Proyecto:	wShifts - Gestión de Turnos de Trabajo
Especialidad:	Admin. de web y de comercio electrónico
Autor:	Ángel Luis García García
Consultor:	Francisco Javier Noguera Otero
Tutor externo:	María del Camino Arias Villanueva
Fecha:	27/06/2017

Licencia

wShifts - Gestión de Turnos de Trabajo

Resumen del proyecto

La gestión de personal en sistemas empresariales con un número elevado de trabajadores necesita estructurarse adecuadamente para poder dar un servicio óptimo a todas las necesidades que se generan de la propia naturaleza del modelo de negocio. Cualquier sistema de gestión de personal de esta índole necesita estructurarse en unidades organizativas, que son por ejemplo cada uno de los centros, departamentos, servicios, secciones o vicerrectorados que componen una Universidad.

Una unidad organizativa está compuesta de personas que se pueden categorizar dependiendo de los servicios que presten y de su función dentro de dichos servicios. La realización de una función se denomina trabajo, la persona es el trabajador y el trabajo realizado en una jornada laboral, entre un rango horario continuo, se identifica como *Turno*. Por tanto la gestión de los turnos de trabajadores en unidades organizativas se denomina *Gestión de Turnos* y tiene como finalidad fundamental el control y gestión de presencias de trabajadores en sus respectivas unidades organizativas.

Normalmente esta gestión de presencias de trabajadores se realizan mediante el uso de hojas de cálculo. Sin embargo cuando la estructura organizativa es compleja por el elevado número de trabajadores y servicios su utilización se vuelve impracticable. Se hace necesario un sistema integral de gestión de turnos de personal para el control y gestión de los recursos más preciados de una unidad empresarial: los trabajadores.

wShifts (*work shifts)* es un proyecto que nace de esta necesidad, cuyo objetivo es la creación de un sistema básico de gestión de turnos de personal de una unidad empresarial estándar para el control y gestión de presencias de trabajadores a partir del uso de planificaciones diarias, también denominados cuadrantes o planillas. El proyecto se basa en los conceptos teóricos de turno o actividad (horario continuo de trabajo), ciclo o patrón (conjunto de actividades agrupadas por semanas), calendarios laborales, puestos de trabajo y planificaciones (expansión de ciclos en calendarios laborales).

wShifts es altamente configurable, basando su diseño en una estructura jerárquica de unidades organizativas. Así se podrán configurar diferentes centros físicos. Cada centro físico podrá tener uno o varios servicios funcionales. Por cada servicio funcional se podrán tener varios equipos de trabajo, que se definen por su propia naturaleza de funcionamiento, más específicamente por la categoría profesional de los trabajadores que conforman dichos equipos. Sin embargo en los equipos de trabajo no se planifican personas sino puestos de trabajo y son los trabajadores los que rotan sobre dichos puestos; esto es, los equipos de trabajo tienen puestos y los trabajadores se asignan a dichos puestos.

Se permite configurar todos los turnos que sean necesarios con las configuraciones horarias que se requieran. La aplicación genera la planificación de los ciclos en el calendario laboral teniendo en cuenta los días festivos, que también son configurables por año y centro físico. Finalmente, una vez configurados todos estos elementos se dispone de una planilla, que será el elemento donde se podrán visualizar los turnos de los trabajadores asignados por días en sus respectivos puestos de trabajo. Una planilla es por tanto un mes de un calendario laboral donde cada columna es un día y cada fila un trabajador de la unidad organizativa. En la planilla se pueden realizar cambios de turno e inserción de ausencias, que es la no realización de un trabajo planificado. Estas dos operaciones afectan directamente al balance horario del trabajador que se define como el total de horas trabajadas anualmente menos la jornada teórica, que es el número de horas anuales que tiene que realizar.

El diseño del proyecto se basa en arquitectura web cliente – servidor. La parte del servidor se ha desarrollado por completo con el lenguaje de programación Python y el marco de trabajo Flask, basado también en Python, para la gestión de los servicios web. La parte cliente se ha desarrollado íntegramente con el marco de trabajo Angular 2, programado con TypeScript. El sistema puede funcionar en modalidad multiusuario, con varios usuarios trabajando a la vez, en donde se ha optado por PostgreSQL como sistema gestor de bases de datos o monousuario, para el uso de un único usuario y en cuyo caso se optado por SQLite como sistema de bases de datos. En este proyecto se presentará el sistema monousuario. **wShifts** tiene licencia Apache 2.0.

Tabla de contenidos

Sumario

Capítulo I – Introducción

Objetivos

En el presente Trabajo Fin de Máster se va a crear un sistema básico de gestión de turnos de personal llamado **wShifts** (work Shifts) enfocado para su uso en unidades empresariales estándar con los objetivos principales de:

- Gestión de actividades de trabajadores.
- Gestión de ausencias de trabajadores.
- Control de balances horarios.
- Control de coberturas de servicio.

Este sistema se ha desarrollado para trabajar en un entorno web, el cual podrá ser accedido desde cualquier navegador, con independencia del sistema operativo del usuario. Se ha diseñado para ser escalable y adaptable en funcionalidades acordes a los requisitos de unidades empresariales, de modo que se podrán adecuar características específicas, incrementar prestaciones e integrarse en nuevos sistemas.

La organización empresarial se podrá moldear atendiendo a una estructura jerárquica de unidades organizativas: centros físicos, servicios funcionales y equipos de trabajo. Así un centro físico estará compuesto de varios servicios funcionales y un servicio de varios equipos de trabajo. Finalmente dentro de los equipos se podrán configurar los puestos de trabajo y estos serán los que se planifiquen en un calendario laboral para ser ocupados por trabajadores que estén dados de alta en el sistema.

Cada trabajador tendrá un balance horario, que será calculado en base a la relación entre el total de horas trabajadas a lo largo del año y una jornada teórica que debe realizar. Es misión del aplicativo mantener y actualizar el control del balance horario dependiendo de las actividades o turnos que realice el trabajador en su jornada laboral, incluyendo si se ausenta del trabajo o cambia el turno laboral. Por ende el gestor de turnos tiene que ser lo suficientemente adaptable para abarcar las casuísticas básicas de cómputo de balance horario, por lo que será necesaria la configuración de:

- Turnos: horario diario de jornada.
- Ciclos: conjunto de turnos de trabajo semanales.
- Planificaciones: expansión de ciclos en calendarios laborales.

Y justamente la planificación junto con los puestos de trabajo generan la planilla que es el elemento sobre el cual el usuario trabajará bien modificando el turno, bien insertando una ausencia, con la consecuente variación de balance horario del trabajador. Por tanto conviene decir que la planilla será el lugar de trabajo donde se gestionen los turnos y ausencias, se controlen los

balances horarios y las coberturas de servicio, que son las necesidades (número de trabajadores) que tiene un equipo dentro de un servicio para su correcto funcionamiento. Esta forma de trabajar no es original, es una evolución de otra herramienta más primitiva, que se mostrará en el siguiente capítulo.

Estado del Arte

Originalmente la gestión de turnos de trabajo se ha llevado a cabo mediante el uso de planillas en hojas de cálculo.

Cuadrante de turnos. MAYO

			lun	mar	mié	jue	vie	sáb	do	lun	mar	mié	jue	vie	sáb	do	lun	mar	mié	jue	vie	sáb	do	lun	mar	mié	jue	vie	sáb	do	lun	mar
#	Personal	Nombre	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	AD01		M	M	M	T	T	F	F	M	M	M	T	T	F	F	M	M	M	T	T	F	F	M	M	M	T	T	F	F	T	T
2	P01		T	T	TN	TN	TN	F	F	F	F	T	T	TN	TN	TN	TN	T	T	T	T	F	F	F	F	M	M	M	M	M	M	TN
3	P02		T	T	T	T	F	F	F	F	F	M	M	M	M	M	M	M	M	M	T	T	T	T	TN	TN	TN	TN	M	M	M	F
4	P03		M	M	M	F	F	F	F	M	M	M	F	F	T	T	T	T	T	F	F	F	F	TN	TN	TN	TN	TN	TN	F	F	F
5	P04		T	T	T	T	T	T	TN	TN	TN	F	F	M	M	F	M	M	M	M	M	M	T	T	F	F	F	F	M	M	M	T
6	P05		M	M	M	TN	TN	TN	TN	TN	TN	F	F	M	M	F	F	T	T	T	T	TN	TN	TN	TN	TN	TN	F	F	F	T	T
7	P06		F	F	F	F	F	F	M	M	M	M	M	M	T	T	T	M	M	M	TN	TN	TN	F	F	T	T	T	T	M	M	M
8	P07		M	M	M	M	F	F	T	T	T	TN	TN	TN	M	M	F	F	F	F	T	T	T	T	T	T	TN	TN	TN	M	M	M
9	P08		M	M	M	T	T	T	T	T	TN	TN	TN	F	F	F	T	T	T	T	T	F	F	TN	TN	TN	M	M	F	F	M	M
10	P09		F	F	F	F	M	M	M	M	M	M	F	F	M	M	M	T	T	T	T	T	T	F	F	F	F	TN	TN	M	M	M
11	P10		T	T	T	T	TN	TN	TN	F	F	F	F	TN	TN	TN	M	M	F	F	M	M	M	TN	TN	TN	M	M	M	F	F
12	P11		TN	TN	TN	F	F	TN	TN	TN	T	T	T	T	F	F	F	F	T	T	T	T	TN	TN	TN	F	F	F	F	M	M	M
13	P12		M	M	M	M	M	M	T	T	T	T	TN	TN	TN	T	T	T	T	TN	TN	TN	F	F	F	F	F	F	T	T	T	T
14	P13		TN	TN	TN	TN	F	F	F	F	M	M	M	M	M	M	M	M	T	T	T	T	T	TN	TN	TN	T	T	T	M	M	M
15	P14		T	T	T	T	F	F	F	F	M	M	M	T	T	TN	TN	TN	M	M	T	T	T	T	T	TN	TN	TN	F	F	F	M
16	P15		M	M	M	TN	TN	TN	TN	TN	TN	T	T	M	M	M	F	F	F	F	F	F	F	M	M	M	M	M	M	TN	F	F

Figura 1. Planilla en hoja de cálculo.

Una **planilla** es una rejilla que representa un mes de un calendario laboral, donde cada columna es un día del mes y cada fila un trabajador de la unidad organizativa. En las celdas producto de las intersecciones de filas y columnas se incluyen las abreviaturas correspondientes a los turnos que los trabajadores realizan y en algunos casos las abreviaturas de las ausencias que dichos trabajadores pudieran tener.

El uso de hojas de cálculo para la gestión de turnos de personal tiene ciertas ventajas:

- Es válido para organizaciones pequeñas.
- De fácil uso en cambios e inserción de turnos.
- Uso de fórmulas configurables para obtener información sobre turnos realizados (horas, presencias, etc).
- Presentación óptima de la relación trabajadores – turnos de trabajo que realizan sobre un calendario laboral, a partir de una rejilla.

Sin embargo su uso adolece de otros puntos:

- Es propenso a errores si se modifican las fórmulas de la plantilla.
- En algunos sistemas empresariales con recursos informáticos limitados y compartidos donde no hay control de acceso, se puede modificar la planilla.

- La inserción de planificación de ciclos en calendario de forma manual para organizaciones medianas y grandes es impracticable, ya que hay una planilla por equipo/servicio.
- Dependiente de software de terceros: Microsoft Excel, OpenOffice Calc, LibreOffice Calc, etc.
- Extracción complicada de metadatos: absentismo, etc.

Debido a estas desventajas han surgido soluciones software de todo tipo, pero que en definitiva tienen ciertos puntos en común:

- Usuario y roles para acceso a la aplicación.
- Visualización en rejilla de trabajadores – turnos en calendario laboral.
- Control automático de balances horarios.
- Gestión de ausencias.
- Estructuración de trabajadores en unidades organizativas.

En el mercado se puede encontrar software con versiones gratuitas limitadas en funcionalidad, aplicaciones completas de pago y en todos los casos con código fuente no accesible. Ejemplos de ello son TempusBasic, aTurnos, Gesturn, Turnos Saint 7, etc. En otros casos la aplicación puede llegar a ser gratuita pero sin acceso al código, como ABC Roster o Wikimetal RRHH.

A día de hoy no existe una solución software que contemple los puntos anteriormente señalados y que además sea licencia Open Source o de Software Libre. El proyecto **wShifts** pretende dar solución completa y extensible a este problema. Sin embargo para llegar a comprender totalmente como se gestionan los turnos de personal hay que analizar los elementos y procesos básicos que lo componen. En el siguiente capítulo se presentarán los conceptos fundamentales de las gestión de turnos de personal.

Conceptos fundamentales

wShifts (work Shifts ó Turnos de Trabajo) es un proyecto con licencia **OpenSource Apache 2.0** (que se adjunta en el **Anexo I**) . Se basa en la idea de gestión de turnos de personal mediante planillas. Para poder realizar esta gestión es necesario definir previamente ciertos elementos o componentes, que serán fundamentales para el uso de la planilla.

Figura 2. Componentes fundamentales del Gestor de Turnos.

Estructura organizativa

Las unidades empresariales se organizan a partir de una estructura arbórea de dependencia. Una organización básica es la planteada por la tripleta centro físico, servicio y equipo. Así un **centro físico** es un emplazamiento real, que se puede organizar funcionalmente mediante servicios. Por ejemplo en un hospital (centro físico) existen los servicios funcionales de Urgencias, Pediatría, Oncología, etc. Sin embargo un **servicio funcional** esta formado por conjuntos de trabajadores que tienen en común su categoría profesional. Siguiendo con el ejemplo del hospital, el servicio de Urgencias tiene el equipo de celadores, el equipo de facultativos, el de enfermería, etc. En definitiva un **equipo de trabajo** se define por la categoría profesional de los trabajadores que componen dicho equipo. Hablar de equipo de trabajo o categoría es sinónimo. A los elementos que conforman la estructura organizativa se les llama unidades organizativas.

Coberturas de servicio

Así se tendrá que definir para cada unidad organizativa de equipo de trabajo las **coberturas de servicio** semanales, es decir, el número de trabajadores que se necesitará para cada día de la semana de manera que la funcionalidad del servicio (para ese equipo) esté cubierta por completo. Por ejemplo, un servicio hospitalario de urgencias de enfermería necesitará de lunes a viernes 5 trabajadores al día y los fines de semana 4. El servicio tiene una necesidad de 5 trabajadores de lunes a viernes y 4 el sábado y el domingo. La cobertura semanal es de 5 trabajadores de lunes a viernes y 4 trabajadores sábado y domingo. Como se puede observar las coberturas dependen de lo realizado por el trabajador y dicho trabajo es el turno o actividad.

Turno

Un trabajador realizará una tarea diaria en un horario continuo que definiremos como **turno**, conocido también como actividad. Un turno puede ser la realización de un trabajo por ejemplo de 8:00 de la mañana a 15:00 de la tarde. Sin embargo no se trabaja sobre días sueltos, sino en semanas.

Ciclo

Un turno se incluye dentro de un patrón, llamado también ciclo semanal. Un **ciclo** es un conjunto de turnos agrupados por semanas. Por ejemplo, un administrativo trabaja en un turno de 7:30 a 15:00 de la tarde en una semana de lunes a viernes, librando sábados y domingos. Este trabajador tiene un ciclo de una semana de 5 días de trabajo por la mañana y dos días libres correspondientes al fin de semana.

Calendario laboral

Los trabajadores desarrollan su actividad en **calendarios laborales,** con días festivos, por lo que los patrones se tendrán que planificar sobre dichos calendarios, esto es, expandiéndose sobre ellos.

Puestos de trabajo

Sin embargo lo que se planifica no son trabajadores, sino los puestos de trabajo, ya que en un puesto pueden rotar varios trabajadores, como por ejemplo ausentes y suplentes. Por tanto podría decirse que un **puesto** es un tipo especial de unidad organizativa, que depende de un equipo de trabajo. Así un equipo de trabajo estará formado por un conjunto de puestos.

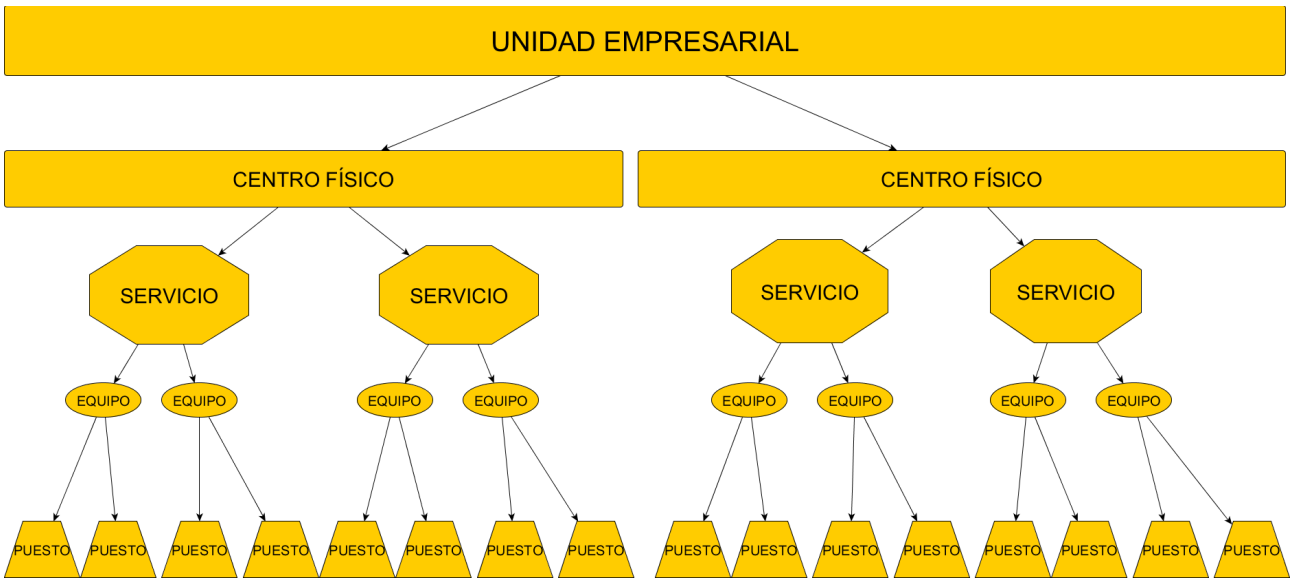


Figura 3. Estructura organizativa.

Planilla

La definición y configuración de estas entidades conforman la creación de la **planilla**, que es una tabla que representa un mes de un calendario laboral, donde cada columna es un día del mes y cada fila un trabajador de la unidad organizativa, siendo la intersección de filas y columnas el turno diario del trabajador, representado por abreviaturas.

Todos estos conceptos teóricos tienen un sentido práctico que se presentan en la siguiente sección.

Flujo de trabajo

Un centro físico se compone de varios servicios funcionales. Por ejemplo, una empresa de venta de gasóleo con varias estaciones de servicio tendrá varios centros físicos, donde cada centro físico será una estación. Cada estación tendrá un servicio funcional cuyo objetivo será el repostaje de vehículos. En dicho servicio habrá expendedores para servir el combustible y administrativos para llevar la gestión y documentación. Posiblemente si la estación sea grande haya un servicio de cafetería, con camareros, cocineros, pinches. Podría darse el caso que haya un hotel asociado a la estación, por lo que habría otro servicio funcional, el hotel, con diversos grupos de trabajadores, administrativos, ayudantes, dirección, etc. Así un servicio funcional tendrá varios grupos de trabajadores, donde un grupo se corresponde con una categoría profesional (pinche, administrativo, informático). Para cada grupo de categorías profesionales hay un conjunto de puestos de trabajo.

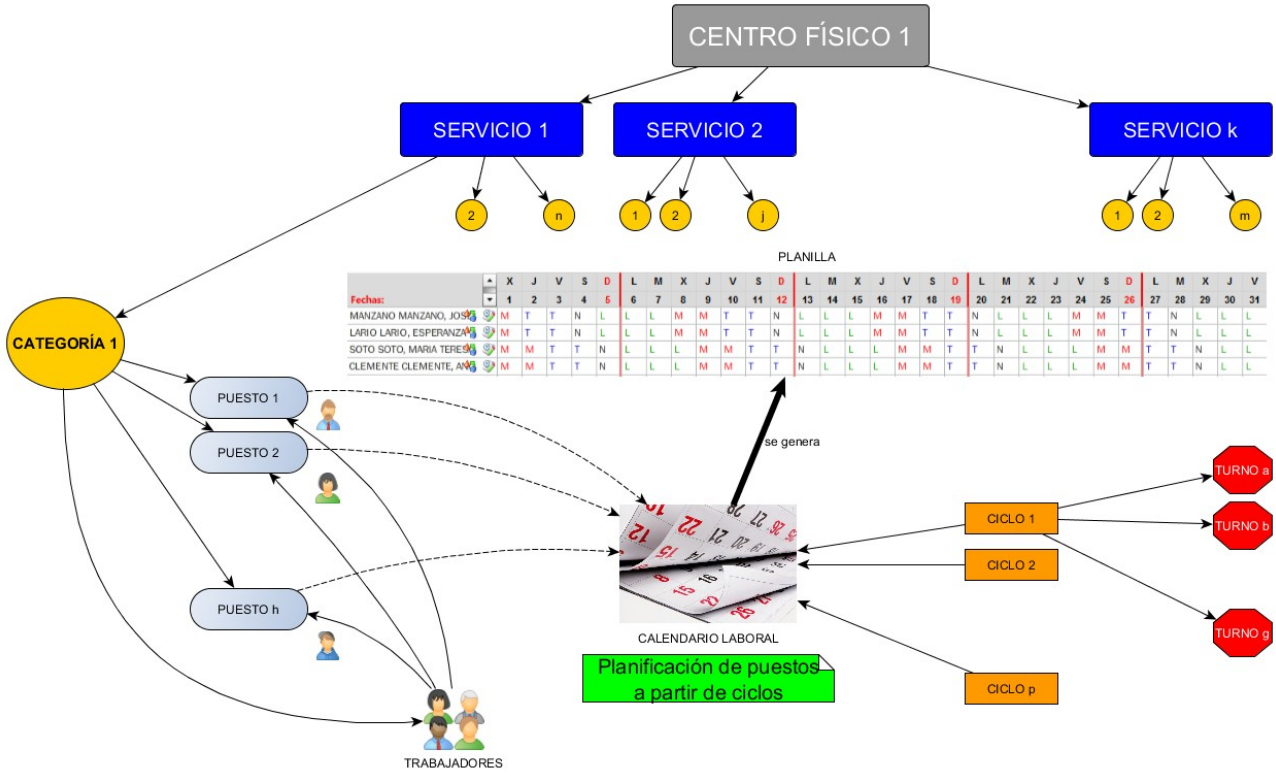


Figura 4. Flujo de trabajo.

Siguiendo con el ejemplo, en una estación de servicio habrá 2 expendedores de combustible, un cajero/a para cobro y tienda y un administrativo para

gestión. Por tanto se necesitará para la categoría expendedor dos puestos de trabajo, 1 puesto para el cajero y 1 puesto para administrativo. Lo normal es que un grupo pueda estar formado por más de una categoría profesional, por tanto una categoría profesional podría estar formada por varias, lo que comúnmente se llama equipos de trabajo.

Por otra parte, tal como se ha comentado un trabajador tiene un turno de trabajo, que se agrupa en ciclos o patrones semanales y que se planifican sobre calendarios laborales. Sin embargo lo que se planifica es el puesto de trabajo, no el trabajador, ya que este es el que rota sobre el puesto o puestos. Una vez que los puestos están planificados, se pasa a asignar a los trabajadores a dichos puestos. Este proceso se realiza sobre una planilla.

wShifts realizará una gestión del cuadrante mediante:

- Cambios de turnos: Un cambio de turno es modificar el turno de un trabajador para un día determinado y cambiarlo por otro turno válido. Por ejemplo, un trabajador cuyo turno original es de mañanas pero lo cambia (no viene por la mañana) y va a ir a trabajar por la tarde. Cambia el turno de mañana por un turno de tarde.

- Inserción de ausencias: Una ausencia es cualquier situación en donde estando el trabajador con un turno planificado no está en su puesto de trabajo. Así para trabajadores que trabajen de lunes a viernes en horario de mañanas los fines de semana no estarán de ausencia, ya que tienen un turno de libranza, es decir, una actividad que implica que no vienen a trabajar. Sin embargo unas vacaciones o días de libre disposición si son ausencias.

- Control de coberturas de servicio.

Y la generación de la plantilla de trabajadores mediante:

- Gestión de puestos de trabajo.
- Resumen de balance horario.
- Gestión de trabajadores.
- Gestión de tareas: Una tarea es la asignación de un trabajador a un puesto de trabajo para un periodo laboral definido por un rango de fechas. El puesto está planificado en la planilla y el trabajador se asocia al puesto. En consecuencia un trabajador puede tener varias tareas.

Capítulo II – Estudio de viabilidad

Estado inicial del sistema

La empresa **M·3 Informática**, que será identificada de aquí en adelante como *el cliente,* es donde trabajo actualmente. Este proyecto y las prácticas externas asociadas al trabajo fin de máster se basan por tanto el la relación laboral con esta empresa.

El cliente está planteando migrar todo el sistema de capa de presentación de sus soluciones software a entorno web con HTML5 y CSS3 debido a que sus productos software utilizan una tecnología que pronto quedará obsoleta. El cliente ha estado estudiando diferentes alternativas teniendo predilección por opciones que estén patrocinadas por grandes empresas y corporaciones, para asegurar en cierta medida que la tecnología tendrá un respaldo lo suficientemente importante como para asegurar la estabilidad del desarrollo de sus productos.

Para ello se requiere el planteamiento de un proyecto basado en web, con las tecnologías anteriores expuestas y que trate sobre el modelo de negocio del cliente, siempre y cuando no haga competencia directa con su software propietario. El cliente basa su modelo de negocio en la gestión de Recursos Humanos y Nóminas para el ámbito sanitario, y requiere desarrollar un sistema básico de gestión de turnos de personal, con licencia de software libre, más concretamente Apache 2.0.

El cliente quiere desarrollar una aplicativo de gestión de turnos de personal con una nueva tecnología en la capa de presentación para verificar que la solución propuesta es válida, y de este modo evaluar la conveniencia o no de una migración total de todos sus productos software.

Situación actual y requisitos planteados

Este gestor de turnos tendrá una funcionalidad básica de gestión de cambios de turnos y ausencias sobre una planilla, compuesta de un conjunto de puestos de trabajo y trabajadores asociados a dichos puestos. Se requiere además la creación de un sistema jerárquico de estructura de unidades organizativas. Los costes deben de ser mínimos, ya que este proyecto tiene que servir como base para la migración de un sistema mayor. Dicha migración podrá basarse, si el cliente así lo considera oportuno, en el trabajo realizado en este proyecto. Sin embargo es importante recalcar que este proyecto de gestión de turnos tiene una línea de desarrollo independiente y podrá ser totalmente funcional.

La parte clave en este desarrollo con licencia de software libre es que no puede afectar al producto de gestión de turnos del cliente por lo que el diseño del aplicativo debe ser suficientemente diferente al gestor de turnos del cliente y con una funcionalidad limitada con respecto a este último.. Además no habrá una gestión de recursos humanos completa un módulo de nóminas. El proyecto únicamente gestionará coberturas de servicio, cambios de turno e inserción de ausencias mediante una planilla.

En el sistema del cliente el servidor del aplicativo de turnos está íntegramente desarrollado en Java, con el uso de procedimientos almacenados, funciones y disparadores de bases de datos de Oracle. En este sentido el proyecto a realizar estará desarrollado en Python, con bases de datos SQLite para sistemas monousuario y postgreSQL para sistemas multiusuario,. Por cuestiones de tiempo se desarrollará el proyecto para un sistema monousuario ya que trabajar con SQLite es más rápido y menos costoso a nivel de requisitos del sistema y de complejidad de carga de datos, al no necesitar de un servidor ejecutándose como un servicio en memoria. La parte cliente del gestor a migrar está desarrollado en **Flex**. En este nuevo proyecto la parte del cliente estará desarrollada íntegramente en **HTML5, CSS3** y un framework para la vista y el controlador de la capa de presentación.

Como requisito fundamental se plantea en un principio el uso del framework Angular 2, de Google, como herramienta de control de vista y controlador y TypeScript como el lenguaje de desarrollo de esta capa ya que son las tecnologías seleccionadas que formarán parte de la migración. El principal problema de esta elección es que a fecha de la propuesta del proyecto esta tecnología estaba aún en fase de pruebas, y con continuos cambios en su diseño. Se obtuvo una versión final y estable en Noviembre de 2016. El segundo requisito es que el nuevo proyecto tiene que estar basado en servicios web, sin embargo no se especificó ninguna tecnología para su desarrollo. En este caso se ha seleccionado Python como lenguaje de desarrollo del modelo y el framework Flask (basado en Python) como controlador de las peticiones de servicios web. La elección de estas herramientas se ha basado en la facilidad de su uso y en la rapidez a la hora de desarrollo de una solución software El tercer requisito es el uso de HTML5 y CSS3. En el siguiente capítulo se presentan esquemáticamente los requisitos del aplicativo así como el análisis de componentes.

Capítulo III – Análisis

Definición del sistema

En la siguiente tabla se muestran los requisitos del sistema junto con los usuarios que intervienen en la definición del requisito del sistema y en su validación:

REQUISITOS BÁSICOS DEL SISTEMA	USUARIOS
Diseño de una aplicación web, con uso de servicios web para acceso a bases de datos. El frontend se desarrollará con el framework Angular 2 y con el lenguaje de programación TypeScript. Se hace necesario el uso de HTML5 y CSS3.	Eduardo Méndez González Camino Arias Villanueva
Diseño de un sistema de gestión de turnos de personal básico, sin gestión de RRHH y Nómina. La solución no puede ser competencia directa del aplicativo propietario del cliente.	Ángel L. García García Camino Arias Villanueva
La planificación diaria debe tener una funcionalidad básica de visualización de turnos de trabajadores, modificación de turnos e inserción de ausencias.	Ángel L. García García Camino Arias Villanueva
La licencia del nuevo proyecto debe ser Apache 2.0	Eduardo Méndez González

Tabla 1. Definición del sistema.

Establecimiento de requisitos

A continuación se completan los requisitos definidos en el punto anterior contando con la información suministrada por los usuarios, en especial Camino Arias Villanueva, jefa de proyecto del cliente, y Eduardo Méndez González, gerente de la empresa cliente. Se crea una tabla donde cada fila es un nuevo requisito.

REQUISITOS	TIPOLOGÍA
La seguridad de la aplicación está basada en el uso de usuarios y roles, donde un usuario se identifica con una persona física y un rol es un conjunto de permisos sobre ciertos recursos de la aplicación. Un rol o perfil puede componerse de uno o más roles. Una persona estará asociada a un rol, a varios roles o a un rol compuesto de otros roles. Un recurso de la aplicación es cualquier componente que por sus características de funcionamiento se *modulariza* para su uso. Así un recurso puede ser un proceso, una ventana, un botón o el acceso a visualizar cierta información.	[SEGURIDAD]

Acceso a la aplicación a través de login con contraseña.	[SEGURIDAD]
El cálculo de balances horarios tiene que hacerse desde código y no mediante consultas SQL a la base de datos, para evitar ralentización del sistema.	[RENDIMIENTO]
Creación de estructura organizativa basada en unidades funcionales jerárquicas dependientes por niveles: Nivel 1: Centro Físico, Nivel 2: Servicio Funcional, Nivel 3: Grupo Funcional (equipo de trabajo), Nivel 4: claves (puestos de trabajo).	[FUNCIONAL]
Posibilidad de configurar turnos a partir de un código y un rango horario.	[FUNCIONAL]
Configuración de patrones o ciclos a partir de los patrones creados.	[FUNCIONAL]
Posibilidad de crear planificaciones a partir de la expansión de ciclos o patrones sobre calendarios laborales.	[FUNCIONAL]
Creación de calendarios laborales por centros físicos, que sirvan en las planificaciones para poder desarrollar los ciclos sobre días festivos.	[FUNCIONAL]
Creación de puestos de trabajo que puedan asignarse a equipos de trabajo y que serán los que se planifiquen, esto es, se planifican puestos y no trabajadores.	[FUNCIONAL]
Configuración de coberturas de servicio por equipos de trabajo, para poder determinar las necesidades del número de trabajadores necesarios por día de la semana.	[FUNCIONAL]
Los equipos de trabajo estarán definidos por las categorías profesionales que los trabajadores tengan por contrato a la hora de asignarlos a los puestos de trabajo.	[FUNCIONAL]
Se registrará una jornada teórica por centro físico, que es el número de horas que tiene que realizar un trabajador en un año natural, descontados los días de vacaciones y libres disposición (asuntos particulares).	[FUNCIONAL]
En la planilla se podrán modificar los turnos que se hayan planificado por otos nuevos, de manera que afecte al total del balance horario del trabajador. De la misma manera se podrán insertar ausencias, que previamente tendrán que estar establecidas y que podrán gestionarse (crear nuevas, configurarlas, etc).	[FUNCIONAL]
Se permitirá dar de alta a trabajadores nuevos, y asignarlos a los equipos de trabajo con la categoría profesional adecuada.	[FUNCIONAL]
Todas las pantallas de la aplicación podrán ser exportadas a formato CSV, siendo el formato de grid (rejilla) lo más parecido a una hoja de cálculo.	[FUNCIONAL]
Se podrán incluir servicios previos en el balance del trabajador, como horas que tienen que contemplarse para el total de horas trabajadas.	[FUNCIONAL]
El sistema tendrá la posibilidad de poder instalarse en entornos Linux y Windows, tanto la parte del servidor como la del cliente.	[IMPLANTACIÓN]
El balance de un trabajador tendrá que tener en cuenta la planificación que hace al año, los cambios de turno, las ausencias que ha sufrido (y que pueden contar horas como trabajadas o no), los servicios previos y la jornada teórica.	[FUNCIONAL]
Se hace necesario incorporar un repositorio de informes que ataque directamente a la base de datos.	[FUNCIONAL]
Creación de un sistema de asignaciones de trabajadores a puestos en los equipos de trabajo atendiendo a su categoría profesional.	[FUNCIONAL]

La creación de ciclos tendrá que tener en cuenta que no se podrán solapar turnos en franjas horarias, al igual que en las planificaciones, que tampoco pueden solaparse.	[FUNCIONAL]
Se podrá configurar qué días de la semana pueden ser festivos.	[FUNCIONAL]

Tabla 2. Establecimiento de requisitos.

Interfaces de usuario

Definición de perfiles

En la siguiente tabla se recoge la relación de perfiles (roles) de acuerdo a las funcionalidades/operativas asignadas dentro de la aplicación. Cada fila corresponde a un único perfil.

NOMBRE DEL PERFIL	DESCRIPCIÓN DEL PERFIL	FUNCIONALIDAD
Administrador	Administrador de la aplicación.	Administración y configuración del aplicativo para su correcto funcionamiento. Tiene permisos de ejecución y visualización de toda la aplicación: módulos de seguridad, configuración, recursos humanos y planilla.
Usuario	Usuario de la aplicación	Uso del módulo de Planilla. Tiene permiso para visualizar el cuadrante (cambiar meses y años), insertar ausencias y modificar turnos.
Usuario (solo lectura)	Usuario de la aplicación en modo solo lectura.	Uso del módulo de Planilla. Tiene permiso únicamente para visualizar la planilla (cambiar meses y años).

Tabla 3. Definición de perfiles.

Componentes de interfaz y perfiles

En este apartado se estudiará el diseño de interfaz según el tipo de perfil que haga uso de dicha interfaz. Esto es, la interfaz variará dependiendo del perfil que haga uso de ella. Se presenta tabla donde cada fila es la relación entre el perfil y una interfaz, describiendo la funcionalidad representada. Un perfil puede tener varias interfaces. Un interfaz puede ser accedida por varios tipos de perfil.

INTERFAZ	PERFIL	FUNCIONAMIENTO
Inicio de aplicación (Login)	• Administrador • Usuario • Usuario (solo lectura)	Inserción de usuario y contraseña para acceso al sistema. Si el usuario no es válido o no se inserta no se podrán acceder a ningún módulo, excepto el de Acerca de y Ayuda.

Tabla 4. Interfaz de inicio de aplicación y perfiles.

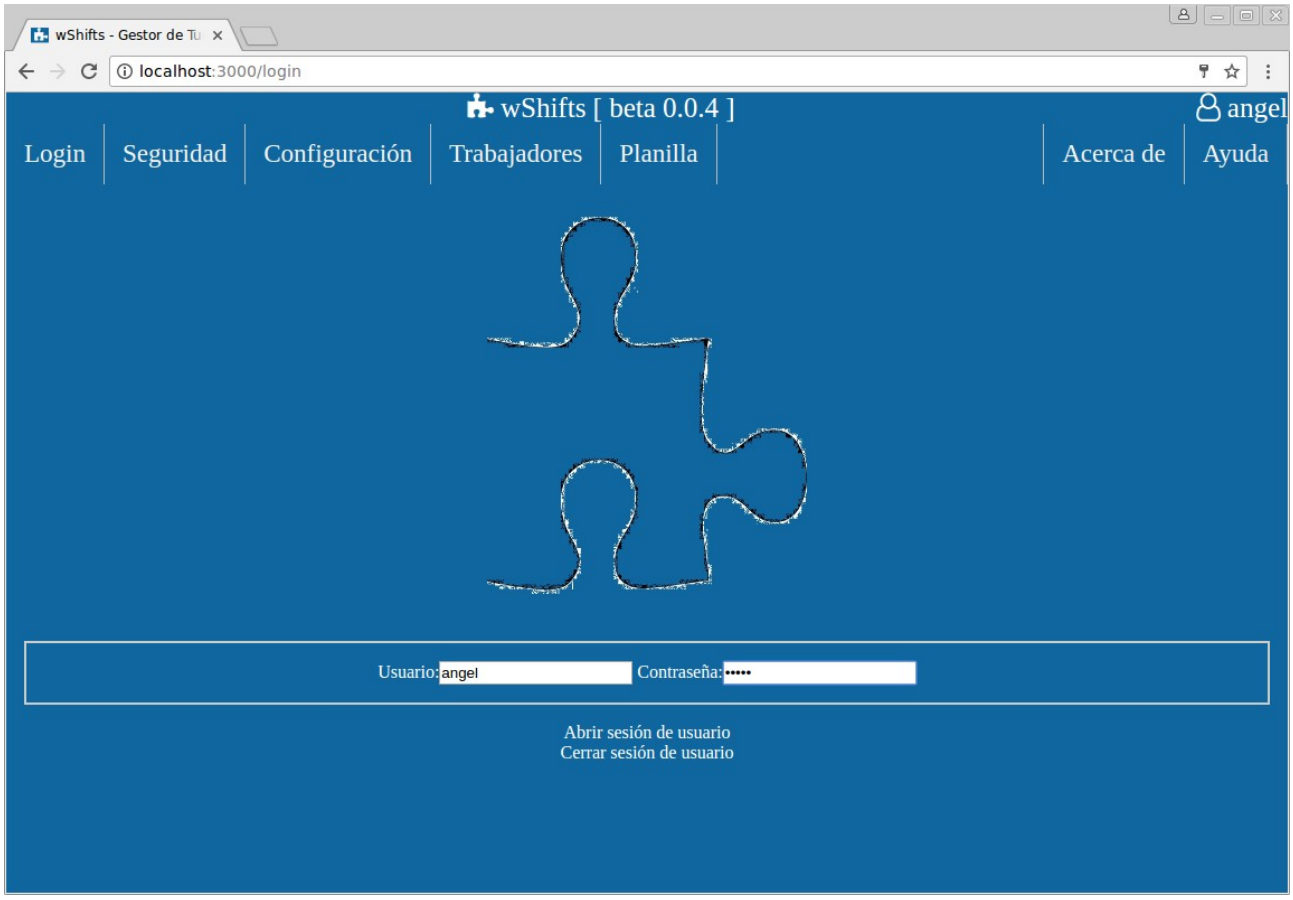


Figura 5. Interfaz de inicio de la aplicación. Para acceder al aplicativo es necesario el ingreso de usuario y contraseña. Si las credenciales no son válidas no se podrá acceder a los módulos de Seguridad, Configuración, Trabajadores y Planilla. Si se accede a la aplicación correctamente en la esquina superior derecha aparecerá el usuario con el que se accedió al aplicativo y se podrán acceder a los recursos en donde el usuario tenga permisos.

Interfaz	**Perfil**	**Funcionamiento**
Seguridad	• Administrador	Gestión de usuarios que acceden al aplicativo. Gestión de roles (perfiles) de usuario, definición de recursos del aplicativo, asociación de roles a recursos, asociación de roles a usuarios y asociación de usuarios a unidades organizativas dentro de la estructura organizativa.

Tabla 5. Interfaz de seguridad y perfiles.

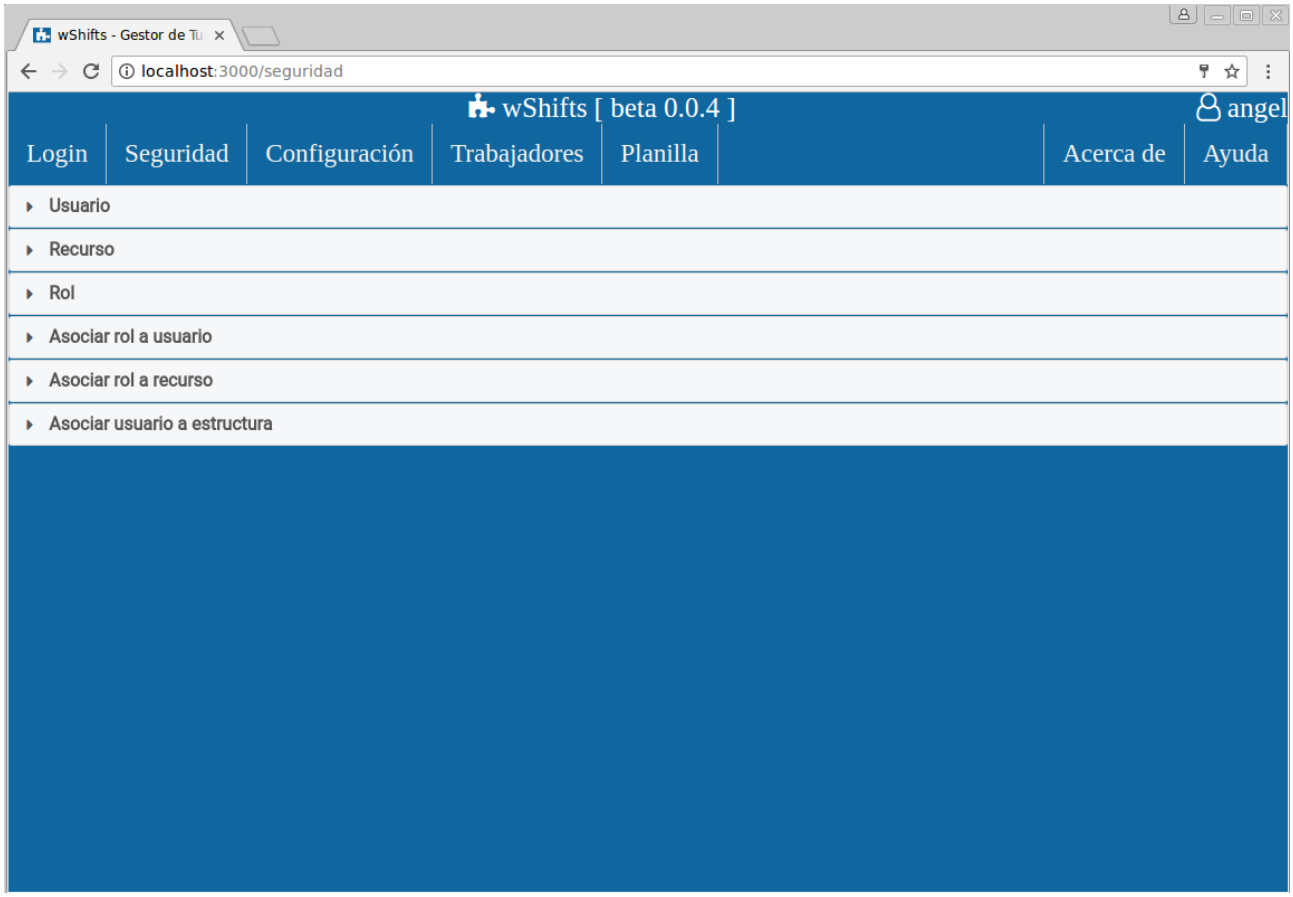


Figura 6. Interfaz de seguridad de la aplicación. Configuración de usuarios y roles del aplicativo. A partir de unos recursos preestablecidos de la aplicación se asocia el acceso de roles a recursos. A su vez, los usuarios se asocian a roles y acceden a través de ellos a los recursos. Un usuario podrá acceder únicamente a las unidades organizativas a las cuales tenga acceso dentro de la estructura organizativa.

INTERFAZ	PERFIL	FUNCIONAMIENTO
Configuración	• Administrador	Configuración del aplicativo. Se divide en la creación de la estructura organizativa (centros físicos, servicios funcionales, equipos de trabajo), la creación de turnos, ciclos, puestos y sus planificaciones, coberturas de servicio, calendario de festivos, jornada teórica anual por centro físico, tipos de ausencias, categorías profesionales que serán asociadas a equipos de trabajo, las categorías profesionales y los cargos que se usan para la configuración de contratos. Además hay un mantenimiento de datos básicos de la aplicación para poder modificar los días festivos de la semana.

Tabla 6. Interfaz de configuración y perfiles.

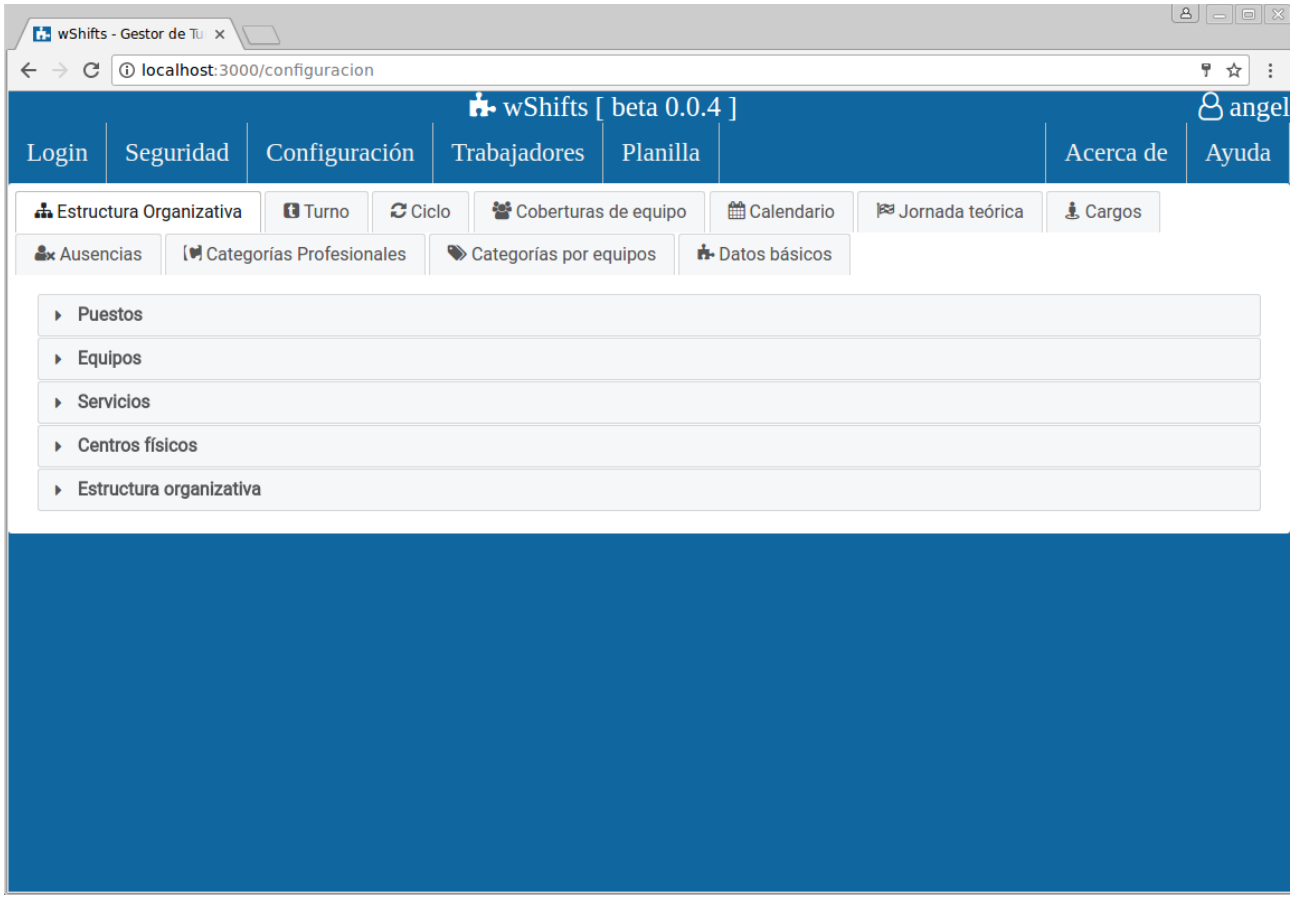


Figura 7. Interfaz de configuración de la aplicación. Configuración de estructura organizativa de la aplicación en unidades organizativas, a partir de puestos, equipos, servicios funcionales y centros físicos. Definición de datos básicos para la gestión de turnos, ciclos, coberturas de servicio de equipos de trabajo, calendarios de festivos por centros físicos y jornadas teóricas anuales por centros físicos. Además se configuran datos básicos para la gestión de recursos humanos, como los tipos de ausencia de trabajadores, las categorías profesionales asociadas a puestos de trabajo y los cargos asociados a contratos laborales. Por último se permite asociar a equipos de trabajo solo trabajadores con una determinada categoría profesional y la configuración de días de la semana que son festivos a nivel de datos básicos del aplicativo.

Interfaz	**Perfil**	**Funcionamiento**
Trabajadores	• Administrador	Gestión de trabajadores. Un trabajador se le creará un contrato entre un periodo temporal con el fin de estar registrado. En dicho contrato se le incluirá la categoría profesional necesaria para ser asignado a un equipo de trabajo y un cargo de responsabilidad. Se podrán gestionar los datos personales de todos los trabajadores, sus servicios previos y las ausencias del trabajador de su puesto de trabajo.

Tabla 7. Interfaz de trabajadores y perfiles.

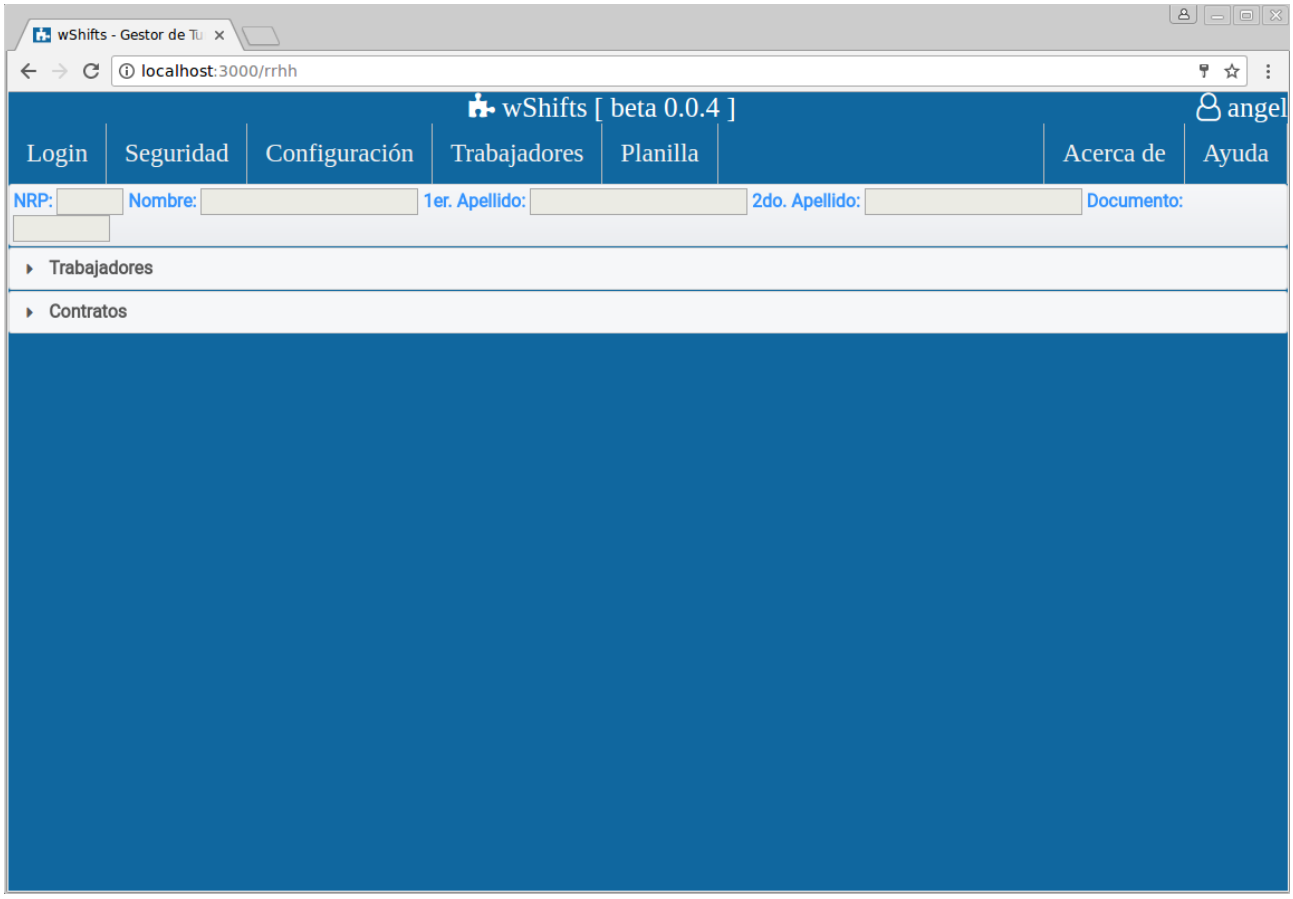


Figura 8. Interfaz de gestión de trabajadores de la aplicación. Gestión de datos de trabajadores así como de sus contratos asociados y servicios previos.

INTERFAZ	PERFIL	FUNCIONAMIENTO
Planilla	• Administrador • Usuario • Usuario (solo lectura)	Asociación de ciclos a puestos de trabajo para su planificación (expansión del ciclo en el calendario laboral). Asignación de puestos de trabajo a trabajadores entre un periodo de fechas, que quedarán reflejados en la Planilla mensual (planificación diaria), que es la rejilla donde está el calendario laboral, con los turnos de los trabajadores y las coberturas de servicio para cada día. En él se podrán insertar ausencias y cambiar los turnos de los trabajadores, si procede. Visualización del balance horario del trabajador, que son las horas trabajadas menos la jornada teórica, para comprobar el cómputo de tiempo trabajado. El *usuario de solo lectura* únicamente podrá acceder a la sección de Planilla, no pudiendo cambiar el turno o insertar una ausencia.

Tabla 8. Interfaz de planilla y perfiles.

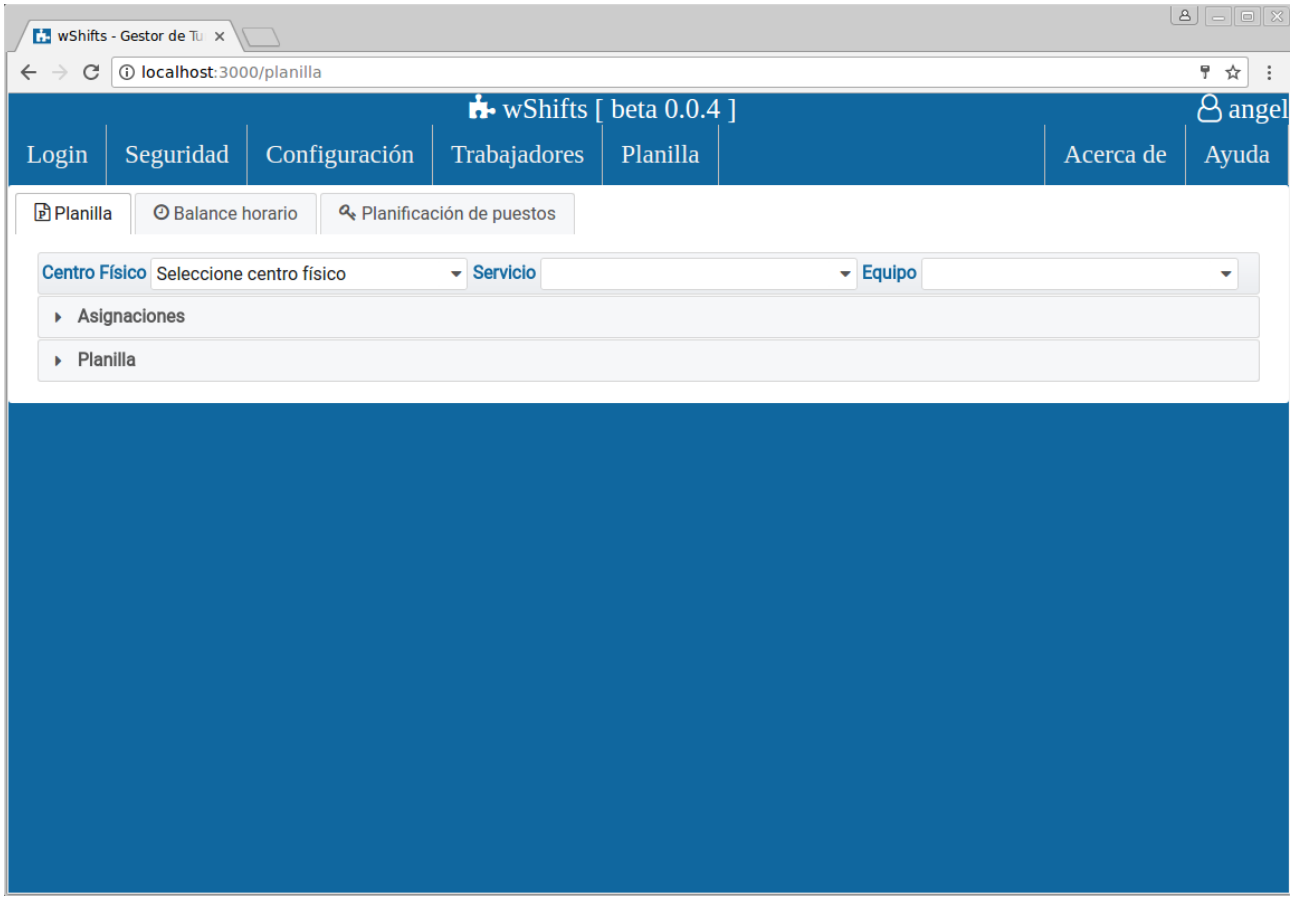


Figura 9. Interfaz de gestión de planilla.

En el **capítulo V**, sección **Funcionamiento Básico** se entra más en detalle sobre los aspectos de cada una de las interfaces principales. Y para poder entender este punto, la operativa básica de funcionamiento, se hace necesaria una introducción elemental a la gestión de turnos de personal, que se detalla a continuación.

Introducción a la teoría de gestión de turnos

Turno o Actividad

Un turno o actividad es cualquier trabajo que se realiza entre un rango de horas continuas para un día determinado. Así un turno estará definido por un nombre, un código y un rango de horas de trabajo. Así por ejemplo para un turno de *Mañana* podemos tener la siguiente configuración:

- Nombre: Mañana
- Abreviatura: M
- Rango horas:
  - ✔ Desde: 8:00
  - ✔ Hasta: 15:00

Coberturas de servicio

Un equipo de trabajo es un conjunto de trabajadores que realizan tareas afines para un mismo objetivo, como por ejemplo un equipo de desarrollo software de una empresa, que podría estar compuesto por diseñadores gráficos, analistas desarrolladores y analistas funcionales. Así un equipo de trabajo, también llamado grupo funcional, tiene una serie de necesidades de cobertura de servicio. La necesidad es la cantidad de personal que es necesario en un día para que el servicio que presta dicho equipo esté cubierto. Siguiendo con el ejemplo anterior el equipo de desarrollo podría necesitar a 4 trabajadores, de forma permanente, de lunes a viernes. Esto significa que para que el servicio esté cubierto se necesitan a 4 trabajadores. La cobertura de servicio es de 4. Si trabajan en horario de mañana, de 8:00 a 15:00, y para un día falta un trabajador de los cuatro, se dice que en dicho día hay una presencia de 3 trabajadores y una deficiencia de 1 trabajador, para una cobertura de 4 trabajadores.

Patrones o ciclos

Una vez que se han definido todas las actividades se pasa a definir los patrones. Un patrón (o ciclo) es un conjunto de actividades (o turnos) que se repiten en un espacio temporal, definido por semanas completas. Por ejemplo, para el personal del equipo anterior de desarrollo se tendría el siguiente ciclo semanal, de lunes a viernes:

L	M	X	J	V	S	D
M	M	M	M	M	L	L

Tabla 9. Ciclo de 1 semana de mañanas de lunes a viernes.

Es decir, trabajan cada día en un turno (actividad) de Mañana (computa 7 horas, de 8:00 a 15:00), y los sábados y domingos tienen una actividad de

Libre (L), que es un turno que no computa horas (computa 0 horas). Este ciclo es de una semana, ya que inicia y termina en un periodo de 7 días. Así cuando un patrón es múltiplo de 7 hará tantas semanas como se definan en dicho patrón. ¿Qué pasa cuando un patrón no es múltiplo de 7? Pues que hará tantas semanas como actividades tenga dicho patrón. Para ver esta casuística se tiene un patrón de Mañana (M), Tarde (T), Noche (N) y Libre (L), esto es, un ciclo compuesto por 4 turnos básicos, tal como sigue:

N.º SEMANA	L	M	X	J	V	S	D
1	M	T	N	L	M	T	N
2	L	M	T	N	L	M	T
3	N	L	M	T	N	L	M
4	T	N	L	M	T	N	L

Tabla 10. Ciclo de 4 semanas, donde se repiten 4 turnos.

El ciclo resultante es de 4 semanas, ya que hay 4 actividades que lo componen, y se tarda justo esa cantidad en iniciar y terminar el patrón por el mismo día de la semana (lunes). En definitiva, se puede observar que la siguiente semana a la finalización de la semana 4 que debe continuar es justamente la semana que empieza por la semana 1, esto es, se ha definido un ciclo.

Planificación

Sin embargo un ciclo no es una unidad para trabajar, puesto que normalmente se trabaja sobre un calendario laboral anual. Para poder trabajar sobre dicho calendario hay que expandir el ciclo sobre él en un proceso denominado planificación, entre un rango de fechas y a partir de la semana del ciclo por la que se desea empezar. Esto es debido a que en un equipo de trabajo, en un momento dado, un trabajador estará en una semana u otra del ciclo dependiendo de cuando haya empezado en su planificación. Se presenta el siguiente ejemplo real de un calendario laboral de un servicio de urgencias de un hospital, para el mes de noviembre de 2016, con un ciclo compuesto por los turnos Mañana (M), Tarde (T), Noche (N) y Saliente de Noche (-) que es un turno especial que no cuenta horas, equivalente al turno Libre anteriormente visto.

L	M	X	J	V	S	D
31 N	1 -	2 M	3 T	4 N	5 -	6 M
7 T	8 N	9 -	10 M	11 T	12 N	13 -
14 M	15 T	16 N	17 -	18 M	19 T	20 N
21 -	22 M	23 T	24 N	25 -	26 M	27 T

28 N	29 -	30 M	1 T	2 N	3 -	4 M

Tabla 11. Ciclo que contiene 4 turnos básicos, M, T, N y - expandido en el mes de noviembre del año 2016.

Si se quisiera planificar a varios trabajadores que empiezan en un ciclo en noviembre habría que comprobar por cual semana del ciclo tiene que empezar, dependiendo del día de la semana en el que se esté y por qué parte del ciclo esté. Así se tendrá:

- Trabajador 1: Está trabajando el día 17 de saliente de noche.

¿Cómo detectar por cuál semana está? El día 17 de noviembre es jueves, y tiene como actividad saliente de noche. En el ciclo se tiene que el único jueves que es saliente de noche es el que corresponde con la semana 1, por tanto el Trabajador 1 estará por la semana 1.

N.º SEMANA	L	M	X	J	V	S	D
1	M	T	N	-	M	T	N
2	-	M	T	N	-	M	T
3	N	-	M	T	N	-	M
4	T	N	-	M	T	N	-

Tabla 12. Ciclo semanal de M, T, N, -. El único jueves cuya actividad es saliente de noche pertenece a la semana 1.

- Trabajador 2: Está trabajando el día 26 de turno de mañana.

¿Cómo detectar por cuál semana empezar? El día 26 de noviembre es sábado, y tiene como actividad Mañana. En el ciclo se tiene un único sábado que es mañana y que corresponde con la semana 2, por tanto el dicho trabajador estará en la semana 2.

N.º SEMANA	L	M	X	J	V	S	D
1	M	T	N	-	M	T	N
2	-	M	T	N	-	**M**	T
3	N	-	M	T	N	-	M
4	T	N	-	M	T	N	-

Tabla 13. Ciclo semanal de M, T, N, -. El único sábado cuya actividad es mañana pertenece a la semana 2.

Este ejemplo es un caso muy básico de detección de semanas por las que empieza el trabajador. Se puede llegar a complicar en exceso cuando los ciclos son de varias semanas (de 8 en adelante) y con varios trabajadores. La semana por la que un trabajador empieza a trabajar es fundamental ya que en un equipo de trabajo lo normal es que no empiecen todos por la misma semana. Por tanto en la planificación se tiene que tener no solo en cuenta el ciclo expandido, sino la semana por la que se empieza a expandir el ciclo en el calendario laboral.

Para poder *montar un equipo de trabajo* basado en la *gestión de turnos* hay que realizar los siguientes pasos:

1. Definir las actividades o turnos que realizarán los trabajadores en su jornada laboral diaria.

2. Definir los ciclos semanales de trabajo.

3. Identificar las semanas por las que empiezan cada uno los trabajadores en un ciclo semanal para poder expandir correctamente los ciclos sobre calendarios laborales (para cada trabajador).

4. Definir el número de trabajadores que se necesitarán por día de la semana para que las necesidades del servicio estén cubiertas.

El proyecto **wShifts** apoya su funcionamiento en esta base teórica, que si bien es simple de entender tiene cierta complejidad en su implantación. Y justo para que la implantación sea correcta se han realizado pruebas para la corrección de errores y test de comprobación de datos que se han ido realizando conforme se ha hecho la construcción de los módulos implementados. Puesto que lo más importante era la generación correcta de datos en el servidor se han hecho scripts de carga de datos, así como de cálculo de planificaciones, ciclos y turnos. En el siguiente capítulo se verá el diseño del aplicativo y se entrará más en profundidad en esta cuestión, esto es, las pruebas de verificación de funcionamiento de la aplicación.

Capítulo IV – Diseño

En el anterior capítulo de *Análisis* se han definido los requerimientos básicos que se tienen que tener en cuenta para el diseño de la aplicación, a saber, la tecnología que debe de usarse y el funcionamiento básico de la aplicación. En este capítulo se mostrará dicha tecnología y en el **capítulo V** su funcionamiento.

Tecnología de diseño

Tal como se especificó en el capítulo anterior hay varias tecnologías que deben de usarse para el diseño del proyecto, en particular para el desarrollo de la parte del cliente se debe de usar el marco de trabajo Angular 2 y el lenguaje de programación TypeScript, que es un super conjunto de JavaScript. Angular utiliza TypeScript porque sus tipos facilitan el soporte a la productividad del desarrollador con herramientas. También se puede escribir código Angular en JavaScript. Además la parte gráfica del cliente debe de ser diseñada con las tecnologías HTML5 y CSS3.

Angular sigue el patrón Modelo Vista Controlador de ingeniería del software, el cual fomenta la articulación flexible entre la presentación, datos y componentes lógicos.

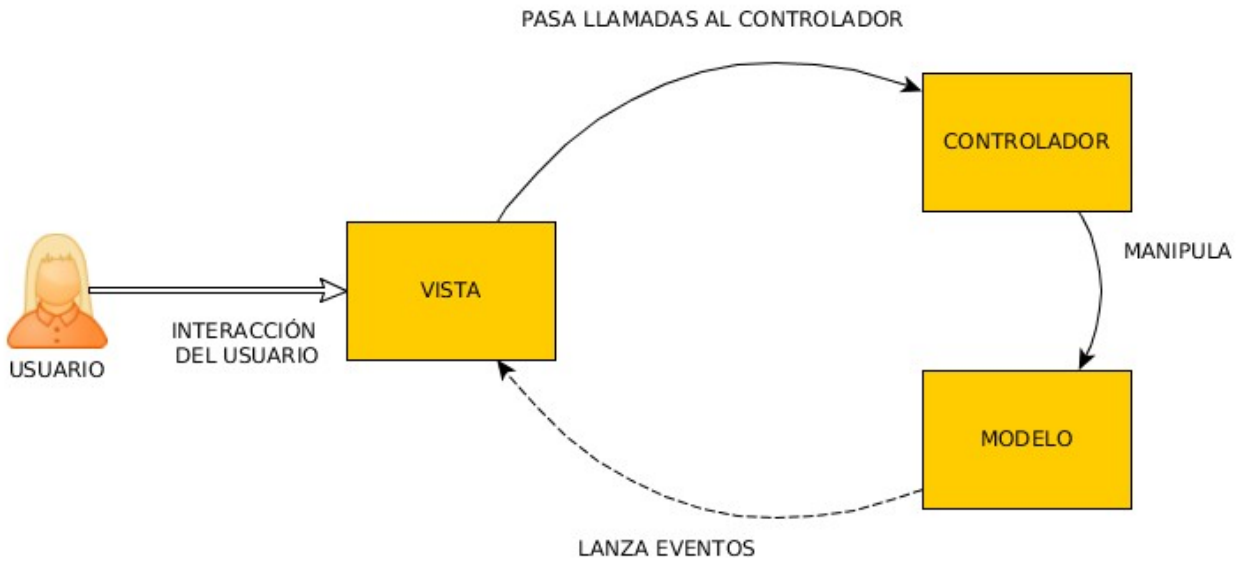


Figura 10. Patrón de diseño Modelo Vista Controlador (MVC). El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.). El controlador recibe la notificación de la acción solicitada por el usuario, gestionando el evento que llega mediante un manejador de eventos. El controlador accede al modelo para recuperar o modificar información de forma adecuada a la acción solicitada por el usuario. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se

reflejan los cambios en el modelo (actualización o recuperación de datos). El modelo no tiene conocimiento directo sobre la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Componentes de Modelo Vista Controlador

Modelo

Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada a un usuario. Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador.

Controlador

Responde a eventos (acciones del usuario) e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta el modelo (por ejemplo que se actualice la forma en la que se muestran los datos). El controlador hace de intermediario entre la vista y el modelo.

Vista

Presenta el modelo (información y *lógica de negocio*) en un formato adecuado para interactuar mediante una interfaz de usuario.

Modelo

Tal como se ha visto en el punto anterior el modelo es la representación de información del sistema, esto es, el repositorio de información, que normalmente viene representado por una base de datos y otros mecanismos para la manipulación de la información del sistema. La forma en la que el controlador accede a los datos del modelo se ha diseñado mediante la implementación de servicios web, que es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Dentro de los servicios web hay diferentes estándares que se pueden utilizar. En este caso, y debido a que es una aplicación web, se opta por el uso de REST (transferencia de estado representacional), que es una arquitectura que haciendo uso del protocolo HTTP proporciona una API (interfaz de programación de aplicaciones) que utiliza cada uno de sus métodos (get, post, put, etc) para la realización de diferentes operaciones entre la aplicación que ofrece el servicio web (parte servidor) y el cliente.

Figura 11. Esquema básico REST API.

Sin embargo el modelo, tal como se ha mencionado anteriormente, no solo está formado por una base de datos sino por otros componentes para la manipulación de los mismos. Además la implementación de REST API debe de realizarse mediante algún lenguaje de programación o marco de trabajo. Para el diseño de ambos elementos se ha optado por el uso de Python, cuya elección se basa en los siguientes elementos que se han evaluado con respecto a otras alternativas:

- Es un lenguaje de programación de propósito general, de muy alto nivel (esto es, un alto nivel de abstracción, con el uso de listas, tuplas, diccionarios).

- Es interpretado (no es necesaria compilación), dinámico (no necesita identificar explícitamente los tipos de datos para inicializar variables, de modo que los tipos se validan durante la ejecución del programa) y fuertemente tipado (no pueden mezclarse tipos, es necesario hacer conversiones).

- Es multiplataforma (Windows, Mac, Linux, etc), multiparadigma (imperativo, orientado a objetos y en menos medida funcional) y con gestión automática de memoria.

- Es un lenguaje de programación con una sintaxis clara y sencilla, fácil de aprender, donde se pueden mezclar los diferentes paradigmas de programación de los que dispone, ampliamente documentado, extensible, que intenta obligar al desarrollador de software a programar de la manera correcta en el menor tiempo posible.

- Tiene licencia de código abierto (PSFL) compatible con GPL de GNU a partir de la versión 2.1.1.

Específicamente para la implementación de REST API se ha optado por el uso del marco de trabajo minimalista Flask, escrito en Python, que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código.

Vista y controlador

El diseño de la vista y controlador están basados en el marco de trabajo Angular 2, que usa el lenguaje de programación TypeScript para su implementación. Características técnicas de este lenguaje son las siguientes:

- Es un lenguaje de programación libre y de código abierto desarrollado y mantenido por **Microsoft**.
- Es un superconjunto de **JavaScript**, que esencialmente añade tipado estático y objetos basados en clases. Puede ser usado para desarrollar aplicaciones JavaScript que se ejecutarán en el lado del cliente o del servidor.
- Extiende la sintaxis de JavaScript, por tanto cualquier código JavaScript existente debe funcionar correctamente. Está pensado para grandes proyectos, los cuales a través de un compilador de TypeScript se traducen a código JavaScript original.

Angular 2 es un marco de trabajo desarrollado y mantenido por Google, para ser usado en diseño de aplicaciones que siguen el patrón Modelo Vista Controlador. El diseño gráfico de la vista, esto es, la capa de presentación donde el usuario interacciona con la aplicación a través de un navegador web, está escrita en el lenguaje de marcas HTML5 y se utiliza CSS3 para incluir hojas de estilo; dar una presentación mejorada visualmente a la página web. Angular tiene como misión la implementación del controlador con el fin de capturar los eventos que provienen de la vista (por ejemplo el usuario pulsa un enlace, un botón, etc.), realizar tareas de validación de datos, implementación de reglas de negocio (por ejemplo cómo mostrar la información, el flujo de visualización de ventanas, etc.), y envío de datos al modelo para actualizar o recuperar información.

En resumen, se tiene que el proyecto sigue un patrón de diseño Modelo Vista Controlador, donde el modelo estará compuesto por una base de datos y otros ficheros, incluido un servicio web, basado en REST API y escrito en Python. Así mismo el controlador estará implementado en el marco de trabajo Angular 2 y la parte de la vista estará escrita en HTML5 y CSS3. La aplicación sigue un modelo de arquitectura cliente – servidor, donde el usuario interacciona con la parte del cliente (vista y controlador) y la parte servidor (modelo) recibe peticiones de información de la parte del cliente. En la siguiente sección se tratará este punto.

Flujo de datos

La arquitectura cliente – servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta.

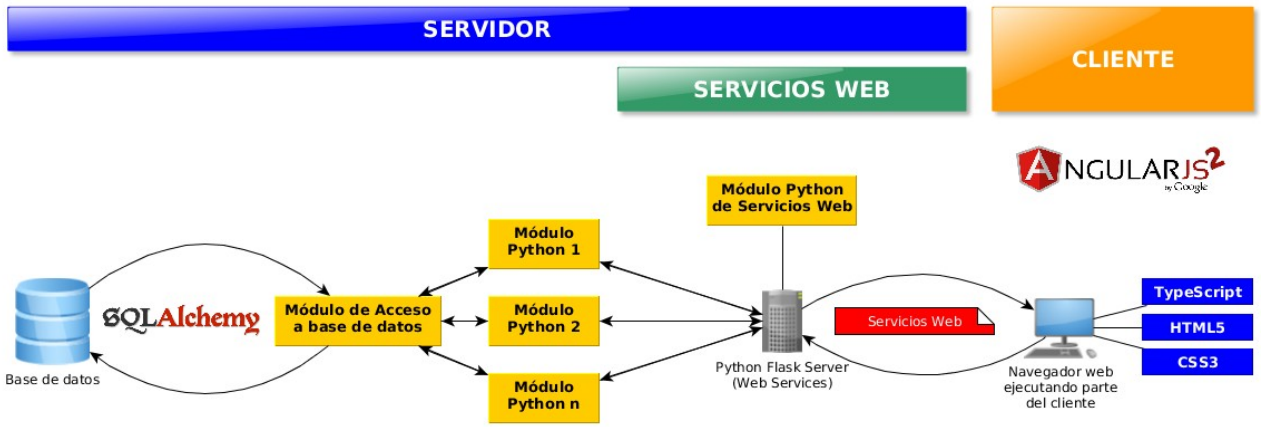


Figura 12. Arquitectura cliente – servidor para wShifts.

El navegador del usuario ejecuta la parte del cliente, que es una página web cuya presentación está hecha con HTML5 y CSS3. El controlador está implementado en Angular 2. Cuando se realiza una petición para obtener información o modificar datos se realiza una petición a un servicio web de un servidor Flask, el cual llama a unos módulos de reglas de negocio, que realizan ciertas tareas de conversión y comprobación, para a su vez enviar la petición a un módulo de acceso a la base de datos. Este módulo a partir del software SQLAlchemy, que es un ORM (mapeador objeto – relacional; un mecanismo de abstracción de base de datos, para simplificar el acceso a información almacenada en un sistema gestor de bases de datos utilizando la programación orientada a objetos), accede a la base de datos, actualizando información o recuperando la misma. Dicha información vuelve al módulo de acceso a base de datos, devolviendo a su vez la información al módulo Python que realizó la petición. Esta información viaja a partir de un servicio web a la parte cliente, y en la vista se presentan los resultados esperados.

Tal como puede apreciarse en la Figura 12 toda la información está almacenada en una base de datos, esto es, un repositorio de información persistente, que en este caso sigue un modelo relacional **Chen – Codd**, y que se presenta en el siguiente apartado.

Modelo de datos

El modelo de datos usado para persistir la información de la aplicación como las planificaciones, turnos o estructura organizativa se basa en el relacional, donde los datos están agrupados en tablas e interrelacionados mediante relaciones. El proyecto se ha diseñado por módulos por lo que el diseño de la base de datos tiene también esta característica. En el **Anexo III** se presenta el *diagrama de Entidad – Relación* completo. Así se pueden diferenciar mediante conjuntos de tablas (entidades) y relaciones entre ellas los siguientes módulos:

- **Módulo de seguridad**: En este módulo se encuentran las tablas y relaciones que componen toda la seguridad de aplicación de usuarios que acceden a la misma. Un usuario de aplicación tiene asignado un rol (función que una persona desempeña en la aplicación). Un rol puede ser un usuario normal que accede a los trabajadores de su grupo de trabajo, un supervisor (responsable de varias unidades organizativas), un administrador (acceso a toda la aplicación), etc. Este rol puede acceder (ejecutar, ver o abrir) a uno o varios recursos, entendiendo como recurso a cualquier parte de la aplicación o elemento (botón, proceso, módulo de aplicación, etc). Este modelo de negocio está implementado por las tablas *recurso*, *recurso_rol*, *rol*, *rol_usuario* y *usuario*.

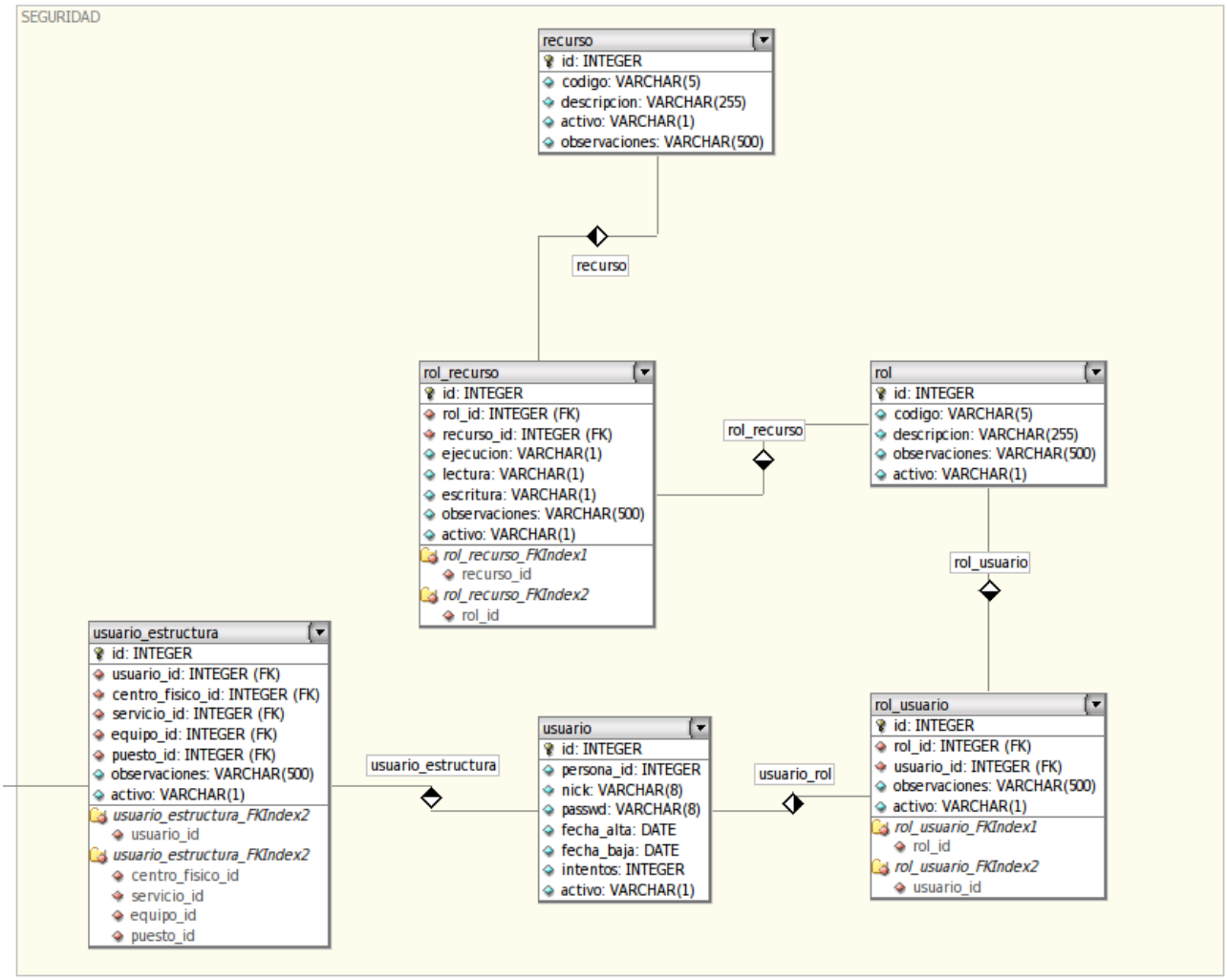


Figura 13. Diagrama de entidad – relación del módulo de Seguridad.

Además un usuario de aplicación puede acceder a determinadas unidades organizativas de la estructura. Por ejemplo, un responsable del grupo de Enfermería del servicio de Urgencias de un hospital podrá acceder a su grupo, pero no tiene porque acceder al grupo de Celadores de dicho servicio. Esta lógica está implementada por la tabla *usuario_estructura* y *usuario.*

- **Módulo de Estructura**: El módulo de estructura representa la jerarquía organizativa de la aplicación, basada en los tipos de unidades organizativas puesto, grupo de trabajo, servicio funcional y centro físico (tabla *tipo_unit*). Cada unidad organizativa deberá ser de uno de estos tipos, con una configuración determinada (tabla *unit*). La estructura organizativa se basará en tuplas (centro físico, servicio, grupo, puesto), teniendo una jerarquía estructural arbórea plana (tabla estructura).

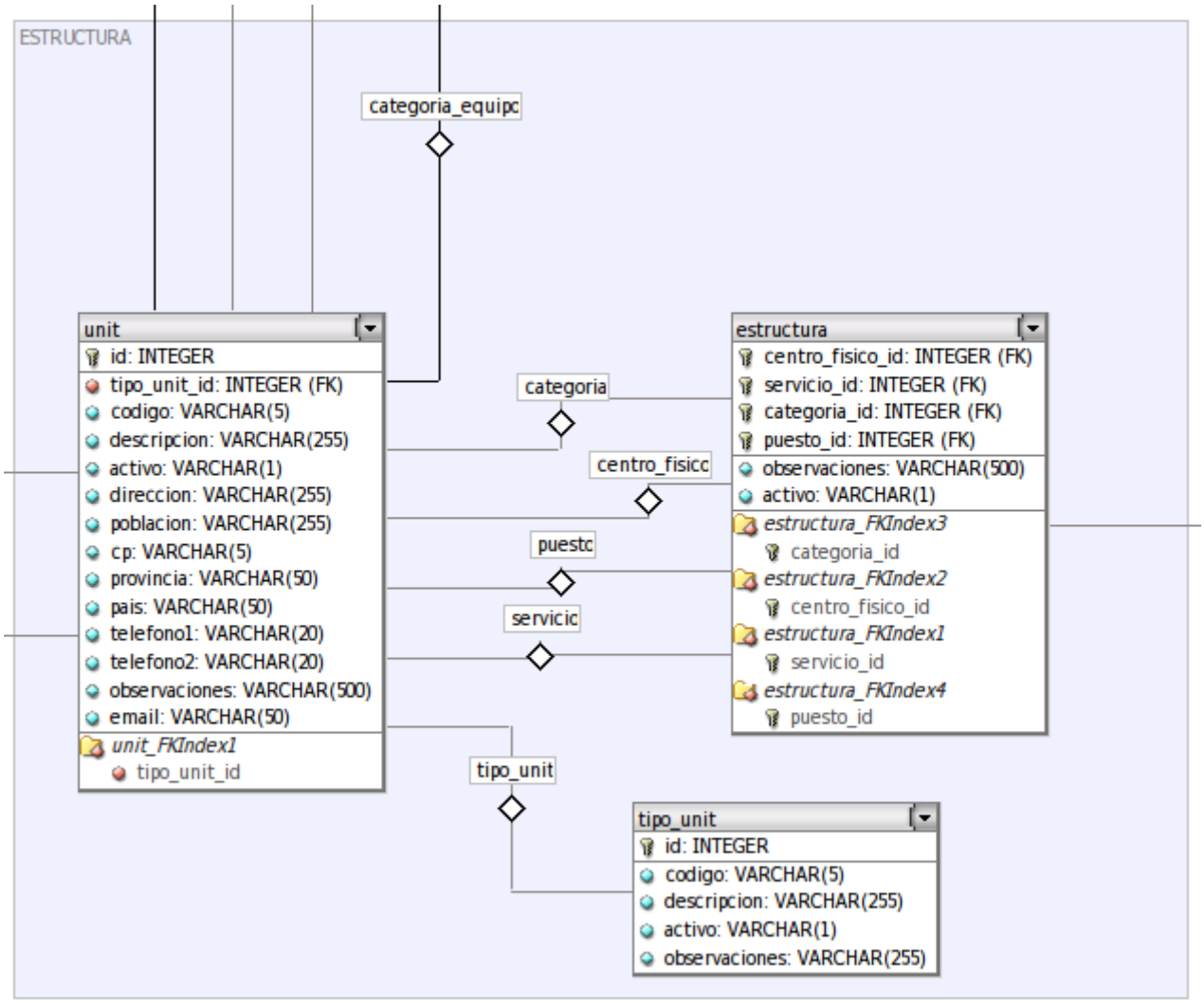


Figura 14. Diagrama de entidad – relación del módulo de Estructura.

- **Módulo de Turnos**: Aquí se guardan las configuraciones de los turnos (tablas *turno_master* y *turno_detail*) y los ciclos, que son los conjuntos de turnos que se repiten en un ciclo semanal (tablas *ciclo_master* y *ciclo_detail*).

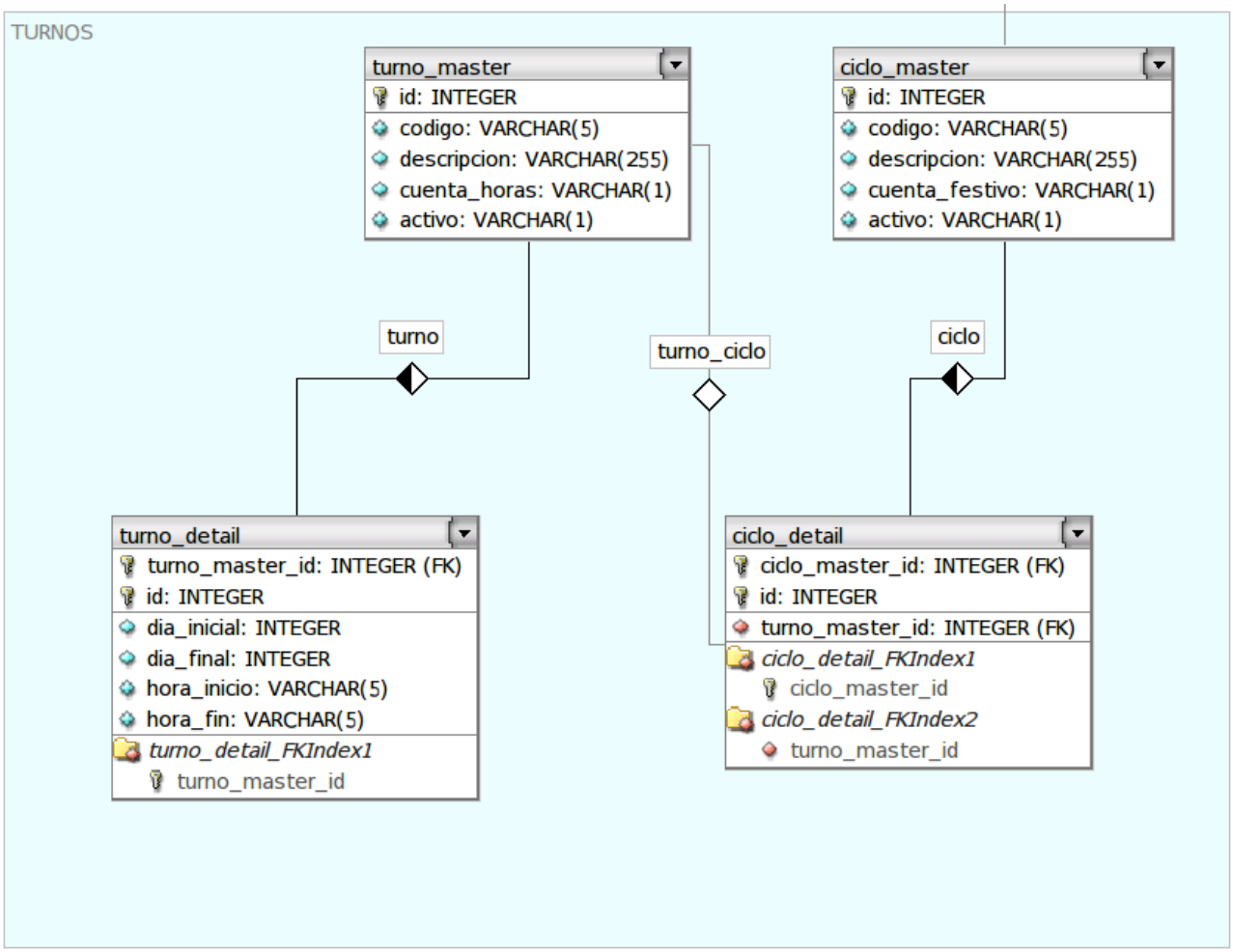


Figura 15. Diagrama de entidad – relación del módulo de Turnos.

- **Módulo de Planificación**: En el módulo de planificación se almacenan las expansiones de los ciclos en el calendario laboral por meses (tabla *planificacion*), los cambios de turno en la planilla (tabla *cambio_turno*), los días festivos por centros físicos (tabla *calendario_festivo*). Por ejemplo un centro físico en Madrid tendrá como festivo el 2 de Mayo (día de la Comunidad de Madrid) y un centro físico en Murcia tendrá como festivo el 9 de Junio (día festivo de la Región de Murcia). Un puesto puede tener asociado una o varias planificaciones (tabla *puesto_ciclo*), y un equipo de trabajo tendrá unas necesidades de coberturas de servicio diarias entre un rango de fechas (tabla *cobertura_equipo*).

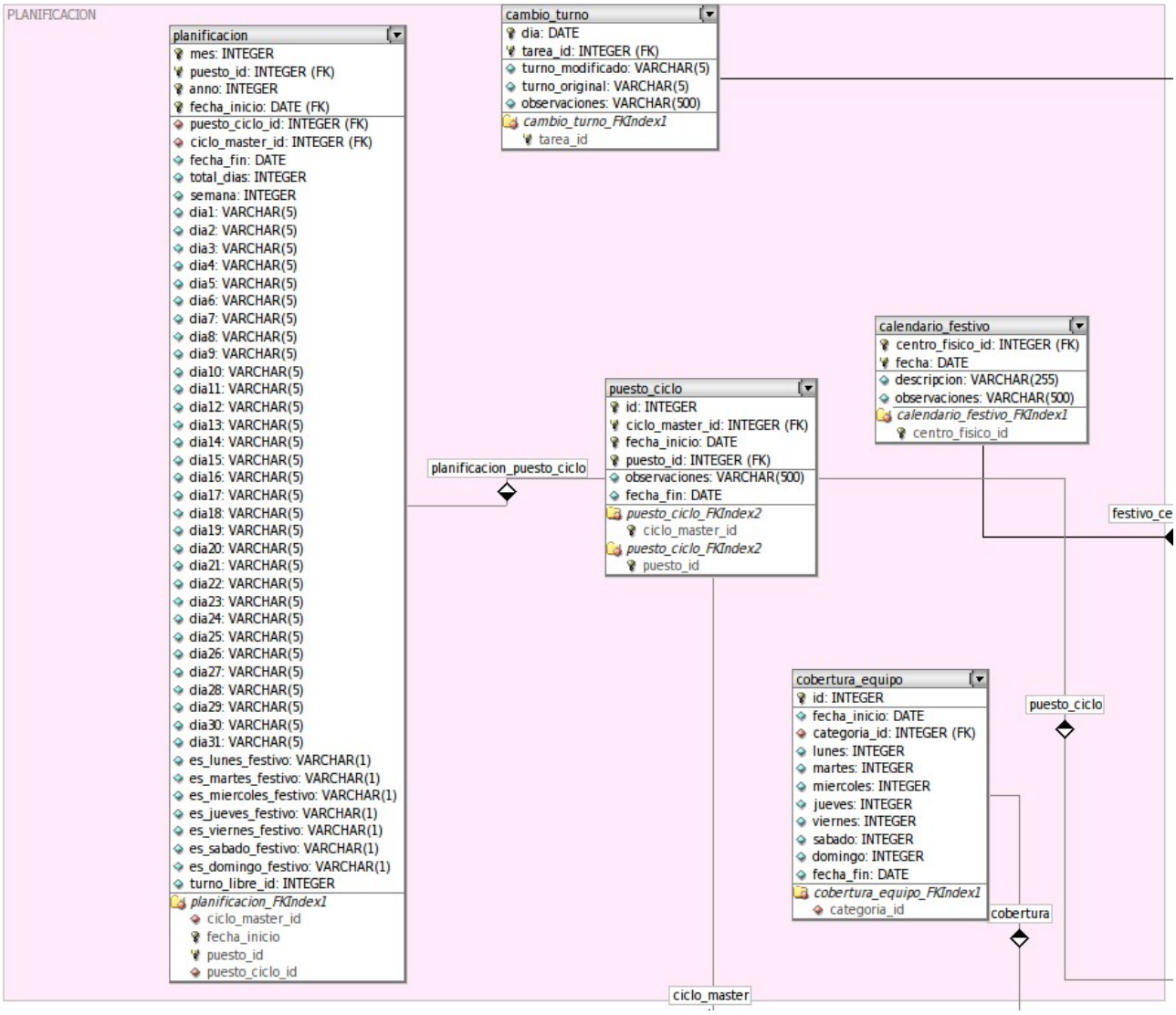


Figura 16. Diagrama de entidad – relación del módulo de Turnos.

- **Módulo de Recursos Humanos**: Toda la información relativa a los trabajadores de la organización se encuentran en este módulo, tales como sus datos personales (tabla *persona*), sus servicios previos por año (tabla *servicios_previos*), que son la cantidad de horas que se deben de tener en cuenta aparte de lo que se calcule por las horas planificadas en la planilla, y sus contratos (tabla *contrato*). Un contrato puede tener asociado una o varias ausencias (tablas *contrato_ausencia*) a partir de un tipo de ausencia predeterminado (tabla *ausencia*). La configuración de un contrato requiere de un tipo de cargo (tabla *cargo*), que puede ser director, gerente, informático, etc. Además cada contrato tiene asociado una categoría profesional, que se define como la capacidad del puesto de trabajo, esto es, familias profesionales (tabla *categoria_profesional*), como por ejemplo ingeniero informático, facultativo, maestro, etc. Finalmente un equipo de trabajo podrá incluir trabajadores de una categoría profesional o de varias (tabla *categoria_equipo*). Lo habitual es que en un equipo de profesionales solo estén los que pertenezcan a una misma familia profesional. Pero podría darse el caso de equipos de

trabajo con diferentes categorías profesionales. Por ejemplo, en un equipo de enfermería de un servicio hospitalario de Neonatos pueden convivir en la misma planilla auxiliares de enfermería y enfermeros, ya que pueden realizar la misma función y realizar las mismas coberturas del servicio.

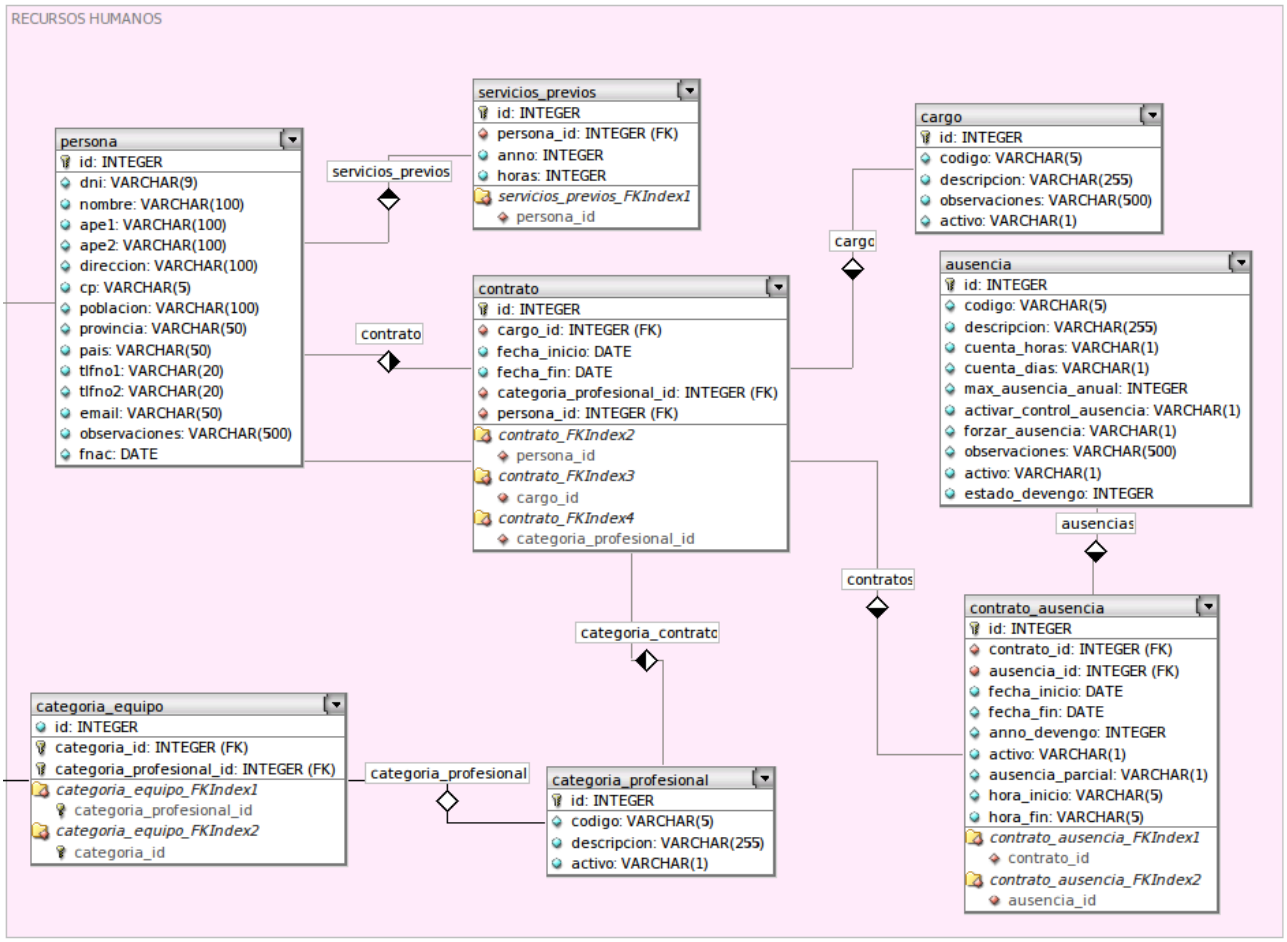


Figura 17. Diagrama de entidad – relación del módulo de Recursos Humanos.

- **Módulo de Jornada Teórica**: En este módulo se registra el total de horas que tiene que realizar un trabajador anualmente por centro físico (tabla *jornada_teorica*).

- **Módulo de Configuración**: La configuración básica de aplicación, como versión, nombre de aplicación, días festivos semanales, etc, están en este módulo (tabla *basica*).

- **Módulo de Asignación**: El proceso de relacionar un puesto planificado con un contrato de un trabajador para un periodo temporal (entre fechas) se denomina asignación. De esta manera se asigna un puesto de trabajo a un trabajador a partir de su contrato (tabla *tarea*). Un trabajador (en realidad su contrato) puede tener una o varias tareas (asignaciones), no solapadas en el tiempo.

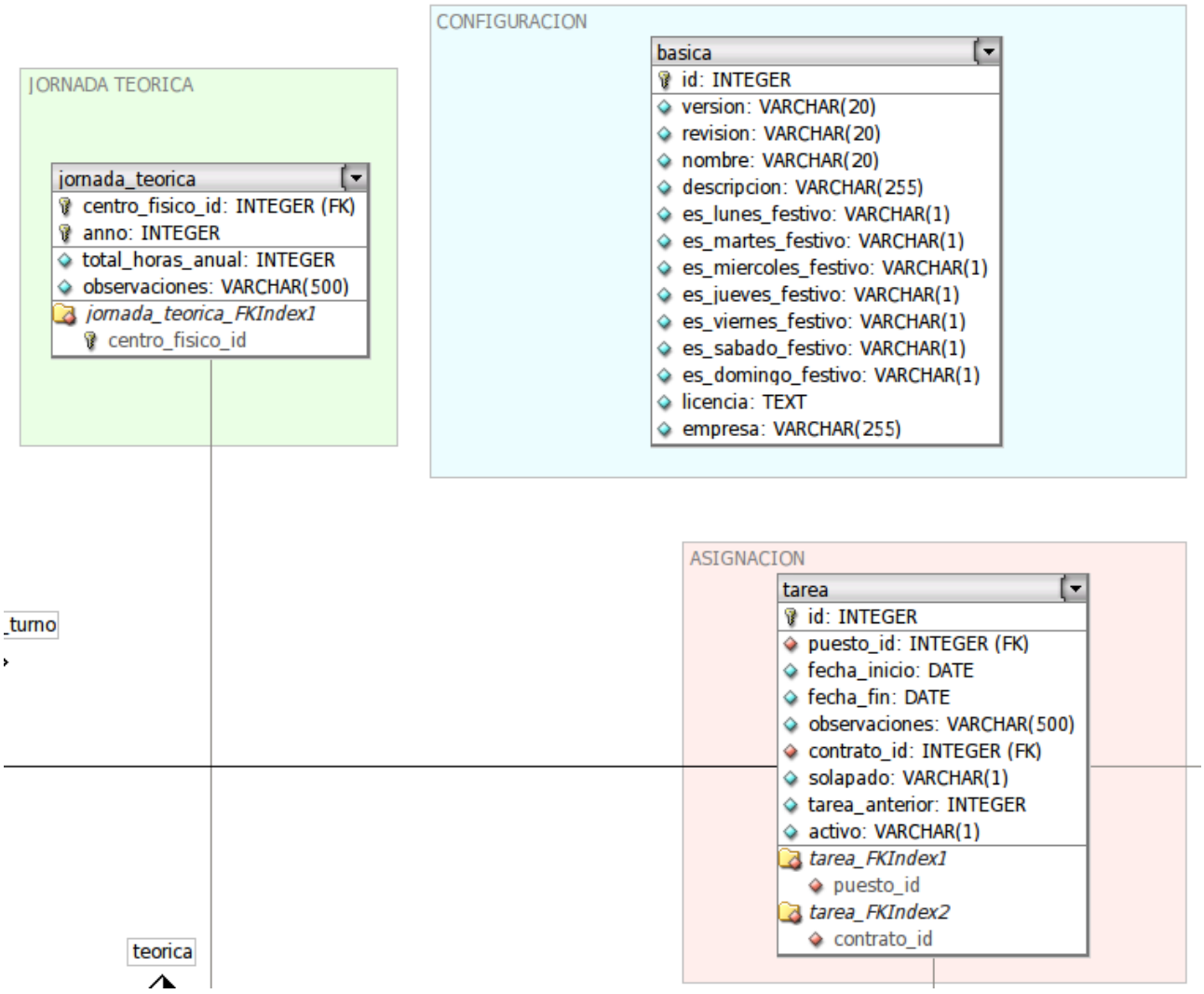


Figura 18. Diagrama de entidad – relación del módulos de Asignación, Básica y Jornada Teórica.

Toda la información persistente en la base de datos es accedida a través del mapeador relacional **SQLAlchemy** a través de módulos **Python**, que a su vez envían dicha información a otros módulos para su tratamiento por reglas de negocio. En el siguiente punto se analiza en profundidad esta cuestión.

Reglas de negocio

Tal como se vio en el punto anterior los datos se almacenan en una base de datos relacional, pero antes deben ser tratados, tanto para su lectura como su escritura por entidades que apliquen reglas de negocio, verificando la información semántica (dando sentido a la información) y sintáctica (evitando almacenar datos que no tengan sentido). Este trabajo se realiza en módulos Python, que implementan, mediante el paradigma de programación orientada a objetos, todo un sistema de validación de reglas de negocio para la problemática de gestión de turnos de personal.

En la figura 12 de la página 35 se presentó el esquema conceptual de arquitectura cliente – servidor para este proyecto. Una vez que los datos son accedidos (ya sea para leer o escribir en el repositorio de información) por el módulo de acceso a base de datos, se envían a otros módulos (también denominadas librerías) intermedios. En dichos módulos están las reglas de negocio, definidas por clases Python.

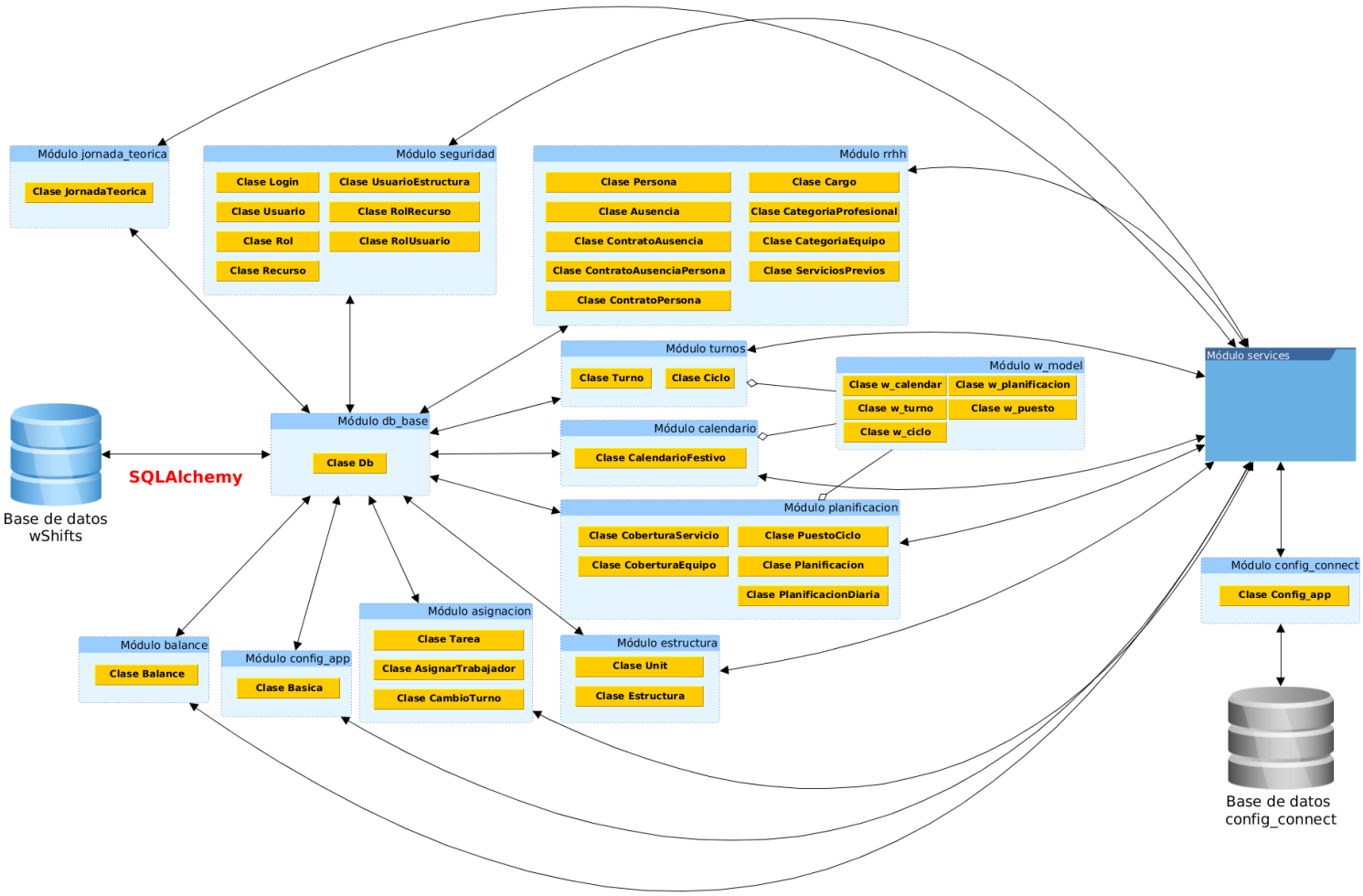


Figura 19. Representación de modelo de negocio en parte servidor.

El módulo *db_base* es el denominado módulo de acceso a bases de datos. Es la única interfaz de conexión con las base de datos, y todas las peticiones de

lectura o escritura en la base de datos deben de ser gestionadas por este módulo a partir del ORM SQLAlchemy.. La aplicación tiene su punto de entrada en el módulo *services*, que es un servidor de peticiones de servicios web. En dicho módulo están instanciadas todas las clases de la aplicación que contienen la lógica de negocio, más la clase *Config_app* del módulo *config_connect*, que es la interfaz de configuración de la aplicación. Dicha configuración se encuentra en una base de datos llamada *config_connect*, y contiene datos sobre la base de datos donde se encontrará la información del gestor, así como la configuración de cadenas de conexión y otros datos básicos para poder ejecutar la aplicación. El resto de módulos componen el núcleo de la aplicación, a saber:

- Módulo seguridad: Incluyen las clases para gestionar los usuarios que accederán a la aplicación (clase *Usuario*), los roles (clase *Rol*), los recursos (clase *Recurso*), la asignación de usuarios a roles (clase *RolUsuario*) y roles a recursos (clase *RolRecurso*), el acceso de usuarios a la aplicación (clase *Login*) y finalmente los permisos de usuarios para acceder a los elementos de la estructura organizativa (clase *UsuarioEstructura*).

- Módulo w_model: Toda la lógica de los conceptos de turno (clase *w_turno*), ciclo (clase *w_ciclo*), planificación (clase *w_planificacion*), calendario laboral (clase w_calendar) y puesto de trabajo (clase *w_puesto*) se recoge en este módulo. La configuración y validación genérica de los turnos y forma en la que se planifican ciclos en calendarios laborales vienen contempladas en estas clases. Es el módulo más importante de la aplicación, ya que contiene la generalización de los componentes básicos. Estas clases son utilizadas por otros módulos, no teniendo acceso al módulo de acceso a bases de datos. Estas clases sirven para crear el modelo de negocio a otros módulos, como se verá mas adelante.

- Módulo turnos: Este módulo genera la lógica de los turnos (clase *Turno*) y ciclos (clase *Ciclo*). Utiliza las clases genéricas del módulo *w_model*, para implementar las reglas de negocio y a partir de ellas poder comunicarse con el módulo de acceso a bases de datos y con el módulo *services*, que es el punto de entrada de la aplicación.

- Módulo planificacion: La forma en la que un ciclo se expande en el calendario laboral, su control, verificación de reglas, como por ejemplo evitar solapamientos temporales entre planificaciones, etc, y las asignaciones de ciclos a puestos de trabajo están contempladas en este módulo (clases *Planificacion* y *PuestoCiclo*). Además este módulo genera las planificaciones de la planilla, que es donde el usuario podrá ver la planificación de los trabajadores (clase *PlanificacionDiaria*) y las coberturas de servicio de dicha planilla (clase *CoberturaServicio*), que recoge el total de personas que hay por día en una planificación. La configuración del número de personas que se necesita por día semanal para cubrir las necesidades de un grupo también se define en este

módulo (clase *CoberturaEquipo*). Este módulo también utiliza las clases genéricas del módulo *w_model*, para implementar las reglas de negocio y a partir de ellas poder comunicarse con el módulo de acceso a bases de datos y con el módulo *services*, que es el punto de entrada de la aplicación.

- Módulo calendario: Aquí se generan los calendarios con días festivos por centro físico, su control y las verificaciones de datos válidos y consistentes (clase *CalendarioFestivo*). Al igual que los dos módulos anteriores también utiliza las clases genéricas del módulo *w_model*, para implementar las reglas de negocio y a partir de ellas poder comunicarse con el módulo de acceso a bases de datos y con el módulo *services*.

- Módulo rrhh: Este es el módulo de recurso humanos, donde se recogen los datos personales de los trabajadores (clase *Persona*), sus servicios previos asociados (clase *ServiciosPrevios*) que son la cantidad de horas por año que se se suman al balance anual del trabajador y los contratos de cada trabajador (clase *ContratoPersona*). Cada contrato podrá tener asociado una o varias ausencias del trabajador (clases ContratoAusencia y *ContratoAusenciaPersona*), a partir de tipos de ausencias (clase Ausencia). Los contratos tienen asociados cargos profesionales (clase *Cargo*) y categorías profesionales (clase *CategoriaProfesional*) y por último un equipo de trabajo puede tener asociada una o varias categorías profesionales (clase *CategoriaEquipo*).

- Módulo estructura: La configuración de unidades organizativas (clase *Unit*) y la estructura organizativa (clase *Estructura*) de dichas unidades en tuplas de centros físicos, servicios, equipos y puestos están implementadas en este módulo.

- Módulo asignacion: En este módulo se recogen las clases para asignar trabajadores a puestos de trabajo que han sido planificados (clases *Tarea* y *AsignarTrabajador*) y el mecanismo para realizar cambios de turnos en la planilla (clase *CambioTurno*).

- Módulo jornada_teorica: Contiene la clase *JornadaTeorica*, que maneja las jornadas teóricas anuales por centro físico para cada trabajador (el total de horas que tiene que realizar por año y centro físico).

- Módulo balance: Contiene las clases necesarias para realizar los cálculos de obtención de balances horarios de trabajadores (clase *Balance*), entendiendo balance como la diferencia entre el total de horas trabajadas y la jornada teórica anual. Si el balance es positivo la empresa “debe” horas al trabajador y si es negativo el trabajador “debe” horas a la empresa.

- Módulo config_app: Módulo de configuración de datos básicos de aplicación (clase Basica).

Visualización: parte del cliente

Tal como se ha comentado en capítulos anteriores esta aplicación es web, basando su funcionalidad en el uso de un navegador web por parte del usuario. Se ha optado por su desarrollo en HTML5 y CSS3 para la parte de presentación gráfica y el marco de trabajo Angular 2 de Google para manejar los eventos producidos por la interacción del usuario con la aplicación (por ejemplo la ejecución de una acción como consecuencia de pulsar sobre un botón de la aplicación). Para escribir código Angular 2 se pueden utilizar varios lenguajes de programación, como Dart o JavaScript. En este caso se opta por desarrollar en TypeScript de Microsoft.

La arquitectura de una aplicación Angular se realiza mediante *componentes*. Desde la aplicación más básica de Angular 2, el *Hola Mundo*, todo tiene que comenzar por un componente. Así una aplicación Angular consta de un árbol de componentes de n niveles, desde el componente padre, componentes hijos, hijos de sus hijos, etc. Un componente es equivalente a una etiqueta HTML para realizar las operaciones que sean oportunas. Un componente puede ser cualquier cosa dentro de una página web, desde una sección de navegación a un formulario, o un campo de formulario.

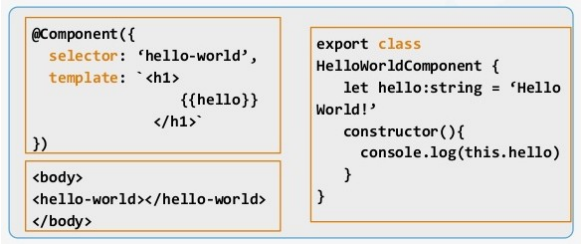


Figura 20. Código Angular 2 basado en TypeScript, HTML y CSS.

Para definir el contenido de esta nueva etiqueta, el componente, se usa HTML, CSS y TypeScript. El uso de componentes mejora la forma de organización de una aplicación, su mantenimiento, reutilización del código, etc.

La parte cliente de este proyecto se basa en un componente específico llamado *ag-grid*, que un elemento de rejilla o tabla, extensamente configurable. Todos los elementos para presentar, modificar o crear cualquier tipo de dato o información se basan en este componente. Y puesto que se utiliza programación orientada a objetos para el desarrollo de este proyecto, se ha creado un componente ag-grid general, que implementa las operaciones de inserción, modificación y eliminación de datos, así como de recuperación de conjuntos de datos a partir de unos parámetros predefinidos. Cualquier parte de la aplicación que requiere de este tipo de funcionalidad implementa la funcionalidad de este componente mediante la herencia del mismo.

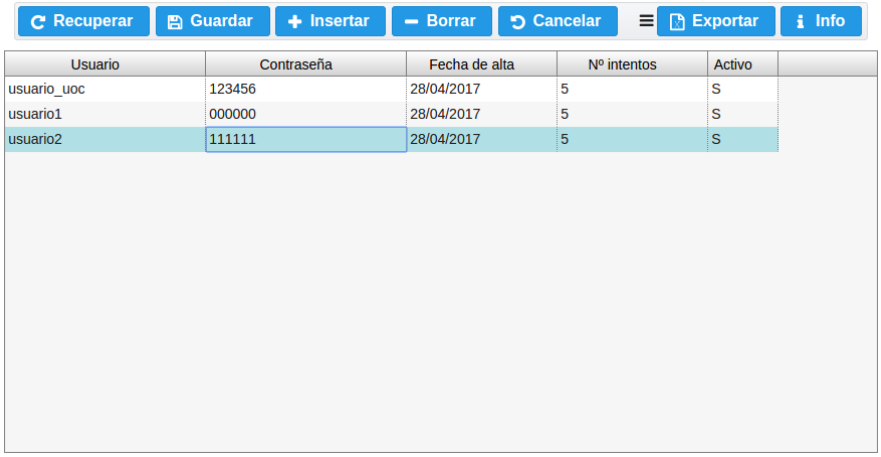


Figura 21. Mantenimiento de usuarios de la aplicación. La tabla con datos hereda del componente *ag-grid*, que contiene las funcionalidades de recuperación, inserción, modificación y borrado de elementos. La acción de pulsar sobre los botones de este mantenimiento lanza eventos que Angular captura e infiere al componente tabla, para la realización de la operación pertinente.

Por tanto todos los mantenimientos de la aplicación se basan en el componente *ag-grid* para la manipulación de datos. Para el resto de componentes gráficos se ha optado por el uso del marco de trabajo *primeNG*, una colección de componentes de interfaz de usuario de la empresa PrimeTek (primeng 2016). Su elección frente a otros paquetes de software se basa fundamentalmente en que en la fecha de realización del proyecto no había otro paquete de software con tantos componentes terminados para Angular 2, tales como paneles, botones, combos de selección, acordeones, visualización de mensajes, contenedores de paneles con pestañas, etc.

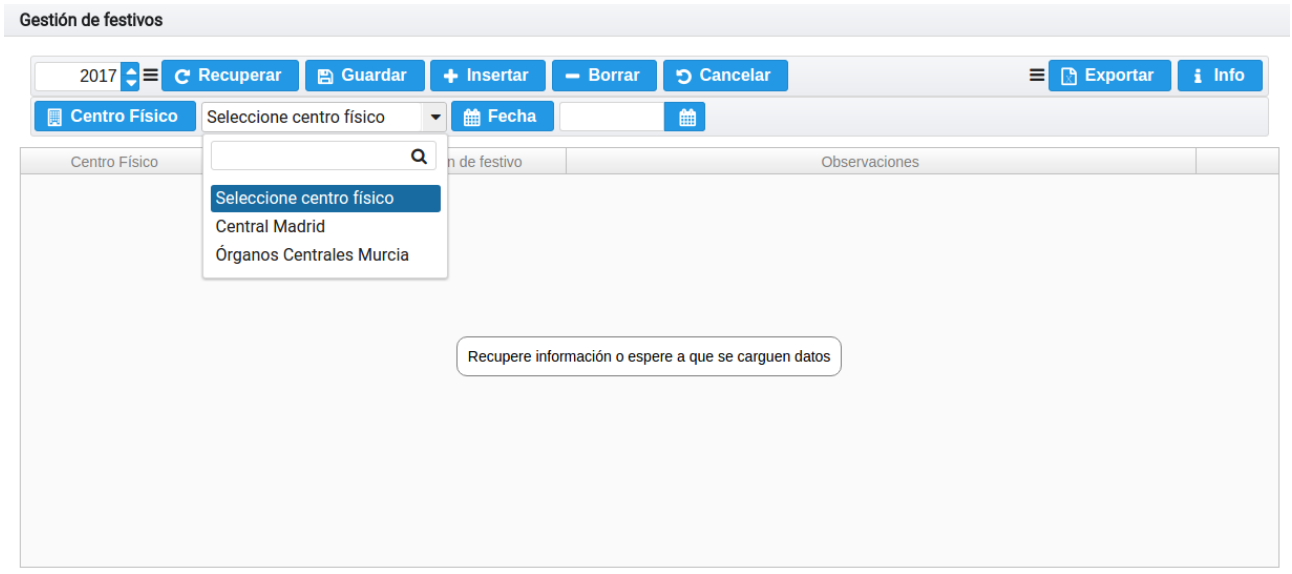


Figura 22. Mantenimiento de gestión de festivos. Se visualizan componentes *primeNG*, tales como botones y combos de selección.

En el siguiente capítulo se ve con mas detalle el uso de estos componentes.

Capítulo V – Desarrollo

Funcionamiento básico

En este apartado se presenta el funcionamiento básico de la aplicación atendiendo a los requisitos establecidos en el **capítulo III**. En primer lugar se deberá realizar una configuración básica antes de poder realizar una planificación de turnos completa. Los pasos a seguir para dicha configuración de la aplicación son:

- Definición de estructura organizativa: Definición unidades organizativas de puestos, equipos, servicios funcionales y centros físicos. Creación de relación de unidades organizativas atendiendo a una dependencia jerárquica de centro físico, servicio, equipo y puesto.

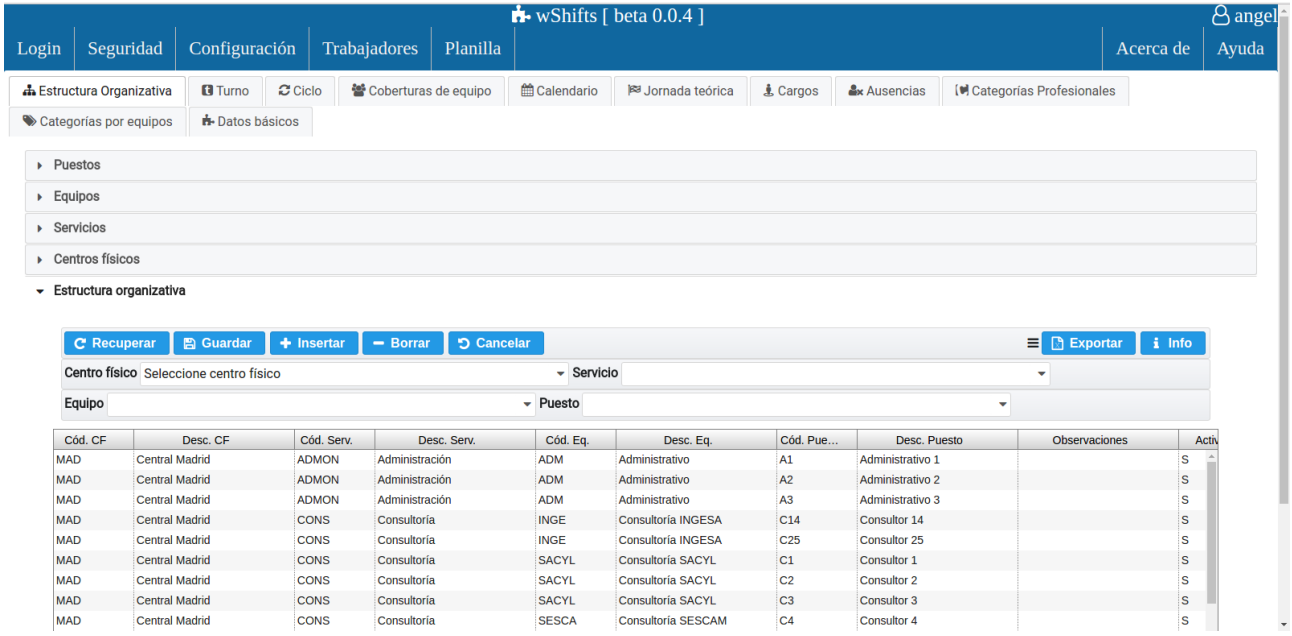


Figura 23. Mantenimiento de estructura organizativa. La estructura organizativa es una jerarquía arbórea de dependencia entre unidades organizativas, que en este mantenimiento se representa mediante las tuplas <centro físico, servicio, equipo, puesto>. Una tupla o relación es única, no pudiendo repetirse. Un centro físico se compone de uno o varios servicios. Cada servicio se compone de uno o varios equipos. Un equipo se compone de uno o varios puestos de trabajo.

- Creación de usuarios de aplicación, roles, recursos y sus relaciones. Un usuario podrá acceder al aplicativo a partir de la inserción de un nombre de usuario y contraseña. No puede haber dos nombres de usuario iguales. El sistema tiene un número limitado de intentos de acceso. Si después de ciertos intentos no se ha insertado correctamente el usuario y contraseña dicho usuario quedará bloqueado y no podrá acceder al sistema. Un rol es un perfil de acceso a la aplicación. Así un rol puede ser "Administrador", que tiene total control de la aplicación,

"Supervisor", para supervisar todas las planillas a las que tenga acceso, "RRHH" para gestión de datos de trabajadores, etc. No puede haber códigos ni descripciones de roles repetidos. Un usuario puede tener uno o varios roles. Un recurso es una parte del aplicativo que contiene una funcionalidad específica. Así un recurso puede ser un botón de una pantalla, un módulo de la aplicación o una opción de un proceso. Un recurso puede estar activo, y se podrá acceder a él, o no estarlo, no pudiendo acceder al mismo. Un rol puede tener asociados uno o varios recursos, donde un recurso es una parte de la aplicación a la cual se puede acceder, visualizar o ejecutar. Tal como se ha comentado la estructura organizativa es una jerarquía arbórea de dependencia entre unidades organizativas. Un usuario podrá acceder a dichas unidades organizativas siempre que estén ligadas al mismo. Las unidades organizativas a las cuales podrá tener acceso son centros físicos, servicios funcionales y equipos de trabajo. Por defecto si se tiene acceso a un equipo de trabajo se tiene acceso a todos sus puestos de trabajo.

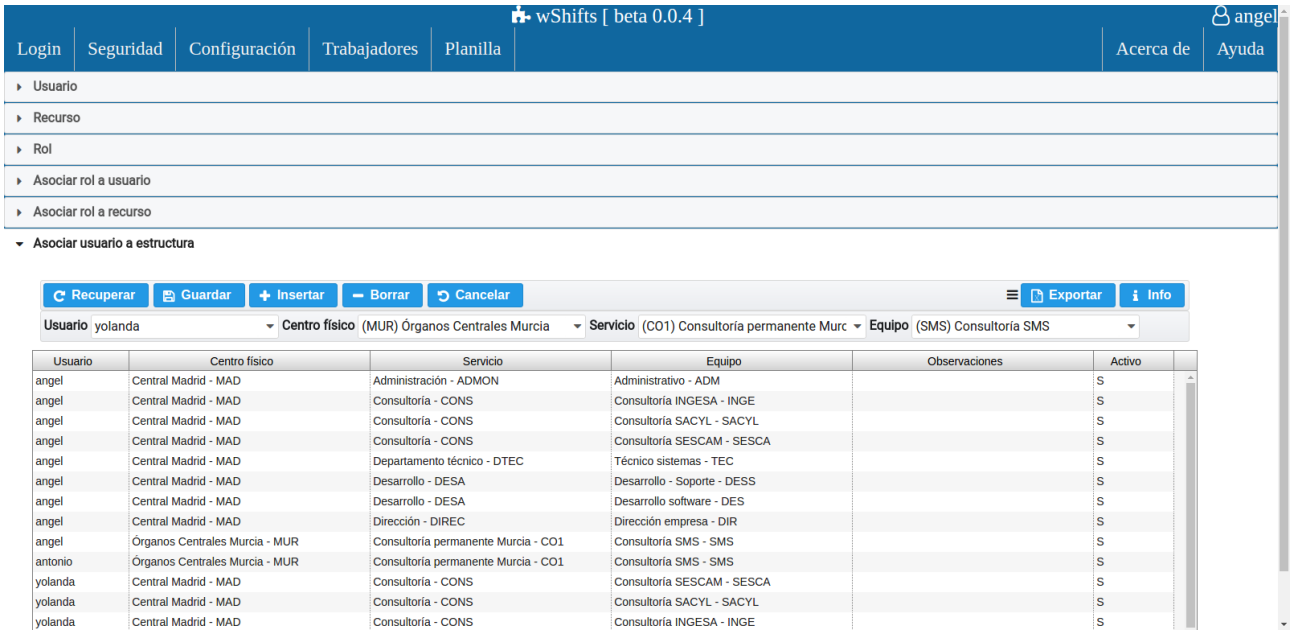


Figura 24. Módulo de Seguridad. Mantenimiento de asociación de usuarios a estructura organizativa.

- Configuración de turnos. Un turno se define como la realización de una actividad de trabajo entre un rango de horas continuo. Así un turno puede ser una jornada laboral de 7 horas que comienza a las 7:00 y finaliza a las 15:00. Este turno computará 7 horas de trabajo efectivo. Un turno puede comenzar o terminar en el día actual o en el día siguiente a la realización de la actividad. Por ejemplo, un turno de noche puede empezar a las 22:00 del día actual y terminar a las 8:00 del día siguiente. Además un turno puede no contar horas, es decir, es una actividad cuyo cómputo horario es 0 horas, definiendo días de libranza.

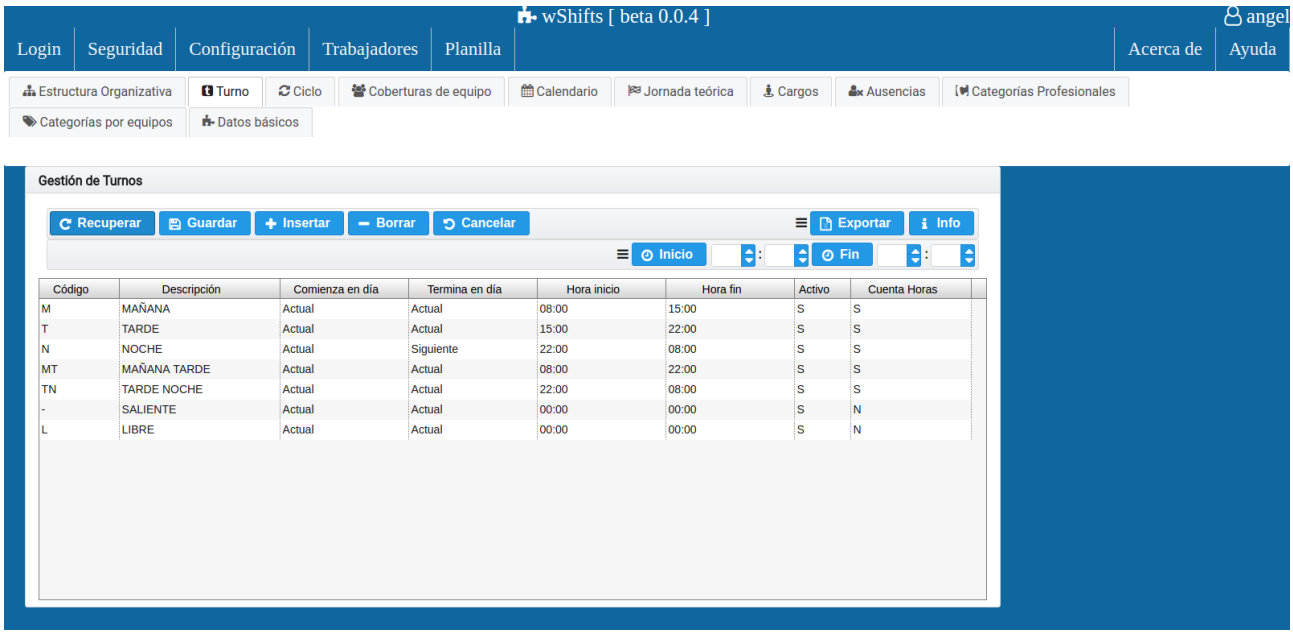


Figura 25. Módulo de Configuración. Mantenimiento de gestión de turnos.

- Configuración de ciclos. Un ciclo o patrón es un conjunto de turnos ordenados que se repiten en un calendario laboral. Así cuando se llegue al último turno del patrón el siguiente será el primer turno de dicho patrón. Por ejemplo un patrón típico de trabajo de 35 horas semanales será el compuesto por 5 turnos de Mañana, de lunes a viernes, y dos turnos que no cuentan horas, para el fin de semana, sábado y domingo: M M M M M L L, donde M es turno de mañana de 8:00 a 15:00 y L es turno de libranza, que no cuenta horas.

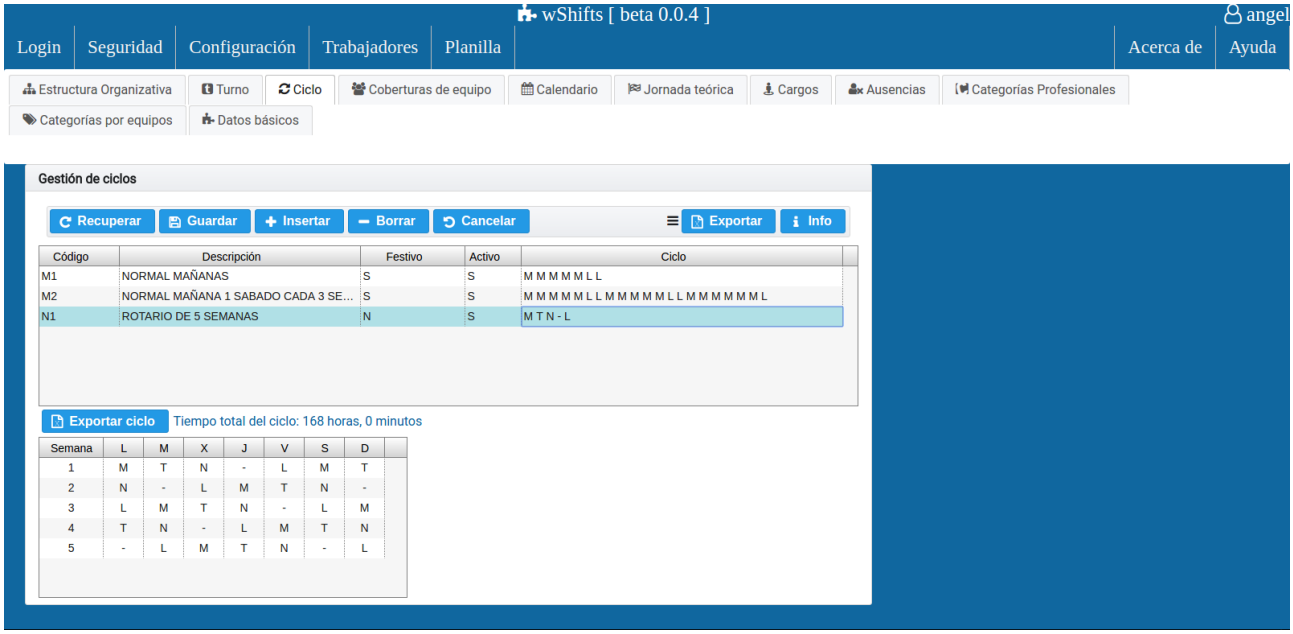
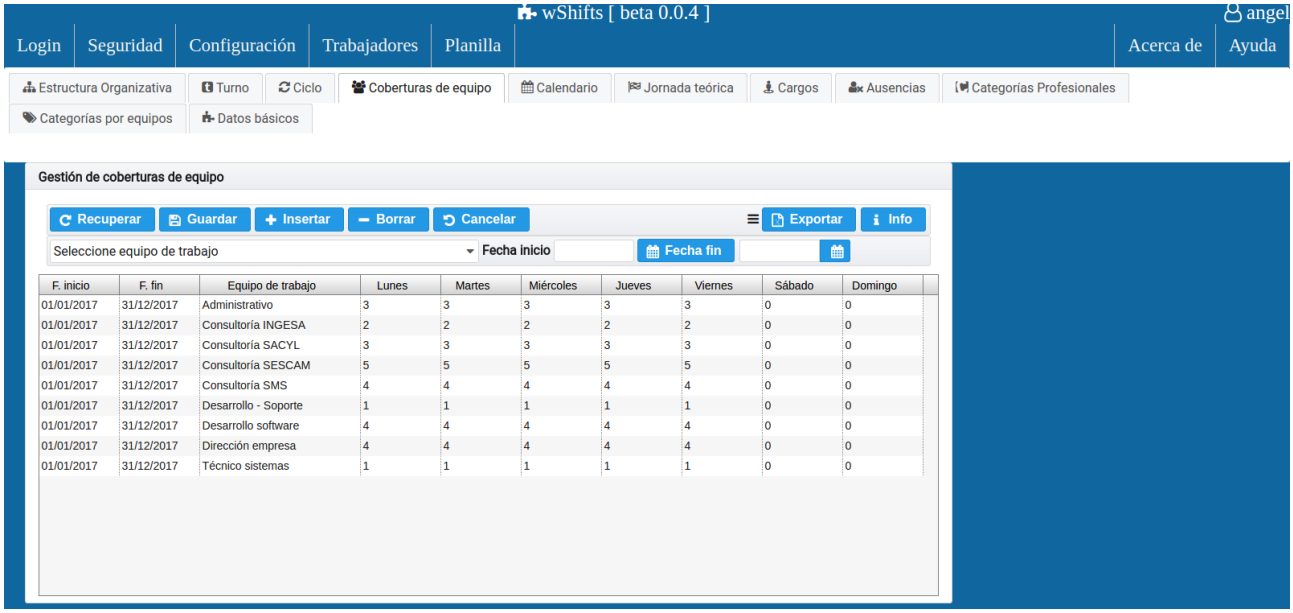


Figura 26. Módulo de Configuración. Mantenimiento de gestión de ciclos. Un ciclo se define como un conjunto ordenado de turnos. Los turnos tienen un orden, de izquierda a derecha. Haciendo 'click' sobre un patrón se puede ver su expansión semanal, así como el cómputo total de horas de dicho ciclo.

- Configuración de coberturas de equipo: Una cobertura de equipo es la cantidad de trabajadores que se necesitan por cada día de una semana, con el objetivo de que el servicio siempre esté cubierto con respecto a la necesidad creada. Se tiene que definir una cobertura por cada equipo, para una fecha de inicio. No puede haber 2 coberturas que empiecen en la misma fecha para un mismo equipo. Tampoco puede haber solapamiento de coberturas. Si se quiere cambiar una cobertura se tendrá que cerrar la actual y crear una nueva en una fecha posterior a la del cierre.



F. inicio	F. fin	Equipo de trabajo	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
01/01/2017	31/12/2017	Administrativo	3	3	3	3	3	0	0
01/01/2017	31/12/2017	Consultoría INGESA	2	2	2	2	2	0	0
01/01/2017	31/12/2017	Consultoría SACYL	3	3	3	3	3	0	0
01/01/2017	31/12/2017	Consultoría SESCAM	5	5	5	5	5	0	0
01/01/2017	31/12/2017	Consultoría SMS	4	4	4	4	4	0	0
01/01/2017	31/12/2017	Desarrollo - Soporte	1	1	1	1	1	0	0
01/01/2017	31/12/2017	Desarrollo software	4	4	4	4	4	0	0
01/01/2017	31/12/2017	Dirección empresa	4	4	4	4	4	0	0
01/01/2017	31/12/2017	Técnico sistemas	1	1	1	1	1	0	0

Figura 27. Módulo de Configuración. Mantenimiento de coberturas de equipo.

- Configuración de calendario laboral: Un calendario laboral incluye días que son festivos, esto es, días en los cuales los turnos que se realizan no cuentan horas (días de libranzas) para algunos colectivos de trabajadores. Así un festivo puede ser el 6 y 8 de diciembre, el 1 de Mayo, o más específicamente para la comunidad de Madrid el 15 de Mayo, para la ciudad de Murcia el 12 de Septiembre, etc. Por tanto cada emplazamiento físico tendrá unos festivos locales además de los nacionales que se podrán tener en cuenta en el gestor a la hora de planificar el calendario laboral. Por tanto los calendarios festivos se configuran por centro físico y por año natural.

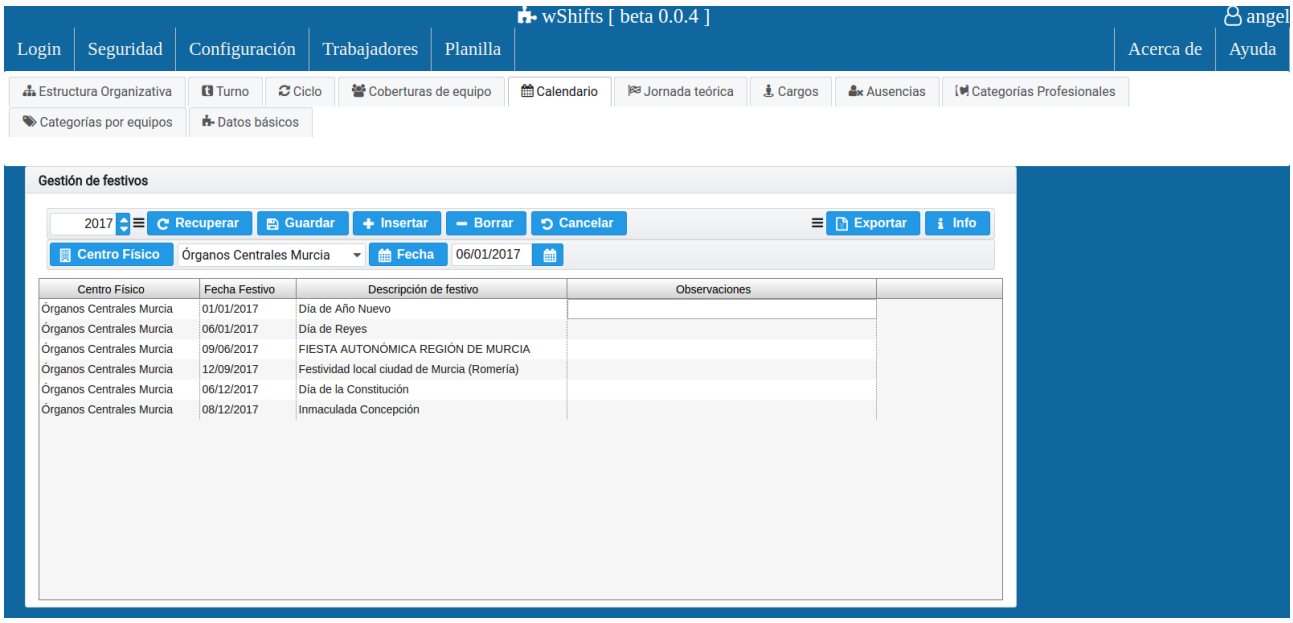


Figura 28. Módulo de Configuración. Mantenimiento de calendario de festivos.

- Configuración de jornada teórica: La jornada teórica es la cantidad de horas que tiene que realizar un trabajador en un año natural. Esta jornada anual es necesaria para el control del balance horario del trabajador, que es la diferencia entre las horas trabajadas y el total de jornada anual. La jornada anual o teórica se define por centro físico y por año. No puede haber más de una jornada teórica para un mismo año y centro físico.

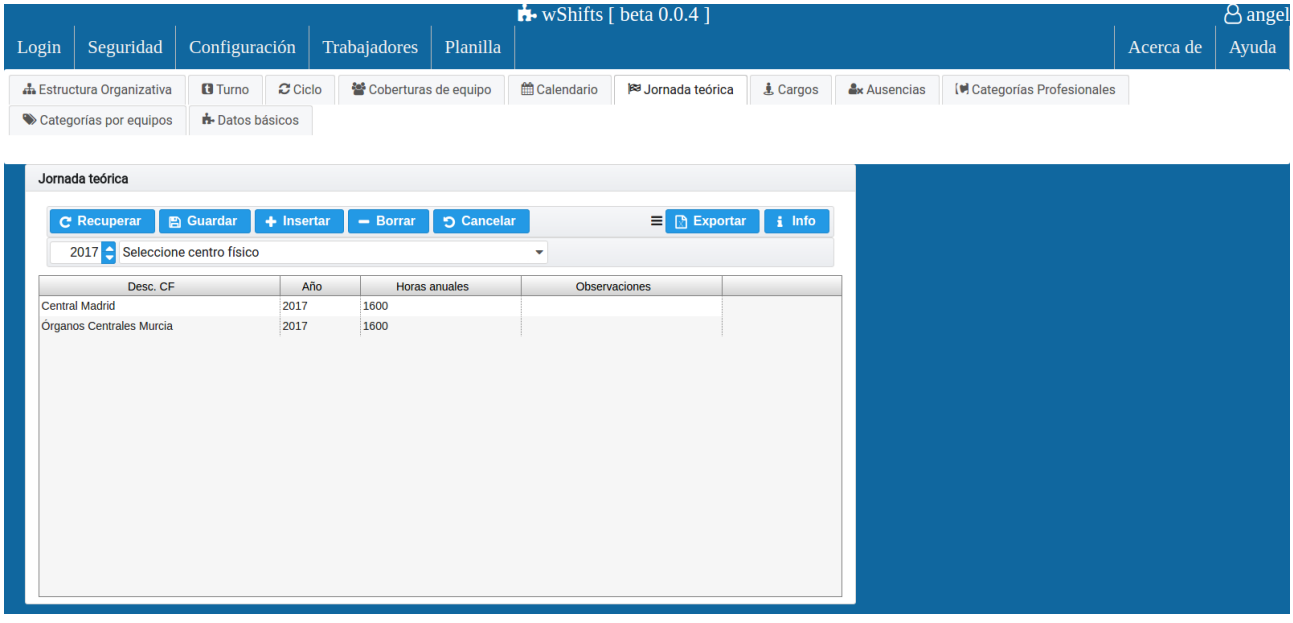


Figura 29. Módulo de Configuración. Mantenimiento de jornada teórica.

- Configuración de módulo de recursos humanos: Los trabajadores que serán planificados para la gestión de sus turnos deberán tener un contrato en vigor. La aplicación gestiona contratos de trabajadores. Así

un contrato de trabajo está definido por el cargo (esto es, la definición del puesto de responsabilidad) que desempeñará el trabajador, por ejemplo "Jefe de almacén", "Supervisor de área", "Repartidor", "Cuidador", etc., la fecha de inicio y fin de contrato y la categoría profesional que se define como la capacidad del puesto de trabajo (definición de familia profesional). Así categorías profesionales son "Ingeniero Informático", "Facultativo", "Maestro", "Logopeda", "Auxiliar Administrativo", etc. Además un trabajador puede ausentarse, por lo que se tienen que definir los tipos de ausencia a los que puede optar como por ejemplo Vacaciones", "Asuntos Particulares", "Compensaciones Horarias", "Formación", "Incapacidad Temporal", etc. Los cargos, categorías profesionales y tipos de ausencia son configurables.

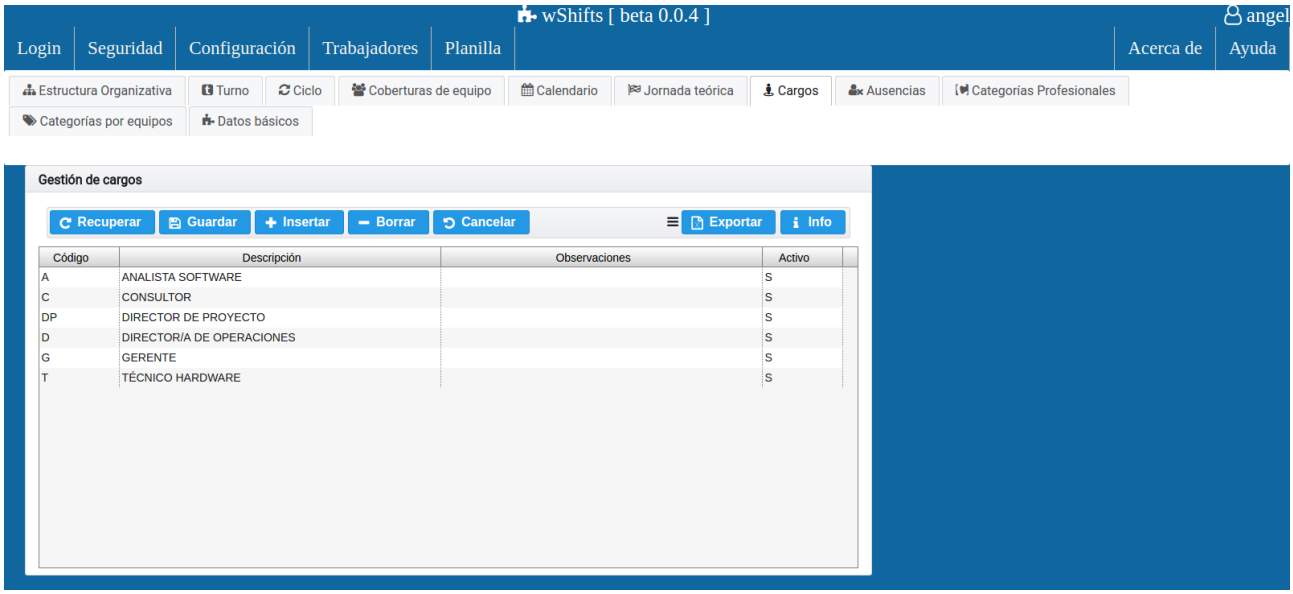


Figura 30. Módulo de Configuración. Mantenimiento de cargos.

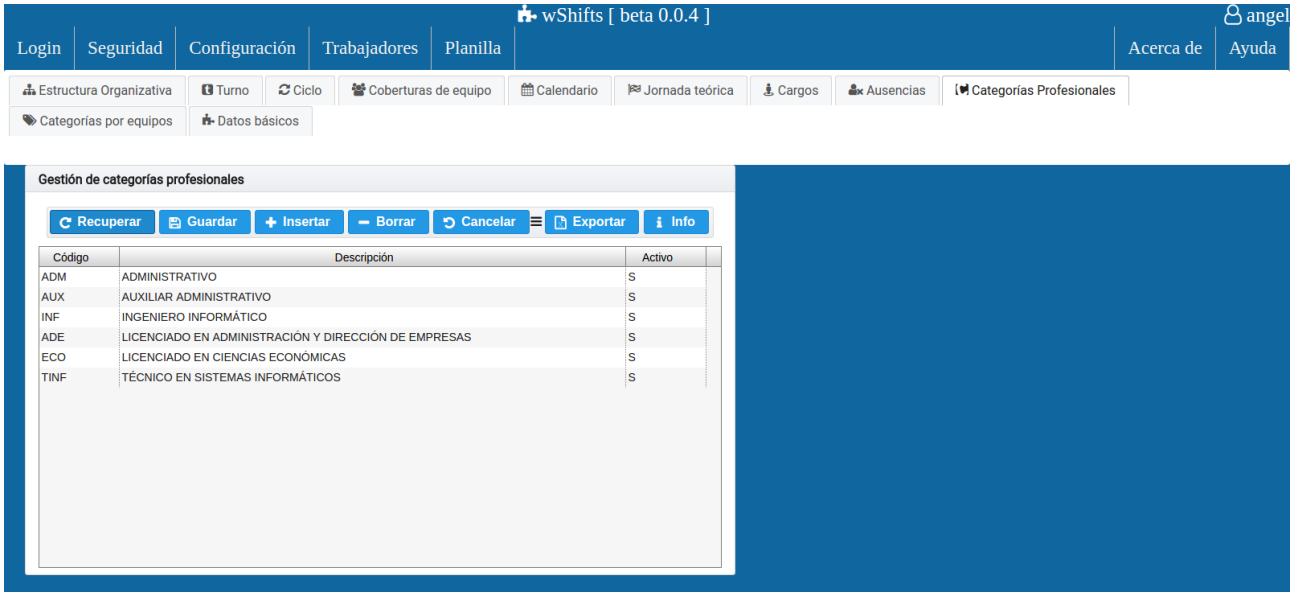


Figura 31. Módulo de Configuración. Mantenimiento de categorías profesionales.

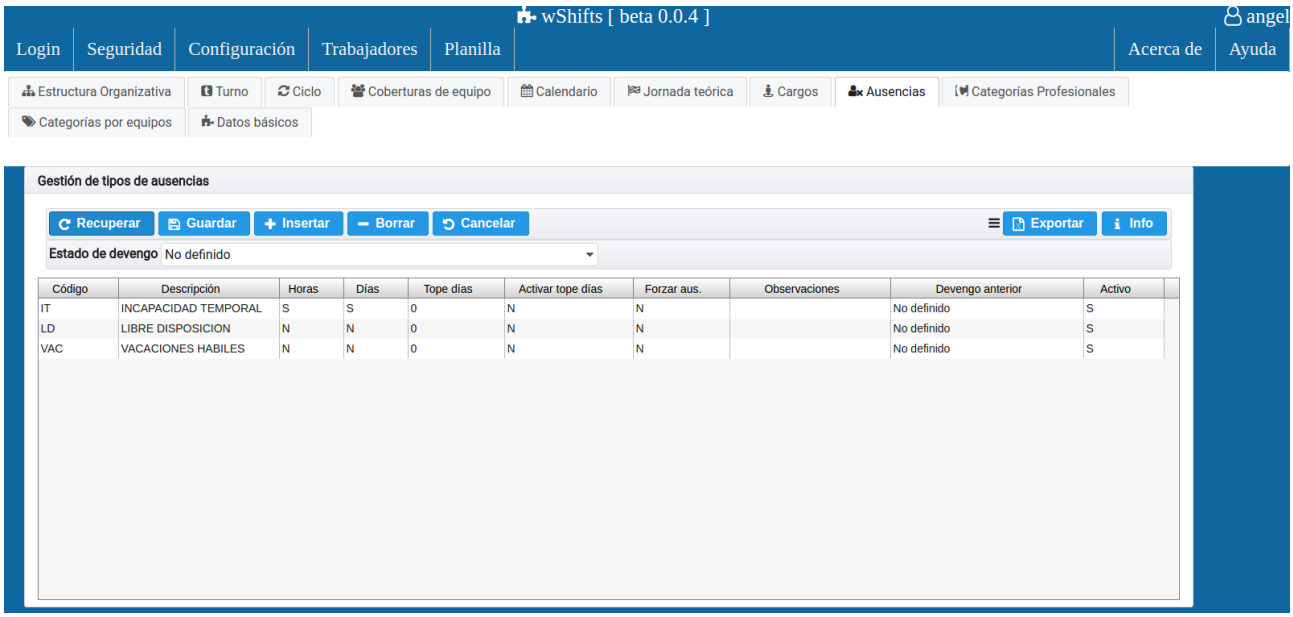


Figura 32. Módulo de Configuración. Mantenimiento de tipos de ausencia.

- Configuración de categorías profesionales asociadas a equipos de trabajo: Un equipo de trabajo está compuesto por un conjunto de trabajadores con una misma categoría profesional, ya que van a desempeñar funciones comunes. Sin embargo para un equipo de trabajo puede haber trabajadores con categorías profesionales diferentes pero que realizan el mismo trabajo. Así por ejemplo en un departamento de desarrollo de software puede haber trabajadores que son ingenieros informáticos, matemáticos, etc, que realizan la misma tarea de programación. Sin embargo en un departamento de desarrollo no debe haber, en principio, trabajadores con categorías profesionales que no concuerden con la finalidad del equipo. En el ejemplo anterior no debe de haber trabajadores con categorías profesionales de enfermero o auxiliar administrativo, por ejemplo.

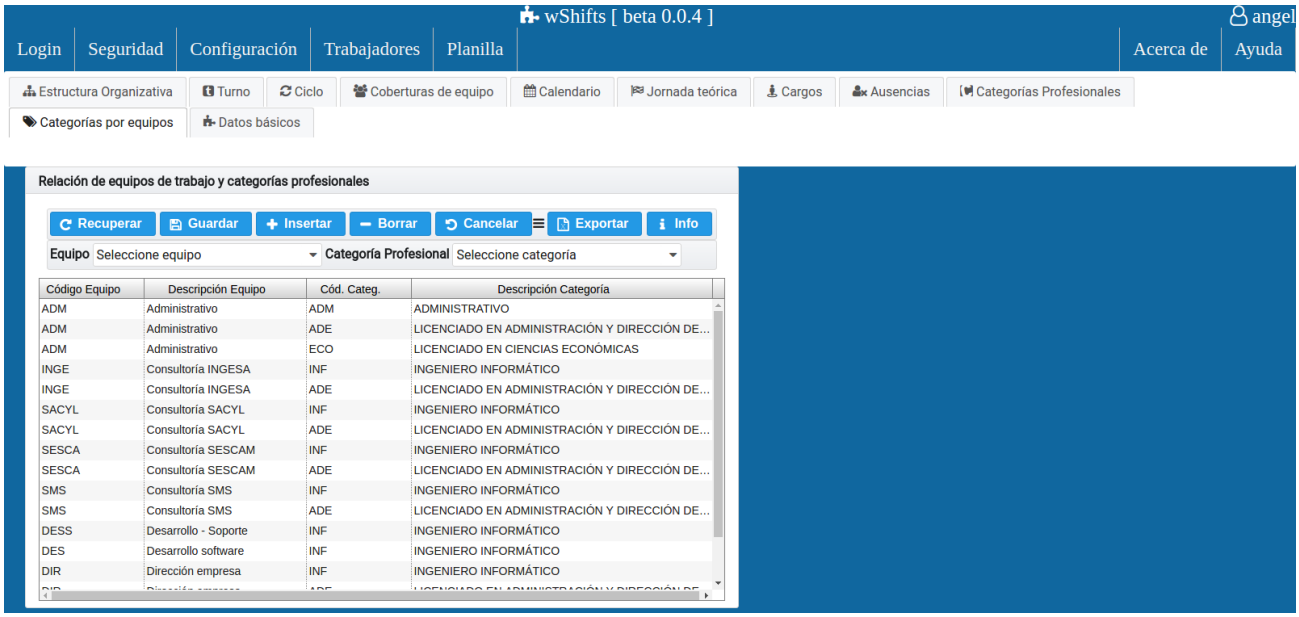


Figura 33. Módulo de Configuración. Mantenimiento de relación entre equipos y categorías profesionales de trabajadores. A un equipo de trabajo se le podrá asignar únicamente trabajadores que tengan una determinada categoría profesional. La relación de categorías profesionales que son permitidas para cada uno de los equipos de trabajo se definen en este mantenimiento.

Una vez finalizada la configuración básica se dispone a dar de alta a los trabajadores, mediante la inserción de sus datos personales y la creación de contratos asociados a cada trabajador.

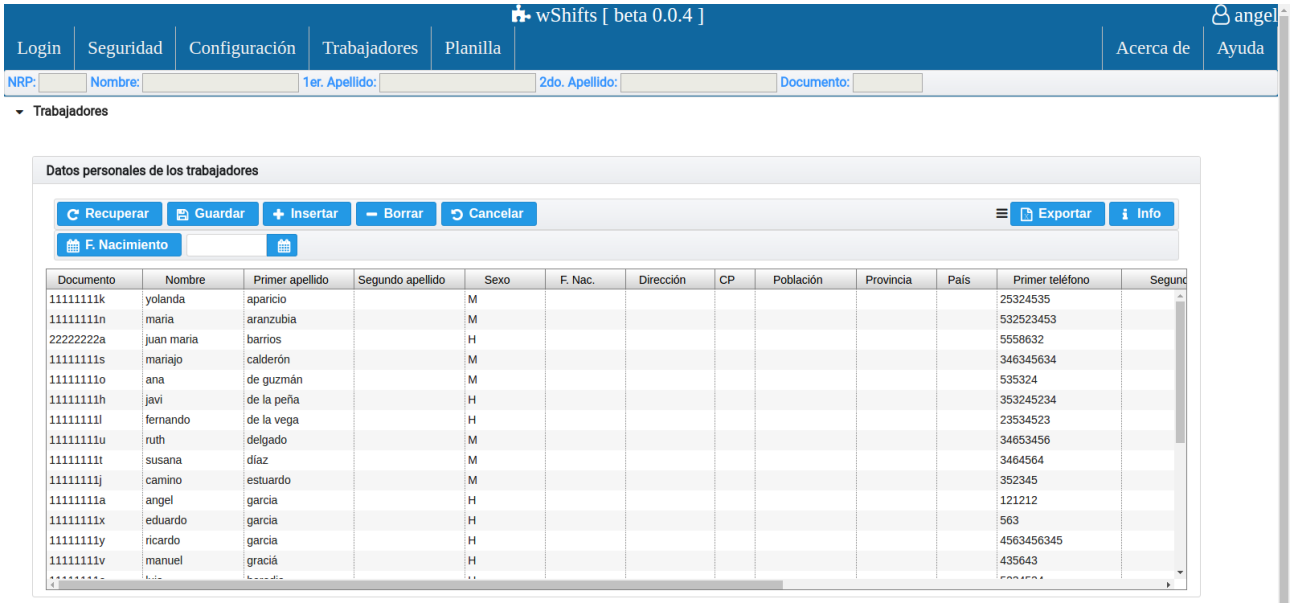


Figura 34. Módulo de Trabajadores.

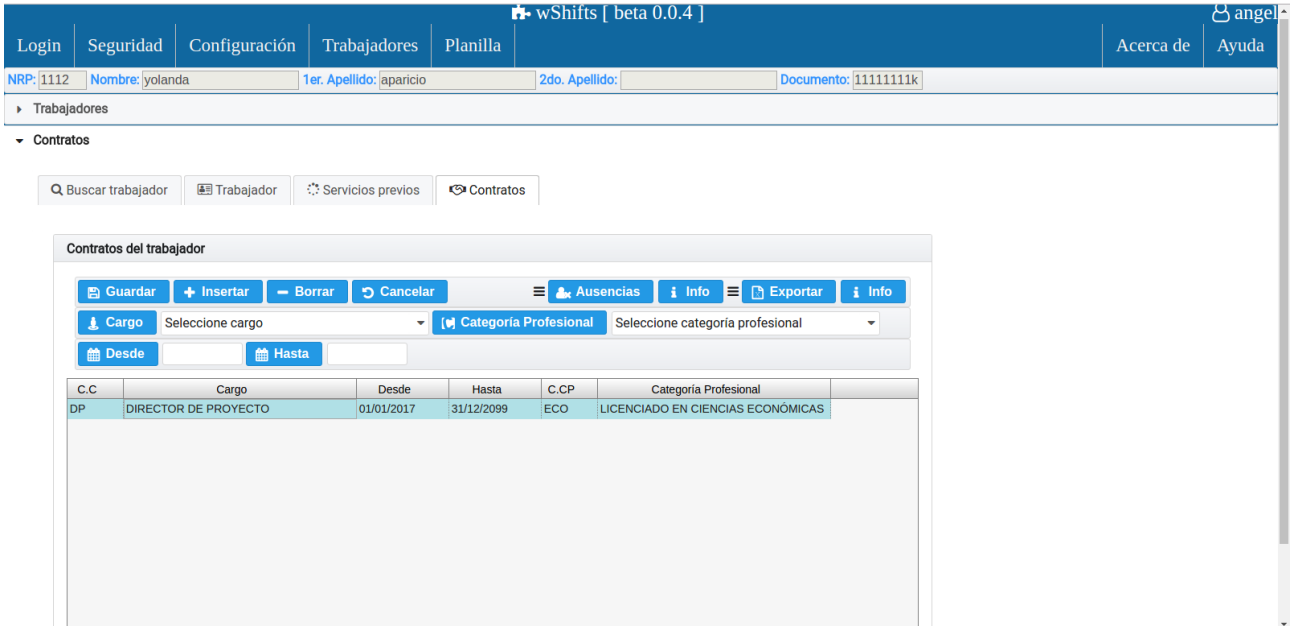


Figura 35. Módulo de Contratos. Se ha seleccionado a un trabajador y se le ha creado un contrato, con cargo Director de Proyecto, categoría profesional de Licenciado en Ciencias Económicas e indefinido desde enero de 2017.

A continuación se planifican los puestos de trabajo. Un puesto de trabajo puede tener asociado uno o varios ciclos semanales de turnos, entendiendo un ciclo como el conjunto de turnos que se repiten en un conjunto de semanas. Un puesto puede tener varios ciclos siempre que no empiecen en el mismo día. Un ciclo tiene que asociarse a un puesto de trabajo entre un rango de fechas, no pudiendo solaparse rangos de fechas de varios ciclos. Así un puesto tendrá asociado varios ciclos, cada uno con un rango de fechas diferente, de modo que se planificarán cada uno de los ciclos (expansión del ciclo en el calendario laboral) automáticamente una vez se hayan asociado.

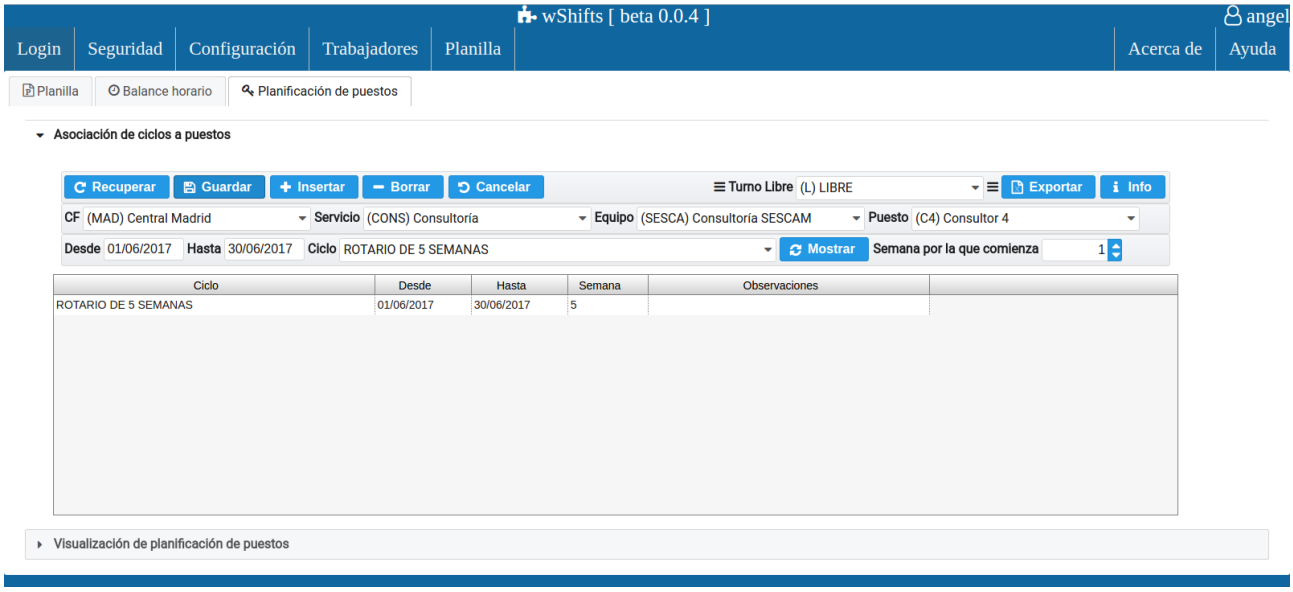


Figura 36. Módulo de Planilla – Planificación de puestos. Se asocia el ciclo Rotatorio de 5 Semanas al puesto Consultor 4 del equipo Consultoría SESCAM (servicio Consultoría perteneciente a la central de Madrid), para el mes de Junio de 2017. Además el ciclo empieza por la semana 1.

Puesto que ya se tiene al trabajador con contrato y configurados todos los servicios y equipos de trabajo se está en disposición de asignar una tarea al trabajador, es decir, incluir al trabajador en un equipo de trabajo, en un puesto específico que ya estará asociado a un ciclo y planificado en el calendario laboral.

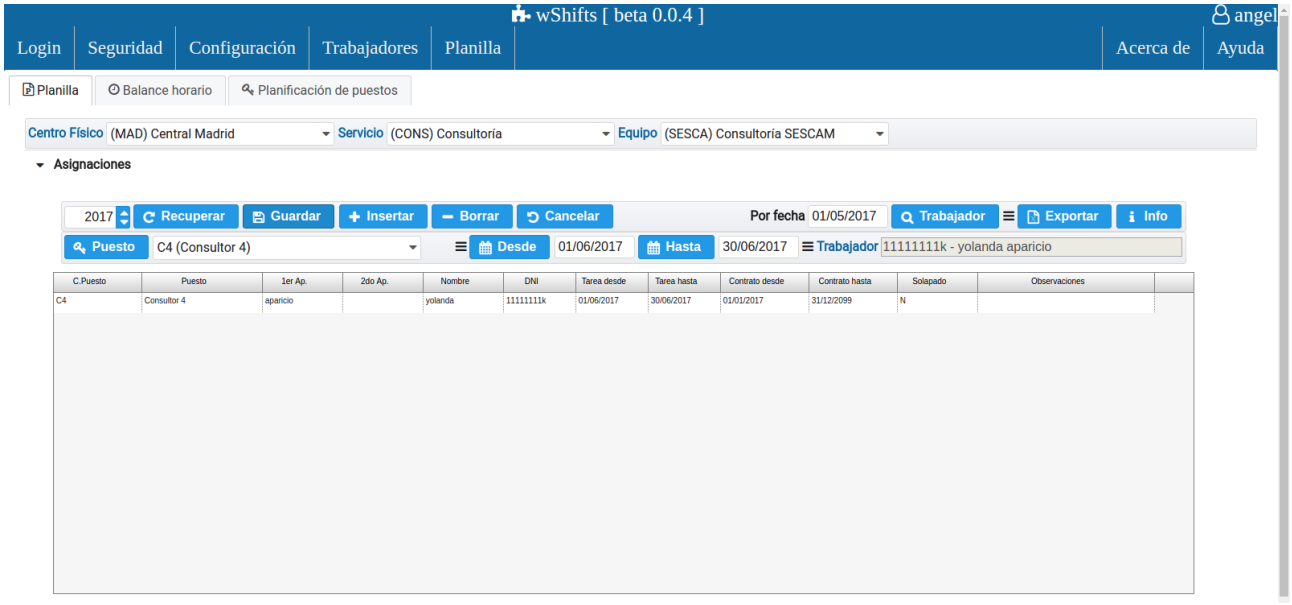


Figura 37. Módulo de Planilla – Planilla – Asignaciones. Se asocia al trabajador con el puesto C4 para el mes de Junio del año 2017.

Y finalmente se puede ver la planilla (planificación diaria) con el trabajador, mostrando el ciclo expandido en el calendario laboral y sus coberturas de servicio para cada día.

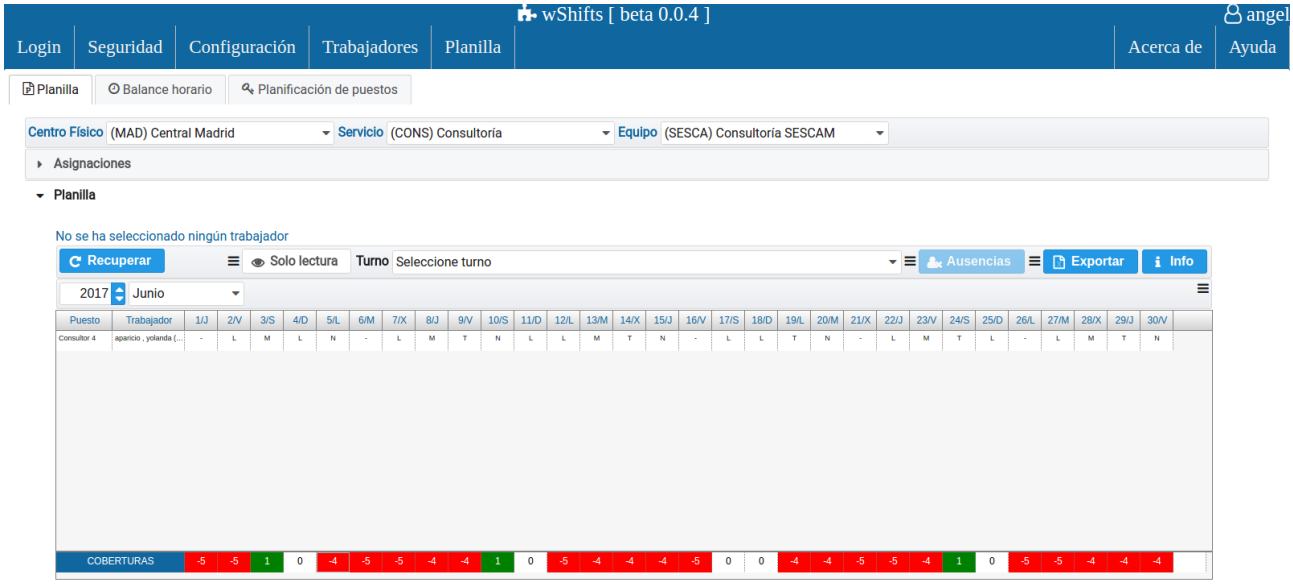


Figura 38. Módulo de Planilla – Planilla – Planilla. El trabajador está planificado en el puesto C4, con un ciclo rotatorio de 5 semanas. En las coberturas de servicio aparecen en rojo el número de trabajadores que necesita el equipo para cubrir las necesidades de servicio. En verde aparecen el número de trabajadores que exceden para la cobertura para ese día. Si la celda de cobertura está en blanco significa que la cobertura está satisfecha y el número de trabajadores para ese día coincide con la necesidad definida.

En la Planilla se pueden hacer cambios de turno e inserción de ausencias. En definitiva todo este trabajo no podría haberse realizado sin una filosofía de desarrollo, que para a verse a continuación.

Filosofía de desarrollo

Este proyecto ha intentado en todo lo posible seguir una **Filosofía Python** (que es bastante análoga a la filosofía de **Unix**). El código que sigue los principios de Python de legibilidad y transparencia se dice que es **pythonico**. Contrariamente, el código opaco u ofuscado es bautizado como **no pythonico**. Estos principios fueron famosamente descritos por el desarrollador de Python *Tim Peters* en **El Zen de Python**:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Ralo es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechazar la tentación de adivinar.
- Debería haber una - y preferiblemente sólo una - manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres son una gran idea ¡Hagamos más de esas cosas!.

Estos principios están implementados en el intérprete de Python, pudiéndose ver mediante la instrucción *import this*.

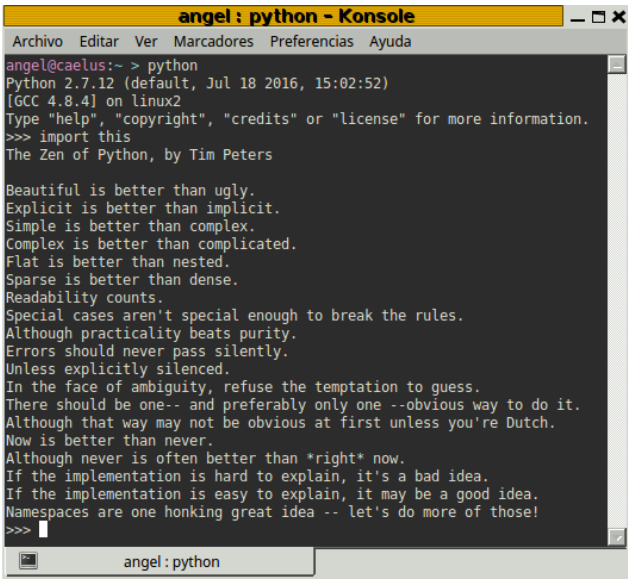


Figura 39. Intérprete Python. Ejecución de *import this*.

Metodología de desarrollo

Definición de metodología ágil

Una metodología de tipo Ágil es una metodología de desarrollo de software que proporciona un proceso de desarrollo rápido y dotado de pocos mecanismos de regulación, con objeto de hacerla más ligera. La filosofía Ágil para el desarrollo del software se especifica en un manifiesto para el desarrollo ágil del software, en el que se priman los aspectos prácticos y orientados a la consecución rápida del objetivo, sobre los aspectos formales y sistemáticos, propios de metodologías “pesadas” (en el sentido de que contienen densos y extensos componentes normativos para guiar la práctica). Las metodologías de desarrollo del software ágiles, se presentan y aparecen históricamente, en contraste con las metodologías de desarrollo de software comúnmente conocidas como “tradicionales”. Por metodologías tradicionales se entiende las que están guiadas por un desarrollo en cascada en el que cada fase no empieza hasta que la anterior haya terminado, y que presuponen que todos los requisitos están disponibles al principio.

Metodología aplicada

wShifts se ha desarrollo a partir de una metodología de tipo *Ágil*, donde prima el resultado, la rapidez del desarrollo y entrega de versiones del producto. Para ello se ha utilizado una variante de Programación Extrema (*Extreme Programming*), donde la construcción del software se ha ido realizando por módulos, inicialmente independientes cuyo objetivo principal es que tuvieran la funcionalidad deseada en el menor tiempo posible. Una vez que el módulo se ha realizado y se ha comprobado que hace lo que se espera de él, es cuando se comenta internamente. De esta manera se han obtenido pequeños programas que tenían una funcionalidad concreta, cuya unión ha dado como resultado el producto final, esto es, una aplicación completa y funcional. Evidentemente este tipo de metodología no podría haberse puesto en práctica si el lenguaje en el que se ha programado la parte servidor no tuviera la filosofía descrita en el apartado anterior, en su idiosincrasia e implementación. Así, Python, con una sintaxis clara y concisa, ha ayudado a la tarea de desarrollo ágil de software.

Herramientas

Parte servidor

El desarrollo de la parte servidor se ha escrito íntegramente en Python, por lo que se ha usado un único IDE (entorno de desarrollo integrado), **Wing Python IDE**, versión 5, de la empresa **Wingware** (Wingware 2016). Contiene un editor de código con ayuda de código automático, intérprete, herramientas auxiliares de desarrollo (creación de documentación, tests, empaquetado de software), depurador, etc.

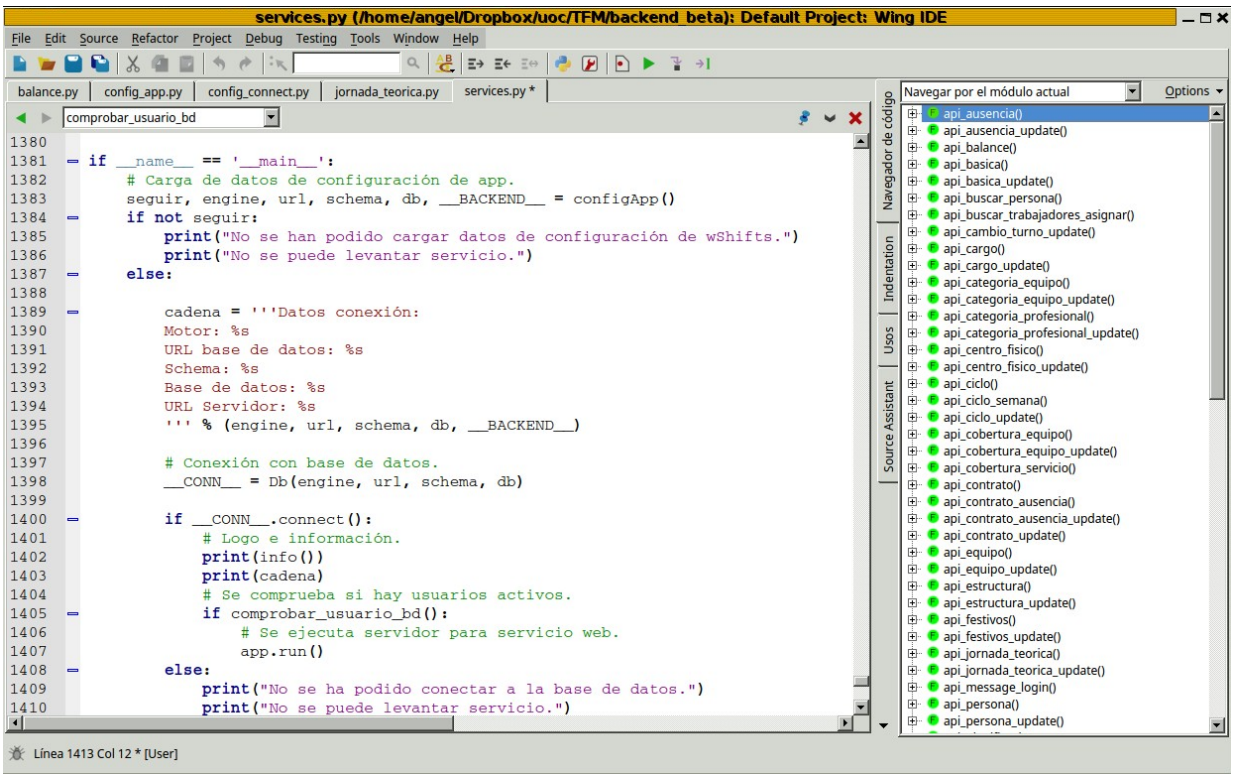


Figura 40. Editor Wing Python IDE, versión 5. Edición de módulo *services.py,* visualizando el punto de entrada de la parte servidor de servicios web de *wShifts*. En la parte derecha se puede observar todos las funciones de acceso de servicios web implementadas.

El diseño entidad – relación de la base de datos se ha creado mediante la aplicación **DB DesignerFork 2009** (Becker 2009), que puede generar código SQL para creación de tablas y relaciones.

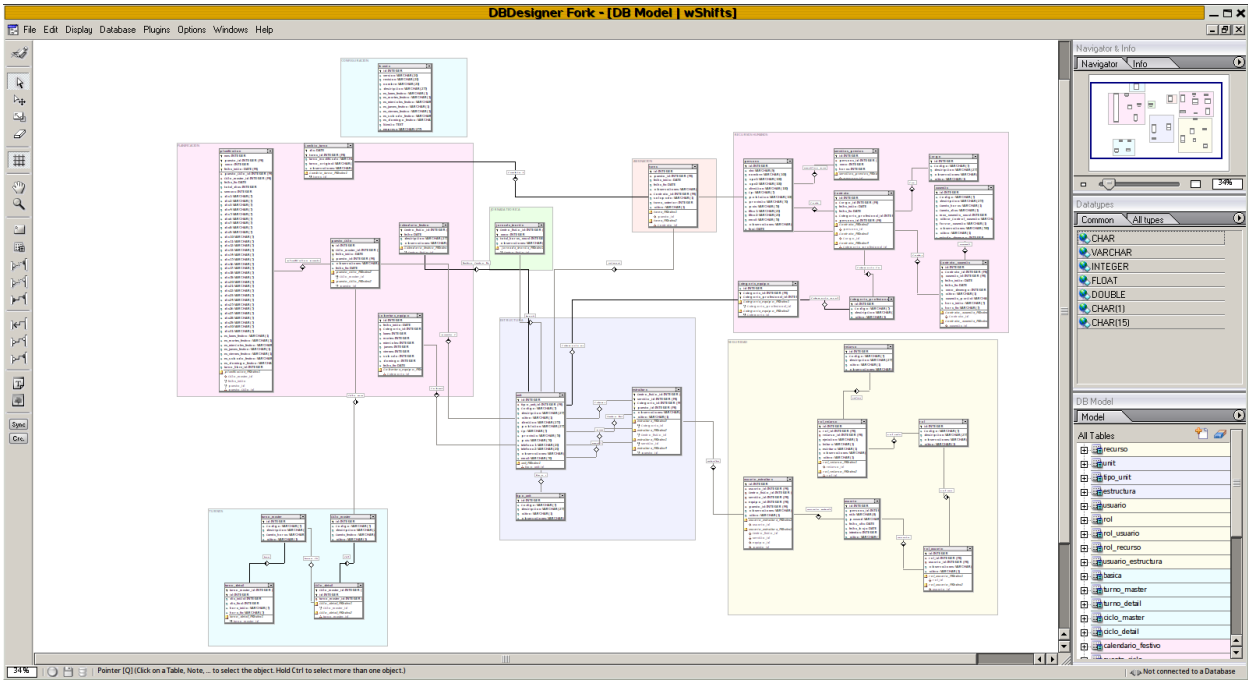


Figura 41. DB Designer Fork 2009. Edición del diagrama entidad – relación del proyecto wShifts.

Tal como se ha comentado en capítulos anteriores este proyecto utiliza, en principio, la base de datos monopuesto **SQLite**. Para su gestión se ha usado el software **DB Browser for SQLite**, versión 3.7.99 (Piacentini *et al*. 2016).

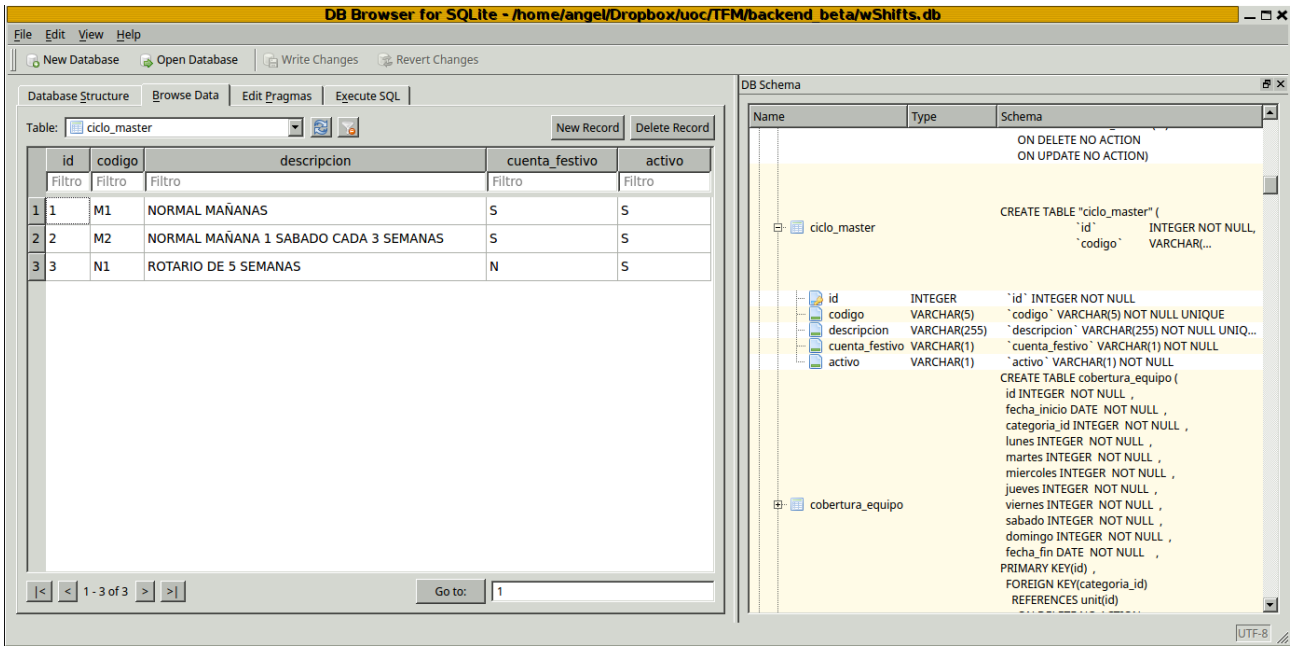


Figura 42. DB Browser for SQLite, versión 4.7.99. Edición de datos de tabla ciclo_master. A la derecha se puede observar la descripción de los campos de dicha la tabla. Este software gestiona tablas, índices, vistas y disparadores de una base de datos.

Puesto que se ha desarrollado un servidor de servicios web era necesario comprobar que dichas llamadas eran válidas para verificar que lo que se enviaba y devolvía eran datos coherentes y esperados. Para tal fin se ha usado el software **Curl** versión 7.4 (Stenberg *et al*. 2016), que es una aplicación que implementa una librería de funciones para conectar con servidores y trabajar con ellos. El trabajo se realiza con formato URL, sirviendo para realizar acciones sobre archivos que hay en URLs de Internet, soportando los protocolos más comunes, como http, ftp, https, etc. De esta manera se pueden simular peticiones de información al servidor simulando llamadas desde el cliente. Esto es las peticiones de información del cliente son simuladas por Curl.

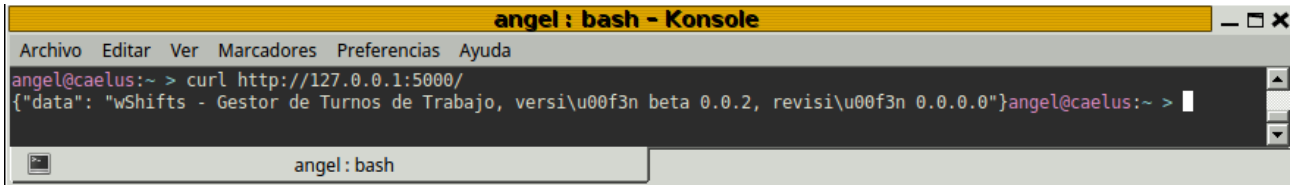


Figura 43. Curl, versión 7.4. Ejecución de instrucción *curl http://127.0.0.1:5000/*. Simula petición web de cliente al servidor (que se está ejecutando en la máquina local, escuchando por el puerto 5000) para que devuelva nombre, versión y revisión de aplicación.

Parte cliente

La parte del cliente se ha codificado en TypeScript, HTML5 y CSS3. Al principio del proyecto se utilizó para la escritura del código *Visual Studio Code*, de *Microsoft*. Sin embargo este editor tenía ciertos problemas de estabilidad (en noviembre de 2016). Por tanto se optó por usar el editor **Atom**, versión 1.10, instalando además algunos complementos para coloración de código HTML y CSS y código TypeScript.

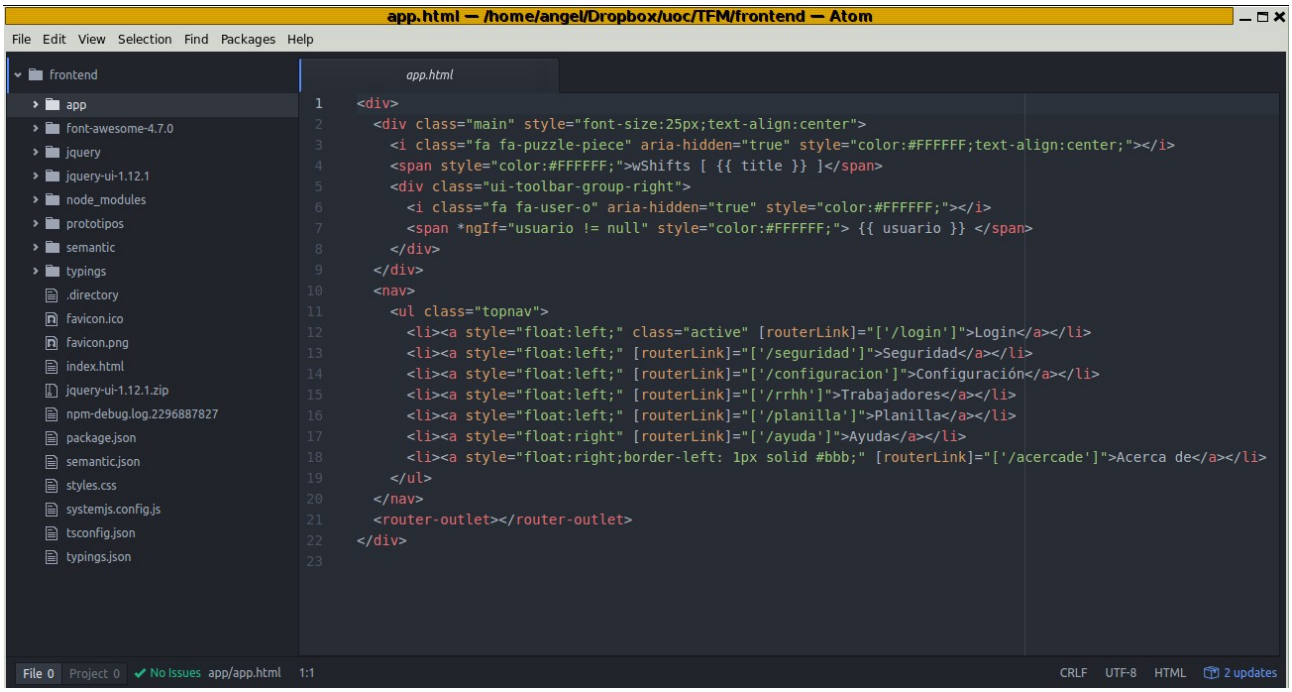


Figura 44. Atom, versión 1.10. Edición de fichero app.html. Se puede observar código HMTL, CSS y etiquetas propias de Angular 2, como <router-outlet></router-outlet> o [routerLink].

La depuración en la parte del cliente ha sido esencial, y si bien la mayoría de los navegadores tienen depuradores excelentes se ha optado por el uso del navegador **Chrome**, de Google, ya que es esta empresa la que ha creado el marco de trabajo Angular 2. Dicho navegador contiene un sistema de depuración extraordinario, con posibilidad de inserción de puntos de ruptura, visualización de carga de trabajo en memoria, carga de red, introspección de código, visualización de código JavaScript y TypeScript, etc.

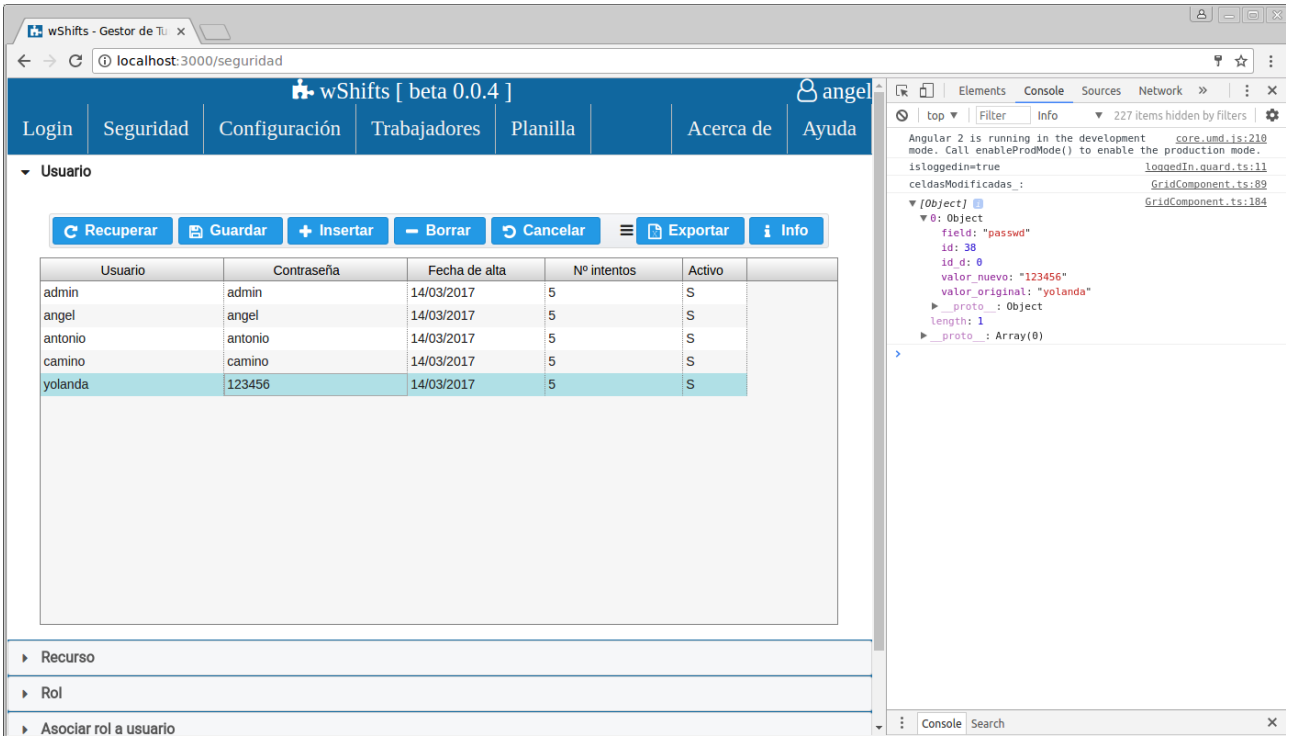


Figura 45. Navegador Chrome. Se muestra consola de desarrollador a la derecha de la pantalla, donde se puede visualizar un objeto. Se ha modificado la contraseña del usuario "yolanda", y se puede ver dentro del objeto los datos persistentes y en qué componente está dicho objeto (GridComponent.ts). La aplicación está en modo depuración, por lo que se pueden ver estos tipos de datos.

Memoria

La memoria de este proyecto se ha realizado con **Writer** de **LibreOffice**, versión 4.4.3.2. Para la creación de diagramas la aplicación **yEd Graph Editor** versión 3.16.2.1, de la empresa yWorks (yWorks 2016). La edición de ficheros de bases datos, ficheros de carga de pruebas, bitácora de desarrollo, etc. se ha optado por el editor de texto **NotePad++** (Do Ho 2016). La creación de la documentación de código (interfaz de aplicación o API) se ha generado con **epydoc** (Loper 2008). Dicho documento puede encontrarse en el **Anexo IV**. El diagrama de Gantt se ha diseñado mediante la aplicación **Serena OpenProj**, versión 1.4 (Openproj 2013).

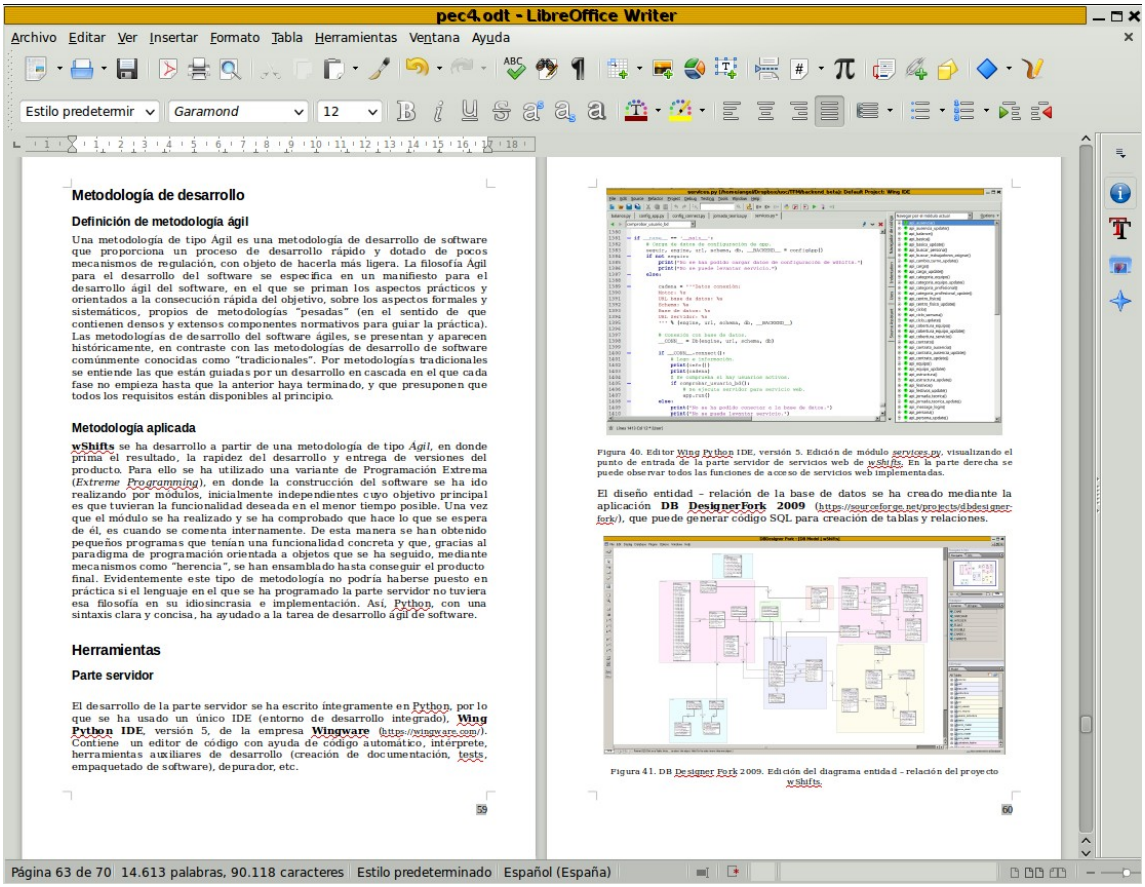


Figura 46. Aplicación Writer de LibreOffice. Editando la sección de metodologías de desarrollo del capítulo V del proyecto wShifts.

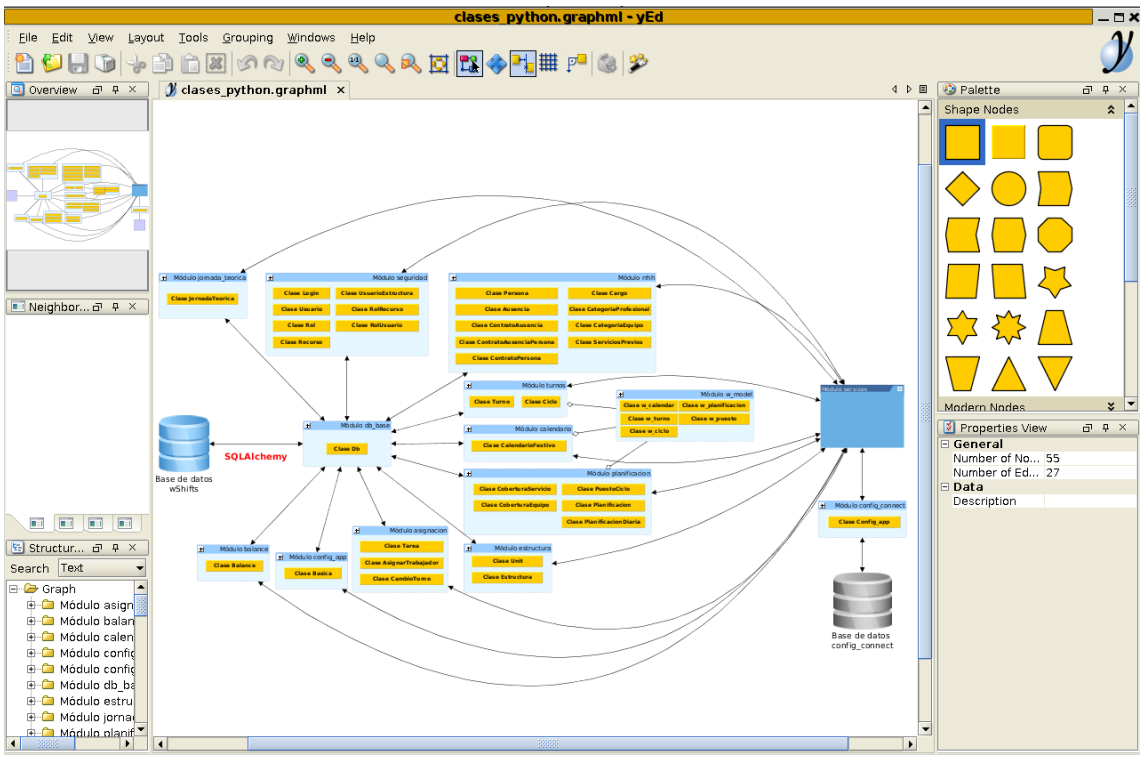


Figura 47. Software yEd. Edición de diagrama de clases Python de la parte servidor.

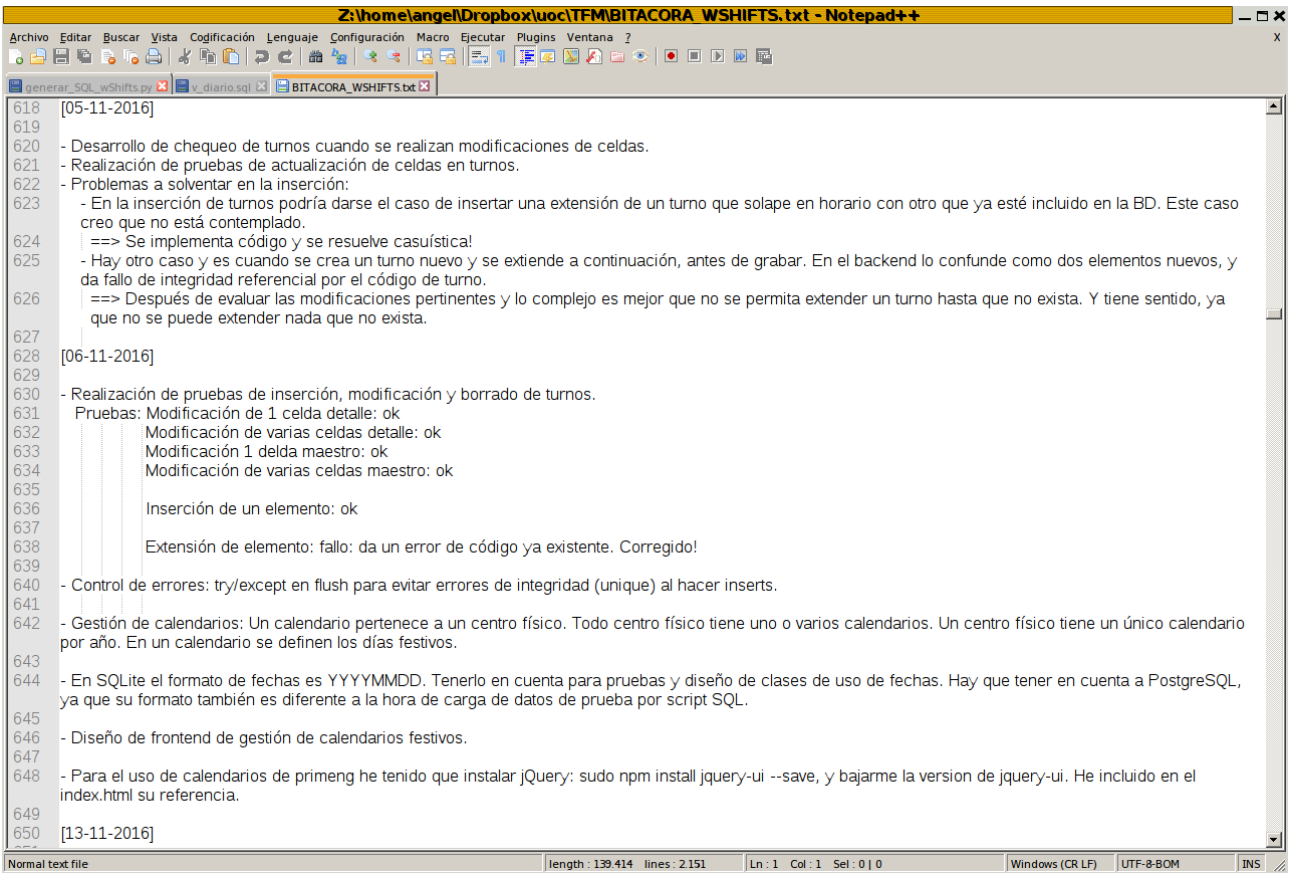


Figura 48. NotePad++. Edición de bitácora de desarrollo.

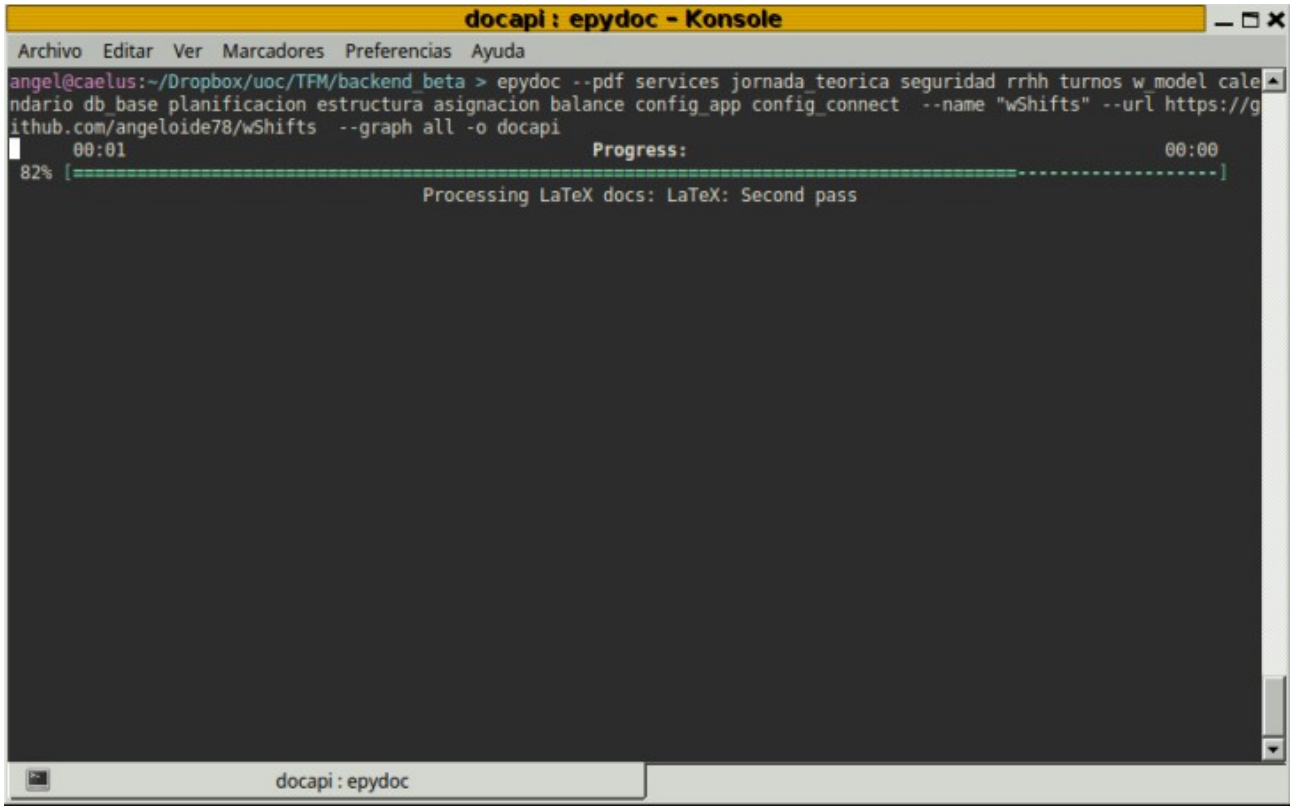


Figura 46. Aplicación epydoc. Generación de documentación API del proyecto wShifts.

Evolución del desarrollo

El desarrollo de este proyecto se basa en dos partes diferenciadas, la parte servidor y la parte cliente, la cual ha tenido mayor complejidad ya que la tecnología utilizada, Angular 2, está recién salida al mercado y por tanto en continua evolución. Los marcos de trabajo como *PrimeNG* o componentes como *ag-grid* están continuamente evolucionando, al igual que la manera de generar código Angular 2 automático. Esto, junto con los errores propios de un nuevo software ha hecho generar cierto retraso y formación continua conforme el proyecto avanzaba. Por tanto en un primer lugar se ha desarrollado la parte servidor, a saber, diseño de base de datos, reglas de negocio y servidor de peticiones web. Al utilizar una metodología ágil la evolución del desarrollo no ha sido ejecutada por etapas estrictas, sino por la creación de módulos funcionales creados en el menor tiempo posible para poder ser utilizados. Mientras se creaban los módulos se iban creando guiones de comprobación de datos y pruebas de integración, para que todo fuera coherente al ensamblar unas partes con otras de la aplicación. Conforme se escribían módulos (librerías) de la aplicación se iba registrando en un fichero de texto plano, llamado **bitácora de desarrollo**, todos los avances, problemas y contratiempos, ideas sobre la marcha que iban surgiendo, etc, para tener un registro diario de lo que se iba haciendo. Si bien se ha hecho una planificación temporal mediante diagramas de Gantt la bitácora refleja el día a día del desarrollo del proyecto. El proyecto se divide en 13 módulos de la parte servidor que se han presentado en capítulos anteriores.

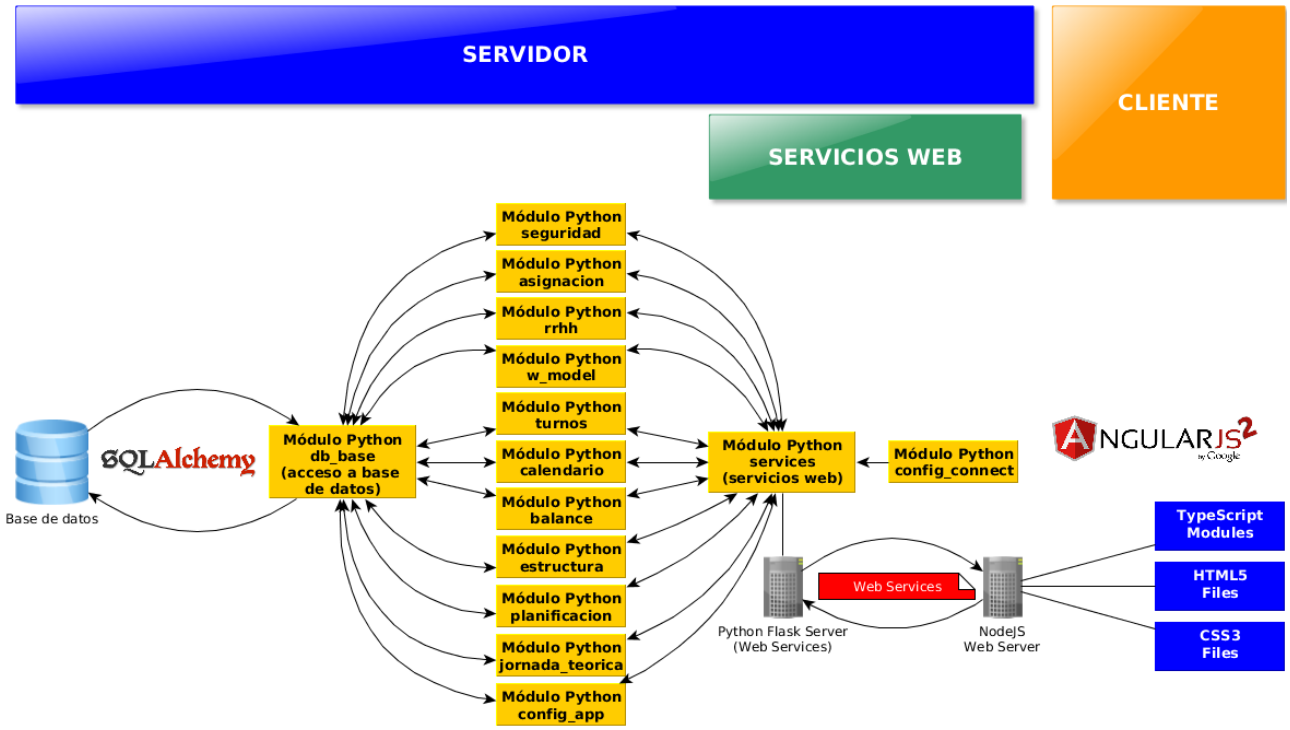


Figura 47. Diagrama de flujo de datos del proyecto durante el desarrollo.

Cada módulo presenta ciertas funcionalidades implementadas mediante clases. Se ha conseguido que tanto el diseño como desarrollo de cada módulo

haya sido independiente, validando su cohesión mediante pruebas de integración con guiones Python de carga de datos y simulación de casuísticas. Así por ejemplo para evaluar la funcionalidad de planificación de puestos de trabajo se han creado guiones Python específicos que hacen uso de las clases que generan y validan dichos elementos, sin tener que tener guardado en base de datos la información ni realizar desde el cliente esta operativa. Incluso para funcionalidades extremadamente específicas se han hecho guiones de validación ya que las propias clases del modelo de negocio se encargan de verificar si los datos que contienen son válidos o no mediante la implementación de métodos de chequeo de datos.

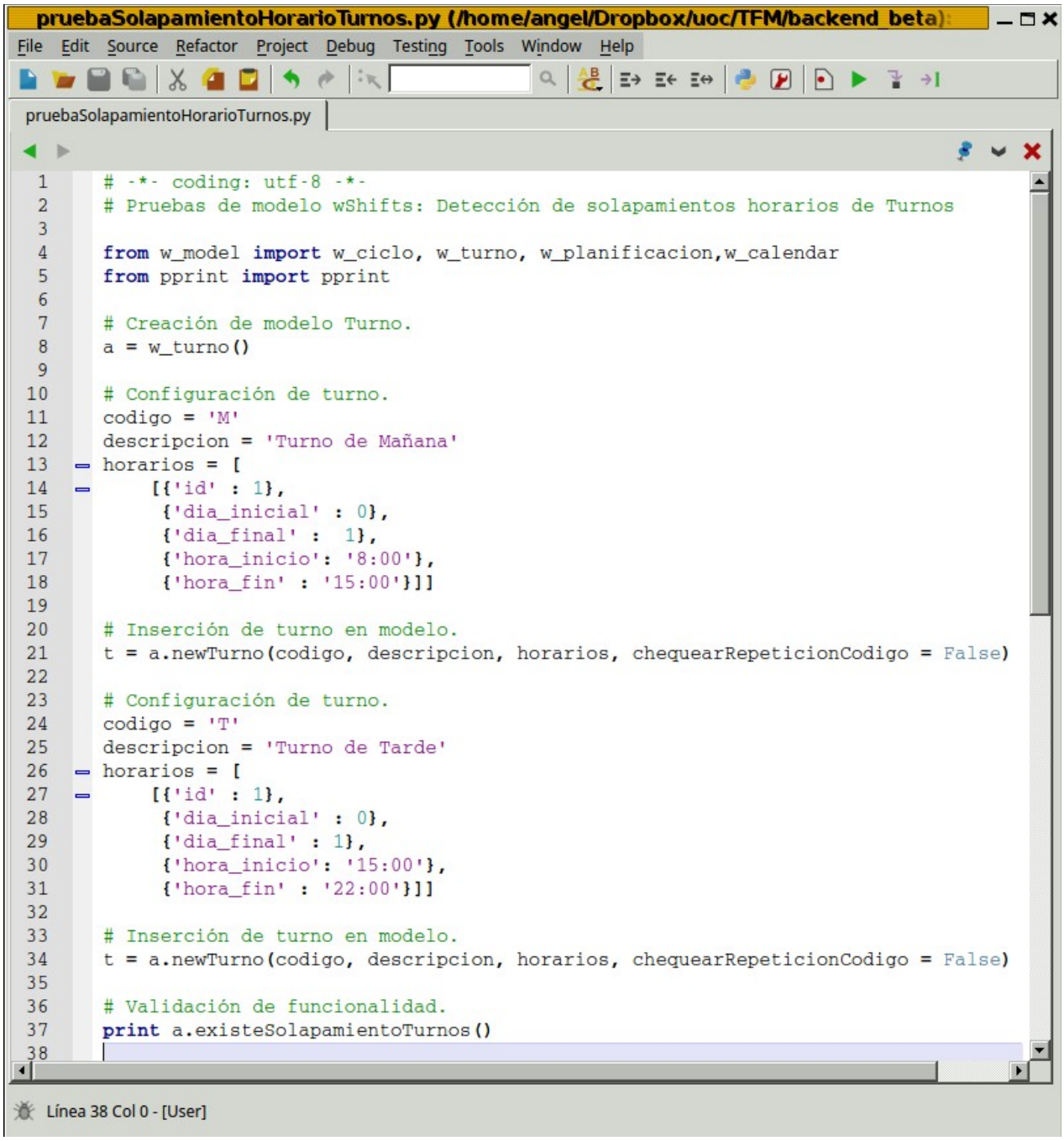


Figura 48. Guión Python de comprobación de funcionalidad de mecanismo de solapamiento horario entre Turnos de trabajo adyacentes. En este caso se está evaluando que un turno de

Mañana (con horario de 8:00 a 15:00) no se solapa temporalmente con el siguiente turno de Tarde (con horario de 15:00 a 22:00). Observar que la instanciación de la clase *w_turno* contempla el método *existeSolapamientoTurnos()*, que devuelve cierto si para los turnos que contiene existe solapamiento horario y falso en caso contrario. En este caso el guión devuelve falso.

Tal como se ha comentado anteriormente el desarrollo se ha dividido en dos partes, la del servidor y la del cliente. En términos generales para la parte del servidor y dependiendo de si se quería implementar una nueva regla de negocio o simular una petición del cliente, el diseño de la nueva funcionalidad se ha realizado siguiendo una metodología ágil, creándola de manera concreta y completa en el menor tiempo posible.

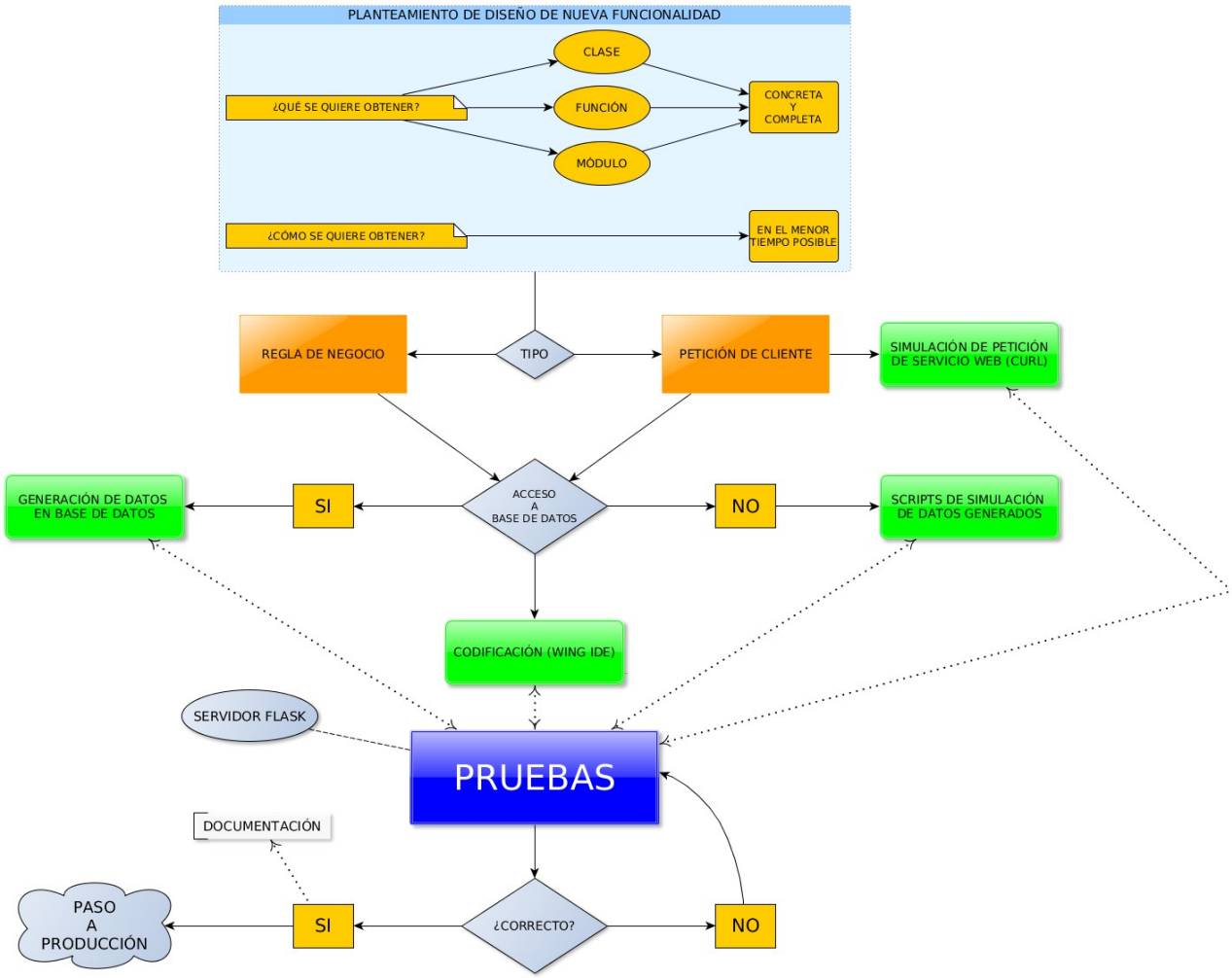


Figura 49. Diagrama de desarrollo general de la parte del servidor.

Para la parte del cliente se ha utilizado la misma filosofía, aunque tal como se ha señalado antes la implementación estaba condicionada al estudio de la nueva herramienta, por lo que la “agilidad” no ha sido la que se hubiese deseado para su diseño y desarrollo.

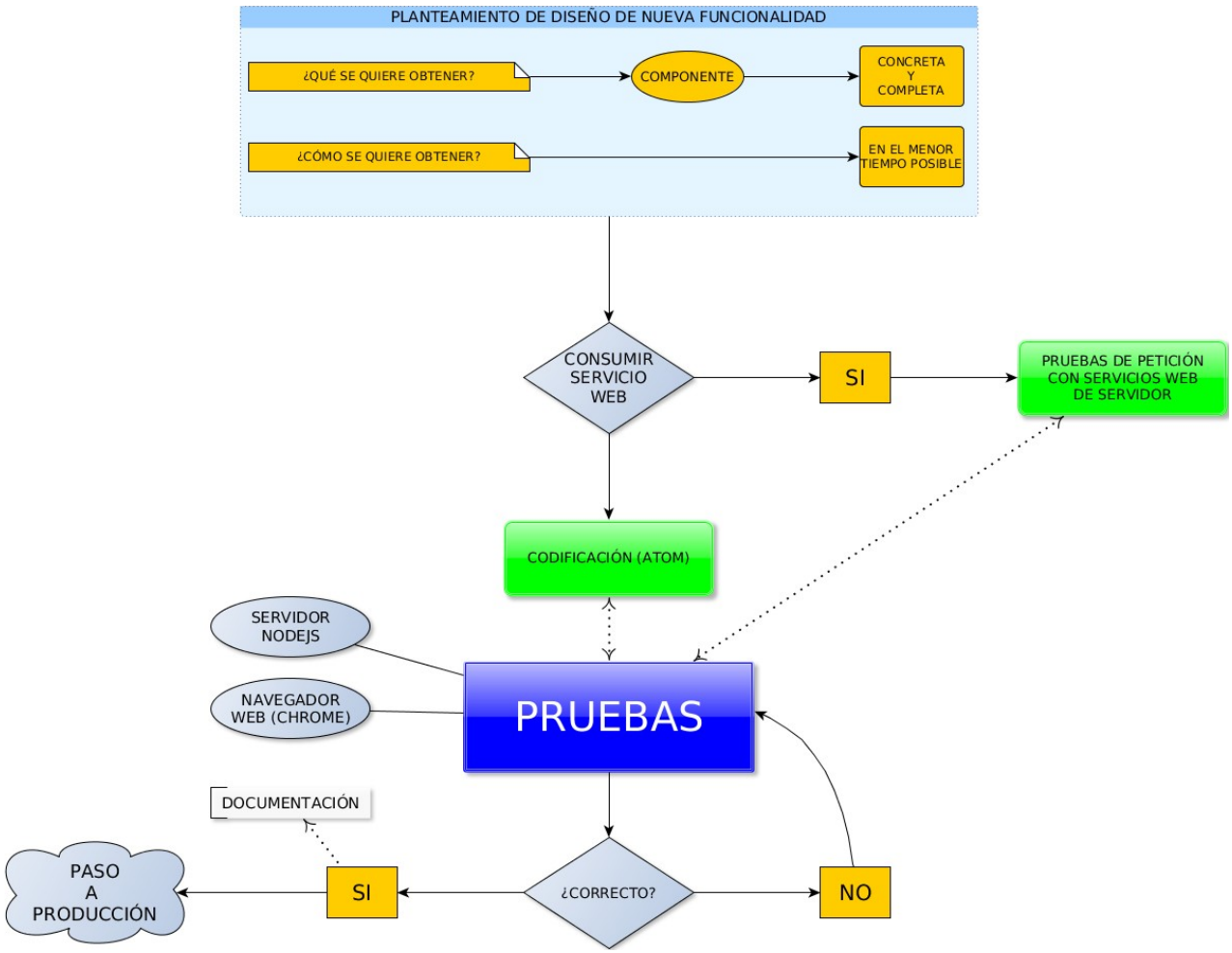


Figura 50. Diagrama de desarrollo general de la parte del cliente.

En el desarrollo del proyecto se han tenido que utilizar dos tipos de servidores, a saber, un servidor de peticiones web implementado por **Werkzeug**, ya que se utiliza el marco de trabajo **Flask** (Ronacher 2016) para realizar dicha funcionalidad y este servidor viene por defecto, y el servidor de la parte del cliente, que viene dado por **NodeJS** (nodejs 2017), que es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Este último servidor se usa para poder desarrollar con Angular 2. En entornos de producción no es necesario.

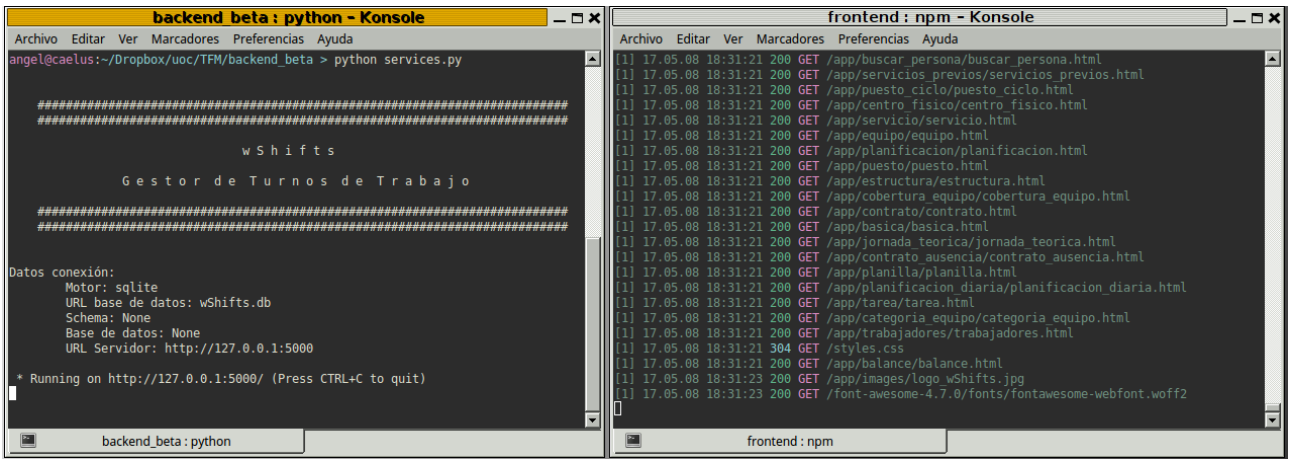


Figura 51. Ejecución de servidores. En el terminal de la izquierda se presenta un servidor Werkzeug esperando peticiones web. En el terminar de la derecha se ha ejecutado un microservidor de NodeJS para el cliente, que hace que se pueda ejecutar en un navegador, mediante la URL oportuna, la parte del cliente.

Todo este trabajo no podría haberse realizado sin previamente estimar los tiempos de desarrollo que normalmente se reflejan mediante un diagrama de Gantt y que se presentan en la siguiente sección.

Temporización

En este capítulo se presenta la temporización del proyecto mediante un **diagrama de Gantt**, que es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo del tiempo total de desarrollo del proyecto.

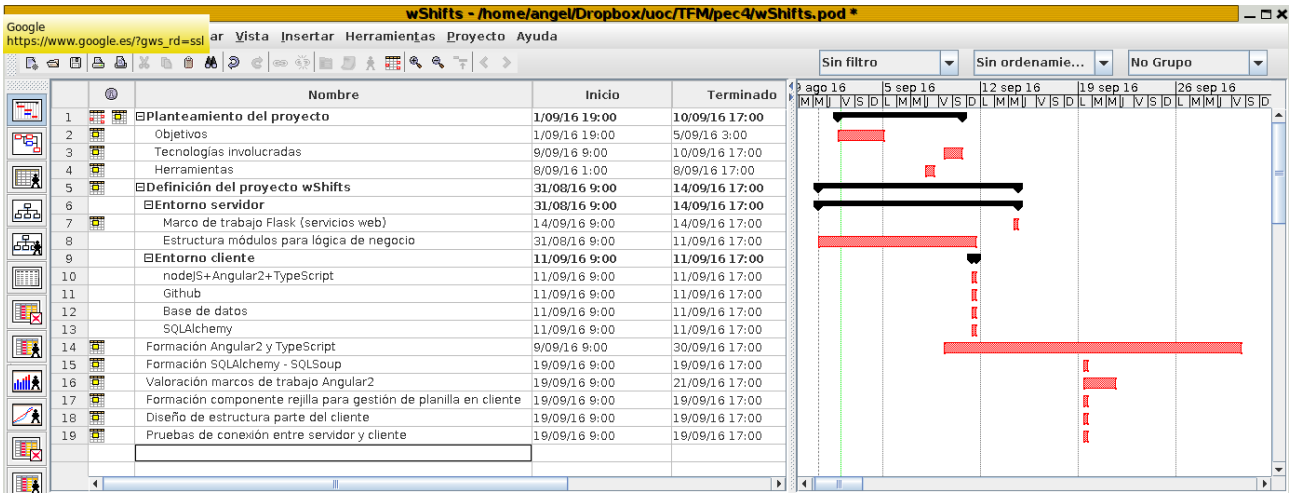


Figura 52. Aplicación Serena OpenProj, versión 1.4. Ejemplo de diseño de diagrama de Gantt.

El proyecto *wShifts* se ha desarrollado en un total de 197 días, a razón de un trabajo en promedio de 4'5 horas diarias, por lo que el total en promedio de trabajo desarrollado ha sido de 886,5 horas. A este dato hay que sumarle el total de horas en la redacción de la memoria, creación de presentación, manual del proyecto y vídeo de presentación, que en promedio son 135 horas.

La realización de este proyecto ha sido intensiva y únicamente los días entre el 28 y el 31 de diciembre de 2016 no se dedicado nada diariamente. Se comenzó con el planteamiento el 1 de septiembre de 2016 y se dio por terminado el 20 de marzo de 2017. La redacción de la memoria se planteó hacerla entre el 21 de marzo y el 31 de mayo.

En el siguiente capítulo se versará sobre el siguiente paso al desarrollo final del producto, esto es, la implantación.

Nombre	Inicio	Terminado
Planteamiento del proyecto	**1/09/16 8:00**	**10/09/16 17:00**
Objetivos	1/09/16 8:00	10/09/16 17:00
Tecnologías involucradas	1/09/16 8:00	10/09/16 17:00
Herramientas	1/09/16 8:00	10/09/16 17:00
Definición del proyecto wShifts	**11/09/16 8:00**	**23/09/16 17:00**
Entorno servidor	**11/09/16 8:00**	**23/09/16 17:00**
Marco de trabajo Flask (servicios web)	11/09/16 8:00	23/09/16 17:00
Estructura de módulos lógica de negocio	11/09/16 8:00	23/09/16 17:00
Entorno cliente	**11/09/16 8:00**	**23/09/16 17:00**
nodeJS + Angular2 + TypeScript	11/09/16 8:00	23/09/16 17:00
Github	**11/09/16 8:00**	**23/09/16 17:00**
Código en repositorio	11/09/16 8:00	23/09/16 17:00
Bases de datos	**11/09/16 8:00**	**23/09/16 17:00**
Pruebas de carga con SQLAlchemy	11/09/16 8:00	23/09/16 17:00

Nombre	Inicio	Terminado
Formación Angular2 y TypeScript.	24/09/16 8:00	5/10/16 17:00
Pruebas de conexión de base de datos (SQLAlchemy).	24/09/16 8:00	5/10/16 17:00
Valoración marcos de trabajo Angular2.	6/10/16 8:00	13/10/16 17:00
Formación en componente de rejilla para gestión de datos de parte del cliente	6/10/16 8:00	13/10/16 17:00
Diseño estructura parte del cliente.	6/10/16 8:00	13/10/16 17:00
Pruebas básicas de conexión entre servidor y cliente.	6/10/16 8:00	13/10/16 17:00
Prototipado de módulo de seguridad parte del servidor: diseño.	14/10/16 8:00	17/10/16 17:00
Prototipado de módulo de seguridad parte del cliente: diseño.	18/10/16 8:00	21/10/16 17:00
Gestión de informes mediante JasperReports.	22/10/16 8:00	26/10/16 17:00
Servidor Tomcat 7. Pruebas.	22/10/16 8:00	26/10/16 17:00
Prototipado de módulo de turnos en parte del servidor: diseño.	27/10/16 8:00	28/10/16 17:00
Prototipado de módulo de planificacion en parte del servidor: diseño e implementación.	27/10/16 8:00	28/10/16 17:00
Primera aproximación a planificación de puestos de trabajo.	27/10/16 8:00	28/10/16 17:00

Figura 53. Diagrama de Gantt, entre 01-09-2016 y 28-10-2016.

Nombre	Inicio	Terminado
Prototipado de módulo de w_model en parte del servidor: diseño.	29/10/16 8:00	1/11/16 17:00
Prototipado de módulo de turnos en parte del cliente: diseño.	2/11/16 9:00	6/11/16 17:00
Prototipado de módulo de planificacion en parte del servidor: diseño.	**6/11/16 9:00**	**18/11/16 17:00**
Calendario de días festivos.	6/11/16 9:00	18/11/16 17:00
Coberturas de servicio.	6/11/16 9:00	18/11/16 17:00
Asignación de ciclos a puestos de trabajo.	6/11/16 9:00	18/11/16 17:00
Pruebas de integración de turnos, ciclos, planificación de puestos.	**19/11/16 9:00**	**29/11/16 17:00**
Parte servidor	19/11/16 9:00	29/11/16 17:00
Parte cliente	19/11/16 9:00	29/11/16 17:00
Prototipado de módulo de recursos humanos (rrhh) en parte del cliente: diseño.	29/11/16 9:00	3/12/16 17:00
Pruebas de integración del módulo de recursos humanos.	**4/12/16 9:00**	**5/12/16 1:00**
Parte cliente	4/12/16 9:00	5/12/16 1:00
Parte servidor	4/12/16 9:00	5/12/16 1:00

Nombre	Inicio	Terminado
Prototipado de módulo de estructura en parte del servidor: diseño	**6/12/16 9:00**	**7/12/16 17:00**
Unidades organizativas.	6/12/16 9:00	7/12/16 17:00
Prototipado de módulo de estructura en parte del cliente: diseño	**6/12/16 9:00**	**7/12/16 17:00**
Unidades organizativas.	6/12/16 9:00	7/12/16 17:00
Prototipado de módulo de planificacion en parte del servidor: diseño	**8/12/16 9:00**	**19/12/16 17:00**
Planificación de puestos de trabajo.	8/12/16 9:00	19/12/16 17:00
Prototipado de módulo de planificacion en parte del cliente: diseño	**8/12/16 9:00**	**19/12/16 17:00**
Planificación de puestos de trabajo.	8/12/16 9:00	19/12/16 17:00
Pruebas de integración en planificación de puestos de trabajo.	8/12/16 9:00	19/12/16 17:00
Prototipado de módulo de estructura en parte del servidor: diseño	**20/12/16 9:00**	**21/12/16 17:00**
Estructura organizativa.	20/12/16 9:00	21/12/16 17:00
Prototipado de módulo de estructura en parte del cliente: diseño	**20/12/16 9:00**	**21/12/16 17:00**
Estructura organizativa.	20/12/16 9:00	21/12/16 17:00
Prototipado de módulo de planificación en parte del cliente: diseño	**22/12/16 9:00**	**23/12/16 17:00**
Coberturas de servicio.	22/12/16 9:00	23/12/16 17:00
Prototipado de módulo de jornada teórica en parte del cliente: diseño	22/12/16 9:00	23/12/16 17:00
Prototipado de módulo de datos básicos de la aplicación en parte del cliente: diseño	22/12/16 9:00	23/12/16 17:00
Pruebas de integración en planificación de puestos de trabajo.	24/12/16 9:00	27/12/16 17:00
Pruebas de integración en creación de ciclos.	24/12/16 9:00	27/12/16 17:00
Prototipado de módulo de recursos humanos (rrhh) en parte del servidor: diseño	1/01/17 9:00	6/01/17 17:00

Figura 54. Diagrama de Gantt, entre 29-10-2016 y 06-01-2017.

Figura 55. Diagrama de Gantt, entre 07-01-2017 y 31-05-2017.

Capítulo VI – Implantación

Implantación del sistema

En esta etapa se llevarán a cabo las actuaciones para implantar el sistema en su entorno operativo definitivo. Mientras que las pruebas unitarias y de integración tienen lugar en un entorno diferente del entorno de operación real, las pruebas de implantación deben realizarse en el sistema en producción. Tal como se ha comentado en capítulos anteriores el proyecto **wShifts** está preparado para sistemas multiusuario (varios usuarios accediendo de manera simultánea a la aplicación) y monousuario (un único usuario accediendo a la aplicación a la vez). Para la presentación e implantación del proyecto se ha elegido la opción monousuario debido a que no se disponía de tiempo para realización de pruebas de integración de sistemas multiusuario. Este proyecto tiene licencia **Apache 2.0**, por lo que es un producto **OpenSource**, y cuyo código fuente está disponible en el repositorio **Github**:

https://github.com/angeloide78/wShifts.git

Este proyecto tiene la posibilidad de integración con el producto Jasper Reports de la empresa TIBCO Jaspersoft (jasper 2017), un sistema de diseño y gestión de informes, escrito en Java, y que necesita de un servidor de aplicaciones, tal como Tomcat o Jboss. En el **Anexo II** se incluyen las clases necesarias para la integración con el aplicativo wShifts, y que no se han incluido en el código fuente.

Planificación de la implantación

La implantación del aplicativo wShifts se basa en 3 fases, que son las siguientes:

- **Infraestructura**: En esta fase se definen los requisitos necesarios para el correcto funcionamiento del aplicativo. En primer lugar se necesitará seleccionar una máquina en donde se instalará el servidor web, la base de datos y los ficheros Python con toda la lógica de negocio de la aplicación. Los sistemas operativos soportados son cualquier distribución Linux actual (por ejemplo, Linux Ubuntu 16.04, Linux Mint 17, etc.) o sistemas Microsoft Windows 7, 8 o 10. Los requerimientos de hardware son mínimos, y cualquier máquina actual soporta perfectamente la ejecución del aplicativo. El proyecto se ha desarrollado en una máquina Intel Core i5, 8 GB RAM, 1GB de disco duro con un sistema Linux Mint 17 y en una máquina Intel Core i7 con 16 GB de RAM y 2GB de disco duro trabajando sobre Windows 10. Sin duda

configuraciones más livianas son igualmente válidas, pero puesto que se ha diseñado con estos parámetros se definen como estándar mínimo para su funcionamiento.

Una vez seleccionada la máquina se dispondrá a instalar el aplicativo, que necesita en este caso software de terceros. El código fuente está disponible en Github, tal como se ha comentado en puntos anteriores, por lo que es necesario tener instalado un cliente Git para poder descargar el código o también es posible descargarlo directamente, empaquetado en formato ZIP, desde:

https://github.com/angeloide78/wShifts/archive/master.zip

Para el funcionamiento de la parte servidor es necesaria la instalación de:

- ✔ El intérprete Python, versión 2.7 (Rossum 2017) ,
- ✔ El mapeador de objetos relacional SQLAlchemy (Bayer 2017).
- ✔ El mapeador de objetos relacionales SQLSoup (Bayer 2016).
- ✔ El marco de trabajo Flask que implementa el servidor web Werkzeug.
- ✔ Flask-Cors, versión 3.0.2 (Dolphin 2016) que es una extensión del marco de trabajo Flask.
- ✔ Requests: HTTP for Humans (Reitz 2016) para la comunicación entre el servidor y el cliente.

Para la parte del cliente será necesario instalar nodejs, que es un entorno de ejecución de JavaScript. Con este software se instalará Angular 2 y TypeScript, y todas las dependencias del proyecto, esto es, el componente de rejilla ag-grid (Crosby 2016) y el marco de trabajo primeNG. El acceso al aplicativo se hace mediante un navegador web. Si bien cualquier navegador funciona correctamente se recomienda el uso de Google Chrome o Firefox, en sus versiones más actualizadas.

Una vez instalados todos los paquetes de software necesarios para la parte servidor, compilados los ficheros Python, instalados Angular, TypeScript, el componente ag-grid y el marco de trabajo primeNG, se realizan pruebas de conexión en la máquina elegida, y se resuelven los problemas que existiesen debidos fundamentalmente a configuraciones en sistemas de cortafuegos que no dejen realizar una correcta conexión o antivirus que no dejen ejecutar el servicio web. Finalmente se pasa a la configuración del aplicativo, que se analizará a continuación.

- **Configuración**: El gestor de turnos **wShifts** es una herramienta dirigida principalmente a los supervisores o coordinadores de unidades organizativas que les permite realizar funciones tales como:

  - ✔ Gestión del control horario de los trabajadores mediante los turnos que realicen.

- ✔ Gestión de permisos, vacaciones y licencias.
- ✔ Generación de listados individuales y colectivos de los recuentos horarios de los trabajadores así como las planificaciones (planillas con los Turnos).

Para poner en marcha el aplicativo es necesario entre 1 y 2 días de trabajo a razón de 6 horas/día para parametrizar los ficheros maestros junto con los usuarios de perfil administrador. Se definen como ficheros maestros a aquellos elementos que son imprescindibles para el funcionamiento básico del aplicativo. En este caso se tiene que definir:

- Estructura organizativa: centros físicos, servicios funcionales, grupos de trabajo y puestos en los que se organiza el sistema empresarial
- Usuarios: usuarios que accederán al aplicativo.
- Roles: perfiles de usuario para tener acceso al aplicativo.
- Acceso de usuarios a recursos del aplicativo: a partir de los roles, los usuarios podrán acceder a diferentes partes del aplicativo.
- Turnos: la configuración del conjunto de actividades que podrán realizar los trabajadores en una jornada ordinaria de trabajo. Por ejemplo mañanas (Código 'M', horario 8:00 a 15:00).
- Ciclos: la configuración del conjunto de ciclos que podrán realizarse en la unidad empresarial (por ejemplo ciclos de mañanas fijas de lunes a viernes).
- Jornada teórica: Total de horas que tiene que realizar un trabajador por año natural.
- Recursos humanos: configuración de tipos de cargos, ausencias y categorías profesionales. Creación e inserción de trabajadores en aplicativo, contratos y ausencias.
- Calendario laboral: definición de festivos por centro físico.

- **Formación**: Cada grupo de trabajo tiene unas propias necesidades con respecto a puestos de trabajo, necesidades de cobertura de servicio y personal. Este tipo de configuración debe de realizarla el responsable de cada grupo, por tanto es necesario realizar una configuración de servicios y grupos con cada supervisor / coordinador responsable. Dependiendo del volumen de la unidad empresarial donde implantar el aplicativo habrá que dedicar mayor o menor tiempo, pero en promedio un mínimo de 2 semanas ya que mientras que se realiza la configuración de grupos se irá formando en el uso del aplicativo. La idea principal es hacer una formación del aplicativo a la misma vez que se va configurando cada equipo de trabajo.

Capítulo VII – Mantenimiento

El código fuente del aplicativo está disponible en un repositorio Github, por lo que siempre se podrá acceder a él y modificarlo libremente, gracias en parte a su licencia de código abierto Apache 2.0. El software de terceros, a saber, Python, SQLAlchemy, SQLSoup, Angular, TypeScript, ag-grid y primeNG utilizan licencias de código abierto, en donde se puede acceder al código y realizar las modificaciones pertinentes, si bien hay opciones de pago para el marco de trabajo primeNG y para el componente ag-grid. En estos dos casos hay opciones de formación y consultoría previo pago.

Si opcionalmente se opta por el uso de Jasper Reports para la gestión de informes hay una comunidad amplia en internet para el soporte de incidencias. Jasper tiene también una versión de pago, en la que se da soporte de software específico antes incidencias, formación y consultoría.

Capítulo VIII – Conclusiones

Se ha creado un sistema básico de gestión de turnos de personal, enfocado para su uso en unidades empresariales estándar, en donde se permite la gestión de actividades de trabajadores, la gestión de sus posibles ausencias y las coberturas de servicio necesarias para que las necesidades de dicho servicio estén cubiertas con respecto al número de trabajadores que son necesarios para tal fin. La aplicación implementa la gestión de usuarios y roles para acceso a la aplicación. Además permite la visualización y gestión de un calendario laboral en donde aparecen todos los trabajadores que están trabajando en un equipo de trabajo, quedando reflejados sus turnos de trabajo, sus posibles ausencias y el total de las coberturas de servicio. Además se puede llevar un control del balance horario de cada trabajador, en función de lo trabajado y de lo que se espera que trabaje a lo largo de un año natural. Por último cabe indicar que la plantilla de trabajadores se gestiona mediante un organigrama empresarial basado en unidades organizativas. Este proyecto tiene Licencia OpenSource (Apache 2.0) y se han usado tecnologías de última generación como Angular 2 y TypeScript combinadas con sistemas más maduros tales como Python, SQLite o Flask. El sistema creado es fácil de utilizar e intuitivo, donde el código fuente está disponible en un repositorio de código abierto (en este caso github).

Como objetivos no conseguidos no se ha llegado a desarrollar directamente la implementación de Jasper Reports en el aplicativo, aunque se ha desarrollado el módulo Python para lanzar los informes desde el mismo, y que puede verse en el **Anexo II**. Además, si bien el sistema está preparado para bases de datos multiusuario, no se ha llegado a implementar en PostgreSQL, implementando la solución en un sistema de base de datos monousuario SQLite. Hay ciertos detalles que cambian de una base de datos a otra, como son el formato de fechas y las funciones para operar sobre ellas (en PostgreSQL hay tipos de datos específicos para fechas y en SQLite no, representándose por una cadena de caracteres), los operandos de definición de campos de autoincremento en la definición de las tablas son distintos (en PostgreSQL el autoincremento se define con *serial* y en SQLite con *autoincrement*), la sintaxis en el uso de “with recursive” para la generación de tablas recursivas también varía levemente, lo que hace que se tenga que especificar para ambos sistemas de manera distinta. En PostgreSQL se hace necesario el uso de usuarios para la conexión a la instancia de base de datos y SQLite no lo necesita. Resumiendo, los cambios son pocos para pasar de un sistema a otro.

El aplicativo puede ampliarse en la funcionalidad de la planilla, ya que debería poder gestionar las coberturas de servicio para horarios de mañana, tarde y noche. Actualmente es para días completos. La funcionalidad del aplicativo en sistemas multiusuario sería la principal ampliación a realizar, basando la persistencia de datos en sistemas gestores de bases de datos, como PostgreSQL, Oracle o MySQL, por ejemplo. La total integración de un sistema

de gestión de informes, como Jasper Reports sería una opción muy válida para poder dar una solución completa al sistema.

La realización de este proyecto ha sido fascinante por lo que he aprendido ya que nunca había desarrollado una aplicación web con HTML5 y CSS3. Angular 2 y TypeScript no los conocía y he tenido que aprender sobre la marcha. El principal problema es que esta tecnología (Angular) ha ido evolucionando desde septiembre de 2016 hasta día de hoy, y cada 6 meses sacan una nueva versión con mejoras. Actualmente Angular está en su versión 4, si bien Google se ha encargado de que las aplicaciones sigan funcionando de una versión a otra, esto no siempre pasa con los marcos de trabajo como PrimeNG, que aún tienen un largo recorrido para ser estables. En definitiva es una tecnología muy joven, que aún falta por depurar. Esto unido a la escasa o nula documentación de calidad en la red ha hecho que el proyecto haya sido duro de realizar en algunos tramos del mismo. Sin embargo la parte del servidor ha sido más liviana de realizar. Nunca había programado servicios web, pero sí he desarrollado en Python, por lo que me ha sido relativamente sencilla esta parte del proyecto. Estos últimos años he estado trabajando como consultor funcional del área de gestión de turnos en sistemas sanitarios públicos, por lo que la implementación en un lenguaje el cual domino ha facilitado la tarea.

Me ha sorprendido lo extenso que puede llegar a ser el desarrollo web, sobretodo en el diseño CSS y en las mejoras que tiene HTML5 sobre su antecesor. En definitiva he disfrutado enormemente haciendo este proyecto, cuyo código fuente comparto con la comunidad en **github** para que sirva de guía en el desarrollo de este tipo de aplicaciones de gestión de turnos y como ejemplo de una aplicación desarrollada con el nuevo marco de trabajo de Google, Angular, y el lenguaje de programación TypeScript de Microsoft.

Referencias bibliográficas

Bayer M. (2017). SQLAlchemy, Mapeador de objetos relacional. http://www.sqlalchemy.org/

Bayer M. (2016). sqlsoup, un mapeador de objetos Python a tablas de bases de datos relacionales. https://pypi.python.org/pypi/sqlsoup/

Bayer, M. & Ellis, J. (2012). SQLSoup's documentation. https://sqlsoup.readthedocs.io/en/latest/

Becker, Fabian (2009). DB Designer Fork 2009. https://sourceforge.net/projects/dbdesigner-fork/

Copeland, R. (2008). Essential SQLAlchemy (1ra ed.). Estados Unidos de América: O'Reilly Media, Inc.

Crosby, Niall (2016). JavaScript Datagrid. https://ag-grid.com/documentation-main/documentation.php

Daniel Stenberg D, Holme S. (2016). curl, una herramienta de línea de comandos y biblioteca para transferir datos mediante URL's. https://curl.haxx.se/

Dolphin C. (2016). Flask-Cors, una extensión de Flask que agrega un decorador para la ayuda de CORS. https://pypi.python.org/pypi/Flask-Cors

Don Ho (2016). Editor Notepad++. https://notepad-plus-plus.org/

Goldwasser. M. & Letscher, D. (2008). Object – Oriented Programming in Python. New Jersey: Pearson Prentice Hall.

Google (2016). One framework – Angular. https://angular.io

Lerner, A., Coury, F., Murray, N. & Taborda C. (2016) ng-book 2 The Complete Book on Angular 2 (1ra ed.). Gran Bretaña: Amazon Distribution.

Loper E. (2008). epydoc. Generación automática de documentación de API para Python. http://epydoc.sourceforge.net/

Node.js Foundation (2017). nodeJS, un entorno de ejecución para JavaScript. https://nodejs.org/es/

Novák, I. (2016). Unraveling Angular 2 (1ra ed.). Leipzig: Amazon Distribution.

Moiseev, A. & Fain, Y. (2016). Angular 2 Development with TypeScript (1ra ed.). Nueva York: Manning.

OpenProj - Project Management (2013). Gestión de proyectos. https://sourceforge.net/projects/openproj/

Piacentini M., Morgan P., Miltner J., Ravanini R., Peinthor R., Kleusberg M., Clift J., Haller J.T (2016). DB Browser for SQLite. http://sqlitebrowser.org/

PrimeTek (2016). PrimeNG. http://www.primefaces.org/primeng

Refsnes Data (2016). W3Schools Online Webs Tutorials. http://www.w3schools.com/

Reitz K. (2016). Requests, una librería HTTP para Python. http://docs.python-requests.org/en/master/

Ronacher A. (2016). Flask, un marco de trabajo Python para desarrollo web. http://flask.pocoo.org/

Rubiales Gómez, M. (2014). HTML5, CSS3 y JavaScript (1ra ed). Madrid: Anaya Multimedia.

Stack Exchange (2016). Stack Overflow. http://stackoverflow.com/

TIBCO Jaspersoft (2017). Jasper Reports. https://www.jaspersoft.com/

van Rossum G. (2017). Python, un lenguaje de programación de propósito general. https://www.python.org/

yWorks (2016). Editor de gráficos yEd. https://www.yworks.com/

Wingware (2016). Wingware Python IDE. https://wingware.com/

Anexo I – Licencia

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking

systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
2. You must cause any modified files to carry prominent notices stating that You changed the files; and
3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or

distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Anexo II – Módulo jasperreports.py

En este anexo se presenta el módulo **jasperreports.py**, necesario para poder conectar informes generados por la aplicación **Jasper Reports** con una aplicación escrita en **Python**.

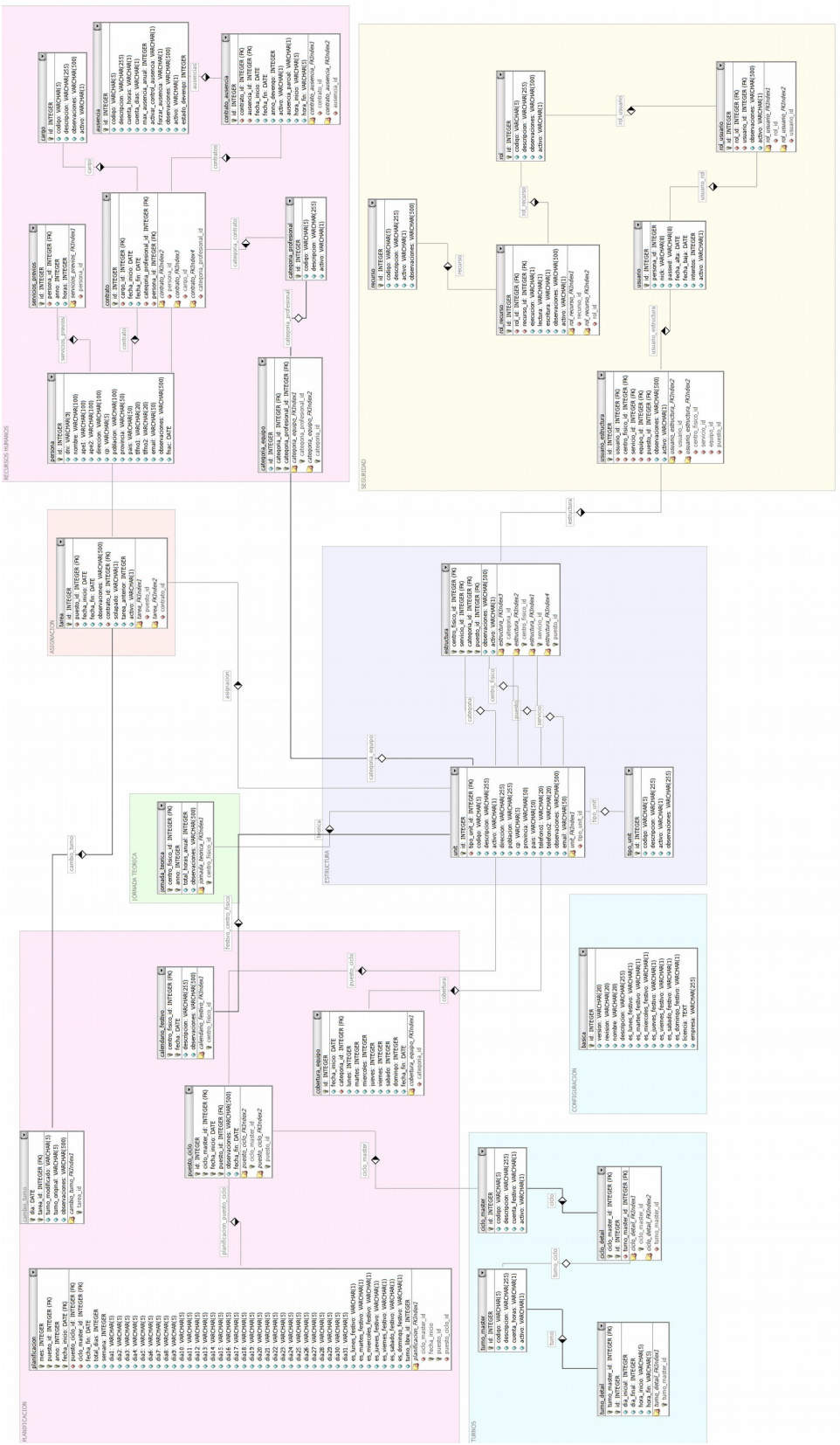
```python
# -*- coding: utf-8 -*-

import requests
import webbrowser
from datetime import datetime
import hashlib
import os
from requests.auth import HTTPBasicAuth

# 23-10-2016 Módulo de Jasper Reports.

class jasperreports(object):
    '''Clase de configuración de informes para Jasper Reports'''

    def __init__(self, maquina, puerto, recurso):
        self.__maquina = maquina
        self.__puerto = puerto
        self.__recurso = recurso
        self.__session = requests.Session()
        self.__token = self.__generador_token()
        self.__usuario = None
        self.__passwd = None
        self.__cookie = None
        self.__url = None

    def __generador_token(self):
        '''Generador de token'''
        return hashlib.sha1(os.urandom(128)).hexdigest()

    def login(self, usuario, passwd):
        '''Clase login para acceso a JasperServer'''

        # Formación de URL para login.
        url = "http://%s:%s/jasperserver/rest/login" % (self.__maquina, \
                                                        self.__puerto)

```

```
37          # Formación de cuerpo y cabecera.
38          data = "j_username=%s&j_password=%s" % (usuario, passwd)
39          header = {"content-type" : "application/x-www-form-urlencoded"}
40
41          # Se envían datos ed inicio de sesión.
42          ret = self.__session.post(url, data, headers = header)
43
44          # Se guarda cookie, usuario y contraseña.
45          self.__cookie = ret.cookies['JSESSIONID']
46          self.__usuario = usuario
47          self.__passwd = passwd
48
49          # Algo de información.
50          print('Cookie.............: %s' % self.__cookie)
51          print('Login response.....:')
52
53      def report(self, ParentFolderUri, reportUnit, params = None):
54          '''Método de lanzamiento de informe'''
55
56          # Cabeceras.
57          headers = {"Authorization": "Basic " + self.__token}
58
59          # Cookie.
60          cookies = dict(cookies=self.__cookie)
61
62          # Autenticación.
63          auth = HTTPBasicAuth(self.__usuario, self.__passwd)
64
65          # Se crea la URL manualmente...
66          url = "http://%s:%s/jasperserver/flow.html?_flowId=" % (self.__maquina,
67                                                                  self.__puerto)
68          url += "viewReportFlow&_flowId=viewReportFlow&ParentFolderUri="
69          url += "%s&reportUnit=%s&standAlone=true" % (ParentFolderUri, \
70                                                       reportUnit)
```

```
71
72          # ALGG 27-10-2016 Y se incluye usuario y contraseña en la url...
73          url += "&j_username=%s&j_password=%s" % (self.__usuario, \
74                                                  self.__passwd)
75
76          # Se lanza el get...
77          ret = self.__session.get(url, auth = auth, cookies = cookies, \
78                                   headers = headers)
79
80          # Damos algo de información.
81          print('Report.............: %s' % ret.url)
82          print('Report Response....: %s' % ret)
83
84          try:
85              webbrowser.open(ret.url)
86          except:
87              nomfich = 'temp_error_%s.html' % str(datetime.now())
88              f = open('temp/%s' % nomfich,'w')
89              f.write(ret.text)
90              f.close()
91              webbrowser.open('temp/%s' % nomfich)
92
93      def info(self):
94          url = 'http://%s:%s/%s/rest_v2/serverInfo' % (self.__maquina, \
95                                                        str(self.__puerto), \
96                                                        self.__recurso)
97
98          # Se realiza petición de información al servidor JasperServer...
99          ret = requests.get(url,headers={"accept":"application/json"}).json()
100
101         # Se recupera información del servidor si todo ha ido bien...
102         data = { 'dateFormatPattern' : ret['dateFormatPattern'],
103                  'datetimeFormatPattern' : ret['datetimeFormatPattern'],
104                  'version' : ret['version'],
105                  'edition' : ret['edition'],
106                  'build' : ret['edition']
107                  }
108
109         # Se devuelven datos.
110         return data
111
112     def logout(self):
113         if self.__cookie is None:
114             print("Logout..........: No hay cookie")
115         else:
116
117             url = "http://%s:%s/jasperserver/logout.html" % (self.__maquina, \
118                                                              self.__puerto)
119
120             ret = self.__session.get(url, \
121                                      headers = {'JSSESIONID' : self.__cookie})
122
123             print('Logout response....: %s' % ret)
124
```

Anexo III – Esquema de base de datos.

En este anexo se presenta el esquema de entidad – relación de la base de datos implementado en el proyecto.



Anexo IV – API.

wShifts

API Documentation

May 7, 2017

Contents

1 Module asignacion

1.1 Variables

Name	Description
___package___	**Value:** `None`

1.2 Class AsignarTrabajador

object
└ **asignacion.AsignarTrabajador**

Trabajadores candidatos a asignación

1.2.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_asignar_trabajador(*self*, *equipo_id*, *fecha*)

Devuelve: Los trabajadores que tienen contrato a "fecha" dada, cuya categoría profesional por contrato está contemplada dentro del equipo de trabajo "equipo_id". Si además tiene asignaciones en esa fecha dada, también saldrán. (trab_id, dni, ape1, ape2, nombre, fini_c, ffin_c, tarea_id, fini_t, ffin_t, eq_id, eq_cod, eq_desc)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

1.2.2 Properties

Name	Description
Inherited from object	
___class___	

1.3 Class CambioTurno

object
└ **asignacion.CambioTurno**

Clase CambioTurno

1.3.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

set_cambio_turno(*self*, *datos*)

Actualización de datos. Tabla: cambio_turno

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

1.3.2 Properties

Name	Description
Inherited from object	
___class___	

1.4 Class Tarea

object
└ **asignacion.Tarea**

Clase Tarea

1.4.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_tarea(*self*, *equipo_id*=`None`, *anno*=`None`)

Devuelve: (id, p_id, p_cod, p_desc, fecha_inicio, fecha_fin, observ, contrato_id, persona_id, nombre, ape1, ape2, dni, solapado) Opciones de filtrado: equipo_id <id equipo>, anno <año en donde está en vigor la asignación

set_tarea(*self*, *datos*)

Actualización de datos. Tabla: tarea

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

1.4.2 Properties

Name	Description
Inherited from object ___class___	

2 Module balance

2.1 Variables

Name	Description
___package___	**Value:** `None`

2.2 Class Balance

object
└ **balance.Balance**

Clase Balance

2.2.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_horas(*self*, *mes*=`None`, *anno*=`None`, *fecha_inicio*=`None`, *fecha_fin*=`None`, *persona_id*=`None`, *cf_id*=`None`, *sf_id*=`None`, *eq_id*=`None`, *p_id*=`None`)

Devuelve el total de horas entre un rango de fechas. Filtro: centro físico <cf_id>, servicio <sf_id>, equipo <eq_id>, puesto <p_id>, ident. del trabajador <persona_id>

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

2.2.2 Properties

Name	Description
Inherited from object	
___class___	

3 Module calendario

3.1 Variables

Name	Description
__package__	**Value:** `None`

3.2 Class CalendarioFestivo

object
└ **calendario.CalendarioFestivo**

3.2.1 Methods

__init__(*self*, *conn*)

Constructor

Overrides: object.__init__

getFestivosByCentroFisicoId(*self*, *idCentroFisico*, *anno*=`None`)

Devuelve días festivos a partir del id de un Centro Físico. Si anno es None devuelve todos los festivos. Si anno es un año devuelve los festivos de ese año

get_calendario_festivo(*self*, *centro_fisico_id*=`None`, *anno*=`None`)

Devuelve: (id, centro_fisico_id, cod_cf, desc_cf, año, fecha_festivo, desc_festivo, observ_festivo) Tabla: calendario_festivo Opciones de filtrado: Id de centro físico <centro_fisico_id>, año de calendario laboral <anno>

newCalendarioFestivo(*self*, *anno*)

Creación de calendario de días festivos

newFestivo(*self*, *centro_fisico_id*, *(anno, mes, dia)*, *descripcion*, *observaciones*)

Creación de día festivo

set_calendario_festivo(*self*, *datos*)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

3.2.2 Properties

Name	Description
Inherited from object	
___class___	

4 Module config_app

4.1 Variables

Name	Description
___package___	**Value:** `None`

4.2 Class Basica

object
└
config_app.Basica

Clase de datos básicos de la aplicación

4.2.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_basica(*self*)

Devuelve: id, version, revision, nombre, descripcion, es_lunes_festivo, es_martes_festivo, es_miercoles_festivo, es_jueves_festivo, es_viernes_festivo, es_sabado_festivo_es_domingo_festivo, licencia, empresa

get_conf_aplic(*self*)

Devuelve datos básicos de configuración de aplicación (version, revision, nombre, descripcion)

set_basica(*self*, *datos*)

Actualización de datos. Tabla: basica

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

4.2.2 Properties

Name	Description
Inherited from object ___class___	

5 Module config_connect

5.1 Functions

configApp()

Función de configuración de app. Devuelve: <True, engine, url, schema, db> si es correcto o <False, None, None, None, None> si hubi error.

5.2 Variables

Name	Description
___FICH_CONF_APP_-_	**Value:** `'config_connect.db'`
___SQL___	**Value:** `'select engine, url, schema, db, backend from conexion wh...`
___package___	**Value:** `None`

5.3 Class Config_app

object
└ **config_connect.Config_app**

Configuración de app

5.3.1 Methods

___init___(*self*, *fich_conf*)

Constructor

Overrides: object.___init___

get_info_connect(*self*, *consultaSQL*)

Recupera información de conexión actual de la app. Devuelve: Es correcto: True, engine, url, schema, db Es incorrecto: False, None, None, None, None

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(),

___str___(), ___subclasshook___()

5.3.2 Properties

Name	Description
Inherited from object	
___class___	

6 Module db_base

6.1 Variables

Name	Description
___package___	**Value:** `None`
func	**Value:** `<sqlalchemy.sql.functions._FunctionGenerator object at 0x...`

6.2 Class Db

object
└ **db_base.Db**

Clase conexión base.

6.2.1 Methods

___init___(*self*, *engine*, *url*, *schema*, *bbdd*)

Constructor

Overrides: object.___init___

actualizar(*self*)

Commit de conexión

actualizar_ausencia(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de ausencia Tabla: ausencia Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_basica(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de datos básicos de aplicación. Tabla: basica Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_calendario_festivo(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de calendario festivo Tabla: calendario_festivo Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_cambio_turno(*self*, *dia*, *tarea_id*, *turno_modificado*, *turno_original*, *observaciones*)

Cambio de turno en planificación diaria Tabla: cambio_turno Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_cargo(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de cargo Tabla: cargo Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_catEq(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de categorías en equipos de trabajo Tabla: categoria_equipo Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_ciclo(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de turno Tablas: ciclo_master y ciclo_detail Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_cobertura_equipo(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de coberturas de equipo Tabla: cobertura_equipo Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_contrato(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de contrato de persona Tabla: contrato Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_contrato_ausencia(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de ausencia en contrato Tabla: contrato_ausencia Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_cp(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de categoría profesional Tabla: categoria_profesional Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_estructura(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de estructura organizativa Tablas: estructura Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_jt(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de jornada teórica Tabla: jornada_teorica Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_persona(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de persona Tabla: persona Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_puesto_ciclo(*self*, *campos*, *reg_nuevo*, *reg_eliminado*, *detalle_reg_eliminado*)

Actualización de ciclos asociados a puestos de trabajo Tablas: puesto_ciclo Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_recurso(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de recurso Tabla: recurso Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_rol(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de rol Tabla: rol Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_rol_recurso(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de roles asociados a recursos Tabla: rol_recurso Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_rol_usuario(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de roles asociados a usuarios Tabla: rol_usuario Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_sp(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de servicios previos Tabla: servicios_previos Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_tarea(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de tarea Tabla: tarea Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_turno(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de turno Tablas: turno_master y turno_detail Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_unit(*self*, *campos*, *reg_nuevo*, *reg_eliminado*, *opcion*=`None`)

Actualización de unidad organizativa Tabla: unit Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_usuario(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de usuario Tabla: usuario Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

actualizar_usuario_estructura(*self*, *campos*, *reg_nuevo*, *reg_eliminado*)

Actualización de usuarios asociados a estructura Tabla: usuario_estructura Actualización a nivel de modificación de campos, inserción de filas, eliminación de filas.

connect(*self*)

Creación de conexión. Devuelve cierto si todo es correcto y false en caso contrario

delete_ausencia(*self*, *_id*, *hacer_commit*=`True`)

Borrado de ausencia Tabla: ausencia

delete_calendario_festivo(*self*, *_id*, *hacer_commit*=`True`)

Borrado de festividad Tabla: calendario_festivo

delete_cargo(*self*, *_id*, *hacer_commit*=`True`)

Borrado de cargo Tabla: cargo

delete_catEq(*self*, *_id*, *hacer_commit*=`True`)

Borrado de relación de equipo y categoría profesional Tabla: categoria_equipo

delete_ciclo(*self*, *ciclo_master_id*, *borrar_cabecera*=`True`, *hacer_commit*=`True`)

Borrado de ciclo Tablas: ciclo_master y ciclo_detail

delete_cobertura_equipo(*self*, *_id*, *hacer_commit*=`True`)

Borrado de cobertura de equipo Tabla: cobertura_equipo

delete_contrato(*self*, *_id*, *hacer_commit*=`True`)

Borrado de contrato Tabla: contrato

delete_contrato_ausencia(*self*, *_id*, *hacer_commit*=`True`)

Borrado de ausencia del contrato Tabla: contrato_ausencia

delete_cp(*self*, *_id*, *hacer_commit*=`True`)

Borrado de categoría profesional Tabla: categoria_profesional

delete_estructura(*self*, *_id*, *hacer_commit*=`True`)

Borrado de relación en estructura organizativa Tablas: estructura

delete_jt(*self*, *_id*, *hacer_commit*=`True`)

Borrado de jornada teórica Tabla: jornada_teorica

delete_persona(*self*, *_id*, *hacer_commit*=`True`)

Borrado de persona Tabla: persona

delete_puesto_ciclo(*self*, *_id*, *detalle*, *hacer_commit*=`True`)

Borrado de relación de ciclos con puesto de trabajo Tablas: puesto_ciclo

delete_recurso(*self*, *_id*, *hacer_commit*=`True`)

Borrado de recurso Tabla: recurso

delete_rol(*self*, *_id*, *hacer_commit*=`True`)

Borrado de rol Tabla: rol

delete_rol_recurso(*self*, *_id*, *hacer_commit*=`True`)

Borrado de rol asociado a recurso Tabla: rol_recurso

delete_rol_usuario(*self*, *_id*, *hacer_commit*=`True`)

Borrado de rol asociado a usuario Tabla: rol_usuario

delete_sp(*self*, *_id*, *hacer_commit*=`True`)

Borrado de servicios previos Tabla: servicios_previos

delete_tarea(*self*, *_id*, *hacer_commit*=`True`)

Borrado de tarea (asignación) Tabla: tarea

delete_turno(*self*, *turno_master_id*, *_id*, *hacer_commit*=`True`)

Borrado de turno Tablas: turno_master y turno_detail

delete_unit(*self*, *_id*, *hacer_commit*=`True`)

Borrado de unidad organizativa Tabla: unit

delete_usuario(*self*, *_id*, *hacer_commit*=`True`)

Borrado de usuario Tabla: usuario

delete_usuario_estructura(*self*, *_id*, *hacer_commit*=`True`)

Borrado de usuario asociado a unidad organizativa Tabla: usuario_estructura

exists_asignacion_in_contrato(*self*, *contrato_id*)

Devuelve: True si el contrato tiene asignaciones y False en caso contrario
Tabla: contrato, tarea

exists_ausencia_en_contrato(*self*, *ausencia_id*)

Devuelve: True si la ausencia existe en un contrato y False en caso contrario.
Tabla: contrato_ausencia

exists_ausencias_not_in_contrato(*self*)

Devuelve: True si existen ausencias que se inician o terminan fuera de las fechas de un contrato, y False en caso contrario. Tabla: contrato

exists_cargo_en_contrato(*self*, *cargo_id*)

Devuelve: True si el cargo existe en un contrato y False en caso contrario.
Tabla: contrato

exists_catProf_en_contrato(*self*, *categoria_profesional_id*)

Devuelve: True si la categoría profesional existe en un contrato y False en caso contrario. Tabla: contrato

exists_catProf_en_equipo(*self*, *categoria_profesional_id*)

Devuelve: True si la categoría profesional existe en la relación de equipo con categoría profesional y False en caso contrario. Tabla: categoria_equipo

exists_catProf_equipo(*self*, *cat_prof_id*, *eq_id*)

Devuelve: True si la relación <categoría profesional, equipo de trabajo> existe y False en caso contrario Tabla: categoria_equipo

exists_cfisico_con_calendario(*self*, *centro_fisico_id*)

Devuelve: True si el centro físico tiene calendarios festivos asociadss y False en caso contrario Tablas: jornada_teorica

exists_cfisico_con_jteorica(*self*, *centro_fisico_id*)

Devuelve: True si el centro físico tiene jornadas teóricas asociadas y False en caso contrario Tablas: jornada_teorica

exists_equipo_con_cobertura(*self*, *equipo_id*)

Devuelve: True si el equipo tiene categorías profesionales asociadas y False en caso contrario Tablas: categoria_equipo

exists_in_estructura(*self*, *cf_id*=`None`, *sf_id*=`None`, *eq_id*=`None`, *p_id*=`None`)

Devuelve: True si la unidad organizativa existe en la tabla estructura y False en caso contrario Tabla: estructura (estructura organizativa)

exists_in_puesto_ciclo(*self*)

Devuelve: True si existe más de un ciclo que empieza el mismo día para el puesto Tabla: puesto_ciclo (asociación de ciclos a puestos de trabajo)

exists_puesto_asignado(*self*, *puesto_id*)

Devuelve: True si el puesto está asignado y False en caso contrario Tabla: tarea

exists_puesto_planificado(*self*, *puesto_id*)

Devuelve: True si el puesto está planificado y False en caso contrario Tablas: puesto_ciclo

exists_rol_recurso(*self*, *rol_id*, *recurso_id*=`None`)

Devuelve: True si existe la tupla (recurso_id, rol_id) y False en caso contrario. Tabla: rol_recurso (asociación de recurso con rol)

exists_rol_usuario(*self*, *rol_id*=`None`, *usuario_id*=`None`)

Devuelve: True si existe la tupla (usuario_id, rol_id) y False en caso contrario. Tabla: rol_usuario (asociación de usuario con rol)

exists_solapamiento_in_cobertura(*self*)

Devuelve: True si existe solapamiento entre coberturas para un grupo funcional y False en caso contrario Tabla: cobertura_equipo

exists_solapamiento_in_contrato(*self*)

Devuelve: True si existe solapamiento entre contratos para un trabajador y False en caso contrario Tabla: contrato

exists_solapamiento_in_puesto_ciclo(*self*)

Devuelve: True si existe solapamiento entre rango de fechas para un puesto y False en caso contrario Tabla: puesto_ciclo (asociación de ciclos a puestos de trabajo)

exists_sp_in_persona(*self*)

Devuelve: True si el trabajador tiene servicios previos para el año y False en caso contrario. Tabla: servicios_previos

exists_turno_en_cambio_turno(*self*, *turno_id*)

Devuelve: True si el turno existe en un cambio de turno y False en caso contrario. Tablas: cambio_turno, turno_master

exists_turno_en_ciclo(*self*, *turno_id*)

Devuelve: True si el turno existe en un ciclo y False en caso contrario. Tablas: ciclo_detail

exists_usuario_estructura(*self*, *usuario_id*, *cf_id*=`None`, *sf_id*=`None`, *eq_id*=`None`)

Devuelve: True si existe la tupla usuario, estructura y False en caso contrario. Tabla: usuario_estructura (asociación de usuario con rol)

fechaPython(*self*, *fecha*)

Devuelve objeto Python Date a partir del año, mes y día

getContratoByAusencia(*self*, *contrato_ausencia_id*)

Devuelve: Id de contrato a partir del identificador de ausencia de contrato. Tabla: contrato_ausencia

get_asignar_trabajador(*self*, *equipo_id*, *fecha*)

Devuelve: Los trabajadores que tienen contrato a "fecha" dada, cuya categoría profesional por contrato está contemplada dentro del equipo de trabajo "equipo_id". Si además tiene asignaciones en esa fecha dada, también saldrán. (trab_id, dni, ape1, ape2, nombre, contrato_id, fini_c, ffin_c, tarea_id, fini_t, ffin_t, eq_id, eq_cod, eq_desc)

get_ausencia(*self*, *id_*=`None`, *codigo*=`None`, *activo*=`None`)

Devuelve: (id, codigo, descripcion, cuenta_horas, cuenta_dias, max_ausencia_anual, activar_control_ausencia, forzar_ausencia, observaciones, activo, estado_devengo) Tabla: ausencia Opciones de filtrado: id_ <id ausencia,>, codigo <cód. ausencia>, activo <'S', 'N'>

get_basica(*self*)

Devuelve: id, version, revision, nombrem, descripcion, es_lunes_festivo, es_martes_festivo, es_miercoles_festivo, es_jueves_festivo, es_viernes_festivo, es_sabado_festivo_es_domingo_festivo, licencia, empresa Tabla: basica (datos básicos de la aplicación)

get_calendario_festivo(*self*, *centro_fisico_id*=`None`, *anno*=`None`)

Devuelve: (centro_fisico_id, cod_cf, desc_cf, año, fecha_festivo, desc_festivo, observ_festivo) Tabla: calendario_festivo Opciones de filtrado: Id de centro físico <centro_fisico_id>, año de calendario laboral <anno>

get_cambios_turnos(*self*, *fini_t*, *ffin_t*, *tarea_id*)

Se recuperan los posibles cambios de turnos de la planificación diaria para una tarea entre un rango de fechas de tarea. Tabla: cambio_turno

get_cargo(*self*, *id_*=`None`, *activo*=`None`)

Devuelve: (id, codigo, descripcion, observaciones, activo) Tabla: cargo Opciones de filtrado: id_ <id. cargo>, activo <'S', 'N'>

get_catEq(*self*, *eq_id*=None, *cat_id*=None)

Devuelve: (id, eq_id, eq_cod, eq_desc, cat_id, cat_cod, cat_desc) Tabla: categoria_equipo

get_ciclo(*self*, *activo*=None, *id_*=None)

Devuelve: (id, codigo, descripcion, cuenta_festivo, activo, id_detalle, numero_dia, turno_master_id Tablas: ciclo_master y ciclo_detail Opciones de filtrado: activo <'S','N'>, codigo <código ciclo>

get_coberturaServicio(*self*, *equipo_id*, *mes*, *anno*)

Devuelve: (dia_trab, dia_semana, descuadre, info) Tabla: cobertura_equipo, tarea, v_planificacion Esta query da las coberturas de servicio, teniendo en cuenta turnos que cuenten o no horas, ausencias que cuenten ó no presencias (días) y cambios de turnos.

get_cobertura_equipo(*self*, *equipo_id*=None, *fecha*=None)

Devuelve: (id, fecha_inicio, categoria_id, lunes, martes, miercoles, jueves, viernes, sabado, domingo, fecha_fin) Tabla: cobertura_equipo Opciones de filtrado: equipo_id <id equipo>, fecha <fecha en donde la cobertura es efectiva>

get_contrato(*self*, *persona_id*)

Recupera todos los contratos de una persona Devuelve: (id, cargo_id, cargo_cod, cargo_desc, fecha_inicio, fecha_fin, cp_id, cp_cod, cp_desc, persona_id) Tabla: contrato Opciones de filtrado: persona_id <id de persona>

get_contrato_ausencia(*self*, *contrato_id*, *activo*=None, *fecha_desde*=None, *fecha_hasta*=None)

Recupera todas las ausencias de un contrato Devuelve: (id, contrato_id, aus_id, aus_cod, aus_desc, fecha_inicio, fecha_fin, anno_devengo, activo, ausencia_parcial, hora_inicio, hora_fin) Tabla: contrato_ausencia Opciones de filtrado: contrato_id <id de contrato>, activo <'S', 'N'>, fecha_desde, fecha_hasta

get_cp(*self*, *id_*=None, *activo*=None)

Devuelve: (id, codigo, descripcion, activo) Tabla: cargo Opciones de filtrado: id_ <id. cargo>, activo <'S', 'N'>

get_estructura(*self*, *puesto_id*=None, *activo*=None)

Devuelve: (cf_id, cf_cod, cf_desc, sf_id, sf_cod, sf_desc, eq_id, eq_cod, eq_desc, p_id, p_cod, p_desc, observ, activo) Tabla: estructura (estructura organizativa)

get_horas(*self*, *fecha_inicio*, *fecha_fin*, *persona_id*=None, *cf_id*=None, *sf_id*=None, *eq_id*=None, *p_id*=None)

Devuelve el total de horas de un trabajador entre un rango de fechas. Filtro: centro físico <cf_id>, servicio <sf_id>, equipo <eq_id>, puesto <p_id>, ident. del trabajador <persona_id> Vista: v_balance

get_horas_ciclo(*self*, *ciclo_semanal*)

Devuelve: (horas, minutos) Tablas: turno_master, turno_detail Opciones de filtrado: ciclo_semanal <lista de semanas, donde cada semana es una lista de los días de la semana y cada elemento es un código de turno>

get_jt(*self*, *centro_fisico_id*=None, *anno*=None)

Devuelve: (id, cf_id, cf_cod, cf_desc, anno, total_horas_anual, observaciones) Tablas: jornada_teorica, unit Opciones de filtrado: Id de centro físico <centro_fisico_id>, año de calendario laboral <anno>

get_persona(*self*, *id_*=None, *dni*=None)

Devuelve: (id, dni, nombre, ape1, ape2, direccion, cp, poblacion, provincia, pais, tlfno1, tlfno2, email, observaciones, sexo, fnac) Tabla: persona (datos personales del trabajador) Opciones de filtrado: id_ <id de persona>, dni <dni de persona>

get_planificacion(*self*, *anno*, *mes*, *puesto_id*=None, *fecha_inicio*=None, *equipo_id*=None, *visualizacion*=0)

Devuelve: () Tabla: planificacion (planificaciones de ciclos asociados a puestos)

get_planificacion_diaria(*self*, *anno*, *mes*, *puesto_id*=None, *fecha_inicio*=None, *equipo_id*=None)

Devuelve: La planilla Tabla: planificacion, tarea (planilla)

get_puesto_ciclo(*self*, *puesto_id*=None, *id_*=None)

Devuelve: (ciclo_id, ciclo_desc, cf_id, cf_desc, sf_id, sf_desc, eq_id, eq_desc, p_id, p_desc, observ, finicio, ffin) Tabla: puesto_ciclo (asignación de ciclos a puestos)

get_recurso(*self*, *codigo*=None, *activo*=None)

Devuelve: (id, codigo, descripcion, activo, observaciones) Tabla: recurso (recurso de aplicación) Opciones de filtrado: código del recurso <codigo>, activo <'S','N'>

get_rol(*self*, *activo*=None, *codigo*=None)

Devuelve: (id, codigo, descripcion, observaciones, activo) Tabla: rol (rol maestro de aplicación) Opciones de filtrado: activo <'S','N'>, codigo <código rol>

get_rol_recurso(*self*, *recurso_id*=None, *activo*=None)

Devuelve: (id, rol_id, rol_desc, recurso_id, recurso_desc, ejecucion, lectura, escritura, observaciones, activo Tabla: rol_recurso (relación entre rol y recurso) Opciones de filtrado: recurso_id <id de recurso>, activo <'S','N'>

get_rol_usuario(*self*, *usuario_id*=None, *activo*=None)

Devuelve: (rol_id, usuario_id, observaciones, activo Tabla: rol_usuario (relación entre rol y usuario) Opciones de filtrado: usuario_id <id de usuario>, activo <'S','N'>

get_sp(*self*, *persona_id*, *anno*=None)

Devuelve: (id, persona_id, anno, horas) Tabla: servicios_previos Opciones de filtrado: anno <año>

get_tarea(*self*, *equipo_id*=None, *anno*=None)

Devuelve: (id, p_id, p_cod, p_desc, fecha_inicio, fecha_fin, observ, contrato_id, persona_id, nombre, ape1, ape2, dni, solapado) Tabla: tarea Opciones de filtrado: equipo_id <id equipo>, anno <año en donde está en vigor la asignación>

get_tipo_unit(*self*, *_id*=None, *activo*=None)

Devuelve: (id, codigo, descripcion, activo, observaciones) Tabla: entidad (tipo de unidad organizativa) Opciones de filtrado: id <identificador>, activo <'S','N'>

get_turno(*self*, *activo*=None, *codigo*=None, *id_m*=None, *solo_libres*=False)

```
Devuelve: (id, codigo, descripcion, cuenta_horas, activo,
id_detalle, dia_inicial, dia_final, hora_inicio, hora_fin)
Tablas: turno_master y turno_detail
Opciones de filtrado: activo <'S','N'>, codigo <código turno>,
                      identificador de maestro <id_m>
```

get_unit(*self*, *tipo_unit_id_*, *id_*=None, *codigo*=None, *activo*=None)

Devuelve: (id, tipo_unit_id, codigo, descripcion, activo, direccion, poblacion, cp, provincia, pais, telefono1, telefono2, observaciones, email) Tabla: unit (unidad organizativa) Opciones de filtrado: id_ <id de unit>, activo <'S','N'>

get_unit_dependences(*self*, *tipo_unit*, *id_*, *asig_pend*=False)

Devuelve las dependencias arbóreas de unidades organizativas, según tipo_unit: 1 - Devuelve todos los servicios funcionales que cuelgan del centro físico con id_ pasado como parámetro. 2 - Devuelve todos los equipos de trabajo que cuelgan del servicio con id_ pasado como parámetro. 3 - Devuelve todos los puestos que cuelgan del equipo de trabajo con identificador id_ pasado como parámetro.

get_usuario(*self*, *activo*=None, *nick*=None)

Devuelve: (id, persona_id, nick, password, fecha_alta, fecha_baja, intentos, activo) Tabla: usuario (usuario de aplicación) Opciones de filtrado: nick <login usuario>, activo <'S','N'>

get_usuario_estructura(*self*, *usuario_id*=None, *cf_id*=None, *sf_id*=None, *eq_id*=None, *activo*=None)

Devuelve:(id, usuario_id, usuario, cf_id, cf_cod, cf_desc, sf_id, sf_cod, sf_desc, eq_id, eq_cod, eq_desc, observ, activo) Tabla: usuario_estructura (relación entre usuario y estructura organizativa) Opciones de filtrado: usuario_id, cf_id, sf_id, eq_id, activo

insert_ausencia(*self*, *codigo*, *descripcion*, *cuenta_horas*, *cuenta_dias*, *max_ausencia_anual*, *activar_control_ausencia*, *forzar_ausencia*, *observaciones*, *activo*, *estado_devengo*, *hacer_commit*=True)

Inserción de ausencia Tabla: ausencia

insert_calendario_festivo(*self*, *centro_fisico_id*, *fecha_festivo*, *desc_festivo*, *observ_festivo*, *hacer_commit*=`True`)

Inserción de nueva festividad Tabla: calendario_festivo

insert_cargo(*self*, *codigo*, *descripcion*, *observaciones*, *activo*, *hacer_commit*=`True`)

Inserción de cargo Tabla: cargo

insert_catEq(*self*, *eq_id*, *cat_id*, *hacer_commit*=`True`)

Inserción de categorías en equipos Tabla: categoria_equipo

insert_ciclo(*self*, *codigo_m*, *descripcion_m*, *activo_m*, *cuenta_festivo_m*, *ciclo_d*, *insertar_solo_detalle*=`False`, *id_m*=`None`, *hacer_commit*=`True`)

Inserción de ciclo Tablas: ciclo_master y ciclo_detail

insert_cobertura_equipo(*self*, *fecha_inicio*, *categoria_id*, *lunes*, *martes*, *miercoles*, *jueves*, *viernes*, *sabado*, *domingo*, *fecha_fin*, *hacer_commit*=`True`)

Inserción de cobertura de equipo Tabla: cobertura_equipo

insert_contrato(*self*, *cargo_id*, *fecha_inicio*, *fecha_fin*, *categoria_profesional_id*, *persona_id*, *hacer_commit*=`True`)

Inserción de contrato Tabla: contrato

insert_contrato_ausencia(*self*, *contrato_id*, *aus_id*, *fecha_inicio*, *fecha_fin*, *anno_devengo*, *activo*, *ausencia_parcial*, *hora_inicio*, *hora_fin*, *hacer_commit*=`True`)

Inserción de ausencia en contrato Tabla: contrato_ausencia

insert_cp(*self*, *codigo*, *descripcion*, *activo*, *hacer_commit*=`True`)

Inserción de categoría profesional Tabla: categoria_profesional

insert_estructura(*self*, *cf_id*, *sf_id*, *eq_id*, *p_id*, *observ*, *activo*, *hacer_commit*=`True`)

Inserción de relación organizativa Tablas: estructura

insert_jt(*self*, *cf_id*, *anno*, *total_horas_anual*, *observaciones*, *hacer_commit*=`True`)

Inserción de jornada teórica Tabla: jornada_teorica

insert_persona(*self*, *dni*, *nombre*, *ape1*, *ape2*, *direccion*, *cp*, *poblacion*, *provincia*, *pais*, *tlfno1*, *tlfno2*, *email*, *observaciones*, *sexo*, *fnac*, *hacer_commit*=`True`)

Inserción de nueva persona Tabla: persona

insert_planificacion(*self*, *mes*, *puesto_id*, *anno*, *fecha_inicio*, *ciclo_id*, *fecha_fin*, *total_dias*, *semana*, *lista_dias_mes*, *es_lunes_festivo*, *es_martes_festivo*, *es_miercoles_festivo*, *es_jueves_festivo*, *es_viernes_festivo*, *es_sabado_festivo*, *es_domingo_festivo*, *turno_libre_id*, *hacer_commit*=`True`)

Inserción de nueva planificación Tabla: planificacion

insert_puesto_ciclo(*self*, *ciclo_master_id*, *fecha_inicio*, *puesto_id*, *observaciones*, *fecha_fin*, *hacer_commit*=`True`)

Inserción de relación entre ciclos y puesto de trabajo Tablas: puesto_ciclo

insert_recurso(*self*, *codigo*, *descripcion*, *observaciones*, *activo*, *hacer_commit*=`True`)

Inserción de nuevo recurso Tabla: recurso

insert_rol(*self*, *codigo*, *descripcion*, *observaciones*, *activo*, *hacer_commit*=`True`)

Inserción de nuevo rol Tabla: rol

insert_rol_recurso(*self*, *rol_id*, *recurso_id*, *ejecucion*, *lectura*, *escritura*, *observaciones*, *activo*, *hacer_commit*=`True`)

Inserción de nueva asociación de rol con recurso Tabla: rol_recurso

insert_rol_usuario(*self*, *rol_id*, *usuario_id*, *observaciones*, *activo*, *hacer_commit*=`True`)

Inserción de nueva asociación de rol con usuario Tabla: rol_usuario

insert_sp(*self*, *persona_id*, *anno*, *horas*, *hacer_commit*=`True`)

Inserción de servicios previos Tabla: servicios_previos

insert_tarea(*self*, *puesto_id*, *fecha_inicio*, *fecha_fin*, *observaciones*, *contrato_id*, *solapado*, *hacer_commit*=`True`)

Inserción de tarea (asignación) Tabla: tarea

insert_turno(*self*, *id_m*, *codigo_m*, *descripcion_m*, *cuenta_horas_m*, *activo_m*, *id_d*, *dia_inicial_d*, *dia_final_d*, *hora_inicio_d*, *hora_fin_d*, *hacer_commit*=`True`)

Inserción de turno Tablas: turno_master y turno_detail

insert_unit(*self*, *tipo_unit_id*, *codigo*, *descripcion*, *activo*, *direccion*, *poblacion*, *cp*, *provincia*, *pais*, *telefono1*, *telefono2*, *observaciones*, *email*, *hacer_commit*=`True`)

Inserción de nueva unidad organizativa Tabla: unit

insert_usuario(*self*, *nick*, *passwd*, *fecha_alta*, *intentos*, *activo*, *hacer_commit*=`True`)

Inserción de nuevo usuario Tabla: usuario

insert_usuario_estructura(*self*, *usuario_id*, *cf_id*, *sf_id*, *eq_id*, *observaciones*, *activo*, *hacer_commit*=`True`)

Inserción de nueva asociación de usuario con estructura Tabla: usuario_estructura

is_fechas_correctas_puesto_ciclo(*self*)

Devuelve True si fecha_desde es menor o igual a fecha_hasta y False en caso contrario en puesto_ciclo

set_usuario_passwd(*self*, *usuario_id*, *quitarIntento*=`False`, *resetearIntento*=`False`, *numeroIntentos*=`0`)

Actualización: intentos Tabla: usuario Filtros de actualización: usuario_id <identificador> Opciones de filtrado: quitarIntento <True: disminuye en 1 el número de intentos, False: Nada> resetearIntento <True: Resetea intentos a numeroIntentos, False: Nada>

update_ausencia(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: ausencia

update_basica(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: basica

update_calendario_festivo(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: calendario_festivo

update_cargo(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: cargo

update_ciclo(*self*, *campos*, *hacer_commit*=`True`)

Actualización de ciclo Tabla: ciclo_master y ciclo_detail

update_cobertura_equipo(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: cobertura_equipo

update_contrato(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: contrato

update_contrato_ausencia(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: contrato_ausencia

update_cp(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: categoria_profesional

update_estructura(*self*, *campos*, *hacer_commit*=`True`)

Actualización de relación organizativa Tabla: estructura

update_jt(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: jornada_teorica

update_persona(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: persona

update_puesto_ciclo(*self*, *campos*, *hacer_commit*=`True`)

Actualización de relación entre ciclos y puesto de trabajo Tabla: puesto_ciclo

update_recurso(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: recurso

update_rol(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: rol

update_rol_recurso(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: rol_recurso

update_rol_usuario(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: rol_usuario

update_sp(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: servicios_previos

update_tarea(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: tarea

update_turno(*self*, *campos*, *hacer_commit*=`True`)

Actualización de turno Tabla: turno_master y turno_detail

update_unit(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: unit

update_usuario(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: usuario

update_usuario_estructura(*self*, *_id*, *campo*, *valor*, *hacer_commit*=`True`)

Actualización de campo Tabla: usuario_estructura

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

6.2.2 Properties

Name	Description
Inherited from object	
___class___	

7 Module estructura

7.1 Variables

Name	Description
__package__	**Value:** `None`

7.2 Class Estructura

object
└ **estructura.Estructura**

Clase Estructura (estructura organizativa)

7.2.1 Methods

__init__(*self*, *conn*)

Constructor

Overrides: object.__init__

get_estructura(*self*, *puesto_id*=`None`, *activo*=`None`)

Devuelve: (cf_id, cf_cod, cf_desc, sf_id, sf_cod, sf_desc, eq_id, eq_cod, eq_desc, p_id, p_cod, p_desc, observ, activo) Tabla: estructura (estructura organizativa)

set_estructura(*self*, *datos*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

7.2.2 Properties

Name	Description
Inherited from object	
__class__	

7.3 Class Unit

object
└ **estructura.Unit**

Clase unit (unidad organizativa)

7.3.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_centro_fisico(*self*, *id_*=`None`, *codigo*=`None`, *es_activo*=`None`)

Devuelve: (id, codigo, descripcion, activo, direccion, poblacion, cp, provincia, pais, telefono1, telefono2, observaciones, email) Tabla: unit (unidad organizativa) Opciones de filtrado: id_ <id de centro físico>, activo <'S','N'>

get_equipo(*self*, *id_*=`None`, *codigo*=`None`, *es_activo*=`None`, *sf_id*=`None`, *asig_pend*=`False`)

Devuelve: (id, codigo, descripcion, activo, direccion, poblacion, cp, provincia, pais, telefono1, telefono2, observaciones, email) Tabla: unit (unidad organizativa) Opciones de filtrado: id_ <id de equipo>, activo <'S','N'>

get_puesto(*self*, *id_*=`None`, *codigo*=`None`, *es_activo*=`None`, *eq_id*=`None`, *asig_pend*=`False`)

Devuelve: (id, codigo, descripcion, activo, direccion, poblacion, cp, provincia, pais, telefono1, telefono2, observaciones, email) Tabla: unit (unidad organizativa) Opciones de filtrado: id_ <id de puesto>, activo <'S','N'>

get_servicio(*self*, *id_*=`None`, *codigo*=`None`, *es_activo*=`None`, *cf_id*=`None`, *asig_pend*=`False`)

Devuelve: (id, codigo, descripcion, activo, direccion, poblacion, cp, provincia, pais, telefono1, telefono2, observaciones, email) Tabla: unit (unidad organizativa) Opciones de filtrado: id_ <id de servicio>, activo <'S','N'>

get_unit(*self*, *tipo_unit*, *id_*=`None`, *codigo*=`None`, *es_activo*=`None`)

Devuelve: (id, codigo, descripcion, activo, direccion, poblacion, cp, provincia, pais, telefono1, telefono2, observaciones, email) Tabla: unit (unidad organizativa) Opciones de filtrado: tipo_unit <tipo de unidad organizativa: 1 - centro físico, 2 - servicio, 3- categoría o equipo 4 - puesto>, id_ <id de unit>, activo <'S','N'>, codigo <codigo de unit>

get_unit_dependences(*self*, *tipo_unit*, *id_*, *asig_pend*=`False`)

Devuelve las dependencias arbóreas de unidades organizativas, según tipo_unit: 1 - Devuelve todos los servicios funcionales que cuelgan del centro físico con id_ pasado como parámetro. 2 - Devuelve todos los equipos de trabajo que cuelgan del servicio con id_ pasado como parámetro. 3 - Devuelve todos los puestos que cuelgan del equipo de trabajo con identificador id_ pasado como parámetro.

set_unit(*self*, *datos*, *opcion*=`None`)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

7.3.2 Properties

Name	Description
Inherited from object	
___class___	

8 Module jornada_teorica

8.1 Variables

Name	Description
___package___	**Value:** `None`

8.2 Class JornadaTeorica

object
└ **jornada_teorica.JornadaTeorica**

8.2.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_jt(*self*, *centro_fisico_id*=`None`, *anno*=`None`)

Devuelve: (id, cf_id, cf_cod, cf_desc, anno, total_horas_anual, observaciones) Tabla: jornada_teorica Opciones de filtrado: Id de centro físico <centro_fisico_id>, año de calendario laboral <anno>

set_jt(*self*, *datos*)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

8.2.2 Properties

Name	Description
Inherited from object	
___class___	

9 Module planificacion

9.1 Variables

Name	Description
__package__	**Value:** `None`

9.2 Class CoberturaEquipo

object
└
planificacion.CoberturaEquipo

Clase cobertura de servicio para equipo de trabajo

9.2.1 Methods

__init__(*self*, *conn*)

Constructor

Overrides: object.__init__

get_cobertura_equipo(*self*, *equipo_id*=`None`, *fecha*=`None`)

Devuelve: (id, fecha_inicio, categoria_id, lunes, martes, miercoles, jueves, viernes, sabado, domingo, fecha_fin) Opciones de filtrado: equipo_id <id equipo>, fecha <fecha en donde la cobertura es efectiva>

set_cobertura_equipo(*self*, *datos*)

Actualización de datos. Tabla: cobertura_equipo

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

9.2.2 Properties

Name	Description
Inherited from object	

continued on next page

Name	Description
___class___	

9.3 Class CoberturaServicio

object
└
planificacion.CoberturaServicio

Cobertura de servicio. Información de presencias en planificación diaria

9.3.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_coberturaServicio(*self*, *equipo_id*, *mes*, *anno*)

Devuelve: (dia_trab, dia_semana, descuadre, info) Tabla: cobertura_equipo, tarea, v_planificacion Esta query da las coberturas de servicio en planificación diaria, teniendo en cuenta turnos que cuenten o no horas, ausencias que cuenten ó no presencias (días) y cambios de turnos. Además soporta diferentes coberturas para un mismo mes.

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

9.3.2 Properties

Name	Description
Inherited from object	
___class___	

9.4 Class Planificacion

object
└ **planificacion.Planificacion**

9.4.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

festivos(*self*, *l*, *m*, *x*, *j*, *v*, *s*, *d*, *total_dias*, *mes*, *anno*, *equipo_id*)

Devuelve estructura de festivos

get_planificacion(*self*, *anno*, *mes*, *puesto_id*=`None`, *fecha_inicio*=`None`, *equipo_id*=`None`, *visualizacion*=`0`)

Devuelve: () Tabla: planificacion (planificaciones de ciclos asociados a puestos)

set_planificacion(*self*, *puesto_id*, *ciclo_id*, *fecha_inicio*, *fecha_fin*, *turno_libre_id*, *semana*=`1`)

Crea una planificación

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

9.4.2 Properties

Name	Description
Inherited from object	
___class___	

9.5 Class PlanificacionDiaria

object
└ **planificacion.PlanificacionDiaria**

9.5.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_ausencias_trab(*self*, *fini_t*, *ffin_t*, *contrato_id*)

Se recuperan las ausencias del trabajador

get_cambios_turnos(*self*, *fini_t*, *ffin_t*, *tarea_id*)

Se recuperan los posibles cambios de turnos de la planificación diaria para una tarea entre un rango de fechas de tarea.

get_planificacion_diaria(*self*, *anno*, *mes*, *puesto_id*=`None`, *fecha_inicio*=`None`, *equipo_id*=`None`)

Devuelve: La planilla Tablas: planificacion, tarea (planilla)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

9.5.2 Properties

Name	Description
Inherited from object	
___class___	

9.6 Class PuestoCiclo

object
└ **planificacion.PuestoCiclo**

9.6.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_puesto_ciclo(*self*, *puesto_id*=`None`)

Devuelve: (ciclo_id, ciclo_desc, cf_id, cf_desc, sf_id, sf_desc, eq_id, eq_desc, p_id, p_desc, observ, finicio, ffin, semana turno_libre_id) Tabla: puesto_ciclo (asignación de ciclos a puestos)

set_puesto_ciclo(*self*, *datos*)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

9.6.2 Properties

Name	Description
Inherited from object	
___class___	

10 Module rrhh

10.1 Variables

Name	Description
___package___	**Value:** `None`

10.2 Class Ausencia

object
└ **rrhh.Ausencia**

Clase ausencia

10.2.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_ausencia(*self*, *id_*=`None`, *codigo*=`None`, *activo*=`None`)

Devuelve: (id, codigo, descripcion, cuenta_horas, cuenta_dias, max_ausencia_anual, activar_control_ausencia, forzar_ausencia, observaciones, activo, estado_ausencia) Opciones de filtrado: id_ <id ausencia,>, codigo <cód. ausencia>, activo <'S', 'N'>

set_ausencia(*self*, *datos*)

Actualización de datos. Tabla: ausencia

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.2.2 Properties

Name	Description
Inherited from object	

continued on next page

Name	Description
___class___	

10.3 Class Cargo

object
└
rrhh.Cargo

Clase cargo de contrato para puesto de un trabajador

10.3.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_cargo(*self*, *id_*=`None`, *activo*=`None`)

Devuelve: (id, codigo, descripcion, observaciones, activo) Opciones de filtrado: id_ <id cargo>, activo <'S', 'N'>

set_cargo(*self*, *datos*)

Actualización de datos. Tabla: cargo

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.3.2 Properties

Name	Description
Inherited from object	
___class___	

10.4 Class CategoriaEquipo

object
└ **rrhh.CategoriaEquipo**

Clase categorías en equipos de trabajo

10.4.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_catEq(*self*, *eq_id*=`None`, *cat_id*=`None`)

Devuelve: (id, eq_id, eq_cod, eq_desc, cat_id, cat_cod, cat_desc) Opciones de filtrado: eq_id <id equipo>, cat_idactivo <id cat.prof.>

set_catEq(*self*, *datos*)

Actualización de datos. Tabla: categoria_equipo

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.4.2 Properties

Name	Description
Inherited from object	
___class___	

10.5 Class CategoriaProfesional

object
└ **rrhh.CategoriaProfesional**

Clase categoría profesional asociada a un contrato

10.5.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_cp(*self*, *id_*=`None`, *activo*=`None`)

Devuelve: (id, codigo, descripcion, activo) Opciones de filtrado: id_ <id cat.prof.>, activo <'S', 'N'>

set_cp(*self*, *datos*)

Actualización de datos. Tabla: categoria_profesional

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.5.2 Properties

Name	Description
Inherited from object	
___class___	

10.6 Class ContratoAusencia

object
└
rrhh.ContratoAusencia

Clase ausencia asociada a un contrato

10.6.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

esValido(*self*)

Método de inicialización de mecanismo de validación de ausencias asociadas a contrato

setContrato(*self*, *contrato_id*)

Asocia identificador de contrato con ausencias

setModificaciones(*self*, *celdas*, *filas_a_eliminar*, *filas_a_insertar*)

Se cargan celdas a modificar, filas a eliminar y filas a insertar de las ausencias asociadas al contrato

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.6.2 Properties

Name	Description
Inherited from object	
___class___	

10.7 Class ContratoAusenciaPersona

object
└
rrhh.ContratoAusenciaPersona

Clase ContratoAusenciaPersona que relaciona las ausencias con un contrato

10.7.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_contrato_ausencia(*self*, *contrato_id*, *activo*=None)

Recupera todas las ausencias de un contrato Devuelve: (id, contrato_id, aus_id, aus_cod, aus_desc, fecha_inicio, fecha_fin, anno_devengo, activo) Opciones de filtrado: contrato_id <id de contrato>, activo <'S', 'N'>

set_contrato_ausencia(*self*, *datos*)

Actualización de datos. Tabla: contrato_ausencia

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.7.2 Properties

Name	Description
Inherited from object	
___class___	

10.8 Class ContratoPersona

object
└ **rrhh.ContratoPersona**

Clase ContratoPersona que relaciona a una persona con sus contratos

10.8.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_contrato(*self*, *persona_id*)

Recupera todos los contratos de una persona Devuelve: (id, cargo_id, cargo_cod, cargo_desc, fecha_inicio, fecha_fin, cp_id, cp_cod, cp_desc, persona_id) Opciones de filtrado: persona_id <id de persona>

set_contrato(*self*, *datos*)

Actualización de datos. Tabla: contrato

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.8.2 Properties

Name	Description
Inherited from object	
___class___	

10.9 Class Persona

object
└ **rrhh.Persona**

Clase persona que cotiene todos los datos personales de un trabajador

10.9.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_persona(*self*, *id_*=`None`, *dni*=`None`, *busqueda*=`False`)

Devuelve: (id, dni, nombre, ape1, ape2, direccion, cp, poblacion, provincia, pais, tlfno1, tlfno2, email, observaciones) Opciones de filtrado: id_ <id de persona>, dni <dni de persona>

set_persona(*self*, *datos*)

Actualización de datos. Tabla: persona

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(),

___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.9.2 Properties

Name	Description
Inherited from object	
___class___	

10.10 Class ServiciosPrevios

object
└
rrhh.ServiciosPrevios

Clase para gestión de servicios previos de un trabajador

10.10.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_sp(*self*, *persona_id*, *anno*=`None`)

Devuelve: (persona_id, anno, horas) Opciones de filtrado: anno <año>

set_sp(*self*, *datos*)

Actualización de datos. Tabla: servicios_previos

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

10.10.2 Properties

Name	Description
Inherited from object	

continued on next page

Name	Description
___class___	

11 Module seguridad

11.1 Variables

Name	Description
___package___	**Value:** `None`

11.2 Class Login

object
└
seguridad.Login

Clase de acceso a la aplicación

11.2.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

existe_login(*self*, *usuario*, *passwd*)

Devuelve <True, msj> si login/passwd es correcto y <False, msj> en caso contrario.

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

11.2.2 Properties

Name	Description
Inherited from object	
___class___	

11.3 Class Recurso

object
└ **seguridad.Recurso**

Clase Recurso

11.3.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_recurso(*self*, *codigo*=`None`, *es_activo*=`None`)

Devuelve: (id, codigo, descripcion, activo, observaciones) Tabla: recurso (recurso de aplicación) Opciones de filtrado: código del recurso <codigo>, activo <'S','N'> Si no se parametrizan opciones de filtrado por defecto devuelve todos los recursos.

set_recurso(*self*, *datos*)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

11.3.2 Properties

Name	Description
Inherited from object	
___class___	

11.4 Class Rol

object
└ **seguridad.Rol**

Clase Rol

11.4.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_rol(*self*, *codigo_rol*=`None`, *es_activo*=`None`)

Devuelve:(id, codigo, descripcion, observaciones, activo) Tabla: rol (rol de aplicación) Opciones de filtrado: activo <'S','N'>, codigo <código rol>

set_rol(*self*, *datos*)

Actualización de datos. Tabla: rol

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

11.4.2 Properties

Name	Description
Inherited from object	
___class___	

11.5 Class RolRecurso

object
└ **seguridad.RolRecurso**

Clase Rol - Recurso

11.5.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_rol_recurso(*self*, *rec_id*=None, *es_activo*=None)

Devuelve:(rol_id, recurso_id, ejecucion, lectura, escritura, observaciones, activo) Tabla: rol_recurso (relación entre recurso y sus roles asociados) Opciones de filtrado: activo <'S','N'>, rec_id <id de recurso>

set_rol_recurso(*self*, *datos*)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

11.5.2 Properties

Name	Description
Inherited from object	
___class___	

11.6 Class RolUsuario

object
└ **seguridad.RolUsuario**

Clase Rol - Usuario

11.6.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_rol_usuario(*self*, *usu_id*=None, *es_activo*=None)

Devuelve:(rol_id, usuario_id, observaciones, activo) Tabla: rol_usuario (relación entre usuario y sus roles asociados) Opciones de filtrado: activo <'S','N'>, usu_id <id de usuario>

set_rol_usuario(*self*, *datos*)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

11.6.2 Properties

Name	Description
Inherited from object	
___class___	

11.7 Class Usuario

object
└
seguridad.Usuario

Clase Usuario

11.7.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_usuario(*self*, *usuario*=`None`, *es_activo*=`None`)

Devuelve (id, persona_id, nick, password, fecha_alta,fecha_baja, intentos, activo) Opciones de filtrado: nick <login usuario>, activo <'S','N'> Si no se parametrizan opciones de filtrado por defecto devuelve todos los usuarios.

set_usuario(*self*, *datos*)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

11.7.2 Properties

Name	Description
Inherited from object	
___class___	

11.8 Class UsuarioEstructura

object
└
seguridad.UsuarioEstructura

Clase Usuario - Estructura

11.8.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

get_usuario_estructura(*self*, *usuario_id*=`None`, *cf_id*=`None`, *sf_id*=`None`, *eq_id*=`None`, *activo*=`None`)

Devuelve:(id, usuario_id, usuario, cf_id, cf_cod, cf_desc, sf_id, sf_cod, sf_desc, eq_id, eq_cod, eq_desc, observ, activo) Tabla: usuario_estructura (relación entre usuario y estructura organizativa) Opciones de filtrado: usuario_id, cf_id, sf_id, eq_id, activo

set_usuario_estructura(*self*, *datos*)

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

11.8.2 Properties

Name	Description
Inherited from object	
___class___	

12 Module services

12.1 Functions

api_ausencia()

api_ausencia_update()

api_balance()

api_basica()

api_basica_update()

api_buscar_persona()

api_buscar_trabajadores_asignar()

api_cambio_turno_update()

api_cargo()

api_cargo_update()

api_categoria_equipo()

api_categoria_equipo_update()

api_categoria_profesional()

api_categoria_profesional_update()

api_centro_fisico()

api_centro_fisico_update()

api_ciclo()

api_ciclo_semana()

api_ciclo_update()

api_cobertura_equipo()

api_cobertura_equipo_update()

api_cobertura_servicio()

api_contrato()

api_contrato_ausencia()

api_contrato_ausencia_update()

api_contrato_update()

api_equipo()

api_equipo_update()

api_estructura()

api_estructura_update()

api_festivos()

api_festivos_update()

api_jornada_teorica()

api_jornada_teorica_update()

api_message_login()

api_persona()

api_persona_update()

api_planificacion()

api_planificacion_diaria()

api_planificacion_update()

api_puesto()

api_puesto_ciclo()

api_puesto_ciclo_update()

api_puesto_update()

api_recurso()

api_recurso_update()

api_rol()

api_rol_recurso()

api_rol_recurso_update()

api_rol_update()

api_rol_usuario()

api_rol_usuario_update()

api_root()

api_servicio()

api_servicio_update()

api_servicios_previos()

api_servicios_previos_update()

api_tarea()

api_tarea_update()

api_turno()

api_turno_update()

api_usuario()

api_usuario_estructura()

api_usuario_estructura_update()

api_usuario_update()

comprobar_usuario_bd()

Se comprueba si existe usuario en base de datos. Por defecto si no hay usuarios, se crea usuario admin, con contraseña admin

info()

info_reports()

reports_usuario()

response_(*json_data*)

Función que incluye cabecera a la respuesta

12.2 Variables

Name	Description
___BACKEND___	**Value:** `None`
___CONN___	**Value:** `None`
___package___	**Value:** `None`
app	**Value:** `<Flask 'services'>`
request	**Value:** `<LocalProxy unbound>`

13 Module turnos

13.1 Variables

Name	Description
__package__	**Value:** `None`

13.2 Class Ciclo

object
└ **turnos.Ciclo**

Clase Ciclo

13.2.1 Methods

__init__(*self*, *conn*)

Constructor

Overrides: object.__init__

get_ciclo(*self*, *id_ciclo*=`None`, *es_activo*=`None`, *semana*=`False`)

Devuelve: (id, codigo, descripcion, cuenta_festivo, activo, id_detalle, numero_dia, turno_master_id Tablas: ciclo_master y ciclo_detail Opciones de filtrado: activo <'S','N'>, codigo <código ciclo>

set_ciclo(*self*, *datos*)

Actualización de datos. Tablas: ciclo_master y ciclo_detail

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

13.2.2 Properties

Name	Description
Inherited from object	

continued on next page

Name	Description
___class___	

13.3 Class Turno

object
└ **turnos.Turno**

Clase Turno

13.3.1 Methods

___init___(*self*, *conn*)

Constructor

Overrides: object.___init___

formarTurno(*self*, *turno*)

get_turno(*self*, *codigo_turno*=`None`, *es_activo*=`None`, *solo_libres*=`False`)

Devuelve: (id, codigo, descripcion, activo, id_detalle, dia_inicial dia_final, hora_inicio, hora_fin) Tablas: turno_master y turno_detail Opciones de filtrado: activo <'S','N'>, codigo <código turno>

set_turno(*self*, *datos*)

Actualización de datos. Tablas: turno_master y turno_detail

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

13.3.2 Properties

Name	Description
Inherited from object	
___class___	

14 Module w_model

14.1 Variables

Name	Description
FRIDAY	**Value:** `4`
MONDAY	**Value:** `0`
SATURDAY	**Value:** `5`
SUNDAY	**Value:** `6`
THURSDAY	**Value:** `3`
TUESDAY	**Value:** `1`
WEDNESDAY	**Value:** `2`
__package__	**Value:** `None`

14.2 Class w_calendar

object
└
w_model.w_calendar

Clase calendario

14.2.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

diferenciaDias(*self*, *fecha1*, *fecha2*, *festivos*=`[]`)

Devuelve una tupla con la diferencia entre dos fechas: (diferencia de días totales (días normales, festivos y fines de semana), diferencia de días sin contar días festivos, sábados y domingos. diferencia de días sin contar los días festivos)

esFestivo(*self*, *festivo*, *festivos*)

Devuelve True si es festivo y False en caso contrario

getCalendario(*self*, *anno*, *festivos*, *lunes_festivo*=`False`, *martes_festivo*=`False`, *miercoles_festivo*=`False`, *jueves_festivo*=`False`, *viernes_fetivo*=`False`, *sabado_festivo*=`True`, *domingo_festivo*=`True`)

```
Devuelve estructura:
 ret = {'calendario' : c,
       'festivos' : f}
de calendario con festivos
```

getDiaSemana(*self*, *anno*, *mes*, *dia*)

Devuelve día se la semana 0..6 <-> Lunes.. Domingo

getMes(*self*, *calendario*, *mes*, *por_pantalla*=`True`)

Devuelve el mes del calendario

getNombreDia(*self*, *anno*, *mes*, *dia*, *mayus*=`False`, *abrev*=`False`)

Devuelve nombre de día de la semana y sigla

getNombreMes(*self*, *mes*, *mayus*=`False`)

Devuelve el nombre del mes

newFestivos(*self*, *anno*, *calendario*, *festivos*=`[]`, *lunes_festivo*=`False`, *martes_festivo*=`False`, *miercoles_festivo*=`False`, *jueves_festivo*=`False`, *viernes_festivo*=`False`, *sabado_festivo*=`True`, *domingo_festivo*=`True`)

Devuelve estructura de festivos

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

14.2.2 Properties

Name	Description
Inherited from object	
___class___	

14.3 Class w_ciclo

object
└ **w_model.w_ciclo**

Clase ciclo

14.3.1 Methods

___init___(*self*)

Constructor

Overrides: object.___init___

cuentaFestivo(*self*)

Devuelve True si el ciclo cuenta festivos como días de trabajo y False en caso contrario

getCicloSemanalPorCodTurno(*self*, *ciclo*=[])

Método que devuelve una estructura con la planificación de códigos de turnos, por semanas

getPlanilla(*self*)

Devuelve la planificación del ciclo sobre el calendario

newCiclo(*self*, *codigo*, *descripcion*, *cuenta_festivo*, *ciclo*, *por_pantalla*=True)

Crea un ciclo

planificarCiclo(*self*, *calendario*, *semana_inicio*, *fecha_desde*, *fecha_hasta*, *turno_libre*, *por_pantalla*=True)

Planifica una ciclo y la expande a lo largo de un calendario. Devuelve el número de última semana planificada.

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

14.3.2 Properties

Name	Description
Inherited from object	
___class___	

14.4 Class w_planificacion

object
└
w_model.w_planificacion

Clase planificación

14.4.1 Methods

___init___(*self*)

Constructor

Overrides: object.___init___

getFechasPlanificacion(*self*)

Devuelve tupla <fecha_inicio, fecha_fin> de planificación

getUltimaSemanaPlanificada(*self*)

Devuelve la última semana planificada del ciclo

newPlanificacion(*self*, *fecha_desde*, *fecha_hasta*, *ciclo*, *turno_libre*, *semana_inicio*=`1`, *lunes_festivo*=`False`, *martes_festivo*=`False`, *miercoles_festivo*=`False`, *jueves_festivo*=`False`, *viernes_festivo*=`False`, *sabado_festivo*=`True`, *domingo_festivo*=`True`, *festivos*=`[]`, *por_pantalla*=`True`)

Creación de una planificación para un año. Devuelve <True, planilla, [fecha_inicio, fecha_fin, ultima_semana_planificada]> si todo es correcto y <False, mensaje, None> si hubo error. Parámetros: fecha_desde: Tupla (año, mes, dia) fecha_hasta: Tupla (año, mes, dia) ciclo: Ciclo tipo w_ciclo turno_libre: Código de turno que no cuenta horas para usar cuando se tenga que cambiar un turno por una actividad que no cuente horas, como Libres o Salientes de Noche. semana_inicio: Nº de semana por la que se quiere planificar sabado_festivo: El sábado se cuenta como día festivo. festivos: Lista de festivos [(año, mes, dia), ..., (año, mes, dia)] por_pantalla: True pintará la planilla en pantalla y False no lo hará

particionarPlanificacion(*self*, *planilla*)

Particionamiento de planilla en meses. Se devuelve una lista, donde cada elemento es un mes, según los elementos de la planilla pasada como parámetro

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

14.4.2 Properties

Name	Description
Inherited from object	
___class___	

14.5 Class w_puesto

object
└
w_model.w_puesto

Clase puesto de trabajo

14.5.1 Methods

___init___(*self*)

Constructor

Overrides: object.___init___

add_planificacion(*self*, *planilla*, *extra*, *chequearSolapamiento*=`True`)

Añadir una planificación a un puesto. Se pasa por parámetro la planilla, el identificador del ciclo, y las fechas de inicio y fin junto con la última semana de planificación del ciclo

diasMes(*self*, *anno*, *mes*)

Devuelve el número de días de un mes

get_planificacion_mensual(*self*, *anno*, *mes*, *por_pantalla*=`True`)

Devuelve la planificación mensual de un mes dado, dentro de un año, para el puesto. Devuelve todos los días del mes, que puede incluir más de una planificación (más de un ciclo). Los días que no se hayan planificado se incluirá un nulo.

get_planificaciones(*self*)

haySolapamiento(*self*, *f1_inicio*, *f1_fin*, *f2_inicio*, *f2_fin*)

Devuelve True si hay solapamiento entre el rango de fechas f1 y f2 y False en caso contrario. Cada uno de los parámetros tiene que tener el formato (año, mes, dia)

ultimaSemanaPlanificacion(*self*)

Devuelve la última semana de la planificación más reciente de entre todas las planificaciones que tiene el puesto

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(), ___str___(), ___subclasshook___()

14.5.2 Properties

Name	Description
Inherited from object	
___class___	

14.6 Class w_turno

object
└ **w_model.w_turno**

Clase turno

14.6.1 Methods

___init___(*self*)

x.___init___(...) initializes x; see help(type(x)) for signature

Overrides: object.___init___ extit(inherited documentation)

esTurnoLibre(*self*, *turno*)

Devuelve True si el turno es Libre (no cuenta horas) y False en caso contrario

existeSolapamientoTurnos(*self*, *patron*=`None`)

Devuelve <True, msj> si hay turnos solapados y <False, msj> en caso contrario, para los turnos agregados dentro de la clase

extenderHorarioTurno(*self*, *turno*, *dia_inicial*, *dia_final*, *hora_inicio*, *hora_fin*, *actualizarTurno*=`True`)

Extiende el horario de un turno, devolviendo <True, msj> si todo ha ido bien y <False, msj> si hubo algún tipo de error

getHoraPatron(*self*, *patron*)

Devuelve el tiempo total de un patrón <horas, minutos, segundos>

getHoraTurno(*self*, *turno*)

Devuelve el tiempo total de un turno <horas, minutos, segundos>

getTurnoByCodigo(*self*, *codigo*)

Devuelve un <True, turno> a partir de su código pasado como parámetro y <False, None> si el turno no existe

getTurnos(*self*, *patron*=`None`)

Devuelve lista con los turnos de la clase. Parámetros: patron = None, devuelve todos los turnos por orden de creación. patron = (<codigo>, <codigo>, <codigo>, ...) devuelve una lista ordenada con los códigos definidos. Por ejemplo, se podría devolver getTurnos(('M','M','T','N','-') para devolver un patrón de un posible ciclo. Devuelve los turnos en ese orden con respecto a sus códigos de turno.

modificarTurno(*self*, *turno*)

Cambia el turno "turno" por el nuevo valor de "turno". Devuelve <False, msj> si no se pudo modificar el turno y <True, ""> si se modificó correctamente.

newTurno(*self*, *codigo*, *descripcion*, *horario*, *cuenta_horas*, *activo*='S', *_id*=None, *chequearRepeticionCodigo*=True)

```
Creación de un nuevo Turno. Devuelve <True, ""> si todo ha ido bien
y <False, mensaje de error> si ha habido fallo. Parámetros:

codigo: Código unívoco identificativo del turno.
descripcion: Descripción del turno.
cuenta_horas: <'S'> El turno cuenta las horas. <'N'> el turno no cuenta
horas, independientemente de lo que tenga en horario. Para definir turnos
que no cuenten horas, como libres o salientes de noche, usar este valor.
activo: <'S'> si el turno está activo y <'N'> si no está activo.
horario: [
          [{'id' : 1},
           {'dia_inicial' : X},
           {'dia_final' : X},
           {'hora_inicio': 'hh:mm'},
           {'hora_fin' : 'hh:mm'}],
           ...
           ...
          [{'id' : n},
           {'dia_inicial' : X},
           {'dia_final' : X},
           {'hora_inicio': 'hh:mm'},
           {'hora_fin' : 'hh:mm'}]
         ]
         donde X = -1 si la franja horaria nace en el día anterior.
               X =  0 si la franja horaria nace en el día actual.
               X =  1 si la franja horaria nace en el día siguiente.
```

sonTurnosSolapados(*self*, *ti*, *tj*)

Método que devuelve <True, msj> si el turno ti se solapa en el tiempo con el turno t2, donde el orden es j = i + 1. Devuelve <False, msj> si no hay solapamiento

Inherited from object

___delattr___(), ___format___(), ___getattribute___(), ___hash___(), ___new___(), ___reduce___(), ___reduce_ex___(), ___repr___(), ___setattr___(), ___sizeof___(),

___str___(), ___subclasshook___()

14.6.2 Properties

Name	Description
Inherited from object	
___class___	

Index