UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistema de automatización para el llenado de la MIR-DICyG

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A N

López Santibáñez Jiménez Luis Gerardo

Moreno Guerra Marco Antonio

DIRECTORA DE TESIS

M.I. Tanya Itzel Arteaga Ricci

Ciudad Universitaria, Cd. Mx., 2021

TABLA DE CONTENIDO

RESUMEN

El proceso de la Matriz de Indicadores para Resultados (MIR), de la Facultad de Ingeniería, es vital para justificar el presupuesto que le brinda la Universidad Nacional Autónoma de México (UNAM) y que sea posible continuar con las actividades académicas y de investigación.

Este proceso es delegado a cada División de la Facultad, confiando en que cumplirán con este trámite en tiempo y forma. Sin embargo, al ser un proceso administrativo que recaba información y la resume, surgen problemas al realizarse de manera manual; de igual forma, la falta de formatos estándares, dificultades para la coordinación y correcciones manuales, finalmente, se traducen en atrasos para generar los resúmenes de cada trimestre.

Estos problemas pueden solucionarse haciendo uso de las herramientas tecnológicas que se tienen actualmente. Este proyecto de tesis plantea una solución a través de un sistema web para automatizar el proceso de solicitud, recabación, integración y generación de formatos en la División de Ingenierías Civil y Geomática, permitiéndole cumplir en tiempo y forma con este trámite ante la Facultad de Ingeniería.

CAPÍTULO 1. ANTECEDENTES DEL PROYECTO

1.1. Antecedentes de la Secretaría Técnica

El proceso de la Matriz de Indicadores para Resultados (MIR) es un proceso establecido en la Universidad Nacional Autónoma de México (UNAM) derivado de una auditoría que se realiza por parte del Gobierno Federal de los Estados Unidos Mexicanos a esta Universidad desde el 2013.

El objetivo de esta auditoría por parte de la Federación es medir la productividad que tiene la UNAM, de manera que se justifique y demuestre el aprovechamiento de los recursos asignados por parte del Erario Público para el desempeño de sus actividades académicas y científicas.

El presupuesto proporcionado a la Universidad, por parte del Gobierno Federal, es de alta importancia para el cumplimiento de sus actividades, debido a esto es que el proceso de la MIR ha cobrado mucha importancia, por lo que para garantizar que este proceso sea realizado de manera correcta se ha delegado esta actividad a cada Facultad, y éstas a su vez a sus respectivas Divisiones.

Dentro de las Divisiones de la Facultad de Ingeniería, se encuentra la División de Ingenierías Civil y Geomática (DICyG), sobre la que se enfocará el análisis del proceso de la MIR. La razón de desarrollarlo para esta División es que se trata de una propuesta por parte de la Jefa de Cómputo de la DICyG, la M. I. Tanya Itzel Arteaga Ricci, quien ha desempeñado ese puesto por varios años y conoce las complicaciones que ha tenido la DICyG en este proceso.

La DICyG está formada por diversos Departamentos y Secretarías, que se encargan de organizar y llevar a cabo actividades para los estudiantes y académicos de esta División.

Entre las Secretarías que conforman a la DICyG se encuentra la Secretaría Técnica, que se encarga de apoyar a la Jefatura de la División en actividades de índole administrativas y es la responsable de llevar a cabo el proceso de solicitud y recabación de información de actividades de los Departamentos para integrarlas en un reporte que se entrega a la Facultad de Ingeniería, autorizado por el Jefe de la División, para continuar con el proceso de la MIR a nivel Facultad.

La MIR se estableció como un proceso anual en la Universidad, midiendo de manera trimestral las actividades de todos los Departamentos que conforman a las Divisiones. Sus indicadores se centran en el desempeño de las actividades de los Departamentos y su atención a los estudiantes y académicos.

1.2. Procesos

El proceso de la MIR dentro de la DICyG es coordinado por la Secretaría Técnica, que se encarga de crear formatos estandarizados que se entregan a cada uno de los Departamentos para concentrar la información de sus actividades trimestrales.

Cada Departamento de la División concentra las actividades que realizaron en cada trimestre en cada uno de los rubros definidos en la MIR, como: cursos realizados, capacitaciones de profesores, prácticas, concursos y digitalización de documentación, entre otros. Todos los rubros a detalle de la MIR se mencionan en la Sección 3.1.

Adicionalmente a la responsabilidad de reportar sus actividades trimestrales, cada Departamento hace una proyección de actividades esperadas para el próximo año, esto con el fin de justificar el presupuesto que será asignado para el siguiente año.

1.3. Problemática por resolver

Como se mencionó en la Sección 1.2, el proceso de captura y concentración de información de los Departamentos debería seguir un formato y ser entregados en tiempo y forma.

Sin embargo, la mayoría de los departamentos utilizan diferentes formatos a su conveniencia, por ejemplo, generan sus reportes como archivos de Word, Excel o incluso PDF. Por lo que, concentrar la información resulta un proceso complejo, tardado y confuso para la Secretaría Técnica.

Adicionalmente, como se necesita entregar un solo concentrado de información, se debe realizar manualmente el conteo de los reportes entregados y en caso de realizar una corrección por parte de un Departamento se deben volver a contar manualmente los reportes individuales.

Debido a lo anterior, se detectó que el problema a resolver es el proceso de captura de las actividades trimestrales de los Departamentos y la concentración de toda la información en reportes, de manera que se busque proveer una solución integral a través de un sistema web, ya que todas las actividades de este proceso son factibles de solucionar computacionalmente para hacerlas más efectivas y automáticas.

El manejo de un sistema web es capaz de establecer las siguientes soluciones a los problemas identificados:

- Establecer un formato estándar de captura de datos.
- Proporcionar apoyo describiendo el contenido de cada uno de los rubros de la MIR.
- Facilitar el proceso de captura de datos a los Departamentos en aquellos rubros con centenas de actividades a registrar, como en el caso de las prácticas.
- Establecer intervalos en los que sea posible capturar y rendir la información.
- Automatizar el proceso de integración de toda la información de los Departamentos.
- Proporcionar estadísticas del progreso de cada Departamento.
- Comparar las actividades proyectadas con las realizadas.
- Generar los reportes y garantizar su validación o Visto Bueno por los Jefes de Departamentos o el Jefe de la División de manera electrónica.

CAPÍTULO 2. MARCO TEÓRICO

2.1. Ciclo de vida del desarrollo de software

Todo proyecto requiere de una organización y comunicación entre el equipo para establecer objetivos en común, establecer fechas límites, costos, posibles errores o inconvenientes que se puedan presentar y reducirlos, en general, se busca maximizar la calidad del producto. Ante esto, se requiere establecer modelos de desarrollo de software, que son metodologías o procesos seleccionados para desarrollar el proyecto de acuerdo con sus objetivos.

Los modelos de desarrollo de software son un apoyo y una guía que conducen todo el proyecto para mejorar la calidad y todo el proceso de desarrollo, desde su inicio hasta la entrega del producto, que se define como el ciclo de vida del desarrollo de software. Este último es un entorno que describe las diferentes etapas por las que pasa un producto de software durante su desarrollo, así como las actividades que se llevan a cabo en cada una de éstas etapas.

El ciclo de vida del software o *Software Development Life Cycle* (SDLC) es un plan, que establece una línea de trabajo para desarrollar las necesidades del cliente. Debido a su generalidad y funcionalidad, se estableció el estándar internacional ISO/IEC 12207[1] para el SDLC. La finalidad de este estándar es definir todas las actividades que se deben realizar para desarrollar y mantener un producto de software.

El SDLC se compone de seis etapas principales y secuenciales [1] que establece el orden en cómo se debe realizar para garantizar un producto de calidad.

1. Análisis de requerimientos y planeación.
Es la primera y más importante etapa del SDLC, debido a que en este punto miembros expertos del equipo, interactúan con el cliente o su organización para establecer sus necesidades y la lógica de negocio, es decir, cómo operan sus procesos y flujos de información. Esta etapa establece las bases del producto a realizar.

[1] *International Organization for Standardization* (ISO) / *International Electrotechnical Commission* (IEC) 12207 es un estándar sobre *Software life-cycle processes*. Provee un marco de referencia con procesos y actividades para el desarrollo de software. https://ingertec.com/iso-iec-12207/

2. Definición de requerimientos.
Después de haber recabado y analizado los requerimientos, se definen claramente y se documentan. Esta información documentada se conoce como *Software Requirement Specification* (SRS) establece los objetivos a cumplir del producto y debe ser aprobada por el cliente.

3. Diseño de la arquitectura del producto.
En esta etapa se toma como referencia el SRS para establecer la arquitectura del producto. Se elige un enfoque de arquitectura y se documenta en un *Design Document Specification* (DDS) que se revisa por todo el equipo. El enfoque elegido debe basarse en parámetros como: evaluación de riesgos, método de diseño, presupuesto y tiempo.

4. Implementación o desarrollo.
Es en esta etapa donde se inicia el desarrollo o codificación del producto, es decir, se comienza su programación. Los programadores deben seguir las especificaciones del producto.

5. Pruebas.
Una vez que se ha codificado el producto, un grupo de *testers* se dedican a probar la aplicación, para garantizar su funcionamiento correcto o dar una retroalimentación a los programadores en cuanto a ciertas fallas.

6. Despliegue y mantenimiento.
Cuando se han pasado las pruebas de manera exitosa, se despliega o entrega el producto al cliente. Dependiendo del acuerdo con el cliente, se puede brindar soporte del producto por un determinado tiempo o realizar cobros por el mismo.

2.2. Metodologías de desarrollo de software

Con el avance tecnológico de las últimas décadas, se ha pasado de tener aplicaciones de software sencillas, sin interfaz gráfica de usuario y sin cantidades altas de información, a aplicaciones con altos volúmenes de datos, que requieren interfaces gráficas de usuario simples para presentar grandes cantidades de información de manera simple e intuitiva. Por ello, también se ha modificado el proceso de creación de software, pasando de ser pocas personas a equipos que se dediquen a realizar múltiples tareas simultáneamente.

Es en ese momento cuando surgió la necesidad de instaurar procesos o metodologías que establecieran la forma de trabajo de todos los miembros de los equipos y la comunicación entre ellos, con la finalidad de evitar atrasos, trabajos duplicados y maximizar la eficiencia del trabajo de todos.

Las metodologías imponen un proceso disciplinado en cada etapa del ciclo de vida del software para hacerlo más estructurado y eficiente. Lo anterior es posible con planificación y comunicación, asignando diferentes roles y líneas de trabajo.

Inicialmente surgieron metodologías de desarrollo de software orientadas a la planeación, hoy en día conocidas como metodologías tradicionales. Posteriormente, surgieron un tipo de metodologías en las que se buscó minimizar la cantidad de procesos burocráticos para dedicar más tiempo a llevar a cabo el desarrollo del software. Estas últimas están orientadas a facilitar los procesos de respuesta a cambios, aceptar un grado de incertidumbre, ofrecer mayor flexibilidad en el arranque de un proyecto y en la liberación de funcionalidades a usuarios finales, lo que mejora la experiencia de desarrollo de software para las personas involucradas. Éstas últimas son conocidas como metodologías ágiles.

2.2.1. Metodologías tradicionales

Este tipo de metodologías inician con un proceso riguroso de planificación, para extraer la mayor cantidad posible de información útil acerca del software a desarrollar, estableciendo metas, requerimientos del cliente y límites de trabajo, para definir qué trabajo está en el alcance establecido.

Estas metodologías son lineales y establecen el proyecto como un flujo unidireccional de trabajo, es decir, que una vez que se haya concluido una etapa del proyecto no se puede regresar a ella; es por ello que se requiere una gran planificación, que no contempla cambios en los requerimientos. También, en las metodologías tradicionales el cliente tiene una participación limitada.

Debido a que muchas de las metodologías tradicionales han sido probadas por muchos equipos y a través de muchos años, las más relevantes que se describirán a continuación son. Cascada y Espiral.

2.2.1.1. Cascada

Fue propuesto en 1970 por Winston Royce [1], establece una secuencia lineal de las etapas del SDLC. Su forma de funcionamiento consiste en que cada una de las etapas del SDLC debe completarse antes de que la siguiente etapa inicie. Además de que al final de cada etapa el producto se revisa para verificar que sigue cumpliendo con los requerimientos.

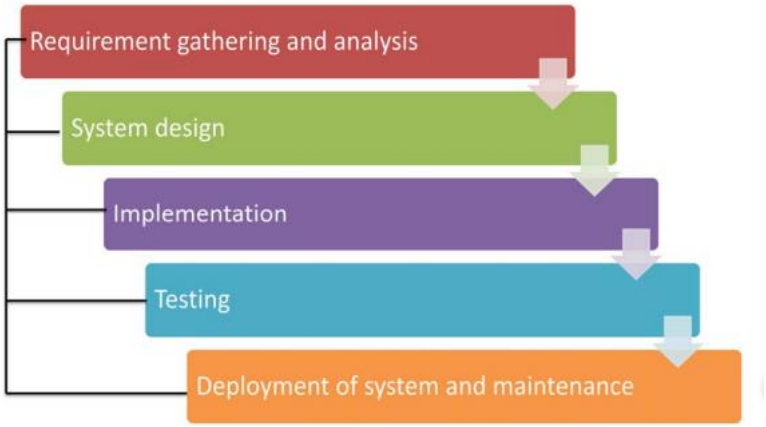


Figura 2.1. Metodología Cascada.[2]

Bajo este esquema se definen las siguientes ventajas de esta metodología:

- Documentación y diseño de arquitectura que facilitan el conocimiento del proyecto a nuevos miembros.
- Fácil comprensión.
- Un mayor monitoreo del desempeño y problemas por la realización de una etapa a la vez
- Funciona muy bien en proyectos pequeños y con requerimientos bien definidos.

Sin embargo, también se presentan los siguientes inconvenientes:

- No contempla que los requerimientos pueden cambiar o agregarse durante el SDLC.
- No se puede dividir el proyecto en diferentes etapas o módulos.
- Estimación de tiempo y costos complicada para cada etapa.
- No hay prototipos.
- Las tareas o problemas pueden no resolverse en una sola etapa.
- No es recomendado en proyectos grandes y complejos.

2.2.1.2. Espiral

Fue propuesto por Barry Boehm en su ensayo *A Spiral Model of Software Development and Enhancement.* [2] También se conoce como desarrollo o Metodología Incremental. Esta metodología surge como

[2] Fuente: http://www.revistaie.ase.ro/content/68/06%20-%20Stoica,%20Mircea,%20Ghilic.pdf

respuesta a ciertos inconvenientes de la metodología en cascada. Define el SDLC por medio de espirales, que se repiten hasta que se logren los objetivos y la calidad del producto.

Una importante característica de Espiral es que los riesgos se minimizan por completo, lo que puede ocasionar un aumento de costos, tiempos y esfuerzo. Esta minimización se logra gracias a la naturaleza de la metodología, debido a que primero se hacen prototipos, simulaciones, pruebas de referencia y entrevistas con usuarios.

El progreso del proyecto depende de cómo se puedan eliminar los riesgos, por lo que, se considera exitoso cuando no hay riesgos.

El objetivo de cada ciclo es producir un producto que mejora continuamente, es decir, se perfecciona agregando o solucionando funcionalidades.

Las repeticiones se llevan a cabo cuando los riesgos o conflictos amenazan el proyecto. En ese momento el producto tiene que pasar por un nuevo ciclo o iteración.

Esta metodología implementa los siguientes ciclos:

Objetivo y determinación alternativa.
Los objetivos se definen en colaboración con el cliente, así como posibles alternativas y condiciones técnicas.

Análisis y evaluación de riesgos.
Se identifican y evalúan riesgos potenciales y alternativas. Los riesgos son registrados, evaluados y minimizados. En este ciclo se usan prototipos como plantillas de diseño.

Desarrollo y pruebas.
Los prototipos se amplían y se agregan funcionalidades. Se escribe código, se prueban en entornos controlados hasta validarlo para su uso en un entorno de producción.

Planificación del siguiente ciclo.
Al finalizar la etapa se planifica el siguiente ciclo. Si existen errores, se busca solucionarlos en el siguiente ciclo.

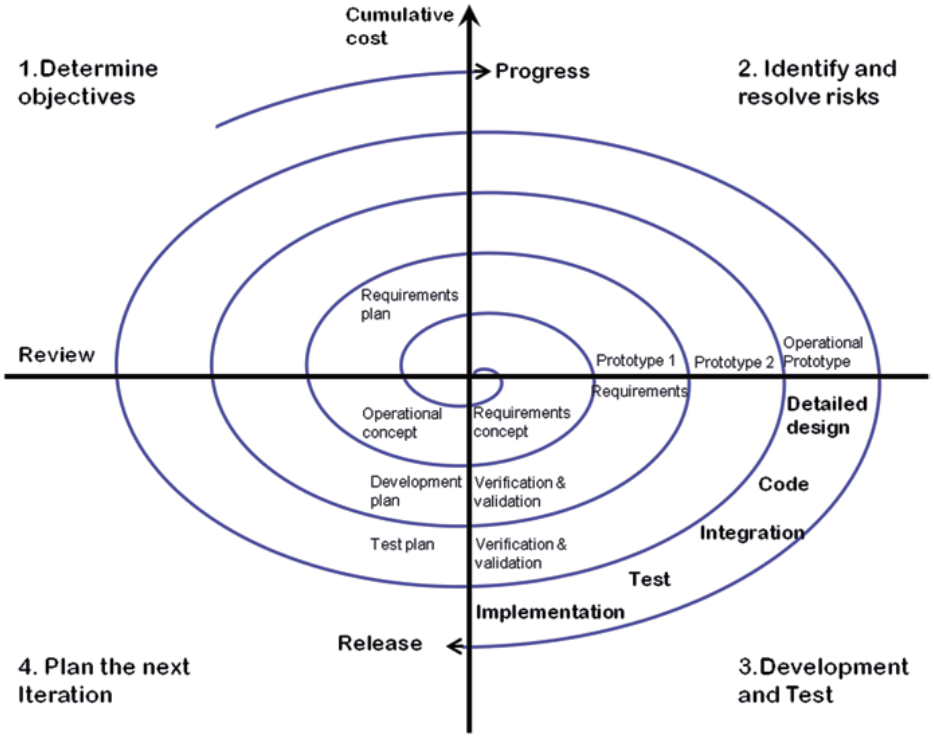


Figura 2.2. Metodología Espiral.[3]

2.2.2. Ágiles

Con el incremento del uso de computadoras y su masificación, surgieron nuevas y diferentes necesidades de los usuarios. Fue por ello que las metodologías tradicionales de desarrollo de software presentaron algunos problemas en cuanto a las nuevas tendencias de productos de software y que a inicios de la década de 1990 [3] se presentaron propuestas para lograr resultados más eficientes sin comprometer la calidad. Estas propuestas metodológicas fueron llamadas ágiles por su versatilidad y dinamismo.

[3] Fuente: https://www.researchgate.net/profile/Benjamin_Cowley/publication/287268554/figure/fig1/AS:614209679261713@1523450407826/The-spiral-model-of-design-This-consists-of-four-phases-1-Determine-objectives.png

Todas las metodologías ágiles parten de la filosofía del Manifiesto Agile de software, en el que se forman los valores fundamentales para desarrollar software, que establecen una jerarquía en cuanto a los recursos:

"Individuos e interacciones, por encima de procesos y herramientas; software funcionando, por encima de documentación excesiva; colaboración con el cliente, por encima de negociación contractual; y respuesta ante el cambio, por encima de seguir un plan" [3]

Este tipo de metodologías tienen la finalidad de solventar algunas deficiencias de las metodologías tradicionales, principalmente la carga de trabajo en la planificación, su poco dinamismo y la capacidad de respuesta ante cambios del cliente.

En las metodologías ágiles los proyectos se estructuran en proyectos más pequeños independientes entre sí, en los que se desarrollan un conjunto de características en pequeños periodos. Además, se requiere una gran participación del cliente, para establecer un canal de comunicación que provea de una retroalimentación a entregas constantes del proyecto.

Otro enfoque importante de estas metodologías es que se espera que haya mejoras o actualizaciones del producto, pues tienen un flujo no secuencial de trabajo, por lo que pueden generarse iteraciones para repetir etapas.

Las metodologías ágiles más destacadas y que se describirán a continuación son: SCRUM, XP, KANBAN. [3]

2.2.2.1. SCRUM

Esta metodología ágil provee un marco de trabajo diseñado para lograr una colaboración eficaz de equipos en proyectos, esto a través de emplear un conjunto de reglas y artefactos que definen roles para establecer una estructura necesaria para su funcionamiento.

Esta metodología se rige por equipos SCRUM que son autogestionados, multifuncionales y que trabajan en iteraciones. La autogestión permite a los miembros del equipo elegir la forma de trabajo que mejor se adapte a ellos en vez de acoplarse a seguir lineamientos. Además, todos los miembros tienen el conocimiento necesario para realizar el trabajo.

Todas las entregas del producto se realizan en iteraciones y cada una de ellas crea nuevas funcionalidades o se trabaja sobre alguna que el cliente requiera.

En SCRUM se definen los siguientes tres roles:

- *Scrum master:* Asegura que el equipo está adoptando la metodología, sus prácticas, valores y normas. Es el líder del equipo, pero no gestiona el desarrollo.

- *Product owner:* Es la única persona que representa a los interesados o clientes. Es el responsable de maximizar el valor del producto y trabajo del equipo de desarrollo. Además, se encarga de gestionar la lista ordenada de funcionalidades requeridas o *Product Backlog*.
- El *Product Backlog* es una lista, ordenada bajo un criterio como valor, riesgo, prioridad o necesidad de los requerimientos del producto. Esta lista nunca está terminada, ya que evoluciona durante el desarrollo del proyecto.
- *Scrum team:* Se encarga de convertir lo que el cliente define en el *Product Backlog*, en iteraciones funcionales del producto. Todos los miembros del equipo tienen el mismo nivel e importancia.

En la metodología SCRUM se define un evento principal o *Sprint*, que corresponde a una ventana de tiempo donde se crea una versión utilizable del producto, también conocido como incremento. Cada *Sprint*, se considera como un proyecto independiente, cuya duración máxima es de un mes.

A su vez, cada *Sprint* se compone de los siguientes elementos:

- Reunión de Planeación del *Sprint*: Aquí se define el plan de trabajo, como qué se va a entregar y cómo se logrará. En otras palabras, el diseño del producto y la estimación de la cantidad de trabajo. Esta reunión dura ocho horas para un *Sprint* de un mes o de manera proporcional.
- *Daily Scrum:* Es un evento o junta del equipo de desarrollo que dura 15 minutos cada día. Su finalidad es explicar qué se logró realizar desde la última reunión, qué se hará antes de la siguiente reunión y los obstáculos que se han presentado.
- Revisión del *Sprint*: Esto ocurre al final del *Sprint* y dura aproximadamente cuatro horas para un proyecto de un mes. El *Product owner* revisa el trabajo realizado, identifica lo que no se logró y discute acerca del *Product Backlog*. El *Scrum team* comparte los problemas que se encontraron y cómo se solucionaron; además, muestra el incremento del producto realizado en dicho *Sprint*.
- Retrospectiva del *Sprint*: Es una reunión de tres horas del *Scrum team* donde se analiza la forma de la comunicación, el proceso y las herramientas usadas. Así como qué estuvo bien, que no y se planifican mejoras para el siguiente *Sprint*.

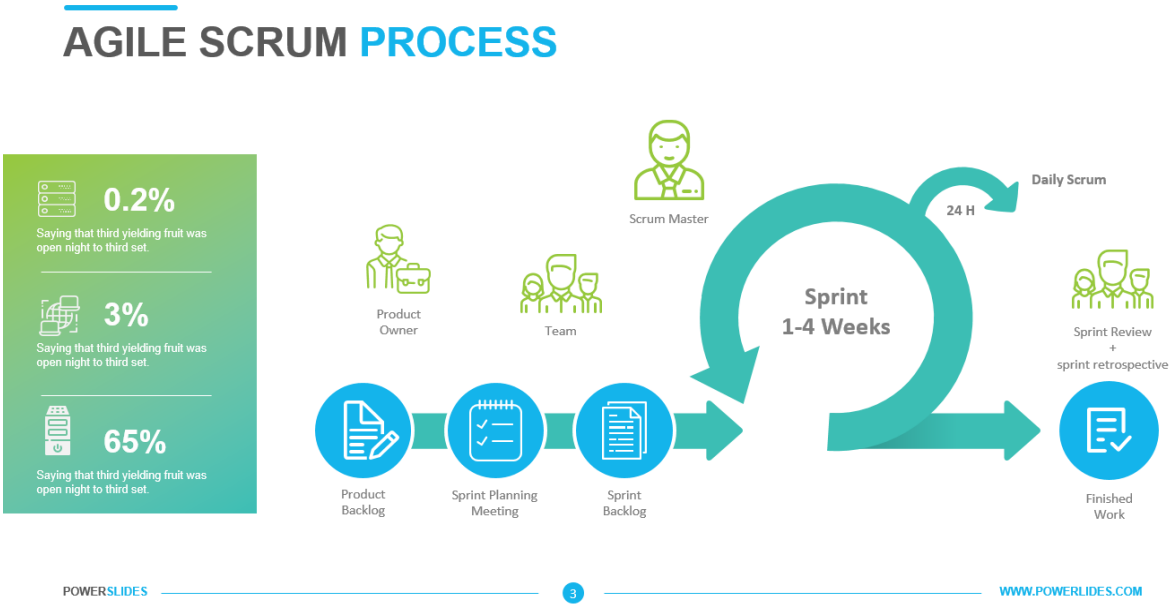


Figura 2.3. Metodología SCRUM.[4]

2.2.2.2. XP

La metodología de Programación Extrema o *eXtreme Programming* (XP) se centra en potenciar las relaciones interpersonales para el éxito del desarrollo de software, es decir, promueve el trabajo en equipo y prioriza el aprendizaje de los programadores a través de un buen entorno de trabajo.

XP fue desarrollada por Kent Beck [3] para guiar a equipos de desarrollo de software pequeños o medianos, entre dos y diez miembros, en ambientes de requerimientos cambiantes. Además, se basa en cinco principales valores: simplicidad, comunicación, retroalimentación, respeto y coraje.

Otro aspecto fundamental de XP se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, así como una comunicación fluida entre todos los participantes y simplicidad en las soluciones implementadas. Otra característica importante es que XP se adapta fácilmente a los cambios en los proyectos, por lo que es útil en aquellos proyectos con requisitos imprecisos y variables.

Historias de usuario

[4] Fuente: https://www.powerslides.com/wp-content/uploads/2019/02/Agile-Scrum-Process-3.png

Se tratan de formatos en los que el cliente describe de manera resumida las características que requiere el producto, sean funcionales o no funcionales. Cada historia de usuario se fragmenta en tareas de programación o *Task Cards*, que se asignan a programadores para implementarse en una iteración.

Cada historia de usuario puede contener diferentes elementos como: fecha, actividad (nueva función, corrección o mejora), número de historia, prioridad, referencia a otra historia, prueba funcional, riesgo, descripción y notas.

Roles

Bajo la metodología XP se identifican principalmente los siguientes roles o puestos, que permite establecer responsabilidades y un mejor control de las tareas:

- Cliente: Es el encargado de escribir las historias de usuario y pruebas funcionales para validar su implementación. Además, se encarga de definir prioridades.
- Encargado de pruebas: Apoya al cliente a escribir las pruebas funcionales. Implementa prueba regularmente y comparte los resultados con el equipo de desarrollo.
- Encargado de seguimiento: Proporciona retroalimentación al equipo. Realiza un seguimiento del progreso de cada iteración.
- Entrenador o *Coach*: Es el responsable del proceso de manera general. Provee guías al equipo para que se continúe el proceso de manera correcta.
- Consultor: Miembro externo al equipo, debe poseer conocimiento en cierto tema que pueda ser de apoyo para el proyecto.
- Gestor: Es el vínculo entre el cliente y los programadores, apoya a que el equipo trabaje efectivamente. Se encarga de la coordinación.

Proceso XP

XP tiene las siguientes prácticas: *planning game*, entregas pequeñas, diseño simple, programación en parejas, pruebas, *refactoring*, integración continua, propiedad común del código, cliente en sitio, metáfora y estándares de código.

Para que XP cumpla el objetivo de disminuir la dificultad del costo de cambios a lo largo del proyecto, sugiere el uso de las tecnologías disponibles para agilizar el desarrollo de software y aplicar las siguientes prácticas:

- Juego de la planificación o *Planning game*: Define el alcance y la fecha de cumplimiento de una entrega funcional completa o fecha de *release*, es decir, la fecha en que se puede poner en funcionamiento. Se dividen las responsabilidades entre el cliente y los desarrolladores.

El cliente emplea las historias de usuario para simplificar los casos de uso, los requerimientos y su importancia para cada iteración. A partir de éstas, los desarrolladores estiman costos y esfuerzos de su implementación, así como el número de iteraciones para completarla.

- Entregas pequeñas: Producción de versiones operativas del producto, aunque no cuenten con toda la funcionalidad del sistema. Cada versión constituye un resultado de valor.
- Metáfora: Una metáfora es una historia compartida que describe cómo debe funcionar el sistema, es decir, todos los miembros del proyecto deben tener la misma visión del sistema. La metáfora es una guía global del desarrollo.
- Diseño simple: Siempre se debe buscar la solución más simple para ser implementada en un momento determinado del proyecto.
- Pruebas: Todo código producido debe pasar por pruebas unitarias que se ejecuten ante cualquier modificación del sistema.
- Refactorización o *Refactoring*: Es la actividad de reestructurar el código para evitar duplicidad, mejorar la legibilidad, simplificarlo y flexible ante cambios.
- Programación en parejas: Todo código a realizar por programadores debe realizarse en parejas para disminuir la tasa de errores y buscar la mejor implementación, a través de compartir experiencia y conocimiento.
- Integración continua: Cada módulo o componente del producto debe integrarse con el resto, una vez que se aprobó su funcionamiento.
- Cliente en sitio: El cliente debe estar disponible en cualquier momento para el equipo, ya que éste se encarga de dirigir el trabajo por el camino que cumpla los objetivos.
- 40 horas por semana: Se debe trabajar como máximo 40 horas por semana y no se deben trabajar horas extras dos semanas seguidas, pues sería un síntoma de que algo está mal. Se busca un trabajo moderado para satisfacción de los programadores y aumentar su eficiencia, pues el trabajo extra los desmotiva.

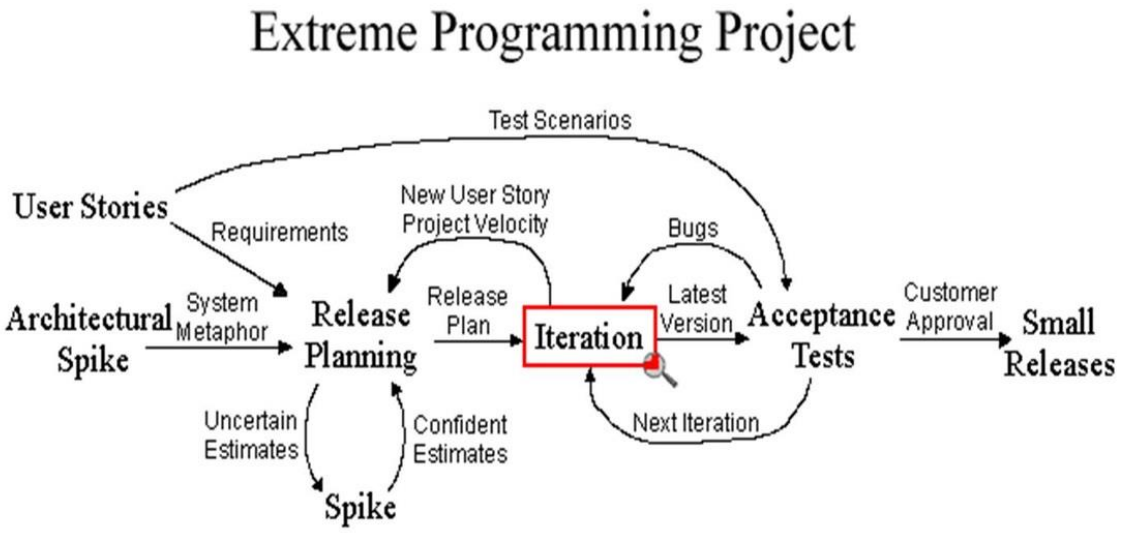


Figura 2.4. Esquema de la Metodología XP.[5]

2.2.2.3. KANBAN

Kanban es una metodología de trabajo que surgió en la industria manufacturera de vehículos en la década de 1940 por la empresa Toyota, que buscó optimizar los procesos y costos de producción basados en la demanda del mercado. [4]

Debido a su éxito, se probó en diferentes industrias y los resultados fueron benéficos. La industria del software adoptó Kanban como parte de las metodologías de desarrollo ágil, debido a su versatilidad y simplicidad, como el hecho de integrar su uso en equipos ya existentes y acoplarse a su forma de trabajo, pues no establece normativas ni roles que requieran de una reestructuración completa del proyecto.

El principio de Kanban se basa en visualizar el flujo de trabajo, limitar el trabajo en proceso o *Work In Progress* (WIP) para dejar de comenzar y comenzar a terminar. Además de brindar flexibilidad en la planificación, generar resultados más rápidos por su fácil entendimiento, establecer un enfoque más claro y transparencia en todo el ciclo de desarrollo del producto.
Los elementos de funcionamiento de Kanban son los *Kanban boards* y los *Kanban cards*.

Kanban cards

[5] Fuente: http://www.sandeepmukkara.com/blog/extreme-programming-xp-core-values-explained

Son una unidad de información representada en forma de tarjeta, donde cada elemento de trabajo a realizar se plasma. Esto permite al equipo completo poder rastrear de manera visual el estado de dicha tarea.

Kanban boards

Es la pieza central, es un tablero físico o virtual en el que todos los miembros del equipo pueden visualizar el flujo de trabajo. El tablero se estructura en al menos tres columnas, que permite identificar el estado de las actividades. Estas tres columnas básicas son: *To Do*, *In Progress* y *Done*.

Sin embargo, esta estructura básica puede modificarse de acuerdo con las necesidades de cada equipo o proyecto, de manera que permita una mejor visualización del flujo de trabajo.

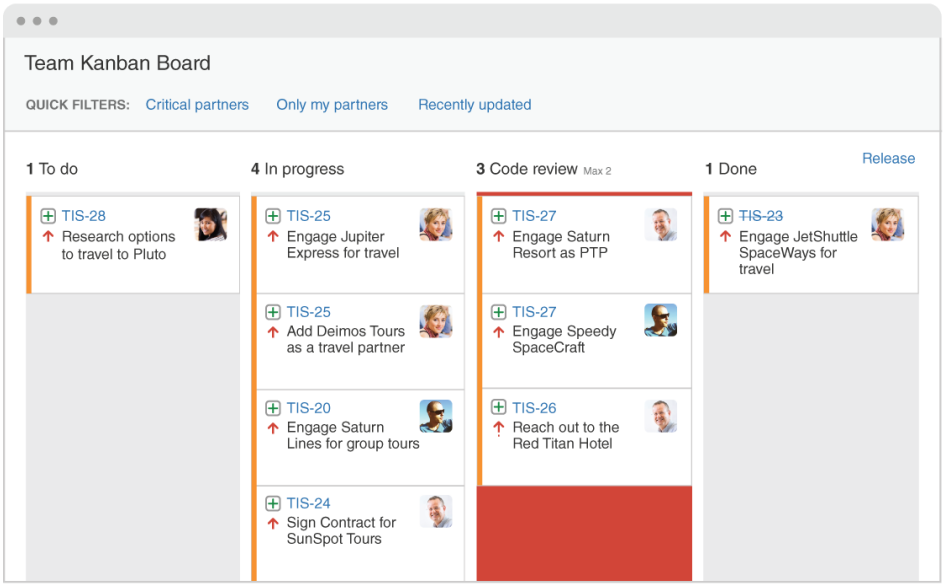


Figura 2.5. Ejemplo de *Cards* en Kanban.[6]

[6] Fuente: https://www.atlassian.com/agile/kanban

2.3. Tecnologías para el Desarrollo Web

2.3.1. Introducción

Tim Berners-Lee creó la *World Wide Web* (WWW) en 1989 mientras se encontraba trabajando como ingeniero de software en el CERN[7]. [5]

El objetivo principal de la WWW fue satisfacer la demanda de compartir de manera automática y eficiente la información entre universidades e institutos a través del mundo. Su idea básica fue fusionar las tecnologías de computadoras, redes de datos y el hipertexto[8] en un sistema global potente pero sencillo.

El diseño de la WWW permitió la creación de un navegador web que permitía el acceso sencillo a información existente y a una página web inicial que contenía información acerca de la guía telefónica del CERN y un servicio de búsqueda basado en palabras claves.

El navegador web original de Berners-Lee, llamado inicialmente *WorldWideWeb* y renombrado como Nexus, se ejecutaba sobre las computadoras NeXT, que contenía características de los actuales navegadores web; además de incluir la capacidad de modificar las páginas web directamente en el navegador.

[7] *Conseil Européen pour la Recherche Nucléaire* u Organización Europea para la Investigación Nuclear, es una organización europea formada en 1954 para realizar trabajos de investigación de la física de partículas. https://home.cern/about

[8] Hipertexto o *HyperText* es un concepto, propuesto por Ted Nelson en 1965, que se refiere a todo texto que no está limitado a ser lineal y puede contener enlaces a otros textos (hipervínculos). https://www.w3.org/WhatIs.html

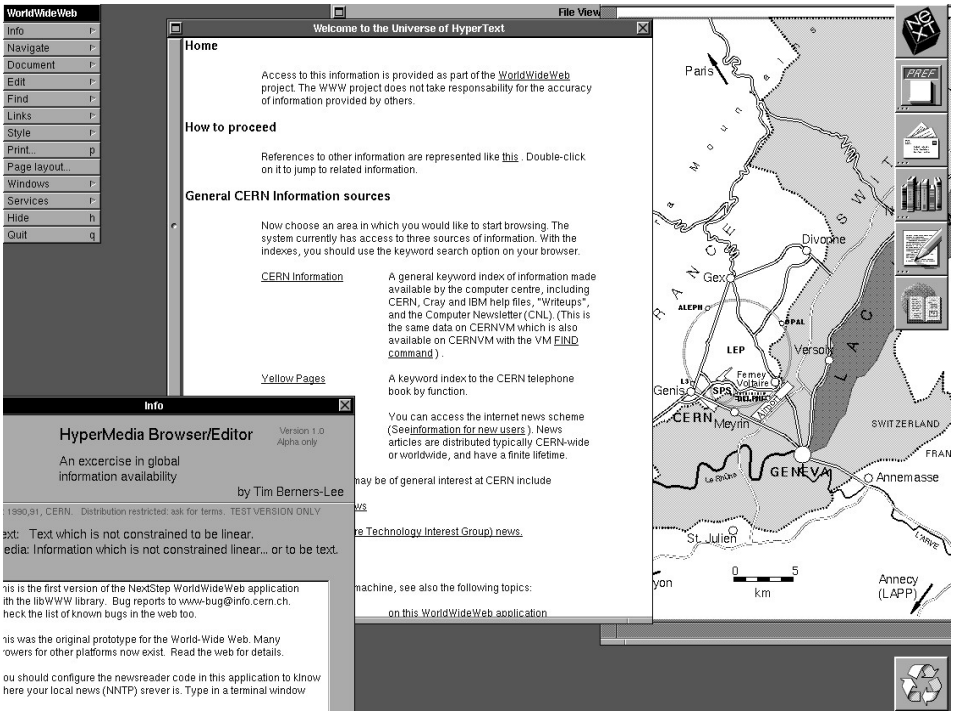


Figura 2.6. Primer navegador web (Nexus) creado por Berners-Lee.[9]

Al ver el potencial que tenía la WWW, se realizaron esfuerzos para crear servidores web para alojar diferentes páginas web que se creaban y se crearon diversos navegadores web que mejoraban continuamente la experiencia en Internet, como lo fueron: Lynx, Mosaic (posteriormente actualizado y renombrado Netscape), Internet Explorer, Opera, Safari, Firefox, Google Chrome y Edge. [6]

Los navegadores web proporcionaron una característica muy importante que los destacó e hizo crecer de manera acelerada: permitían el acceso multiplataforma, es decir, sin importar el sistema operativo desde el cual se accedía a la WWW.

Conforme fueron evolucionando los primeros navegadores, se fueron complementando con más capacidades, como la visualización de imágenes y contenido multimedia, que se implementó en el navegador *Netscape*, haciéndolo en su momento el más popular del mercado.

Sin embargo, surgió la necesidad de crear una herramienta que permitiera agregar dinámica a las páginas web, es decir, ser capaz de manipular y controlar la visualización de contenido en función de la interacción de los usuarios. Como solución, Brendan Eich en 1995, creó en 10 días un lenguaje de

[9] Fuente: https://home.cern/science/computing/birth-web/short-history-web

programación débilmente tipado llamado JavaScript, para complementar al navegador *Netscape* y poder manipular páginas web de una manera sencilla. [7]

Debido a la gran importancia que tomó JavaScript, se crearon diversos lenguajes de programación similares a éste. Por ello, en 1996, JavaScript comenzó a estandarizarse para tener un control de las diferentes implementaciones como *JScript* de Microsoft y así evitar dificultades de compatibilidad entre los diferentes navegadores.

La estandarización fue realizada por *ECMA International* [10], estableciendo el estándar ECMAScript 1 en 1997, basado en JavaScript. Por lo que, desde ese momento, JavaScript y las demás implementaciones pasaron a ser dialectos de ECMAScript.

ECMAScript 5, lanzado a finales del 2009, implementó mejoras significativas de compatibilidad y un modo estricto para proporcionar mejor control de comprobación de errores, que tiene soporte en Firefox 4 (2011), Chrome 19 (2012), Safari 6 (2012), Opera 12.10 (2012) e Internet Explorer 10 (2012). [8]

A partir de ECMAScript 6 se optó realizar actualizaciones anuales y nombrarlas con el año de su liberación, es decir, ECMAScript 6 es ECMAScript 2015. Esta versión fue una de las actualizaciones más importantes debido a que implementaron cambios significativos que permitieron hacer aplicaciones más complejas.

Esto permitió que los *frameworks* [11] de desarrollo web modernos puedan traducir a ECMAScript 5 para tener compatibilidad con cualquier versión de navegador web desarrollado después del 2012. Esto es especialmente importante cuando se hace uso de bibliotecas como React o *frameworks* como Angular, ya que utilizan funciones que no son compatibles con versiones de navegadores web rezagadas o que no han implementado nuevas versiones de ECMAScript.

Desde la creación de ECMAScript han surgido diferentes dialectos, entre ellos se ha destacado recientemente la creación de *TypeScript* creado en 2012 por Microsoft.

Typescript es un lenguaje de programación opcionalmente tipado y que es un *superset* de JavaScript [9], es decir que es totalmente compatible con JavaScript, y que permite una mayor productividad, ya que obtiene información del comportamiento del código gracias al tipado y las interfaces que pueden implementarse a través de objetos basados en clases. Este lenguaje no puede ser interpretado por los

[10] *European Computer Manufacturers Association* (ECMA) es una organización internacional fundada en 1961 encargada de la estandarización de información y sistemas de comunicación. https://www.ecma-international.org/

[11] *Framework* o marco de trabajo, es un esquema establecido y ordenado para organizar el software, garantizando la calidad y ahorrando tiempo por sus herramientas incluidas. https://neoattack.com/neowiki/framework/

navegadores web directamente por lo que se traduce generalmente a ECMAScript 5 para poder ejecutarse en cualquier navegador o en servidores con Node.js

En la actualidad el desarrollo web se encuentra en constante innovación por el surgimiento de nuevos *frameworks* y, gracias a su popularidad, se ha incursionado en plataformas móviles con el uso de *frameworks* como *React Native, Ionic* y *NativeScript*. Estos *frameworks* permiten utilizar el mismo lenguaje de programación, como Javascript o *Typescript*, en aplicaciones web y aplicaciones móviles. De esta manera no se tiene que utilizar lenguajes de programación como *Java* o *Kotlin* para Android y *Objective-C* o *Swift* para iOS, lo que permite un desarrollo más ágil, una curva de aprendizaje menor y un mantenimiento más sencillo.

2.3.2. Frontend

Frontend es la parte lógica de la aplicación que puede ver y con la que puede interactuar visualmente el usuario. En el ámbito del desarrollo web es la aplicación que se despliega en el navegador del usuario y con la que puede interactuar. [10]

El desarrollo del *frontend* básico se compone de *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) y el lenguaje de programación JavaScript. Sin embargo, es posible hacer uso de bibliotecas de JavaScript, como React, y *frameworks* como Angular y Vue, que facilitan la tarea de desarrollar interfaces dinámicas y atractivas. Por otro lado, existen *frameworks* para CSS, como *Bootstrap* y *SASS,* que facilitan la tarea implementar diseños atractivos.

Utilizando estas tecnologías es posible crear interfaces de usuario complejas (UI o *User Interface*) y una gran experiencia de usuario (UX o *User Experience*). UX se refiere a la facilidad y eficiencia con la que puede llevarse a cabo la interacción e intención de los usuarios en la aplicación. [11]

En el pasado sólo existían las *Multi-Page Applications* (MPA), las cuales requerían descargar el HTML, CSS y JavaScript, además de recursos como imágenes o videos cada vez que se navegaba de una página a otra dentro de un mismo sistema o aplicación. La evolución de JavaScript ha permitido la creación un nuevo paradigma de desarrollo web que son las *Single-Page Applications* (SPA). [12]

Las SPA permiten reescribir las páginas HTML de manera dinámica y sin interrumpir el flujo de la aplicación haciendo uso de *Asynchronous JavaScript And XML* (AJAX) con el que es posible hacer peticiones de datos al servidor sin necesidad de recargar una página para actualizar la vista con la nueva información. Gracias a esto, es posible sólo cambiar una parte de la página con JavaScript en vez de descargar completamente una página HTML. [13]

2.3.3. Backend

Backend se refiere a lo que el usuario no puede acceder directamente, sino por medio del *frontend*. En éste se implementa toda la lógica de operación de la aplicación, como la autenticación, el almacenamiento de datos, control de permisos y se almacenan archivos.

Los componentes principales del *backend* son: el sistema operativo, el servidor de archivos, la base de datos y el lenguaje de programación que se utiliza para hacer todas estas operaciones. Generalmente se hace uso de un conjunto de herramientas tecnológicas relacionadas y altamente compatibles, a lo que se conoce como un *stack* de aplicaciones.

Un *stack* es una colección de elementos para desarrollar una aplicación web, dos de los más populares son MEAN y LAMP. Ambos requieren un sistema operativo, un servidor web, un *framework,* una base de datos y un lenguaje de programación, como: PHP, Python, Java, JavaScript, C# o Ruby. [14]

LAMP es un acrónimo de:

- GNU/Linux (Sistema operativo)
- Apache (Servidor web)
- MySQL (Base de datos)
- PHP. (Lenguaje de programación)

Algunas variaciones pueden sustituir el uso de PHP por Python

MEAN es un acrónimo de:

- MongoDB (Base de Datos)
- Express (Web *framework* de *backend*)
- Angular (Web *framework* de *frontend*)
- Node.js. (Entorno de ejecución)

Algunas variaciones pueden utilizar React en vez de Angular, al cual se le conoce como MERN *stack*.

Existen dos paradigmas cuando se trata de un desarrollo *backend*, el primero funciona como una *Application Programming Interface* (API) y se usa generalmente para proveer de funcionalidad a las SPA, ya que una API web permite establecer localizaciones a los recursos de la base de datos a través del protocolo *Hypertext Transfer Protocol* (HTTP), los cuales se pueden consumir por la SPA al lanzar

peticiones HTTP a la URL deseada. El segundo paradigma hace uso del motor de *templates* del *framework* para crear MPA. [12]

2.3.4. Patrones MVC y MVT

2.3.4.1. MVC

El patrón de diseño *Model View Controller* (MVC) se encarga de separar la lógica de una aplicación de la lógica de la vista. Especifica que una aplicación consta de un modelo de datos, información de presentación e información de control. Este patrón de diseño requiere que cada uno de estos se separe en diferentes objetos para permitirles trabajar independientemente y sin afectarse entre ellos. [15]

Es un patrón de diseño arquitectónico utilizado para desarrollar aplicaciones, que se separa en Modelo, Vista y Controlador, permitiendo una reutilización de código eficiente y un desarrollo paralelo. Cada uno de sus tres componentes se describen a continuación:

- Modelo: se encarga de la parte de representación de datos del sistema, así como de la lógica de negocio o funcionalidad. Debe ser independiente del sistema de almacenamiento.
- Vista: también conocida como la Interfaz de Usuario, se compone de la información que se muestra al usuario y los mecanismos de interacción que puede tener con ésta. Recibe los datos y sólo los muestra.
- Controlador: es el intermediario entre el Modelo y la Vista, coordinando el flujo de información entre ellos. Maneja los eventos y gestiona sus acciones.

Este patrón se puede encontrar en muchos lenguajes de programación populares, como: Java, C#, Ruby y PHP. Además, muchos *frameworks* modernos implementan este patrón de arquitectura o alguna variante, como el MVT.

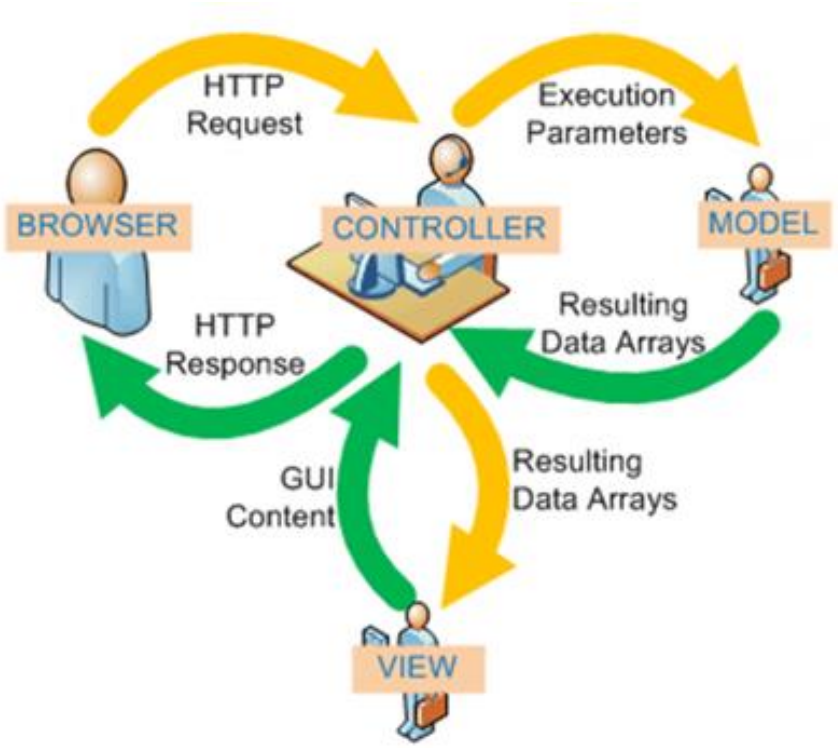


Figura 2.7. Arquitectura del MVC.[12]

2.3.4.2. MVT

La arquitectura de diseño de software *Model View Template* (MVT) es ligeramente diferente del MVC, ya que, aunque también es una colección de tres componentes esenciales estos juegan un rol diferente. [16]

- *Model* (Modelo): es una interfaz o capa de acceso a datos, que contiene los campos y comportamientos necesarios de los datos que serán almacenados. Es la estructura de datos lógica entre toda la aplicación.
- *View* (Vista): se utiliza para ejecutar la lógica de negocio e interactuar con el modelo para transportar datos y presentarlos mediante un *Template*. La vista obtiene datos de un *Model* y le da a cada *Template* acceso a datos específicos para mostrar. También es la encargada de aceptar y procesar las solicitudes HTTP y proporcionar una respuesta al usuario. Corresponde a la parte análoga del *Controller* del patrón MVC.

[12] Fuente: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

- *Template* (Plantilla): es una capa de presentación que maneja completamente la parte de la interfaz de usuario. Estos son archivos HTML, que se utilizan para representar datos. El contenido de estos archivos puede ser estático o dinámico.

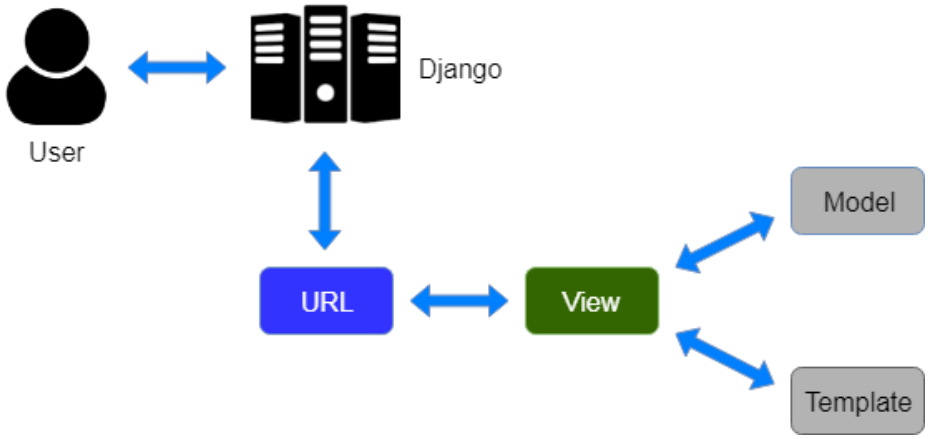


Figura 2.8. Arquitectura del MVT.[13]

2.3.5. Frontend *frameworks*

2.3.5.1. React

Es una biblioteca *Open Source* de JavaScript lanzada en 2013 por Facebook enfocada en el desarrollo de interfaces de usuario interactivas y con un gran crecimiento y adopción por desarrolladores en los últimos años. [17]

A pesar de ser una biblioteca que complementa a JavaScript, es tan popular y comparada con *frameworks* debido a su capacidad de desarrollo de funcionalidades de manera sencilla y el paradigma orientado a componentes. [18]

Algunas de las características más importantes de React son:

- Componentes: son clases o funciones de JavaScript que contienen lógica y el contenido HTML que se encapsula para generar un componente que puede ser reutilizado.
- JSX o JavaScript XML: es una extensión de JavaScript que permite crear vistas interactivas y dinámicas, que permiten escribir HTML en JavaScript.
- Propiedades y estados: los componentes reciben propiedades y contienen un estado único que puede ser cambiado a través de eventos.

[13] Fuente: https://www.javatpoint.com/django-mvt

- *Document Object Model* (DOM) virtual: permite que React genere el DOM de manera dinámica eficiente haciendo los cambios en memoria y compararlos con la versión actual del DOM real.

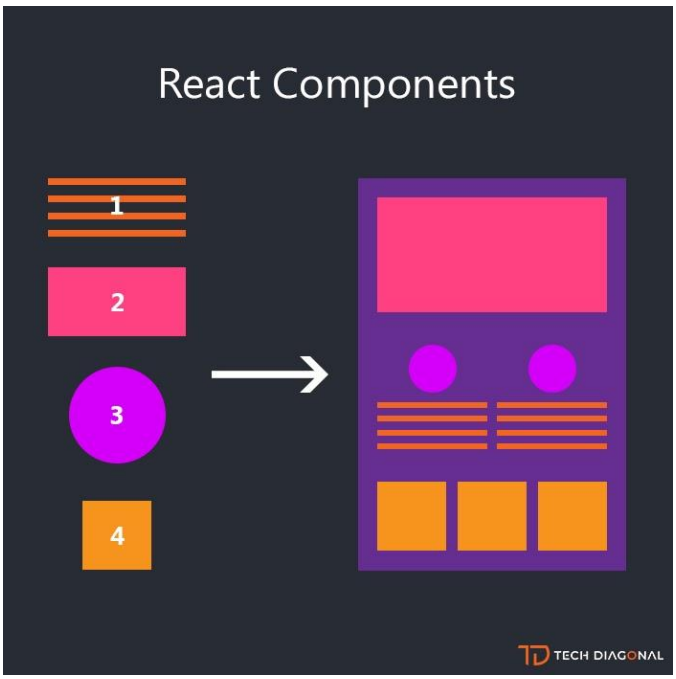


Figura 2.9. Arquitectura de una aplicación con React.[14]

A pesar de que React es una biblioteca, provee de beneficios como la facilidad de desarrollo de aplicaciones con buen rendimiento, flexibilidad y organización de código. Sin embargo, al ser una biblioteca especializada en la construcción de manejo de interfaces, no implementa el patrón *Model View Controller* (MVC) y requiere de la integración de otras herramientas y complementos. Lo anterior no ocurre en un *Framework*.

2.3.5.2. Angular

Es un web *framework* de *Open Source* desarrollado y sustentado por Google y colaboradores, el cual tiene el objetivo de facilitar la creación, mantenibilidad y escalabilidad de aplicaciones web eficientes; permite la creación de sistemas web de tipo *Single Page Application* (SPA), que muestran contenido nuevo a través de la carga de módulos y componentes sin necesidad de recargar el sitio. [19]

[14] Fuente: https://www.techdiagonal.com/reactjs_courses/beginner/understanding-reactjs-components/

Este *framework* es para el control de aplicaciones *Frontend* y define claramente la separación de la lógica de esta capa y la del *Backend*. Además, gracias a la estructura que provee previene la duplicidad de código e implementa el patrón *Model View Controller* (MVC) descrito en la Sección 2.3.4.1.

Angular emplea el lenguaje de programación *TypeScript* que es un lenguaje *Open Source* desarrollado por Microsoft; además, es totalmente compatible con la sintaxis de JavaScript debido a que está construido sobre él, pero agrega funcionalidades que lo hacen una excelente opción para el desarrollo de proyectos grandes, como: integración de *Object Oriented Programming* (OOP) y el manejo de tipos estáticos para las variables.

Algunas de las características más importantes del Angular son [20]:

- Módulos: permiten empaquetar los diferentes servicios, componentes y directivas de una aplicación para generar unidades funcionales que son reutilizables.
- Componentes: son clases que contienen datos, lógica y están asociados a un *template* que define lo que el usuario puede ver.
- *Templates*: son archivos HTML con instrucciones especiales que permiten modificar la vista antes de ser mostrada al usuario.
- Directivas: son atributos propios de HTML que permiten agregar funcionalidades a otros elementos HTML.
- Servicios: son funcionalidades que no están relacionadas con una vista específica pero que funcionan como interfaz que permita consumir o enviar recursos a una API; pueden ser reutilizados múltiples componentes por medio de inyección de dependencias.
- Inyección de dependencias: es una manera de integrar funcionalidades extras a componentes como los servicios.

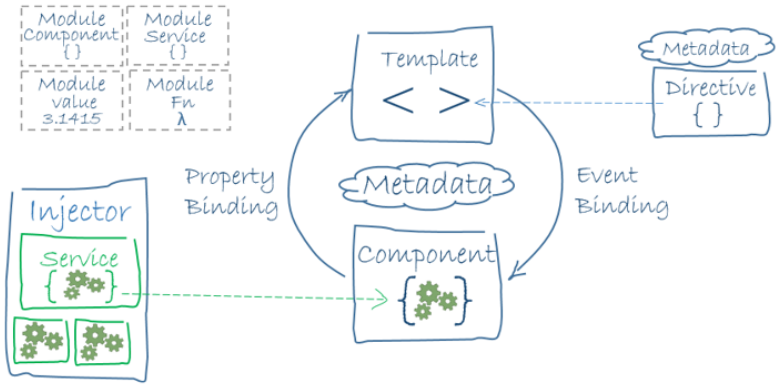


Figura 2.10. Arquitectura de una aplicación con Angular.[15]

2.3.6. Backend *frameworks*

2.3.6.1. Django

Django es un web *framework* de código abierto enfocado en seguridad y escalabilidad creado por Adrian Holovaty y Simon Willison en 2003 y fue liberado al público bajo la licencia *Berkeley Software Distribution* (BSD) en 2005 [21]. Actualmente es mantenido por la *Django Software Foundation.*

Django es un *framework* web de alto de nivel de Python enfocado en que sus módulos tengan alta cohesión y bajo acoplamiento; es decir, que al cambiar un módulo, aquellos que hacen uso de éste no tengan que ser modificados o muy poco, de manera que puedan agregarse, quitarse o intercambiarse diferentes componentes de la aplicación [22].

Por ejemplo, el módulo encargado de la Base de Datos no debe preocuparse de las peticiones web y no importa qué tipo de *Relational Data Base Manager System* (RDBMS) se utilice, ya que el funcionamiento de los demás componentes debe trabajar de la misma manera.

Una de las ventajas de utilizar Django es que está escrito en Python y esto abre la posibilidad de utilizar todas las bibliotecas y complementos que existen para este lenguaje de programación.

Adicionalmente, Django implementa el patrón de diseño *Model View Template* (MVT) descrito en la Sección 2.3.4.2, lo que permite tener una estructuración lógica de cada componente y definir su trabajo a realizar.

[15] Fuente: https://angular.io/guide/architecture

2.3.6.2. ExpressJS (NodeJS)

NodeJS es un entorno de tiempo de ejecución JavaScript de código abierto y multiplataforma desarrollado en 2009 por Ryan Dahl [23] con el enfoque de ser útil en la creación de programas de red altamente escalables como servidores web. Éste es ejecutado sobre el motor V8 de Google, el mismo que utiliza Google Chrome para ejecutar JavaScript por lo que es muy eficiente.

NodeJS se ejecuta sobre un solo proceso y no crea nuevos hilos con cada nueva petición ya que implementa operaciones primitivas asíncronas de entrada y salida (I/O). Esto permite evitar el bloqueo cuando se realiza una consulta a una Base de Datos o se lee del sistema de archivos, ya que reanudará la operación pendiente una vez que se tengan respuesta de la petición asíncrona.

Esto permite a NodeJS recibir miles de peticiones concurrentes en un solo servidor sin la necesidad de manejar concurrencia de hilos que a menudo son el origen de muchos errores.

Debido a que NodeJS está escrito en JavaScript, permite que los desarrolladores *Frontend* puedan crear *Backend* sin cambiar de lenguaje de programación, lo que generalmente es una barrera en la industria.

Algunos de los *frameworks* más populares para Node.js son:

- *ExpressJS*: un *Framework* de desarrollo web minimalista y ligero que permite crear *APIs* de manera muy rápida.
- *SailsJS*: un *Framework* MVC para proyectos empresariales.

2.3.7. Servidor

Un servidor es una computadora especial equipada con una serie de características de *Hardware* y programas de software que permite ofrecer servicios, recursos, datos o programas a otras máquinas o clientes a través de una conexión de red.

Debido a que un servidor puede ofrecer una gran cantidad de elementos, se tiene una clasificación de tipos de servidores en función de su propósito principal:

- **Servidor de archivos:** almacena y administra archivos y carpetas que pueden ser accedidos por otros clientes.
- **Servidor de impresión:** ofrece un servicio dedicado que permite a los clientes acceder a impresoras.
- **Servidor web:** está equipado con el *HyperText Transfer Protocol* (HTTP) para servir páginas como respuestas a las solicitudes de clientes, como los navegadores web.

- **Servidor de aplicación:** almacena y gestiona aplicaciones entre los usuarios de una organización.

2.3.7.1. Sistemas operativos para servidores

Dado que un servidor es una computadora especializada en proveer de ciertas funcionalidades o elementos a otras, requiere un sistema operativo para poder desempeñar sus actividades de manera correcta.

Un sistema operativo es un conjunto de programas y utilidades básicas para que una computadora funcione, siendo el núcleo uno de sus componentes principales; éste es el programa que realiza la comunicación entre los recursos de *Hardware*, como la gestión de memoria y de los procesos, con el software; permitiendo ejecutar otros programas haciendo uso de los recursos materiales.

Existe una gran diversidad de sistemas operativos que se especializan en potenciar los recursos de *Hardware* de los servidores, siendo los derivados de GNU/Linux los más efectivos y usados.

A continuación, se listan las dos opciones de sistemas operativos de GNU/Linux más populares, *Open Source* y efectivos para usarlos en un servidor.

2.3.7.1.1. Debian

Debian es una distribución de GNU/Linux creada en 1993 por Ian Murdock. Debian es uno de los sistemas operativos más populares basado en GNU/Linux y es la base de muchas otras distribuciones muy populares como Ubuntu.

Al igual que todas las distribuciones de GNU/Linux, su núcleo está basado en éste y tiene una amplia compatibilidad con diferentes arquitecturas y *Hardware*, por lo que es difícil encontrar una configuración que no sea compatible. [24]

Debian cuenta con más de 59 mil paquetes *Open Source* disponibles para descargar y un sistema de manejador de paquetes que permite realizar actualizaciones del sistema operativo con un comando y de una manera muy sencilla. [24]

Debido a su buen ciclo de actualización del sistema y sus paquetes a través del gestor de paquetes apt, su seguridad, estabilidad, preconfiguración y *Software Open Source*, lo hacen una opción excelente para emplearlo como sistema operativo de un servidor. [25]

2.3.7.1.2. CentOS

CentOS (Community ENTerprise Operating System) es una distribución de GNU/Linux basada en Red Hat Enterprise Linux (RHEL). Su primera versión fue publicada en 2004 y fue una bifurcación de RHEL versión 2.1AS. [26]

La principal diferencia con RHEL es que es mantenido por la comunidad y su distribución es libre pero dado que está basado en RHEL, contiene estándares de seguridad empresariales muy altos, lo cual le permite ser muy estable y cuenta con periodos de soporte largos.

Su gestor de paquetes es yum, con el que es posible obtener una gran cantidad de *Software* complementario. Sin embargo, apt de Debian es más avanzado y estable, además de tenerlos mejor organizados en sus repositorios. [25]

2.4. Seguridad Web

Como se mencionó en la Sección 2.3.1, Internet ha tenido un gran crecimiento y junto con él se han presentado vulnerabilidades que han sido aprovechadas por personas con fines lucrativos y maliciosos. Por ello, Internet no es un lugar totalmente seguro y se requieren tomar medidas preventivas para salvaguardar los recursos de un sistema web y de sus clientes.

Al hablar de seguridad web se refiere a la protección de bienes como principalmente aplicaciones web y Bases de Datos. Esta última es muy esencial pues los datos que se almacenan ahí se han convertido en un activo o recurso muy importante de las empresas y entidades gubernamentales.

La manera en que la seguridad web busca la prevención de diferentes ataques malignos es a través de una estrategia de diseño que evite accesos no autorizados, uso indebido, modificaciones indebidas o destrucción de recursos.

Los conceptos principales involucrados en la seguridad web son cubiertos por los servicios de seguridad que se describen a continuación [27] [28] [28]:

Autenticación: es el proceso de identificar de manera única a cada usuario o entidad que utiliza una aplicación o servicio.

Autorización: proceso que se encarga de verificar las capacidades que tiene un usuario autenticado sobre un sistema o recurso restringido.

Confidencialidad: se encarga de asegurar que la comunicación entre la aplicación y el usuario sea privada.

Integridad: asegura que los datos están protegidos ante modificaciones accidentales o maliciosas cuando circulan del sistema al usuario final o en su almacenamiento en una Base de Datos.

Disponibilidad: implica garantizar que el sistema tenga la capacidad de atender las peticiones de usuarios o entidades legítimas en cualquier momento.

La manera en que es posible aplicar la mayoría de los servicios de seguridad descritos anteriormente es a través de mecanismos que provee el campo de la Criptografía, como el cifrado de información.

2.4.1. Cifrado

El cifrado es el proceso de transformar datos de manera que la información clara o legible para los humanos sea convertida a un formato codificado o ilegible para cualquier persona; de manera que únicamente sea posible recuperar la información en claro a través de la realización de una transformación inversa, llamada descifrado, por las entidades autorizadas.

Es un elemento fundamental de seguridad de datos y es la forma más simple e importante de impedir que alguien no autorizado robe o tenga acceso a la información de un sistema web. Además, se usa ampliamente en Internet para garantizar la confidencialidad de la información personal enviada entre navegadores web (clientes) y servidores. [29]

Un mecanismo muy común para la implementación del proceso de cifrado son las funciones *hash*; éstas son un algoritmo matemático que al ejecutarse contra cualquier contenido devuelve una salida única y de tamaño fijo para cada valor de entrada. Los elementos a los que se le puede aplicar una función *hash* son: cadenas de texto, documentos, sonidos, videos, imágenes, entre otros.

Los *hashes* criptográficos son la base fundamental de los procesos de autenticación e integridad, es decir, para las firmas electrónicas. Sin embargo, al ir evolucionando y mejorando la capacidad de los equipos de cómputo, las herramientas tecnológicas y técnicas criptográficas, se tienen diversos algoritmos de funciones *hash* que han solucionado vulnerabilidades y deficiencias. Por ejemplo, SHA-2 resuelve problemas de ataques de sus antecesores MD5, SHA-0 y SHA-1.

2.4.1.1. SHA-2

Secure Hash Algorithm 2 (SHA-2) es el conjunto de funciones *hash* criptográficas SHA-224, SHA-256, SHA-384 y SHA-512, diseñadas por la *National Security Agency* (NSA) en Estados Unidos a través de un *Federal Information Processing Standard* (FIPS). [30]

Las funciones *hash* SHA-2 están implementadas en una gran variedad de aplicaciones y protocolos de seguridad como: *Secure Shell* (SSH), *Transport Layer Security* (TLS), *Secure Sockets Layer* (SSL), *Pretty Good Privacy* (PGP) y en criptomonedas como Bitcoin.

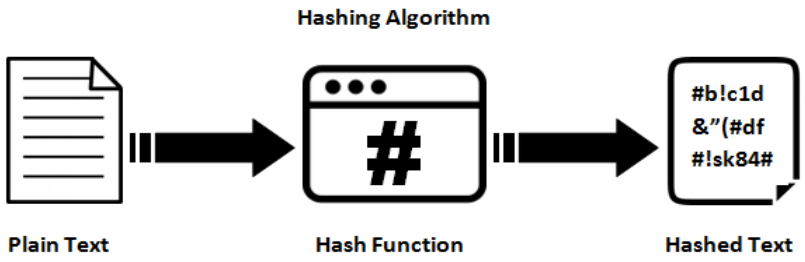


Figura 2.11. Funcionamiento de un algoritmo *Hash.*[16]

La manera en que se obtiene un *hash* de un mensaje de forma general es la siguiente:

1. Se convierte el mensaje en ASCII para tener una representación de cada letra de manera numérica
2. Se convierte el número de cada letra de base Hexadecimal a base Binaria.
3. Se une la cadena de bits y se completa con ceros para obtener un número de la longitud de acuerdo con la implementación del algoritmo.
4. Se aplican diversas transformaciones, definidas en el algoritmo *hash* empleado, que generan una cadena única y este es el código *hash* resultante.

2.4.1.2. PBKDF2

Password-Based Key Derivation Function 2 (PBKDF2) es una función de derivación con un costo computacional variable que se utiliza para reducir las vulnerabilidades a los ataques de fuerza bruta en los algoritmos *hash*. Su estructura es la siguiente:

```
<algoritmo>$<iteraciones>$<salt>$<hash>
```

Donde:

- **<algoritmo>:** es el algoritmo con el que se va a cifrar nuestra cadena por ejemplo SHA256 o SHA512.
- ***<salt>:*** es una cadena corta aleatoria que se agrega a la cadena que se quiere cifrar
- **<iteraciones>**: es el número de veces que se volverá a aplicar el *hash* a la cadena.
- ***<hash>:*** es la cadena resultante después de las iteraciones.

[16] Fuente: https://www.thesslstore.com/blog/difference-sha-1-sha-2-sha-256-hash-algorithms/

Por ejemplo, un *hash* almacenado se vería de la siguiente manera:

```
<algoritmo>$<iteraciones>$<salt>$<hash>
pbkdf2_sha256$36000$Fmyyx***$8ICNz5JduW***
```

La dificultad de un ataque de fuerza bruta aumenta con el número de iteraciones, ya que esto requiere que cada proceso se ejecute miles de veces para cada prueba.

Por otra parte, el uso de *salt* previene que los atacantes pre calculen un diccionario de claves derivadas. Esto no impide la posibilidad de realizar un ataque por fuerza bruta pero sí vuelve miles de veces más costoso computacionalmente realizarlo. [31]

Gracias a los beneficios de seguridad que complementa a las funciones *hash*, los PBKDF2 son empleados en distintas aplicaciones, por ejemplo: Django lo emplea junto con SHA-256 para el almacenamiento de las contraseñas de los usuarios. [32]

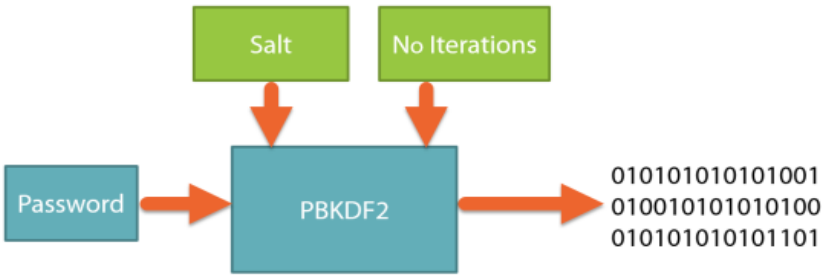


Figura 2.12. Funcionamiento de PBKDF2.[17]

2.4.1.3 JSON Web *Token*

JSON Web Token (JWT) es un estándar de Internet para crear *tokens* de acceso que proporcionan un mecanismo para la identificación y asignación de privilegios a ciertos datos, proporcionando además la opción de incluir firma o un cifrado. Los datos se conocen como carga útil (*payload*) y emplea el estándar para la representación e intercambio de información conocido como JavaScript Object Notation (JSON).

JWT parte de otros estándares basados en JSON, como *JSON Web Signature* (JWS) y *JSON Web Encryption* (JWE). [33]

[17] Fuente: https://stephenhaunts.com/2014/10/08/password-based-key-derivation-functions-in-net/

JWS es una forma de garantizar la integridad de la información en un formato altamente serializable y legible por máquina. Se utiliza un algortimo de cifrado o *hash* para generar una firma con el contenido del mismo, de manera que si es modificado es facilmente detectable, en el caso del *hash* unicamente se genera con el contenido del encabezado y la carga util y esta es la firma, para el caso del cifrado se cifran los dos elementos anteriores y se obtiene un *hash* del resultado que se vuelve la firma. El algoritmo que se utiliza esta definido en el encabezado. Se puede utilizar para enviar información de un sitio web a otro y está especialmente dirigido a las comunicaciones en la web.

JWT se representa como una secuencia de caracteres o partes seguras separadas por el carácter punto ('.'), que será la representación de *tokens* resultantes de este estándar.

Cada parte contiene un valor codificado en Base64URL, que es una modificación del estándar Base64 para permitir los caracteres empleados en las URLs (*Uniform Resource Locator*) y nombres de archivos. Base64URL está detallado en el RFC 4648 propuesto por la *Network Working Group*. [34]

Los tres componentes que conforman a un JWT son:

- Encabezado (*Header*)
- Carga útil (*Payload*)
- Firma (*Signature*)

A continuación, se presenta un ejemplo de la estructura de un JWT en base64URL:

```
<header>.<payload>.<hash>
eyJ0eXAi***.eyJzZXJ2ZX***.Oeh39yCq***
```

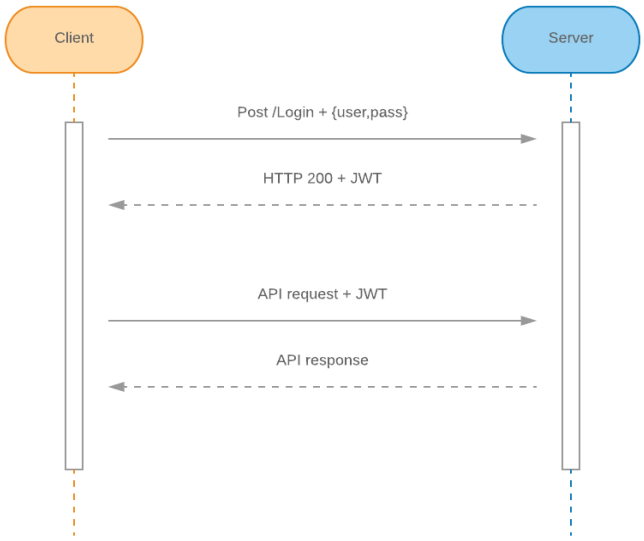


Figura 2.13. Generación y funcionamiento de JWT.[18]

2.4.2. Universally Unique Identifier (UUID)

Los *Universally Unique IDentifier* (UUID), también conocidos como *Globally Unique IDentifier* (GUID) son cadenas de caracteres de 16 Bytes (128 bits) generadas a partir de ciertas variables del contexto que garantizan la unicidad en el espacio y el tiempo. Se encuentran definidas en el RFC 4122.

Los UUID surgieron en proyectos de años anteriores y han ido evolucionando de manera que incluso se han desarrollado diferentes versiones que proveen de mejoras y metodologías distintas en la formación de los propios UUID.

En la versión 4, el valor del UUID se genera aleatoriamente empleando cuatro bits para indicar esta versión, mientras que 2 bits se emplean para indicar la variante. Por lo tanto, se tendrán 6 bits de variante y versión predeterminadas. Restando los 6 bits anteriores se tienen 122 bits restantes para la parte generada de forma aleatoria. Por lo tanto, se tiene un total de 2^{122} posibilidades diferentes. [28]

Debido a que proveen de identificadores prácticamente únicos por la pequeña probabilidad de colisiones (repetición de ID) se emplean en muchas aplicaciones que requieran la asociación de

[18] Fuente: https://miro.medium.com/max/858/0*g84vZ5p-9qMftyFy

archivos o recursos a IDs. Por ejemplo, se emplean como Folio Fiscal para las facturas digitales en México. [35]

2.5. Bases de Datos

2.5.1. Bases de Datos relacionales

Una Base de Datos relacional es una recopilación de elementos de datos con relaciones predefinidas entre ellos. Estos elementos se organizan de forma lógica en un conjunto de tablas con columnas y filas. Las tablas se utilizan para guardar información sobre los objetos que se van a representar en la Base de Datos.

Cada columna de una tabla guarda un determinado tipo de dato y almacena el valor real de un atributo. Las filas de la tabla representan una recopilación de valores relacionados de una instancia de una entidad. Cada fila de una tabla podría marcarse con un identificador único denominado llave primaria, mientras que filas de varias tablas pueden relacionarse con llaves primarias externas. Se puede obtener acceso a estos datos de muchas formas distintas sin reorganizar las propias tablas de la base de datos. [36]

2.5.1.1. PostgreSQL

Los orígenes de PostgreSQL se remontan a 1986 como parte del proyecto POSTGRES en la Universidad de California en Berkeley y tiene más de 30 años de desarrollo activo en la plataforma central. [37]

PostgreSQL es un potente sistema de Base de Datos relacional de objetos de código abierto que usa y amplía el lenguaje SQL combinado con muchas características que almacenan y escalan de manera segura las cargas de trabajo de datos más complicadas. [37]

PostgreSQL se ha convertido en la base de datos relacional de código abierto preferida por muchos desarrolladores empresariales y *startups*, impulsando aplicaciones empresariales y móviles líderes. Ofrece diversa solidez, administración de transacciones y paralelismo de consultas. [38]

2.5.1.2. SQLite

D. Richard Hipp diseñó SQLite en el año 2000 mientras trabajaba para *General Dynamics* por contrato con la Marina de los Estados Unidos.

Los objetivos de diseño de SQLite fueron permitir que el programa funcionará sin instalar un sistema de gestión de Base de Datos o sin necesidad de un administrador de Base de Datos.

SQLite es un motor de base de datos SQL incorporado. A diferencia de la mayoría de las demás bases de datos SQL, SQLite no tiene un proceso de servidor separado. SQLite lee y escribe directamente en archivos de disco ordinarios, es decir, se monta sobre el mismo dispositivo sobre el que se va a utilizar la aplicación. Es una Base de Datos SQL completa con múltiples tablas, índices, *triggers* y vistas que se encuentra contenido en un solo archivo de disco. [39]

La portabilidad la hace ideal para aplicaciones en dispositivos móviles como Base de Datos de prueba en entornos de desarrollo locales.

2.5.2. Bases de Datos NoSQL

Las Bases de Datos NoSQL surgen por las deficiencias encontradas en los modelos relacionales para manejar gigantescas cantidades de información de una manera rápida y eficaz. Un sistema de almacenamiento NoSQL no implementa los conceptos del modelo relacional, sino que trabaja con otro tipo de modelos de datos, como: objetos organizados por llave-valor o grafos.

Las Bases de Datos NoSQL se adaptan muy bien a muchas aplicaciones modernas, como dispositivos móviles, web y juegos, que requieren Bases de Datos flexibles, escalables, de alto rendimiento y altamente funcionales para proporcionar excelentes experiencias de usuario. [40]. Sin embargo, su principal ventaja radica en su uso para datos no estructurados, sean orientado a documentos, orientado a columnas, orientado a graficas o a llave-valor. En estos casos toda la información se almacena en una misma particion, por lo que realizar operaciones de escritura y/o lectura de estrcuturas complejas de datos en una sola operación.

- **Flexibilidad**: ofrecen esquemas flexibles que permiten un desarrollo más rápido e iterativo. El modelo de datos flexible hace que las bases de datos NoSQL sean ideales para datos semiestructurados y no estructurados.

- **Escalabilidad**: están diseñadas para escalar usando clústeres distribuidos de hardware en lugar de escalar añadiendo servidores.
- **Alto rendimiento**: están optimizados para modelos de datos específicos y patrones de acceso que permiten un mayor rendimiento que el intento de lograr una funcionalidad similar con bases de datos relacionales.
- **Altamente funcional**: proporcionan interfaces altamente funcionales y tipos de datos que están diseñados específicamente para cada uno de sus respectivos modelos de datos.

2.5.2.1. MongoDB

El desarrollo de MongoDB comenzó en 2007 de la mano de 10gen Inc. (ahora llamada MongoDB Inc.). MongoDB es un sistema de Base de Datos NoSQL, orientado a documentos y de código abierto. Se encuentra construido sobre JavaScript y escrito en C++.

En lugar de guardar los datos en tablas, como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos *Binary JSON* o BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida. [41]

Algunas de sus características son:

- **Consultas ad hoc**: soporta la búsqueda por campos, consultas de rangos y expresiones regulares. Las consultas pueden devolver un campo específico del documento, pero también puede ser una función definida por el usuario para su mejor ocupación
- **Indexado**: cualquier campo en un documento de MongoDB puede ser indexado, al igual que es posible hacer índices secundarios. El concepto de índices en MongoDB es similar al empleado en base de datos relacionales
- **Replicación**: soporta el tipo de replicación primario-secundario. Cada grupo comparte la misma información y los secundarios se denominan *replica set*. El grupo primario puede ejecutar comandos de lectura y escritura mientras que el secundario únicamente de lectura.
- **Balanceo de cargas**: puede escalar de forma horizontal ya que tiene la capacidad de ejecutarse en múltiples servidores, balanceando la carga y/o replicando los datos para poder mantener el sistema funcionando en caso de que exista un fallo de *Hardware*.

La estructura de almacenamiento de los datos se realiza a través de colecciones y bajo el esquema de documentos. Lo anterior permite tener flexibilidad en cuanto a la carencia de la definición de un esquema o campos requeridos para almacenar para una entidad del mismo tipo. [42]

2.6. Sistemas de Control de Versiones

Un Sistema de Control de Versiones (SCV) registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo de manera que es posible recuperar versiones específicas en el futuro.

En programación comúnmente se utiliza para hacer un seguimiento de la evolución del código fuente del software, aunque en realidad puede realizarse con casi cualquier tipo de archivo en una computadora.

Utilizar estos sistemas permite tener un historial de todos los cambios sobre un proyecto y la habilidad de desplazarse entre diferentes versiones que han existido, lo que posibilita incluso recuperar archivos si son borrados por equivocación. [43]

De manera general existen distintos tipos de SCV:

- Locales: contienen una Base de Datos sencilla que contiene el registro de los cambios realizados. Presentan el problema al trabajar de manera colaborativa en los proyectos. Un ejemplo de este tipo de SCV es RCS.
- Centralizados: tienen un servidor único con todos los archivos versionados y con distintos clientes que descargan los archivos a su entorno local. Soluciona el problema de trabajo en equipo, pero es vulnerable a fallos al ser un servidor centralizado. Ejemplos de este tipo son: CVS, Subversion y Perforce.
- Distribuidos: soluciona las carencias de los dos tipos anteriores, de manera que los clientes no sólo descargan la última copia de los archivos, sino que se hace una réplica exacta de todo el repositorio. Ejemplos de este tipo son: Git, Mercurial, Bazaar y Darcs.

2.6.1. Git

Git es un SCV distribuido *Open Source* distribuido bajo los términos de la *GNU General Public License* versión 2. Fue creado por Linus Torvalds en 2005 para el control del proceso de desarrollo y actualización del kernel de GNU/Linux; actualmente es el SCV más famoso y usado. Posee una gran comunidad de apoyo y soporte, así como una gran documentación y libros. [44]

Git es un sistema distribuido de control de versiones para rastrear cambios en el código fuente durante el desarrollo de software. Está diseñado para coordinar el trabajo entre programadores, aunque puede usarse para rastrear cambios en cualquier conjunto de archivos.

Sus principales ventajas respecto a otros SCV son la velocidad, seguridad de datos, flexibilidad y soporte para flujos de trabajo no lineales distribuidos. Mientras que su desventaja respecto a otros sistemas es que puede ser un *Software* complejo de aprender. [45]

2.6.2. GitHub

GitHub es un Sistema de Control de Versiones basado en la nube, es decir, es un servicio de nube que permite el almacenamiento y administración de repositorios de proyectos que emplean Git como SCV, lo que permite tener un seguimiento y control de los cambios a los archivos.

Sus características permiten que el trabajo de proyectos individuales o colaborativos y profesionales o educativos sea muy fácil, debido a que posee una interfaz amigable y una gran integración de aplicaciones que hacen eficiente la metodología de trabajo. [46]

2.7 Pruebas de *Software*

Las pruebas en software son una actividad para corroborar que los resultados obtenidos coinciden con los resultados esperados en una aplicación y para garantizar que no hay *bugs* en la misma. Esto permite identificar errores o deficiencias de los requerimientos y pueden ser realizados de forma manual o automática.

La aplicación de pruebas de software es importante porque permite detectar de manera temprana errores que pueden ser costosos y peligrosos. Por lo que al aplicar estas pruebas se obtienen los siguientes beneficios:

- **Reducción de costos:** al prevenir futuros errores se puede tener un ahorro al evitar reparaciones o evitando pérdidas económicas, dependiendo el fin del producto.
- **Seguridad:** se obtiene al remover de manera temprana vulnerabilidades y posibles problemas.
- **Calidad del producto:** al realizar pruebas se garantiza que no hay errores y se incrementa la calidad del producto.
- **Satisfacción del cliente:** se logra previniendo fallas, generando una mejor *User Interface* (UI) y *User Xperience* (UX).

Los tipos de prueba más comunes son los siguientes: [47]

- **Pruebas de funcionalidad**

    - Pruebas unitarias
    - Pruebas de integración
    - Pruebas de Aceptación del Usuario (UAT)
    - Pruebas de localización
    - Pruebas de interoperabilidad
- **Pruebas de no funcionalidad**
    - Pruebas de desempeño
    - Pruebas de resistencia
    - Pruebas de carga
    - Pruebas de escalabilidad
    - Pruebas de usabilidad
- **Pruebas de mantenimiento**
    - Pruebas de regresión
    - Pruebas de mantenimiento

2.7.1 Pruebas unitarias

Las *Unit Testing* o pruebas unitarias es un tipo de prueba de *Software* donde se prueban unidades o componentes de software de manera aislada. Su propósito es validar que cada unidad se comporta como se espera.

Este tipo de pruebas suelen ser las más sencillas y las primeras a realizar de todas las pruebas existentes. Su principio se basa en probar pequeños componentes para tener una granularidad de estos y poder garantizar su funcionamiento y reutilización.

La realización de estas pruebas es importante para garantizar el funcionamiento correcto al desarrollar o modificar unidades, las cuales pueden ser una función, método, procedimiento, módulo u objeto.

2.7.2 Pruebas *End-To-End*

Las pruebas *End-To-End* (E2E) o de extremo a extremo también son conocidas como pruebas de integración y tienen el propósito de buscar validaciones en el *Software* incluyendo todas sus integraciones y replicar las condiciones de un entorno productivo con configuraciones y datos reales.

Este tipo de pruebas suelen aplicarse posterior a las pruebas unitarias, ya que las anteriores garantizan el funcionamiento correcto de manera aislada de cada componente, mientras que las pruebas de

integración garantizan la obtención de los resultados esperados de los componentes de manera conjunta.

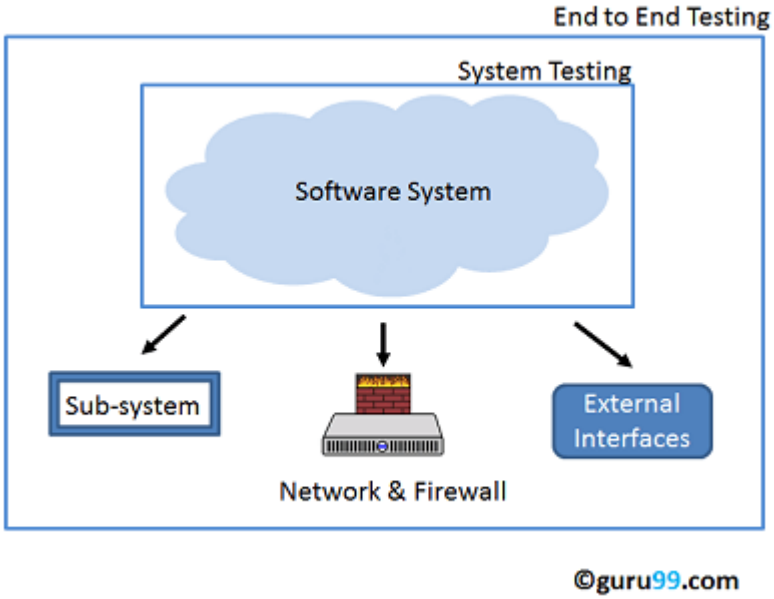


Figura 2.14. Funcionamiento de las pruebas E2E.[19]

[19] Fuente: https://www.guru99.com/end-to-end-testing.html

CAPÍTULO 3. ANÁLISIS Y PLANIFICACIÓN DEL PROYECTO

3.1. Análisis de requerimientos

Inicialmente, la Jefatura de la División de Ingenierías Civil y Geomática (DICyG) de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México (UNAM) se encarga de solicitar a cada uno de los departamentos que conforman a la División su información de actividades realizadas durante cada periodo trimestral como se describió en la Sección 1.2. Las actividades reportadas por cada departamento se clasifican en cada uno de los trece apartados o rubros, que se listan a continuación:

- A21: Cursos disciplinares, cursos de cómputo, conferencias y talleres extracurriculares e intersemestrales impartidos.
- A31: Alumnos atendidos en cursos disciplinares, cursos de cómputo, conferencias y talleres extracurriculares e intersemestrales impartidos.
- A22: Programas de apoyo a la disminución del rezago académico y recuperación de los estudiantes irregulares.
- A32: Alumnos atendidos en programas de apoyo a la disminución del rezago académico y recuperación de los estudiantes irregulares.
- A61: Personal académico de carrera que participa en programas de actualización y superación académica.
- A62: Personal académico de asignatura que participa en programas de actualización y superación académica.
- A73: Actividades académicas de reforzamiento realizadas: prácticas, laboratorios, concursos nacionales e internacionales.
- A83: Asistentes de actividades académicas de reforzamiento realizadas: prácticas, laboratorios, concursos nacionales e internacionales.
- A74: Promoción de la oferta sociocultural, deportiva y recreativa realizada. Son los medios de difusión y comunicación usados para promover las actividades deportivas y socioculturales en los que la Entidad participe.
- A91: Material de apoyo didáctico impreso que contribuya a hacer más eficiente el proceso de enseñanza-aprendizaje.
- A94: Tiraje de publicaciones realizadas. Ejemplares producidos periódica o unitaria.

- A92: Material de apoyo didáctico digitalizado, disponible en medios de almacenamiento ópticos o magnéticos, que contribuya a hacer más eficiente el proceso de enseñanza-aprendizaje.
- A93: Material de apoyo didáctico en línea. Material de apoyo didáctico disponible en plataformas computacionales para apoyar el proceso de enseñanza-aprendizaje.

Al finalizar la recaudación de datos, la Jefatura de la DICyG se encarga de realizar un concentrado con la información de todos los departamentos de cada uno de los apartados. Dicho reporte se entrega a la Facultad de Ingeniería para el proceso de captura de la Matriz de Índice de Resultados (MIR) a nivel Facultad.

Este proceso se realizaba a través de un proceso manual, el cual fue descrito en el Capítulo 1, donde la Secretaría Técnica de la División procesaba y concentraba toda la información a través de documentos escritos y con el software Microsoft Office Excel.

Al ser un proceso repetitivo que solicita información del mismo tipo cada trimestre, es posible automatizar todo este proceso de la MIR de la División a través de un sistema web que agilice tiempos, reduzca esfuerzos y aumente la disponibilidad, gestión y transparencia de la información. Para lograrlo se definen roles de usuario, diferentes actividades y generación de reportes mediante archivos PDF y estadísticas.

Las diferentes secciones con las que se contará en el sistema dependen del rol para interactuar con la información; estas secciones se describen a continuación en función de los dos roles.

Para el rol de usuario normal se tienen las siguientes secciones:

- **Inicio**: vista principal e informativa.
- **Capturar**: sección en la que pueden capturar los registros del periodo actual.
- **Revisar**: sección en la que pueden consultar y actualizar registros capturados previamente para el registro actual.
- **Capturar esperados del año**: sección en la que cada uno de los usuarios captura toda la información referente a los registros que se esperan capturar para el siguiente año al que se está usando el sistema.
- **Estadísticas**: sección en la que se puede consultar la información relativa al desempeño esperado contra el obtenido.

- **PDF**: sección donde pueden generar un reporte en PDF de sus actividades reportadas para cada apartado.

Por otra parte, el rol de usuario administrador contiene todas las secciones del usuario normal, pero con la capacidad de visualizar el contenido de todos los departamentos y además agrega los siguientes módulos administrativos:

- **Funcionarios**: módulo en el que es posible gestionar a los usuarios (jefes de Departamentos) y hacer movimientos del departamento asignado a cada uno.
- **Usuarios**: Este módulo se encarga de la gestión de disponibilidad de acceso al sistema a todos los usuarios o solamente algunos.
- **Calendario**: En esta vista se pueden dar de alta un nuevo calendario anual, modificarlo y se permite gestionar los periodos.

La Figura 3.1 ilustra los actores y casos de uso que se identificaron del análisis de requerimientos anterior, lo que permite tener una idea general de las reglas de negocio y alcance del sistema web a desarrollar.

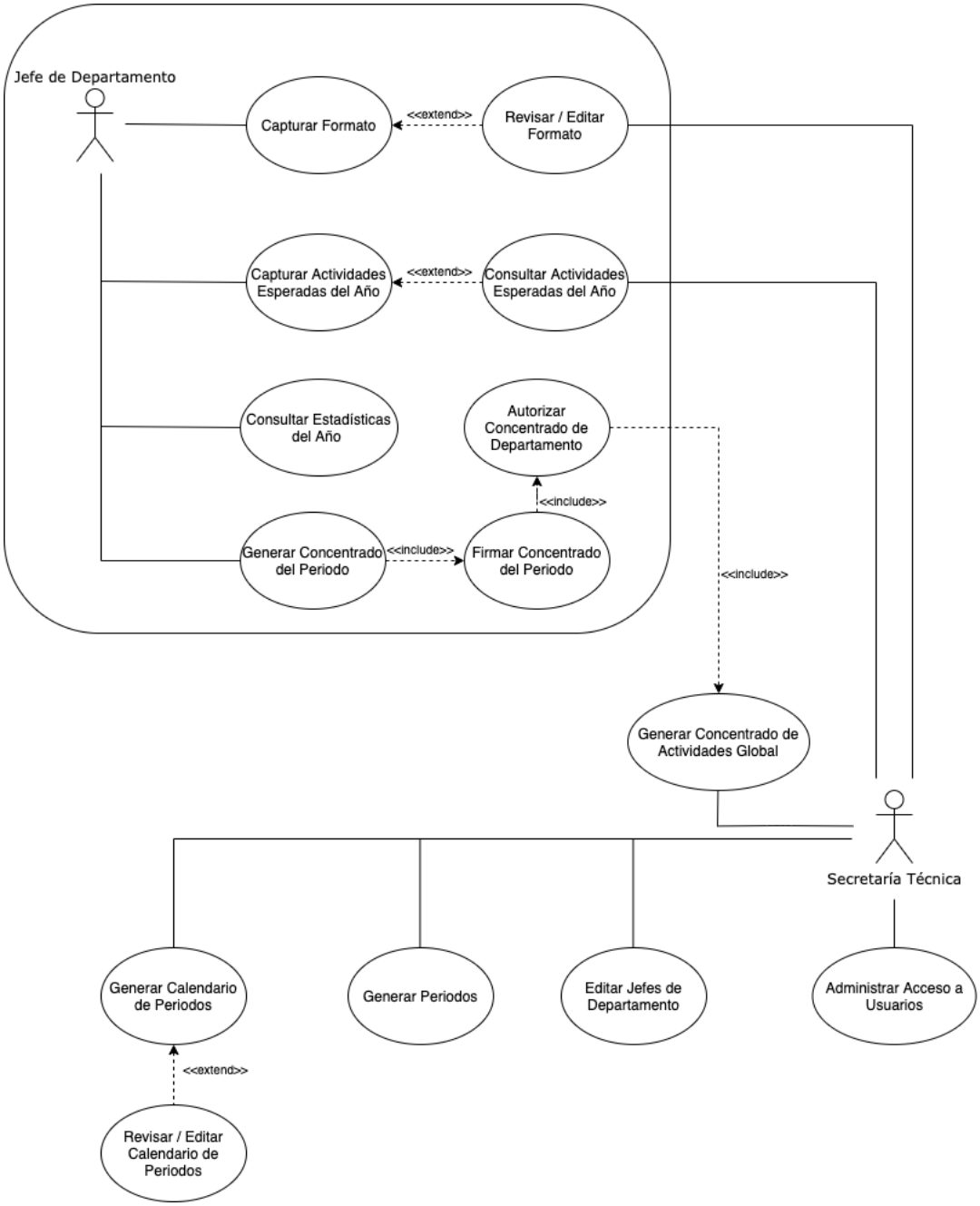


Figura 3.1. Diagrama UML de casos de uso.

3.1.1. Requerimientos funcionales

Se requiere un control de acceso, donde se provea un módulo de *login* solicitando credenciales de acceso para su autenticación, asignación de una identidad y rol en el sistema, con lo que es posible identificar los movimientos que realice.

Cada usuario podrá ser un jefe de departamento de los que conforman a la División. También se tendrá un usuario administrador, para la Secretaría Técnica, a la cual se le proveerán sus credenciales y podrá tener acceso para realizar tareas de gestión del sistema.

Se requiere crear un formulario para cada uno de los trece apartados que se solicitan en la MIR con sus respectivos campos requeridos. Estos formularios estarán disponibles tanto para sección de Captura de nuevos registros, así como en el módulo Revisar de la información previamente capturada.

Es necesaria la generación de reportes en formato PDF, bajo cierta estructura y con los nombres a quienes se dirigen, según sea el caso. Además, deberán contener una firma electrónica que permita determinar su validez por parte de las partes interesadas, es decir, Jefes de Departamentos y Jefatura de la DICyG.

Los módulos de gestión, que son Funcionarios, Usuarios y Calendario, deben estar disponibles y accesibles únicamente para los usuarios con los privilegios del rol administrador.

La información deberá ser confidencial en función de los roles de cada involucrado y totalmente restringida para entes externos a la División. Cada departamento será capaz de capturar, ver y modificar información del periodo en turno de su propio departamento y no de otros. El rol administrador será capaz de visualizar la información de todos los departamentos y todos los periodos.

El sistema debe ser capaz de generar informes en PDF que tengan un formato estándar que recabe la información de cada apartado de cada Departamento, el cual tendrá validez e interés a nivel interno o de División. Mientras que también se debe generar reportes que concentren toda la información de todos los Departamentos, los cuales serán de interés a nivel externo pues son los que se entregarán de manera formal a la Facultad.

Dado que el proceso del sistema y la generación de reportes será de manera digital se deberá implementar un sistema que asegure la validez y aprobación de los reportes por los jefes de Departamento y por el Jefe de la división, que otorguen la validez de las firmas autógrafas.

3.1.2. Requerimientos no funcionales

El sistema debe tener una interfaz amigable y minimalista, con la información espacialmente distribuida para maximizar la claridad de los pasos e información solicitada.

Debe ser de fácil uso, enfocado principalmente a usuarios que no tengan necesariamente conocimiento alguno del proceso de la MIR y que tampoco cuenten con experiencia en el manejo y operación de sistemas web.

Se debe facilitar el proceso de captura, concentrado y aprobación de información para todos los roles en el sistema. Al ser tareas repetitivas, se debe automatizar la mayor cantidad de subprocesos posibles.

Es necesario que exista información descriptiva de cada paso a realizar para los usuarios, así como de las descripciones de cada sección, apartado o campo a capturar en algún formulario. No debe prestarse a diversas interpretaciones.

El sistema deberá garantizar su disponibilidad durante los periodos de captura que se llevan a cabo cada trimestre. Se considera que puedan capturar formatos simultáneamente todos los usuarios del sistema, que son los Jefes de Departamento. Además, se garantizará su disponibilidad para consulta por parte de la Secretaría Técnica en todo momento.

En caso de presentarse algún inconveniente o interrupción, se restaurará el sistema lo más pronto posible. Sin embargo, al tener una baja demanda de uso, no resulta prioritario contar con servidores de respaldo fuera del área de cómputo de la División.

3.2. Análisis del alcance

El proceso que lleva a cabo la DICyG para la MIR abarca todo el proceso administrativo de manejo de la información, es decir, está enfocado en la Secretaría Técnica. Este proceso está dirigido a la recaudación y gestión de información de los diferentes Departamentos con los que cuenta la División:

1. Construcción
2. Sistemas, Planeación y Transporte
3. Topografía
4. Sanitaria y Ambiental
5. Hidráulica
6. Geotecnia

7. Estructuras
8. Geodesia y Fotogrametría
9. Computación
10. Coordinación de Especialidades

Cada Departamento actúa bajo sus propios programas y planificaciones, dentro de lo acordado por la propia DICyG, por lo que es responsabilidad de cada Departamento el crear mecanismos para recabar la información de todas sus actividades trimestrales para reportarlas a la MIR de la División.

El proceso inicia por la Secretaría Técnica de la Jefatura de la División al solicitar a cada uno de los Departamentos la declaración de todas sus actividades desempeñadas y planificadas, y finaliza con la validación por parte del Jefe de la División al término del trimestre de actividades para formarse el informe de la Facultad de Ingeniería.

El sistema web es un facilitador del proceso, permitirá agilizar tareas repetitivas que pueden ser programadas, con la finalidad de disminuir el esfuerzo por parte de todos los involucrados en el sistema y aumentar la disponibilidad, confidencialidad y transparencia de información; así como la formación de históricos de datos que permitan generar estadísticas descriptivas que permitan mejorar la toma de decisiones de la División.

Por lo tanto, el sistema automatizará una parte del proceso, iniciando con la solicitud de información de la Secretaría Técnica y apertura del sistema a los usuarios, hasta la generación de reportes en PDF con la validación y visto bueno de la Secretaría Técnica y el Jefe de la División, con lo que se pueda garantizar la validez de los informes.

3.3. Diagrama de actividades

Las actividades que componen el proceso de la MIR de la DICyG se describen en esta sección con un diagrama UML, específicamente un diagrama de actividades que describe de manera visual el evento que dispara el proceso, el flujo de actividades, su finalización y las partes involucradas.

En la Figura 3.2. se observa que el evento que inicia el proceso de la MIR de la División es la solicitud de la información trimestral de los Departamentos por parte de la Secretaría Técnica. Todas las actividades involucran a las siguientes tres entidades:

1. Jefe de la División.

2. Secretaría Técnica.
3. Departamentos de la División

El proceso finaliza con la firma de autorización y visto bueno por parte del Jefe de la División. Es en ese momento cuando los informes son posibles de enviarse a la Alta Dirección de la Facultad de Ingeniería para registrar el cumplimiento de esta actividad de la DICyG.

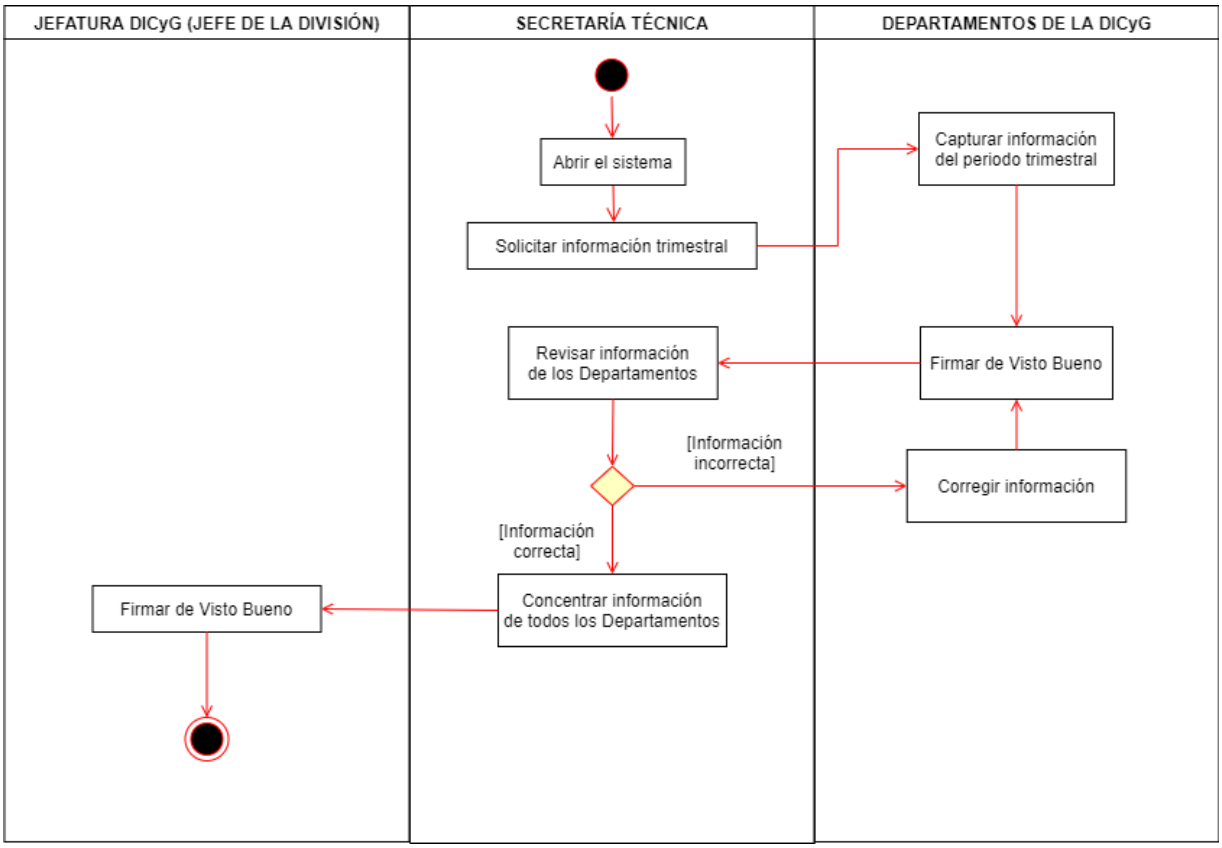


Figura 3.2. Diagrama de actividades del proceso de la MIR de la DICyG.

3.4. Propuesta del *Frontend*

Otra parte fundamental en un sistema web, consiste en la parte tangible de la aplicación, es decir, con la visualización de la información a un usuario final. Esta parte con la que el usuario puede interactuar y hacer peticiones se conoce como *frontend*.

El *frontend* puede desarrollarse empleando las tecnologías clásicas de HTML5 para la estructura lógica del sitio, CSS3 para el ajuste de elementos visuales y JavaScript para agregar dinamismo, aunque existen *frameworks* que facilitan esta tarea como se mencionó en la Sección 2.3.2 y 2.3.5.

Por la naturaleza del proyecto, cualquiera de los *frameworks* mencionados en la Sección 2.3.5 son convenientes para el desarrollo del proyecto. Sin embargo, por la experiencia previa con Angular, la estabilidad que ha demostrado, la facilidad de actualización entre versiones, el mantenimiento de los módulos más esenciales, por parte de su equipo y su seguridad, se optó por utilizar éste. También, posee una robusta documentación, facilidad de integración en proyectos, adopción de estándares al implementar de manera nativa el lenguaje *TypeScript*, que potencia el uso de JavaScript al agregar tipado estático y evitar futuros errores en el desarrollo.

Otras características que favorecieron el uso de Angular son sus características por defecto, las cuales se encuentran listas para implementarse, tales como: inyección de dependencias e integración de servicios, como se mencionó en la Sección 2.3.5.

De acuerdo con los requerimientos solicitados y a la adopción de Angular, se propone la construcción del sistema como una *Single Page Application* (SPA) moderna y capaz de comunicarse de manera segura por medio de servicios que sean capaces de procesar peticiones de datos bajo *JavaScript Object Notation* (JSON), que es efectivo para el intercambio de recursos con el *backend* por medio de una API.

El *frontend* del sistema web se fragmentará en secciones o módulos, que permitan identificar a los usuarios las distintas unidades de trabajo con las que podrán interactuar en el sistema. Los siguientes módulos serán accesibles por los usuarios bajo un rol normal, es decir, para los jefes de Departamentos:

- **Iniciar sesión:** primera vista en la que se mostrará información de los Calendarios del sistema para el año en curso. Además de solicitar las credenciales de autenticación para entrar al sistema una vez que se ha autenticado.
- **Inicio:** vista principal e informativa del sistema.
- **Capturar:** sección en la que es posible capturar los diferentes formatos o actividades del periodo actual.
- **Revisar:** sección en la que se puede consultar y actualizar registros capturados previamente para el periodo actual.
- **Esperados del año:** módulo en el que cada uno de los usuarios captura toda la información referente a los registros que se esperan capturar para el siguiente año al que se está usando

el sistema. Esta información permitirá hacer un contraste de resultados y rendimiento de cada uno de los Departamentos.

- **Estadísticas:** sección en la que se puede consultar la información relativa al desempeño esperado contra el obtenido.
- **PDF:** módulo en el que se pueden generar los reportes de actividades en PDF para cada apartado.

Los usuarios con el rol de administrador podrán observar la información relativa a todos los Departamentos en todos los módulos anteriores; con ello, podrá revisar y dar seguimiento a cada Departamento para la toma de decisiones por parte de la Jefatura de la División.

Además de los módulos anteriores, los usuarios administradores tendrán acceso a los siguientes:

- **Funcionarios:** módulo en el que es posible gestionar a los usuarios (jefes de departamentos) y hacer movimientos del departamento asignado a cada uno.
- **Usuarios:** módulo encargado de la gestión de disponibilidad de acceso al sistema a los usuarios de rol estándar que se deseen.

3.5. Propuesta de *Backend*

Un sistema web puede realizar conexiones a una Base de Datos (BD) existente, a través de un método de autenticación como el tener el usuario y contraseña de la Base de Datos a la que desea conectarse. Este método de acceso debe ser controlado por un lenguaje de programación de *Backend*, como los descritos en la Sección 2.3.3.

Sin embargo, debido a que las aplicaciones web modernas requieren de un proceso más robusto de construcción y proveer métodos que garanticen la mantenibilidad y escalabilidad efectiva, se determinó hacer uso de un *framework* de *backend* para el desarrollo de este sistema. Los *frameworks* a considerar fueron los mencionados en la Sección 2.3.6, de los que se determinó la elección de Django para Python, debido a la gran simplicidad que ofrece para la creación de una aplicación web, la extensa diversidad de bibliotecas que existen de Python que pueden complementar el desarrollo de funcionalidades y, otro importante factor fue la excelente documentación que posee en su sitio web oficial: https://docs.djangoproject.com/

Para este proyecto se requiere que el *backend* de la aplicación se comunique con la Base de Datos para obtener los datos, los roles de usuario se administraran desde Django. Además, se deberá ofrecer

comunicación entre el *frontend* y el *backend* por medio de una *Application Programming Interface* (API) para consumir sus recursos.

La mejor propuesta para proveer las características anteriores es hacer una *Representational State Transfer API* (RESTful API) con control de accesos para la consulta y escritura de recursos. Esto será posible de adecuarse con Django dado que existen bibliotecas que ofrecen una capa de abstracción mayor de los datos para proveerlos a través de una RESTful API, como *Django Rest Framework* (DRF).

Los detalles técnicos del uso e implementación de Django y DFR se describen en el Capítulo 4 de este trabajo.

Debido a que los requerimientos descritos en la Sección 3.1 que detallan el uso de cada uno de los apartados de actividades a llenar en la MIR, se observó que muchos pares de apartados comparten muchas características en común, con la diferencia de tener un campo adicional o distinto.

La identificación de esta particularidad permite proponer el manejo lógico de los datos de los apartados en pares, en los casos que se comparten dichas características. Esto podrá facilitar el control de la información y evitará su repetición, así como el garantizar la consistencia en la BD.

Esta fusión será tratada de manera interna. La información correspondiente a cada formato podrá ser accesible individualmente al momento de capturar y revisar para cada Departamento de la División. De igual manera, el concentrado de actividades se generará de forma individual. Como resultado de esta unión de actividades, se manejará el siguiente listado de actividades:

- A21/A31
- A22/A32
- A61/A62
- A73/A83
- A74
- A91/A94
- A92
- A93

Con el uso de los *frameworks* Django y DRF para el *backend*, será posible crear métodos que permitan consumir y modificar los datos. Estos métodos serán los que permitan conectar los datos con los definidos en los modelos de la BD y serán los siguientes:

- **Formatos:** será la forma lógica en la que se operarán las diferentes actividades agrupadas.
    - A2131
    - A2232
    - A6162
    - A7383
    - A74
    - A9194
    - A92
    - A93
- **Divisiones:** corresponderá con la relación de la BD de mayor jerarquía que identifica de qué división son los datos. Para este sistema web, bastará con tener a la DICyG, pero permite la escalabilidad de la aplicación.
- **Departamentos:** mapeará los datos de cada división en sus subdependencias, que son las encargadas de rendir la información de sus actividades trimestrales.
- **Usuarios:** permitirá acceder a los datos de los usuarios (Jefes de Departamentos) registrados en la BD para que puedan acceder al sistema bajo el rol de usuario normal o estándar.
- **Periodos:** proveerá los diferentes periodos, que fungirán como ventanas de tiempo que permitan la localización temporal de los datos registrados.
- **Apartados:** será empleada como datos de sólo lectura, pues serán informativos dando un resumen de cada apartado de actividad existente y su descripción.
- **Firmas:** será la interfaz de generación y actualización de registros únicos que validen la información de manera electrónica, por los jefes de Departamentos y el jefe de la División, en los reportes de actividades en PDF.
- **Profesores:** corresponderá con los registros creados en la BD para los profesores de la Facultad de Ingeniería (FI) que hayan sido involucrados en las actividades reportadas.
- **Profesores Externos:** tendrá la misma función que el punto anterior, pero para profesores externos a la FI.
- **Calendario:** permitirá acceder y modificar los datos que hagan referencia a los calendarios en que deberán acceder los usuarios al sistema para registrar sus actividades. La lectura de este apartado será pública, para permitir mostrar la información incluso antes de iniciar sesión o identificarse en el sistema.
- **Totales Esperados:** será la conexión con los datos que correspondan al módulo de números de actividades que se proyectan tener para el siguiente año al que esté en curso.

3.6. Propuesta de la Base de Datos

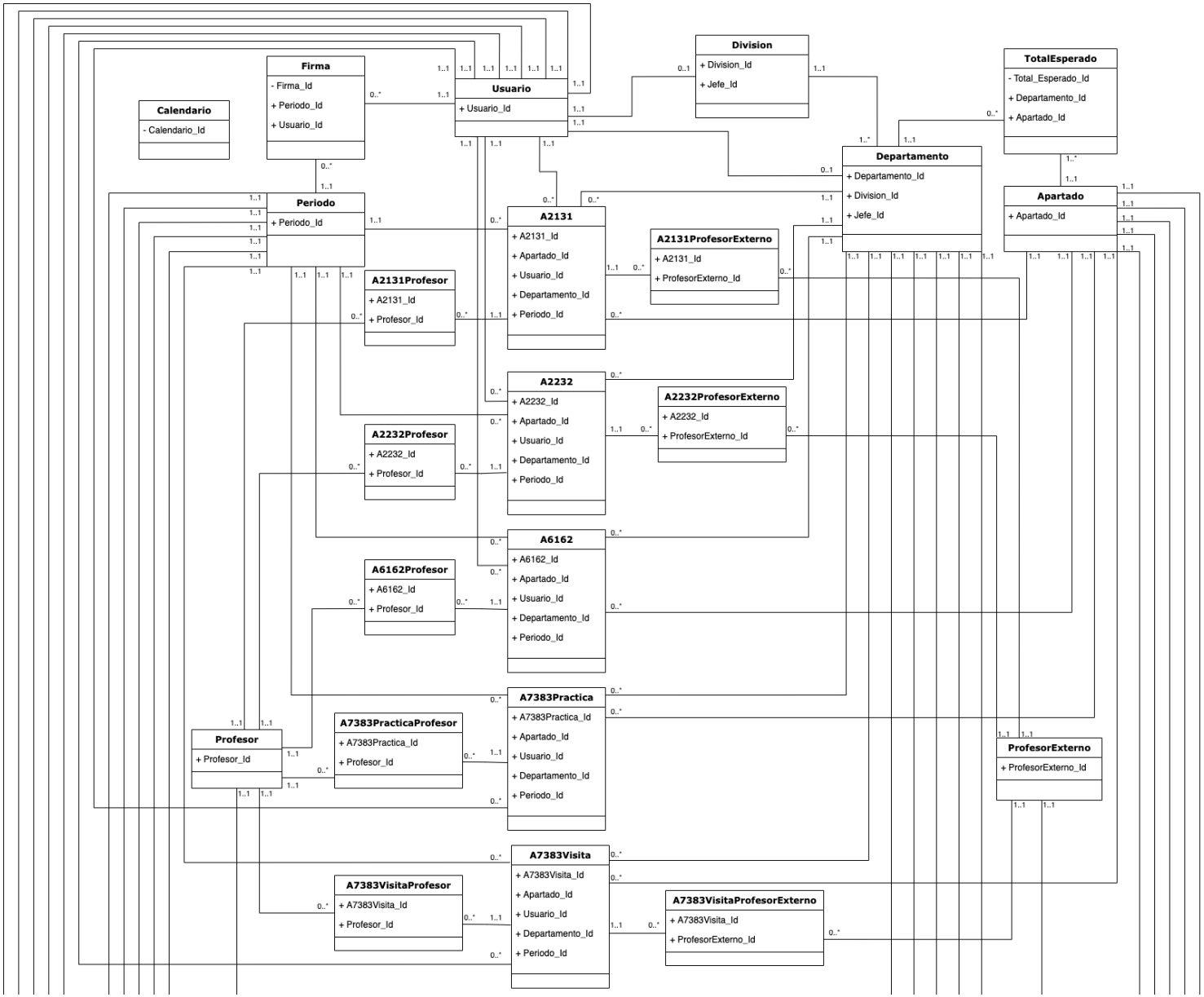
La construcción de un sistema web que sea capaz de cumplir con todos los requerimientos de los usuarios y garantizar las mejores prácticas, requiere el uso de una Base de Datos para cumplir con la consistencia, durabilidad y seguridad de las grandes cantidades de información que puedan generarse.

En el Capítulo 2 se mencionaron las opciones viables de *Relational DataBase Management Systems* (RDBMS) para sistemas web y debido a sus características de alto rendimiento, cumplimiento con estándares SQL, seguridad, independencia de un propietario privativo (*open source*), su apoyo y gran comunidad con que cuenta, la mejor opción para usar es el RDBMS PostgreSQL.

Para el modelado de la base de datos es importante conocer las relaciones que existen entre los diferentes módulos y tomar en consideración los requerimientos funcionales y no funcionales, por lo que es posible identificar la necesidad de contar con los siguientes modelos de datos:

- Formatos: A21, A31, A22, A32, A61, A62, A73, A83, A74, A91, A94, A92, A93
- Divisiones
- Departamentos
- Usuarios
- Periodos
- Apartados
- Firmas
- Profesores
- Profesores Externos
- Calendario
- Totales Esperados

La Figura 3.3 muestra el diagrama de clases de las entidades de la Base de Datos que cumple con los datos requeridos para su almacenamiento. Lo anterior se derivó del análisis realizado en la Sección 3.5.


Calendario
- Calendario_Id
Firma
- Firma_Id
+ Periodo_Id
+ Usuario_Id
Usuario
+ Usuario_Id
Division
+ Division_Id
+ Jefe_Id
TotalEsperado
- Total_Esperado_Id
+ Departamento_Id
+ Apartado_Id
Departamento
+ Departamento_Id
+ Division_Id
+ Jefe_Id
Apartado
+ Apartado_Id
Periodo
+ Periodo_Id
A2131
+ A2131_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A2131ProfesorExterno
+ A2131_Id
+ ProfesorExterno_Id
A2131Profesor
+ A2131_Id
+ Profesor_Id
A2232
+ A2232_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A2232ProfesorExterno
+ A2232_Id
+ ProfesorExterno_Id
A2232Profesor
+ A2232_Id
+ Profesor_Id
A6162
+ A6162_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A6162Profesor
+ A6162_Id
+ Profesor_Id
A7383Practica
+ A7383Practica_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
Profesor
+ Profesor_Id
A7383PracticaProfesor
+ A7383Practica_Id
+ Profesor_Id
ProfesorExterno
+ ProfesorExterno_Id
A7383Visita
+ A7383Visita_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A7383VisitaProfesor
+ A7383Visita_Id
+ Profesor_Id
A7383VisitaProfesorExterno
+ A7383Visita_Id
+ ProfesorExterno_Id

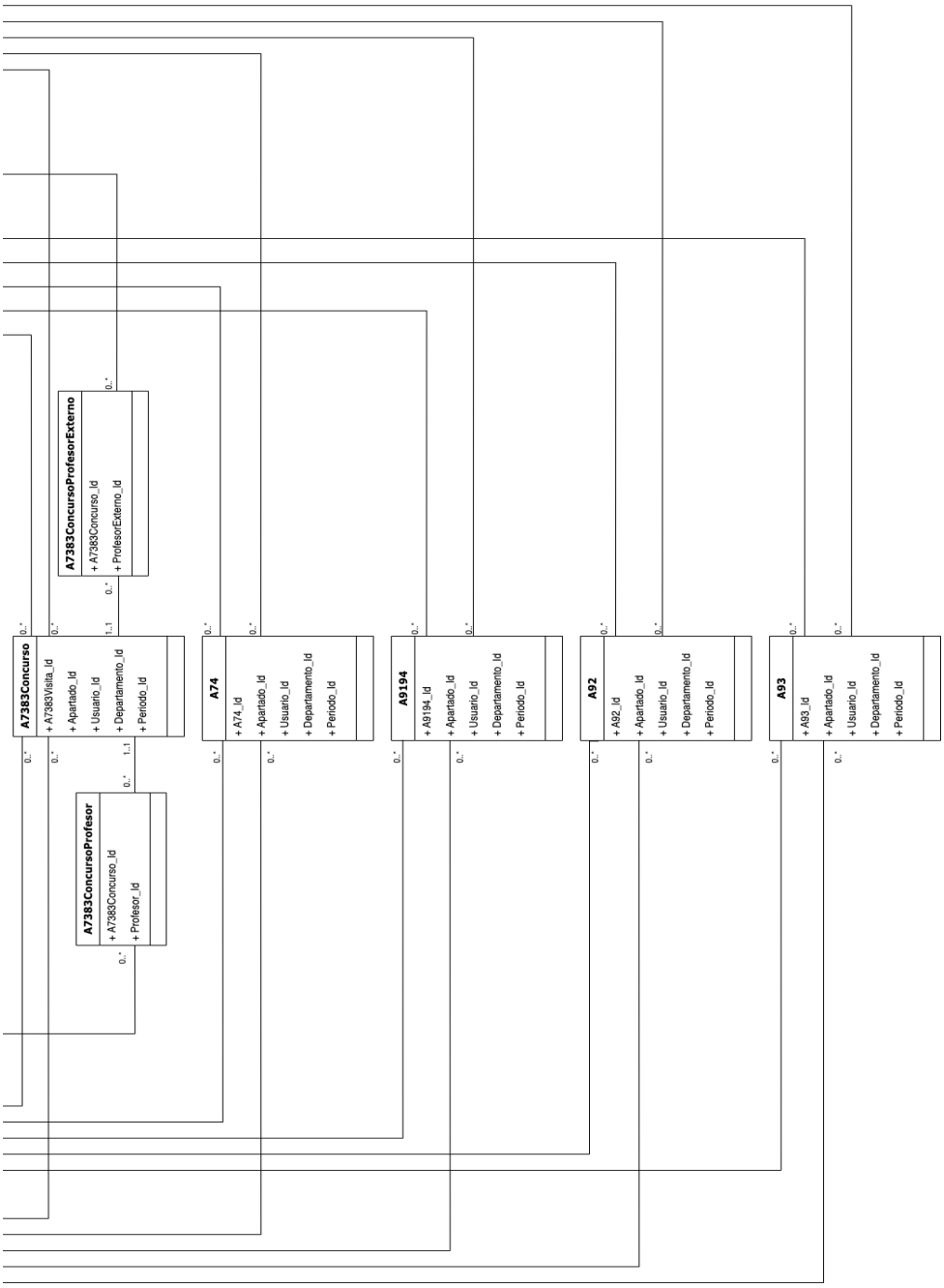


Figura 3.3. Diagrama de clases de la Base de Datos para la MIR de la DICyG.[20]

[20] Por motivos de seguridad, el diagrama de clases muestra sólo los campos esenciales para visualizar la estructura y relación de entidades. La información detallada está plasmada en el Modelo Relacional de la Base de Datos, mismo que se encuentra bajo resguardo de la M.I. Tanya Itzel Arteaga Ricci, Jefa de la Unidad de Cómputo de la DICyG.

Se debe poder generar formatos con sus diferentes atributos de manera que se puedan relacionar con el usuario que lo creó, el apartado que corresponde, el periodo en que se capturó, los profesores involucrados y poder realizar consultas de manera independiente a cualquiera de las instancias antes mencionadas.

La Base de Datos contendrá únicamente a un usuario administrador, el cual será gestionado por el *framework* Django. Debido a esto no se requiere tener diferentes roles de usuario a nivel Base de Datos, ya que los permisos de lectura y escritura están manejados a nivel aplicación.

CAPÍTULO 4. IMPLEMENTACIÓN DEL PROYECTO

En el capítulo anterior se detallaron las propuestas de solución a los requerimientos y necesidades que se llevarán a cabo en el sistema web que se desarrollará. En este capítulo se detallará a nivel técnico cada una de las fases implementadas en el servidor, la Base de Datos, el *backend* y el *frontend* de la aplicación.

De manera general, la arquitectura de la aplicación empleando el servidor web NGNIX, el RDBMS PostgreSQL, Django y *Django Rest Framework* (DRF) para *backend* y Angular para *frontend*, como se muestra en la Figura 4.1.

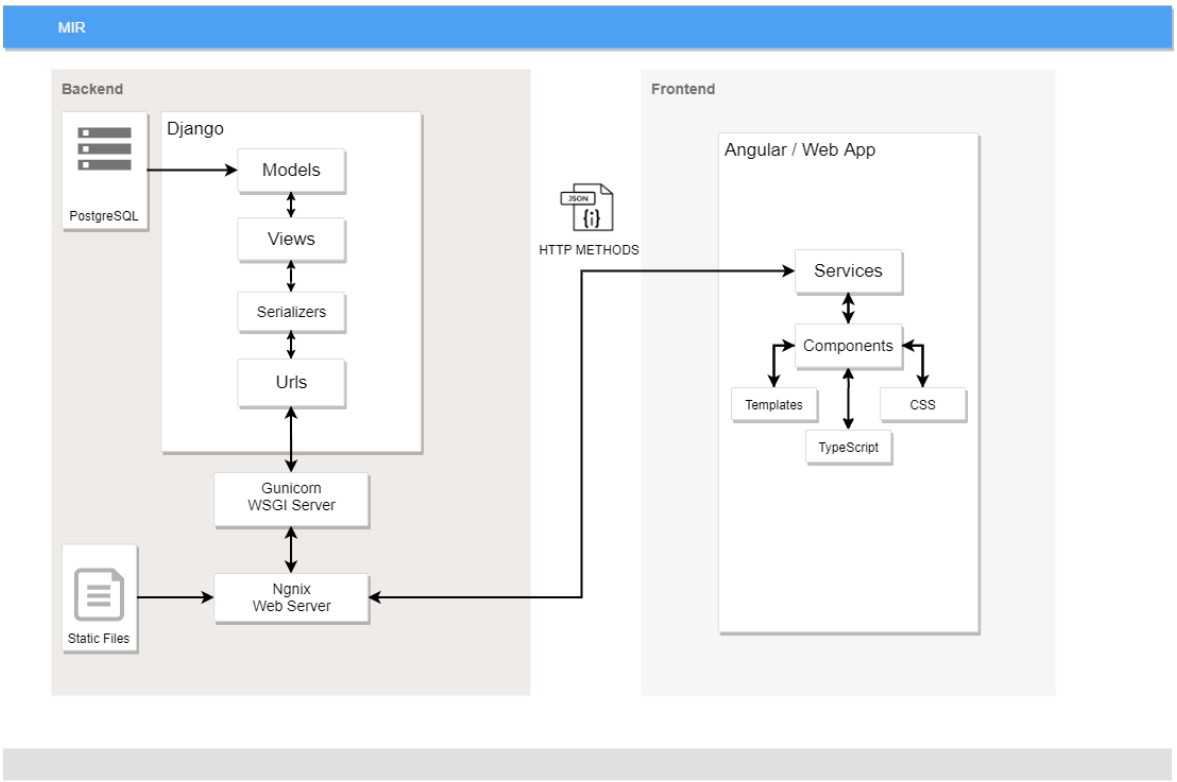


Figura 4.1. Arquitectura general de la aplicación web "mirapp".

4.1. Infraestructura

El sistema web creado se alojó en uno de los servidores de la DICyG, para garantizar su control y gestión por parte de esta División a través de la Unidad de Cómputo.

Debido a su alta confiabilidad y sencillez, se optó por usar un servidor GNU/Linux, específicamente la distribución Debian, con las siguientes características:

- **Sistema Operativo**
  - **Nombre del servidor:** mirserver
  - **Sistema Operativo:** Debian GNU/Linux 9.6 (stretch)
  - **Disco duro virtual:** 28 GiB
  - **Memoria RAM:** 1 GB
  - **Arquitectura:** x86_64
  - **Orden de bytes:** *Little Endian*
  - **CPU(s):** 1
  - **Kernel:** Linux 4.9.0-6-amd64
- **Base de Datos**
  - PostgreSQL 9.6.10
- **Servidor Web**
  - nginx 1.10.3
  - gunicorn 19.9.0
- **Software**
  - Python 3.5.3
  - Django versión 1.11.6 LTS
  - Angular versión 8.2

Debido a las características y software requerido para el funcionamiento del sistema web, se optó por usar un servidor virtual dedicado para este proyecto. La Jefa de la Unidad de Cómputo de la DICyG, la M.I. Tanya Itzel Arteaga Ricci, fue quien creó este servidor web virtual y proporcionó el acceso al mismo a través de SSH. Las direcciones del servidor son las siguientes:

IP interna: 192.168.xxx.yyy

IP externa: 132.248.xxx.yyy

Una vez que se tuvo el acceso al servidor, fue posible hacer la instalación del software, servidor web y la Base de Datos que se describieron anteriormente.

4.2. Implementación en Software

4.2.1. Base de Datos

Como se mencionó en el Capítulo 3, se optó por utilizar el *framework* Django, debido a muchas ventajas que ofrece como el uso de bibliotecas de Python. Otro factor importante fue que tiene de manera nativa un *Object-Relational Mapping* (ORM), el cual permite hacer un mapeo de los de los recursos de una BD relacional, como la seleccionada en este proyecto (PostgreSQL), a una estructura de objetos del paradigma de *Object-Oriented Programming* (OOP).

Lo anterior permite tener un control más eficiente de los datos a través de una interfaz, como las APIs, y permite gestionar y controlar los esquemas de la Base de Datos desde el *backend*.

Otra característica fundamental de Django es que provee de manera nativa, el uso del manejador de bases de datos SQLite3, que gracias a su portabilidad y facilidad de uso, es adecuada para realizar pruebas y emplearla en un ambiente de desarrollo, es decir, en el proceso de creación y actualización de características del sistema.

Sin embargo, SQLite3 no es una opción recomendada para emplearla como una Base de Datos en un entorno productivo, donde se realiza un uso intensivo realizando muchas solicitudes y procesamiento de datos. Es por esto que se eligió el RDBMS PostgreSQL 9.6.10 para el entorno productivo.

4.2.1.1. Creación de Base de Datos en PostgreSQL

En la Sección 4.2.1 se justificó el uso del RDBMS PostgreSQL en su versión 9.6.10, por lo que debe procederse a la creación de la BD en el servidor o entorno de producción.

Dentro del entorno de producción se realizó la descarga e instalación del manejador de PostgreSQL a través de línea de comandos. Una vez que se concluyó la instalación del RDBMS PostgreSQL, se crea de manera automática un usuario en el servidor, con el que se tendrá acceso como *DataBase Administrator* (DBA) para proceder a la creación de usuarios y bases de datos. Para acceder a dicho perfil DBA se hace el cambio de usuario bajo la sintaxis de GNU/Linux:

```
$sudo -u postgres psql
```

Posteriormente, se hizo la creación de la BD, el usuario y se le otorgaron los privilegios necesarios para poder emplear acciones *Data Definition Language* (DDL), *Data Manipulation Language* (DML) y *Data Query Language* (DQL) sobre la BD de la aplicación. Lo anterior se logró con la ejecución de los siguientes comandos dentro de la terminal psql bajo el perfil DBA:

```
>CREATE DATABASE <database_name>;
>CREATE USER <admin_user> WITH PASSWORD '<app_password>';
>ALTER ROLE <admin_user> SET client_encoding TO 'utf8';
>ALTER ROLE <admin_user> SET default_transaction_isolation TO 'read
committed';
>ALTER ROLE <admin_user> SET timezone TO 'UTC';
>GRANT ALL PRIVILEGES ON DATABASE <database_name> TO <admin_user>;
```

Hasta este punto, se ha creado la BD con las configuraciones mínimas recomendadas para la operación correcta y manipulación de los datos, como la definición del esquema de codificación de datos a UTF8 con el que opera Django, el establecimiento de aislamiento de transacciones para bloquear la lectura y escritura de las transacciones que no han llegado a un punto en el que garantice la integridad de los datos (*rollback* o *commit*). Finalmente, se estableció la zona horaria con base en UTC, mismo con el que opera Django.

Posterior a este punto, se deben materializar las relaciones presentadas en la Sección 3.4 para introducirlas en la BD de PostgreSQL y en SQLite3. Sin embargo, aun cuando ambos RDBMS tienen una sintaxis SQL que deriva del estándar, no son iguales, por lo que no es posible emplear las mismas sentencias SQL para el DDL. Para evitar la creación de SQL manualmente sobre cada RDBMS y como se mencionó anteriormente, se aprovechará la el ORM de Django para realizar esta tarea sobre la BD a través de modelos.

4.2.1.2. Modelos de Django

Como se explicó anteriormente, gracias al uso del ORM de Django es posible hacer la definición de las relaciones de la BD desde este mismo. En la Sección 3.4 se analizó la propuesta de la BD y se definió el modelo relacional que muestra las diferentes entidades participantes en el sistema.

Con base en el modelo relacional propuesto, las entidades o relaciones se definen como modelos en un archivo de Django llamado *models.py* ubicado en el siguiente directorio:

```
mirapp/models.py
```

Por ejemplo, el modelo que define la relación para Profesores en Django es el siguiente:

```
class Profesor(models.Model):
  nombre = models.CharField(max_length=255, null=False, blank=False)
  clave = models.CharField(max_length=255, null=False, blank=False)
  rfc = models.CharField(max_length=25, null=True, blank=True)
  num_trabajador = models.CharField(max_length=255, blank=True)
  cargo = models.CharField(max_length=255, null=True, blank=True)
```

Django, a través del ORM se encarga de generar de manera interna las instrucciones SQL para el modelo Profesor:

```
-- Create model Profesor
CREATE TABLE "mirapp_profesor" (
  "id" serial NOT NULL PRIMARY KEY,
  "nombre" varchar(255) NOT NULL,
  "num_trabajador" varchar(255) NULL,
  "cargo" varchar(255) NULL,
  "rfc" varchar(25) NULL,
  "clave" varchar(255) NOT NULL
);
```

Las sentencias SQL para cada modelo se ajustan de acuerdo con el RDBMS que se ha configurado para el entorno del proyecto, es decir, SQLite3 para el entorno de desarrollo y PostgreSQL para el entorno de producción de la aplicación.

La selección del RDBMS se debe establecer en un archivo de configuraciones del proyecto de Django. Este archivo se encuentra en el siguiente directorio:

```
settings/settings.py
```

Dentro de este archivo existe el parámetro DATABASES, que permite establecer el RDBMS a emplear. Debido a que se busca optimizar el proceso de desarrollo y maximizar el tiempo invertido en desarrollar las funcionalidades de la aplicación y no realizar muchos cambios manualmente en cada entorno, se aprovechó la biblioteca de Python llamada os, que permitirá determinar si existe el directorio de configuración para el entorno productivo.

Lo anterior se realizó en código de la siguiente forma:

```
DATABASES = {
  'default': {
      'ENGINE': 'django.db.backends.sqlite3',
      'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
      'TEST': {
          'NAME': os.path.join(BASE_DIR, 'db_test.sqlite3'),
      },
  }
}
if(os.path.exists('/../../mirapp/<production-config-directory>')):
  DATABASES = {
    'default': {
      'ENGINE': 'django.db.backends.postgresql_psycopg2',
      'NAME': '<database_name>',
      'USER': '*******',
      'PASSWORD': '*******',
      'HOST': 'localhost',
      'PORT': '',
  }
}
```

4.2.1.3. Migraciones

Una vez que se han establecido en las características de las entidades dentro de los modelos de Django se procede a generar el código SQL, el cual es generado a partir de los cambios detectados en el archivo `models.py`.

Django detecta los cambios que se realizan en dicho archivo y los compara con la última definición que tomó, es decir, actúa como un sistema de control de versiones, y a partir de las últimas modificaciones será el código SQL que generará. Esto facilitará los cambios en la BD si se realizan cambios o se quiere replicar la BD.

Cada vez que se realizan modificaciones en los modelos, se deben ejecutar migraciones que generan las sentencias SQL con dichas modificaciones.

Por ejemplo, si se agrega una nueva *Foreign Key* a una entidad de la BD y se desean aplicar esos cambios, se realizará la migración con la ejecución del siguiente comando:

```
$python manage.py makemigrations
```

La ejecución del comando anterior creará un archivo con las especificaciones por aplicar a la BD, es decir, una migración. Este archivo está ubicado en `mirapp/migrations/` y luce como el siguiente archivo:

```
# -*- coding: utf-8 -*-
# Generated by Django 1.11
from __future__ import unicode_literals
from django.db import migrations, models
class Migration(migrations.Migration):
  dependencies = [
    ('mirapp', '0001_...'),
  ]
  operations = [
    migrations.AddField(
      model_name=test,
      name='profesor_ext',
      field=models.ForeignKey(
        null=True,
        on_delete=django.db.models.deletion.CASCADE,
        to='mirapp.ProfesorExt'
      )
    )
  ]
```

Este archivo contiene las instrucciones a realizar sobre la BD, pero no se ha aplicado aún, únicamente detectó los cambios en la definición de los modelos y los almacenó en su registro como cambios pendientes por aplicar.

Para observar las migraciones pendientes y realizadas, se puede consultar el comando: `$python manage.py showmigration`. Este comando mostrará con una equis "X" las migraciones que han sido aplicadas.

Es recomendable realizar las migraciones en el entorno local, para hacer pruebas y validaciones de las afectaciones que realiza sobre la BD y el impacto sobre la aplicación.

Después de verificar y confirmar las migraciones a realizar, se procede a aplicarlas con la ejecución del siguiente comando: `$python manage.py migrate`

Este comando a su vez ejecutará las sentencias SQL sobre la BD para aplicar las modificaciones. Por ejemplo, una de las migraciones que agrega un índice para optimizar consultas y que agrega un *constraint* de *Foreign Key* es el siguiente:

```
CREATE INDEX "mirapp_test_profesor_ext_id_"
ON "mirapp_test" ("profesor_ext_id");

ALTER TABLE "mirapp_test"
ADD CONSTRAINT "mirapp_test_profesor_ext_id_fk"
FOREIGN KEY ("profesor_ext_id")
REFERENCES "mirapp_profesorext" ("id") DEFERRABLE INITIALLY DEFERRED;
```

Para el ejemplo mostrado, se tiene el objetivo de crear una relación de tipo *Many to Many* entre la relación *mirapp_test* y *mirapp_profesorext*. Sin embargo, el ORM de Django se encarga de generar los *constraints* e índices necesarios en la sintaxis SQL adecuada a cada RDBMS.

Para obtener el SQL a partir de la migración se puede emplear el siguiente comando:
`$python manage.py sqlmigrate <nombre_app> <nombre_de_migracion>`

Este comando genera las sentencias SQL de acuerdo con el RDBMS seleccionado en el archivo de configuración, de manera que puede ser aplicado manualmente si así se desea o simplemente corroborar las sentencias generadas.

4.2.2. Backend

La versión de Django empleada es la 1.11.6; mientras que la versión de Python es la 3.5.3.

Como se describió en la Sección 2.3.6.2, Django es un *Framework* para aplicaciones web que permite una rápida implementación, escalabilidad, seguridad y mantenibilidad. Sin embargo, es importante realizar algunas configuraciones para su implementación y que se detallan a continuación.

4.2.2.1. Servidor Web

El servidor web elegido para esta aplicación, NGINX, recibe peticiones HTTP de los usuarios a través del puerto designado, mismas que transmite a gunicorn [48], el cual es un Web Server Gateway Interface (WSGI) diseñado para interconectar al servidor web con la aplicación web de Python (Django), esta configuración se realiza en el archivo ubicado en
`/etc/nginx/sites-available/`

Y su contenido debe lucir así:

```
server {
  listen <port_number>;
  …
  location / {
    include proxy_params;
    proxy_pass http://unix:/.../settings.sock;
  }
}
```

Esta configuración permite la comunicación entre clientes y la aplicación, adicionalmente ofrece la capacidad de reaccionar a altas tasas de solicitudes, distribuir la carga y mantener múltiples procesos de la aplicación web que está ejecutándose de una manera muy sencilla y con configuraciones mínimas, ya que Django viene por defecto configurado para funcionar de esta manera.

A su vez, este servidor web NGNIX permitirá devolver archivos estáticos a las peticiones de los clientes; dentro de estos archivos se encuentran HTML, CSS, JavaScript e imágenes.

4.2.2.2. Vistas

Como se mencionó en la Sección 2.3.4.2, Django utiliza una arquitectura *Model View Template* (MVT), que es análogo al *Model View Controller* (MVC). De hecho, el papel que desempeñan los *Controllers* del MVC es muy similar al que desempeña las *Views* del MVT, por lo que la lógica de la aplicación *Backend*, como atención de peticiones y respuestas al usuario, se encuentra en los *Views*.

Los *Views* principalmente se encargarán de procesar las peticiones que llegan desde los *endpoints*, en forma de métodos HTTP (GET, POST, UPDATE, PUT y DELETE).

Con el uso de DRF se proveen los llamados *ViewSets*, los cuales son métodos genéricos que permiten implementar los *endpoints* para un RESTful API en unas pocas líneas. Por ejemplo, para el caso del formato A21/31 se tendrá el siguiente *ViewSet*:

```
class A2131ViewSet(viewsets.ModelViewSet):
  """ API endpoint that allows A2131 forms to be viewed or edited """
  serializer_class = A2131Serializer

  def get_queryset(self, *args, **kwargs):
    queryset = A2131.objects.all()
    ...
```

```
    limit = self.request.query_params.get('limit')
    ...
    if is_superuser is False:
      queryset = queryset.filter(usuario=user_id)
    if limit:
      queryset = queryset[:int(limit)]
    return queryset
```

Este *ViewSet* permite crear, editar y actualizar registros; también brinda la posibilidad de filtrar por múltiples campos y restringe los registros que pueden ver los usuarios con un rol básico, en caso de ser un usuario con rol administrador pueden ver la información de todos los usuarios.

4.2.2.3. Serializadores

Los serializadores son una herramienta fundamental de DRF que permiten representar los datos de una API como objetos en formato JSON en vez de una respuesta directa de la BD mediante sentencias SQL. Esto brinda una interfaz para que el cliente (aplicación *frontend*) pueda lanzar peticiones sobre la BD, pero de manera indirecta y segura.

Los serializadores permiten definir los campos con los que contará la API de la aplicación, de manera que sea posible elegir los campos de los modelos que estarán disponibles en la API como objetos JSON. A continuación, se describen los serializadores para el formato A21/31 y las Firmas Electrónicas.

```
class A2131Serializer(serializers.ModelSerializer):
  class Meta:
    model = A2131
    fields = ('__all__')

class FirmaSerializer(serializers.ModelSerializer):
  usuario = UserDeptoSerializer()
  class Meta:
    model = Firma
    fields = (
      'key',
      'usuario',
      'formato',
      'nombre',
      'updated_at',
      'created_at',
      'periodo',
      'is_approved'
```

)

4.2.2.4. URLs

Las URLs (de *Uniform Resource Locator*) permiten definir la dirección a la que será posible lanzar peticiones HTTP que se encuentren asociadas con alguna *View*, misma que se encargará de ejecutar una acción asociada acorde con el método que se utilice en la petición, es decir, con el verbo HTTP. Éstos ya están definidos en los *ViewSets* antes mencionados.

DRF implementa una funcionalidad para registrar los *ViewSets* de manera que crea automáticamente una RESTful API, como se detalla en la siguiente tabla:

Método	URL	Descripción
GET	<ruta/id/>	Obtener un elemento por su id
PUT	<ruta/id/>	Actualizar completamente un elemento por su id
PATCH	<ruta/id/>	Realizar una actualización parcial a un elemento por su id
DELETE	<ruta/id/>	Borrar un elemento por su id
POST	<ruta/>	Crear un nuevo elemento
GET	<ruta/>	Obtener una lista de todos los elementos

La manera en que se define cada URL y se asocia con un *ViewSet* que se encarga de atender las peticiones de los usuarios está dentro del archivo $settings/urls.py, como se muestra a continuación para el caso del formato A21/31:

```
router = routers.DefaultRouter()
router.register(r'a2131', views.A2131ViewSet, base_name='A2131ViewSet')
```

4.2.2.5. Firmas Electrónicas

Dentro de toda la arquitectura del *backend* se han explicado los componentes principales en los apartados anteriores. Dentro de las funcionalidades a implementar, se hace un énfasis especial sobre las Firmas Electrónicas que se generarán a través de códigos QR. Éstos estarán dentro de los reportes en PDF que se generen al término de cada periodo trimestral de uso del sistema.

Esta implementación de Firmas Electrónicas permitirá la sustitución de las firmas autógrafas, pero se garantizarán los servicios de seguridad que ofrecen éstas últimas: autenticidad e integridad.

La definición de las Firmas Electrónicas se realizará creando un modelo en Django, pero utilizando un algoritmo que garantice que la clave que identifica cada registro de este modelo sea única y con una baja probabilidad de repetirse.

Para lograr lo anterior, se hará uso de los *Universally Unique Identifier* (UUID) que se describieron en la Sección 2.4.2

La probabilidad de colisión es muy pequeña, pero aun así dado que en el modelo definido de las Firmas Electrónicas es la llave primaria, si llegara a haber una colisión, Django generaría un error de integridad y pedirá repetir la operación; por lo que es posible asegurar la unicidad de cada registro realizado y teniendo un tratamiento correcto de las colisiones.

```
class Firma(models.Model):
  ...
  key = models.UUIDField(
          primary_key=True,
          default=uuid.uuid4,
          editable=False
        )
  ...
```

Los QR generados contienen una URL con el UUID generado, de manera que al escanearlos desde una aplicación móvil se direcciona a un módulo de validación dentro del servidor, el cual se encarga de obtener la información asociada al registro con ese UUID, como son el título del reporte, el nombre de quien firma, fecha en que se realizó la firma y el periodo en que se emitió, entre otros campos. Como se muestra en la Figura 4.2.

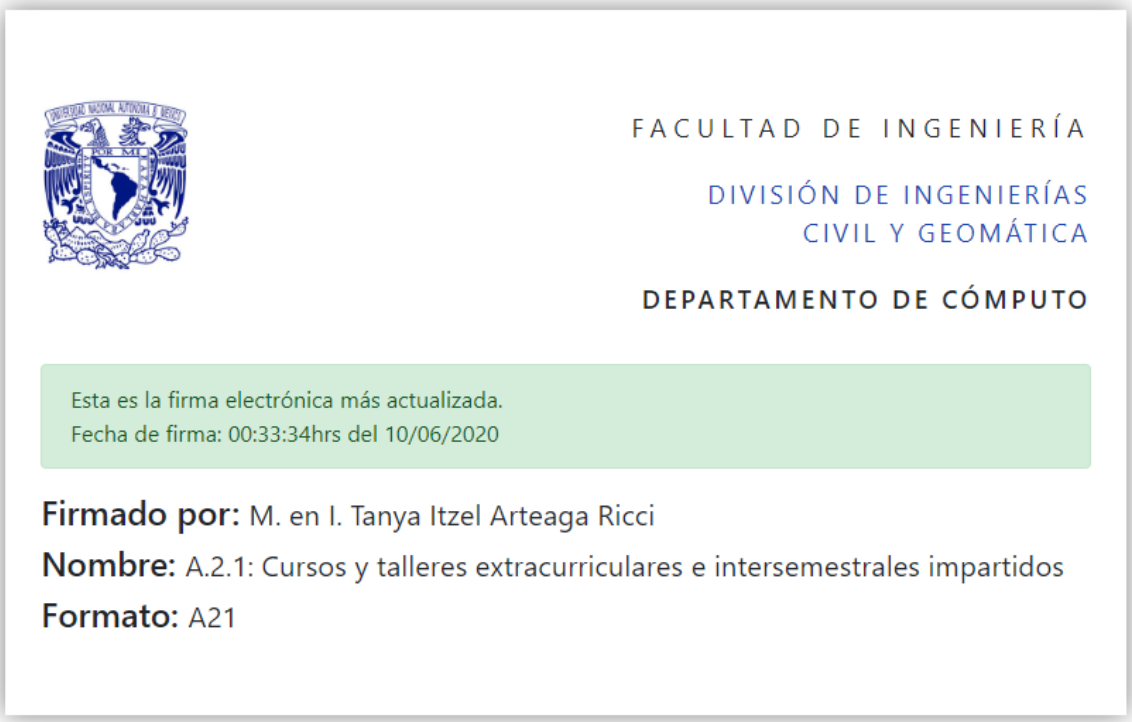


Figura 4.2. Vista de validación correcta de Firma Electrónica.

Cada Firma Electrónica posee dos marcas de tiempo: creación y actualización de la firma. De esta forma, el módulo de validación es capaz de verificar el valor de la fecha de actualización y determinar si existe una nueva firma más actualizada, en caso de que se hayan modificado los registros del periodo, sin perder validez el documento que ya fue firmado. Se muestra un ejemplo de esto en la Figura 4.3.

Figura 4.3. Vista de existencia de una Firma Electrónica más reciente.

Por otra parte, en caso de que la validación sea incorrecta, el módulo de validación mostrará una vista como la de la Figura 4.4

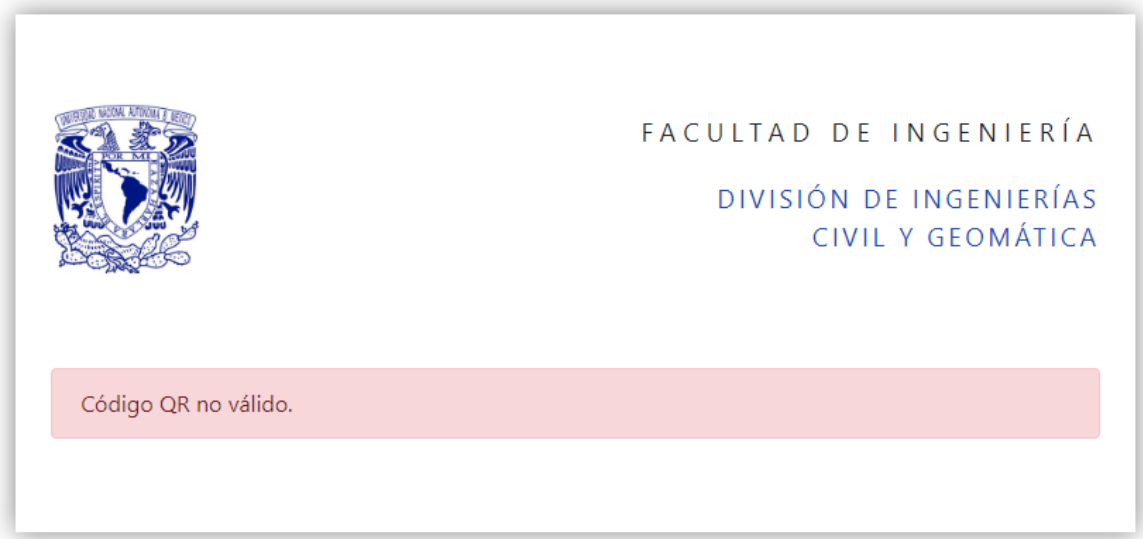


Figura 4.4. Vista de validación de Firma Electrónica incorrecta.

4.2.2.6. Autenticación

Para la autenticación se utilizó una biblioteca que extiende la funcionalidad de DRF llamada *REST Framework JWT Auth*. Ésta implementa *PyJWT,* que es una biblioteca de Python que cumple con el estándar RFC 7519 [21]de *JSON Web Token* (JWT) y que garantiza la autenticación e integridad de los datos por medio de *Message Authentication Codes* (MACs) como se explicó en la Sección 2.4.1.3.

Por lo que cada vez que un usuario introduce sus credenciales se genera un *token* en el servidor con expiración de 24 horas, el cual se le devuelve al cliente para su almacenamiento temporal. De esta manera no es necesario crear y almacenar datos de sesiones en el servidor web, sino que el usuario al hacer peticiones HTTP a la REST API, envía en la cabecera de las peticiones el *JSON Web Token*, lo que permitirá autenticarlo y otorgarle acceso a recursos.

4.2.3. Frontend

La versión de Angular que se utilizó es la 8.2 junto con *NodeJS* 12.14. En conjunto, estas dos tecnologías permitieron desarrollar la aplicación web para el cliente

Como se mencionó en la Sección 2.3.5.2, Angular es un *Framework* web que brinda muchas facilidades para la creación de aplicaciones e interfaces web modulares, expandibles y modernas. Dentro de las

[21] RFC 7519 JSON Web Token (JWT). Fuente: https://tools.ietf.org/html/rfc7519

características que tiene este *Framework*, se aprovechó principalmente el enfoque de componentes, servicios y formularios.

4.2.3.1. Componentes

Los componentes de Angular están basados en el estándar de los *web components*, que son un conjunto de APIs de plataformas web que permiten la creación de nuevas etiquetas HTML que encapsulan un contenido o funcionalidad específica, de manera que sean reutilizables y personalizables. [49]

Los componentes corresponden a la definición de vistas o conjunto de elementos de pantalla que pueden mostrar información, reaccionar y desencadenar eventos en función de su lógica y de los datos. Es decir, corresponde con aquellos elementos que son tangibles en el navegador web para el usuario, mostrándole información y permitiéndole interactuar con toda la aplicación. [50]

La estructura de los componentes en Angular se basa en tres elementos principales para su funcionamiento:

1. *Template*: es la parte empleada para renderizar los elementos visuales o de la *User Interface*. Es decir, contiene la estructura HTML del componente y tiene la capacidad de enlazarse con datos definidos en la *Class*.
2. *Class*: es la parte que contiene la lógica y datos de los componentes que se conectan con el *Template*. Se define como una clase en el paradigma de OOP y en lenguaje *TypeScript*. Esta parte es la encargada de conectar la parte visual con algún *Backend* a través de servicios, que se detallan en la Sección 4.2.3.2.
3. *Metadata*: contiene la definición de datos y de la clase por defecto empleada en Angular para el funcionamiento de los componentes. Se define a través de un decorador llamado @Component.

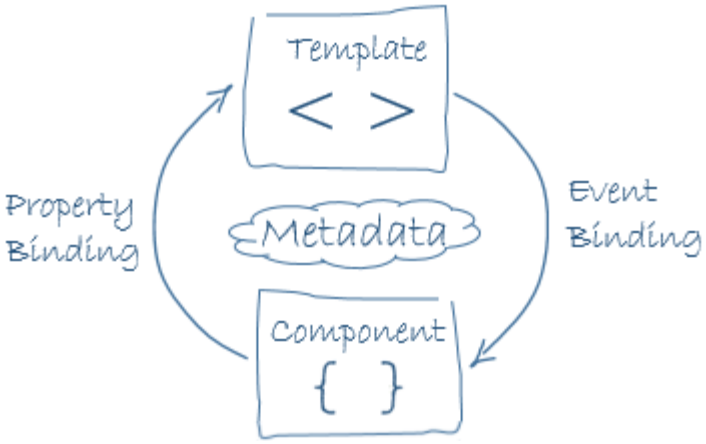


Figura 4.5. Arquitectura básica de un componente en Angular.[22]

Cada componente creado se encapsula en etiquetas o un *selector*, con lo que es posible hacer uso de éstos en otras partes de la aplicación.

A continuación, se presenta una definición básica de ejemplo de un componente en *capturar.component.ts*.

```
@Component({
  selector: 'app-capturar',
  templateUrl: './capturar.component.html',
  styleUrls: ['./capturar.component.scss']
})
export class CapturarComponent implements OnInit { ... }
```

El uso de este componente se realiza a través de la siguiente etiqueta:

<app-capturar></app-capturar>

4.2.3.2. Servicios

Un servicio en Angular es habitualmente una clase con un propósito bien definido, de manera que realice una tarea en particular como obtener datos de un servicio externo. Al igual que los componentes, los servicios se definen como clases, pero empleando un decorador `@Injectable()`. [51]

[22] Fuente: https://angular.io/guide/architecture

Al crearse una instancia de un componente, se determinan los servicios y otras dependencias que ésta requiera a través de los parámetros de su constructor.

Figura 4.6. Integración de servicios con componentes.[23]

Cuando se inicializa un servicio, Angular crea una sola instancia que se inyecta en los componentes que lo soliciten, de esta manera optimiza la memoria utilizada y permite que haya coherencia entre los componentes que importan un mismo servicio.

El siguiente ejemplo define un servicio llamado *FormatoService*, que a su vez inyecta otros servicios como *HttpClient* y *HelperService*.

```
@Injectable()
export class FormatoService {
  public baseUrl = environment.apiEndpoint;
  constructor(
    private http: HttpClient,
    public helperService: HelperService
  ) {}
  ...
}
```

4.2.3.3. Autenticación

Aprovechando las propiedades de los servicios, se definió uno con el propósito de brindar la funcionalidad de control de la autenticación de los usuarios. En este se define una variable privada `loggedIn` a la que no pueden acceder los componentes pero si recibir actualizaciones por medio del *observable* `@Output() updates`, en caso de que ésta cambie.

Con esto será posible conocer el estado actual del usuario en la aplicación, es decir, permite saber si el usuario está autenticado y si tiene los permisos para acceder a los módulos de la aplicación.

[23] Fuente: https://angular.io/guide/architecture

```
@Injectable()
export class LoginService {
  private loggedIn:
    BehaviorSubject<boolean> = new BehaviorSubject<boolean>(false);
  @Output() updates = new EventEmitter();

  constructor(
    public router: Router,
    private http: HttpClient,
    public helperService:HelperService
  ) {
    this.validateToken(token).subscribe(
      res => {
        ...
      },
      error => {
        this.loggedIn.next(false);
        this.logOut();
        return this.loggedIn.asObservable(), false;
      }
    )
  }
  ...
}
```

Aprovechando la funcionalidad de los servicios, se definió uno llamado *HelperService* el cual tiene la finalidad de proveer las cabeceras de autenticación a otros servicios que lo requieran.

```
@Injectable()
export class HelperService {
  constructor() { }

  public getAuthData(): any {
    let jwttoken = localStorage.getItem('token');
    try {
      const requestHeaders: any = new HttpHeaders()
      .append('Authorization', `JWT ${jwttoken}` || '')
      return {headers: requestHeaders}
    } catch (error) {
      const requestHeaders: any = new HttpHeaders()
      return {headers: requestHeaders}
    }
  }
```

```
}
```

El siguiente método se realiza una petición HTTP que utiliza `getAuthData` de HelperService, lo que permite que el código sea más limpio y en caso de modificar el uso de las cabeceras de autenticación sólo se realizará en este lugar.

```
createFormato(ref, data): any {
  let url = this.baseUrl + ref;
  let options = this.helperService.getAuthData();
  return this.http.post(url, data, options);
}
```

4.2.3.4 Formularios

Los formularios en Angular se generan a través de los *FormBuilder*, que son servicios que facilitan su creación ya que, implementan métodos muy útiles como: establecimiento de campos requeridos, tipo de datos, validaciones de campos y expresiones regulares.

El *FormBuilder* permite el desacoplamiento de elementos de la vista creando un *FormGroup* o grupo de controles a los que le aplicará un seguimiento de su estado y cambios para hacer las validaciones establecidas. [52]

Por ejemplo, un formulario en el módulo de captura tiene la siguiente sintaxis:

```
export class CapturarComponent implements OnInit {
  this.a2232Form = this.formBuilder.group({
    nombre: ['', Validators.compose(
      [
        Validators.minLength(5),
        Validators.maxLength(100),
        Validators.required,
        Validators.pattern('[a-zA-Z ]*')
      ]
    )],
    correo: ['', Validators.email],
    fecha_inicio: ['', Validators.required],
    fecha_fin: ['', Validators.required],
    hora_inicio: ['', Validators.required],
    hora_fin: ['', Validators.required]
  });
  ...
}
```

4.3. Módulo de inicio de sesión

Representa el módulo principal con el que tendrá contacto el usuario con el sistema web, ya que es la primera vista en la que se le muestra un formulario de *login* e información acerca del calendario de periodos de apertura del sistema.

Por medio del formulario se obtiene la información que se envía utilizando el servicio de *login* para autenticar la identidad del usuario y otorgarle los permisos de acceso a los demás módulos del sistema.

La información del calendario se obtiene a través de un servicio que consulta la información desde la API. Éste es el único servicio que brinda información de manera pública, es decir, sin necesidad de haber iniciado sesión.

Figura 4.7. Vista del módulo de inicio de sesión.

En caso de que el usuario ingrese sus claves de acceso de manera errónea o que se encuentre deshabilitado, se mostrará una alerta como en la Figura 4.8. Es importante mencionar que un usuario

con el rol de administrador tendrá capacidad de ingresar en cualquier momento y puede habilitar o deshabilitar el acceso al sistema para el resto de los usuarios con rol estándar.

Figura 4.8. Inicio de sesión erróneo.

4.4. Módulo de Captura

Este módulo es el encargado de capturar la información con la que se alimenta el sistema, es decir, corresponde al registro de actividades trimestrales de los Departamentos. Este módulo se conforma por un formulario dinámico que adecua los campos necesarios en función del rubro que se desea capturar.

Figura 4.9. Vista principal del módulo de captura.

En caso de que no exista algún error en el envío del formulario, entonces se obtiene una respuesta que confirma que se guardó el registro con un mensaje como el que se muestra en la Figura 4.10.

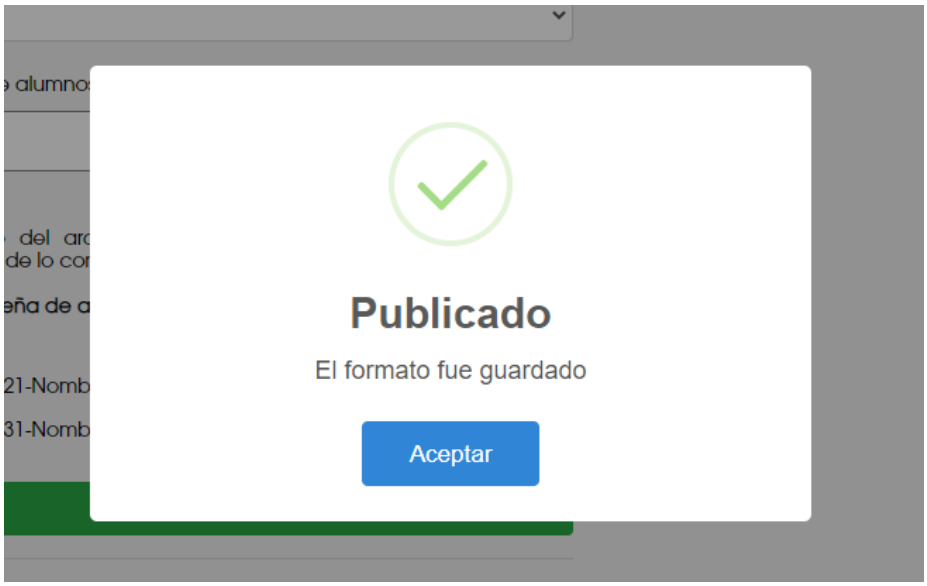


Figura 4.10. Proceso de captura exitoso.

Los diferentes apartados o rubros que conforman a la MIR, detallados en la Sección 3.1, se encuentran disponibles como primera opción del formulario dinámico de este módulo con el nombre de "Formato".

Para la gran mayoría de los Formatos, el formulario emplea campos de texto libre para que los llene el usuario. Una gran diferencia se encuentra en el formato "A73/83 - Actividades académicas de reforzamiento realizadas" de tipo prácticas, ya que se habilita la opción de descargar un formato llamado "MIR_Practicas_A7383.csv" en el que es posible llenar todos los registros de prácticas realizadas.

La Figura 4.11 muestra la vista de la funcionalidad descrita anteriormente.

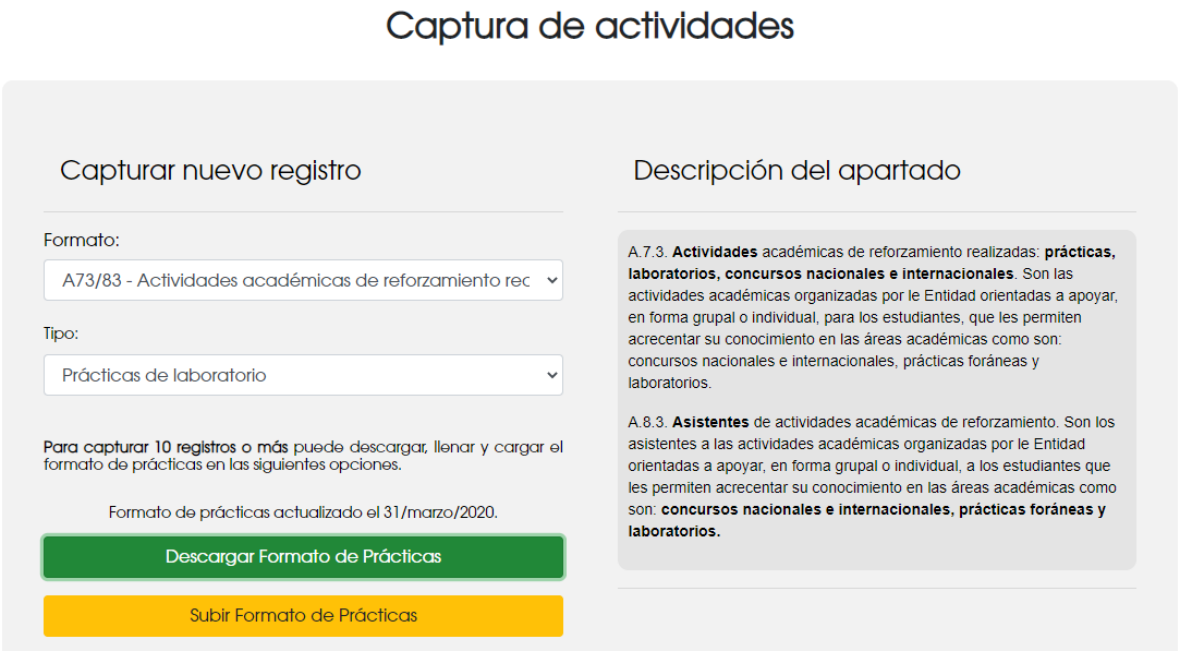


Figura 4.11. Vista de captura de actividades para el apartado A73/83.

Posterior a su descarga y llenado del archivo CSV, se habilita la opción de cargar todos esos registros con un par de *clics*, mediante la carga de este archivo como se muestra en la Figura 4.12.

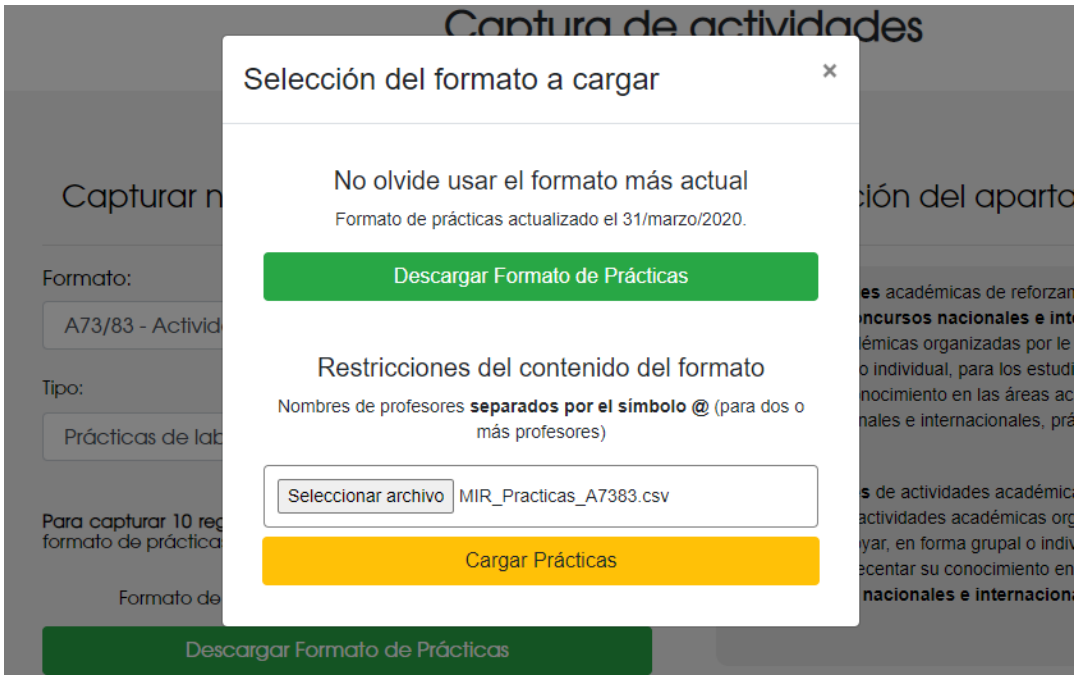


Figura 4.12. Vista del modal para la carga de prácticas mediante el archivo CSV.

Al presionar el botón "Cargar Prácticas" se hace un proceso de validación de los datos y en caso de que todo esté correcto se muestra la vista de la Figura 4.13.

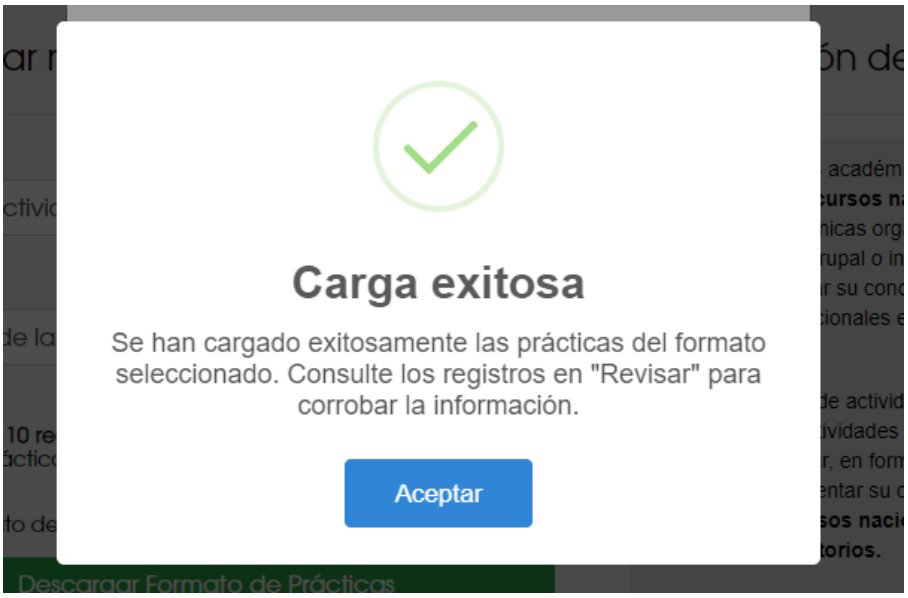


Figura 4.13. Vista de carga exitosa de registros de prácticas del CSV.

En caso de existir algún problema en la validación de los datos, se muestra una alerta detallando el error como en la Figura 4.14.

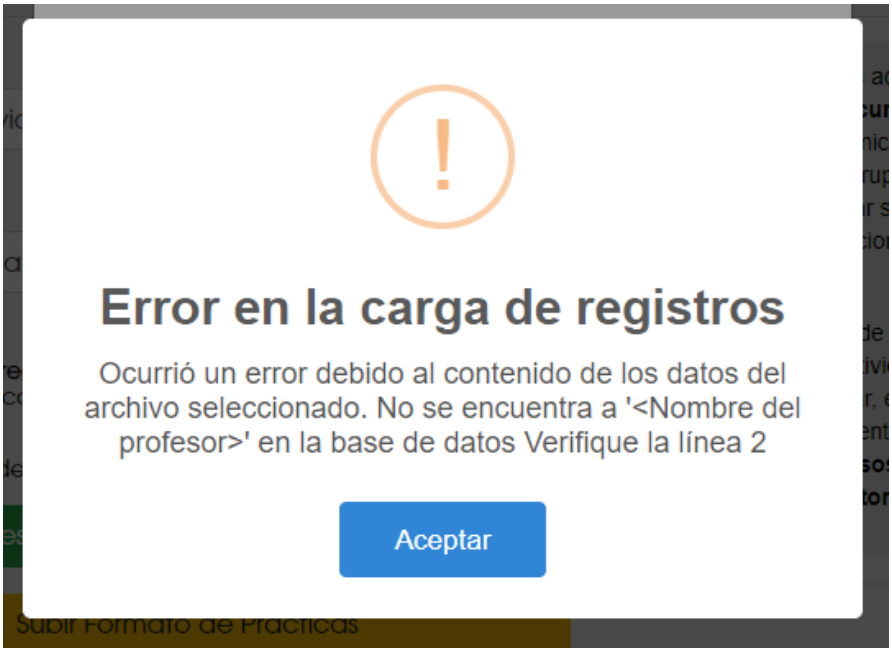


Figura 4.14. Vista de carga fallida de registros de prácticas del CSV.

La funcionalidad anterior tiene como objetivo la disminución de tiempo y reducción de carga de trabajo para el usuario al tener que llenar muchos registros de manera manual.

De manera general, el funcionamiento de este módulo de carga consiste en implementar validaciones en el formulario y en los datos que se van capturando y se envían mediante un servicio por medio de peticiones HTTP a la API del *Backend*.

A continuación, se presenta la estructura del servicio que se encarga de enviar los datos del formulario:

```
export class FormatoService {
  ...
  createFormato(ref, data):any {
    let url = this.baseUrl + ref;
    let options = this.helperService.getAuthData();
    return this.http.post(url, data, options);
  }
  ...
}
```

4.5. Módulo de Revisión

En este módulo es posible visualizar todos los registros capturados en el sistema a través de los diferentes periodos. Adicionalmente, es posible editar aquellos que corresponden al periodo actual.

La forma de interacción consiste en dos campos: Periodo y Apartado. Con éstos, es posible recabar aquellos registros de cierto periodo y para un particular apartado o rubro de actividades. Lo anterior se plasma en la Figura 4.15.

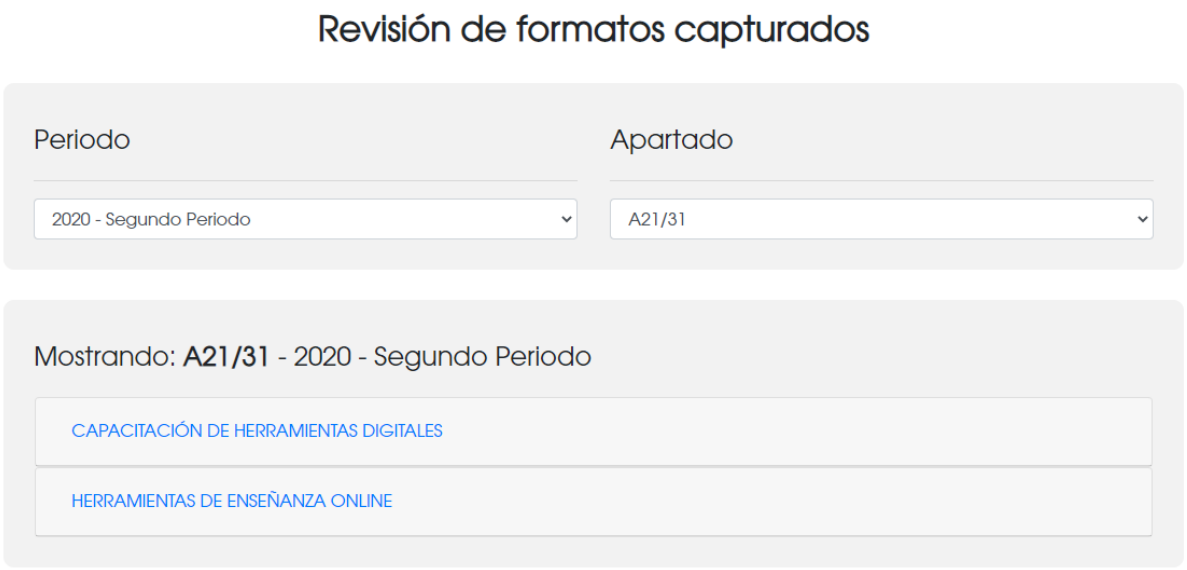


Figura 4.15. Vista de interacción con el módulo de revisión.

Se tiene la posibilidad de dar clic a cada uno de los registros mostrados al inferior de la Figura 4.15. Al realizarlo, se muestra una sección con más información y la posibilidad de editar el registro si se trata del periodo actual, como se muestra en la Figura 4.16.

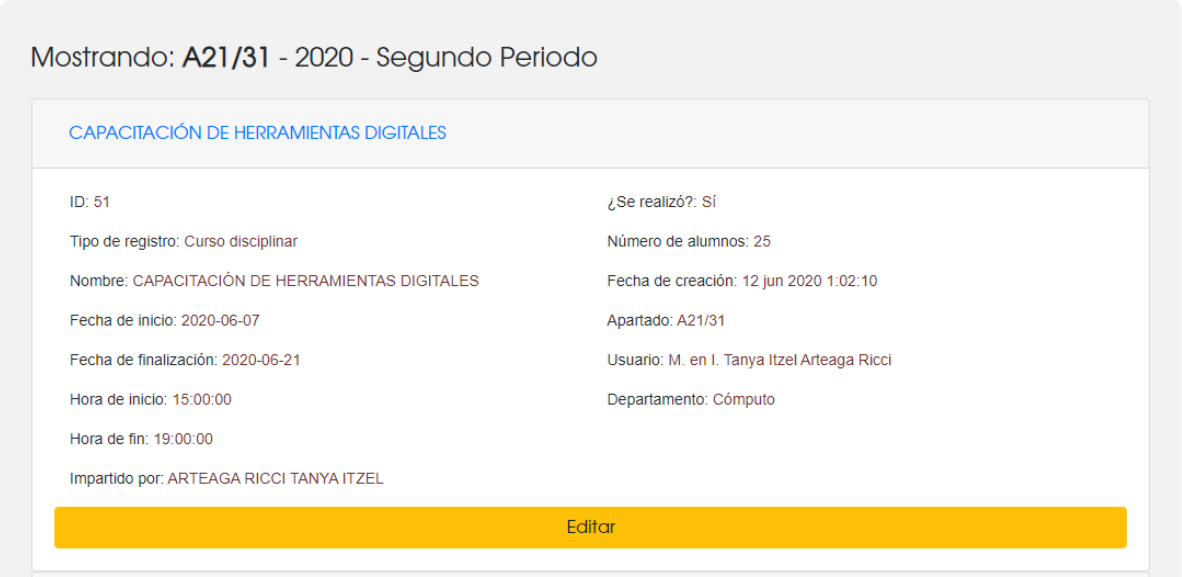


Figura 4.16. Información de registros en el módulo de revisión.

Al darle clic al botón "Editar" se desplegará un formulario en función del rubro que se haya seleccionado donde se muestran los campos que se permite modificar como en la Figura 4.17.

Figura 4.17. Vista de modificación de datos de registros.

4.6. Módulo de Esperados del Año

El propósito de este módulo consiste en brindar la capacidad a los usuarios de capturar la cantidad de registros que se esperan tener para cada uno de los apartados para el siguiente año al que está en curso. Estos datos recolectados permiten hacer una comparativa en el módulo de estadísticas, detallado en la Sección 4.7, de los datos esperados capturados contra los reales que se obtengan, de manera que se tengan indicadores de rendimiento

Para un usuario con rol estándar, únicamente es posible ingresar datos de proyecciones del próximo año y una vez capturados, no pueden modificarse, como se observa en la Figura 4.18.

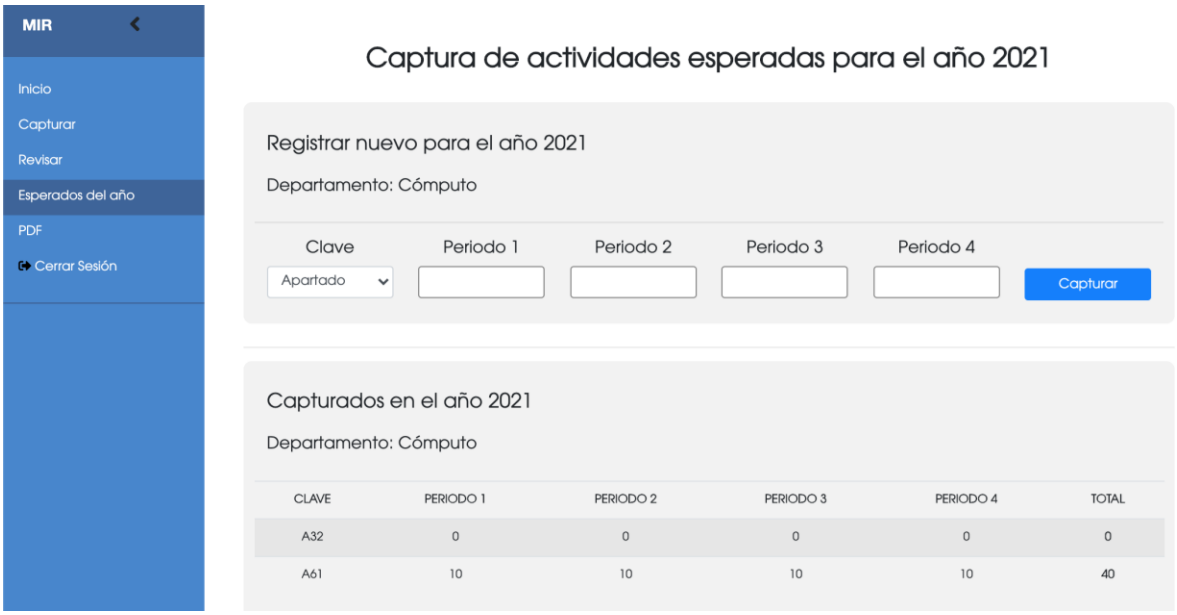


Figura 4.18. Vista del módulo de esperados del año para un usuario estándar.

Mientras que, para un usuario con rol de administrador, este módulo le permite obtener una visualización de estos datos, pero de todos los Departamentos y filtrarlos por años pasados. Además, tiene la capacidad de editar los registros si se requiere. Todo esto se muestra en la Figura 4.19.

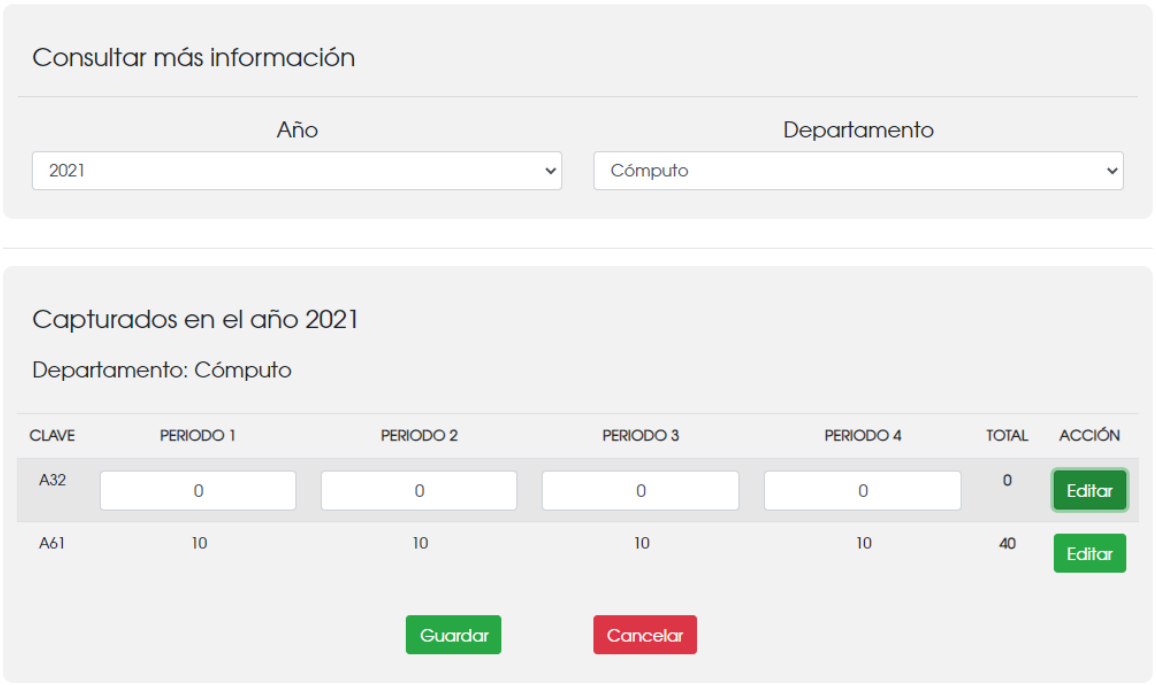


Figura 4.19. Vista del módulo de esperados del año para un usuario administrador.

4.7. Módulo de Estadísticas

El propósito de este módulo es mostrar una interfaz sencilla donde se concentren datos esperados y logrados, de manera que se pueda hacer un contraste entre éstos para obtener indicadores de resultados.

Su funcionamiento es sencillo; inicialmente se despliega un gráfico con los datos esperados y reales que se tengan recabados, de manera que sea visualmente fácil obtener estos índices. Ejemplos de esta descripción son los mostrados en las Figuras 4.20 y 4.22.

En la parte inferior de los gráficos estadísticos se muestra la misma información, pero a través de una matriz, como las mostradas en las Figuras 4.21 y 4.23. De esta forma se tienen dos medios para que se pueda tener un control y seguimiento a cada Departamento y rubro.

Este módulo puede ser accedido tanto por administradores del sistema (Jefatura) como por usuarios con rol estándar (Jefes de Departamentos).

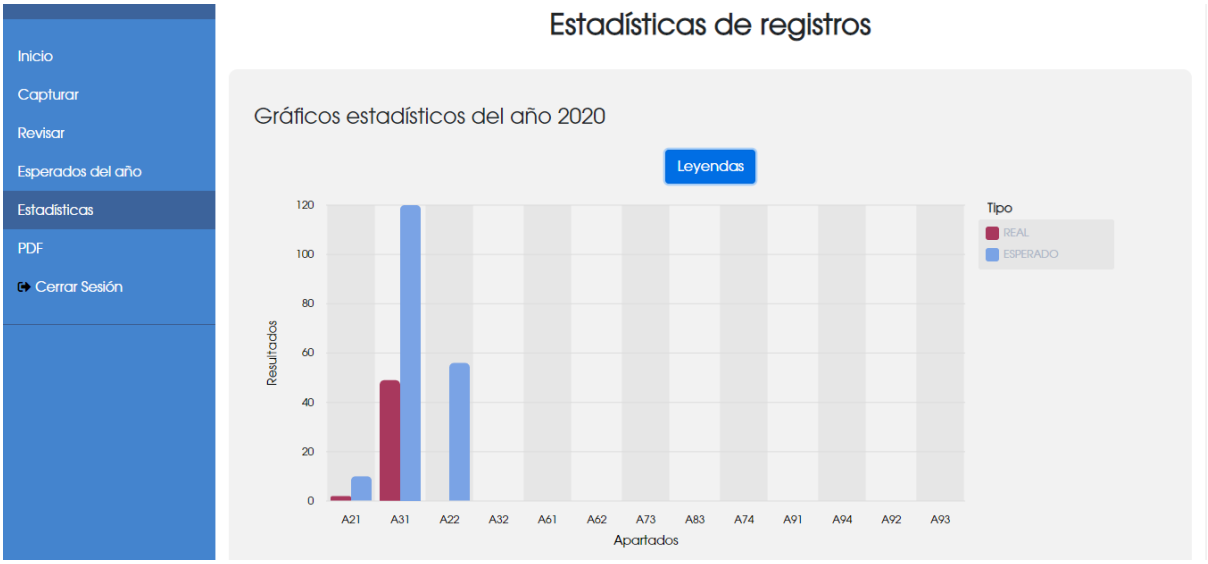


Figura 4.20. Vista de gráficos del módulo de estadísticas del año para un usuario estándar.

Registros del año 2020

Departamento: Geodesia y Fotogrametría

Año

2020

APARTADO	DATO	PERIODO 1	PERIODO 2	PERIODO 3	PERIODO 4	ANUAL
A21	REAL	1	1	0	0	1
	ESPERADO	4	3	0	0	7
A31	REAL	4	0	0	0	4
	ESPERADO	0	0	0	0	0
A22	REAL	0	0	0	0	0
	ESPERADO	0	0	0	0	0
A32	REAL	0	0	0	0	0
	ESPERADO	0	0	0	0	0

Figura 4.21. Matriz de datos del módulo de estadísticas del año para un usuario estándar.

Debido a que la cantidad de información mostrada para un usuario administrador es bastante, se tiene la posibilidad de filtrar los registros estadísticos por año y por periodo. Estas opciones son mostradas en la Figura 4.23.

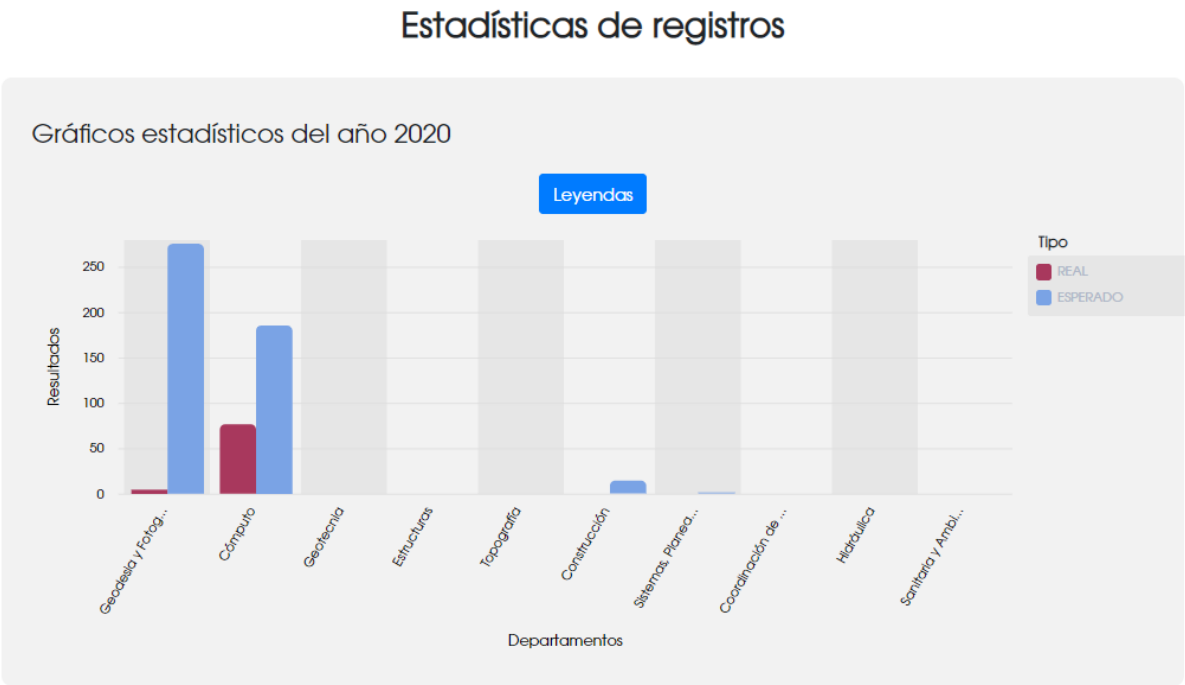
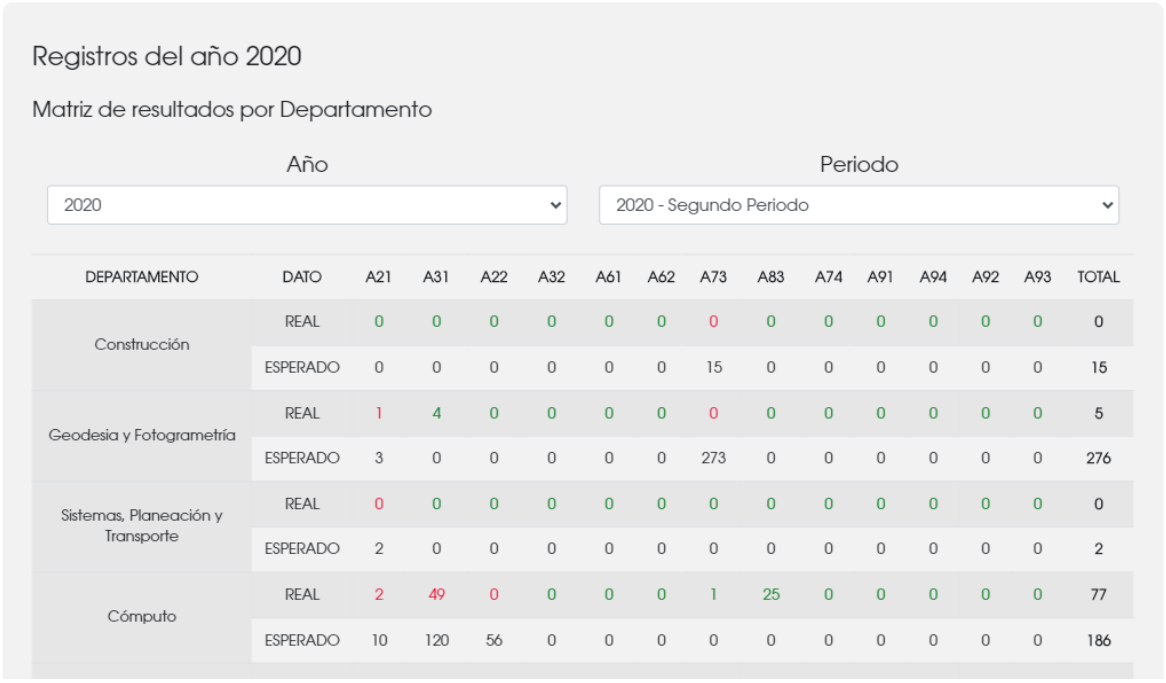


Figura 4.22. Vista de gráficos del módulo de estadísticas del año para un usuario administrador.



DEPARTAMENTO	DATO	A21	A31	A22	A32	A61	A62	A73	A83	A74	A91	A94	A92	A93	TOTAL
Construcción	REAL	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	ESPERADO	0	0	0	0	0	0	15	0	0	0	0	0	0	15
Geodesia y Fotogrametría	REAL	1	4	0	0	0	0	0	0	0	0	0	0	0	5
	ESPERADO	3	0	0	0	0	0	273	0	0	0	0	0	0	276
Sistemas, Planeación y Transporte	REAL	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	ESPERADO	2	0	0	0	0	0	0	0	0	0	0	0	0	2
Cómputo	REAL	2	49	0	0	0	0	1	25	0	0	0	0	0	77
	ESPERADO	10	120	56	0	0	0	0	0	0	0	0	0	0	186

Figura 4.23. Matriz de datos del módulo de estadísticas del año para un usuario administrador.

4.8. Módulo de Funcionarios

Este módulo puede ser accedido únicamente por un usuario administrador y su funcionamiento está enfocado para realizar cambios administrativos de las personas a cargo de los Departamentos.

Esto es requerido debido a que el sistema fue diseñado tomando como entidades principales a los Departamentos, de manera que es posible hacer una reasignación de la persona que se encuentra a cargo como su encargado o jefe sin la necesidad de borrar usuarios pasados. Esto permitirá mantener un histórico correcto de los datos.

Figura 4.24. Vista del módulo de funcionarios en el sistema.

4.9. Módulo de Reportes en PDF

Como una parte del proceso de la MIR consiste en concentrar la información capturada de cada periodo por los Departamentos se creó este módulo, el cual tiene la finalidad de recabar y concentrar esta información en un formato estándar a través de un PDF.

Como se mencionó en la Sección 3.5, al capturar algunos apartados se trataron de manera conjunta o se fusionaron; sin embargo, al generarse los reportes probatorios los registros se operan de manera separada por cada rubro o formato.

Figura 4.25. Vista principal del módulo de funcionarios en el sistema para un usuario estándar.

Se observa en la parte inferior de la Figura 4.25 un conjunto de botones que permite generar cada PDF por rubro del Departamento al que pertenece la sesión.

Mientras que para un usuario administrador se muestra una opción de hacer el filtrado por Departamento para generar el reporte de cada uno de éstos y se presenta una opción autogenerada llamada "Actividades concentradas de la DICyG" como se muestra en la Figura 4.26, que integra los datos de todos los Departamentos.

Figura 4.26. Vista principal del módulo de funcionarios en el sistema para un usuario administrador.

Para los dos escenarios mostrados en las Figuras 4.25 y 4.26 se tiene un conjunto de campos prellenados que serán plasmados en los reportes. Se tiene la posibilidad de editarlos si es necesario. Estos campos corresponden al Departamento, Nombre y Puesto de la persona a quien se dirige el reporte (oficio), así como el Nombre y Puesto de quien lo firma.

La generación del documento en PDF se muestra en la Figura 4.27, mismo que será posible firmarlo electrónicamente como se detalla en la Sección 4.9.1, imprimirlo o descargarlo según las necesidades del usuario.

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍAS CIVIL Y GEOMÁTICA

DEPARTAMENTO DE CÓMPUTO

Asunto: Tabla resumen Matriz MIR - 2020 - Segundo Periodo

DR. JESÚS HUGO MEZA PUESTO
SECRETARIO ACADÉMICO DE LA DICyG
FACULTAD DE INGENIERÍA, UNAM

P r e s e n t e:

Se anexa la siguiente información del apartado **A.2.1: Cursos y talleres extracurriculares e intersemestrales impartidos** y a la matriz de indicadores MIR implementado por el departamento para el periodo 2020 - Segundo Periodo

#	Actividad	Fecha/hora de inicio	Fecha/hora de finalización	¿Realizado?
1	Herramientas de enseñanza online	2020-05-04 14:00:00	2020-05-07 16:00:00	Sí
2	Capacitación de herramientas digitales	2020-06-07 15:00:00	2020-06-21 17:00:00	Sí

Figura 4.27. Vista parcial del reporte generado en PDF de un usuario estándar.

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍAS
CIVIL Y GEOMÁTICA

JEFATURA

Asunto: Tabla resumen Matriz MIR - 2020 - Segundo Periodo

ING. LUIS JIMÉNEZ ESCOBAR
SECRETARIO ADMINISTRATIVO FI
FACULTAD DE INGENIERÍA, UNAM

P r e s e n t e:

Se anexa la siguiente información del apartado **A.2.1: Cursos y talleres extracurriculares e intersemestrales impartidos** y a la matriz de indicadores MIR implementado por el departamento para el periodo 2020 - Segundo Periodo

#	Actividad	Fecha/hora de inicio	Fecha/hora de finalización	¿Realizado?	Departamento
1	Herramientas de enseñanza online	2020-05-04 14:00:00	2020-05-07 16:00:00	Sí	Cómputo
2	Capacitación de herramientas digitales	2020-06-07 15:00:00	2020-06-21 17:00:00	Sí	Cómputo

Figura 4.28. Vista superior del PDF generado por un usuario administrador.

Figura 4.29. Vista inferior del PDF generado por un usuario administrador.

4.9.1. Firma Electrónica

Los reportes generados en PDF descritos en la Sección 4.8. requieren ser firmados y garantizar su validez; para lograr esto se implementaron las Firmas Electrónicas descritas en la Sección 4.2.2.5, con lo que basta presionar el botón de firmar (mostrado en la parte inferior izquierda de la Figura 4.29) para generar un identificador único asociado con los datos del usuario y asociarlo a un QR, de manera que esta funcionalidad puede reemplazar la firma autógrafa.

La generación de Firmas Electrónicas está disponible tanto para jefes de Departamentos como para el Jefe de la División.

Una vez que se ha generado la Firma Electrónica, se muestra un código QR como firma y en la parte inferior se muestra un botón para imprimir o guardar el PDF. Lo anterior se encuentra en la Figura 4.30.

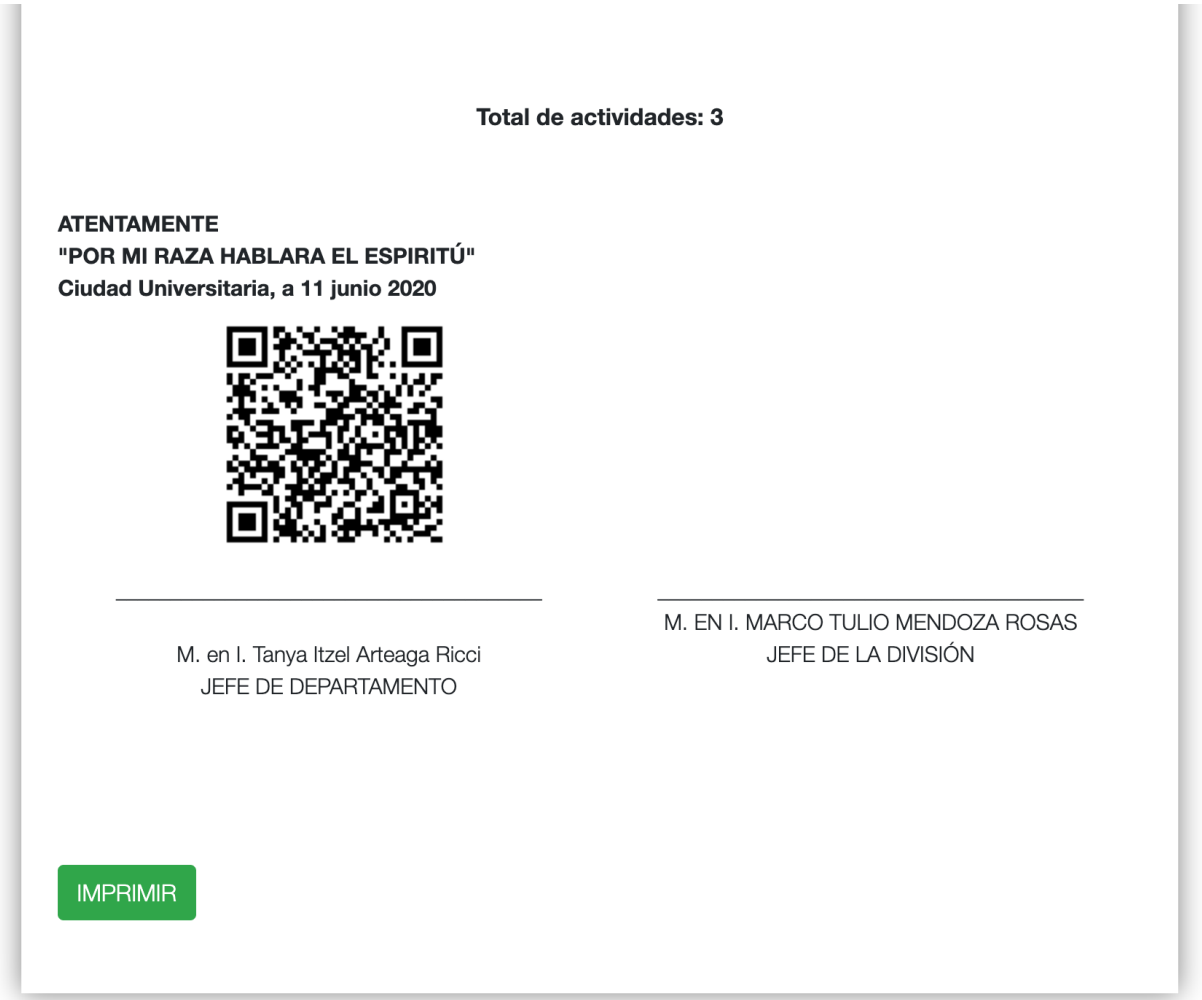


Figura 4.30. Vista inferior del PDF firmado por un usuario administrador.

4.10. Revisión de Reportes

Este módulo sólo puede ser accedido por administradores del sistema. Dado que los jefes de cada Departamento capturan sus actividades y generan reportes que firman para validarlos, se requiere la supervisión y aprobación por parte de la Secretaría Técnica.

Para lograr este objetivo dentro del sistema, la Secretaría Técnica tiene la facultad de visualizar y autorizar los reportes que generó cada uno de los Departamentos, de manera que, si se tienen indicadores de los reportes erróneos, podrá solicitarle a su correspondiente jefe de Departamento que haga las correcciones necesarias.

Esta funcionalidad se logra a través de un panel de revisión, que muestra para cada formato cada Departamento y el estado de su proceso: si ha firmado su reporte y si lo ha aprobado la Secretaría Técnica. En caso de que no se haya aprobado aún, se puede visualizar (si existe) el reporte de cada Departamento; posteriormente, al ver que todo está correcto puede aprobarlo con el botón de Autorizar.

Todas las opciones anteriores se muestran en la Figura 4.31.

Figura 4.31. Vista del panel de revisión de reportes de los Departamentos.

4.11. Módulo de Usuarios

Este módulo únicamente puede ser accedido por administradores del sistema. El objetivo principal de este módulo es poder modificar el acceso al sistema web por parte de los usuarios no administradores, lo que permite establecer periodos más estrictos de entrega de información para los Departamentos.

La habilitación y deshabilitación del acceso a los recursos del sistema se puede realizar de manera individual o grupal. La Figura 4.32. muestra este módulo.

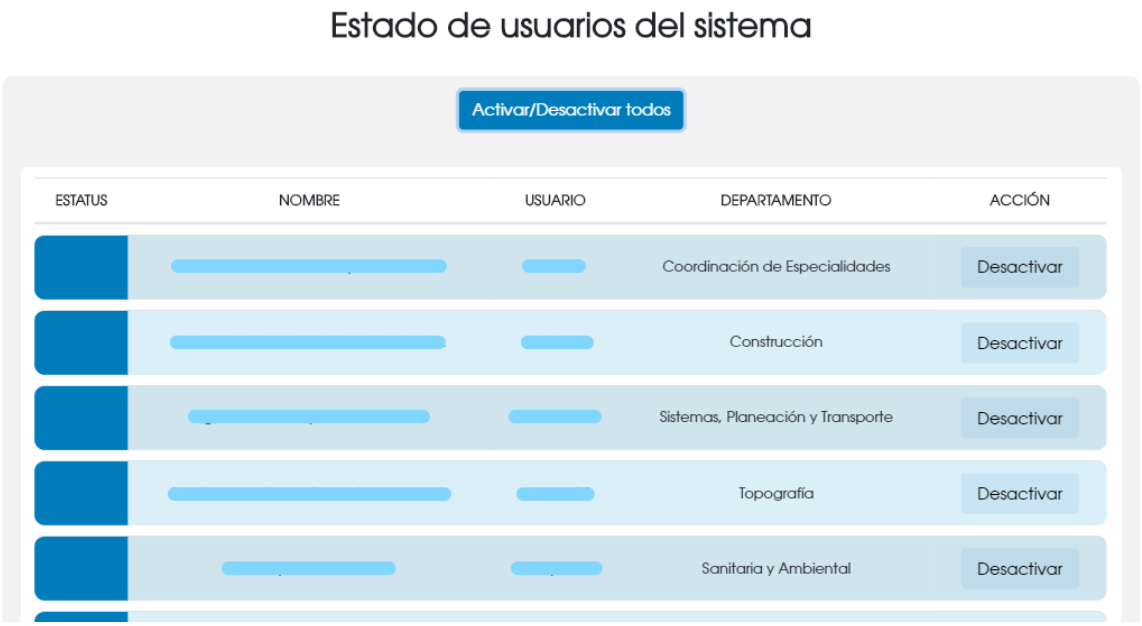


Figura 4.32. Vista del módulo de activación de usuarios con usuarios habilitados.

Figura 4.33. Vista del módulo de activación de usuarios con usuarios deshabilitados.

4.12. Módulo de Calendarios y Periodos

Este módulo es de acceso restringido para los administradores del sistema. El objetivo principal de este es permitir la captura y edición de las fechas de inicio y fin del año en curso y del año inmediato próximo, como se muestra en la Figura 4.34.

Figura 4.34. Vista de edición de fechas de los periodos del calendario del año 2020.

Además, permite la creación de un nuevo periodo, que al crearse de manera inmediata todas las actividades que se capturen por los jefes de Departamentos estarán asociadas a éste. Lo anterior se muestra en la Figura 4.35.

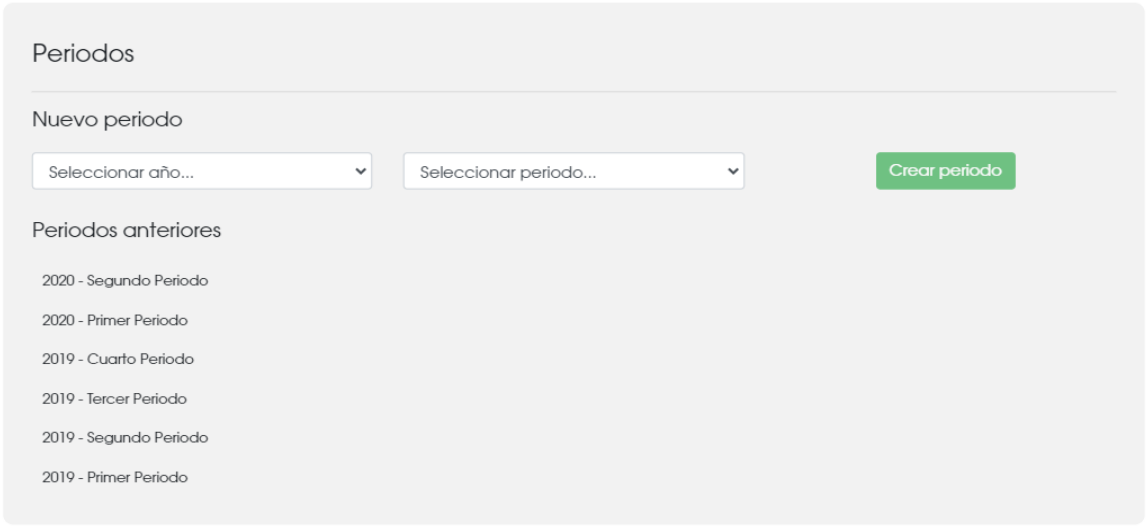


Figura 4.35. Vista del histórico y creación de periodos.

CAPÍTULO 5. PRUEBAS, VALIDACIÓN Y VERIFICACIÓN

5.1. Pruebas unitarias

Como se mencionó en la Sección 2.7.1. las pruebas unitarias tienen el objetivo de probar funciones o procedimientos específicos de manera aislada, de manera que garantice su correcto funcionamiento.

En el sistema web desarrollado para la MIR se hicieron pruebas del módulo de la carga de prácticas, explicado en la Sección 4.4, como una unidad, aunque involucra una serie de pasos previos por parte del usuario. Debido a que esta funcionalidad de cargar las prácticas mediante un archivo CSV se desempeña totalmente a través de funciones dentro *views.py* en el *Backend,* se utilizaron herramientas de *Django Rest Framework* (DRF) para ayudar a la realización de pruebas unitarias; su documentación explica la creación y uso de este tipo de pruebas, misma que se puede encontrar en el siguiente enlace: https://www.django-rest-framework.org/api-guide/testing/

Se realizaron diferentes pruebas para comprobar tanto la obtención y existencia de los usuarios en la Base de Datos con respecto de los que devuelve DRF, así como pruebas para garantizar la respuesta correcta empleando el verbo GET de una petición HTTP de todos los *endpoint* de la API. Finalmente se realizaron pruebas para garantizar el funcionamiento correcto del *endpoint* del módulo de carga de prácticas para el formato A73/83.

De manera general, todas las pruebas con Django y DRF hacen uso de la *APIClient* proporcionada por el DRF, autenticar a un usuario, lanzar una petición HTTP con el cliente y corroborar que el estatus de esta petición sea exitoso, es decir, que tenga un código HTTP_200_OK. Para lograr el último paso se emplea el método assertEqual que se incluye en la paquetería de *Django TestCase*, el cual tiene una estructura como la siguiente para su funcionamiento:

```
class TestClass(BaseTest):
  def func(self):
    ...
    self.assertEqual(<client_status_response>, <expected_http_res>)
    ...
```

assertEqual hace la comparación entre el valor obtenido en el primer parámetro y el segundo parámetro, de manera que si los valores no son iguales la prueba realizada falla. También se hace uso del método *assertTrue* que recibe una expresión y si ésta es falsa entonces la prueba falla. [53]

Las pruebas unitarias de carga de prácticas consisten en autenticar un usuario, cargar un archivo CSV de prueba, guardarlo en la Base de Datos y después corroborar que el número de registros antes y después de ejecutar la petición al *endpoint* de cargar las prácticas, hayan incrementado.

A continuación, se muestra el código realizado para implementar las pruebas unitarias; primero se establece una clase padre llamada *BaseTest* que herede la clase *APITestCase* e implemente las configuraciones de crear un cliente con *APIClient* y asignar el usuario que se usará para esta actividad.

Posteriormente, se creó una clase hija llamada *CheckAPITest*, que hereda a la clase *BaseTest*, donde cada uno de sus métodos es un tipo de prueba a realizar.

```
class BaseTest(APITestCase):
  def setUp(self):
    self.client = APIClient()
    self.user = User.objects.get(username='<test_user>')

class CheckAPITest(BaseTest):
  def test_users_count(self):
    url = '/api/users/'
    self.client.force_authenticate(self.user)
    response = self.client.get(url)
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(User.objects.count(), len(response.data))

  def test_get_api(self):
    self.client.force_authenticate(self.user)
    self.assertEqual(
      self.client.get('/api/<endpoint>/').status_code, status.HTTP_200_OK
    )
    ...

  def test_carga_practicas_get(self):
    url = "/api/<endpoint-to-load-practices>/"
    self.client.force_authenticate(self.user)
    response = self.client.get(url)
    self.assertTrue(response)
    self.assertEqual(response.status_code, status.HTTP_200_OK)


  def test_carga_practicas_post(self)
    url = "/api/<endpoint-to-load-practices>/"
    self.client.force_authenticate(self.user)
    last_periodo = Periodo.objects.last()
    initial_count = A7383Practica.objects.count()
    ...
    data = open("<path-to-test-csv-file>", 'rb')
```

```
csv_file = SimpleUploadedFile(
  content = data.read(),
  name = data.name,
  content_type = "multipart/form-data"
)
response = self.client.post(url, csv_file, ... )
self.assertTrue(response)
self.assertEqual(response.status_code, status.HTTP_200_OK)
after_count = self.client.get(a7383url).json()
self.assertTrue(len(after_count) > initial_count)
```

Para ejecutar el código anterior para las pruebas unitarias se utiliza el siguiente comando:

```
>./manage.py test -k
```

Figura 5.1. Demostración de ejecución de pruebas unitarias con Django y DRF.

Como se demuestra en la Figura 5.1. al finalizar la ejecución de las pruebas, por defecto únicamente mostrará salida en la terminal si existen errores, de otra manera solo los ejecuta y muestra la cantidad de pruebas realizadas y el tiempo que tomó hacerlas.

5.2. Pruebas de E2E

Las pruebas *End-To-End* (E2E) o de integración explicadas en la Sección 2.7.2. permiten validar la funcionalidad de un conjunto de componentes, por lo que se optó por realizarlas a los módulos de la aplicación de *Frontend*, es decir, con el *Framework* Angular, debido a que hay una gran interacción de varios componentes para lograr una determinada tarea.

Las pruebas de E2E se hicieron para el módulo de Inicio de Sesión, descrito en la Sección 4.3, y el módulo de Captura de Formatos, explicado en la Sección 4.4.

Para la realización de estas pruebas en Angular se hace uso de un *Framework* auxiliar llamado *Protractor*, el cual hace uso de un *WebDriver*[24] para hacer todas las tareas programadas en los siguientes archivos:

```
app.po.ts
app.e2e-spect.ts
```

Además, *Protractor* emplea por defecto otro *Framework* de pruebas llamado *Jasmine* para crear la interfaz de pruebas. [54]

El primer archivo mencionado se emplea para definir las clases a exportar para cada uno de los módulos que se van a probar; dentro de cada clase se definen métodos convenientes, como la obtención de ciertos elementos y la navegación entre rutas del sistema. Dicha estructura se ejemplifica en con el siguiente código:

```
import { browser, by, element } from 'protractor';
export class AppPage {
  navigateTo() {
    return browser.get('/');
  }

  getParagraphText() {
    return element(by.css('.title-text')).getText();
  }
}

export class LoginPage {
  navigateTo() {
    return browser.get('/#/login');
  }

  getParagraphText() {
    return element(by.css('h2.title-text')).getText();
  }

  getEmailTextbox() {
    return element(by.css("input[formControlName=username]"));
  }
```

[24] Un *WebDriver* es una interfaz que sirve para manipular el DOM en un documento web y controlar el comportamiento de un agente usuario https://www.w3.org/TR/webdriver/

```
  ...
}
```

Por otra parte, el archivo *app.e2e-spect.ts* importa las clases definidas en *app.po.ts* y el *Framework Protractor* y, a través de la sintaxis de *Jasmine*, describe las funcionalidades y resultados deseados. El siguiente fragmento de código define lo explicado anteriormente:

```
describe('LOGIN PAGE', () => {
  let page: LoginPage;

  beforeEach(() => {
    page = new LoginPage();
  });

  it('Should login', () => {
    page.navigateTo();
    let email = page.getEmailTextbox()
    let pass = page.getPasswordTextbox()
    let btn = page.getLoginButton()
    expect(email).toBeTruthy()
    expect(pass).toBeTruthy()
    expect(btn).toBeTruthy()

    email.sendKeys('<test_user>');
    pass.sendKeys('<user_password>');

    btn.click()
    let modal = page.getLoginModal()

    modal.isPresent().then(function (result) {
      if (result) {
        expect(modal.getText()).not.toContain("Inválido")
      } else {
        expect(modal.isPresent()).toBeFalsy()
      }
    });
  });
});

describe('CAPTURAR PAGE', () => {
  let page: CapturarPage;

  beforeEach(() => {
    page = new CapturarPage();
  });

  it('Have a subtitle', () => {
    page.navigateTo();
    expect(page.getParagraphText().getText()).toBeTruthy()
```

```
  });

  it('should count the number of option', () => {
    expect(page.getOptionsCount()).toBe(9);
  });

  it('Should count a2131 options', () => {
    page.getOption2().click();
    expect(page.getA2131Count()).toBe(9)
  });

  it('Should have a nombre field', () => {
    page.getA2131Option2().click();
    expect(page.getInput("nombre")).toBeTruthy()
  });
});
```

Para ejecutar las pruebas E2E se ingresa el comando ng e2e, el cual ejecutará el test en un navegador web y realizará las interacciones de forma automática haciendo uso del *WebDriver*, como se muestra en la Figura 5.2.

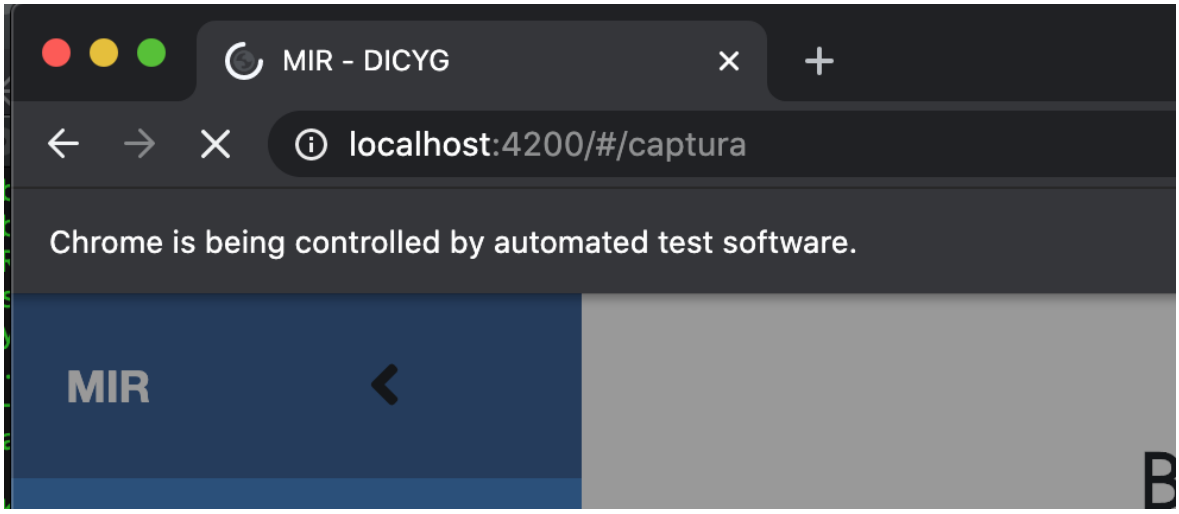


Figura 5.2. Demostración de ejecución de pruebas E2E con Angular.

Al finalizar se muestra en la terminal un reporte de los test ejecutados, tal como se muestra en la Figura 5.3.

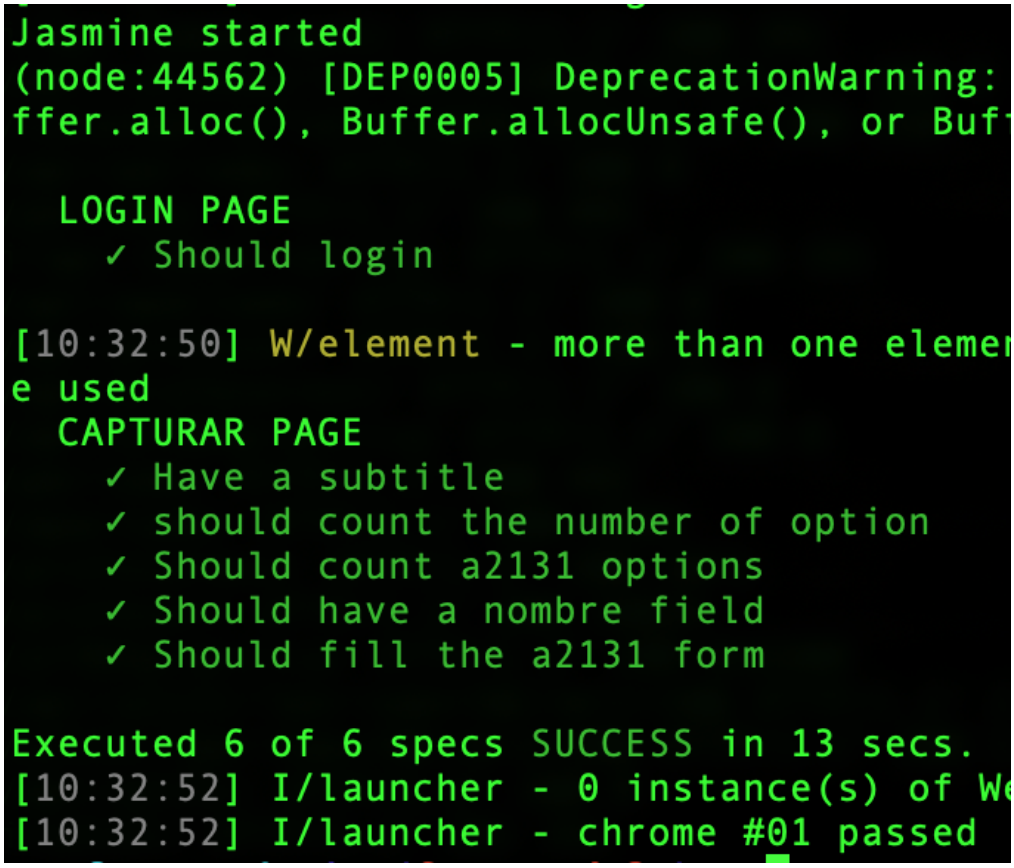


Figura 5.3. Reporte de ejecución de pruebas E2E con Angular.

5.3. Validación de los usuarios

El sistema web realizado en este proyecto fue validado a través de pruebas unitarias y de integración descritas en las Secciones 5.1. y 5.2. Sin embargo, una vez que fue desplegado y habilitado para su uso, los usuarios han tenido oportunidad de interactuar con el mismo y proporcionar una serie de comentarios y retroalimentaciones en función de su experiencia; algunos de los comentarios recibidos son los siguientes:

M.I. Octavio García Domínguez:

> En relación con el uso del sistema desarrollado por la Unidad de Cómputo de esta División para la captura de la información académica requerida por la MIR le comento que ha sido de gran ayuda ya que, en el pasado reciente, esta tarea era tediosa y se capturaba de forma manual en formatos que cada área diseñaba acorde a las particularidades del tipo de actividad reportada.

A partir del sistema con que se cuenta actualmente, no sólo se puede capturar más rápido la información e incorporar más fácilmente las evidencias de las actividades en archivos electrónicos, sino que se pueden generar automáticamente reportes en un formato oficial, para ser entregados a la Facultad.

El sistema ha mostrado sus beneficios y se ha ido adaptando a los requerimientos de las áreas. Sería deseable poder contar con más desarrollos como este para atender otras tareas en las que se requieren un manejo eficiente y preciso de la información.

Muchas gracias por su apoyo.

M.I. Octavio García Domínguez
Jefe del Departamento de Estructuras
DICyG, FI-UNAM

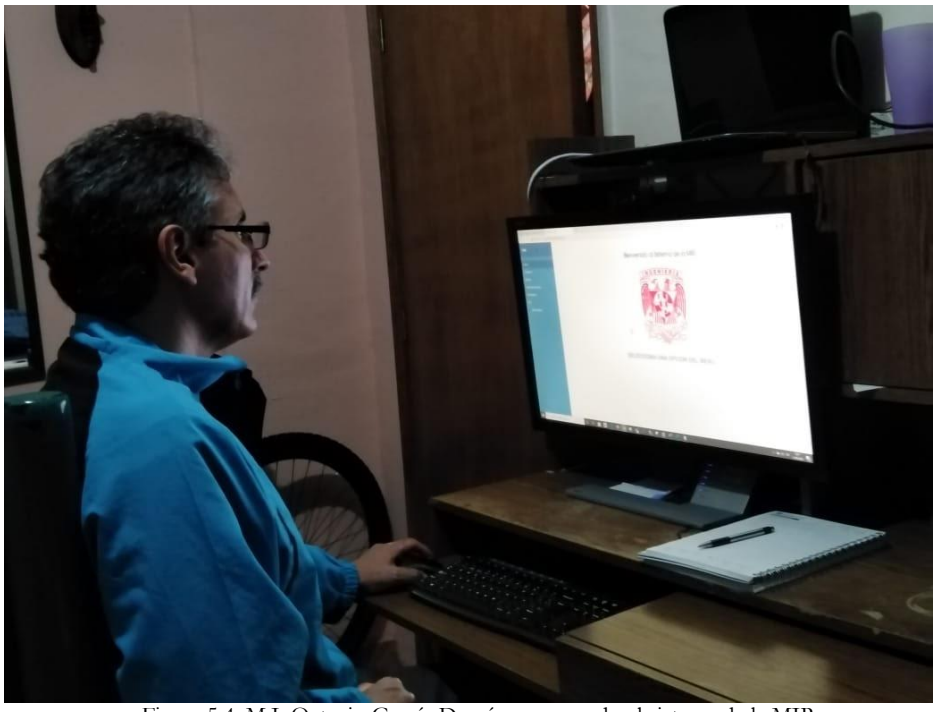


Figura 5.4. M.I. Octavio García Domínguez usando el sistema de la MIR.

Ing. Roberto Carlos De La Cruz Sánchez:

La Matriz de Indicadores para Resultados (MIR) es una herramienta de planeación que en forma resumida, sencilla y armónica establece con claridad los objetivos de un programa, incorpora los indicadores que miden dichos objetivos y sus resultados esperados.
Opino que el sistema permite el acceso de una manera amigable y la descarga de información es sencilla una vez que se ubica el rubro que se va a trabajar. La subida a la nube de las evidencias también es de fácil acceso.

Ing. Roberto Carlos De La Cruz Sánchez
Jefe del Departamento de Topografía
DICyG, FI-UNAM

Figura 5.5. Ing. Roberto Carlos De La Cruz Sánchez usando el sistema de la MIR.

CAPÍTULO 6. CONCLUSIONES

El objetivo de la Matriz de Indicadores de Resultados (MIR), es medir la productividad que tiene la UNAM de manera que se justifique y demuestre el aprovechamiento de los recursos asignados por parte del Erario Público, por medio del desempeño de sus actividades académicas y científicas.

El presupuesto proporcionado a la Universidad por parte del Gobierno Federal es derivado del cumplimiento de sus actividades, por lo que se debe garantizar que este proceso sea realizado de manera correcta y a tiempo. Por lo tanto, fue necesario implementar un sistema que asegure cumplir con los objetivos planteados de facilitar el llenado, la validación y la generación de reportes de la MIR ante la Facultad de Ingeniería y de tener un mejor control de los datos capturados para realizar históricos, estadísticas y revisiones de manera sencilla, lo cual se consiguió con el sistema web desarrollado.

Algunas de las características a resaltar son las siguientes:

- La disponibilidad de la información en un medio electrónico.
- La implementación de la firma electrónica que permitió tener una constancia de autenticidad de la información, de manera que puede ser validada y realizada de manera electrónica.
- Se facilitó el cambio de personal cuando se requiere, sin perder ningún tipo de información.
- Se tuvo muy buena aceptación por parte de los usuarios del sistema durante su desarrollo y se realizaron mejoras constantes gracias al uso temprano de nuestros usuarios y su retroalimentación.

Adicionalmente, el uso de herramientas tecnológicas como Django, Angular y Git permitieron la flexibilidad al realizar ajustes durante el desarrollo y resolverlos de manera efectiva, sin perder la línea base con la que se planeó el proyecto. Y dado que se tuvo siempre en mente la mantenibilidad, escalabilidad y replicación de este proyecto a diferentes Divisiones o a nivel Facultad, se dejó la posibilidad a futuras expansiones y/o adecuaciones por parte de otras Divisiones que se puedan ver beneficiadas, ya sea por la automatización del proceso de captura y generación de informes o por la generación de firmas electrónicas para validar sus documentos.

Esta disponibilidad y uso del sistema para otras dependencias, se permite debido a la cesión de derechos del código fuente a la propia Unidad de Cómputo de la División de Ingenierías Civil y Geomática bajo el licenciamiento MIT. En caso de requerirse acceso al mismo, se puede poner en contacto con la directora de la presente tesis y se anexa a continuación:

Contacto de la Jefa de la Unidad de Cómputo, la M.I. Tanya Itzel Arteaga Ricci para licenciamiento: tanya.arteaga@ingenieria.unam.edu

GLOSARIO

1. Matriz de Indicadores para Resultados (MIR).
2. División de Ingenierías Civil y Geomática (DICyG).
3. Software Development Life Cycle (SDLC).
4. Software Requirement Specification (SRS).
5. Design Document Specification (DDS).
6. World Wide Web (WWW).
7. HyperText Markup Language (HTML).
8. Cascading Style Sheets (CSS).
9. User Interface (UI).
10. User Experience (UX).
11. Multi-Page Applications (MPA).
12. Single-Page Applications (SPA).
13. Asynchronous JavaScript And XML (AJAX).
14. Application Programming Interface (API).
15. Model View Controller (MVC).
16. Model View Template (MVT).
17. HyperText Transfer Protocol (HTTP).
18. Document Object Model (DOM).
19. Object Oriented Programming (OOP).

20. Berkeley Software Distribution (BSD).
21. Secure Hash Algorithm 2 (SHA-2).
22. Secure SHell (SSH).
23. Transport Layer Security (TLS).
24. Secure Sockets Layer (SSL).
25. Pretty Good Privacy (PGP).
26. American Standard Code for Information Interchange (ASCII).
27. JavaScript Object Notation (JSON).
28. JSON Web Token (JWT).
29. Universally Unique IDentifier (UUID).
30. Sistema de Control de Versiones (SCV).
31. Unified Modeling Language (UML).
32. Tiempo Universal Coordinado (UTC).
33. Object-Relational Mapping (ORM).
34. Web Server Gateway Interface (WSGI).

BIBLIOGRAFÍA

[1] M. Stoica, M. Mircea y B. Ghilic-Micu, «Software Development: Agile vs. Traditional,» *Informatica Economica,* pp. 1-13, 2013.

[2] «RYTE WIKI,» [En línea]. Available: https://es.ryte.com/wiki/Modelo_en_Espiral. [Último acceso: 03 Abril 2020].

[3] A. Navarro Cadavid, J. D. Fernández Martínez y J. Morales Vélez, «Revisión de metodologías ágiles para el desarrollo de software,» *PROSPECTIVA,* vol. 11, pp. 1-11, 2013.

[4] «Agile Alliance - What is Kanban?,» 15 01 2010. [En línea]. Available: https://www.agilealliance.org/glossary/kanban. [Último acceso: 04 Abril 2020].

[5] S. S. McPherson, Tim Berners-Lee: Inventor of the World Wide Web., Minneapolis: Twenty-First Century Books, 2009.

[6] «A Brief History of Web Browsers and How They Work,» crossbrowsertesting, 28 Enero 2018. [En línea]. Available: https://crossbrowsertesting.com/blog/test-automation/history-of-web-browsers/. [Último acceso: 06 Abril 2020].

[7] D. Cassel, «Brendan Eich on Creating JavaScript in 10 Days, and What He'd Do Differently Today,» thenewstack, 26 Agosto 2018. [En línea]. Available: https://thenewstack.io/brendan-eich-on-creating-javascript-in-10-days-and-what-hed-do-differently-today/. [Último acceso: 06 Abril 2020].

[8] P. Sebastian, «A Brief History of JavaScript,» auth0, 16 Enero 2017. [En línea]. Available: https://auth0.com/blog/a-brief-history-of-javascript/. [Último acceso: 06 Abril 2020].

[9] «Typescript,» Microsoft, 2020. [En línea]. Available: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html. [Último acceso: 06 Abril 2020].

[10] N. Chapaval, «Qué es Frontend y Backend,» Platzi, 2018. [En línea]. Available: https://platzi.com/blog/que-es-frontend-y-backend/. [Último acceso: 06 Abril 2020].

[11] «User Experience (UX) Design,» Interaction Design Foundation, [En línea]. Available: https://www.interaction-design.org/literature/topics/ux-design. [Último acceso: 07 Abril 2020].

[12] «A Blended Single Page and Multi Page Application Approach,» a2i2, 1 Marzo 2019. [En línea]. Available: https://a2i2.deakin.edu.au/2019/03/01/a-blended-single-page-and-multi-page-application-approach/. [Último acceso: 07 Abril 2020].

[13] M. Wasson, «ASP.NET - Single-Page Applications,» Microsoft, 01 Noviembre 2013. [En línea]. Available: https://docs.microsoft.com/en-us/archive/msdn-magazine/2013/november/asp-net-single-page-applications-build-modern-responsive-web-apps-with-asp-net. [Último acceso: 07 Abril 2020].

[14] C. Heike, «LAMP vs. MEAN: Which stack is right for you?,» Bitbucket, 02 Mayo 2019. [En línea]. Available: https://bitbucket.org/blog/lamp-vs-mean-which-stack-is-right-for-you. [Último acceso: 07 Abril 2020].

[15] «Modelo vista controlador (MVC),» Universidad de Alicante, [En línea]. Available: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html. [Último acceso: 07 Abril 2020].

[16] «Django MVT,» javatpoint, [En línea]. Available: https://www.javatpoint.com/django-mvt. [Último acceso: 07 Abril 2020].

[17] «¿Qué es React y para qué sirve?,» drauta, 27 Enero 2020. [En línea]. Available: https://www.drauta.com/que-es-react-y-para-que-sirve#. [Último acceso: 08 Abril 2020].

[18] «React,» Facebook, 2020. [En línea]. Available: https://reactjs.org/. [Último acceso: 07 Abril 2020].

[19] «Angular About,» Google, 2020. [En línea]. Available: https://angular.io/about?group=Angular. [Último acceso: 07 Abril 2020].

[20] «Angular Architecture,» Google, 2020. [En línea]. Available: https://angular.io/guide/architecture. [Último acceso: 07 Abril 2020].

[21] S. Willison, «Quora,» Quora, 05 Enero 2014. [En línea]. Available: https://www.quora.com/What-is-the-history-of-the-Django-web-framework-Why-has-it-been-described-as-developed-in-a-newsroom. [Último acceso: 07 Abril 2020].

[22] «Django Project,» Django Software Foundation, 2020. [En línea]. Available: https://docs.djangoproject.com/en/2.2/misc/design-philosophies/. [Último acceso: 07 Abril 2020].

[23] «About Node.js,» training.com, 11 Septiembre 2016. [En línea]. Available: http://blog.training.com/2016/09/about-nodejs-and-why-you-should-add.html. [Último acceso: 07 Abril 2020].

[24] «Reasons to Choose Debian,» Debian, 31 May 2020. [En línea]. Available: https://www.debian.org/intro/why_debian. [Último acceso: 01 Junio 2020].

[25] C. Pontikis, «Five Reasons to use Debian as a Server,» pontikis.net, 25 Febrero 2013. [En línea]. Available: https://www.pontikis.net/blog/five-reasons-to-use-debian-as-a-server. [Último acceso: 07 Abril 2020].

[26] «CentOS-2 Final finally released,» The CentOS Project, 2019. [En línea]. Available: https://lists.centos.org/pipermail/centos/2004-May/000153.html. [Último acceso: 08 Abril 2020].

[27] J. Meier y A. Mackman, «Chapter 1- Web Application Security Fundamentals,» Microsoft, 2003. [En línea]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648636(v=pandp.10). [Último acceso: 08 Abril 2020].

[28] IETF, «A Universally Unique IDentifier (UUID) URN Namespace,» julio 2005. [En línea]. Available: https://www.ietf.org/rfc/rfc4122.txt. [Último acceso: junio 2020].

[29] Kaspersky, «¿Qué es el cifrado de datos?,» [En línea]. Available: https://latam.kaspersky.com/resource-center/definitions/encryption. [Último acceso: junio 2020].

[30] W. Penard y T. van Werkhoven, «The Wayback Machine,» [En línea]. Available: https://web.archive.org/web/20160330153520/http://www.staff.science.uu.nl/~werkh108/docs/study/Y5_07_08/infocry/project/Cryp08.pdf. [Último acceso: junio 2020].

[31] IETF, «PKCS #5: Password-Based Cryptography Specification,» septiembre 2000. [En línea]. Available: https://tools.ietf.org/html/rfc2898#section-5.2. [Último acceso: junio 2020].

[32] django, «Password management in Django,» [En línea]. Available: https://docs.djangoproject.com/en/3.0/topics/auth/passwords/. [Último acceso: junio 2020].

[33] IETF, «JSON Web Token (JWT),» mayo 2015. [En línea]. Available: https://tools.ietf.org/html/rfc7519. [Último acceso: junio 2020].

[34] IETF, «The Base16, Base32, and Base64 Data Encodings,» octubre 2006. [En línea]. Available: https://tools.ietf.org/html/rfc4648. [Último acceso: junio 2020].

[35] Instituto Nacional Electoral, «Sistema Integral de Fiscalización,» [En línea]. Available: https://portalanterior.ine.mx/archivos2/tutoriales/sistemas/ApoyoInstitucional/SIF/docs/candidatos/folioFiscalXML.pdf. [Último acceso: junio 2020].

[36] Amazon Web Services, «¿Qué es una base de datos relacional?,» [En línea]. Available: https://aws.amazon.com/es/relational-database/. [Último acceso: junio 2020].

[37] PostgreSQL, «What is PostgreSQL?,» [En línea]. Available: https://www.postgresql.org/about/. [Último acceso: junio 2020].

[38] Amazon Web Services, «Amazon RDS for PostgreSQL,» [En línea]. Available: https://aws.amazon.com/es/rds/postgresql/. [Último acceso: junio 2020].

[39] SQLite, «About SQLite,» [En línea]. Available: https://www.sqlite.org/about.html. [Último acceso: junio 2020].

[40] Amazon Web Services, «¿Qué es NoSQL?,» [En línea]. Available: https://aws.amazon.com/es/nosql/. [Último acceso: junio 2020].

[41] mongoDB, «10gen Announces Company Name Change to MongoDB, Inc.,» 27 agosto 2013. [En línea]. Available: https://www.mongodb.com/press/10gen-announces-company-name-change-mongodb-inc. [Último acceso: junio 2020].

[42] mongoDB, «mongoDB Documentation,» [En línea]. Available: https://docs.mongodb.com/manual/introduction/. [Último acceso: junio 2020].

[43] git, «1.1 Inicio - Sobre el Control de Versiones - Acerca del Control de Versiones,» [En línea]. Available: https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones. [Último acceso: junio 2020].

[44] git, «1.2 Getting Started - A Short History of Git,» [En línea]. Available: https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git. [Último acceso: junio 2020].

[45] atlassian, «What is Git,» [En línea]. Available: https://www.atlassian.com/git/tutorials/what-is-git. [Último acceso: junio 2020].

[46] kinsta, «What Is GitHub? A Beginner's Introduction to GitHub,» 2 marzo 2020. [En línea]. Available: https://kinsta.com/knowledgebase/what-is-github/. [Último acceso: junio 2020].

[47] Guru99, «What is Software Testing? Definition, Basics & Types,» [En línea]. Available: https://www.guru99.com/software-testing-introduction-importance.html. [Último acceso: junio 2020].

[48] vsupalov.com, «What Is Gunicorn, and What Does It Do?,» 2019. [En línea]. Available: https://vsupalov.com/what-is-gunicorn/. [Último acceso: 11 07 2020].

[49] webcomponents.org, «What are web components?,» 2020. [En línea]. Available: https://www.webcomponents.org/introduction. [Último acceso: julio 2020].

[50] Angular, «Introduction to Angular concepts,» [En línea]. Available: https://angular.io/guide/architecture. [Último acceso: 09 julio 2020].

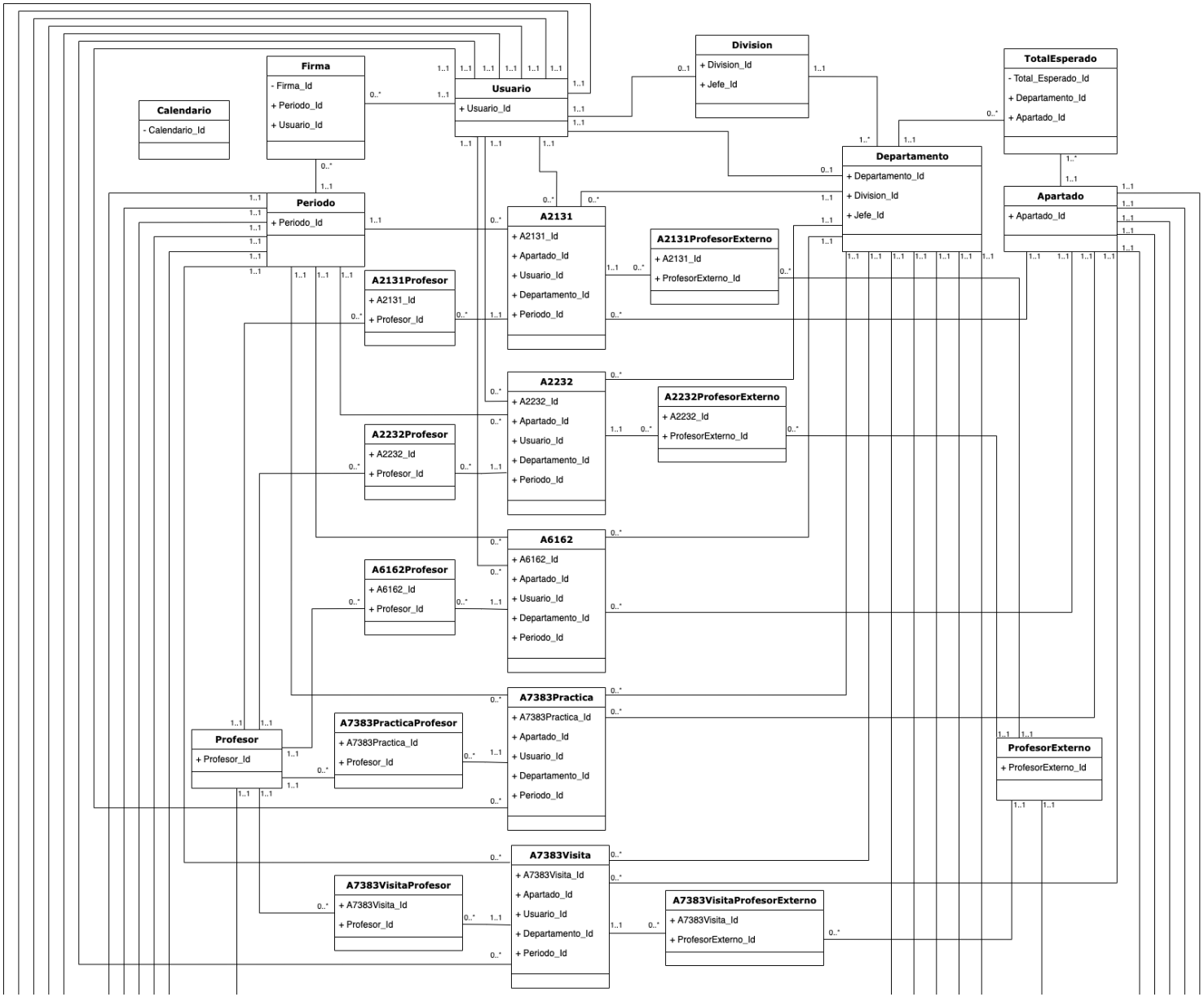
[51] Angular, «Introduction to services and dependency injection,» [En línea]. Available: https://angular.io/guide/architecture-services. [Último acceso: 09 julio 2020].

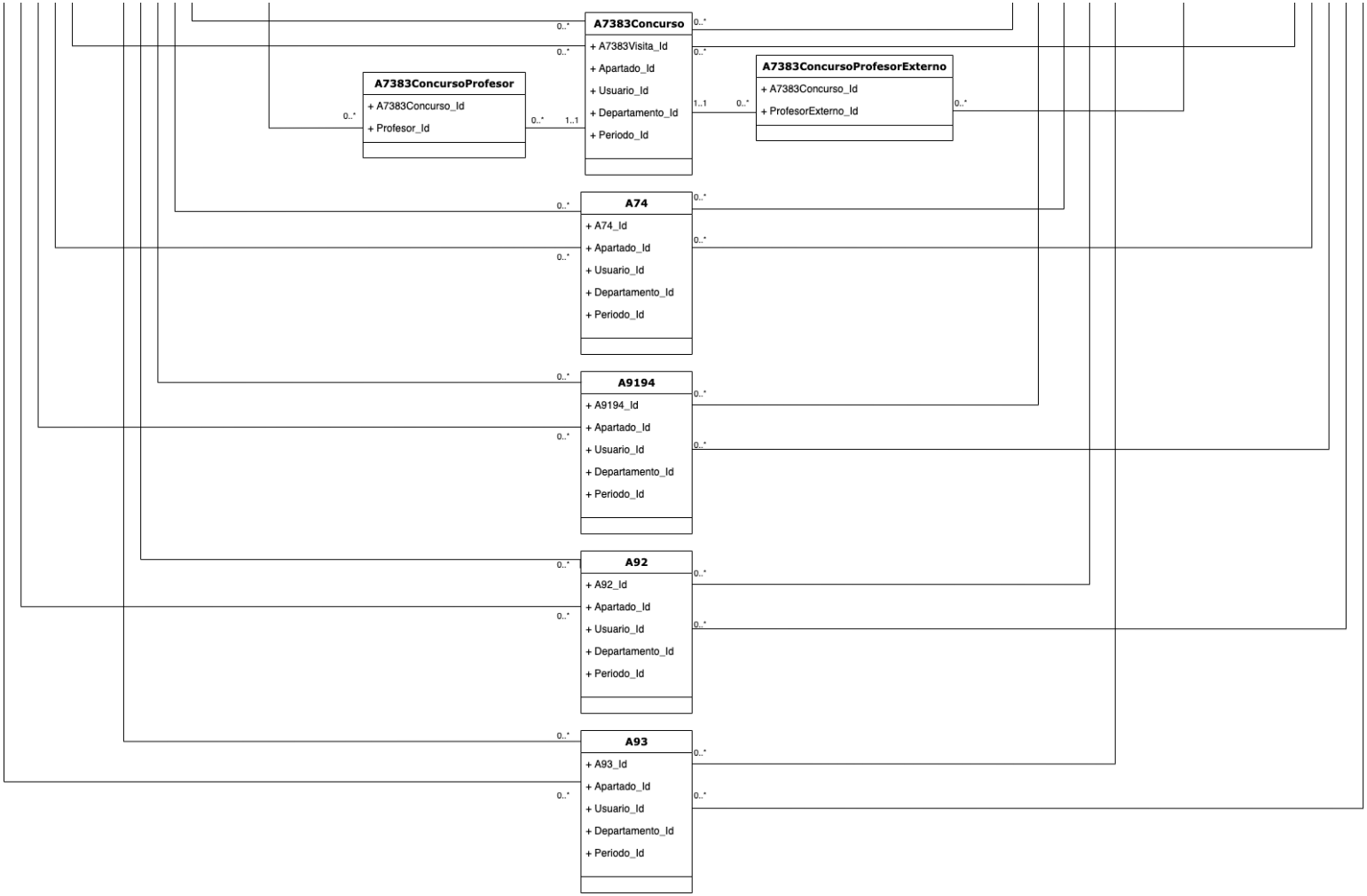
[52] Angular, «Start forms,» [En línea]. Available: https://angular.io/start/start-forms. [Último acceso: 10 julio 2020].

[53] Python, «Unit testing framework,» [En línea]. Available: https://docs.python.org/3/library/unittest.html. [Último acceso: 27 julio 2020].

[54] Protractor, «Protractor end to end testing for Angular,» [En línea]. Available: https://www.protractortest.org/#/. [Último acceso: 25 julio 2020].

ANEXOS


Firma
- Firma_Id
+ Periodo_Id
+ Usuario_Id
Usuario
+ Usuario_Id
Division
+ Division_Id
+ Jefe_Id
TotalEsperado
- Total_Esperado_Id
+ Departamento_Id
+ Apartado_Id
Calendario
- Calendario_Id
Departamento
+ Departamento_Id
+ Division_Id
+ Jefe_Id
Apartado
+ Apartado_Id
Periodo
+ Periodo_Id
A2131
+ A2131_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A2131ProfesorExterno
+ A2131_Id
+ ProfesorExterno_Id
A2131Profesor
+ A2131_Id
+ Profesor_Id
A2232
+ A2232_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A2232ProfesorExterno
+ A2232_Id
+ ProfesorExterno_Id
A2232Profesor
+ A2232_Id
+ Profesor_Id
A6162
+ A6162_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A6162Profesor
+ A6162_Id
+ Profesor_Id
A7383Practica
+ A7383Practica_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
Profesor
+ Profesor_Id
A7383PracticaProfesor
+ A7383Practica_Id
+ Profesor_Id
ProfesorExterno
+ ProfesorExterno_Id
A7383Visita
+ A7383Visita_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A7383VisitaProfesor
+ A7383Visita_Id
+ Profesor_Id
A7383VisitaProfesorExterno
+ A7383Visita_Id
+ ProfesorExterno_Id


A7383Concurso
+ A7383Visita_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A7383ConcursoProfesor
+ A7383Concurso_Id
+ Profesor_Id
A7383ConcursoProfesorExterno
+ A7383Concurso_Id
+ ProfesorExterno_Id
A74
+ A74_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A9194
+ A9194_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A92
+ A92_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id
A93
+ A93_Id
+ Apartado_Id
+ Usuario_Id
+ Departamento_Id
+ Periodo_Id

Manual de usuario del Sistema de Matriz de Indicadores para Resultados (MIR) de la DICyG

Unidad de Cómputo

05/10/2020

Índice

Acceso

Para acceder al Sistema MIR se necesita entrar a su navegador de preferencia e ingresar la siguiente URL:

http://dicyg.fi-c.unam.mx/~mir/

Posteriormente, se desplegará la vista de inicio de sesión al sistema, donde adicionalmente se despliega la información de las fechas de apertura/disponibilidad de uso del sistema (ver Figura 1.1).

Figura 1.1

En los campos *Usuario* y *Contraseña* se deberán colocar sus credenciales de acceso, que previamente se le otorgaron por parte de la DICyG, y presionar el botón *Acceder*. Lo anterior se describe en la Figura 1.2.

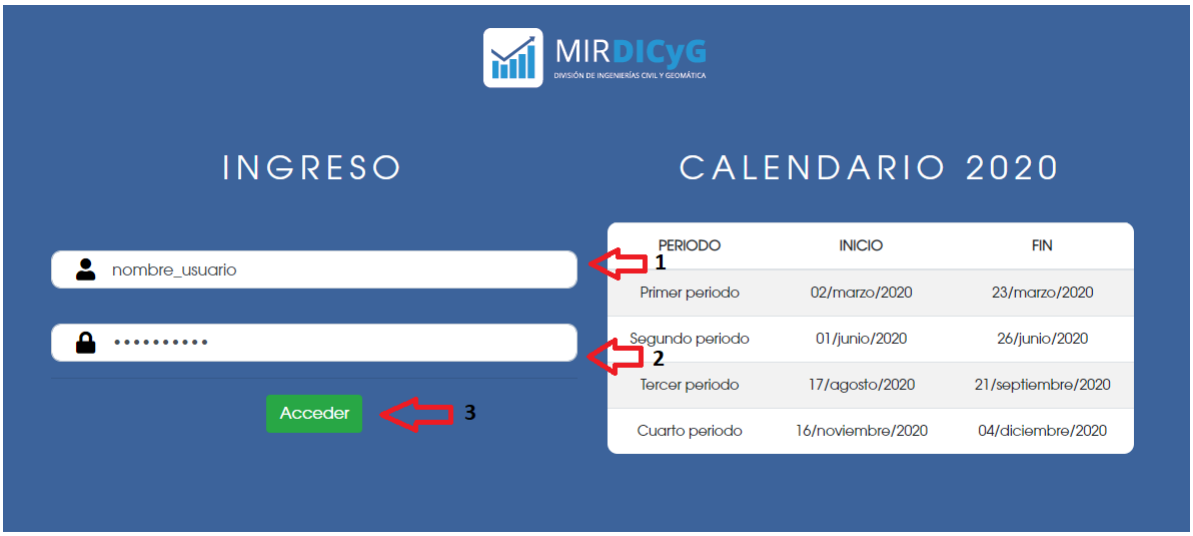


Figura 1.2

En caso de que el sistema se encuentre cerrado debido a que se ha concluido el periodo de uso o debido a que su contraseña es errónea, se mostrará una alerta como la de la Figura 1.3.

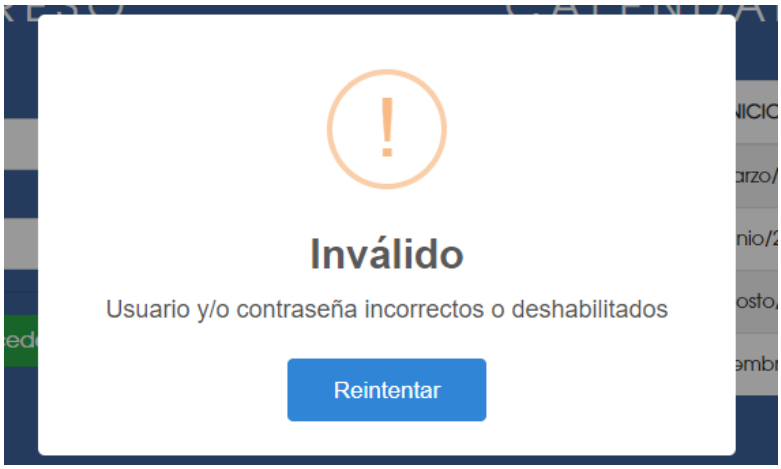


Figura 1.3

Por el contrario, si el sistema está disponible para su uso y las credenciales fueron correctas, se mostrará la vista de inicio (ver Figura 1.4).

Figura 1.4

Se cuenta con las siguientes opciones del menú de navegación:

- Inicio
- Capturar
- Revisar
- Esperados del año
- Estadísticas
- PDF
- Cerrar Sesión

Inicio

En la opción *Inicio* únicamente se desplegará el mensaje de bienvenida al sistema (ver Figura 2.1).

Figura 2.1

Capturar

En la opción *Capturar* se tiene la posibilidad de poder realizar la captura de actividades que se reportarán a la MIR para el Departamento que pertenezca; estos datos los puede corroborar en la sección B de la Figura 3.1.

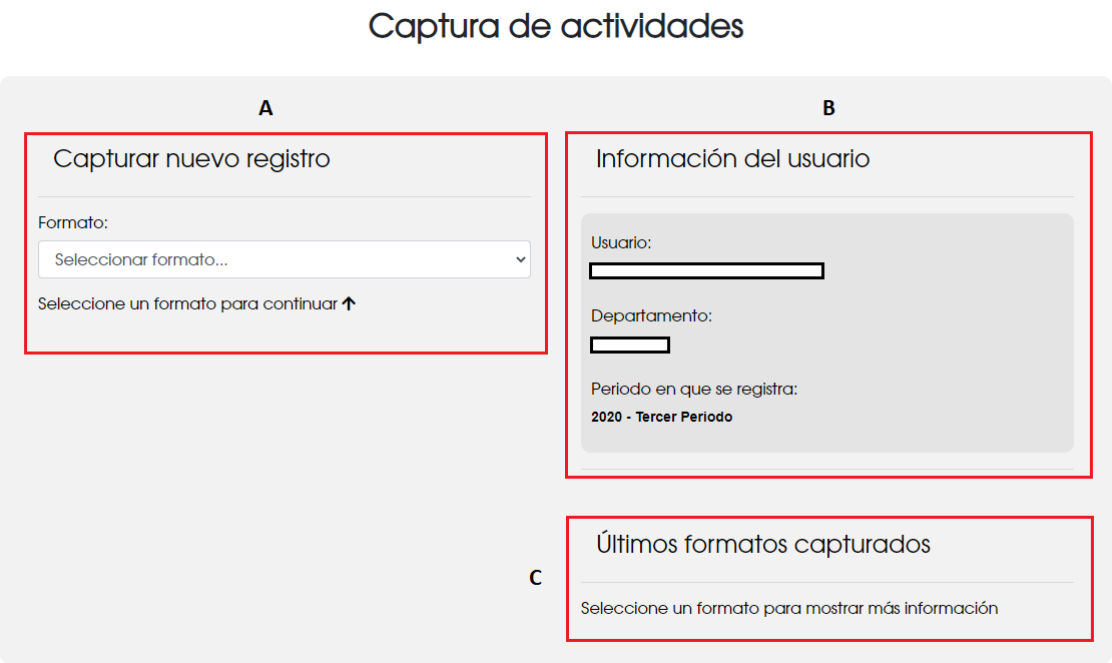


Figura 3.1

En la sección A de la Figura 3.1 se eligen los distintos formatos/apartados en los que se pueden registrar actividades, las cuales de muestran en la Figura 3.2.

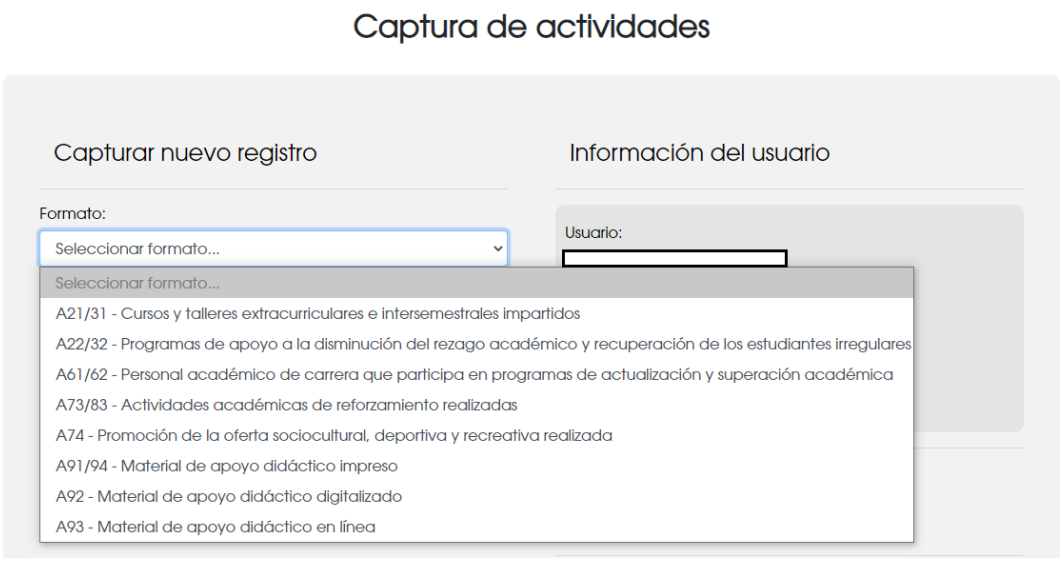


Figura 3.2

Al seleccionar un formato, en la sección C de la Figura 3.1 se mostrarán los últimos registros capturados para el periodo en curso (ver Figura 3.3).

Figura 3.3

De manera general, el proceso de captura es idéntico para todos los formatos. Se mostrará un formulario con todos los campos necesarios para capturarlo.

En el campo *Nombre del curso/taller* se debe escribir en texto, mientras que para los campos de *Fecha de inicio* y *Fecha de finalización* se abrirá un calendario para indicar la fecha (ver Figura 3.5).

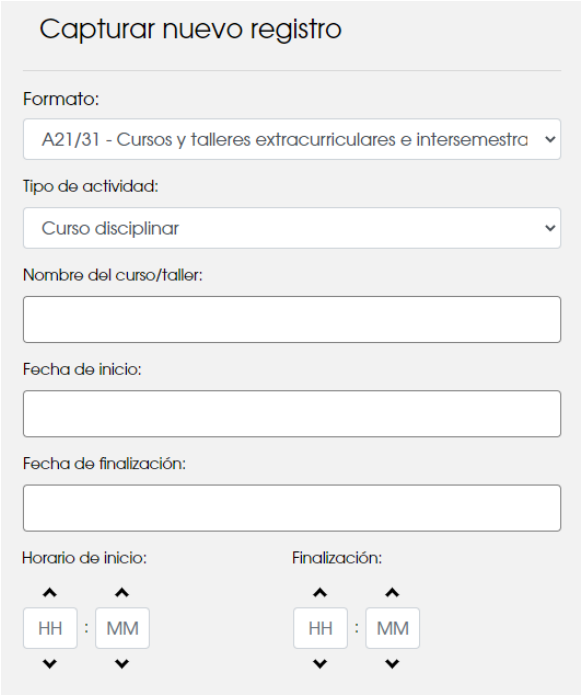


Figura 3.4

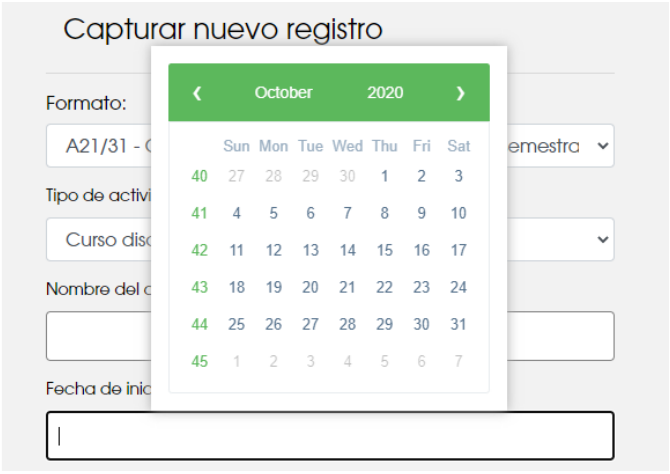


Figura 3.5

El campo de *Horario de inicio* y *Finalización* tiene formato de 24 horas, el cual se puede llenar de manera manual o modificar con las flechas (ver Figura 3.4).

Posteriormente, se cuenta con un campo *Tipo de profesor*, que permite elegir si el profesor que impartió la actividad es interno a la Facultad de Ingeniería o externo (ver Figura 3.6).

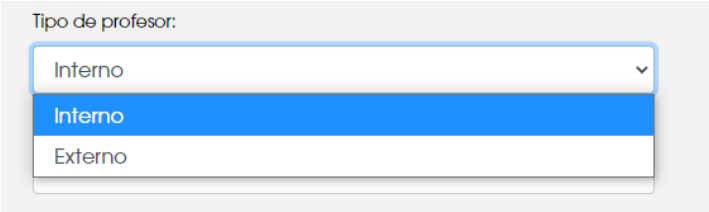


Figura 3.6

Dependiendo de la opción elegida de Profesor Interno o Externo de la Figura 3.6 se mostrará el campo *Profesor Interno que impartió* o *Profesor Externo que impartió*, donde se podrá ir escribiendo el nombre del profesor y se tendrá una ventana de sugerencia y autocompletado (ver Figura 3.7).

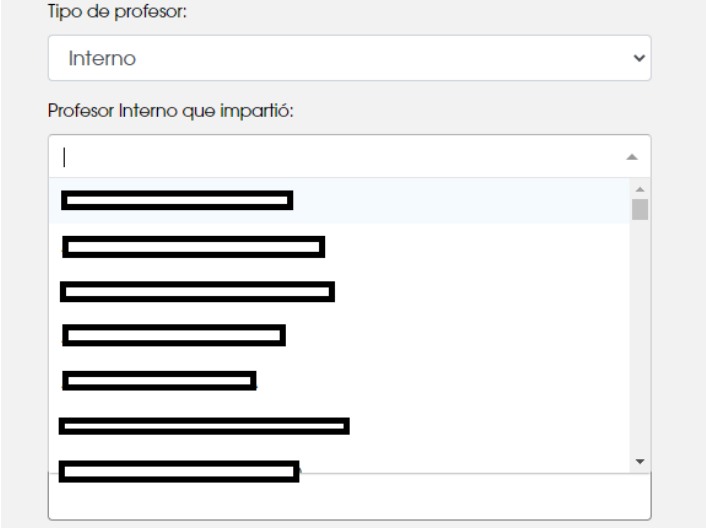


Figura 3.7

Si no se encuentra el profesor deseado, es posible dar clic en *Agregar Profesor Interno* (ver Figura 3.8) para llenar un formulario y dar de alta el profesor (ver Figura 3.9), para después elegirlo.

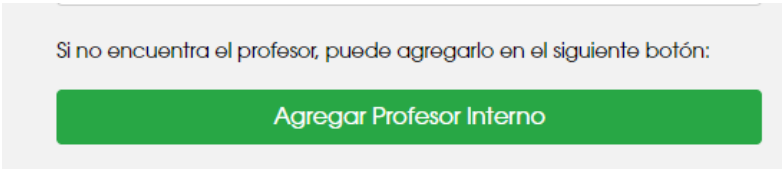


Figura 3.8

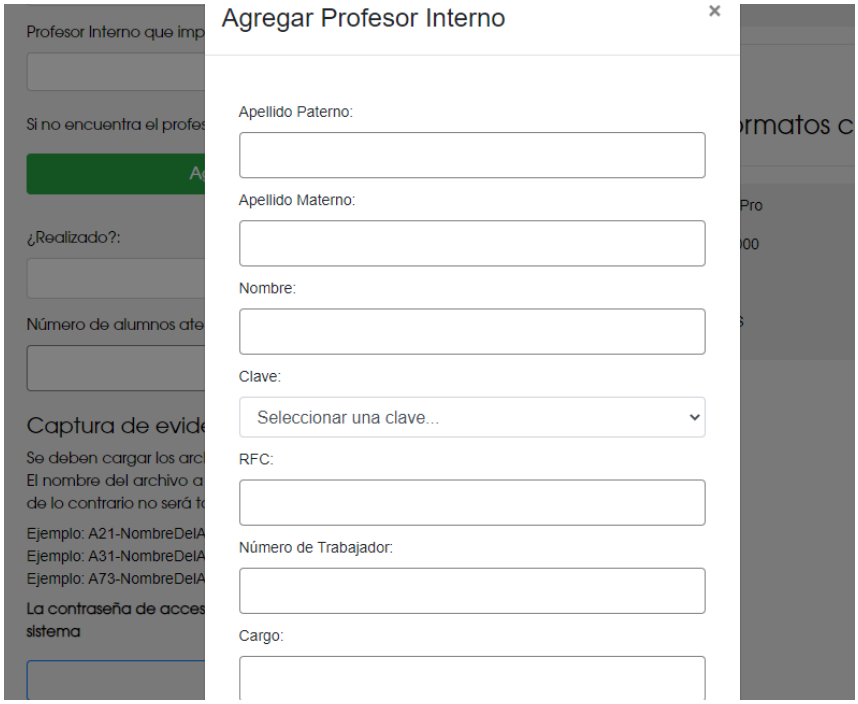


Figura 3.9

Dependiendo el formato elegido, se contará con otros campos como *¿Realizado?* y *Número de alumnos atendidos* (ver Figura 3.10).

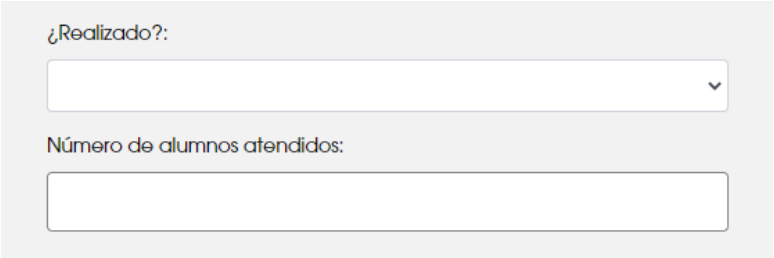


Figura 3.10

El campo *¿Realizado?* sirve para indicar si la actividad fue realizada o no; mientras que el *Número de alumnos atendidos* indicará a la cantidad de personas involucradas en la actividad.

Para todos los formatos, se cuenta al final con la capacidad de subir evidencia a la plataforma CloudDICyG de la DICyG (ver Figura 3.11).

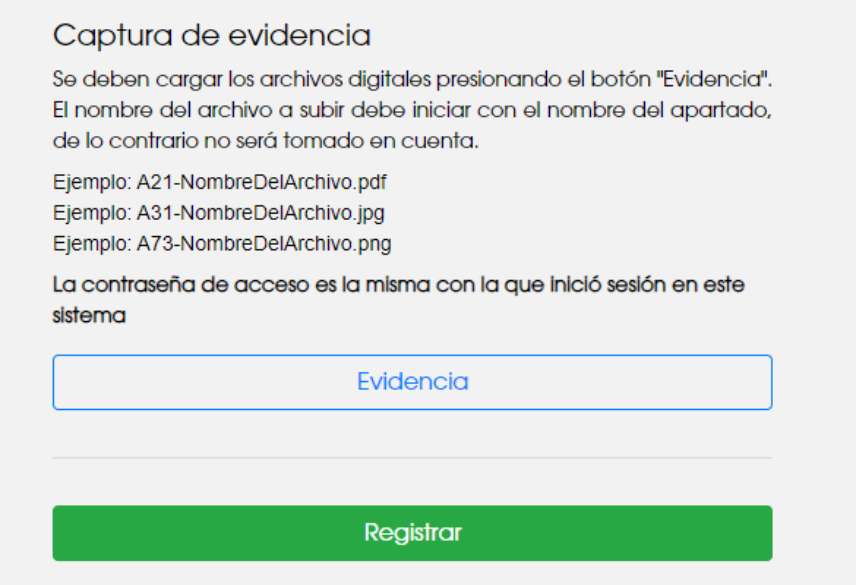


Figura 3.11

Al dar clic en *Evidencia* direccionará a otra pestaña donde se subirán las evidencias correspondientes.

Figura 3.12

Para ingresar se pedirá la contraseña, la cual es la misma que usó para acceder al sistema.

Una vez que se haya accedido al sistema, se mostrarán carpetas para cada uno de los periodos (ver Figura 3.13).

Figura 3.13

Al elegir la carpeta del periodo correspondiente, se podrán cargar los archivos de evidencia dando clic en el símbolo de “+” que se encuentra en la parte superior

izquierda (ver Figura 3.14) o arrastrar directamente los archivos o carpetas a la ventana.

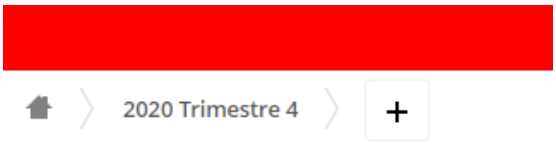


Figura 3.14

Para finalizar, se dará clic en *Registrar*. Si todos los campos fueron llenados correctamente se mostrará un mensaje de éxito (ver Figura 3.15).

Figura 3.15

El procedimiento anterior es idéntico para los diferentes formatos; únicamente para el formato A73/83 se tiene una opción adicional: cargar los registros de prácticas desde un formato CSV (o Excel), debido a que pueden ser muchas ya que son de las actividades más frecuentes en la DICyG.

Para ello, se deberá elegir en *Tipo* la opción de *Prácticas de laboratorio* dentro del formato A7383, con lo que mostrará dos botones extras: *Descargar Formato de Prácticas* y *Subir Formato de Prácticas* (ver Figura 3.16).

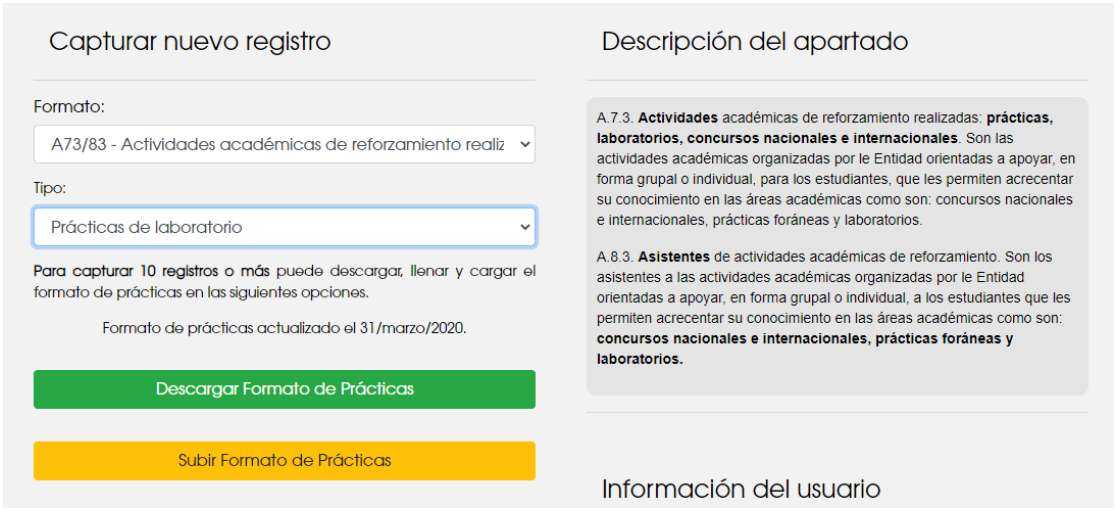


Figura 3.16

Una vez que se tenga el formato descargado y llenado con la información en las columnas indicadas (ver Figura 3.17), se procederá a subirlo al sistema; para ello se dará clic en el botón *Subir Formato de Prácticas* y mostrará una vista como la de la Figura 3.18.

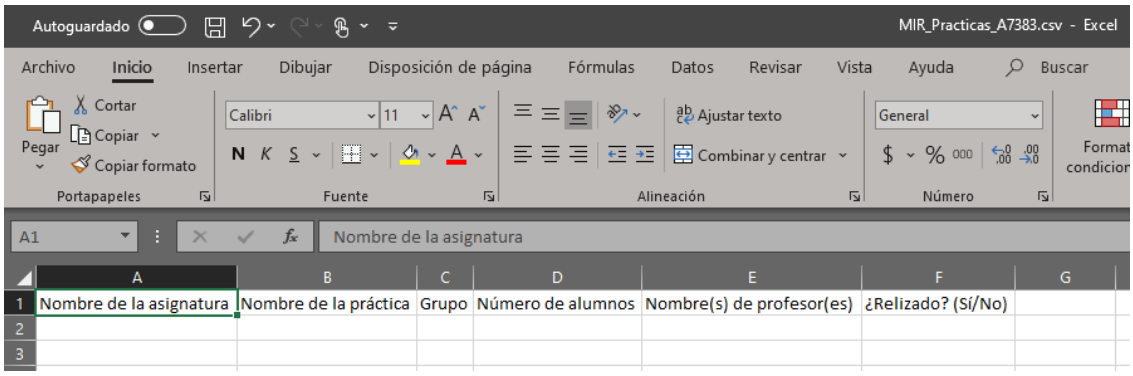


Figura 3.17

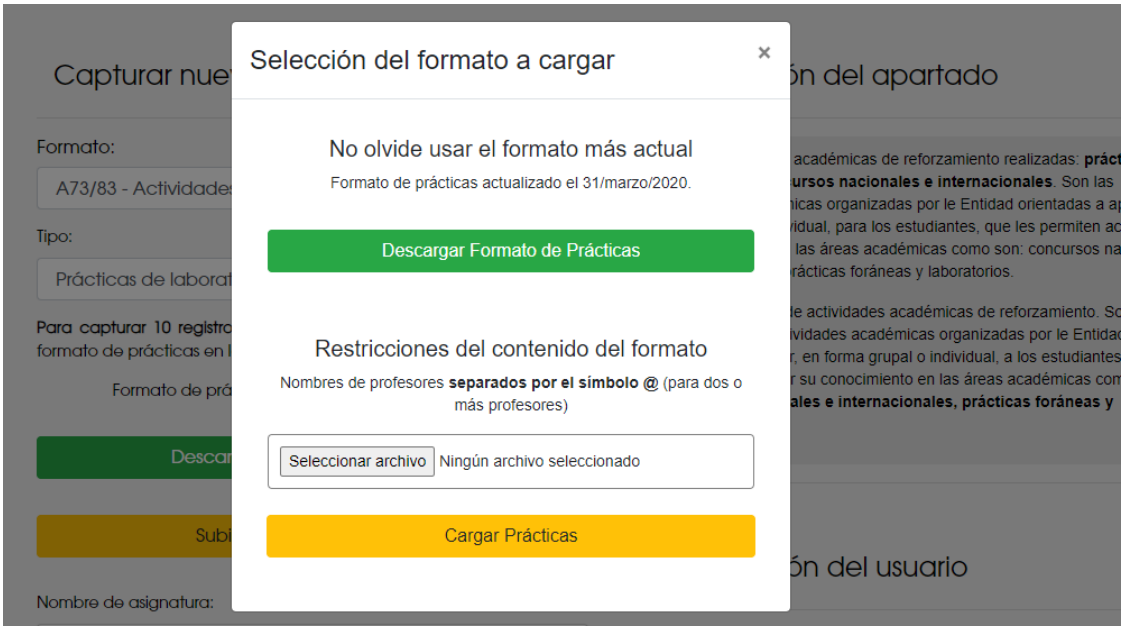


Figura 3.18

Para subir el formato previamente llenado, se dará clic en *Seleccionar archivo*, elegirlo y presionar el botón *Cargar Prácticas* (ver Figura 3.19).

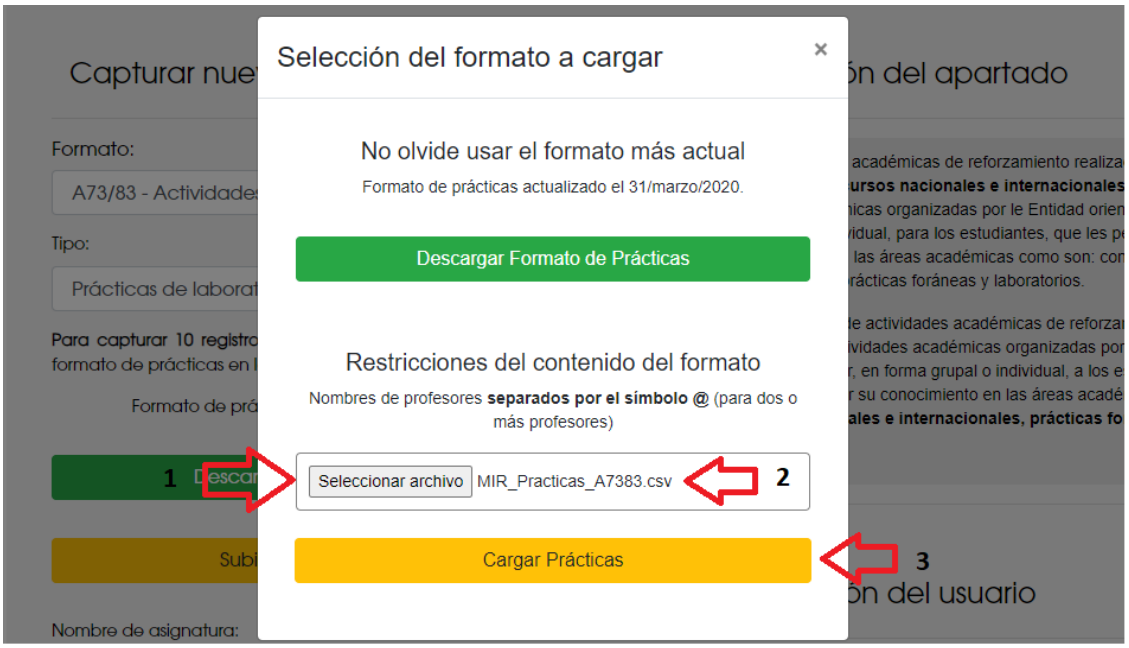


Figura 3.19

Si el proceso de carga resultó exitoso o existió algún error, se mostrará un mensaje dando instrucciones.

Revisar

En la opción *Revisar* es posible hacer una consulta y revisión de los formatos capturados hasta el momento, ya sean de periodos anteriores o del actual.

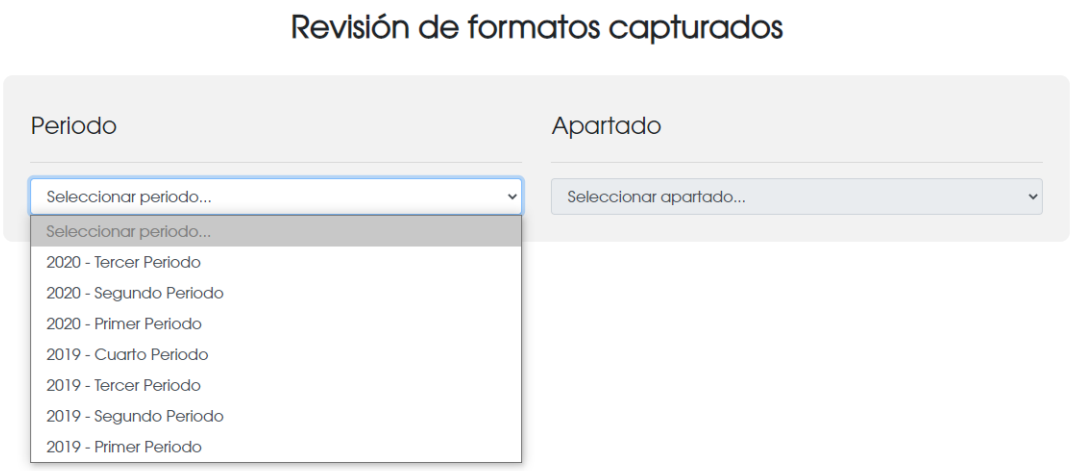


Figura 4.1

Al seleccionar un periodo, se podrá elegir el apartado o formato que se desea revisar (ver Figura 4.2).

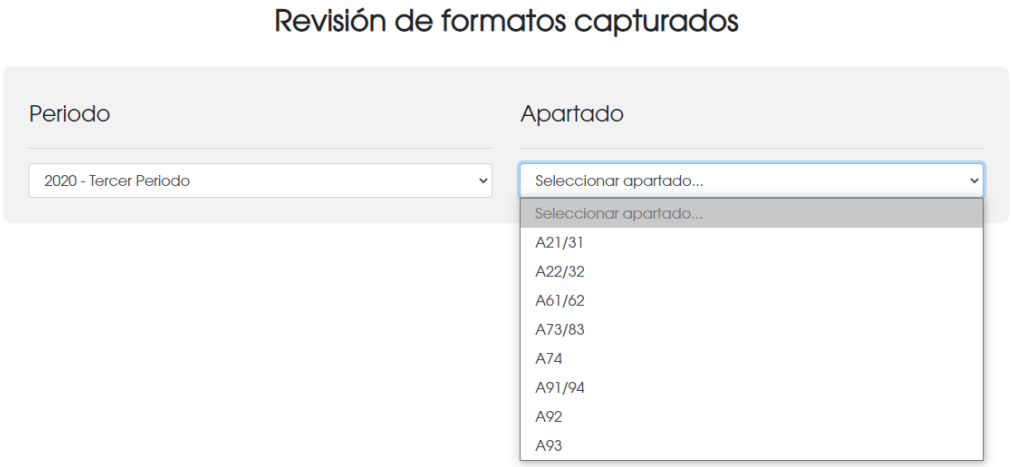


Figura 4.2

Al elegir las opciones anteriores, en la parte inferior se desplegará una ventana con todos los registros de dichas opciones (ver Figura 4.3).

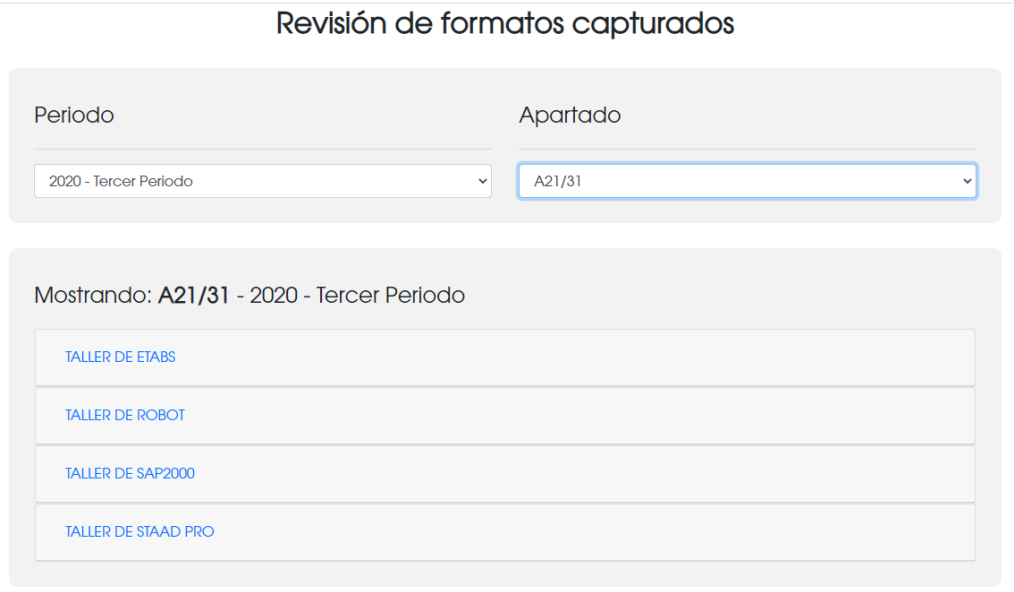


Figura 4.3

Al elegir alguno de los registros de la Figura 4.3 se mostrará su información y, sólo si el registro pertenece al periodo actual, se contará con la opción de *Editar* (ver Figura 4.4).

Figura 4.4

En caso de ser posible la edición de un registro, se dará clic en *Editar*, con lo que abrirá un formulario con los datos correspondientes a dicho registro como en las Figuras 4.5 y 4.6.

Si se efectúan cambios y se desean almacenar o se desea cancelar la edición, en la parte inferior del formulario se encontrarán dos opciones para ello (ver Figura 4.6): *Actualizar* y *Regresar*.

Figura 4.5

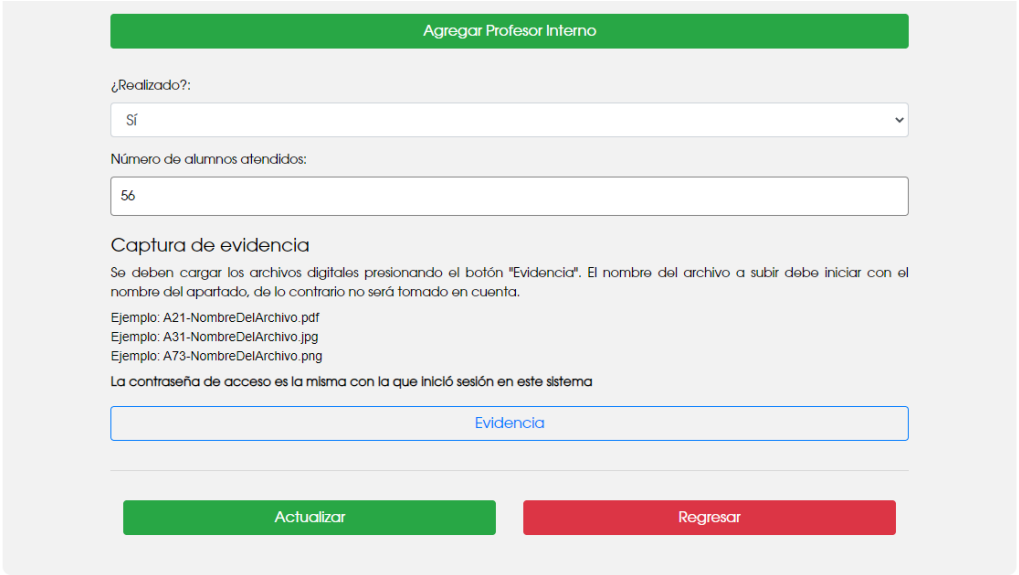


Figura 4.6

Esperados del año

En este módulo se deberá cargar la información de la cantidad de formatos esperados o proyectados para el próximo año al que esté en curso. Esto permitirá generar estadísticas de registros esperados y los obtenidos, para medir el rendimiento de cada Departamento.

Los registros capturados hasta el momento se mostrarán en la parte inferior; en caso de no existir aún registros capturados, se mostrará un mensaje *Sin registros* (ver Figura 5.1).

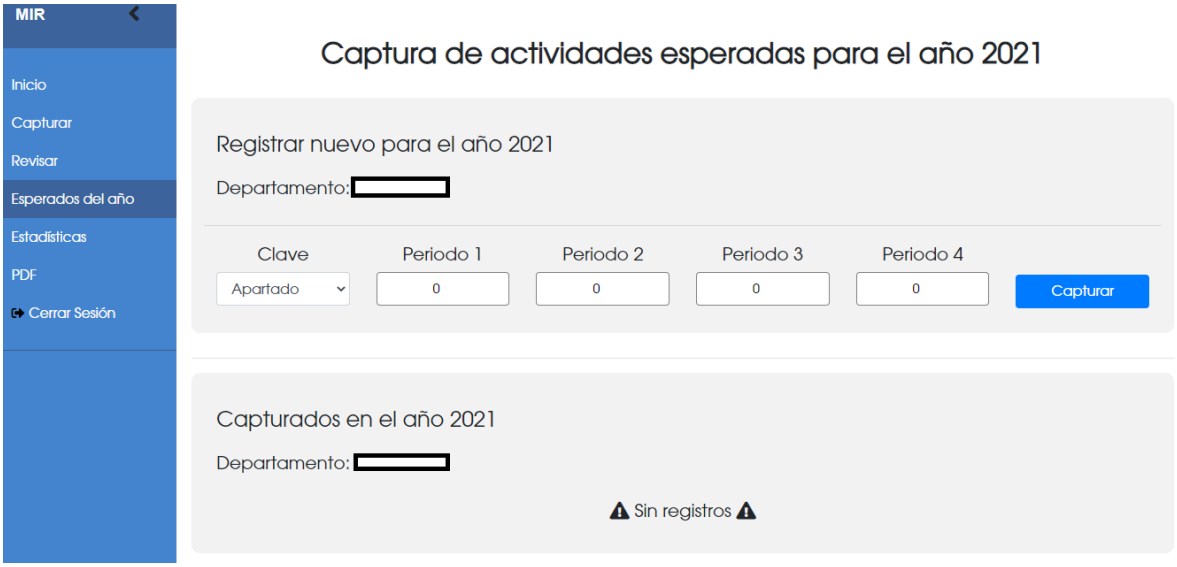


Figura 5.1

La captura de actividades esperadas deberá realizarse indicando la Clave del apartado y la cantidad de registros para cada trimestre/periodo en dicho año (ver Figura 5.2).

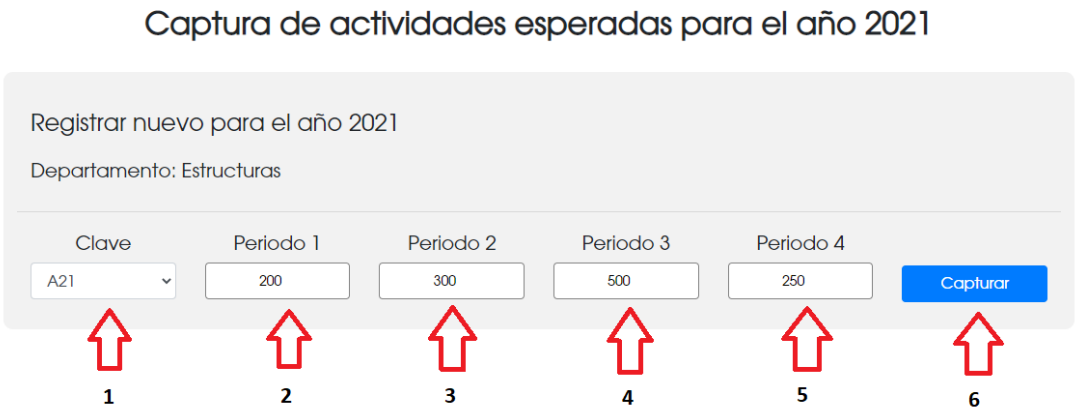

Captura de actividades esperadas para el año 2021
Registrar nuevo para el año 2021
Departamento: Estructuras
Clave
Periodo 1
Periodo 2
Periodo 3
Periodo 4
A21
200
300
500
250
Capturar
1
2
3
4
5
6

Figura 5.2

Estadísticas

En este módulo se presentan la cantidad de registros o actividades capturadas en el sistema y la cantidad de registros que se esperaban para el año en curso.

Esto se realiza a través de dos interfaces, una es mediante un gráfico en la parte superior y con una matriz de números en la parte inferior (ver Figuras 6.1 y 6.2). También es posible seleccionar el año (ver Figura 6.2) para poder visualizar el rendimiento en dicho año.

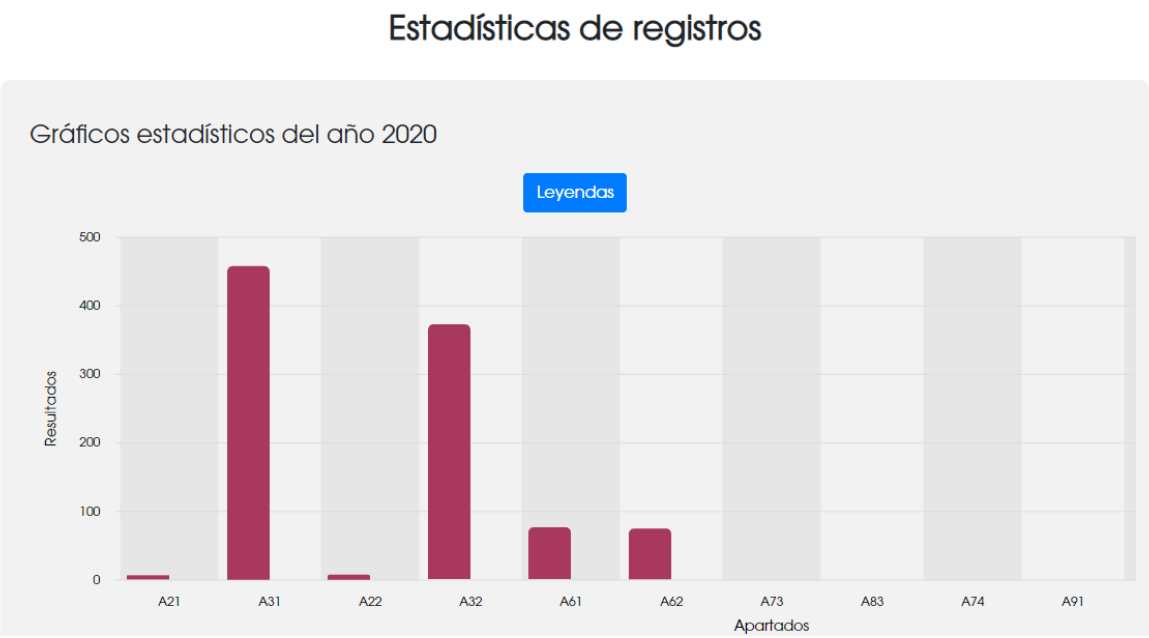


Figura 6.1

Registros del año 2020

Departamento: Estructuras

Año

2020

APARTADO	DATO	PERIODO 1	PERIODO 2	PERIODO 3	PERIODO 4	ANUAL
A21	REAL	3	0	4	0	7
	ESPERADO	0	0	0	0	0
A31	REAL	306	0	152	0	458
	ESPERADO	0	0	0	0	0
A22	REAL	0	5	3	0	8
	ESPERADO	0	0	0	0	0
A32	REAL	0	253	120	0	373
	ESPERADO	0	0	0	0	0

Figura 6.2

PDF

En el módulo de PDF es posible generar los reportes de actividades del periodo que se seleccione.

Figura 7.1

Al elegir un periodo, se mostrará en la parte inferior la información prellenada respecto hacia el autor y remitente del reporte a generar (ver Figura 7.2). Si se desea cambiar la información es posible realizarlo seleccionando dicho texto; esto se reflejará en el PDF que se genere (ver Figura 7.4).

Figura 7.2

En la parte inferior a la Figura 7.2 se desplegarán una serie de botones para generar el reporte PDF para cada uno de los apartados de la MIR, como se visualiza en la Figura 7.3.

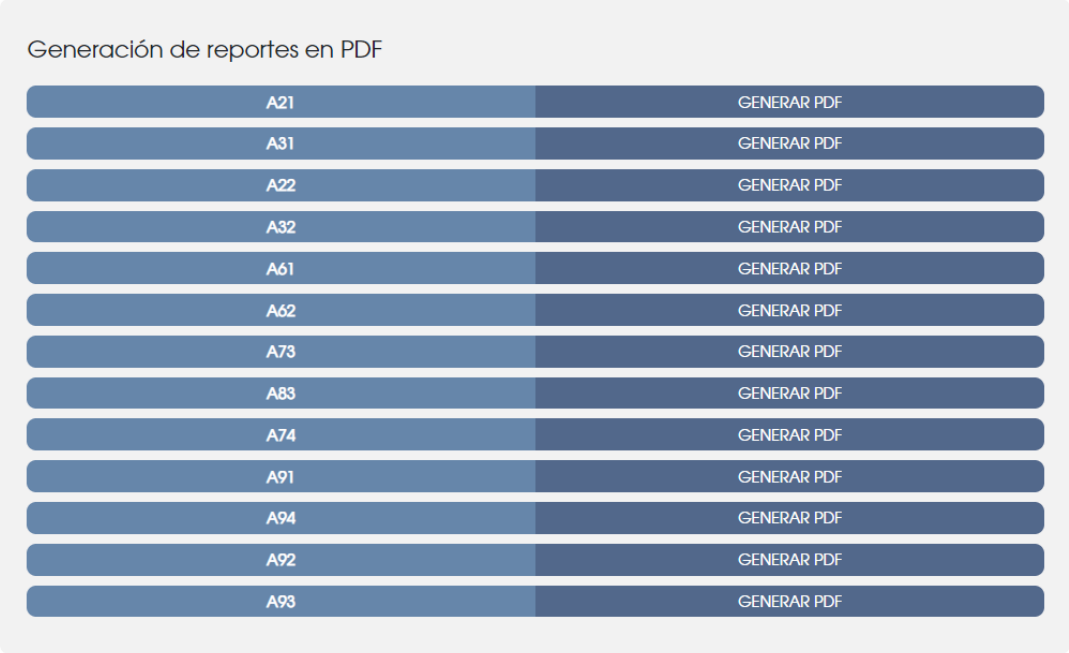


Figura 7.3

Al dar clic en *Generar PDF* del apartado deseado, se abrirá una ventana nueva con la vista previa del PDF y la información correspondiente a dicho periodo, apartado y el remitente (ver Figura 7.4).

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍAS
CIVIL Y GEOMÁTICA

DEPARTAMENTO DE ESTRUCTURAS

Asunto: Tabla resumen Matriz MIR - 2020 - Tercer Periodo

DR. JESÚS HUGO MEZA PUESTO
SECRETARIO ACADÉMICO DE LA DICyG
FACULTAD DE INGENIERÍA, UNAM

P r e s e n t e:

Se anexa la siguiente información del apartado **A.2.1: Cursos y talleres extracurriculares e intersemestrales impartidos** y a la matriz de indicadores MIR implementado por el departamento para el periodo 2020 - Tercer Periodo

#	Actividad	Fecha/hora de inicio	Fecha/hora de finalización	¿Realizado?
1	Taller de Staad Pro	2020-09-07 10:00:00	2020-09-10 15:00:00	Sí
2	Taller de SAP2000	2020-09-14 10:00:00	2020-09-18 12:00:00	Sí
3	Taller de Robot	2020-09-15 10:00:00	2020-09-19 15:00:00	Sí

Figura 7.4

Adicionalmente, es posible generar una firma electrónica que garantice el Visto Bueno y la aprobación del Jefe del Departamento de la información del reporte PDF. Esto se logra al dirigirse en la parte inferior del reporte y dando clic en *Firmar* (ver Figura 7.5).

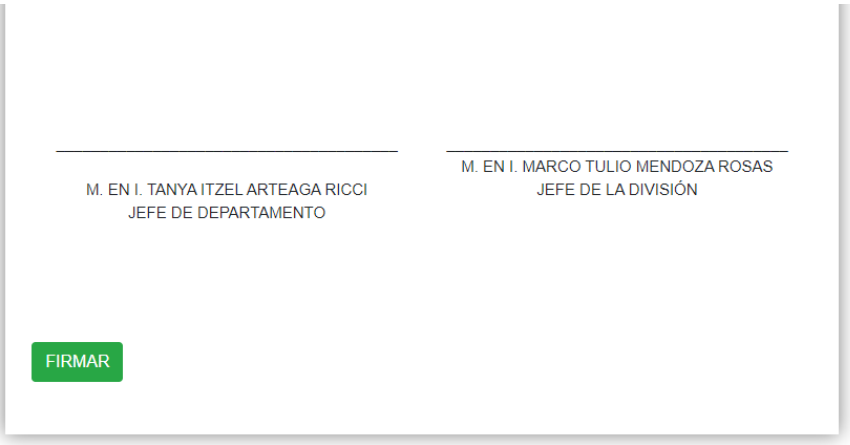


Figura 7.5

Una vez que se haya firmado el reporte, se generará un código QR asociado a su usuario (ver Figura 7.6) y se tendrá la opción de *Imprimir* dicho reporte PDF o guardarlo en su equipo de cómputo (ver Figura 7.7).

Figura 7.6

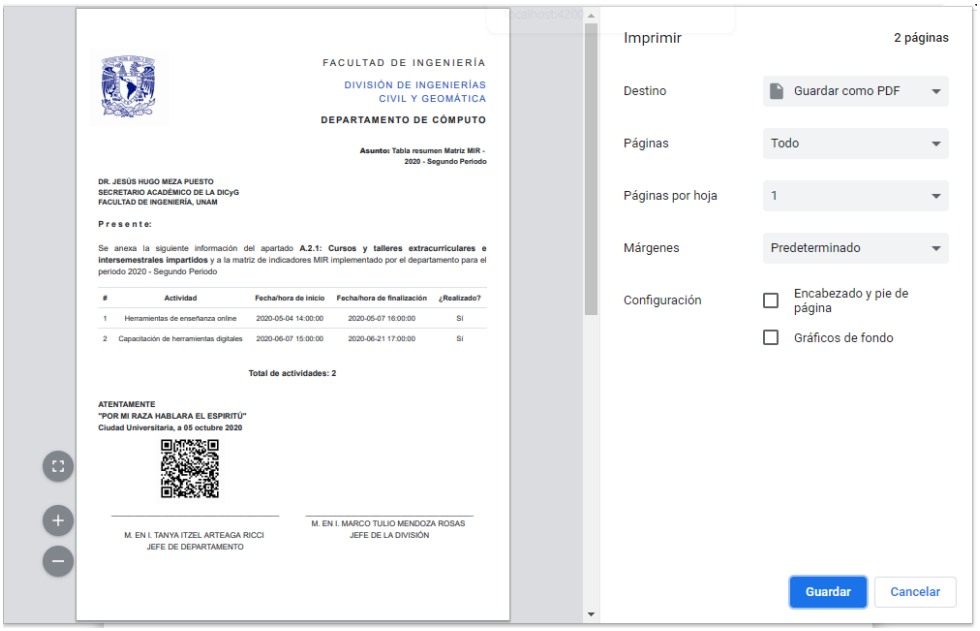


Figura 7.7

Al escanear el código QR, se direccionará a un módulo de validación para mostrar la información de la firma electrónica (ver Figura 7.8).

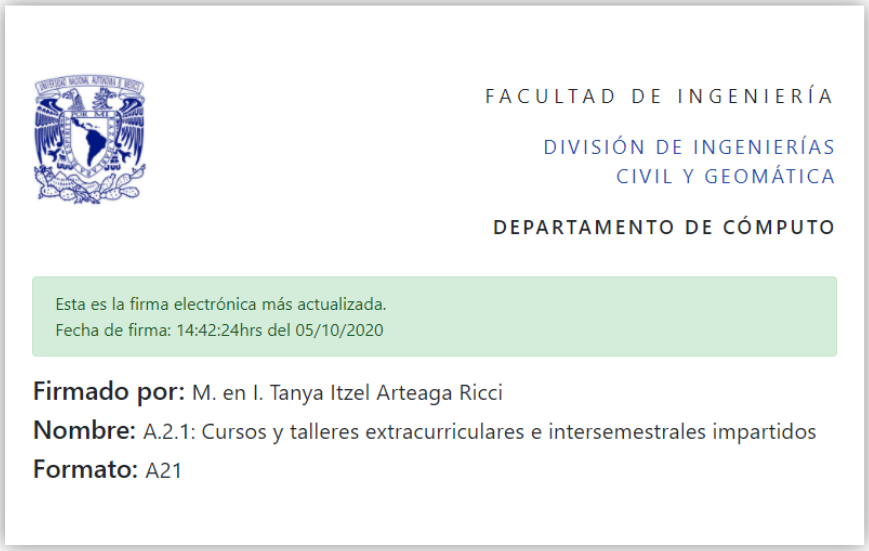


Figura 7.8

Cerrar Sesión

Para cerrar sesión del sistema, basta con dar clic en *Cerrar Sesión* que se encuentra en la barra de navegación del lateral izquierdo (ver Figura 8.1).

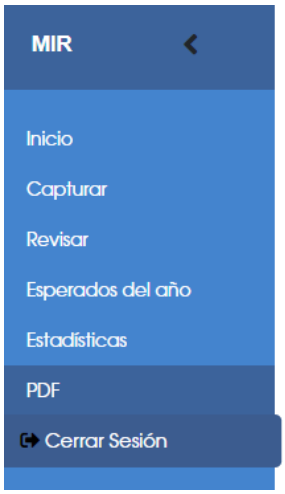


Figura 8.1

Lo cual redireccionará a la vista de inicio de sesión del sistema (ver Figura 8.2).

Figura 8.2