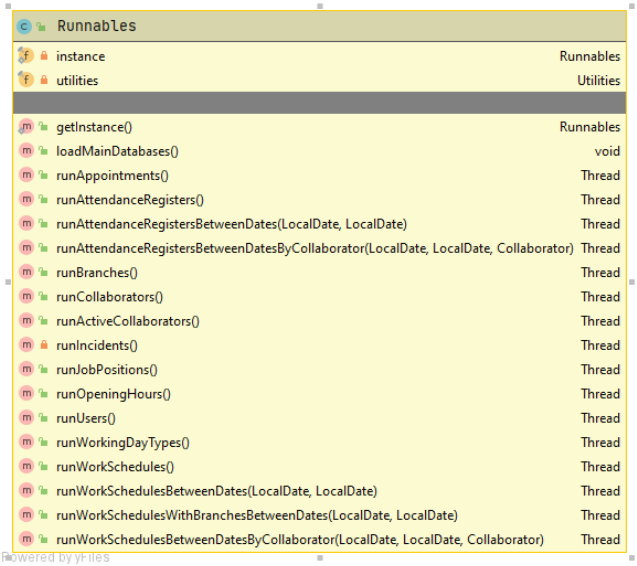


*Ilustración 24. Diagrama de clase. Fuente: Elaboración propia*

*HibernateConnection*

Esta clase cuenta con un único método que tiene por objeto realizar la configuración de Hibernate y hacer la conexión, lo cual se realiza antes de iniciar la venta principal, por lo que, en lo que se desarrolla, se muestra una ventana previa de espera.

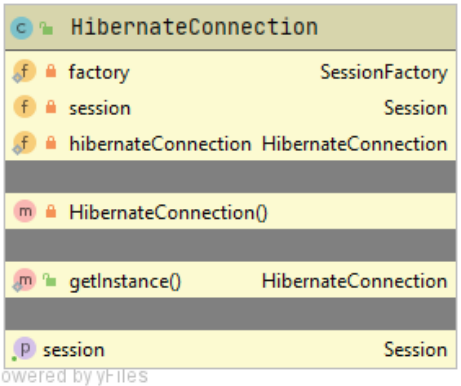


*Ilustración 25. Diagrama de clase. Fuente: Elaboración propia*

*HoursByDateByBranch*

Esta clase es exclusivamente de apoyo a una vista especial del módulo WorkSchedule, por medio de la cual se hacen distintos cálculos para determinar cuántas horas al día están cubiertas por el personal médico.

*OpeningHoursDetailed*

Esta clase es exclusivamente de apoyo a la vista principal del módulo WorkSchedule, particularmente sirve para generar la lista de horas de apertura por cada día de trabajo y poder validar si las jornadas de trabajo de los colaboradores se ajustan o no a estas.

*WorkScheduleError*

Esta clase es exclusivamente de apoyo a una vista especial del módulo WorkSchedule, por medio de la cual se enlistan los diferentes errores que pudieron ocurrir en la programación del calendario de trabajo.

#### 5.3.1.3 Controller

En este directorio se conservan los controladores de todas las vistas y se estructura en los diferentes módulos con los que cuenta la aplicación.

##### 5.3.1.3.1 Attendance Control Controller

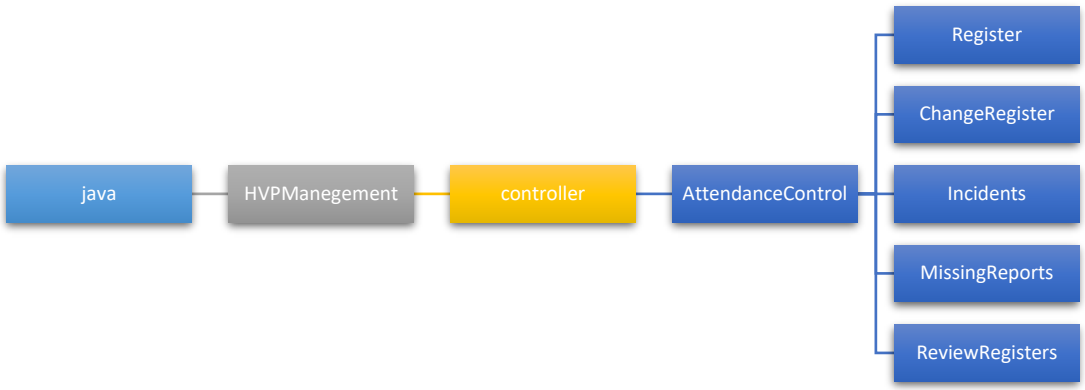


*Ilustración 26. Esquema de contenido. Fuente: Elaboración propia*

*Register*

Es el controlador de la vista principal del módulo de registro de entradas y salidas. Esta vista inicial permite al usuario registrar su entrada y salida, conocer cuál es el estado de esta y poder acceder a otra información como sus registros pasados. Sus campos se refieren a instancias de otras clases, a componentes y a objetos que se usan durante el funcionamiento del módulo. Los métodos principales son los siguientes:

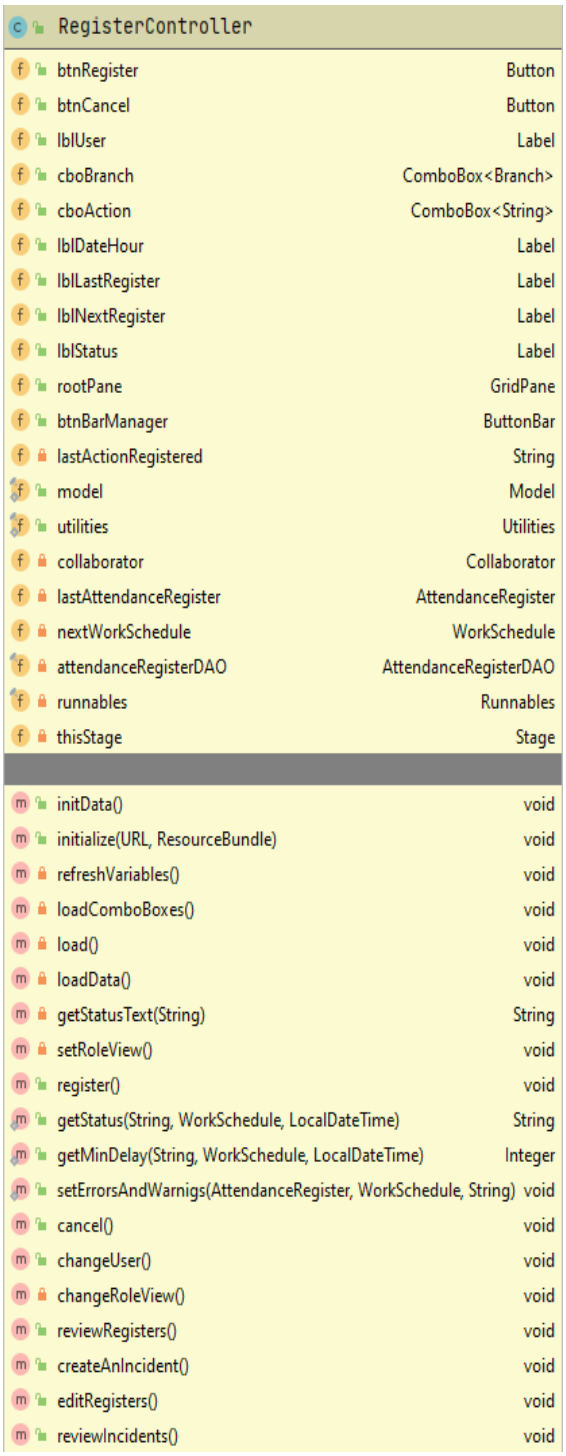


*Ilustración 27. Diagrama de clase. Fuente: Elaboración propia*

a) *Initialize:* con este se cargan los métodos iniciales para la vista: *refreshVariables, loadComboboxes, loadData, setView.*
b) *RefreshVariables:* se cargan en memoria los registros de la base de datos que sean útiles para la visualización y manipulación.
c) *Load:* sirve para hacer una llamada a refreshvariables cada vez que se haga un cambio de parámetro.
d) *LoadData:* carga en la interfaz los datos que correspondan de acuerdo con el usuario, la fecha y sus registros previos.
e) *GetStatusText:* es un método accesorio para la visualización, pero se requiere hacer varios cálculos complejos con respecto a la puntualidad del colaborador.

*f)* *Register:* este es realmente el método más importante, pues es el que se activa cuando el usuario oprime el botón registrar, con lo que se realizan los cálculos y validaciones necesarias para registrar su entrada o su salida en la base de datos.

g) Métodos de validación: existen varios métodos que tienen como finalidad determinar los errores y advertencias al usuario antes de realizar el registro.

h) Métodos de cambio de vista: se activan por medio de botons y permiten al usuario entrar a una función auxiliar de este módulo, por ejemplo la visualización de registros anteriores o la edición.

*ChangeRegister*

Es el controlador de la vista para cambiar los registros de entradas y salidas. Sus campos se refieren a instancias de otras clases, a componentes y a objetos que se usan durante el funcionamiento de la vista.

Sus métodos se refieren exclusivamente a la visualización de la interfaz y a aquellos útiles para hacer las ediciones correspondientes con sus validaciones relacionadas.

*ReviewRegisters*

Es una clase muy similar a la anterior, pero esta, a diferencia de aquella, no permite la edición de la información y solo puede visualizar los datos del usuario que haya iniciado sesión.

*Incidents*

Permite un rápido acceso al módulo de incidentes para que los usuarios puedan crear uno y sea atendido por la administración.

*MissingReports*

Es una clase accesoria de las clases *reviewRegisters* y *changeRegisters* cuya función principal es una vista de los registros faltantes sea del colaborador o de todos los colaboradores.

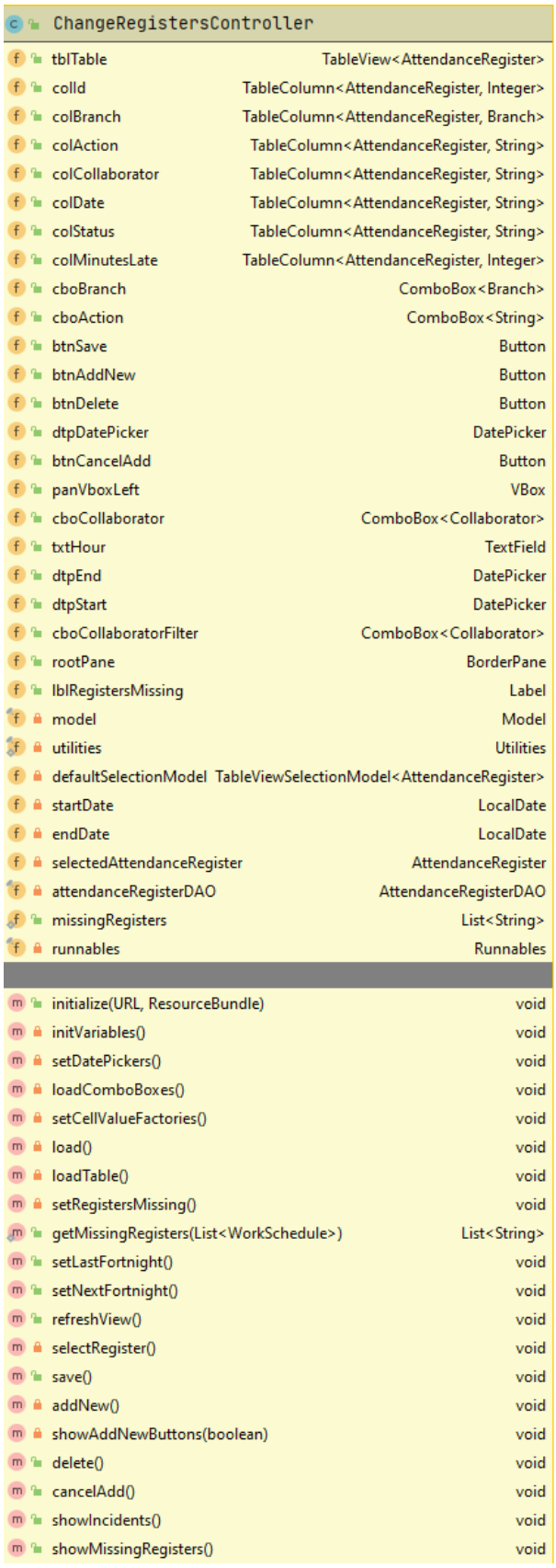


*Ilustración 28. Diagrama de clase. Fuente: Elaboración propia*

#### 5.3.1.3.2 Collaborator Controller


*Ilustración 29. Esquema de contenido. Fuente: Elaboración propia*

Este directorio solo cuenta con una clase en virtud de que todas las funciones: visualizar, agregar y editar no se manipulan en vistas separadas, sino que la visualización de la interfaz cambia dependiendo de la función deseada. En la imagen no se incluyen los campos, para no sobrecargar la información.

Los métodos principales se refieren a dar inicio a la visualización, cargar la información de los usuarios, cambiar de vista entre agregar nuevo, editar o visualizar, generar información automática, agregar una imagen de usuario y validar que en determinados campos solo se acepten números.

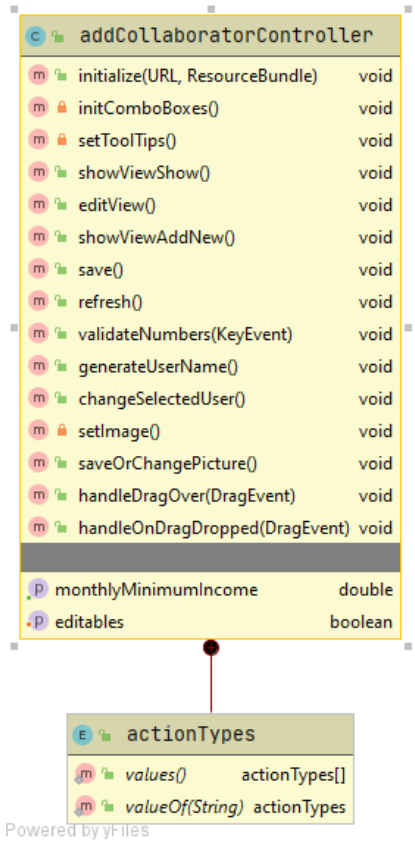


*Ilustración 30. Diagrama de clase. Fuente: Elaboración propia*

#### 5.3.1.3.3 Configuration Controller

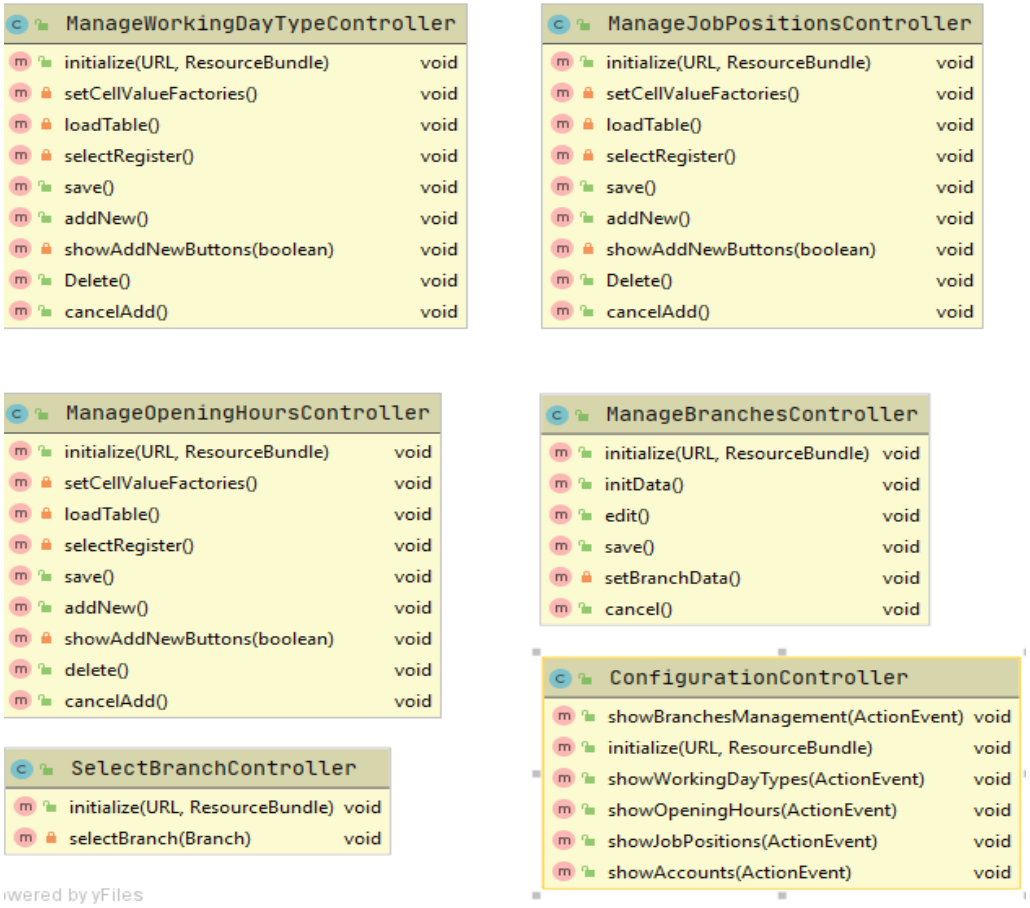


*Ilustración 31. Diagrama de clase. Fuente: Elaboración propia*

Este directorio cuenta con varias clases que todas son muy similares entre sí pues son controladores de vistas que tienen por objeto agregar, editar o eliminar información de algunas entidades secundarias de la base de datos: sucursales, puestos de empleos, horarios de apertura y tipos de jornada de trabajo.

Los métodos principales se refieren a dar inicio a la visualización, cargar la información de los usuarios, cambiar de vista entre agregar nuevo, editar o visualizar, generar información automática, agregar una imagen de usuario y validar que en determinados campos solo se acepten números.

#### 5.3.1.3.4 Incident Controller


*Ilustración 33. Esquema de contenido. Fuente: Elaboración propia*

*Incident*

Esta clase es el controlador de la vista principal del módulo incidentes, lo que hace es únicamente mostrar al usuario los incidentes que ha realizado y su estado; y de igual forma le permite introducir un nuevo incidente.

*ManageIncidents*

Esta clase es el controlador de la vista para resolver los incidentes, su función principal es dar a conocer al usuario los incidentes registrados así como los datos auxiliares; y con base en estos, le permite al administrador abrir los módulos que se relacionan con los parámetros del incidente y poder solucionarlos; y una vez realizado esto marcar el incidente como concluido.

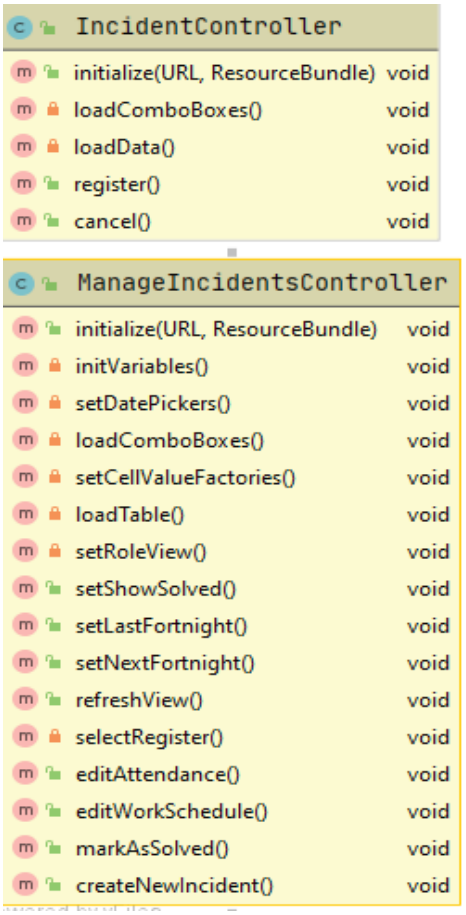


*Ilustración 32. Diagrama de clase. Fuente: Elaboración propia*

#### 5.3.1.3.5 Main Controller

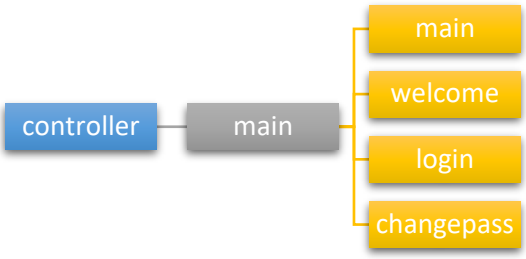


*Ilustración 35. Esquema de contenido. Fuente: Elaboración propia*

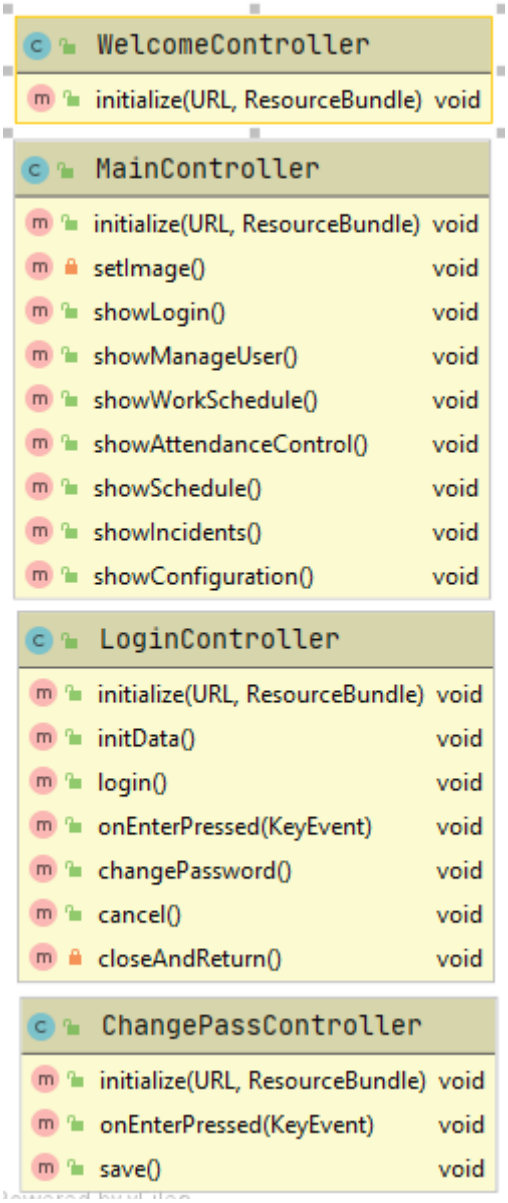


*Ilustración 34. Diagrama de clase. Fuente: Elaboración propia*

*Welcome*

Es el controlador de la primera ventana que visualiza el usuario, que sirve de espera en lo que se realiza la conexión con Hibernate

*Main*

Es el controlador de la ventana principal de la aplicación, permite al usuario acceder a cualquiera de los módulos, verificando previamente si requiere haber accedido previamente o no o si requiere algún rol especial. Por otro lado también informa el nombre y la foto del usuario que ha accedido.

*Login*

Es el controlador de la ventana de login, para que el usuario ponga su usuario y su contraseña, autentica que sea correcto y finalmente le otorga la oportunidad de cambiar su contraseña.

*ChangePass*

Es el controlador de la ventana que otorga al usuario la posibilidad de cambiar su contraseña.

#### 5.3.1.3.6 ScheduleController

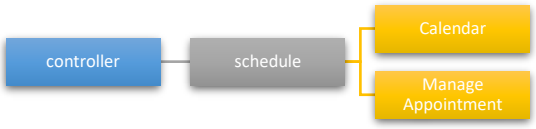


*Ilustración 37. Esquema de contenido. Fuente: Elaboración propia*

*Calendar*

Es el controlador de la ventana principal del módulo *schedule,* que tiene por objeto programar citas para servicios veterinarios. Entre sus métodos más relevantes encontramos:

a) Métodos relacionados con el inicio de la aplicación: tienen por objeto cargar los comboboxes, refrescar las variables de acuerdo con el día de hoy y cargar las citas de la semana, así como los valores que correspondan.
b) Modificadores de parámetros: son aquellos que sirven para modificar los parámetros de visualización, a saber: la fecha, los colaboradores o la sucursal.
c) Método para abrir la pestaña de edición o inserción de citas.

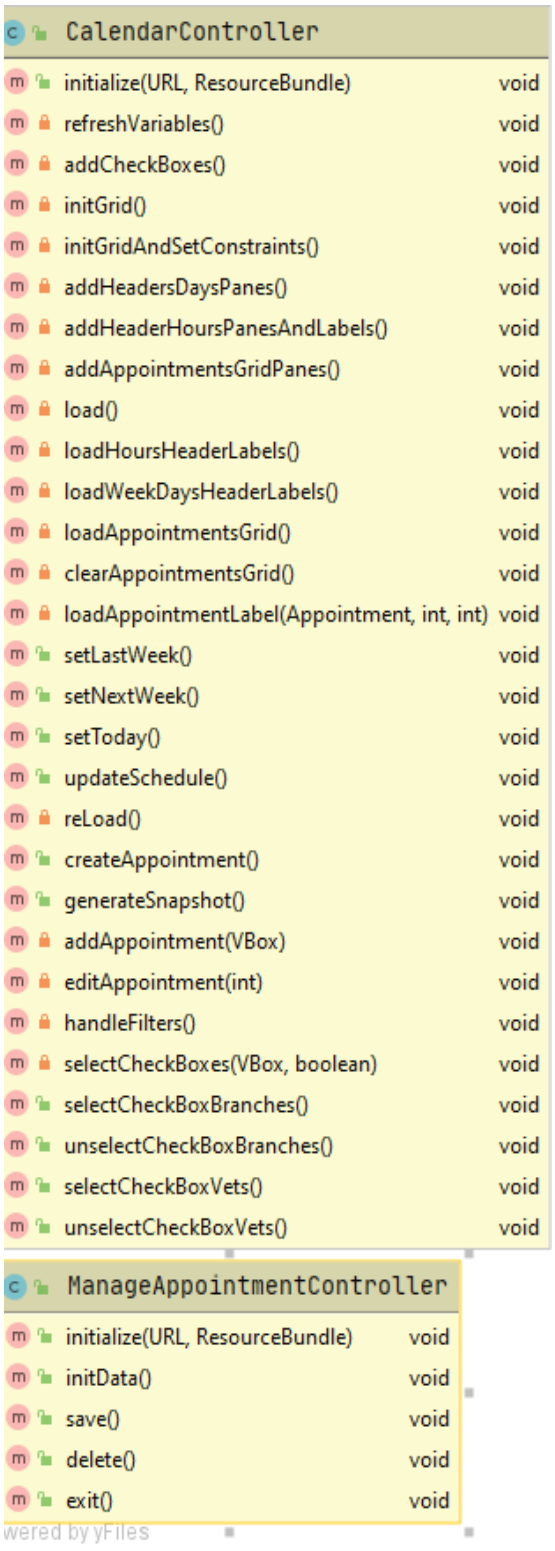


*Ilustración 36. Diagrama de clase. Fuente: Elaboración propia*

*ManageAppointment*

Esta es la clase del controlador de la vista de inserción o edición de citas. Entre sus métodos se encuentra la validación de datos y las funciones relacionados con los ajustes de las bases de datos.

### 5.3.1.3.7 WorkSchedule Controller

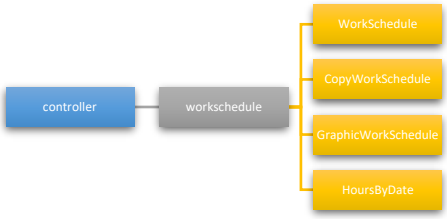


*Ilustración 39. Esquema de contenido. Fuente: Elaboración propia*

*Workschedule*

Esta es la clase controladora de la vista principal del módulo calendario de trabajo, este módulo consiste en otorgar herramientas al administrador para calendarizar los horarios del personal para facilitar la validación de los parámetros que le acompañan. Esta clase cuenta con veintiséis campos y cerca de cincuenta métodos. Entre los métodos más importantes encontramos:

a) Métodos iniciadores: son los métodos que sirven para cargar inicialmente la vista como los comboboxes, las variables iniciales, el tipo de vista de acuerdo con el rol del usuario y los datos existentes de acuerdo con la base de datos.
b) Cambiadores de vista y métodos accesorio: la interfaz cuenta con una vista que organiza los horarios de acuerdo con la sucursal y otro de acuerdo con los colaboradores, ambas se relacionan con la

| WorkScheduleController | |
|---|---|
| initData() | void |
| initialize(URL, ResourceBundle) | void |
| setRoleView() | void |
| loadComboBoxes() | void |
| initVariables() | void |
| refreshVariables() | void |
| loadView() | void |
| loadBranchView() | void |
| loadCalendarHeader(GridPane) | void |
| loadCalendarDaysHeader(GridPane, int) | void |
| addRowsToGrid(GridPane, int) | void |
| addGridPanesToGridPaneRest() | void |
| loadInternalGridsBranchView() | void |
| loadInternalDataBranchView() | void |
| insertRestLabels() | void |
| loadCollaboratorsView() | void |
| loadCollaboratorsNames(GridPane) | void |
| loadInternalGridsCollaboratorView() | void |
| loadInternalDataCollaboratorView() | void |
| changeDateOrView() | void |
| setLastWeek() | void |
| setNextWeek() | void |
| setToday() | void |
| loadDataBase() | void |
| refresh() | void |
| retrieveData() | void |
| setRestToCollaboratorsWithoutRegister() | void |
| retrieveDataFromPaneBranchView(GridPane) | void |
| retrieveDataFromCollaboratorPane() | void |
| generateRestDays() | void |
| validateData() | void |
| validateInternally() | void |
| validateReceptionistAtClose() | void |
| validateHourlyIfTherelsAVetOrAsistA() | void |
| validateUniqueUsers() | void |
| validateWithDataBase() | void |
| showHoursByDateAndBranch() | void |
| showGraphic() | void |
| saveIntoDB() | boolean |
| setWorkSchedulesToSaveOrUpdate() | boolean |
| generateSnapshot() | void |
| showCopyFromAnotherWeek() | void |
| emptyWeek() | void |
| showIncident() | void |
| addCollaboratorRow(ActionEvent) | void |
| removeCollaboratorRow(ActionEvent) | void |
| handleSelectUserBranchView(ChoiceBox<Collaborator>) | void |
| validateHBoxBranchView(HBox) | void |
| handleSelectWorkingDayTypeCollaboratorView(HBox) | void |
| handleSelectBranchCollaboratorView(HBox) | void |
| validateHBoxCollaboratorView(HBox) | void |
| getWorkScheduleFromWorkSchedules(WorkSchedule) | WorkSchedule |
| addOrUpdateTempWorkSchedules(WorkSchedule) | void |
| getGridPaneByBranchName(String, GridPane, GridPane, GridPane) | GridPane |

*Ilustración 38. Diagrama de clase. Fuente: Elaboración propia*

misma base de datos, pero se permite la modificación para facilitar la manipulación de la información.

c) Métodos para ajustar los parámetros: métodos para actualizar la interfaz dependiendo de la fecha.
d) Métodos para agregar o eliminar filas a una sucursal: son métodos que adecúan el número de filas de la tabla de horarios de una sucursal para poner o quitar colaboradores.
e) Métodos de validación: son diversos métodos que validan varias cosas: como que el horario se ajuste al horario de apertura, que no haya más de dos jornadas por trabajador por día, que no se exceda la jornada máxima semanal de trabajo, que siempre haya un médico, que el horario de salida no sea previo al horario de entrada, entre otras cosas.
f) Métodos relacionados con la base de datos: se refiere a la inserción, modificación o eliminación de registros en la base de datos.
g) Métodos cambiadores de ventana: se refiere a métodos para abrir ventanas auxiliares.

*CopyWorkSchedule*

Es el controlador de la vista para copiar un calendario, esto permite al usuario copiar el calendario de una semana a otra y determinar cuántas semanas quiere repetirlo. Para esto, cuenta con métodos de validación y cálculos internos para solicitar al usuario que confirme si quiere reemplazar los datos existentes.

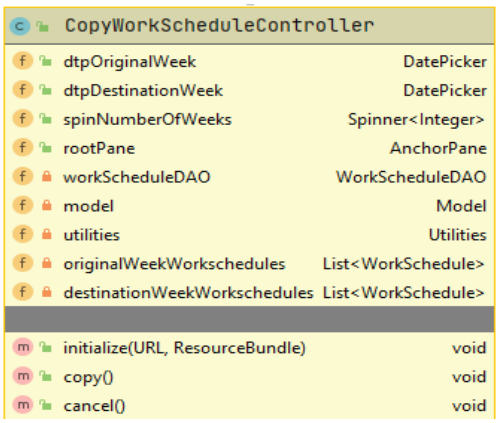


*Ilustración 40. Diagrama de clase. Fuente: Elaboración propia*

*Errors*

Es el controlador de la vista errores, que es una vista emergente auxiliar que le aparece al usuario cuando solicita la validación de la información o intenta hacer ediciones o modificaciones en la base de datos. En este sentido le muestra los posibles errores y advertencias que tiene el calendario con respecto a los parámetros de validación.

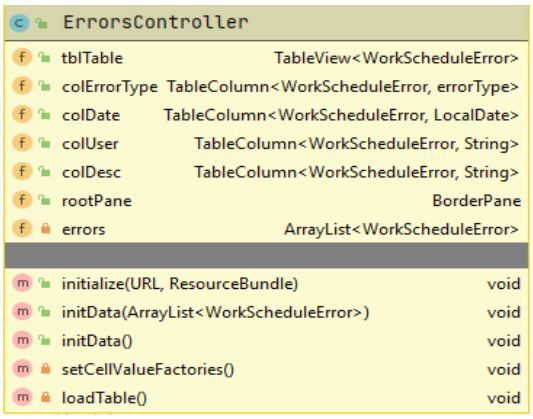


*Ilustración 41. Diagrama de clase. Fuente: Elaboración propia*

*GraphicWorkSchedule*

Esta vista no permite al usuario manipular información sobre la calendarización, pero le da una vista que le permite observar de una forma más gráfica y esquematizada quienes cubren cada una de las horas de apertura de cada sucursal. Los métodos principales se refieren a cargar la información llenada por el usuario en tres calendarios diferentes, y que se distingan por colores de acuerdo con el puesto de trabajo de cada persona, asi mismo genera tantas filas como horas tenga el día y columnas como colaboradores asistan a cada una de las sucursales.

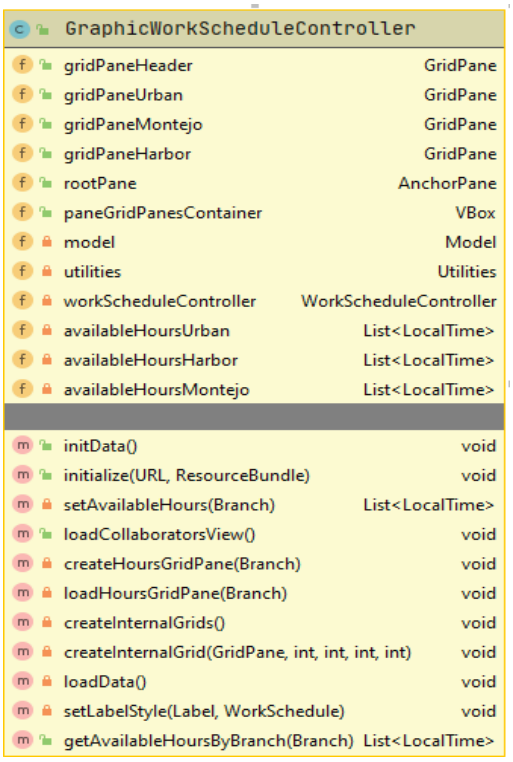


*Ilustración 42. Diagrama de clase. Fuente: Elaboración propia*

*HoursByDateAndBranch*

Esta es una vista auxiliar emergente que, de forma similar a la vista gráfica, hace un cálculo del número de horas que se cuenta por día y por sucursal para que el administrador verifique si hay algún día que esté sobrecargado o sin suficiente personal.

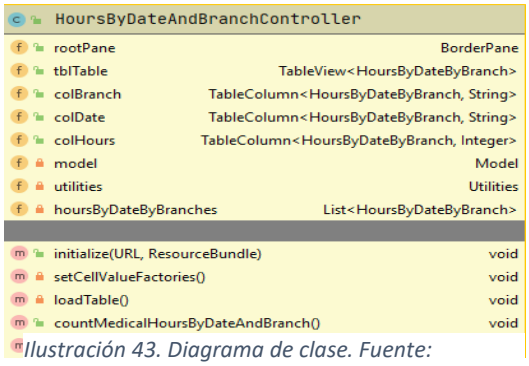


*Ilustración 43. Diagrama de clase. Fuente: Elaboración propia*

#### 5.3.1.4 Entity

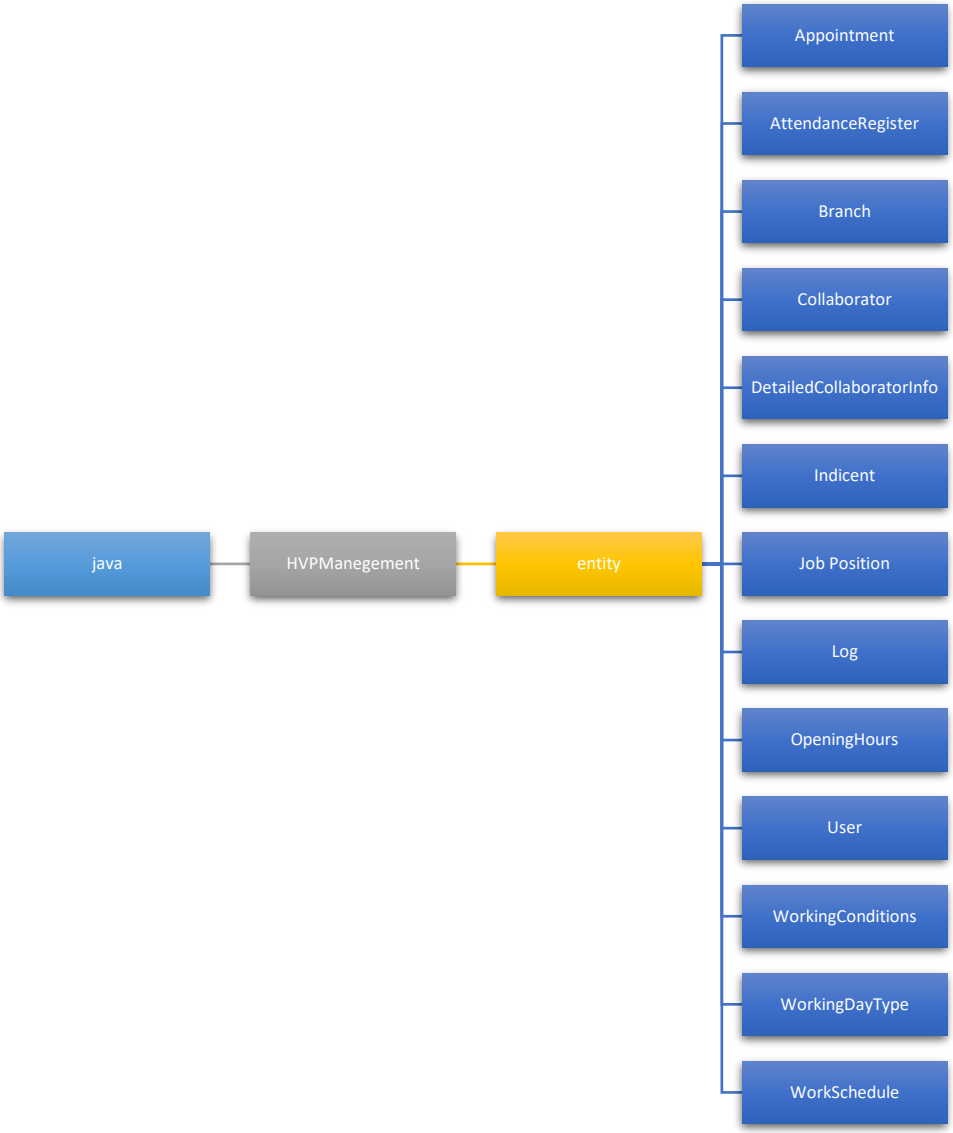


*Ilustración 44. Esquema de contenido. Fuente: Elaboración propia*

Las clases de tipo entidad coinciden naturalmente con entidades de las bases de datos, por lo que sus campos son los de los registros de las bases de datos y sus métodos son escasos, mayormente se refieren a las funciones hash, equals y a la manipulación de listas en los casos de relaciones de uno a muchos.

Con respecto al contenido de sus campos, se pueden observar los diagramas entidad-relación en el apartado 5.1.

#### 5.3.1.5 DAO

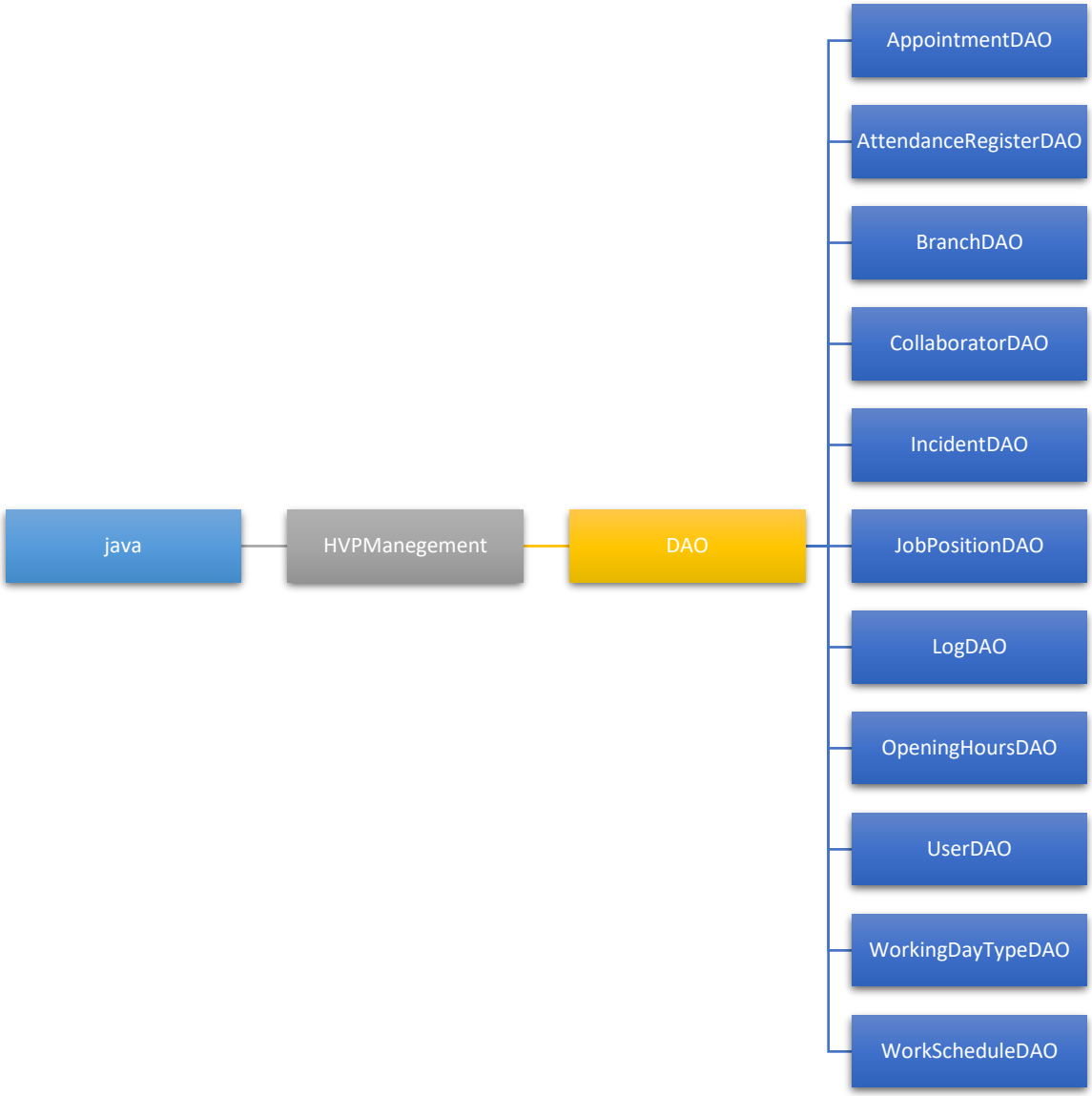


*Ilustración 45. Esquema de contenido. Fuente: Elaboración propia*

Las clases DAO (Data Access Object u objetos de acceso a datos), son clases que contienen únicamente métodos que realizan una conexión, por medio de Hibernate a la base de datos con el objetivo de hacer los métodos CRUD en distintas variantes.

### 5.3.2 Resources

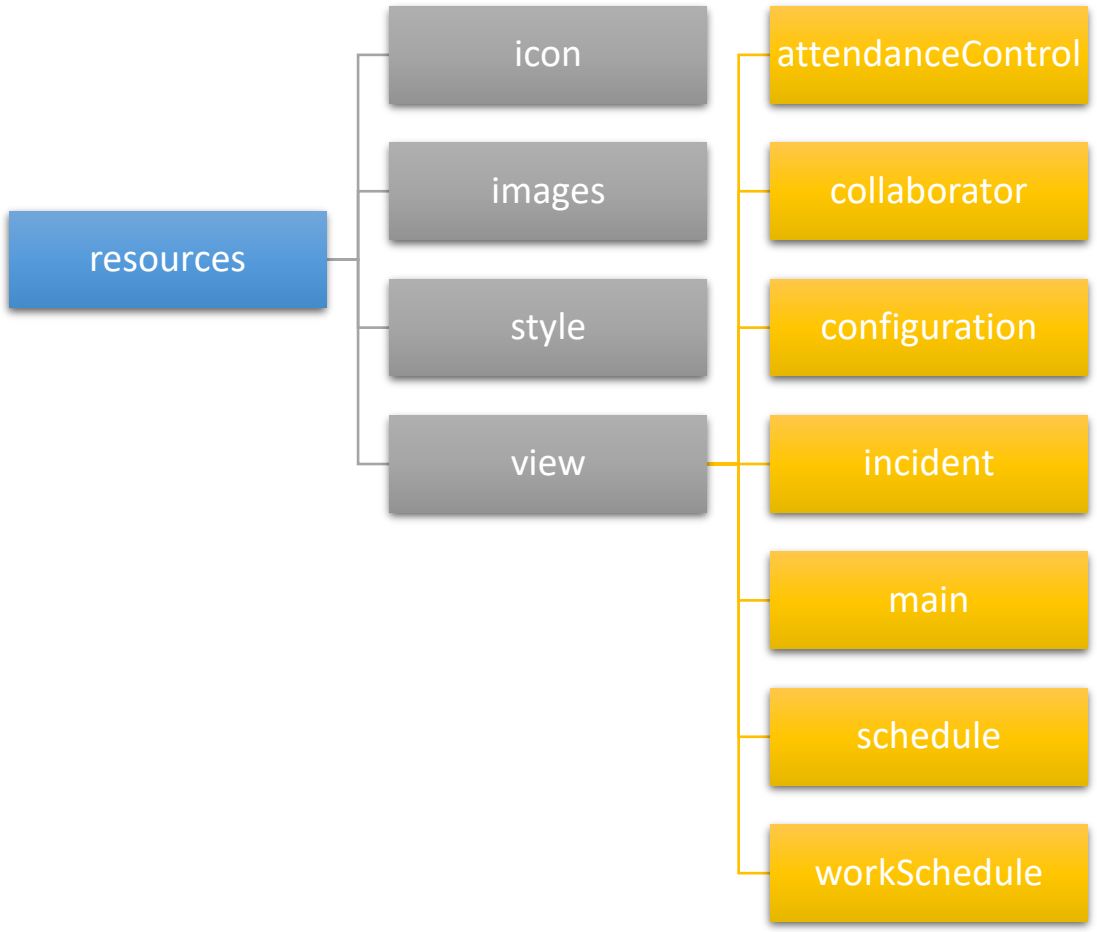


*Ilustración 46. Esquema de contenido. Fuente: Elaboración propia*

En este directorio se resguardan los recursos de la aplicación.

*Icon*

En este directorio se conservan los íconos que se usan durante la aplicación, como el ícono de la misma aplicación o cursores para avanzar al siguiente o al anterior, así como los relacionados con cada uno de los módulos.

*Images*

En este directorio ser conservan las imágenes

*Style*

En este directorio se conservan los archivos de estilo de tipo CSS, en realidad solo es uno con indicaciones para toda la aplicación.

*View*

En esta carpeta se guardan todas las vistas en formato fxml de la aplicación, no se detallarán pues coincide con lo explicado para los controladores.

# 6. Despliegue y pruebas

## 6.1. Plan de prueba

| NÚM. | ESPECIFICACIÓN DE PRUEBAS |
|---|---|
| 1 | **Objetivo probado:** Realizar la autenticación de los usuarios y permitir que puedan acceder a distintos módulos y vistas específicas dependiendo de si se han autenticado y de su rol.<br><br>**Requisitos probados:**<br><br>a) El usuario puede acceder a la aplicación con su usuario y contraseña.<br>b) El usuario puede cambiar su contraseña.<br>c) En caso de que el usuario se equivoque con su nombre de usuario o contraseña, deberá aparecer una ventana emergente señalando el error.<br>d) En caso de que sea exitoso, en la ventana principal aparecerán sus datos y fotografía.<br>e) El usuario deberá estar registrado para acceder a cualquier módulo con excepto de la agenda.<br>f) Solo los administradores podrán acceder al módulo configuración.<br><br>**Pruebas a realizar:**<br><br>a) Introducir los datos de nombre de usuario y contraseña correctamente.<br>b) Introducir los datos de nombre de usuario y contraseña incorrectamente.<br>c) En caso de que el usuario se equivoque con su nombre de usuario o contraseña, deberá aparecer una ventana emergente señalando el error.<br>d) En caso de que sea exitoso, en la ventana principal aparecerán sus datos y fotografía.<br>e) Intentar acceder sin registrar a los módulos que requieren inicio de sesión previo.<br>f) Intentar acceder al módulo configuraciones con un usuario que no tenga rol de administrador. |
| 2 | **Objetivo probado:** Registrar y editar la información de los colaboradores de la empresa, incluyendo además de sus datos personales, sus datos como usuario de la aplicación y sus condiciones de trabajo.<br><br>**Requisitos probados:**<br><br>a) En la interfaz se pueda elegir a todos los colaboradores activos para visualizar su información.<br>b) Editar la información de los colaboradores y se refleje en la base de datos.<br>c) Agregar nuevos usuarios. |

| | |
|---|---|
| | d) Los cálculos relacionados a si es activo o el bono de antigüedad se realicen automáticamente.<br><br>**Pruebas a realizar:**<br><br>a) Seleccionar cada uno de los usuarios y verificar que la información se cargue.<br>b) Agregar o editar información en la base de datos y verificar su representación en la interfaz y viceversa.<br>c) Cambiar aleatoriamente las fechas de entrada y salida para determinar que los cálculos se realicen |
| **3** | **Objetivo probado:** Permitir a la administración calendarizar los turnos de trabajo del personal y contar con mecanismos de validación útiles.<br><br>**Requisitos probados:**<br><br>a) Se visualicen correctamente los días, a pesar de que se utilice en diversas zonas horarias.<br>b) Se ajuste el número de filas por sucursal dependiendo del día que tenga más colaboradores en la semana.<br>c) La generación automática de los colaboradores que descansarán un determinado día de la semana.<br>d) La información de la vista colaborador y de la vista sucursal coincide.<br>e) La representación gráfica del calendario semanal se ajusta en columnas y horas a lo calendarizado.<br>f) Las diferentes validaciones se realizan correctamente.<br><br>**Pruebas a realizar:**<br><br>a) Visualizar y editar la información tanto en un ordenador de México como de España.<br>b) Agregar o quitar colaboradores para verificar si se acomoda la tabla de la sucursal.<br>c) Insertar un número de colaboradores aleatorio y dejar otros insertar, y revisar si en la base de datos se registra como día de descanso.<br>d) Cambiar entre la vista colaborador y sucursal.<br>e) Hacer ajustes entre colaboradores y sucursales y verificar su cambio en el apartado gráfico.<br>f) Meter datos erróneos sobre dobles jornadas el mismo día, horario de salida previo al de apertura, formatos incorrectos de hora, entre otros. |
| **4** | **Objetivo probado:** Registrar las entradas y salidas en las diversas sucursales de la empresa, así como llevar un control sobre sus retardos y bonos de puntualidad.<br><br>**Requisitos probados:** |

| | |
|---|---|
| | a) La ventana inicial del módulo muestra la información relacionada con el último registro relacionado y la próxima entrada o salida.<br>b) El cálculo de los minutos de retardo es correcto.<br>c) La edición de registros funciona correctamente.<br>d) Las validaciones son eficaces.<br><br>**Pruebas a realizar:**<br><br>a) Registrar varias entradas y salidas en la base de datos y acceder en la aplicación para determinar que por cada usuario los datos de último registro y próxima entrada o salida son correctos.<br>b) Intentar ingresar entradas o salidas con diferentes horas de diferencia a la que corresponde.<br>c) Editar registros cambiando cada uno de los parámetros.<br>d) Introducir datos erróneos como entrada en un día que no está calendarizado, una salida sin que exista entrada o una entrada en una sucursal diferente a la que corresponde. |
| **5** | **Objetivo probado:** Agendar citas de los servicios que presta la empresa en cada una de sus sucursales.<br><br>**Requisitos probados:**<br><br>a) La agregación de dos citas en la misma hora y día es posible.<br>b) Las citas se identifican entre sucursales por diferentes colores.<br>c) Las opciones de filtrado de información funcionan perfectamente.<br><br>**Pruebas a realizar:**<br><br>a) Insertar dos citas a la misma hora.<br>b) Insertar citas de diferentes sucursales.<br>c) Filtrar información por sucursal y por colaborador. |
| **6** | **Objetivo probado:** Ingresar y gestionar incidentes relacionados con la calendarización de jornadas de trabajo y registro de horarios, así como cualquier situación relacionada con la aplicación.<br><br>**Requisitos probados:**<br><br>a) La interfaz solo muestra los incidentes del usuario que ha iniciado sesión.<br>b) El administrador puede seleccionar una incidencia y pasar directamente a la ventana de edición de entradas y salidas o de calendario de jornadas de trabajo para realizarlas.<br><br>**Pruebas a realizar:** |

| | |
|---|---|
| | a) Ingresar distintas incidencias en la base de datos y verificar la visualización según el usuario.<br>b) Proceder desde el módulo de incidencias a los otros módulos para verificar que los parámetros de edición se incorporen correctamente. |
| **7** | **Objetivo probado:** Agregar y modificar aspectos relacionados con la empresa, accesorios para el resto de los módulos como son las sucursales, los horarios de apertura, los puestos de trabajo y los tipos de jornadas de trabajo.<br><br>**Requisitos probados:**<br><br>a) La inserción y edición de las entidades que se mencionan en el objetivo se realizan eficazmente.<br><br>**Pruebas a realizar:**<br><br>a) Insertar y editar cada una de las entidades referidas. |

# 7. Conclusiones

## 7.1. Objetivos alcanzados

Los objetivos planteados se cumplieron en su totalidad.

| Tipo de objetivo | Objetivo | Estado |
|---|---|---|
| **General** | Diseñar e implementar una aplicación que apoye a la administración y a los colaboradores en la gestión del Hospital Veterinario Peninsular, en las actividades de registro de nuevos colaboradores, la programación de calendarios de trabajos, el registro de entradas y salidas y la agenda de citas. | Cumplido |
| **Específico** | Realizar la autenticación de los usuarios y permitir que puedan acceder a distintos módulos y vistas específicas dependiendo de si se han autenticado y de su rol. | Cumplido |
| **Específico** | Registrar y editar la información de los colaboradores de la empresa, incluyendo además de sus datos personales, sus datos como usuario de la aplicación y sus condiciones de trabajo. | Cumplido |
| **Específico** | Permitir a la administración calendarizar los turnos de trabajo del personal y contar con mecanismos de validación útiles. | Cumplido |
| **Específico** | Registrar las entradas y salidas en las diversas sucursales de la empresa, así como llevar un control sobre sus retardos y bonos de puntualidad. | Cumplido |
| **Específico** | Agendar citas de los servicios que presta la empresa en cada una de sus sucursales. | Cumplido |
| **Específico** | Ingresar y gestionar incidentes relacionados con la calendarización de jornadas de trabajo y registro de horarios, así como cualquier situación relacionada con la aplicación. | Cumplido |
| **Específico** | Agregar y modificar aspectos relacionados con la empresa, accesorios para el resto de los módulos como son las sucursales, los horarios de apertura, los puestos de trabajo y los tipos de jornadas de trabajo. | Cumplido |

## 7.2. Conclusiones del trabajo

El proyecto implicó un gran reto en todas sus fases de desarrollo, pues si bien se contaban con conocimientos previos sólidos, la inmensa cantidad de clases y librerías disponibles, así como el replanteamiento constante de otras maneras de obtener el mismo resultado, hicieron que este desarrollo apoyara enormemente al desarrollo de habilidades y conocimientos del equipo.

La cercanía del equipo desarrollador con la empresa en la que se pretende implementar la aplicación, así como la familiarización de aquel con el software previo fueron muy útiles y productivos durante todo el desarrollo.

Podríamos calificar al proyecto como ambicioso, pues en una etapa preliminar se planteó como idea incorporar un mayor número de módulos y funcionalidades, pero se llegó a la conclusión que era mejor desarrollar un módulo a la vez e ir determinando una entrega realista.

Un parámetro fundamental durante todo el desarrollo fueron los tiempos de entrega, en este sentido, un objetivo que se tuvo siempre claro fue hacer las entregas correspondientes en las fechas requeridas.

La implementación de JavaFX y de Hibernate jugaron un papel muy importante pues se encontraron varios problemas de compilación y ejecución que tuvieron que ser revisados en diversas fases, por ejemplo el mapeo de las entidades para Hibernate en los objetos de acceso a datos tuvieron que reconstruirse numerosas veces, sea como consultas SQL o consultas de la librería persistence, pues el rendimiento y los problemas conocidas cono N+1 hacían que la aplicación perdiera calidad.

### 7.3. Vías futuras

Las vías futuras de desarrollo del proyecto son numerosísimas, por lo que enlistaremos las más importantes:

a) Diseñar e implementar entidades relacionadas con contabilidad, tanto de caja como general.
b) Incorporar un módulo de cuentas de caja, que permita a los recepcionistas registrar las entradas y salidas de dinero.
c) Registrar los servicios que presta la empresa, así como sus características especificas en relación con los colaboradores.
d) Calcular las comisiones que corresponden a los colaboradores por los servicios que prestan, pues estas obedecen a diversos factores como el puesto ocupado y su antigüedad.
e) Calcular la nómina quincenal de los colaboradores.

f) Hacer inventario de los productos y un sistema de traspaso de inventarios entre sucursales y entre cada sucursal con su bodega y el mostrador, así como generar avisos en caso de faltantes o escasez.
g) Registrar clientes y sus mascotas.
h) Generar expedientes médicos.

## 8. Glosario

Para efectos de este proyecto, se entenderá por:

a) **Advertencia:** anotación que ocurre cuando el usuario pretende realizar una acción que podría no corresponder con lo deseado, por lo que se suele pedir al usuario su confirmación. *Véase error.*
b) **Calendario:** el calendario semanal.
c) **Calendario semanal:** el calendario semanal que comprende las jornadas de trabajo de cada colaborador, incluyendo el tipo de jornada de trabajo, la sucursal, la hora de entrada y la hora de salida.
d) **Código de colaborador:** Si bien existe un código alfabético y un código numérico de cada colaborador, en caso de no distinguirse se entenderá que se refiere al código alfabético.
e) **Error:** anotación que ocurre cuando el usuario pretende realizar una acción que no puede registrarse por ser incompatible con los parámetros, en este caso no se permite la acción. *Véase advertencia.*
f) **Incidente:** hecho que puede registrar cualquier usuario para manifestar la existencia de un error o de una inconsistencia, independientemente de la causa de esta.
g) **Sucursal:**
h) **Sucursal exprés:** aquella sucursal de menor tamaño, donde se prestan servicios menores y que es atendida generalmente por un único colaborador.
i) **Ventana inicial:** la ventana cargada cuando se abre la aplicación que tiene por objeto anunciar al usuario que esta se está cargando, en lo que se realizan tareas adicionales, para evitar que el usuario piense que hay un error por la demora en abrir la aplicación.
j) **Ventana principal:** la ventana que permite el acceso a los diferentes módulos de la aplicación.

## 9. Bibliografía

# 10. Anexos

## 10.1 Manual de instalación

## 10.2 Manual de usuario

https://www.youtube.com/watch?v=hbOJdMtLvus&list=PLDKYjVmIS0jg69A3Zcff0DvqF-sOHZTDg

## 10.3 JavaDoc

# 11. Pendientes

Bibliografía

Hacer Javadoc

Hacer un manual de instalación

Hacer un manual de usuario: videos