Rosatto, Daniel

Práctica profesional supervisada, proyecto GIDAPPF

2019

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)
Proyecto GIDAPPF
Informe Final

ÍNDICE GENERAL

ÍNDICE DE IMÁGENES

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

DATOS DEL ESTUDIANTE

Apellido y Nombres: Rosatto, Daniel Alejandro

DNI: 26927057

Nº de Legajo: 3787

Correo electrónico: danielrosatto@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 42

DOCENTE SUPERVISOR

Apellido y Nombres: Dr. Ing. Morales, Martín

Correo electrónico: martin.morales@unaj.edu.ar

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

Apellido y Nombres: Dra. Ferrari, Mariela

Correo electrónico: mariela_c_ferrari@hotmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Programa “Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social”. Ing. en Informática. Instituto de Ingeniería y Agronomía.

Dirección: Avenida Calchaquí 6200, Florencio Varela.

Teléfono: +54 11 4275-6100

Sector: Sistemas de Cómputos.

TUTORES DE LA ORGANIZACIÓN

Apellido y Nombres: Ing. Cortes Bracho, Oscar
Correo electrónico: ocortesbracho@unaj.edu.ar

Apellido y Nombres: Mg. Ing.Encinas, Diego Omar
Correo electrónico: dencinas@unaj.edu.ar

FIRMA DEL COORDINADOR DE LA CARRERA

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

Resumen

La Práctica Profesional Supervisada (PPS) es un desarrollo informático que consiste en la creación de un software original, a nombre del Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social, para la carrera de Ingeniería en Informática, perteneciente al Instituto de Ingeniería y Agronomía de la Universidad Nacional Arturo Jauretche (UNAJ). El proyecto gestionará los datos de alumnos para mantener la historia educativa de cada estudiante del Plan FinEs. Dejará a disposición de los administradores, coordinadores y secretarios, información estadística sobre los ingresantes y estudiantes, para realizar el seguimiento y planificar las comisiones del siguiente ciclo.

El software, solicitado por la coordinadora de la Unidad de Vinculación y Calidad Educativa Regional, profesora María Elena Zambella, tiene la finalidad de otorgar soporte informático a medida al sector de Administración, que se encarga de dar de alta y realizar el seguimiento de los estudiantes del Plan Fines. Actualmente, este sector documenta el rendimiento de estos estudiantes con un sistema implementado mediante herramientas de ofimática estándares. La Administración del Plan Fines, con quienes, como se mencionó, también se realizará la presente práctica profesional, está compuesta por el Secretario del Plan Fines, el profesor Rubén Romero, y por el Coordinador del Plan, el profesor Christian Pidalá.

El soporte informático consiste en el desarrollo de una aplicación web que mantenga los datos registrados actualmente y permita manipularlos eficazmente para beneficio de toda la comunidad educativa del Plan Fines. Para ello, se deberá interpretar el diseño y la funcionalidad de la aplicación web, realizando una investigación sobre la administración actual del Plan Fines. Se debe lograr un acuerdo sobre todas las funcionalidades del sistema a fin de que este que cumpla con las siguientes condiciones:

- Debe ser lo suficientemente útil para reemplazar al sistema actual.

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

- Debe ser pasible de ser mantenido o sustentable por el soporte de los servidores de la Universidad Nacional Arturo Jauretche.
- Debe ser finalizado a tiempo para su uso en el año en curso y los subsiguientes, dado el plazo de la Práctica Profesional Supervisada y las necesidades del sector.
- Además del proyecto de desarrollo de la aplicación web a medida para el Plan Fines, utilizando una metodología ágil, deberá crearse un manual del usuario.

En primera instancia, se generarán módulos de implementación que responderán a las funcionalidades administrativas que se releven durante la investigación. Estos módulos de la aplicación representan cada rol que intervendrá, de manera que se reflejen los aportes individuales en el espacio compartido. Los cuatro módulos a diseñar son "Miembros", "Estudiantes", "Reportes y estadísticas" y "Sistema de alertas".

El módulo "Miembros" se encargará del alta, baja y modificación del personal del Plan Fines. El módulo "Estudiantes" estará dedicado a las altas, bajas y modificación de su condición. También se encargará de la gestión de las comisiones, las aulas y los estudiantes libres. El tercer módulo que se desarrollará mostrará los reportes y estadísticas de las comisiones y la actuación en clase de los estudiantes, reflejando la asistencia y la calificación. Por último, se incorporará un módulo que permita establecer notificaciones emergentes configurables, que respondan ante condiciones que requiera el administrador, con el objetivo de realizar la atención especial ante prioridades devueltas por las estadísticas. Como se dijo, este módulo se denominará "Sistema de Alertas".

Objetivos

El objetivo general de la PPS es desarrollar un software original y a medida, que implemente un sistema nuevo de gestión de Estudiantes para el Plan Fines.

Los objetivos específicos del presente trabajo son los siguientes:

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

- Se deberá identificar el circuito administrativo, basándose en una investigación sobre la administración actual del Plan Fines, en los sectores de Secretaría y de Coordinación. Se definirán las funciones y los perfiles de los miembros del Plan Fines y se incorporarán diagramas del circuito administrativo.
- Se efectuará un desglose de las funciones encontradas en el diagrama y se documentará el alcance del proyecto, que será tomado en cuenta para la planificación de las siguientes etapas.
- Se realizará un prototipo que será sometido a evaluación por los miembros del equipo de co-creación del nuevo sistema.
- Se desarrollará un Diagrama de Entidad - Relación para comenzar a implementar la base de datos. Este diagrama consiste en la identificación de datos que se considerarán categóricamente agrupables en la entidad, que llevará la denominación de la categoría. Además, se debe establecer la correspondencia entre las otras entidades que se encuentren, es decir "la relación". En el gráfico, se describen, por una parte, las entidades como bloques titulados con su denominación e ítems de datos en el interior; y, por otra, las relaciones, como líneas que unen entidades, indicando en cada extremo la proporción de cada una.
- Se implementará asimismo el servidor de la aplicación, según lo dispuesto por la documentación del alcance del proyecto.

Tareas a ejecutar

Las tareas a realizar serán de dos tipos distintos: en primer lugar, las que hacen a la delimitación de necesidades por parte de la coordinación del Plan Fines, el profesor tutor de la UNAJ y la tutora del Taller de Apoyo a la Producción de Textos Académicos (TAPTA), de la UNAJ, y, por otra, las que se relacionan directamente con la

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

implementación del software. A continuación, se explican ambas, de acuerdo con el orden cronológico de su realización:

- Se presentará ante el profesor tutor de la UNAJ la planilla de inscripción a la Práctica Profesional Supervisada Nº 12.
- Se deberá formar un equipo multidisciplinario con los coordinadores del Plan Fines para realizar la co-creación del software a medida de las necesidades del Plan Fines, quienes serán los usuarios finales del software entregado.
- El estudiante realizará un reconocimiento de las actuales funciones y limitaciones del sistema administrativo del Plan Fines, teniendo en cuenta las necesidades y posibles soluciones a problemas vinculados al uso, desde el punto de vista informático, junto con el equipo de co-creación del nuevo sistema, mencionado en el punto “Resumen” de esta presentación.
- Se realizarán reuniones presenciales y virtuales con la profesora tutora de TAPTA, para establecer objetivos y estructura del trabajo a presentar, en primer lugar, y, luego, para la revisión y corrección de la documentación y del informe de la Práctica Profesional Supervisada.
- Se realizarán reuniones con los tutores de la institución para la revisión y corrección del aspecto tecnológico del avance desarrollado.
- Se presentará el plan de trabajo preliminar ante el profesor tutor de la UNAJ.
- Se coordinarán reuniones periódicas con los responsables del Plan Fines, para informar avances, aceptar cambios y sugerencias que serán documentadas en minutas.
- Se realizará una planificación del proyecto, basada en una Metodología de desarrollo de software aplicable para este caso. Esta planificación es necesaria para resolver de antemano la estructura básica que definirá el orden de ejecución de la escritura del código fuente.
- Se ejecutará el desarrollo planificado, implementando con el lenguaje de

programación Ruby y la base de datos PostgreSQL, de acuerdo con los pasos a desarrollar que resultaron de la planificación y que se describen a continuación:

- El primer desarrollo de la aplicación es el del módulo del acceso y de registro de "Usuarios", que se implementará de manera que el ingreso al sistema sea confirmable, recuperable, registrable, recordable y validable desde la interfaz de usuario.
- En segundo lugar, se trabajará en el módulo "Miembros", que, en primer lugar, brinda la información pública de todos los integrantes, y, en segundo lugar, en el nivel de acceso de administradores, permite la modificación del plantel, con la posibilidad de mostrar una foto de la persona.
- Luego, se elaborará el módulo "Estudiantes", que se divide en la exposición en tres submenús, llamados "Ingresantes", "En comisión" y "Libres", correspondiendo a todas las condiciones en las que se pueda encontrar un estudiante registrado en el sistema. Además, este módulo debe dar soporte de búsqueda, alta, baja y modificación a la gestión de los estudiantes, según el nivel de acceso ingresado.
- A continuación, se trabajará en el módulo "Reportes y estadísticas", que muestra tablas y gráficos de los resultados del día a día, según el desempeño de los miembros en cada nivel. Dentro de este módulo, el perfil "Docente" es el único que contiene los formularios de asistencia y calificación de cada estudiante para generarlos desde aquí.
- Finalmente, el módulo "Sistema de alertas" es el menú al que se accede desde el perfil administrador para configurar las condiciones y el alcance de diferentes tipos de alerta. El "Sistema de alertas" para los demás roles será una notificación emergente sobre el pie de página de la aplicación web. Dicha alerta se produce si las estadísticas que surgen de los estudiantes o las comisiones presentan coincidencias con alguna condición

configurada por el administrador.

- Se presentará el informe de avance ante la tutora de TAPTA y ante el profesor tutor de la UNAJ.
- Se realizará una Planificación del Proyecto de la etapa final.
- Se realizará la ejecución del desarrollo de la etapa final, además de realizarse un balance del trabajo ejecutado, se propondrán posibles mejoras o se establecerán posibles desarrollos ulteriores y extensiones de la tarea de la PPS.
- Se presentará el informe final ante la tutora de TAPTA y ante el profesor tutor de la UNAJ.
- Se realizará la presentación y defensa oral del informe.

Cronograma de trabajo

Como paso previo a la realización de la PPS (pero parte integral del presente trabajo), se presenta a continuación un cronograma pormenorizado, desde el inicio de la tarea. Habiendo sido analizada y aceptada la propuesta del profesor tutor de la UNAJ, dr. ing. Martín Morales, se establece la aceptación del estudiante, ap. Daniel Alejandro Rosatto, para la realización de la Práctica Profesional Supervisada, en los términos y condiciones que se detallaron previamente. Se acuerda que los resultados del desarrollo deben ser documentados y presentados en tiempo y forma para poder ser evaluados y calificados por los Tutores de la Práctica Profesional Supervisada.

En esta instancia, es necesaria una reunión con la coordinadora de la Unidad de Vinculación y Calidad Educativa Regional, Profesora María Elena Zambella y con los coordinadores del Plan Fines, el profesor Rubén Romero y el profesor Christian Pidalá, los primeros días de septiembre, para establecer el equipo de trabajo y co-creación del desarrollo y para definir, asimismo, la motivación de este desarrollo.

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

También es necesaria una reunión con la tutora del Taller de Apoyo a la Producción de Textos Académicos de la UNAJ, dra. Mariela Ferrari, a mediados de septiembre, para establecer los conceptos a transmitirse en la documentación de la Práctica Profesional Supervisada y también acordar el formato de esta.

Además, se debe realizar una tercera reunión, a fines de septiembre, con los tutores de la Institución, ing. Oscar Cortes Bracho y el mg. ing. Diego Omar Encinas, que verificarán la viabilidad del sistema y corregirá detalles técnicos. El resto del cronograma, que hace al desarrollo en sí, se presenta en la Imagen 1.

TAREAS	Septiembre	Octubre	Noviembre	Diciembre	Febrero	Marzo
Reuniones para delimitar alcances y objetivo del sistema con...	Coordinadora de la Unidad de Vinculación y Calidad Educativa Regional, Profesora María Elena Zambella y Coordinadores del Plan Fines, Profesores Rubén Romero y Christian Pidalá	Coordinadora de la Unidad de Vinculación y Calidad Educativa Regional, Profesora María Elena,Zambella y Coordinadores del Plan Fines, Profesores Rubén Romero y Christian Pidalá		Coordinadora de la Unidad de Vinculación y Calidad Educativa Regional, Profesora María Elena,Zambella Coordinadores del Plan Fines, Profesores Rubén Romero y Christian Pidalá		Coordinadora de la Unidad de Vinculación y Calidad Educativa Regional, Profesora Zambella, María Elena, Coordinadores del Plan Fines, Profesores Rubén Romero y Christian Pidalá
Coordinar con...	Tutor de la Institución, Ing. Oscar Cortes Bracho, Tutora del Taller de Apoyo a la producción de Textos Académicos de la UNAJ, Dra. Mariela Ferrari		Tutor de la Institución, Ing. Oscar Cortes Bracho, Tutora del Taller de Apoyo a la producción de Textos Académicos de la UNAJ, Dra. Mariela Ferrari		Tutor de la Institución, Ing. Oscar Cortes Bracho, Tutora del Taller de Apoyo a la producción de Textos Académicos de la UNAJ, Dra. Mariela Ferrari	Tutor de la Institución, Ing. Oscar Cortes Bracho Tutora del Taller de Apoyo a la producción de Textos Académicos de la UNAJ, Dra. Mariela Ferrari
Establecer	Alcance del proyecto	Validación del Prototipo y del alcance del proyecto.	Servidor y entorno de desarrollo	Primera Entrega del Plan del informe de PPS...	Versiones de entrega final de la aplicación web	Entrega final
Investigar	Circuito administrativo	Informe de avance	Informe de avance	Versiones de entrega de la aplicación web	Manual del usuario	Manual del usuario
Informe de la PPS	Plan preliminar			Informe de avance		
Configurar	Metodología de desarrollo.	Canales de comunicación		Soporte de testeo		
Implementar	Prototipo	Diagrama Entidad-Relación	Base de datos	Versiones de prueba de la aplicación web	Versiones de prueba de la aplicación web	
Escribir última versión de informe				Informe de avance		Informe Final
Defensa oral del informe						Días finales del mes

Imagen 1: *Tabla con el cronograma de trabajo detallado* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

Implementación de la metodología de desarrollo ágil

La variante de la metodología de desarrollo se explica en siete apartados que se considera necesario desarrollar por separado, para clarificar de avance de la Práctica Profesional Supervisada, en la implementación del sistema de Gestión Integral de Alumnos para el Plan Fines (GIDAPPF).

En el primer apartado, se describe el establecimiento de los canales de información, que alcanzan a la coordinadora de la Unidad de Vinculación y Calidad Educativa Regional, profesora María Elena Zambella, a la administración del Plan Fines, compuesta por el secretario del Plan Fines, el profesor Rubén Romero, y por el coordinador, el profesor Christian Pidalá. También se incluye la comunicación con la tutora del Taller de Apoyo a la Producción de Textos académicos de la UNAJ, dra. Mariela Ferrari; y con los mencionados tutores de la organización ing. Oscar Cortes Bracho y mg. ing. Diego Omar Encinas.

El segundo apartado explica la aplicación de las herramientas de desarrollo, elaborando un análisis en función de la metodología ágil aplicada en la implementación a entregar.

En el tercer apartado, se describe la implementación del prototipo y sus aspectos destacados, desgloses y consideraciones a tener en cuenta para el desarrollo en el Lenguaje de programación *Ruby*.

El cuarto apartado es el detalle de la solución informática que se desarrolla, incluyendo, los servicios utilizados, la tecnología aplicada y restricciones del sistema.

El quinto apartado se refiere a la documentación generada, para los usuarios finales y para el personal de soporte de sistemas, que se encargará de la configuración y el mantenimiento.

Finalmente, en el sexto apartado, se exponen las conclusiones y el trabajo futuro, mientras que en el séptimo apartado, se establecen las referencias bibliográficas utilizadas para la implementación.

1. Establecimiento de los canales de información

Cualquier actividad organizada entre distintos actores comienza interactuando en ámbitos que proporcionan un medio de comunicación suficientemente eficaz para cumplir con las planificaciones y realizar los objetivos. Previamente, en el informe preliminar, se estableció que la metodología de producción de software debía ser ágil, por lo tanto, el canal de comunicación tiene que convertirse en un medio de comunicación para trabajar en equipo.

El equipo de trabajo es interdisciplinario; en este sentido, la comunicación se producirá con destinatarios del área académica, de la administración y de sistemas informáticos.

Los medios de comunicación disponibles son reuniones presenciales y virtuales, el intercambio de correos electrónicos y diferentes formatos de mensajería instantánea.

La eficacia del desarrollo ágil depende en gran medida de los resultados de las reuniones, es decir, se deben planificar cuidadosamente los temas a tratar y registrar los temas pendientes. La documentación se intercambia a través de correos electrónicos. Los mensajes instantáneos son poco utilizados en este caso, ya que son útiles para comentarios muy puntuales o en el caso de tener un mayor número de desarrolladores de software.

La documentación requerida por la Práctica Profesional Supervisada se comparte mediante servicios de computación en la nube con todos los tutores; en dicho espacio virtual, también están disponibles las minutas de reunión y los diagramas de análisis.

Como se utiliza en paralelo una herramienta de publicación de versiones, se genera documentación técnica de orden público. Se detalla este tema en el apartado 2.

2. Herramientas de desarrollo

Desarrollar un software a medida no implica que se tenga que programar instrucción por instrucción la funcionalidad entera de este, ya que esto sería una producción muy costosa

en horas de desarrollo. La ingeniería en informática aplicada permite producir software a medida sin malgastar recursos materiales, ni humanos.

Como reacción a la forma tradicional en la que se desarrollaba el software en la década de los ochenta, nace la ingeniería de software y, junto con ella, los modelos generales de proceso de software que permiten la producción de software con mayor calidad para proveer de soluciones según las necesidades. Uno de los modelos generales del proceso de software surgidos es el que responde a las necesidades del ambiente moderno de negocios, el cual genera sistemas basados en computadora y productos de software que evolucionan rápida y constantemente, un proceso al que comúnmente se llama "Desarrollo Ágil". Ésta metodología pone énfasis en la reutilización del código fuente, es decir, lo que se programó y se publicó debe ser aplicado en los casos que corresponda, ya que la solución publicada es mejorada continuamente por los autores o colaboradores. Estas soluciones genéricas son conocidas como "librerías" y están diseñadas para ser reutilizadas por programadores de sistemas de cómputo.

Hay una diversidad enorme de soluciones informáticas publicadas bajo distintos acuerdos y en diversos soportes informáticos de publicación, que se conocen como "Sistemas de Código abierto", o de "Código Libre", en cuyo desarrollo las universidades juegan un papel muy importante tanto en aportes de recursos, como también en innovación e investigación académica. Sin embargo, también existen organizaciones y empresas que intervienen generando solidez en estos sistemas.

El lenguaje de programación elegido, que se menciona en el informe preliminar, es *Ruby* (Matsumoto, 2001: 11). Fundamentalmente, se eligió este lenguaje para utilizar la herramienta *Rails* (Hansson Heinemeier, 2005: 1) que depende de éste y que se explica a continuación, en el apartado 2.1.

El principal criterio de selección que se tuvo en cuenta es que *Ruby* es un lenguaje de Programación Orientado a Objetos, es decir, un paradigma que indica cómo se diseñan los algoritmos partiendo del postulado "Todo es un objeto". Otra ventaja es el gran soporte

de librerías propias de *Ruby*, que se denominan "Gemas". Son muy fáciles de instalar a través del comando *Gem install*, como se muestra en la Imagen 2, o, también, desde el archivo de creación del proyecto *Gemfile*.

```
$ gem install orats
```

Imagen 2: *Bloque de código con el comando de instalación de librerías de extensión de Ruby* (imagen propia)

En la imagen 2, se muestra la instalación de la Gema Orats versión 5.2.3, que es la indicada para el proyecto GIDAPPF. *Orats* provee de proyecto de aplicación web en *Ruby*, guardado dentro de un contenedor *Docker*, que es el sistema de virtualización que se explica en el apartado 2.3.

En la imagen 3, se muestra la ejecución de *Orats* porque lo significativo, en este caso, es que cualquier computadora se puede transformar en un servidor de desarrollo con los comandos necesarios. Para crear el proyecto GIDAPPF, se comienza desde `$`, que es el *prompt* de la terminal del sistema operativo, es decir, los comandos se ingresan seguidamente. La primera y segunda línea son comandos de configuración del entorno de

```
$ export PATH="$PATH:$(ruby -e 'print Gem.user_dir')/bin"
$ export GEM_HOME=$HOME/.gem
$ orats new gidappf

        task  Check if path exists
        task  Check if template exists
        task  Create 'base' project

        path  gidappf

        task  Prepare and run everything
        open  carefully read and edit the `.env` file
        move  cd gidappf
        run  docker-compose up --build

        task  Initialize the database in a 2nd Docker-enabled terminal
        note  OSX / Windows users can skip the --user flag
        run  docker-compose exec --user "$(id -u):$(id -g)" website rails db:reset
        run  docker-compose exec --user "$(id -u):$(id -g)" website rails db:migrate
```

Imagen 3: *Bloque de código con la creación del proyecto Gidappf* (imagen propia).

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

programación para cumplir con requisitos de *Orats*. Luego, el comando Orats new gidappf genera el proyecto y termina mostrando la ayuda sobre los comandos básicos para continuar luego del arranque del servidor de desarrollo.
Se nombró entonces al proyecto *gidappf* basado en *Orats*, dentro de la carpeta GIDAPPF que es la principal de dicho proyecto. Los detalles de los componentes se explicarán en los siguientes apartados.

2.1. Lenguaje de programación *Ruby*

El autor del lenguaje de programación *Ruby*, Yukihiro Matsumoto, mezcló partes de sus lenguajes favoritos *Perl, Smalltalk, Eiffel, Ada* y *Lisp,* para formar un nuevo lenguaje orientado a objetos muy robustos, a fin de que la respuesta de los algoritmos programados fuera fiel incluso si se contara con requisitos mínimos en las dependencias recomendadas. Matsumoto ha manifestado que está "tratando de hacer que *Ruby* sea natural, no simple" (Matsumoto, 2000: 1407), de una forma que se asemeje a la vida real.
La programación orientada a objetos es un paradigma que se aplica a cualquier elemento que se quiera definir como un objeto con propiedades y acciones delimitadas computacionalmente. Los lenguajes de programación tratan de implementar este paradigma, limitados por los requerimientos del sistema y de hardware. En el caso de *Ruby*, se denomina a las propiedades "variables de instancia" y las acciones son conocidas como "métodos".
La orientación a objetos pura de *Ruby* se suele demostrar con un simple código que aplica una acción a un número.

```
5.times { print "Nos *encanta* Ruby -- ¡es fuera de serie!" }
```

En este lenguaje, el punto representa el llamado a un método o atributo del objeto a su izquierda, que, en este caso, es el número 5. En otros lenguajes de programación orientados a objetos, los números no califican como objetos, sólo tienen valor constante y no tienen otro propósito. Sin embargo, este caso posibilita el método *times* que ejecuta el

contenido entre llaves 5 veces (o "5 times", en inglés). Esta particularidad es una gran ventaja que se propaga directamente al desarrollo.

El método *times* no es el único posibilitado por el número, existe todo tipo de métodos utilizables. *Ruby* sigue la influencia del lenguaje *Smalltalk,* ya que permite asignar métodos y variables de instancia a todos sus tipos de datos.

Estas características de *Ruby* facilitan al desarrollador proveer de claridad y síntesis en el código, una cualidad muy necesaria en sistemas en constante evolución y desarrollados en equipo. La primera versión se publicó en 1995 y se popularizó rápidamente mediante la licencia pública y el código abierto. A continuación, se crearon comunidades de soporte y desarrollo de complementos adicionales, que dotaron al lenguaje de extensibilidad y flexibilidad.

2.2. *Rails*

Las aplicaciones web constituyen la mayor parte de los servicios de internet y cuentan con una amplia aceptación, siendo particularmente popular entre los desarrolladores. Es muy variada la gama de tecnologías de implementación de aplicaciones web, tanto de código libre, o de su contrapartida, de software propietario.

Dentro de las implementaciones, existen los *frameworks* (marcos). Son soluciones informáticas diseñadas para ser utilizadas por los programadores y que desarrollan soluciones muy demandadas, tales como videojuegos, sistemas de seguridad informática, testeos de calidad del software y aplicaciones web, entre otros. Proporcionan el conjunto de librerías específico para comenzar con el sistema a desarrollar, pero con la arquitectura básica lista para recibir los cambios específicos demandados por el presente proyecto.

Rails es uno de los *frameworks* de desarrollo de aplicaciones web de código libre de mayor aceptación y con gran soporte de colaboradores. Es muy versátil y contempla la integración de las últimas tecnologías. Brinda la posibilidad de hacer los cambios al código

fuente utilizando cualquier editor de texto. Los comandos de *Rails* se ejecutan en terminales de línea de comando estándar. La solidez de las aplicaciones desarrolladas en *Rails* se da porque el *framework* está diseñado para generar un proyecto basado en una arquitectura de software muy estudiada.

Se trata de la arquitectura "Modelo - Vista - Controlador" o MVC, que, en la Imagen 4, se presenta en un diagrama en bloques. Se muestra que la arquitectura interactúa directamente con el navegador web del usuario que está a la izquierda. El bloque de la vista es el encargado de disponer adecuadamente los elementos visuales y las líneas de texto, que proporciona el bloque del modelo, si son variables. El bloque del modelo brinda las instancias de los objetos que contienen datos que necesariamente deben ser respaldados, según se requiera en la etapa de diseño. Con respecto al bloque controlador, cumple la función de interpretar los eventos de entrada del navegador web realizados por el usuario y convertirlos en estados de objetos del modelo.

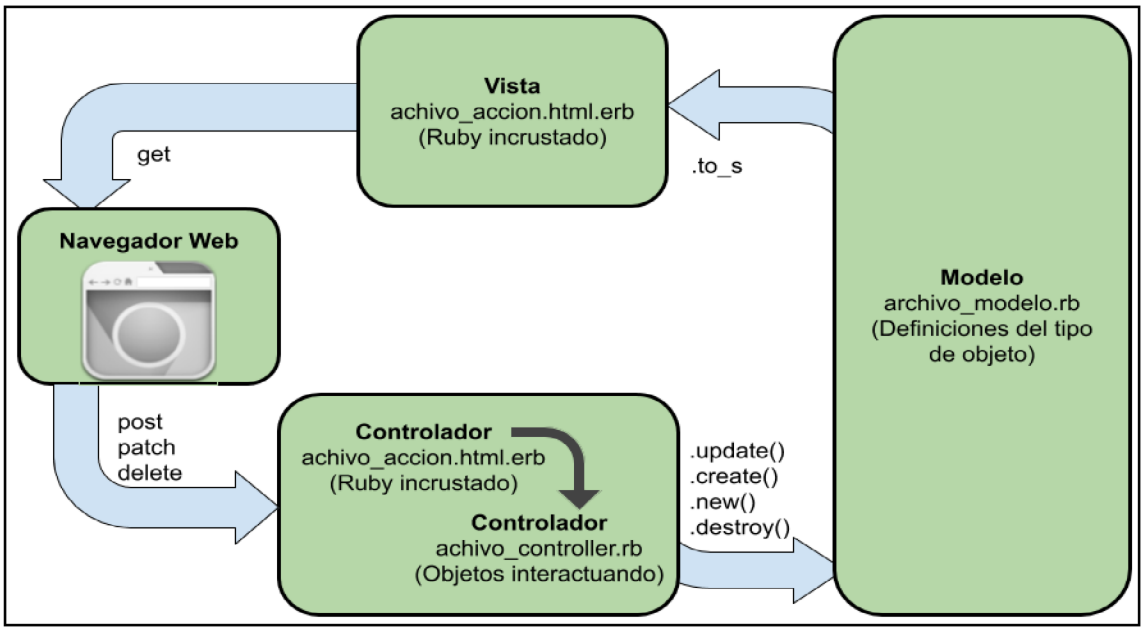

Imagen 4: *Arquitectura MVC como un diagrama en bloques* (imagen propia).

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

Rails genera dicha arquitectura en su funcionamiento y permite preparar el código fuente estructural básico, trabajando en el lenguaje de programación *Ruby*, para ser incorporado de manera directa al proyecto, junto con diversos comandos que posibilitan el desarrollo ágil, sin comprometer la arquitectura del sistema y, sobre todo, sin malgastar recursos.
Ejemplos de los comandos más usados son el generador de código de *Rails*, console y test. A continuación, explicaremos brevemente cada uno de estos comandos y su uso en nuestro proyecto.

2.2.1. Generador de código *Rails*

Es una herramienta que soluciona de manera escalonada los conceptos elementales de la arquitectura Modelo - Vista - Controlador. Es una manera ágil de establecer el esquema necesario de los componentes de proyecto incorporados a la arquitectura en el lenguaje *Ruby*.
El primer paso es generar los códigos fuente de las migraciones. Las migraciones son códigos fuente que actúan automáticamente para crear el modelo de datos específico del proyecto, es decir, la equivalencia informática de las definiciones obtenidas por el análisis funcional. Este paso es necesario porque el resguardo de los datos y la persistencia están a cargo del sistema de base de datos. El sistema de base de datos es independiente de *Rails*, pero se comunican constantemente mediante el sistema operativo. La capa de persistencia de los datos es protegida por el sistema de gestión de bases de datos, que tiene su propio control de acceso y sincronismo y *Rails* tiene los comandos para manejarlos integrados. Estos comandos se detallan en el apartado 2.2.2., pero aquí es necesario aclarar que, en *Rails*, las migraciones creadas se registran en una carpeta del proyecto y se aplican al proceso de programación del sistema.
El segundo paso corresponde a la sincronización del modelo de datos; se refiere a que el modelo debe tener una instancia entre los objetos de *Rails*, y los cambios de estado de los objetos se deben sincronizar automáticamente con la base de datos. Se guarda un

archivo en el proyecto dentro de la carpeta "Models" con la clase que sincroniza el modelo.

En la imagen 5, que se presenta a continuación, se muestra un ejemplo práctico de generación de código.

```
$ rails generate model Role name:string description:text  level:float
enabled:boolean
```

Imagen 5: *Bloque de código con el comando Rails para la creación de modelo de datos* (imagen propia).

Con el comando descripto en la imagen 5, se crean los archivos de migración y las clases del modelo; en este caso, Role es el modelo con los atributos *name*, *level* y *enabled*.

Otros elementos que el generador de *Rails* permite realizar son los controladores y las vistas. Estos son comandos a los que solamente se les indica el modelo de referencia y generan una página web para crear valores y modificarlos dentro del modelo de datos. También existen los generadores de testeos, con una variedad de plantillas que será vista en detalle en el apartado 2.2.3.

2.2.2. Comando *Rails console*

Las aplicaciones web no sólo mantienen páginas web. Otra forma de acceso a los datos es mediante la línea de comandos del servidor. *Rails console* es un ambiente en línea de comandos, destinado a ser utilizado por los desarrolladores y que permite interactuar directamente con las instancias y las clases de objetos Ruby cargados en la memoria de la computadora. Se trata de un entorno que favorece la prueba de pequeños algoritmos, antes de considerarse como parte del código fuente. En la siguiente captura de pantalla, la imagen 6, se muestra la línea de comandos del servidor anfitrión del proyecto GIDAPPF, comunicándose con el comando *Rails console.*

Imagen 6: *Acceso a rails console* (imagen propia).

Como puede verse, el proyecto está encapsulado en un sistema de virtualización, y, por esto, es necesario indicar el prefijo docker-compose exec, que se explicará detalladamente en el apartado 2.4. El indicador de acceso a *Rails console* es el cursor *irb(main)* que muestra el resultado de consultar el primer objeto *Rails* de clase *User*, (*User.first*). Se accede al modelo de datos desde esta línea de comandos, a través de un patrón de arquitectura diseñada para estos casos. Este patrón es *Active Record* (Fowler, 2003: 160) y es estándar para muchos *frameworks* de desarrollo de aplicaciones web. Por lo tanto, los objetos del modelo de *Rails* responden de acuerdo a Active Record.

2.2.3. Comando *Rails test*

Las distintas características de los elementos de una arquitectura de software Modelo - Vista - Controlador deben interactuar entre validaciones y restricciones que podrían estar en el proceso de desarrollo. Cuando se desarrolla en grupos grandes, es difícil medir los tiempos de desarrollo y, muchas veces, se dan por finalizados algoritmos incompletos, debido a fallas de comunicación.

Una manera de evitar estas fallas es generar los testeos del proyecto, y, así, saber cuándo finaliza correctamente el desarrollo. Los testeos deben ser diseñados para establecer los márgenes de calidad del software. De esa manera, cuando se corren los testeos y pasan exitosamente, se establece que la respuesta del sistema es óptima.

Los testeos son fundamentales a la hora de integrar desarrollos grupales, ya que los aportes individuales testeados son integrables al proyecto en forma automática, si previamente se diseñaron dichos testeos.

En la versión 5, *Rails* integra la herramienta de testeo Minitest y se accede a esta por consola. Para ejecutar todos los testeos disponibles, se utiliza el comando "rails test".

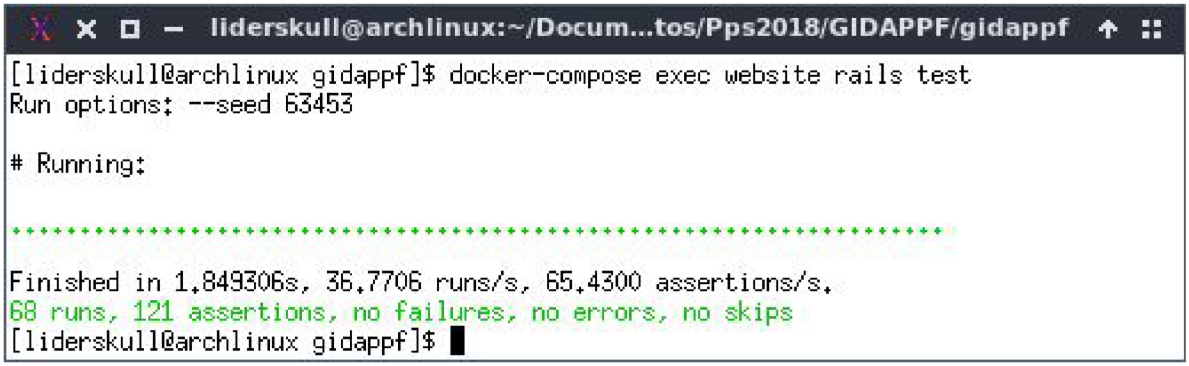

Imagen 7: *Mensajes de salida de rails test interactuando a través del contenedor docker* (imagen propia).

La utilización del comando "rails generate model", que se ve en la Imagen 5, resulta particularmente eficaz, ya que también genera un archivo plantilla de testeo del modelo, dentro de la carpeta "test/models" del proyecto.

Se debe observar que *Rails* está contenido dentro de un *Docker*, herramienta que se detallará en el apartado 2.4. Los comandos vistos anteriormente funcionan si se antepone el gran prefijo: *docker-compose exec --user "$(id -u):$(id -g)" website,* como se observa en la Imagen 7.

2.3. Base de datos

Una característica de las aplicaciones web es que los datos que se generan por la actividad de esta se resguardan en sistemas específicos de gestión de datos. A estos sistemas se los conoce como "Bases de datos" (Silberschatz, 1999: 24), que cuentan con

un lenguaje de consultas y modificación de datos estándar, llamado “SQL”, por las siglas *Structured Query Language*.

De la gran variedad de tecnologías de implementación de Bases de datos, se elige implementar el proyecto *GIDAPPF* con *PostgeSql* (Stonebraker, 1996), ya que este es de código libre, puede ser utilizado en forma centralizada o distribuida y, además de tener un amplio soporte, es muy usado por grandes organizaciones.

Para interactuar con la base de datos durante el desarrollo, se utilizan múltiples formas de acceso para verificar que el modelo de datos responde adecuadamente, es decir, que la comunicación con *PostgereSql* no sólo se logra con el acceso al servidor contenido en el Sistema de Virtualización *Docker* (Hykes, 2013; Turnbull, 2016), sino que se aprovechan los comandos de rails console y las migraciones y los valores de los requisitos iniciales del sistema.

En particular, el comando que se usa es *rails db: setup*, que establece el inicio del sistema tomado del archivo seeds.rb (Imagen 8), pero es un comando que se debe utilizar por única vez al instalar el sistema, si no, se perderán datos críticos que se hayan ingresado durante la actividad productiva.

```
25  ##################################################################
26  # Valores de Roles. Se debe jerarquizar a mayor level, mayor
27  ##################################################################
28  Role.destroy_all
29  Role.create!([
30    {
31      name: "Ingresante",#id1
32      description: "Comienza el tramite para participar del Pla
33      created_at: gidappf_start_time,
34      enabled: true, level: 10.0
35    },
36    {
37      name: "Estudiante",#id2
38      description: "Usuario que participa de las cursadas y est
39      created_at: gidappf_start_time,
40      enabled: true, level: 20.0
41    },
```

Imagen 8: *Archivo seeds, de valores iniciales del modelo* (imagen propia).

2.4. Sistemas de virtualización

Para que el proyecto *GIDAPPF* se pueda integrar con los servidores de la Universidad Nacional Arturo Jauretche, un requisito de sistema es que debe ser contenido por el sistema *Docker*.

Para explicar el sistema de contenedores *Docker*, primero, se deben entender los sistemas de virtualización.

Un sistema de virtualización informático es una solución que permite obtener el máximo desempeño y seguridad de los *hardwares* que se disponen. Se utilizan para que una computadora sea capaz de hacer funcionar más de un sistema operativo a la vez, y, entonces, así, multiplicar las actividades que se realizan en paralelo. Son seguros porque

están controlados por un sistema hipervisor, de manera tal que las posibles fallas de cada sistema operativo virtualizado queden aisladas.

Hay gran variedad de sistemas de virtualización y son ampliamente utilizados en los Data Centers que funcionan con una alta disponibilidad de servicio.

La virtualización a través del sistema *Docker* es una tecnología de vanguardia que está siendo aceptada gracias a los casos exitosos. El mayor logro de *Docker* consiste en crear una plataforma de funcionamiento e intercambio de sistemas operativos, guardados en contenedores. La idea de los contenedores ya existía desde antes y estos eran utilizados por expertos, pero *Docker* popularizó su uso.

La diferencia entre los sistemas de virtualización y los sistemas de contenedores es que, para virtualizar, es necesario indicarle al hipervisor la disponibilidad reservada de recursos de *hardware* que se utilizarán para cada sistema operativo virtual. Los contenedores *Docker* no funcionan con hipervisores, sólo con gestores que dan soporte a la ejecución, creación e interacción de dichos contenedores, por lo tanto, el trabajo en paralelo se divide en forma dinámica.

En la Imagen 9, que se presenta a continuación se observa a la derecha el diagrama en bloques de un sistema de virtualización tradicional, y, a la izquierda, el diagrama de *Docker*. Es notable la menor cantidad de sistemas operativos funcionando en paralelo.

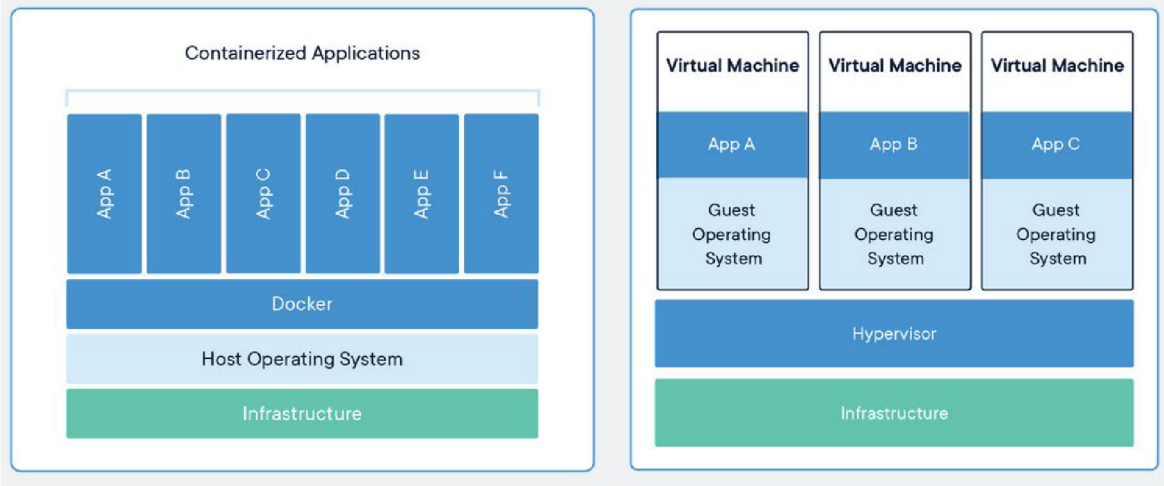

Imagen 9: *Comparación entre Docker y los sistemas de virtualización tradicionales*.
(imagen de www.docker.com)

La plataforma de *Docker* es de software propietario, pero los contenedores y el gestor son de código libre, entonces, es posible hacer cambios en contenedores, probarlos en cualquier computadora, compartirlos y publicarlos. También es posible generar códigos fuente con instrucciones para el motor del gestor *Docker*, lo cual posibilita que la creación a medida de contenedores pueda ser incluida en las metodologías de desarrollo ágil.

Se ha vuelto una herramienta de desarrollo en equipo, además de ser un sistema de virtualización que evoluciona continuamente. *Docker* es capaz de organizar un conjunto de servidores en forma sistemática mediante la herramienta docker-compose y el lenguaje de etiquetado *YAML*, que también es considerado un código fuente del proyecto.

Es posible realizar un desarrollo de aplicación web y, luego, realizar el contenedor *Docker*, pero se produce una mejor reutilización del código fuente, si se incluyen en el proyecto los códigos fuente de *Docker* y si se integran las configuraciones específicas.

Al incluir *Docker* en el desarrollo, se debe tener en cuenta que no se accede a la terminal de línea de comandos de un servidor virtualizado o "dockerizado"; para comprender este

proceso, en la Imagen 10, se resumen todos los accesos a los servicios Docker con los comandos específicos para cada capa de servicio.

Comando de acceso al servidor, desde GIDAPPF/gidappf:	Terminal de acceso
`$ docker-compose exec --user "$(id -u):$(id -g)" cable /bin/sh`	Servidor cable
`$ docker-compose exec --user "$(id -u):$(id -g)" postgres /bin/sh`	Servidor postgres
`$ docker-compose exec --user "$(id -u):$(id -g)" redis /bin/sh`	Servidor redis
`$ docker-compose exec --user "$(id -u):$(id -g)" sidekiq /bin/sh`	Servidor sidekiq
`$ docker-compose exec --user "$(id -u):$(id -g)" website /bin/sh`	Servidor rails

Imagen 10: *Tabla con la lista de comandos de accesos a las diferentes terminales del proyecto GIDAPPF* (imagen propia)

También es necesario remarcar que solamente es posible ejecutar un comando simple, como se muestra en las imágenes 6 y 7, presentadas con anterioridad.

2.5. Sistemas de control de versiones y soporte del código fuente

El desarrollo colaborativo es de gran importancia para las metodologías ágiles. La sinergia del trabajo en equipo y la buena comunicación no sólo son deseables para ejecutar eficientemente las planificaciones del desarrollo, sino que resultan imprescindibles para este fin. La integración de cada aporte individual es un problema al que los sistemas de control de versiones ofrecen solución. Los sistemas de control de versiones cuentan con un espacio de almacenamiento seguro del código fuente, al que se denomina “repositorio”.

Los repositorios poseen dos variantes: pueden ser locales o remotos. A continuación, explicaremos sucintamente la diferencia entre ambos.

La lógica de los sistemas de versionado necesita que cada desarrollador adquiera una cuenta de acceso individual. En consecuencia, el repositorio local es individual, pero el repositorio remoto es en donde se produce automáticamente la integración de cada trabajo que se publica. El repositorio remoto es un servicio de internet y existen muchos proveedores con distintos planes de acceso y formas de publicación.

De las diferentes opciones, se elige mantener el código fuente en el repositorio de *Github* bajo una licencia pública *Gpl v3*. Este repositorio es capaz de mantener diversos sistemas de control de versiones y, por esta razón, resulta el más adecuado para la función a realizar. Se elige utilizar a *Git* como el sistema de control de versiones.

La ventaja de *Git* es que no sólo es capaz de mantener una integración del proyecto, también puede integrar, en el repositorio remoto, ramas distintas de integración con distintos ciclos del proyecto. Para desarrollar el trabajo en equipo y su coordinación, es muy útil que cada integrante tenga su propia rama publicada.

En el caso del proyecto GIDAPPF, se crea la rama *master*, que contiene la versión a entregar. También se crea la rama *development*, en donde se guardan los módulos a medida que se van completando y, por último, la rama *newfeature_a,* que es la que contiene los últimos aportes inestables del sistema GIDAPPF. Entonces, cada vez que *development* publica un cambio es porque se terminó con un desarrollo de *newfeature_a*. A su vez, cada vez que master publica un cambio, es porque *development* terminó de realizar los cambios de todas las planificaciones planeadas y se generó una nueva versión.

La ventaja de utilizar al repositorio *Github* para publicar el código fuente es que no tiene restricciones para la cantidad de colaboradores ni para la cantidad de ramas de un proyecto público; la seguridad en el acceso es implementada de múltiples formas. Además, permite crear el perfil de la organización que publica en el repositorio. También

cuenta con la posibilidad de incluir descripciones en los encabezados del soporte del repositorio y espacios destacados para visualizaciones de archivos *Readme*.

Estas características de *Github*, acordes a las técnicas de autodocumentación del código fuente, son fundamentales en el despliegue de las metodologías ágiles de desarrollo. La autodocumentación consiste en utilizar de manera adecuada los caracteres etiquetadores de comentarios que todos los lenguajes de programación poseen. En el caso del proyecto GIDAPPF, es necesario generar un modelo de datos acorde a los diagramas obtenidos en el análisis funcional; de esta forma, es fácil comentar de qué se trata cada atributo de ese modelo. Los modelos de datos adquieren funcionalidad por separado en *Rails*; a saber, a un modelo le corresponde un controlador que aporta métodos. A dichos métodos se los debe comentar con un encabezado que comunique cuáles son las condiciones necesarias para aplicar el método, describir qué parámetros de entrada necesita y detallar cuáles son todos los posibles resultados que se pueden obtener luego de su ejecución. Estos comentarios de autodocumentación son importantes para el mantenimiento del sistema a largo plazo y también para la inclusión de colaboradores en el equipo de desarrollo. Asimismo, *Github* hace un resaltado al código fuente, que produce una legibilidad muy agradable del código fuente y de los comentarios.

3. Implementación del prototipo

Al abordar el análisis del circuito administrativo del Plan Fines, teniendo en cuenta la información brindada por el secretario y por el coordinador del Plan, surgió como resultado el diagrama que se muestra en la Imagen 11.

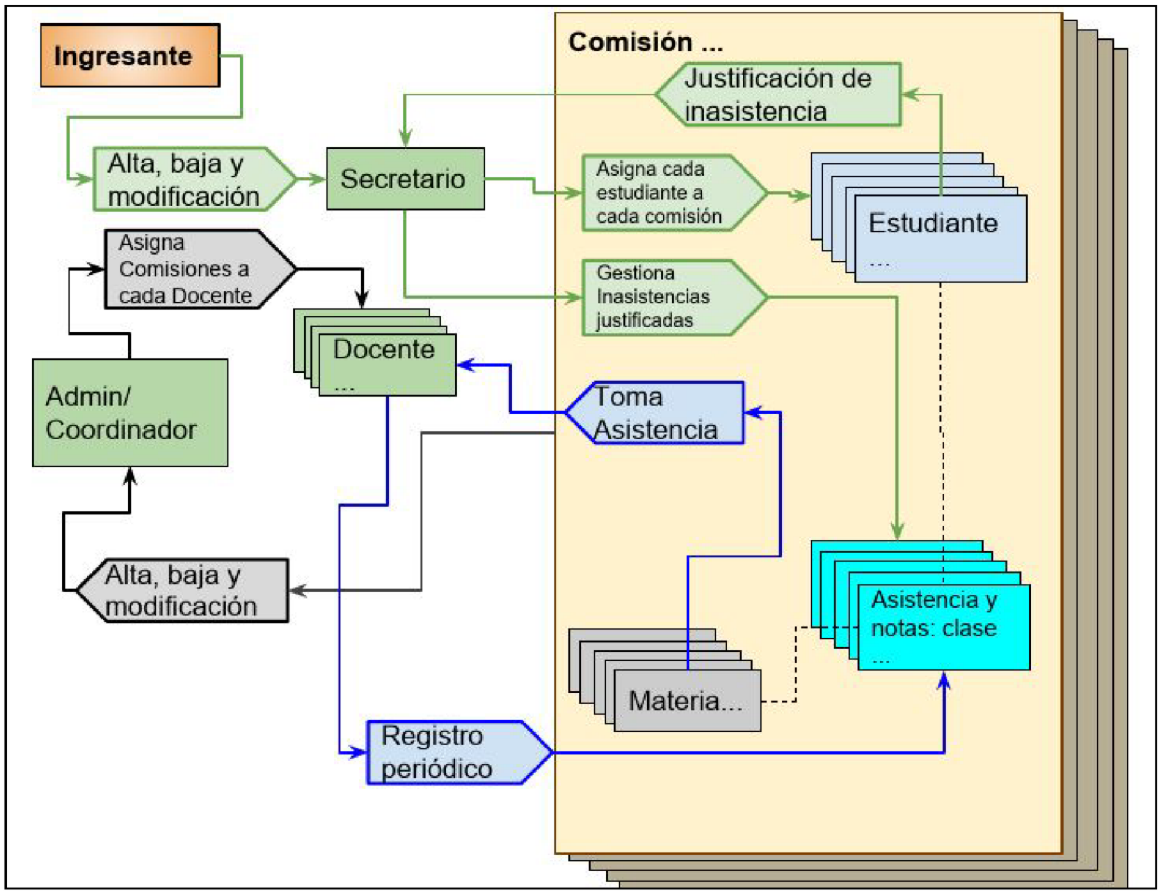

Imagen 11: *Diagrama del circuito administrativo* (imagen propia)

Dicho diagrama es representativo del sistema que se debe desarrollar, pero no detalla qué recursos gráficos se tienen en cuenta, ni cuántas pantallas aproximadas serán producidas. Para lograr una rápida aproximación al resultado final de las vistas del proyecto GIDAPPF, y con el fin de que todos los integrantes del equipo multidisciplinario tomen contacto con el sistema, se realizó un prototipo del proyecto, utilizando el plan básico de la herramienta de prototipado *Marvel*. Esta herramienta es una solución de software propietario que permite realizar un diseño con funcionalidad básica de aplicaciones web y móviles y requiere conocimientos básicos de interfaz de usuario. En la

Imagen 12, se muestra la captura de pantalla de algunas de las cincuenta y un pantallas que se publicaron.

Se presentó el prototipo y se dejó sujeto a comentarios, para que todos los integrantes pudieran realizar aportes. El aspecto general y la funcionalidad fueron aprobados por la coordinadora de la Unidad de Vinculación y Calidad Educativa Regional, profesora María Elena Zambella y por el Secretario del Plan Fines, el profesor Rubén Romero. Se especificaron cambios, como los datos personales, que se deben registrar en cada perfil de estudiante y la prioridad de que un ingresante pueda figurar de alta en las comisiones ante una situación de incumplimiento o falta de datos no primordiales. Estos cambios serían integrados en el proyecto GIDAPPF, desde la etapa siguiente del desarrollo.

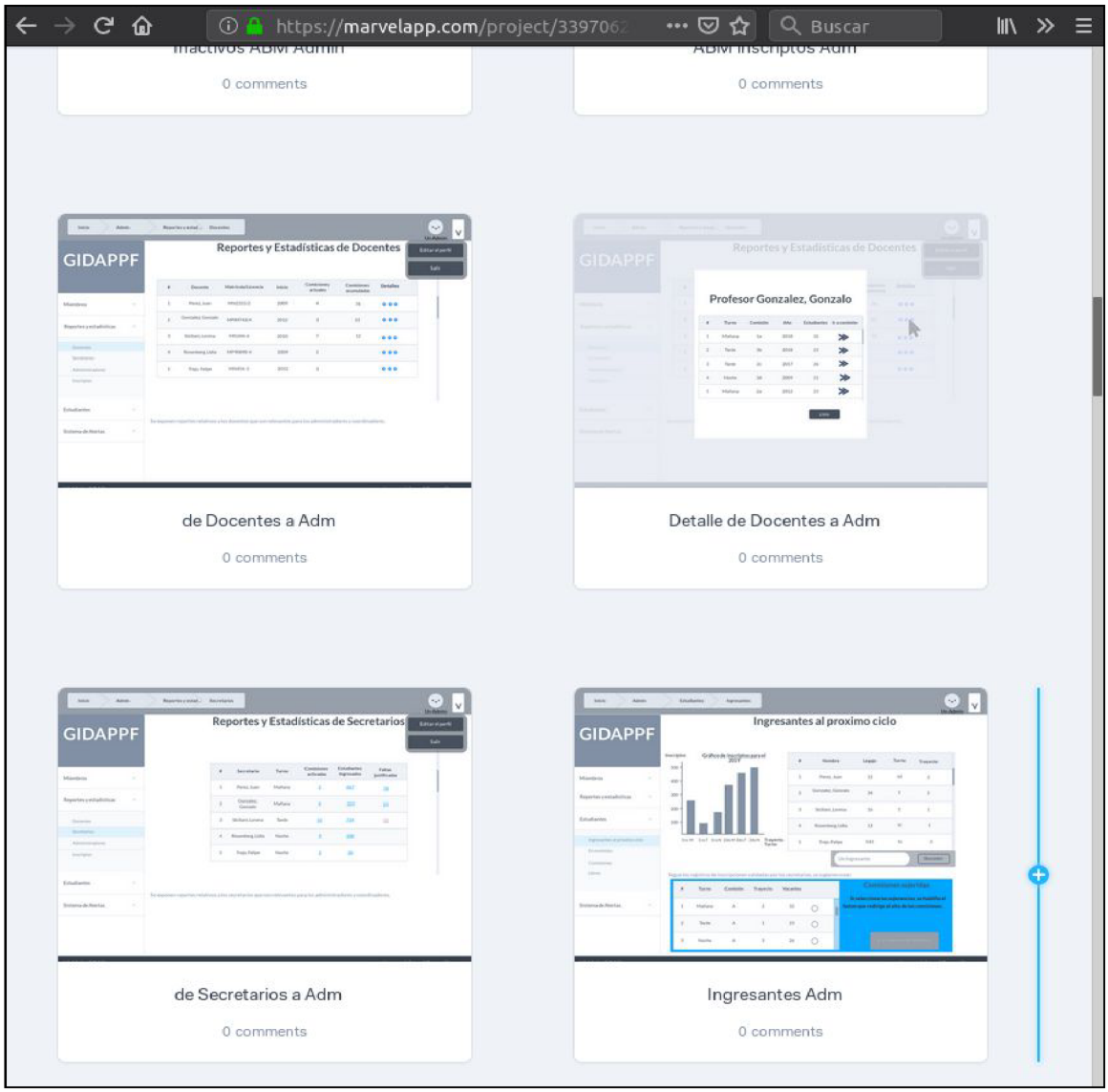

Imagen 12: *Desarrollo del prototipo de GIDAPPF*
(imagen de https://marvelapp.com/eigf43j)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

4. Solución informática en detalle

Para dejar constancia del alcance del proyecto, se enumeran a continuación los módulos y submódulos del desarrollo completo, teniendo en cuenta que el rol indicado entre corchetes es el que tendrá acceso a la funcionalidad descripta:

1. Registro Inicial de primer administrador.
2. Entrada de la cuenta del usuario registrado.
3. Modulo “Miembros”
    a. Alta, baja y modificación de Docentes.
        i. [Administración]: Presentación de Docentes registrados.
        ii. [Secretarios]: Presentación de Docentes registrados.
        iii. [Docentes]: Presentación de Docentes registrados.
        iv. [Administración]: Formulario de nuevo Docente.
        v. [Administración]: Formulario de modificación de datos de Docentes.
    b. Alta, baja y modificación de Secretarios.
        i. [Administración]: Presentación de Secretarios registrados.
        ii. [Secretarios]: Presentación de Secretarios registrados.
        iii. [Docentes]: Presentación de Secretarios registrados.
        iv. [Administración]: Formulario de nuevo Secretario.
        v. [Administración]: Formulario de modificación de datos de Secretarios.
    c. [Administración]: Alta, baja y modificación de Administradores.
        i. [Administración]: Presentación de Administradores registrados.
        ii. [Secretarios]: Presentación de Administradores registrados.
        iii. [Docentes]: Presentación de Administradores registrados.
        iv. [Administración]: Formulario de nuevo Administrador.
        v. [Administración]: Formulario de modificación de datos de Administradores.
    d. [Administración]: Crear nuevo, reactivar y modificación de inactivos de los miembros.
        i. [Administración]: Presentación de inactivos.
        ii. [Administración]: Controles de inactivos registrados.
4. Módulo “Reportes y estadísticas”
    a. [Administración]: Presentación de datos relevantes al administrador sobre los docentes. Detalle de las comisiones a cargo.

b. [Administración]: Presentación de la actuación de los secretarios.
c. [Administración]: Presentación de inscriptos al próximo ciclo.
d. [Secretarios]: Presentación del reporte de asistencia de todas las comisiones.
    i. [Secretarios]: Presentación del detalle de la asistencia en cada comisión con el formulario para seleccionar un estudiante y justificar una falta.
e. [Secretarios]: Presentación de los reportes de los vencimientos de inscripciones, comisiones y estudiantes.
f. [Docentes]: Presentación de las comisiones asignadas para reportar asistencia.
    i. [Docentes]: Formulario de asistencia de cada materia asignada a la comisión con la lista de estudiantes.
g. [Docentes]: Presentación de las comisiones asignadas para reportar notas.
    i. [Docentes]: Formulario para reporte de notas de los estudiantes asignados en la comisión seleccionada.

5. Módulo “Estudiantes”
    a. [Administración]: Creación de nuevas comisiones para el próximo ciclo.
    b. [Administración]: Presentación de estudiantes con comisión asignada. Posibilidad de mostrar el historial de cada estudiante.
    c. [Secretarios]: Presentación de estudiantes con comisión asignada. Posibilidad de mostrar el historial de cada estudiante.
    d. [Administración]: Alta, baja y modificación de comisiones.
        i. [Administración]: Presentación de las comisiones vigentes y las vencidas.
        ii. [Secretarios]: Presentación de las comisiones vigentes y las vencidas.
            1. [Secretarios]: Formulario de asignación de estudiantes a las comisiones.
        iii. [Administración]: Formulario para generar nuevas comisiones.
            1. [Administración]: Formulario y presentación para alta, baja y modificación de aulas.
    e. [Administración]: Presentación de los estudiantes libres comisión.
        i. [Administración]: Formulario de reincorporación con edición de motivo.
    f. [Secretarios]: Presentación de los estudiantes libres de comisión.

g. [Secretarios]: Alta, baja y modificación de nuevo ingresante al próximo ciclo.
    i. [Secretarios]: Formulario para crear nuevo ingresante para el próximo ciclo.
6. Módulo “Sistema de Alertas”:
    a. [Administración]: Alta, baja y modificación de alertas configurables en todos los roles.
        i. [Administración]: Presentación de los alertas vigentes y control de Alta, baja y modificación.
            1. [Administración]: Formulario de creación de nuevo alerta.
        ii. [Administración]: Presentación de alertas cumplidas; Ya no se producen porque la condición ya no se cumplirá.
        iii. [Administración]: Presentación de alertas vistas; El alerta es clickeado por el usuario pero la condición produce nuevas instancias de alerta.
        iv. [Administración]: Presentación de alertas pendientes; El alerta es cerrado por el usuario pero la condición produce nuevas instancias de alerta, indicaría irrelevancia del alerta.

Todos los módulos listados previamente se integran en el lenguaje de programación *Ruby*, bajo la arquitectura Modelo - Vista – Controlador, proporcionada por *Rails*, como se mencionó anteriormente. En el desarrollo, se utiliza de guía un diagrama que se produce mediante la aplicación de técnicas de administración de bases de datos. El Diagrama de Entidad – Relación (DER) obtenido se muestra en la imagen 13.

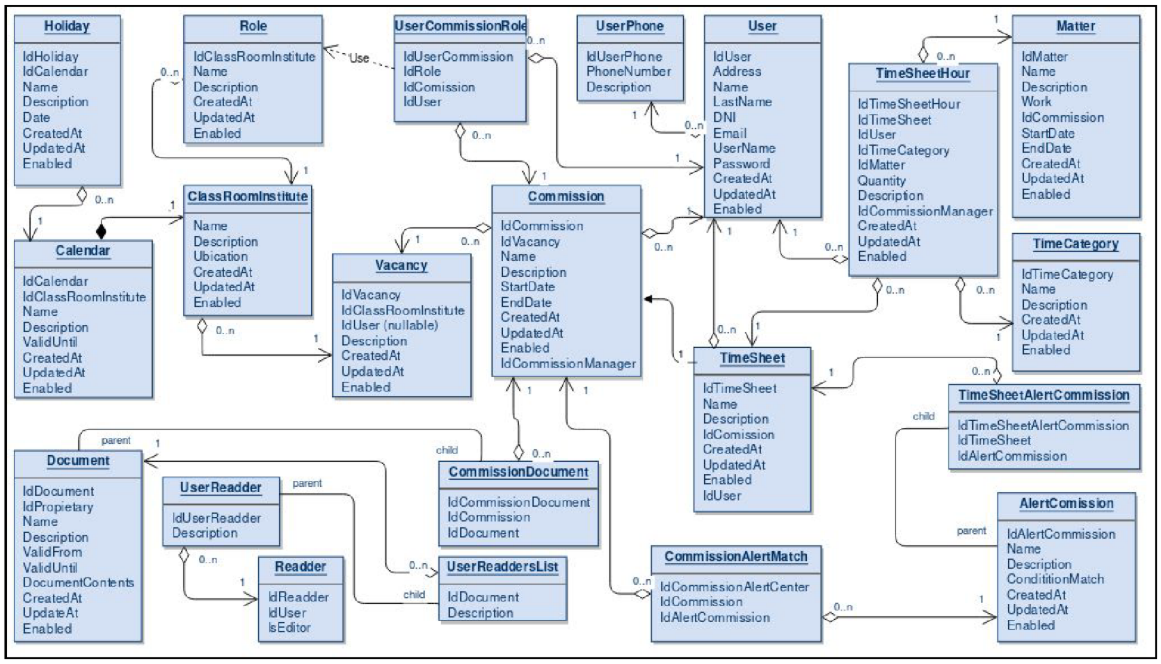

Imagen 13: *D.E.R. que se utiliza de guía para el proyecto GIDAPPF* (imagen propia)

A simple vista, el DER puede resultar complejo en su interpretación, basándose en todas las variables que se desprenden del circuito administrativo. Pero, en realidad, esto es así porque el DER es un esquema clasificatorio de los datos que se van a respaldar: entidades que permanecen inalterables, sin importar si el sistema está en pleno funcionamiento o si no se activó todavía. Así, por ejemplo, la entidad Role reserva el espacio en memoria para el objeto *Ruby* con valores "Docente", "Secretario", "Administrador", etc. según se produzca la ejecución.

El protagonismo se enfoca en la entidad "commission", que es el objeto que vincula alumnos y docentes, y que, además, generará las estadísticas y los alertas.

4.1. Problemas de diseño

El proyecto es una solución a medida, es decir, es necesario reconocer los planteos que derivan del desarrollo de algoritmos que resuelven situaciones específicas del circuito administrativo del plan FinEs.

Una implementación destacable es el cambio de diseño de entidad *"Document"* a relación *"Documents"*. En este sentido, la entidad *"Document"*, que estuvo diseñada desde el inicio del análisis funcional, en la práctica, resultó insuficiente para contemplar los documentos de entrada y salida de datos estadísticos. Además, la asignación de permisos de lectura y escritura en cada instancia *Document* se debía desarrollar en otra entidad.

La administración de FinEs proporcionó el formulario de ingreso y los campos a completar forman el perfil del estudiante. La implementación del perfil del estudiante debe ser flexible, dado que debe estar compuesta por campos opcionales y opcionalmente editables, pero también por campos con datos críticos, como el DNI, número de legajo, etc. Además, no es necesario que el perfil de estudiante tenga un usuario del sistema.

Sin embargo, los docentes deben generar documentación con la información de los perfiles presentes en clase. Asimismo, se toma en cuenta la posibilidad de que un estudiante se convierta en docente, por lo tanto, la referencia a la entidad usuario es necesaria para migrar de perfil.

Otra característica deseable en la administración FinEs, que fue implementada en el transcurso del proyecto, era que existiera la posibilidad de que los alumnos pudieran cambiar sus horarios de clase con facilidad, porque es muy complicado actualizar con cada cambio los listados fijos de cada comisión.

4.2 Implementaciones específicas

Los problemas de diseño de la entidad *"Document"* fueron analizados y se encontró otra alternativa.

La solución es que los documentos se asocien al usuario y también al perfil. De esta manera, se logra que todos los usuarios asocien su perfil por medio de su primer documento imborrable.

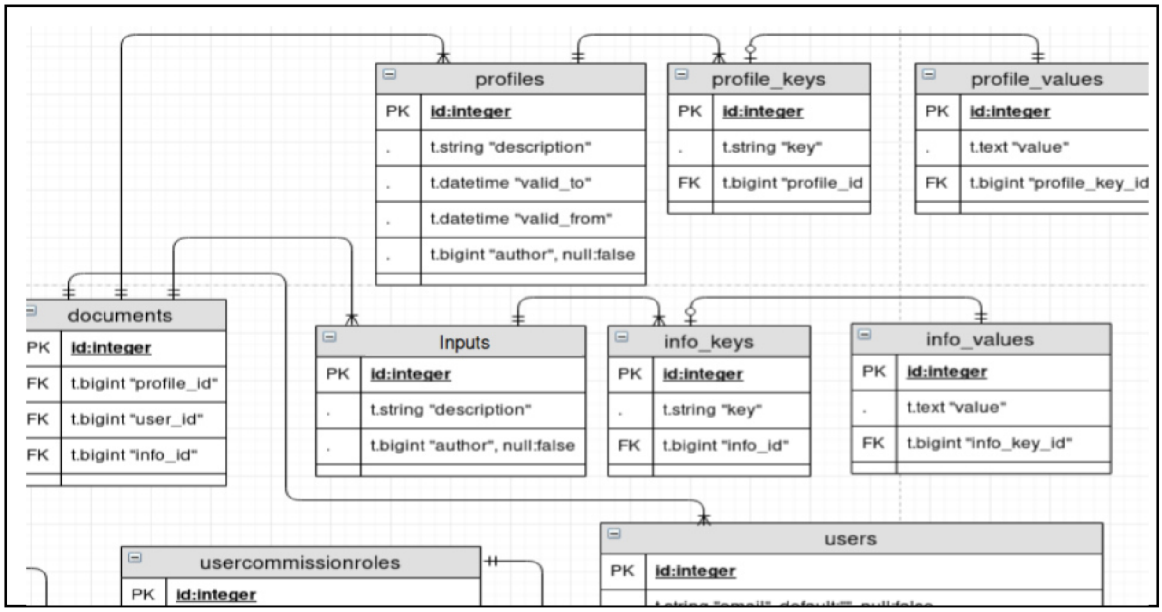

Imagen 14: *Rediseño del DER* (imagen propia).

Como se ve en la imagen 14, la entidad *Document* de la Imagen 12 cambia y se relaciona con las entidades *Input* y *Profile*. Así, *Documents* deja de ser una entidad y se convierte en relación.

Con esta alternativa, *"Documents"* serían las relaciones que vinculan al usuario con sus contenidos generados, ya sea por su perfil o por la información de sus actividades; gracias a esto, no es imprescindible generar un modelo para *"Document"*, pero sí para *"Profile"* y para *"Input"*, por lo que, necesariamente, se debe modificar el DER, como se muestra en la Imagen 14.

En este contexto, es posible crear plantillas de documentos que pueden ir generando nuevos valores, según lo requiera el sistema, sin estar limitado por el número de campos de la entidad *"Document"* o de la entidad *"Profile"*.

```
<%= form_with(model: profile, local: true) do |form| %>
  ...
  <%= form.fields_for :profile_keys, profile.profile_keys do |nested| %>
    <%= nested.hidden_field :key, :value =>
Profile.first.profile_keys[nested.index].key %>
    <%= nested.fields_for :profile_values,
profile.profile_keys[nested.index].profile_values do |nested2| %>
      <div class="field">
        <%= nested2.label Profile.first.profile_keys[nested.index].key %>
          <% unless nested.index == 2 then %>
            <%= nested2.text_field :value %>
          <% else %>
            <%= nested2.text_field :value, :readonly => true %>
          <% end %>
        </div>
    <% end %>
  <% end %>
  <div class="actions"><%= form.submit %></div>
<% end %>
```

Imagen 15: *Bloque de código fuente que utiliza el método fields_for proporcionado por el módulo nested_attributes del modelo* (imagen propia)

Esta manera de redefinir a *"Document"* es útil para reflejar en una sola vista (ver apartado 2.2), por una parte, *"Profile"*, *"ProfileKey"* y *"ProfileValue"* para la visualización de los perfiles, y, por otra, *"Input"*, *"InfoKey"* e *"InfoValue"*, para la visualización del contenido de los documentos. Para lograrlo, se utiliza una herramienta proporcionada por *Rails*: los *"nested_attributes"*. La herramienta consiste en modificar el controlador de cada entidad *"Profile"*, e *"Input"* para que tome en cuenta los sub-modelos asociados. En el caso de la vista de acción edición, la herramienta nested_attributes proporciona el método fields_for

para desglosar apropiadamente cada campo del objeto dependiente, como se muestra en la Imagen 15.

Al implementar *nested_attributes* en el bloque de la vista de la arquitectura MVC código fuente, como se observa en la Imagen 15, se explica brevemente que, en la línea 1, se declara el inicio del formulario asignado al modelo *Profile* y está finaliza en la línea 16; luego, en la línea 2, es posible invocar al método *fields_for*, porque en *Profile* se activó al módulo *nested_attributes* con el modelo *ProfileKey* y finaliza en la línea 14. En la línea 4, es implementado un segundo nivel de *fields_for*, activado sucesivamente en *ProfileKey* con el modelo *ProfileValue,* en un ciclo interno, hasta la línea 13. Esto significa que el perfil tendrá tantos campos como la cantidad de *ProfileKey* asociados y los nombres de los campos son dados por el atributo *key*. Además, los valores de los campos son almacenados en la entidad *ProfileValue* asociada a *ProfileKey*, sucesivamente asociada a *Profile*.

Esto constituye una solución a la dinámica necesaria para los perfiles y los contenidos generados, ya que, por ejemplo, los nombres de los campos de *Profile* podrían ser editables por el usuario. Esta solución no afecta solo al bloque de la vista y al modelo de la arquitectura MVC, sino que el controlador también participa.

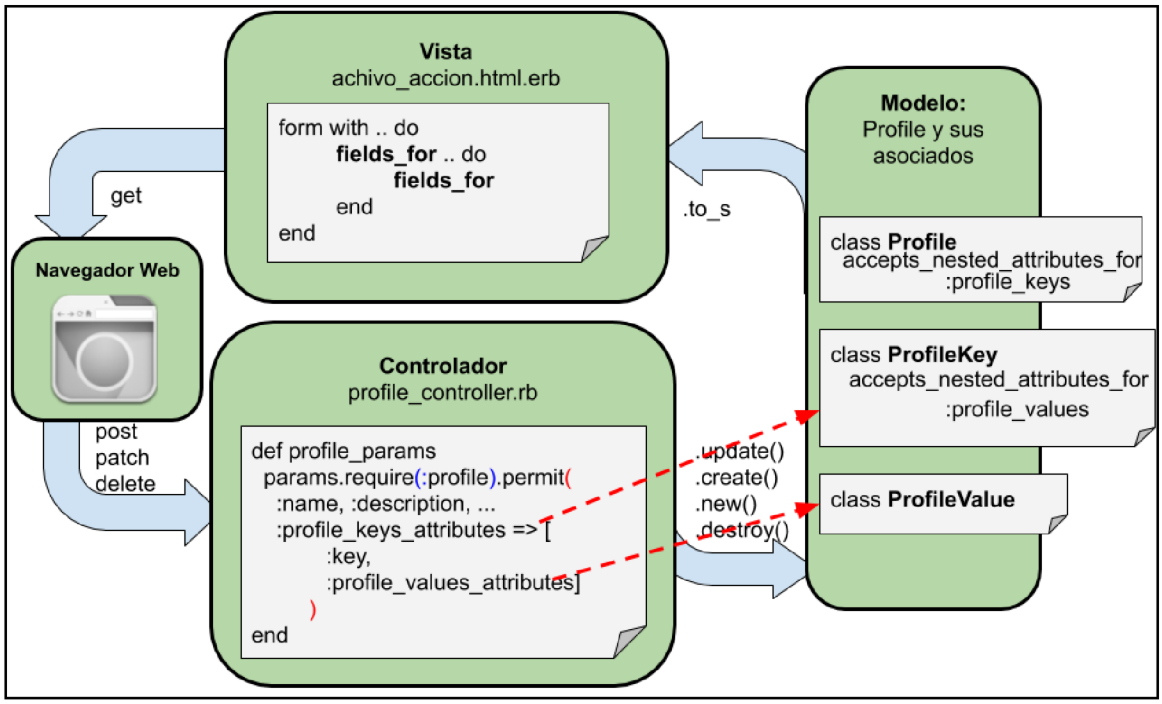

Imagen 16: *Implementación de nested_attributes vista desde el diagrama en bloques de la arquitectura Modelo - Vista - Controlador (MVC)* (imagen propia)

Es necesario implementar los permisos a los campos de las subclases de objetos desde el controlador, por tratarse de un controlador para un modelo, en este caso, Profile, al que se le agrega el permiso al campo key del modelo *ProfileKey* y el permiso al campo value de *ProfileValue*, como se observa en el bloque controlador de la Imagen 16.

4.2.1 Implementación dinámica del perfil e inconvenientes sometidos a revisión

La implementación de perfiles dinamizados con clave - valor (*ProfileKey - ProfileValue*) es exclusiva de GIDAPPF, ya que soluciona características a los problemas específicos que se reflejaron durante el análisis funcional y el diseño. Sin embargo, la etapa de desarrollo continúa en fase de diseño y no es posible avanzar hasta resolver la validación del objeto *Profile*. Es decir, el objeto *Profile* se considera como tal si se pueden validar todos los

atributos, entre ellos, el nuevo atributo que es un conjunto de objetos *ProfileKeys*. A su vez, *ProfileKey* está compuesto entre sus atributos por un conjunto de objetos *ProfileValues*.

Con el diseño del apartado anterior, los campos generados por *nested_attributes*, pasados el conjunto de objetos que componen a *ProfileKey*, se editan con campos de texto con todo el conjunto de caracteres. Esto genera un problema porque, de esta manera, el usuario puede tipear números en campos "Apellido" o letras en campos fechas.

4.2.2 Validación del lado del cliente

La solución definitiva para los perfiles dinamizados con clave - valor fue la incorporación del concepto de validaciones del lado del cliente para los *nested_attibutes*. Es un concepto en el cual la vista utiliza una segunda interfaz de conexión en segundo plano con la base de datos, para lograr un cruce de información con los datos válidos de la base de datos y, así, validar en tiempo real los datos tipeados por el usuario, antes de guardar el perfil utilizando la compatibilidad de *rails* con el lenguaje de programación *Javascript*.

Este tipo de validación es útil si se implementan diversos tipos de validaciones según el tipo de *ProfileKey* que desea el usuario. Entonces, es necesario modificar nuevamente el diseño de la base de datos a fin de categorizar cada tipo de *ProfileKey* con una referencia a la entidad *ClientSideValidator*.

Esta implementación, mostrada en el DER de la Imagen 17, también es extensible para rediseñar los contenidos *Input* de *Documents* que fueron dinamizados, asimismo, con clave - valor en las entidades asociadas *InfoKey* e *InfoValue* como se vio, previamente, en la Imagen 14.

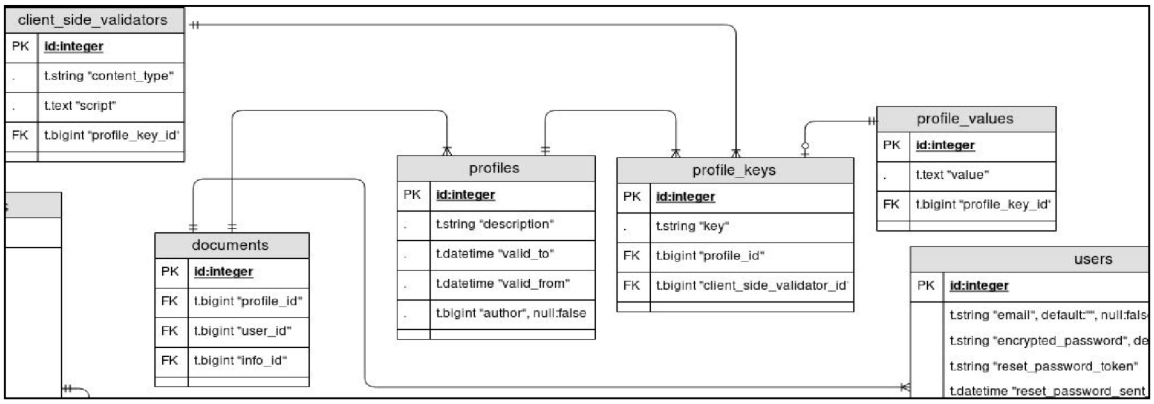

Imagen 17: *Implementación de ProfileKey categorizado según su ClientSideValidator desde el punto de vista del rediseño de la base de datos* (imagen propia)

La integración desde el bloque de la vista de la arquitectura MVC se produce utilizando el método *render* de *Rails* que se utiliza en los códigos fuente de las vistas para generar formularios parciales desde un segundo archivo de código fuente de vista, con la diferencia que el nombre del archivo debe comenzar con guión bajo.

Archivo: GIDAPPF/gidappf/app/views/profiles/_form.html.erb

```
1  <%= form_with(model: profile, local: true) do |form| %>
2    ...
3          <%= render 'fields_for_profile_keys', profile: profile, form: form  %>
4    <div class="actions">
5          <%= form.submit %>
6    </div>
7  <% end %>
```

Imagen 18: *Implementación de la vista de edición de perfiles renderizando la vista de componentes ProfileKeys* (imagen propia)

En la nueva versión de la vista, se delegan los llamados a *fields_for* de los componentes *nested_attributes*, en un nuevo formulario parcial que se visualiza cuando la ejecución pasa por la línea 3 del bloque de código, mostrado en la Imagen 18. De esta forma, luego, se implementa el bloque de código fuente en forma cíclica para que puedan

visualizarse los campos de *ProfileKey* en el archivo llamado "*_fields_for_profile_keys.html.erb*".

Archivo: GIDAPPF/gidappf/app/views/profiles/_fields_for_profile_keys.html.erb

```
1 <% unless profile.errors.any? then %>
2   <%= form.fields_for :profile_keys, profile.profile_keys do |nested| %>
3     <%= nested.hidden_field :key, :value =>
Profile.find(profile.id).profile_keys[nested.index].key %>
4 <%=nested.fields_for:profile_values,
  profile.profile_keys[nested.index].profile_values do |nested2|%>
5         <%= render "fields_for_profile_values", form: nested2, profile_id: profile.id,
    nested_index: nested.index %>
6     <% end %>
7   <% end %>
8 <% end %>
```

Imagen 19: *Bloque de código fuente de vista parcial fields_for_profile_keys* (imagen propia)

En la Imagen 19, específicamente, en la línea 2, se resalta el ciclo de *fields_ for* para cada componente visual de cada *ProfileKey* asociado con el perfil; este termina en la línea 7. Sucesivamente, en la línea 5, se implementa el renderizado al parcial que se encarga de los *profile_values* vinculados a cada *ProfileKey,* en otro bloque de código fuente, para poder ser reutilizado todas las veces que sea necesario, pero, principalmente, este se vincula al código fuente de la validación del lado del cliente. La Imagen 20 muestra el bloque de código fuente de la vista parcial que integra el llamado a *ClientSideValidator.script* en la línea 6, mediante el cual se vincula al método de validación del campo a través de código javascript que se renderize según el archivo de la vista parcial que lo indique script.

La validación Javascript actúa con el evento *onfocusout*, inmediatamente después de completar el campo, como se muestra en la línea 4 de la Imagen 20. Como cada subobjeto *profile_key* tendrá asignado una referencia categórica a su *ClientSideValidator* adecuado al contenido del campo, entonces, la entidad *Profile* es válida.

Archivo: GIDAPPF/gidappf/app/views/profiles/_fields_for_profile_values.html.erb

```
1  <div class="field">
2    <%= form.label Profile.find(profile_id).profile_keys[nested_index].key %>
3      <% if
   Profile.find(profile_id).profile_keys[nested_index].client_side_validator_id >
     ClientSideValidator.find_by(content_type: 'GIDAPPF words').id then %>
4        <%= form.text_field :value, onfocusout:
     ClientSideValidator.find(Profile.find(profile_id).
       profile_keys[nested_index].client_side_validator_id).script %>
5      <% else %>
6        <%= render  ClientSideValidator.find(Profile.find(profile_id).
         profile_keys[nested_index].client_side_validator_id).script,
            valuable_gidappf_field: form %>
7    <% end %>
8  </div>
```

Imagen 20: *Bloque de código fuente de vista parcial fields_for_profile_values* (imagen propia)

Los valores iniciales insertados en seeds.rb, para que *ClientSideValidator* funcione, deben proporcionar rutas válidas de vistas parciales existentes, y, como norma, se indica en el atributo *content_type* un prefijo GIDAPPF y, luego, se especifica una aclaración breve del tipo al que será comprobado el contenido, como se muestra en la Imagen 21.

Las vistas parciales de *ClientSideValidator* son accedidas tanto desde los controladores de *ProfileKey* como de InfoKey, produciendo *ProfileValues* e *InfoValues* válidos. Las consultas en segundo plano de las validaciones se producen directamente en la validación de onfocusout, como ocurre, por ejemplo, en la validación de materias a las que un docente puede indicar en su perfil.

Los códigos fuente de los controladores de *ClientSideValidator* se diseñaron para que se asigne, en una variable *target_reg_exp_strategy,* el código fuente de validación en lenguaje de programación *Javascript*, compatible con el *framework* JQuery, tal como se muestra en la Imagen 22, en las líneas 1 a 9.

Archivo: GIDAPPF/gidappf/db/seeds.rb

```
1  ClientSideValidator.destroy_all
2  ClientSideValidator.create!([{
3    content_type: 'GIDAPPF read only',#1
4    script: 'client_side_validators/gidappf_read_onlys'#REQUERIDO POR SISTEMA
5  },{
6    content_type: "GIDAPPF alphanumerics",#2
7    script: 'client_side_validators/gidappf_alphanumerics'
8  },{
9    content_type: "GIDAPPF checks",#3
10   script: 'client_side_validators/gidappf_checks'
11 },{
12   content_type: "GIDAPPF dates",#4
13   script: 'client_side_validators/gidappf_dates'
14 },{
15   content_type: "GIDAPPF matters",#5
16   script: 'client_side_validators/gidappf_matters'
17 },{
18   content_type: "GIDAPPF numbers",#6
19   script: 'client_side_validators/gidappf_numbers'
20 },{
21   content_type: "GIDAPPF trayects",#7
22   script: 'client_side_validators/gidappf_trayects'
23 },{
24   content_type: "GIDAPPF words",#8
25   script: 'client_side_validators/gidappf_words'
26 },{
27   content_type: "GIDAPPF validator example",#9
28   script: "$(document).ready(function() {
29     if($(event.target).val().match(/^[a-zA-Z\\s]+$/) == null) {
30         $(event.target).val('');
31         $( \"<p class='validation-error'>Error detected.</p>\").
32               appendTo($(event.target).parent());
33         }
34   });"}
```

Imagen 21: *Bloque de código fuente de valores iniciales referidos a la entidad ClientSideValidator en la base de datos* (imagen propia)

En la línea 6, es destacable la manera de insertar lenguaje *Ruby* gracias a su compatibilidad. Luego, en la línea 10, es posible implementar un elemento de selección gráfica, pero, de todas maneras, seguirá funcionando la validación en paralelo. Es

importante observar que, en la línea 11, se realiza la consulta en segundo plano, para establecer cuáles son los nombres de las materias y sus identificadores, entonces, el selector gráfico se ejecuta correctamente.

Archivo: GIDAPPF/gidappf/app/views/client_side_validators/_gidappf_matters.html.erb

```
1  <% target_reg_exp_strategy = "$(document).ready(function() {
2    str=String($(event.target).val());
3    rexp=String($(event.target).val().match(/[\\w\\W]+/gim));
4    if( str !== rexp ) {
5          $(event.target).val('');
6          $( \"<p class='validation-error'>
         #{t('body.gidappf_entity.client_side_validator.error.matter_eror')}
    </p>\" ).appendTo($(event.target).parent());
7          }
8    });"
9  %>
10 <%= valuable_gidappf_field.select :value,
11       Matter.where(enable: true).collect { |m| [m.name, m.id ] },
12         include_blank: true, onfocusout: target_reg_exp_strategy
13 %>
```

Imagen 22: *Bloque de código fuente de un ClientSideValidator* (imagen propia)

Cada uno de los distintos tipos de ClientSideValidator funcionan en paralelo en las vistas que los requieran. En la Imagen 23, se aprecia el resultado.

La implementación de perfiles dinamizados con clave - valor (*ProfileKey - ProfileValue*), exclusiva de GIDAPPF, en combinación con la implementación de *ClientSideValidators,* soluciona características a los problemas específicos que se reflejaron durante el análisis funcional y el diseño, y también es una propuesta innovadora y original de este sistema. Se deja pendiente un rediseño de estas funcionalidades que podrían ser una solución genérica a sistemas *Rails* que necesiten manipular entidades con *nested_attributes* en forma gráfica con distintos tipos de datos en sus campos, es decir, la producción de una Gema de *Ruby* como las que se describen en el apartado 2 (cf. “6. Conclusiones”).

Imagen 23: *Vista de edición del perfil dinamizado con clave - valor funcionando y validando con múltiples ClientSideValidator en la misma vista* (Imagen propia)

4.2.3 Cambios de comisiones dinámicamente controladas

La necesidad de dar soporte al hecho de que, en cualquier momento del curso de las clases, puedan realizarse cambios en la lista de alumnos, fue un obstáculo a la hora de implementar el circuito de control de asistencia. El problema es que los documentos están vinculados a los usuarios y, para registrar cambios en documentos de listas de

estudiantes, es muy difícil realizar su trazabilidad para asegurar los datos cambiados, sobre todo, si se realizan múltiples cambios al mismo tiempo. Se complica aún más si el que se cambia de comisión es el docente.

La solución que se encontró fue la de no utilizar lista de estudiantes por comisión. Se cambió de paradigma, sabiendo que cada usuario tiene una referencia a la entidad *Usercommissionrole*, entonces, es posible realizar la consulta de esas referencias en cada horario, al inicio de cada clase. De esa manera, el procedimiento es seguro, previniendo el choque de datos y los sabotajes. La desventaja es que se necesita una herramienta que permita crear respuestas a eventos temporales.

Por lo antedicho, se resuelve usar la sincronización temporal, utilizando la herramienta característica de *Rails 5* llamada *Active Jobs* y el servidor *Sidekiq*, ambos integrados por defecto al *framework Rails*, pero que funcionan como servicios de ejecución independientes para crear trabajos en segundo plano. El diagrama de la interacción se presenta en la Imagen 24.

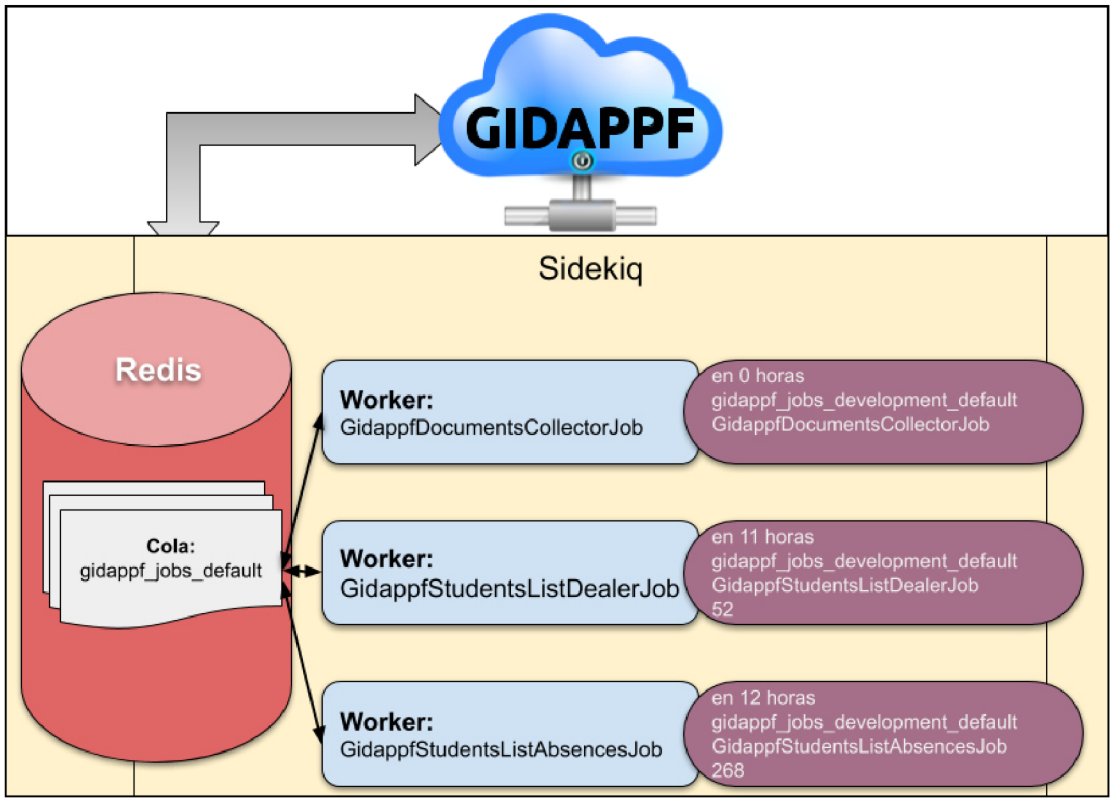

Imagen 24: *Diagrama de la integración de trabajos en segundo plano con GIDAPPF*
(imagen propia)

La implementación de colas de trabajo *ActiveJobs* se utiliza para generar algoritmos que se ejecutan en hilos de ejecución por separado; de esta forma, es posible implementar trabajadores que ejecutan una tarea según la condición que debe cumplirse para iniciar la realización.

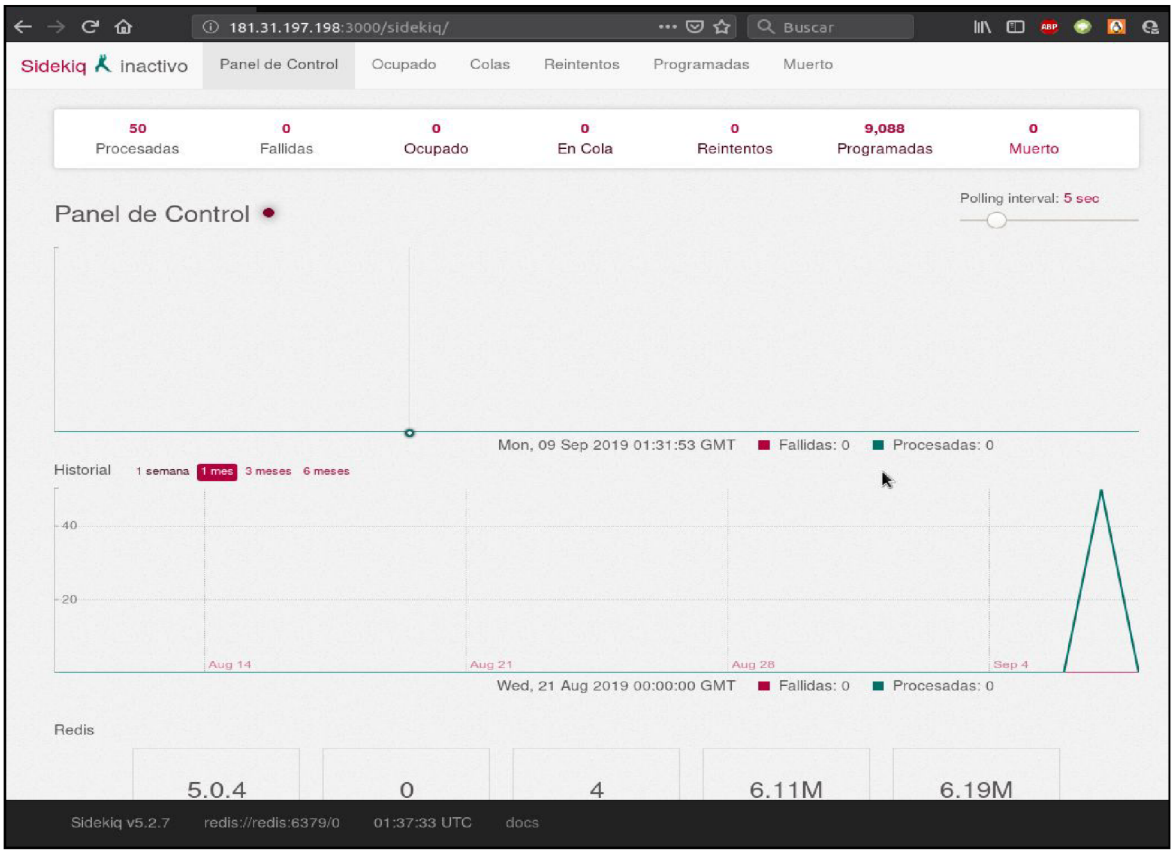

Imagen 25: *Histograma de tareas procesadas que provee Sidekiq* (imagen propia)

En la Imagen 25, se muestra una interfaz de *Sidekiq* que centraliza el histograma de las ejecuciones de *ActiveJobs*. La solapa “Fallidas” resulta muy útil para el desarrollador, ya que, cuando ocurre una excepción, esta vista muestra la trazabilidad de dicha excepción. El número de tareas programadas también es un indicador de control para verificar que el algoritmo de encolamiento de tareas funciona correctamente. Esta interfaz de sidekiq no está incluida en herramientas de administración para GIDAPPF, es sólo para la utilización en modo de desarrollador.

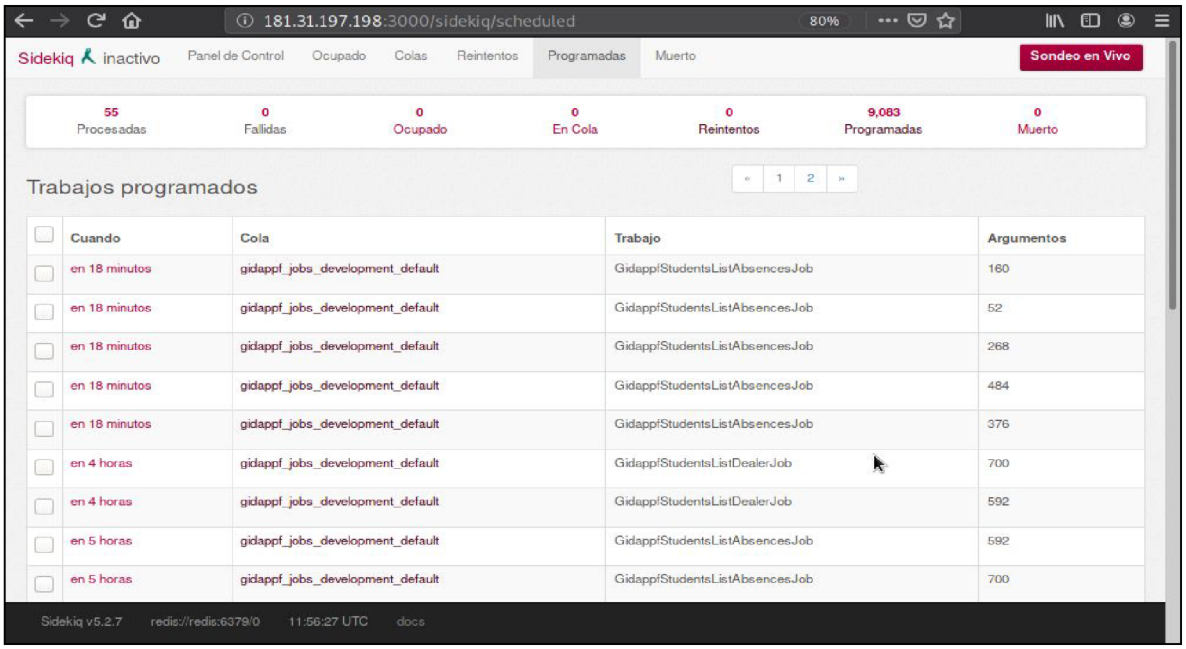

Imagen 26: *Visualización de la cola de ActiveJobs de Sidekiq* (imagen propia)

Otra vista útil de *Sidekiq* es la de tareas programadas, mostrada en la Imagen 26. Allí pueden verse los detalles de la cola y los distintos *Workers* pendientes. El tiempo programado es actualizado con cada refresco del navegador web. El sondeo en vivo produce las actualizaciones automáticas de los datos, pero interfiere con la sesión de *rails*.

```
def self.to_queue_sidekiq(c, t)
  time_until_in = Time.new(c.year.to_i,c.month.to_i,c.day.to_i,t.from_hour.to_i,
     t.from_min.to_i,00).in_time_zone(ENV['TZ'])
  time_until_out = Time.new(c.year.to_i,c.month.to_i,c.day.to_i,t.to_hour.to_i,
     t.to_min.to_i, 00).in_time_zone(ENV['TZ'])
  GidappfStudentsListDealerJob.set(wait_until: time_until_in).
     perform_later(time_sheet_hour.id)
  GidappfStudentsListAbsencesJob.set(wait_until:time_until_out).
     perform_later(time_sheet_hour.id)
end
```

Imagen 27: *Bloque de código fuente de GidappfTemplatesTools, definición del método que inserta trabajos al servidor Sidekiq* (imagen propia)

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

La forma de insertar nuevos *ActiveJobs* a la cola de *Sidekiq*, es tal y como se implementó en el bloque de código, que aparece en la Imagen 27. Esta también es una implementación exclusiva de GIDAPPF.
En dicha imagen, se explica brevemente el método de los *Workers* resaltados; con la sentencia *wait_until,* se indica el tiempo del sincronismo como condición de realización. Luego, la sentencia *perform_later* es la que inserta el trabajo en la cola de *Sidekiq*.

4.2.4 Diagrama entidad relación definitivo

Las correcciones del diseño implementadas fueron modificando el Diagrama Entidad - Relación que se presentó en el apartado 4. Es muy importante para futuras implementaciones mantenerlo actualizado. En la Imagen 28, se muestra la última versión con las implementaciones de perfiles y entradas dinamizadas por clave - valor y sus validaciones del lado del cliente implementadas.

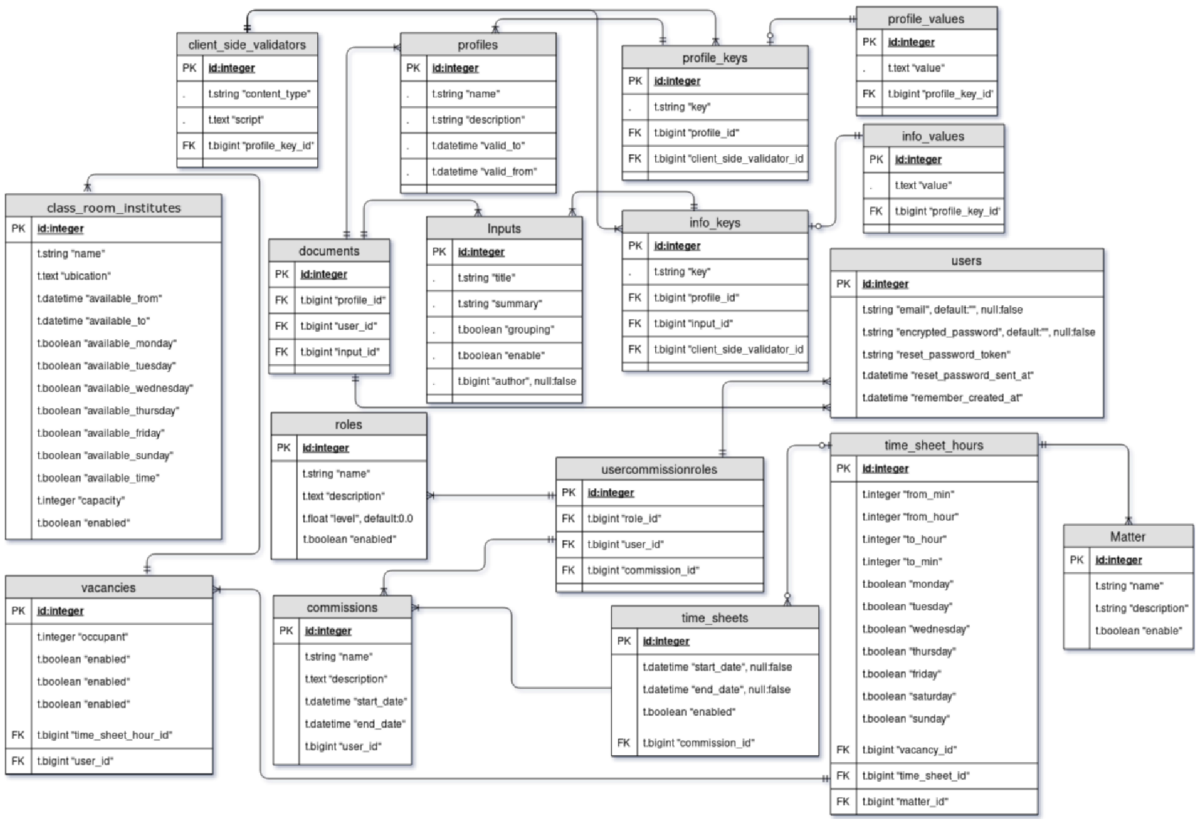

Imagen 28: *DER actualizado al momento de finalizar la versión de desarrollo de GIDAPPF* (imagen propia)

4.3 Pruebas de aceptación

La primera prueba de aceptación se realizó en una reunión con coordinadores del Plan FinEs, Profesores Rubén Romero y Christian Pidalá, en la sede de FinEs. Se compartió una primer versión del manual de usuario GIDAPPF y se realizó una comprobación de todas las acciones. Las primeras devoluciones resultaron en el acuerdo de que el desarrollo es adecuado para las necesidades planteadas y que se podría organizar una segunda prueba, pero que se necesitaba que, en la vista de cada perfil de estudiante, se tuviera acceso a la historia académica.

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

El desarrollador plantea la posibilidad de probar GIDAPPF instalado en un servidor público de prueba, para que los usuarios interactúen directamente y propongan mejoras. Los coordinadores estuvieron de acuerdo con la propuesta.
La versión preliminar es la que se presenta como desarrollo obtenido del resultado de la actual Práctica Profesional Supervisada número 12.

5. Documentación generada

Como se adelantó, en este desarrollo, resultó importante generar documentación adecuada según el destinatario, ya que los informes académicos representan el detalle de la Práctica Profesional Supervisada y son uno de los objetivos fundamentales de esta.
Cabe destacar que los reportes de minutas de reunión, diagramas y tutoriales propios de cada módulo desarrollado se comparten con los diferentes tutores y coordinadores, mencionados previamente.
El intercambio de emails con los miembros del equipo de desarrollo se usó para distribuir las minutas de reuniones de avance, acompañadas por el reporte de la minuta.
Asimismo, el código fuente se publicó en el repositorio cuyo enlace es https://github.com/unajpps2018asc/GIDAPPF, en el cual, los cambios y la autodocumentación se producen al momento de un aporte individual y verificando la integridad de código fuente mediante los testeos. La *Wiki* publicada en el repositorio contiene información destinada a los programadores y al personal de soporte de sistemas.

6. Conclusiones y trabajos futuros

GIDAPPF es un software libre que logró obtener su ciclo de producción inicial; actualmente se encuentra en una etapa en la cual se han dejado de añadir nuevas características para comenzar una nueva etapa de pruebas y aceptación por parte de la Dirección De Informática de la UNAJ, a cargo de Gustavo Alejandro Pilla, que evalúa el código fuente para la integración con los servidores. Una primera devolución de la

gerencia indica que es necesario realizar el arranque de *ActiveJobs* y la configuración de sincronismo del tiempo dentro del código fuente de la creación del contenedor *Docker*, y no como un comando concatenado de *linux*, como se realiza actualmente. Es una tarea de configuración avanzada e indica que el resto de las dependencias fueron integradas correctamente. Es decir, GIDAPPF comienza con la fase de lanzamiento de la primer versión.

El sistema cumple con las expectativas básicas de los coordinadores de FinEs y podría utilizarse de inmediato en el entorno de la producción; todas las tecnologías que se integraron son compatibles a la perfección, garantizando el constante mejoramiento. Además, la característica de naturalidad del lenguaje de programación *Ruby* (referenciado en el apartado 2.1.), facilita el reconocimiento de patrones repetitivos para resolverlos de manera creativa y eficiente.

Sin embargo, existen muchas reglas elegidas arbitrariamente por el desarrollador que están pendientes de confirmación, por ejemplo, la cantidad máxima de comisiones por temporada que pueden ser asignadas a los docentes; no obstante, estos son detalles muy fáciles de implementar y que pueden corregirse a demanda en cuestión de horas.

Por otra parte, las implementaciones de los tests se realizaron en forma insuficiente y solo son testeadas las acciones del controlador en forma individual, pese a que *Rails* provee, además, marcos de testeo de modelo, de vista y de integración que no están siendo aprovechadas. La futura implementación de estos incrementará la calidad del software al tener más parámetros de control.

Existe un malfuncionamiento conocido, o *bug*, como se lo indica en el ámbito del desarrollo de *software*, pero es necesario someter al sistema a una auditoría externa, en el caso ideal, para saber si existen más y obtener una visión objetiva de estos aspectos, o bien esperar a la devolución de los reclamos de los usuarios.

Dicho *bug* se evidencia en usuarios que, al conectarse con credenciales provisorias, son redireccionados por GIDAPPF para editar sus credenciales. La implementación actual no

desconecta al usuario luego de actualizar las credenciales y tampoco le avisa que debe desconectarse. Asimismo, el sistema vuelve a detectar su nueva clave como provisoria si el usuario reingresa al sistema luego de más 90 segundos, sin notificar dicha consecuencia. Igualmente, en cualquier momento, el usuario puede entrar al sistema de recuperación de contraseña mediante el link *Forgot your password?*, como se muestra en la Imagen 29.

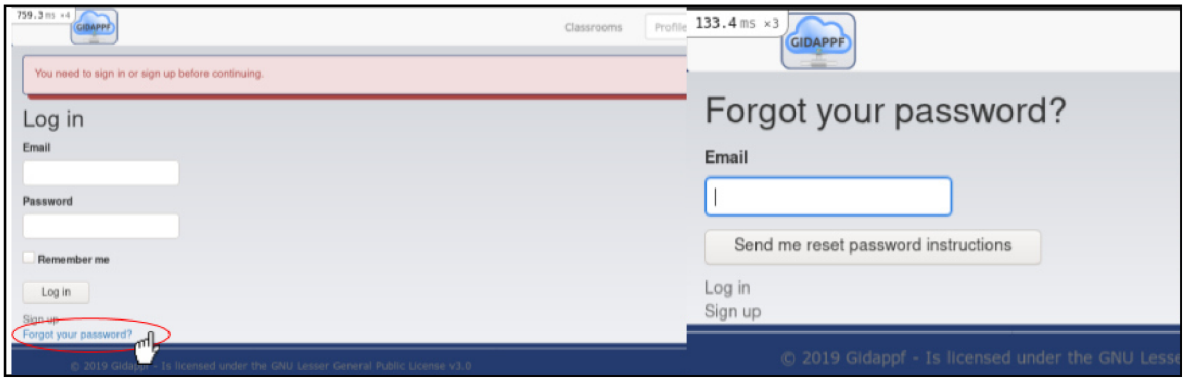

Imagen 29: *Alternativa que soluciona cualquier problema con las credenciales provisorias* (imagen propia)

La planificación del proyecto fue lograda y ampliada, pero en plazos muy distintos a los calculados, puesto que no se reconoció una referencia similar al caso para estimar los tiempos al inicio de la planificación. El incumplimiento con los plazos es una presión que se debe aprender a manejar a nivel profesional, ya que estos se pagan con la disminución de recursos. En el caso de GIDAPPF, durante la entrega del alcance del proyecto ya se había aclarado a los coordinadores de FinEs que el modelo de desarrollo utilizado provee de un software que evolucionará de acuerdo a los sucesivos lanzamientos de nuevas versiones.

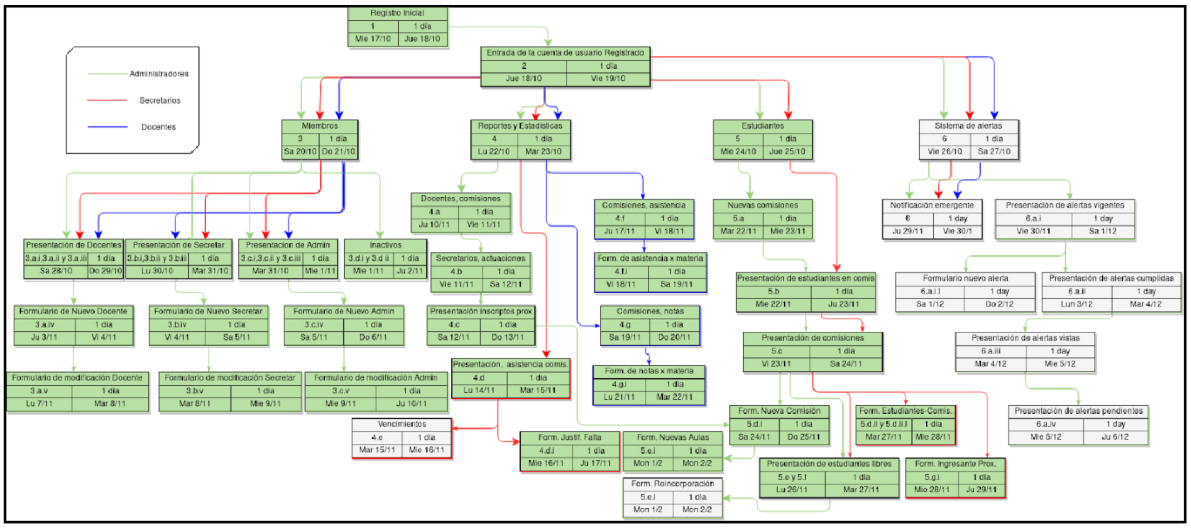

Imagen 30: *Referencia gráfica del progreso de GIDAPPF versión 1, basado en el desglose inicial producido en la etapa del análisis funcional* (imagen propia)

Desde el punto de vista de la primer versión entregada, se cumplió con un 78% del objetivo según lo que se había propuesto durante la etapa del análisis funcional, dejando pendiente el desarrollo del sistema de alertas configurables, como se observa en la Imagen 30 en donde se resaltan en verde los bloques que representan las implementaciones completas y se dejaron los bloques blancos para representar a los módulos del análisis funcional que no entraron en el desarrollo de GIDAPPF versión 1. Es un cumplimiento que está dentro de los parámetros de satisfacción de los coordinadores de FinEs, ya que los módulos pendientes estuvieron pensados como características complementarias desde el comienzo.

El rol del practicante también necesita ser analizado en esta conclusión, para tener en cuenta todos los aspectos y tiene que ver con el producto final. Los perfiles de cargos relacionado con tecnologías en el desarrollo de software no son colegiados en Argentina, pero la institución con la mejor referencia es la Cámara de la Industria Argentina del Software, CESSI (Busso, 1982). Está institución publica y actualiza periódicamente las características y competencias necesarias para cada tarea a desarrollar en siete distintos

perfiles ocupacionales de industria TI (Tecnologías de la Información). Dichos perfiles ocupacionales publicados son los siguientes:

- Desarrollo de software: esta categoría describe los diferentes perfiles de programación que intervienen en el ciclo de vida de desarrollo de software.
- Análisis: quienes desarrollan esta actividad tienen la responsabilidad de tener un fuerte conocimiento del producto o negocio, tanto para describir requerimientos o la historia de usuarios, así como para analizar datos o desarrollar análisis predictivo.
- Diseño: esta categoría agrupa los perfiles altamente creativos. En su mayoría, todos poseen buen manejo de herramientas gráficas, efectos y animaciones.
- Calidad de software: estos perfiles buscan garantizar la calidad del trabajo en el área de sistemas, tanto en la organización de una metodología y estandarización de la empresa utilizando las mejores prácticas, como así también en los procesos de testeo y depuración de errores, antes de implementar una nueva aplicación o cambios en una ya existente.
- Soporte de infraestructura: estos perfiles se responsabilizan de garantizar la estabilidad, seguridad y el buen funcionamiento de la infraestructura de las terminales de trabajo, servidores y redes de la compañía, asistiendo, por una parte, a los usuarios en las dificultades técnicas, y, por otra parte, a los programadores, en la comunicación, colaboración e integración entre desarrollos y el sector de IT, siguiendo los principios de DevOps.
- Implementadores/Soporte a usuario: estos son responsables de las implementaciones y personalizaciones de software de Gestión.
- Comunicación online: estos perfiles están orientados al desarrollo de la comunicación digital de una marca o producto. Agrupados en diferentes especialidades como posicionamiento en buscadores, redes sociales y contenidos, todos buscan el mayor alcance y el público exacto al que se desea comunicar.

Estos perfiles ocupacionales son especializaciones que se requieren en organizaciones dedicadas al desarrollo de software y cada una de estos se dividen en categorías con distintas responsabilidades.

Las habilidades que fueron aplicadas para desarrollo pueden ser identificadas en las competencias de múltiples categorías de los perfiles ocupacionales, es decir, se puede comprobar que, en el ámbito empresarial, las tareas se dividen en seis profesionales en nivel semi senior de experiencia en casi todos los casos, salvo en el caso de la comunicación online.

No existe una categoría indicada en CESSI en la cual un solo puesto sea responsable de todas las especializaciones, pero, internacionalmente, en las búsquedas laborales, hay descripciones de competencias similares en puestos de *Full Stack Developer*. Dicho puesto es caracterizado por la amplia experiencia en las competencias que lo componen, mayor a cinco años.

Al desarrollo de GIDAPPF, se le dió prioridad al aspecto de base tecnológica desempeñado en las tareas de *back end*, lo que significa que, de esta manera, se asegura la funcionalidad, las posibles extensiones funcionales y las probables integraciones tecnológicas. Por una cuestión de tiempo, no se investigó sobre los mejores estilos visuales para cada acción y tampoco se realizó un diseño gráfico a medida del sistema, sólo se desarrollaron las tareas básicas de *front end*, o implementaciones del aspecto visual.

El trabajo a futuro está claramente direccionado en la corrección del *bug*, la implementación de los módulos pendientes, y la categorización visual del historial académico accedido desde el perfil de cada estudiante. De acuerdo con este lineamiento, la siguiente será una versión intermedia.

En la versión 2 del sistema aquí desarrollado, se planifica la implementación de una aplicación móvil que soporte las tareas administrativas del docente en modo en línea y en modo sin conexión, implementando una estrategia de sincronización adecuada. También

es necesario integrar un sistema de recopilación de errores para documentar los reclamos de los usuarios y recapitular las soluciones a los problemas frecuentes.

REFLEXIÓN SOBRE LA PRÁCTICA PROFESIONAL SUPERVISADA COMO ESPACIO DE FORMACIÓN

El espacio de formación de la Práctica Profesional Supervisada es una oportunidad única para promocionar el profesionalismo individual del estudiante. Una reflexión adecuada se manifiesta en vinculación con el desarrollo profesional, pero también hace referencia al desarrollo personal, a lo largo de dicha práctica.

A nivel personal, la formación de la Práctica Profesional Supervisada tomó un protagonismo extra con respecto a otras materias. Durante la carrera, siempre pude repartir el tiempo de cada cursada entre dos o tres materias, pero, en este caso, el desarrollo implicó a la necesidad de establecer certezas individuales. Algunas de las incertidumbres que me planteó la PPS fueron las siguientes:

- Si me toca seguir desarrollando este tipo de sistemas una vez recibido ¿estaría satisfecho con esas tareas?
- Si tuviera que delegar el desarrollo a alguien más ¿sería una capacitación comprensible?
- ¿Sería más eficiente dividir o armar un grupo con una sinergia general e interdisciplinaria?

Con respecto a las oportunidades que representa en el aspecto profesional, la PPS generó un espacio de trabajo interdisciplinario, pero con mucha libertad de producción. La envergadura del proyecto era la apropiada para desarrollarlo desde la etapa de análisis funcional. El objetivo profesional de la PPS era que GIDAPPF funcionara y fuera útil para la comunidad FinEs. Este objetivo pragmático y de

aplicación inmediata también resultó un incentivo para el trabajo, ya que, de su eficacia, dependía, asimismo, el funcionamiento conjunto de diversos sectores de dicha comunidad.

El desarrollo fue establecido teniendo en cuenta que, si el sistema se expande, y el equipo de trabajo tiene que incorporar personal con distintas especialidades, se podría dividir fácilmente en distintas áreas. De acuerdo con nuestra observación y análisis, estás podrían ser las áreas de *DBA*, Integración, *Testing*, *Front - end*, *Back - end*, Análisis funcional y Aplicación Móvil. Además, se podría desprender la rama del desarrollo que se dedica a la implementación del código fuente de la gema *Ruby*, que generalice la interacción de los perfiles dinámicamente implementados con validación del lado del cliente como se propuso en el apartado 4.2.2.

El apoyo y la colaboración otorgados por los tutores y los integrantes de FinEs fueron esenciales para establecer la relación similar al entorno de producción de sistemas de una organización.

Para concluir, la envergadura del desarrollo me permitió sentir la responsabilidad del trabajo en distintas áreas de especializaciones en el ámbito empresarial.

7. Bibliografía

Bourque, Pierre y Fairley, Richard. *Guide to the Software Engineering Body of Knowledge*. Washington D. C.: IEEE Computer Society, 2014.

Busso, Mauro (1982). *Cámara de la Industria Argentina del Software*. Disponible en: <http://www.cessi.org.ar>. [Consulta: 26/09/2019].

Fernández, Obie. *The Rails 5 Way*. Boston: Addison-Wesley, 2017.

Fowler, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, 2003.

Hansson Heinemeier, David (2005). *Imagine what you could build if you learned Ruby on Rails...* Disponible en: <http://www.rubyonrails.org/ >. [Consulta: 01/03/2019].

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional:

Hykes, Salomón. (2013). *Shape your digital future at DockerCon 2019*. Disponible en: <https://www.docker.com/> [Consulta: 01/03/2019].

Matsumoto, Yukihiro (Miércoles, 16 Feb 2000 08:50:42). "[ruby-talk:01407] Re: Say Hi". En: *Lista de correo Ruby - talk, Archivos (1407)*, 440400 páginas. Disponible en: http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/1407. [Consulta: 19/09/2019].

Matsumoto, Yukihiro. *Ruby in a nutshell*. Sebastopol: O'Reilly, 2001.

Pressman, Roger. *Ingeniería del software. Un enfoque práctico*. México D.F.: McGraw-Hill, 2010.

Silberschatz, Abraham. *Fundamentos de bases de datos*. México D.F.: Mc Graw Hill, 1999.

Sommerville, Ian. *Ingeniería del software*. Madrid: Addison-Wesley, 2005.

Stonebraker, Michael (1996). *PostgreSQL: The World's Most Advanced Open Source Relational Database*. Disponible en: <https://www.postgresql.org/>. [Consulta: 01/03/2019].

Turnbull, Jammes (2016). *The Docker Book, Containerization is the new virtualization*. Disponible en: <https://dockerbook.com>. [Consulta: 26/02/2019].

Firma Estudiante:	Firma Docente Supervisor:	Firma docente tutor TAPTA:	Firma tutor Organizacional:	Firma tutor Organizacional: