Autores:
AYALA, Cintia Antonela
SANCHEZ, Enzo David

Título del Proyecto Final: Diseño y Desarrollo de un Sistema Integral para una Empresa de actividad petrolera

Grado Académico alcanzado: Ingeniero en Sistemas

Unidad Académica a la cual pertenece: Facultad de Ingeniería de la UNLPam

Fecha de aprobación: 11/09/2020

Nombre y apellido del director: Carolina Salto **Cátedra:** Sistemas Operativos

Nombre, apellido y filiación institucional del Jurado:
María de los Angeles Martin
Maria Belen Rivera
Fernanda Papa

Resumen:

En el presente documento se describe la experiencia en el diseño y desarrollo de un sistema orientado a la web y de una aplicación nativa en Android bajo la metodología ágil denominada Programación Extrema (XP). El proyecto se efectuó con el objetivo de dar solución a la falta de control y seguimiento de los trabajos desarrollados por una empresa de servicios de reparación y mantenimiento de motores en los yacimientos petrolíferos de la localidad de 25 de Mayo.

En este trabajo se hace una revisión bibliográfica de la metodología ágil y se detallan todos los procesos por los que se transitó para obtener el producto final ajustándose a la práctica de XP, teniendo en cuenta los recursos del equipo de desarrollo, los requerimientos solicitados por el cliente y factores externos que influyen directamente en el proyecto.

La aplicación nativa permite recolectar de manera simple los datos de las actividades que realiza un equipo de operadores en una jornada usual de trabajo en la zona petrolera. La aplicación interactúa con el sistema web, el cual gestiona esta información y ofrece la posibilidad de visualizarla y, a su vez, generar informes, índices y estadísticas de interés para el personal administrativo y gerencial de la empresa. Para el desarrollo del sistema web y la aplicación nativa se utilizaron tecnologías innovadoras, con el fin de experimentar con nuevas herramientas y aprovechar sus múltiples beneficios.

Palabras claves: Metodologías agiles, Programación Extrema (XP), 25 de Mayo, yacimiento, petróleo.

Abstract:

This document describes experience in design and development of a web system and a native Android application using the agile methodology called Extreme Programming (XP). The aim's project was to solve the lack of control and monitoring of the work carried out by an engine repair and maintenance service company in the oil area of 25 de Mayo town.

In this work, there is the agile methodology's bibliographic review and all the stages which it passed through to obtain the final product, adjusted to the XP practice, taking into account the resources of the development team, client's requirements and external factors that impacted directly on the project.

The native application provides a simple way to collect data of the activities performed by an operator team in a usual working day in the oil zone. The application interacts with the web system, which manages this information and offers the possibility of viewing briefing papers, generating reports, indexes and statistics of interest to the administrative personnel and company's manager. For the web system and the application developments were used innovative technologies, in order to experiment with new tools and take advantage of their multiple benefits.

Key words: Agile methodologies, Extreme programming (XP), 25 de Mayo, oilfield, oil.

Universidad Nacional de La Pampa
Facultad de Ingeniería

Diseño y Desarrollo de un Sistema Integral para una Empresa de actividad petrolera

Apellido, Nombre: Ayala Cintia
DNI: 34222043
Legajo N°: 3668

Apellido, Nombre: Sanchez Enzo
DNI: 33494414
Legajo N°: 3473

Modalidad: Proyecto de desarrollo
Carrera: Ingeniería en Sistemas
Directora de tesis: Salto, Carolina

Resumen

En el presente documento se describe la experiencia en el diseño y desarrollo de un sistema orientado a la web y de una aplicación nativa en Android bajo la metodología ágil denominada Programación Extrema (XP). El proyecto se efectuó con el objetivo de dar solución a la falta de control y seguimiento de los trabajos desarrollados por una empresa de servicios de reparación y mantenimiento de motores en los yacimientos petrolíferos de la localidad de 25 de Mayo.

En este trabajo se hace una revisión bibliográfica de la metodología ágil y se detallan todos los procesos por los que se transitó para obtener el producto final ajustándose a la práctica de XP, teniendo en cuenta los recursos del equipo de desarrollo, los requerimientos solicitados por el cliente y factores externos que influyen directamente en el proyecto.

La aplicación nativa permite recolectar de manera simple los datos de las actividades que realiza un equipo de operadores en una jornada usual de trabajo en la zona petrolera. La aplicación interactúa con el sistema web, el cual gestiona esta información y ofrece la posibilidad de visualizarla y, a su vez, generar informes, índices y estadísticas de interés para el personal administrativo y gerencial de la empresa. Para el desarrollo del sistema web y la aplicación nativa se utilizaron tecnologías innovadoras, con el fin de experimentar con nuevas herramientas y aprovechar sus múltiples beneficios. El trabajo se realizó bajo la práctica Proyecto de desarrollo, cubriendo un total de 461 horas trabajadas en el mismo.

Agradecimientos

A mis padres, hermanos, compañero e hija por creer en mí, acompañarme y por el esfuerzo que significó patrocinarme en este largo pero gratificante proceso. A mi familia y amigos por el apoyo incondicional.

Cintia

A mi papá y a mi mamá, que me dieron la posibilidad de iniciar mis estudios y estuvieron siempre tanto en los buenos como en los malos momentos, ya sea apoyando, aconsejando o motivando. Sin ellos, nada de esto sería posible. Al resto de mi familia y también amigos, quien cada uno a su manera se hizo presente en esta etapa importante de mi vida.

Enzo

A nuestros compañeros de carrera, que esta etapa universitaria nos hizo coincidir, y a través de días de estudio, mates y charlas, formamos una hermosa amistad haciendo más fácil llegar al objetivo.

A Carolina, a quien admiramos profundamente, por su tiempo, dedicación y por habernos guiado en todo momento.

Y finalmente, a todo el personal, docentes y no docentes y a la Universidad Nacional de la Pampa por abrirnos las puertas, contenernos y formarnos como profesionales. A todos GRACIAS.

Contenido

RESUMEN **4**

AGRADECIMIENTOS **5**

TABLA DE IMÁGENES **9**

TABLA DE TABLAS **12**

CAPITULO 1 ..... **13**

1.1 Introducción 13
1.2 Identificación del problema 15
1.3 Objetivos 16
1.4 Organización del trabajo 17

CAPITULO 2: METODOLOGÍA **19**

2.1 Introducción 19
2.2 Metodología ágil para el desarrollo del software 19
2.3 Principales metodologías ágiles 20
2.4 Programación Extrema 22

CAPITULO 3: PLANEACIÓN **25**

3.1 Historias de usuarios 25
3.2 Iteraciones 27
3.3 Plan de entregas 29
3.4 Reuniones 30
3.5 Mover personal 31
3.6 Modificar XP 31
3.7 División de tareas 32

CAPITULO 4: DISEÑO **35**

4.1 Simplicidad en el diseño 35
4.2 Metáfora del sistema 37
4.3 Tarjetas CRC (Clase-Responsabilidad-Colaboración) 37
4.4 Soluciones *spike* 39
4.5 No aumentar funcionalidades antes de tiempo 40
4.6 Refactorización 40

CAPITULO 5: CODIFICACIÓN **41**

5.1 Cliente siempre presente 41
5.2 Código siguiendo estándares 41
5.3 Codificar la prueba primero 42
5.4 Programación en parejas 42
5.5 Integración secuencial 43
5.6 Integraciones frecuentes 43
5.7 Máquina dedicada a integraciones 43
5.8 Propiedad colectiva 44
5.9 Consideraciones finales 44

CAPITULO 6: PRUEBAS **45**

6.1 Pruebas unitarias 45

6.1.1 Pruebas unitarias en Laravel *46*
6.1.2 Pruebas unitarias en Kotlin *47*
6.2 Pruebas de aceptación 50
6.3 Cuando se encuentra un error 50

CAPÍTULO 7. DESCRIPCIÓN DE PROCESOS PRIORITARIOS 52

7.1 Identificación de los perfiles de usuario 52
7.2 Circuito de carga de una jornada de trabajo 52
7.3 Emitir reportes y estadísticas 59

CAPÍTULO 8. DESARROLLO DE SGPDM Y PDMAPP SEGÚN ITERACIONES DEL PROYECTO 61

8.1 Etapa de codificación y pruebas de la Iteración 1 61
8.1.1 Creación del diagrama de entidad relación (DER) *61*
8.1.2 Presentación de un prototipo de interfaz tanto del sistema SGPDM como de la aplicación PDMApp *63*
8.1.3 Acceso a SGPDM a través de autenticación de usuarios *64*
8.1.4 Implementación de módulos del sistema SGPDM *66*
8.1.5 Testing de los módulos desarrollados *70*
8.2 Etapa de codificación y pruebas de la iteración 2 73
8.2.1 Implementación de la carga de Ordenes de Trabajo (OT) desde la aplicación PDMApp *73*
8.2.2 Listar y visualizar Órdenes de Trabajo (OT) de acuerdo a determinados criterios en SGPDM *76*
8.2.3 Testing del módulo Ordenes de Trabajo *79*
8.3 Etapa de codificación y pruebas de la iteración 3 80
8.3.1 Implementación de módulo de Compresiones en el sistema SGPDM *80*
8.3.2 Implementación de la toma de compresiones cilíndricas de un motor desde la aplicación PDMApp *81*
8.3.3 Generación de reportes en el sistema SGPDM *82*
8.3.4 Leer código QR desde la aplicación PDMApp *85*
8.4 Codificación y pruebas de iteración 4 86
8.4.1 Implementación en el sistema SGPDM del módulo de recesos laborales *86*
8.4.2 Desarrollo en la aplicación PDMApp de la funcionalidad de agregar recesos laborales *87*
8.4.3 Implementación de carga de tareas de almacén en el sistema SGPDM *87*
8.4.4 Desarrollo en la aplicación PDMApp de la funcionalidad de agregar tareas de almacén *88*
8.4.5 Generación de Parte Diarios de Mantenimiento de Motores de Campo (PDMM) y de nuevos reportes *89*
8.4.6 Creación módulos de Ordenes de trabajo y Jornadas *91*

9. CONCLUSIONES 94

10. REFERENCIAS 96

ANEXOS 97

Anexo 1: Glosario 97
Anexo 2: Kotlin 99
2.1 Kotlin frente a Java *99*
Anexo 3: Laravel 101
3.1 Características de Laravel *101*
Anexo 4: Patrón MVC 102
Anexo 5: Plantillas Blade 106
Anexo 6: Historias de usuarios 107
Anexo 7: Tarjetas CRC 113
Anexo 8: Documentos en papel 117
Anexo 9: Estándares 120
9.1 Estándares en el desarrollo del sistema SGPDM *120*

9.2 Estándares en el desarrollo de la aplicación PDMApp 120

Tabla de imágenes

Imagen 1: Mapa del área El Medanito SE 14
Imagen 2: Pozos petroleros en El Medanito SE 14
Imagen 3: Circuito administrativo y operativo en papel 16
Imagen 4: Ciclo de la metodología XP 24
Imagen 5: Ejemplo 1 de historia de usuario 26
Imagen 6: Ejemplo 2 de historia de usuario 27
Imagen 7: Diagrama de Gantt 34
Imagen 8: Diagrama de casos de uso 36
Imagen 9: Diagrama de Entidad Relación (DER) 37
Imagen 10: Tarjeta CRC. 38
Imagen 11: Tarjeta CRC Equipo 38
Imagen 12: Tarjeta CRC Orden 39
Imagen 13: Pruebas instrumentadas (1) y las pruebas de JVM locales (2), en la vista Project. 47
Imagen 14: Configuración de dependencias en archivo gradle. 48
Imagen 15: Estructura del proyecto 49
Imagen 16: Circuito administrativo y operativo actual 53
Imagen 17: Pantalla “Jornada” 53
Imagen 18: Pantalla “Almacén”: Entrega de insumos 54
Imagen 19: Pantalla “Almacén”: Descarga de aceite usado 55
Imagen 20: Etiqueta con código QR 55
Imagen 21: Tipo de órdenes de trabajo 56
Imagen 22: Orden de trabajo (1). 57
Imagen 23: Orden de trabajo (2). 57
Imagen 24: Receso laboral 58
Imagen 25: Edición de receso laboral 58
Imagen 26: Finalizar jornada laboral 59
Imagen 27: Pantalla login 63
Imagen 28: Librería recyclerview en el archivo build.gradle 63
Imagen 29: Pantalla principal de la app 64
Imagen 30: Pantalla inicio para usuario supervisor 66
Imagen 31: Pantalla inicio usuario administrador 66
Imagen 32: Pantalla Maquinas 67
Imagen 33: Método index de MaquinaController 67
Imagen 34: Vista maquinas.index 68
Imagen 35: Método create de MaquinaController 68
Imagen 36: Pantalla de creación de una máquina 68
Imagen 37: Método store de MaquinaController 69
Imagen 38: Método edit de MaquinaController 69
Imagen 39: Pantalla de edición de una máquina 69
Imagen 40: Método update de MaquinaController 70
Imagen 41: Método show de MaquinaController 70
Imagen 42: Pantalla para mostrar una máquina en particular 70
Imagen 43: Test de Login correcto 71

Imagen 44: Test de Login correcto (Resultado) 71
Imagen 45: Test crear_equipo_sin_ser_usuario 72
Imagen 46: Resultado del test crear_equipo_sin_ser_usuario 72
Imagen 47: Test crear_equipo_siendo_usuario 73
Imagen 48: Resultado del test crear_equipo_siendo_usuario 73
Imagen 49: ViewPager con FragmentPagerAdapter de dos páginas: Info. Equipo y Servicios/Materiales 74
Imagen 50: Multichoice AlertDialog con listado de servicios. 75
Imagen 51: AlertDialog con listado de materiales. 75
Imagen 52: Método index de Orden 76
Imagen 53: Scopes en el modelo Orden 77
Imagen 54: Pantalla Ordenes 77
Imagen 55: Pantalla para mostrar una Orden en particular 78
Imagen 56: Pantalla para mostrar una Orden en particular (2) 78
Imagen 57: Test mostrar_listado_ordenes 79
Imagen 58: Resultado test mostrar_listado_ordenes 79
Imagen 59: Test editar_orden_siendo_usuario 79
Imagen 60: Resultado test editar_orden_siendo_usuario 80
Imagen 61: Método index de Compresion 81
Imagen 62: Pantalla index de Compresiones 81
Imagen 63: Compresión Cilíndrica en OT 82
Imagen 64: Información estadística 82
Imagen 65: Gráfico de barras de materiales más utilizados 83
Imagen 66: Gráfico de torta de OT realizadas 83
Imagen 67: Reporte materiales 84
Imagen 68: Ventana modal 84
Imagen 69: Implementando librería ZXing 85
Imagen 70: Método onActivityResult() 85
Imagen 71: Iniciar el proceso de escaneo 86
Imagen 72: Método index de RecesoLaboral 86
Imagen 73: Pantalla index de Recesos Laborales 87
Imagen 74: Método index de TareaAlmacenController 88
Imagen 75: Pantalla index de Tareas de Almacén 88
Imagen 76: TimePickerDialog 89
Imagen 77: Método parteDiarioAdministracion 89
Imagen 78: Búsqueda de Partes Diarios 90
Imagen 79: Parte Diario para Administración 90
Imagen 80: Retornar datos de PDF 91
Imagen 81: Método edit de OrdenController 91
Imagen 82: Pantalla de edición de una Orden 92
Imagen 83: Agregar material 92
Imagen 84: Ventana modal para agregar materiales 93
Imagen 85: AIB – Bomba de Varilla 98
Imagen 86: Ejemplo de MVC 102
Imagen 87: Asignación Segura de Trabajo 117
Imagen 88: Orden de Trabajo 118

Imagen 89: Parte Diario de Mantenimiento de Motores de Campo 119

Tabla de tablas

Tabla 1: Comparación entre metodologías ágiles y tradicionales 20
Tabla 2: Plantilla para las historias de usuario 26
Tabla 3: Historias de usuario por iteraciones 28
Tabla 4: Plan de entregas 29
Tabla 5: Estimación de tiempo de tareas 33

CAPITULO 1

1.1 Introducción

El presente informe tiene como modalidad de trabajo Proyecto de Desarrollo y será llevado a cabo de forma conjunta por dos estudiantes. Se tratará el diseño y desarrollo de un sistema de gestión orientado a la web y una aplicación nativa en Android. En conjunto deben mantener el control y seguimiento de trabajos realizados en un yacimiento petrolífero. También debe generar reportes estadísticos e intercomunicar los entes involucrados. Explicaremos las diferentes etapas por las que transitaron ambas implementaciones y la metodología de desarrollo elegida de acuerdo a las necesidades surgidas. Los requerimientos llegan desde la empresa local YAL S.A., dedicada a reparación, mantenimiento y control de motores (automotores, maquinarias y equipos de uso agropecuario y forestal). Esta empresa está contratada, entre otras, por una compañía multinacional que se encarga de la exploración y producción de petróleo crudo. En la Argentina posee derechos de exploración y producción en cinco yacimientos ubicados en la Cuenca Neuquina[1]:(i) “El Sosneado” mediante una concesión en la Provincia de Mendoza, (ii) “El Medanito“ en la Provincia de La Pampa a través de un contrato de locación de obras y servicios, (iii) “Jaguel de los Machos” mediante una co-concesión y contrato con una empresa provincial, (iv) “Gobernador Ayala V” y (v) “25 de Mayo – Medanito Sudeste” (ver imagen 1), bajo las mismas condiciones.

Actualmente, cuatro equipos de operarios de la empresa YAL S.A. se encuentran en el área productiva La Pampa El Medanito (LPEM) de la localidad 25 de Mayo. Se estima que existen alrededor de 800 pozos perforados en un área de 177 km^2 con más de la mitad activos en producción. En la imagen 2 se los puede apreciar, donde cada marcador representa un pozo y el color los agrupa por zona. YAL S.A. realiza servicios de mantenimiento a más de 450 motores de cigüeñas extractoras de petróleo desde finales del año 2017 [1] [2].

En este informe se explicarán las funcionalidades de la *app* nativa y del sistema de gestión profundizando en los siguientes módulos:

- Identificación de perfiles de usuario
- Circuito de carga de una jornada de trabajo
- Generación de una orden de trabajo
- Emisión de reportes y estadísticas

[1] La Cuenca Neuquina es un área de la vertiente oriental de la cordillera de los Andes, comprendida aproximadamente entre los 32° y 42° de latitud sud, mayoritariamente comprendida en la provincia de Neuquén. Es un área rica en yacimientos de hidrocarburos.

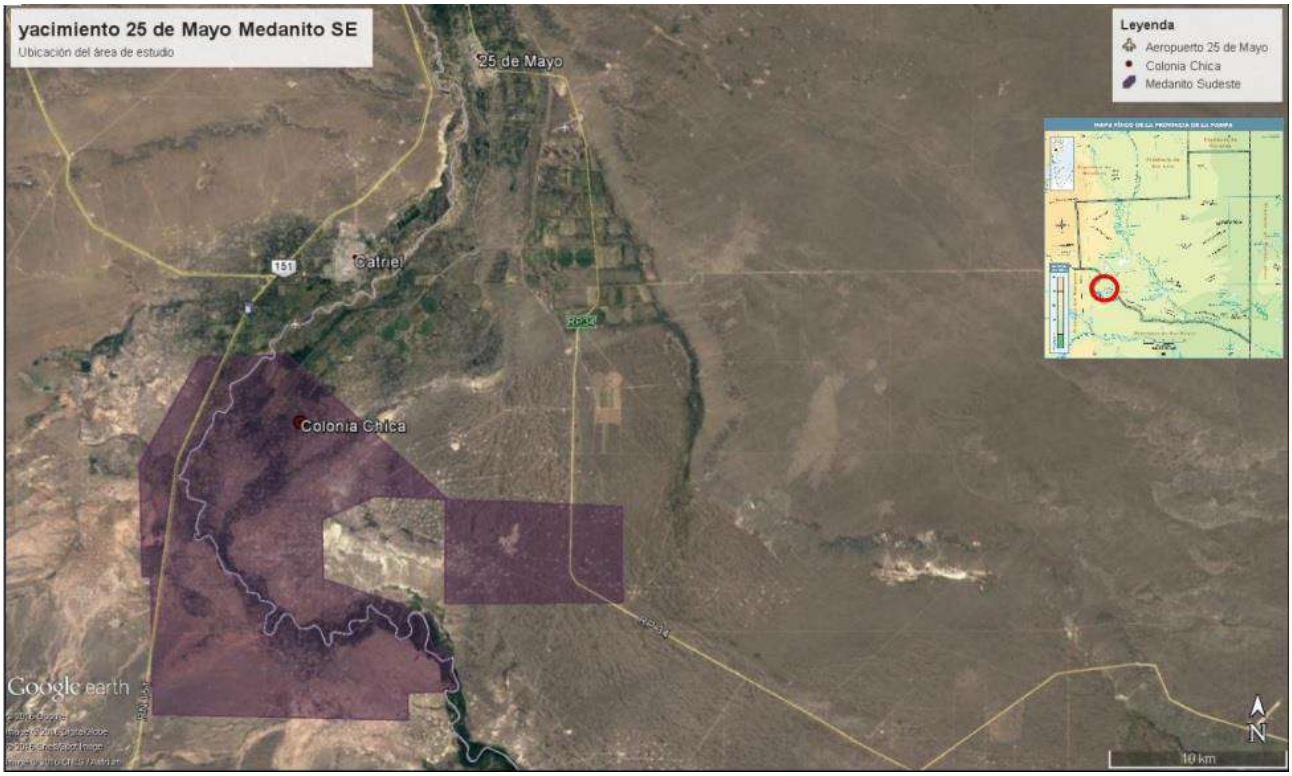


Imagen 1: Mapa del área El Medanito SE

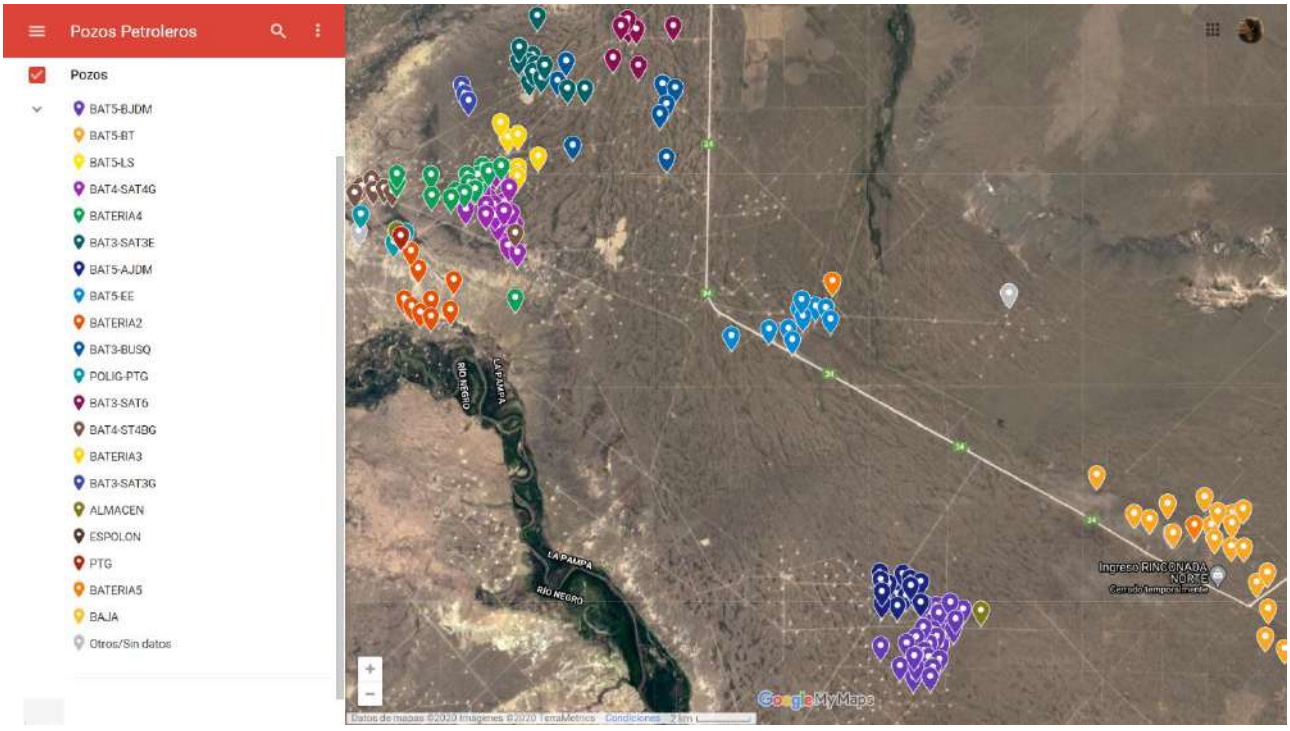


Imagen 2: Pozos petroleros en El Medanito SE

1.2 Identificación del problema

En la actualidad, la empresa YAL S.A. no cuenta con un módulo que registre las distintas actividades que llevan a cabo su grupo de empleados en los pozos petroleros de la localidad de 25 de Mayo. Todas las tareas se registran en planillas de papel, lo que conlleva a un exceso de documentación, inexistencia de control en las distintas áreas de trabajo y dificultad para sostener un historial de servicios prestados.

Los operarios llegan al área de trabajo y completan el documento de Asignación Segura de Trabajo (AST) para garantizar que aceptan y conocen las medidas de seguridad como también los riesgos que se pueden presentar. Una vez hecho esto ingresan al almacén, lugar donde se depositan los insumos que reciben de la empresa contratante, y retiran los suministros necesarios para realizar la labor de la jornada. Las acciones que se realizan dentro de este lugar no quedan asentadas, tarea fundamental para mantener un registro de materiales. Luego, se dirigen al pozo donde se encuentra el motor a ser atendido, efectúan las acciones pertinentes y registran en la planilla denominada Orden de Trabajo (OT) los servicios prestados y la cantidad de material utilizado. En esta instancia, además, se toman diferentes mediciones de presión, pero tampoco se plasman en la planilla por falta de espacio y organización. Este proceso se repite por cada motor que necesite servicio en la jornada laboral. Posteriormente todos los equipos entregan sus planillas al supervisor, quien se encarga de hacer llegar este material al personal administrativo que radica en General Pico. Este circuito operativo y administrativo puede observarse en la imagen 3. La tarea del supervisor es repetitiva y retrasa sus labores de relevancia, puesto que cada noche debe escanear las planillas del día para poder enviarlas por correo electrónico, y que el personal de administración los imprima y cuente con la información de todos los trabajos realizados. A su vez, estos grandes volúmenes de información en papel dificultan el trabajo administrativo debido a que resulta imposible contabilizar cuántos motores están activos a la fecha, en qué pozos se encuentran, el tipo de servicios que han prestado, la mano de obra y materiales que se han suministrado. La falta de informes y estadísticas imposibilita al nivel gerencial visualizar el desempeño real de sus trabajadores, contar con la información organizada para la toma de decisiones y poner en marcha un plan de mejora.

La empresa contratante también recibe copias de los documentos y cargan manualmente la información en su sistema. Este proceso requiere una persona dedicada casi exclusivamente a esta tarea por el tiempo que demanda.

En la zona de El Medanito existe, además, la limitación de la señal de internet, por lo que surge la necesidad de llevar los registros en un dispositivo (*tablet*) de manera *offline*. Debido a esto la aplicación debe ajustarse a las capacidades de hardware y software del dispositivo.

Imagen 3: Circuito administrativo y operativo en papel

1.3 Objetivos

El objetivo general del proyecto YAL es el diseño y desarrollo de un sistema de gestión orientado a la web (SGPDM) y una aplicación nativa para Android (PDMApp). Se tendrán en cuenta las etapas del ciclo de vida del software, integrando nuevas tecnologías que se adecuen a las necesidades de las empresas involucradas. El sistema estará alojado en un servidor de la empresa YAL S.A., con posibilidad de acceso las 24 horas del día.

Mediante el uso del sistema y de la aplicación se espera:

- **Tener registro de datos / eliminar uso de papel:** Todos los datos que se ingresen a la aplicación y al sistema se almacenarán en una misma base de datos. De esta manera, se reduce el volumen físico de documentos archivados. Se tendrá registro de la cantidad de motores que existen, dónde se encuentran y qué tareas se realizaron con ellos sin necesidad revisar las planillas archivadas o de movilizarse a la ubicación actual del motor.
- **Control de tareas realizadas:** Se permitirá un seguimiento de la jornada completa de trabajo, teniendo en cuenta las personas responsables, tiempos (duración de la jornada, del servicio realizado a un motor, entradas y salidas del almacén, etc.) y geolocalización (lugar donde se llevó a cabo la acción).
- **Visión clara / organización de la información:** Se organizará el sitio web y la aplicación en forma modular, ayudando a reducir el desorden de información y evitando contenidos

duplicados. Al mantener los datos en una buena estructura se hace más fácil, en un futuro, la ampliación con contenidos nuevos. A su vez, se permitirá encontrar y agrupar la información haciendo uso de diferentes filtros.

- **Creación de perfiles de usuario:** Para delimitar sobre las funciones que posee cada usuario del sistema, se crearán perfiles. Cada usuario tendrá permisos según el cargo que desempeñe dentro de la empresa.
- **Agilizar tiempo del circuito de la información:** Evitar el escaneo de documentos físicos (papel) entre los operarios y administración, teniendo en cuenta la existencia de una lejanía entre los involucrados. Asimismo, se reduce el tiempo de ingreso de la información tanto en la aplicación como en el sistema.
- **Facilitar la comunicación entre YAL S.A. y la empresa que la contrata:** El personal administrativo de YAL S.A. elige qué información enviar a la multinacional (en formato .csv) y, a su vez, esta última, evita cargar los registros manualmente en su propio sistema.
- **Análisis del trabajo mediante reportes y estadísticas:** El sistema permitirá generar estadísticas de las diferentes labores realizadas. Estos reportes ayudan a analizar el progreso y el trabajo diario de los equipos, brindando información fundamental al sector gerencial para la toma de decisiones.
- **Permitir el registro de las tareas sin acceso a internet:** Dado que los lugares donde trabajan los operarios no cuentan con señal de internet, la aplicación debe permitir el ingreso de los datos de manera *offline*.
- **Escalabilidad:** La aplicación se adaptará de acuerdo a las necesidades de los operarios, permitiendo cambios en el sistema por parte del usuario con permisos para hacerlo (agregar un motor, un pozo, modificar servicios, materiales, etc.) sin necesidad de requerir para esa tarea a los desarrolladores.

1.4 Organización del trabajo

El documento consta de 8 capítulos que se distribuyen de la siguiente forma.

Capítulo 2: Metodología. Ofrece una descripción teórica de las diferentes metodologías ágiles existentes para luego detallar el motivo de la metodología elegida, enumerando principios y características de la misma.

Capítulo 3: Planeación. Se expone la planificación llevada a cabo, teniendo en cuenta el relevamiento de datos, la generación de documentos funcionales, la comunicación con el cliente y la estimación de tiempo. Además, se detalla la experiencia del equipo de desarrollo.

Capítulo 4: Diseño. Se explican los aspectos referidos al diseño de la aplicación y del sistema, exponiendo los diagramas realizados que facilitaron obtener el esquema de persistencia de los datos y la estructura de la interfaz de ambos desarrollos.

Capítulo 5: Codificación. Se exponen los aspectos a considerar para el desarrollo de tareas especificadas en la etapa de planeación y el resultado obtenido por los integrantes.

Capítulo 6: Pruebas. Se detallan los diferentes tipos de pruebas que se deben tener en cuenta para el desarrollo de un software y el diseño que provee el entorno para ejecutarlas.

Capítulo 7: Descripción de procesos prioritarios. Se amplían los conceptos de las funcionalidades más relevantes del circuito operativo y administrativo.

Capítulo 8: Desarrollo de SGPDM y PDMApp según iteraciones del proyecto. Se realiza un recorrido por las iteraciones del proyecto, haciendo hincapié en la implementación, especificando los componentes y las herramientas utilizadas.

Finalmente se exponen las conclusiones a las que se arribaron con el desarrollo del trabajo y apreciaciones personales respecto de la metodología XP.

CAPITULO 2: Metodología

2.1 Introducción

En un proceso de software existen numerosas propuestas metodológicas que inciden en el transcurso del desarrollo. No existe una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable.

Existen aquellas propuestas más tradicionales[2] que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros [3].

Una posible mejora es centrarse en otras dimensiones como por ejemplo, el factor humano o el producto software. Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones cortas.

En este capítulo presentaremos la metodología elegida por el equipo de trabajo para desarrollar ambas implementaciones. Tanto para la creación del sistema de gestión de partes diarios de mantenimiento como para la aplicación móvil, la metodología ágil es la que mejor se adecua al proyecto. A continuación explicaremos en detalle la razón de la selección y características del enfoque.

2.2 Metodología ágil para el desarrollo del software

La metodología ágil es una metodología de gestión de proyectos que da mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque muestra su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad [4].

Esta metodología cuenta como punto de partida con un manifiesto ágil [5], un documento que resume la filosofía ágil y posee doce principios que son características que diferencian un proceso ágil de uno tradicional. Estos son:

i. La prioridad esta puesta en satisfacer al cliente mediante la entrega temprana y continua de software con valor
ii. Aceptar que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente
iii. Entregar software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible
iv. Los responsables de negocio y los desarrolladores deben trabajar juntos de forma cotidiana durante todo el proyecto
v. Los proyectos se desarrollan en torno a individuos motivados. Deben contar con el entorno y el apoyo que necesiten, y brindarles oportunidades para la ejecución del trabajo

[2] La metodología tradicional o pesada hace énfasis en la planificación total del trabajo a realizar y una vez todo detallado, comienza el ciclo de desarrollo de software.

vi. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara
vii. El software funcionando es la medida principal de progreso
viii. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener un ritmo constante de forma indefinida
ix. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad
x. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial
xi. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados
xii. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia

En la tabla 1 se observa la comparación entre las metodologías ágiles y las metodologías tradicionales, mediante esta se afirma que las metodologías ágiles son más orientadas a proceso de desarrollo de software con pocas semanas de desarrollo y bajos niveles de formalización en la documentación requerida.

Tabla 1: Comparación entre metodologías ágiles y tradicionales

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Preparados para el cambio durante el proyecto	Cierta resistencia a los cambios
Reglas de trabajo impuestas internamente (por equipos)	Reglas de trabajo impuestas externamente
Proceso menos controlado, con pocos principios	Procesos mucho más controlados, con numerosas políticas/normas
El cliente es parte del desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones en determinadas etapas del proceso
Pocos roles	Más roles

2.3 Principales metodologías ágiles

Al comienzo de los 90 surgieron propuestas metodológicas para lograr resultados más rápidos en el desarrollo de software sin disminuir su calidad.

Entre las principales metodologías ágiles, se encuentran las siguientes:

- **SCRUM:** especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones denominadas Sprint, con una duración de 30 días, el resultado de cada Sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de quince minutos del equipo de desarrollo para coordinación e integración [6].
- **Metodologías Crystal:** se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por encontrarse centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se debe invertir esfuerzos en

mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores [7].

- **Desarrollo de software adaptativo (*Adaptative Software Development - ASD*):** presupone que las necesidades del cliente son cambiantes. La iniciación de un proyecto involucra definir una misión para él, determinar las características, las fechas y descomponer el proyecto en una seria de pasos individuales, cada uno de los cuales puede abarcar entre cuatro y ocho semanas. Los pasos iniciales deben verificar el alcance del proyecto, los tardíos tienen que ver con el diseño de la arquitectura, la construcción del código, la ejecución de las pruebas finales y el despliegue [8].
- **Proceso Unificado de Desarrollo de Software:** el proceso unificado es un conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema de software. Este proceso se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo consta de cuatro fases: Inicio, Elaboración, Construcción y Transición [9].
- **Método de Desarrollo de Sistemas Dinámicos (*Dynamic Systems Development Methods - DSDM*):** metodología de dominio público, independiente de las herramientas, utilizadas en proyectos de tipo RAD (*Rapid Application Development*). En DSDM se definen cinco fases: estudio de factibilidad, estudio de negocio, iteración del modelo funcional, iteración del diseño y construcción/implantación [10].
- **Kanban:** la estrategia Kanban es especialmente útil para los responsables de proyectos. Consiste en la elaboración de un cuadro o diagrama en el que se reflejan tres columnas de tareas: pendientes, en proceso y terminadas. Es indispensable que el cuadro esté ubicado en un lugar visible, o en una herramienta de software compartida, para que los miembros de los equipos sepan la evolución del proceso y eviten repetir tareas. De esta manera, se logra una mejor coordinación de tiempos, talentos y habilidades.
- **Programación Extrema (*eXtreme Programming*):** pone el acento en las relaciones personales de los miembros del equipo y entre éstos y los clientes o destinatarios del proyecto. Es especialmente oportuna para *startups* o empresas que aún no están consolidadas en sus respectivos mercados. Además, dado que el foco son las relaciones entre los miembros, lo ideal es que se acoja en escenarios con equipos de trabajo reducidos. Sus fases principales son: El cliente decide lo que quiere del proceso, objetivos y resultados. El equipo divide el trabajo en acciones pequeñas y le asigna un tiempo a cada una, el cliente decide qué acciones se realizan primero, y el equipo de trabajo realiza lo que el cliente ha decidido [11].

En particular, para el desarrollo del proyecto YAL, en las primeras reuniones con el cliente se manifestaron los requerimientos iniciales dejándose entrever que habría cambios a los que el sistema debería adaptarse luego. La comunicación con el cliente fue frecuente, permitiendo evacuar dudas y aportar sugerencias en cada iteración, tanto por su parte como de la nuestra. Por lo tanto, dentro de las modalidades disponibles la que más se ajustó al contexto del proyecto es Programación Extrema.

2.4 Programación Extrema

La programación extrema (XP) es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck[3]. Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, XP se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

Los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Ser capaz de adaptarse a los cambios de requisitos, en cualquier punto de la vida del proyecto, es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos. La programación extrema es la metodología de desarrollo que mejor se adecúa a lo que se pretende llevar a cabo en este proyecto, y será aplicada de manera dinámica durante el ciclo de vida del software.

Los equipos de un proyecto de esta tipología y magnitud tienen normalmente las siguientes figuras y roles [12]:

- **Clientes:** escribe las historias de usuario y las pruebas funcionales para validar su implementación. Asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración, centrándose en aportar el mayor valor de negocio. Suelen ser los usuarios finales del producto.
- **Programadores:** escriben las pruebas unitarias y producen el código del sistema. Definen las tareas que conlleva cada historia de usuario y estiman el tiempo que requerirá cada una.
- **Encargado de Prueba (*Tester*):** ayuda al cliente a escribir las pruebas funcionales. Ejecuta pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- **Encargado de seguimiento (*Tracker*):** proporciona retroalimentación al equipo. Debe verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones.
- **Entrenador (*Coach*):** es responsable del proceso global. Sirve de guía a los miembros del equipo para que se apliquen las prácticas XP y se siga el proceso correctamente. Determina la tecnología y metodologías a usar por el equipo de desarrollo.
- **Gestor (*Big boss*):** vínculo entre clientes y programadores. Construye el plantel del equipo, obtiene los recursos necesarios y maneja los problemas que se generan. Administra a su vez las reuniones (planes de iteración, agenda de compromisos, etc.).

Roles opcionales:

- **Consultor:** es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Guía al equipo para resolver un problema en concreto.
- **Predictor *(Doomsayer)*:** es el responsable de asegurarse que se conocen los riesgos involucrados, que las malas noticias se den a conocer en la medida correcta, y que no se sobredimensionen. Debe buscar posibles riesgos en forma constante, presentándolos al equipo siempre con una alternativa para tratar el problema.

XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. A continuación detallaremos los 4 valores de la programación extrema:

[3] Kent Beck (1961-), autor del primer libro sobre la metodología programación extrema, Extreme Programming Explained: Embrace Change (1999- ISBN 978-0321278654, ISBN 84-7829-055-9)

- **Comunicación:** la comunicación se realiza de diferentes formas y el objetivo es dar a todos los intervinientes una visión compartida del sistema. Para el grupo de programadores es recomendable que cada integrante conozca las tareas del resto para así, si alguno tiene un problema que otro ha solucionado con anterioridad puedan resolverlo juntos (no reinventar la rueda). La comunicación es importante para crear un sentido de equipo y cooperación eficiente. El código comunica mejor cuanto más simple sea, si es complejo hay que esforzarse para hacerlo inteligible. El código autodocumentado es más fiable que los comentarios ya que éstos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo el objetivo de una clase o la funcionalidad de un método. Las pruebas unitarias son otra forma de comunicación, ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad. La colaboración entre usuarios y programadores es constante permitiendo satisfacer, en caso de surgir, nuevos requisitos y responder rápidamente a los mismos. Muchos problemas que surgen en los proyectos se deben a que después de concretar los requerimientos que debe cumplir el programa no hay una revisión de los mismos, pudiendo dejar olvidados puntos importantes.
- **Simplicidad:** se desarrollan las soluciones más sencillas necesarias para los requerimientos que se están viendo en ese momento mitigando las fallas a medida que ocurren. No se invierte esfuerzo en futuros requerimientos que podrían cambiar o que no se vayan a necesitar. Así mismo un diseño y programación simple mejora la calidad de la comunicación y comprensión del código por todos los integrantes del equipo facilitando el mantenimiento.
- **Realimentación (*Feedback*):** al estar el cliente integrado en el proyecto su opinión sobre el mismo se conoce en tiempo real. Las pruebas de aceptación se realizan tras ciclos cortos de tiempo y minimiza el tener que rehacer partes que no cumplen con los requisitos y centrarse en lo importante. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Ejecutar periódicamente pruebas unitarias e integradas permite descubrir fallos debidos a cambios recientes en el código.
- **Coraje:** se debe tener coraje o valentía para cumplir los tres puntos anteriores. Hay que tener valor para comunicarse con el cliente y enfatizar algunos puntos, a pesar de que esto pueda dar sensación de ignorancia por parte del programador. Hay que tener coraje para mantener un diseño simple y no optar por el camino más fácil. Esto es un esfuerzo para evitar empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar todo lo demás del proyecto. La valentía permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario. Por último hay que tener valor y confiar en que la realimentación sea efectiva.

Los pasos fundamentales inmersos en las fases del método son:

- ❖ Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- ❖ Pruebas unitarias continuas: son frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- ❖ Programación en parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
- ❖ Frecuente integración del equipo de programación con el cliente o usuario: se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.

- ❖ Corrección de todos los errores antes de añadir nueva funcionalidad: Hacer entregas frecuentes.
- ❖ Refactorización del código: es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- ❖ Propiedad del código compartido: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- ❖ Simplicidad del código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

La simplicidad y la comunicación son extraordinariamente complementarias. Con mayor comunicación resulta más fácil identificar qué se debe y qué no se debe hacer. Cuanto más simple es el sistema, menos se tendrá que comunicar sobre éste, lo que lleva a una comunicación más completa, especialmente si se puede reducir el equipo de programadores.

El método XP propone potenciar las actividades que se desempeñan en las fases que atraviesa un ciclo XP, definiendo los requisitos, arquitectura y diseño cada día, y no en un periodo de tiempo determinado y acotado. XP apuesta por hacer simultáneas las fases y llevarlas a cabo en paralelo para que, de este modo se vaya adaptando el producto y el sistema a las necesidades según van surgiendo. En un periodo corto, puede ser una semana, se realiza un ciclo completo o iteración (ver imagen 4) en la que se aborda algo de cada una de las fases tradicionales de un desarrollo software, trabajando con las historias de usuario, que son requisitos que dan valor al cliente. En cada ciclo se planifica en qué historias trabajar y se llevan a cabo de forma completa con su análisis, diseño, pruebas y desarrollo. Al final de la iteración se podrá mostrar al cliente el resultado de la funcionalidad obtenida para que pueda formarse una opinión sobre ella.

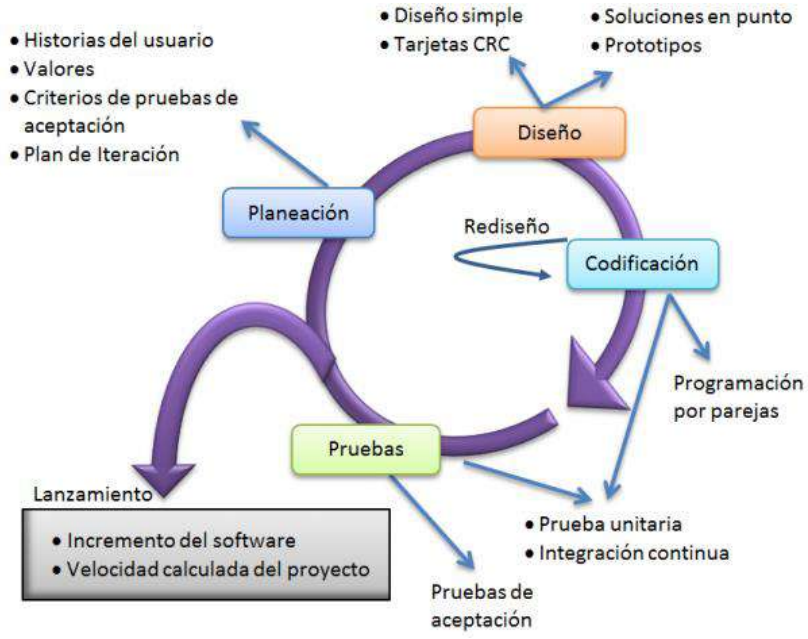


Imagen 4: Ciclo de la metodología XP

CAPITULO 3: Planeación

En la fase inicial de la metodología XP se establece una comunicación continua entre el equipo de desarrollo y el cliente, para obtener principalmente los requisitos del sistema. Además permite establecer el alcance del proyecto y fechas de entrega, tomando en cuenta la prioridad y tiempo estimado para el desarrollo de cada historia de usuario. En muchos casos, los proyectos de software no salen como debieran porque se han hecho malas previsiones o malas ejecuciones del desarrollo, o simplemente porque se han definido sus funcionalidades de forma inapropiada. Una adecuada planificación resulta imprescindible para el proyecto.

Esta fase puede consistir en elaborar un plan de desarrollo del software, estableciendo las estimaciones de recursos, costos y tiempo del proyecto. También puede implicar el desarrollo de un plan de etapas de entregas, identificando el número y tamaño de las iteraciones, requerimientos y documentos de diseño detallado, al igual que se plantean ajustes necesarios a la metodología según las características del proyecto. En definitiva, la planificación consiste en organizar y documentar aquello que queremos hacer con el objetivo de entregar valor incrementalmente. En este caso se consigue agregando funcionalidad en cada iteración y manteniendo siempre el producto en marcha con la funcionalidad que haya sido implementada hasta ese momento.

Por lo tanto, en este capítulo se desarrollarán los siguientes elementos de la metodología: historias de usuario, plan de entrega, iteraciones, reuniones, mover personal y ajustar XP. En cada caso, se planteará la definición y posteriormente se describirá cómo se implementaron en el desarrollo del proyecto. O, en caso de no utilizarse, se explicará el motivo por el cual no se aplicó.

3.1 Historias de usuarios

El primer paso en la estimación y planificación es la definición del proyecto a realizar. Se puede definir en objetivos expresados como historias de usuarios, cada una aportando valor de negocio, incremental e individual. Una historia es un requisito de negocio visto desde el punto de vista de un usuario. Constan de tres o cuatro líneas escritas por el cliente en un lenguaje no técnico sin hacer mucho hincapié en los detalles; no se debe hablar ni de posibles algoritmos para su implementación, ni de diseños de base de datos adecuados, etc. Son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia. El tiempo de desarrollo ideal para una historia de usuario es entre una y tres semanas.

Las condiciones de satisfacción de los objetivos suelen ponerse en forma de criterios de aceptación, pruebas que se realizarán para verificar si el sistema se comporta de la manera esperada. Esta técnica permite evitar la aparición de errores por malos entendidos y evadir repetición de tareas. Por ello es recomendable no empezar a desarrollar una iteración sin antes haber escrito los casos de prueba, especialmente porque es más barato escribir texto y pensar en cómo desambiguar los requisitos que arreglar errores importantes debido a su mal entendimiento [11].

En la definición de las historias de usuario para el desarrollo del proyecto YAL, si bien las historias no fueron escritas por el cliente, fue quien diseñó y dirigió su contenido, ya sea en las reuniones o a través de comunicación vía correo electrónico. A pesar de que no hayan sido elaboradas por el cliente por falta de conocimiento del formato, el propósito de las historias de usuario no se vio

alterado. En aquellos casos que la terminología no resultaba completamente entendida, se hablaba nuevamente con el cliente para terminar de comprender lo que se solicitaba.

Además, las historias de usuario jugaron un papel importante en la estimación de los tiempos requeridos para el desarrollo de este proyecto, pudiendo plantear tiempos necesarios para su implementación. Finalmente, resultó que las estimaciones fueron aproximadas a los tiempos requeridos para el desarrollo.

En lo que respecta a los criterios de aceptación, el cliente por cuestiones de disponibilidad de tiempo, no pudo diseñarlos. No obstante, como el sistema y la aplicación eran bien conocidos por los miembros del equipo de desarrollo y se disponía de la información para la reconstrucción de estas pruebas, no fue necesaria la intervención del cliente.

Finalmente, se obtuvo un total de 26 historias de usuario, considerando que el número es adecuado con respecto a la magnitud del proyecto. La plantilla a utilizarse durante el proyecto para las historias de usuario se muestra en la tabla 2, con sus componentes debidamente explicados.

Tabla 2: Plantilla para las historias de usuario

Historia de Usuario	
Número: Permite identificar a una historia de usuario	
Usuario: Persona que utilizará la funcionalidad del sistema descripta en la historia de usuario	
Nombre Historia: Describe de manera general a una historia de usuario	
Prioridad en Negocio: Grado de importancia que el cliente asigna a una historia de usuario	**Riesgo en Desarrollo:** Valor de complejidad que una historia de usuario representa al equipo de desarrollo
Tiempo estimado: Horas que se necesitara para el desarrollo de una historia de usuario	**Iteración asignada:** Numero de iteración, en que el cliente desea que se implemente una historia de usuario
Programador responsable: Persona encargada de programar cada historia de usuario	
Descripción: Información detallada de una historia de usuario	

A continuación, a modo de ejemplo, se presentan las imágenes 5 y 6 que son dos historias de usuario que se escribieron en el proyecto. El resto de ellas, se detallan en el Anexo 6.

Historia de usuario	
Número: 5	
Usuario: SOP	
Nombre historia: Alta/Baja de equipos de trabajo	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Generar un nuevo grupo de trabajo y discontinuarlo en caso de que ya no exista esa formación de operarios.	

Imagen 5: Ejemplo 1 de historia de usuario

Historia de usuario	
Número: 20	
Usuario: Operario	
Nombre historia: Escaneo de código QR	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 16 horas	**Iteración asignada:** 3
Programador responsable: Ayala, Cintia	
Descripción: Poder escanear desde la aplicación un código QR que proveerá información del motor.	

Imagen 6: Ejemplo 2 de historia de usuario

3.2 Iteraciones

En la metodología XP, el desarrollo del software se divide en etapas para facilitar su realización. Por lo general, los proyectos constan de más de tres etapas, las cuales toman el nombre de iteraciones. De allí se obtiene el concepto de metodología iterativa. La duración ideal de una iteración es de una a tres semanas.

Por cada iteración se define un módulo o conjunto de historias que se van a implementar. Al final de la iteración se obtiene como resultado la entrega del módulo correspondiente, el cual debe haber superado las pruebas de aceptación que establece el cliente para verificar el cumplimiento de los requisitos. Las tareas que no se realicen en una iteración se tomarán en cuenta en la siguiente iteración, en donde se define, junto al cliente, si se deben realizar o deben ser removidas de la planeación [11].

Para el desarrollo del proyecto solicitado por la empresa YAL S.A., se dividió el mismo en 4 iteraciones, en las cuales se entregaron partes del sistema y de la aplicación, completamente funcionales. Las distintas iteraciones fueron las siguientes:

Iteración 1:

- Creación del diagrama de entidad relación (DER) que permita visualizar un modelo de base de datos
- Presentación de un prototipo de la interfaz tanto del sistema SGPDM como de la aplicación PDMApp
- Acceso al sistema SGPDM a través de autenticación de usuario
- Implementación de los distintos módulos del sistema SGPDM, con sus funcionalidades básicas de “Crear”, “Editar” y “Eliminar”, para que el cliente pueda cargar los datos pertinentes a usar. En esta iteración, los módulos a desarrollar son: Aplicaciones, Bombas/AIB, Equipos, Máquinas, Materiales, Móviles, Operarios, Pozos, Servicios de Mantenimiento y Zonas

Iteración 2:

- Implementación de la carga de Órdenes de Trabajo (OT) desde la aplicación PDMApp
- Creación de las funcionalidades para listar y visualizar OT, de acuerdo a determinados criterios simples de búsqueda (ejemplo: por fecha o equipo) en el sistema SGPDM

Iteración 3:

- Implementación de módulo de Compresiones (con las funciones de “Crear”, “Editar” y “Eliminar”) en el sistema SGPDM

- Implementación de la toma de compresiones cilíndricas de un motor desde la aplicación PDMApp
- Generación de reportes de datos de: Materiales utilizados, Motores atendidos, Horas trabajadas en el sistema SGPDM
- Leer códigos QR desde la aplicación PDMApp

Iteración 4:

- Implementación en el sistema SGPDM del módulo para la carga de recesos laborales
- Desarrollo en la aplicación PDMApp de la funcionalidad de agregar recesos laborales a una jornada de trabajo
- Implementación para la carga de las tareas realizadas en el almacén en el sistema SGPDM
- Desarrollo en la aplicación PDMApp de la funcionalidad de agregar tareas realizadas en el almacén
- Generación de Partes Diarios de Mantenimiento de Motores de Campo (PDMM) y de nuevos reportes, de acuerdo a las últimas peticiones
- Creación de módulos de Órdenes de Trabajo y Jornadas

En la tabla 3 se detallan las historias de usuario que pertenecen a cada iteración.

Tabla 3: Historias de usuario por iteraciones

Iteración	Historia de usuario
1	1. Acceso a SGPDM
	2. ABM de aplicaciones
	3. ABM de bombas AIB
	4. ABM de operarios
	5. Alta/Baja de equipos de trabajo
	6. ABM de móviles
	7. ABM de zonas
	8. ABM de pozos
	9. ABM de servicios de mantenimiento
	10. ABM de materiales
	11. ABM de máquinas
2	12. Cargar una orden de trabajo
	13. Listado de órdenes de trabajo según distintos criterios
3	14. ABM de compresiones cilíndricas
	15. Toma de compresiones cilíndricas de un motor desde la app
	16. Reporte de materiales
	17. Reporte de servicio de motores
	18. Reporte de horas trabajadas
	19. Imágenes de orden de trabajo
	20. Escaneo de código QR
4	21. Recesos laborales en la app
	22. ABM de recesos laborales
	23. Tareas de almacén en la app
	24. ABM de tareas de almacén
	25. Parte Diario de Mantenimiento
	26. Listar/Modificar/Eliminar jornada

3.3 Plan de entregas

Habiendo definido las historias de usuarios es necesario crear un plan de entregas, donde se indiquen cuáles se crearán para cada versión del sistema y las fechas en las que las mismas se entregarán.

En un plan de entregas, los desarrolladores y el cliente establecen los tiempos de implementación ideales de las historias de usuario, la prioridad con las que serán implementadas y cuáles serán desarrolladas en cada iteración del proyecto. Al finalizar esta etapa tienen que estar claros estos cuatro factores: los objetivos que se deben cumplir (que son principalmente las historias que se deben desarrollar en cada iteración), el tiempo que tardarán en desarrollarse y entregarse las distintas versiones del sistema, el número de personas que trabajarán en el desarrollo y cómo se evaluará la calidad del trabajo realizado [11].

Volviendo al proyecto, se elaboró el siguiente plan (tabla 4), el cual muestra las historias de usuario que se llevaron a cabo en cada iteración, teniendo en cuenta la prioridad de cada una. Para aproximar el tiempo, se tomó como medida tres semanas. Según la cantidad de historias con las que constaba la iteración, en algunos casos se pudo reducir ese tiempo, mientras que en la iteración 3 y 4 se extendió el plazo debido a retrasos en el desarrollo y/o indisponibilidad del cliente para verificar si dicha iteración cumplía los criterios de aceptación. Cada semana de trabajo constaba de 5 días en las que se trabajaban 4 horas sin distracciones. Esta decisión fue acogida por el equipo debido a otras responsabilidades externas al proyecto.

Tabla 4: Plan de entregas

Historias de Usuario	Iteración	Prioridad
Historia 1	1	Media
Historia 2	1	Alta
Historia 3	1	Alta
Historia 4	1	Alta
Historia 5	1	Alta
Historia 6	1	Alta
Historia 7	1	Alta
Historia 8	1	Alta
Historia 9	1	Alta
Historia 10	1	Alta
Historia 11	1	Alta
Historia 12	2	Alta
Historia 13	2	Media
Historia 14	3	Media
Historia 15	3	Media
Historia 16	3	Alta
Historia 17	3	Alta
Historia 18	3	Media
Historia 19	3	Media
Historia 20	3	Media
Historia 21	4	Media
Historia 22	4	Media
Historia 23	4	Media
Historia 24	4	Media
Historia 25	4	Alta
Historia 26	4	Alta

3.4 Reuniones

El planeamiento es esencial para cualquier tipo de metodología, es por ello que XP requiere de una revisión continua del plan de trabajo. Al comenzar el proyecto se realiza una reunión entre el equipo de trabajo y el cliente. En dicha reunión se define el marco temporal de la realización del sistema. El cliente expone las historias de usuario a los integrantes del grupo, quienes estimarán el grado de dificultad de la implementación de cada historia. Las historias de usuario son asignadas a las diferentes iteraciones según su orden de relevancia para el proyecto. En esta reunión se predicen los tiempos que se utilizarán en la realización de las diferentes etapas del proyecto, los cuales no son datos exactos pero proporcionan una base del cronograma.

También se llevan a cabo reuniones al inicio de cada iteración, donde se organizan las actividades de programación a realizar, traduciendo las historias de usuario a tareas y asignándolas a los desarrolladores. Estos estiman los tiempos de realización de cada tarea. XP propone que el tiempo ideal o sin distracción es de uno a tres días. A su vez, es necesario que los desarrolladores se reúnan diariamente (*stand-up meeting*) y expongan sus problemas, soluciones e ideas en forma conjunta con el fin de que el equipo construya una mejor solución. Las reuniones deben ser fluidas, todos deben poder hablar y se debe evitar discusiones largas y conversaciones separadas [11].

En lo que respecta a nuestro proyecto, se realizaron tres reuniones iniciales. En la primera, el cliente se presentó con una idea poco clara de lo que necesitaban. Solamente contaba con una planilla en papel, la cual debía ser cargada de manera digital a través de la *tablet*. Todos los participantes del proyecto debieron indagar exhaustivamente para obtener mayor precisión de los objetivos. Para la segunda reunión, se contactó a un representante del cliente. El equipo de trabajo expuso las historias de usuario creadas (como se explicó anteriormente, el cliente no disponía de tiempo para hacerlas) y estos aceptaron lo propuesto por el equipo y complementaron añadiendo nuevos requerimientos. En la última reunión de esta etapa inicial, nuevamente el equipo presentó todas las historias desarrolladas después de realizadas las reuniones anteriores. El cliente confirmó que se cubrían todos los requerimientos, y además, se profundizó en las definiciones de las historias.

La realización de las reuniones retrasó varios días el desarrollo del proyecto, principalmente debido a que en la primera reunión no se pudo obtener demasiada información. No todos los representantes del cliente tenían en claro los procesos que se incluirían en el proyecto e incluso, en la segunda reunión, debatían entre ellos con respecto a los requerimientos a solicitar. Esto dejaba en claro que no estaban organizados y perjudicaban los tiempos destinados a las reuniones. Como solución, el equipo solicitó que solo una persona presenciara las reuniones y expusiera los requerimientos para no retrasar el proyecto.

Las reuniones diarias se realizaban en los hogares de alguno de los miembros del equipo de trabajo, siendo esto posible debido a la cercanía entre los participantes del proyecto. Se intercambiaban e-mails para aquellos días en que no era posible reunirse. La comunicación de problemas y soluciones se realizaban a lo largo de la jornada ya que se contaba con la disponibilidad de la red en todo momento.

Al comienzo, el diseño de la base de datos demandó más tiempo de lo estimado pero aun así se creyó necesario para la comprensión del contexto y continuidad del proyecto. Posteriormente los temas planteados en muchos de los casos se relacionaban con la codificación, lo cual no implicó discusiones largas para encontrar su solución.

3.5 Mover personal

La metodología XP recomienda mover al personal de sus tareas para evitar pérdidas de conocimiento y cuellos de botella. Si solo una persona del equipo puede trabajar en un área determinada, y por algún motivo debe retirarse o se encuentra con mucha labor, el proyecto se retrasará. La capacitación cruzada[4] es considerada importante en las compañías que tratan de evitar islas de conocimiento, las cuales son susceptibles de generar pérdida de información importante.

Un equipo es mucho más flexible si todos saben lo suficiente sobre cada parte del sistema, evitando que algunos estén sobrecargados de tareas mientras que otros miembros tengan poco que hacer. La programación en pareja se convierte en una herramienta muy importante para lograr el objetivo del traslado del personal sin perder productividad y rendimiento [11].

Para el desarrollo del presente proyecto se debió adquirir conocimiento acerca de dos herramientas: un nuevo lenguaje de programación y un *framework*[5]. El primero fue Kotlin[6], con el cual se implementó la aplicación nativa y fue designado a un miembro del grupo. El segundo, Laravel[7], se utilizó para crear el sistema de gestión y se destinó al restante miembro. Al finalizar la capacitación, ambos compartieron el conocimiento de modo que todo el equipo fuera capaz de hacer implementaciones cuando fuera necesario.

Al tratarse de dos integrantes, la posibilidad de hacer una rotación de programadores resultaba imposible. Sin embargo cada decisión de diseño fue tomada por ambos miembros y los detalles de implementación fueron permanentemente compartidos con el compañero, evitando las islas de conocimiento y cuellos de botella.

3.6 Modificar XP

Todos los proyectos tienen características específicas por lo cual XP puede ser modificado para ajustarse adecuadamente al proyecto en cuestión. Al inicio, se aplican las reglas de XP, pero no se debe dudar en cambiar aquellos aspectos en los que no funcione. Es de utilidad que se discuta qué es aplicable al proyecto y que no, y diseñar mejoras a XP en conjunto [11].

En lo que respecta a nuestro trabajo, la principal razón de las modificaciones a la metodología, surgió del hecho de que se encuentran involucrados solo 2 programadores. A continuación se detallan los aspectos que fueron ajustados por parte del equipo.

Las historias de usuario no fueron escritas por el cliente solo: Al cliente se le dificulta llevar a cabo este proceso por sí solo. Surge la necesidad de capacitarlo para esta labor y trabajar en conjunto con el equipo de desarrollo. Exteriorizar las necesidades del cliente no es tarea sencilla, aun así se encontró predispuesto en todo momento para dialogar y canalizar las ideas y finalmente completar esta tarea.

No fueron escritas pruebas de aceptación: Teniendo en cuenta la buena predisposición y conocimiento por parte del cliente, en lugar de crear criterios de aceptación para determinar si una

[4] Capacitación cruzada: Modalidad de capacitación de los empleados que se realiza intercambiando puestos de trabajo que tienen poca relación con el suyo.

[5] *Framework*: Estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado.

[6] Kotlin: Lenguaje de programación fuertemente tipado desarrollado por JetBrains.

[7] Laravel: *Framework* de aplicaciones web con sintaxis expresiva y elegante.

historia es correcta o no, fue el mismo cliente quien probó manualmente tanto la *app* como el sistema a medida que era entregada una versión funcional del proyecto.

Mover personal: Esta tarea no fue llevada cabo, dado que se trataba solamente de dos programadores. No obstante, la propiedad colectiva del código no se vio afectada.

Uso de metáfora del sistema: El proyecto no requiere de metáforas debido a que es de fácil entendimiento tanto para un nuevo desarrollador como para el cliente. Igualmente, se implementaron buenas prácticas de programación, empleando nombres significativos a la funcionalidad que llevan a cabo.

Cliente siempre presente en la codificación: La presencia física del cliente durante el desarrollo del proyecto no fue siempre posible. El lugar de residencia del representante del cliente es distante con respecto al equipo de trabajo, sin embargo en todo momento hubo una comunicación fluida con el mismo.

Codificar primero la prueba: El equipo llevó a cabo la codificación sin tener en consideración esta recomendación de XP, pero luego se comprendió que el desarrollo de las mismas era importante para una buena programación.

Programación en parejas: Ambos miembros del equipo desarrollaban desde su propia computadora, dado que cada uno estaba cargo de una tecnología en particular. Asimismo, si bien la mayor parte del tiempo se compartía el espacio físico, existían momentos donde el equipo trabajaba distanciado. De igual manera, se mantuvo una constante comunicación entre ambos.

Integración secuencial: Dado que cada programador estaba a cargo de una parte específica del proyecto, no existía código que se compartiera. Por lo tanto, asegurar la integración secuencial no era un aspecto preponderante para el proyecto.

Integraciones frecuentes: Cada desarrollador mantenía su repositorio local actualizado, pero como se explicó en el punto anterior no existía la necesidad de compartir uno.

Máquina dedicada a integraciones: Nuevamente, como cada integrante del equipo llevaba su propio versionado, no era necesaria una computadora única dedicada a esto.

3.7 División de tareas

La estimación de un proyecto de software sigue siendo al día de hoy una tarea compleja. Sin embargo, con el conjunto de recursos presentados en este capítulo fue posible obtener las tareas que se deben llevar a cabo y, en base a ello, elaborar un plan de trabajo. El equipo estimó un total de 461 horas, divididas en 9 semanas. A continuación se detallan las tareas que se desarrollaron dentro del proyecto y en la tabla 5 se especifica quién la realizó y su duración en el tiempo.

Tarea 1: Conocimiento del entorno laboral, el circuito administrativo y los documentos que se utilizan
Tarea 2: Determinar los requerimientos principales del proyecto
Tarea 3: Dividir los requerimientos en distintas tareas más pequeñas y estimar el tiempo de desarrollo de las mismas
Tarea 4: Desarrollar el modelo de entidad-relación (DER), que facilitará el diseño de la base de datos
Tarea 5: Seleccionar el motor de base de datos a utilizar, los lenguajes de programación que mejor se adecuen tanto a la aplicación como al Sistema web e investigar y definir los *frameworks* con los cuales se trabajará
Tarea 6: Diseñar y maquetar la interfaz de la *app* para un dispositivo móvil y del sistema en un navegador web

Tarea 7: Desarrollar tanto la aplicación nativa como el sistema de manera iterativa e incremental, de acuerdo a las tareas detalladas anteriormente e incluyendo las peticiones recibidas en el *feedback* del cliente
Tarea 8: Implementar el *testing* y si las pruebas resultan positivas, pasar a la siguiente tarea
Tarea 9: Implementar el proyecto en un ambiente de producción
Tarea 10: Realizar un seguimiento de sistema y de la aplicación

Tarea	Duración (en horas)	Desarrollador
Tarea 1		
Conocimiento del entorno laboral, el circuito administrativo y los documentos que se utilizan	8	Cintia / Enzo
Tarea 2		
Determinar los requerimientos principales del proyecto	8	Cintia / Enzo
Tarea 3		
Dividir los requerimientos en distintas tareas más pequeñas y estimar el tiempo de desarrollo de las mismas	12	Cintia / Enzo
Tarea 4		
Desarrollar el modelo de entidad-relación (DER)	12	Cintia / Enzo
Tarea 5		
Seleccionar lenguaje de programación y entorno de desarrollo para la App	16	Cintia
Seleccionar lenguaje de programación e investigar y decidir frameworks para el Sistema Web	16	Enzo
Tarea 6		
Diseño y maquetación de la app	20	Cintia
Diseño y maquetación del sistema web	20	Enzo
Tarea 7		
Desarrollar la aplicación nativa de acuerdo a las especificaciones dadas	140	Cintia
Desarrollar el sistema web de acuerdo a las especificaciones dadas	140	Enzo
Tarea 8		
Implementar el testing de la app	16	Enzo
Implementar el testing del sistema web	16	Cintia
Tarea 9		
Implementar el proyecto en un entorno de producción	12	Cintia / Enzo
Tarea 10		
Realizar un seguimiento del proyecto	25	Cintia / Enzo
Total	461	

Tabla 5: Estimación de tiempo de tareas

El proceso de desarrollo de las tareas se puede observar a través del siguiente diagrama de Gantt (imagen 7):

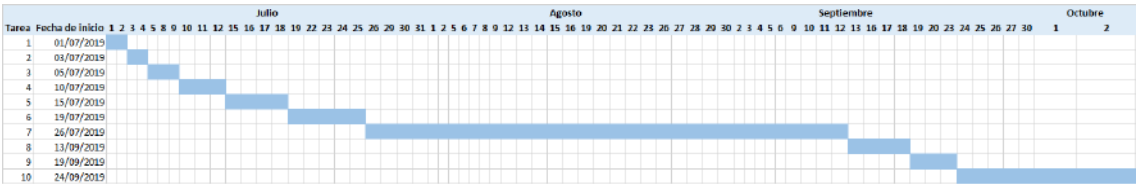


Imagen 7: Diagrama de Gantt

CAPITULO 4: Diseño

A diferencia de las metodologías tradicionales, el diseño se realiza durante todo el tiempo de vida del proyecto, siendo constantemente revisado y muy probablemente modificado debido a cambios presentados durante el desarrollo. En XP solo se diseñan aquellas historias de usuario que el cliente ha seleccionado para la iteración actual por dos motivos:

- Por un lado se considera que no es posible tener un diseño completo del sistema y sin errores desde el principio.
- Dada la naturaleza cambiante del proyecto, el hacer un diseño muy extenso en las fases iniciales para luego modificarlo, se considera un desperdicio de tiempo.

Los aspectos que se tratarán a continuación son: simplicidad en el diseño, metáfora del sistema, tarjetas CRC (Clase-Responsabilidad-Colaboración), refactorización y soluciones *spike*.

4.1 Simplicidad en el diseño

Una de las partes más importantes de la filosofía XP es la simplicidad en todos los aspectos. Un diseño sencillo requiere menos tiempo que uno complejo, por lo tanto si se encuentra dificultoso se debe reemplazar en el momento detectado antes de perder mucho tiempo con él.

La simplicidad es difícil de medir, es una cualidad muy subjetiva. XP propone que sea el equipo quien decida que es “simple” y juntos juzguen su código subjetivamente. Recomienda cuatro cualidades subjetivas: probables, navegables, comprensibles y explicables. Probable significa que se pueden escribir pruebas unitarias y de aceptación para verificar automáticamente los problemas. Esto impacta en todo el diseño y al acoplamiento de objetos en el sistema. Se aconseja dividir el sistema en pequeñas unidades comprobables. Navegable es la calidad de encontrar lo que se desea cuando se desea. Una forma conveniente de lograr esto es utilizando nombres significativos, y aplicando correctamente el polimorfismo, la delegación y la herencia. Comprensible y explicable son características obvias pero altamente subjetivas. Se pueden explicar como cualidades que significan que es fácil mostrar cómo funciona todo.

Los diseños simples a menudo surgen después de que el proyecto ha estado funcionando por un tiempo. El mejor enfoque es codificar lo planificado para la iteración hasta adquirir el conocimiento suficiente para emplear un diseño más simple. Una vez logrado se recomienda refactorizar incrementalmente. Se sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración que se esté trabajando [11].

Para el proyecto se siguió lo que recomienda XP, solo invirtiendo el tiempo exclusivamente necesario en elaboración de diagramas y diseño de interfaz gráfica. Para el diseño de las interfaces, se procedió a realizar un bosquejo en papel de la ubicación de los distintos elementos tanto en la app como en el sistema, siguiendo las indicaciones del cliente.

En lo que respecta a la elaboración de diagramas, se realizaron esquemas manuales de tarjetas CRC, el diagrama de casos de uso (imagen 8), algunos diagramas de secuencia y el modelo de Entidad-Relación (DER) (imagen 9), del cual surgieron varias versiones a medida que se incorporaban funcionalidades al proyecto. Más allá de haberse tratado de diagramas muy simples fueron útiles para una mayor comprensión del contexto y del proyecto en sí. En el caso del DER se usó MySQLWorkbench[8]. Esta herramienta facilita visualizar las relaciones entre las entidades de

[8] MySQL Workbench es una herramienta visual unificada para arquitectos de bases de datos, desarrolladores y DBA. Proporciona modelado de datos, desarrollo de SQL y herramientas de administración integrales para configuración del servidor, administración de usuarios, copia de seguridad y mucho más.

manera gráfica. Una vez finalizada la creación del DER, permite aplicar ingeniería directa, es decir, partiendo del diagrama se puede generar la base de datos con todas sus tablas y correspondientes claves primarias y foráneas. Esto agilizó los tiempos ante la necesidad de realizar nuevas modificaciones al diagrama, dando la posibilidad de regenerar automáticamente las tablas ya creadas.

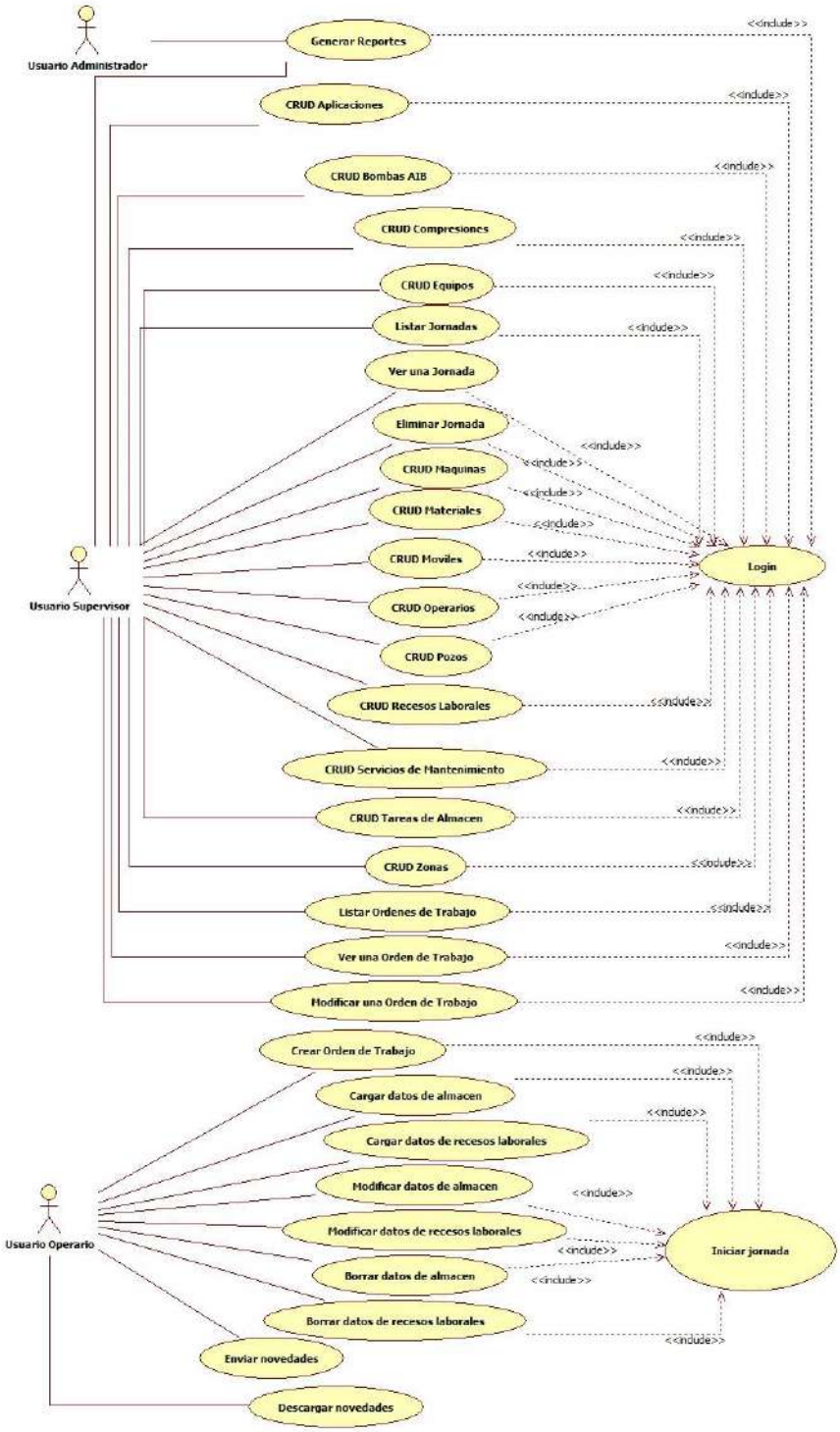


Imagen 8: Diagrama de casos de uso

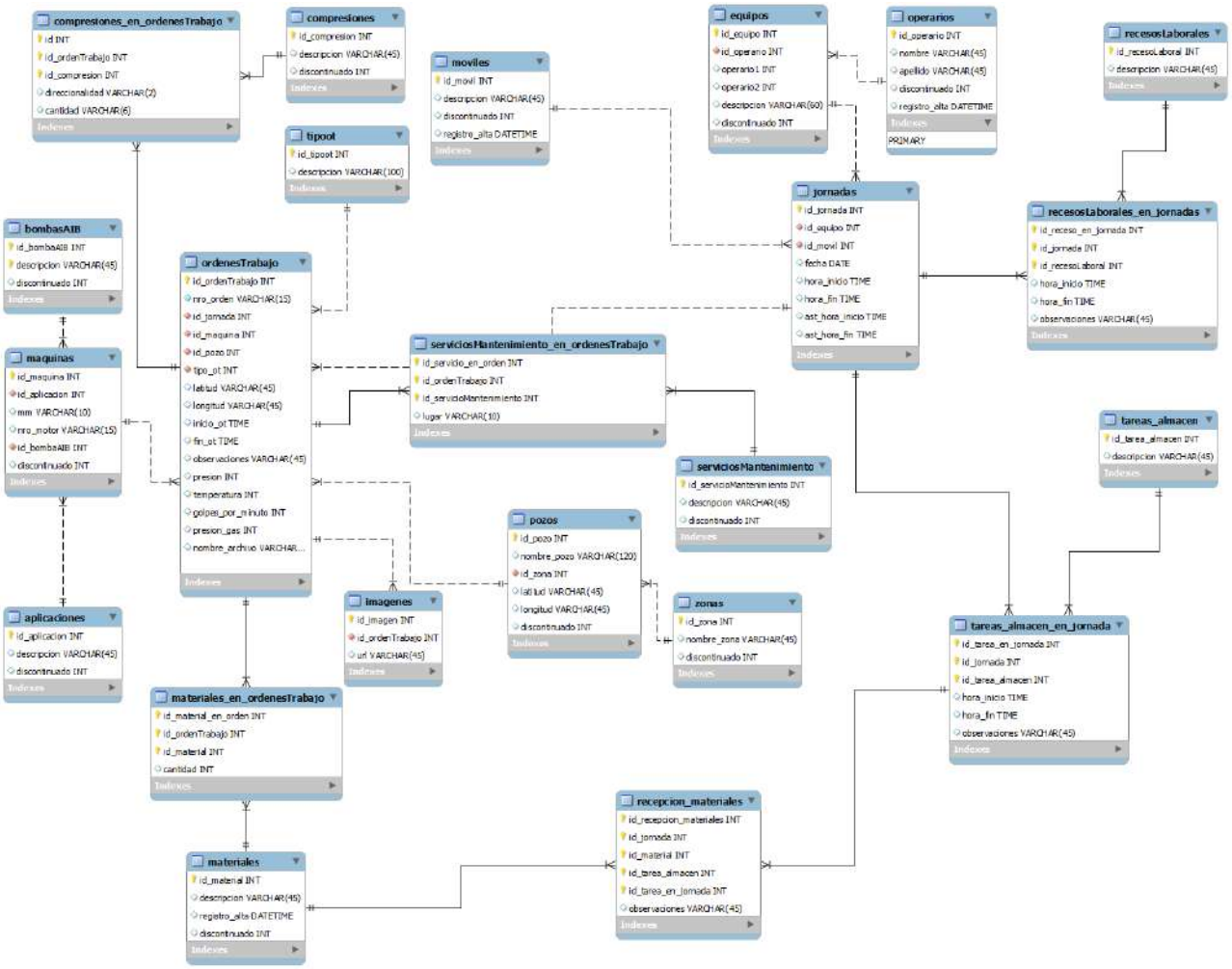


Imagen 9: Diagrama de Entidad Relación (DER)

4.2 Metáfora del sistema

Los equipos desarrollan una visión común sobre cómo funciona el programa, llamada “historia”. La cualidad más importante es poder explicar el diseño del sistema a nuevas personas sin que tengan que leer enormes documentos. El diseño debe tener una estructura que ayude a nuevos integrantes a empezar a contribuir rápidamente. Otra característica es la elección de nombres de las clases y sus métodos. La consistencia de estos es muy valiosa para entender el diseño general del sistema y la reutilización del código [11].

En el desarrollo de nuestro proyecto no se requirió del empleo de metáforas, debido a que el sistema y su contexto son de fácil comprensión para los clientes y desarrolladores. La convención de nombres claros facilitó enormemente el logro de la propiedad colectiva del código, con solo ver el nombre de la clase o método se puede entender la función que desempeña y el lugar que ocupa en el sistema.

4.3 Tarjetas CRC (Clase-Responsabilidad-Colaboración)

Las tarjetas de Clase, Responsabilidad y Colaboración (CRC) se utilizan para diseñar el sistema en equipo. Cuantas más personas puedan contribuir en el diseño, mayor cantidad de buenas ideas

se podrán incorporar. La principal funcionalidad es ayudar a dejar el pensamiento procedural para incorporarse al enfoque orientado a objetos.

Cada tarjeta representa un objeto. El nombre de la clase puede ser escrito en la parte superior y define sus responsabilidades en la sección izquierda y las clases colaboradoras en su parte derecha. Se dice que "puede ser escrito" porque normalmente los participantes solo necesitan pocas tarjetas con el nombre de la clase y prácticamente ninguna está escrita de manera completa. De todas formas, en caso de ser necesario, pueden escribirse en su totalidad y conservarse como documentación. En la imagen 10 se muestra la estructura de las tarjetas CRC.

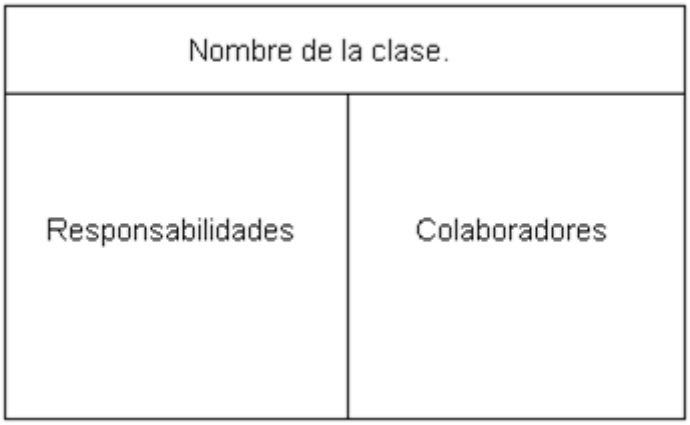


Imagen 10: Tarjeta CRC.

En el desarrollo del proyecto YAL, las tarjetas fueron elaboradas en forma conjunta, de este modo el diseño pudo ser fácilmente asimilado por ambos desarrolladores favoreciendo la propiedad colectiva del código. En las primeras iteraciones del proyecto, las tarjetas CRC fueron de mucha utilidad. Las tarjetas iniciales indicaban clases que contenían métodos propios que no hacían llamados a otras clases. Se siguió con aquellas que hacían llamados a las ya implementadas y así sucesivamente en las siguientes iteraciones. La manipulación de las tarjetas permitió obtener el DER con mayor facilidad y agilizar su modificación a medida que se planteaban nuevas necesidades. A continuación, en las imágenes 11 y 12 se presentan algunos ejemplos:

Equipo	
Mostrar Equipos **Crear Equipos** **Mostrar un Equipo en particular** **Modificar un Equipo** **Discontinuar un Equipo** **Eliminar un Equipo**	Operario

Imagen 11: Tarjeta CRC Equipo

Orden	
Crear una Orden de Trabajo	Compresión
Mostrar Ordenes de Trabajo	Jornada
Mostrar una Orden de Trabajo en particular	Máquina
Modificar una Orden de Trabajo	Material
Eliminar una Orden de Trabajo	Pozo
	Servicio

Imagen 12: Tarjeta CRC Orden

En el Anexo 7 se encuentran el resto de las tarjetas CRC escritas para el proyecto YAL.

4.4 Soluciones *spike*

En muchas ocasiones los equipos de desarrollo se enfrentan a requerimientos de los clientes los cuales generan problemas desde el punto de vista del diseño o la implementación. La solución *spike* o de punta, es una herramienta de XP para abordar este inconveniente. Se trata de una aplicación muy simple, completamente desconectada del proyecto, que explora el problema y propone soluciones potenciales. El objetivo es reducir el riesgo de un problema técnico o aumentar la confiablidad de la estimación de una historia de usuario. Una vez logrado el propósito, el *spike* es descartado [11].

El desarrollo del sistema SGPDM requirió del estudio del *framework* Laravel. Se destinó a uno de los desarrolladores para esta tarea, mientras el otro también destinaba horas de investigación al lenguaje de programación Kotlin para implementar en la aplicación PDMApp. La instalación de las nuevas tecnologías también se incluye en este periodo, que aproximadamente fue de 14 horas cada uno. Esta investigación inicial comprende indagar en la correcta instalación y las principales funcionalidades propias del lenguaje o el *framework* para comenzar con la implementación. Si bien durante el desarrollo surgieron inquietudes que necesitaron ser suprimidas, el tiempo no fue significante. Algo similar surgió para implementar las pruebas tal como XP las recomienda. Se debió recurrir a las librerías JUnit4 y Mockito para el caso de la aplicación y el *framework* PHPUnit para el sistema.

Al término de la investigación, en ambos casos, cada desarrollador compartió sus conocimientos con su compañero en el empleo de herramientas y tecnologías para no generar cuellos de botella. La capacitación de cada desarrollador demandó un periodo no mayor a tres horas.

XP recomienda asignar estas soluciones *spike* por parejas, sin embargo tal como se explicó en el capítulo de planeación no fue hecho de esta forma, lo que no representó un problema. El objetivo de lograr una comprensión rápida de cada uno de estos asuntos fue logrado, asegurando el cumplimiento de los plazos del proyecto. El equipo de desarrollo fue cauteloso para compartir el conocimiento adquirido, aspecto importante que procura garantizar XP, permitiendo que ambos miembros pudieran hacer implementaciones si lo necesitaban, lo cual no fue necesario en ambos casos.

Uno de los aportes más importantes del concepto de soluciones *spike* fue la posibilidad de desligarse completamente del proyecto mientras se trabaja en él, lo que facilita la concentración del esfuerzo personal en adquirir conocimiento en lugar de disiparlo intentando solucionar un problema sin saber cómo enfrentarlo.

4.5 No aumentar funcionalidades antes de tiempo

La mayoría de los desarrolladores tienden a predecir necesidades futuras e implementarlas antes. Solo el 10% de ese material adicional se utilizará, por lo tanto se estaría desperdiciando tiempo de desarrollo y complicando el diseño innecesariamente. En XP se analiza lo que se desarrollará en la iteración actual, olvidando por completo cualquier necesidad futura que se pueda presentar [11].

Durante el desarrollo del proyecto YAL se cumplió con esta recomendación. Teniendo en claro los objetivos de cada iteración y no adicionando funcionalidades antes de tiempo, se consiguió un mejor rendimiento en los tiempos de entrega. Sin embargo, la idea de no adicionar funcionalidades antes de tiempo atrajo pequeños inconvenientes. En la reunión junto al cliente, en la que se diseñó el plan de entregas, quedaron claros algunos aspectos del proyecto. En los encuentros de cada iteración sólo se discutían temas concernientes a dicha iteración, por tanto cualquier detalle de otra posterior fue omitido, aun cuando se tenía en claro que tarde o temprano debía ser considerado. Por ejemplo, en las primeras reuniones se sabía sobre las mediciones de compresión de un motor pero no se implementó hasta la tercera iteración por decisión del cliente. El costo de incluirlas fue considerablemente superior a que si se hubiese realizado de una vez en la primera iteración. Se debieron modificar las secciones de la aplicación y los informes que esta arrojaba, al igual que los reportes del sistema SGPDM que se relacionaban con esta información. Algo similar ocurrió con el tipo de orden de trabajo, que si bien estaba claro en un principio, no se implementó en la iteración siguiente. El hecho de hacer modificaciones en el diseño implica realizar pruebas, las cuales requieren atención y tiempo del equipo de desarrollo.

4.6 Refactorización

Resulta normal que los desarrolladores mantengan el diseño, incluso luego de que se dificulte su lectura y entendimiento, por temor a modificarlo y arruinar el trabajo realizado. En XP se parte de un diseño general y simple, que no debe tardar en conseguirse, al cual se le hacen adiciones y correcciones a medida que el proyecto avanza. La refactorización en el código pretende conservarlo tan sencillo y fácil de mantener como sea posible. Cuando se elimina la redundancia, se eliminan funcionalidades no utilizadas y se logran las metas de simplicidad tanto en el código en sí mismo como en la lectura y en mantenimiento. La refactorización a lo largo de todo el ciclo de vida del proyecto ahorra tiempo y aumenta la calidad justificando el esfuerzo extra que se realiza para corregir funcionalidades innecesarias [11].

Surgieron situaciones en el proyecto YAL que condujeron a una refactorización. Por ejemplo, en un principio el tipo de orden de trabajo era un atributo de la tabla “ordenestrabajo” en el DER. Posteriormente se quitó y se creó una nueva tabla (tipoot) y se la relacionó con la mencionada con anterioridad. De esta manera, se logró normalizar la base de datos, mejorar el diseño y facilitar la lectura a los desarrolladores, aunque para el cliente no implicaba un cambio visual o de comportamiento en el sistema y aplicación. También se eliminaron métodos que se encontraban repetidos dentro del proyecto. Suprimir la redundancia simplificó la implementación de modificaciones y el agregado de funciones. Si bien llevar a cabo esta práctica resultó difícil, se logró gracias a que constantemente se realizaban las pruebas correspondientes ante cada modificación, para asegurar el correcto funcionamiento.

CAPITULO 5: Codificación

En metodologías tradicionales, la codificación es un proceso al cual solo se llega después de largas fases de análisis y diseño. En XP el proceso es muy diferente. Prácticamente desde un principio se inicia con la codificación, favoreciendo el logro del objetivo de hacer entregas frecuentes al cliente.

Los aspectos a tener en cuenta en este apartado son: cliente siempre presente, código siguiendo estándares, codificar la prueba primero, programación en parejas, integración secuencial, integraciones frecuentes, maquina dedicada a integraciones y propiedad colectiva.

5.1 Cliente siempre presente

Uno de los requerimientos de XP es tener el cliente siempre disponible. No solo para ayudar al desarrollo del equipo, sino también para ser parte de él. Todas las fases de un proyecto XP requieren comunicación con el cliente, preferentemente cara a cara. Lo mejor es que se integren al proyecto una o dos personas que tengan en claro el circuito administrativo/operativo de la empresa, que ayuden a evacuar las dudas que puedan surgir y que garanticen que la implementación desarrollada cubre las necesidades planteadas.

Las historias de usuarios son escritas por los clientes, con ayuda de los desarrolladores, para permitir estimar el tiempo y las prioridades de las mismas. Este trabajo cooperativo asegura que las necesidades funcionales del sistema van a estar cubiertas en estas historias de usuarios. Además, durante las reuniones de planeación de entrega será el cliente quien negociará cuáles historias serán incluidas en cada iteración y los tiempos de las mismas.

El cliente también será necesario para ayudar con las pruebas funcionales. Se deberán crear datos de prueba y calcular o verificar los resultados logrados, y de esta forma obtener la retroalimentación correspondiente. Puede ocasionar inconvenientes que más de un cliente frecuente las reuniones de planeación, generando discusiones por diferencia de opiniones. Para resolver este problema, es recomendable organizar reuniones grupales ocasionales [11].

Trasladando este requerimiento de XP a nuestro contexto, el cliente si bien se hizo presente en muchas de las reuniones de planeación, no era posible tenerlo a tiempo completo en caso de que surgieran dudas. Por tal motivo, se debió implementar una estrategia de comunicación distinta, en la cual se realizaron llamadas o intercambiaron correos electrónicos, priorizando un lenguaje claro, sencillo y directo en el contenido. Si bien esta estrategia no fue igual de efectiva que haber tenido al cliente acompañando el desarrollo, fue suficiente para lograr una buena comunicación.

Cabe destacar, que en nuestro caso tuvimos inconvenientes en algunas reuniones, en las cuales los representantes del cliente debatían intereses y requerimientos para el proyecto, sin tener un concepto claro de lo que solicitaban. Por lo tanto, en lugar de resultar productivas, se generaba una pérdida de tiempo valioso. A partir de esto, se le solicitó al cliente, que solamente envíe un representante con el conocimiento necesario para transmitir a los desarrolladores.

5.2 Código siguiendo estándares

El código debe seguir un formato según los estándares de codificación acordados. Estos mantienen el código consistente y fácil para todo el equipo, para su lectura o refactorización. El código que se ve igual fomenta la propiedad colectiva [11].

Para el equipo de desarrollo del proyecto YAL, llevar adelante este requerimiento de XP no resultó una complicación. En proyectos anteriores solíamos seguir ciertas reglas en lo que respecta a la escritura de código. Particularmente en este caso, acordamos un esquema de estándares generales y luego surgieron otros ligados al lenguaje de programación usado por cada desarrollador. Por razón de seguir una disciplina, se formalizaron en un documento que se adjunta en el Anexo 9.

5.3 Codificar la prueba primero

Cuando se crean primero las pruebas antes que el código, la codificación resulta más fácil, rápida y ahorra mucho tiempo. Las pruebas unitarias ayudan al programador a considerar realmente lo que hay que hacer. Además, generan una retroalimentación inmediata y permiten definir el alcance y la finalización de nuestra codificación. En el momento que todas las pruebas funcionen correctamente, se sabe que el trabajo está terminado.

También hay un beneficio para el diseño del sistema. A menudo, es muy difícil probar ciertos sistemas, debido a que son codificados primero y se testean luego por un equipo completamente diferente al que lo codificó. Creando las pruebas primero, el diseño se verá influenciado por el deseo de probar todo lo que sea valioso para el cliente y será más fácil de testear.

Hay un procedimiento para desarrollar primero la prueba. Se crea una prueba para definir algunos pequeños aspectos del problema en cuestión. Luego se crea el código más simple que pase esa prueba. Seguidamente, se crea una segunda prueba. Ahora se agrega al código que se acaba de crear para pasar esta nueva prueba. Y sucesivamente se continúa hasta que no quede nada por testear [11].

En cuanto a la experiencia en el proyecto YAL, seguir este planteamiento de XP no resulto fácil por diversos motivos. El principal y común a ambos desarrollos, es que los integrantes indagaban con tecnologías nuevas. Para codificar la prueba primero se debía adicionar conocimiento y tiempo de investigación, en tal sentido se optó por no codificar las pruebas primero.

5.4 Programación en parejas

Todo el código que se enviará a producción es creado por dos personas que trabajan juntas en una sola computadora. La programación en pareja aumenta la calidad del software sin afectar el tiempo de entrega, dado que dos personas trabajando juntas agregarán tanta funcionalidad como dos que trabajan por separado. Con mayor calidad, se obtienen grandes ahorros a futuro en el proyecto. La programación en pareja es una habilidad social que lleva tiempo aprender. Se esfuerza por lograr una forma cooperativa de trabajo que incluya dar y recibir de ambos integrantes. Es útil si alguien del equipo con experiencia muestra al resto como debería ser [11].

En nuestra experiencia, este requerimiento de XP no fue llevado a cabo. Como se nombró al inicio, el equipo de trabajo estuvo conformado por dos personas, donde cada una poseía conocimientos elevados en distintas herramientas empleadas en este proyecto. Dado que cada sistema se desarrolló en un lenguaje de programación diferente, y cada integrante tenía mayor entendimiento de uno en particular por experiencias anteriores, resultó más productivo que uno se encargara de la aplicación Android, mientras que el otro hacía lo propio con el sistema web. Aunque no se trabajó en una sola máquina, el estar en el mismo espacio geográfico facilitó la comunicación y el apoyo constante entre los integrantes, como así también revisar en pareja cada nueva funcionalidad agregada con el objetivo de mitigar errores, mejorar la calidad y discutir alternativas.

Es muy probable que en el caso de no tener dominio de las herramientas que se usan para desarrollar, sea más productivo trabajar a la par en una misma computadora. De esta forma, asimilar conceptos nuevos, mejorar la calidad y organizar el código resulta más sencillo y ágil.

5.5 Integración secuencial

Sin tener un orden de integración, los desarrolladores testean localmente su código e integran creyendo que todo está bien. En la integración en paralelo hay una combinación de código que no ha sido testeada antes en conjunto, y eso puede generar problemas y no ser detectados. Otros inconvenientes suceden cuando no hay una última versión clara. Esto aplica no solo al código sino también al conjunto de pruebas unitarias que debe verificar la correcta funcionalidad del código. El conjunto de pruebas debe ser completo, correcto y consistente para no estar persiguiendo errores que no existen y no detectando errores que si existen.

Muchos proyectos utilizan una herramienta de repositorio de código fuente para controlar la integración. La mayoría de estos parecen fomentar el desarrollo secuencial y la integración paralela, bloqueando un archivo para que solo un desarrollador pueda agregar funcionalidad a la vez. Esta solución no aborda el problema raíz. Lo deseable es que los desarrolladores puedan proceder en paralelo realizando cambios en cualquier parte del sistema pero también una integración libre de errores que no pierda cambios.

La integración estrictamente secuencial en combinación con la propiedad del código colectivo es una solución simple a este problema. El código nuevo se lanza al repositorio de código fuente por turnos. Esto no implica que no pueda integrar sus propios cambios con la última versión en su propia computadora en cualquier momento que lo desee.

El desarrollo continúa en paralelo mientras que la integración es secuencial. Lo más simple es una sola computadora dedicada a la integración. Los desarrolladores se secuencian implícitamente por turnos para usarlo [11].

5.6 Integraciones frecuentes

Los desarrolladores deberían integrar código al repositorio con la mayor frecuencia posible. La integración continua evita esfuerzos de desarrollos divergentes o fragmentados, donde los desarrolladores no se comunican entre sí sobre lo que se puede reutilizar o lo que se puede compartir. Cada equipo de desarrolladores es responsable de integrar su propio código cada cierto tiempo. Esto puede ser cuando las pruebas unitarias se ejecutan al 100% o se termina una parte más pequeña de la funcionalidad planificada.

La integración continua evita o detecta tempranamente problemas de compatibilidad [11].

5.7 Máquina dedicada a integraciones

Una sola computadora dedicada a la integración secuencial funciona realmente bien cuando el equipo de desarrollo está situado en el mismo lugar. Esta computadora actúa como un *token* físico que controla la integración y permite a los desarrolladores ver quien está subiendo cambios y cuando. Así, la estabilidad está asegurada. Al tratarse de una sola computadora, el conjunto de pruebas se encuentra siempre actualizado. Si al ejecutarse las pruebas unitarias resultan exitosas al 100%, los cambios son confirmados. Si fallan, los cambios se restituyen y se depuran en la propia máquina del desarrollador [11].

5.8 Propiedad colectiva

La propiedad colectiva alienta a todos a aportar nuevas ideas al proyecto. Cualquier desarrollador puede agregar funcionalidad, corregir errores, mejorar diseños o refactorizar. Esto genera que nadie se convierta en un cuello de botella para los cambios. Al tratarse de varios programadores trabajando en un mismo sistema no se puede atribuir la responsabilidad solo a una persona. Cada desarrollador debe crear sus propias pruebas unitarias al código que escribe y asegurarse que se ejecutan satisfactoriamente. En la práctica, la propiedad colectiva es más confiable que poner una sola persona a cargo de clases específicas. Especialmente porque una persona puede abandonar el proyecto en cualquier momento [11].

5.9 Consideraciones finales

Dado que las últimas cuatro reglas de XP no fueron aplicadas por motivos similares, decidimos realizar una explicación general. En nuestro caso, cada integrante puso su enfoque en un sistema diferente, bajo un lenguaje distinto. El propio desarrollador era el encargado de efectuar las pruebas unitarias necesarias para asegurar la consistencia de todo el código al agregarse una nueva funcionalidad. Asimismo, es pertinente destacar que no existía una computadora dedicada a la integración, dado que ambos programadores contaban con su propia computadora como repositorio de ese sistema en sí.

CAPITULO 6: Pruebas

El *testing* es una actividad desarrollada para evaluar la calidad del producto y para mejorarlo al identificar defectos y problemas. El *testing* de software consiste en la verificación dinámica del comportamiento de un programa sobre un conjunto finito de casos de prueba, apropiadamente seleccionados a partir del dominio de ejecución que usualmente es infinito, en relación con el comportamiento esperado. XP enfatiza mucho en aspectos relacionados con las pruebas. Todo el código que se escribe debe ir acompañado por *test* que se ejecuten continuamente. Representan la mejor documentación del sistema y del buen uso depende el éxito de otras prácticas que buscamos como la propiedad colectiva del código y la refactorización.

Los *tests* son fundamentales porque nos permite desarrollar incrementos de funcionalidad muy rápidamente sin temor a introducir *bugs*. Al heredar un proyecto y tener que agregar una nueva funcionalidad sin el conocimiento completo del comportamiento, o simplemente porque el proyecto evoluciona y podemos olvidar detalles de implementaciones anteriores, escribir *test* brinda *feedback* inmediato y seguridad de que el sistema continúa siendo sustentable.

6.1 Pruebas unitarias

Las pruebas unitarias se enfocan en validar que la unidad más pequeña en un sistema cumple con el comportamiento esperado. Una vez que cada una de estas unidades ha sido probada y todas las pruebas son pasadas, se pueden aplicar otras formas de prueba para validar que el *software* cumple con sus otros requisitos.

Las pruebas unitarias son una parte esencial de XP, pero en esta metodología, a diferencia de otras, son un poco diferentes. En primer lugar se debe crear o descargar un *framework* de pruebas unitarias para poder crear pruebas automatizadas. En segundo lugar, se deben probar todas las clases en el sistema. El código que no haya sido probado, no puede ser integrado a producción.

Durante la vida de un proyecto, una prueba automatizada puede ahorrar cientos de veces el costo de crearla al encontrar y protegerse contra los errores. Cuanto más difícil sea crear la prueba, más se necesitará porque serán mayores los beneficios a futuro.

Las pruebas unitarias permiten:

- **La propiedad colectiva:** Requerir que todo el código pase todas las pruebas unitarias antes de que se pueda enviar a producción, garantiza que toda la funcionalidad sigue actuando correctamente
- **La refactorización:** Después de cada pequeño cambio, las pruebas unitarias pueden verificar que un cambio en la estructura no introdujo un cambio en la funcionalidad
- **La integración frecuente:** Es posible integrar rápidamente cualquier cambio reciente y luego ejecutar su propia versión más reciente del conjunto de pruebas. Si una prueba falla, significa que esta última versión es incompatible con la versión del resto del equipo de desarrollo. Solucionar problemas pequeños cada pocas horas lleva menos tiempo que solucionar problemas enormes justo antes de la fecha límite

Además, el empleo de pruebas unitarias completas facilitan la liberación continua de versiones por cuanto al implementar algo nuevo y actualizar la última versión, solo es cuestión de ejecutar de forma automática las pruebas ya creadas para saber que la nueva versión no contiene errores. De esta manera, puede decirse que las pruebas unitarias proporcionan una red de seguridad de pruebas de regresión y de validación para que pueda refactorizar e integrarse de manera efectiva [11].

En nuestro caso, este tipo de pruebas fueron llevadas a cabo una vez implementados los módulos correspondientes. La creación de las mismas fue una experiencia nueva para el equipo de

trabajo, sin embargo tanto el lenguaje de programación utilizado en la aplicación como el usado en el sistema, contaban con componentes que facilitaban la ejecución de *tests*. En ambos contextos, fue necesario un tiempo de adaptación y demandó más tiempo de lo planeado poder escribir el código correspondiente a las pruebas. Pero finalmente, la tarea pudo desarrollarse exitosamente.

6.1.1 Pruebas unitarias en Laravel

Para el caso del sistema SGPDM, Laravel ha sido de mucha utilidad, ya que fue desarrollado teniendo en cuenta el *testing*. Desde que se crea el proyecto, proporciona un directorio **/tests** donde se guardaran las pruebas llevadas a cabo. Además, el soporte para pruebas con PHPUnit[9] viene incluido y un archivo "phpunit.xml" ya está configurado para su aplicación. El *framework* también incluye métodos convenientes que permiten probar el proyecto de forma expresa. Cuando se ejecutan pruebas a través de PHPUnit, Laravel ajustará automáticamente el entorno de configuración a *testing* debido a las variables de configuración definidas en el archivo "phpunit.xml". Laravel también configura automáticamente la sesión y la caché en el *driver array* mientras se realizan las pruebas, lo que significa que no se mantendrán los datos de la sesión o la caché durante la ejecución [13].

Ante la pregunta cómo comenzar a probar las aplicaciones Laravel, la respuesta es "HTTP Tests". Con este tipo de pruebas, es sencillo crear cualquier tipo de solicitud para *get*, *post*, *put*, etc. Se pueden enviar datos de solicitud, establecer un usuario autenticado y establecer datos de sesión y encabezado utilizando una API fluida. El objeto de respuesta devuelto tiene métodos de aserción para probar el comportamiento común. Se puede verificar el estado HTTP, la vista devuelta, la sesión o el conjunto de datos de encabezado, o si se produjo una redirección.

Las pruebas HTTP son la forma más fácil de enviar una solicitud a la aplicación y hacer afirmaciones sobre la respuesta. Además, esta prueba de alto nivel proporciona una amplia cobertura, ya que pone en juego muchas capas de la aplicación, incluidos middleware, controladores, modelos, servicios y vistas.

Para crear un nuevo *test*, se debe utilizar el siguiente comando de Artisan[10]:

php artisan make:test NombreTest –unit

Este comando crea una clase que hereda de TestCase, por lo tanto todos sus métodos están disponibles para ser utilizados en los casos de prueba. Como se va a probar cada método de cada clase, lo indicado es nombrar a la clase de la misma forma que el módulo a testear, anexando la palabra *Test*. Por ejemplo: para probar los métodos de la clase *Equipo* se crea la clase *EquipoTest*, y para que se ejecuten correctamente, los métodos deben comenzar con el prefijo "test_", como se muestra a continuación:

```
public función test_editar_datos_equipo(){
    //Código
}
```

O bien, anteponer la notación "@test" antes del método:

[9] PHPUnit es un *framework* de testeo orientado a PHP. Es una instancia de la arquitectura xUnit para *frameworks* de pruebas unitarias

[10] Artisan es el nombre de la interfaz de línea de comandos incluida en Laravel. Ésta proporciona una serie de comandos útiles que podrán ayudar mientras se desarrolla la aplicación

```
/** @test */
public función editar_datos_equipo(){
    //Código
}
```

Una vez generado el *test*, se pueden definir los métodos de prueba como se haría normalmente con PHPUnit. Finalmente, para ejecutar los *test*, se posiciona en la ruta raíz del proyecto y se usa el siguiente comando en la terminal:

vendor/bin/phpunit

En el capítulo 8 se explicará con mayor profundidad los *test* realizados en cada iteración.

6.1.2 Pruebas unitarias en Kotlin

En lo que respecta a Kotlin, Android Studio es el IDE (Entorno Integrado de Desarrollo*)* que se utilizó para el desarrollo de la aplicación Android. Este entorno está diseñado para simplificar las pruebas. Al crearse el proyecto, el entorno genera automáticamente dos directorios en donde se deben colocar las pruebas: androidTest y test.

En el directorio ***/src/androidTest/java/*** (punto 1 de la imagen 13) se deben colocar las pruebas que se ejecutan en un dispositivo real o emulador de hardware. Estas pruebas tienen acceso a las API de instrumentación, las cuales permiten controlar la app desde el código de prueba. De este modo se logra automatizar la interacción de usuarios o cuando las pruebas tengan dependencias de Android que los objetos ficticios no pueden contemplar.

En la ubicación ***/src/test/java/*** (punto 2 de la imagen 14) están las pruebas que se ejecutan en el JVM (*Máquina Virtual de Java*) local sin la necesidad de ejecutarlas en un dispositivo real o un emulador [14].

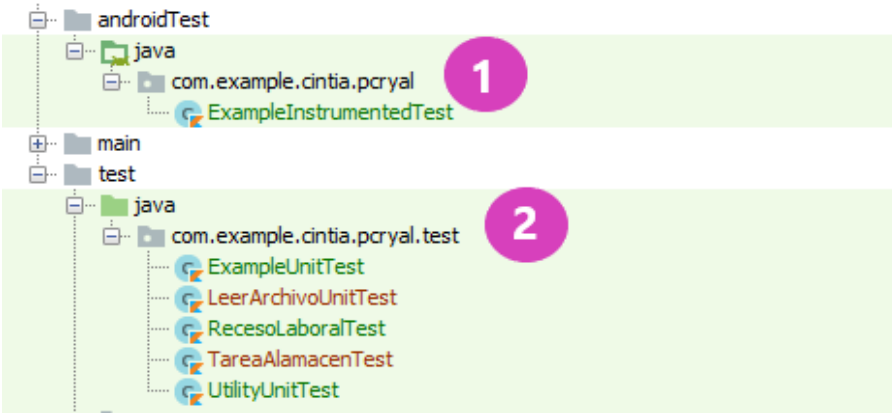


Imagen 13: Pruebas instrumentadas (1) y las pruebas de JVM locales (2), en la vista Project.

JUnit es el marco de trabajo de prueba de unidades más populares y más usadas para Java, y permite escribir las pruebas de una manera más simple y flexible que las versiones anteriores, ya que con JUnit 4 no es necesario hacer lo siguiente [14]:

- Extender de la clase *junit.framework.TestCase*
- Agregar un prefijo al nombre del método de prueba con la palabra clave 'test'

- Usar clases de los paquetes *junit.framework* o *junit.extensions*

Para poder crear test unitarios con JUnit 4 se añadieron las dependencias de JUnit al archivo "gradle" del módulo de la aplicación. Como es habitual crear objetos dobles, se debe añadir también una dependencia de una librería de mocks como Mockito[11], que es muy popular en Android. En el archivo "build.gradle" del módulo de la *app* se especifican las dependencias de bibliotecas de pruebas, como se puede observar en la imagen 14.

```
dependencies {
    implementation 'com.android.support:recyclerview-v7:26.1.0'
    implementation 'com.squareup.picasso:picasso:2.5.2'
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin_version"
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.0'
    implementation 'com.google.android.gms:play-services-location:15.0.1'
    testImplementation 'junit:junit:4.12'
    testImplementation 'org.mockito:mockito-core:2.18.3'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'

}
```

Imagen 14: Configuración de dependencias en archivo gradle.

Para crear una clase de prueba básica de JUnit 4, se debe generar una clase que contenga uno o más métodos de prueba. Un método de prueba comienza con la anotación @Test y contiene el código para ejercitar y verificar una funcionalidad única en el componente que se desea probar.

Para evaluar si los componentes de la app muestran los resultados esperados, se utiliza la biblioteca de Assert[12], que contiene métodos como *is()* y *equals()* permitiendo comparar los resultados obtenidos con los reales.

Una de las principales dificultades a la hora de incluir *tests* en una aplicación no son los *tests* en sí, sino la arquitectura de la misma. Es esencial que esta tenga una estructura lo suficientemente desacoplada como para permitir controlar todo el contexto del *test*. Con el fin de cumplir el objetivo, se deben aislar los métodos y controlar las entradas y salidas de las clases colaboradoras. Para facilitar esto, se visualiza la aplicación como una serie de módulos, donde cada uno son tareas que los usuarios realizan dentro de la *app*, con un área de enfoque definida. Finalmente, la estructura resultó como se observa en la imagen 15.

[11] Mockito es un *framework* de prueba de open source para Java lanzado bajo la Licencia MIT. El *framework* permite la creación de objetos dobles de prueba en pruebas unitarias automatizadas con el propósito de desarrollo basado en pruebas o desarrollo basado en el comportamiento.(https://site.mockito.org/)

[12] Assert es una clase que contiene diferentes métodos que permiten realizar verificaciones en los *test*

Imagen 15: Estructura del proyecto

Las pruebas unitarias instrumentadas o pruebas de la interfaz de usuario (UI) no se realizaron en este proyecto. Para crearlas se debía estudiar Esspreso, un *framework* desconocido para el equipo. Siendo una aplicación con una interfaz sencilla, se creyó conveniente probar cada nueva funcionalidad manualmente. Comprendiendo que no es lo recomendado por la metodología XP, el equipo interactuó con la UI en cada incremento de funcionalidad para obtener el *feedback* en pantalla. De este modo ante la detección de fallas o comportamientos no deseados se realizaban los cambios necesarios para obtener los resultados esperados.

Para las pruebas unitarias se utilizó Mockito *framework*, logrando configurar objetos ficticios para mostrar algún valor específico cuando se lo invocaba. Mockito permite probar interacciones específicas entre un componente y su dependencia dentro de la *app*, usando un marco de trabajo ficticio con el fin de eliminar las dependencias externas del código. Al sustituir las dependencias de Android con objetos ficticios, se puede aislar la prueba de unidad del resto del sistema Android mientras se verifica que se llame a los métodos correctos en esas dependencias [15].

Además de incluir la dependencia en el archivo "*build.gradle"*, como se indicó anteriormente, hay un modelo de programación a seguir:

1. Al comienzo de la definición de la clase de prueba de la unidad, se agrega la anotación *@RunWith(MockitoJUnitRunner.class)*. Ese objeto le indica al panel de pruebas de Mockito que valide el correcto uso del marco de trabajo y simplifique la inicialización de los objetos ficticios.

2. Con el fin de crear un objeto ficticio para una dependencia de Android, se agrega la anotación *@Mock* antes de la declaración del campo.

3. Para indicar el comportamiento de la dependencia, se puede especificar una condición y mostrar un valor cuando se cumpla con la condición mediante los métodos *when()* y *thenReturn().*

4. Por defecto las clases en Kotlin son *final*. Esto significa que si se quiere crear un objeto mock de una clase se debe abrir con la palabra reservada *open* o extender una interfaz. Ambas opciones son tediosas y una limitación a la hora de usar este lenguaje, pero afortunadamente la versión Mockito 2 ha eliminado esta restricción. La dependencia que incluimos está actualizada a las últimas versiones pero también se necesita una configuración manual.

Luego, se deben crear dentro del directorio **/test/** los subdirectorios **/resources/mockito-extensions/**. Seguidamente, generar un archivo con el nombre "org.mockito.pluguins.MockMacker" y escribir:

mock-maker-inline

De esta forma las pruebas corren sin problemas.

6.2 Pruebas de aceptación

Las pruebas de aceptación son creadas desde las historias de usuarios. Durante una iteración las historias de usuarios seleccionadas en la reunión de planeación de iteración serán traducidas a *tests* de aceptación. El cliente especifica escenarios a corroborar cuando una historia de usuario ha sido correctamente implementada. Una historia puede tener una o varias pruebas, las que sean necesarias para asegurar su correcto funcionamiento.

Las pruebas de aceptación son *tests* de sistemas de caja negra. Cada *test* representa una resultado esperado del sistema. Los clientes son responsables de verificar la exactitud y revisar los puntajes de las pruebas para decidir qué pruebas reprobadas son de mayor prioridad. También se usan como pruebas de regresión antes de enviar a producción.

Una historia de usuario no se considera completa hasta que no ha pasado sus pruebas correspondientes. Esto significa que se deben crear nuevos *test* de aceptación en cada iteración o el equipo de desarrollo no reportará progreso.

El control de calidad es una parte esencial del proceso XP. En algunos proyectos, este control es realizado por un grupo separado, mientras que en otros está integrado en el equipo de desarrollo. Las pruebas deben automatizarse para que puedan ejecutarse con frecuencia. El puntaje de la prueba de aceptación se publica al equipo. Es responsabilidad del mismo fijar un tiempo en cada iteración para corregir cualquier prueba que haya fallado. El nombre de estas pruebas surge de los *test* de funcionalidad. Esto refleja mejor la intención, que es garantizar que se cumplan los requisitos y que el sistema sea aceptable por nuestro cliente [11].

En el presente proyecto, este tipo de pruebas no fueron documentadas de acuerdo a las historias de usuario. Sino, que una vez que se completa la iteración, se entrega al cliente tanto la aplicación como el sistema para que la pruebe. De esta manera, otorgará la retroalimentación correspondiente, señalando detalles y/o consideraciones para solucionar problemas encontrados o para tener en cuenta en la siguiente iteración.

6.3 Cuando se encuentra un error

Cuando se encuentra un error, se crean pruebas para evitar que ocurra de nuevo. Un error en producción requiere que se escriba una prueba de aceptación para evitarlo. Crear una prueba de

aceptación antes de la depuración ayuda a los clientes a definir de manera concisa el problema y comunicarlo a los programadores. Los programadores podrán así enfocar sus esfuerzos y solucionar el problema.

Cuando una prueba de aceptación falla, los desarrolladores crean pruebas unitarias para mostrar el defecto desde un punto de vista más específico del código fuente. Las pruebas unitarias que fallan dan una respuesta inmediata al esfuerzo de desarrollo cuando el error ha sido reparado. Cuando las pruebas unitarias se ejecutan al 100%, la prueba de aceptación fallida se puede volver a ejecutar para validar la corrección del error.

Capítulo 7. Descripción de procesos prioritarios

En el presente capítulo se explicarán las tareas principales del proyecto, para comprender el circuito administrativo y operativo de la empresa. Por un lado, se detallarán los distintos perfiles de usuarios existentes en el sistema SGPDM y las diferentes funcionalidades que pueden realizar. También, se explicarán cómo se desarrolla la carga de una jornada de trabajo desde la aplicación PDMApp, acción esencial para los operarios. Por último, se especificarán los reportes que se pueden generar desde el sistema SGPDM.

7.1 Identificación de los perfiles de usuario

Dentro del proyecto tenemos diferentes actores que interactúan con el sistema SGPDM y la aplicación, en esta sección se nombrarán y explicarán las funcionalidades de cada uno.

Supervisor Mecánico de la Operadora (SOP): Este usuario interactúa con el sistema SGPDM, es responsable de alimentar al sistema con datos. Solo este tipo de usuario tiene el permiso de editar, agregar o eliminar cualquier información, como por ejemplo formar un nuevo equipo de trabajo, editar una OT, dar entrada a un nuevo motor, etc. Hoy en día solo una persona tiene este acceso en el sistema.

Administración Yal: Este usuario opera con el sistema SGPDM, analiza los reportes y estadísticas que genera el sistema para diferentes fines: supervisar las actividades y sus empleados, como también para analizar estratégicamente la información para mejorar el desarrollo de las actividades y la empresa.

Operarios: Este usuario opera con la aplicación PDMApp, es responsable de ingresar todas las actividades que realiza en la jornada habitual de trabajo, que luego se verán reflejadas también en el sistema SGPDM. En el próximo apartado se explica con mayor detalle las tareas del usuario Operario.

7.2 Circuito de carga de una jornada de trabajo

Todas las mañanas los equipos se reúnen con su supervisor. Este hace entrega del formulario AST (Asignación Segura Trabajo), una *tablet* e informa qué pozos deberán recorrer en la jornada, siendo cuatro la cantidad máxima de pozos posibles de asignar a cada equipo. Se puede observar el circuito administrativo y operativo en la imagen 16.

Cada equipo debe asentar el inicio de jornada en la aplicación PDMApp, indicando su identificación, el utilitario que usarán en la jornada y horarios (ver imagen 17). No podrán realizar ninguna otra operación en la aplicación sin antes completar esta información.

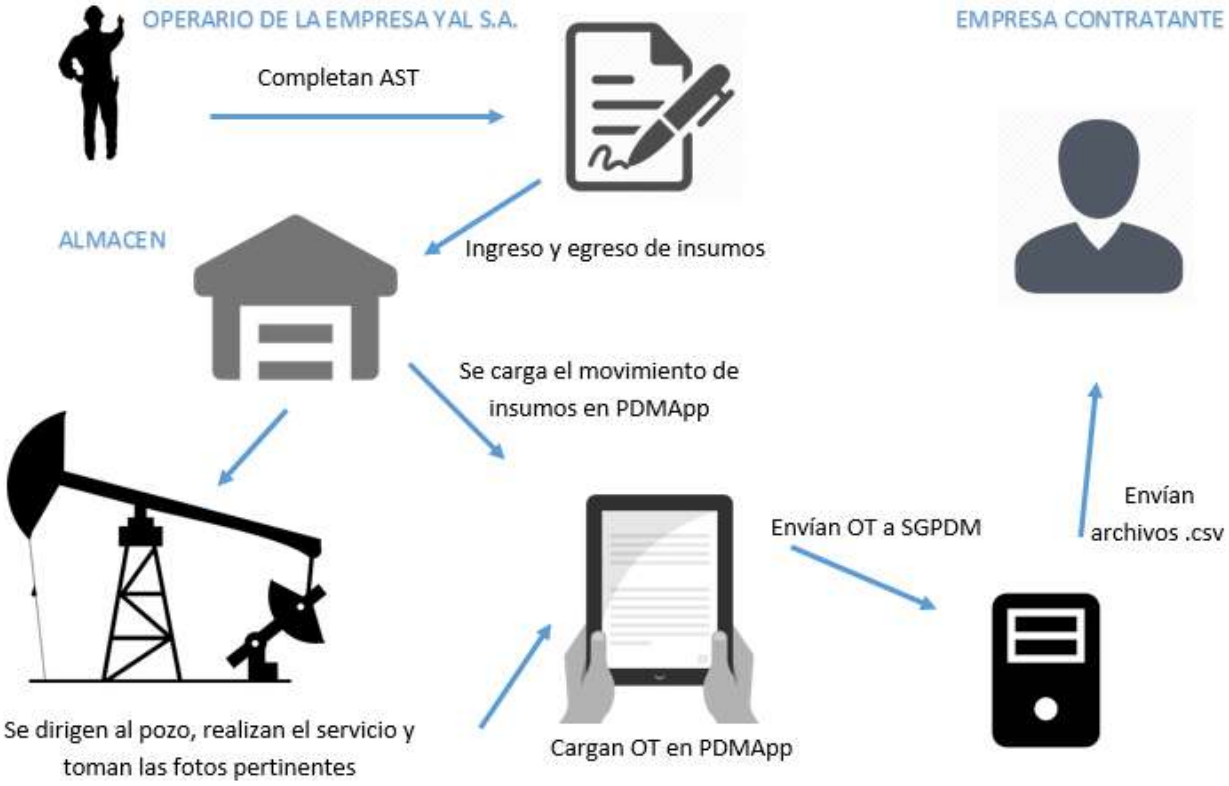


Imagen 16: Circuito administrativo y operativo actual

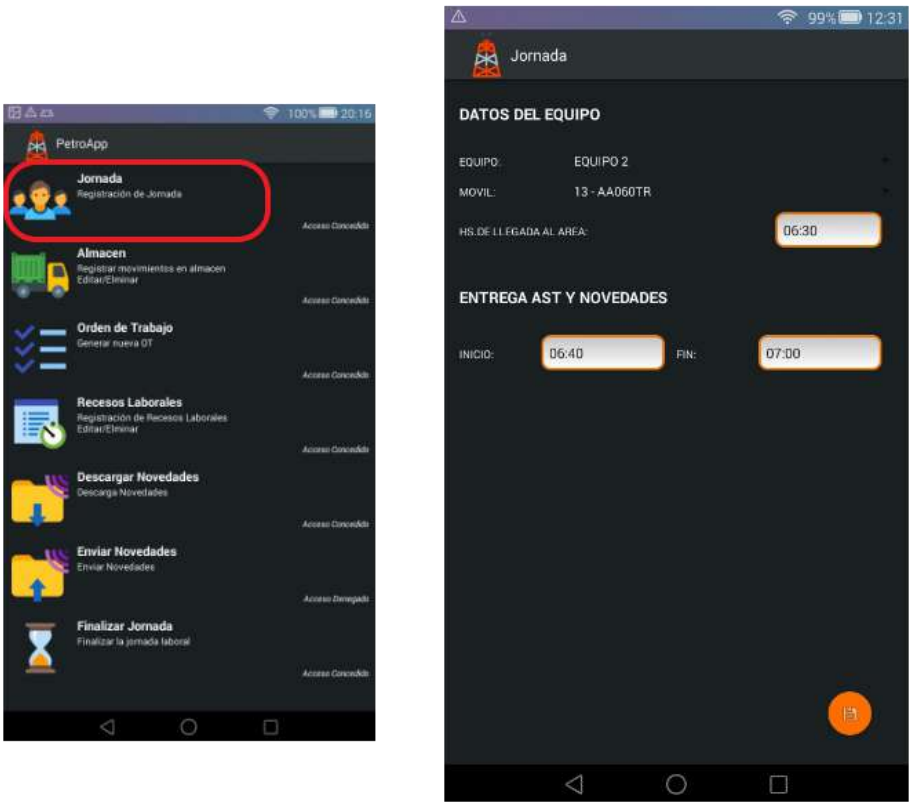


Imagen 17: Pantalla "Jornada"

Los materiales son proporcionados por la empresa contratante y los repuestos que entregan se ingresan en la sección “Almacén” de la *app*. La recepción de materiales debe cargarse con el horario de inicio y fin, descripción de los materiales que se reciben y las cantidades de cada uno (imagen 18). Otra tarea que se desarrolla es la descarga de aceite usado. El aceite "quemado" o "usado" se extrae de los motores cuando se le coloca nuevo aceite. La cantidad de este aceite usado no es de interés, ya que al no utilizar un caudalímetro[13], no se sabe con exactitud cuánto se extrae. El aceite usado se almacena, y cada cierta cantidad de tiempo es llevado a una planta de la empresa contratante para descargarlo y que lo procesen. La descarga lleva un cierto tiempo, de manera que se debe justificar este período en una jornada. Hay una tercera opción genérica, para el caso de que deban ingresar una actividad extraordinaria a lo cotidiano, dando detalles en la sección “observaciones”. Esto se puede apreciar en la imagen 19. Todos los movimientos deben quedar registrados. En caso de querer editar/eliminar una entrada pueden hacerlo, siempre y cuando no hayan finalizado la jornada.

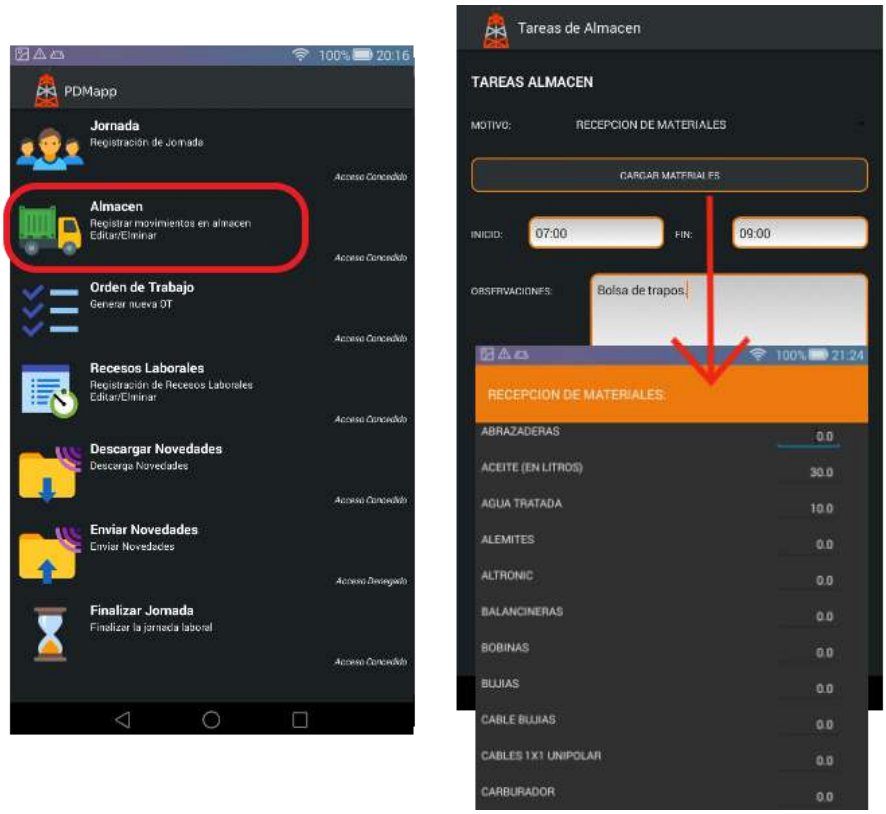


Imagen 18: Pantalla “Almacén”: Entrega de insumos

[13] Un caudalímetro es un instrumento usado para medir lineal, no lineal, la masa o caudal volumétrico de un líquido o un gas.

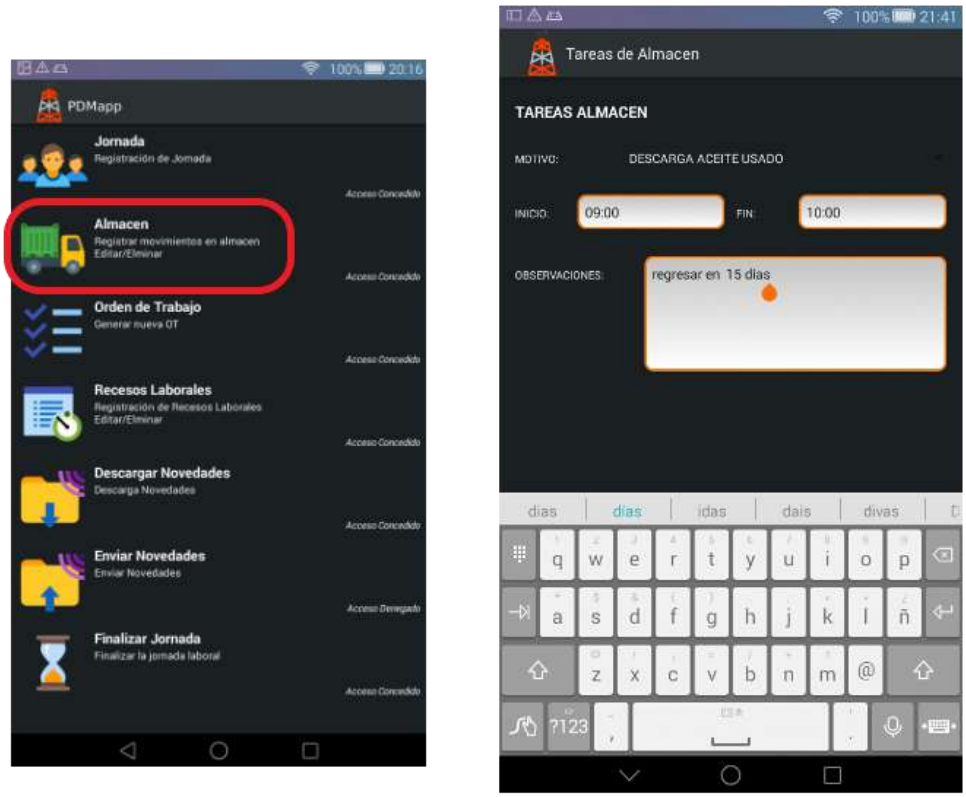


Imagen 19: Pantalla "Almacén": Descarga de aceite usado

Para realizar su labor en los motores y completar las órdenes de trabajo, los operarios se dirigen hacia los pozos que se le han asignado. Cada motor tiene una etiqueta con un código QR único (ver imagen 20) que la aplicación permite escanear y al procesarla presenta automáticamente la OT en pantalla. Se deben completar algunos datos más, como el tipo de orden, el pozo en el que se ubica, mediciones e imágenes pertinentes, los servicios brindados, materiales utilizados y el tiempo demandado.

Imagen 20: Etiqueta con código QR

El tipo de orden indica el tipo de trabajo que se le realiza al motor. Hoy en día solo hay cuatro, como se ve en la imagen 21:

descripcion
MANTENIMIENTO PREVENTIVO
MANTENIMIENTO CORRECTIVO
PUESTA EN MARCHA POR PRODUCCION
MANTENIMIENTO CORRECTIVO PTG

Imagen 21: Tipo de órdenes de trabajo

- **Mantenimiento correctivo:** reparación del motor por presentar falla, desgaste o averías. Necesita intervención para cumplir satisfactoriamente con su función en la producción.

- **Mantenimiento preventivo:** con el objetivo de reducir el riesgo de falla o caída del rendimiento del motor, hay servicios que se realizan cada un cierto periodo de tiempo planificado. El mantenimiento preventivo es el más frecuente.

- **Puesta en marcha por producción:** proceso para poner en producción un motor que estaba en depósito o para la explotación de un nuevo pozo. En cualquier caso requiere:
    1. Realizar al motor los servicios que necesite,
    2. Comprobar que funcione correctamente,
    3. Ponerlo activo en lugar y corroborar que en conjunto con el resto del equipamiento realicen sus funciones exitosamente.

  Es un proceso complejo que requiere de días de trabajo.

- **Mantenimiento correctivo PTG:** No es lo más habitual pero suelen realizar trabajos de mantenimiento en instrumentos de la planta de gas.

La carga de OT se visualiza en dos pantallas (imágenes 22 y 23). Su desarrollo se explica en profundidad en el capítulo 8.2.1. Cuando finalizan de cargar los datos, solo deben guardar la información.

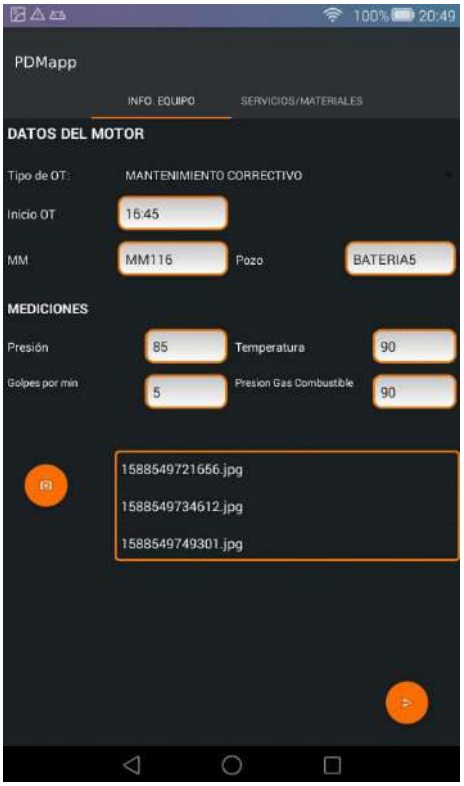


Imagen 22: Orden de trabajo (1).

Imagen 23: Orden de trabajo (2).

Los recesos laborales también se cargan en la aplicación, ya sea el caso de almorzar o interrumpir su trabajo por mal clima (imagen 24). También pueden editarse o eliminar una entrada siempre y cuando no hayan finalizado la jornada (imagen 25).

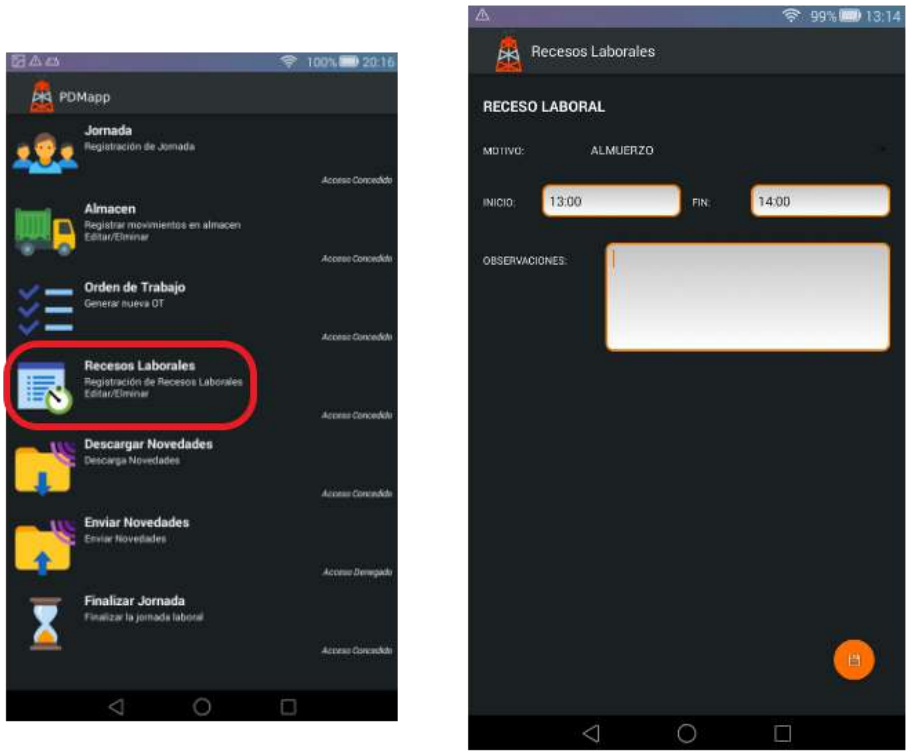


Imagen 24: Receso laboral

Imagen 25: Edición de receso laboral

Cuando se terminan todas las actividades del día se debe presionar en <<Finalizar jornada>> y verificar el horario (imagen 26).

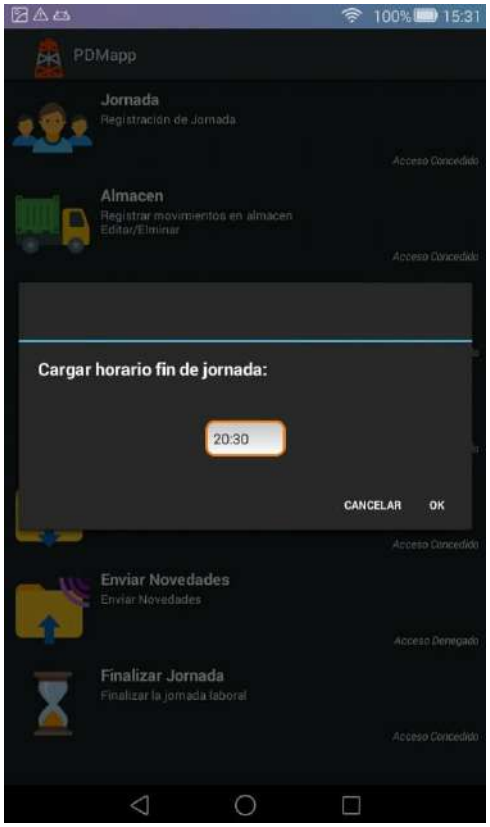


Imagen 26: Finalizar jornada laboral

Antes de retirarse, cada equipo entrega la *tablet* al supervisor, quien cuando cuente con una conexión a internet estable, se dirige a la sección <<Enviar Novedades>> de la aplicación. Realizada esta acción, se verá reflejada la información cargada por cada equipo en el sistema SGPDM.

7.3 Emitir reportes y estadísticas

Uno de los requerimientos peticionados desde el cliente, fue la posibilidad de obtener reportes que aporten información relevante para la toma de decisiones. Entre ellos se encuentran: horas trabajadas por operario, materiales utilizados, motores atendidos, tipos de órdenes de trabajo realizadas y parte diario de mantenimiento de motores de campo.

Horas trabajadas por operario

En este informe se muestra la cantidad de tiempo que cada empleado ha trabajado entre las distintas jornadas, distinguiendo sus horas ejercidas como Operario Oficial y como Operario Asistente. También se ofrece la posibilidad de observar en detalle en cuáles órdenes de trabajo se vio involucrado el empleado elegido. En el sistema se encuentran distintos filtros de búsqueda que permiten obtener un resultado más específico del reporte. Los filtros disponibles son: por operario, fecha de inicio y fecha de finalización. De este modo, se puede obtener la cantidad de horas que un empleado trabajó en un periodo determinado. Este reporte es muy útil a la parte administrativa de

la empresa para obtener el importe que debe abonarle a cada empleado por su cantidad de horas de trabajo.

Materiales utilizados

El reporte detalla la cantidad utilizada de un material en un motor en particular. Además, se ofrece la posibilidad de visualizar la cantidad total utilizada dentro de distintos parámetros elegidos. Entre los posibles filtros se encuentran nombre del material, equipo de trabajo, MM (Módulo Motor), fecha de inicio y fecha de finalización. Así, se puede obtener la cantidad utilizada de un material en un motor en particular. Con este informe es posible hacer un seguimiento de los materiales y llevar un control acerca de cuáles insumos se requieren en mayor medida y durante qué época.

Motores atendidos

Este reporte muestra el listado de motores a los que se le ha realizado mantenimiento, especificando la fecha y el equipo que lo hizo. Se puede realizar una búsqueda más exhaustiva mediante el MM del motor, el equipo que realizó el servicio, fecha de inicio y fecha de finalización. De esta manera, se puede buscar qué motores recibieron atención dentro de un rango de fechas. El informe le sirve a la parte administrativa para llevar un control de los mantenimientos realizados y conocer el estado de cada uno de sus motores.

Parte diario de mantenimiento de motores de campo

Este reporte permite visualizar una jornada completa realizada por un equipo. Está conformado por un encabezado en el que se detalla el equipo que realizó la jornada, especificando cuál operario estaba a cargo (oficial) y quienes eran los ayudantes, y el móvil con el cual se transportaron. Seguidamente se listan las tareas que se hicieron, detallando el tipo de orden de trabajo realizada, el pozo donde se llevó a cabo, el MM al que se le realizó el servicio, la hora de inicio y de finalización. Entre las tareas, también se encuentran las relativas al tiempo transcurrido con la entrega de AST y los recesos laborales, tales como el almuerzo. Sin embargo, estas últimas no entran en el total de horas contabilizado que se muestra al final del reporte. Este quizás es el informe de mayor importancia para la empresa YAL S.A., ya que tiene en detalle toda la información de las jornadas. Y es el documento que se presenta a la empresa contratante, para que ésta liquide el pago correspondiente.

Estadísticas

Además de los reportes mencionados anteriormente, existe un apartado en el sistema que permite observar dos gráficos: uno de barras presentando datos de materiales y uno de torta relacionado a las OT. La primera representación muestra los 10 materiales más utilizados y el consumo de cada uno. En tanto que el segundo, detalla la cantidad de órdenes de trabajo elaboradas y la proporción de cada tipo. Como se ha explicado en un apartado anterior, existen diferentes tipos de órdenes para detallar el trabajo realizado, por lo tanto la información precisa aquí es importante porque permite observar cuales tipo de OT fueron llevadas a cabo en mayor medida. En ambos gráficos, es posible segmentar los datos utilizando los parámetros de fecha de inicio y de finalización.

Capítulo 8. Desarrollo de SGPDM y PDMApp según iteraciones del proyecto

En el presente capítulo explicaremos en detalle los desarrollos llevados a cabo en cada iteración tanto para el sistema web como para la aplicación nativa, de acuerdo a las historias de usuario.

A continuación se enumeran las tareas que se implementaron en cada iteración y se explica detalladamente cuáles fueron los componentes, herramientas y tecnologías seleccionadas para el desarrollo de la funcionalidad y de las pruebas unitarias, los recursos que brindan, el motivo de la elección y los resultados obtenidos.

8.1 Etapa de codificación y pruebas de la Iteración 1

Tal como se mencionó en el capítulo 3, las tareas correspondientes a la iteración 1 son las siguientes:

- Creación del diagrama de entidad relación (DER) que permita visualizar un modelo de base de datos.
- Presentación de un prototipo de la interfaz tanto del sistema SGPDM como de la aplicación PDMApp
- Acceso al sistema SGPDM a través de autenticación de usuario
- Implementación de los distintos módulos del sistema SGPDM, con sus funcionalidades básicas de “Crear”, “Editar” y “Eliminar”, para que el cliente pueda cargar los datos pertinentes a usar En esta iteración, los módulos a desarrollar son: Aplicaciones, Bombas/AIB, Equipos, Máquinas, Materiales, Móviles, Operarios, Pozos, Servicios de Mantenimiento y Zonas

Las mismas, surgen de las historias de usuarios 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 y 11 (Ver Anexo 6).

8.1.1 Creación del diagrama de entidad relación (DER)

En lo que respecta al modelo de base de datos, en el capítulo 4 se mostró el DER creado con la herramienta MySQL Workbench. A partir de ello, se realizó “ingeniería directa” para crear las tablas en la base de datos MySQL, utilizando PhpMyAdmin[14] como gestor de administración de base datos. Finalmente, la misma quedó constituida por 25 tablas.

A continuación se procede a explicar cada una de las entidades del DER, las cuales terminaron siendo tablas en la base de datos.

Operarios: Contiene los datos correspondientes a cada uno de las personas que trabajan en la empresa. YAL S.A. no necesita contar con demasiada información detallada, por lo tanto solo se guarda el apellido, el nombre, el día en que comenzó el operario a trabajar en la empresa y un campo que determina si se encuentra discontinuado o no.

Equipos: Almacena todos los equipos que se conforman en la empresa. Los campos de esta tabla son: descripción, para guardar un nombre del equipo, operario0, operario1 y operario2 para

[14] PhpMyAdmin: es una herramienta de software gratuita escrita en PHP, destinada a manejar la administración de MySQL en la Web

los id de los operarios que lo conforman (siendo operario0 el operario oficial y operario1 y operario2 los ayudantes) y también el campo discontinuado.

Aplicaciones: Son modelos de motores existentes. Contienen una descripción y un atributo para determinar si se encuentra discontinuado o no.

Bombas AIB: Son las bombas que tiene la empresa instaladas en una perforación petrolera. Constan de una descripción y de un campo de discontinuado.

Máquinas: Son los motores que se encuentran en los pozos, a los cuales los operarios prestan sus servicios. Esta tabla contiene los atributos propios de MM (módulo de motor), número de motor y discontinuado para indicar si está activo o no. A su vez, posee las claves foráneas de las tablas con las que se relaciona: aplicaciones y bombasAIB.

Materiales: Son los insumos que utilizan los operarios para prestar mantenimiento a los motores. La tabla se compone de una descripción y un campo para determinar si se encuentra discontinuado o activo.

Móviles: Esta tabla contiene todos los utilitarios que usan los operarios para transportarse entre los distintos pozos en la jornada laboral. Contiene una descripción, un campo para determinar si se encuentra activo o discontinuado y otro para determinar la fecha de adquisición por parte de la empresa.

Recesos laborales: Se guardan los diferentes tipos de interrupciones que pueden existir en una jornada de trabajo a través de un campo de descripción.

Tareas de almacén: Contiene el listado de todas las acciones que pueden llevar a cabo los operarios en el almacén. Solo es necesario guardar una descripción de la misma.

Zonas: Son todos los lugares donde se encuentra los pozos petroleros. Se almacenan con el nombre de la zona y un campo para determinar si se encuentra discontinuada o no.

Pozos: Son todas las perforaciones de suelo con el objetivo de extraer petróleo. Se guardan con el nombre del pozo, las coordenadas de latitud y longitud para determinar su ubicación, el campo para establecer si se encuentra activo o discontinuado y la clave foránea resultante de su relación con la tabla zonas.

Servicios de Mantenimiento: La tabla contiene todas las acciones que pueden llevar a cabo los operarios al atender un motor. Los mismos se registran con una descripción y un atributo de discontinuado.

Compresiones: Contiene la numeración de los cilindros de un motor. Poseen una descripción y un campo de discontinuado.

Imágenes: En esta tabla se guardan todas las fotografías tomadas en los pozos. Poseen un atributo propio que es la url que determina el nombre de la imagen y un campo resultante de su relación con la tabla órdenes de trabajo.

Tipos de Ordenes de Trabajo: Contiene el listado de los tipos de órdenes de trabajo que se pueden aplicar.

Jornadas: Contiene toda la información pertinente al accionar laboral en el día por parte de un equipo. Los campos propios de la tabla son fecha, hora de inicio, hora de finalización, hora de inicio de la AST y hora de finalización de la AST. Y, dado que se relaciona con la tabla equipos (el equipo que participa en esa jornada) y con la tabla móviles (el móvil que usa el equipo para trasladarse durante la jornada), posee las claves foráneas que hacen referencia a dichas tablas.

Órdenes de Trabajo: Esta tabla representa la planilla en papel que debían completar los operarios cuando realizaban cada tarea de mantenimiento. Es por ello que guardan los datos propios de la orden: número de orden, tipo de orden, latitud, longitud, hora de inicio de la OT, hora de finalización de la OT, observaciones, presión, temperatura, golpes por minuto y presión de gas.

También posee las claves foráneas resultantes de la relación con las tablas jornadas, máquinas y pozos.

8.1.2 Presentación de un prototipo de interfaz tanto del sistema SGPDM como de la aplicación PDMApp

Para los prototipos de la *app* y del sistema se hicieron bosquejos en papel y luego se trasladaron a código. En el caso del sistema web, se utilizó Bootstrap[15] para el maquetado. La amplia cantidad de componentes y recursos que brinda esta herramienta, agilizó mucho el proceso de diseño. Por ejemplo, la pantalla de *login* se visualiza en la imagen 27.

Imagen 27: Pantalla login

Para la visualización de las funcionalidades de la *app* se optó por crear un *Adapter*[16] de un *RecyclerView*[17]. Los *RecyclerViews* son listas más personalizables, potentes y eficientes que las antiguas *ListView*[18]. Para esto se debió implementar la dependencia *Recyclerview* que por defecto no lo tiene. Así que en las *dependencies{}* se añadió la línea que se muestra en la imagen 28.

```
dependencies {
    implementation 'com.android.support:recyclerview-v7:26.1.0'
```

Imagen 28: Librería recyclerview en el archivo build.gradle

El adaptador es simple y permite la modificación de ítemes. Esto quiere decir que en caso de que en un futuro se requieran de más funcionalidad en la estructura principal solo implicaría agregar un nuevo *adapter* y asignarlo al *RecyclerView* de manera muy sencilla. Cada ítem de la lista de funcionalidades está compuesto por los siguientes componentes:

- **ivFuncion**: Es un componente *ImageView*. Es el logo de la función

[15] Bootstrap: Es un kit de herramientas de código abierto para desarrollar con HTML, CSS y JS.(https://getbootstrap.com)

[16] Un objeto Adapter actúa como un puente entre un AdapterView y los datos subyacentes para esa vista. El adaptador proporciona acceso a los elementos de datos. El adaptador también es responsable de crear una vista para cada elemento del conjunto de datos.

[17] RecyclerView es un ViewGroup que representa cualquier vista basada en adaptador de manera similar. Permite mostrar una lista de desplazamiento de elementos sobre la base de grandes conjuntos de datos (o datos que cambian con frecuencia).

[18] Una vista ListView visualiza una lista deslizable verticalmente de varios elementos, donde cada elemento puede definirse como un Layout.

- **tvNombreFuncion**: Un *TextView* indicando el nombre de la función
- **tvDescripcion**: *TextView* aportando una breve explicación de lo que se puede realizar en esa actividad
- **tvAcceso**: *TextView* indicando el estado de acceso a esa funcionalidad. Este estado varía entre acceso denegado y acceso concedido durante la interacción con la *app*

Finalmente, la imagen que se observa al inicio de la aplicación PDMApp se muestra en la imagen 29.

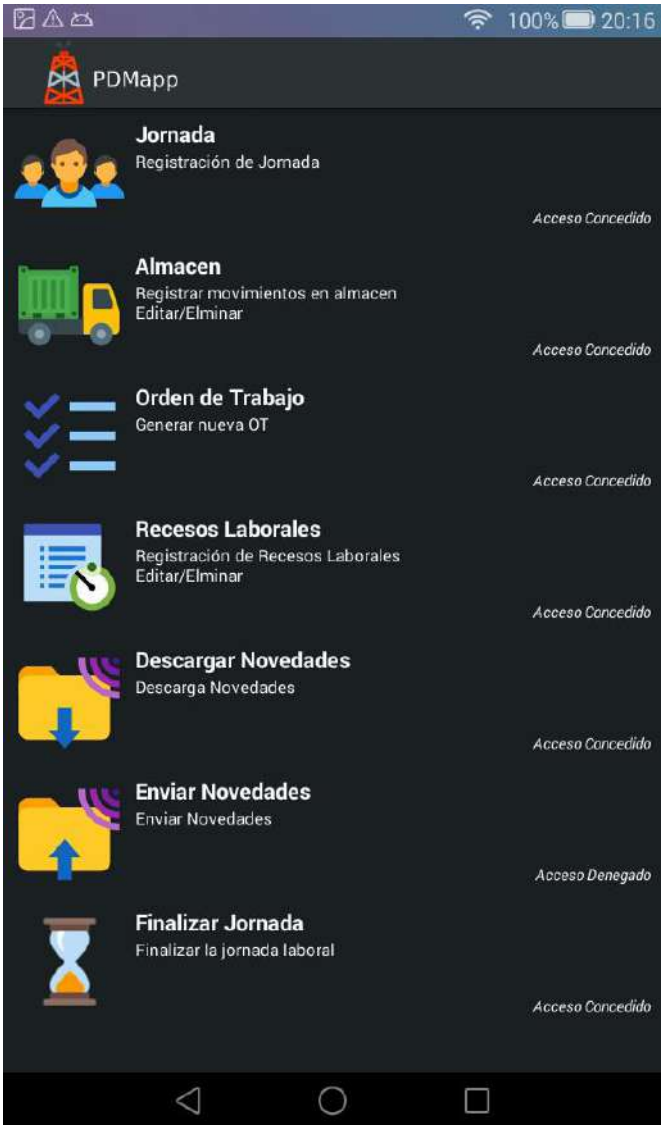


Imagen 29: Pantalla principal de la app

8.1.3 Acceso a SGPDM a través de autenticación de usuarios

En el sistema web, a través de la creación de perfiles, se puede controlar el acceso de usuarios y restringir su ingreso cuando no cuenten con los permisos necesarios. Particularmente, se cuentan con dos tipos de perfiles: Usuario Supervisor y Usuario Administrador. El primero es el que tiene los permisos para crear, modificar y eliminar diferentes aspectos de los módulos del sistema. En tanto que el segundo, solamente puede generar y visualizar reportes e informes estadísticos.

Laravel hace que la implementación de la autenticación sea muy simple. De hecho, prácticamente todo viene configurado de serie. Solamente, se debe ejecutar en la terminal el comando:

php artisan make:auth

Una vez ejecutado el comando, se generan varias vistas en la ruta **resources/views/auth**. Además existe un archivo "layout" que contiene la plantilla general de la aplicación en la ruta **resources/views/layouts**. También se incluyen varios controladores pre-desarrollados que se pueden encontrar en el *namespace* **App\Http\Controllers\Auth**. El controlador *RegisterController* gestiona el registro de nuevos usuarios, *LoginController* gestiona la autenticación, *ForgotPasswordController* se encarga del envío de e-mails para el restablecimiento de contraseñas y *ResetPasswordController* contiene la lógica para restablecer las contraseñas. Cada uno de estos controladores utiliza un *trait* para incluir los métodos necesarios. Para muchas aplicaciones no será necesaria la modificación de estos controladores. [13]

Para la protección de rutas, Laravel utiliza un middleware. Este se puede utilizar para permitir el acceso a ciertas rutas únicamente a usuarios autenticados. Laravel incluye un middleware auth, definido en **Illuminate\Auth\Middleware\Authenticate** y todo lo que hay que hacer es asignarlo a la definición de una ruta. Por supuesto, si se están utilizando controladores, se puede llamar al método middleware desde el constructor del controlador en lugar de asignarlo en la definición de la ruta directamente:

```
public function __construct() {
    $this->middleware('auth');
}
```

Finalmente, para determinar si el usuario ha iniciado sesión en la aplicación, se puede utilizar el método *check()*, el cual retornará *true* si el usuario está autenticado. Luego, de acuerdo al perfil que posea el usuario, determina el acceso a un conjunto de funcionalidades u otras. Cuando ingresa un usuario supervisor, la pantalla que se muestra es la que se observa en la imagen 30. En tanto, que si se trata de un usuario administrador, se verá como en la imagen 31.

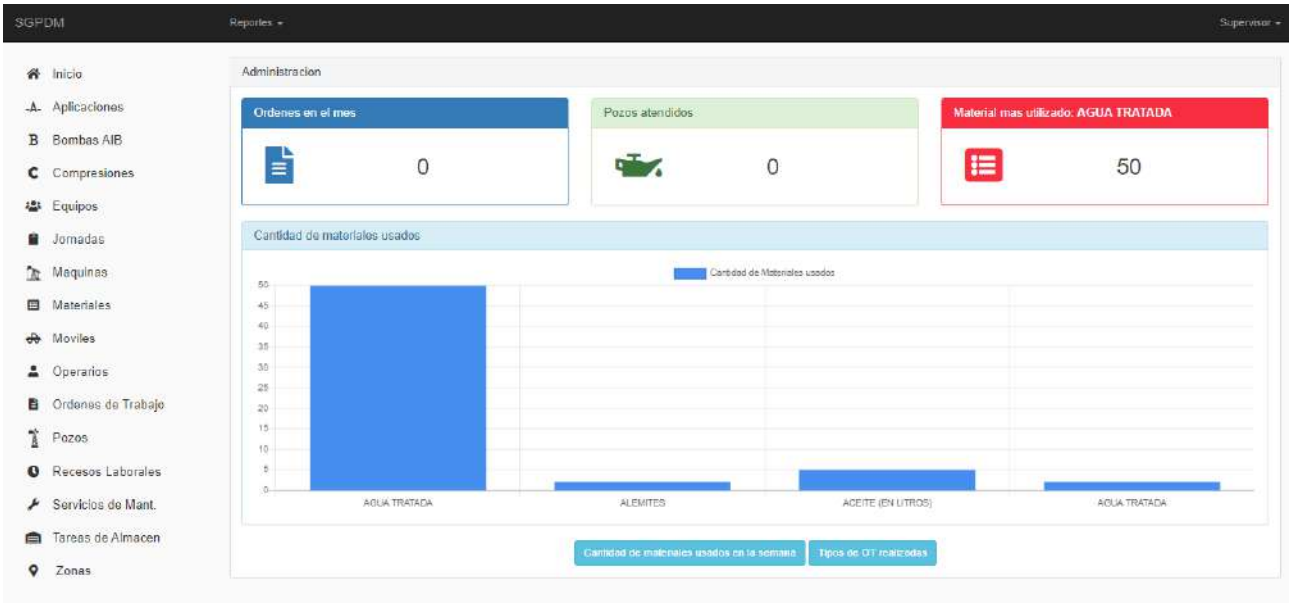


Imagen 30: Pantalla inicio para usuario supervisor

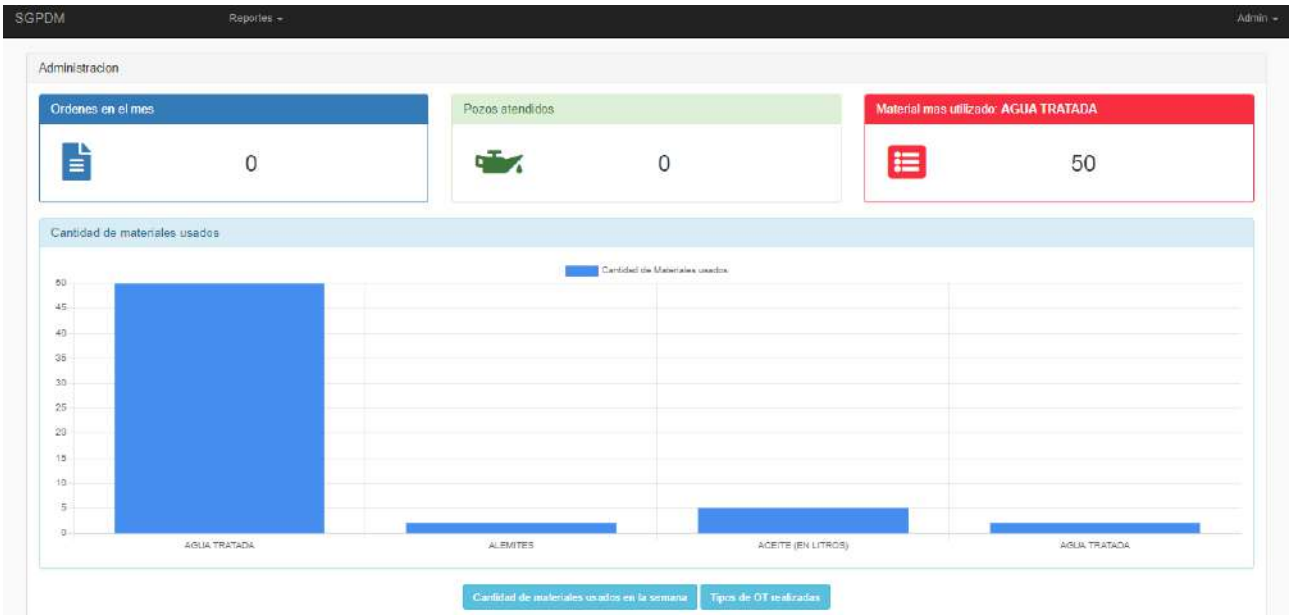


Imagen 31: Pantalla inicio usuario administrador

8.1.4 Implementación de módulos del sistema SGPDM

En lo que respecta a la implementación de los módulos, tomaremos uno como modelo y lo usaremos para desarrollar las acciones de Crear/Editar/Eliminar.

Por ejemplo, al acceder al módulo Máquinas se observa la pantalla que muestra la imagen 32. La misma muestra el listado de máquinas activas en la empresa, con las funcionalidades de visualizar, editar, dar de baja o eliminar. En la parte superior derecha, se ubica el botón que permite la creación de una nueva máquina. Y, además, se encuentra un buscador para listar máquinas que cumplan con el MM y/o número de motor especificado.

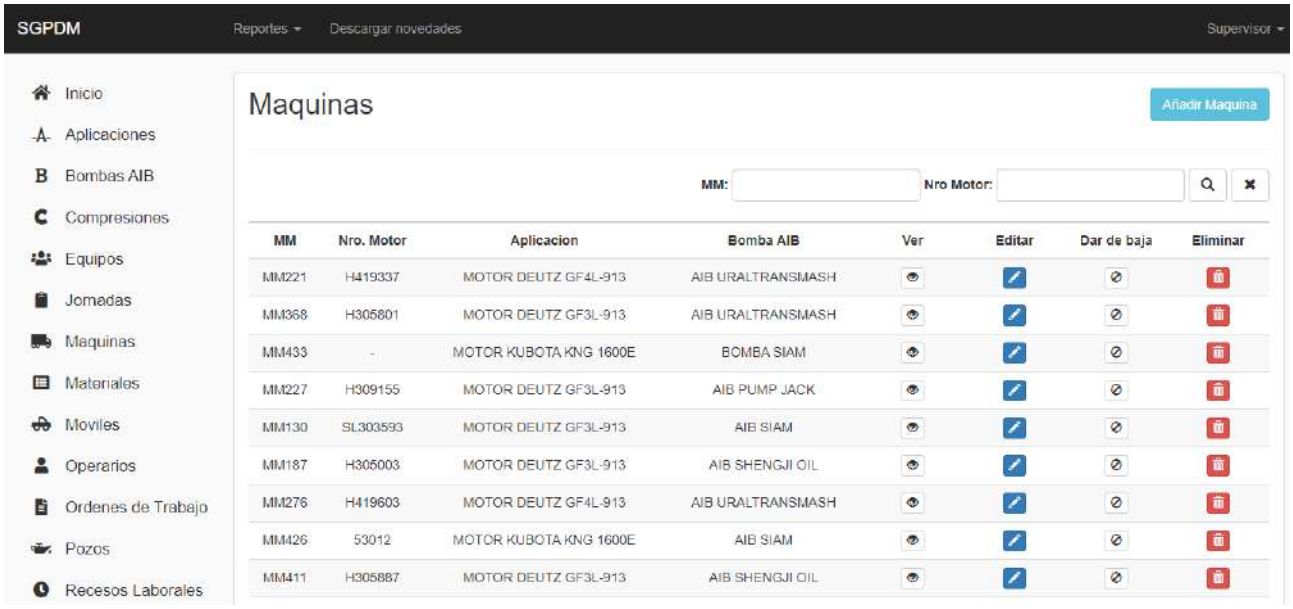


Imagen 32: Pantalla Maquinas

Laravel, con su patrón MVC (ver Anexo 4), permite realizar esta acción de manera muy simple. Se determina, en primer lugar, que en el momento en que el usuario acceda a la ruta **maquinas/index** sea el controlador *MaquinaController*, con su método *index()*, quien se encargue de resolver la petición. En la imagen 33 se observa el código del mismo.

```
public function index(Request $request){
    $mm = $request->mm;
    $motor = $request->nro_motor;
    $maquinas = Maquina::where('discontinuado','=',0)
                        ->mm($mm)
                        ->motor($motor)
                        ->orderBy('id_maquina','DESC')
                        ->paginate();
    return view('maquinas.index',compact('maquinas'))->with('mm',$mm)->with('nro_motor',$motor);
}
```

Imagen 33: Método index de MaquinaController

El método recibe como parámetro un **$request** con lo que haya introducido el usuario en el cuadro de búsqueda. En el primer acceso dicha variable estará vacía. Por ello, las variables **$mm** y **$motor**, también lo estarán. Seguidamente, se hace uso del modelo Maquina para consultar a la base de datos. Finalmente, se retorna a la vista “index” enviando las variables **$maquinas** (a través de la función *compact*), **$mm** y **$motor** (estas dos últimas con *with*).

Y luego, en la vista, se muestran los datos como se observa en la imagen 34.

```
<tbody>
    @foreach($maquinas as $maquina)
    <tr>
        <td class="text-center">{{$maquina->mm}}</td>
        <td class="text-center">{{$maquina->nro_motor}}</td>
        <td class="text-center">{{$maquina->aplicacion->descripcion}}</td>
        <td class="text-center">{{$maquina->bombaaib->descripcion}}</td>
        <td class="text-center"><a class="btn btn-default btn-xs" href="{{route('maquinas.show',$maquina->id_maquina)}}"><span class="glyphicon
        <td class="text-center"><a class="btn btn-primary btn-xs" href="{{route('maquinas.edit',$maquina->id_maquina)}}" ><span class="glyphicon
        <td class="text-center"><a class="btn btn-default btn-xs" href="{{route('maquinas.dar-baja',$maquina->id_maquina)}}"><span class="glyph
        <td class="text-center">
          <form action="{{ route('maquinas.destroy',$maquina->id_maquina)}}" method="post">
           {{csrf_field()}}
           <input name="_method" type="hidden" value="DELETE">

           <button class="btn btn-danger btn-xs" type="submit"><span class="glyphicon glyphicon-trash"></span></button>
            </form>
        </td>
    </tr>
    @endforeach
</tbody>
```

Imagen 34: Vista maquinas.index

Al pulsar el botón <<Añadir Máquina>>, se hace una petición a la ruta **maquinas/create**. Nuevamente, es el controlador quien maneja el acceso, pero esta vez mediante su método *create* (imagen 35).

```
public function create(){
    $aplicaciones = Aplicacion::where('discontinuado','=',0)
                              ->orderBy('descripcion','ASC')
                              ->pluck('descripcion','id_aplicacion');
    $bombas = BombaAIB::where('discontinuado','=',0)
                      ->orderBy('descripcion','ASC')
                      ->pluck('descripcion','id_bombaAIB');
    return view('maquinas.create',compact('aplicaciones'),compact('bombas'));
}
```

Imagen 35: Método create de MaquinaController

En este caso, se consulta a la base de datos las aplicaciones y las bombas que se encuentran activas (no discontinuadas) y se guardan los datos en sus respectivas variables. Luego, se retorna la vista “create”, enviando las variables correspondientes. La pantalla que se observa, es la que aparece en la imagen 36.

Imagen 36: Pantalla de creación de una máquina

El motivo por el que se busca las aplicaciones y las bombas, es para poner valores en los *selects* correspondientes. Cuando se completan todos los datos y se presiona el botón <<Enviar>>, otra

vez es el controlador quien se encarga de guardar los datos, a través de su método *store()* que se detalla en la imagen 37.

```php
public function store(MaquinaRequest $request){
    $maquina = new Maquina;
    $maquina->mm = strtoupper($request->mm);
    $maquina->nro_motor = strtoupper($request->nro_motor);
    $maquina->id_aplicacion = $request->id_aplicacion;
    $maquina->id_bombaAIB = $request->id_bombaAIB;
    $maquina->save();
    return redirect()->route('maquinas.index')->with('info','La maquina fue guardada');
}
```

Imagen 37: Método store de MaquinaController

Dentro del método, se instancia el modelo Maquina, y se procede a cargar sus atributos con lo que se obtiene de la variable **$request** (lo ingresado por el usuario en el formulario de *create()*). Seguidamente se guarda, y se redirecciona a la vista “index”, con el mensaje correspondiente.

En lo que respecta a la edición, es el método *edit()* quien devuelve la vista donde se pueden modificar los datos (imagen 38).

```php
public function edit($id){
    $maquina = Maquina::find($id);
    $aplicaciones = Aplicacion::where('discontinuado','=',0)
                        ->orderBy('descripcion','ASC')
                        ->pluck('descripcion','id_aplicacion');
    $bombas = BombaAIB::where('discontinuado','=',0)
                        ->orderBy('descripcion','ASC')
                        ->pluck('descripcion','id_bombaAIB');
    return view('maquinas.edit',compact('maquina'),compact('aplicaciones'))->with('bombas',$bombas);
}
```

Imagen 38: Método edit de MaquinaController

Como se puede observar, el método es muy parecido a *create()*, con la diferencia de que aquí en primer lugar se busca la máquina de acuerdo al **$id** solicitado. Luego, se devuelve la vista “edit” con las variables correspondientes. El resultado se observa en la imagen 39.

Imagen 39: Pantalla de edición de una máquina

Cuando se presiona el botón <<Enviar>>, se ejecuta el método *update()* (imagen 40).

```php
public function update(MaquinaRequest $request, $id){
    $maquina = Maquina::find($id);
    $maquina->mm = strtoupper($request->mm);
    $maquina->nro_motor = strtoupper($request->nro_motor);
    $maquina->id_aplicacion = $request->id_aplicacion;
    $maquina->id_bombaAIB = $request->id_bombaAIB;
    $maquina->save();
    return redirect()->route('maquinas.index')->with('info','La maquina fue actualizada');
}
```

Imagen 40: Método update de MaquinaController

El método es muy parecido al *create()*, nuevamente con la diferencia de que en primer lugar se busca la maquina por su **$id** y seguidamente se modifica sus atributos por los ingresados por el usuario. Por último, se retorna a la vista "index", con el mensaje correspondiente.

Cuando se desea visualizar una máquina en particular, el método *show()* es el que maneja la solicitud. El código del mismo, se muestra en la imagen 41.

```php
public function show($id){
    $maquina = Maquina::find($id);
    return view('maquinas.show',compact('maquina'));
}
```

Imagen 41: Método show de MaquinaController

Se trata de un método sencillo, cuya funcionalidad es buscar la máquina a través de su **$id** y enviar los datos a la vista correspondiente. La pantalla resultante que se obtiene es la que muestra la imagen 42.

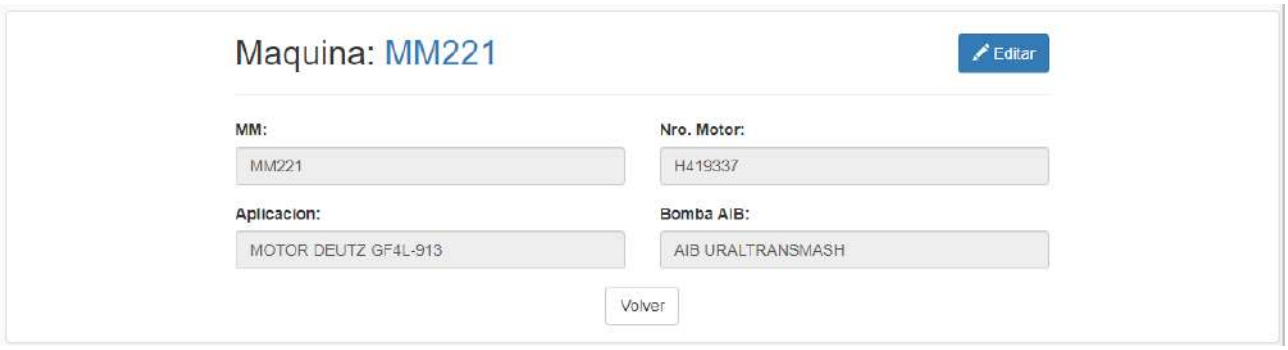


Imagen 42: Pantalla para mostrar una máquina en particular

8.1.5 Testing de los módulos desarrollados

Para llevar a cabo los *tests*, se replicó la base de datos del proyecto. De esta forma, se pueden ver los resultados sin afectar la base de datos original. Las primeras pruebas que se llevaron a cabo fueron las referidas a la autenticación de usuarios. Los casos a probar fueron los siguientes:

- Mostrar formulario de *Login* al acceder a dicha página.
- Hacer una autenticación correcta y que se redireccione a la página correspondiente.
- Hacer una autenticación incorrecta y que se redireccione nuevamente a la página de *Login*, mostrando los errores correspondientes.

- Registrar un usuario y que se redireccione a la página correspondiente.

Para el segundo caso de prueba, el código es el que se muestra en la imagen 43. El *test* está dividido en dos partes: en la primera se hace una llamada a la ruta **/login** con los datos de un usuario existente en la base de datos y la respuesta se guarda en la variable **$response**. En segundo lugar, el método *assertRedirect()* indica que lo que se espera de la respuesta es una redirección hacia la acción que ejecuta el método *inicio()* del controlador *OrdenController*. Este método se ejecuta cuando la autenticación es exitosa.

```php
/** @test */
public function login_correcto_y_redireccion(){
    //Inicio de sesion exitoso con un usuario creado
    $response = $this->post(route('login'), [
        'email' => 'admin@correo.com',
        'password' => 'admin1234',
    ]);
    $response->assertRedirect(action('OrdenController@inicio'));
}
```

Imagen 43: Test de Login correcto

Y la ejecución y resultado del caso de prueba, se puede observar en la imagen 44:

```
C:\wamp\www\SGPDM>vendor\bin\phpunit --filter login_correcto_y_redireccion
PHPUnit 5.7.27 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 655 ms, Memory: 13.75MB

OK (1 test, 2 assertions)

C:\wamp\www\SGPDM>
```

Imagen 44: Test de Login correcto (Resultado)

Como muestra la imagen, el resultado del *test* es OK. Es decir, cuando se ingresaron los datos, se redireccionó hacia la página de inicio. Por lo tanto, el *login* fue exitoso.

Posteriormente se realizaron las pruebas de cada módulo que contiene el sistema. Se tomará como ejemplo, el módulo de Equipos. Los *tests* que se aplicaron fueron los siguientes:

- Imposibilidad de crear un equipo al no ser usuario
- Crear un equipo siendo usuario
- Mostrar el listado de equipos siendo usuario
- Mostrar el listado de equipos sin ser usuario
- Mostrar datos de un equipo en particular
- Editar datos de un equipo en particular
- Eliminar un equipo

En primer lugar, se mostrará el caso en el que se quiera crear un equipo sin ser usuario. Para ello, se crea el método que se muestra en la imagen 45.

```php
/** @test */
public function crear_equipo_sin_ser_usuario(){
    $data = [
            'descripcion' => "Nuevo Equipo",
            'operario0' => 1,
            'operario1' => 2,
            'operario2' => 3,
            ];

    $response = $this->json('POST', '/equipos/create',$data);
    $response->assertStatus(405);
}
```

Imagen 45: Test crear_equipo_sin_ser_usuario

Se comienza cargando la variable **$data** con los valores que se desea que tenga el equipo. En este caso, que se cree un equipo de nombre “Nuevo Equipo” que lo conformen los operarios con id 1, 2 y 3. Realizamos una inserción mediante el método POST en la ruta **equipos/create** con la variable **$data**. Y la respuesta que se espera es un código de estado 405, es decir, que no está permitido. Se procede a ejecutar el *test* y se obtiene lo que muestra la imagen 46.

```
C:\wamp\www\SGPDM>vendor\bin\phpunit --filter crear_equipo_sin_ser_usuario
PHPUnit 5.7.27 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 413 ms, Memory: 9.75MB

OK (1 test, 1 assertion)
```

Imagen 46: Resultado del test crear_equipo_sin_ser_usuario

Como se observa, el resultado es satisfactorio. Es decir, se llega a un código de estado 405, por lo tanto no es posible crear un equipo sin estar autenticado.

Ahora se mostrará el mismo caso, pero con un usuario *logueado*. El código del *test* que se utiliza, es el de la imagen 47.

```php
/** @test */
public function crear_equipo_siendo_usuario(){
    $data = [
            'descripcion' => "Nuevo Equipo",
            'operario0' => 1,
            'operario1' => 2,
            'operario2' => 3,
            ];
    $user = factory(User::class)->create();

    $response = $this->actingAs($user)->json('POST', '/equipos/create',$data);
    $response = $this->json('POST', '/equipos',$data);
    $response->assertStatus(302); //Cuando se carga un equipo, se redirecciona a la pagina equipos.index
    $response->assertRedirect('/equipos');

}
```

Imagen 47: Test crear_equipo_siendo_usuario

Como se puede apreciar, la diferencia con respecto al *test* anterior, radica en que se genera un usuario a través del método *factory()* que provee Laravel. Luego, se le dice a Laravel que, actuando como el usuario previamente creado, realice una inserción en la ruta **equipos/create** de los datos cargados en **$data**. Y la respuesta que se espera en este caso, es una redirección (código de estado 302) hacia la ruta **equipos/index**. Se procede a ejecutar el *test* y se obtiene lo expone la imagen 48.

```
C:\wamp\www\SGPDM>vendor\bin\phpunit --filter crear_equipo_siendo_usuario
PHPUnit 5.7.27 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 717 ms, Memory: 14.50MB

OK (1 test, 3 assertions)

C:\wamp\www\SGPDM>
```

Imagen 48: Resultado del test crear_equipo_siendo_usuario

El resultado del *test* es satisfactorio, es decir, se crea el equipo y se redirige a la página de inicio. Si se dirige a la base de datos para las pruebas, se puede apreciar que se encuentra el equipo cargado mediante el *test*.

8.2 Etapa de codificación y pruebas de la iteración 2

Las tareas correspondientes a la iteración 2 son las siguientes:

- Implementación de la carga de Órdenes de Trabajo (OT) desde la aplicación PDMApp
- Creación de las funcionalidades para listar y visualizar OT, de acuerdo a determinados criterios simples de búsqueda (ejemplo: por fecha o equipo) en el sistema SGPDM

Estas surgen de las historias de usuario 12 y 13 (Ver Anexo 6).

8.2.1 Implementación de la carga de Ordenes de Trabajo (OT) desde la aplicación PDMApp

Una orden de trabajo contiene toda la información del mantenimiento que se le efectúo a un motor en particular, completar una OT requiere ingresar muchos datos. La planilla en papel era extensa y compleja (Ver Anexo 8), por eso para evitar una pantalla que requiriera de desplazamiento

vertical (*scroll*) se optó por implementar un *ViewPager* con un *FragmentPagerAdapter*. El *ViewPager* permite al usuario visualizar toda la planilla en varias páginas o *Tabs* con la posibilidad de deslizarse hacia los lados y cambiar de página (imagen 49). [16]

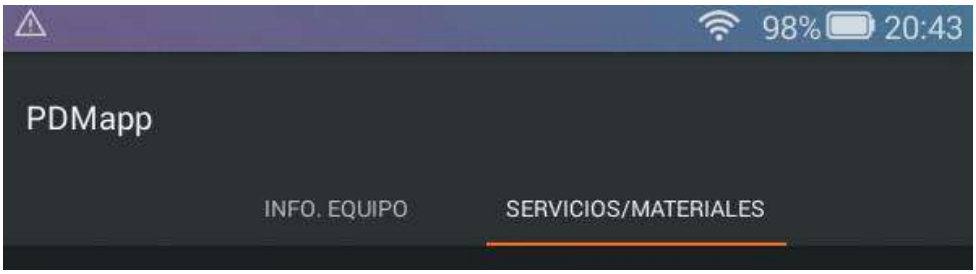


Imagen 49: ViewPager con FragmentPagerAdapter de dos páginas: Info. Equipo y Servicios/Materiales

Para esto se necesitó definir un adaptador, *FragmentPagerAdapter*, que determinara correctamente la cantidad de páginas y qué fragmento mostrar para cada página del adaptador [16]. Finalmente, se asoció el *ViewPager* con una nueva instancia de nuestro adaptador. Cada página es un *fragment* diferente, aquí solo se tienen dos, pero en caso de necesitarse más se pueden crear nuevas instancias de la clase *Fragment* en la actividad principal y se obtendría una página nueva para mostrar al usuario.

En el primer *fragment* Ot1 (imagen 22), los operarios cargan datos de medición del motor (presión, golpes por minuto, presión gas del combustible, etc.), como también el tipo de orden de trabajo. En el *fragment* Ot2 (imagen 23), se cargan las mediciones de los cilindros del motor, como también cuales fueron los servicios que se le realizaron, los materiales y cantidades que se necesitaron para completar la tarea.

Para listar los servicios se creó un *Multichoice AlertDialog*[19], este componente nos permite ingresar el listado de servicios en forma de *Array* y seleccionar varias opciones, que es exactamente lo que se precisa para elegir todas las acciones que se realizaron al motor (imagen 50). Para listar los materiales también se implementó un *AlertDialog,* pero este requirió de un componente contenedor más complejo para mantener el *Array* de materiales y sus cantidades. Se resolvió con un *layout ScrollView*[20] que permite deslizar hacia abajo y arriba cuando la pantalla no permite visualizar el listado completo y un *LinearLayout* [21] para mantener, como se nombró anteriormente, el *Array* de materiales y el *Array* de cantidades (imagen 51).

[19] Ventana pequeña que le indica al usuario que debe tomar una decisión o ingresar información adicional. Puede mostrar uno, dos o tres botones.

[20] Un grupo de vista que permite desplazar verticalmente la jerarquía de vista ubicada dentro de él.

[21] Grupo de vista que alinea todos los componentes secundarios en una única dirección, de manera horizontal o vertical.

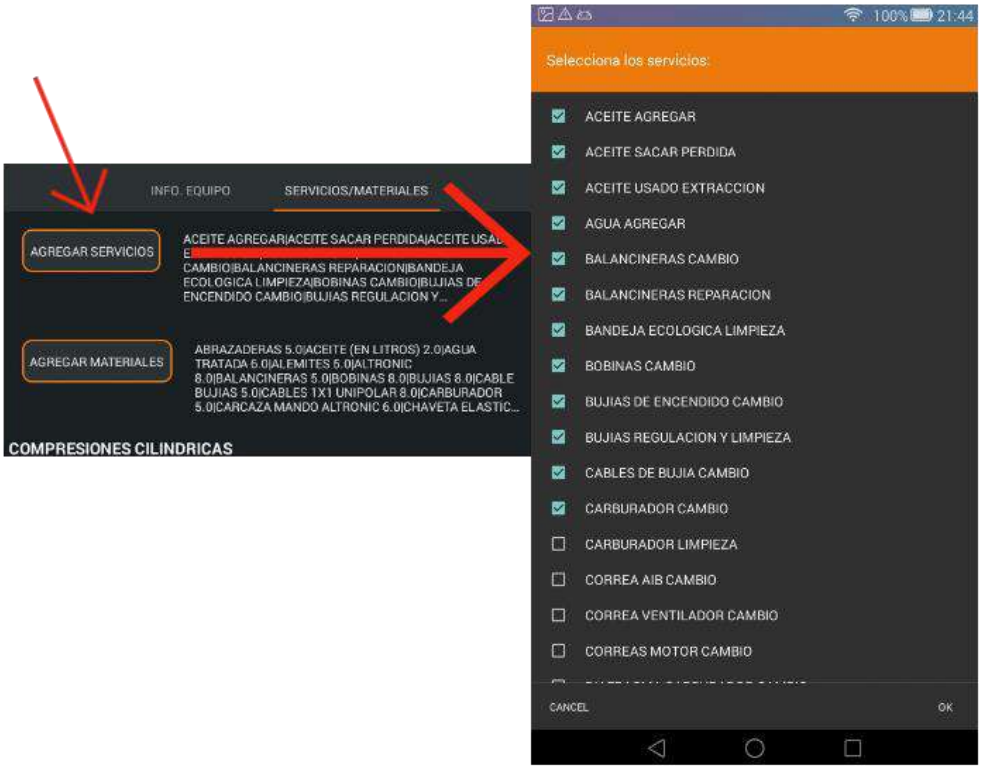


Imagen 50: Multichoice AlertDialog con listado de servicios.

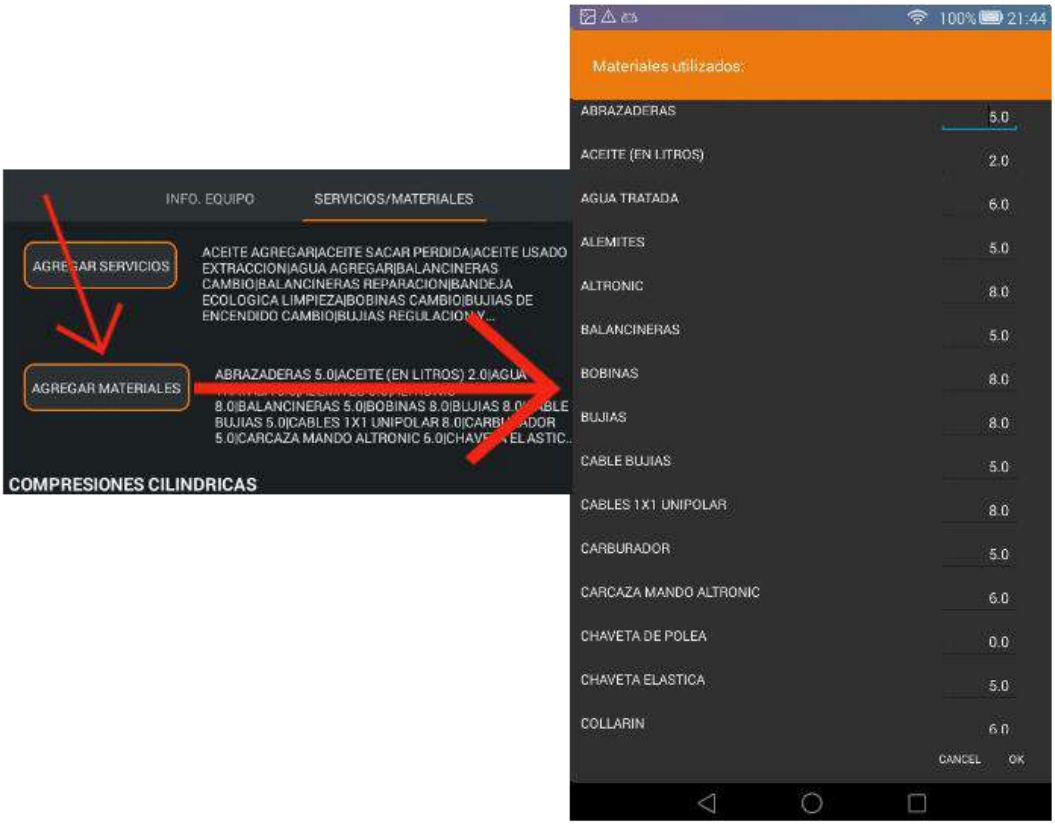


Imagen 51: AlertDialog con listado de materiales.

8.2.2 Listar y visualizar Órdenes de Trabajo (OT) de acuerdo a determinados criterios en SGPDM

Cuando se accede al módulo de órdenes de trabajo se realiza una petición al controlador *OrdenController*, el cual a través de su método *index()* (imagen 52) devuelve el conjunto de órdenes en el sistema.

```
public function index(Request $request){
    $fecha_inicio = $request->get('fecha_inicio');
    $fecha_fin = $request->get('fecha_fin');
    $equipo = $request->get('equipo');
    $mm = $request->get('mm');

    $ordenes = Orden::join('jornadas','jornadas.id_jornada','=','ordenestrabajo.id_jornada')
                        ->mm($mm)
                        ->inicio($fecha_inicio)
                        ->fin($fecha_fin)
                        ->equipo($equipo)
                        ->orderBy('id_ordenTrabajo','DESC')
                        ->paginate();

    $arr_equipos = Equipo::orderBy('descripcion','ASC')->get()->pluck('descripcion','id_equipo')->toArray();

    return view('ordenes.index',compact('ordenes'))->with('arr_equipos',$arr_equipos);
```

Imagen 52: Método index de Orden

En el código se puede apreciar que en primer lugar se obtienen los datos de la variable **$request**. En caso que el usuario haya realizado una búsqueda, cada variable se cargará con el valor correspondiente. Seguidamente, se procede a realizar la búsqueda en la base de datos a través del modelo Orden, pero como también se necesitan datos referidos a la jornada en la que se realizaron dichas órdenes se realiza un *join* con la tabla jornadas mediante la columna que comparten. Luego se ejecutan los métodos inicio, fin y equipo que sirven para filtrar los datos insertados en el buscador. También, como se ha mencionado en la iteración anterior, se buscan los datos referidos a los equipos para poder mostrarlos en el *select* del buscador. Una vez obtenidas las ordenes, se devuelven a la vista correspondiente.

Es pertinente mencionar, que los métodos inicio, fin y equipo se tratan de una funcionalidad que ofrece Laravel que se denomina *scopes*. Los *scopes* permiten definir un conjunto de restricciones que se pueden reutilizar a lo largo de la aplicación. Estos se definen dentro de la clase que los va a utilizar. En el caso de este proyecto, los scopes utilizados se muestran en la imagen 53.

```php
//Scopes

public function scopeMM($query,$mm){
    if ($mm) {
        $query->where('maquinas.mm', 'LIKE', '%'.$mm.'%')
              ->join('maquinas', 'ordenestrabajo.id_maquina', '=', 'maquinas.id_maquina');
    }
    return $query;
}

public function scopeInicio($query,$fecha_inicio){
    if($fecha_inicio){
        $query->where('jornadas.fecha','>=',$fecha_inicio);
    }
    return $query;
}
public function scopeFin($query,$fecha_fin){
    if($fecha_fin){
        $query->where('jornadas.fecha','<=',$fecha_fin);
    }
    return $query;
}
public function scopeEquipo($query,$equipo){
    if($equipo){
        $query->where('jornadas.id_equipo','=',$equipo);
    }
    return $query;
}
```

Imagen 53: Scopes en el modelo Orden

Por definición, deben denominarse con el prefijo *scope* y luego el nombre que se le desee colocar. Posteriormente, al momento de utilizarlos, se omite el prefijo y se utiliza el nombre en minúscula. Como se puede apreciar, resulta muy útil a la hora de realizar consultas anidadas, ya que en el caso que se coloque el dato en el buscador se ejecutará el *scope* correspondiente. Caso contrario, no ingresa a ejecutar la consulta. Cuando se retorna a la vista, se observa la pantalla de la imagen 54.

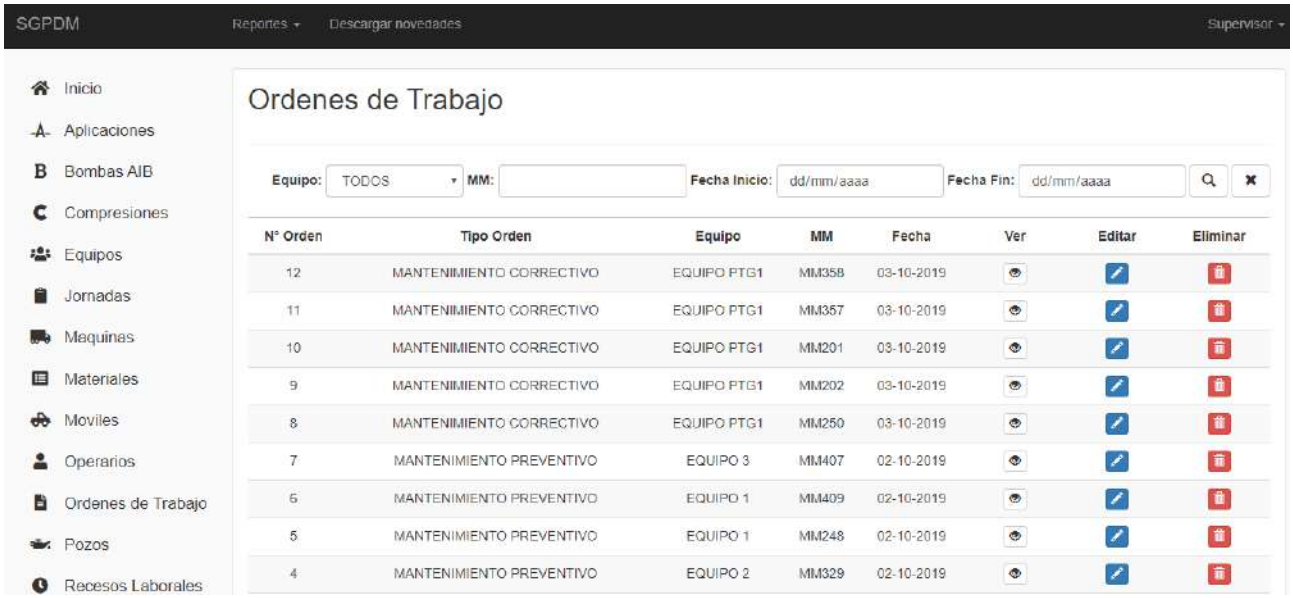


Imagen 54: Pantalla Ordenes

Al seleccionar la opción Ver se muestran todos los datos correspondientes a la orden de trabajo elegida. Este resultado es el que muestra la imagen 55.

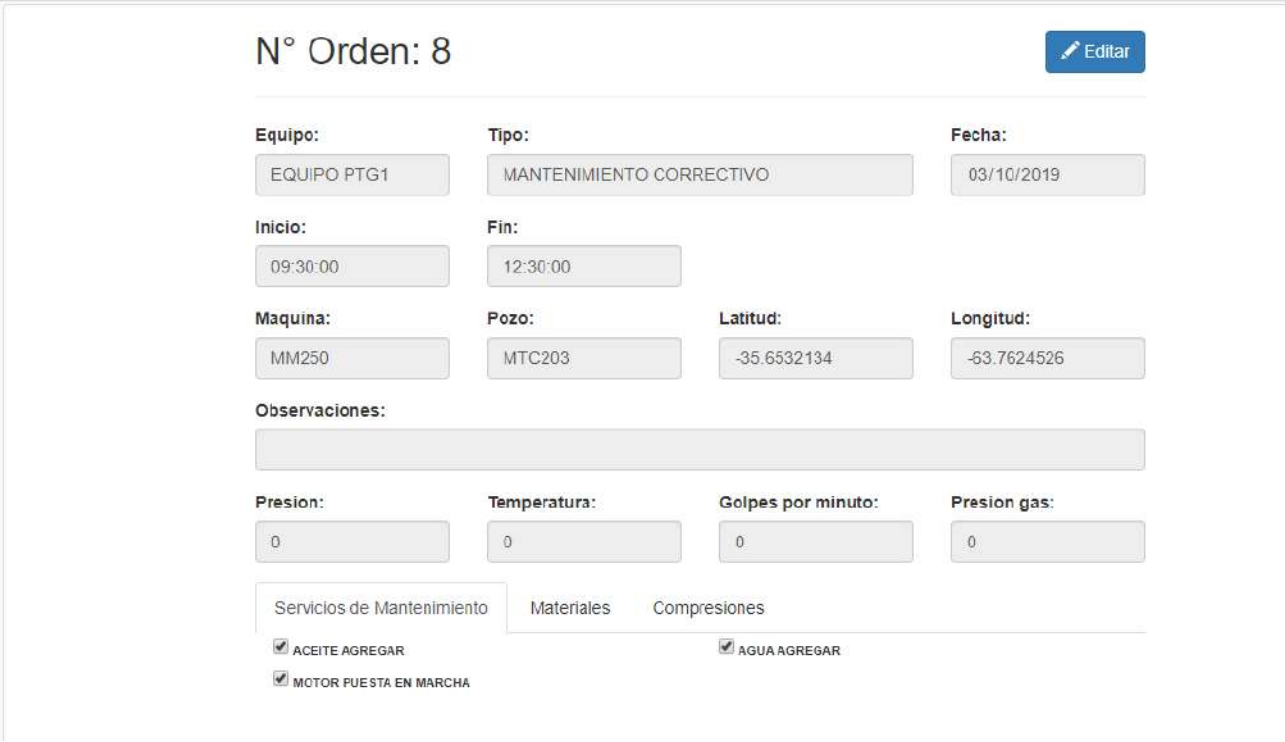


Imagen 55: Pantalla para mostrar una Orden en particular

Como muestra la imagen los servicios de mantenimiento, los materiales y las compresiones se introducen en distintas pestañas. Además, si la orden cuenta con fotografías tomadas por parte de los operarios se podrán observar en la parte inferior de la pantalla, como muestra la imagen 56.

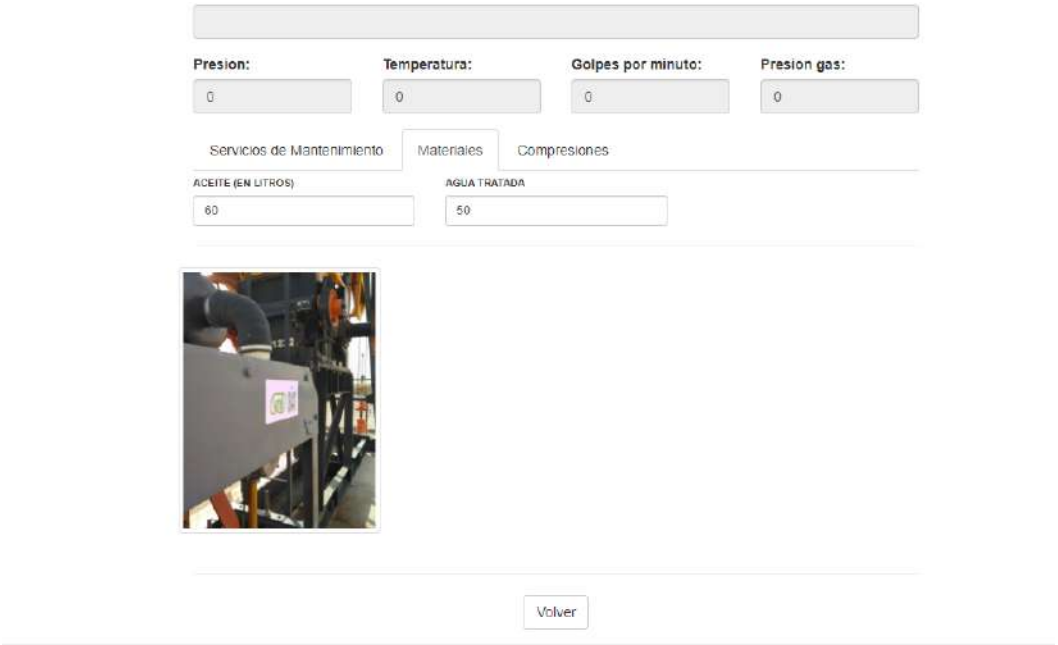


Imagen 56: Pantalla para mostrar una Orden en particular (2)

8.2.3 Testing del módulo Ordenes de Trabajo

En el caso del módulo de OT se ejecutaron un conjunto de *tests* para probar cada uno de los métodos del controlador correspondiente. En primer lugar, se desarrollará la prueba que verifique que se puede acceder al listado de órdenes. El código es el que muestra la imagen 57.

```php
/** @test */
public function mostrar_listado_ordenes(){
    $user = factory(User::class)->make(); //creamos el usuario con factory
    $response = $this->actingAs($user)->get(route('ordenes.index'));
    $response->assertViewIs('ordenes.index');
    $response->assertViewHas('ordenes');
}
```

Imagen 57: Test mostrar_listado_ordenes

Como se ha explicado anteriormente, en primer lugar se debe crear un usuario para poder acceder a las funcionalidades del sistema. Luego, se realiza una petición a la ruta **ordenes/index** y se espera acceder a la vista correspondiente. Al ejecutarse el *test*, se obtiene el siguiente resultado que se muestra en la imagen 58. Como se aprecia, el resultado obtenido es satisfactorio.

```
C:\wamp\www\SGPDM>vendor\bin\phpunit --filter mostrar_listado_ordenes
PHPUnit 5.7.27 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 675 ms, Memory: 14.50MB

OK (1 test, 2 assertions)

C:\wamp\www\SGPDM>
```

Imagen 58: Resultado test mostrar_listado_ordenes

Otro de los *test* llevados a cabo fue el de editar una orden de trabajo siendo usuario. Para ello, el código correspondiente es el de la imagen 59.

```php
/** @test */
public function editar_orden_siendo_usuario(){
    $user = factory(User::class)->make();
    $data = [
            'id_jornada' => 1,
            'nro_orden' => 1,
            'id_maquina' => 1802,
            'id_pozo' => 14,
            'inicio_ot' => "08:00:00",
            'fin_ot' => "18:00:00",
            'latitud' => "-35.6532167",
            'longitud' => "-63.7624447",
            'presion' => 85,
            'temperatura' => 90,
            'golpes_por_minuto' => 5,
            'presion_gas' => 85
            ];

    $response = $this->actingAs($user)->put(route('ordenes.update', 1), $data);
    $response->assertStatus(302);
}
```

Imagen 59: Test editar_orden_siendo_usuario

Como en pruebas anteriores, en primer lugar se crea un usuario mediante el método *factory* de Laravel. Luego, se accede a la ruta **update** enviando como parámetro la variable **$data**. Finalmente, se espera obtener un código de estado 302, es decir, de redireccionamiento hacia la ruta **index**. Como se observa en la imagen 60, el resultado es satisfactorio nuevamente.

```
C:\wamp\www\SGPDM>vendor\bin\phpunit --filter editar_orden_siendo_usuario
PHPUnit 5.7.27 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 643 ms, Memory: 14.75MB

OK (1 test, 1 assertion)
```

Imagen 60: Resultado test editar_orden_siendo_usuario

8.3 Etapa de codificación y pruebas de la iteración 3

Las tareas correspondientes son:

- Implementación de módulo de Compresiones (con las funciones de “Crear”, “Editar” y “Eliminar”) en el sistema SGPDM
- Implementación de la toma de compresiones cilíndricas de un motor desde la aplicación PDMApp
- Generación de reportes de datos de: Materiales utilizados, Motores atendidos, Horas trabajadas en el sistema SGPDM
- Leer códigos QR desde la aplicación PDMApp

Estas surgen de las historias de usuario 14, 15, 16, 17, 18, 19 y 20 (ver Anexo 6).

8.3.1 Implementación de módulo de Compresiones en el sistema SGPDM

Una vez desarrollada la funcionalidad de la carga de órdenes de trabajo desde la aplicación, se puede comenzar con la implementación de una de las solicitudes del cliente en esta iteración y que aporta información relativa a la OT. Este es el caso de las compresiones.

La toma de compresión en un motor es un valor que permite parametrizar a través del tiempo el estado del mismo. Si los valores son desiguales o bajos se puede decir que el motor requiere atención. En motores en V, donde hay 2 bancos de cilindros, en general la numeración de los cilindros se hace por lado izquierdo y derecho, visto desde el lado del volante del motor. Por lo tanto, en caso de motores de 12 cilindros en V, hay 6 cilindros por lado.

En lo que respecta al sistema, similar a los casos del apartado 8.1.4, cuando se accede a la ruta ***/*compresiones**, es el método *index()* del controlador *CompresionController* quien se encarga de la petición. El código se muestra en la imagen 61. Como se puede observar, se trata de un método sencillo donde solamente se procede a buscar aquellas compresiones de la base de datos que se encuentran activas (no discontinuadas) y devolver a la vista “index” para mostrarlas en pantalla con las opciones de visualización, edición, dar de baja, eliminación y creación de una compresión en particular. Esta pantalla se presenta en la imagen 62.

```php
public function index(){
    $compresiones = Compresion::where('discontinuado','=',0)
                    ->orderBy('id_compresion','DESC')
                    ->paginate();
    return view('compresiones.index',compact('compresiones'));
}
```

Imagen 61: Método index de Compresion

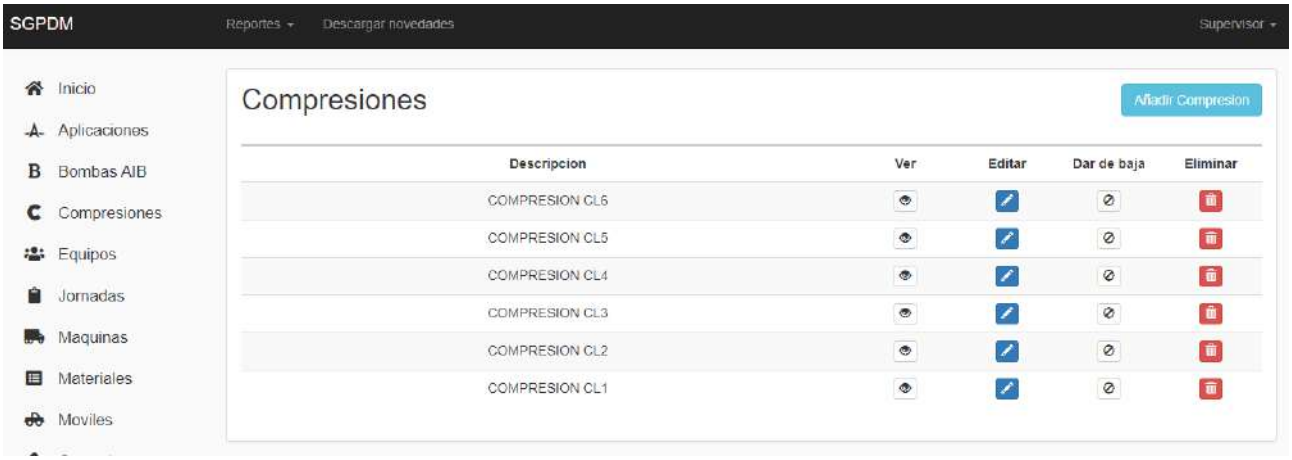


Imagen 62: Pantalla index de Compresiones

El controlador que se nombró al inicio, cuenta con los métodos *create()*, *store()*, *edit()*,*update()*,*show()*, *darBaja()* y *destroy()*, cuyo código de cada uno es muy similar a su homónimo en los módulos del apartado 8.1.4 por ello se obvia su explicación.

8.3.2 Implementación de la toma de compresiones cilíndricas de un motor desde la aplicación PDMApp

Los operarios deben poder guardar las mediciones de compresiones cilíndricas de un motor en una OT. Para agregar esta sección se utilizó el componente TableLayout. Este ViewGroup es el más adecuado para implementar esta tarea: fácilmente organiza cada compresión con sus medidas en filas y columnas. Las mediciones son valores numéricos, por lo tanto a cada componente EditTextView se lo personalizó para que solo permitiera ingresar caracteres numéricos con la siguiente definición:

```
android:inputType="number"
```

Finalmente esta sección agregada en las órdenes de trabajo se visualiza como en la imagen 63.

Imagen 63: Compresión Cilíndrica en OT

8.3.3 Generación de reportes en el sistema SGPDM

Si bien con la entrega de la primera iteración el cliente ya contaba con un producto funcional, en el cual podía cargar datos de los diferentes módulos, fue la segunda iteración quien permitió que se pudieran ir agregando ordenes de trabajo documentadas en papel desde la aplicación para luego ser enviadas al sistema. Con esa información almacenada en el sistema SGPDM, ya era posible la obtención de estadísticas y reportes de interés para el cliente. Pero para ello, había que implementar el módulo correspondiente.

En primer lugar, se explica el proceso llevado a cabo para la generación de estadísticas. Se crea el controlador *GraficoController* para recolectar la información. El cliente solicitó tener acceso en forma rápida a ciertos datos de interés. Por esa razón, es que luego de autenticarse, se muestra un panel con 3 valores correspondientes al mes vigente: cantidad de órdenes de trabajo realizadas, cantidad de pozos atendidos y material más utilizado. Esto se puede apreciar en la imagen 64. A su vez, en el mismo panel se pueden contemplar dos gráficos: uno de barras, que se observa en la imagen 65, en el que se detallan los 10 materiales más usados en el mes (especificando la cantidad consumida) y uno de torta, que se muestra en la imagen 66, que describe la proporción de tipos de OT que se ejecutaron. Es pertinente destacar que para la generación de los gráficos se usó la herramienta Charts[22] que proporciona Google. Existe una amplia galería de tipos y de personalizaciones posibles. Una vez ejecutada la consulta en el controlador y devuelto los resultados a la vista, se procede a "dibujar" mediante Charts. Asimismo, se le ofrece la oportunidad al cliente de ver información entre un rango de fechas ingresado manualmente.

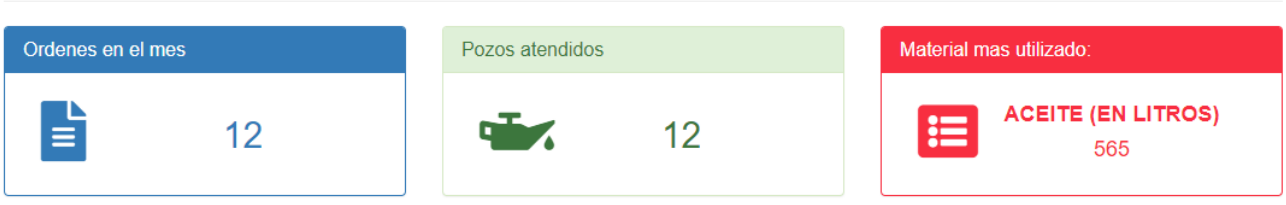


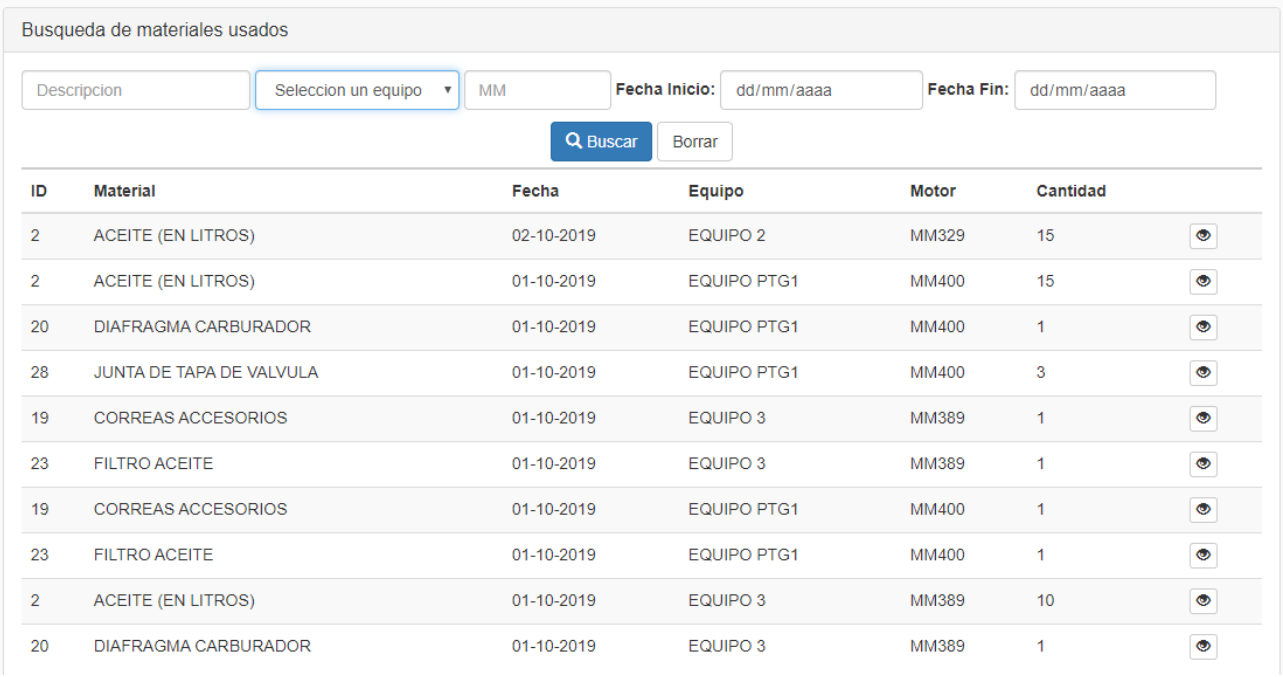
Imagen 64: Información estadística

[22] Charts: https://developers.google.com/chart

Imagen 65: Gráfico de barras de materiales más utilizados

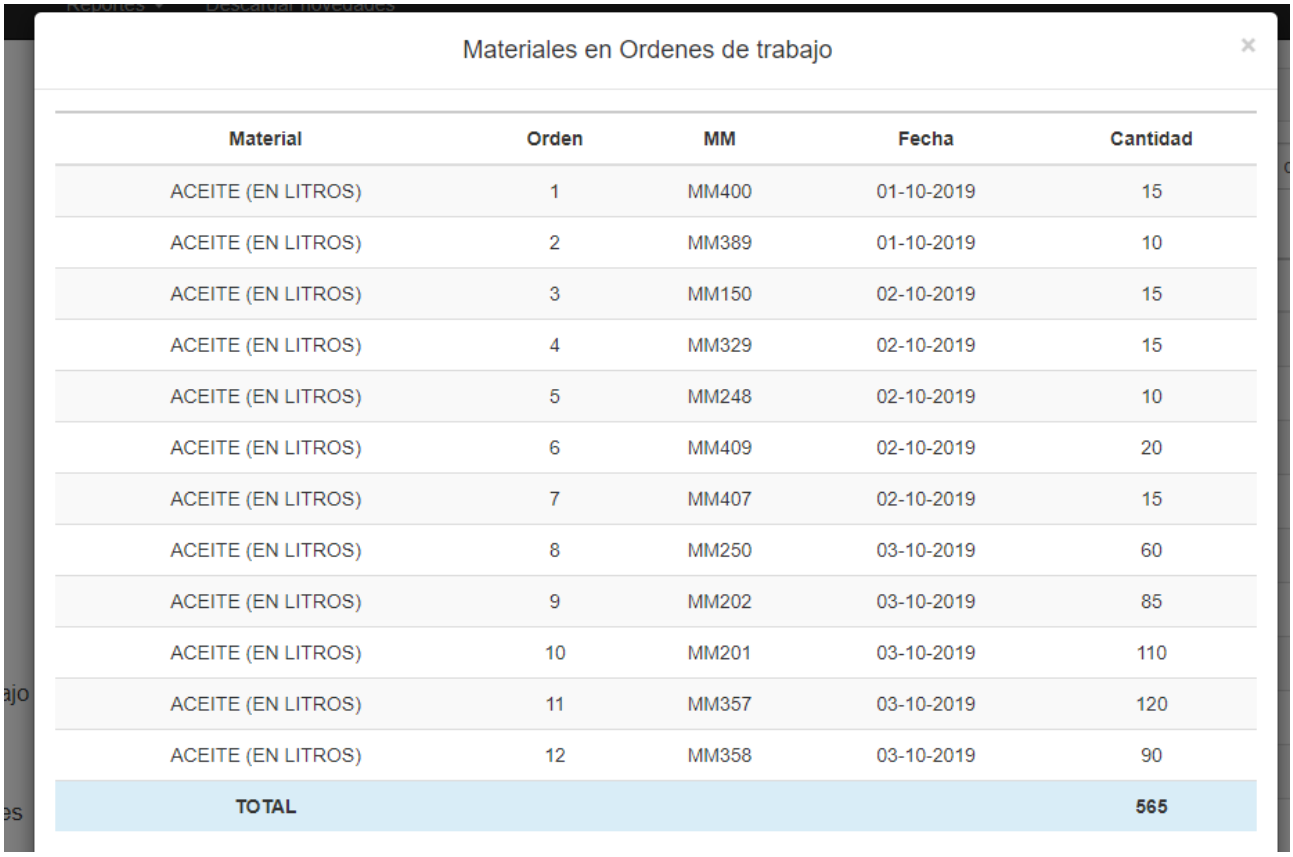
Imagen 66: Gráfico de torta de OT realizadas

En lo que respecta a los reportes, se creó un controlador que será el encargado de regular todas las tareas: *ReporteController*. Como se mencionó en el apartado anterior, se continuará explicando acerca de los materiales. Para ello, cuando se accede a la ruta **/reporte/materiales** es el método *materiales()* quien retorna todos los datos a la vista. Al ingresar al reporte, la pantalla que se observa contiene todos los materiales utilizados en una orden determinada (imagen 67). Sin embargo, también contiene un buscador para fijar parámetros tales como: descripción (nombre del material), equipo, MM, fecha de inicio y fecha de finalización. Además, contiene una opción para observar en detalle todas las ocurrencias de ese material y el total utilizado. Si se presiona el botón que se ubica a la derecha de cada fila, se despliega una ventana modal (imagen 68) con la información correspondiente. Si el cliente completa valores en el campo de búsqueda, se ejecuta el método *reporteMateriales()* quien se encarga de recopilar todos los datos de acuerdo a los criterios ingresados y devolverlos nuevamente a la vista “materiales”.



ID	Material	Fecha	Equipo	Motor	Cantidad
2	ACEITE (EN LITROS)	02-10-2019	EQUIPO 2	MM329	15
2	ACEITE (EN LITROS)	01-10-2019	EQUIPO PTG1	MM400	15
20	DIAFRAGMA CARBURADOR	01-10-2019	EQUIPO PTG1	MM400	1
28	JUNTA DE TAPA DE VALVULA	01-10-2019	EQUIPO PTG1	MM400	3
19	CORREAS ACCESORIOS	01-10-2019	EQUIPO 3	MM389	1
23	FILTRO ACEITE	01-10-2019	EQUIPO 3	MM389	1
19	CORREAS ACCESORIOS	01-10-2019	EQUIPO PTG1	MM400	1
23	FILTRO ACEITE	01-10-2019	EQUIPO PTG1	MM400	1
2	ACEITE (EN LITROS)	01-10-2019	EQUIPO 3	MM389	10
20	DIAFRAGMA CARBURADOR	01-10-2019	EQUIPO 3	MM389	1

Imagen 67: Reporte materiales



Material	Orden	MM	Fecha	Cantidad
ACEITE (EN LITROS)	1	MM400	01-10-2019	15
ACEITE (EN LITROS)	2	MM389	01-10-2019	10
ACEITE (EN LITROS)	3	MM150	02-10-2019	15
ACEITE (EN LITROS)	4	MM329	02-10-2019	15
ACEITE (EN LITROS)	5	MM248	02-10-2019	10
ACEITE (EN LITROS)	6	MM409	02-10-2019	20
ACEITE (EN LITROS)	7	MM407	02-10-2019	15
ACEITE (EN LITROS)	8	MM250	03-10-2019	60
ACEITE (EN LITROS)	9	MM202	03-10-2019	85
ACEITE (EN LITROS)	10	MM201	03-10-2019	110
ACEITE (EN LITROS)	11	MM357	03-10-2019	120
ACEITE (EN LITROS)	12	MM358	03-10-2019	90
TOTAL				**565**

Imagen 68: Ventana modal

Para el caso de los demás reportes, el mecanismo es muy similar, debiendo cada método ejecutar las consultas respectivas a cada informe.

8.3.4 Leer código QR desde la aplicación PDMApp

Para este requerimiento se hizo uso de la clase *IntentIntegrator*. Es una clase de utilidad que facilita la integración con *Barcode Scanner* a través de *Intents*. Esta es una manera simple de invocar el escaneo de códigos de barras y recibir el resultado, sin necesidad de integrar, modificar o aprender el código fuente del proyecto. Para ello primero se debió agregar la librería ZXing en el archivo "build.gradle", como se muestra resaltado con un recuadro rojo en la imagen 69.

```
dependencies {
    implementation 'com.android.support:recyclerview-v7:26.1.0'
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin_version"
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.0'
    implementation 'com.google.android.gms:play-services-location:15.0.1'
    implementation 'com.android.support:design:26.1.0'
    implementation 'com.android.support:support-v4:26.1.0'
    implementation 'com.journeyapps:zxing-android-embedded:3.6.0'
    testImplementation 'junit:junit:4.12'
    testImplementation 'org.mockito:mockito-core:2.18.3'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'
    implementation 'com.github.d-max:spots-dialog:1.1@aar'
    implementation files('libs/commons-net-3.5.jar')
}
```

Imagen 69: Implementando librería ZXing

Para integrarlo, se creó una instancia de *IntentIntegrator*, se llama al método *initiateScan()* y se espera por el resultado en la aplicación. Para esta funcionalidad, se requiere que la aplicación Barcode Scanner esté instalada en el dispositivo. El método *initiateScan()* solicitará que el usuario la descargue, en caso de ser necesario.

Hay algunos pasos que se deben seguir para usar esta integración. Primero en la *Activity* se debe implementar el método *onActivityResult(int, int, Intent)* e incluir las siguientes líneas de código que se muestran en la imagen 70.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    if(resultCode == Activity.RESULT_OK){
        val result : IntentResult! = IntentIntegrator.parseActivityResult(requestCode, resultCode, data)
        if (result != null) {
            // Manejar el resultado escaneado
        } else{
            //continuar con cualquier otro manejo que se necesite
        }
    }
}
```

Imagen 70: Método onActivityResult()

En segundo lugar, en respuesta a la acción del operario de acceder a la sección Orden de Trabajo, comienza el proceso de escaneo con la llamada que determina la imagen 71.

```
val scanner = IntentIntegrator( activity: this)
scanner.setDesiredBarcodeFormats(IntentIntegrator.QR_CODE)
scanner.setBeepEnabled(false)
scanner.initiateScan()
```

Imagen 71: Iniciar el proceso de escaneo

El método *initiateScan()* devuelve un *AlertDialog* que no es nulo si se le solicita al usuario que descargue la aplicación. La librería permite personalizar el título del cuadro de dialogo que solicita la descarga del Barcode Scanner. Del mismo modo se puede modificar el mensaje que se le muestra al usuario y las etiquetas de los botones <<Si>>/<<No>>. Se puede usar el método *addExtra(String Object)* para agregar más parámetros a la intención para invocar el escáner. Esto se puede utilizar para establecer opciones adicionales que no están expuestas directamente por esta API simplificada. El método *setDesiredBarcodeFormats(IntentIntegrator.QR_CODE)* limita a escanear solo códigos QR y *setBeepEnabled(false)* impide que al obtenerse un resultado emita un pitido.

8.4 Codificación y pruebas de iteración 4

Las tareas desarrolladas en la iteración 4 son las siguientes:

- Implementación en el sistema SGPDM del módulo para la carga de recesos laborales
- Desarrollo en la aplicación PDMApp de la funcionalidad de agregar recesos laborales a una jornada de trabajo
- Implementación para la carga de las tareas realizadas en el almacén en el sistema SGPDM
- Desarrollo en la aplicación PDMApp de la funcionalidad de agregar tareas realizadas en el almacén
- Generación de Partes Diarios de Mantenimiento de Motores de Campo (PDMM) y de nuevos reportes, de acuerdo a las últimas peticiones
- Creación de módulos de Órdenes de Trabajo y Jornadas

Las historias asociadas a estas tareas son 21, 22, 23, 24, 25 y 26 (Ver Anexo 6).

8.4.1 Implementación en el sistema SGPDM del módulo de recesos laborales

Uno de los aspectos que se deben considerar cuando se genera una OT es el tiempo que un operario no realiza una acción laboral durante la jornada. Este tiempo es denominado “Receso laboral” e incluye momentos que los empleados hacen una pausa para almorzar, por algún tipo de conflicto social o deben detener sus tareas debido a condiciones climáticas.

En el sistema SGPDM, este módulo fue desarrollado del mismo modo que los módulos en la iteración 1. Al ingresar a la ruta ***/***recesoslaborales**, el método *index()* del controlador *RecesoLaboralController* busca la información almacenada en la base de datos y retorna la vista que mostrará todos los recesos del sistema. El código puede verse en la imagen 72.

```
public function index(){
    $recesos_laborales = RecesoLaboral::orderBy('id_recesoLaboral','ASC')->paginate();
    return view('recesoslaborales.index',compact('recesos_laborales'));
}
```

Imagen 72: Método index de RecesoLaboral

Luego, en “recesoslaborales.index” procedemos a mostrar los datos y la pantalla resultante es la que se observa en la imagen 73.

Imagen 73: Pantalla index de Recesos Laborales

Nuevamente, como en los anteriores módulos, se cuenta con la posibilidad de visualizar, editar y eliminar un receso en particular. Asimismo, se pueden crear nuevos a través del botón <<Añadir Receso Laboral>>.

En este caso, se trata de un módulo más simple, dado que el Receso Laboral solo cuenta con una descripción, así que la información que se muestra o se puede editar al respecto es menor. Lo mismo sucede al momento de crear uno nuevo: solamente se solicita una breve descripción o nombre para el receso. Los pasos correspondientes para la visualización, edición, eliminación y creación de un receso resultan muy similares a los explicados en la iteración 1.

8.4.2 Desarrollo en la aplicación PDMApp de la funcionalidad de agregar recesos laborales

La acción de presionar sobre esta sección genera una nueva actividad. Para poder mostrar las opciones de recesos laborales se implementó el componente *Spinner*. Se crea un *ArrayAdapter* que es alimentado por un arreglo de elementos, este arreglo son los diferentes motivos de recesos laborales. Por último se define el *Adapter* al *Spinner* por medio del método *setAdapter()*. La ventaja de utilizar esta metodología es que si en un futuro se decide agregar algún otro motivo que implique un receso laboral, solo se debe agregar este en el módulo Recesos Laborales del sistema SGPDM, que se explicó en el apartado anterior. También se le da ingreso a los horarios de inicio y fin del receso y un cuadro de texto para el caso que se quiera agregar un comentario.

8.4.3 Implementación de carga de tareas de almacén en el sistema SGPDM

Como se explicó en el apartado de carga de una OT, los operarios llevan a cabo diferentes tareas referidas al almacén. Por ello es necesario llevar un control de las mismas. De igual manera que en los módulos expresados anteriormente, se ejecuta un método *index()* que se encarga de recopilar los datos y mostrar todas las tareas del sistema SGPDM (imagen 74).

```php
public function index(){
    $tareas_almacen = TareasAlmacen::orderBy('id_tarea_almacen','ASC')->paginate();
    return view('tareasalmacen.index',compact('tareas_almacen'));
}
```

Imagen 74: Método index de TareaAlmacenController

Cuando se retorna a la vista, se observa la pantalla con cada una de las tareas y las acciones correspondientes de visualización, edición y eliminación (imagen 75). También se ofrece la posibilidad de añadir una nueva tarea.

Imagen 75: Pantalla index de Tareas de Almacén

Al igual que en el caso de los recesos laborales, el dato que se almacena es solamente una breve descripción de las tareas. De igual manera, los procesos para la visualización, edición, creación y eliminación se llevan a cabo de forma similar a los explicados en la iteración 1.

8.4.4 Desarrollo en la aplicación PDMApp de la funcionalidad de agregar tareas de almacén

Cuando un operador elije la opción Almacén la aplicación abre una nueva actividad, que al igual que en la pantalla de Recesos Laborales, las opciones de las tareas que se pueden realizar son presentadas en un *Spinner*.

Los horarios de inicio y fin solicitan que el operador registre un horario. Estos *EditText*, al ser manipulados por el operador, disparan el *widget Time Picker*. *TimePickerDialog* permite elegir y obtener la hora con suma facilidad gracias a su interfaz gráfica (imagen 76).

Para crear un selector de hora se requiere:

1. Definir un *EditText* como no seleccionable:

```
android:focusable="false"
```

2. Hacer una llamada al método *getTime(EditText, Context).* Este método recibe como parámetro el componente *EditText* que se definió con anterioridad. Para especificar la función que debe realizar cuando se lo presiona se configura el método *onClickListener().* Esta función crea un objeto *Time Picker Dialog* con el tema especificado, pasado también como parámetros. El método *show()* lo hace visible y se cerrará automáticamente cuando el usuario seleccione un horario. La función *getTime*, por su implementación, es reusable y muy fácil de emplear. En la aplicación todos los elementos *EditText*, que requieren el ingreso de un horario por parte del operario, hacen uso del mismo.

Imagen 76: TimePickerDialog

8.4.5 Generación de Parte Diarios de Mantenimiento de Motores de Campo (PDMM) y de nuevos reportes

Como ya se mencionó en el aparto 8.3.3, para la generación de los reportes se creó un controlador que se encargue de este módulo. El método para este informe en particular, es *parteDiarioAdmin()*, que se muestra en la imagen 77. Al acceder a dicho reporte, en primer lugar se observa un cuadro de búsqueda (imagen 78) para que el usuario complete con el equipo que llevó a cabo la jornada y la fecha de la misma. Cuando el usuario presiona <<Buscar>> se ejecuta el método nombrado anteriormente, el cual toma los parámetros ingresados y realiza la búsqueda del parte diario, los recesos laborales involucrados y las tareas de almacén llevadas a cabo, y los devuelve a la vista para que muestre los resultados.

```
public function parteDiarioAdmin(Request $request){
    $equipo = $request->equipo;
    $fecha = $request->fecha;
    $equipos = Equipo::orderBy('id_equipo','ASC')->pluck('descripcion','id_equipo')->toArray();
    $equipo_jornada = Equipo::find($equipo);
    $parte_diario="1";
    $arr_ordenes="";
    $arr_recesos = "";
    $arr_tareas = "";
    if($equipo!=null){
        $parte_diario = Jornada::where('id_equipo',$equipo)
                               ->where('fecha',$fecha)
                               ->first();
        if($parte_diario!=null){

            $arr_ordenes = Orden::where("ordenestrabajo.id_jornada",$parte_diario->id_jornada)
                        ->get();
            $arr_recesos = RecesoLaboral::
                        join('recesoslaborales_en_jornadas','recesoslaborales.id_recesoLaboral','=','recesoslaborales_en_jornadas.id_recesoLaboral')
                                ->where('recesoslaborales_en_jornadas.id_jornada',$parte_diario->id_jornada)
                                ->orderBy('hora_inicio')
                                ->get();
            $arr_tareas = TareasAlmacen::
                        join('tareas_almacen_en_jornada','tareas_almacen_en_jornada.id_tarea_almacen','=','tareasalmacen.id_tarea_almacen')
                                ->where('tareas_almacen_en_jornada.id_jornada',$parte_diario->id_jornada)
                                ->orderBy('hora_inicio')
                                ->get();
        }
    }
    return view('reportes.partes-diarios',compact('equipo_jornada'))->with('equipos',$equipos)
                                    ->with('parte_diario',$parte_diario)
                                    ->with('equipo_jornada',$equipo_jornada)
                                    ->with('arr_ordenes',$arr_ordenes)
                                    ->with('arr_recesos',$arr_recesos)
                                    ->with('arr_tareas',$arr_tareas)
                                    ->with('fecha',$fecha)
                                    ->with('equipo',$equipo);
}
```

Imagen 77: Método parteDiarioAdministracion

Imagen 78: Búsqueda de Partes Diarios

Una vez que se tienen los datos, en primer lugar se observa el encabezado del parte diario con la información correspondiente al equipo que realizó la labor. Seguidamente, se procede a mostrar todas las tareas que se llevaron a cabo en orden cronológico. Por último, se computa el total de horas acumuladas en todo el proceso laboral (sin tener en cuenta los recesos laborales). El resultado se expone en la imagen 79.

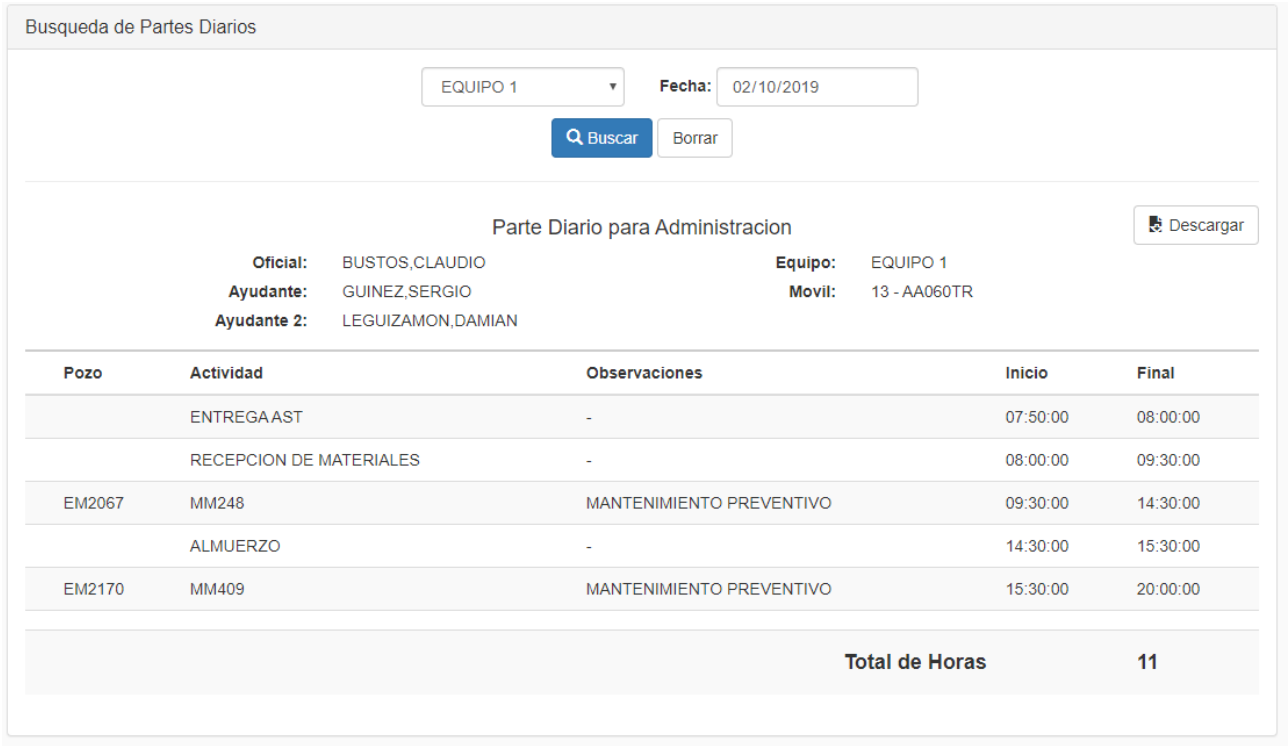


Imagen 79: Parte Diario para Administración

Además de la información mostrada en pantalla, se ofrece la posibilidad de acceder al parte diario en formato PDF. Para ello, se debía instalar al proyecto un paquete externo que permitiera llevar a cabo esa tarea. El componente elegido fue Dompdf[23] debido a su compatibilidad con CSS y, de esta manera, se puede dar estilo al PDF como si se tratara de una página HTML del sistema web.

[23] Dompdf es (principalmente) un motor de diseño y renderización HTML compatible con CSS 2.1 escrito en PHP. Es un renderizador basado en el estilo: descargará y leerá hojas de estilo externas, etiquetas de estilo en línea y los atributos de estilo de elementos HTML individuales. También es compatible con la mayoría de los atributos HTML de presentación. (https://github.com/dompdf/dompdf)

Una vez agregada la herramienta, se procede a escribir el código para la generación del documento. Cuando se presiona el botón <<Descargar>>, se ejecuta el método *exportarParteDiario()* de *ReporteController*. En lo que respecta a la búsqueda de los datos, es muy similar al descripto anteriormente para mostrar la información en pantalla. La diferencia radica al momento de retornar los datos (imagen 80).

```
$pdf = PDF::loadView('pdf.partes-diarios',
                    ['arr_ordenado'=>$arr_ordenado,'equipo_jornada'=>$equipo_jornada,
                        'parte_diario'=>$parte_diario,'total_horas'=>$total_horas
                    ]);
return $pdf->stream('parte-diario.pdf');
```

Imagen 80: Retornar datos de PDF

Para esta tarea, se instancia al objeto PDF y se ejecuta el método *loadView()*. Este recibe dos parámetros: en primer lugar la vista que se encargará de renderizar los datos y en segundo lugar, las variables que se deseen enviar a esta vista. Por último, existen 3 formas de mostrar el documento: guardarlo como un archivo con el método *save()*, mostrarlo en el navegador con el método *stream()* o descargarlo directamente con el método *download()*. En nuestro caso se escogió la segunda opción, donde se le pasa por parámetro el nombre que llevará el documento.

8.4.6 Creación módulos de Ordenes de trabajo y Jornadas

Al igual que en el caso del módulo Maquinas, cuando se selecciona la opción editar se ejecuta el método *edit()*, pero de *OrdenController*, como se visualiza en la imagen 81. El método recibe el **$id** de la orden que se desea editar. La misma se busca en la base de datos y se envía como variable a la vista “edit”. Además, también se envían los datos de los equipos, máquinas, pozos, servicios, materiales, compresiones y tipos de OT.

```
public function edit($id){
    $orden = Orden::find($id);
    $equipos = Equipo::where('discontinuado','=',0)->orderBy('id_equipo','ASC')->pluck('descripcion','id_equipo');
    $maquinas = Maquina::where('discontinuado','=',0)->orderBy('id_maquina','ASC')->pluck('mm','id_maquina');
    $pozos = Pozo::where('discontinuado','=',0)->orderBy('id_pozo','ASC')->pluck('nombre_pozo','id_pozo');
    $servicios = Servicio::where('discontinuado','=',0)->orderBy('id_servicioMantenimiento','ASC')->get();
    $materiales = Material::where('discontinuado','=',0)->orderBy('id_material','ASC')->get();
    $compresiones = Compresion::where('discontinuado','=',0)->orderBy('id_compresion','ASC')->get();
    $tipos_ot = DB::table('tipoot')->select('id_tipoot','descripcion')->pluck('descripcion','id_tipoot');
    return view('ordenes.edit',compact('servicios'),compact('materiales'))->with('orden',$orden)
                                                                            ->with('equipos',$equipos)
                                                                            ->with('maquinas',$maquinas)
                                                                            ->with('pozos',$pozos)
                                                                            ->with('compresiones',$compresiones)
                                                                            ->with('tipos_ot',$tipos_ot);
}
```

Imagen 81: Método edit de OrdenController

La pantalla que se observa seguidamente es similar a la de visualización (mostrada en 8.2.2), con la diferencia de que los campos pueden ser modificados (imagen 82).

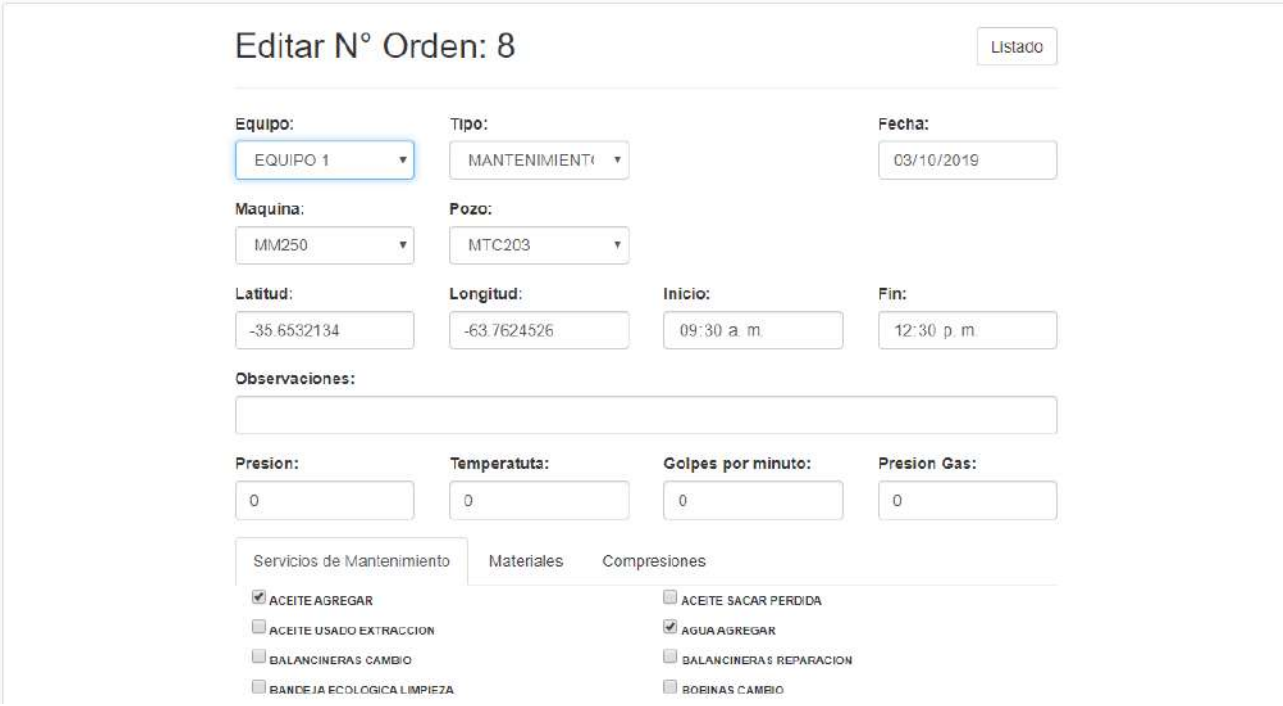


Imagen 82: Pantalla de edición de una Orden

Para el caso en que se quieran agregar nuevos materiales o compresiones, existe un botón, como muestra la imagen 83, que al presionarlo despliega una ventana modal con los datos que se deseen añadir (imagen 84). Se completan los valores y al pulsar <<Guardar>> se cargan en la orden.

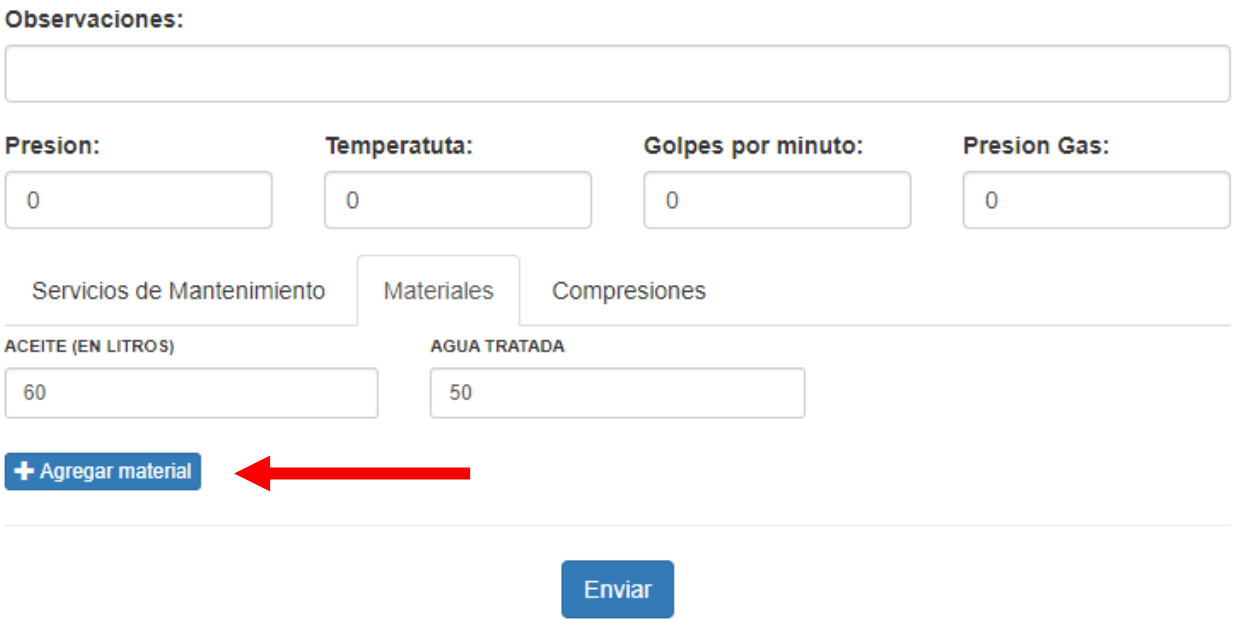


Imagen 83: Agregar material

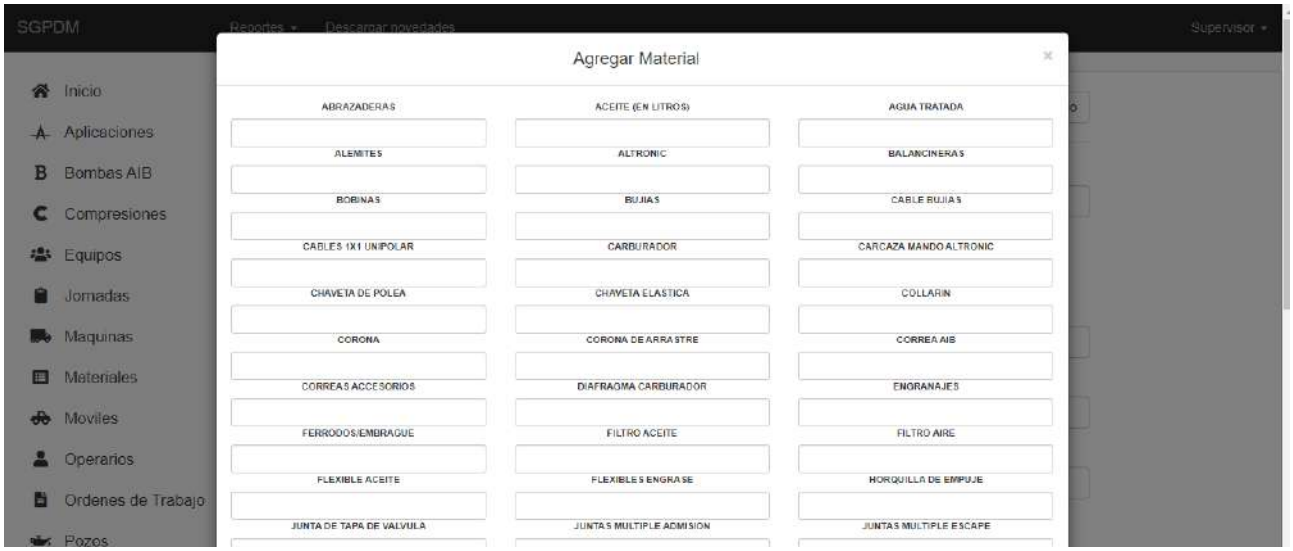


Imagen 84: Ventana modal para agregar materiales

Realizadas las modificaciones deseadas, se presiona el botón <<Enviar>>, se ejecuta el método *update()* y se guardan los cambios en la base de datos.

9. Conclusiones

La práctica de desarrollo del proyecto resultó satisfactoria por varios motivos. Desarrollar bajo nuevas tecnologías, lenguajes de programación y *frameworks* era un desafío enorme, una batalla contra el tiempo que aun así se logró, adquiriendo gran experiencia al hacerlo. Como equipo, comprendimos el proceso de desarrollo del proyecto utilizando la metodología ágil XP. Las técnicas que aporta fueron de vital importancia para cumplir con cada tarea en forma ordenada y en tiempo de entrega pactado. Hoy, el proyecto YAL está implantado y en uso, concluyendo en un resultado positivo en términos de satisfacción del cliente, brindándole un conjunto de sistemas de acuerdo a sus necesidades.

Asimismo se deduce que XP tiene tanto ventajas como desventajas. A continuación, se exponen las mismas, no con intención de calificarla pero son aspectos útiles y de referencia a la hora de tratar con proyectos de condiciones similares al que se desarrolló en este documento.

Uno de los aspectos más importantes que manifiesta XP es el de realizar entregas cortas y rápidas, incrementando la funcionalidad en cada una de ellas. Esto posibilitó que el cliente tuviera en sus manos una versión funcional tanto del sistema como de la aplicación para ir haciendo uso de los mismos en el corto plazo. Las entregas frecuentes, permiten que el cliente realice aportes o manifieste funcionalidades que quizás no fueron considerados al inicio del proyecto. Asimismo, generan retroalimentación inmediata en cada iteración, agilizando los procesos de modificaciones y mejoras. A su vez, al observar cambios de manera continua, el cliente siempre se encuentra motivado e interesado y esto conllevó a generar un buen ambiente de trabajo y la continuidad del proyecto.

Otro de los puntos destacables, es que no había un contrato laboral rígido ni una relación formal, lo cual permitió un trato más fluido y personal con el cliente. Aunque la metodología recomienda que éste se encuentre siempre presente, enfatizando en el diálogo “cara a cara”, se pueden lograr los mismos resultados con una comunicación continua utilizando herramientas como correo electrónico y llamadas telefónicas.

En nuestra experiencia, al inicio el cliente estaba seguro de las necesidades mayores del proyecto, sin embargo no tenía muy en claro los requerimientos en su totalidad y las reuniones continuas permitieron que los mismos fueran surgiendo. En la actualidad, se continúa desarrollando nuevos requerimientos y existen otros en iteraciones futuras, como por ejemplo: informe de control de stock de materiales, reporte de liquidación de sueldo según el tipo de OT y horas trabajadas, organización del recorrido de pozos de cada equipo desde el sistema SGPDM y se refleje en la aplicación, etc. En una metodología tradicional, esto conllevaría un retroceso a las etapas iniciales, no obstante XP tiene la flexibilidad y/o adaptabilidad de ajustarse a las nuevas peticiones y posibles modificaciones.

Debido a que el equipo de trabajo estaba conformado solamente por dos integrantes, se debieron considerar cambios en la metodología, que aunque se encuentra ideada para un grupo numeroso de programadores, el uso de la misma no se vio afectada. Esto se obtuvo gracias a la continua comunicación entre ambos, estrategias de integración y capacitación, la adjudicación de responsabilidades de cada uno y el cumplimiento de plazos individuales y grupales.

Al hacer modificaciones en XP, se corre el riesgo de alterar la metodología. Por ello, cada uno de los cambios se pensó cuidadosamente para que no se pierda la esencia de la misma y se realizaron porque eran sumamente necesarios para el proyecto.

Para desarrollar la aplicación en Kotlin se debió estudiar el lenguaje. Kotlin tiene muchos beneficios pero para la estructura y magnitud de este proyecto no eran relevantes. Aun así aprender una nueva tecnología siempre aporta experiencia y formación que podrá ser aplicada en otros

proyectos. En lo que respecta a la elección del lenguaje de programación para llevar a cabo el sistema orientado a la web, si bien era conocido por el programador responsable de dicha tarea, era la primera vez que utilizaba un *framework* de PHP. Aunque demandó tiempo el conocimiento y entendimiento del mismo, fue de mucha utilidad y se adaptó en forma satisfactoria a las necesidades requeridas. Contiene numerosas ventajas que fueron beneficiosas (utilización del patrón MVC, funciones de fácil comprensión, entre otras) en el proyecto y seguramente será utilizado en el futuro.

10. Referencias

[1] «Revista Petroquimica,» 20 Abril 2017. [En línea]. Available: https://www.revistapetroquimica.com/una-empresa-estatal-licitara-dos-las-areas-petroleras-mas-importantes-la-pampa/. [Último acceso: 17 Enero 2020].

[2] «Gobierno de La Pampa,» [En línea]. Available: https://www.lapampa.gob.ar/images/Archivos/BoletinOficial/2017/Anexo_Decreto_2131.pdf. [Último acceso: 14 Enero 2020].

[3] R. S. Pressman, Software Engineering. A Practitioner's Approach, McGraw-Hill International, 2001.

[4] F. Zuppa, Desarrollo de software ágil, 2019.

[5] K. Beck y J. Grenning, «Manifiesto ágil,» 2001. [En línea]. Available: http://agilemanifesto.org/iso/es/manifesto.html. [Último acceso: 28 Marzo 2020].

[6] K. Schwaber, M. Beedle y R. Martin, «Agile Software Development with SCRUM,» Prentice Hall, 2001.

[7] A. Cocburn, «Agile Software Development,» Addison-Wesley, 2001.

[8] J. Highsmith y K. Orr, «Adaptive Software Development: A Collaborative Approach to Managing Complex Systems,» Dorset House, 2000.

[9] G. A.U.S Torossi, El proceso unificado del desarrollo de software, 2010.

[10] J. Stapleton, Dsdm Dynamic Systems Development Method: The Method in Practice, Addison-Wesley, 1997.

[11] D. Wells, «Extreme Programming: A gentle introduction,» [En línea]. Available: http://www.extremeprogramming.org/. [Último acceso: 15 Abril 2020].

[12] [En línea]. Available: https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema#Roles. [Último acceso: 23 Enero 2020].

[13] «Laravel: The PHP Framework for web artisans,» [En línea]. Available: https://laravel.com/. [Último acceso: 16 Abril 2020].

[14] «Crea lo que quieras en Android,» [En línea]. Available: https://developer.android.com/studio/test. [Último acceso: 20 Mayo 2020].

[15] «Mockito framework site,» [En línea]. Available: https://site.mockito.org/. [Último acceso: 30 Abril 2020].

[16] «Android,» [En línea]. Available: https://www.android.com/intl/es_es/. [Último acceso: 19 Mayo 2020].

[17] «Kotlin,» [En línea]. Available: https://kotlin.es/sobre-kotlin/. [Último acceso: 16 Abril 2020].

Anexos

Anexo 1: Glosario

AIB: Aparato Independiente de Bombeo. Bomba de varilla, también conocida como guanaco, cigüeña, unidad de bombeo, válvula para pozos petroleros, bomba de viga. Es la parte superficial de una bomba impelente de pistón, instalada en una perforación petrolera (ver imagen 85).

Aplicación de motor: Modelo del motor MM. Marca del motor. Por ejemplo: Deutz, Kubota.

AST: Asignación Segura de Trabajo. La Asignación Segura de Trabajo es básicamente una lista de verificación de seguridad. Tiene como finalidad identificar los agentes de riesgo a los cuales están expuestos los trabajadores en la ejecución de sus tareas diarias y poder ganar el compromiso del personal hacia los procedimientos seguros. Las partes básicas de una AST son:

1. Pasos básicos del trabajo e identificación del personal.
2. Accidentes potenciales o condiciones peligrosas.
3. Recomendaciones de acciones y procedimientos.

Discontinuado: Variable de valor 0 o 1.
0 - No discontinuado, activo.
1 - Discontinuado, no activo.
Para mantener el historial hay parámetros de la base de datos que no pueden eliminarse de manera permanente, pero si es necesario que no sean visibles para futuras entradas de información mientras mantenga ese estado. Por ejemplo: Si un operario de desvincula de la empresa, no eliminamos esta entrada en la base de datos, sino que la discontinuamos y ya no figurara como mecánico para futuros trabajos.

Equipo: Grupo de operarios trabajando en conjunto. Un equipo está conformado por un mecánico oficial, un mecánico ayudante y puede tener un segundo ayudante.

LPEM: La Pampa El Medanito.

MM: Módulo Motor.

MMxxx: Identificación única a un motor. Ejemplo: MM124.

Móvil: Vehículo utilitario. Se identifican por su dominio.

OT: Orden de Trabajo. Planilla con descripción detallada del servicio realizado a un motor especifico.

PDMM: Parte Diario de Mantenimiento Motores de Campo. En este reporte se resume la jornada diaria de un equipo en particular. Este informe es de interés para la empresa petrolera, cuenta con la información necesaria y de utilidad para archivar y liquidar el pago a la empresa contratada.

PTG: Plantas de Tratamiento de Gas. No es lo más habitual pero suelen realizar trabajos de mantenimiento en instrumentos de la planta de gas.

PTC: Plantas de tratamiento de Crudo. Planta de Petróleo.

SOP: Supervisor Mecánico de la Operadora.

Tipo de OT: Tipo de servicio que se le realiza a un motor. Las opciones son Mantenimiento Preventivo, Mantenimiento Correctivo y Puesto en marcha por producción.

Imagen 85: AIB – Bomba de Varilla

Anexo 2: Kotlin

Kotlin es el lenguaje de programación elegido para el desarrollo de la aplicación Android. Es un lenguaje open source relativamente joven que gradualmente está tomando fuerza en la comunidad de desarrolladores por sus ventajas comparado con otros lenguajes de programación. Kotlin fue creado por JetBrains en 2012 y colaboró junto con Google para crear Android Studio. En la actualidad, Google ya le provee soporte como lenguaje oficial en el desarrollo de Android. Kotlin es un lenguaje orientado a objetos de calidad pensado para funcionar con Máquina Virtual de Java (JVM) y Android [17].

2.1 Kotlin frente a Java

- Más rápido en compilar: En sus primeras versiones de la librería, el tiempo de compilación de un proyecto realizado en Kotlin requería de más tiempo respecto a uno puramente Java. Pero cuando se incluyó las construcciones incrementales, cambió notoriamente y ha demostrado ser incluso superior.
- Menos código: Kotlin nos permite escribir menos código que en Java. JetBrains se esforzó en hacer el lenguaje lo más conciso posible. Menos código mejora la legibilidad, y si se hace correctamente implica menos errores. El *framework* toma el control de ciertos aspectos rutinarios del código, permitiendo enfocarse en la productividad. Kotlin prioriza la máxima legibilidad y simplicidad, facilitando el proceso de desarrollo de la aplicación.
- Interoperabilidad: El código de Java y Kotlin pueden convivir en un mismo proyecto sin problemas. La solución de Kotlin es compilar su código a un bytecode que se puede ejecutar en la JVM. Por lo que todas las librerías y *frameworks* hechos en Java se pueden importar y ejecutar en un proyecto Kotlin. Gracias a su versatilidad, cualquier librería desarrollada en Kotlin es usable en un proyecto Java. Es fácilmente integrable con Maven, Gradle u otras herramientas.
- Integración en Gradle: Kotlin fue elegido por Gradle para escribir *scripts*, como una alternativa a Groovy. Por lo que podemos escribir estos para configurar nuestro proyecto en Android Studio y tener ayudas de autocompletado y detección de errores en tiempo de compilación.
- Seguridad: Los desarrolladores de Kotlin integraron en su semántica errores comunes en la ejecución del programa. Otra razón por la que Kotlin tiene un alto nivel de seguridad, comparado con Java. A su vez es seguro a NullPointerException. Todas las situaciones nulas de nuestro código serán avisadas en tiempo de compilación. Hay que especificar explícitamente, al lenguaje de programación, que un objeto puede ser nulo y luego comprobar su nulidad antes de usarlo. Las referencias nulas son controladas por el sistema de tipos, y verificará si el código contiene una instanciación nula. Esto evitará horas de depuración innecesarias, además de que hará el código mucho más estable.
- Co-rutinas: Existen muchas formas de realizar una tarea asíncrona desde Java, pero todas ellas llenas de inconvenientes. En su versión 1.1, Kotlin incorpora las co-rutinas, además de la interoperabilidad con JavaScript para el desarrollo web. Con las co-rutinas, se puede escribir código asíncrono, que se ha escrito tradicionalmente usando el patrón Callback, pero utilizando estilo síncrono. El valor de retorno de la función proporcionará el resultado de la llamada asíncrona.

- Su evolución está cubierta: JetBrains es la empresa detrás de Kotlin y ella misma lo usa para sus propios productos, de forma que están más que interesados en seguir desarrollando y mejorándolo. Tiene un numeroso y muy competente equipo comprometido trabajando en él, manteniendo un modelo de negocio estable.
- Comunidad de soporte: Si bien la sintaxis de Kotlin es breve, clara e intuitiva, puede que se vuelva un poco difícil de leer y entender al principio. Al ser un lenguaje joven, la comunidad de desarrolladores es escasa comparado a otros lenguajes bien consolidados. Por lo tanto, la resolución de dudas, documentación, blogs, tutoriales o información sobre Kotlin es mucho menor.

Anexo 3: Laravel

El sistema web fue desarrollado mediante la herramienta Laravel. Se trata de un *framework* de código abierto para crear aplicaciones y servicios web con PHP 5 y PHP 7.

3.1 Características de Laravel

El *framework* Laravel trabaja con una arquitectura de carpetas avanzada, de modo que promueve la separación de los archivos con un orden correcto y definido. A su vez, dispone también de una arquitectura de clases muy adecuada, que promueve la separación del código por responsabilidades. Su estilo arquitectónico es MVC.

Contiene además un amplio conjunto de características, que sirven para realizar la mayoría de las aplicaciones web. Entre ellas podemos encontrar:

- Un sistema de rutas, mediante las cuales es fácil crear y mantener todo tipo de URLs amistosas a usuarios y buscadores, rutas de API, etc.
- Sistema Blade como motor de plantillas
- Un sistema de abstracción de base de datos, con un ORM potente pero sencillo de manejar, mediante el que se puede tratar los datos de la base de datos como si fueran simples objetos
- Soporte para MVC
- Un sistema para creación de colas de trabajo, de modo que es posible enviar tareas para ejecución en *background* y aumentar el rendimiento de las aplicaciones
- Varias configuraciones para envío de e-mail, con proveedores diversos
- Un sistema de notificaciones a usuarios, mediante e-mail, base de datos y otros canales
- Una abstracción del sistema de archivos, mediante el cual podemos escribir datos en proveedores *cloud*, y por supuesto en el disco del servidor, con el mismo código
- Gestión de sesiones
- Sistema de autenticación, con todo lo necesario como recordatorios de clave, confirmación de cuentas, recordar un usuario logueado, etc.
- La posibilidad de acceder a datos en tiempo real y recibir notificaciones cuando éstos se alteran en la base de datos

Anexo 4: Patrón MVC

MVC (Modelo-Vista-Controlador) es un patrón arquitectónico de software que separa una aplicación en tres capas descritas como su acrónimo lo indica. Laravel, así como la mayoría de *frameworks* en PHP, implementan este patrón de diseño en donde cada capa maneja un aspecto de la aplicación. Pero antes de ver cómo Laravel está diseñado para implementar este patrón de software, se tratará de dejar este concepto un poco más claro definiendo cada una de sus partes.

Modelo: Hace referencia a la estructura de datos de la aplicación. Los datos pueden ser transferidos desde la base de datos, una clase, un servicio, u otros, directamente a la vista o ser transformados en el controlador para ser actualizados nuevamente al origen.

Vista: Es la representación de la información en una interfaz de usuario. Por lo general, en interfaces no estáticas se representan los datos que vienen directamente del modelo o estos son transformados en un proceso intermedio en el controlador. En vistas estáticas, normalmente, no hace falta que las vistas sean renderizadas con datos enviados del controlador.

Controlador: Contiene la lógica de la aplicación, los procedimientos, algoritmos y rutinas que hacen que funcione el software. Actúa como interfaz entre los componentes de modelo y vista aplicando las transformaciones y lógica necesarias.

En la imagen 86 se muestra un ejemplo relacionado con una petición de un usuario.

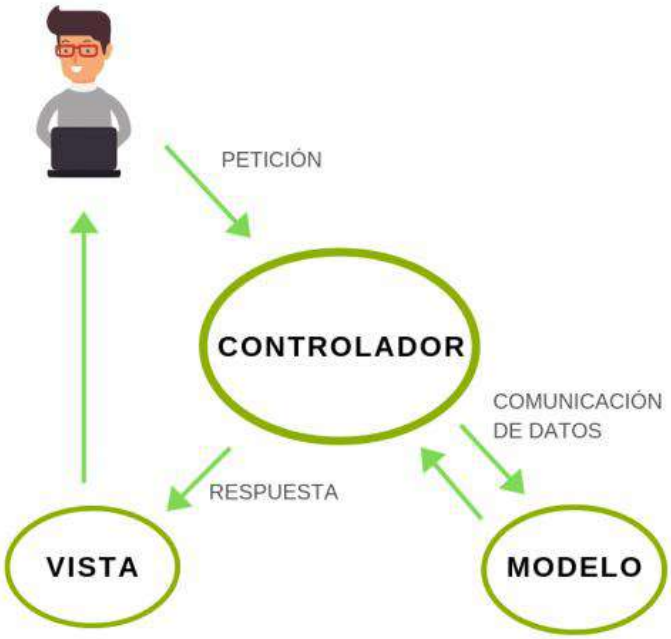


Imagen 86: Ejemplo de MVC

1. El usuario realiza una petición. En nuestro caso desea modificar el nombre de una tabla.
2. El Controlador recibe dicha petición (de la interfaz gráfica) y procede a ejecutar la acción pertinente enviando los datos al Modelo.
3. El Modelo realiza las modificaciones con la base de datos (mysql, postgresql, sqlite).
4. Luego, devuelve al Controlador los cambios solicitados.
5. Cuando el Controlador recibe todos los datos del cambio, envía una respuesta a la Vista.
6. La Vista aplica la modificación en la interfaz del navegador mostrando al usuario.

MVC en Laravel

El MVC en Laravel está implementado de la siguiente manera. En una aplicación web, los controladores estarán situados en la carpeta **app/Http/Controllers**, los modelos directamente en **app** y las vistas en **resources/views**. Para crear un nuevo controlador, basta ejecutar el comando:

php artisan make:controller HelloWorld

Este comando creará el archivo **app/Http/Controllers/HelloWorld.php** que tendrá el siguiente contenido:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HelloWorld extends Controller
{
    //
}
```

Si se conoce de antemano esta estructura, se puede codificar el controlador directamente. Pero es mucho más dinámico crearlo desde Artisan. Como se puede ver este comando creó un controlador que hereda directamente de la clase *App\Http\Controllers* de Laravel. Para que el controlador sea funcional se agregará el siguiente método:

```php
public function sayHello(){
    return view('hello');
}
```

Los métodos en los controladores de Laravel por lo general suelen retornar arrays, objetos o vistas. En este caso, se ha utilizado el *helper* view para retornar la vista **hello** la cual se creará a continuación.

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>App title</title>
<body>
    <h1>HELLO WORLD!</h1>
</body>
</html>
```

Este archivo debe ser guardado con el nombre **hello.blade.php** en la carpeta **resources/views**. Finalmente se agrega la siguiente línea en el archivo **routes/web.php** para poder acceder al método *sayHello*, el cual retornará la vista **hello**.

```
Route::get('/helloworld', 'HelloWorld@sayHello');
```

Al acceder a la ruta **/helloworld** junto con el dominio raíz de la aplicación en el navegador deberá visualizarse el mensaje HELLO WORLD!

Para agregar el componente del modelo, se ejecuta el siguiente comando en la terminal:

php artisan make:model Hello

Esto creará el archivo **app/Hello.php** con el siguiente contenido:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Hello extends Model
{
    //
}
```

Después de esto, se agrega el siguiente método al modelo para retornar el saludo:

```
public static function helloWorldMessage(){
    return "HELLO WORLD!";
}
```

A continuación, se realiza la siguiente adecuación en el controlador para enviar el mensaje del modelo a la vista:

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Hello;  // aquí importamos el modelo

class HelloWorld extends Controller
{
    public function sayHello()
    {
        return view('hello', ["message" => Hello::helloWorldMessage()]);
    }
}
```

Finalmente se modifica la vista para imprimir el mensaje enviado por el controlador:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>App title</title>
<body>
    <h1> {{ $message }} </h1>
</body>
</html>
```

Y con esto al acceder nuevamente a la ruta **/helloworld** se observará nuevamente el mensaje HELLO WORLD! Pero esta vez traído directamente desde el modelo.

Anexo 5: Plantillas Blade

Son archivos de texto plano que contiene todo el HTML de la página con etiquetas que representan elementos o zonas a incluir en la plantilla, o vistas parciales como se conocen en otros *frameworks* PHP. Sin embargo, en Blade estos elementos incrustados se organizan en un solo archivo. Esta es una idea muy interesante de Laravel que mejora la organización de las vistas y su rendimiento. Sobre todo cuando las vistas pueden llegar a ser muy complejas incluso con elementos anidados. En el *render* de una vista completa en Laravel se usan dos archivos: la plantilla definiendo el HTML global y las zonas a incluir. Un sólo archivo, la vista, con los elementos (partial views).

Un ejemplo típico para una plantilla HTML5 de Blade:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <title>@yield('titulo')</title>
    </head>
    <body>
        @yield('navegacion')
    </body>
</html>
```

En la plantilla presentada, el código **@yield()** identifica al método donde como parámetro se indica el nombre de la zona desplegar. Por otro lado, el código de la vista, donde se define la plantilla a usar y la información de las distintas zonas a desplegar:

```
<!-- identificando la plantilla a utilizar -->
@extends('template')

<!-- definiendo una zona -->
@section('titulo')

@endsection

<!-- definiendo otra zona -->
@section('navegacion')

@endsection
```

Anexo 6: Historias de usuarios

En este anexo se incluyen todas las historias de usuario desarrolladas en el presente proyecto (incluidas la 5 y la 20 a modo de ejemplo en el capítulo 3) implementadas en las iteraciones del proyecto, según se especificó en el capítulo 8.

Historia de usuario	
Número: 1	
Usuario: Administrador / SOP	
Nombre historia: Acceso a SGPDM	
Prioridad: Alta	**Riesgo en desarrollo:** Alta
Tiempo estimado: 8 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Existen 2 tipos de usuario en el sistema: Supervisor y Administrador. Cada uno de ellos cuenta con acceso a diferentes funcionalidades. Se debe desarrollar el sistema teniendo en cuenta esta distinción	

Historia de usuario	
Número: 2	
Usuario: SOP	
Nombre historia: ABM de aplicaciones	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Ingresar una nueva aplicación de motor, modificar una existente o dar de baja para que deje de estar vigente.	

Historia de usuario	
Número: 3	
Usuario: SOP	
Nombre historia: ABM de bombas AIB	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Dar de alta un AIB que no esté en sistema, modificar uno existente o dar de baja una bomba que ya no se desee que esté vigente.	

Historia de usuario	
Número: 4	
Usuario: SOP	
Nombre historia: ABM de operarios	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Dar ingreso a un nuevo operario, posibilidad de modificar sus datos y dar baja en caso que no trabaje más en la empresa. Si se da de baja, se debe dar de baja el equipo que integre	

Historia de usuario	
Número: 5	
Usuario: SOP	
Nombre historia: Alta/Baja de equipos de trabajo	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Generar un nuevo grupo de trabajo y discontinuarlo en caso de que ya no exista esa formación de operarios.	

Historia de usuario	
Número: 6	
Usuario: SOP	
Nombre historia: ABM de móviles	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Dar ingreso a sistema a un vehículo, posibilidad de modificar sus datos o dar de baja en caso que ya no se use para las jornadas laborales.	

Historia de usuario	
Número: 7	
Usuario: SOP	
Nombre historia: ABM de zonas	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Posibilidad de dar ingreso al sistema una zona petrolera, modificar sus datos y también debe poder darse de baja.	

Historia de usuario	
Número: 8	
Usuario: SOP	
Nombre historia: ABM de pozos	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Posibilidad de dar ingreso al sistema un pozo petrolero, identificando la zona en la que se encuentra y la ubicación exacta del mismo. Debe poder modificarse los datos ingresados y baja en caso que ya no se explote.	

Historia de usuario	
Número: 9	
Usuario: SOP	
Nombre historia: ABM de servicios de mantenimiento	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Dar ingreso al sistema de los diferentes tipos de mano de obra que se realiza a los motores, pudiéndose modificar sus datos y dar de baja los mismos.	

Historia de usuario	
Número: 10	
Usuario: SOP	
Nombre historia: ABM de materiales	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Dar ingreso al sistema de los diferentes materiales que se pueden utilizar al prestar servicio a un motor, poder modificar sus datos y también dar de baja los mismos.	

Historia de usuario	
Número: 11	
Usuario: SOP	
Nombre historia: ABM de máquinas	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 1
Programador responsable: Sanchez, Enzo	
Descripción: Se debe poder cargar en el sistema una nueva máquina con datos de relevancia tales como su número de motor, la aplicación y la bomba correspondiente. Esto se debe poder modificar y dar de baja en caso de ser necesario.	

Historia de usuario	
Número: 12	
Usuario: Operario	
Nombre historia: Cargar una orden de trabajo	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 32 horas	**Iteración asignada:** 2
Programador responsable: Ayala, Cintia	
Descripción: Cargar desde una dispositivo móvil una orden de trabajo, considerando que no hay señal, ni conexión a internet.	

Historia de usuario	
Número: 13	
Usuario: SOP	
Nombre Historia: Listado de Ordenes de Trabajo según distintos criterios	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Bajo
Tiempo estimado: 20 horas	**Iteración asignada:** 2
Programador responsable: Sanchez, Enzo	
Descripción: En el sistema de gestión, el usuario puede listar todas las órdenes de trabajo que cumplan determinados criterios de búsqueda. Dentro de los posibles filtros, se encuentran: por fecha (todas las órdenes de una fecha en particular u órdenes entre dos fechas), por equipo y por MM	

Historia de usuario	
Número: 14	
Usuario: SOP	
Nombre historia: ABM de compresiones cilíndricas	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 3
Programador responsable: Sanchez, Enzo	
Descripción: Crear en el sistema un nuevo compresor, posibilidad de modificarlo y dar de baja si es necesario.	

Historia de usuario	
Número: 15	
Usuario: Operario	
Nombre historia: Toma compresiones cilíndricas de un motor desde la app	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 8 horas	**Iteración asignada:** 3
Programador responsable: Ayala, Cintia	
Descripción: Implementar en la aplicación la posibilidad de agregar una compresión a la orden de trabajo	

Historia de usuario	
Número: 16	
Usuario: Administrador	
Nombre historia: Reporte de materiales	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 8 horas	**Iteración asignada:** 3
Programador responsable: Sanchez, Enzo	
Descripción: Obtener un reporte que informe y liste los materiales y cantidad utilizados al realizar servicios, ya sea uno en específico o todos los materiales. Tener la posibilidad de filtrar un rango de fechas, equipo de trabajo y un motor mediante su identificador.	

Historia de usuario	
Número: 17	
Usuario: Administrador	
Nombre historia: Reporte servicio de motores	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 8 horas	**Iteración asignada:** 3
Programador responsable: Sanchez, Enzo	
Descripción: Obtener un reporte respecto a que motores se les prestó servicio. Debe poder filtrarse si se desea un motor particular o todos, por un rango de fechas, equipo de trabajo y por el tipo de servicio.	

Historia de usuario	
Número: 18	
Usuario: Administrador	
Nombre historia: Reporte horas trabajadas	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 8 horas	**Iteración asignada:** 3
Programador responsable: Sanchez, Enzo	
Descripción: Obtener un reporte de horas trabajadas por operario. Se debe poder filtrar operario, un rango de fechas o un motor.	

Historia de usuario	
Número: 19	
Usuario: Operario	
Nombre historia: Imágenes de orden de trabajo	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 8 horas	**Iteración asignada:** 3
Programador responsable: Ayala, Cintia	
Descripción: Debe poder sacar imágenes (3) al cargar una orden de trabajo.	

Historia de usuario	
Número: 20	
Usuario: Operario	
Nombre historia: Escaneo de código QR	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 16 horas	**Iteración asignada:** 3
Programador responsable: Ayala, Cintia	
Descripción: Poder escanear desde la aplicación un código QR que proveerá información del motor.	

Historia de usuario	
Número: 21	
Usuario: Operarios	
Nombre historia: Recesos laborales en la app	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 32 horas	**Iteración asignada:** 4
Programador responsable: Ayala, Cintia	
Descripción: Desde la aplicación se deben poder cargar, modificar o eliminar un receso en la jornada laboral, ya sea por almuerzo, mal clima, etc. Debe quedar registrada la hora de ingreso y egreso	

Historia de usuario	
Número: 22	
Usuario: SOP	
Nombre historia: ABM recesos laborales	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 4
Programador responsable: Sanchez, Enzo	
Descripción: Desde el sistema se debe poder cargar, modificar o eliminar un receso laboral	

Historia de usuario	
Número: 23	
Usuario: Operario	
Nombre historia: Tareas de almacén en la app	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 16 horas	**Iteración asignada:** 4
Programador responsable: Ayala, Cintia	
Descripción: Desde la aplicación se deben poder cargar, modificar o eliminar una tarea realizada en almacén, ya sea para dar ingreso a materiales, descarga de aceite usado, etc.	

Historia de usuario	
Número: 24	
Usuario: SOP	
Nombre historia: ABM tareas de almacén	
Prioridad: Media	**Riesgo en desarrollo:** Baja
Tiempo estimado: 4 horas	**Iteración asignada:** 4
Programador responsable: Sanchez, Enzo	
Descripción: Desde el sistema se debe poder cargar, modificar o eliminar una tarea de almacén	

Historia de usuario	
Número: 25	
Usuario: Administrador	
Nombre historia: Parte Diario de Mantenimiento	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 8 horas	**Iteración asignada:** 4
Programador responsable: Sanchez, Enzo	
Descripción: Informe por equipo con información concisa de la jornada laboral.	

Historia de usuario	
Número: 26	
Usuario: SOP	
Nombre historia: Listar/Modificar/Eliminar Jornada	
Prioridad: Alta	**Riesgo en desarrollo:** Baja
Tiempo estimado: 8 horas	**Iteración asignada:** 4
Programador responsable: Sanchez, Enzo	
Descripción: Desde el sistema se debe poder agregar, modificar o eliminar cualquier información de una jornada de un equipo. Implica una orden de trabajo, recesos laborales, tareas de almacén, etc.	

Anexo 7: Tarjetas CRC

En este anexo se detallan todas las tarjetas CRC que se diagramaron en el proyecto (incluidas la de Equipo y Orden en el capítulo 4).

Aplicación	
Crear Aplicación **Mostrar Aplicaciones** **Mostrar una Aplicación en particular** **Modificar una Aplicación** **Discontinuar una Aplicación** **Eliminar una Aplicación**	

BombaAIB	
Crear Bomba AIB **Mostrar Bombas AIB** **Mostrar una Bomba AIB en particular** **Modificar una Bomba AIB** **Discontinuar una Bomba AIB** **Eliminar una Bomba AIB**	

Compresión	
Crear Compresión **Mostrar Compresiones** **Mostrar una Compresión en particular** **Modificar una Compresión** **Discontinuar una Compresión** **Eliminar una Compresión**	

Equipo	
Mostrar Equipos **Crear Equipos** **Mostrar un Equipo en particular** **Modificar un Equipo** **Discontinuar un Equipo** **Eliminar un Equipo**	Operario

Jornada	
Crear Jornada **Mostrar Jornadas** **Mostrar una Jornada en particular** **Modificar una Jornada** **Eliminar una Jornada**	Equipo Material Movil RecesoLaboral RecepcionMaterial TareaAlmacen

Maquina	
Crear Maquina	Aplicacion
Mostrar Maquinas	BombaAIB
Mostrar una Maquina en particular	RecesoLaboral
Modificar una Maquina	TareaAlmacen
Discontinuar una Maquina	
Eliminar una Maquina	

Material	
Crear Material	
Mostrar Materiales	
Mostrar un Material en particular	
Modificar un Material	
Discontinuar un Material	
Eliminar un Material	

Movil	
Crear Móvil	
Mostrar Móviles	
Mostrar un Móvil en particular	
Modificar un Móvil	
Discontinuar un Móvil	
Eliminar un Móvil	

Operario	
Crear Operario	
Mostrar Operarios	
Mostrar un Operario en particular	
Modificar un Operario	
Discontinuar un Operario	
Eliminar un Operario	

Orden	
Crear una Orden de Trabajo	Compresión
Mostrar Ordenes de Trabajo	Jornada
Mostrar una Orden de Trabajo en particular	Máquina
Modificar una Orden de Trabajo	Material
Eliminar una Orden de Trabajo	Pozo
	Servicio

Pozo	
Crear Pozo **Mostrar Pozos** **Mostrar un Pozo en particular** **Modificar un Pozo** **Discontinuar un Pozo** **Eliminar un Pozo**	Zona

RecesoLaboral	
Crear Receso Laboral **Mostrar Recesos Laborales** **Mostrar un Receso Laboral en particular** **Modificar un Receso Laboral** **Eliminar un Receso Laboral**	

Servicio	
Crear Servicio **Mostrar Servicios** **Mostrar un Servicio en particular** **Modificar un Servicio** **Discontinuar un Servicio** **Eliminar un Servicio**	

TareasAlmacen	
Crear Tarea Almacén **Mostrar Tareas Almacén** **Mostrar una Tarea Almacén** **Modificar una Tarea Almacén** **Eliminar una Tarea Almacén**	

Zona	
Crear Zona **Mostrar Zonas** **Mostrar una Zona** **Modificar una Zona** **Discontinuar una Zona** **Eliminar una Zona**	

MainActivity	
Listar Funcionalidades	

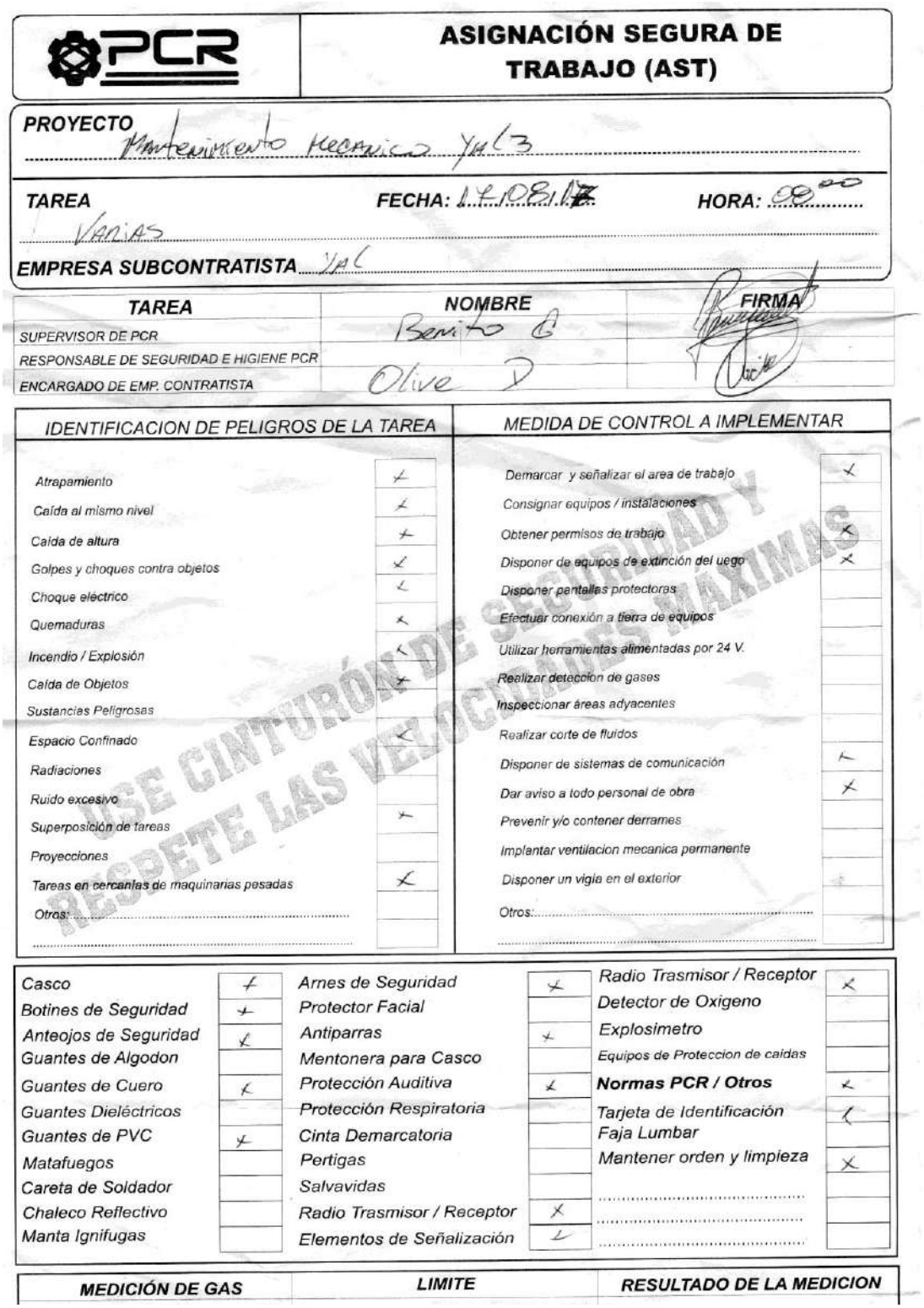
DescargarNovedades	
Descargar Archivos	AsyncTask

EnviarNovedades	
Enviar Archivos	AsyncTask

MenuOT	
Scannear QR	Orden

Anexo 8: Documentos en papel

En este anexo se incorporan imágenes de los documentos en papel involucrados en el circuito administrativo y operativo de la empresa YAL S.A y la empresa contratante.



PCR

ASIGNACIÓN SEGURA DE TRABAJO (AST)

PROYECTO Mantenimiento Mecanico YAL 3

TAREA VARIAS **FECHA:** 17/08/17 **HORA:** 09:00

EMPRESA SUBCONTRATISTA YAL

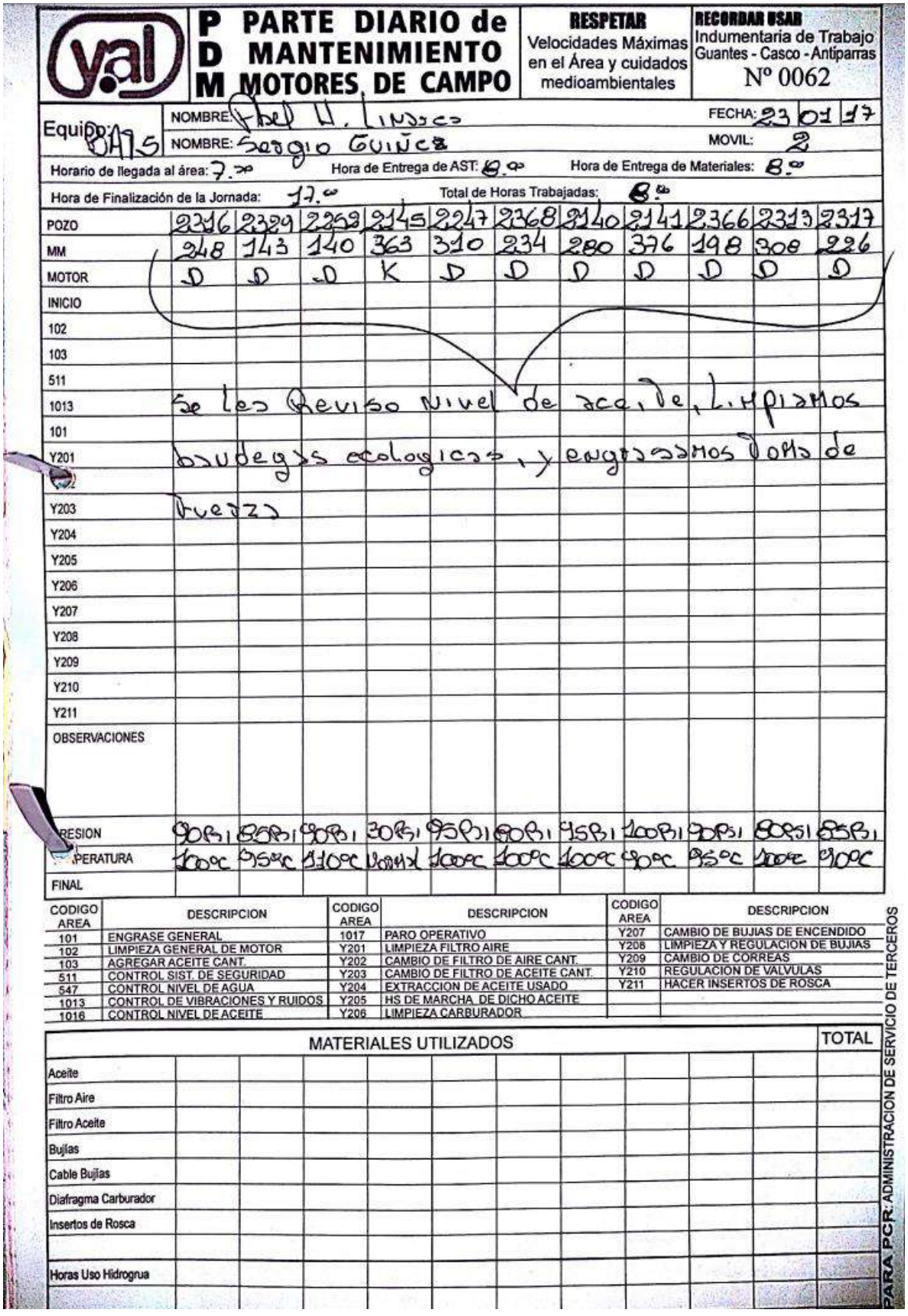
TAREA	NOMBRE	FIRMA
SUPERVISOR DE PCR	Benito C	
RESPONSABLE DE SEGURIDAD E HIGIENE PCR		
ENCARGADO DE EMP. CONTRATISTA	Olive D	

IDENTIFICACION DE PELIGROS DE LA TAREA		MEDIDA DE CONTROL A IMPLEMENTAR	
Atrapamiento	✓	Demarcar y señalizar el area de trabajo	✓
Caída al mismo nivel	✓	Consignar equipos / instalaciones	
Caida de altura	✓	Obtener permisos de trabajo	✓
Golpes y choques contra objetos	✓	Disponer de equipos de extinción del uego	✓
Choque eléctrico	✓	Disponer pantallas protectoras	
Quemaduras	✓	Efectuar conexión a tierra de equipos	
Incendio / Explosión	✓	Utilizar herramientas alimentadas por 24 V.	
Caída de Objetos	✓	Realizar deteccion de gases	
Sustancias Peligrosas		Inspeccionar áreas adyacentes	
Espacio Confinado	✓	Realizar corte de fluidos	
Radiaciones		Disponer de sistemas de comunicación	✓
Ruido excesivo		Dar aviso a todo personal de obra	✓
Superposición de tareas	✓	Prevenir y/o contener derrames	
Proyecciones		Implantar ventilacion mecanica permanente	
Tareas en cercanías de maquinarias pesadas	✓	Disponer un vigia en el exterior	
Otros:		Otros:	

Casco	✓	Arnes de Seguridad	✓	Radio Trasmisor / Receptor	✓
Botines de Seguridad	✓	Protector Facial		Detector de Oxigeno	
Anteojos de Seguridad	✓	Antiparras	✓	Explosimetro	
Guantes de Algodon		Mentonera para Casco		Equipos de Proteccion de caidas	
Guantes de Cuero	✓	Protección Auditiva	✓	**Normas PCR / Otros**	✓
Guantes Dieléctricos		Protección Respiratoria		Tarjeta de Identificación	✓
Guantes de PVC	✓	Cinta Demarcatoria		Faja Lumbar	
Matafuegos		Pertigas		Mantener orden y limpieza	✓
Careta de Soldador		Salvavidas			
Chaleco Reflectivo		Radio Trasmisor / Receptor	✓		
Manta Ignifugas		Elementos de Señalización	✓		

MEDICIÓN DE GAS	LIMITE	RESULTADO DE LA MEDICION

Imagen 87: Asignación Segura de Trabajo



P PARTE DIARIO de
D MANTENIMIENTO
M MOTORES DE CAMPO

RESPETAR Velocidades Máximas en el Área y cuidados medioambientales

RECORDAR USAR Indumentaria de Trabajo Guantes - Casco - Antiparras

Nº 0062

Equipo: BA15

NOMBRE: Abel H. Linzoc

NOMBRE: Sergio Guiñez

FECHA: 23 01 17

MOVIL: 2

Horario de llegada al área: 7.00 Hora de Entrega de AST: 9.00 Hora de Entrega de Materiales: 8.00

Hora de Finalización de la Jornada: 17.00 Total de Horas Trabajadas: 8.00

POZO	2316	2329	2252	2145	2247	2368	2140	2141	2366	2313	2317
MM	248	143	140	363	310	234	280	376	198	308	226
MOTOR	D	D	D	K	D	D	D	D	D	D	D
INICIO											
102											
103											
511											
1013	Se les Revisa Nivel de Aceite, Limpiamos										
101											
Y201	bandejas ecologicas, y engrasamos bomba de										
Y202											
Y203	fuerza										
Y204											
Y205											
Y206											
Y207											
Y208											
Y209											
Y210											
Y211											
OBSERVACIONES											
PRESION	90PSI	85PSI	90PSI	30PSI	95PSI	90PSI	95PSI	100PSI	90PSI	90PSI	85PSI
TEMPERATURA	100°C	95°C	110°C	Normal	100°C	100°C	100°C	90°C	95°C	100°C	90°C
FINAL											

CODIGO AREA	DESCRIPCION	CODIGO AREA	DESCRIPCION	CODIGO AREA	DESCRIPCION
101	ENGRASE GENERAL	1017	PARO OPERATIVO	Y207	CAMBIO DE BUJIAS DE ENCENDIDO
102	LIMPIEZA GENERAL DE MOTOR	Y201	LIMPIEZA FILTRO AIRE	Y208	LIMPIEZA Y REGULACION DE BUJIAS
103	AGREGAR ACEITE CANT.	Y202	CAMBIO DE FILTRO DE AIRE CANT.	Y209	CAMBIO DE CORREAS
511	CONTROL SIST. DE SEGURIDAD	Y203	CAMBIO DE FILTRO DE ACEITE CANT.	Y210	REGULACION DE VALVULAS
547	CONTROL NIVEL DE AGUA	Y204	EXTRACCION DE ACEITE USADO	Y211	HACER INSERTOS DE ROSCA
1013	CONTROL DE VIBRACIONES Y RUIDOS	Y205	HS DE MARCHA DE DICHO ACEITE		
1016	CONTROL NIVEL DE ACEITE	Y206	LIMPIEZA CARBURADOR		

MATERIALES UTILIZADOS											TOTAL
Aceite											
Filtro Aire											
Filtro Aceite											
Bujías											
Cable Bujías											
Diafragma Carburador											
Insertos de Rosca											
Horas Uso Hidrogrua											

PARA P.C.R: ADMINISTRACION DE SERVICIO DE TERCEROS

Imagen 88: Orden de Trabajo

Equipo (3)

PDMM **PARTE DIARIO de MANTENIMIENTO MECANICO**

RESPETAR Velocidades Máximas en el Área y cuidados medioambientales

RECORDAR USAR Indumentaria de Trabajo - Guantes Casco - Antiparras

Nº 0905

FECHA: 21 / 03 / 17

NOMBRE: Victor Cerna. Mov. 13

NOMBRE: Cristian Coli

Pozo	M.M.	Observaciones	Inicio	Final
		se acondiciona movil se retira Ast. Manceri: cargar lubes de gas y lucha comp.	8:00	9:00
2117	240	pozo 12 3cil. Limpia bandeja cabezal lucha comp. Agrega Aceite 23lts. saca perdida de aceite.	9:00	10:40
2493	384	pozo 4 cil limpia bateria. Engrasa tensor correa. Cambia tubo de gas.	10:40	13:00
		Almuerzo	13:00	14:00
2102	366	Pozo lubo. limpieza cabezal engrasa bancadas. Aceite 4Lts	14:00	15:30
2101	318	Pozo lubo. limpia cabezal engrasa bancadas. Agrega Aceite 3Lts	15:30	17:00

(M331304051)101

Horas . Normales	
Horas al 50 %	
Horas al 100 %	
TOTAL de Horas	8hs

PCR 29 MAR 2017 RECIBIDO

Agosto 2016 - DEL 661-1150 - ORIGINAL-BLANCO DUPLICADO-COLOR TRIPLICADO-COLOR

Imagen 89: Parte Diario de Mantenimiento de Motores de Campo

Anexo 9: Estándares

A continuación se detallan los estándares utilizados por el equipo de desarrollo, respondiendo a una serie de buenas prácticas recomendadas para los lenguajes Java y PHP.

Si bien se siguieron muchas reglas, también se implementaron otras nuevas, buscando facilitar la compresión y lograr la propiedad colectiva del código.

9.1 Estándares en el desarrollo del sistema SGPDM

- Toda clase inicia con mayúscula
- Todo método inicia con minúscula
- Toda variable inicia con minúscula
- Sea clase o método, si consta de dos o más palabras todas van sin espacios y de la segunda en adelante con mayúscula inicial.
- Las variables si constan de dos o más palabras van separadas por guion bajo (_)
- Las variables que son arrays (arreglos) comienzan con el prefijo “arr”.
  Ejemplo:
  $arr_materiales
- Las llaves de apertura de las clases se escriben en la línea siguiente.
- Las llaves de cierre se escriben en la línea siguiente al cuerpo de la clase
  Ejemplo:

```
class Operario extends Models
{
        /* cuerpo de la clase */
}
```

- La llave de apertura de métodos y estructuras de control van en la misma línea que el nombre del método o estructura. Se comienza a escribir en línea inferior
- La llave de cierre de los métodos y estructura de control se escriben solas en la última línea
  Ejemplo:

```
public function reporteMaquinas{
        /* cuerpo del método */
}
```

- Todo el código debe estar identado
  /**********/

9.2 Estándares en el desarrollo de la aplicación PDMApp

- Todo paquete inicia con minúscula
- Toda actividad inicia con mayúscula. Si consta de más de una palabra, se coloca una mayúscula al inicio de la misma y no se dejan espacios ni caracteres especiales
- Todo método inicia con minúscula y al igual que la clase si consta de dos o más palabras, se coloca una mayúscula inicial
- Todo archivo .xml creado en la ruta res/layout que define la arquitectura de la UI de una actividad/fragmento se escriben con minúscula y con la siguiente estructura:

layout_nombredeactividad.xml
fragment_nombredefragmento.xml
Por ejemplo:
layout_tareasalmacen.xml
fragment_ot1.xml
A diferencia de activity_main que está creada por defecto.

- Todo archivo .xml creado en la ruta res/drawable que indica el estilo de un componente UI se escribe todo en minúscula y con el formato:
  La palabra "style" + el componente UI al que le estamos dando estilo y el atributo en particular que estemos modificando.
  style_componente_atributo.xml
  Por ejemplo:
  style_edittext_border.xml
  style_spinner_background.xml

- Las imágenes que se encuentran en la ruta **res/drawable** conservan la misma extensión (.png). Sus nombres están en minúscula y sin espacios.
- Los comentarios en un método, en el caso que algo no sea intuitivo se antepone doble barra "//"
- Los comentarios antes de una clase /** comentario */

- Todo el código debe estar correctamente indentado

- Nombres de variables
  Todo elemento de componente visual comienza con un conjunto de letras indicando el componente que representa.
  Por ejemplo:
  -spTipoOt --> Spinner
  -edPozo --> EditText
  -tvRta --> TextView
  -btnActualizar --> Button
  -ivFuncion --> ImageView
  -lvFotos --> ListView