

Silva, Lilian Leonor

# Sigarhu (Sistema de Gestión y Administración de Recursos Humanos)

2019

*Instituto: Ingeniería y Agronomía*

*Carrera: Ingeniería en Informática*



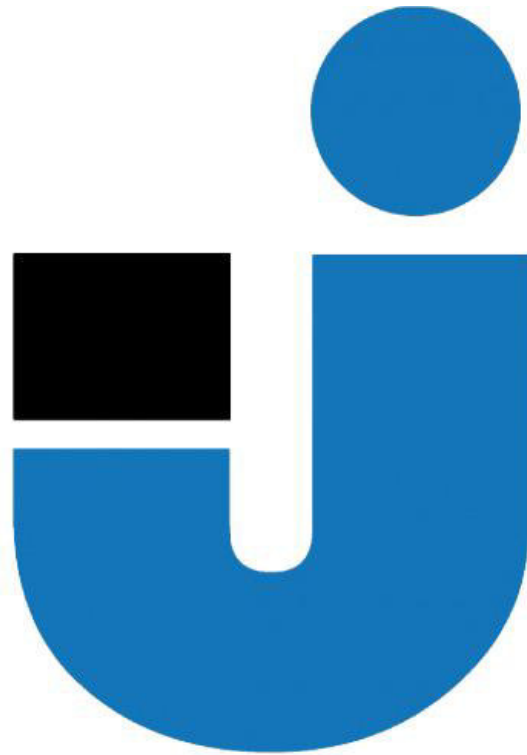
Esta obra está bajo una Licencia Creative Commons Argentina.  
Atribución - No Comercial - Compartir Igual 4.0  
<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Documento descargado de RID - UNAJ Repositorio Institucional Digital de la Universidad Nacional Arturo Jauretche

*Cita recomendada:*

*Silva, L.L. (2019) Sigarhu: sistema de gestión y administración de recursos humanos [informe de la Práctica Profesional Supervisada] Universidad Nacional Arturo Jauretche.*

Disponible en RID - UNAJ Repositorio Institucional Digital UNAJ <https://biblioteca.unaj.edu.ar/rid-unaj-repositorio-institucional-digital-unaj>



Universidad Nacional  
**ARTURO JAURETCHE**

**Informe Final**

**Lilian Leonor Silva**

**Universidad Nacional Arturo Jauretche**

**Instituto de Ingeniería y Agronomía**

**Ingeniería Informática**

**PRÁCTICA PROFESIONAL SUPERVISADA (PPS)**  
**Informe Final**

**DATOS DEL ESTUDIANTE**

Apellido y Nombres: Silva, Lilian Leonor

DNI: 30974662

Nº de Legajo: 3960

Correo electrónico: [silvalilian662@gmail.com](mailto:silvalilian662@gmail.com)

Cantidad de materias aprobadas al comienzo de la PPS: cuarenta y tres (43)

PPS enmarcada en artículo Nº 7 de la Resolución del Consejo Superior 103/16, inciso b.

**DOCENTE SUPERVISOR**

Apellido y Nombres: prof.dr. Conde, Sergio

Correo electrónico: [drcondesergio@gmail.com](mailto:drcondesergio@gmail.com)

**DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ**

Apellido y Nombres: prof.dra. Ferrari, Mariela

Correo electrónico: [mariela\\_c\\_ferrari@hotmail.com](mailto:mariela_c_ferrari@hotmail.com)

**DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS**

Nombre o Razón Social: Ministerio de Transporte de la Nación

Dirección: Paseo Colon Nº 315

Teléfono: 5289-3800 int. 4096/4097

Sector: Dirección de Integración de Sistemas

**TUTOR DE LA ORGANIZACIONAL**

Apellido y Nombres: Lic. PerezArrieu, Federico

Correo electrónico: [fperezarrieu@transporte.gob.ar](mailto:fperezarrieu@transporte.gob.ar)

**FIRMA DEL COORDINADOR DE LA CARRERA**

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Contenido

1. Introducción	6
1.1 Presentación .....	6
1.2 Objetivos .....	7
1.3 Contexto .....	7
1.4 Planteamiento de problema .....	10
2. Forma de trabajo	12
3. Análisis y relevamiento	14
3.1 Instituciones involucradas en el sistema .....	14
3.2 Usuarios del sistema .....	16
4. Tareas a ejecutar	19
4.1 Bloque de antigüedad .....	19
4.2 Bloque de administración .....	20
4.3 Bloque de embargos .....	22
4.4 Bloque de anticorrupción.....	23
4.5 Observaciones de gestión .....	24
5. Diseño e implementación	25
5.1 Ambientes de trabajo.....	25
5.2 Tecnologías a utilizar .....	27
PHP (Hypertext Preprocessor) .....	27
Programación Orientada a Objetos (POO/OOP).....	28
<i>JavaScript</i> .....	29
<i>AJAX</i> .....	32

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

SQL .....	32
MySQL.....	32
MVC (Modelo, Vista y Controlador) .....	33
5.3 Herramientas .....	36
<i>Sublime Text</i> .....	36
<i>MySQL Workbench</i> .....	37
Poncho .....	37
<i>Framework</i> del Ministerio de Transporte (FMT).....	38
Repositorio de librerías compartidas (CDN).....	39
VirtualBox .....	40
<i>GIT</i> .....	41
GITEA.....	41
5.4 Metodología de programación .....	44
6. Desarrollo de la PPS .....	47
6.1 Implementación de permisos a los roles.....	48
6.2 Gestión de legajos .....	51
6.3 Implementación Bloque de antigüedad .....	57
6.4 Implementación del Bloque de embargo.....	64
6.5. Implementación del Bloque de administración .....	79
6.6 Implementación del Bloque de anticorrupción .....	97
6.7 Implementación de la gestión de observaciones.....	107
7. Conclusión .....	115
7.1. Validación personal del trabajo realizado .....	116

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

<b>7.2. Posibles mejoras y ampliaciones .....</b>	<b>117</b>
8. Bibliografía	119
9. Glosario	122

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

## 1. Introducción

### 1.1 Presentación

El presente trabajo propone el diseño e implementación de los requerimientos asignados como parte del equipo de desarrollo del proyecto SIGARHU(Sistema de Gestión y Administración de Recursos Humanos), herramienta destinada a mejorar la gestión de los agentes<sup>1</sup> del Ministerio de Transporte de la Nación.

El proyecto consiste en crear una aplicación web que sustituirá la gestión de los recursos humanos que actualmente se lleva a través de un sistema de archivos Excel, por parte de distintas direcciones dependientes del Ministerio.

Para llevar a cabo el proyecto, se utilizará el lenguaje de programación web *PHP*, el cual se usa como estándar para todos los desarrollos que se realizan en la Dirección de Integración de Sistemas (DIS). Se utiliza, además, un *Framework*<sup>2</sup> propio del Ministerio de Transporte con el patrón de diseño de arquitectura Modelo, Vista, y Controlador (MVC).

En este informe, se describirá en detalle el sistema a desarrollar y sus objetivos, las instituciones intervinientes y los roles para el acceso al sistema, a fin de comprender la información que estos incorporan y gestionan como parte del nuevo programa informático.

---

<sup>1</sup>Término utilizado por las áreas intervinientes del sistema para denominar a los empleados del Ministerio.

<sup>2</sup>Esquema para el desarrollo o implementación de una aplicación. Se compone de un conjunto de módulos listos para la creación de elementos recurrentes estandarizados en el desarrollo de los sistemas (Ortiz, 2018: s/p).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Además, se explicará el papel desempeñado dentro del equipo de desarrollo, el entorno de trabajo, las metodologías de desarrollo y las herramientas utilizadas para la implementación, a fin de plasmar los conocimientos adquiridos como estudiante de la carrera de Ingeniería Informática, con miras a completar la Práctica Profesional Supervisada (PPS).

### **1.2 Objetivos**

El objetivo general del sistema es mantener actualizada y disponible la información referente a los agentes que ingresan a trabajar al Ministerio de Transporte. El software va a permitir a los usuarios del sistema realizar búsquedas, consultas, crear o modificar información de los agentes, según el área de su competencia. El ingreso al sistema se realizará mediante diferentes roles con permisos de acceso, representando las direcciones del Ministerio involucradas.

Asimismo, el presente proyecto procurará brindar un sistema sencillo e intuitivo para los usuarios, con el propósito de facilitar las tareas de las direcciones y también permitir tener una base Integral de información que manipulan las direcciones de manera que sea posible evitar la desactualización, la duplicación de datos o la desincronización al cargar estos. Esto posibilitará obtener una mayor eficiencia que con la gestión actual.

### **1.3 Contexto**

El proyecto será realizado por la Dirección de Integración de Sistemas. Esta se encuentra bajo la órbita de la Dirección General Técnica y Administrativa (DGTA), dependiente de la Subsecretaría de Coordinación Administrativa (SSCA) del Ministerio de Transporte de la Nación.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



El Ministerio de Transporte de la República Argentina, creado en 2015, es resultado de una división del antiguo Ministerio del Interior y Transporte. Al Ministerio le compete asistir a la Presidencia de la Nación en todo lo inherente al transporte aéreo, ferroviario, automotor, fluvial y marítimo, y la actividad vial.

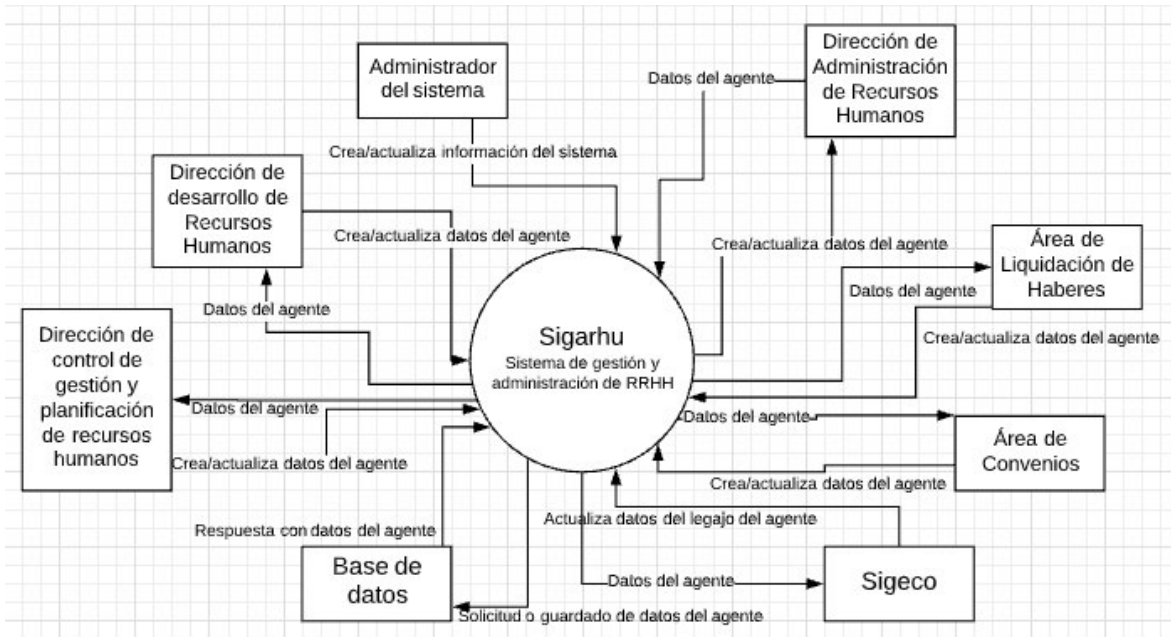
El sistema beneficiará a distintas direcciones y áreas dependientes de la Dirección General de Recursos Humanos dentro de la órbita de la Subsecretaría de Coordinación Administrativa. Dichas direcciones se citan a continuación:

- a. Dirección de Administración de Recursos Humanos
- b. Dirección de Desarrollo de Recursos Humanos
- c. Dirección de control de Gestión y Planeamiento de Recursos Humanos
- d. Área de convenio, dependiente de la Dirección General de Recursos Humanos.
- e. Área de Liquidación de Haberes, dependiente de la Dirección General de Recursos Humanos.

Las direcciones beneficiarias del sistema podrán utilizarlo a través de la asignación de distintos roles de usuarios registrados, lo que les permitirá acceder a una mayor o menor funcionalidad dentro del sistema. Las direcciones citadas se describen en más detalle en el apartado “Áreas intervinientes” y los roles de usuarios se detallan en la sección “Usuarios del sistema”.

Para presentar el modo en que las direcciones y áreas intervinientes, como actores principales, se relacionan con el sistema, se realiza el siguiente diagrama de contexto:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



**Figura 1.1** Diagrama de contexto del sistema SIGARHU (imagen propia)

El presente diagrama muestra todo el sistema como único proceso; indica los elementos externos con que el sistema interactúa y el intercambio de información entre estos; además, delimita en términos generales el alcance del sistema. En el diagrama, se muestra cómo SIGARHU interactúa con otros agentes. Se observa las Direcciones del Ministerio, citadas anteriormente. El sistema también interactúa con la base de datos como única fuente de información y con el Sistema de Gestión de Contratos (SIGECO).

Inicialmente, la gestión de contratos formaba parte de los objetivos principales de SIGARHU, pero, por decisión de los solicitantes, la funcionalidad de gestión de contratos se separó del proyecto para formar

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

parte de un sistema independiente. Este sistema recibirá los datos de los legajos a través de una API REST<sup>3</sup> que permite utilizar la información de los legajos de SIGARHU en SIGECO.

#### 1.4 Planteamiento de problema

La información de los empleados que las direcciones gestionan a través de un sistema de planillas Excel es un sistema con alto nivel de inseguridad en la integridad de datos y archivos, y, además, es un sistema que ha quedado anacrónico ante el surgimiento de nuevas herramientas de software que hacen la labor mucho más sencilla, eficiente y segura. Adaptar la gestión actual de los funcionarios implica hacerlos cambiar de un paradigma de datos planos a un sistema con dimensiones, pero que sea intuitivo o de fácil uso.

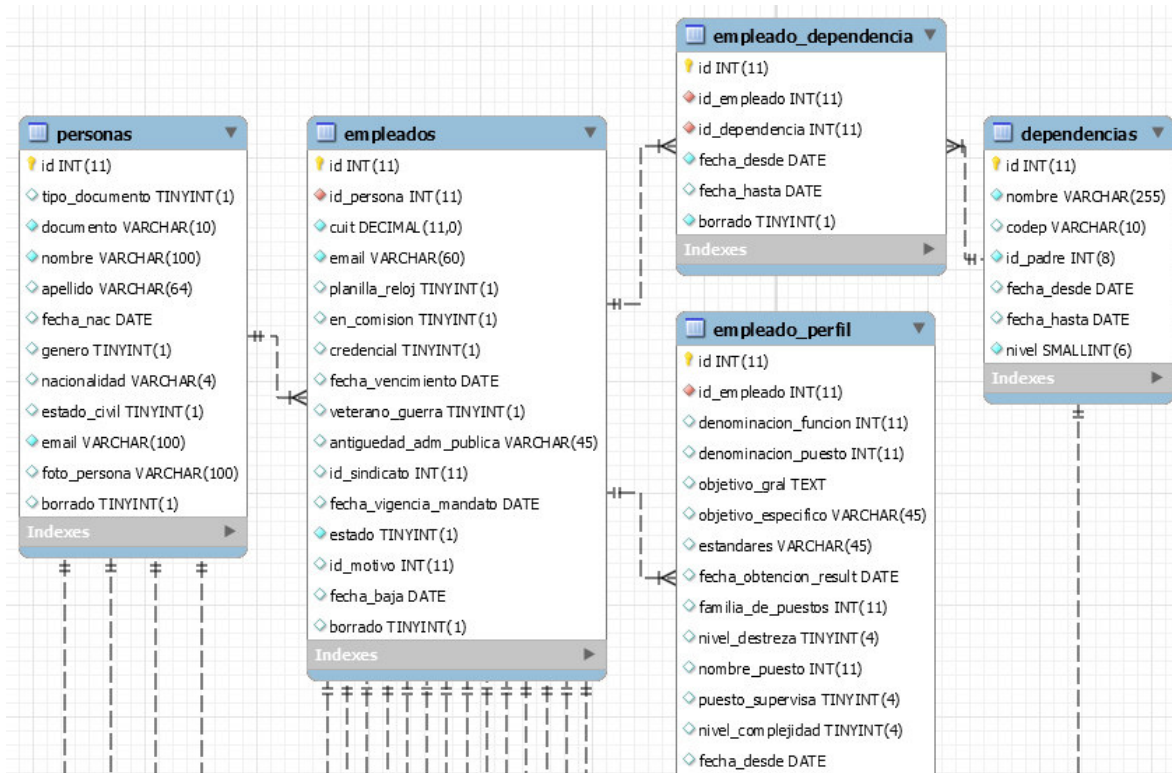
El planteamiento para la operatividad del sistema nace luego del análisis de los datos que se manejan de cada agente. A partir de esto, se detecta que cada agente es el núcleo de los datos analizados y toda la información gira en torno a él; por lo tanto, se va a representar como un modelo de entidad principal. A continuación, en la Figura 1.2., se muestra la entidad principal "Empleados" en relación con sus propiedades. En el ejemplo de la imagen, se puede ver la relación del empleado con las entidades "Perfil" y "Dependencia". Las entidades principales están pensadas como abstracciones de lo que

---

<sup>3</sup>Una API REST es un servicio HTTP que puede ser llamado mediante librerías y herramientas web estándar. API (*Application Programming Interface* - Interfaz de Programación de Aplicaciones) representa una interfaz de comunicación entre componentes de software. REST (*Representational State Transfer* - Transferencia de Estado representacional) permite que un recurso sea transferido con su estado y relaciones mediante formatos y operaciones estandarizadas. Finalmente, HTTP (*Hypertext Transfer Protocol*) es el protocolo estándar para el intercambio de datos en web (AA.VV., 2019f: s/p).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

representa un agente y sus características o propiedades. Esta abstracción de los datos apunta a crear referencias, sin repetirlos.



**Figura 1.2** Fragmento del Diagrama de Entidad - Relación (DER) de SIGARHU (imagen propia)

La gestión de legajos consistirá en la administración de las entidades principales, conformada por solapas de datos que gestionan las distintas direcciones.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

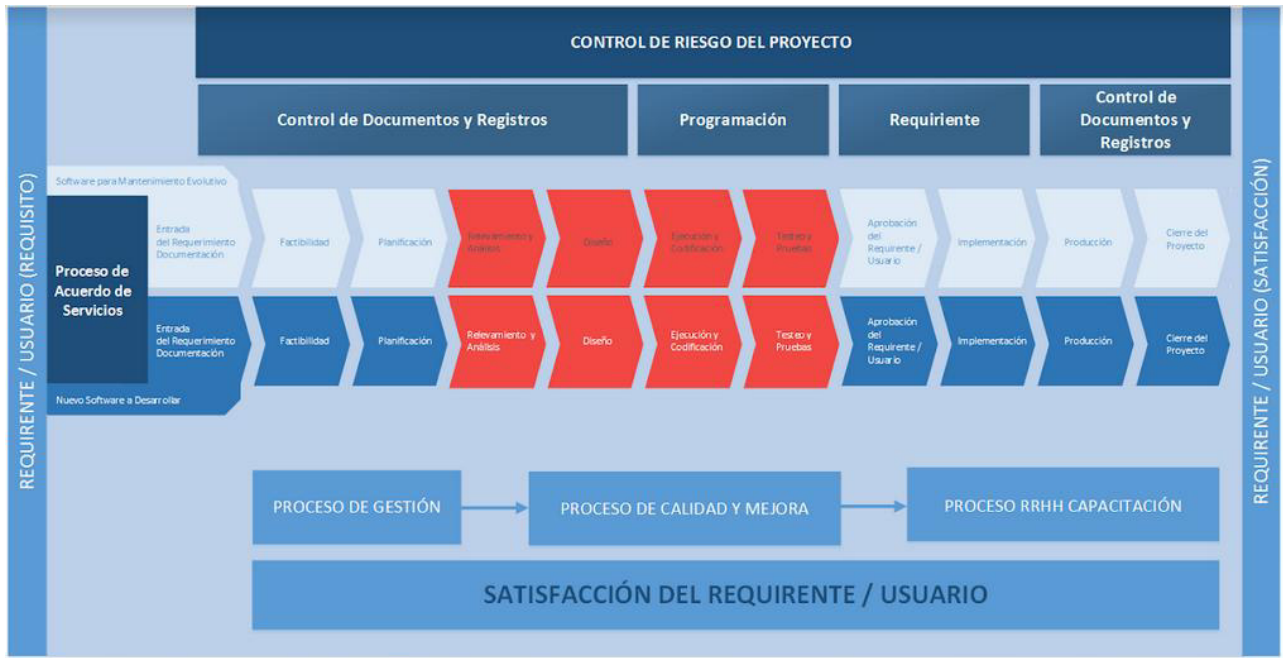
## 2. Forma de trabajo

En esta sección, se describirá la forma en que se desarrollarán las actividades dentro de la DIS. La primera parte del trabajo dentro del equipo de desarrollo es realizar reuniones con el líder técnico, y los demás desarrolladores del equipo, para comprender en conjunto las necesidades y requerimientos del sistema proporcionados por el líder del proyecto (LP). Para ello, fue necesario trabajar a la par con el equipo en la estrategia de desarrollo a seguir en conjunto y ajustarse a los requerimientos concernientes al proyecto.

Una vez claras las reglas, políticas y restricciones, que ayudan a ejecutar las operaciones de las direcciones de manera correcta, proporcionadas por el LP, y acordadas con el líder técnico, se asignan las tareas de desarrollo. A estas tareas se les designan tiempos de realización en la herramienta de seguimiento y gestión de proyectos *Redmine*, para su posterior desarrollo. La Herramienta *Redmine* se describe con detalle en el apartado de Metodologías de desarrollo.

Para comprender las etapas del proceso de desarrollo de este nuevo software en la DIS, se presenta el siguiente diagrama:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



**Figura 2.1** Mapa de Procesos (Pereira, 2019:20)

En la figura 2.1, se presenta la intervención en las etapas de mapa del proceso, resaltadas en color rojo. Estas se describen a continuación:

Relevamiento y análisis: en esta etapa, se analizarán requisitos (nuevos campos, tipos de datos, formularios, listados, etc.) de las tareas de desarrollo solicitadas por el líder técnico.

Diseño: para el diseño, se utilizarán las herramientas y la documentación complementaria (Diagrama de Entidad-Relación, Documento de acuerdo etc.), según lo requiera el proyecto.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Ejecución y codificación: en esta etapa, se llevarán adelante las tareas de programación necesarias para dar solución a los requisitos enunciados en la etapa de diseño, tomando como base los diferentes documentos generados en dicha etapa.

Testeo y pruebas: en esta etapa, se ejecutarán pruebas funcionales y de integración, antes de cerrar las tareas asignadas.

### **3. Análisis y relevamiento**

#### **3.1 Instituciones involucradas en el sistema**

A continuación, se describen los sectores que forman parte del Ministerio de Transporte y son solicitantes del sistema SIGARHU. Asimismo, se brinda una breve información de sus principales tareas y actividades, a fin de comprender las operaciones en las que intervendrá el sistema, ayudando a realizar estas actividades de forma más rápidas, flexible y cómoda para los usuarios.

##### **a. Subsecretaría de Coordinación Administrativa**

La Subsecretaría de Coordinación Administrativa tiene como función la asistencia a otras áreas de la Secretaría en los trámites administrativos, para conseguir los recursos humanos que resulten necesarios y obtener los materiales, el equipamiento tecnológico y cualquier otro insumo imprescindible para el cumplimiento de las tareas dentro del Ministerio de Transporte.

##### **b. Dirección General de Recursos Humanos**

La Dirección General de Recursos Humanos forma parte de la Subsecretaría de Coordinación Administrativa del Ministerio de Transporte y es la encargada de administrar y actualizar los sistemas de información relacionados con las herramientas de administración de recursos humanos, donde se

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

aplican los controles que permiten la correcta liquidación de haberes del personal. También gestiona las actividades correspondientes al desarrollo de la carrera administrativa de los empleados del Ministerio, como, por ejemplo, la selección, integración, evaluación de desempeño, promoción y capacitación. La Dirección General tiene a su cargo la Dirección de Administración de Recursos Humanos, la Dirección de Desarrollo de Recursos Humanos y la Dirección de Control de Gestión y Planificación de Recursos Humanos. Estas últimas se describen a continuación.

**c. Dirección de Administración de Recursos Humanos**

La Dirección de Administración de Recursos Humanos coordina la aplicación de los estatutos o normativas para la relación contractual entre el empleado público y la administración pública. Asimismo, se ocupa de la relación escalafonaria o rango en que se agrupan las personas integradas en la institución, además de controlar el cumplimiento de las reglamentaciones relativas a licencias, compensaciones, situaciones especiales de revista, asignaciones familiares, incompatibilidades, y proponer las adecuaciones pertinentes.

**d. Dirección de Desarrollo de Recursos humanos**

La Dirección de Desarrollo de Recursos Humanos es la encargada de coordinar y evaluar la aplicación de los sistemas de selección y evaluación de desempeño. También establece los estudios que permiten la clasificación de puestos de trabajo y especifica los perfiles básicos aconsejables para la ocupación de puestos.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



**e. Dirección de control de Gestión y Planeamiento de Recursos Humanos**

La Dirección de Control de Gestión y Planeamiento de Recursos Humanos asesora a las áreas dependientes de su jurisdicción en el diseño de sistemas, métodos y medición de trabajo administrativos y sistemas de información que posibiliten la adecuada aplicación de las acciones de cada uno de los sectores funcionales del Ministerio y que facilitan un funcionamiento compatible con la obtención de los resultados esperados.

También realiza la evaluación técnica y asesora en materia de administración de espacios físicos y equipamiento de oficinas, respetando las condiciones óptimas de trabajo de las personas.

**3.2 Usuarios del sistema**

Una vez presentadas las direcciones o áreas involucradas, se exponen, en lo que sigue, los roles que harán uso del sistema. Los roles de usuario son seis. Cada uno de ellos deberá ser dado de alta por el rol "Administrador". Estos roles segmentan la administración y actualización de los datos de los agentes, según las respectivas responsabilidades de cada una de las direcciones a las que pertenecen y la información sobre los agentes que les compete.

- **Usuario Administrador:** es el encargado de administrar las altas y bajas de los usuarios, asignando módulos, opciones del menú y permisos de quienes deban operar el sistema.
- **Usuario Administrador RRHH:** este usuario pertenece a la Dirección de Administración de Recursos Humanos; podrá dar de alta o modificar los datos personales, la situación escalafonaria, los datos sobre la antigüedad del agente y también los datos referidos a las cargas horarias.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- **Usuario Desarrollo RRHH:** usuario perteneciente a la Dirección de Desarrollo de Recursos Humanos. Tendrá acceso a la información del agente en lo que concierne a la situación escalafonaria y formación. Además, deberá convalidar la información que administra la Dirección de Control de Gestión y Planificación de Recursos Humanos, así como el Área de Convenios, para ciertas modalidades de vinculación y situación de revista, aceptando lo definido por la otra Dirección o Área u observando alguno de los campos, a fin de que se corrija el error hasta estar ambas Direcciones o Áreas de acuerdo.

- **Usuario Convenios:** el Área de Convenios es un sector que forma parte de la Dirección General de Recursos Humanos y se encarga de actualizar información correspondiente a los datos personales de los agentes contratados, situación escalafonaria, perfil del puesto a ocupar, formación, etc., entre otras categorías.

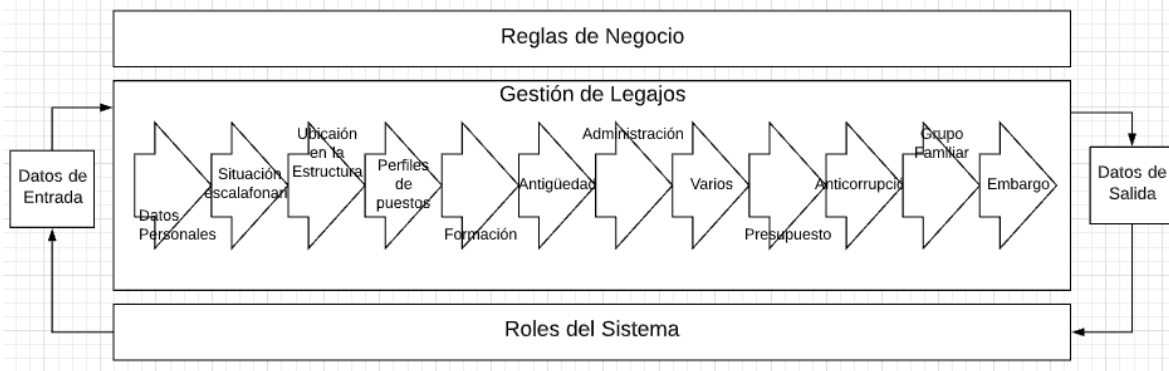
- **Usuario Control RRHH:** el usuario control de RRHH pertenece a la Dirección de Gestión y Planificación de Recursos Humanos y será el encargado de actualizar información concerniente a la ubicación en la estructura del Ministerio, la situación escalafonaria, y los perfiles del puesto del agente. Además, deberá resolver las observaciones que la Dirección de Desarrollo haga sobre algún agente, dado que esta Dirección convalida la información que administra la Dirección de Control de Gestión y Planificación de Recursos Humanos.

- **Usuario Liquidación:** este usuario pertenece al Área de Liquidaciones de Haberes de la Dirección General de Recursos Humanos y se encarga de actualizar en el sistema información sobre

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

el presupuesto del agente. Puede aplicar un presupuesto a un Agente, siempre y cuando este tenga un atributo que permita asignar tal presupuesto.

Las reglas impuestas para los requerimientos limitarán tanto a los distintos roles del sistema, así como la posibilidad de visualizar y editar solapas o bloques del sistema. En líneas generales, la figura 2.2, a continuación, muestra un panorama del flujo de información en el sistema, por medio de las distintas solapas de la gestión de legajos.



**Figura 2.2** Flujo de información del sistema SIGARHU (imagen propia)

La figura expone una serie de elementos articulados que forman parte de un conjunto integrado. Cuando alguno de estos elementos se modifica, modifica al conjunto en su totalidad.

Este sistema de información, coordinado por las personas que representan las distintas áreas del Ministerio, releva y procesa los datos, transformándolos en información, y va a ayudar a una mejor toma de decisiones y desarrollo de acciones correspondientes a cada Dirección.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

#### 4. Tareas a ejecutar

Como se dijo previamente, en el presente trabajo, se explica el desarrollo de determinados bloques pertenecientes al sistema para la gestión de legajos. Estos bloques estarán divididos por solapas, ya que la información sobre el empleado que se maneja está segmentada según la potestad de los distintos roles. A continuación, se describen los requerimientos que debe cumplir el sistema por medio de las tareas asignadas.

##### 4.1 Bloque de antigüedad

Como parte de las tareas a desarrollar, se solicita la creación del *template*<sup>4</sup> y *back-end*<sup>5</sup> necesarios para el formulario de alta y modificación de los datos de la pestaña de "Antigüedad". Los campos de datos de este bloque se delimitan a continuación:

- Años y meses: campos de datos donde se podrá seleccionar el año y el mes correspondientes a la antigüedad del empleado en la administración pública. Esta combinación de datos se guardará como la antigüedad en la administración pública. Los datos que se guarden en el campo serán en formato *Json*<sup>6</sup>, por ejemplo, {"anio":"8","mes":"0"}
- Fecha Ingreso MTR: campo de datos para el ingreso de fechas.

---

<sup>4</sup>Los *Templates* son plantillas para hacer páginas con un diseño general. Es la parte visible de un sitio web. (Barzanallana, 2016: s/p).

<sup>5</sup>El *back-end* es la parte del código que se encarga de la lógica del funcionamiento y la que se conecta con la base de datos y el servidor. El *front-end* a diferencia del *back-end* se encarga de la interacción con el usuario. (Arjonilla, 2019: s/p)

<sup>6</sup>Estándar basado en texto plano para el intercambio de información (AA.VV., 2019j: s/p).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- Contador de Antigüedad MTR: campo de datos informativo que se completa calculando la diferencia entre la fecha actual y la fecha del campo "Fecha Ingreso MTR". El formato será "XX años XX meses XX días".

El formulario tendrá un doble comportamiento: si el CUIT suministrado refiere a un empleado existente, será un formulario de modificación; si no existe, será un formulario de alta.

En el caso de que no existan datos del empleado, se mostrará un mensaje que lo informe, sin mostrar el formulario, porque no es posible cargar datos huérfanos. Este formulario sólo será utilizable para usuarios con Rol administrador control RRHH y Rol administrador convenios. Para los usuarios que no tengan los roles anteriormente citados, los datos serán visibles, pero no accesibles. La carga de datos en el Bloque de antigüedad es dependiente de los datos del Bloque de datos personales y del Bloque de ubicación en la estructura. En caso de que estos datos no estén disponibles, se mostrará un mensaje de advertencia, por ejemplo: "Los datos básicos del empleado, necesarios para definir la antigüedad, no existen."

#### **4.2 Bloque de administración**

Tal como se dijo previamente, como parte de las tareas a desarrollar, se solicita la creación del *template* necesario para mostrar la pestaña de "administración". En esta pestaña, se gestionan datos sobre los siguientes ítems: 1) la carga horaria; 2) la ubicación y 3) licencias especiales del agente. Los campos de datos se enumeran y explican a continuación, separados respecto de cada uno de los tres ítems.

##### 1) Carga horaria:

- Plantilla Reloj: campo de datos seleccionable del tipo de plantilla horaria.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- Grilla horaria: grilla con los días y horarios de trabajo del empleado.

2) Ubicación:

- Edificio: campo de datos seleccionable del edificio.
- Calle/número: campo de datos para la calle y número; se muestra a modo informativo.
- Piso: campo de datos seleccionable del piso; depende de la selección del edificio.
- Oficina: campo de datos seleccionable de oficina; depende de la selección del edificio y del piso.

3) Licencias especiales:

- Licencias especiales: campo de tipo seleccionable para el ingreso del sindicato al que pertenece el agente.
- Fecha inicio: campo de datos para el ingreso de fechas.
- Fecha fin: campo de datos para el ingreso de fechas.

Este formulario tendrá un doble comportamiento: si el CUIT suministrado refiere a un empleado existente, será un formulario de modificación; si no existe, será un formulario de alta.

En el caso de que no existan datos del empleado, se mostrará un mensaje que lo informe, sin mostrar el formulario, porque no es posible cargar datos huérfanos.

Estos formularios serán sólo utilizables para usuarios con Rol Administrador Control RRHHyRol Administrador Convenios. Para los usuarios que no tengan este rol, tal como en el caso anterior, dichos datos serán visibles, pero no accesibles. En caso de que no existan datos, se mostrará un mensaje que lo informe.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

### 4.3 Bloque de embargos

Se solicita también la creación de una nueva solapa en gestión de legajos, titulada “Embargos”. El rol que podrá administrar estos datos será el de administrador de RRHH. La solapa contendrá un listado de embargos con "fecha de cancelación" mayor o igual al día de la fecha de ingreso, con la posibilidad de crear nuevos, modificar existentes o dar de baja erróneos (baja lógica<sup>7</sup>).

A su vez, esta solapa también tendrá un listado denominado "Historial", en donde se verán los embargos con "fecha de cancelación" menor al día de la fecha de ingreso.

Por un lado, el agente puede tener más de un tipo de embargo vigente y, por otro lado, embargos concurrentes.

Los campos de datos correspondientes en este caso se listan a continuación:

- Autos: campo que corresponde al nombre de la resolución judicial de un embargo por lo que el campo para el ingreso de texto.
- Fecha de alta y Fecha de cancelación: campos para el ingreso de fechas.
- Tipo de Embargo: campo de datos de tipo seleccionable para el tipo de embargos, las opciones serán “ejecutivo” y “familiar”.

---

<sup>7</sup>La “baja lógica” consiste en que el dato no esté disponible para el usuario, como si se hubiera borrado, pero, internamente, sólo se lo marca para indicar que está dado de baja, a diferencia del borrado físico, que es la capacidad de borrar completamente el registro (De Seta, 2009: s/p).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- Monto: campo para el ingreso de valores numéricos. Si el tipo de embargo es “ejecutivo”, es una cantidad en pesos (\$), y si el tipo de embargo es “familia”, el monto es una cantidad porcentual (%).

#### **4.4 Bloque de anticorrupción**

Se solicita crear una nueva solapa para la funcionalidad de anticorrupción. Esta solapa solo será visible para el rol de Control de RRHH y para el administrador de la aplicación. Se solicita el alta, baja y modificación de los datos que integran la solapa. Estos datos, relativos a los agentes, se listan en los siguientes campos, explicados en cada caso:

- Obligado a presentar declaración jurada: campo de datos en el que se declara si el agente está obligado a presentar una declaración por su carácter (SI/NO).
- Tipo de Declaración Jurada: campo de datos seleccionable en tres posibilidades: anual, inicial, baja.
- Fecha de designación: campo de datos para el ingreso de fechas.
- Fecha de publicación de la designación: campo de datos para el ingreso de fechas.
- Fecha de aceptación de la renuncia: campo de datos para el ingreso de fechas (solo se permitirá completar este campo si el legajo no está activo y se da de baja).
- Fecha de última presentación: campo de datos para el ingreso de fechas.
- Número de transacción: campo de datos para el ingreso de valores numéricos.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



Por una parte, además, esta solapa permitirá adjuntar el comprobante asociado a la presentación en formato PDF. Por otra, se podrán visualizar las presentaciones anteriores y la información que se brindará es la siguiente:

- Fecha de presentación
- Tipo de presentación
- Número de transacción
- Visualizar PDF

Se solicita crear un nuevo formulario para alta y modificación de presentaciones anticorrupción, separado del formulario principal.

#### **4.5 Observaciones de gestión**

Se solicita agregar a la vista de legajos una sección de “observaciones” independiente de los formularios de los bloques de legajos, que sea ‘colapsable’, es decir, que sea usada para mostrar u ocultar el contenido. La sección contendrá un campo para el ingreso de texto que sirva para la carga de las nuevas observaciones. También deberá incluir un listado de las observaciones existentes ordenadas por fecha en orden descendente y que el listado contenga las siguientes columnas de datos:

- Fecha
- Usuario
- Bloque
- Observación

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Los roles intervinientes serán todos los participantes en la gestión de legajos. Los usuarios podrán hacer observaciones sobre los bloques en los que tienen permisos de edición y podrán ver todas las observaciones del legajo, pero sólo modificarán las propias.

En la base de datos, quedarán registrados los id de usuarios, el bloque, la fecha y la observación específica, en cada caso.

### **5. Diseño e implementación**

En esta etapa, el líder técnico del equipo ayuda a definir estándares para el desarrollo y el acompañamiento de la metodología de trabajo adoptada. Se especifica qué debe hacer la aplicación y se definen los modelos de datos que permitan garantizar la integridad de la información y los datos almacenados.

A continuación, se describe el entorno de desarrollo, el cual consiste en todo lo que necesita el proyecto para ser desplegado, como los ambientes de trabajo, las herramientas a utilizar y la metodología de programación.

#### **5.1 Ambientes de trabajo**

La Dirección de Integración de Sistemas es responsable del desarrollo de otros proyectos y cuenta con otros equipos trabajando simultáneamente, por lo que se administran distintos ambientes de trabajo para el desarrollo de los proyectos.

Los ambientes de trabajo en el desarrollo de software son una práctica recomendada para realizar cambios en el código que puedan ser probados antes de ponerlos en uso real, con el fin de obtener

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

aplicaciones de calidad que satisfagan las necesidades de los clientes. El objetivo de tener varios ambientes es minimizar la cantidad de fallas en el ambiente de producción.

Para lograr esto, la dirección utiliza cuatro distintos ambientes: el ambiente de desarrollo, el ambiente de pruebas, el ambiente de preproducción y el ambiente de producción. Los cuatro se describen a continuación:

**a) Ambiente de desarrollo**

El ambiente de desarrollo es un ambiente de trabajo de uso exclusivo para los programadores, donde residen todos los recursos informáticos necesarios para efectuar tareas de análisis y programación de sistemas (desarrollo, pruebas y mantenimiento).

Este ambiente es donde se realizan los cambios que surjan como requerimientos de nuevas tareas o ajustes del cliente. Nada de lo que se realice en este ambiente afectará al resultado final.

El entorno de desarrollo consiste en la virtualización, en la máquina local, del desarrollador basado en un ambiente de producción, para hacer que el escenario sea lo más parecido al que luego hay que implementar.

Una vez listos los desarrollos de las tareas solicitadas, estos pasan por un *reléase* o lanzamiento, para ser homologados por los QA (es decir, los responsables encargados de asegurar el correcto funcionamiento del software que se desarrolla).

**b) Ambiente de pruebas**

El ambiente de pruebas es donde se hacen las pruebas funcionales por parte del QA. En este entorno, no se altera el código de ninguna manera, ya que el objetivo es asegurar que el software sea puesto en

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

producción con un desperdicio mínimo. Con esto, se puede evaluar su comportamiento y luego planificar su implementación en el entorno de producción que se describe posteriormente.

### **c) Ambiente de preproducción**

El ambiente de preproducción es un entorno intermedio, similar al de producción y con los mismos datos, donde los usuarios finales pueden verificar que el programa se desempeña del modo necesitado. Pertenece a las últimas etapas, pero es previo a la liberación de la versión nueva del software. La finalidad de este entorno es determinar si se cumplen las necesidades o requerimientos de las empresas y sus usuarios.

### **d) Ambiente de producción**

En el ambiente de producción, se encuentran los datos operativos y los usuarios finales realizan transacciones con dichos datos en él. En los ambientes previamente descritos, se homologan los nuevos cambios del sistema y son aprobados por el área o usuario solicitante.

Una vez explicados los ambientes, se procede a explicar las tecnologías para la implementación del desarrollo.

## **5.2 Tecnologías a utilizar**

### **PHP (HypertextPreprocessor)**

Para el desarrollo del sistema, se utilizará el lenguaje de programación PHP v5.5., bajo el paradigma orientado a objeto. PHP es un lenguaje de código abierto, lo que significa que puede ser usado por cualquier programador que quiera usarlo, adecuado para el desarrollo WEB y que puede ser incrustado en HTML. Esto implica que, en un mismo archivo, se puede usar PHP y combinarlo con HTML.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

PHP se utiliza para generar páginas web dinámicas, por lo tanto, el contenido de la página cambia, a diferencia de las páginas web estáticas, donde el contenido siempre es el mismo. Por ejemplo, los contenidos pueden cambiar en función de los cambios que haya en una base de datos, de búsquedas o aportaciones de los usuarios, etc.

PHP es un lenguaje altamente permisivo y esto puede llevar a desarrollar código de forma desordenada, por lo que se decide programar bajo el paradigma orientado a objeto. Esta es una forma de construir, pensar, diseñar e implementar programas basados en objetos que combinan estados, comportamientos e identidades. Esto permite agilizar y dinamizar los códigos para hacerlos más accesibles.

### **Programación Orientada a Objetos (POO/OOP)**

La “Programación orientada a objetos” (POO, en español; OOP, en inglés) es una metodología de programación avanzada y bastante extendida, en la que los sistemas se modelan creando “clases”, que son un conjunto de datos y funcionalidades. Las clases son definiciones a partir de las que se crean objetos. Los objetos son ejemplares de una clase determinada y, como tales, disponen de los datos y funcionalidades definidos en la clase.

La POO difiere de la programación estructurada tradicional donde los datos y los procedimientos están separados y sin relación, ya que lo único que se busca en esta última es el procesamiento de unos datos de entrada para obtener otros de salida. En cambio, la programación estructurada anima al programador a pensar, sobre todo, en términos de procedimientos o funciones, en primer lugar, y, en segundo lugar, en las estructuras de datos que esos procedimientos manejan. En la programación estructurada, sólo se escriben funciones que procesan datos. Los programadores que emplean POO, sin

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

embargo, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

### **JavaScript**

Se utilizará *JavaScript* para hacer que la aplicación sea más amigable con el usuario. *JavaScript* permite crear contenido nuevo y dinámico, agregando acciones en la página web. A diferencia de PHP, este lenguaje funciona del lado del cliente; los navegadores son los encargados de interpretar estos códigos.

Se dice que *JavaScript* es un lenguaje del lado del cliente (en la computadora del usuario), ya que al ingresar a una página web se está realizando una petición HTTP a un servidor asociado al dominio o dirección que se escribe en el navegador y que trae como resultado un documento HTML, con todas sus asociaciones y referencias, entre las que puede estar un archivo CSS y un archivo *JavaScript*, estos archivos se descargan de forma temporal en la computadora, y son interpretados por el navegador para mostrar la página como se desee.

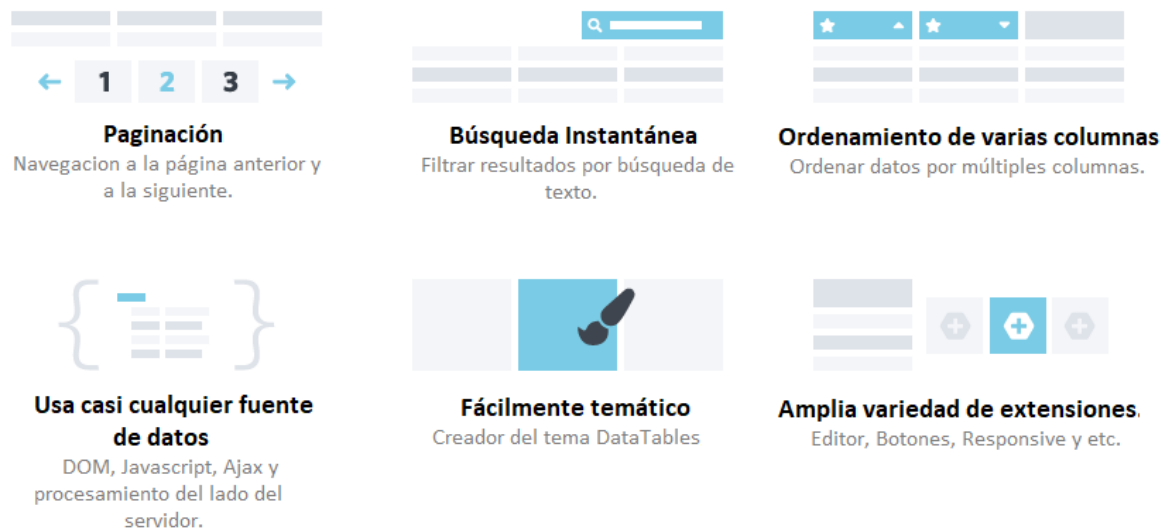
A través de JavaScript, la página web que contiene elementos proporciona comportamiento al sitio web, usando la información que el documento HTML despliega y permitiendo al usuario tener control sobre los elementos. Esto hace que la presentación sea más atractiva al usuario, más intuitiva y que la información pueda mostrarse de una mejor forma. *JavaScript* también permite al desarrollador tener más control sobre los elementos y ser más creativo a la hora de desarrollar una solución web.

Se integran al proyecto algunos elementos de programación pre-programados de *jQuery*, que se trata de una amplia biblioteca basada en *JavaScript* que abarca desde efectos dinámicos, hasta calendarios,

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

entre otros. *jQuery* está optimizado para realizar muchas funciones de *script* frecuentes, permitiendo el uso de menos líneas de código.

Se integra el complemento *DataTabledejQuery*, una herramienta flexible que agrega controles de interacción avanzados a cualquier tabla HTML. En la imagen siguiente, se muestran algunas de las características principales de *DataTable* con sus ejemplos:



**Figura 5.2.1.** Lista de Ejemplos de las características principales de *DataTable*(<https://datatables.net/>)

También se utilizará *Bootstrap*. Esta es una herramienta para crear interfaces de usuarios simples, que otorga la característica de adaptarse a todo tipo de dispositivos y pantallas, sea cual sea su tamaño. Además, *Bootstrap* ofrece las herramientas necesarias para crear cualquier tipo de sitio web utilizando los estilos y elementos de sus librerías.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

*Bootstrap* es un *framework* CSS desarrollado inicialmente en el año 2011 por *Twitter*, que permite dar forma a un sitio web mediante librerías CSS. Estas librerías incluyen tipografías, botones, cuadros, menús y otros elementos que pueden ser utilizados en cualquier sitio web.

*Select2Bootstrap* es otro complemento que se utilizará y que ofrece un cuadro de selección de opciones personalizable con soporte para búsqueda, etiquetado, y muchas otras opciones.

También se utilizaba *bootstrap-fileinput*, que permite convertir un campo simple de tipo "file" en un control avanzado para la subida de archivos. Este complemento, o *plugin*, trabaja con la versión 3 de *Bootstrap* y posibilita la previsualización de los archivos para las imágenes y el texto de los archivos que se desea cargar; además, puede cargar uno o más archivos desde el almacenamiento local.

El proyecto también utiliza el *plugin bootstrap-datetimepicker*, que es una solución rápida para añadir un calendario en un formulario. Este complemento proporciona un calendario totalmente personalizable, en el que se pueden seleccionar fechas y se las puede asociar a elementos HTML, como entradas de formularios.

Entre otras opciones de las librerías de *JavaScript* a utilizar, debe mencionarse *Moment.js*, una librería que permite solucionar problemas y tener un manejo de las fechas mucho más cómodo. Algunas de las posibilidades que brinda dicha librería es el cálculo entre fechas, restar o agregar días, formatearlas fechas apoyándose en diferentes localizaciones (idioma/cultura), etc.

El *plugin datetime-moment.js* es un complemento para *DataTables* basado en un formato dado, de manera que el *datatable* detectará automáticamente su información temporal y ordenará correctamente las cadenas de fecha y horas correctamente.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



## **AJAX**

Para actualizar datos de las tablas sin necesidad de refrescar la página, se utiliza *AJAX*. Esta tecnología permite modificar los contenidos de la página creando efectos dinámicos y rápidos a través de *Javascript*. *AJAX* permite que las páginas web se actualicen de forma asíncrona mediante el intercambio de datos con un servidor web en segundo plano. Esto significa que es posible actualizar partes de una página web sin tener que volver a cargar toda la página.

*AJAX*, acrónimo de *AsynchronousJavaScript And XML* (*JavaScript* asíncrono y *XML*), no es un lenguaje de programación. Aunque *AJAX* se pensó inicialmente para transferir datos en un solo formato (*XML*), actualmente, permite la transmisión de datos en múltiples formatos: *XML*, *JSON*, *EBML*, texto plano, *HTML*, etc.

## **SQL**

El lenguaje de consulta estructurado (*SQL* o *StructuredQueryLanguage*) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre estas. Una de sus características es el manejo del álgebra y el cálculo relacional que permitelanzar consultas con el fin de recuperar de una forma sencilla información de interés de una base de datos, así como también hacer cambios enesta.

## **MySQL**

Como gestor de base de datos, se utiliza *MySQL*. Desarrollado a mediados de los años 90, *MySQL* fue uno de los primeros gestores de bases de datos de código abierto disponibles en el mercado. Hoy en día, existen muchas alternativas o variantes de *MySQL*. Sin embargo, las diferencias entre las variantes

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

no son significativas ya que usan la misma sintaxis y la funcionalidad básica también permanece igual. Este motor de base de datos proporciona un acceso multiusuario a las bases de datos. MySQL se usa con la combinación de PHP y Apache *Web Server*, además de una distribución de Linux. MySQL usa el lenguaje SQL para consultar la base de datos.

### **MVC (Modelo, Vista y Controlador)**

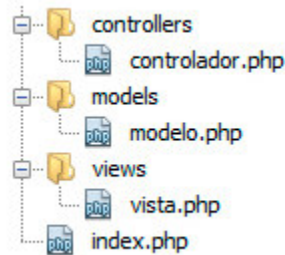
El lenguaje de programación PHP provee libertad a la hora de programar aplicaciones, por lo que podría ocurrir que se descuide el desarrollo del código, se obtenga como resultado un código desordenado y, muchas veces, mal aplicado. Para solucionar este tipo de problema y que resulte posible aplicar buenas prácticas de programación, se utiliza el patrón de desarrollo MVC, el cual posibilita mantener un código estructurado.

MVC es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control, en tres componentes distintos, es decir, el código de los programas queda separado por sus diferentes responsabilidades. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, la reutilización del código y la separación de conceptos.

MVC ayuda a mejorar la manera de creación de software y esto se logra, como se dijo, mediante la arquitectura basada en capas que separan el código en función de responsabilidades o conceptos, lo que contribuye a crear aplicaciones con mayor calidad.

A continuación, se presenta la imagen de un ejemplo de la estructura ordenada del código que sigue el patrón MVC.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



**Figura 5.2.2.** Estructura de carpetas de código respetando el patrón MVC  
(<https://www.adaweb.es/modelo-vista-controlador-mvc-php/>)

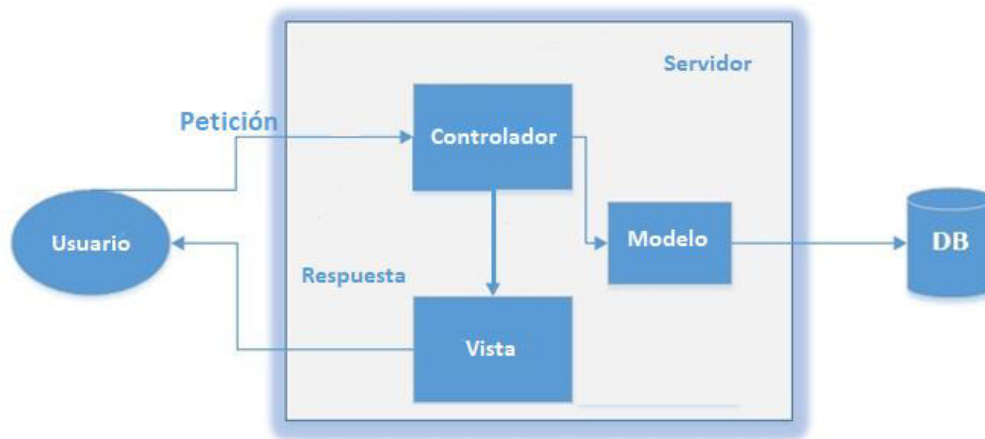
A continuación, se detallan las funciones de los componentes:

- El modelo es el responsable de las siguientes funciones:
  - Acceder a la capa de almacenamiento de datos.
  - Definir las reglas de negocio (la funcionalidad del sistema). Lleva un registro de las vistas y controladores del sistema.
- El controlador es responsable de las funciones que se detallan aquí:
  - Recibir los eventos de entrada, para responder a las acciones solicitadas, por ejemplo, las acciones como el alta, la baja, etc.
  - Contener las reglas de gestión de eventos, del tipo "Si Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas.
  - Servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo. Su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida.
- Las vistas son responsables de las funciones que se sintetizan en lo que sigue:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- Recibir datos del modelo y mostrarlos al usuario.
- Requerir los datos a los modelos y generar la salida.

A continuación, se presenta un diagrama donde se visualiza cómo colaboran las distintas capas que componen la arquitectura de desarrollo de software en el patrón MVC.



**Figura 5.2.3.** Flujo de trabajo característico en un esquema MVC (imagen propia)

En la imagen, se representan con flechas los modos de colaboración entre los distintos elementos que formarían una aplicación MVC, junto con el usuario o cliente. Como se puede ver, el usuario realiza la solicitud al sitio web y ésta llega al controlador; a su vez, se comunica con los modelos y con las vistas. Luego, solicita los datos a los modelos y, una vez que los obtiene, el controlador es el responsable de pasar la información a las vistas para que estas muestren la información al usuario.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

### 5.3 Herramientas

En la siguiente sección, se describen los programas informáticos que se utilizan para el desarrollo del nuevo sistema. Estos son *Sublime Text*, *MySQLWorkbench* y *Framework* del Ministerio de Transporte (FMT), entre otros.

#### **Sublime Text**

*SublimeText* es el editor de código elegido para el trabajo, ya que es multiplataforma, ligero y sencillo, con una interfaz de color oscuro y riqueza en el coloreado de la sintaxis para centrar la atención. La herramienta permite tener varios documentos abiertos mediante pestañas.

El programa dispone de auto-guardado y muchas opciones de personalización; además, cuenta con un buen número de herramientas para la edición del código y automatización de tareas. Soporta *macros*<sup>8</sup>, *Snippets*<sup>9</sup> y autocompletar, entre otras funcionalidades que hacen que los desarrolladores puedan adaptarlo a su comodidad, a fin de incrementar la productividad.

*Sublime Text* es un programa pago, aunque se puede descargar una versión de prueba, plenamente funcional y sin limitación de tiempo.

---

<sup>8</sup>Las *macros* permiten automatizar operaciones repetitivas de manera sencilla. Se pueden grabar o escribir a mano, por ejemplo, “pulsar Ctrl+s” para guardar en archivo.

<sup>9</sup>Los *snippets* son secciones de código que pueden traer a lo que se está escribiendo usando un *keyword* y presionando *tab*. Por ejemplo, si se escribe cientos de `console.log's` en *JavaScript* al día, sería útil usar un *snippet*, de tal manera que, con solo escribir `cl`, `console` u otro *keyword* que se elija y, al presionar *tab*, este código se insertará después del cursor.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

**MySQLWorkbench**

*MySQLWorkbench* v.6.3 es un software creado por la empresa *Sun Microsystems*. Esta herramienta permite modelar diagramas de Entidad-Relación para bases de datos *MySQL*<sup>10</sup>. Asimismo, con ella, se puede elaborar una representación visual de las tablas, vistas, procedimientos almacenados y claves foráneas de la base de datos. Además, es capaz de sincronizar el modelo en desarrollo con la base de datos real. Se puede realizar una ingeniería directa e ingeniería inversa para exportar e importar el esquema de una base de datos ya existente que haya sido guardado o hecho copia de seguridad con *MySQLAdministrador*.

**Poncho**

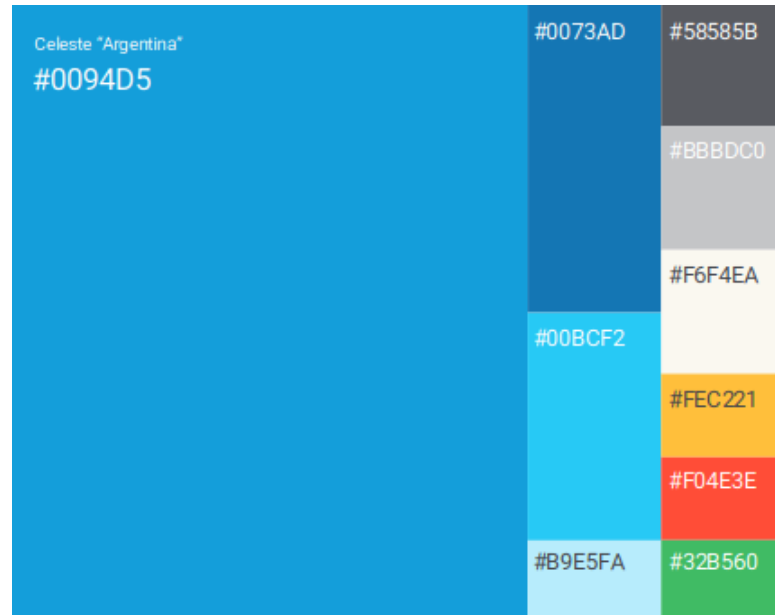
*Poncho* es la librería de estilos, componentes y patrones de uso para diseñar y desarrollar sitios web del Gobierno de la República Argentina. Los colores de *Poncho* están basados en los lineamientos básicos de aplicación de logotipos, paletas de colores y piezas de comunicación de uso institucional de la Presidencia de la Nación argentina.

La tipografía de *Poncho* es *Roboto*. Las tablas de *Poncho* usan *HTML*, clases de *Bootstrap* y tienen funcionalidades opcionales para optimizar su vista en dispositivos móviles.

---

<sup>10</sup>Sistema de gestión de base de datos relacional (RDBMS), de código abierto, basado en lenguaje de consulta estructurado (SQL) (Ramírez Navia, 2018).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



**Figura 5.3.1.** Paleta cromática del Ministerio de Transporte  
([https://www.transporte.gob.ar/\\_img/logos\\_web.pdf](https://www.transporte.gob.ar/_img/logos_web.pdf))

### **Framework del Ministerio de Transporte (FMT)**

El *Framework* es el entorno de trabajo propio del Ministerio de Transporte. Es una carpeta repositorio donde se almacena código que implementa el patrón MVC. FMT contiene componentes comúnmente utilizados para normalizar la estética de las aplicaciones desarrolladas por la DIS, además de vistas genéricas (Cabecera, Footer).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

### **Panel**

El Proyecto *Panel* se usa para entrar o iniciar sesión en la aplicación y se trata de una herramienta única de autenticación centralizada; tiene una conexión por medio de *Active Directory* (Dominio de usuario). Como un sub-proyecto, se cuenta con “Api de usuarios” para configurar los permisos de uso de software.

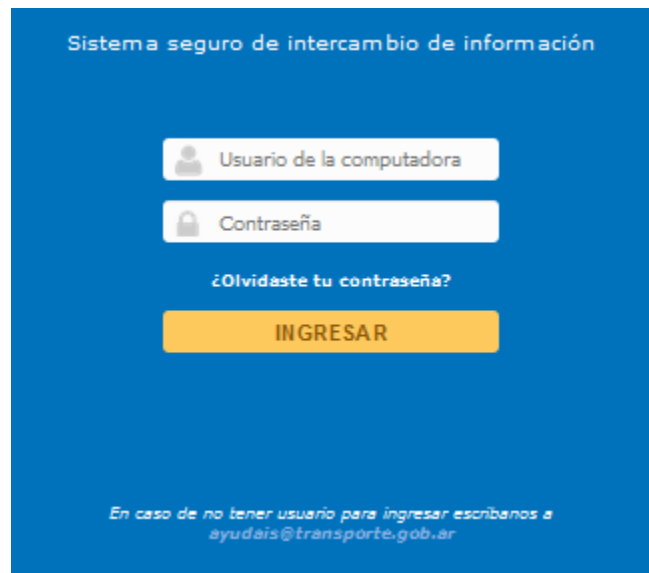


Figura 5.3.2. Flujo de trabajo característico en un esquema MVC (imagen propia)

### **Repositorio de librerías compartidas (CDN)**

Un repositorio de librerías compartidas o CDN (*Content Delivey Network*, por sus siglas en inglés) es una red de distribución de contenidos que almacena información de los sitios web y que sirve al usuario final.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



En el Ministerio, el CDN incluye las librerías de estilos usados para los proyectos desarrollados en la DIS, contiene archivos CSS, archivos *JavaScript*, fuentes, imágenes, etc. Además, al contener las bibliotecas de mayor importancia, se requiere que éstas sean actualizadas periódicamente hacia las últimas versiones estables.

### **VirtualBox**

Para desarrollar las tareas en el ambiente de desarrollo y pruebas, se utiliza la herramienta de virtualizaciones *VirtualBox* que permitirá simular en la máquina local de cada desarrollador un ambiente de producción, pero usado estrictamente para el desarrollo con datos de prueba que no son críticos para el negocio.

La virtualización se refiere a la creación mediante *software* de una versión virtual de recursos tecnológicos, como podrían ser un sistema operativo, un dispositivo de almacenamiento y algún recurso de red.

El *software* de virtualización implementa lo que se llama un “hipervisor” o *Virtual Machine Monitor* (VMM), que consiste en una capa de abstracción entre el hardware de la máquina física (host o anfitrión) y la máquina virtual (MV), formada por *hardware* y *software* virtualizado.

El hipervisor maneja y gestiona los cuatro recursos principales de una computadora (CPU, memoria, almacenamiento y conexiones de red), para, así, poder repartir dinámicamente dichos recursos entre todas las máquinas virtuales creadas en el host anfitrión. Esto permite que se puedan tener varias máquinas virtuales ejecutándose en la misma computadora física.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

**GIT**

Como herramienta de control de versiones se utiliza *Git*. Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan recuperar versiones específicas, más adelante. También tiene como propósito coordinar el trabajo que varias personas realizan sobre archivos compartidos. Los cambios del sistema se guardan en una rama (*branch*).

**GITEA**

Como interfaz de *Git* se utiliza *Gitea*. *Gitea* es un paquete de software de código abierto para alojar el control de versiones de desarrollo de software, utilizando *Git*, así como otras características de colaboración como el seguimiento de errores.

*Gitea* soporta autenticación de doble factor (*two factor authentication*), más funcionalidades relativas a la gestión del código, granularidad en los roles, firma de *commits*<sup>11</sup> con *GPG*, restricción de *push*<sup>12</sup> y *merge*<sup>13</sup> a usuarios específicos, etc.

Para trabajar en *Git* y subir las versiones del código al repositorio, se utiliza un *gitFlow* o “flujo de trabajo”, en *Git*, compuesto por una serie de ramas. Existen dos distinciones principales para trabajar en *Git*; estas son las ramas principales y las ramas auxiliares. La siguiente figura muestra la totalidad de estas.

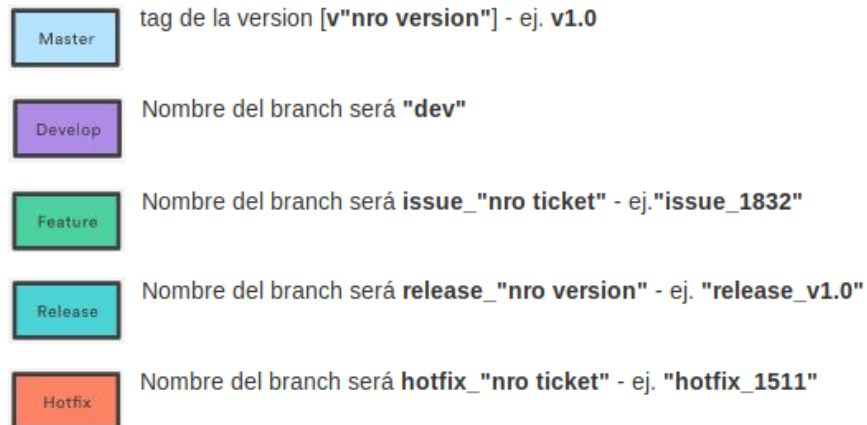
---

<sup>11</sup>El comando *Commit* sirve para incluir cambios en el repositorio (Dudler., s.f.: s/p).

<sup>12</sup>El comando *Push* sirve para enviar los cambios al repositorio remoto (Dudler., s.f.: s/p).

<sup>13</sup>El comando *Merge* sirve para fusionar los cambios de otra rama con la rama activa (Dudler., s.f.: s/p).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



**Figura 5.3.4. Ramas de GitFlow**

(<https://desa.transporte.gob.ar/recursos/gitflow.php>)

En lo que sigue, se las explicará pormenorizadamente.

Las ramas principales son la rama *Master* y la rama *Develop*, que se describen a continuación:

**Master**: es la rama fundamental del repositorio y es la que se crea por defecto, cuando se inicia un proyecto en *Git*. En esta rama, solo estará el código de la última versión del desarrollo preparado para desplegarse en producción.

**Develop**: esta rama es la encargada de contener el último estado del código en desarrollo con las últimas funcionalidades finalizadas para el próximo *reléase* o lanzamiento. Las ramas auxiliares son la rama *Feature*, *Release* y la rama *Hotfix*, las cuales se explican seguidamente:

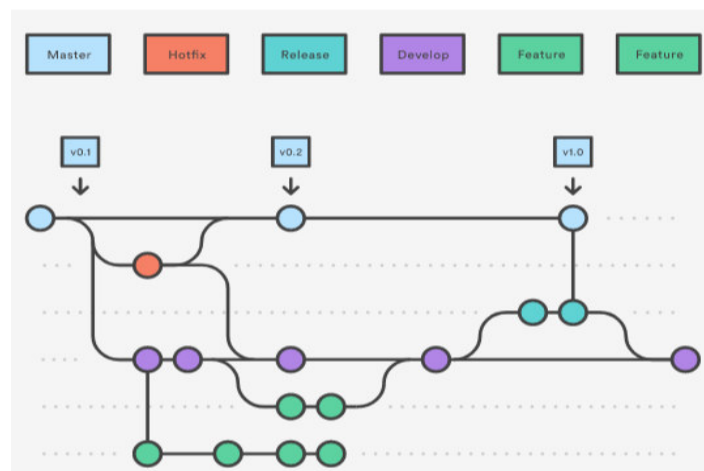
**Feature**: rama con nuevas características o requisitos.

**Release**: rama que se utiliza para estandarizar o cortar una serie de código que se ha estado desarrollando en la rama *Develop*.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

*Hotfix*: rama usada para depurar el código que venga del ambiente de producción, por haberse detectado un defecto crítico.

Las ramas principales no reciben código de manera directa, sino que lo reciben a través de las ramas auxiliares, con el objeto de que no exista el riesgo de generar códigos defectuosos en la subida a producción. Las ramas auxiliares tienen un principio y un fin, ya que se unen a las ramas *Master* y *Develop* y desaparecen. La figura 5.3.5 muestra el flujo de trabajo con *Git*. A continuación, se describe este flujo:



**Figura 5.3.5. GitFlow o flujo de trabajo con Git**  
<https://desa.transporte.gob.ar/recursos/gitflow.php>

Todas las tareas que se asignen para el desarrollo se realizan en la rama *Feature*; una vez cumplidas por los desarrolladores y pre aprobadas, se unirán con la rama *Develop*. Una vez unidas, las ramas *Feature* se borrarán.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:
-------------------	---------------------------	----------------------------	-----------------------------

Cuando se complete un bloque de trabajo, se creará una rama *Release* que se enviará a QA para su homologación. Cualquier corrección se agregará en un nuevo *commit* sobre la rama *Release*. Una vez aprobado, se unirá con *Master*, creando un *tag* de versión (ejemplo: v1.0). Simultáneamente, se unirá con la rama *Develop*. Una vez unido, el *Release* se borrará.

Si se encontraran *bugs*<sup>14</sup> en producción, se creará una rama *Hotfix* para solucionarlos. Una vez aprobada la homologación QA, se unirá con *Master*, creando un *tag* de subversión (v1.1). Al mismo tiempo, se unirá con *Develop*. Una vez unidas las ramas *Hotfix*, se borrarán.

#### 5.4 Metodología de programación

El modelo de metodología de programación será “ágil”, lo cual implica agilizar el flujo de trabajo y crear un equipo capaz de manejar los requisitos siempre cambiantes. Aunque no existe una metodología declarada en la Dirección, puede decirse que la existente es comparable a la metodología “Kanban” por las siguientes características:

- Se intenta que, al realizar las tareas, las cosas salgan bien en el primer intento. Aunque se tarde más en una tarea asignada, esta debe realizarse de manera completa, a fin de evitar la pérdida de tiempo y recursos.
- También se procura reducir el desperdicio, por lo que no se necesita hacer nada extra o superficial por cada tarea asignada, solamente lo necesario para que la tarea resulte bien. De este modo, se optimizan recursos.

---

<sup>14</sup>Un *bug* en un software ocasiona que el programa colapse o que de errores. La mayoría de los *bugs* son fallos humanos (AA.VV., 2015: s/p).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- Ante tareas imprevistas, se busca la forma de adaptarse al cambio, de manera que exista una "cola de espera" de tareas en las que la prioridad se establece en función de las necesidades de cada momento y de la urgencia de cada una de dichas tareas.

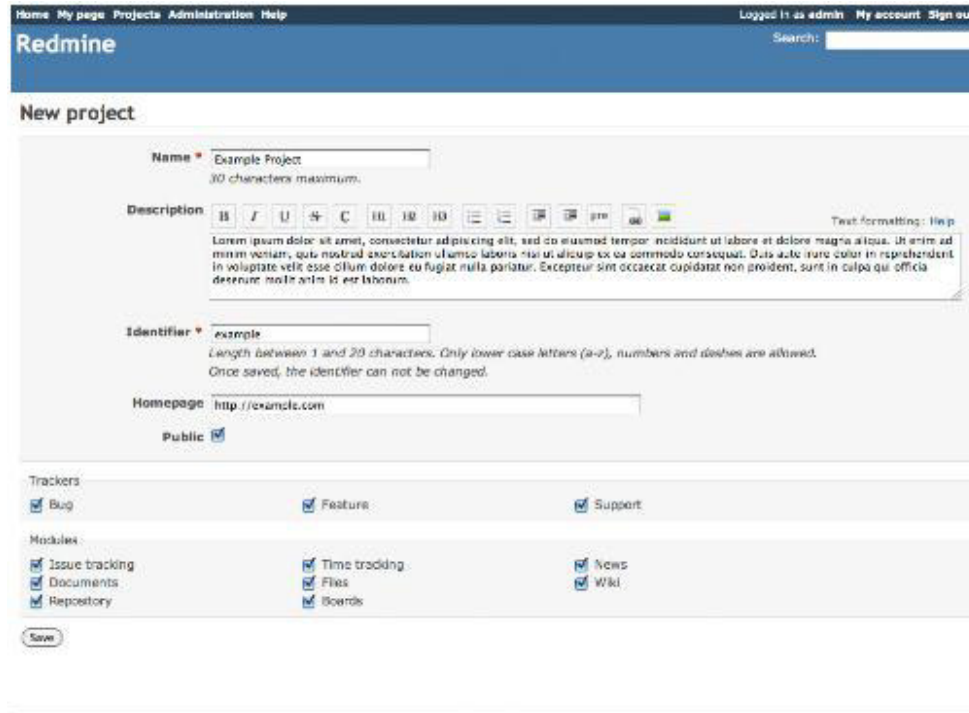
Para llevar a la práctica la metodología elegida, se utilizará la herramienta de control de tareas *Redmine*, la cual recoge y gestiona las peticiones a desarrollar.

### ***Redmine***

La herramienta *Redmine* permite implantar un sistema de gestión de proyectos que ayuda a tener un vistazo rápido de las tareas terminadas, las que están en progreso y las próximas tareas planificadas, así como estimaciones de tiempos para la resolución de tareas, reflejando la realidad del equipo de desarrollo.

*Redmine* es de código libre. Como gestor de proyectos, cuenta con la ventaja de poder tener toda la información asociada a un usuario, acotada dentro del mismo proyecto. Además, permite su control a través de una interfaz web sencilla y muy práctica. A continuación, se presenta, a modo ilustrativo, la interfaz del sistema *Redmine*.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



**Figura 5.4.1.** Interfaz de la herramienta de gestión de proyectos Redmine  
(<https://www.arsys.es/blog/programacion/redmine-gestion-proyectos-cloud/>)

Las características de Redmine son las siguientes:

- Soporta distintos proyectos, en función del sistema de permiso que se aplique.
- Cada proyecto puede llevar asociado documentos, archivos o noticias.
- Posee un sistema de notificaciones a los usuarios mediante correo electrónico, ya sea porque se le ha asignado una tarea o porque una parte del proyecto se ha modificado.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- Brinda la posibilidad de reflejar el desarrollo del proyecto a través de diagramas *Gantt* que permiten plasmar de forma visual el progreso de los proyectos.

Una vez explicadas las tecnologías, metodologías de programación y el entorno de desarrollo, desplegadas hasta la presente sección del trabajo, se explicará la implementación de los requerimientos para la gestión de legajos del sistema SIGARHU, que, como se dijo, tiene por objetivo mejorar la manipulación de los datos referentes a legajos de los empleados. Esta gestión está dividida en bloques o solapas, por lo que los requerimientos serán detallados respecto de estos.

## **6. Desarrollo de la PPS**

En esta etapa, se pone en práctica el diseño y desarrollo de las soluciones a los requerimientos solicitados, teniendo en cuenta la metodología ágil adoptada. Los *tickets* elaborados por el líder técnico y asignados al desarrollo se ponen en curso, y se agregan los tiempos de comienzo y finalización. En cada tarea, el líder técnico asiste en tal sentido, ayudando a validar el correcto uso de los estándares definidos previamente.

Para cumplir con los tiempos acordados, se reporta el avance de la tarea, agregando las horas de trabajo en cada ticket asignado diariamente, e informando sobre los avances o las dificultades para la resolución de las tareas al líder técnico, de manera que pueda reportar desvíos al LP, en el caso de que fuera necesario, y para que el último pueda gestionar los recursos materiales o humanos a fin de obtener los resultados esperados en los plazos previstos y con la calidad necesaria.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



Antes de comenzar con el desarrollo, en primer lugar, se explica cómo se implementarán los permisos a roles de acceso al sistema y cómo se configuran las acciones a las que tendrán acceso dichos roles.

### 6.1 Implementación de permisos a los roles

La implementación de los roles en el sistema se basa en una clase “FMT/Roles”. En el proyecto, se crea un modelo llamado “AppRoles” que hereda (recibe las mismas propiedades y métodos) de la clase mencionada. Es en este modelo donde se definirá la propiedad “permisos”, que consiste en un *array*<sup>15</sup> con roles y permisos y que tiene la siguiente estructura:

- nombre: nombre del rol de tipo *string*.
- padre: id del rol padre del cual heredan sus permisos.
- permisos: *array* con los controladores a los que tiene acceso el rol.
- controlador: *array* con las acciones permitidas para el controlador autorizado.

La figura a continuación presenta la implementación de los permisos correspondiente a cada usuario:

---

<sup>15</sup>Los *arrays* (arreglos) o matrices son un tipo de estructuras de datos que nos permiten almacenar múltiples elementos de un tipo de datos similar, en una sola variable, a los que se accede utilizando su índice o clave (Leodorf y Tatroe, 2002).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```
1  <?php
2  namespace APP\modelo;
3  use FMT\Roles;
4
5  class AppRoles extends Roles
6  {
7      static $permisos = [
8          1 => [
9              'nombre' => 'Usuario principal',
10             'permisos' => [
11                 'PrincipalControlador' => [
12                     'alta' => 1,
13                     'baja' => 1,
14                     'modificacion' =>1
15                 ]
16             ]
17         ],
18         2 => [
19             'nombre' => 'Usuario Acceso',
20             'padre' => 1,
21             'permisos' => [
22                 'AccesoControlador' => [
23                     'alta' => 1,
24                     'baja' => 1,
25                     'modificacion' =>1
26                 ]
27             ]
28         ]
29     ]
30
31 ]
32 }
```

Figura 6.1.1. Implementación de los permisos por usuarios (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Como se puede ver, la figura 6.1.1 representa una estructura que tiene un esquema hereditario, de manera tal que la clase “Roles” evalúa los permisos que tiene cada rol que posee el usuario logueado<sup>16</sup> actualmente, para el controlador y acción analizados. Si no se encuentran permisos configurados y tiene rol padre, se evaluarán los permisos de este, y así, sucesivamente, hasta encontrar la configuración, es decir el rol que no tiene herencia. En el ejemplo, el rol “usuario acceso” hereda del “usuario principal”. El siguiente paso es agregar en el archivo *bootstrap* el *init()* de la clase FMT/FMT, tal como se ilustra en la figura 6.1.2

```

1  FMT/FMT::init([
2  |   'roles' => '\\App\\Modelo\\AppRoles'
3  | ]);

```

**Figura 6.1.2.** Inicialización de la clase *AppRoles* (imagen propia)

El objetivo de esto es propagar la configuración hacia la clase FMT\Controlador. Con esto, se da flexibilidad a la aplicación, que puede definir libremente la ruta y nombre de su clase de roles y también se evita en FMT\Controlador, el ‘hardcodeo’<sup>17</sup> de la ruta a la clase “roles local”.

En este punto de la implementación, todas las funcionalidades se bloquean, si el usuario no tiene los permisos correspondientes. Como resultado de esto, la aplicación redireccionará a una vista de error. Cuando la necesidad sea ocultar todas las funcionalidades para las que el usuario no tiene permisos, se evalúa previamente en el controlador el destino de todos los enlaces, íconos y botones que conforman

<sup>16</sup>El término “logueado” viene de “*login*” que es el nombre dado al momento de autenticarse e ingresar a un sistema (Alegsa, Leandro. 2017).

<sup>17</sup>“Hardcodeo” es un término que hace referencia a una mala práctica en el desarrollo de software, que consiste en incrustar datos directamente en el código fuente del programa (AA.VV., 2019h: s/p).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

la vista. La evaluación es por medio del método `AppRoles::puede()`, como se muestra en el siguiente ejemplo:

```
1 $flag_mod = AppRoles::puede('AccesoControlador', 'alta');
```

**Figura 6.1.3.** Condición para acciones restringidas (imagen propia)

Otra propiedad que se agrega a las listadas más arriba es el “atributo”. Esta propiedad tiene la posibilidad de asignarle permisos a un rol para acceder a campos de datos, pestañas de la aplicación, así como también al estado específico de una entidad. Por ejemplo, el usuario con rol de convenio solo podrá visualizar información de los agentes cuya modalidad de vinculación sea “Prestación de Servicios”.

## 6.2 Gestión de legajos

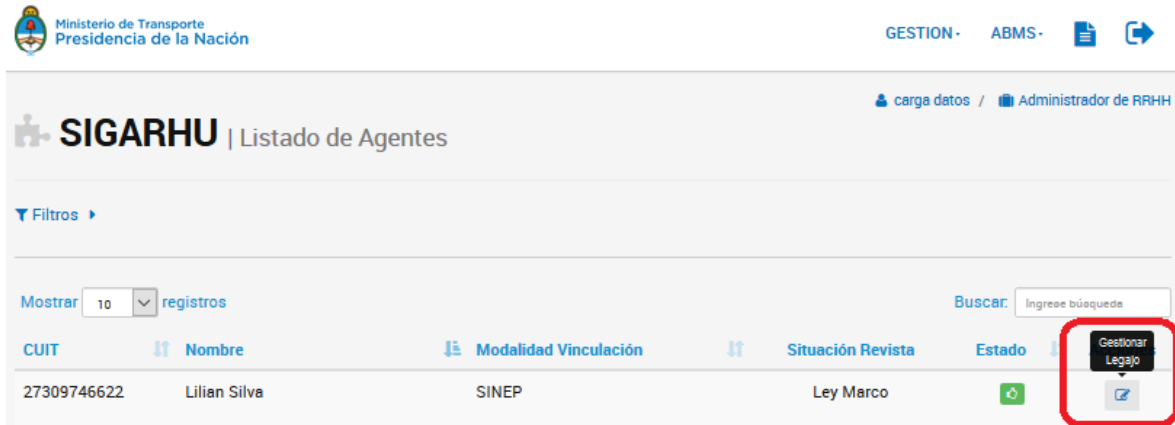
La vista principal del sistema cuenta con un listado de los agentes por CUIT, nombre, modalidad de vinculación, situación de revista, estado y acciones. Los agentes que mostrará el listado dependerán del rol que esté logueado en ese momento, ya que los diferentes roles gestionarán los datos de los agentes, dependiendo de las distintas modalidades de vinculación o situación de revista.

La modalidad de vinculación se refiere al tipo de contratación que posee el agente. Se puede mencionar como ejemplos “Autoridad superior”, “Extraescalafonario”, “Personal embarcado”, “Prestación de servicios”, “SINEP”, entre otras.

La situación de revista es el contrato que posee el agente; se establecen las siguientes posibilidades: “Comisión de servicios”, “Planta permanente con designación transitoria”, “CLM (Contrato Ley Marco)”, “Planta permanente1109/17” y “Asistencia técnica”.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

A continuación, se muestra el ejemplo de este listado de agentes:



**Figura 6.2.1.** Listado de agentes (imagen propia)

La gestión de legajos se inicia en el listado de agentes, en el botón de la acción “Gestionar Legajo” que se muestra resaltado en la figura 6.2.1; este botón lleva a las pestañas de la gestión. A continuación, se describe la secuencia:

1. Por medio del botón “Gestionar legajo”, el usuario realiza una solicitud a la siguiente URL **[http://local.transporte.gob.ar/sigarhu/index.php/legajos/gestionar/<CUIT\\_del\\_usuario>](http://local.transporte.gob.ar/sigarhu/index.php/legajos/gestionar/<CUIT_del_usuario>)**. El servidor web se encarga de la solicitud mediante la ejecución del script de arranque en index.php.
2. El archivo index.php establece el controlador y acción por default del sistema para el usuario logueado en el momento.
3. Además, el archivo index.php captura los parámetros que se le pasan a la URL (controlador, acción y un tercer parámetro, si existiese).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

4. La aplicación procesa la orden del usuario a través de los parámetros y esto lleva a un controlador y a una acción.
5. Para este ejemplo, el controlador es “Legajos” y la acción es “gestionar” además de agregarse el parámetro “CUIT” del agente. La acción “accion\_gestionar()” se muestra a continuación:

```
protected function accion_gestionar() {
    $empleado = Empleado::obtener($this->request->query('id'));

    $select_tab = '';

    switch ($this->request->post('gestionar_form')) {
        case 'datos_personales':
            $this->am_datos_personales($empleado);
            $select_tab = 'tab_ubicacion_estructura';
            break;
        case 'ubicacion_estructura':
            $this->am_ubicacion_estructura($empleado);
            $select_tab = 'tab_ubicacion_estructura';
            break;
        case 'antiguedad':
            $this->m_antiguedad($empleado);
            $select_tab = 'tab_ubicacion_estructura';
            break;
        default:
            # code...
            break;
    }

    $permisos['antiguedad'] = AppRoles::puede('antiguedad','alta');
    $vista = $this->vista;
    $parametricos = $this->datos_parametricos();
    $parametricos['sum_antiguedad'] = $this->sum_antiguedad($empleado->dependencia->fecha_desde);

    (new Vista($this->vista_default,compact('empleado','permisos', 'vista', 'parametricos', 'select_tab'))->pre_render());
}
```

Figura 6.2.2. Acción Gestionar en el Controlador “Legajos” (imagen propia)

La acción “gestionar”, que se muestra en la figura 6.2.2, obtiene una instancia de la clase empleado, que contiene todos los atributos del empleado, desde el modelo “Empleado”, a partir de la información ingresada por el usuario por medio de la URL, a través del *Get*, cuyo ID es el CUIT del agente.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:
-------------------	---------------------------	----------------------------	-----------------------------

Esta acción, además, permite mostrar otras solapas conforme la solapa que se seleccione. También se obtienen los datos sobre los permisos que tenga el usuario logueado para el acceso a las solapas y acciones por medio de la variable `$permiso`. La variable `$parametricos` obtiene los datos paramétricos necesarios de los que se hará uso en cada solapa.

La variable `$vista` va a 'instanciar' una vista, con la 'inicialización' de un *template*. Con esta variable, se muestran las cabeceras y el *footer* generales para toda la gestión de legajos. Por último, se crea una nueva instancia de la vista default, esto quiere decir que el archivo con extensión *php* que generará la vista llevará el mismo nombre de la acción; en este caso, se llamará "gestionar.php". La función "compact" toma las variables mencionadas anteriormente para pasarlas a la vista, convirtiendo la lista de variables en una matriz asociativa, donde la clave es el nombre de la variable

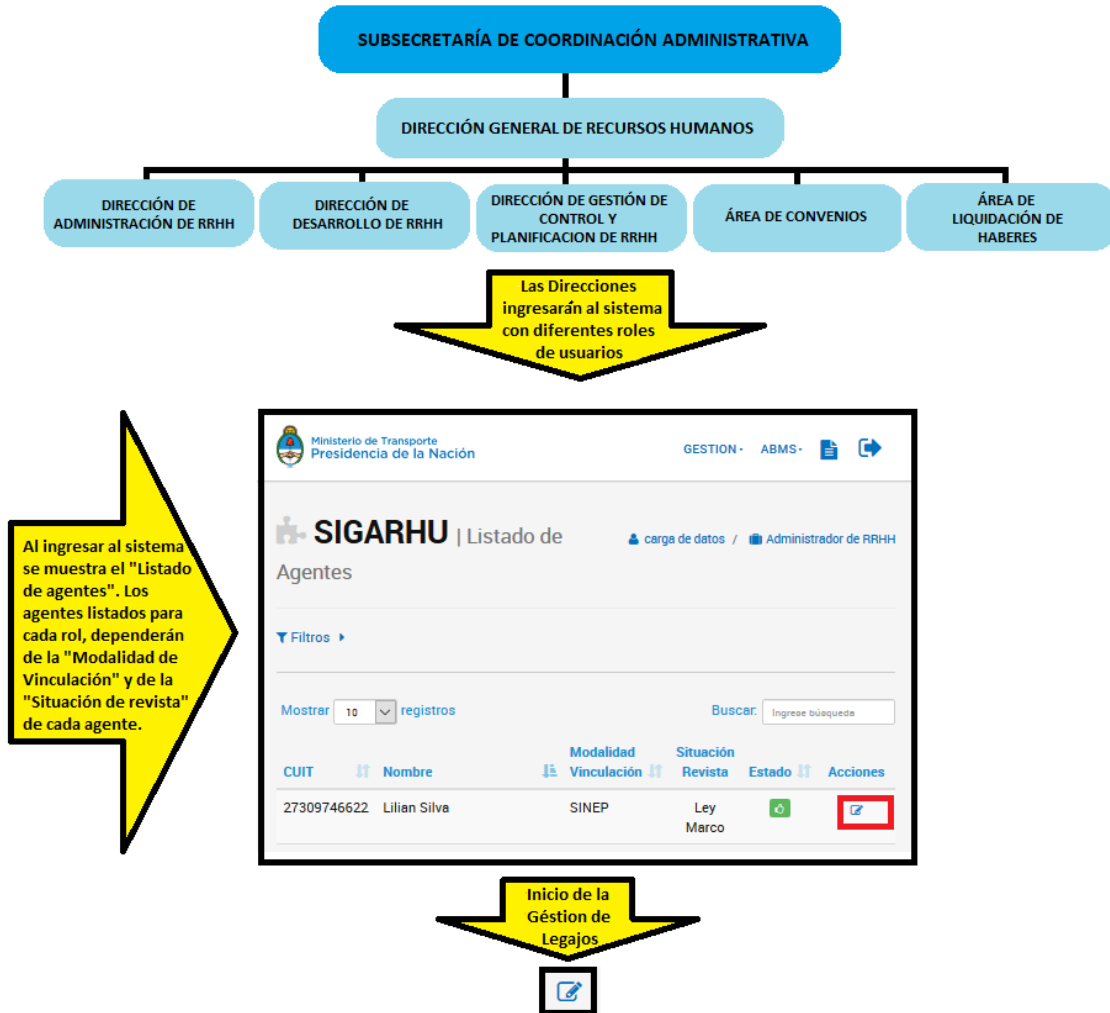
y el valor es el valor de ésta. Con lo anterior, se crea una vista base para componer lo que se va a mostrar al 'renderizar'<sup>18</sup>.

A continuación, se desarrolla un gráfico descriptivo para comprender las partes del sistema que se desarrollarán en el presente apartado.

---

<sup>18</sup>"Renderizar" es un término utilizado para referirse al proceso de generar una vista.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



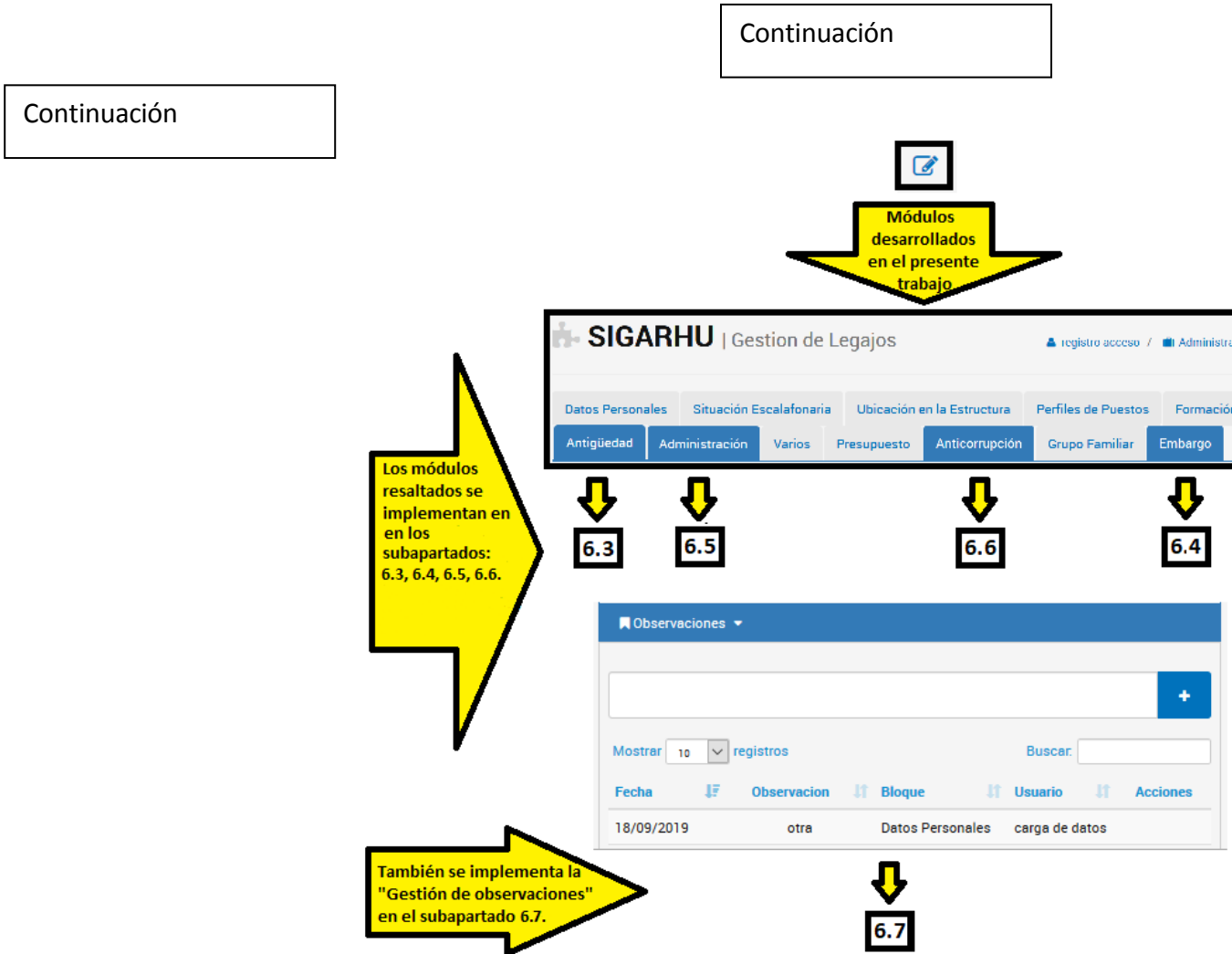


Figura 6.2.3. Esquema del Sistema con los módulos a desarrollar (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

A partir de aquí, se explicarán los bloques desarrollados dentro de la gestión de Legajos.

### 6.3 Implementación Bloque de antigüedad

El bloque de antigüedad permitirá agregar datos del agente respecto de su antigüedad como empleado en la administración pública y los años como empleado del Ministerio de Transporte.

Para que el formulario de antigüedad resulte accesible solo a los roles mencionados en la definición de tareas, se va a configurar el acceso, en el modelo “AppRoles” dentro de la propiedad permiso en el rol base, mediante el atributo de los correspondientes campos configurados con la constante booleana<sup>19</sup> en “0” (falso), para el controlador y acción. Todos los demás roles que heredan de Base padre, no tendrán acceso a modificarlos datos sobre la antigüedad, pero sí podrán visualizarlos. Los roles que tendrán permiso de edición a esta solapa serán el Rol administrador control RRHH y el Rol administrador convenios, por lo que, en estos, se redefinen los permisos para el controlador y la acción configurados con la constante 1 (verdadero).

Los campos que contiene la solapa de antigüedad solicitada son los siguientes:

- Campos años y meses: aquí se podrá seleccionar el año y el mes; esta combinación de datos se guardará como la antigüedad en la administración pública. Los datos que se guarden en el campo serán en formato *JSON*. Ejemplo: {"anio":"8","mes":"0"}
- Campo “Fecha Ingreso MTR”: el campo será un elemento HTML <INPUT>, de tipo “text”, con una clase “fecha”. Se utilizará para las fechas el componente *datePicker*. Los componentes de todas las

<sup>19</sup>Una “constantebooleana” es un valor perteneciente al conjunto {0,1} (Silva, 2010: 1).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

librerías se agregan en el controlador “Base” por medio del método “head”, y se incluyen al generar las vistas.

- Campo “Contador de Antigüedad MTR”: es únicamente informativo, lo que significa que no se guardará en la base de datos. Se mostrarán de los años, meses y días, y se calculará a partir de la fecha del campo “Fecha Ingreso MTR” hasta la fecha actual.

A continuación, se muestra una imagen de los campos de la solapa “Antigüedad”:



Agente: Silva Lilian | Cuit: 27309746622 | Cargo: -

**Antigüedad en la Administración Pública**

Años: 2 | Meses: 5

**Antigüedad en el Ministerio**

Fecha Ingreso MTR: 13/03/2019 | Contador Antigüedad MTR: 0 años, 4 meses, 2 días

Figura 6.3.1. Campos de la Solapa de antigüedad (imagen propia)

Para comenzar con el desarrollo de la pestaña “Antigüedad”, se muestra el flujo de datos:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

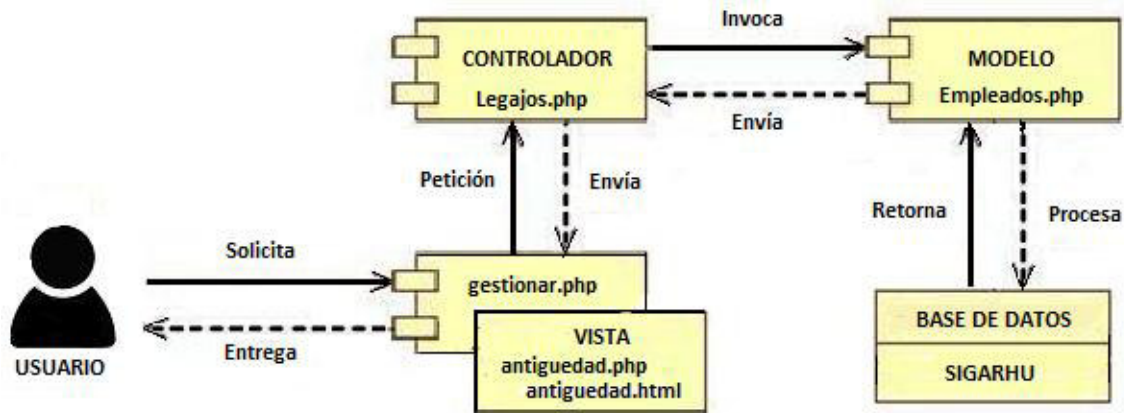


Figura 6.3.2. MVC para el Bloque de antigüedad (imagen propia)

El método privado “m\_antigüedad” es invocado en la acción “gestionar” del controlador “Legajos.php” al momento de enviar los datos cargados en el formulario por medio del *POST*. Este comportamiento se ejemplifica en la figura 6.2.2. El método “m\_antigüedad” recibe la instancia del empleado, luego de asignar los valores que el usuario ingrese o modifique desde la interfaz (Vista), a partir del formulario de antigüedad; luego, los valores pasan por la validación en el modelo para que la información ingresada sea correcta y, de esa manera, se ingresan o modifican los datos desde el modelo “Empleado” en la base de datos.

Un ejemplo de lo descrito anteriormente se muestra en la siguiente figura 6.3.3.:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```
private function m_antiguedad(Empleado $Empleado=null){
    $control = clone $Empleado->antiguedad;
    $antiguedad = ($this->request->post('antiguedad_adm_publica')) ?
        $this->request->post('antiguedad_adm_publica') : ['anio' => 0, 'mes' => 0];
    $Empleado->antiguedad->antiguedad_adm_publica = (object)$antiguedad;
    $Empleado->antiguedad->fecha_ingreso = \DateTime::createFromFormat('d/m/Y H:i:s',
        $this->request->post('fecha_ingreso_mtr'). ' 00:00:00');

    if($Empleado->validar()){
        if($Empleado->cuit) {
            if($control->antiguedad_adm_publica !== $Empleado->antiguedad->antiguedad_adm_publica )
                $Empleado->modificacion();
            if($control->fecha_ingreso !== $Empleado->antiguedad->fecha_ingreso)
                $Empleado->modificacion_ingreso();
            $this->mensajería->agregar(
                "La antigüedad del empleado <strong>{$Empleado->persona->nombre} {$Empleado->persona->apellido}</strong>
                fue modificada exitosamente.",
                \FMT\Mensajería::TIPO_AVISO,
                $this->clase
            );
        }
    } else {
        foreach ($Empleado->errores as $text) {
            $this->mensajería->agregar($text, \FMT\Mensajería::TIPO_ERROR, $this->clase);
        }
    }
}
```

Figura 6.3.3. Método “m\_antiguedad” (imagen propia)

En el controlador “Legajos” y en la acción “gestionar”, se agrega un nuevo dato paramétrico “sum\_antiguedad”, que va a permitir mostrar el campo informativo “contador antigüedad MTR” a partir de la fecha de ingreso que forma parte de los atributos de la antigüedad del empleado debidamente configurado (si los datos del empleado se encuentran configurados con las constantes).

Para realizar el cálculo de la antigüedad, se utilizará una función protegida “sum\_fecha()” dentro del mismo controlador, que se encargará tanto de recibir la fecha como parámetro y a partir de la fecha actual, como de calcular el tiempo transcurrido desde entonces.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Por una parte, el generador de vista se compone de una vista en código PHP la cual obtiene los datos de la consulta y, por otra parte, de una vista en código HTML, a fin de evitar incrustar código PHP dentro del HTML y mantener un código separado y más ordenado.

Para ello, se crean las vistas, en primer lugar, el archivo llamado “antigüedad.php”, que contiene la información de la consulta obtenida por medio del controlador “legajo”; se preparan las etiquetas que se van a reemplazar cuando se crea, en segundo lugar, una vista “antigüedad.html”. La imagen que se muestra a continuación es del archivo “antigüedad.php” y servirá de ejemplo para entender, de ahora en más, cómo se arman las vistas php.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```

<?php
use App\Helper\Vista;
$vars_template = [];
$campos_antiguedad = [
    'FECHA_INGRESO_MTR' => !empty($temp = $empleado->antiguedad->fecha_ingreso) ? $temp->format('d/m/Y') : '',
    'CONTADOR_ANTIGUEDAD' => \FMT\Helper\Arr::get($parametricos, 'sum_antiguedad') ? $parametricos['sum_antiguedad'] : '',
];

$anios = [];
for ($i=1; $i <= 50 ; $i++) {
    $anios[$i] = $i;
}
$campos_antiguedad['ANIO'] = \FMT\Helper\Template::select_block($anios, $empleado->antiguedad->antiguedad_adm_publica->anio);

$meses = [];
for ($i=1; $i <= 12 ; $i++) {
    $meses[$i] = $i;
}
$mes = (isset($empleado->antiguedad->antiguedad_adm_publica->mes)) ? $empleado->antiguedad->antiguedad_adm_publica->mes : 0;
$campos_antiguedad['MES'] = \FMT\Helper\Template::select_block($meses, $mes );

if (!empty($empleado->id) && !empty($empleado->cuit)){
    if($permisos['antiguedad']){
        $vars_template['CAMPOS_ANTIGUEDAD'][0] = [
            'FORM' => \App\Helper\Vista::get_url("index.php/legajos/gestionar/{$empleado->cuit}"),
        ];
        foreach ($campos_antiguedad as $key => $value) {
            $vars_template['CAMPOS_ANTIGUEDAD'][0][$key] = $value;
        }
        $vars_template['CAMPOS_ANTIGUEDAD'][0]['CONTADOR_ANTIGUEDAD'] = $parametricos['sum_antiguedad'];
    } else {
        $campos_antiguedad['ANIO'] = $empleado->antiguedad->antiguedad_adm_publica->anio;
        $campos_antiguedad['MES'] = $mes;
        foreach ($campos_antiguedad as $key => $value) {
            $vars_template['SPAN_ANTIGUEDAD'][0][$key] = $value;
        }
        $vars_template['SPAN_ANTIGUEDAD'][0]['CONTADOR_ANTIGUEDAD'] = $parametricos['sum_antiguedad'];
    }
} else {
    $vars_template['AVISO'][0]['MSJ'] = 'PARA DEFINIR LA <strong>ANTIGÜEDAD</strong>,
    ES REQUISITO TENER LOS DATOS BÁSICOS DEL <strong>AGENTE</strong> COMPLETOS,
    <strong>SITUACIÓN ESCALAFONARIA</strong> y <strong>UBICACIÓN EN LA ESTRUCTURA.</strong>.';
}
$antiguedad = new \FMT\Template(TEMPLATE_PATH.'/legajos/antiguedad.html', $vars_template,['CLEAN'=>false]);
  
```

Figura 6.3.4. *Antigüedad.php* (imagen propia)

Como se ve en la Figura 6.3.4, se inicializa la variable `$vars_template`, en principio, como un *array* vacío. Esta va a contener los valores a reemplazar en las etiquetas de reemplazo en el archivo “antiguedad.html”.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

La variable `$antiguedad_adm_publica` contiene la información sobre el año y mes de ingreso del empleado. Como anteriormente había sido guardada en formato *JSON*, al obtenerla de la base de datos, habrá que volver el contenido a una variable PHP para poder mostrarla en el formulario de la vista. Para eso, se utiliza la función PHP denominada “*json\_decode*” la cual convierte un *string*<sup>20</sup> codificado en *JSON* a una variable de PHP.

La variable `$campos_antiguedad` va a contener el bloque de datos que encierra al código HTML y que contiene las etiquetas que serán reemplazadas.

El bloque de antigüedad debe contener datos del empleado precargados para poder acceder; para ello, se pone la condición de que el id o CUIT del empleado no pueden estar vacíos; caso contrario, se envía un mensaje de aviso, donde se exprese que esos datos no existen.

También hay una validación para los permisos de acceso a la pestaña. Si estos permisos existen, se muestran los datos para agregar o modificar; en caso contrario, serán campos de tipo `<SPAN>` inhabilitados.

En la última línea de la figura 6.3.4, se muestra la vista al usuario, utilizando la Clase `FMT/Template`, pasando, en primer lugar, la ruta de la ubicación del archivo base para realizar los reemplazos, y el arreglo que contiene las etiquetas y valores que se reemplazarán en el *template* o archivo HTML. La variable que contiene el *array* de datos es `$vars_template`. El resultado de la vista va a ser el contenido del archivo HTML final. La acción del controlador completa la vista realizada y se la muestra al usuario.

---

<sup>20</sup>Un *string* es una cadena de caracteres (Leodorfy Tatroe, 2002).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



## 6.4 Implementación del Bloque de embargo

### 6.4.1. Listado de embargos

Para este bloque, se muestra un listado de los embargos vigentes que tiene el agente, con otras opciones: 1) la opción de cargar nuevos embargosa través del botón “nuevo”; 2) poder ver un listado de antiguos embargos del agente por medio del botón“historial”;o 3) volver a la gestión de Legajos mediante el botón “volver”.A continuación, se muestra la vista principal de la solapa de embargos:



SIGARHU | Gestion de Legajos  
 Administrador de RRHH  
 Agente: Silva Lilian | Cuit:27309746622 | Cargo:-  
 Embargos

Tipo Embargo	Autos	Fecha Alta	Fecha Cancelación	Monto	Acciones
Familiar	afdsfsad	27/07/2019	20/09/2019	45 %	 

Observaciones ▶  
 HISTORIAL NUEVO VOLVER

**Figura 6.4.1.1.**Listado de Embargos (imagen propia)

Como se muestra en la figura 6.4.1.1., se solicita que los embargos vigentes se muestren en forma de lista, con información sobre el tipo de embargo, autos (nombre de la resolución judicial de un embargo), fecha de alta, fecha de cancelación, monto y acciones posibles frente a los embargos, tales como modificación y baja o eliminación.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Por una parte, con la acción “gestionar”, del controlador legajos, se obtiene un listado de los embargos del agente y, por otra, se accede al listado de los embargos en el “historial” por medio del botón “Historial”. Este botón solo es visible si el agente analizado tiene embargos vencidos; en caso contrario, el botón permanece oculto en la vista del usuario. A continuación, se muestra cómo se hace uso de los métodos del modelo “Embargo” que obtiene la información pasando como parámetro el “id” del agente analizado en el controlador.

```
$lista_embargo = Modelo\Embargo::listar($empleado->id);
$lista_historial_embargo = Modelo\Embargo::listar_historial($empleado->id);
```

**Figura 6.4.1.2.** Variables que obtienen los listados de embargo y embargos en historial en el controlador “Legajos” y la acción “Gestionar” (imagen propia)

En la acción “gestionar”, también se configuran los permisos, tanto para agregar un nuevo embargo, como para poder ver o no la pestaña de embargos, ya que, en este caso, algunos usuarios con otros permisos, no deben ver la pestaña.

```
$permisos['embargo'] = AppRoles::puede('Legajos','alta_embargo');
$permisos['bloque_embargo'] = $solo_lectura && AppRoles::puede_atributo('Embargos','index','tab_visible','embargo');
```

**Figura 6.4.1.3.** Permisos de acceso o edición de datos para la pestaña de embargos (imagen propia)

La vista que recibe la lista de embargos se muestra en un *Datatable*. Para las acciones del listado, se consideran los permisos del usuario logueado; como se dijo anteriormente, estas acciones van a ser modificar o dar de baja los datos de los embargos cargados para el empleado.

A continuación, se muestra el archivo “embargo.php” que compone la vista para mostrar el listado de embargos.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```

<?php
use \FMT\Helper\Template;
use \FMT\Helper\Arr;
if($permisos['bloque_embargo']) {
    $vars_template = [];
    $vars_template['TITULOS'] = [
        ['TITULO' => 'Tipo Embargo'],
        ['TITULO' => 'Autos'],
        ['TITULO' => 'Fecha Alta'],
        ['TITULO' => 'Fecha Cancelación'],
        ['TITULO' => 'Monto'],
    ];
}

foreach ($lista_embargo as $key => $em) {
    ($em['tipo_embargo'] == 'Ejecutivo') ? $signo = ' $' : $signo = ' %';
    $vars_template['ROW'][$key] = [
        ['COL' => [
            ['CONT' => $em['tipo_embargo']],
            ['CONT' => $em['autos']],
            ['CONT' => $em['fecha_alta']],
            ['CONT' => $em['fecha_cancelacion']],
            ['CONT' => $em['monto'] . $signo],
        ]],
    ];
    if($permisos['embargo']) {
        $acciones = '
        <span class="acciones">
        <a href="'.\App\Helper\Vista::get_url("index.php/legajos/modificar_embargo/{$em['id']}")"
        ' data-toggle="tooltip" data-placement="top" data-id="" title="Modificar" data-toggle="modal"><i class="fa fa-pencil"></i></a>
        <a href="'.\App\Helper\Vista::get_url("index.php/legajos/baja_embargo/{$em['id']}")"
        ' data-toggle="tooltip" data-placement="top" data-id="" title="Eliminar" data-toggle="modal"><i class="fa fa-trash"></i></a>
        </span>
        ';
        $vars_template['ROW'][$key]['COL'][0]['CONT'] = $acciones;
    }
}

if(!empty($lista_historial_embargo)) {
    $vars_template['HISTORIAL'][] = ['URL_HISTORIAL' =>
    \App\Helper\Vista::get_url("index.php/legajos/historial_embargo/{$empleado->cuit}");
}

if($permisos['embargo']) {
    $vars_template['TITULOS'][]['TITULO'] = 'Acciones';
    $vars_template['BOTON_NUEVO'][] = ['LINK' =>
    \App\Helper\Vista::get_url("index.php/legajos/alta_embargo/{$empleado->cuit}"), ];
}
$vars_template['BOTON_VOLVER'][] = ['VOLVER' =>
    \App\Helper\Vista::get_url("index.php/legajos/gestionar/{$empleado->cuit}"), 'CLASS' => "btn btn-default"];
$vars_template['TABLA'][] = new \FMT\Template(TEMPLATE_PATH.'/tabla.html', $vars_template, ['CLEAN'=>false]);
$embargos = new \FMT\Template(TEMPLATE_PATH.'/legajos/embargo.html', $vars_template, ['CLEAN'=>false]);
}
    
```

Figura 6.4.1.4. Archivo "embargo.php" (imagen propia)

En la Figura 6.4.1.4, se puede ver que la variable `$vars_template` almacena toda la información necesaria para reemplazar en el *template* o plantilla, en principio, como un arreglo vacío;

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

asimismo, se puede ver que se asignan los títulos que va a tener la tabla. La variable \$lista\_embargo, como habíamos dicho, obtiene la lista de embargos del empleado, que se itera para completar la tabla. También se puede ver la etiqueta para el botón historial, que va a dirigir al usuario hacia el listado de historial de embargos; se va a mostrar solo si existen datos en el listado de historial. También se mostrará el botón “Nuevo”, si el usuario cuenta con los permisos para agregar un nuevo embargo. Para mostrar todo lo mencionado, en la figura se puede ver el uso de la clase *Template*<sup>21</sup>, que se encarga de completar una plantilla llamada “tabla.html”, pasando como parámetro la ruta completa hasta a ésta, junto con la variable \$vars\_template, compuesta por todos los valores a reemplazar para componer una tabla, lo que rellena las etiquetas mediante la iteración mencionada. A continuación, se muestra la figura de la plantilla “tabla.html”.

```

<table id="tabla" class="display table table-striped {{{CLASS}}}" cellspacing="0" width="100%">
  <thead>
    <tr>
      {{{TITULOS}}}
      <th>{{{TITULO}}}</th>
    </tr>
  </thead>
  <tbody>
    {{{ROW}}}
    <tr>
      {{{COL}}}
      <td {{{EXTRAS}}}>{{{CONT}}}</td>
      {{{/COL}}}
    </tr>
  </tbody>
</table>
  
```

Figura 6.4.1.5. Archivo *template* “tabla.html” (imagen propia)

<sup>21</sup> Clase *Template* es una clase para el manejo y composición de *templateo* plantillas HTML (ORIGEN DE LA INFO).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

En la figura 6.4.1.5 del archivo “tabla.html”, se observan las etiquetas a reemplazar; todas las etiquetas están envueltas por llaves triples {{{ETIQUETA\_SIMPLE}}}, o también por{{{ETIQUETA\_SIMPLE.nombre\_propiedad}}}, incrustadas en un documento *HTML*, al que denominamos *template*, lo que nos permite separar el código *PHP* del *HTML*.

Finalmente, se compone la plantilla final “embargo.html” mediante la clase *Template*. Esta se muestra a continuación:

```
<div class="row subtítulo">
  <div class="col-xs-12 text-left"><h6 class="dom"><i class="fa fa-gavel"></i> Embargos</h6></div>
</div>
<div class="row">
  <div class="col-md-12">
    {{{TABLA}}}
  </div>
  <div class="col-md-12 text-right">
    {{{HISTORIAL}}}
    <a class="btn btn-info" href="{{URL_HISTORIAL}}" data-toggle="tooltip"
      data-placement="top" data-id="" title="Historial de Embargos" data-toggle="modal" target = "_self" ><i>HISTORIAL</i></a>
    {{{/HISTORIAL}}}
    {{{BOTON_NUEVO}}}
    <a class="btn btn-primary" href="{{LINK}}">NUEVO</a>
    {{{/BOTON_NUEVO}}}
    {{{BOTON_VOLVER}}}
    <a href="{{HREF}}" id="{{ID}}" class="{{CLASS}}" data-bloque="{{BLOQUE}}" data-ref="{{VOLVER}}">VOLVER</a>
    {{{/BOTON_VOLVER}}}
  </div>
</div>
```

Figura 6.4.1.6. Archivo *template* “embargo.html” (imagen propia)

La etiqueta “{{{TABLA}}}" hace referencia al archivo “tabla.html”, donde se cargan los datos previamente definidos. También se pueden ver las etiquetas que muestran los botones, con el objetivo de ver el historial, por medio de las etiquetas {{{HISTORIAL}}}; para ver el botón nuevo a través de las etiquetas {{{BOTON\_NUEVO}}}; y, a fin de observar el botón volver, con las etiquetas {{{BOTON\_VOLVER}}}.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

### 6.4.2. Estructura de datos para embargos del empleado

Para preparar la pestaña de embargo, en primer lugar, se creó una nueva tabla en la base de datos denominada “Embargos”, con los campos solicitados en los requerimientos. A partir de los campos presentados en la figura 6.4.2.1, se crea un nuevo modelo “Embargos”.

```

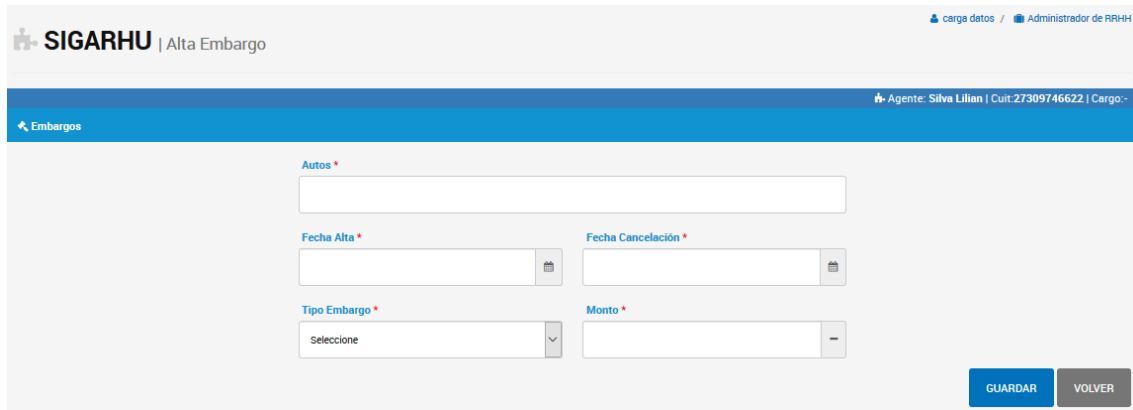
CREATE TABLE `embargos` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `id_empleado` INT(11) NOT NULL,
  `tipo_embargo` TINYINT(1),
  `autos` VARCHAR(255) NULL,
  `fecha_alta` DATE NULL DEFAULT NULL,
  `fecha_cancelacion` DATE NULL DEFAULT NULL,
  `monto` VARCHAR(45) NULL,
  `borrado` TINYINT(1) NULL DEFAULT '0',
  PRIMARY KEY (`id`),
  INDEX `index1` (`id_empleado` ASC),
  CONSTRAINT `fk_embargos_1`
    FOREIGN KEY (`id_empleado`)
    REFERENCES `empleados` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
  
```

Figura 6.4.2.1. Script SQL para la creación de la tabla Embargos (imagen propia)

### 6.4.3. Formulario de Alta de embargos

El botón “Nuevo” que se muestra en el listado de Embargos y que presentamos anteriormente, conduce a la acción “alta\_embargo” en el controlador “Legajos”, mostrando un formulario con los campos solicitados, tal como puede verse en la siguiente figura:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



The screenshot shows a web interface for 'SIGARHU | Alta Embargo'. At the top right, there are links for 'carga datos' and 'Administrador de RRHH'. Below the header, a blue bar indicates the user is 'Agente: Silva Lilian | Cuit: 27309746622 | Cargo:'. The main content area is titled 'Embargos' and contains the following form fields:

- Autos \***: A text input field.
- Fecha Alta \***: A date picker field.
- Fecha Cancelación \***: A date picker field.
- Tipo Embargo \***: A dropdown menu with 'Seleccione' as the current selection.
- Monto \***: A numeric input field with a minus sign on the left.

At the bottom right of the form, there are two buttons: 'GUARDAR' (highlighted in blue) and 'VOLVER' (greyed out).

**Figura 6.4.3.1** Formulario para el alta y modificación de embargos (imagen propia)

Los campos para el ingreso de datos de la figura se describen a continuación:

**Autos:** nombre de la resolución judicial de un embargo, por lo que el campo será un elemento HTML <TEXTAREA> para el ingreso de datos de tipo texto.

**Fecha de Alta y Fecha cancelación:** estos campos serán para la fecha de alta del embargo y la fecha de cancelación del embargo; ambos serán elementos HTML <INPUT>. En este caso, se vuelven a utilizar las librerías *bootstrap-datetimepicker* para añadir a los campos de tipo fecha, formatos de calendarios.

**Tipo Embargo:** el campo será un seleccionable entre las opciones “Ejecutivo” y “Familiar”.

**Monto:** este campo estará relacionado con el campo de tipo de “embargo”; en él, se permitirá introducir texto en un elemento HTML <INPUT>. El campo tendrá visible un elemento SPAN que se verá modificado de acuerdo con lo que se seleccione en el campo de tipo embargo; allí se mostrará un signo “\$”, si se selecciona la opción “Ejecutivo”, o un signo “%”, en el caso de la opción “Familiar”.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Además de los campos, el formulario tiene los botones “Guardar” y “Volver”, que se describen seguidamente:

Guardar: redirige al alta de un nuevo embargo.

Volver: retorna a la vista anterior respecto de la posición actual.

En la figura 6.4.3.2, se muestra la configuración para el envío de los datos cargados en el formulario, que se mandan al servidor al presionar el botón “Guardar”.

```

<form action="{{FORM}}" method="POST" autocomplete="off" enctype="multipart/form-data" >
  <button class="btn btn-primary" type="submit" data-toggle="tooltip" data-placement="top"
    name="boton_embargo" value="{{OPERACION}}" id='boton_embargo' title="Guardar" {{DISABLED}}>Guardar</button>
</form>
  
```

**Figura 6.4.3.2.** Ejemplo de configuración para el procesamiento de datos que se envía al servidor, fragmento del archivo *formulario\_embargo.html*(imagen propia)

El `type="submit"` del elemento `<button>` de la figura permite enviar los datos del formulario una vez que el usuario haya rellenado todos sus campos. Al pulsar el botón, se deben procesar los datos por servidor. Para esto, se configuran los atributos (`method = "POST"`).

La operación a configurar en el atributo `"value"` del `<button>` será `"Alta"`. A continuación, se muestra la configuración de la operación en el archivo `alta_embargo.php`:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



```
<?php
use App\Helper\Vista;
use \FMT\Helper\Arr;

$vars_vista['SUBTITULO'] = 'Alta Embargo';
$vars_template = [];
$vars_template['OPERACION'] = 'alta';
$vars_template['AUTOS'] = $embargo->autos;
$vars_template['TIPO_EMBARGO'] = \FMT\Helper\Template::select_block($tipo_embargo, $embargo->tipo_embargo);
$vars_template['FECHA_ALTA'] = !empty($temp = $embargo->fecha_alta) ? $temp->format('d/m/Y') : '';
$vars_template['FECHA_CANCELACION'] = !empty($temp = $embargo->fecha_cancelacion) ? $temp->format('d/m/Y') : '';
$vars_template['MONTO'] = $embargo->monto;
$vars_template['VOLVER'] = Vista::get_url("index.php/legajos/gestionar/{$empleado->cuit}") ;
$vars_template['BLOQUE'] = \App\Helper\Bloques::EMBARGO;
$form_embargos = new \FMT\Template(TEMPLATE_PATH.'/legajos/formulario_embargo.html', $vars_template,['CLEAN'=>false]);

$vars_vista['CSS_FILES'][][] = ['CSS_FILE' => \App\Helper\Vista::get_url('legajos.css')];
$vars_vista['JS_FOOTER'][][]['JS_SCRIPT'] = Vista::get_url('embargo.js');
$vars_vista['CONTENT'] = "{$form_embargos}";

$vista->add_to_var('vars',$vars_vista);
return true;
```

Figura 6.4.3.3. Archivo “alta\_embargo.php” (imagen propia)

En la figura, la variable `$vars_vista` configura los valores a reemplazar en el `template` base, donde se incluyen la cabecera, pie y títulos que se muestran en toda la gestión de legajos; y `$vars_template` acumula todos los valores para la solapa embargo que va a conformar el `<content>` en el documento completo, lo que significa incorporarlo al `template` base y obtener el `template` completo.

Similar a este archivo de la vista, también habrá un archivo para el cambio o modificación del embargo, en el que cambia la operación (en lugar de “alta”, la operación será “modificación”) y se utilizará el mismo `template` previamente mencionado (`formulario_embargo.html`).

A continuación, se muestra el archivo “embargo.js”, donde se observa el código *JavaScript* con las funcionalidades añadidas a los campos de “fecha”, “tipo embargo” y “monto” y el botón “volver”.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```

$(document).ready(function () {
  $(function () {
    $('#fecha').datetimepicker({
      format: 'DD/MM/YYYY'
    });

    var $btn = $('#volver_legajo');
    var f = $('<form/>', {id:'form_l' , action : $btn.data('ref'), method : 'POST'});
    var input = $('<input />', { name: 'id_bloque', type: 'hidden', value: $btn.data('bloque') });
    f.append(input);
    $btn.after(f);
    $btn.click(function(){
      $('#form_l').submit();
    });
    var tipo_embargo= $('#tipo_embargo').val();
    span_monto(tipo_embargo);

    $('#tipo_embargo').on('change', function(){
      var tipo_embargo= $(this).val();
      span_monto(tipo_embargo);
    });
  });
});

function span_monto(tipo_embargo){
  if(tipo_embargo == ''){
    $('#span_monto').removeClass().addClass( "fa fa-minus" );
  }
  if(tipo_embargo == 1){
    $('#span_monto').removeClass().addClass( "fa fa-usd" );
  }
  if(tipo_embargo == 2){
    $('#span_monto').removeClass().addClass( "fa fa-percent" );
  }
}

```

Figura 6.4.3.4. Archivo “embargo.js” (imagen propia)

En la parte superior de la figura, se inicializa la librería *datetimepicker* asignando la clase “fecha” del elemento <input> para la fecha de alta y cancelación.

Más abajo, se declara una variable \$btn con selector de *jQuery*. Aquí, se utiliza el “Id” del elemento HTML del botón “volver” con un mecanismo para volver a la vista anterior a la actual.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Posteriormente, se determina una variable llamada “tipo\_embargo” que obtiene el valor del elemento con id “tipo\_embargo” en el código HTML y, luego, llama a la función “span\_monto”, cuyo parámetro es el tipo de embargo obtenido, mostrando tres posibles signos: en primer lugar, el signo “-”, en el caso de que el valor está vacío; el signo “\$”, si el tipo de embargo es “Ejecutivo”, o ID 1 y, finalmente, el signo “%”, en el caso de que el tipo de embargo sea “Familiar”, o ID 2. El sistema se comportará de igual forma cuando el campo de tipo de embargo sea modificado por el usuario para obtener el signo que corresponda mediante el valor seleccionado.

Una vez cargados los campos en el formulario de embargos, se presiona el botón “Guardar”, con la operación alta. En esta instancia, se guardarán en la base de datos los nuevos datos cargados, siendo previamente validados. En la siguiente imagen, se muestra la acción alta\_embargo en el controlador Legajos.php:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```

protected function accion_alta_embargo() {
    $empleado = Empleado::obtener($this->request->query('id'));
    $embargo = Embargo::obtener();
    if (empty($empleado->id) && empty($empleado->cuit)){
        $tipo_embargo = App\Modelo\Embargo::getParam('TIPO_EMBARGO');
        $embargo->id_empleado = $empleado->id;
        $embargo->tipo_embargo = (empty($this->request->post('tipo_embargo')) ? (int)$this->request->post('tipo_embargo') : $embargo->tipo_embargo);
        $embargo->autos = ($temp = $this->request->post('autos')) ? $temp : null;
        $embargo->fecha_alta = ($temp = $this->request->post('fecha_alta')) ? \DateTime::createFromFormat('d/m/Y', $temp) : null;
        $embargo->fecha_cancelacion = ($temp = $this->request->post('fecha_cancelacion')) ? \DateTime::createFromFormat('d/m/Y', $temp) : null;
        $embargo->monto = ($temp = $this->request->post('monto')) ? $temp : null;

        if($this->request->post('boton_embargo') == 'alta') {
            if($embargo->validar()){
                $embargo->alta();
                $this->mensajeria->agregar(
                    "El Embargo del empleado <strong>{$empleado->persona->nombre} {$empleado->persona->apellido}</strong> fue cargado exitosamente.",
                    \FMT\Mensajeria::TIPO_AVISO,
                    $this->clase
                );
                $select_tab = 'tab_embargo';
                $_SESSION['data_legajo']['select_tab'] = $select_tab;
                $redirect = Vista::getUrl("index.php/legajos/gestionar/{$empleado->cuit}");
                $this->redirect($redirect);
            }else {
                $serr = $embargo->errores;
                foreach ($serr as $text) {
                    $this->mensajeria->agregar($text, \FMT\Mensajeria::TIPO_ERROR, $this->clase);
                }
            }
        }

        $vista = $this->vista;
        $nombre_de_puesto = App\Modelo\Perfil::listarNombrePuestos();
        $puesto = \FMT\Helper\Arr::path($nombre_de_puesto, "{$empleado->perfil_puesto->nombre_puesto}.nombre", '-');
        $vars = ['AGENTE' => $empleado->persona->apellido, 'EMPLEADO' => $empleado->persona->nombre, 'CUIT' => $empleado->cuit];
        if($puesto != '') {
            $vars['DENOMINACION'] = $puesto;
        }
        $vista->add_to_var('vars', $vars);
        (new Vista($this->vista_default, compact('vista', 'permisos', 'empleado', 'tipo_embargo', 'embargo'))->pre_render());
    }else{
        $this->mensajeria->agregar("PARA DEFINIR EL <strong>EMBARGO</strong>, ES REQUISITO TENER LOS DATOS BÁSICOS DEL <strong>AGENTE</strong>.", \FMT\Mensajeria::TIPO_ERROR, $this->clase);
    }
}

```

Figura 6.4.3.5. Acción "alta\_embargo()" en el controlador "Legajos" (imagen propia)

En la acción "alta\_embargo" del controlador, se puede ver la manera en que se reciben los datos que vienen del formulario, y, antes de guardar, por medio del llamado al método "Alta" del modelo "Embargos", se analiza si la acción es alta y si pasa las validaciones que corroboran que los valores que se completan en los campos son correctos.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

#### 6.4.4. Formulario de Modificación de embargos

Como establecimos en la sección previa, el formulario para la modificación es el mismo que el utilizado para el alta, ejemplificado en la figura 6.4.3.1; la diferencia es que este formulario vendrá con campos de datos cargados previamente. Se ingresa a la modificación a través del listado de embargos de la figura 6.4.1.1, por medio del icono “modificar” expuesto en la siguiente figura:



**Figura 6.4.4.1.** Icono para el acceso al formulario de modificación de embargos (imagen propia)

La “operación” en la figura 6.4.3.3 esta vez será “modificación” para el atributo “value” del <button>

La siguiente figura muestra la acción “modificación”, con un comportamiento similar a la acción “alta\_embargo”.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```
protected function accion_modificar_embargo() {
    $embargo = Embargo::obtener_embargo($this->request->query('id'));
    $empleado = Empleado::obtener($embargo->id_empleado, true);
    if (!empty($empleado->id) && !empty($empleado->cuit)) {
        $tipo_embargo = \App\Modelo\Embargo::getParam('TIPO_EMBARGO');
        $embargo->id_empleado = $empleado->id;
        $embargo->tipo_embargo = !empty($this->request->post('tipo_embargo')) ?
            (int)$this->request->post('tipo_embargo') : $embargo->tipo_embargo;
        $embargo->autos = !empty($temp = $this->request->post('autos')) ? $embargo->autos;
        $embargo->fecha_alta = !empty($temp = $this->request->post('fecha_alta')) ?
            \DateTime::createFromFormat('d/m/Y', $temp) : $embargo->fecha_alta;
        $embargo->fecha_cancelacion = !empty($temp = $this->request->post('fecha_cancelacion')) ?
            \DateTime::createFromFormat('d/m/Y', $temp) : $embargo->fecha_cancelacion;
        $embargo->monto = !empty($temp = $this->request->post('monto')) ? $temp : $embargo->monto;
        if ($this->request->post('boton_embargo') == 'modificacion') {
            if ($embargo->validar()) {
                $embargo->modificacion();
                $this->mensajeria->agregar(
                    "El embargo del empleado <strong>{$empleado->persona->nombre} {$empleado->persona->apellido}</strong>
                    fue modificado exitosamente.", \FMT\Mensajeria::TIPO_AVISO, $this->clase);

                $select_tab = 'tab_embargo';
                $_SESSION['data_legajo']['select_tab'] = $select_tab;
                $redirect = Vista::get_url("index.php/legajos/gestionar/{$empleado->cuit}");
                $this->redirect($redirect);
            } else {
                $serr = $anticorruptcion->errores;
                foreach ($serr as $stext) {
                    $this->mensajeria->agregar($stext, \FMT\Mensajeria::TIPO_ERROR, $this->clase);
                }
            }
        }
        $vista = $this->vista;
        $nombre_de_puesto = \App\Modelo\Perfil::listarNombrePuestos();
        $puesto = \FMT\Helper\Arr::path($nombre_de_puesto, "{$empleado->perfil_puesto->nombre_puesto}.nombre", '-');
        $vars = ['AGENTE' => $empleado->persona->apellido, 'EMBA' => $empleado->persona->nombre, 'CUIT' => $empleado->cuit];
        if ($puesto != '') {
            $vars['DENOMINACION'] = $puesto;
        }
        $vista->add_to_var('vars', $vars);
        (new Vista($this->vista_default, compact('vista', 'empleado', 'embargo', 'tipo_embargo'))->pre_render();
    } else {
        $this->mensajeria->agregar("PARA DEFINIR EL <strong>EMBARGO</strong>,
        ES REQUISITO TENER LOS DATOS BÁSICOS DEL <strong>AGENTE</strong>.", \FMT\Mensajeria::TIPO_ERROR, $this->clase);
    }
}
```

Figura 6.4.4.2. Acción "modificar\_embargo()" en el controlador "Legajos" (imagen propia)

En la acción "modificar\_embargo()" del controlador, se observa la manera en que se reciben los datos que vienen del formulario de manera similar a la acción "alta\_embargo()".

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:
-------------------	---------------------------	----------------------------	-----------------------------

#### 6.4.5. Baja de embargos

La acción baja de embargos no se muestra en un formulario, puesto que la baja se realiza por medio del ícono “eliminar” de la siguiente figura:



Figura 6.4.5.1. Icono para la baja de embargos (imagen propia)

Al presionar el ícono “eliminar”, se muestra un aviso para confirmar antes de eliminar el embargo seleccionado; la lógica para la baja se muestra a continuación:

```

protected function accion_baja_embargo() {
    $embargo = Modelo\Embargo::obtener_embargo($this->request->query('id'));
    $empleado = Empleado::obtener($embargo->id_empleado, true);
    if (!empty($empleado->id) && !empty($empleado->cuit)){
        if($embargo->id) {
            $select_tab = 'tab_embargo';
            $_SESSION['data_legajo']['select_tab'] = $select_tab;
            if ($this->request->post('confirmar')) {
                $res = $embargo->baja();
                if ($res) {
                    $this->mensajeria->agregar('AVISO: El embargo
                    se eliminó de forma exitosa.', \FMT\Mensajeria::TIPO_AVISO, $this->clase);
                } else {
                    $this->mensajeria->agregar('AVISO: No es posible
                    eliminar el embargo con agentes asignados.', \FMT\Mensajeria::TIPO_ERROR, $this->clase);
                }
                $redirect = Vista::get_url("index.php/legajos/gestionar/{$empleado->cuit}");
                $this->redirect($redirect);
            }
        }
        $vista = $this->vista;
        $vars = ['AGENTE' => $empleado->persona->apellido, '$empleado->persona->nombre, 'CUIT' => $empleado->cuit];
        $vista->add_to_var('vars', $vars);
        (new Vista($this->vista_default, compact('vista', 'empleado', 'embargo'))->pre_render();
    }
}
  
```

Figura 6.4.5.2. Acción “baja\_embargo()” en el controlador “Legajos” (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Como se ve en la figura, se recibe el “id” del embargo seleccionado y, a partir de esto, se obtiene el “id” del empleado. También se muestran los mensajes correspondientes en el caso de éxito o falla en la eliminación.

#### 6.4.6. Historial de embargos

A continuación, se muestra la acción de “historial”. La vista que muestra esta acción tiene un comportamiento similar al listado de embargos.

```

protected function accion_historial_embargo() {
    $empleado = Empleado::obtener($this->request->query('id'));
    $historial = Embargo::listar_historial($empleado->id);
    $vista = $this->vista;
    $nombre_de_puesto = \App\Modelo\Perfil::listarNombrePuestos();
    $puesto = \FMT\Helper\Arr::path($nombre_de_puesto, "{$empleado->perfil_puesto->nombre_puesto}.nombre", '-');

    (new Vista($this->vista_default, compact('vista', 'historial', 'empleado', 'puesto'))->pre_render());
}
  
```

**Figura 6.4.6.1.** Acción “historial\_embargo()” en el controlador “Legajos” (imagen propia)

En la figura, la acción “historial\_embargo()” se obtiene a partir del CUIT del empleado, los datos de su “id”, y, a su vez, el “id” se pasa como parámetro para buscar los embargos que tiene el empleado y que se van a mostrar en la vista de lista de historial, de forma similar al procedimiento que muestra la lista de embargos.

#### 6.5. Implementación del Bloque de administración

En este apartado, se presenta el desarrollo del Bloque de administración. Este contiene información variada sobre el agente en un único formulario. Para ello, se divide el formulario en secciones.

A continuación, se explicará la implementación de la sección de carga horaria del empleado, ubicaciones del empleado y licencias especiales del empleado.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



### 6.5.1. Carga horaria del empleado

Para comenzar con el desarrollo de la sección “Carga Horaria”, se muestra el flujo de datos en lo que sigue:

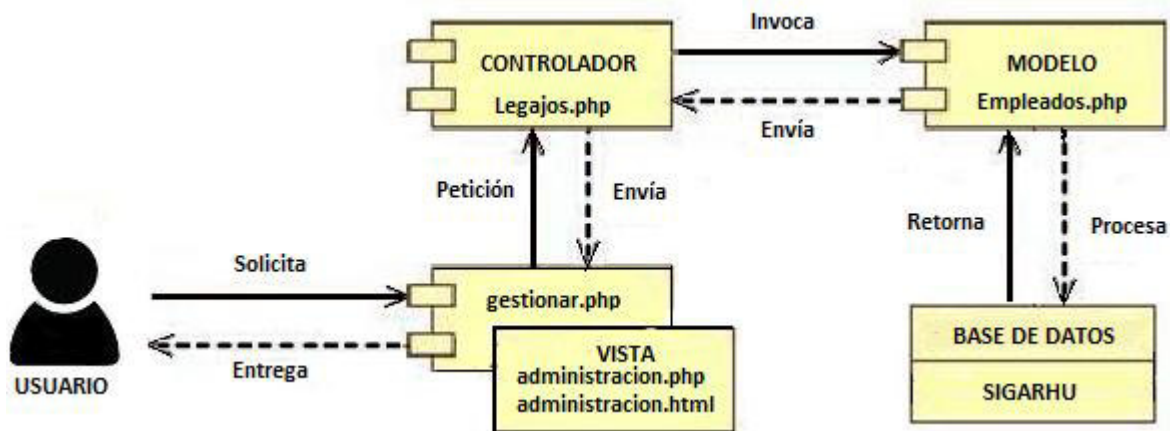


Figura 6.5.1.1. MVC para el Bloque de Administración para la sección de CargaHoraria (imagen propia)

En el controlador “Legajo.php”, en la acción “Gestionar”, se invoca el método “am\_administracion” en el momento del POST o envío de los datos del formulario para el guardado de alta o modificación, similar a la pestaña de embargos.

Se pretende mostrar una grilla horaria, con los días de la semana y los horarios de la jornada, o sea, el horario de entrada y de salida por cada día de trabajo. La grilla se cargará mediante la selección de un campo de datos llamada “Plantilla Horaria”, el cual tendrá listadas distintas plantillas precargadas mediante un ABM de plantillas horarias. En la siguiente imagen, se muestra la sección para la carga horaria:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



**Figura 6.5.1.2.** Vista de la sección Carga Horaria en la Solapa de Administración (imagen propia)

Además, del campo de selección de la plantillas horarias y de la grilla de horarios, se solicita un campo para el turno, otro campo informativo para la carga del total de horas semanales, el cual se calcula dependiendo de la selección de horarios en la grilla a través de una sumatoria de los horarios, y el campo “firma presentismo”.

#### 6.5.1.1. Estructura de datos para la carga horaria del empleado

Para la carga del horario del empleado, fue necesario modelar la tabla en la base de datos, donde se asignan los datos de horario al empleado. En la figura 6.5.1.1.1 se muestra la nueva tabla “empleado\_horarios” y su relación con la tabla “empleados”.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

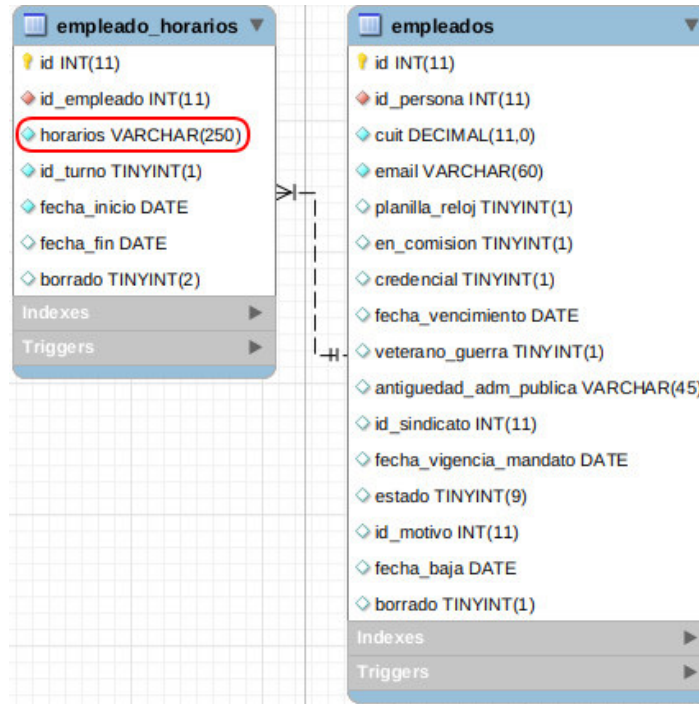


Figura 6.5.1.1.1. Modelado de la tabla “empleado\_horarios”, fragmento del DER de SIGARHU (imagen propia)

El campo “horarios”, resaltado en color rojo en la figura 6.5.1.1.1 de la tabla “empleados”, almacena los horarios cargados en la grilla en un formato de tipo *JSON*, como se muestra en el siguiente ejemplo:

```
(([], [{"09:00", "18:00"}, {"09:00", "18:00"}, {"09:00", "18:00"}, {"09:00", "18:00"}, {"09:00", "18:00"}], []))
```

El campo “id\_turno” de la tabla “empleados” podrá ser id “1”, para el turno mañana, o un id “2”, para el turno tarde. El campo de “fecha\_inicio” es la fecha actual en el alta de horarios o la de la modificación.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

### 6.5.1.2. Alta de las cargas horarias del empleado

A continuación, en la figura 6.5.1.2.1, se muestra un fragmento del método “am\_administracion()” para el manejo de alta y modificación de horarios del empleado:

```

protected function am_administracion($empleado){

    $control_horario      = clone $empleado->horario;
    $empleado->horario->horarios      = json_encode($this->request->post('horarios'));
    $empleado->horario->id_turno      = $this->request->post('id_turno');
    $empleado->horario->fecha_inicio  = $empleado->horario->fecha_inicio;
    $empleado->horario->fecha_fin     = $empleado->horario->fecha_fin;
    $post_p_reloj        = $this->request->post('planilla_reloj');

    if($empleado->cuit){
        $grilla = $this->request->post('horarios');
        $horas = false;
        foreach ($grilla as $value) {
            if(!empty($value[0]) && !empty($value[1])){
                $horas = true;
            }
        }
    }
    if($horas && ($control_horario != $empleado->horario)){

        if($empleado->validar()){
            if($empleado->horario->id){
                $empleado->modificacion_horario();
            }else{
                $empleado->horario->fecha_inicio = \DateTime::createFromFormat('d/m/Y H:i:s',
                    gmdate('d/m/Y').'0:00:00');
                $empleado->alta_horario();
            }
        }else{
            $err = $empleado->errores;
            foreach ($err as $text) {
                $this->mensajeria->agregar($text, \FMT\Mensajeria::TIPO_ERROR, $this->clase);
            }
            return false;
        }
        $camb_horario = true;
    }else {
        $camb_horario = false;
    }
}
  
```

Figura 6.5.1.2.1. Controlador Legajo, método am\_administracion()(imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

En la figura, la variable \$control\_horario guarda una copia de la instancia actual del empleado y se reasignan los datos en \$empleado->horario desde la vista, con datos que ingresa el usuario.

Los datos para “horarios”, que llegan de la vista en un formato de *array*, se muestran en la siguiente imagen:

```

array (size=7)
  0 =>
    array (size=2)
      0 => string '' (length=0)
      1 => string '' (length=0)
  1 =>
    array (size=2)
      0 => string '10:00' (length=5)
      1 => string '18:00' (length=5)
  2 =>
    array (size=2)
      0 => string '09:00' (length=5)
      1 => string '18:00' (length=5)
  3 =>
    array (size=2)
      0 => string '09:00' (length=5)
      1 => string '18:00' (length=5)
  4 =>
    array (size=2)
      0 => string '09:00' (length=5)
      1 => string '18:00' (length=5)
  5 =>
    array (size=2)
      0 => string '09:00' (length=5)
      1 => string '18:00' (length=5)
  6 =>
    array (size=2)
      0 => string '' (length=0)
      1 => string '' (length=0)
  
```

**Figura 6.5.1.2.2.** Arreglo de los valores de los horarios (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Luego de verificar que el empleado tenga los datos básicos, al evaluar si su CUIT está en el sistema, se verifica que la grilla tenga datos en todas las posiciones del *array*. A partir de esto, puede saberse si hubo o no modificaciones en la grilla.

Posteriormente, se revisa que `$control_horario` o la copia de instancia original del empleado sea distinta de la nueva instancia, con los valores reasignados en `$empleado->horario`.

Después de revisar los datos, se valida que sean correctos y se evalúa si el empleado aún no tiene horarios cargados, a fin de dar alta los datos para los horarios ingresados por medio de los métodos del modelo "alta\_horario()".

#### **6.5.1.3. Modificación de las cargas horarias del empleado**

Como se mencionó anteriormente, en la figura 6.5.1.2.1, se muestra un fragmento del método "am\_administracion()" para el manejo de alta y modificación de horarios del empleado. En este caso, luego de validar que los datos ingresados sean correctos y chequear que estos sean distintos a los ya ingresados, se modifican a través del método "modificacion\_horario()" del modelo.

#### **6.5.2. Ubicación del empleado**

A continuación, se describe la implementación de la sección de ubicación dentro de la pestaña de administración.

##### **6.5.2.1. Estructura de datos para la ubicación del empleado**

Para comenzar la implementación de esta sección, fue necesario modelar los datos que se van a guardar, acerca de la ubicación del empleado, para lo que se crea la tabla "empleados\_x\_ubicacion". Se hace uso, además, de las tablas "ubicacion\_edificios" y "ubicaciones". Ambas tablas contienen datos previamente

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

cargados por el administrador del sistema, a partir un “ABM<sup>22</sup> de ubicación” y el “ABM de ubicaciones Edificios”.

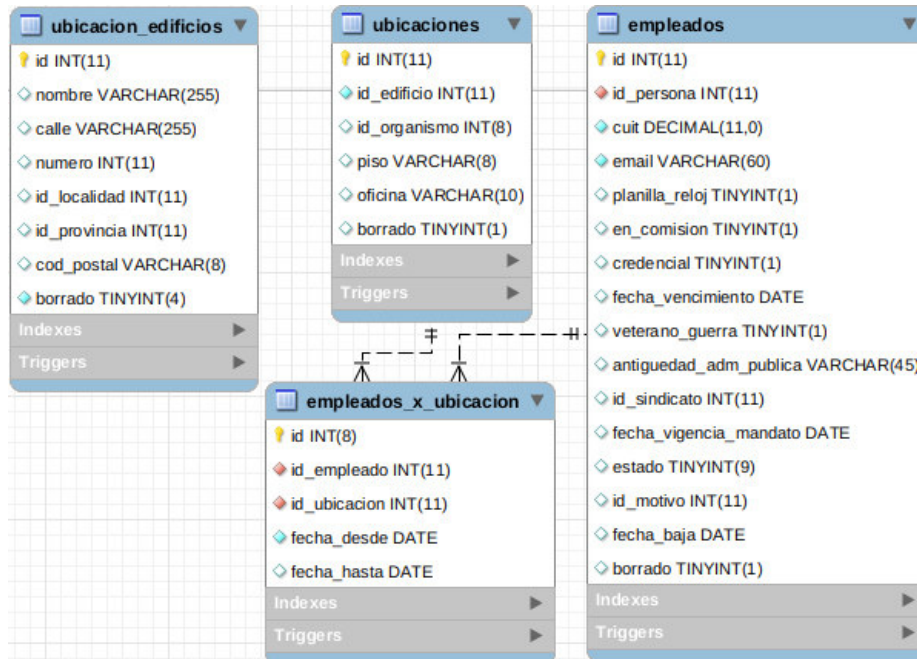


Figura 6.5.2.1.1. Modelado de la tabla “empleado\_x\_ubicacion”, fragmento del DER de SIGARHU (imagen propia)

Se requiere que, cuando el empleado no tenga datos sobre una ubicación, pueda seleccionar un edificio, y, a partir de esta selección, se completen automáticamente los demás campos de la ubicación.

Los campos de datos para la ubicación del empleado se describen a continuación:

<sup>22</sup>La abreviatura ABM significa “Altas, Bajas y Modificaciones”, y se refiere al sistema mediante el cual las aplicaciones de bases de datos se mantienen actualizadas (AA.VV, 2019a: s/p).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Edificio: campo con un elemento <select> de HTML. Representa un control que muestra un menú de opciones. Estas serán una lista de edificios.

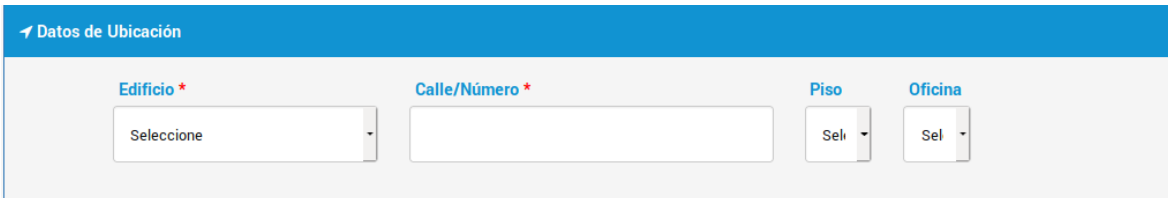
Calle/número: campo con un elemento <span> de HTML.

Piso: campo con un elemento <select> de HTML. Las opciones serán una lista de pisos que dependen del “id\_edificio” seleccionado.

Oficina: campo con un elemento <select> de HTML. Las opciones serán una lista de oficinas que dependen del “id\_edificio” y del piso seleccionado.

id ubicacion: campo de un elemento HTML <input> oculto (type = “hidden”).

Los campos descritos se exponen en la figura siguiente.

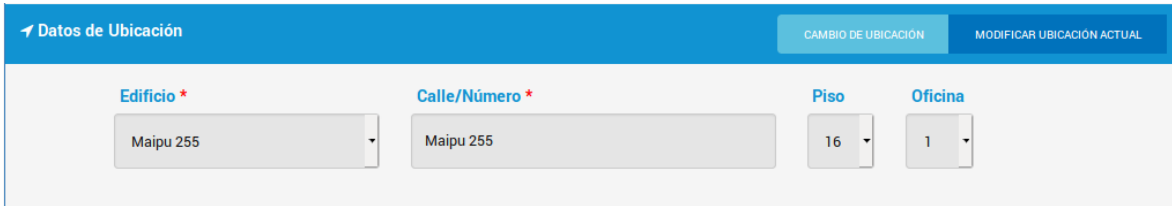


**Figura 6.5.2.1.2.** Campos de la sección “Datos de Ubicación” en la Pestaña “Administración” (imagen propia)

Cuando el empleado tenga datos sobre una ubicación, estos se muestran inhabilitados a la edición, pero se enseñan los datos sobre la ubicación precargada con los campos que aparecen en gris; en la siguiente figura, se muestra un ejemplo de esto.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:





**Figura 6.5.2.1.3.** Campos de la sección “Datos de Ubicación” en la Pestaña de “Administración” con los botones de “Cambio de Ubicación” y “Modificar ubicación actual” (imagen propia)

Como se ve en la figura 6.5.2.1.3, se requiere, además, que aparezcan unos botones en la parte superior derecha de la sección, que ofrecerán las siguientes opciones:

Cambio de Ubicación: limpia la selección, dejando los campos vacíos y permite la selección de otro edificio.

Modificación de la ubicación actual: habilita los campos, mostrando la selección actual, con la posibilidad de modificar los campos existentes.

En el archivo de la vista “gestionar.php”, se declara la variable `$empleado_ubicacion`, la cual devuelve como resultado “true”, si el empleado tiene una ubicación, y “false”, en caso contrario. Posteriormente, se utiliza el recurso de inyectar *JavaScript* para obtener datos del servidor en el cliente, como se muestra en la siguiente imagen, con la variable `$empleado_ubicacion`:

```

$empleado_ubicacion = (!empty($empleado->ubicacion->id)?'true':'false');
$vars_vista['JS'][]['JS_CODE'] = <<<JS
var \ $empleado_ubicacion = {$empleado_ubicacion};
JS;
  
```

**Figura 6.5.2.1.4.** Variable `$empleado_ubicacion` en inyección JavaScript (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

La variable `$empleado_ubicacion`, que se inyecta en el archivo *PHP*, va a poder ser usada en el archivo *JavaScript*, ya que el valor es necesario para generar la lógica que muestra los botones mencionados anteriormente. El archivo *JavaScript* en cuestión se muestra a continuación.

```
(function() {
$('#div_adm_ubicacion').ready(function(){
  var $rollback_ubi = $('#id_ubicacion').val();
  var campos_ubicacion = {
    'id_edificio'      : $('#id_edificio').val(),
    'ubicacion_piso'  : $('#ubicacion_piso').val(),
    'ubicacion_oficina' : $('#ubicacion_oficina').val(),
  };
  var span_ubicacion = {
    'ubicacion_calle_numero' : $('#ubicacion_calle_numero').text(),
  };

  $('#div_adm_ubicacion *').attr('disabled', true);
  if(!$empleado_ubicacion){
    $('#div_adm_ubicacion *').attr('disabled', false);
    $('#ubicacion_accion_alta').hide();
    $('#ubicacion_accion_modificar').hide();
    $('#adm_ubicacion_accion').val('alta');
  }

  $('#ubicacion_accion_alta').on('click', function(){
    $('#adm_ubicacion_accion').val('alta');
    $('#div_adm_ubicacion *').find("span").empty();
    $.each(campos_ubicacion, function(campo, valor){
      $('#'+campo).val('');
    });
    $('#div_adm_ubicacion *').attr('disabled', false);
    $('#id_ubicacion').val('');
    $('#div_adm_ubicacion #ubicacion_calle_numero').attr('disabled', true);
  });

  $('#ubicacion_accion_modificar').on('click', function(){
    $('#adm_ubicacion_accion').val('modificacion');
    $('#div_adm_ubicacion *').attr('disabled', false);
    $('#id_ubicacion').val($rollback_ubi);
    $.each(campos_ubicacion, function(campo, valor){
      $('#'+campo).val(valor);
    });
    $.each(span_ubicacion, function(campo, valor){
      $('#'+campo).text(valor);
    });
    $('#div_adm_ubicacion #ubicacion_calle_numero').attr('disabled', true);
  });
});
```

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

**Figura 6.5.2.1.5.** Archivo “administracion.js” (imagen propia)

En la vista, los botones tienen asignados identificadores: “ubicacion\_accion\_alta”, para el botón de “cambio de ubicación”, y “ubicacion\_accion\_modificacion”, para el botón “modificación ubicación actual”. Los identificadores de los elementos son necesarios para generar los comportamientos de los campos solicitados. En la figura 6.4.10, puede verse la variable *JavaScript* \$rollback\_ubi, donde se captura el valor del elemento cuyo identificador es “id\_ubicacion”. También se presenta la variable “campos\_ubicacion”, que obtiene los valores de los campos de datos y la variable “span\_ubicacion”, que muestra el valor del campo deshabilitado.

Más abajo, en la imagen, se analiza el elemento HTML <div> a través de su identificador “div\_adm\_ubicaciones”. Este elemento contiene los elementos que corresponden a los campos de edificio, calle/numero, piso, oficina. El método “attr” agrega un atributo que deshabilita los elementos de “div\_adm\_ubicaciones”.

Después, se comprueba si la variable “\$empleado\_ubicacion”, que se pasa desde el archivo *PHP*, es “false”, lo que quiere decir que está vacía, por lo que va a mantener los campos deshabilitados y ocultos los botones de alta y modificación.

El método “on” controla el evento “Click”, y detecta si es un alta o modificación, al presionar los botones respectivos. En el caso de alta, activa los campos y los limpia, para cargar nuevos datos, a partir de una nueva selección. Se asigna el valor “alta” al elemento “adm\_ubicacion\_accion” y un valor vacío al elemento con el id “id\_ubicacion”. En el caso de modificación, se desactivan los campos de datos de la ubicación del empleado; datos contenidos en el elemento cuyo identificador es “div\_adm\_ubicaciones”,

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

se muestran todos los valores actuales de los campos, a fin de ofrecer la posibilidad de modificar uno o varios. Así, se asigna el valor “modificación” al elemento “adm\_ubicacion\_accion”, y, también, se asigna el valor de “\$rollback\_ubi” al elemento con el identificador “id\_ubicacion”.

#### 6.5.2.2. Alta de ubicación del empleado

Al momento de guardar los cambios de la ubicación del empleado, se invoca el método “am\_administracion” en el controlador “Legajos”, junto con la acción “Gestionar”. A continuación, se muestra el método “am\_administracion”, representado en la figura 6.5.2.2.1.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```
protected function am_administracion($empleado){
    $control_ubi = $empleado->ubicacion->id_ubicacion;
    $empleado->ubicacion->id_ubicacion = $this->request->post('id_ubicacion');
    if($empleado->cuit){
        if($this->request->post('adm_ubicacion_accion') && ($control_ubi != $empleado->ubicacion->id_ubicacion)){
            if($this->request->post('adm_ubicacion_accion') == 'alta'){
                if($empleado->validar()){
                    if($empleado->ubicacion->id){
                        $aux = $empleado->ubicacion->id_ubicacion;
                        $aux2 = \App\Modelo\Ubicacion::obtener($aux);
                        $empleado->ubicacion->id_ubicacion = $control_ubi;
                        $empleado->ubicacion->fecha_hasta = \DateTime::createFromFormat('d/m/Y H:i:s', gmdate('d/m/Y').'.0:00:00');
                        $empleado->modificacion_ubicacion();
                        $empleado->ubicacion->id_ubicacion = $aux;
                        $empleado->ubicacion->id_edificio = $aux2->id_edificio;
                        $empleado->ubicacion->nombre = $aux2->nombre;
                        $empleado->ubicacion->calle = $aux2->calle;
                        $empleado->ubicacion->numero = $aux2->numero;
                        $empleado->ubicacion->piso = $aux2->piso;
                        $empleado->ubicacion->oficina = $aux2->oficina;
                    }
                    $empleado->ubicacion->fecha_desde = \DateTime::createFromFormat('d/m/Y H:i:s', gmdate('d/m/Y').'.0:00:00');
                    $empleado->ubicacion->fecha_hasta = null;
                    $empleado->alta_ubicacion();
                } else {
                    $serr = $empleado->errores;
                    foreach ($serr as $text) {
                        $this->mensajeria->agregar($text, \FMT\Mensajeria::TIPO_ERROR, $this->clase);
                    }
                    return false;
                }
            }
            if($this->request->post('adm_ubicacion_accion') == 'modificacion'){
                if($empleado->validar()){
                    $aux = \App\Modelo\Ubicacion::obtener($empleado->ubicacion->id_ubicacion);
                    $empleado->ubicacion->id_edificio = $aux->id_edificio;
                    $empleado->ubicacion->nombre = $aux->nombre;
                    $empleado->ubicacion->calle = $aux->calle;
                    $empleado->ubicacion->numero = $aux->numero;
                    $empleado->ubicacion->piso = $aux->piso;
                    $empleado->ubicacion->oficina = $aux->oficina;
                    $empleado->modificacion_ubicacion();
                } else {
                    $serr = $empleado->errores;
                    foreach ($serr as $text) {
                        $this->mensajeria->agregar($text, \FMT\Mensajeria::TIPO_ERROR, $this->clase);
                    }
                    return false;
                }
            }
        }
        $scamb_ubi = true;
    } else {
        $scamb_ubi = false;
    }
}
```

Figura 6.5.2.2.1. Controlador “Legajo”, método “am\_administracion” para alta y modificación de ubicación del empleado (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

La variable “\$control\_horario”, que se muestra en la figura 6.5.2.2.1, guarda una copia de la instancia actual del empleado y se reasignan los datos en “\$empleado->horario” con datos que ingresa el usuario desde el formulario.

Luego, se evalúa que el empleado tenga los datos básicos, mediante el análisis del CUIT, y se comprueba que la acción sea de alta y que la copia de la instancia de empleados->ubicacion->id\_ubicacion (\$control\_ubi) esté vacía para realizar el alta. Antes de guardar los cambios, se valida que los datos ingresados sean correctos y se dan de alta mediante el método “alta\_ubicacion” del modelo “Empleado”.

### **6.5.2.3. Modificación de la ubicación del empleado**

Para el caso de la modificación, y siguiendo con el ejemplo de la figura 6.5.2.2.1, se comprueba que la acción sea de modificación y que la copia de la instancia de empleados->ubicacion->id\_ubicacion (\$control\_ubi) sea distinta, para que resulte posible realizar cambios. En caso de que no lo sea, no hay cambios en la ubicación del empleado. Antes de guardar los cambios, se valida que los datos ingresados sean correctos y se utiliza el método “modificacion\_ubicacion” del modelo “Empleado”.

### **6.5.3. Licencias especiales del empleado**

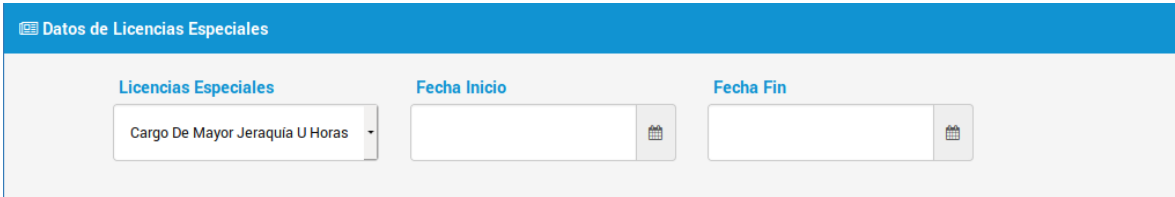
La implementación de la carga de datos de licencias especiales es similar a la implementación de los datos de ubicación. A continuación, se describen los campos necesarios para esta sección.

Licencias especiales: el campo será un seleccionable con opciones de licencias especiales.

Fecha de inicio y fecha fin: estos campos serán para la fecha de inicio de la licencia y la fecha de fin de la licencia; ambos serán elementos HTML <INPUT> de tipo fecha.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

En la imagen siguiente, se representa la sección de licencias especiales de la solapa de administración.



**Figura 6.5.3.1.** Campos de Datos de la sección “Datos de Licencias Especiales” en el bloque Administración (imagen propia)

### 6.5.3.1. Estructura de datos para las licencias especiales del empleado

A continuación, se muestran las figuras del modelado de datos necesarios para el guardado de información de licencias especiales y su relación con las tablas de “empleados” y “licencias\_especiales”.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

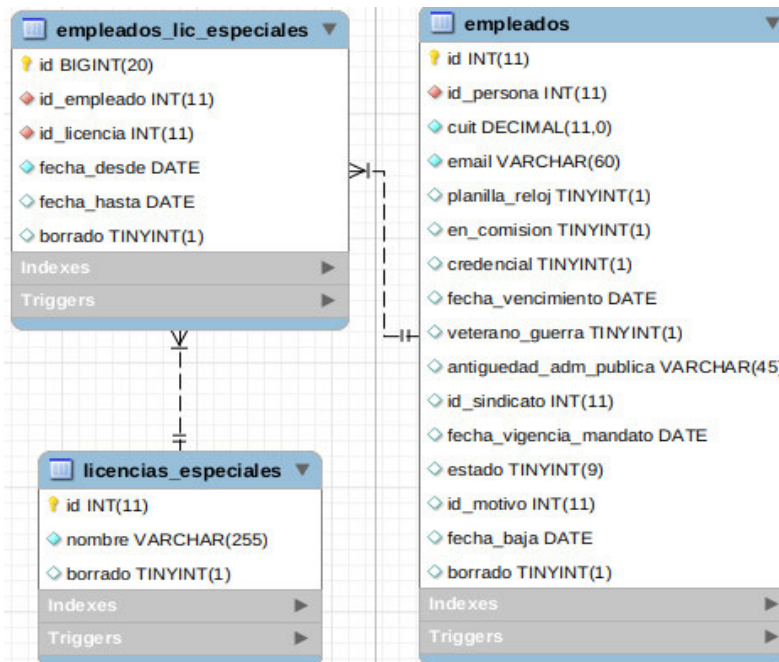


Figura 6.5.3.1.1. Modelado de la tabla “empleados\_lic\_especiales”, fragmento del DER de SIGARHU (imagen propia)

### 6.5.3.2. Alta de licencias especiales del empleado

Como las demás secciones de la solapa administración, presentada anteriormente, para el guardado de datos, vuelve a ser necesario invocar el método “am\_administracion” en el controlador “Legajos” y la acción “gestionar”. A continuación, se muestra el fragmento diseñado para la lógica de alta o modificación de la sección “licencias especiales” en el método “am\_administracion”.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



```
protected function am_administracion($empleado){
    $control_lic = clone $empleado->licencia;
    $empleado->licencia->id_licencia = $this->request->post('id_licencia');
    $empleado->licencia->fecha_desde = ($temp = \DateTime::createFromFormat('d/m/Y H:i:s', $this->request->post('fecha_desde')).'0:00:00') ? $temp : '';
    $empleado->licencia->fecha_hasta = ($temp = \DateTime::createFromFormat('d/m/Y H:i:s', $this->request->post('fecha_hasta')).'0:00:00') ? $temp : '';

    if($empleado->cuit){
        if($this->request->post('adm_licencia_accion') && $control_lic != $empleado->licencia) {
            if($this->request->post('adm_licencia_accion') == 'alta'){
                if($empleado->validar()){
                    if($control_lic->id){
                        $aux = clone $empleado->licencia;
                        $empleado->licencia = $control_lic;
                        $empleado->baja_licencias_especiales();
                        $empleado->licencia = $aux;
                    }
                    $empleado->alta_licencias_especiales();
                }else{
                    $serr = $empleado->errores;
                    foreach ($serr as $text) {
                        $this->mensajeria->agregar($text, \FMT\Mensajeria::TIPO_ERROR, $this->clase);
                    }
                    return false;
                }
            }
            if($this->request->post('adm_licencia_accion') == 'modificacion'){
                if($empleado->validar()){
                    $empleado->modificacion_licencias_especiales();
                }else{
                    $serr = $empleado->errores;
                    foreach ($serr as $text) {
                        $this->mensajeria->agregar($text, \FMT\Mensajeria::TIPO_ERROR, $this->clase);
                    }
                    return false;
                }
            }
            $camb_licencia = true;
        }else{
            $camb_licencia = false;
        }
    }
}
```

Figura 6.5.3.2.1. Controlador “Legajo”, metodo *am\_administracion* para el alta y modificación de Licencias Especiales (imagen propia)

La lógica para el guardado de datos en el código de la figura 6.5.3.2.1 es similar a la realizada para la sección de “Ubicación del empleado”. Se evalúa que el empleado tenga los datos básicos, mediante el análisis del CUIT, y se comprueba que la acción sea de alta y que la copia de la instancia de empleados->licencia (\$control\_lic) este vacía para realizar el alta. Antes de guardar los cambios, se validan los datos ingresados como correctos y se dan de alta mediante el método “alta\_licencias\_especiales” del modelo “Empleado”.

### 6.5.3.3. Modificación de licencias especiales del empleado

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Para el caso de la modificación, se comprueba que la copia de la instancia de empleados->licencia (\$control\_lic) sea distinto para realizar cambios; ya que, en caso contrario, no se realizan cambios en la licencia del empleado. Antes de guardar los cambios, se valida que los datos ingresados sean correctos y se utiliza el método "modificacion\_licencias\_especiales" del modelo "Empleado".

### 6.6 Implementación del Bloque de anticorrupción

El sistema llevará el control y gestión del personal que debe presentar su Declaración Jurada Patrimonial Integral. El formulario para la carga de presentaciones de las declaraciones juradas en el bloque de anticorrupción muestra los campos deshabilitados, en principio, en el caso de que no haya declaraciones juradas cargadas. En el caso de que sí existan presentaciones, se mostrará la presentación con fecha actual o la última cargada. Los campos se describen a continuación:

El formulario de anticorrupción muestra los campos de datos que se describen a continuación:

Obligado a presentar declaración jurada: el elemento HTML es `<input type="checkbox">`. El tipo CHECKBOX representa un control de dos estados que permite a los usuarios marcar uno de ellos ("seleccionado" o "deseleccionado"). Si se elige "seleccionado", se habilitan los campos de "fecha de designación" y "fecha de publicación de la designación" y el botón "guardar", previamente deshabilitados.

Fecha de designación: campo con un elemento `<input disabled="disabled"/>`. Cuando está presente el elemento "disabled", el elemento `<input>` debe estar deshabilitado. El campo "fecha de designación" no puede ser mayor a la fecha de publicación de la designación.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Fecha Publicación de la designación: campo con un elemento `<input disabled="disabled" />` no menor a la fecha de designación.

Fecha de aceptación de la renuncia: campo con un elemento `<input>` que solo se podrá completar si el legajo no está activo y se da de baja. Mientras el empleado esté activo, será un campo con un elemento `<input disabled="disabled" />`.

Fecha de última presentación: campo con un elemento `<input disabled="disabled" />`.

Tipo de Declaración Jurada: campo con un elemento `<input disabled="disabled" />`.

Período Presentado: campo con un elemento `<input disabled="disabled" />`.

Número de transacción: campo con un elemento `<input disabled="disabled" />`.

En lo que sigue, se describen los botones en el formulario de anticorrupción:

Nueva Presentación: en principio, este botón no es visible hasta que se haya guardado primero la fecha de designación y la fecha de publicación de la designación. Cuando esto ocurre, se redirige al usuario a un formulario de alta de una nueva presentación.

Ver comprobante: no es visible, hasta que haya una presentación actual cargada, con su respectivo comprobante. El botón muestra el comprobante de la Declaración Jurada únicamente si existe en la presentación actual.

Historial: no es visible hasta que existan presentaciones anteriores cargadas. El botón redirige a un listado de las presentaciones cargadas con anterioridad.

A continuación, se presenta el formulario principal para la carga de anticorrupción, con los campos y botones anteriormente descriptos.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



Figura 6.6.1. Formulario principal del bloque de Anticorrupción (imagen propia)

La figura 6.6.1 muestra los datos de la presentación de declaración jurada vigente del empleado.

Al presionar el botón “Nueva Presentación”, se mostrará el formulario con los siguientes campos:

Tipo de Declaración Jurada: campo con un elemento `<select>` de HTML. Representa un control que muestra un menú de opciones. Las opciones son inicial, anual y baja.

Fecha de presentación: elemento HTML `<input>` de tipo “text”, con una clase fecha. Para este campo, se utilizará la librería *moment.js* para restringir la selección solo a una fecha durante el presente año. Para ejemplificar lo anterior, se muestra a continuación el archivo *JavaScript*:

```

var y = moment().get('year');
$('.fecha').datetimepicker({
  maxDate: moment(y, "YYYY").add(1, 'year').subtract(1, 'days'),
  format: 'DD/MM/YYYY'
});

```

Figura 6.6.2. Inicialización de la librería “datetimepicker” para la clase “fecha” y uso de la librería momento (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Período presentado: también es un elemento HTML <INPUT> de tipo fecha. Aquí se muestra el ejemplo del archivo *JavaScript* definido con la inicialización de la librería.

```

$('#periodo').datetimepicker({
  viewMode: 'years',
  format: 'YYYY',
});

```

**Figura 6.6.3.** Inicialización de la librería “datetimepicker” para el id periodo (imagen propia)

En la figura anterior, se puede ver que se toma el “id” del campo, en este caso, “periodo”, y se inicializa la librería *datetimepicker* con una configuración que permite mostrar solo el año, por ejemplo, “2019”.

Número de transacción: campo con un elemento <INPUT> de tipo *text*.

Comprobante: campo representado por un elemento HTML <INPUT> de tipo “file”. Luego de haber agregado las correspondientes librerías *fileinput*, se procede a inicializar el proceso, mediante la clase del elemento <INPUT> que se muestra en la figura a continuación.

```

$(".filestyle").fileinput({
  language: 'es',
  browseLabel: '',
  showRemove: false,
  showUpload: false,
  previewFileIcon: '<i class="glyphicon glyphicon-eye"></i>',
  previewFileIconClass: 'file-icon-4x'
});

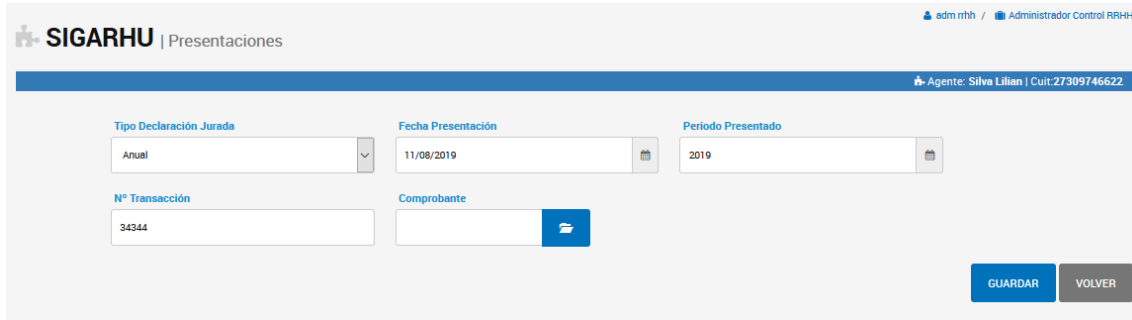
```

**Figura 6.6.4.** Inicialización de la librería “fileinput” para la clase “filestyle” (imagen propia)

En la figura 6.6.4., se muestra la sintaxis para la inicialización del *pluginfileinput* en *jQuery*, siendo *filestyle* la clase del campo de tipo “file” del formulario y, más abajo, las opciones de configuración *fileinput* separadas por coma (,).

Presentados los campos, se representan en la siguiente figura:

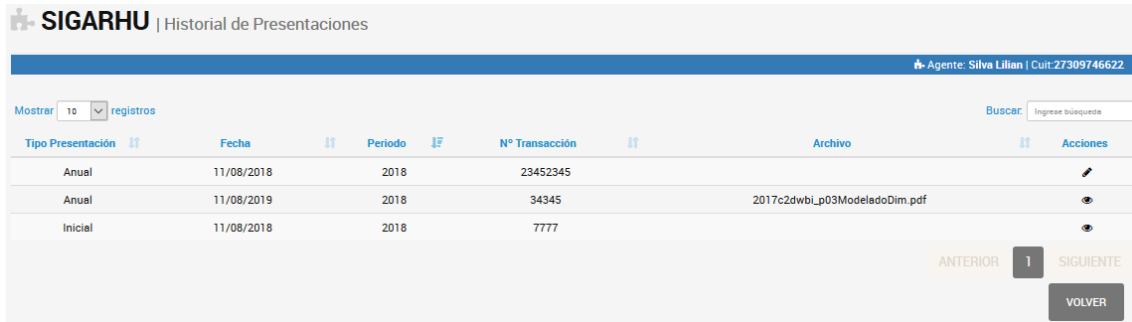
Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



**Figura 6.6.5.**Formulario de alta de Presentación (imagen propia)

La última presentación que se cargue desde este formulario actualizará el formulario de anticorrupción presentado en la figura 6.6.5.

También se podrán visualizar las presentaciones anteriores con el botón “Historial”, pudiendo modificar solo la última presentación agregada. A continuación, se muestra el ejemplo de un listado para el historial de presentaciones:



Tipo Presentación	Fecha	Período	N° Transacción	Archivo	Acciones
Anual	11/08/2018	2018	23452345		
Anual	11/08/2019	2018	34345	2017c2dwbLp03ModeladoDim.pdf	
Inicial	11/08/2018	2018	7777		

**Figura 6.6.6.**Listado Historial de Presentaciones (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

En la última columna de la figura, con título “acciones”, se puede ver que es posible modificar la última presentación realizada que se muestra en orden ascendente; las demás solo pueden verse, sin posibilidad de modificación.

#### **6.6.1. Estructura de datos para el Bloque de anticorrupción**

Para la implementación de las funcionalidades en el Bloque de anticorrupción, se inicia el modelado de los datos, para lo cual se crearon dos nuevas tablas en la base de datos: “anticorrupcion” y “anticorrupcion\_presentacion”, como se detalla en la imagen a continuación.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



Figura 6.6.1.1. Tablas de empleados, anticorrupcion y anticorrupcion\_presentacion(imagen propia)

Cuando se crea una nueva tabla anticorrupción, esta podrá tener varias presentaciones relacionadas, referidas a la misma tabla anticorrupción; cada registro en la tabla "anticorrupcion\_presentacion" tendrá un "id\_anticorrupcion" que hace referencia a la primera tabla, la de "anticorrupcion". Más adelante, se crea un nuevo modelo "anticorrupción" cuyas propiedades incluyen los campos de las dos tablas presentadas.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



### 6.6.2. Alta de anticorrupción

Al momento de guardar en el controlador "Legajos" y la acción "gestionar", se invoca una nueva función "am\_anticorrupcion" que se presenta a continuación.

```

protected function am_anticorrupcion(Modelo\Anticorrupcion $anticorrupcion=null) {
    $control = Clone $anticorrupcion;
    $empleado = Empleado::obtener($this->request->query('id'));
    $modificacion = false;
    $anticorrupcion->id_empleado = $empleado->id;
    $anticorrupcion->fecha_designacion = empty($temp = $this->request->post('fecha_designacion')) ?
    \DateTime::createFromFormat('d/m/Y H:i:s', $temp . ' 00:00:00') : null;
    $anticorrupcion->fecha_publicacion_designacion = empty($temp = $this->request->post('fecha_publicacion_designacion')) ?
    \DateTime::createFromFormat('d/m/Y H:i:s', $temp . ' 00:00:00') : null;
    $anticorrupcion->fecha_aceptacion_renuncia = empty($temp = $this->request->post('fecha_aceptacion_renuncia')) ?
    \DateTime::createFromFormat('d/m/Y H:i:s', $temp . ' 00:00:00') : null;
    if($anticorrupcion->id) {
        if($this->request->post('obligado_dj') == 1){
            if($control->fecha_designacion != $anticorrupcion->fecha_designacion
            || $control->fecha_publicacion_designacion != $anticorrupcion->fecha_publicacion_designacion
            || $control->fecha_aceptacion_renuncia != $anticorrupcion->fecha_aceptacion_renuncia
            || $control->tipo_presentacion != $anticorrupcion->tipo_presentacion
            || $control->fecha_presentacion != $anticorrupcion->tipo_presentacion
            || $control->nro_transaccion != $anticorrupcion->nro_transaccion)
            {
                if($anticorrupcion->validar()){
                    $aux = clone $anticorrupcion;
                    $anticorrupcion = $control;
                    $aux->modificacion();
                    $anticorrupcion = $aux;
                    $modificacion = true;
                    $this->mensajería->agregar(
                        "la Anticorrupcion para el empleado <strong>{$empleado->persona->nombre}</strong>
                        {$empleado->persona->apellido}</strong> fue ".($modificacion ? 'modificada':'cargada')." exitosamente.",
                        \FMT\Mensajería::TIPO_AVISO,
                        $this->clase
                    );
                }
            }
        }
        $err = $anticorrupcion->errores;
        foreach ($err as $text) {
            $this->mensajería->agregar($text, \FMT\Mensajería::TIPO_ERROR, $this->clase);
        }
        return false;
    }
}

```

Figura 6.6.2.1. función "am\_anticorrupcion" en el controlador "Legajos" (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```

    $anticorrupcion->baja();
    $this->mensaje->agregar(
        "la Anticorrupcion para el empleado <strong>{$empleado->persona->nombre}</strong>
        {$empleado->persona->apellido}</strong> fue dada de baja exitosamente.",
        \FMT\Mensajeria::TIPO_AVISO,
        $this->clase
    );
  }
}
else{
  if($anticorrupcion->validar()){
    if ($anticorrupcion->alta()){
      $this->mensaje->agregar(
          "la Anticorrupcion para el empleado <strong>{$empleado->persona->nombre}</strong>
          {$empleado->persona->apellido}</strong> fue ".($modificacion ? 'modificada':'cargada')." exitosamente.",
          \FMT\Mensajeria::TIPO_AVISO,
          $this->clase
      );
    }
  }
  if($anticorrupcion->errores){
    foreach ($anticorrupcion->errores as $value) {
      $this->mensaje->agregar($value,\FMT\Mensajeria::TIPO_ERROR,$this->clase);
    }
  }
}
}
}

```

Figura 6.6.2.2. Continuación función "am\_anticorrupcion" en el controlador "Legajos" (imagen propia)

Como se puede ver en las figuras 6.6.2.1 y 6.6.2.2<sup>23</sup>, la variable "\$control" guarda una copia de la instancia actual de la anticorrupción del empleado; además, permite obtener datos sobre el empleado analizado y los valores que vienen de los campos cargados en el formulario de anticorrupción, que se guardan en la variable "\$anticorrupcion".

Se comparan los valores guardados en la variable \$control contra los de la variable \$anticorrupcion, de modo que sea posible guardar los nuevos datos, luego de validar la correcta inserción de la información desde el formulario.

### 6.6.3 Modificación de anticorrupción

<sup>23</sup> Las figuras se presentan separadas en el presente informe, a fin de posibilitar su mejor comprensión al lector.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Para la modificación, se comparan los valores guardados en la variable \$control y los de la variable \$anticorrupcion, a fin de establecer las diferencias entre los valores previamente cargados a través del alta y los nuevos valores, para luego de validar la correcta inserción de la información desde el formulario y, finalmente, modificar dichos datos.

#### 6.6.4 Presentaciones de la declaración Jurada

Un procedimiento similar se utiliza para el guardado de datos de las presentaciones; dicho procedimiento se muestra a continuación:

```

protected function accion_presentacion() {
    $empleado = Empleado::obtener($this->request->query('id'));
    $anticorrupcion = Anticorrupcion::obtener($empleado->id);
    $tipo_presentacion = \App\Modelo\Anticorrupcion::getParam('TIPO_D3');
    $anticorrupcion->fecha_presentacion = ($temp = $this->request->post('fecha_presentacion')) ?
        \DateTime::createFromFormat('d/m/Y', $temp) : null;
    $anticorrupcion->periodo = ($temp = $this->request->post('periodo')) ? $temp : null;
    $anticorrupcion->nro_transaccion = ($temp = $this->request->post('nro_transaccion')) ? $temp : null;
    $anticorrupcion->archivo = null;
    if($this->request->post('boton_presentacion') == 'alta') {
        $anticorrupcion->id_empleado = $empleado->id;
        $anticorrupcion->tipo_presentacion = empty($this->request->post('tipo_presentacion')) ? (int)$this->request->post('tipo_presentacion') : $
            anticorrupcion->tipo_presentacion;
        $anticorrupcion->archivo = ($FILES['anticorrupcion_file']['error'] == UPLOAD_ERR_OK) ? $FILES['anticorrupcion_file'] : null;
        if($anticorrupcion->id) {
            if($anticorrupcion->validar()){
                $anticorrupcion->alta_presentacion();
                $this->mensajeria->agregar(
                    "la Presentación del empleado <strong>{$empleado->persona->nombre} {$empleado->persona->apellido}</strong> fue cargada exitosamente.",
                    \FMT\Mensajeria::TIPO_AVISO,
                    $this->clase
                );
                $select_tab = 'tab_anticorrupcion';
                $_SESSION['data_legajo']['select_tab'] = $select_tab;
                $redirect = Vista::get_url("index.php/legajos/gestionar/{$empleado->cuit}");
                $this->redirect($redirect);
            }
        }
        $err = $anticorrupcion->errores;
        foreach ($err as $text) {
            $this->mensajeria->agregar($text, \FMT\Mensajeria::TIPO_ERROR, $this->clase);
        }
    }
}

$vista = $this->vista;
$vars = ['AGENTE' => $empleado->persona->apellido, 'empleador' => $empleado->persona->nombre, 'CUIT' => $empleado->cuit];
$vista->add_to_var('vars', $vars);
(new Vista($this->vista_default, compact('vista', 'empleado', 'tipo_presentacion', 'anticorrupcion'))->pre_render());

```

Firma estudiante:

Firma docente supervisor:

Firma docente tutor TAPTA:

Firma tutor organizacional:

Figura 6.6.4.1. Acción “presentación” en el controlador “Legajos” (imagen propia)

En este caso, se analizan los campos del formulario de presentaciones de declaraciones juradas que se presentaron previamente en la figura 6.6.5.

### 6.7 Implementación de la gestión de observaciones

Para la gestión de observaciones, se prepara la tabla en donde se guardarán las observaciones. El nombre de la tabla será “Observaciones” y los campos son los siguientes:

1. id: identificador auto incremental.
2. id\_employado: hace referencia al id del empleado a quien se le agregan las observaciones.
3. id\_usuario: id del usuario del sistema que realiza las observaciones del agente.
4. id\_bloque: id del bloque o pestaña a la cual se le agrega la observación del agente.
5. fecha: fecha de cuando se realiza la observación.
6. descripción: información sobre la observación en sí.
7. borrado: campo para la eliminación lógica del registro; se considerará borrado cuando el valor del registro sea un 1 o “true”; en caso contrario, será un 0 o “false”.

A continuación, se muestra la figura 6.7.1 que contiene la tabla “Observaciones” y su relación con la tabla principal “Empleados”:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

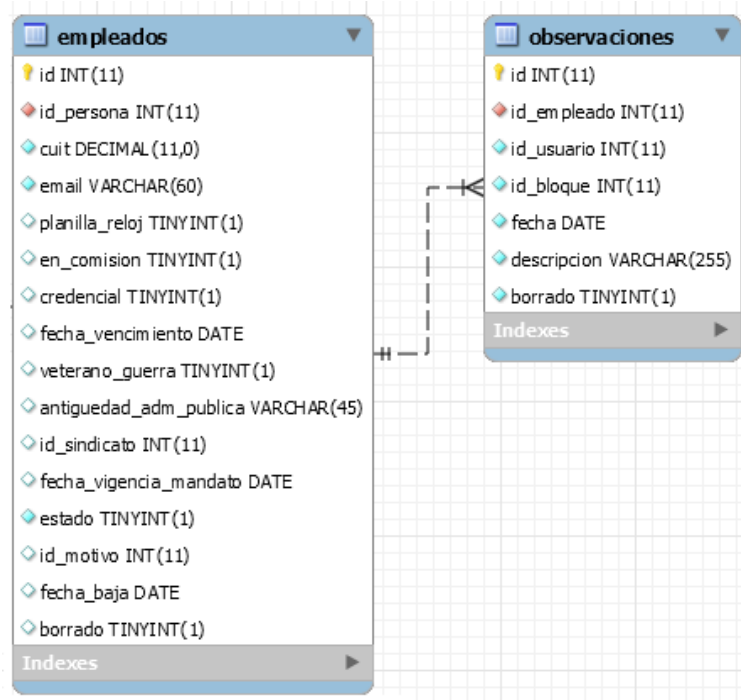


Figura 6.7.1. Modelado de la tabla de Observaciones, fragmento del diagrama de entidad - relación (DER) SIGARHU (imagen propia)

Con el modelado de la tabla de observaciones preparado, y, luego del análisis junto con el líder técnico, se concluye que es necesario agregar observaciones a través de un elemento HTML <textarea> para el ingreso de texto con un elemento HTML <span> con el signo más (+) en cada bloque de la gestión de legajos. Además, esto debe reflejarse automáticamente en un listado en la parte inferior del *textarea*. Para obtener este comportamiento, se decide utilizar AJAX.

El elemento <textarea> tendrá un id que lo identifique, titulado "legajo\_observacion\_ajax" y la acción que dispara el alta será mediante el elemento <span> que se identifica por su clase "span-observacion".

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

El listado para las observaciones se muestra en la figura a continuación:



Fecha	Observacion	Bloque	Usuario	Acciones
10/09/2019	nueva observacion de ejemplo	Datos Personales	carga de datos	<a href="#">✎</a>
10/05/2019	nueva direccion	Ubicación en la Estructura	Auditor Gonzalez	
02/02/2015	Actualizar datos	Situacion Escalafonaria	admin Cortez	
02/02/2013	corroborrar infromacion	Datos Personales	admin Cortez	

Mostrando registros del 1 al 4 de un total de 4 registros

**Figura 6.7.2.** Vista de la Gestión de Observaciones (imagen propia)

En la figura 6.7.2 se puede ver, en la parte superior del listado, el campo para el ingreso de observaciones sobre el legajo del empleado.

Se utilizará un nuevo archivo observaciones-ajax.js que contiene a la función `click()`, que se ejecuta cada vez que se hace “click” en el elemento `<span>` del DOM.

El primer control detecta el “click” sobre el `<span>` a través de su clase “span-observación” y este se encarga de lanzar la petición *AJAX*.

El código *Javascript* que contiene la lógica mencionada es el siguiente:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```
$(".span-observacion").click(function(){
    var $bloque;
    var $id;
    var $cuit      = $('#div_agente .ct').eq(0).text();
    var $observacion = $("#legajo_observacion_ajax");
    $("#alert_observacion").hide();
    if ($observacion.data('id') == '') {
        $bloque = $('li.active a').data('id');
        $id = '0';
    }else{
        $bloque = $observacion.data('bloque');
        $id = $observacion.data('id');
    }
}

lsilva (7 months ago) · avance Gestion de Observaciones

$.ajax({
    url: $url_base + '/index.php/legajos/observaciones',
    data: {
        id_ob: $id,
        cuit: $cuit,
        descripcion: $observacion.val(),
        bloque: $bloque
    },
    method: "POST"
})
.done(function (data) {
    if (!data.error) {
        $observacion = $("#legajo_observacion_ajax");
        $observacion.val('');
        $observacion.data('id','');
        $observacion.data('bloque','');
        $("#alert_observacion").show();
        $("#alert_observacion").removeClass().addClass("alert alert-success");
        $("#alert_observacion i").removeClass().addClass("fa fa-check");
        $("#alert_observacion span").html(" La nueva Observación se cargo con éxito.");

        update();
    }else{
        $("#alert_observacion").show();
        $("#alert_observacion").removeClass().addClass("alert alert-danger")
        $("#alert_observacion i").removeClass().addClass("fa fa-times-circle");
        $("#alert_observacion span").html(" No es posible Cargar una Observación vacía");
    }
});
});
```

Figura 6.7.3. Función clicken el archivo observaciones-ajax.js. (imagen propia)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

En la imagen, se pueden ver las propiedades bloque, id, CUIT y observación. Más abajo, en el método *AJAX*, se pasan los siguientes parámetros: en primer lugar, el parámetro URL, con la dirección a la que se envía la petición; luego, el parámetro *"data"*, con los datos a enviar al servidor; y, por último, el parámetro *"method"*, donde se especifica que será una petición de tipo *"POST"*, el cual envía los datos y causa efectos en el servidor.

La función *"done"* es la forma de implementar una ejecución posterior al éxito. Si la devolución de datos es exitosa, se mostrará un mensaje de éxito y se actualizará la tabla con los nuevos datos de la observación cargada, por medio de la función *"update"*, que vuelve a dibujar la tabla de datos en el contexto actual; en caso contrario, se muestra un mensaje de error y no se refresca la tabla.

A continuación, se muestra la acción observación en el controlador *"legajos"*, que se configura en el parámetro URL del método *AJAX*.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



```
protected function accion_observaciones(){
    Empleado::contiene();
    $empleado = Empleado::obtener($this->request->post('cuit'));
    $observacion = \App\Modelo\Observacion::obtener($this->request->post('id_ob'));
    $observacion->id_empleado = $empleado->id;
    $observacion->descripcion = $this->request->post('descripcion');
    $user = Usuario::obtenerUsuarioLogueado();
    $observacion->id_usuario = $user->id;
    $observacion->id_bloque = $this->request->post('bloque');

    if(!$observacion->id) {
        $observacion->fecha = \DateTime::createFromFormat('d/m/Y H:i:s', gmdate('d/m/Y').'.0:00:00');
        if(!empty($observacion->descripcion)){
            $datos = $observacion->alta();
        }else{
            $data['error'] = true;
            (new Vista(VISTAS_PATH.'/json_response.php',compact('data'))->pre_render());
        }
    } else {
        $datos = $observacion->modificacion();
    }

    $data = [ 'result' => true];
    (new Vista(VISTAS_PATH.'/json_response.php',compact('data'))->pre_render());
}
```

Figura 6.7.4. Acción observaciones en el archivo del controlador Legajos.php (imagen propia)

En la figura 6.7.4, la acción “observaciones” obtiene la instancia del empleado a través del CUIT que se pasa al parámetro “data”. También se obtiene una instancia del objeto “observación” por medio del id “id\_ob”. El método “obtener” del modelo observación establece una instancia de la observación a partir del id; si el id es nulo, brinda una instancia vacía del objeto observación. A partir de este objeto, se agregan sus propiedades (id\_empleado, descripción, id\_usuario, y id\_bloque), para poder darlo de alta, en el caso de que no exista el id de la observación, o para realizar una modificación, si existiera el id.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

La tabla en la vista contendrá un id llamado “observación” que permitirá inicializar el *dataTable* en el archivo *JavaScript* y cargar los datos mediante un *AJAX* que apunta a una dirección, pasando como parámetro el CUIT del agente; proceso que se muestra a continuación:

```
url:$url_base + '/index.php/legajos/ajax_lista_observaciones/'+$cuit,
```

Figura 6.7.5.URL establecida como la fuente de datos Ajax para la tabla (imagen propia)

El método “ajax\_lista\_observaciones” va a preparar la “data” para refrescar la tabla cuando obtenga la información del CUIT del agente.

Al inspeccionar el navegador, se obtiene la información del servidor a través de la petición *AJAX* del listado; proceso representado en la imagen siguiente:

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

```
array (size=3)
  'recordsTotal' => string '2' (length=1)
  'recordsFiltered' => string '2' (length=1)
  'data' =>
    array (size=2)
      0 =>
        array (size=8)
          'id' => string '2' (length=1)
          'id_usuario' => string '5' (length=1)
          'usuario' => string 'adm rrhh' (length=8)
          'id_bloque' => string '6' (length=1)
          'bloque' => string 'Antigüedad' (length=11)
          'fecha' => string '29/07/2019' (length=10)
          'descripcion' => string 'dsdsdsdf' (length=8)
          'id_logueado' => int 5
      1 =>
        array (size=8)
          'id' => string '1' (length=1)
          'id_usuario' => string '5' (length=1)
          'usuario' => string 'adm rrhh' (length=8)
          'id_bloque' => string '4' (length=1)
          'bloque' => string 'Perfiles de Puestos' (length=19)
          'fecha' => string '29/07/2019' (length=10)
          'descripcion' => string 'nueva observacion' (length=17)
          'id_logueado' => int 5
```

Figura 6.7.6 Datos devueltos por el método "ajax\_lista\_observaciones" (imagen propia)

Los datos que se muestran en la Figura 6.7.6 serán los que se muestren en el listado de las observaciones.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

## 7. Conclusiones

Lo expuesto a lo largo de este informe de Práctica Profesional Supervisada permite arribar a las siguientes conclusiones:

La aplicación desarrollada reemplaza la gestión de los datos de los agentes que antes se realizaba por medio de archivos Excel. La configuración de restricciones de acceso a los usuarios de las distintas direcciones se realizó asignando permisos a través de la clase AppRoles. Con esta configuración, según las reglas en los requerimientos y roles, se restringe el acceso a diferentes acciones, solapas, secciones, campos o botones de la aplicación. Con estas restricciones, además, se limita el uso de la información del sistema a las direcciones responsables de tal información.

La manipulación de los archivos Excel permitía que el ingreso de la información, en muchos casos, no fuera válida, pudiendo ingresar datos incompletos, erróneos o diferentes (incoherentes entre sí), por parte de cada usuario o rol. En el nuevo sistema, se validan los datos que son ingresados, asegurando su integridad y coherencia. El sistema de almacenamiento relacional normalizado permite tener bases de datos más limpias y se esclarece la estructura, para que los datos sean más fáciles de localizar. Previamente a el desarrollo del sistema, el uso que hacían las diferentes direcciones de los archivos con información referente a cada empleado admitía que estos archivos quedaran desactualizados o incompletos, lo que ponía en riesgo la integridad de los datos, o posibilitaba el almacenamiento de datos inconsistentes; en el nuevo sistema, se genera una única fuente de información estructurada que permite la cooperación entre las distintas direcciones y que estas puedan acceder a información actualizada siempre que lo necesiten.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Asimismo, el sistema es escalable en lo que se refiere a funcionalidad, lo que significa que es posible agregar nuevos módulos y funcionalidades. Al contener datos globales de los empleados, se podrá, en un futuro, generar reportes estadísticos concernientes al nivel educativo, puestos de trabajo, horarios de trabajo, etc. conforme surjan nuevas necesidades en los usuarios y estos detecten las capacidades que brinda la nueva estructura diseñada, objeto de la PPS.

### 7.1. Validación personal del trabajo realizado

Durante el desarrollo del sistema, se evidenció la necesidad de una colaboración conjunta entre los desarrolladores y el líder de proyecto, quien mantiene una relación directa con los usuarios finales de la aplicación web, para la definición correcta de los requerimientos, la utilidad y funcionamiento del sistema, y, así, detallar las necesidades de los principales actores. También se consideró fundamental la colaboración con el líder técnico, quien aporta al equipo de desarrollo su experiencia y conocimiento técnicos, sirviendo de guía para la correcta resolución y desempeño en las tareas.

Con el desarrollo de la práctica, se consolidan los conocimientos adquiridos en los años de cursada, ya que, por medio de estos, se profundizan los conocimientos sobre el diseño de los detalles técnicos para la implementación de las tareas que se llevan a cabo en la actividad profesional; asimismo, se desarrollan en detalle las metodologías de trabajo utilizadas, se analizan las relaciones laborales, distinguiendo los perfiles dentro del equipo de desarrollo y la practicante, se familiariza con las tecnologías y herramientas de desarrollo de *software*, gracias a la investigación para la resolución de tareas, sumada a las bases de la formación académica previa.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

Entre los conocimientos adquiridos en la Universidad, se destaca lo aprendido en las asignaturas de Ingeniería del Software, Base de Datos, Metodologías de Programación, Proyecto de *Software*, Administración de proyecto, Algoritmos y Programación y Complejidad Temporal.

Cabe destacar que existen muchas mejoras y ampliaciones que se realizarán a futuro, como las que se citan en el siguiente punto, pero que quedan fuera del alcance de este proyecto de PPS, por su duración y extensión. Sin embargo, las funcionalidades descritas en el apartado de “Tareas a ejecutar” están implementadas. Luego, desde el punto de vista de las áreas vinculadas a su uso y los líderes del proyecto, el trabajo fue realizado satisfactoriamente.

## 7.2. Posibles mejoras y ampliaciones

- A futuro, será necesario mejorar el tiempo de respuesta de la aplicación en la presentación de la gestión de legajos, para lo que será primordial analizar la forma en que se generan las vistas y buscar la manera de mejorar esta *performance*.
- En la siguiente etapa de desarrollo del sistema, debe diseñarse la posibilidad de exportar la información referida a los agentes a modo de informe, pudiendo seleccionar la información de las distintas pestañas. Para ello, será necesario mejorar la estructura de los datos que se requieran en los informes a fin de que sean de alta accesibilidad o de que tengan una *performance* aceptable, generando vistas en Mysql o creando nuevas estructuras de datos.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- Desde la perspectiva del usuario, el cambio de paradigma que plantea el sistema ha de abrir nuevas posibilidades que pudiera brindarle el sistema. En términos concretos, esto significa que SIGARHU es un proyecto preparado para agregar nuevos módulos de gestión y los nuevos módulos pueden adaptarse a la estructura de datos que existe y realizar una tarea independiente y específica, encaminada a resolver un problema sin depender o afectar otros módulos.
- El proyecto SIGARHU es pensado como un proyecto que interactúa con otros proyectos, por los que se desarrollará una API. Esta facilitará la relación con otras aplicaciones, para el intercambio de mensajes y datos. En un principio, la API proporcionará datos del legajo del agente a la aplicación para la gestión de contratos SIGECO, pero, fuera de esto, actualmente, las áreas de Gestión participantes están revisando en qué otros proyectos desarrollados en la DIS podrían utilizar los datos de SIGARHU, de manera ágil.

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

## 8. Bibliografía

- AA.VV. (2010) "Tooltips en HTML". En: *Trucos Informaticos* Disponible en <<https://trucosinformaticos.wordpress.com/2010/09/28/tooltips-en-html/>>[Consulta: 19 de octubre de 2019].
- AA.VV. (2013) *Guía de los fundamentos para la dirección de proyectos (Guía del PMBOK) 5ta. Edición*. NewtownSquare, EE.UU. Project Management Institute, Inc.
- AA.VV. (2014). "Programación Orientada a Objetos". En: *Wikibooks*. Disponible en <[https://es.wikibooks.org/wiki/Programaci%C3%B3n\\_Orientada\\_a\\_Ojetos/Texto\\_completo](https://es.wikibooks.org/wiki/Programaci%C3%B3n_Orientada_a_Ojetos/Texto_completo)> [Consulta: 20 de agosto de 2019].
- AA.VV. (2015). "Bug". En: *Telecom ABC*. Disponible en <<http://www.telecomabc.com/b/bug.html>> [Consulta: 20 de agosto de 2019].
- AA.VV. (2016). "Resolución 111/2016 del Ministerio de Transporte". En: *Infoleg*. Disponible en: <<http://servicios.infoleg.gob.ar/infolegInternet/anexos/260000-264999/261609/norma.htm>> [Consulta: 17 de abril de 2019].
- AA.VV. (2017). *The JSON Data Interchange Syntax. Standart ECMA-404, 2,1*. Disponible en: <<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>>[Consulta: 19 de octubre de 2019].
- AA.VV. (2018). "Ambientes de desarrollo de software: Buenas prácticas". En: *PMOInformatica*. Disponible en: <<http://www.pmoinformatica.com/2012/09/ambientes-de-desarrollo-de-software.htm>>[Consulta: 20 de agosto de 2019].
- AA.VV. (2019a). "ABM - Sección BD/Programación". En: *GlosarioIT.com*. Disponible en <<https://www.glosarioit.com/ABM>>[Consulta: 25 de octubre de 2019].
- AA.VV. (2019b). *AJAX Introduction*. Disponible en: <[https://www.w3schools.com/xml/ajax\\_intro.asp](https://www.w3schools.com/xml/ajax_intro.asp)>[Consulta: 20 de agosto de 2019].
- AA.VV. (2019c). "Cadenas de caracteres (Strings)". En: *PHP*. Disponible en <<https://www.php.net/manual/es/language.types.string.php#language.types.string>> [Consulta: 20 de agosto de 2019].
- AA.VV. (2019d). "Concepto de Plugin". En: *Neoattack*. Disponible en: <<https://neoattack.com/neowiki/plugin/>> [Consulta: 19 de octubre de 2019].

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:



- AA.VV. (2019e). “Función Callback”. En: *MDN web docs* Disponible en <[https://developer.mozilla.org/es/docs/Glossary/Callback\\_function](https://developer.mozilla.org/es/docs/Glossary/Callback_function)> [Consulta: 19 de octubre de 2019].
- AA.VV. (2019f). *Generalidades del protocolo HTTP*. Disponible en: <<https://developer.mozilla.org/es/docs/Web/HTTP/Overview>> [Consulta: 15 de octubre de 2019].
- AA.VV. (2019g). *Gitea compared to other Git hosting options*. Disponible en: <<https://docs.gitea.io/en-us/comparison/>> [Consulta: 20 de agosto de 2019].
- AA.VV. (2019h). “Hardcode”. En: *Wikipedia*. Disponible en <[https://es.wikipedia.org/wiki/Hard\\_code](https://es.wikipedia.org/wiki/Hard_code)> [Consulta: 19 de octubre de 2019].
- AA.VV. (2019i). “Introducción a las APIs web”. En: *MDN Web Docs*. Disponible en: <[https://developer.mozilla.org/es/docs/Learn/JavaScript/Client-side\\_web\\_APIs/Introducci%C3%B3n](https://developer.mozilla.org/es/docs/Learn/JavaScript/Client-side_web_APIs/Introducci%C3%B3n)> [Consulta: 15 de octubre de 2019].
- AA.VV. (2019j). *Introducing JSON*. Disponible en: <<http://json.org/>> [Consulta: 19 de octubre de 2019].
- AA.VV. (2019k). *JavaScript*. Disponible en: <<https://developer.mozilla.org/es/docs/Web/JavaScript>> [Consulta: 20 de agosto de 2019].
- AA.VV. (2019l). *jQuery.ajax()*. Disponible en <<https://api.jquery.com/jquery.ajax/>> [Consulta: 19 de setiembre de 2019].
- AA.VV. (2019m). *Ministerio de Transporte, Administración Centralizada - Desconcentrada*. Disponible en: <<https://mapadelestado.jefatura.gob.ar/organigramas/transporte.pdf>> [Consulta: 17 de abril de 2019].
- AA.VV. (2019n). *PHP Arrays*. Disponible en <[https://www.w3schools.com/php/php\\_arrays.asp](https://www.w3schools.com/php/php_arrays.asp)> [Consulta: 19 de octubre de 2019].
- AA.VV. (2019ñ). *PHP Tutorial*. Disponible en <<https://www.w3schools.com/php/>> [Consulta: 20 de agosto de 2019].
- AA.VV. (2019o). *PHP*. Disponible en <<https://www.php.net/manual/es/index.php>> [Consulta: 20 de agosto de 2019].
- AA.VV. (2019p). *¿Qué es JavaScript?*. Disponible en: <https://www.nextu.com/blog/que-es-javascript/> [Consulta: 19 de octubre de 2019].

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- AA.VV. (2019q). *REST*. Disponible en <<https://developer.mozilla.org/es/docs/Glossary/REST>>. [Consulta: 19 de octubre de 2019].
- AA.VV. (2019r). *SQL Tutorial*. Disponible en <<https://www.w3schools.com/sql/>> [Consulta: 20 de agosto de 2019].
- AA.VV. (s.f.). “Definición de Login (informática)”. En: *Alegsa*. Disponible en <<http://www.alegsa.com.ar>> [Consulta: 20 de agosto de 2019].
- AA.VV. (s.f.). “About”. En: *Git*. Disponible en: <<https://git-scm.com/about>> [Consulta: 19 de octubre de 2019].
- AA.VV., (s.f.). *Poncho*. Disponible en: <<http://argob.github.io/poncho/>> [Consulta: 19 de octubre de 2019].
- Alegsa, Leandro. (2010). “En informática ¿es lo mismo postear que setear?”. En: *Alegsa*. Disponible en <<http://www.alegsa.com.ar/>> [Consulta: 19 de octubre de 2019].
- Alegsa, Leandro (2019). *Definición de Login (informática)*. Disponible en: <<http://www.alegsa.com.ar/Dic/login.php>> [Consulta: 20 de agosto de 2019]
- Álvarez, Miguel Ángel (2004). *Programación orientada a objetos en PHP*. Disponible en: <<https://desarrolloweb.com/articulos/1540.php>> [Consulta: 20 de agosto de 2019].
- Arjonilla, Rafa (2019). *BackEnd*. Disponible en: <<https://rafarjonilla.com/que-es/backend/>> [Consulta: 19 de octubre de 2019].
- Barzanallana, Rafael (2016). *Plantillas para el desarrollo de sitios web*. Disponible en: <<https://www.um.es/docencia/barzana/DAWEB/Desarrollo-de-aplicaciones-web-templates.html>> [Consulta: 19 de octubre de 2019].
- Chapaval, Nicole (2017). *Qué es Frontend Backend*. Disponible en: <<https://platzi.com/blog/que-es-frontend-y-backend/>> [Consulta: 17 de Abril de 2019].
- De Seta, Leonardo (2009). *No se recomienda borrar datos*. Disponible en <<https://dosideas.com/noticias/base-de-datos/712-no-se-recomienda-borrar-datos>> [Consulta: 19 de octubre de 2019].
- Dudler, Roger. (s.f.). *Gitla guía sencilla*. Disponible en: <<https://rogerdudler.github.io/git-guide/index.es.html>> [Consulta: 20 de agosto de 2019].

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

- Larriba, Luis Miguel (s.f.). *Metodología Kanban: ventajas y características*. Disponible en:<<https://www.getbillage.com/es/blog/metodologia-kanban-ventajas-y-caracteristicas>>[Consulta: 19 de octubre de 2019].
- Leodorf, Rasmus y Tatroe, Kevin. *Programing PHP*. Sebastopol:O'Reilly, 2002.
- Nator, Andres (2016). *MVC (Model, View, Controller) explicado*. Disponible en:<<https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>> [Consulta: 20 de agosto de 2019].
- Ordoñez, Miguel Alejandro Esteban (2019). *GIT FLOW: Tipos de Ramas*. Disponible en:<<https://openwebinars.net/blog/git-flow-tipos-de-ramas/>> [Consulta: 19 de octubre de 2019].
- Ortiz, Euriles (2018). *Qué es un framework en informática o programación*. Disponible en:<<https://blog.hostdime.com.co/que-es-un-framework-informatica-programacion/>> [Consulta: 19 de octubre de 2019].
- Pereira, Jose Javier (2019). "Mapa de procesos". En: *Manual de Procedimientos para Nuevo Desarrollo, Mantenimiento Evolutivo y Correctivo de Software*. Buenos Aires: Dirección de Integración de Sistemas, pp. 20-27.
- PerezArrieu, Federico (2019). "Tipos de usuario del sistema". En: *Documento de acuerdo F-02*. Buenos Aires: Dirección de Integración de Sistemas, pp. 7-17.
- PeriañezGomez, Francisco (2016) "Características de virtualbox". *Tutorial de VirtualBox*. En: *Formación Profesional a través de Internet*. Disponible en:<[http://fpg.x10host.com/VirtualBox/caractersticas\\_de\\_virtualbox.html](http://fpg.x10host.com/VirtualBox/caractersticas_de_virtualbox.html)> [Consulta: 20 de agosto de 2019].
- Ramirez Navia, Fernando (2018). *¿Qué es y para qué sirve MySQLDatabase?*. Disponible en:<<https://itsoftware.com.co/content/que-es-y-para-que-sirve-mysql/>>[Consulta: 19 de octubre de 2019].
- Silva Bijit, Leopoldo (2010). "Capítulo 1. Funciones Booleanas". En: *Sistemas Digitales*. Disponible en:<<http://www2.elo.utfsm.cl/~lsb/elo211/clases/c01.pdf>> [Consulta: 5 de diciembre de 2019].

## 9. Glosario

### A

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

**Agente:** término utilizado por las áreas intervinientes del sistema para denominar a los empleados del Ministerio.

**API REST:** una API REST es un servicio HTTP que puede ser llamado mediante librerías y herramientas web estándar. API (Application Programming Interface - Interfaz de Programación de Aplicaciones) representa una interfaz de comunicación entre componentes de software. REST (Representational State Transfer - Transferencia de Estado representacional) permite que un recurso sea transferido con su estado y relaciones mediante formatos y operaciones estandarizadas. HTTP (Hypertext Transfer Protocol) es el protocolo estándar para el intercambio de datos en web. (AA.VV., 2019i: s/p).

**Array:** los arrays (arreglos) o matrices son un tipo de estructuras de datos que nos permiten almacenar múltiples elementos de un tipo de datos similar, en una sola variable, a los que se accede utilizando su índice o clave. (AA.VV., 2019n: s/p)

## B

**Back-end:** el back-end es la parte del código que se encarga de la lógica del funcionamiento interno y la que se conecta con la base de datos y el servidor. (Arjonilla, Rafa, 2019: s/p)

**Baja lógica:** la baja lógica consiste en que el dato no esté disponible para el usuario, como si se hubiera borrado, pero internamente sólo se lo marca para indicar que está dado de baja, a diferencia del borrado físico, que es la capacidad de borrar completamente el registro. (De Seta, Leonardo, 2009: s/p)

**Bug:** un bug en un software ocasiona que el programa colapse o que de errores. La mayoría de los bugs son fallos humanos (AA.VV., 2015: s/p)

## C

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

**Callback:** es una función que se ejecutará después de que otra función haya terminado de ejecutarse (AA.VV, 2019e: s/p)

**Commit:** el comando *Commit* sirve para registrar cambios en el repositorio (Dudler., s.f: s/p).

**Constante booleana:** Una constante booleana es un valor perteneciente al conjunto {0,1}. (Silva, 2010: s/p)  
**F**

**Framework:** esquema para el desarrollo o implementación de una aplicación. Se compone de un conjunto de módulos listos para la creación de elementos recurrentes estandarizados en el desarrollo de los sistemas (Ortiz, 2018: s/p).

**Front-end:** el front-end a diferencia del back-end se encarga de la interacción con el usuario. (Arjonilla, 2019: s/p)

**H**

**Hardcodeo:** es un término que hace referencia a una mala práctica en el desarrollo de software que consiste en incrustar datos directamente en el código fuente del programa (AA.VV., 2019h: s/p)

**J**

**JSON:** Estándar basado en texto plano para el intercambio de información. (AA.VV., 2019j: s/f)

**L**

**Loguearse:** el terminologueado viene de “login” que es el nombre dado al momento de autenticarse e ingresar a un [sistema](#) (Alegsa, 2017: s/p)

**M**

**MYSQL:** sistema de gestión de base de datos relacional (RDBMS), de código abierto, basado en lenguaje de consulta estructurado (SQL). (Ramirez Navia, 2018, s/p)

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional:

**Merge:** el comando Merge une dos o más ramas de desarrollo (Dudler., s.f: s/p)

## P

**Plugin:** un *Plugin* es un fragmento o componente de código hecho para ampliar las funciones de un programa o de una herramienta (AA.VV., 2019d: s/p)

**Push:** el comando Push actualizar referencias remotas junto con objetos asociados repositorio (Dubler., s.f: s/p)

## S

**Seteo:** El termino setear viene de "set", que significa configurar (Alegsa, 2010: s/p)

**String:** es una cadena de caracteres (Leodorf y Tatroe, 2002: s/p)

## T

**Template:** los *Templates* son plantillas para hacer páginas con un diseño general. Es la parte visible de un sitio web. (Barzanallana, 2016: s/p)

**Tooltip:** Los tooltips son unas pequeñas etiquetas emergentes que se muestran cuando el cursor del ratón queda parado durante unos instantes encima de un componente de la página, como por ejemplo, sobre un campo o un botón (AA.VV., 2010:s/p).

Firma estudiante:	Firma docente supervisor:	Firma docente tutor TAPTA:	Firma tutor organizacional: