



**UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN**

**DESARROLLO DEL MÓDULO PRINCIPAL Y ESTADÍSTICAS DE LA
LIBRERÍA AUDITORÍAS TURPIAL**

Por:
Stefani Carolina Castellanos Torres

INFORME DE PASANTÍA

Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de la Computación

Sartenejas, Enero 2018



**UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN**

**DESARROLLO DEL MÓDULO PRINCIPAL Y ESTADÍSTICAS DE LA
LIBRERÍA AUDITORÍAS TURPIAL**

Por:
Stefani Carolina Castellanos Torres

Realizado con la asesoría de:
Tutor Académico: Prof. Angela Di Serio
Tutor Industrial: Ing. Pedro Romero

INFORME DE PASANTÍA
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de la Computación

Sartenejas, Enero 2018



UNIVERSIDAD SIMÓN BOLÍVAR
VICERRECTORADO ACADÉMICO
DECANATO DE EXTENSIÓN
Coordinación de Cooperación Técnica

79e750ca69

ACTA DE EVALUACIÓN


Período: Julio-Diciembre 2017 **Fecha:** 15/01/2018 **Tipo Pasantía:** Pasantía larga EP-3420

Título: Desarrollo del módulo principal y estadísticas de la librería Auditorias Turpial

Nombres y Apellidos: Stefani Carolina Castellanos Torres **CI::** 25385981

Carnet: 11-11394 **Carrera:** Ingeniería de la Computación 0800

APROBADO: ☒ **REPROBADO:** ☐

JURADO EXAMINADOR		
TUTOR ACADÉMICO Nombre: Angela Di Serio	Firma: <i>Angela Di Serio</i> CI : 6544924	Sello del Departamento de Computación y Tecnología de la Información
TUTOR INDUSTRIAL Nombre: Pedro Romero	Firma: <i>Pedro Romero</i> CI : 19045406	Sello de la Coordinación de Carrera
JURADO Nombre: Kity Alvarez	Firma: <i>Kity Alvarez</i> CI: 10829324	Sello del Departamento de Computación y Tecnología de la Información
Art. 17. Reglamento C.4 Cursos en Cooperación: Cuando la evaluación global del tutor industrial de la pasantía sea igual o superior a 4,5 puntos en la escala de 5, el Jurado Examinador podrá reconocer el trabajo aprobado como SOBRESALIENTE , justificándolo en forma razonada en el Acta:		
Justificación: <i>El trabajo cumple cabalmente con los tiempos establecidos. El trabajo excede en cuanto a su alcance y resultados y contiene elementos que califican como aporte susceptibles de publicación</i>		Validación de CCT Firma y Sello 
Firmas Jurado Examinador: <i>Kity Alvarez</i> <i>Pedro Romero</i> <i>Angela Di Serio</i>		

Nota: Este documento no puede llevar tachaduras ni enmiendas y todas las firmas del Jurado Examinador deben estar acompañadas de sus correspondientes sellos en húmedo. En caso de inasistencia del tutor industrial, se debe colocar **NO-PRESENTE** en el espacio reservado para su firma. La evaluación por parte del tutor industrial estará documentada mediante los respectivos procedimientos que establezca el Decanato de Extensión. El Jurado Examinador firmará nuevamente en la zona de justificación al otorgar la mención SOBRESALIENTE.

Puerta de Comunidad a Comunidad

Sartenejas, Baruta, Edif. Comunicaciones Telf: (0212) 906.3157al 64 Apartado postal 89000, Zip Code 1080-A. www.usb.ve



UNIVERSIDAD SIMÓN BOLÍVAR

VICERRECTORADO ACADÉMICO
DECANATO DE ESTUDIOS PROFESIONALES
Coordinación de Ingeniería de la Computación

ACTA DE EVALUACION DEL INFORME DE PASANTIA LARGA

Código de la asignatura: EP3420

Fecha: 12-01-2018

Nombre del estudiante: Stefani Carolina

Carnet: 11-11394

Castellanos Torres

Título del proyecto: Desarrollo del módulo Principal y Estadísticas de la librería Auditorías Turpial

Tutor: Prof. Angela Di Serio

Jurados: Profesores Kity Álvarez

Nosotros, miembros del Jurado designado por la Coordinación Docente de Ingeniería de la Computación, en la fecha indicada para la evaluación del Proyecto de Pasantía Larga en cuestión, hemos evaluado el contenido de la misma y hacemos constar nuestra decisión de **aceptar el documento para su presentación en acto público**. Se convoca al estudiante, Stefani Carolina Castellanos Torres, a la presentación pública del Proyecto de Pasantía Larga "Desarrollo del módulo Principal y Estadísticas de la librería Auditorías Turpial" en el edificio "MyS" sala "Sala de coloquios del Departamento".

Fecha: 15/11/2018

Hora: 10:30 am

Lugar: Sede de Sartenejas

Prof. Kity Álvarez
Jurado (Presidente)
C.I. 10.829.324

Prof. Angela Di Serio
Tutor Académico
C.I. 6.544.924



Nota: Colocar los sellos de los respectivos Departamentos Académicos. Para jurados externos usar el sello de la Coordinación Docente. Este documento debe entregarse en original y sin enmiendas.



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

DESARROLLO DEL MÓDULO PRINCIPAL Y ESTADÍSTICAS DE LA
LIBRERÍA AUDITORÍAS TURPIAL

INFORME DE PASANTÍA LARGA

Realizado por: Stefani Carolina Castellanos Torres
Con la asesoría de: Prof. Angela Di Serio, Ing. Pedro Romero.

Auditorías Turpial es una librería para Django que permite mantener un historial de las operaciones realizadas sobre alguna tabla de base de datos, a través de auditorías. La empresa Turpial Development, requiere una librería de Django que le permita acelerar el desarrollo de aplicaciones que necesiten un sistema de este tipo, y esté desarrollado bajo la arquitectura de microservicios para que sea fácil de integrar. Esta librería consta de tres módulos: Principal (*Core*), Estadísticas y Reportes. Adicionalmente, estos microservicios pueden ser actualizados mediante una herramienta de Integración Continua. En el presente informe, se describen los procesos de diseño, implementación y pruebas de parte del módulo *Core* y completamente el módulo Estadísticas de Auditorías Turpial, desarrollado como pasantía larga en Turpial Development siguiendo la metodología *Turpial Agile Unified Process*. Así como, la implantación del sistema de Integración Continua utilizando Jenkins. Como resultado, se lograron todos los objetivos planteados y otros originados durante la fase de construcción.

Palabras claves: Auditoría de sistemas, librería, Django, Integración Continua, microservicio.

DEDICATORIA

*“Es mucho más difícil juzgarse a sí mismo,
que juzgar a los otros.*

*Si consigues juzgarte rectamente
es que eres un verdadero sabio.”*

- Antoine de Saint-Exupéry. El Principito

*“No vivas imaginando problemas
que no han ocurrido ni van a suceder;
disfruta del presente,
la vida hay que aprovecharla”*

- Enrique Barrios. Ami, el niño de las estrellas.

AGRADECIMIENTOS

A Dios, por llenar mi vida de oportunidades y lecciones que me han convertido en la persona que soy hoy.

A mis padres, Leslie Torres y Armando Castellanos por acompañarme desde mi nacimiento, ser mi apoyo, mis guías y mis consejeros. Por enseñarme que el trabajo duro rinde frutos y que se debe intentar ser mejor cada día. No hay suficiente espacio para expresar mi gratitud y amor por ustedes.

A mi novio, Jesús Herrera, por tantos años a mi lado llenando mi vida de felicidad. Por escucharme cuando lo necesito y darme ánimos cuando me sentí derrotada por esta carrera. La familia no es sólo de sangre y eres parte de la mía.

A mi familia, por su apoyo incondicional.

A mis amigos de la infancia, en especial a Daniela Zarzalejo y Ana De Lucca por crecer junto a mi. Más que amigas, mis hermanas.

A mis amigos de la universidad. A Arleyn Goncalves por compartir esta experiencia conmigo, este proyecto no sería posible sin tu valiosa participación. A Francisco Sucre por su ayuda durante estos años.

A mi tutor industrial, Pedro Romero, por estar siempre dispuesto a ayudarme, por preocuparse cuando me sentí agobiada y por velar por el éxito de esta pasantía.

A mi tutora académica, Angela Di Serio, por guiarme en este proceso y durante mis años en la Universidad.

A mis profesores, por ser tan exigentes y formarme tanto en el aspecto técnico como en el espiritual.

A mi alma máter, mi segundo hogar.

A Andrea Sucre, Jacobo Martínez y al todo el equipo de Turpial Development, por iniciarme en el campo profesional de la mejor manera posible y ampliar mis conocimientos.

ÍNDICE

RESUMEN	v
ÍNDICE	viii
LISTA DE TABLAS	xi
LISTA DE FIGURAS	xii
LISTA DE ABREVIACIONES	xiv
GLOSARIO	xv
INTRODUCCIÓN	1
1 ENTORNO EMPRESARIAL	3
1.1 Descripción	3
1.2 Misión	3
1.3 Visión	3
1.4 Estructura	3
2 DEFINICIÓN DEL PROBLEMA	6
2.1 Antecedentes	6
2.2 Planteamiento del problema	7
2.3 Justificación	7
2.4 Objetivo general	8
2.5 Objetivos específicos	8
3 MARCO TEÓRICO	9
3.1 Auditoría	9
3.2 Acciones auditables	9
3.3 Microservicio	9
3.4 Integración Continua	10
3.5 Pruebas automatizadas	10
3.6 Patrón Modelo-Vista-Controlador	10
3.7 Patrón Modelo-Vista-Plantilla	11
3.8 Señales	11
3.9 Mixins	12
3.10 Serializadores	12

4	MARCO TECNOLÓGICO	13
4.1	Python	13
4.2	Django	13
4.3	Pytest	14
4.4	JavaScript Object Notation (JSON)	14
4.5	PostgreSQL	14
4.6	MySQL	14
4.7	SQLite	14
4.8	Jsonfield	15
4.9	Git	15
4.10	Jenkins	15
4.11	Lenguaje de marcado de hipertexto	15
4.12	Javascript	16
4.13	jQuery	16
4.14	Amcharts	16
4.15	Datatables	16
4.16	Bootstrap	16
5	MARCO METODOLÓGICO	17
5.1	Turpial Agile Unified Process (TAUP)	17
5.1.1	Fase de Concepción	17
5.1.2	Fase de Construcción	18
5.1.3	Fase de Transición	18
6	DESARROLLO	20
6.1	Fase de concepción	20
6.1.1	Análisis de requerimientos	20
6.1.2	Adaptación de la metodología a la pasantía	22
6.1.3	Arquitectura propuesta del sistema	23
6.1.4	Diseño del módulo <i>Core</i>	24
6.1.5	Diseño del módulo Estadísticas	25
6.1.6	Propuesta para la integración continua	27
6.1.7	Plan de pruebas	28
6.1.8	Planificación del desarrollo del proyecto	29

6.2	Fase de construcción	29
6.2.1	Construcción del módulo <i>Core</i>	29
6.2.1.1	Preparación del ambiente de desarrollo	29
6.2.1.2	Estructura de la tabla de auditorías	29
6.2.1.3	Selección de un modelo auditable	30
6.2.1.4	Registro de una acción auditable	30
6.2.1.5	Instalación de la librería utilizando PIP	33
6.2.1.6	Listados de auditoría	34
6.2.1.7	Reestructuración de la arquitectura	34
6.2.1.8	Integración continua	36
6.2.1.9	Pruebas del módulo <i>Core</i>	38
6.2.1.10	Permisología	40
6.2.1.11	Análisis de desempeño	40
6.2.1.12	Prototipo de tabla de auditorías descentralizada	41
6.2.2	Construcción del módulo Estadísticas	41
6.2.2.1	Creación del formulario de filtros	42
6.2.2.2	Cálculos estadísticos	43
6.2.2.3	Construcción de gráficos	45
6.2.2.4	Plantillas del módulo Estadísticas	46
6.2.2.5	Pruebas del módulo Estadísticas	48
6.3	Fase de transición	49
6.3.1	Integración de la librería con Turpial Calendar	49
6.3.2	Puesta en producción	51
	CONCLUSIONES Y RECOMENDACIONES	52
	Conclusiones	52
	Recomendaciones	53
	REFERENCIAS	54
	APÉNDICES	55
	A MANIFIESTO ÁGIL	56
	B MANUAL DE METODOLOGÍA DE TURPIAL (RESUMEN)	58
	C RELEASE PLAN	63
	D PLAN DE PRUEBAS	74

E	RESULTADO DE LAS PRUEBAS	82
F	RESULTADO DE LAS PRUEBAS DE DESEMPEÑO	102
G	MANUAL DE AUDITORÍAS TURPIAL <i>CORE</i>	118
H	MANUAL DE AUDITORÍAS TURPIAL UI (EXTRACTO)	122
I	MANUAL DE INSTALACIÓN DE JENKINS	127
J	MANUAL PARA CREAR UNA LIBRERÍA	134
K	PROTOTIPO TABLAS DESCENTRALIZADAS	137
L	MÓDULO PERMISOLOGÍA	140
M	COMPARACIÓN DE LIBRERÍA AUDITORÍAS TURPIAL CON OTRAS	144

LISTA DE TABLAS

6.1	Comparación de Amcharts vs Charts.js	26
6.2	Comparación de Jenkins vs Fabric.	27
6.3	Modelo de datos de la tabla “AuditableAction”	30

LISTA DE FIGURAS

1.1	Diagrama de la estructura de Turpial Development	4
3.1	Diagrama del patrón MVC.	11
3.2	Diagrama del patrón MVT.	11
6.1	Arquitectura de la librería Auditorías Turpial	23
6.2	Arquitectura de tecnologías a utilizar	24
6.3	Modelo Entidad-Relación del ejemplo.	32
6.4	Gráfico de flujo del orden en el que se activan las señales para guardar un cambio en los campos “ManyToManyField”.	32
6.5	Arquitectura final de la librería.	35
6.6	Estructura del módulo <i>Core</i> en el patrón MVT.	36
6.7	Modelo Entidad-Relación de la base de datos de prueba.	38
6.8	Ejemplo del formato JSON para cargar un <i>Fixture</i>	39
6.9	Estructura del formulario de filtros.	43
6.10	Ejemplo de “stats”.	44
6.11	Definición de “StatisticsView” incluyendo el campo “charts_theme”.	48
6.12	Estructura del módulo Estadísticas.	48
6.13	Diagrama Entidad-Relación de los nuevos modelos en las pruebas.	50

LISTA DE ABREVIACIONES

AJAX	<i>Asynchronous JavaScript And XML</i> . JavaScript asíncrono y XML.
API	<i>Application Programming Interface</i> . Interfaz de programación de aplicaciones.
CSV	<i>Comma Separated Values</i> . Archivo de valores separados por comas.
HTML	<i>HiperText Markup Language</i> . Lenguaje de marcado de Hipertexto
HTTP	<i>Hypertext Transfer Protocol</i> . Protocolo de Transferencia de Hipertexto
INVEST	<i>Independent Negotiable Valuable Estimable Small Testable</i> . Independiente Negociable Estimable Small Testable.
MVC	<i>Model-View-Controller</i> . Modelo-Vista-Controlador.
MTV	<i>Model-Template-View</i> . Modelo-Plantilla-Vista.
PIP	<i>Pip Installs Packages</i> . Pip instala paquetes.
PDF	<i>Portable Document Formats</i> . Formato de documento portátil.
SVG	<i>Scalable Vector Graphics</i> . Gráficos Vectoriales Escalables.
URL	<i>Uniform Resource Locator</i> . Localizador Uniforme de Recursos

GLOSARIO

AJAX: técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente y mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Backend: parte de las aplicaciones que procesa la entrada desde la interfaz de usuario.

Callback: cualquier código ejecutable que es suministrado como argumento a otro código. Este es ejecutado en cualquier momento.

Candlestick: tipo de gráfico financiero que es utilizado para describir la fluctuación de los precios.

Fixture: colección de datos que pueden ser utilizadas para poblar la base de datos.

Framework: conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Frontend: parte del software que interactúa con los usuarios.

In-Process: programa ejecutable que funciona como servicio para otro en lugar de la versión instalable. Ejemplo: DLL.

Look-and-feel: aspecto del sitio para el usuario y lo que siente cuando él al interactuar con el mismo.

PIP: repositorio de software (paquetes) para el lenguaje de programación Python.

Plugin: aplicación o programa que se relaciona con otro para agregarle una función nueva específica.

Portable: aplicación informática que puede ser utilizada, sin instalación previa, en un ordenador que posea el sistema operativo para el que fue programada.

Query: consulta realizada contra una base de datos. Se usa para obtener datos, modificarlos o bien borrarlos.

Queryset: en Django, representa una colección de objetos de la base de datos. Puede tener uno o muchos filtros. En términos de SQL, es equivalente a una sentencia SELECT y los filtros equivalen a la cláusula WHERE.

Responsive: filosofía de diseño y desarrollo cuyo objetivo es adaptar la apariencia de las páginas web al dispositivo que se esté utilizando para visitarlas.

Sass: un metalenguaje de Hojas de Estilo en Cascada (CSS).

Scatter: diagrama matemático que utiliza coordenadas cartesianas para mostrar valores, típicamente de un conjunto de datos con dos variables.

Script: archivo de texto, generalmente ligero, que es ejecutado en tiempo real por un intérprete.

Sprints: bloques temporales cortos y fijos (iteraciones). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto que sea potencialmente entregable.

Triggers: acciones (funciones) que se ejecutan cuando sucede algún evento sobre las tablas de una base de datos, a las que se encuentra asociado.

Webhook: método de alteración del funcionamiento de una página o aplicación web con *callbacks* personalizados.

INTRODUCCIÓN

Con el auge del desarrollo de *software*, surgieron entes que controlan y supervisan el uso de la información y garantizan la confiabilidad de los mismos, por lo que algunas aplicaciones requieren mecanismos para registrar las operaciones ejecutadas. También se pueden utilizar estas trazas para caracterizar el comportamiento del programa a desarrollar.

La gran cantidad de clientes que necesitan un servicio de este tipo, ha llevado a la empresa Turpial Development a querer agilizar el desarrollo de un proyecto con estas características. Para esto, se han valido del uso de librerías de Django, sin embargo, ninguna de ellas cumple sus expectativas.

Motivado por esto, la empresa decidió construir una librería que lleve las trazas de auditorías del sistema en el cuál se instale. El objetivo de esta pasantía es construir parcialmente el módulo Principal (*Core*) y completamente el módulo Estadísticas de la librería Auditorías Turpial.

Por otro lado, la empresa considera que esta librería debe utilizar una arquitectura de microservicios que facilite la inclusión de la misma, y que posea un servidor de Integración Continua. Este último con la intención de que la librería se encuentre disponible la mayoría del tiempo, sin afectar a los que la usen.

El presente informe está conformado por seis capítulos:

- En el capítulo 1 se describe el entorno empresarial en donde se desarrolla el proyecto.
- En el capítulo 2 se presenta el problema y los objetivos de la pasantía.
- En el capítulos 3 se describen los aspectos teóricos relacionados con el proyecto que son indispensables para la comprensión y desarrollo del mismo.
- En el capítulo 4 se detallan los aspectos tecnológicos que permiten que el proyecto sea llevado a cabo.
- En el capítulo 5 se muestra la metodología utilizada en la empresa, que fue creada por ellos mismos.
- En el capítulo 6 se explica el proceso de ejecución de la pasantía, así como la evolución del producto y las pruebas realizadas.

Luego, se ofrecen las conclusiones y recomendaciones que podría tomar en cuenta la empresa. Finalmente, se presentan todos los apéndices necesarios para profundizar sobre la información relevante del proyecto.

CAPÍTULO 1

Entorno empresarial

En este capítulo se describe en ambiente de la empresa Turpial Development, en la cual se llevó a cabo la pasantía. Se presenta la misión, visión y estructura de la empresa, así como el cargo desempeñado por el pasante durante este período.

1.1 Descripción

Turpial Development es una empresa mediana, con cuatro años en el mercado que está enfocada en el desarrollo de sistemas y aplicaciones web y móviles. Fundada e integrada por jóvenes venezolanos, ofrece soluciones que cumplen con altos estándares de usabilidad, diseño y funcionalidad [27].

1.2 Misión

La empresa tiene como misión “prestar servicios y consultoría en diseño y desarrollo de soluciones web, a la medida del cliente, caracterizadas por una alta calidad, excelente soporte y experiencia de usuario” [27] .

1.3 Visión

Su visión es “servir de plataforma para el desarrollo y éxito de nuevos emprendimientos en el área web” [27].

1.4 Estructura

En la figura 1.1 se muestra la estructura organizacional de Turpial Development, la cual está conformada por cuatro departamentos:

Dirección de Operaciones

Ente encargado del “funcionamiento de todos los procesos de soporte de la empresa, como administración, recursos humanos, contabilidad y compras con el fin de garantizar el correcto desarrollo de los procesos principales de la empresa” [27].

Dirección de Proyectos

Ente encargado de “atender a las necesidades y requerimientos de los clientes, sirviendo de enlace con las direcciones de desarrollo y diseño para garantizar la calidad del producto o entrega” [27].

Dirección de Desarrollo

Ente encargado de “la conceptualización y desarrollo de las necesidades del cliente, implementando e integrando el diseño acordado y las funcionalidades requeridas por dicho cliente. Se divide en tres departamentos: Conceptualización, *Backend* y *Frontend*” [27].

Dirección de Diseño

Ente encargado de “la conceptualización y diseño de la interfaz gráfica de los proyectos en función de las necesidades del cliente y de la mano con las decisiones tomadas por la Dirección de Desarrollo” [27].

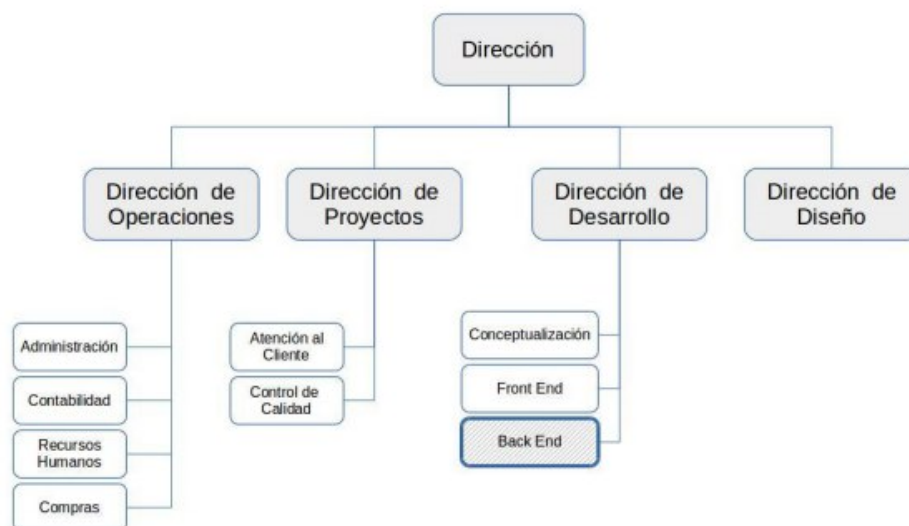


Figura 1.1: Diagrama de la estructura de Turpial Development

Ubicación del pasante

El desarrollo de la pasantía fue llevado a cabo en la Dirección de Desarrollo de la empresa. El pasante fue asignado al departamento de *Backend* con el cargo de Pasante, bajo la supervisión del Tutor Industrial, el Ingeniero Pedro Romero. El pasante debe cumplir tareas tales como: levantar requerimientos, diseñar y desarrollar las funcionalidades de la solución propuesta para el sistema.

CAPÍTULO 2

Definición del problema

En este capítulo se explica la motivación de la empresa para desarrollar el proyecto y la justificación del mismo. Además, se establecen los objetivos que se esperan alcanzar durante la pasantía.

2.1 Antecedentes

En ocasiones, las aplicaciones necesitan un sistema que controle y supervise el uso de la información, bien sea para cumplir con regulaciones o para observar el comportamiento de la información considerada sensible para el negocio. Turpial Development desea contar con una librería confiable y de fácil instalación que provea estas funcionalidades, de manera que puedan ser ofrecidas como un servicio a sus clientes. Esta librería debe poder instalarse en el *framework* escrito en Python, Django, puesto que la empresa estableció como estándar de desarrollo para sus aplicaciones estas tecnologías.

Existen algunas librería disponibles que son capaces de realizar estos procedimientos. Una de las más populares es django-reversions, una extensión de Django que provee control de versiones para los modelos y la posibilidad de restaurar la base de datos a una versión anterior. La empresa utilizó esta librería en algunos proyectos, sin embargo, su integración delega demasiada responsabilidad al programador por lo que es susceptible a errores humanos y agrega una complejidad, que en muchos casos, es innecesaria para el comportamiento que se desea lograr.

Además, de esta extensión, hay otras que no han sido utilizadas por Turpial Development (Apéndice M) que ofrecen estas características, utilizando otras funciones que provee el *framework*, sin embargo los problemas explicados anteriormente se mantienen.

2.2 Planteamiento del problema

Motivados por la cantidad de clientes que requieren un sistema de auditorías y porque la extensiones de Django no son fáciles de implementar, Turpial Development, se vio en la necesidad de crear su propia librería con la intención de agilizar el desarrollo de proyectos con estas funcionalidades. Esta permitirá llevar la trazabilidad de la información sensible del negocio a través de auditorías de manera sencilla.

Adicionalmente, ninguna de las librerías disponibles cuentan con un módulo que permita visualizar los datos obtenidos a través de estadísticas o reportes. El desarrollador debe realizar esta labor por él mismo; analizando la información recaudada y ajustando la estructura de datos para hacer uso de otras librerías que dispongan de la capacidad de realizar gráficas o generar PDF. En este sentido, se desea que el producto a desarrollar proporcione un módulo capaz de realizar los cálculos necesarios para mostrar estadísticas referentes a la información recaudada por las auditorías y sus respectivas gráficas; y un módulo que genere reportes CVS o PDF de los diferentes listados y que sean personalizables.

2.3 Justificación

La empresa Turpial Development decidió invertir en el desarrollo de una librería para la gestión de auditorías que se ajuste tanto a las necesidades de sus programadores, como a la de sus clientes; que sea fácil de integrar en otro proyecto, permita identificar cuáles acciones del sistema deben ser auditables y almacenar las auditorías en una base de datos relacional. Asimismo, debe crear listados, generar reportes, mostrar estadísticas y facilitar la personalización su apariencia sin necesidad de que el desarrollador cree las plantillas desde cero.

La empresa plantea que esta librería sea desarrollada bajo una arquitectura de microservicios. De esta manera, podrá incluirse fácil y rápido dentro de las aplicaciones, a través de configuraciones y ajustes menores que no interfieran con su correcto funcionamiento. Se desarrollará como una aplicación en Django que se agregará como un módulo más al sistema que lo instale.

Adicionalmente, se desea utilizar alguna herramienta automatizada para integrar el trabajo de manera continua, empaquetar, probar y desplegar la librería de manera que se actualice con rapidez y esté disponible en todo momento. De igual manera, la empresa considera que es de vital importancia realizar pruebas automatizadas que aseguren la calidad de sistema que se desea entregar y que las pruebas finales se realicen sobre un proyecto de

uso interno.

2.4 Objetivo general

Implementar, probar y presentar las funcionalidades de selección, gestión y listados de auditorías y el módulo Estadísticas de la librería de Auditorías Turpial.

2.5 Objetivos específicos

- Diseñar el módulo *Core* de la librería Auditorías Turpial y construir parcialmente de dicho módulo, específicamente las funcionalidades de selección de modelos a auditar y gestión de las auditorías, bajo una arquitectura de microservicios.
- Diseñar y construir el módulo Estadísticas de la librería Auditorías Turpial, bajo una arquitectura de microservicios.
- Implantar un sistema de integración continua con el repositorio donde se dispondrá la librería Auditorías Turpial.

CAPÍTULO 3

Marco teórico

En este capítulo se presentan los aspectos teóricos que sustentan y respaldan el desarrollo del proyecto de pasantía. A continuación se describen los conceptos que permiten explicar el problema planteado.

3.1 Auditoría

Es una revisión sistemática para determinar si la calidad de las actividades cumplen con los acuerdos planificados y si estos están implementados efectivamente y son adecuados para alcanzar sus objetivos. Ofrecen una oportunidad de mejora al sistema y pueden ser llevadas a cabo para cumplir con normas regulatorias. Las auditorías se pueden aplicar a sistemas, procesos, programas o servicios y pueden ser internas (realizadas por la misma empresa) o externas (realizadas por proveedores) [28].

3.2 Acciones auditables

Se considera una acción auditable todo flujo del sistema que cree, edite o borre alguna información sensible para el mundo de negocio. Son registradas incluyendo información sobre quién realizó la acción, qué acción se intentó realizar y cuándo ocurrió la acción.

3.3 Microservicio

Es un tipo de arquitectura que consiste en desarrollar una aplicación como un conjunto de pequeños servicios. Dichos servicios son procesos autónomos, cohesivos e independientes que suelen interactuar con otros componentes a través de mensajes [8]. Se enfocan en resolver un único problema y funcionan de manera aislada; si se presentan una falla no se propaga y puede ser atendido más rápidamente [18].

Estos servicios se manejan de manera descentralizada y hacen uso de un despliegue completamente automatizado. Cada uno de ellos pueden ser escritos en diferentes lenguajes

y tecnologías para almacenar información y aún así, interactuar y compartir información [18].

3.4 Integración Continua

Es una práctica de desarrollo de *software* en la que los miembros del equipo combinan su trabajo de forma periódica, usualmente cada día. Cada integración está verificada por una herramienta automatizada que empaquete y pruebe el código para detectar errores lo más rápido posible [11], de esta manera se puede mejorar la calidad del *software* y reducir el tiempo en validar y publicar actualizaciones.

3.5 Pruebas automatizadas

Las pruebas tienen como objetivo ejercitar el código para detectar errores y verificar que el *software* satisface los requerimientos especificados para asegurar su calidad. Estas son realizadas desde el punto de vista del usuario y las funcionalidades son probadas ingresando información y examinando la salida [9].

Esta labor puede ser larga y repetitiva, por lo que es conveniente contar con herramientas que provean métodos que faciliten el proceso de escribir y ejecutar casos de prueba y así, reducir significativamente el esfuerzo y tiempo invertido por los desarrolladores [9]. A este conjunto de casos de pruebas se le conoce como prueba automatizada.

3.6 Patrón Modelo-Vista-Controlador

Este patrón de diseño asigna objetos en una aplicación a uno de tres roles: modelo, vista o controlador y define cómo se comunican entre ellos. La colección de objetos de cada tipo puede ser referido como una “capa” [2].

- Modelo: controla el comportamiento y la data de la aplicación, responde las solicitudes de información (usualmente desde la vista) y las instrucciones de cambio de estado (usualmente desde el controlador).
- Vista: maneja la presentación de la información.
- Controlador: interpreta las entradas del usuario y le informa al modelo y/o vista para cambiar lo que sea apropiado [16].

Es importante notar que la vista y el controlador dependen del modelo, sin embargo el modelo es independiente. Esta separación permite probar el modelo aparte de la presentación visual.

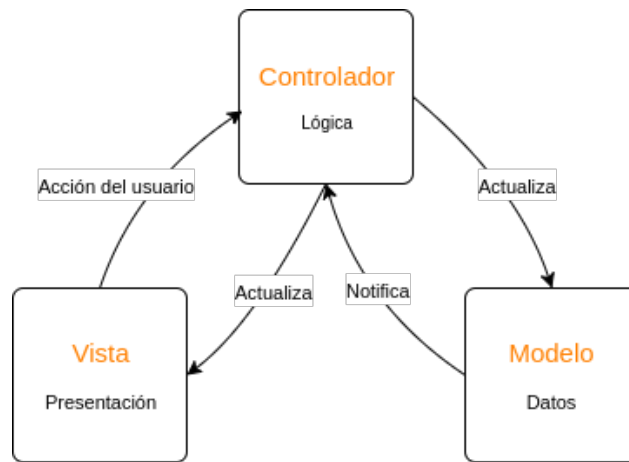


Figura 3.1: Diagrama del patrón MVC.

3.7 Patrón Modelo-Vista-Plantilla

Es una adaptación del patrón MVC, en el cual la “vista” define cuál dato es presentado y su comportamiento, no como se muestra. El dato es obtenido a través de la función de *callback* para pedir una URL en particular. Por otro lado, la “vista” delega a la “plantilla” la presentación de la información.

En el caso de Django, el “controlador” es el *framework* en sí: “la maquinaria que envía una petición a la vista apropiada, de acuerdo a la configuración de URL de Django” [6].

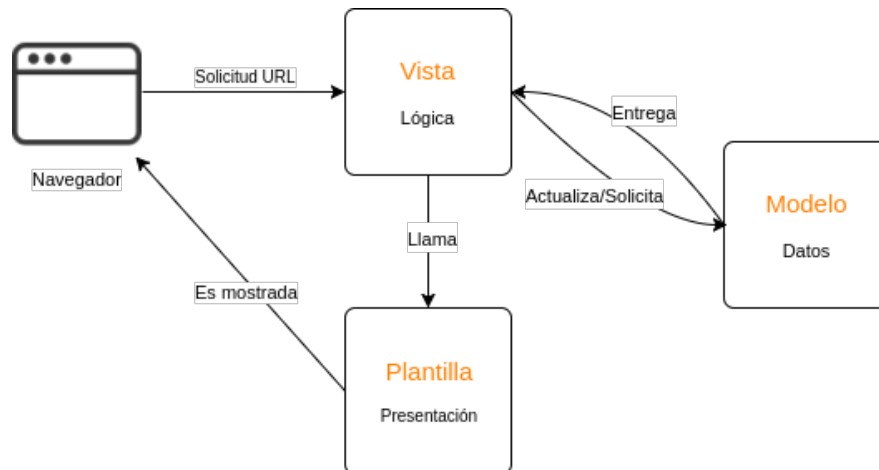


Figura 3.2: Diagrama del patrón MVT.

3.8 Señales

Permiten a las aplicaciones ser notificadas cuando ocurra una acción en algún otro lugar, es decir, las señales permiten a ciertos emisores notificar a un conjunto de receptores que alguna

acción está siendo ejecutada [7]. En términos de *software*, son el análogo a interrupciones de *hardware* [25].

3.9 Mixins

Es un estilo de programación en el cual las unidades de funcionalidad son creadas en una clase y se incorporan en otras [10]. Pueden entenderse como “un subclase abstracta que puede ser usada para especializar el comportamiento de una variedad de padres” [4].

Usualmente, los *Mixins*, definen nuevos métodos que realizan alguna acción y luego llaman a los métodos del padre correspondiente; pueden utilizarse en varias clases de la jerarquía y sin importar en qué clases sean usadas.

Hay varias razones para usar *Mixins*: extienden las clases existentes sin tener que editar, mantener o combinar el código fuente; mantienen el proyecto en componentes separados; facilitan la creación de nuevas clases con funcionalidades pre-fabricadas y que se puede combinar según sea la necesidad [10].

3.10 Serializadores

La *Serialization* es un proceso que consiste en convertir un objeto en un formato que sea fácilmente transmitido a través de la red o que pueda alojarse en un lugar de almacenamiento consistente (i.e base de datos). Los *serializers* pueden ser utilizados, simplemente, para que un objeto sea legible por humanos.

CAPÍTULO 4

Marco tecnológico

En este capítulo se describen los aspectos tecnológicos relevantes para la comprensión del proyecto, así como las herramientas utilizadas durante la pasantía.

4.1 Python

Lenguaje de programación interpretado y fácil de entender. Posee estructuras de datos de alto nivel y una aproximación sencilla al paradigma de programación orientado a objetos. Cuenta con una amplia variedad de librerías que fomentan la reutilización de código; también posee tipos de datos dinámicos y una sintaxis simple para facilitar el rápido desarrollo de aplicaciones [22]

4.2 Django

Framework de código abierto, escrito en Python y está basado en el patrón Modelo-Plantilla-Vista. Proporciona diversas funcionalidades reutilizables para desarrollar, rápidamente, aplicaciones web escalables [6].

Django está equipado con un conjunto de Clases y *Mixins* que pueden ser heredados y proveen el comportamiento estándar de alguna acción particular (i.e crear un elemento). Adicionalmente, el programador puede crear otros nuevos para extender estas funcionalidades.

También incluye un despachador de señales que ayuda a las aplicaciones desacopladas a recibir notificaciones cuando alguna acción ocurra en algún otro lugar en el *framework*. Algunos de los eventos sobre los que notifica Django son la inserción, actualización y eliminación de elementos de la base de datos y migración de la misma.

4.3 Pytest

Framework escrito en Python, que facilita la escritura de complejas pruebas de funcionalidad para aplicaciones y librerías y así, asegurar la calidad del *software* que será entregado [21]. Pytest, permite controlar la ejecución de pruebas automatizadas y comparar los resultados obtenidos con los esperados [21].

4.4 JavaScript Object Notation (JSON)

Formato para intercambiar datos, ligero, independiente del lenguaje y está basado en textos. JSON, define un pequeño conjunto de reglas para la representación de datos estructurados que sean portables [5]. También, puede representar cuatro tipos primitivos (cadenas de caracteres, números, booleanos y null) y dos tipos estructurados (objetos y arreglos) [5].

4.5 PostgreSQL

Sistema de base de datos relacional de código abierto, con una arquitectura que goza de buena reputación gracias a su confiabilidad e integridad de los datos [26]. Puede ser implantada en la mayoría de los sistemas operativos y soporta claves foráneas, conjunciones, vistas, *triggers*, procedimientos y la mayoría de los tipos de datos [26].

Este manejador de base de datos posee un tamaño ilimitado (depende del *hardware*), con un máximo de 32 TB por tabla, 1 GB por campo, entre 250 y 1600 columnas por tabla y sin límites en la cantidad de índices por tablas [26].

4.6 MySQL

Sistema de base de datos relacional de código abierto que soporta múltiples plataformas y las todas la operaciones de SQL [19]. Soporta grandes volúmenes de datos, más de 50 millones de registros; puede manejar 200.000 tablas y hasta 64 índices por tablas[19].

4.7 SQLite

Librería *in-process* que implementa un motor de base de datos SQL transaccional, de código abierto, que es autocontenido, no necesita configuración y no utiliza servidores puesto que escribe directamente a los archivos de disco [24]. Posee todas las funcionalidades de una base de datos SQL completa: múltiples tablas, índices, *triggers* y vistas [24]. Es

multi-plataformas, se puede copiar libremente los archivos entre sistemas con diferentes arquitecturas [24].

El tamaño máximo de base de datos es de 140 TB y 1 GB por fila, máximo 32767 columnas por tabla dependiendo del tipo de columna [24]. La cantidad máxima de índices y tablas está estrechamente relacionada con la cantidad máxima de páginas (un poco más de 2 billones) ya que estos ocupan al menos una página del archivo de la base de datos [24].

4.8 Jsonfield

Plugin que crea un campo en Django que permite almacenar JSON en los modelos. Este, se ocupa de la serialización y la validación del campo en cuestión [13]. Aunque PostgreSQL tiene soporte nativo de los campos tipo JSON, Jsonfield, utiliza una abstracción para asegurar la compatibilidad con el resto de los manejadores de base de datos relacionales, con los que Django tiene integración [13]. Esto es de suma importancia si se desea crear una librería. El campo se traduce a uno de tipo texto.

4.9 Git

Sistema de control de versiones de código abierto, diseñado para administrar cualquier tipo de proyecto con rapidez y eficiencia [12]. Git, dispone de facilidades para llevar el seguimiento de los cambios realizados, soportar el desarrollo no-lineal, cambiar fácilmente de contexto y realizar experimentos sin afectar las versiones entregables del proyecto [12].

4.10 Jenkins

Servidor de automatización de código abierto y de fácil instalación que puede ser utilizado para automatizar tareas relacionadas con el empaquetado, pruebas y despliegue de un *software* [14]. Puede ser utilizado como un servidor de integración continua en donde la versión más reciente del proyecto sea descargada, se ejecuten las tareas descritas anteriormente y si ocurre algún fallo se le notifique a los programadores [14]. Adicionalmente, Jenkins dispone de un gran número de *plugins*, por lo que se adapta a casi cualquier proyecto sin importar qué tecnología se estén usando [14].

4.11 Lenguaje de marcado de hipertexto

Por sus siglas en inglés, *HiperText Markup Language* (HTML), es un formato de datos simple, usado para crear documentos portables de una plataforma a otra y ha sido utilizado

ampliamente en la World Wide Web desde 1990 [3]. HTML, describe la manera en la que la información es presentada en las plantillas, usualmente una página web.

4.12 Javascript

Lenguaje interpretado, multi-paradigma y dinámico que soporta estilos de programación funcional, orientado a objetos e imperativa, así como funciones de primera clase [17]. Es comúnmente utilizado como el lenguaje de *script* para páginas web.

4.13 jQuery

Librería de JavaScript que simplifica la manipulación de documentos HTML, manejo de eventos, animaciones y AJAX [15]. Cuenta con una API fácil de usar que funciona en una gran cantidad de navegadores [15].

4.14 Amcharts

Librería de JavaScript que permite añadir fácilmente gráficos interactivos a los sitios web y aplicaciones [1]. Es compatible con todos los navegadores modernos y la mayoría de los antiguos. Posee facilidades para crear gráficos de torta, barras, línea, *candlestick*, *scatter* entre otros, y cualquier combinación de ellos, exportar e importar los datos [1]. También cuenta con *plugins* para extender su funcionalidad, es *responsive*, fácil de personalizar y soporta múltiples lenguajes [1].

4.15 Datatables

Plugin para la librería jQuery de JavaScript. Añade interacción a cualquier tabla en HTML, dispone de funciones de búsqueda, paginación, ordenamiento, filtros, entre otras [23]. Además, provee facilidades para exportar diferentes tipos de archivo como CSV y PDF e incluso imprimir el contenido de la tabla [23].

4.16 Bootstrap

Framework de código abierto para desarrollar rápidamente aplicaciones, tanto web como móviles, utilizando HTML, CSS y JavaScript [20]. También usa variables y *mixins* de Sass, tiene un sistema de cuadrícula (*grid*) *responsive*, gran cantidad de componentes pre-fabricados y poderosos plugins construidos con jQuery [20].

CAPÍTULO 5

Marco metodológico

En este capítulo se presenta la metodología utilizada para el desarrollo del proyecto, indicando sus características, fases y documentos elaborados. Además, se explican las actividades que se realizan en cada etapa y la planificación de cada una.

5.1 Turpial Agile Unified Process (TAUP)

TAUP es una metodología ágil creada por la empresa Turpial Development basada en los principios del “manifiesto ágil” que brinda a su equipo de trabajo una manera eficaz de llevar a cabo el desarrollo de *software*. En el apéndice A, se encuentra mayor detalle sobre el manifiesto ágil.

TAUP considera que la prioridad es la satisfacción del cliente mediante entregas continuas, las cuales deben realizarse en lapsos entre una semana y un mes. Está conformada por tres fases: concepción, construcción y transición. En el apéndice B se especifican los diferentes aspectos relacionados a la metodología.

5.1.1 Fase de Concepción

El objetivo de esta etapa es que el equipo de desarrollo profundice la comprensión de los requerimientos del sistema. Para ello, se escriben las Historias de Usuario (HU), establecidas por el cliente y el equipo, con su respectiva prioridad y nivel de dificultad, haciendo uso de la siguiente escala: alta, media alta, media, media baja y baja. También, el equipo, asigna *Story Points* a cada HU que ponderan el esfuerzo que se requiere para culminarla.

Las HU cuentan con criterios de aceptación bien definidos que brindan más detalle sobre los aspectos que validan su completitud y poseen pruebas de aceptación para asegurar que el producto cumpla con los estándares establecidos. Adicionalmente, deben cumplir con ciertos requisitos para que se comience a desarrollar (*Definition of Ready* o DoR) y otros para que sea considerada como culminada (*Definition of Done* o DoD).

Por último, se construye la lista de todas las Historias de Usuarios (*Backlog*) en el orden que se van a ejecutar tomando en cuenta la prioridad y el riesgo de la tarea, se elabora un calendario con los *Sprints* que sean necesarios para que todas las Historias de Usuarios sean cumplidas a cabalidad, llamado *Release Plan* (Apéndice D).

5.1.2 Fase de Construcción

El enfoque de esta fase es desarrollar todo lo planteado en la Concepción, validando así, la arquitectura planteada. La metodología se basa en *Sprints* o iteraciones que son bloques de tiempo de duración corta y fija, en los que al final se puede obtener un producto potencialmente entregable.

El primer día de cada *Sprint* se lleva a cabo la Reunión de Planificación, en la que todos los miembros del equipo revisan lo que se tiene planteado en el *Release Plan*. Además, se verifica que cada una de esas Historias de Usuario estén en estado DoR para que sean divididas en tareas y se aclare cualquier duda acerca de lo que se va a realizar.

Durante el *Sprint*, diariamente, se lleva a cabo una reunión corta llamada *Daily Stand Ups Meeting*, en la cual se discute el estado de las tareas para el momento, que se va a hacer y cuáles obstáculos se han presentado en el desarrollo de alguna HU.

El último día de cada *Sprint*, los miembros del equipo deben reunirse para realizar la Revisión de Iteración, en la cual estará presente el cliente para mostrarle todos los avances. Luego, se realiza la Reunión de Retrospectiva, en donde se discute qué hizo bien, qué se puede mejorar y a qué se compromete el equipo.

5.1.3 Fase de Transición

En esta fase se probará todo lo desarrollado en la fase de Construcción. Estas pruebas funcionales y no funcionales, verifican que el proyecto pueda ser utilizado en un ambiente de producción, además se creará un manual para el usuario, el cual le dará la capacitación necesaria al cliente para poder utilizar sin problemas el sistema. También se validará la documentación, comprobando que todo el código esté correctamente explicado para que pueda ser entendido con facilidad en caso de que sea necesario realizar cambios.

Luego que el sistema esté completo, se realiza el despliegue en el ambiente de producción. Al terminar este proceso se otorgará un tiempo prudencial al cliente para efectuar pruebas y

se llevan a cabo las correcciones pertinentes.

CAPÍTULO 6

Desarrollo

En el presente capítulo se detalla la planificación siguiendo la metodología *Turpial Agile Unified Process* (TAUP). Adicionalmente, se describe la evolución del proyecto y sus dificultades, así como las actividades realizadas para cumplir objetivos planteados y logros adicionales.

6.1 Fase de concepción

En esta sección se explican las funcionalidades de los módulos Principal (*Core*) y Estadísticas de la librería Auditorías Turpial, tomando en cuenta los requerimientos del cliente; y se muestra el diseño de la solución y planteamiento de la arquitectura. También, se elaboran los documentos según TAUP y se definen las tecnologías necesarias para el desarrollo del proyecto. Este proceso abarcó las primeras cuatro semanas de la pasantía.

6.1.1 Análisis de requerimientos

Antes de tomar alguna decisión de implementación, fue necesario establecer la tecnología a la que va dirigida el producto final y cuáles son las funcionalidades mínimas que debe poseer. En primer lugar, se decidió que se desarrollaría una librería en Django, para Django, ya que es el estándar de la empresa para el desarrollo de sus aplicaciones; y utilizaría una base de datos relacional, en particular PostgreSQL, MySQL o SQLite porque se integran fácilmente al *framework*.

En segundo lugar, se determinaron las Historias de Usuario (HU). La librería consta de tres módulos: *Core*, Estadísticas y Reportes. El líder del proyecto se encargó de las HU correspondientes al módulo *Core* y el pasante realizó el levantamiento de requerimientos del módulo Estadísticas (ver apéndice C) según las necesidades del cliente. Adicionalmente, elaboró los documentos correspondientes, Documento de Requerimientos y *Release Plan*, siguiendo las plantillas de la empresa.

En este caso en particular, el rol del cliente lo interpretó la empresa misma, puesto que el producto será ofrecido como un servicio a clientes externos. El rol de *product owner* lo desempeñó el tutor industrial para gestionar el desarrollo de la pasantía.

Para escribir las HU, es indispensable contar con el actor que ejecuta alguna acción específica, por lo que se distinguieron dos tipos de usuarios:

- El programador, quien descargará la librería y la incluirá en la aplicación de Django que está desarrollando.
- Los “usuarios” finales, quienes utilizarán la aplicación en donde se instale la librería y la interfaz gráfica provista.

En líneas generales, el módulo *Core* debe contar con las siguientes características:

- Seleccionar cuál modelo (tabla) es auditable.
- Registrar el autor, acción, fecha, estado anterior y estado actual de una instancia particular en formato JSON. Las acciones auditables son: Crear, Actualizar y Eliminar.
- Listar todas las operaciones, filtrarlas y ordenarlas.
- Restringir el acceso del personal no autorizado a los listados.
- Proveer etiquetas personalizadas para las plantillas de los listados que faciliten la inclusión de los mismos.

El módulo Estadísticas debe proveer el cálculo del total de auditorías, cantidad de modelos auditables, porcentaje de cobertura, porcentaje de crecimiento de los datos, promedio por día, mínimo y máximo. Dichos resultados pueden estar filtrados por un rango de tiempo, por acción, por autor y por modelo. Asimismo, debe contar con facilidades para incluir los gráficos que representen los cálculos mencionados anteriormente.

El módulo de Reportes ofrece la posibilidad de generar archivos sobre los listados en diversos formatos (CSV y PDF) y personalizar su apariencia con opciones como modificar los márgenes, espacios, incluir el nombre y logo del sistema, entre otros.

Esta librería debe ser mantenible, eficiente, simple, confiable, escalable y fácil de integrar y configurar. Es indispensable que se instalen automáticamente sus dependencias en el sistema que la utilice para facilitar su uso y evitar errores. También, se requiere que la

librería se actualice mediante el uso de una herramienta de integración continua.

En esta pasantía se abarcarán las funcionalidades correspondientes a la selección del modelo auditable, registro de traza de auditoría y listados (sin filtros ni ordenamiento) del módulo *Core* y el módulo Estadísticas con sus respectivas pruebas automatizadas. También se incluye la instalación y configuración del sistema de integración continua. El módulo de Reportes está fuera del alcance.

6.1.2 Adaptación de la metodología a la pasantía

En el capítulo anterior se explicó la metodología TAUP, sin embargo, dependiendo del proyecto que se desea desarrollar, se pueden realizar algunas modificaciones que mejoren la dinámica y la velocidad del equipo o porque el cliente así lo requiera.

Se ideó un código compuesto por una letra y un número que facilita referenciar las HU. La letra representa el módulo a la que pertenece, *Core* o Estadísticas, y el número denota el orden en que fue concebida. Adicionalmente, se utilizó una modificación para las escalas de prioridad y riesgo, que está conformada por tres valores: alta, media y baja. Para más información sobre las HU desarrolladas, leer el Apéndice C.

Por otro lado, se agregaron nuevos criterios a la *Definition of Ready*, con lo que se tiene lo siguiente:

- Debe ubicarse dentro de uno de los módulos del proyecto.
- Debe tener asignado una prioridad por el cliente.
- Debe tener asignado un riesgo por el equipo de desarrollo.
- Debe tener asignado su respectivo puntaje.
- (Opcional pero deseado) Debe tener una breve descripción, aclaratoria o criterio adicional según sea el caso.

Asimismo, se amplió la *Definition of Done* para agregar las pruebas automatizadas de cada HU. Posee los siguientes estatutos:

- Debe realizarse la codificación respectiva
- El código generado debe estar debidamente documentado para facilidad de programador.

- Debe de realizarse la documentación respectiva de todos los aspectos de configuración asociados al desarrollo y buen funcionamiento de la historia de usuario.
- Deben realizarse pruebas automatizadas a la codificación generada.
- Debe presentarse la nueva funcionalidad al cliente.
- Debe estar disponible en el repositorio.

6.1.3 Arquitectura propuesta del sistema

El planteamiento inicial (figura 6.1) consistía en desarrollar cada módulo de la librería en una aplicación de Django distinta, las cuales se instalarían por separado en el sistema, este será referido como “Host” para simplificar la notación. Los módulos de Estadística y Reportes serían ofrecidos como microservicios dependientes del módulo *Core* pero independientes entre ellos, de manera que si alguno falla, el otro no sea afectado.

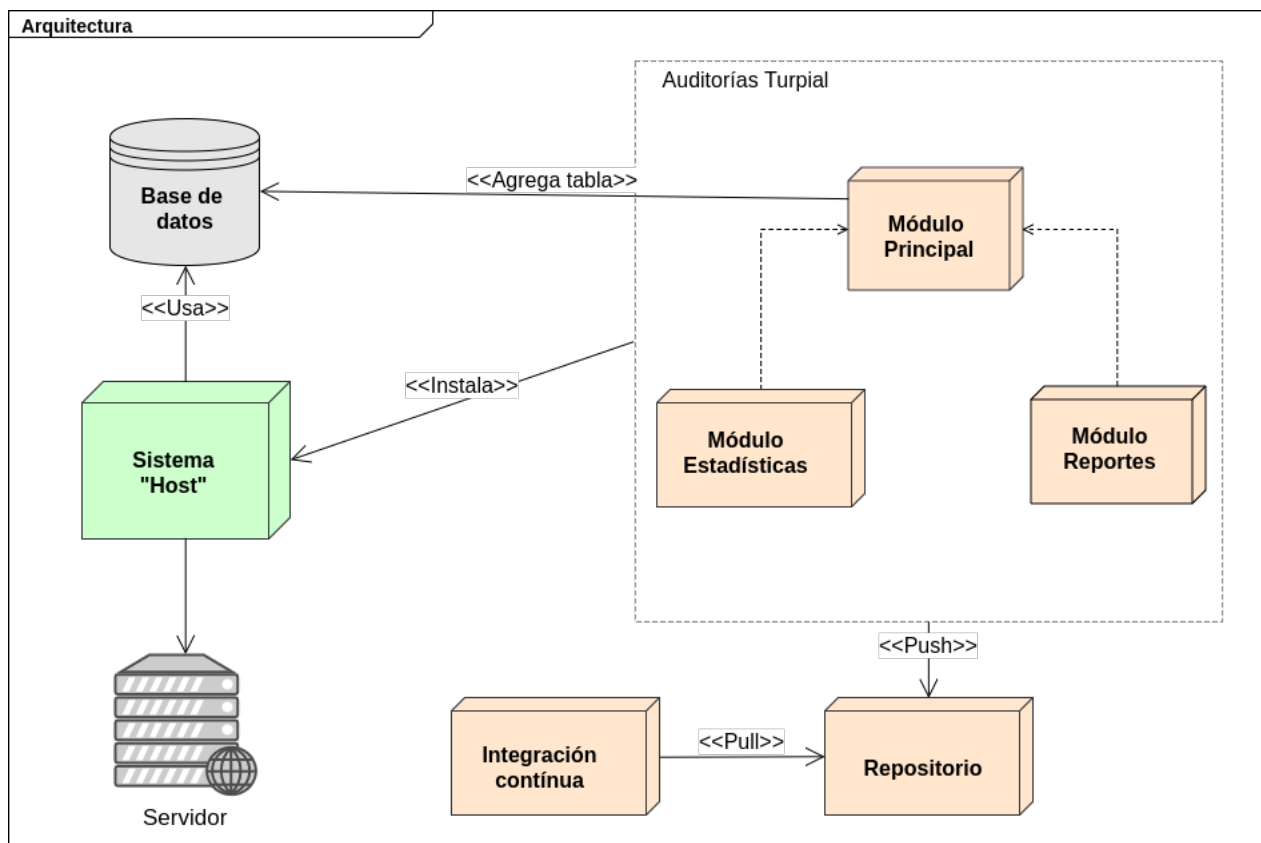


Figura 6.1: Arquitectura de la librería Auditorías Turpial

Se decidió que el módulo *Core* agregaría una nueva tabla en la base de datos ya existente del “Host” en la que mantendrá la información acerca de las auditorías y los otros módulos

podrían leerla para procesarla y mostrarla como sea pertinente.

No obstante, la arquitectura mostrada en la figura 6.1 sufrió modificaciones durante el desarrollo del proyecto para simplificarla. En lugar de crear tres aplicaciones, una para cada módulo, se separó Auditorías Turpial en dos: *backend* y *frontend*. En la sección que explica la fase de construcción se ofrecerán mayores detalles de estos cambios y sus razones.

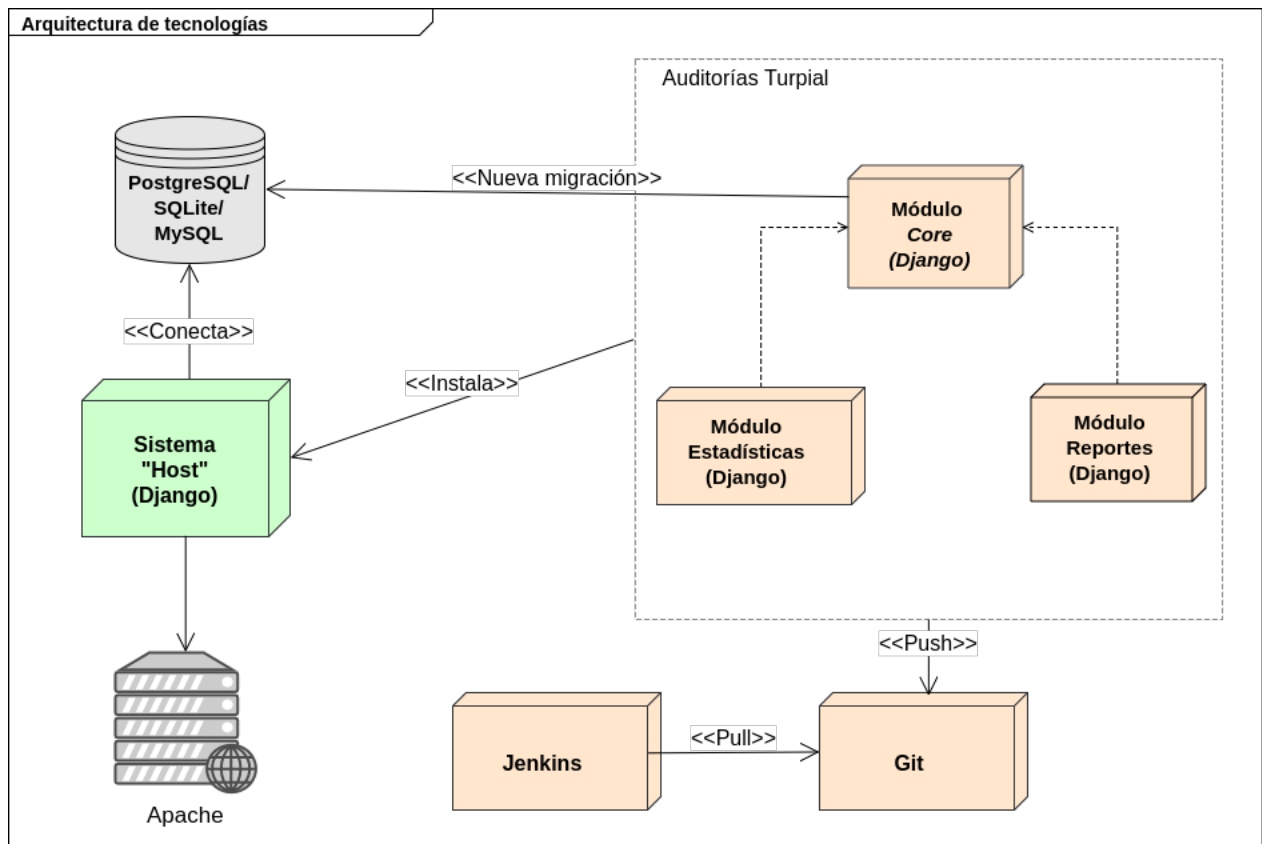


Figura 6.2: Arquitectura de tecnologías a utilizar

En la figura 6.2 se muestran las tecnologías a utilizar en el proyecto. Como se mencionó anteriormente, se utilizará Django para el desarrollo. Como herramienta de control de versiones se utilizará Git y para automatizar la integración continua se usará Jenkins. El sistema “Host” será un proyecto de la empresa que utilizará como servidor web, Apache y una base de datos relacional.

6.1.4 Diseño del módulo *Core*

Uno de los problemas más significativos que la empresa encuentra en otros *plugin* de Django disponibles, es el hecho de que el registro de auditoría se guarda a nivel de la vista y es responsabilidad del programador colocar el código para esta funcionalidad. Esto da

cabida a que se olvide colocar en alguna de las vistas correspondientes a algún modelo, por lo que podría no guardarse todos los tipos de operaciones.

La librería Auditorías Turpial pretende evitar este problema guardando el registro de auditoría cada vez que ocurre alguna operación sobre el modelo, a través del uso de las señales que ofrece el *framework*. Éstas, necesariamente deben distinguir entre un modelo auditable y uno que no lo es; se contempló el uso de un *mixin* que pueda ser heredado por cualquier modelo y así proveer todas las funcionalidades mencionadas.

Por otro lado, se desea incluir en las auditorías el usuario que realizó la acción. La solución mencionada anteriormente no es suficiente para lograr esto, puesto que a nivel de modelos no se posee información sobre el *request* y no se puede saber qué usuario está en sesión. Por esta razón, se planteó agregar otro *mixin* a nivel de vistas, que completara la información antes de guardarla en la base de datos. Sería responsabilidad del programador heredarlo en todas las vistas cuyos modelos mantienen un historial de transacciones.

Django posee mecanismos para traducir los modelos a otro formato con una estructura bien definida, más conocidos como *serializers*. En particular, posee maneras de convertirlos a JSON, por lo que se decidió utilizar dicha funcionalidad para cumplir con el requisito de mostrar los cambios en los campos del modelo auditable en este formato.

En cuanto a los listados, se acordó utilizar *Datatables* para mostrarlos como una tabla que se pueda ordenar y filtrar fácilmente. Este *plugin* puede manejar aproximadamente diez mil filas en sus tablas, procesándolas del lado del cliente. Sin embargo, las auditorías serán potencialmente millones de registros, es por esta razón que debe realizarse el procesamiento del lado del servidor.

Para la autenticación se consideró utilizar una tabla de usuarios propia con su respectiva permisología, con la intención de que no interfiriera con la del sistema “Host”, sin embargo, esto no fue posible. Se profundizará sobre la explicación de esta decisión en la fase de desarrollo del presente capítulo.

6.1.5 Diseño del módulo Estadísticas

Uno de los requisitos mínimos con el que debe contar el módulo Estadísticas, es exponer un conjunto de gráficas que permitan visualizar e interpretar fácilmente los resultados de los cálculos de las auditorías. Para esto, se investigaron dos librerías de JavaScript: Amcharts y Charts.js. En la siguiente tabla se muestran las características principales de cada una de

ellas:

Tabla 6.1: Comparación de Amcharts vs Charts.js

	Amcharts	Chart.js
Código abierto	Sí.	Sí.
Tipos de gráficos	Línea, área, barras, torta, <i>Scatter</i> , Gantt, Radar, de vela, entre otras	Línea, área, barras, torta, <i>Scatter</i> , Radar, entre otras.
Tecnología para mostrar los gráfico	SVG	HTML5 Canvas
Líneas de tendencia	Sí.	Sí.
Responsive	Sí.	Sí.
Exportar los datos	Sí, mediante un <i>plugin</i> .	No.
Zoom	Sí.	Sí. Utilizando un <i>plugin</i> .
Soporta múltiples lenguajes	Sí	No.
Librería integrada con Django	No.	Sí.
Manejo de grandes volúmenes de datos	Sí.	No. Según opiniones de los usuarios, disminuye el desempeño.

Como los desarrolladores de Turpial Development son los principales usuarios de este proyecto, el pasante investigó si existía alguna librería que utilicen usualmente. En la empresa han utilizado varias, incluyendo las dos presentadas anteriormente, por lo que no tienen una preferencia en este sentido.

Debido a que la cantidad de registros de auditorías pueden crecer rápidamente, es necesario escoger una librería de gráficos que maneje adecuadamente grandes volúmenes de datos, por lo que se decidió utilizar Amcharts. Otra característica que inclinó la balanza a favor de esta solución, es el hecho de que posee un *plugin* que permite exportar datos a un archivo CSV o PDF, una funcionalidad que se compenetra bien con el módulo de Reportes.

Aunque Charts.js posee una extensión de Django que facilita la creación de los gráficos a través de las vistas, este proyecto requiere una lógica muy específica para entregar el conjunto de datos que se utilizará para mostrarlos. Por esta razón, es mejor tener total control sobre su implementación.

Los cálculos estadísticos serán entregados a la plantilla a través de una vista, que incluirá el formulario de los filtros y la manipulación de los datos para crear los gráficos pertinentes

y según los requiera la librería escogida.

6.1.6 Propuesta para la integración continua

Dado que en la empresa no existe precedente sobre el uso de una herramienta automatizada para integrar el código de manera continua, se le otorgó al pasante la libertad de escoger entre dos herramientas sugeridas por el líder del proyecto: Jenkins o Fabric. Para decidir cuál herramienta se adecuaba más al proyecto y a las necesidades de la empresa, fue indispensable que el pasante investigara las opciones en profundidad, sus fortalezas, limitaciones y recomendaciones de la comunidad. A continuación se muestra una comparación entre ellas:

Tabla 6.2: Comparación de Jenkins vs Fabric.

	Jenkins	Fabric
Descripción	Un servidor de automatización que puede ser utilizado para automatizar cualquier clase de tarea cómo construir, probar y desplegar software	Es una librería y una herramienta de línea de comandos para coordinar el uso de SSH para despliegues de aplicaciones o tareas de sistemas de administración.
Código abierto	Sí.	Sí.
Lenguaje	Escrito en Java	Escrito en Python
Extensible	Si. Cuenta con una gran cantidad de <i>plugins</i> para ampliar sus funcionalidades básicas y una tienda en donde pueden adquirirse	No.
Interfaz gráfica	Sí.	No.
Integración con Gitlab	Sí.	No
Ejecutar un script	Sí.	Sí.
Empaquetamiento y despliegue programado	Si. Se puede configurar un horario para ejecutar algún script que contenga todas las instrucciones para empaquetar, probar y desplegar el sistema.	No. Se debe ejecutar el archivo de configuración a través de la consola.
Observaciones	Altamente recomendado por la comunidad. Puede generar reportes de las pruebas ejecutadas.	Fácil de aprender, de configurar y de instalar.

Considerando las características que posee cada herramienta, se decidió utilizar Jenkins,

puesto que se pueden ejecutar *scripts* de manera programada, generar reportes, observar el estado del despliegue de varios proyectos e incluso integrarlo con Gitlab. Gitlab es un servicio web de control de versiones y desarrollo de *software* colaborativo basado en Git, que es utilizado por la empresa.

Si bien Fabric es más simple, no es realmente un servidor de integración continua, se asemeja más a *scripts* que permiten automatizar tareas, y se requiere de una herramienta suficientemente general que pueda ser usado tanto en la pasantía como en otros proyectos de la empresa.

6.1.7 Plan de pruebas

Para verificar que cada funcionalidad posea el comportamiento esperado, se realizaron pruebas de manera automatizada utilizando Pytest. Más específicamente, se llevaron a cabo pruebas unitarias, de regresión y de integración.

Adicionalmente, se convino que uno de los criterios de aceptación de las HU, era validar el producto con el cliente para asegurar el cumplimiento de las reglas de negocio y que se cumplen los estándares. A esto se le conoce como pruebas de aceptación, y fueron realizadas en cada cierre de *Sprint*.

Como el proyecto en cuestión es una librería, no puede funcionar por sí sola, sino que necesita instalarse en otro. Para esto, se creó una aplicación sencilla en Django y se instaló la librería en él. En cada *Sprint*, se constató que el programador pueda usar las funcionalidades desarrolladas sin ningún inconveniente. Este mismo proyecto, se utilizó para mostrar los avances al cliente.

En la fase de transición, se planificó la realización de pruebas en una aplicación desarrollada para uso interno de la empresa, llamado Turpial Team. No obstante, debido a que aún se encuentra en desarrollo, no fue posible utilizarla. En su lugar, se seleccionó otro proyecto de uso interno de Turpial Development, llamado Turpial Calendar. La aplicación sirve para planificar eventos de la empresa y enviar notificaciones. Esta decisión no afecta de ninguna manera la pasantía, puesto que la librería debe poder ser instalada en cualquier proyecto con las características mencionadas en la sección de requerimientos.

En el apéndice D se encuentra el documento de Plan de Pruebas creado para la empresa, para ofrecer más detalle de lo explicado en esta sección.

6.1.8 Planificación del desarrollo del proyecto

Luego de finalizar el proceso de investigación, aclarar los requerimientos y determinar las HU, se procedió a planificar la fase de construcción del proyecto. Para ello, se tomó en cuenta la prioridad, precedencia y puntaje de cada HU para determinar el orden. Se decidió iniciar con las HU relacionadas con el *Core* y la instalación de la librería.

Se planificaron ocho *Sprints* con una duración de dos semanas laborales (diez días) cada uno, los cuales abarcan la fase de construcción del módulo *Core* y Estadísticas de Auditorías Turpial, así como la fase de transición.

6.2 Fase de construcción

Una vez culminada la fase de concepción, se procedió con la instalación del ambiente de desarrollo e implementación de los módulos que abarca la pasantía. También, se incluyen las pruebas pertinentes para cada funcionalidad desarrollada según indica el Plan de Pruebas.

6.2.1 Construcción del módulo *Core*

En esta sección se describe el proceso para desarrollar cada HU relacionada con el módulo *Core*, los problemas encontrados y sus soluciones. También se explica el nuevo diseño de la arquitectura y las pruebas efectuadas.

6.2.1.1 Preparación del ambiente de desarrollo

Antes de iniciar con la implementación de la librería, se instalaron y configuraron todas las herramientas necesarias para esto. En primer lugar, se instaló Python 2.7 y luego PIP. Se instaló el *plugin* de Python, Virtualenv, el cual permitió configurar el ambiente virtual. Este, se utilizó para instalar los requerimientos de la librería, en particular, Django 1.10 y Pytest 3.2. Se registraron en el archivo “requirements.txt” para facilitar futuras instalaciones y determinar las dependencias de la librería. Luego, se procedió a crear la aplicación de Django y la estructura del módulo *Core*.

6.2.1.2 Estructura de la tabla de auditorías

Para registrar apropiadamente la información de la auditoría como fue establecida en la fase anterior, se creó un modelo en Django cuyo nombre es “AuditableAction”. Éste se agregará como una tabla adicional en la base de datos del sistema “Host” y posee los siguientes campos:

Tabla 6.3: Modelo de datos de la tabla “AuditableAction”

Nombre del campo	Tipo de dato (Django)	Descripción
ID	<i>AutoField</i>	Identificador del registro auditable. Entero de 32 bits. Los valores van desde -2147483648 a 2147483647.
author	<i>ForeignKey</i>	Referencia al usuario que realizó la acción.
action	<i>CharField</i> . Limitado a las opciones: CREATED, UPDATED, DELETED.	El tipo de acción efectuada sobre un modelo auditable.
datetime	<i>DateTimeField</i>	Fecha en la que se registró la acción.
model_name	<i>CharField</i>	Nombre del modelo auditable.
model_json_old	<i>JSONField</i>	Antiguo valor. Un JSON que representa el valor que tenía el registro que se ve afectado por la operación. En caso de que la acción sea “creado”, el JSON representará el objeto vacío.
model_json_new	<i>JSONField</i>	Nuevo valor. Un JSON que representa el valor que tiene el registro que se ve afectado por la operación. En caso de que la acción sea “eliminado”, el JSON representará el objeto vacío.

En cuanto a los campos tipo JSON, se utilizó el *plugin* de Django, *Jsonfield*, que provee todos los validadores necesarios para que el campo de sea considerado un JSON, y así, mantener la integridad de la base de datos. El campo es traducido a uno tipo texto.

6.2.1.3 Selección de un modelo auditable

Para que un programador pueda escoger qué modelo es auditable y cuál no, se creó un *mixin* a nivel de modelos, llamado “AuditableMixin”. Este permite heredar el comportamiento necesario para registrar las auditorías en la base de datos.

6.2.1.4 Registro de una acción auditable

Como bien se ha dicho antes, Django posee *signals* que son detonadas por diferentes eventos; dos de ellos están relacionados con la creación y actualización de objetos, “pre_save” y “post_save”. Estas señales se activan antes de guardar un objeto en la base de datos y después, respectivamente; bien sea para crearlo o actualizarlo. Cada una de ellas posee información sobre la instancia que será guardada y el modelo que envía la señal (*sender*). El *sender* es de gran utilidad puesto que permite determinar cuál es su clase, y así auditar

solo los modelos que sean una subclase del “AuditableMixin” definido anteriormente.

En principio, se optó por hacer un manejador para la señal “pre_save” debido a que, antes de guardar en base de datos, se puede obtener tanto el valor actual de la instancia como el que tendrá. Desafortunadamente, esta solución no asegura que se almacene la instancia a auditar en la base de datos, pero sí la traza de auditoría, porque podría ocurrir algún error justo en medio de estas dos operaciones, lo que generaría problemas de consistencia.

Utilizar la señal “post_save” tampoco fue una opción, ya que no se podía obtener el valor del registro anterior, lo cual es de suma importancia para comparar los cambios ocurridos.

Motivado por esto, se decidió sobrescribir el método “save()”, con lo que se puede realizar el procesamiento para capturar el valor anterior, guardar el objeto auditable y generar la auditoría. Se distingue el tipo de acción si la instancia posee o no una clave primaria; si no la posee se está creando.

El primer inconveniente encontrado con esta solución es el hecho de que, al cargar la base de datos con un *Fixture*, el *framework* no utiliza “save()”. Asume que los datos dentro del archivo son íntegros y los inserta, activando las señales correspondientes. El segundo, es que este comportamiento se replica con las inserciones y actualizaciones masivas, “bulk_create” “bulk_update”. Los mencionados métodos se traducen directamente a sentencias de SQL, por lo que tampoco activan *signals*. En la documentación de la librería se explican estos problemas y esperan ser corregidos en una próxima versión.

Con respecto a la acción DELETE, no hubo mayores contratiempos, se utilizó la señal “post_save”, la cual se activa luego de que se elimina una instancia, dado que no se requiere registrar el valor posterior.

Relación muchos-a-muchos Django cuenta con la facilidad de, a nivel de modelos, crear relaciones muchos-a-muchos sin necesidad de que el programador explícitamente genere la tabla intermedia que los conecta; a través del campo “ManyToManyField”. Los cambios ocurridos en ellos, activan una señal llamada “m2m_changed”, la cual se utilizó para llevar control de los cambios de apuntadores (llaves primaria) hechos. No obstante, fue descartado debido a que el comportamiento natural del *framework* cuando se crea por primera vez el registro, es guardar la instancia sin los cambios en estos campos y luego guardarlos nuevamente.

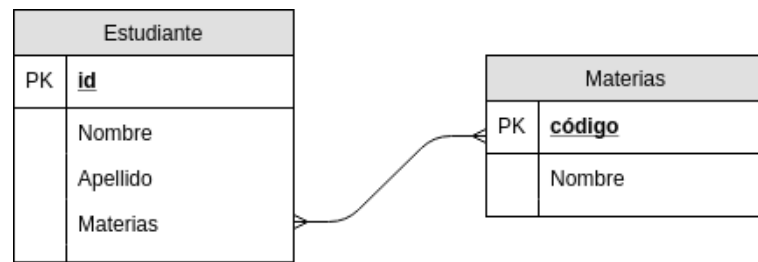


Figura 6.3: Modelo Entidad-Relación del ejemplo.

Por ejemplo, si se tiene un tabla Estudiante, que tiene una relación muchos-a-muchos con Materias como muestra la figura 6.3, al intentar guardar una instancia de Estudiante con algunas Materias, se sigue el siguiente flujo:

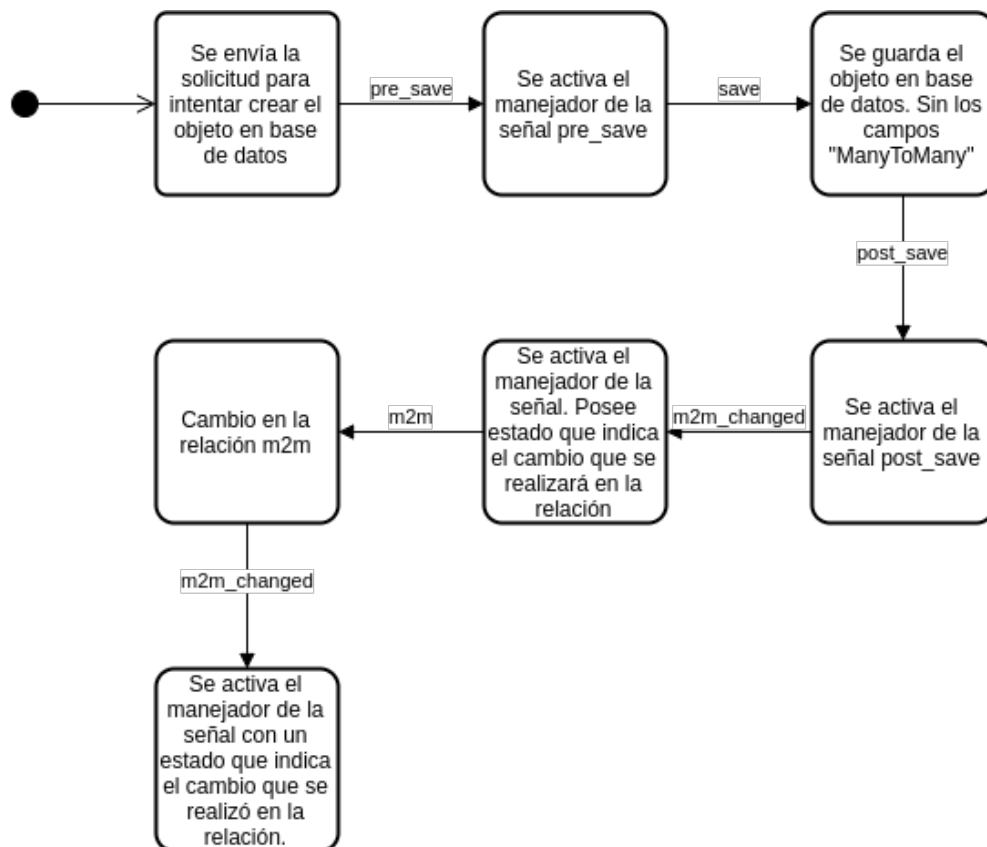


Figura 6.4: Gráfico de flujo del orden en el que se activan las señales para guardar un cambio en los campos “ManyToManyField”.

Debido al orden en que se activan la señales (figura 6.4), se almacena el objeto en base de datos y luego, cuando cambia el campo “materias”, se guarda nuevamente, por lo que era imposible obtener en un sólo registro de auditoría todos los cambios.

El pasante propuso al cliente dos opciones: la primera es permitir que se generen dos

registros de auditoría, el primero con acción `CREATE` y el segundo con `UPDATE`, puesto que en realidad es una actualización. La segunda, es excluir esta relación de la auditoría, es decir, no utilizar `“m2m_changed”`.

Esta última, fue la escogida por el cliente y se documentó apropiadamente. Si el programador desea llevar auditorías de los cambios ocurridos en casos como estos, puede escribir su propia tabla intermedia para relacionar ambos modelos y utilizar la opción `“through”` provista por `“ManyToManyField”` como indica la documentación de Django. Si, esta tabla, hereda el `“AuditableMixin”` se replica el comportamiento buscado inicialmente al incluir los campos muchos-a-muchos

Serializers Como se diseñó en la fase de concepción, se utilizaron los *serializers* del *framework* para almacenar la estructura completa de la instancia en cuestión, con el formato JSON. Los *serializers* incluyen la clave primaria del objeto, el nombre del modelo y los campos que este contiene.

Automáticamente, Django incluye la representación de los campos `“ManyToManyField”` y al decidir no rastrear los cambios en estos, fue necesario excluirlos explícitamente para evitar inconsistencias y confusiones.

Inclusión del usuario en la traza de auditoría Lo explicado anteriormente funciona perfectamente para registrar la auditoría y verificar los cambios, sin embargo no es suficiente para obtener el usuario, puesto que a nivel de modelos no se cuenta con esta información. Para solucionar esto, se creó otro *mixin* llamado `“AuditableMixinView”`, que inyecta la información del usuario en un campo oculto incluido en el `“AuditableMixin”`. De esta manera, en las señales y en el método `“save()”` se puede poseer la información del usuario en sesión y todo lo relacionado con él.

El pasante se aseguró de que el nombre de este campo fuese suficientemente específico para evitar conflictos con el modelo a auditar, porque si poseen el mismo nombre podría ocurrir algún error en el sistema `“Host”`. Se utilizó el nombre `“_turpial_user”` y está documentado en la librería para alertar al programador que la use.

6.2.1.5 Instalación de la librería utilizando PIP

Una vez obtenida una versión suficientemente sólida de la librería, se prosiguió a satisfacer uno de los requerimientos más importantes: que sea capaz de instalarse utilizando PIP. El pasante debió documentarse al respecto, debido a que existía gran desconocimiento en todo el equipo. En la página web oficial de Django se encuentra un tutorial bastante detallado

para construir extensiones el cual sirvió de guía para el proceso.

Lo primero que se hizo fue construir un archivo llamado “setup.py”, que contiene la información de la librería, el nombre (“turpial_auditor”), la versión, los autores, la licencia y los paquetes que requiere, entre otros. En el caso de “turpial_auditor”, el paquete requerido fue Jsonfield.

Luego, se creó el archivo “MANIFEST.in” para incluir aquellos archivos que no son detectados automáticamente por el “setup.py”, como las plantillas de Django, los estáticos (JavaScript, CSS), la licencia, entre otros.

Por último, en la consola, posicionados en la carpeta en la que se encuentra el archivo “setup.py”, se procede a empaquetar la librería. Una vez culminado este proceso, se tiene un archivo comprimido que se puede utilizar para instalar la librería con PIP.

El pasante documentó con gran detalle este proceso (Apéndice J) para que el resto de los miembros del equipo pudiesen seguir los pasos. Esta documentación está disponible para la empresa, en caso de que deseen crear una nueva librería o actualizar Auditorías Turpial.

6.2.1.6 Listados de auditoría

Culminada toda la construcción de los *mixins*, *signals* e instaladores de la librería, se procedió a implementar la funcionalidad de listados de todas las auditorías registradas en la base de datos. Para esta funcionalidad se decidió utilizar el *plugin* de JavaScript, *Datatables*, que permite integrar de forma fácil y sencilla una tabla a la plantilla. Los listados son procesados en el servidor y tienen paginación, puesto que las auditorías pueden poseer millones de registros. Con esta decisión se evitan dos aspectos relevantes, sobrecargar el navegador de altos volúmenes de contenidos e incrementar el tiempo de espera del usuario ocasionado por cargado de los listados.

Al usuario presionar alguno de los botones en la plantilla del listado, se realiza una solicitud GET de dicha página utilizando AJAX. De esta manera, se refresca ese componente sin necesidad de recargar completamente la página, lo que mejora la experiencia de usuario.

6.2.1.7 Reestructuración de la arquitectura

Antes de iniciar con la elaboración de los otros módulos de la librería, se notó que la funcionalidad de generar reportes está íntimamente relacionada con los listados de auditoría.

Los filtros y el ordenamiento aplicados a estos, deben aparecer en los CSV y PDF generados. Es por esta razón que, al finalizar el desarrollo de esta funcionalidad, el equipo se replanteó la estructura establecida en la fase de concepción.

En lugar de crear tres aplicaciones en Django, una para cada módulo, se decidió crear sólo dos, como muestra la figura 6.5. La primera de ellas sería el backend de Auditorías Turpial, que mantiene toda la lógica para registrar las auditorías, explicada en los puntos anteriores; mientras que la segunda tendría los módulos Reportes y Estadísticas. A esta última la llamaremos Auditorías Turpial UI (User Interface) y contendrá todo el frontend del proyecto. Las funcionalidades de los listados fueron colocadas en el módulo Reportes para facilitar la generación de PDF y CSV.

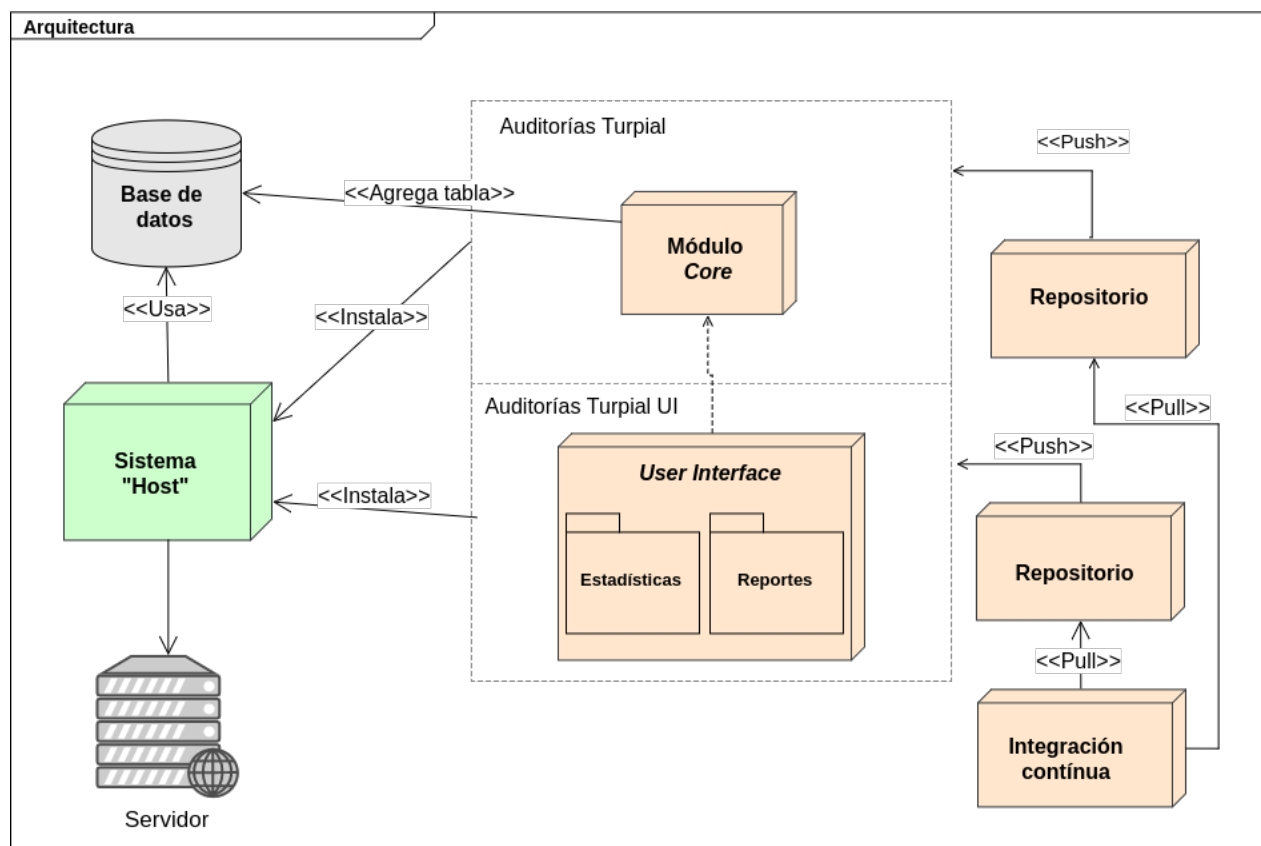


Figura 6.5: Arquitectura final de la librería.

Esta reestructuración permite separar la captura de información, de la manera en la que se muestra, sin que se vean afectados los beneficios que ofrecen los microservicios. Aunque los módulos Estadísticas y Reportes se encuentren dentro de la misma aplicación, el programador puede elegir no instalar alguno de ellos en su sistema si lo excluye de las "INSTALLED_APPS" (aplicaciones instaladas) de Django.

De aquí en adelante nos referiremos a la librería completa como Auditorías Turpial, y a las aplicaciones originada de estas, las llamaremos Auditorías Turpial Core y Auditorías Turpial UI, para el backend y frontend, respectivamente.

Para Auditorías Turpial UI, se realizó el procedimiento descrito en la sección de instalación con PIP, utilizando el nombre “turpial_auditor_ui” y colocando como dependencia “datatables-view” utilizada en el módulo Reportes.

En este punto del desarrollo, se dispone de un módulo *Core* que cumple todas las funcionalidades planteadas según la nueva arquitectura, exceptuando los permisos. En la figura 6.6 se muestra la estructura de este módulo dentro del patrón MVT.

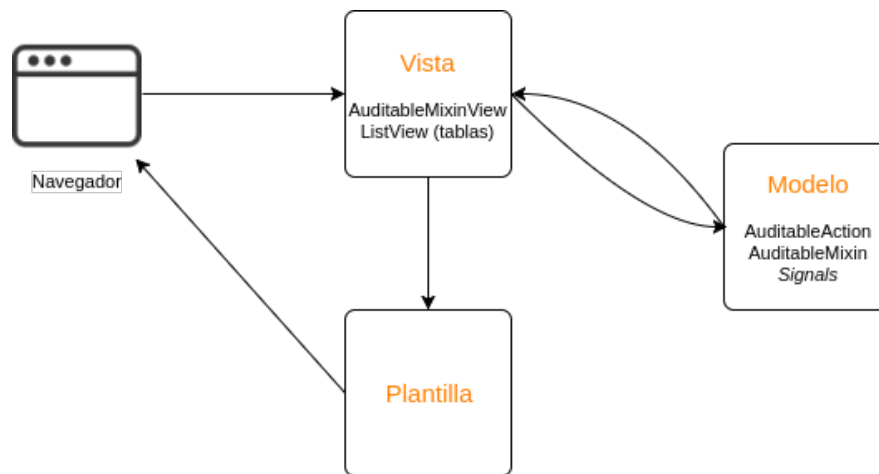


Figura 6.6: Estructura del módulo *Core* en el patrón MVT.

6.2.1.8 Integración continua

Luego de haber terminado gran parte del módulo *Core*, se prosiguió a instalar Jenkins. Para esto, se ingresó al terminal de la computadora, se agregó el repositorio de la herramienta y se instaló.

Una vez realizado esto, el pasante ingresó, mediante la interfaz gráfica, al sistema de integración continua para configurar el repositorio del proyecto, ejecutando los siguientes pasos:

1. Instalar el *plugin* de Git en Jenkins.
2. Agregar las credenciales de Git del pasante (usuario y contraseña) en el sitio administrador. De esta manera quedan almacenadas dentro de Jenkins para futuras acciones que las requieran.

3. Incluir una “nueva tarea” para crear un nuevo proyecto.
4. Clonar el repositorio, ingresando a la sección de “Configuración” del proyecto creado y proporcionando el enlace HTTPS. La herramienta permite seleccionar una rama específica de la cual hacer *pull*; se escogió la rama “develop”.
5. En la configuración del proyecto, agregar la ejecución de un *script* de shell luego de hacer *pull* del repositorio para desplegar el proyecto.

El pasante creó un *script* que permite instalar el ambiente virtual y los *plugins* necesarios para que el proyecto pueda ejecutarse, migrar la base de datos y ejecutar las pruebas. Esto es lo mínimo requerido por la HU. Adicionalmente, construyó otro *script* que automatiza el empaquetamiento de la librería, el cual podría incluirse en el despliegue de la misma.

Por último, se probó la configuración de Jenkins mediante la interfaz. El pasante utilizó el botón de “build” y comprobó en la consola de la herramienta que se había descargado la información más reciente de la rama “develop” y que se había ejecutado el *script* exitosamente.

Con el uso de Jenkins, se puede llevar el control de los despliegues de la librería y, en caso de que alguno falle, se puede detectar rápidamente verificando el estado del “build”. De esta manera, gran parte del tiempo, la librería está disponible para la empresa y pueden comprobar que sea estable sin necesidad de descargarla.

El comportamiento logrado con el *script* mencionado anteriormente, no evita que en el repositorio exista una versión inestable, puesto que el *build* se efectúa luego de que los cambios han sido incluidos en el repositorio. Es por esta razón que el pasante decidió investigar sobre posibilidad de evitar que se incluya código que no satisfaga las pruebas.

Dado que se utilizó Gitlab, se investigó sobre la configuración de Jenkins con este y se encontró que existe un *plugin* que permite la integración de ambos sistemas. El pasante procedió a instalarlo, realizando las siguientes configuraciones sobre ambos:

- En Gitlab, fue necesario generar un identificador único llamado “Gitlab API Token” creado para cada usuario. Adicionalmente, se debe crear un “Webhook” con el URL en el que está instalado Jenkins que incluye el proyecto en cuestión.
- En Jenkins, se utiliza el “Gitlab API Token” como credencial para conectarse con Gitlab. Luego, se procede a configurar el proyecto. En la sección de “Configuración”,

el *plugin*, agrega una nueva opción para accionar el empaquetamiento cuando ocurra un evento en Gitlab. En particular, el evento *push*.

Para crear el “Webhook”, era indispensable instalar Jenkins en un servidor de la empresa, para que el URL mencionado pueda accederse desde Gitlab. No obstante, el líder técnico consideró que no era necesario el comportamiento adicional descrito, así que se descartó.

El pasante dejó una documentación detallada acerca del proceso llevado a cabo para referencias futuras de algún otro miembro de la empresa, así como de la integración con Gitlab. Con esto, se pueden replicar fácilmente los pasos e instalar Jenkins en un servidor de la compañía e incluir otros proyectos, además de esta pasantía.

6.2.1.9 Pruebas del módulo *Core*

Para el módulo Core, se generaron veintidós (22) casos de prueba automatizados utilizando Pytest (ver Apéndice E). Dentro de estas pruebas, once (11) consistían en validar el funcionamiento del “AuditableMixin”, verificando que se guardara apropiadamente todos los campos en la traza de auditoría.

Dentro del módulo de pruebas de la librería se agregaron cuatro (4) modelos con las relaciones mostradas en la figura 6.8, que sólo existen cuando se crean las pruebas. Al inicializar la ejecución de los casos, se efectúa una migración de la base de datos, que permite a estos modelos ser incluidos en el esquema. Estos, poseen relaciones suficientemente complejas para poder realizar casos de pruebas con uno-a-uno, uno-a-muchos y muchos-a-muchos.

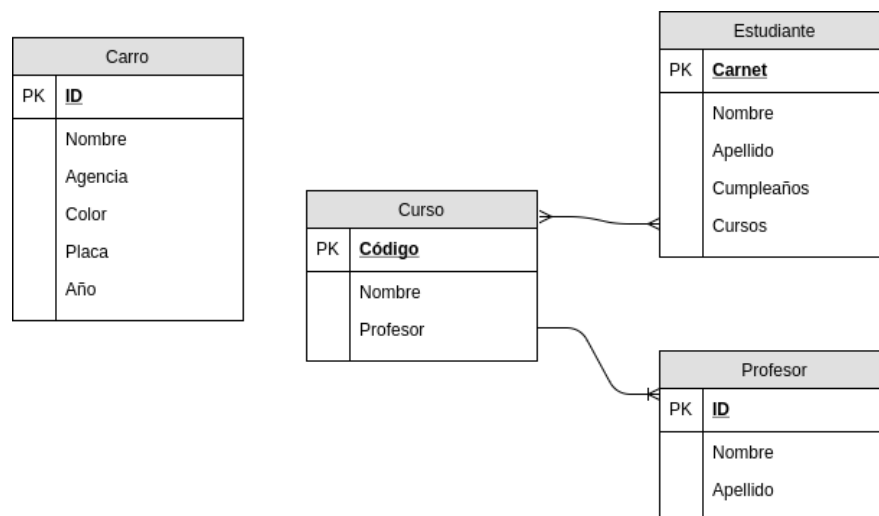


Figura 6.7: Modelo Entidad-Relación de la base de datos de prueba.

Antes de que comiencen a ejecutarse las pruebas, se precargan los registros con un *Fixture*

que utiliza el formato JSON. Estos están escritos siguiendo la estructura de los *serializers*; contienen la clave primaria, el modelo al que pertenece y el valor de cada campo como muestra la figura 6.9. Luego, se configuró Pytest para que utilizara el comando de Django “loaddata” y así, cargar el *Fixture*. La base de datos es desechada al finalizar la ejecución.

```
{
  "model": "tests.carsmodel",
  "pk": 1,
  "fields": {
    "name": "Twingo",
    "manufacturer": "Renault",
    "color": "Red",
    "license_plate": "ADB02F",
    "date": "12-02-2000"
  }
},
```

Figura 6.8: Ejemplo del formato JSON para cargar un *Fixture*.

De estos casos de prueba, se encontró que fallaron dos, debido a que el *serializer* no excluía los campos “ManyToManyField”. Se corrigió el código según lo explicado en la sección sobre *serializers*, y se obtuvo que todas las pruebas fueron exitosas.

Once de los casos de pruebas están dedicados a verificar que el usuario se agregue correctamente con el “AuditableMixinView”. Se configuró “RequestFactory” para simular un *request* que solicitara un URL particular. De esta manera, se pueden probar la vista como se haría con cualquier otra función; con una entrada fija y una salida esperada, sin inspeccionar el contexto de la petición.

Todas estas pruebas resultaron exitosas exceptuando el caso C04-P06 (ver Apéndice E) puesto que al eliminar un modelo no-auditable con una referencia a otro que sí lo es, no se realiza la llamada a la función de la vista del modelo auditable. Por esta razón, no se almacena el usuario en estos registros. Esto fue documentado y se aceptó esta limitación puesto que se conserva la integridad del sistema de auditorías.

Una vez terminada las pruebas unitarias y de integración, el módulo *Core* cumple todos los requisitos del cliente, incluyendo el servidor de Integración Continua. De esta manera, se alcanzan los objetivos planteados.

6.2.1.10 Permisología

Con relación a la permisología de las auditorías, la HU C11 (ver Apéndice C), especifica que se desea contar con un sistema de autenticación (login) y que cada usuario autenticado posea un conjunto de permisos asociados. Estos, permitirán restringir ciertas acciones dentro de las auditorías, en particular visualizar listados.

Para crear el sistema de autenticación, se quería crear un modelo de usuarios para la librería. No obstante, al pasante investigar sobre esto, encontró que la documentación de Django indica que los *plugins* no deben crear su propio modelo de usuarios. Debido a que podría generar conflictos con la aplicación que lo instale. Por consiguiente, se descartó la HU relacionada con la autenticación (C11.1).

Tomando en cuenta esto, se sugirió realizar un módulo Permisología que incluyera, tanto las restricciones de los listados, como los de generación de reportes (PDF y CSV).

Dado que la implementación de este módulo no se encontraba en el objetivo de la pasantía, se decidió no incluirlo para esta versión de la librería. Sin embargo, el pasante decidió realizar el diseño del mismo con la intención de que sirva de referencia.

Este módulo también sería un microservicio, como Estadísticas y Reportes, que incluya permisos básicos y brinde facilidades para generar nuevos permisos. Para restringir una funcionalidad específica, se utilizaría el mismo patrón de desarrollo del resto de la librería; ofrecer un *mixin* que encapsule el comportamiento. Para obtener mayor detalle sobre el módulo Permisología ver el Apéndice L.

6.2.1.11 Análisis de desempeño

Con la intención de evaluar cuánto tiempo de computo agregaba el registro de las auditorías, se realizaron pruebas de desempeño. El diseño de la prueba consistió en cargar tabla de auditorías con mil, diez mil, cien mil y un millón de registros. Cada corrida, ejecutaba cien nuevas inserciones tomando el tiempo, tanto para un modelo auditable como para uno que no lo era. De esta manera se pueden comparar los resultados.

Estas pruebas se realizaron con MySQL, PostgreSQL y SQLite, para tener una referencia del comportamiento de la librería para cada uno de estos manejadores de base de datos.

Adicionalmente, se compararon los resultados con la base de datos indexada y sin índices.

Se repitió el mismo procedimiento descrito anteriormente y se concluyó que, aunque los índices incrementen aproximadamente 267% 786% y 2968% de tiempo, (SQLite, PostgreSQL y MySQL respectivamente) es imperceptible para el usuario. Además, los índices agilizan las consultas lo que beneficia a los módulos de Reportes y Estadísticas. En el apéndice F, se explica cómo fue el proceso para realizar estas pruebas y se muestran los resultados de cada corrida para generar las estadísticas.

6.2.1.12 Prototipo de tabla de auditorías descentralizada

Durante el desarrollo del módulo *Core* surgió la duda sobre si era posible crear diferentes tablas de auditoría, una por cada modelo auditable. Esto con la intención de distribuir la información dado que los manejadores de base de datos tienen almacenamiento limitado para cada tabla. Así se podría evitar que se excediera su capacidad rápidamente.

El pasante decidió diseñar e implementar un prototipo funcional de este comportamiento, aprovechando la facilidad de Python de crear clases de manera dinámica. Para esto se decidió capturar la señal “class_prepared”. Cada vez que se genera un nuevo modelo, se activa esta señal y, si este es auditable, se crea una nueva tabla con el mismo nombre pero con el sufijo “_audit”.

Para crear los modelos es necesario rellenar la meta información, que incluye el “app_label”, que es el nombre del módulo al que pertenece el modelo en cuestión. Se colocó el módulo *Core* de las Auditorías Turpial para que todas las tablas viviesen en el mismo espacio de nombres.

Se decidió continuar el desarrollo utilizando una tabla centralizada, puesto que generar varios modelos complicaban las consultas que se realizarían en los módulos Reportes y Estadísticas. Para más información sobre la implementación de este prototipo y sus limitaciones, ver el Apéndice K.

6.2.2 Construcción del módulo Estadísticas

En esta sección, se explica el desarrollo del módulo Estadísticas; cómo se generaron los filtros, cálculos y gráficos necesarios, para presentar al usuario final la información recopilada por el módulo *Core*.

6.2.2.1 Creación del formulario de filtros

Para cumplir con las HU que se refieren al filtrado de los registros de auditoría, se procedió a crear un formulario. Inicialmente, se incluyeron aquellos campos relacionados con el rango de tiempo. De esta manera se puede construir el flujo de datos básico, sin preocuparse por el resto de los filtros. Es importante destacar que ninguno de los campos es requerido y si no se selecciona, se obtiene toda la información.

Para crear los campos fecha de inicio, hora de inicio, fecha de fin y hora de fin; se aprovecharon los tipos “DateField” y “TimeField” de Django. Estos manejan la validación del formato de los valores ingresados y, en caso de ser correctos, procede a convertirlos en un objeto “datetime” de Python para facilitar su manipulación. Aunque Django posee un campo “DateTimeField”, en el que se puede obtener tanto la fecha como la hora, se decidió mantener estos campos separados, para tener completo control sobre los validadores y rellenar la información no requerida.

El método “clean()” del formulario fue sobrescrito para manejar la lógica de los validadores y evitar el envío de un formulario incorrecto que pueda generar una excepción. Este método se encarga de:

- Validar que la fecha de inicio no sea mayor que la de fin, tomando en cuenta la hora.
- Validar que la fecha de fin no sea mayor a la actual, tomando en cuenta la hora.
- Arrojar un error en el formulario cuando se coloca la hora (de fin o inicio) pero no la fecha relacionada. Este caso es muy importante puesto que no se puede filtrar únicamente con las horas porque generaría un error en el *query*.
- En caso de que no sea ingresado, rellenar el campo de la hora de inicio con el valor 00:00.
- En caso de que no sea ingresado, rellenar el campo de la hora de fin con el valor 23:59.

Una vez estructurado el formulario con el rango de tiempo, se prosiguió a armar el *query* asociado. Para esto, se utilizó la vista prefabricada del *framework* “ListView”, que maneja toda la lógica de listar objetos utilizando un *queryset* particular. La vista resultante se denominó “StatisticsView”. Este *queryset* se configuró para que obtuviese todos los resultados de la tabla de auditorías, no obstante, el programador puede cambiar este atributo.

El formulario se incluye en una vista adicional llamada “FilterView” que hereda “StatisticsView”. De esta manera, en el código se distingue fácilmente la lógica relacionada

a los filtros y la relacionada con los gráficos. En “FilterView”, se colocó el formulario en el contexto y si el mismo es válido, se procede a llamar a la función “build_filters”, la cual añade nuevos filtros al *queryset* inicial, según los valores enviados en el formulario a través del método POST.

Es importante destacar que el *framework* maneja los *queries* de manera “lazy”, lo que quiere decir que no serán evaluadas hasta que realmente sean necesarias. Este comportamiento permite filtrar el *queryset* inicial tantas veces como indique el formulario, sin necesidad de cargar todos los objetos en memoria ni aumentar el tiempo de respuesta del usuario.

También se agregaron tres botones que permiten agilizar el envío del formulario cuando se desea filtrar por la fecha, mes o año actual. Estos botones rellenan los campos de fecha según sea el caso, haciendo uso de funciones de JavaScript.

Una vez creada la estructura que permite filtrar por rango de tiempo, se procedió a agregar al formulario, los campos “action”, “model” y “author” para filtrar por tipo de acción, modelo y autor respectivamente. Con cada uno de ellos, se repitió el proceso descrito anteriormente. En definitiva, el formulario cuenta con los campos mostrados en la figura 6.10.

```
class FilterForm(forms.Form):
    """Filter form for statistics."""

    start_date = forms.DateField()
    start_time = forms.TimeField()
    end_date = forms.DateField()
    end_time = forms.TimeField()
    author = forms.ChoiceField()
    action = forms.ChoiceField()
    model = forms.ChoiceField()
```

Figura 6.9: Estructura del formulario de filtros.

6.2.2.2 Cálculos estadísticos

Luego de obtener el *queryset* filtrado, se procedió a construir los cálculos estadísticos indicados por los requerimientos: el promedio, máximo, mínimo y total de elementos. Para esto, se procesa el *queryset* creando un diccionario, llamado “stats”, que posee toda la información referente a los cálculos. Incluye otro diccionario llamado “data”, cuya clave es el día en que fue realizada la operación y su valor es la cantidad de transacciones en el

mismo, como se refleja en la figura 6.11. La variable “data” se utiliza para la creación de gráficos como se explica en la siguiente sección.

```
{
  "all": {
    "count": 4,
    "max": 2,
    "min" : 1,
    "avg" : 0.010,
    "data": {
      "2017-10-16": 1,
      "2017-10-19": 1,
      "2017-10-25": 2
    }
  }
}
```

Figura 6.10: Ejemplo de “stats”.

En el formulario, se creó la función “days_elapsed” para obtener la cantidad de días transcurridos entre la fecha de inicio y la de fin. Este valor se utiliza para calcular el promedio apropiadamente, dividiendo el total de operaciones entre la cantidad de días transcurridos. Si no se provee fecha de inicio, se toma la fecha del primer objeto en la tabla de auditorías para calcular la diferencia de fechas.

Para calcular el porcentaje de crecimiento, en “build_filters” se determina la cantidad de registros que ocurrieron antes de la fecha de inicio ingresada en el formulario. Este valor, denominado “offset”, se utiliza para conocer cuántas auditorías se registraron en el periodo anterior. El porcentaje de crecimiento se calcula restando el valor de periodo anterior con el actual, y dividiéndolos entre la cantidad total de auditorías.

En cuanto a la cantidad de modelos auditables, se utilizó la función “get_models” de Django que retorna una lista de todos los modelos de la aplicación. Cada modelo es representado por un objeto que incluye su clase, con el cual se puede contar cuántos heredan el “AuditableMixin”. Siguiendo este mismo procedimiento se puede determinar el porcentaje de cobertura de la librería, dividiendo la cantidad de modelos auditables entre el total de modelos.

Todos los cálculos de estadísticas se colocan en el contexto de la vista “StatisticsView” para que puedan ser utilizados en las plantillas.

6.2.2.3 Construcción de gráficos

Antes de procesar los datos necesarios para crear un gráfico, se investigó en profundidad las funcionalidades que provee Amcharts. Este *plugin*, requiere crear un elemento en HTML con la etiqueta “<div>”, que posea un identificador único (atributo id) con el que se pueda referenciar para construir el gráfico con JavaScript. Para esto, se utiliza la función “Amcharts.makeChart” pasando como parámetros el identificador y un objeto con las opciones para crearlo.

Los gráficos de Amcharts son altamente configurables, pues cuentan con una gran cantidad de opciones que permiten crear comportamientos complejos. Entre estas opciones se destacan:

- **dataProvider**: datos para mostrar el gráfico en formato JSON. Puede contener cualquier cantidad de campos.
- **type**: tipo de gráfico. Los posibles valores de esta opción son: “serial”, “pie”, “xy”, “radar”, “funnel”, “gauge”, “map”, “gantt” y “stock”.
- **theme**: tema o estilo del gráfico. Amcharts posee temas propios creados en JavaScript y algunos de estos son “chalk” “black” “light”.
- **balloon**: crea los mensajes que aparece al posicionar el cursor sobre los ítems del gráfico. Amcharts crea automáticamente la información que aparece, sólo se necesita ajustar la apariencia.
- **graphs**: crea la visualización de los datos en los siguientes subtipos de gráfico: “line”, “column”, “step line”, “smoothed line”, “ohlc” y “candlestick”. Cuenta con varias opciones de personalización, incluyendo la posibilidad de modificar la información mostrada en el “balloon”
- **legend**: leyenda del gráfico en cuestión.
- **export**: habilita un menú que permite exportar los datos en diferentes formatos.
- **responsive**: ajusta el tamaño del gráfico dependiendo de la pantalla.

Para construir los gráficos con Amcharts, se creó una clase llamada “ChartData” que encapsula toda la lógica necesaria para obtener la información relacionada con él. La clase cuenta con atributos como el identificador, tipo, tema, y los datos. Este último, denominado “data”, se inicializa con el resultado de procesar “stats”, y convertirlo en un objeto JSON. De esta manera, puede ser utilizado como “dataProvider”. La mencionada clase puede ser

heredada por el programador y ampliada para ajustarlo a sus necesidades.

La clase “ChartData” es usada para crear una variable en el contexto de la vista “StatisticsView” que posea toda la información, y así, pueda ser utilizada en la plantilla. En la vista se incluyen seis gráficos:

- Gráfico de línea con la evolución de las auditoría.
- Gráficos de torta con la comparación entre cantidad de operaciones por autor, por modelo y por acción.
- Gráfico de columnas apiladas con la comparación de cada operación realizada por cada usuario.
- Gráfico de columnas apiladas con la comparación de cada operación realizada sobre cada modelo.

6.2.2.4 Plantillas del módulo Estadísticas

Dado que el sistema en cuestión es una librería, es sumamente importante que las plantillas generadas sean fáciles de utilizar y personalizar por el programador. Por esta razón, se crearon varias plantillas que pueden agregarse en el sistema “Host”, haciendo uso de la etiqueta “include”. Esta etiqueta permite a Django, cargar y mostrar un plantilla utilizando el contexto, e incluso se puede añadir nuevos valores a este.

Entre las plantillas creadas, la principal es “turpial_statistics.html” que incluye varios componentes:

- El formulario del filtro.
- Todos los cálculos estadísticos.
- Gráfico de línea con la evolución de las auditoría.
- Gráficos de torta con la comparación entre cantidad de operaciones por autor, por modelo y por acción.
- Gráfico de columnas apiladas con la comparación de cada operación realizada por cada usuario.
- Gráfico de columnas apiladas con la comparación de cada operación realizada sobre cada modelo.

Si el programador desea una plantilla con toda la información relacionada con las estadísticas puede incluir esta plantilla en las propias.

En cuanto a los gráficos, se observó que la creación de cada uno era similar; por lo que se decidió fabricar una plantilla reutilizable, “turpial_chart.html”, que recibe como parámetro el objeto de la clase “ChartData”. Con esto, se le ofrece al programador una manera sencilla de crear gráficos de Amcharts. Además, es utilizado en “turpial_statistics.html” para generar los gráficos que ofrece.

Esta idea fue aplicada a los cálculos estadísticos en las plantillas “turpial_cards.html” y “turpial_card_stats.html”. La primera presenta la información general relacionada con la librería, como la cantidad de modelos auditables, porcentaje de cobertura y de crecimiento. La segunda, muestra las estadísticas correspondientes a la cantidad de registros auditables (promedio, máximo, mínimo y total). El programador puede utilizar cada plantilla para presentar la información como mejor le convenga, en caso de que “turpial_statistics.html” no se ajuste a sus necesidades.

Todas las plantillas utilizan Bootstrap para mostrar la información, dado que la empresa posee este *framework* como estándar de desarrollo.

Adicionalmente, se creó un tema propio para los gráficos (“turpial.js”), aprovechando las facilidades que ofrece Amcharts. En este tema, se configuraron los colores de los gráficos, el estilo de las leyendas, el tamaño de las letras, entre otros.

El programador también puede crear su propio tema de Amcharts y utilizar el atributo del “StatisticsView” llamado “chart_theme”. Este requiere un diccionario con el nombre del tema y su ubicación en el proyecto (dirección). La dirección fue incluida dado que el tema debe cargarse después que el JavaScript principal de Amcharts, pero antes de generar el gráfico o no estará disponible. El valor de la variable “chart_theme” se utiliza en el atributo del tema en la clase “ChartData”. En la figura 6.12 se muestra “StatisticsView” incluyendo este campo.

El programador tiene completa libertad de cambiar el resto de los estilos al sobrescribir los de Bootstrap, como lo indica la documentación de la librería.

Luego de culminar las plantillas, se posee un módulo Estadísticas que cumple con todos los requerimientos establecidos. En la figura 6.13 se muestra la estructura final del módulo

```
class StatisticsView(ListView):
    """Handles variables related to statistics."""

    queryset = AuditableAction.objects.all()
    charts_theme = {"name": "turpial", "path": "amcharts/themes/turpial.js"}
```

Figura 6.11: Definición de “StatisticsView” incluyendo el campo “charts_theme”.

Estadísticas y en dónde se ubica cada función del mismo.

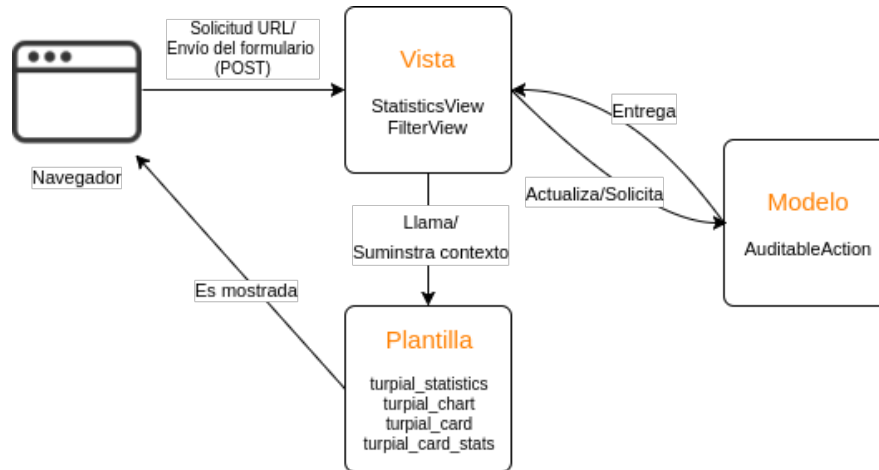


Figura 6.12: Estructura del módulo Estadísticas.

6.2.2.5 Pruebas del módulo Estadísticas

Para el módulo Estadísticas se escribieron 106 casos de prueba, utilizando los mismos modelos que en el módulo *Core* y el mismo *Fixture*. De estos casos, 73 corresponden al formulario de filtros y 33 a los cálculos estadísticos en la vista “StatisticsView”.

Las pruebas del formulario se enfocan en determinar si es válido, y si lo es, proceder a filtrar el *queryset* inicial según los valores provistos. Para comparar que los resultados obtenidos en el *queryset* son los esperados, se utilizó la función “filter” de Python sobre la lista de todos los elementos de la tabla. También, se incluyeron los casos de pruebas en el que el formulario es inválido, en ellos se verifica que ocurra una excepción que incluya un mensaje de error que notifique al usuario. Esto permite asegurar que los filtros funcionan correctamente. Al ejecutar estas pruebas, se obtuvo que todas fueron exitosas.

Para los cálculos estadísticos se deben probar las funciones que entregan estos resultados, haciendo una solicitud POST al URL que llama a la vista “FilterView”. Para esto, se utilizó “Client()” una clase de Django que permite enviar la petición, simulando el navegador. De

esta manera se puede inspeccionar el contexto e incluso el HTML de la respuesta, y así constatar que los valores son los esperados.

Al ejecutar estas pruebas fallaban todas, excepto aquellas que obtenían los resultados sin filtrar. Esto sucedió debido a que, en el “Client()”, no se enviaban correctamente los datos del formulario que están relacionados con el tiempo. Para corregir el error, todos los valores fueron convertidos a *string*.

Luego, fallaron las relacionadas con el filtro de “acción”. No se estaba calculando el “offset” en este caso, por lo que el porcentaje de crecimiento era incorrecto. Se corrigió el código y las pruebas fueron exitosas.

Otro problema que surgió fue que, como el promedio cambia a medida que pasan los días cuando no se envía la fecha de fin, las pruebas comenzaron a fallar. Para corregir esto, se creó una función en lugar de colocar un valor fijo con el cual comparar el promedio, como se estaba haciendo con el resto de las estadísticas. Esta función divide la cantidad de registros esperados entre días transcurridos para calcular el promedio. Los días transcurridos fueron obtenidos con la función “days_elapsed”. Luego se obtuvo que las pruebas fueron exitosas.

Finalmente, se crearon casos de pruebas en los que se genera un modelo al momento de ejecutar el caso en particular. Con la intención de validar que se calculan correctamente la cantidad de modelos auditables y el porcentaje de cobertura. Estas pruebas resultaron exitosas.

6.3 Fase de transición

En esta sección, se explica el proceso de integración de la librería en el proyecto Turpial Calendar, los errores encontrados y sus correcciones. Así como la puesta en producción de ese sistema. Este proceso duró una semana.

6.3.1 Integración de la librería con Turpial Calendar

Antes de iniciar el proceso de integración, se descargó el proyecto Turpial Calendar y se instalaron los paquetes de los cual depende. Este proyecto utiliza Django 1.11 y MySQL.

A continuación, se instaló la librería Auditorías Turpial *Core* (turpial_auditor); utilizando la facilidad que ofrece PIP para esto, a través del enlace del repositorio. Este proceso se repitió para Auditorías Turpial UI (turpial_auditor_ui).

Con respecto a la configuración de las librerías, se siguió paso a paso la documentación correspondiente. En primer lugar, se incluyeron en “INSTALLED_APPS” los módulos necesarios para registrar las auditorías y para visualizar las estadísticas. Luego, agregaron modelos auditables utilizando el *mixin* para esto. Los modelos que presentan auditorías son: Eventos, Proyectos, Empleado y Categoría (de un calendario).

De estos modelos, se ubicaron las vistas que permiten crear y eliminarlos y se incluyó el *mixin* para agregar el usuario en la traza de auditorías, “AuditableMixinView”.

Después, se incluyó un URL en Turpial Calendar que llamase al “FilterView” y que permitiera ver la plantilla de estadísticas. Se repitió este proceso para crear otra, que utilice los estilos de Turpial Calendar y reorganice la información de otra manera, con la finalidad de probar esta funcionalidad.

Errores encontrados Se ejecutó el proyecto y se intentó crear un evento en el calendario para verificar que las auditorías se estaban realizando. Sin embargo, al tratar de enviar el formulario, ocurrió una excepción relacionada con el campo “ForeignKey”. Django, por defecto, incluye en los modelos un campo oculto para realizar una referencia “hacia atrás” al modelo con el cual esta relacionado. Al excluir los campos “ManyToManyField” en el *serializer*, se utiliza una función que permite determinar cuáles son y se accede al nombre de dicho atributo. Este campo oculto, no posee la función que accede al nombre y esto ocasionaba la falla.

Una vez detectado el error, se corrigió y se constató que el *Core* funcionaba como se esperaba en Turpial Calendar. Luego, se escribieron tres nuevas pruebas que validan que el error fue solventado para el caso general. Para esto, se agregaron dos nuevas tablas en la base de datos de prueba: Compañía y Persona, como se muestra en la figura 6.14. Estas pruebas fueron exitosas.

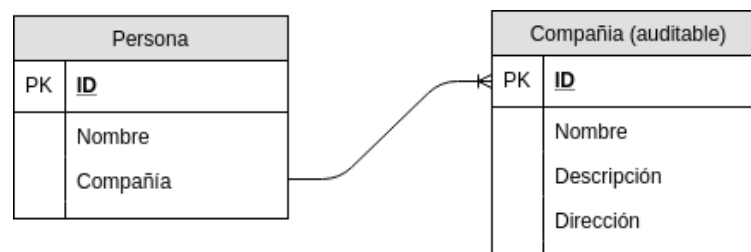


Figura 6.13: Diagrama Entidad-Relación de los nuevos modelos en las pruebas.

En el módulo Estadísticas, se encontró un error en el cálculo de la cantidad de días transcurridos entre la fecha de inicio y la de fin. Cuando no se envía la fecha de inicio en el formulario, se asigna la fecha del primero objeto auditable. Sin embargo, si es borrado de la base de datos, no se obtiene ninguna fecha para realizar el cálculo. Esto se solucionó utilizando el *queryset* ya filtrado y obteniendo el primer registro que posee.

6.3.2 Puesta en producción

Finalizado el proceso de integración, se hizo el despliegue del proyecto Turpial Calendar en el servidor de Turpial Development, Webfaction. Para esto, se ingresó al sistema y se creó un sitio web, agregando el dominio en el que estaría disponible el proyecto. Después, se configuró el tipo de aplicación (Django), se creó la base de datos con su respectivo usuario y contraseña y se migró.

Para clonar el repositorio de Turpial Calendar, se utilizó el protocolo SSH para ingresar al servidor y crear el ambiente virtual del proyecto. Se instalaron las dependencias y las librerías “turpial_auditor” y “turpial_auditor_ui” desde su respectivos repositorios.

A continuación, se configuró el servidor web, Apache, asignándole un nombre, un alias y una tarea para activarlo (*daemon*).

Finalmente, se comprobó que el sistema funcionara correctamente, incluyendo las librerías y todas las funciones relacionadas con ellas.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Brindar soluciones oportunas a la medida de sus clientes, constituyen parte de la misión de Turpial Development, es por ello que inició el desarrollo de una librería para la gestión de auditorías. Esta librería se realizó en Django y es capaz de registrar las operaciones efectuadas sobre la base de datos y mostrarlas a través de estadísticas.

La librería Auditorías Turpial, se desarrolló bajo la arquitectura de microservicio, con lo cual se fomenta su escalabilidad. En su módulo *Core* es posible seleccionar los modelos auditables de manera sencilla mediante el uso de los *mixins* que provee, y generar las auditorías correspondientes según las acciones ejecutadas sobre los mismos, lo que permite llevar la trazabilidad de la información sensible.

El análisis de la información y toma de decisiones por parte del cliente es posible con el uso de esta librería, pues cuenta con un módulo que proporciona resultados estadísticos y facilita la visualización de los resultados capturados por el *Core*, a través de gráficos. El módulo Estadísticas posee plantillas que son fáciles de utilizar por los programadores en cualquier proyecto que use estas tecnologías.

La implantación de un servidor de integración continua fue beneficioso para el desarrollo de este proyecto, puesto que permite unificar las diferentes versiones que poseen los programadores y detectar errores en ellas. También, facilita la automatización de tareas como el despliegue y actualización de la librería, a través de un *script*.

El conjunto de objetivos establecidos para este proyecto de pasantía, fueron alcanzados exitosamente, cumpliendo los estándares, expectativas y lineamientos de la empresa.

Adicionalmente, al realizar las pruebas de desempeño se pudo comprobar que, agregar índices a la tabla de auditorías incrementó el tiempo de inserción aunque no es perceptible por el usuario. Sin embargo, esta decisión permite agilizar las consultas.

Dada las facilidades que ofrece Python para crear clases de manera dinámica, se implementó un un prototipo que mostró que es posible generar tablas por modelo auditable, y con ello, descentralizar las tablas de auditorías con la intención de evitar la sobrecarga que tiene el uso de una tabla centralizada.

Las Historias de Usuario relacionadas con la permisología no fueron llevadas a cabo, sin embargo, se realizó el diseño de un módulo capaz de manejar los privilegios sobre el uso del sistema, el cual puede servir como referencia para la creación del mismo en una versión futura de la librería.

Recomendaciones

Además de los resultados obtenidos, se proponen sugerencias y nuevas ideas a ser consideradas por la empresa para mejorar el sistema:

- Cambiar el identificador numérico del registro de auditoría por un, Identificador Único Universal (UUID, por sus siglas en inglés *Universally Unique Identifier*). De esta manera, se incrementa la cantidad de claves primarias que se pueden direccionar (16^{32}), actualmente es de 2^{32} . Con esto, se evita tener que respaldar la información tan frecuentemente.
- Ampliar las posibles acciones auditables del sistema, incluyendo otras como CREATE TABLE, ALTER TABLE, DROP TABLE para tener un mayor alcance en las auditorías.
- Implementar el módulo de Permisología para restringir funcionalidades de la librería a ciertos usuarios.
- Comparar el desempeño del prototipo de tablas de auditorías descentralizado, con el que está actualmente implementado en la librería.
- Registrar auditorías al crear, actualizar o eliminar datos a través de “bulk_create”, “bulk_update” o “bulk_delete” respectivamente.

REFERENCIAS

- [1] amCharts. Amcharts. <https://www.amcharts.com/>, 2017. Consultado el 28/10/2017.
- [2] Apple Inc. Model-view-controller. <https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>, 2015. Consultado el 27/08/2017.
- [3] Berners, L. Hypertext markup language - 2.0. <https://tools.ietf.org/html/rfc1866>, 1995. Consultado el 15/07/2017.
- [4] Bracha, G. Cook, W. Mixin-based inheritance. <http://www.bracha.org/oops1a90.pdf>, 1990. Consultado el 29/09/2017.
- [5] Bray, Ed. Json. <https://tools.ietf.org/html/rfc7159>, 2014. Consultado el 17/07/2017.
- [6] Django Software Foundation. Faq: General. <https://docs.djangoproject.com/es/1.11/faq/general/>, 2017. Consultado el 27/08/2017.
- [7] Django Software Foundation. Faq: General. <https://docs.djangoproject.com/en/1.11/topics/signals/>, 2017. Consultado el 28/08/2017.
- [8] Dragoni, N. Giallorenzo, S. Microservices: yesterday, today, and tomorrow. <https://arxiv.org/pdf/1606.04036.pdf>, 2017. Consultado 28/07/2017.
- [9] Esmite, I. Farías, M. Automatización y gestión de las pruebas funcionales usando herramientas open source. http://sedici.unlp.edu.ar/bitstream/handle/10915/21787/Documento_completo.pdf?sequence=1. Consultado el 25/07/2017.
- [10] Esterbrook, C. Using mix-ins with python. <http://www.linuxjournal.com/article/4540>, 2001. Consultado el 29/09/2017.
- [11] Fowler, M. Continuous integration. <https://www.martinfowler.com/articles/continuousIntegration.html>, 2006. Consultado el 28/07/2017.
- [12] Git. Git. <https://git-scm.com/>, 2017. Consultado el 17/07/2017.
- [13] Jasper, B. Django-jsonfield. <https://pypi.python.org/pypi/jsonfield>, 2017. Consultado el 04/12/2017.
- [14] Jenkins CI. Jenkins. <https://jenkins.io/doc/>, 2017. Consultado el 18/10/2017.
- [15] jQuery. Jquery. <https://jquery.com/>, 2017. Consultado el 10/11/2017.

- [16] Microsoft. Model-view-controller. <https://msdn.microsoft.com/en-us/library/ff649643.aspx>. Consultado el 27/08/2017.
- [17] Mozilla. Javascript. <https://developer.mozilla.org/es/docs/Web/JavaScript,1995>. Consultado el 15/07/2017.
- [18] Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015.
- [19] Oracle. The main features of mysql. <https://dev.mysql.com/doc/refman/5.7/en/features.html>, 2017. Consultado el 04/11/2017.
- [20] Otto, M. Bootstrap. <https://getbootstrap.com/>, 2017. Consultado el 10/11/2017.
- [21] Pytest. Pytest. <https://docs.pytest.org/en/latest/>, 2017. Consultado el 17/07/2017.
- [22] Python Software Foundation. The python tutorial. <https://docs.python.org/3/tutorial/index.html>, 2017. Consultado el 15/07/2017.
- [23] SpryMedia Ltd. Django-jsonfield. <https://datatables.net/>, 2017. Consultado el 10/11/2017.
- [24] SQLite. About sqlite. <https://www.amcharts.com/>, 2017. Consultado el 04/11/2017.
- [25] Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition, 2014.
- [26] The PostgreSQL Global Development Group. About postgresql. <https://www.postgresql.org/about/>, 2017. Consultado el 28/10/2017.
- [27] Turpial Development. Manual informativo de turpial development, 2016. Documento Confidencial.
- [28] M.B. Weinstein. *Total Quality Safety Management and Auditing*. Taylor & Francis, 1997.

APÉNDICE A

Manifiesto Ágil

Es un estatuto que promueve modelos organizacionales basados en la colaboración entre personas sin documentar de manera excesiva el proceso. Posee sistema que se adapta a las necesidades del cliente y lo incluyen de manera activa en el proceso de desarrollo. Se rige por los siguientes 12 principios.

1. La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
3. Aceptar que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
4. Entregar software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
5. Los responsables de negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.
6. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
7. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
8. El software funcionando es la medida principal de progreso.
9. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
10. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
11. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

12. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
13. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

APÉNDICE B

Manual de metodología de Turpial (resumen)

Descripción de la metodología TAUP

TAUP o Turpial Agile Unified Process es una metodología ágil creada por la empresa Turpial Development basada en los principios del "manifiesto ágil" que brinda a su equipo de trabajo una manera eficaz de llevar a cabo el desarrollo de software.

Fase de Concepción

El objetivo de esta etapa es que el equipo de desarrollo profundice la comprensión de los requisitos del sistema y validación de la arquitectura propuesta.

Historias de Usuario (HU)

Es una técnica para expresar requerimientos que se caracteriza por ser sencilla de escribir, leer y evaluar. El usuario final o cliente le dará valor a esta HU ya que describe una funcionalidad que él necesita.

Las historias de usuarios se escriben en oraciones que poseen tres componentes: el rol del usuario que quiere realizar la acción, que desea hacer y por qué. Una buena HU está caracterizada por ser independiente, negociable, valiosa, estimable, pequeña y comprobable (INVEST).

Adicionalmente, deben contar con criterios de aceptación bien definidos que brinden más detalle respecto a los aspectos de la Historia de Usuario que validan su completitud. Deben ser escritos en lenguaje natural y cada oración debe decir una funcionalidad que tendrá que ser ejecutada por la HU.

Por último, una HU, debe contar con pruebas de aceptación ya que van a permitir que el producto cumpla con los estándares y que se satisfagan los requerimientos establecidos.

Corresponden a una condición que se pueda verificar de los criterios de aceptación y una vez que han sido validadas, se puede dar por culminada la Historia de Usuario.

Priorizar Historias de Usuario

El cliente establecerá una prioridad para las Historias de usuario, la cual es uno de los factores que influyen en el orden de ejecución de las mismas. Se puede utilizar la siguiente escala de cinco valores: Alta, Media alta, Media, Media Baja y Baja.

Análisis de riesgo de Historias de Usuario

El equipo de desarrollo debe establecer un nivel de dificultad para cada HU, para esto, los desarrolladores deben tomar en cuenta diversos factores como: aprender nuevas tecnologías, manejar herramientas desconocidas o investigar un tema en particular. La escala cuenta con cinco valores: Alto, Medio Alto, Medio, Medio Bajo, Bajo.

Para este proyecto se utilizó, al igual en la sección anterior, una escala reducida para puntuar el riesgo: Alto, Medio y Bajo.

Análisis de Esfuerzo de Historias de Usuario

Representa una estimación de cuánto esfuerzo se requiere para culminar una Historia de Usuario, a esto se le denominará Story Points. Al momento de asignar los Story Points se deben considerar diversos factores como el riesgo, complejidad y desconocimiento de la tarea a realizar, entre otros. La escala de ponderación viene dada por la secuencia de Fibonacci.

Cuando una Historia de Usuario consigue un Story Point muy alto (usualmente un valor mayor a 8), deberá ser revisada nuevamente ya que podría dividirse en varias Historias de Usuario que puedan más manejables. Cada Story Point corresponde a lo que se le denomina un Login Point que posee el valor de uno y corresponde a un módulo de autenticación de usuarios de un sistema.

El equipo de desarrollo debe estimar de manera consensuada los Story Point a través de una dinámica denominada Planning Poker; cada miembro puntuará las Historias de Usuario, se eliminarán el valor más alto y el más bajo, el resto se promedian.

Backlog

A la lista de todas las Historias de Historia de Usuarios construidas a partir de toda la información suministrada en los puntos anteriores se le denominará Backlog, al construirla se determina el orden se van a ejecutar tomando en cuenta la prioridad y el riesgo de la tarea, sin embargo, puede cambiar.

Release Plan

Es un calendario en el cual se agregarán los Sprint que sean necesarios para que todas las Historias de Usuarios sean cumplidas a cabalidad.

Definición de Listo “DoR”

DoR (Definition Of Ready) definido así por sus siglas en inglés, es un acuerdo que se establece en el equipo de desarrollo para cada Historia de Usuario que responde a la pregunta ‘¿Qué tiene que estar listo antes de empezar a trabajar en una Historia de Usuario y la misma pueda ser ejecutada en el próximo Sprint?’.

Se debe tener en cuenta los siguientes aspectos:

- Que sea independiente o de lo contrario que la(s) Historia de Usuario que lo preceden ya están implementadas o van a desarrollarse en el mismo Sprint.
- Que la Historia de Usuario tiene que ser desarrollada en un solo Sprint.
- Que los criterios de aceptación para la Historia de Usuario sean detallados
- Que las pruebas de aceptación puedan realizarse en el Sprint.
- Que la Historia de Usuario tenga asignado su Story Point.
- Tomar en cuenta si existe alguna dependencia externa para la Historia de Usuario.

Definición de Terminado “DoD”

DoD (Definition of Done) definido así por sus siglas en inglés, responde a la pregunta “¿Qué tiene que cumplir una Historia de Usuario para que se considere finalizada?”, también puede incluir otros aspectos que sean considerados por el equipo de desarrollo como primordiales para que una HU sea considerada terminada. Este acuerdo explica con detalle los criterios para que toda HU sea considerada terminada.

- La Historia de Usuario ya ha sido analizada.
- El código tiene que estar escrito.
- El código tiene que estar documentado o comentado.
- El código se ha integrado a todo lo desarrollado con anterioridad.
- La Historia de Usuario ha pasado por todas sus pruebas de aceptación con éxito.
- Haberle mostrado la Historia de Usuario al cliente y que este la acepte.

Fase de Construcción

El enfoque de esta fase es desarrollar todo lo planteado en la Concepción, validando así, la arquitectura planteada. La metodología se basa en Sprints o iteraciones.

Sprints

El desarrollo del software se tiene que ejecutar por bloques de duración corta y fija, de 5 días (semana laboral) hasta 1 mes. Cada Sprint tiene que proporcionar un resultado concreto, un incremento de producto que sea potencialmente entregable, de tal manera que pueda estar disponible para ser utilizado por el cliente y así, proporcionar observaciones de lo que se le está mostrando.

El primer día de cada Sprint se lleva a cabo la Reunión de Planificación, en la que todos los miembros del equipo revisan lo que se tiene planteado en el Release Plan para ese Sprint, verifican que cada una de esas Historias de Usuario estén en estado DoR para que sean divididas en tareas y se aclare cualquier duda acerca de lo que se va a realizar. Luego de que se conozcan cuáles son las tareas a desarrollar, se distribuyen a cada miembro del equipo y se colocan en un Kanban Board para saber cuáles están por hacer (To Do), cuales se están haciendo (Doing) y cuales están finalizadas (Done).

Durante el Sprint se debe realizar todos los días el Daily Stand Ups Meeting cuyos integrantes son el equipo desarrollador y su líder de proyecto. Tienen un horario (no debería de durar más quince minutos) y lugar fijo y los participantes tendrán tiempo para hablar y responder tres preguntas: qué ha realizado desde el último Daily, qué se está haciendo y qué bloquea su progreso.

El último día de cada Sprint, los miembros del equipo se deben reunir para realizar una Revisión de Iteración, en la cual estará presente el cliente para mostrarle todos los avances

y que dé su opinión para que el equipo realice las correcciones pertinentes lo más pronto posible. Luego, se realiza la Reunión de Retrospectiva, en donde se discute que hizo bien, que se puede mejorar y a que se compromete

Para cada Sprint es necesario ver qué cantidades de Story Points se van a realizar y cuáles se están trabajando y cuales están ya en DoD. Para esto se crea una gráfica que sirve para saber si se puede o no terminar lo planteado para el Sprint, llamada Burndown Chart, en la cual se coloca en el eje X el tiempo que dura el Sprint y en el eje Y la cantidad de Story Points. Esta práctica también sirve para referirse al avance de todo el proyecto, haciendo el mismo procedimiento antes explicado, pero cambiando los ejes de manera que en ellos se encuentre el total de Story Point del desarrollo del proyecto y todos los Sprint del mismo.

Fase de Transición

En esta fase se probará todo lo desarrollado en la fase de construcción. Estas pruebas funcionales y no funcionales verifican que el proyecto pueda ser utilizado en un ambiente de producción, además se creará un manual para el usuario, el cual le dará la capacitación necesaria al cliente para poder utilizar sin problemas el sistema.

Pruebas

Se realiza la validación del sistema a través de pruebas de integración, despliegue y comportamiento y que el mismo pueda ser desplegado en un ambiente de preproducción. También se validará la documentación, en donde se verificará que todo el código esté correctamente documentado para que pueda ser entendido con facilidad en caso de que sea necesario realizar cambios. Se crea el Manual de Usuario, en el que se especifica todas las funcionalidades del sistema para que el cliente pueda consultar si tiene alguna duda con respecto a la funcionalidad.

Por último, se verifica que todo lo anterior se haya cumplido, de no ser así se tendrá que hacer pruebas de regresión y modificaciones a la validación de la documentación hasta que el cliente esté satisfecho con el trabajo realizado.

Puesta en Producción

Luego de que el sistema esté completo, se tendrá que desplegar en el ambiente de producción, al terminar este proceso se otorgará un tiempo prudencial al cliente para efectuar pruebas. El reporte de estos defectos será más formal ya que llevarán un registro, junto con sus detalles para que los desarrolladores puedan corregirlos.


APÉNDICE C

Release Plan

Librería Auditorías Turpial

Plan de Proyecto

Versión 1.0

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 11/07/2017	
LAT_v<1.0>		

Historial de Revisiones

Fecha	Versión	Descripción	Autor(es)
30/07/2017	<0.1>	<i>Versión Inicial del Plan de Proyecto.</i>	<i>Arleyn Goncalves y Stefani Castellanos</i>
07/08/2017	<0.2>	<i>Versión revisada del Plan de Proyecto.</i>	<i>Arleyn Goncalves y Stefani Castellanos</i>
11/08/2017	<1.0>	Versión aprobada por el Tutor Industrial.	<i>Arleyn Goncalves y Stefani Castellanos</i>

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 11/07/2017	
LAT_v<1.0>		

Propósito

El propósito de este documento es establecer el conjunto de historias de usuario (o épicas) que componen el proyecto Librería Auditorías Turpial. Este plan sirve como referencia para decidir cuánto se debe desarrollar, cuánto tiempo se tardará antes de tener un producto entregable y cómo podría evolucionar el producto en cada Sprint.

Referencias

- Documento de Requerimientos.


Épicas

ID	Epica	Total de historias
C	Módulo Principal (Core)	15
E	Módulo Estadísticas	8
R	Módulo Reportes	12

Backlog

Módulo principal (Core)

ID	Historia
C01	Como programador, quiero indicar cuáles elementos de mi base de datos quiero auditar para poder indicar a la librería cuales modelos debe llevarle seguimiento.
C02	Como programador, quiero poder almacenar cuando un elemento auditable fue creado así podré saber cuándo, quién y qué fue ingresado en el sistema.
C03	Como programador, quiero poder almacenar cuando un elemento auditable fue editado así podré saber cuándo, quién y cómo fue alterada la información en el sistema.
C04	Como programador, quiero poder almacenar cuando un elemento auditable fue eliminado así podré saber cuándo, quién y cuál la información fue borrada en el sistema.
C05	Como usuario, quiero disponer de listas de cada elemento auditable para poder observar el comportamiento de mi información.

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 11/07/2017	
LAT_v<1.0>		

C06	Como usuario, quiero que las listas de mis auditorías puedan ordenarse según autor, fecha de creación, acción efectuada sobre la data
C07	Como usuario, quiero que las listas de mis auditorías puedan filtrarse según autor, fecha de creación, acción efectuada sobre la data; para así facilitar la búsqueda de auditorías en específico.
C08	Como programador, quiero poder configurar el orden en que se desplegarán las columnas de mis listados, así mantener los patrones de experiencia de usuario de mi sistema.
C09	Como programador, quiero contar con etiquetas personalizadas que me permitan incluir cualquier listado de las auditorías en mi sistema
C10	Como programador, quiero contar con un medio para personalizar el Look and Feel de mis listados de auditorías para ajustarlos al estilo de mi sistema.
C11	Como usuario, quiero contar con un control de acceso a los listados de las auditorías y así solo brindar a un conjunto de usuarios las auditorías (Dividida en C11.1, C11.2 y C11.3)
C11.1	Como usuario, quiero contar con un login para restringir el acceso a usuarios no autorizados a mis auditorías.
C11.2	Como usuario, quiero poder autorizar/desautorizar usuarios al acceso de auditorías.
C11.3	Como usuario, quiero poder listar a todos los usuarios autorizados a acceder a las auditorías.
C12	Como usuario, quiero que el sistema se actualice con la última versión de la librería por medio de integración continua para garantizar el soporte al sistema en donde se instale la librería.
C13	Como desarrollador quiero que se instalen automáticamente las dependencias necesarias en el sistema que se instale la librería.

Módulo de Estadísticas


ID	Historia
E01	Como usuario quiero ver las estadísticas globales entre todas las acciones auditables a lo largo de un rango de tiempo.
E02	Como usuario quiero disponer de gráficas con las estadísticas globales.

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 11/07/2017	
LAT_v<1.0>		

E03	Como usuario quiero ver las estadísticas por acción auditable a lo largo de un rango de tiempo.
E04	Como usuario quiero disponer de gráficas con las estadísticas por acción auditable.
E05	Como usuario quiero ver las estadísticas por autor a lo largo de un rango de tiempo.
E06	Como usuario quiero disponer de gráficas con las estadísticas por autor.
E07	Como usuario quiero ver las estadísticas de una tabla de auditoría específica de la base de datos a lo largo de un rango de tiempo.
E08	Como usuario quiero disponer de gráficas con las estadísticas de una tabla de auditoría específica.

Módulo de Reportes

ID	Historia
R01	Como usuario quiero generar reportes CSV de los diferentes listados de la auditoría del sistema.
R02	Como usuario quiero generar reportes PDF de los diferentes listados de la auditoría del sistema.
R03	Como usuario quiero generar reportes CSV con las listas de mis auditorías, donde se muestre fecha, hora, autor y acción.
R04	Como usuario quiero generar reportes PDF con las listas de mis auditorías, donde se muestre la fecha, hora, autor y acción.
R05	Como usuario, quiero personalizar el orden de las columnas de mis reportes CSV.
R06	Como usuario, quiero personalizar el orden de las columnas de mis reportes PDF.
R07	Como usuario quiero personalizar los reportes PDF de los listados con la incorporación del logo del sistema.
R08	Como usuario quiero personalizar los reportes PDF de los listados con la incorporación del nombre del sistema.
R09	Como usuario quiero personalizar los reportes PDF de los listados en referencia al <i>Look and Feel</i> del sistema.
R10	Como usuario quiero llevar control de los reportes generados por el sistema, así conocer

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 11/07/2017	
LAT_v<1.0>		

	quién y cuándo se solicitó un reporte al sistema.
R11	Como usuario quiero seleccionar cuales listados de la auditoría se pueden mostrar en los reportes CSV generados.
R12	Como usuario quiero seleccionar cuales listados de la auditoría se pueden mostrar en los reportes PDF generados.

Sprint 1

ID	Historia	Prioridad	Dependencia	Riesgo	Story Points
C01	Como programador, quiero indicar cuáles elementos de mi base de datos quiero auditar para poder indicar a la librería cuales modelos debe llevarle seguimiento.	Alta	-	Alto	5
C02	Como programador, quiero poder almacenar cuando un elemento auditable fue creado así podré saber cuándo, quién y qué fue ingresado en el sistema.	Alta	C01	Alto	5

Sprint 2

ID	Historia	Prioridad	Dependencia	Riesgo	Story Points
C03	Como programador, quiero poder almacenar cuando un elemento auditable fue editado así podré saber cuándo, quién y cómo fue alterada la información en el sistema.	Alta	C01	Alto	5
C04	Como programador, quiero poder almacenar cuando un elemento auditable fue eliminado así podré saber cuándo, quién y cuál la información fue borrada en el sistema.	Alta	C01	Alto	5


Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 11/07/2017	
LAT_v<1.0>		

Sprint 3

ID	Historia	Prioridad	Dependencia	Riesgo	Story Points
C05	Como usuario, quiero disponer de listas de cada elemento auditable para poder observar el comportamiento de mi información	Alta	C02, C03 y C04	Bajo	2
C12	Como usuario, quiero que el sistema se actualice con la última versión de la librería por medio de integración continua para garantizar el soporte al sistema en donde se instale la librería.	Alta	C01	Medio	5
C13	Como desarrollador quiero que se instalen automáticamente las dependencias necesarias en el sistema que se instale la librería.	Alta	C01	Medio	3

Sprint 4

ID	Historia	Prioridad	Dependencia	Riesgo	Story Points
C06	Como usuario, quiero que las listas de mis auditorías puedan ordenarse según autor, fecha de creación, acción efectuada sobre la data.	Alta	C05	Bajo	1
C07	Como usuario, quiero que las listas de mis auditorías puedan filtrarse según autor, fecha de creación, acción efectuada sobre la data.	Alta	C05	Bajo	1
C09	Como programador, quiero contar con etiquetas personalizadas que me permitan incluir cualquier listado de las auditorías en mi sistema.	Alta	C05	Medio	3


Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 11/07/2017	
LAT_v<1.0>		

Sprint 5

ID	Historia	Prioridad	Dependencia	Riesgo	Story Points
C08	Como usuario, quiero poder configurar el orden en que se desplegarán las columnas de mis listados, así mantener los patrones de experiencia de usuario de mi sistema.	Media	C05	Medio	2
C10	Como programador, quiero contar con un medio para personalizar el <i>Look and Feel</i> de mis listados de auditorías para ajustarlos al estilo de mi sistema.	Media	C05	Alto	5
C11.1	Como usuario, quiero contar con un login para restringir el acceso a usuarios no autorizados a mis auditorías.	Baja	C05	Medio	2

Sprint 6

ID	Historia	Prioridad	Dependencia	Riesgo	Story Points
C11.2	Como usuario, quiero poder autorizar/desautorizar usuarios al acceso de auditorías.	Baja	C05	Medio	5
C11.3	Como usuario, quiero poder listar a todos los usuarios autorizados a acceder a las auditorías.	Baja	C05	Medio	1
E01	Como usuario quiero ver las estadísticas globales entre todas las acciones auditables a lo largo de un rango de tiempo.	Alta	C02, C03 y C04	Medio	2
R01	Como usuario quiero generar reportes CSV de los diferentes listados de la auditoría del sistema.	Alta	C05	Medio	2
R03	Como usuario quiero generar reportes CSV con las listas de mis auditorías, donde se muestre fecha, hora, autor y acción.	Media	R01	Medio	2


Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 11/07/2017	
LAT_v<1.0>		

Sprint 7

ID	Historia	Prioridad	Dependencia	Riesgo	Story Points
E02	Como usuario quiero disponer de gráficas con las estadísticas globales.	Media	E01	Medio	2
E03	Como usuario quiero ver las estadísticas por acción auditable a lo largo de un rango de tiempo.	Alta	C02, C03 y C04	Medio	2
E04	Como usuario quiero disponer de gráficas con las estadísticas por acción auditable.	Media	E03	Medio	2
E05	Como usuario quiero ver las estadísticas por autor a lo largo de un rango de tiempo.	Alta	C02, C03 y C04	Medio	2
R02	Como usuario quiero generar reportes PDF de los diferentes listados de la auditoría del sistema.	Media	C05	Medio	3
R04	Como usuario quiero generar reportes PDF con las listas de mis auditorías, donde se muestre la fecha, hora, autor y acción.	Media	R02	Medio	3
R05	Como usuario, quiero personalizar el orden de las columnas de mis reportes CSV.	Media	R01	Medio	2
R06	Como usuario, quiero personalizar el orden de las columnas de mis reportes PDF.	Media	R01	Medio	2

Sprint 8

ID	Historia	Prioridad	Dependencia	Riesgo	Story Points
E06	Como usuario quiero disponer de gráficas con las estadísticas por autor.	Media	E05	Medio	1
E07	Como usuario quiero ver las estadísticas de una tabla de auditoría específica de la base de datos a lo largo de un rango de tiempo.	Alta	C02, C03 y C04	Medio	3

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 11/07/2017	
LAT_v<1.0>		

E08	Como usuario quiero disponer de gráficas con las estadísticas de una tabla de auditoría específica.	Media	E07	Medio	1
R07	Como usuario quiero personalizar los reportes PDF de los listados con la incorporación del logo del sistema.	Media	R02	Medio	2
R08	Como usuario quiero personalizar los reportes PDF de los listados con la incorporación del nombre del sistema.	Media	R02	Alto	0.5
R09	Como usuario quiero personalizar los reportes PDF de los listados en referencia al <i>Look and Feel</i> del sistema.	Media	R02	Alto	5
R10	Como usuario quiero llevar control de los reportes generados por el sistema, así conocer quién y cuándo se solicitó un reporte al sistema.	Baja	R01 y R02	Medio	1


Sprint 9

ID	Historia	Prioridad	Dependencia	Riesgo	Story Points
R11	Como usuario quiero seleccionar cuales listados de la auditoría se pueden mostrar en los reportes CSV generados.	Baja	R01	Medio	2
R12	Como usuario quiero seleccionar cuales listados de la auditoría se pueden mostrar en los reportes PDF generados.	Baja	R02	Medio	3

APÉNDICE D

Plan de Pruebas

Librería Auditorías Turpial
Documento de Plan de Pruebas
Versión 1.0

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 15/07/2017	
LAT_v<1.0>		

Historial de Revisiones

Fecha	Versión	Descripción	Autor(es)
07/08/2017	<0.1>	<i>Versión Inicial del Plan de Pruebas.</i>	<i>Arleyn Goncalves y Stefani Castellanos</i>
11/08/2017	<0.2>	<i>Versión revisada del Plan de Pruebas.</i>	<i>Arleyn Goncalves y Stefani Castellanos</i>
15/08/2017	<1.0>	Versión aprobada por el Tutor Industrial.	<i>Arleyn Goncalves y Stefani Castellanos</i>

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 15/07/2017	
LAT_v<1.0>		

Propósito

Este documento pretende recopilar toda la información necesaria para planificar las pruebas que se realizarán sobre la librería Auditorías Turpial con la finalidad de entregar un sistema que posea la menor cantidad de fallas posible.

Objetivos

- Detectar errores en etapas tempranas del proyecto.
- Identificar los aspectos en los que se enfocarán los casos de prueba.
- Delimitar el enfoque de las pruebas que será usado.
- Identifica los recursos requeridos para la realización de las pruebas.

Referencias

- Documento de Plan de Proyecto
- Documento de Plan de Requerimientos

Módulos

- Módulo *Core* (Principal)
- Módulo de Reportes
- Módulo de Estadísticas

Usuarios

- Programador
- Usuario


Tipos de pruebas

Tipo	Descripción
Pruebas de Unitarias	Se focaliza en probar cada módulo, lo que provee un mejor modo de manejar la integración de las unidades en componentes mayores.
Pruebas de Regresión	Se vuelven a probar el sistema debido a cambios realizados durante el mantenimiento o desarrollo de la nueva versión del sistema buscando efectos adversos en otras partes.
Pruebas de Integración	Este tipo de pruebas verifican que los componentes de la aplicación funcionan correctamente actuando en conjunto.
Pruebas de Aceptación	La prueba de aceptación es generalmente desarrollada y ejecutada por el cliente o un especialista de la aplicación y es conducida a determinar como el sistema satisface sus criterios de aceptación validando los requisitos que han sido levantados para el desarrollo, incluyendo a documentación y procesos de negocio.
Pruebas de Campo	Correr el sistema en el ambiente real para encontrar errores y validar el producto contra sus especificaciones originales.
Pruebas de Desempeño	Las pruebas de desempeño miden tiempos de respuesta, índices de procesamiento de transacciones y otros requisitos sensibles al tiempo.

Enfoque de las Pruebas

Técnicas y tipos de prueba


Tipo de prueba:	Unitarias.
Técnica:	Comparar el resultado esperado con el resultado obtenido de una función en particular. Si existen errores, reportarlo.
Objetivo de la técnica:	Probar cada una de las funcionalidades de la aplicación.
Herramientas requeridas:	Pytest
Criterios de éxito:	Que todas las pruebas sean exitosas. Cada error encontrado, sea corregido.

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 15/07/2017	
LAT_v<1.0>		

Consideraciones especiales:	No aplica.
------------------------------------	------------

Tipo de prueba:	Regresión.
Técnica:	Repetir los casos de pruebas luego de que se realicen cambios en el código.
Objetivo de la técnica:	Determinar si los cambios recientes en una parte de la aplicación tienen efecto adverso en otras partes.
Herramientas requeridas:	Pytest
Criterios de éxito:	Que todas las pruebas sean exitosas. Cada error encontrado, sea corregido
Consideraciones especiales:	Deben existir pruebas previamente.

Tipo de prueba:	Integración.
Técnica:	Comparar el resultado esperado con el resultado obtenido. Si existen errores, reportarlo.
Objetivo de la técnica:	Probar el comportamiento en conjunto de varias funcionalidades de la aplicación.
Herramientas requeridas:	Pytest
Criterios de éxito:	Que todas las pruebas sean exitosas. Cada error encontrado, sea corregido.
Consideraciones especiales:	No aplica.

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 15/07/2017	
LAT_v<1.0>		

Tipo de prueba:	Aceptación.
Técnica:	Mostrar la aplicación al cliente para validar que funcione como él lo desee.
Objetivo de la técnica:	Probar el comportamiento en conjunto de varias funcionalidades de la aplicación.
Herramientas requeridas:	Pytest
Criterios de éxito:	Que el cliente esté satisfecho con el comportamiento de la aplicación.
Consideraciones especiales:	No aplica.

Tipo de prueba:	De campo.
Técnica:	Utilizar una aplicación en producción e instalar la librería.
Objetivo de la técnica:	Verificar que la librería Auditorías Turpial funcione adecuadamente en el sistema de la empresa.
Herramientas requeridas:	Sistema de la empresa.
Criterios de éxito:	No ocurran errores en el funcionamiento en el sistema de la empresa.
Consideraciones especiales:	No aplica.

Tipo de prueba:	De desempeño.
Técnica:	Crear un <i>script</i> para generar datos, cargarlos en la base de datos y tomar el tiempo que toma auditar cada registro. Se plantean dos escenarios:

Librería Auditorías Turpial	Versión: 1.0	
	Fecha: 15/07/2017	
LAT_v<1.0>		

	sin índices y con índices en el modelo.
Objetivo de la técnica:	Comparar el tiempo que toma una inserción del registro de auditoría.
Herramientas requeridas:	Pytest, Python.
Criterios de éxito:	Que el tiempo agregado por la inserción del registro de auditoría sea despreciable de manera que sea imperceptible al usuario.
Consideraciones especiales:	No aplica.

Alcance de las pruebas

Pruebas a incluir

Las pruebas incluidas en este plan inicial se enfocan en comprobar el correcto funcionamiento de cada historia de usuario desarrollada en el proyecto de pasantía, entre las cuales se implementaran las pruebas unitarias, de regresión, de integración, de aceptación y de campo.

Potencialmente a ser incluidas

Se considera incluir pruebas de desempeño, con la finalidad de evaluar el tiempo de procesamiento de la información bajo las condiciones de que existe volumen de información anticipada. Estas pruebas se ejecutan varias veces, utilizando en cada una carga diferente en el sistema, con el objetivo de verificar el tiempo de procesamiento, mejorar las métricas de desempeño y proyectar la capacidad futura de carga del sistema.

También se consideraron incluir la pruebas *end to end* de manera que se pueda verificar algunas funcionalidades que no se pueden probar implementando pruebas unitarias y que el flujo de distintos componentes funcionan correctamente entre sí.

Pruebas a excluir

Se van a excluir las pruebas de usabilidad, de stress, de carga, de compatibilidad y de seguridad, ya que no forman parte del alcance del proyecto de pasantías y además no tienen alta prioridad para el cliente.

APÉNDICE E

Resultado de las pruebas

APÉNDICE
RESULTADOS DE PRUEBAS AUTOMATIZADAS

Módulo Core

CREAR				
Código	Nombre	Descripción	Resultado Esperado	Resultado
C02-P01	test_simple_element_model	Crear una instancia en el modelo "Cars".	Se crea una instancia en el modelo "AuditableAction" con action=CREATED, model_json_old={} y model_json_new es igual a los datos suministrados.	Exitoso
C02-P02	test_one_to_many_model	Crear una instancia del modelo "Courses" con una referencia al modelo "Teacher".	Se crea una instancia en el modelo "AuditableAction" con action=CREATED, model_json_old={} y model_json_new es igual a los datos suministrados.	Exitoso
C02-P03	test_many_to_many_model	Crear una instancia del modelo "Student" con una lista de referencias al modelo "Courses".	Se crea una instancia en el modelo "AuditableAction" con action=CREATED, model_json_old={} y model_json_new es igual a los datos suministrados.	Exitoso
C02-P04	test_simple_element_view	Crear una instancia en el modelo "Cars", haciendo un post del formulario (desde la vista)	Se crea una instancia en el modelo "AuditableAction" con action=CREATED, model_json_old={} y model_json_new es igual a los datos suministrados.	Exitoso
C02-P05	test_one_to_many_view	Crear una instancia del modelo "Courses" con una referencia al modelo "Teacher", haciendo un post del formulario (desde la vista)	Se crea una instancia en el modelo "AuditableAction" con action=CREATED, model_json_old={} y model_json_new es igual a los datos suministrados.	Exitoso

T01-P01	test_O2M_model_v2	Crear una instancia del modelo "Company" que es referenciado por "Person".	Se crea una instancia en el modelo "AuditableAction" con action=CREATED, model_json_old={} y model_json_new es igual a los datos suministrados.	Exitoso
---------	-------------------	--	---	---------

Nota: Las pruebas PA-C05 y PA-C06 fallaron debido a que no se manejaban apropiadamente los casos con campos Muchos a Muchos.

ACTUALIZAR				
Código	Nombre	Descripción	Resultado Esperado	Resultado
C03-P01	test_simple_element_model	Actualizar una instancia en el modelo "Cars".	Se crea una instancia en el modelo "AuditableAction" con action=UPDATED, model_json_old es igual a los datos inicialmente suministrados y model_json_new es igual a los datos actualizados.	Exitoso
C03-P02	test_simple_element_view	Actualizar una instancia en el modelo "Cars" a través de la "vista".	Se crea una instancia en el modelo "AuditableAction" con action=UPDATED, model_json_old es igual a los datos inicialmente suministrados y model_json_new es igual a los datos actualizados.	Exitoso
C03-P03	test_one_to_many_model_v1	Actualizar una instancia en el modelo "Courses" con una referencia al modelo "Teacher". Se actualizan todos los datos menos la referencia.	Se crea una instancia en el modelo "AuditableAction" con action=UPDATED, model_json_old es igual a los datos inicialmente suministrados y model_json_new es igual a los datos actualizados.	Exitoso

C03-P04	test_one_to_many_view_v1	Actualizar una instancia en el modelo "Courses" con una referencia al modelo "Teacher" a través de la "vista". Actualizando todos los datos menos la referencia.	Se crea una instancia en el modelo "AuditableAction" con action=UPDATED, model_json_old es igual a los datos inicialmente suministrados y model_json_new es igual a los datos actualizados.	Exitoso
C03-P05	test_one_to_many_model_v2	Actualizar una instancia en el modelo "Courses" con una referencia al modelo "Teacher". Se actualiza solo la referencia.	Se crea una instancia en el modelo "AuditableAction" con action=UPDATED, model_json_old es igual a los datos inicialmente suministrados y model_json_new es igual a los datos actualizados.	Exitoso
C03-P06	test_one_to_many_view_v2	Actualizar una instancia en el modelo "Courses" con una referencia al modelo "Teacher" a través de la "vista". Se actualiza solo la referencia.	Se crea una instancia en el modelo "AuditableAction" con action=UPDATED, model_json_old es igual a los datos inicialmente suministrados y model_json_new es igual a los datos actualizados.	Exitoso
C03-P07	test_many_to_many_model	Actualizar una instancia del modelo "Student" con una lista de referencias al modelo "Courses".	Se crea una instancia en el modelo "AuditableAction" con action=UPDATED, model_json_old es igual a los datos inicialmente suministrados y model_json_new es igual a los datos actualizados.	Exitoso
C03-P08	test_many_to_many_view	Actualizar una instancia del modelo "Student" con una lista de referencias al modelo "Courses" a través de la "vista".	Se crea una instancia en el modelo "AuditableAction" con action=UPDATED, model_json_old es igual a los datos inicialmente suministrados y model_json_new es igual a los datos actualizados.	Exitoso

T01-P0 2	test_O2M_model	Crear una instancia del modelo "Company" que es referenciado por "Person".	Se crea una instancia en el modelo "AuditableAction" con action=UPDATED, model_json_old es igual a los datos inicialmente suministrados y model_json_new es igual a los datos actualizados.	Exitoso
-------------	----------------	--	---	---------

Nota: Las pruebas fallaron debido a que los *serializers* tomaban en cuenta los campos m2m, para solucionar esto fueron excluidos y las pruebas fueron exitosas.

ELIMINAR				
Código	Nombre	Descripción	Resultado Esperado	Resultado
C04-P0 1	test_simple_model	Eliminar una instancia, previamente creada, del modelo "Cars".	Se crea una instancia en el modelo "AuditableAction" con action=DELETED, model_json_new={} y model_json_old es igual al registro eliminado.	Exitoso
C04-P0 2	test_simple_view	Eliminar una instancia, previamente creada, del modelo "Cars" a través de la "vista".	Se crea una instancia en el modelo "AuditableAction" con action=DELETED, model_json_new={} y model_json_old es igual al registro eliminado.	Exitoso
C04-P0 3	test_one_to_many_model	Eliminar una instancia del modelo "Courses" con una referencia al modelo "Teacher".	Se crea una instancia en el modelo "AuditableAction" con action=DELETED, model_json_new={} y model_json_old es igual al registro eliminado.	Exitoso
C04-P0 4	test_one_to_many_view	Eliminar una instancia del modelo "Courses" con una referencia al modelo "Teacher" a través de la "vista".	Se crea una instancia en el modelo "AuditableAction" con action=DELETED, model_json_new={} y model_json_old es igual al registro eliminado.	Exitoso

C04-P05	test_one_to_many_casca de_model	Eliminar una instancia del modelo "Teacher" que elimine un "Course" relacionado a él (Cascada).	Se crea una instancia en el modelo "AuditableAction" con action=DELETED, model_json_new={} y model_json_old es igual al registro eliminado.	Exitoso
C04-P06	test_one_to_many_casca de_view	Eliminar una instancia del modelo "Teacher" que elimine un "Course" relacionado a él (Cascada) a través de la "vista".	Se crea una instancia en el modelo "AuditableAction" con action=DELETED, model_json_new={} y model_json_old es igual al registro eliminado.	Exitoso
C04-P07	test_many_to_many_mod el	Eliminar una instancia del modelo "Student" con una lista de referencias al modelo "Courses".	Se crea una instancia en el modelo "AuditableAction" con action=DELETED, model_json_new={} y model_json_old es igual al registro eliminado.	Exitoso
C04-P08	test_many_to_many_vie w	Eliminar una instancia del modelo "Student" con una lista de referencias al modelo "Courses" a través de la "vista".	Se crea una instancia en el modelo "AuditableAction" con action=DELETED, model_json_new={} y model_json_old es igual al registro eliminado.	Exitoso

Nota: La primera prueba falló porque estaban invertidos los campos model_json_new y model_json_old.

Módulo Estadísticas

Formulario de los filtros. Pruebas con sólo las fechas				
Código	Nombre	Descripción	Resultado Esperado	Resultado
E01-P01	test_all_results_empty_start	Llenar el formulario con la fecha y hora de finalización que coincidan con el tiempo del último registro.	Obtener todos los resultados.	Exitoso

E01-P02	test_all_results_empty_end	Llenar el formulario con la fecha y hora de inicio que coincidan con el tiempo del primer registro.	Obtener todos los resultados.	Exitoso
E01-P03	test_all_results	Llenar el formulario con la fecha y hora de inicio y fin que coincidan con el tiempo del primer y último registro respectivamente.	Obtener todos los resultados.	Exitoso
E01-P04	test_all_results_but_first	Llenar el formulario con la fecha y hora de inicio y fin que coincidan con el tiempo del segundo y último registro respectivamente.	Obtener todos los resultados excepto el primero.	Exitoso
E01-P05	test_all_results_but_last	Llenar el formulario con la fecha y hora de inicio y fin que coincidan con el tiempo del primer y penúltimo registro respectivamente.	Obtener todos los resultados excepto el último.	Exitoso
E01-P06	test_first_result	Llenar el formulario con la fecha y hora de inicio y fin que coincidan con el tiempo del primer registro.	Obtener únicamente el primer resultado	Exitoso
E01-P07	test_last_result	Llenar el formulario con la fecha y hora de inicio y fin que coincidan con el tiempo del último registro.	Obtener únicamente el último resultado	Exitoso
E01-P08	test_first_ten_result	Llenar el formulario con la fecha y hora de inicio y fin que coincidan con el tiempo del primer y del décimo registro respectivamente.	Obtener los diez primeros resultados.	Exitoso
E01-P07	test_last_ten_result	Llenar el formulario con la fecha y hora de inicio y fin que coincidan con el tiempo de los últimos diez registros.	Obtener los últimos diez resultados.	Exitoso
E01-P08	test_many_results	Llenar el formulario con la fecha y hora de inicio y fin que coincidan con el octavo y décimo segundo resultado.	Obtener todos los resultados excepto el último.	Exitoso

E01-P09	test_none_results_before_recording	Llenar el formulario con la fecha y hora de inicio y fin con unos valores que sean menores al tiempo del primer registro.	Ningún resultado.	Exitoso
E01-P10	test_none_results_before_recording	Llenar el formulario con la fecha y hora de inicio y fin con unos valores obtengan resultados.	Ningún resultado.	Exitoso

Formulario de los filtros. Pruebas con sólo el tipo de acción

Código	Nombre	Descripción	Resultado Esperado	Resultado
E03-P01	test_all_results	Llenar el formulario sin enviar una acción.	Obtener todos los resultados .	Exitoso
E03-P02	test_all_create_results	Llenar el formulario con la acción de Create	Obtener todos los resultados creados .	Exitoso
E03-P03	test_all_update_results	Llenar el formulario con la acción de Update	Obtener todos los resultados actualizados .	Exitoso
E03-P04	test_all_delete_results	Llenar el formulario con la acción de Delete	Obtener todos los resultados eliminados .	Exitoso

Formulario de los filtros. Pruebas con sólo el autor

Código	Nombre	Descripción	Resultado Esperado	Resultado
E05-P01	test_all_results	Llenar el formulario sin enviar una el autor.	Obtener todos los resultados .	Exitoso
E05-P02	test_all_anonymous_results	Llenar el formulario enviando -1, que representa a un usuario anónimo.	Obtener todos los resultados del usuario anónimo.	Exitoso
E05-P03	test_all_user_results	Llenar el formulario enviando el identificador de un usuario .	Obtener todos los resultados del usuario.	Exitoso

Formulario de los filtros. Pruebas con sólo el modelo

Código	Nombre	Descripción	Resultado Esperado	Resultado
E07-P01	test_all_results	Llenar el formulario sin enviar el modelo.	Obtener todos los resultados .	Exitoso
E07-P02	test_all_car_results	Llenar el formulario enviando el nombre de un modelo.	Obtener todos los resultados del modelo.	Exitoso
Formulario de los filtros. Formulario inválido.				
Código	Nombre	Descripción	Resultado Esperado	Resultado
E01-P11	test_start_gt_end	La fecha de inicio es mayor a la final.	Error: La fecha de inicio no puede ser mayor a la de finalización.	Exitoso
E01-P12	test_end_gt_today	La fecha de fin es mayor a la actual.	Error: La fecha de fin no puede ser mayor a la actual.	Exitoso
E01-P13	test_start_gt_today	La fecha de inicio es mayor a la actual.	Error: La fecha de inicio no puede ser mayor a la de finalización.	Exitoso
E01-P14	test_start_date_empty	Se envía el tiempo de inicio pero no la fecha.	Error: No se puede enviar la hora si la fecha es vacía.	Exitoso
E01-P15	test_end_date_empty	Se envía el tiempo de fin pero no la fecha.	Error: No se puede enviar la hora si la fecha es vacía.	Exitoso
E01-P16	test_wrong_model_name	Se envía un nombre de modelo que no se encuentra en la lista de modelos.	Error: Seleccione una opción válida.	Exitoso
Formulario de los filtros. Combinaciones de valores de los filtros.				
Código	Nombre	Descripción	Resultado Esperado	Resultado
E01-P19	test_all_results_empty_form	Enviar el formulario vacío.	Obtener todos los resultados .	Exitoso
E01-P20	test_all_results_empty_start	Enviar el formulario sin la fecha de inicio.	Obtener todos los resultados en donde la fecha es menor a la de fin.	Exitoso

E03-P05	test_all_results_empty_end	Enviar el formulario sin la fecha de fin.	Obtener todos los resultados .	Exitoso
E03-P06	test_create_results_empty_end	Enviar el formulario sin la fecha de fin y con la acción "C".	Obtener todos los elementos creados.	Exitoso
E03-P07	test_update_result_empty_end	Enviar el formulario sin la fecha de fin y con la acción "U".	Obtener todos los elementos actualizados.	Exitoso
E03-P08	test_delete_result_empty_end	Enviar el formulario sin la fecha de fin y con la acción "D".	Obtener todos los elementos eliminados.	Exitoso
E03-P09	test_create_results_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "C" excepto el primero.	Obtener todos los elementos creados excepto el primero.	Exitoso
E03-P10	test_update_results_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "U" excepto el primero.	Obtener todos los elementos actualizados excepto el primero.	Exitoso
E03-P11	test_delete_results_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "D" excepto el primero.	Obtener todos los elementos eliminados excepto el primero.	Exitoso
E03-P12	test_create_results_but_last	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "C" excepto el último.	Obtener todos los elementos creados excepto el último.	Exitoso
E03-P13	test_update_results_but_last	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "U" excepto el último.	Obtener todos los elementos actualizados excepto el último.	Exitoso
E03-P14	test_delete_results_but_last	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "D" excepto el último.	Obtener todos los elementos eliminados excepto el último.	Exitoso

E03-P15	test_first_create_result	Enviar el formulario con todos los campos en donde se obtenga el primer resultado creado.	Obtener el primer resultado creado.	Exitoso
E03-P16	test_first_update_result	Enviar el formulario con todos los campos en donde se obtenga el primer resultado actualizado.	Obtener el primer resultado actualizado.	Exitoso
E03-P17	test_first_delete_result	Enviar el formulario con todos los campos en donde se obtenga el primer resultado eliminado.	Obtener el primer resultado eliminado.	Exitoso
E03-P18	test_last_create_result	Enviar el formulario con todos los campos en donde se obtenga el último resultado creado.	Obtener el primer resultado creado.	Exitoso
E03-P19	test_last_update_result	Enviar el formulario con todos los campos en donde se obtenga el último resultado actualizado.	Obtener el primer resultado actualizado.	Exitoso
E03-P20	test_last_delete_result	Enviar el formulario con todos los campos en donde se obtenga el último resultado eliminado.	Obtener el primer resultado eliminado.	Exitoso
E03-P21	test_many_create_result	Enviar el formulario con todos los campos en donde se obtenga los resultados desde el décimo segundo al décimo sexto.	Obtener los elementos creados, desde el 12 hasta el 16.	Exitoso
E03-P22	test_many_update_result	Enviar el formulario con todos los campos en donde se obtenga los resultados desde el tercero al séptimo.	Obtener los elementos actualizados, desde el 3 hasta 7.	Exitoso
E03-P23	test_many_delete_result	Enviar el formulario con todos los campos en donde se obtenga los resultados desde el segundo al tercero.	Obtener los elementos creados, desde el 2 hasta el 3.	Exitoso
E05-P01	test_results_created_userA_empty_end	Enviar el formulario sin la fecha de fin, con usuario anónimo y con la acción "C".	Obtener todos los elementos creados por un usuario anónimo.	Exitoso
E05-P02	test_results_created_user_empty_end	Enviar el formulario sin la fecha de fin, con algún usuario y con la acción "C".	Obtener todos los elementos creados por un usuario específico.	Exitoso
E05-P03	test_results_updated_user_empty_end	Enviar el formulario sin la fecha de fin, con algún usuario y con la acción "U".	Obtener todos los elementos creados	Exitoso

			actualizados por un usuario específico.	
E05-P04	test_results_updated_userA_empty_end	Enviar el formulario sin la fecha de fin, con usuario anónimo y con la acción "U".	Obtener todos los elementos actualizados por un usuario anónimo.	Exitoso
E05-P05	test_results_deleted_user_empty_end	Enviar el formulario sin la fecha de fin, con algún usuario y con la acción "D".	Obtener todos los elementos creados eliminados por un usuario específico.	Exitoso
E05-P06	test_results_deleted_userA_empty_end	Enviar el formulario sin la fecha de fin, con usuario anónimo y con la acción "D".	Obtener todos los elementos eliminados por un usuario anónimo.	Exitoso
E05-P07	test_create_user_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "C" y un usuario específico excepto el primero.	Obtener todos los elementos creados por un usuario específico, excepto el primero.	Exitoso
E05-P08	test_create_userA_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "C" y el usuario anónimo excepto el primero.	Obtener todos los elementos creados por el usuario anónimo, excepto el primero.	Exitoso
E05-P09	test_update_user_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "U" y un usuario específico excepto el primero.	Obtener todos los elementos actualizados por un usuario específico, excepto el primero.	Exitoso
E05-P10	test_update_userA_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "U" y el usuario anónimo excepto el primero.	Obtener todos los elementos actualizados por el usuario anónimo, excepto el primero.	Exitoso
E05-P11	test_delete_user_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "D" y un usuario específico excepto el primero.	Obtener todos los elementos eliminados por un usuario específico, excepto el primero.	Exitoso

E05-P12	test_delete_userA_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con la acción "D" y el usuario anónimo excepto el primero.	Obtener todos los elementos eliminados por el usuario anónimo, excepto el primero.	Exitoso
E07-P01	test_all_cars_model	Enviar el formulario con todos los campos en donde se obtengan todos los resultados de un modelo en particular.	Obtener todos los elementos de un modelo en particular.	Exitoso
E07-P02	test_all_cars_model_user	Enviar el formulario con todos los campos en donde se obtengan todos los resultados de un modelo y usuario en particular.	Obtener todos los elementos de un modelo y usuario en particular.	Exitoso
E07-P03	test_all_cars_model_userA	Enviar el formulario con todos los campos en donde se obtengan todos los resultados de un modelo en particular y usuario anónimo.	Obtener todos los elementos de un modelo en particular con el usuario anónimo.	Exitoso
E07-P04	test_create_cars_model_user	Enviar el formulario con todos los campos en donde se obtengan todos los resultados creados de un modelo y usuario en particular.	Obtener todos los elementos de un modelo en particular creados por un usuario.	Exitoso
E07-P05	test_create_cars_model_userA	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con acción "C" de un modelo en particular y usuario anónimo.	Obtener todos los elementos de un modelo en particular creados por un usuario anónimo.	Exitoso
E07-P06	test_update_cars_model_user	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con acción "U" de un modelo y usuario en particular.	Obtener todos los elementos de un modelo en particular actualizados por un usuario.	Exitoso
E07-P07	test_update_cars_model_userA	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con acción "U" de un modelo en particular y usuario anónimo.	Obtener todos los elementos de un modelo en particular actualizados por un usuario anónimo.	Exitoso

E07-P08	test_delete_cars_model_user	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con acción "D" de un modelo en particular y usuario anónimo.	Obtener todos los elementos de un modelo en particular eliminados por un usuario.	Exitoso
E07-P09	test_delete_cars_model_userA	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con acción "D" de un modelo y usuario en particular.	Obtener todos los elementos de un modelo en particular eliminados por un usuario anónimo.	Exitoso
E07-P10	test_create_cars_model_user_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con acción "C" de un modelo en particular y usuario anónimo excepto el primero.	Obtener todos los elementos de un modelo en particular creados por un usuario, excepto el primero.	Exitoso
E07-P11	test_update_cars_model_user_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con acción "U" de un modelo y usuario en particular.	Obtener todos los elementos de un modelo en particular actualizados por un usuario, excepto el primero.	Exitoso
E07-P12	test_delete_cars_model_user_but_first	Enviar el formulario con todos los campos en donde se obtengan todos los resultados con acción "D" de un modelo y usuario en particular.	Obtener todos los elementos de un modelo en particular eliminados por un usuario anónimo, excepto el primero.	Exitoso
E07-P13	test_none_results	Enviar el formulario con todos los campos en donde no se obtengan resultados.	Ningún resultado.	Exitoso

Estadísticas. Combinaciones de valores de los filtros.

Código	Nombre	Descripción	Resultado Esperado	Resultado
E01-P21	test_all_results_empty_form	Enviar el request vacío.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo,	Exitoso

			máximo, cantidad de modelos)	
E01-P22	test_all_results_empty_start	Enviar el request sin la fecha de inicio.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P24	test_all_results_empty_end	Enviar el request sin la fecha de fin.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P25	test_create_results_empty_end	Enviar el request sin la fecha de fin y con la acción "C".	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P26	test_update_result_empty_end	Enviar el request sin la fecha de fin y con la acción "U".	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P27	test_delete_result_empty_end	Enviar el request sin la fecha de fin y con la acción "D".	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P28	test_create_results_but_first	Enviar el request con todos los campos en donde se obtengan todos los resultados con la acción "C" excepto el primero.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso

E03-P29	test_update_results_but_first	Enviar el request con todos los campos en donde se obtengan todos los resultados con la acción "U" excepto el primero.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P30	test_delete_results_but_first	Enviar el request con todos los campos en donde se obtengan todos los resultados con la acción "D" excepto el primero.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P31	test_create_results_but_last	Enviar el request con todos los campos en donde se obtengan todos los resultados con la acción "C" excepto el último.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P32	test_update_results_but_last	Enviar el request con todos los campos en donde se obtengan todos los resultados con la acción "U" excepto el último.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P33	test_delete_results_but_last	Enviar el request con todos los campos en donde se obtengan todos los resultados con la acción "D" excepto el último.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P34	test_first_create_result	Enviar el request con todos los campos en donde se obtenga el primer resultado creado.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P35	test_first_update_result	Enviar el request con todos los campos en donde se obtenga el primer resultado actualizado.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo,	Exitoso

			máximo, cantidad de modelos)	
E03-P36	test_first_delete_result	Enviar el request con todos los campos en donde se obtenga el primer resultado eliminado.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P37	test_last_create_result	Enviar el request con todos los campos en donde se obtenga el último resultado creado.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P38	test_last_update_result	Enviar el request con todos los campos en donde se obtenga el último resultado actualizado.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P39	test_last_delete_result	Enviar el request con todos los campos en donde se obtenga el último resultado eliminado.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P40	test_many_create_result	Enviar el request con todos los campos en donde se obtenga los resultados desde el décimo segundo al décimo sexto.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E03-P41	test_many_update_result	Enviar el request con todos los campos en donde se obtenga los resultados desde el tercero al séptimo.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso

E05-P13	test_results_created_userA_empty_end	Enviar el request sin la fecha de fin, con usuario anónimo y con la acción "C".	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E05-P14	test_results_created_user_empty_end	Enviar el request sin la fecha de fin, con algún usuario y con la acción "C".	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E05-P15	test_results_updated_user_empty_end	Enviar el request sin la fecha de fin, con algún usuario y con la acción "U".	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E05-P16	test_results_updated_userA_empty_end	Enviar el request sin la fecha de fin, con usuario anónimo y con la acción "U".	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E05-P17	test_results_deleted_user_empty_end	Enviar el request sin la fecha de fin, con algún usuario y con la acción "D".	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E05-P18	test_results_deleted_userA_empty_end	Enviar el request sin la fecha de fin, con usuario anónimo y con la acción "D".	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E07-P08	test_all_cars_model_user	Enviar el request con todos los campos en donde se obtengan todos los resultados de un modelo y usuario en particular.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo,	Exitoso

			máximo, cantidad de modelos)	
E07-P09	test_all_cars_model_userA	Enviar el request con todos los campos en donde se obtengan todos los resultados de un modelo en particular y usuario anónimo.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E07-P10	test_create_cars_model_user	Enviar el request con todos los campos en donde se obtengan todos los resultados creados de un modelo y usuario en particular.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E07-P13	test_update_cars_model_userA	Enviar el request con todos los campos en donde se obtengan todos los resultados con acción "U" de un modelo en particular y usuario anónimo.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E07-P14	test_delete_cars_model_user	Enviar el request con todos los campos en donde se obtengan todos los resultados con acción "D" de un modelo en particular y usuario anónimo.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E07-P15	test_delete_cars_model_userA	Enviar el request con todos los campos en donde se obtengan todos los resultados con acción "D" de un modelo y usuario en particular.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E07-P16	test_create_cars_model_user_but_first	Enviar el request con todos los campos en donde se obtengan todos los resultados con acción "C" de un modelo en particular y usuario anónimo excepto el primero.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso

E07-P17	test_update_cars_model_user_but_first	Enviar el request con todos los campos en donde se obtengan todos los resultados con acción "U" de un modelo y usuario en particular.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E07-P18	test_delete_cars_model_user_but_first	Enviar el request con todos los campos en donde se obtengan todos los resultados con acción "D" de un modelo y usuario en particular.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso
E07-P19	test_none_results	Enviar el request con todos los campos en donde no se obtengan resultados.	Obtener los valores correctos de las estadísticas (cantidad, promedio, mínimo, máximo, cantidad de modelos)	Exitoso

Estadísticas. Cantidad de modelos auditables.

Código	Nombre	Descripción	Resultado Esperado	Resultado
E01-P21	test_new_model	Crear un nuevo modelo que no sea auditable.	El contador de modelos auditables se mantenga igual y el porcentaje de cobertura disminuya.	Exitoso
E01-P22	test_new_audit_model	Crear un nuevo modelo que sea auditable.	El contador de modelos auditables aumente y porcentaje de cobertura aumente.	Exitoso

APÉNDICE F

Resultado de las pruebas de desempeño

RESULTADOS DE PRUEBAS DE DESEMPEÑO

Justificación

Dado que las consultas acerca de los listados (búsqueda, filtrado y ordenamiento) en el modelo de auditoría, se realizarán frecuentemente sobre los campos “author”, “action” y “model_name”, se evaluar qué ventajas o desventajas ofrece agregar índices a estos campos. No obstante, hay que tener en cuenta que la operación más frecuente a efectuar es la inserción sobre la tabla y los índices tienden a ralentizarla puesto que es necesario preservar el orden.

Diseño de la prueba

Para cargar la tabla de acciones auditables, se elaboró un script parametrizado, en Python, que genera un archivo tipo JSON que puede ser utilizado como un *Fixture*. El *script*, toma como entrada un archivo de configuración, llamado “*fixture_settings.py*”, en el que se debe indicar el nombre del modelo a agregar (“name”), cuantos elementos (“size”) y qué campos posee (“fields”). El *script* también crea usuarios en la base de datos para que puedan ser referenciados por el campos “author”.

En la figura 1 se muestra un ejemplo del archivo utilizado para cargar la base de datos con un millón de elementos en la tabla de acciones auditables. Se pueden incluir otras tablas si así se desea.

```
MODELS = [  
    {  
        'size': 1000000,  
        'name': 'auditor_core.AuditableAction',  
        'fields': [  
            'author',  
            'action',  
            'datetime',  
            'model_name',  
            'model_json_old',  
            'model_json_new'  
        ]  
    },  
]
```

Figura 1. Ejemplo de “*fixture_settings.py*”

La prueba se realizó cuatro veces, con mil, diez mil, cien mil y un millón de registros en la tabla de acciones auditables. Luego, se insertaron cien registros en dos tablas, una auditable y otra no auditable, y se comparan los tiempos obtenidos cuando la tabla posee índices y cuando no.

Este proceso se realizó con diferentes manejadores de base de datos relacionales, SQLite, MySQL y PostgreSQL para obtener el desempeño en cada uno de ellos.

Análisis de resultados

Cada uno de los resultados presentados fueron generados siguiendo lo explicado anteriormente sobre un computador con las siguientes especificaciones:

- Memoria RAM de 8GB.
- Procesador Intel® Core™ i5-4300M CPU @ 2.60GHz × 4.
- Tarjeta de vídeo Intel® Haswell Mobile.
- Sistema Operativo de 64 bits.
- Espacio en disco de 512GB.

SQLite

A continuación se presentan los resultados de las pruebas para el manejador de base de datos SQLite.

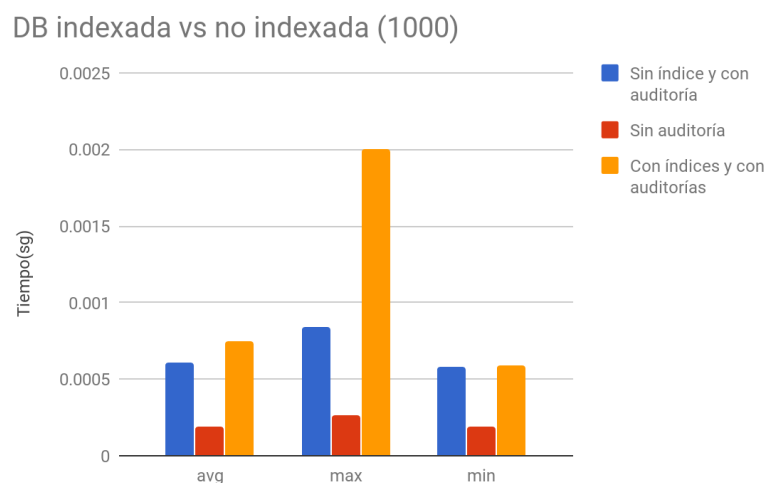


Figura 2. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar mil registros.

El color azul representa el tiempo inserción de un registro en una tabla que es auditable, “AuditableDummyModel” sin agregar los índices. El color rojo representa el tiempo de inserción en una tabla que no es auditable, “DummyModel” y sin índices. El color amarillo representa el tiempo inserción de un registro en “AuditableDummyModel” agregando índices a la base de datos.

Con los resultados mostrados en la figura 2, se observa un incremento poco significativo entre el agregar o no índices a las tablas. En promedio, este valor es del 22%. Al comparar ambos valores, con

el tiempo de inserción en la base de datos de un registro sin auditar, se presenta un crecimiento de 212% y 283% con respecto a sus promedios.

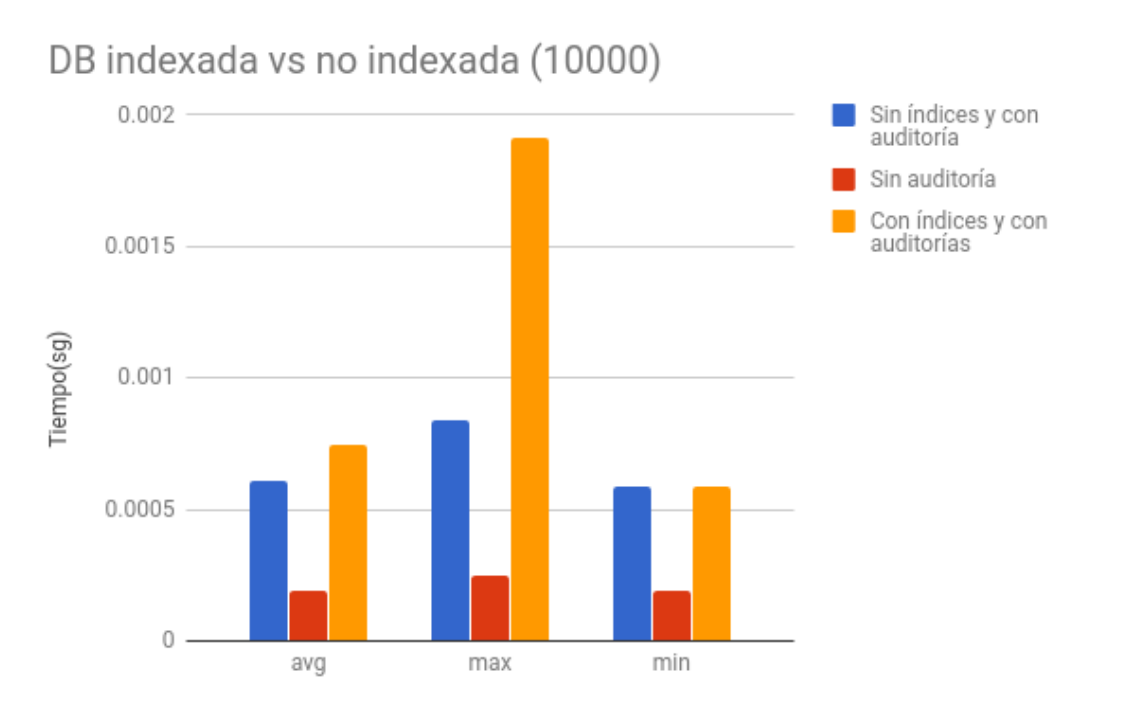


Figura 3. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar diez mil registros.

En este caso, el la diferencia entre las tabla indexada y la que no lo estaba fue de 23% y comparados con el tiempo de inserción sin trazas de auditoría se incrementó en 213% 286% respectivamente. La tabla indexada tiene un tiempo promedio de inserción (tomando en cuenta el del registro auditable) de 0.000748589 segundos.

DB indexada vs no indexada (100000)

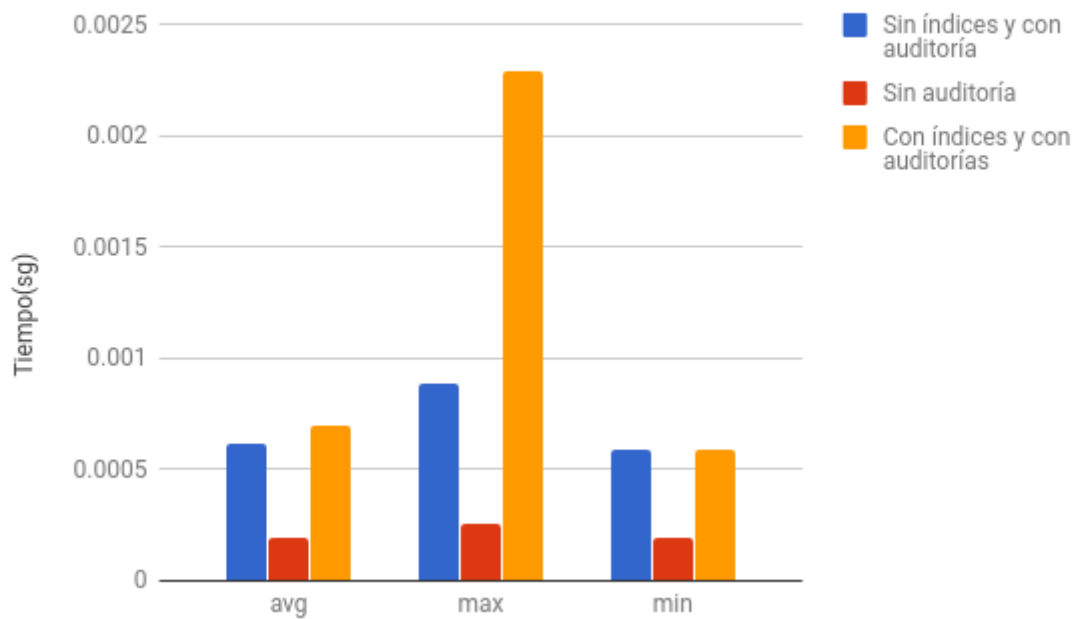


Figura 4. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar cien mil registros.

El comportamiento presentado tanto para la inserción de mil registros, como para la de cien mil, se mantiene al aumentar la carga de la base de datos SQLite a cien mil registros.

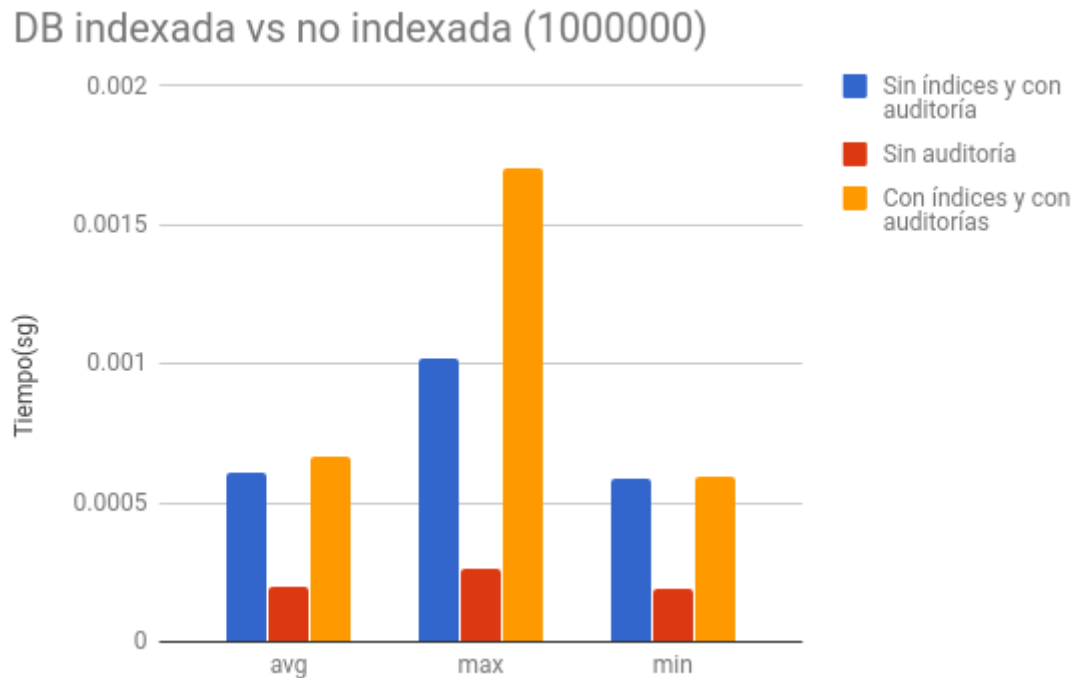


Figura 5. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar un millón de registros.

Las inserciones en la tabla auditable con índices conllevan un mayor tiempo de ejecución que el resto, como era esperado. Considerando todos los resultados anteriores, se puede concluir que para el manejador de base de datos SQLite el tiempo de inserción de un registro auditable cuando la tabla posee índices es de 0.0007, en promedio. Este valor, representa un aumento sobre el valor sin índices de aproximadamente 17% y sobre la inserción sin auditorías de 250%.

Sin embargo, esta diferencia es prácticamente indetectable por un humano por lo que, agregar índices, es una solución viable para mejorar el tiempo de consulta en el futuro con la base de datos SQLite. Es importante resaltar que los tiempos no desmejoraron al aumentar la carga del manejador.

Tabla 1. Resumen de los promedios de crecimiento usando SQLite

Resumen		
%Crecimiento sin indices vs sin auditorías	%Crecimiento con indices vs sin auditorías	% Crecimiento sin índices vs con índices
213.2649828	267.2000846	17.21908549

PostgreSQL

Luego de efectuar las pruebas sobre SQLite, se modificó la configuración en Django, para utilizar PostgreSQL, en donde se obtuvo los resultados mostrados a continuación:

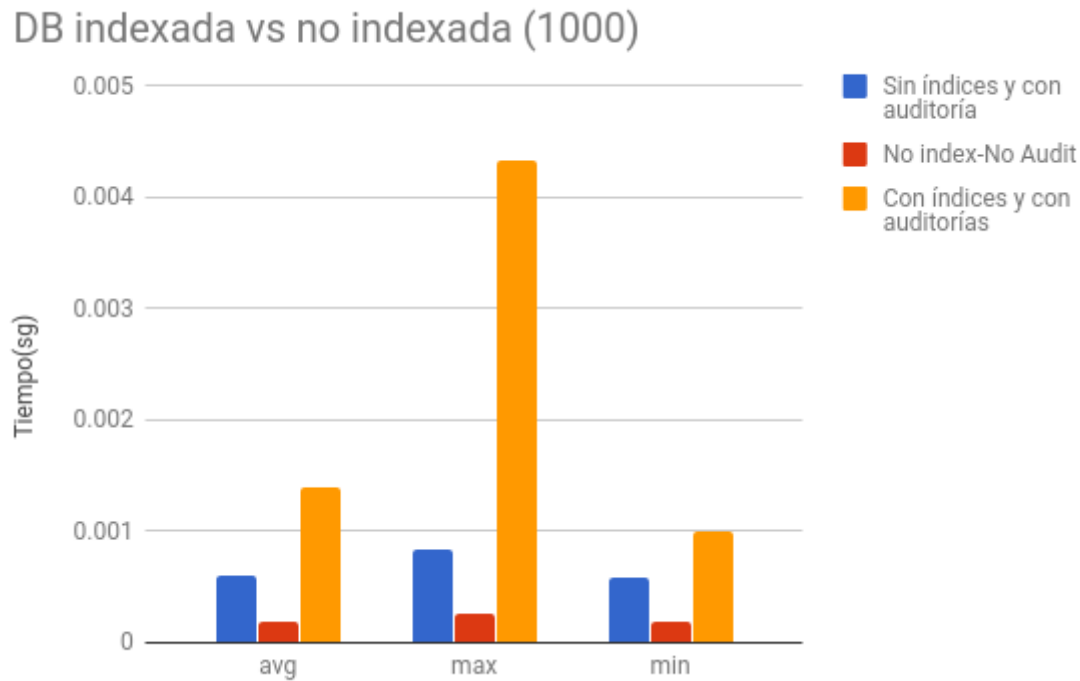


Figura 6. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar mil registros.

En este caso también se observa que el tiempo de inserción en la tabla sin índices es menor, en un 129%. El tiempo promedio de creación de un registro sin auditorías es de 0.0001940346 segundos y con respecto a este valor, agregar índices a la tabla aumenta el tiempo en 616%.

DB indexada vs no indexada (10000)

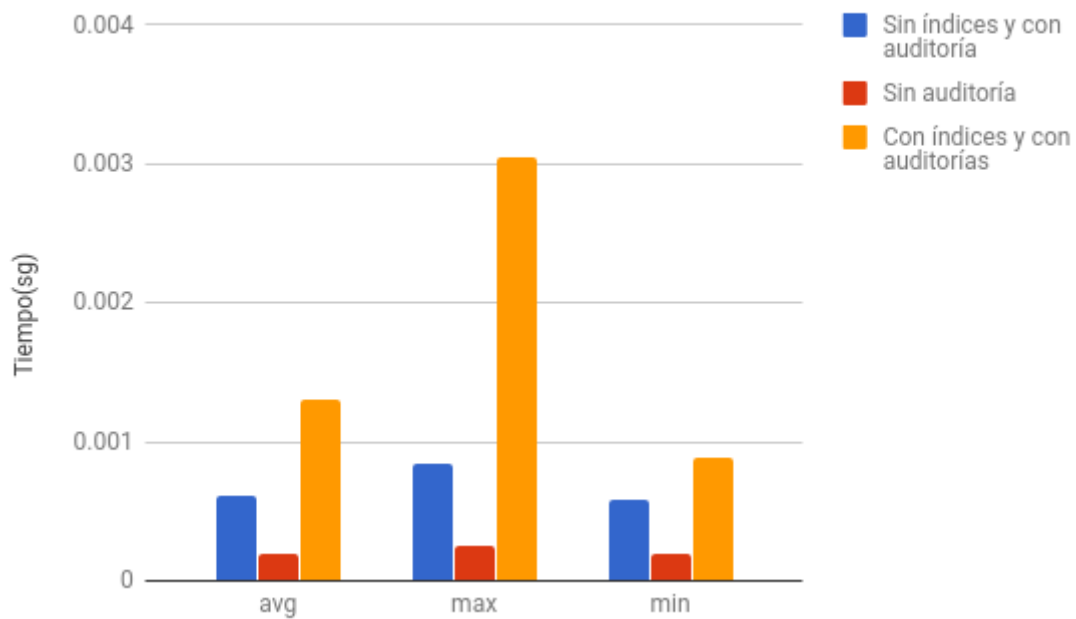


Figura 7. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar diez mil registros.

Al insertar un registro auditable se obtiene un aumento de 213% y 569%, con y sin índices respectivamente. Entre estos valores, se tiene una diferencia de 113%, siendo el segundo mayor que el primero.

DB indexada vs no indexada (100000)

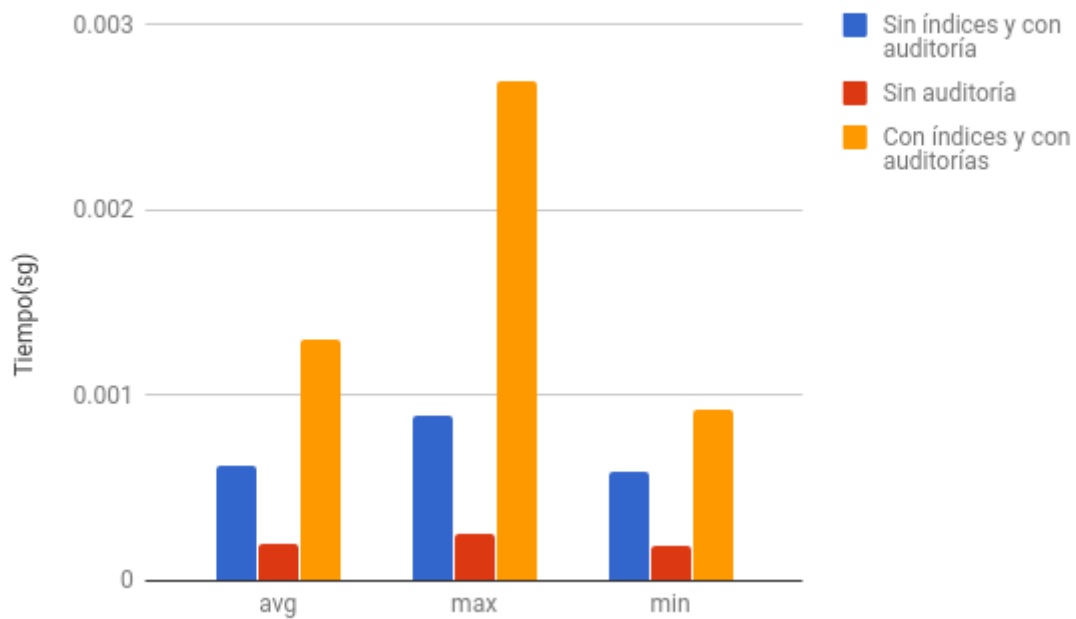


Figura 8. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar cien mil registros.

DB indexada vs no indexada (1000000)

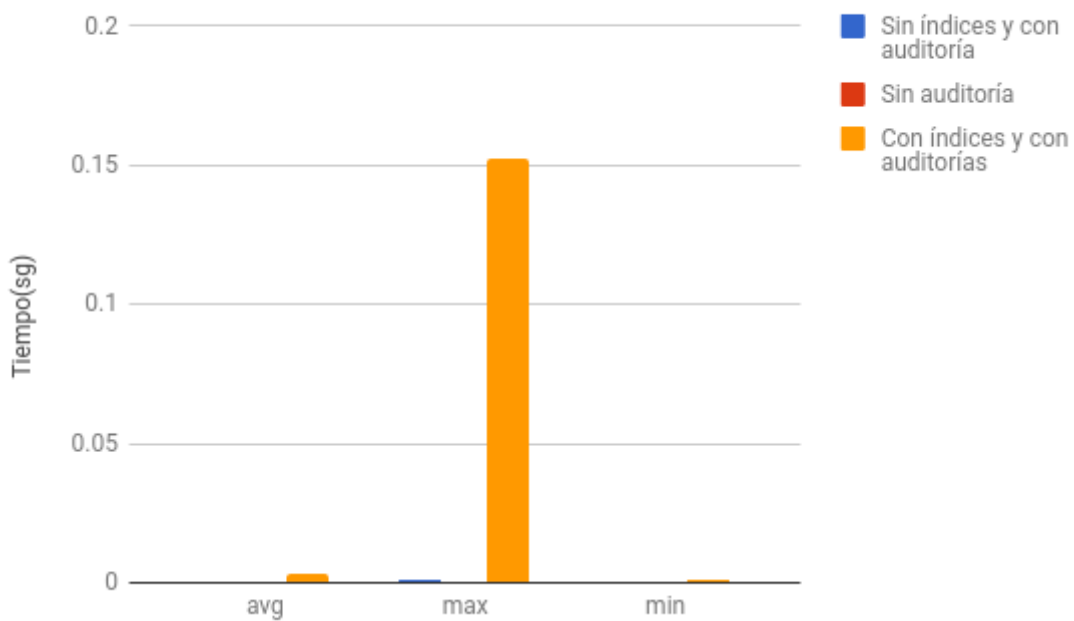


Figura 9. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar un millón de registros.

En las figuras 8 y 9 se observa que la creación del registro con índices y con auditorías, es mayor al resto. En general, se puede concluir que para PostgreSQL se presenta un incremento promedio de 182% al comparar los valores de ambas inserciones con auditorías.

A diferencia de SQLite, PostgreSQL, muestra una pequeña desmejora en el desempeño al cargar un millón de registros.

Tabla2. Resumen de los promedios de crecimiento usando PostgreSQL

Resumen		
%Crecimiento sin indices vs sin auditorías	%Crecimiento con indices vs sin auditorías	% Crecimiento sin índices vs con índices
213.2649828	786.5247172	182.9698376

MySQL

Por último, se efectuaron las pruebas sobre MySQL, son lo que se obtuvo las gráficas mostradas a continuación.

DB indexada vs no indexada (1000)

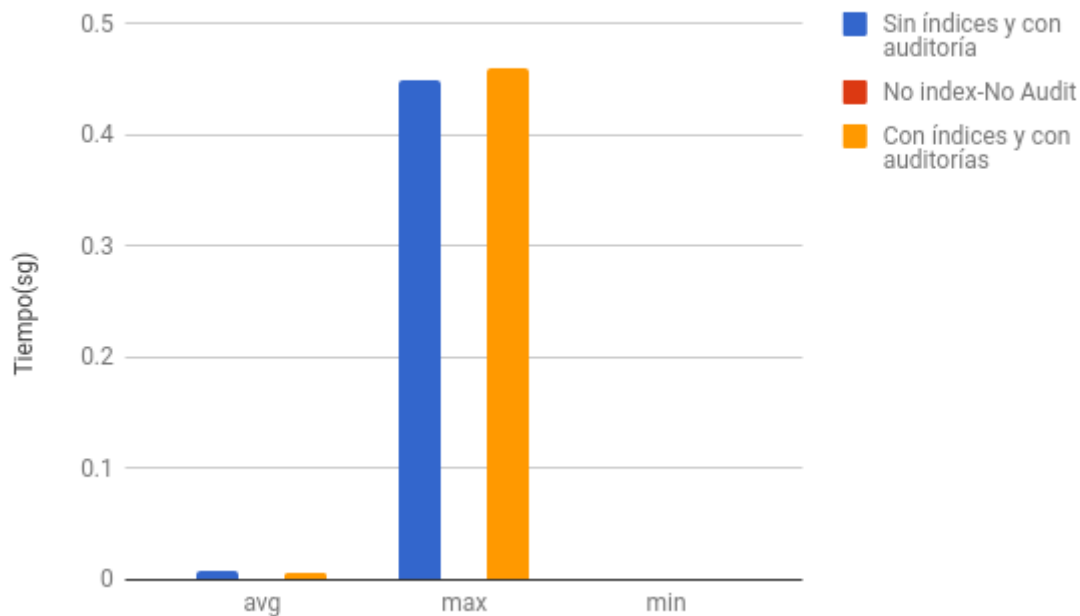


Figura 10. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar mil registros.

Este manejador de base de datos es significativamente más lento que el resto, sin embargo el comportamiento descrito se mantiene. El uso de auditorías aumenta el tiempo de inserción en 1532% y 1751% sin índices y con índices, respectivamente.

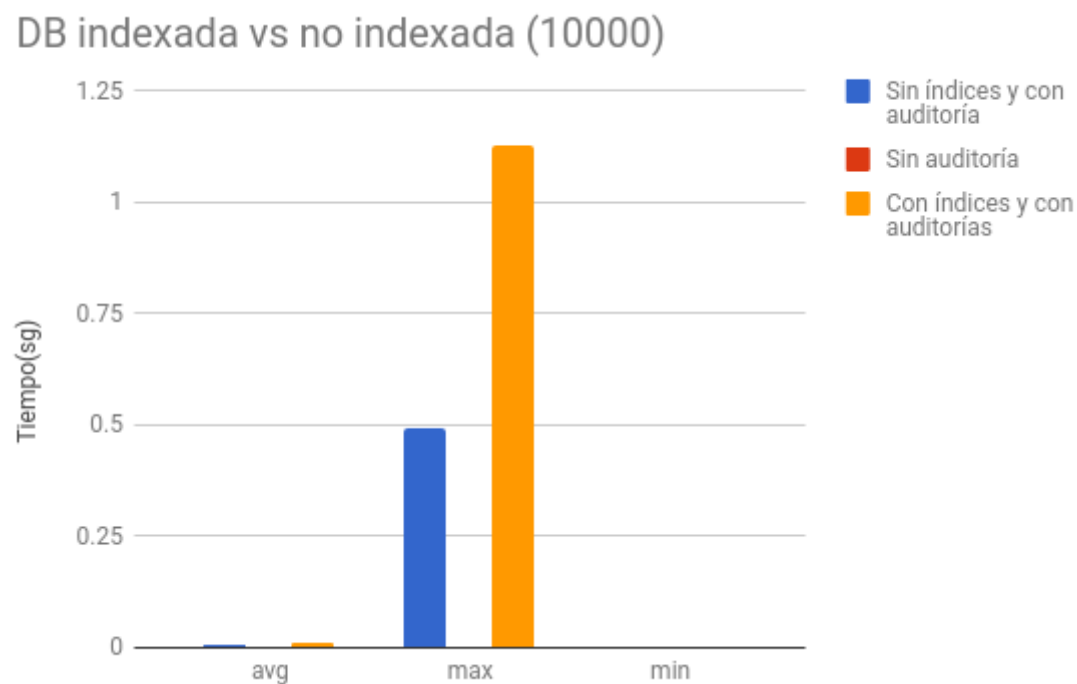


Figura 11. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar diez mil registros.

El máximo con auditorías y sin índices es 0.4914271832 segundos y el mínimo es 0.0006959438 segundos. Dado que estos valores son lejanos entre sí, se muestra que MySQL no tiene un desempeño consistente durante la pruebas cuando se incluye la traza de auditoría. Ocurre de igual manera al agregar los índices.

DB indexada vs no indexada (100000)

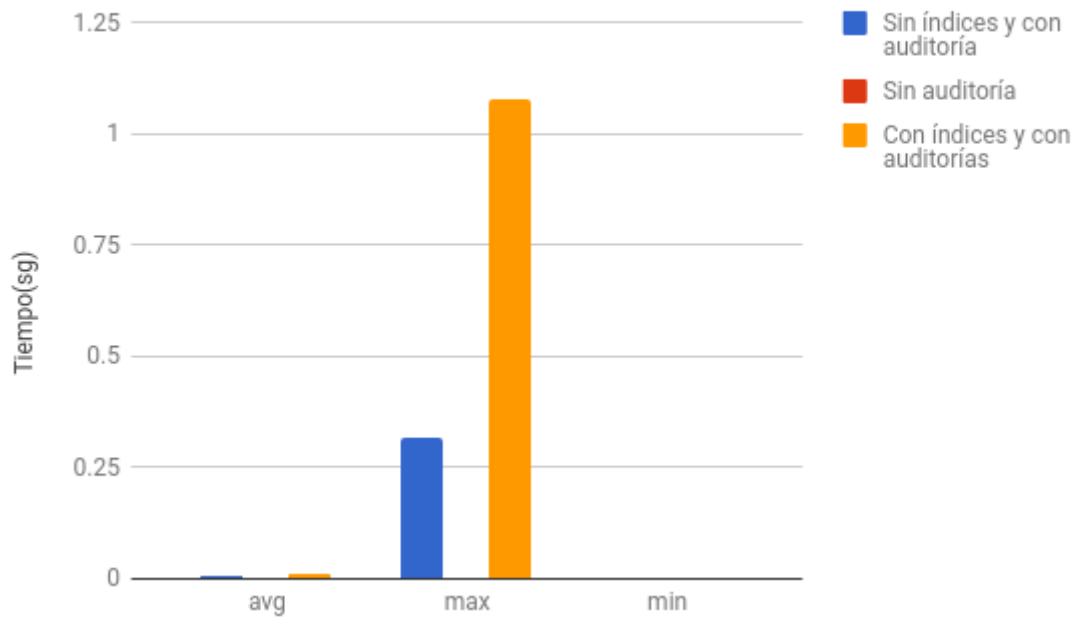


Figura 12. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar cien mil registros.

El porcentaje de crecimiento de la tabla indexada sobre la no indexada, en este caso, es del 90%. El primero, incrementa el tiempo de inserción en 3161% y el segundo en 1612% sobre el registro sin auditorías.

DB indexada vs no indexada (1000000)

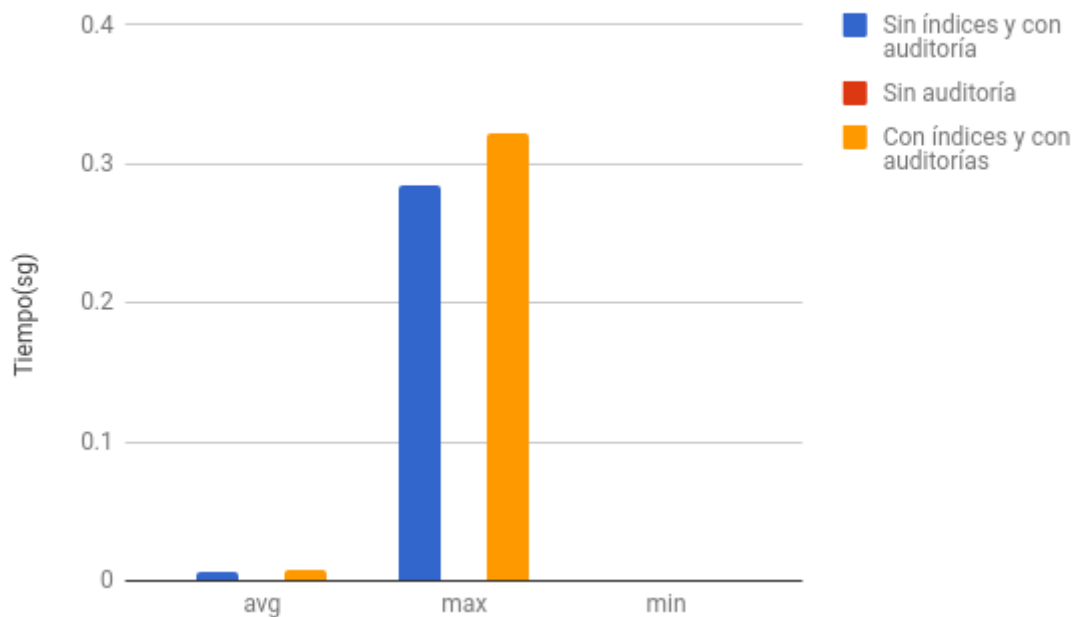


Figura 13. Resultados del promedio, mínimo y máximo para la tabla indexada comparada con la no-indexada al cargar un millón de registros.

Con este análisis se puede concluir que entre los manejadores de base de datos estudiados, MySQL muestra el peor desempeño al agregar trazas de auditorías. Considerando la robustez y confiabilidad de PostgreSQL, este es probablemente el mejor para llevar las auditorías con la librería Auditorías Turpial. Aunque SQLite presenta el menor tiempo de manera general, los índices prácticamente no afecta el tiempo.

Para caracterizar de manera más fiable el uso de índices, es necesario, en el futuro, realizar un estudio sobre las consultas.

Tabla 3. Resumen de los promedios de crecimiento usando MySQL.

Resumen		
%Crecimiento sin índices vs sin auditorías	%Crecimiento con índices vs sin auditorías	% Crecimiento sin índices vs con índices
1894.912972	2968.319394	53.66512553

Resultados obtenidos

SQLite

N	Sin índices y con auditoría	Sin auditorías	Con índices y con auditorías
Carga de la base de datos: 1000 registros			
sum	0.060614109	0.0194034576	0.0743262768
avg	0.0006061411	0.0001940346	0.0007432628
max	0.0008399487	0.0002629757	0.0020010471
min	0.0005831718	0.0001890659	0.000590086
Carga de la base de datos: 10000 registros			
sum	0.0608172417	0.0193741322	0.0748589039
avg	0.0006081724	0.0001937413	0.000748589
max	0.000838995	0.0002458096	0.0019161701
min	0.0005860329	0.0001890659	0.0005879402
Carga de la base de datos: 100000 registros			
sum	0.061671257	0.0196828842	0.0701274872
avg	0.0006167126	0.0001968288	0.0007012749
max	0.0008909702	0.0002541542	0.0022900105
min	0.0005850792	0.0001897812	0.0005891323
Carga de la base de datos: 1 millón de registros			
sum	0.0611720085	0.019516468	0.066955328
avg	0.0006117201	0.0001951647	0.0006695533
max	0.0010168552	0.0002639294	0.0017039776
min	0.0005850792	0.0001888275	0.0005939007

PostgreSQL

N	Sin índices y con auditoría	Sin auditorías	Con índices y con auditorías
Carga de la base de datos: 1000 registros			
sum	0.060614109	0.0194034576	0.1389353275
avg	0.0006061411	0.0001940346	0.0013893533
max	0.0008399487	0.0002629757	0.0043299198
min	0.0005831718	0.0001890659	0.0009920597

Carga de la base de datos: 10000 registros			
sum	0.0608172417	0.0193741322	0.1297955513
avg	0.0006081724	0.0001937413	0.0012979555
max	0.000838995	0.0002458096	0.00304389
min	0.0005860329	0.0001890659	0.0008819103
Carga de la base de datos: 100000 registros			
sum	0.061671257	0.0196828842	0.130531311
avg	0.0006167126	0.0001968288	0.0013053131
max	0.0008909702	0.0002541542	0.0027000904
min	0.0005850792	0.0001897812	0.0009260178
Carga de la base de datos: 1 millón de registros			
sum	0.0611720085	0.019516468	0.2921519279
avg	0.0006117201	0.0001951647	0.0029215193
max	0.0010168552	0.0002639294	0.1524419785
min	0.0005850792	0.0001888275	0.0009269714

MySQL

N	Sin índices y con auditoría	Sin auditorías	Con índices y con auditorías
Carga de la base de datos: 1000 registros			
sum	0.7648100853	0.0413048267	0.674380064
avg	0.0076481009	0.0004130483	0.0067438006
max	0.4487261772	0.0010368824	0.4603960514
min	0.0007300377	0.0002479553	0.0007231236
Carga de la base de datos: 10000 registros			
sum	0.5675678253	0.0261428356	1.258504391
avg	0.0056756783	0.0002614284	0.0125850439
max	0.4914271832	0.0007078648	1.126851082
min	0.0006959438	0.0002331734	0.0007381439
Carga de la base de datos: 100000 registros			
sum	0.6389715672	0.0373146534	1.217111826
avg	0.0063897157	0.0003731465	0.0121711183
max	0.3159770966	0.0013489723	1.078650951
min	0.000688076	0.0002319813	0.0007259846

Carga de la base de datos: 1 millón de registros			
sum	0.6576070786	0.0292971134	0.7514355183
avg	0.0065760708	0.0002929711	0.0075143552
max	0.2842690945	0.0007591248	0.3214919567
min	0.0007238388	0.000246048	0.0007269382

APÉNDICE G

Manual de Auditorías Turpial *Core*

MANUAL DE INSTALACIÓN Y CONFIGURACIÓN DE AUDITOR TURPIAL

Esta librería denominada `turpial_auditor` es una extensión del *framework* Django que proporciona y monitorea las trazas de las instancias de los modelos auditables.

REQUISITOS

- Python 2.7
- Django 1.10
- jsonfield 2.2

INSTALACIÓN

Agregue "turpial_auditor.auditor_core" a su configuración `INSTALLED_APPS` de la siguiente manera:

```
INSTALLED_APPS = (  
    ...  
    'turpial_auditor.auditor_core',  
)
```

Incluya el URLconf del auditor en su proyecto `urls.py` de la siguiente manera:

```
url(r'^auditor/', include('turpial_auditor.urls')),
```

Ejecutar “python manage.py makemigrations” para crear el archivo de migración para el modelo de auditor.

Ejecutar “python manage.py migrate” para aplicar la migración y crear el modelo de Auditor.

USO Y EJEMPLOS

EL MODELO

Esta biblioteca proporciona un modelo que registra la acción (`CREATED`, `UPDATED`, `DELETED`), el valor anterior de la instancia, el nuevo valor, la fecha, la hora y el autor (la persona que ejecuta la acción), que se llama **AuditableAction**.

Nota: Los valores, antiguos y nuevos, de la instancia se representan como un JSON que excluye la representación de muchos campos.

HACER AUDITABLE UN MODELO EN DJANGO

Importar **AuditableMixin** de `turpial_auditor`. Una herencia en tu modelo, asegúrate de que sea el primero, el orden importa:

```
from turpial_auditor.auditor_core.models import AuditableMixin

class CarsModel(AuditableMixin, ..., models.Model, ...):
    ...
```

Si desea auditar un modelo muchos a muchos (*many to many*), deberá definir el suyo utilizando la propiedad **through**:

```
from turpial_auditor.auditor_core.models import AuditableMixin

class Person(models.Model):
    ...

class Group(models.Model):
    ...
    members = models.ManyToManyField(Person, through='Membership')
    ...

class Membership(AuditableMixin, models.Model):
    person = models.ForeignKey(Person, on_delete=models.CASCADE)
    group = models.ForeignKey(Group, on_delete=models.CASCADE)
    ...
```


Ahora su modelo puede ser auditado, sin embargo, no guarda el usuario registrado en el Modelo **AuditableAction**. Puedes lograr esto agregando a tus vistas otra *mixin* llamado **AuditableMixinView**:

```
from turpial_auditor.auditor_core.views import AuditableMixinView

class CarsView(AuditableMixinView, ...):
    ...
```

APÉNDICE H

Manual de Auditorías Turpial UI (extracto)

MANUAL DE INSTALACIÓN Y CONFIGURACIÓN DE TURPIAL AUDITOR UI

Esta librería proporciona una forma de visualizar los datos generados por la librería **turpial_auditor**. Tiene un módulo "Reportes" donde se puede listar las auditorías y generar informes de PDF y CSV; y un módulo "Estadísticas" para visualizar los datos por medio de gráficos y conocer el comportamiento de su aplicación.

REQUISITOS

- Python 2.7
- Django 1.10
- django-datatables-view 1.14
- turpial_auditor

INSTALACIÓN

Agregue "turpial_auditor_ui" a su configuración `INSTALLED_APPS` de esta manera:

```
INSTALLED_APPS = (  
    ...  
    'turpial_auditor_ui.auditor_commons',  
    'turpial_auditor_ui.auditor_reports',  
    'turpial_auditor_ui.auditor_statistics',  
    ...  
)
```

Nota: Puede no incluir **auditor_statistics** o **auditor_reports** si no desea las “Estadísticas” o “Reportes”, respectivamente.

MÓDULO ESTADÍSTICAS

Para incluir el módulo "Estadísticas" cree una plantilla, la llamaremos "yourTemplate.html", esta plantilla puede incluir que provee la librería, "turpial_statistics.html", que brinda toda la información relacionada con las estadísticas y su presentación:

yourTemplate.html

```
...
<body>
    ...

    {% include "turpial_statistics.html" %}

    ...
</body>
...
```

Luego, herede la vista **FilterView** en su archivo views.py y habilite una URL que llame a dicha vista:

views.py

```
from turpial_auditor_ui.auditor_statistics.views import FilterView

class YourView(FilterView):

    template_name = 'yourTemplate.html'
```

url.py

```
url(r'^url$', YourView.as_view(), name="other_name"),
```

Incluir gráficos

Puede que no desee utilizar la plantilla que proporcionamos en esta librería, pero es posible que desee utilizar los gráficos sin tener que lidiar con la lógica que lo respalda. Puede incluir solo los gráficos que son útiles para su negocio haciendo algo como:

```
{% include 'turpial_chart.html' with chart=< chart_data > %}
```

donde: <chart_data> es el JSON con la información de datos.

Las variables proporcionadas por la biblioteca son:

- line_chart_data.
- action_pie_chart_data.
- model_pie_chart_data.
- author_pie_chart_data.
- mod_act_chart_data.
- aut_act_chart_data.

Si desea agregar sus propios datos de gráficos, consulte la sección, estas variables deben heredar de ChartData "Crear sus propias variables de datos para gráficos".

Por ejemplo:

```
{% include 'turpial_chart.html' with chart_data=line_chart_data %}
```

Crea tus propios gráficos

Para agregar su propia variable de gráfico de datos, importe la clase **ChartData** y anule el método **get_context_data** agregando una nueva variable que contenga el objeto de su gráfico:

views.py

```
from turpial_auditor_ui.auditor_statistics.utils import ChartData

class YourView(FilterView):

    template_name = 'yourTemplate.html'

    def get_context_data(self, **kwargs):
        context = super(YourView, self).get_context_data(**kwargs)
        stats = build_stats()
        context['your_chart'] = ChartData(stats)
```

Nota: la variable **stats** puede ser de cualquier tipo que se pueda convertir a un JSON válido y que coincida con el **dataProvider** de Amcharts.

Si desea crear un gráfico más complejo, herede la clase ChartData. Asegúrese de ajustar el *type*, *id* y *theme* de su gráfico.

LISTA DE PLANTILLAS DISPONIBLES

- turpial_base.html - Ubicación: commons.
- turpial_statistics.html - Ubicación: estadísticas.
- turpial_chart.html - Ubicación: estadísticas
- turpial_card_stats.html - Ubicación: estadísticas.
- turpial_card.html - Ubicación: estadísticas.
- turpial_report.html - Ubicación: reportes.

Nota: No nombre sus archivos html con ninguno de estos nombres.

APÉNDICE I

Manual de instalación de Jenkins

MANUAL DE INSTALACIÓN DE JENKINS

Para instalar *Jenkins* se siguen los siguientes pasos:

Agregar el repositorio de *Jenkins*

```
wget -q -O - http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -  
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ > /etc/apt/sources.list.d/jenkins.list'
```

Actualizan los paquetes de *linux*

```
sudo apt-get install jenkins
```

Instalar *Jenkins*

```
sudo apt-get install jenkins
```

Después de la instalación, se verifica que el servidor corra con el comando:

```
sudo service jenkins status
```

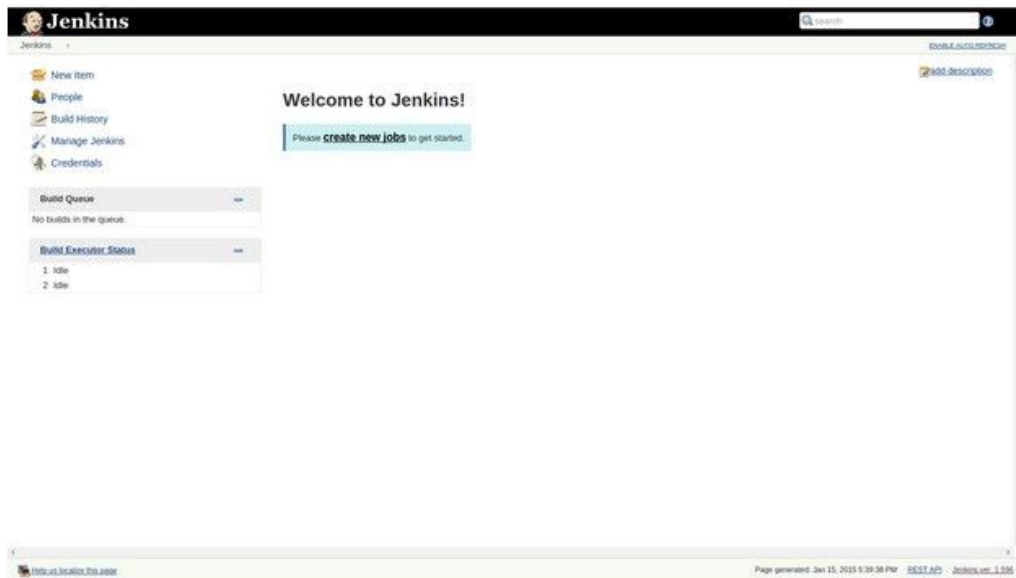
Si el servidor corre correctamente se observará el siguiente mensaje:

```
`Jenkins Continuous Integration Server is running with the pid XXX`
```

Para iniciar *jenkins*:

```
sudo service jenkins start
```

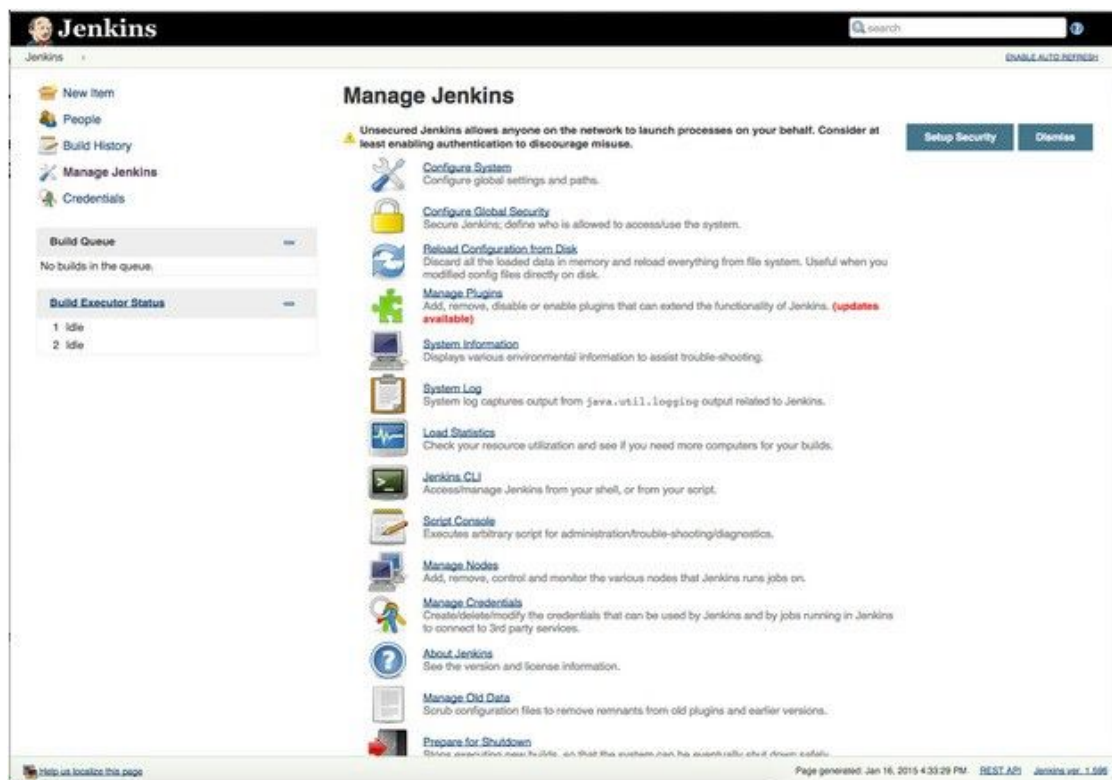
Ahora podemos acceder a Jenkins en el puerto 8080 desde el servidor. Vaya a su navegador y vaya a localhost:8080 (donde localhost es el nombre o la IP de su servidor).



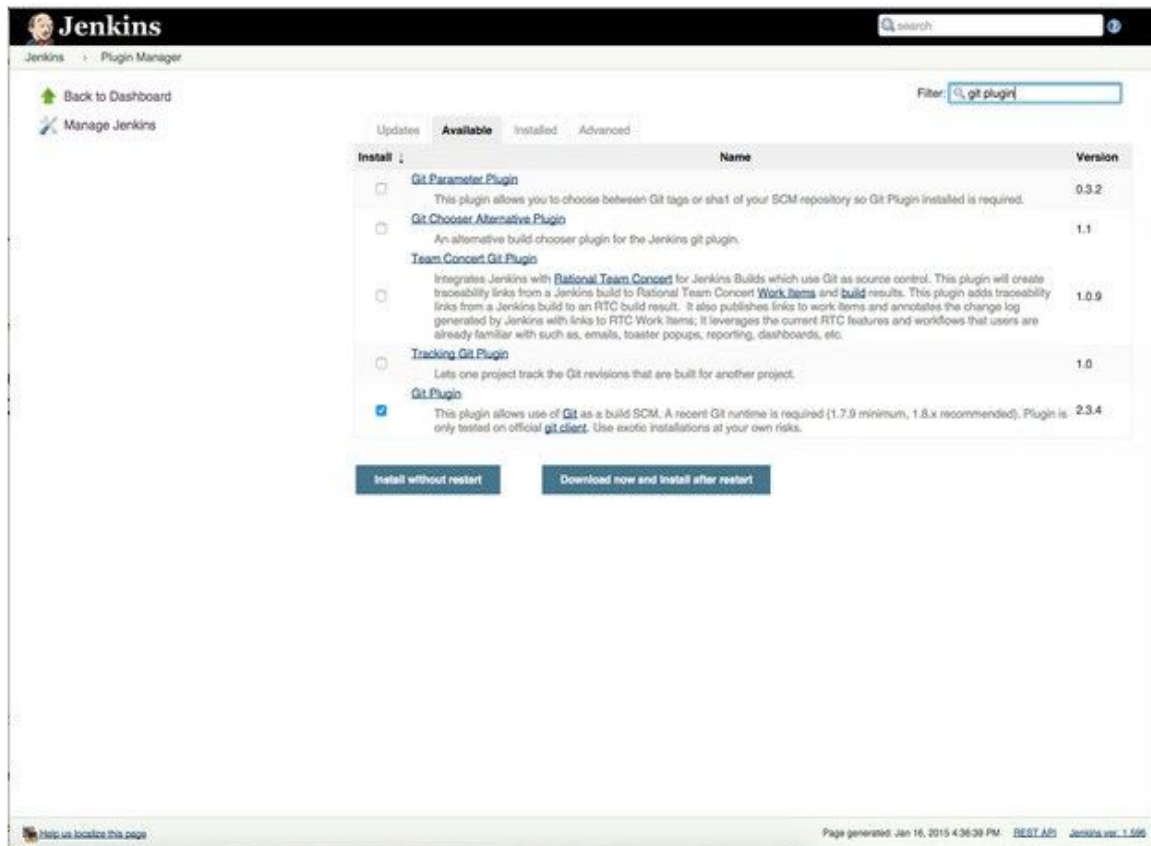
Integración con Git

1. Instalación del plugin de GIT

Jenkins, por defecto no accede a los repositorios de Git, por ello se debe instalar un plugin de Git. Por ello se debe ir a el servidor de *Jenkins*, se selecciona la opción de “*Manage Jenkins*” y en la próxima página hacer click en “*Manage Plugins*”.



En la página de plugins, hacer click en el pestaña de “Available” y filtrar por “git plugin”. Entonces se selecciona el plugin y luego se hace click en “Download now and install after restart”.



2. Configuración del plugin GIT

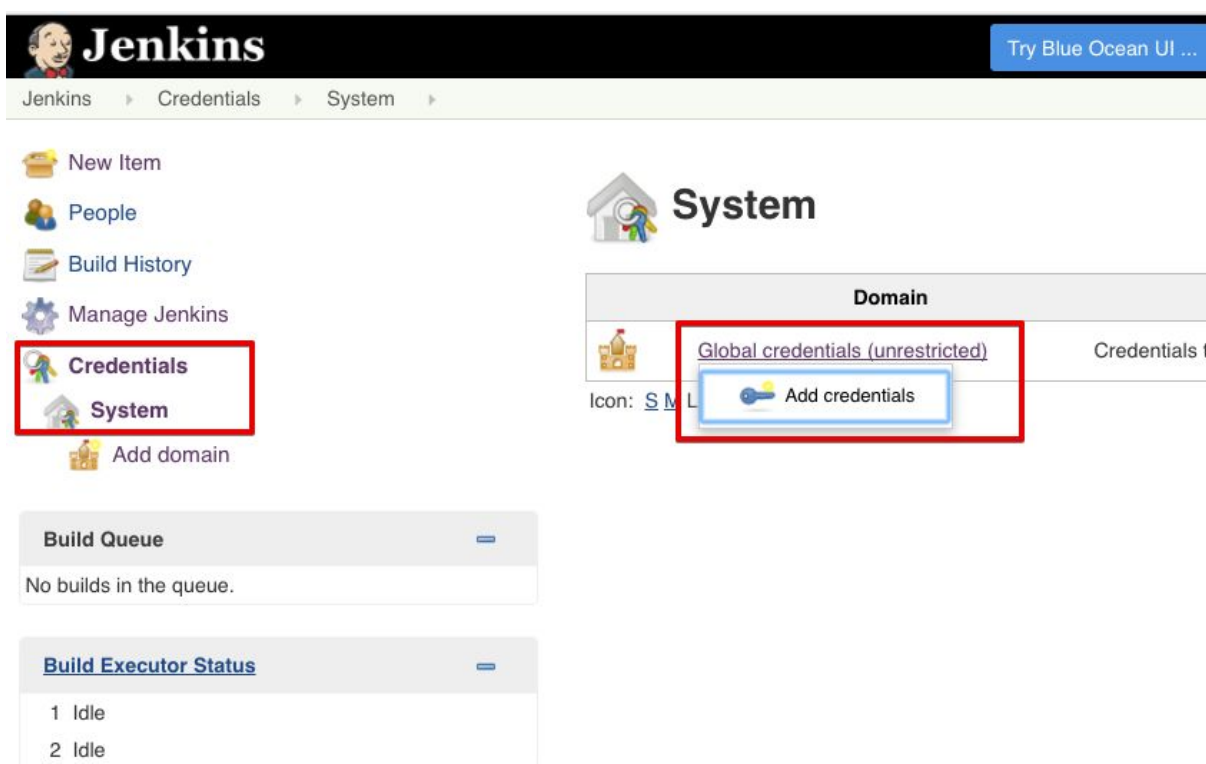
Para configurar el plugin del *Jenkins*, seguir los siguientes pasos:

1. En la página principal de *Jenkins*, haga clic en “New Item” para crear un nuevo trabajo. Escriba el nombre del trabajo y seleccione la opción “Freestyle project”.
2. En la página siguiente, seleccione la opción de “Discard Old Builds” para que las versiones anteriores se descarten automáticamente.
3. Configure el acceso al repositorio. Seleccione el tipo de repositorio (Git, Subversion, CVS), url, bifurcación para recoger, y así sucesivamente.

Credenciales de GIT

Si el repositorio es **privado** , deberá configurar las **credenciales** . Para hacerlo, haga lo siguiente:

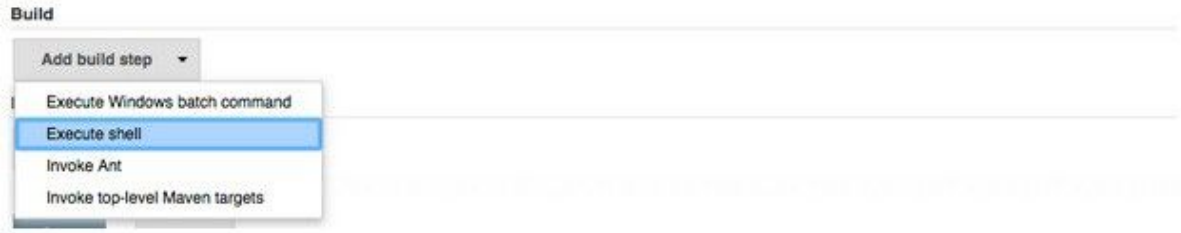
1. Vaya a "*Credentials*" (está en las opciones a la derecha). Debajo de "*Domain*" busque la opción "global".
2. Haga clic en el menú desplegable, luego haga clic en la opción "*Add credentials*".



3. Seleccione el "*king*" "*Username with password*" y escriba sus credenciales de Git o "*SSH user name with private key*". Si elige la última opción, tendrá que establecer una "*SSH key*".

Construir

Se agrega un paso en la compilación que ejecuta un script de shell luego de hacer *pull* del repositorio. Aseguresé de que los *scripts* ejecute todos los comandos que necesita para asegurarse de que la construcción sea exitosa.



Para verificar el resultado de la construcción, haga clic en él y busque la opción "*Console output*".

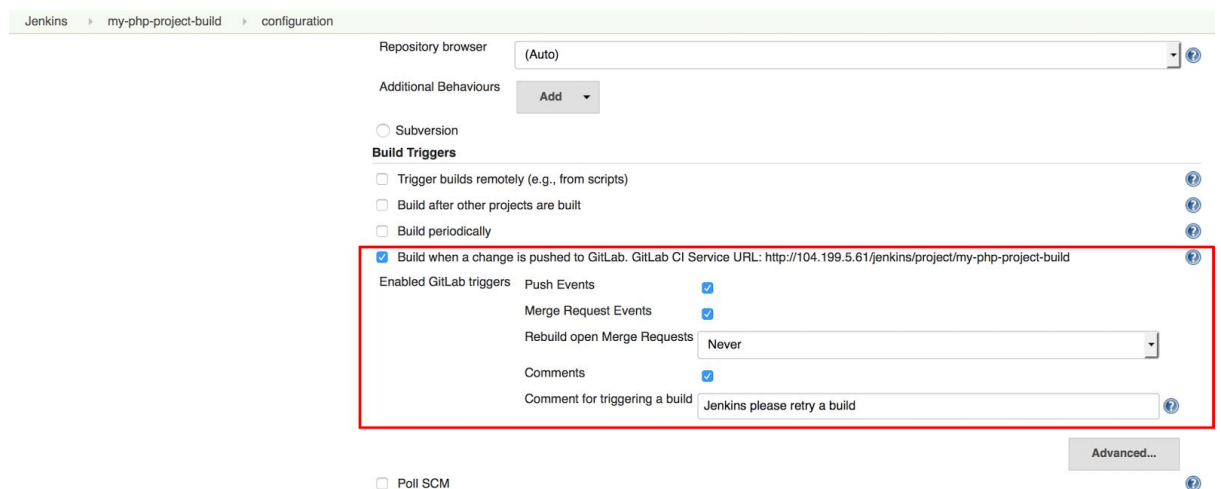
Integración de GitLab

1. Instalación del plugin GitLab

Vea la sección "*Git plugin installation*". Busque la palabra "*gitlab plugin*" en lugar de "*git plugin*".


2. Configuración del plugin Gitlab

En la página de configuración del proyecto, busque la sección "*Build triggers*" y marque la opción "*Build when a change is pushed to GitLab*".



3. Wewhooks

En la página del proyecto Gitlab, vaya a "*Settings*" y encuentre la opción "*Integration*" y haga clic en ella. Luego, desplácese hacia abajo y busque la opción "*Jenkins CI*",

Jenkins CI 

An extendable open source continuous integration server

You must have installed the Git Plugin and GitLab Plugin in Jenkins and enabled the GitLab project's 'Builds' feature.

Active

☒

Trigger

☒ **Push events**
This url will be triggered by a push to the repository

☐ **Merge Request events**
This url will be triggered when a merge request is created

Jenkins url

Project name

The URL-friendly project name. Example: my_project_name

Username

Change Password

SAVE CHANGES

TEST SETTINGS

CANCEL

Marque la opción "*Active*" y el "*Trigger*" que desea agregar. Complete la entrada de "*Jenkins URL*" con la proporcionada en la sección anterior. Completa los campos restantes.

Ahora todo debería estar establecido. Si envía una confirmación al repositorio, debería ver que el trabajo de Jenkins comienza a ejecutarse. Una vez que el trabajo finalice, debería ver el estado al lado de la confirmación en GitLab.

APÉNDICE J

Manual para crear una Librería

Cómo hacer que una librería de Django se pueda instalar usando PIP

Creación del archivo setup.py

Antes que nada, se necesita crear un archivo que contenga el nombre, descripción, autores, licencia, librerías requeridas etc. Por ejemplo:

```
import os
from setuptools import find_packages, setup

with open(os.path.join(os.path.dirname(__file__), 'README.rst')) as readme:
    README = readme.read()

# allow setup.py to be run from any path
os.chdir(os.path.normpath(os.path.join(os.path.abspath(__file__), os.pardir)))

setup(
    name='example',
    version='0.1',
    packages=find_packages(),
    include_package_data=True,
    license='BSD License', # example license
    description='A simple library example',
    long_description=README,
    url='http://an.example.com',
    author='Example author',
    author_email='author@example.com',
    classifiers=[
        'Environment :: Web Environment',
        'Framework :: Django',
        'Framework :: Django :: X.Y', # replace "X.Y" as appropriate
        'Intended Audience :: Developers',
        'License :: OSI Approved :: BSD License', # example license
        'Operating System :: OS Independent',
        'Programming Language :: Python',
        'Programming Language :: Python :: 2.7',
        'Programming Language :: Python :: 3.4',
        'Topic :: Internet :: WWW/HTTP',
        'Topic :: Internet :: WWW/HTTP :: Dynamic Content',
    ],
    install_requires=[
        'a_dependency', #A list of dependencies
    ],
)
```

Creación del archivo MANIFEST.in

A continuación, se debe crear el archivo **MANIFEST.in**, utilizado para incluir archivos adicionales en el paquete de la librería. Ejemplo:

```
include LICENSE
include README.rst

recursive-include path/to/directory/to/include *
```

Nota: El comando *recursive-include* permite agregar varios archivos dentro de un directorio; aquellos que coincidan con la expresión regular colocada luego del *path*.

Empaquetamiento e instalación de la librería

Para empaquetar la librería, en el terminal, se debe estar sobre el directorio en donde se encuentra `setup.py` y se procede a ejecutar el siguiente comando:

```
python setup.py sdist
```

Esto, crea un archivo *tar.gz* bajo un directorio llamado **dist**. A continuación, se puede instalar la librería utilizando el siguiente comando:

```
pip install path_to_library/dist/library_name.tar.gz
```

En donde el *path* es el que dirige al *tar.gz* mencionado anteriormente.

Referencias:

[Django](#)

[Stackoverflow](#)

APÉNDICE K

Prototipo tablas descentralizadas

Prototipo de tablas de auditorías descentralizadas

Justificación

La librería Auditorías Turpial, cuenta con una única tabla para almacenar las trazas de auditoría de cada modelo considerado auditable. Dado que cada manejador de base de datos tiene un límite de cantidad de información que puede almacenar, esta aproximación evita que esta tabla crezca.

Es por esta razón que se decidió experimentar con la creación de un prototipo en el que no se posea una única tabla de auditoría, sino que cada modelo auditable tenga la suya. En este documento se presenta el diseño e implementación de la solución, así como las ventajas y desventajas que posee frente al modelo centralizado.

Diseño e implementación

La librería Auditorías Turpial utiliza para alguna de sus funcionalidades la *signals* provistas por Django. Al investigar respecto a estas, se descubrieron otras que permiten capturar otros eventos. En particular, esta solución se enfoca en el uso de la señal “*class_prepared*” que es activada cuando un modelo está listo para ser utilizado.

Por otro lado, Python cuenta con la capacidad de crear clases de manera dinámica. Esto es beneficioso en caso de que se deseen usar técnicas como “metaprogramación”. En particular, resulta útil para crear una clase cuya estructura interna, dependa de la ejecución del programa en cuestión. La función “*type*” puede recibir tres argumentos que permiten crear una clase:

- *classname*: es un *string* que define el nombre de la clase.
- *superclasses*: es una lista o tupla con las superclases de la que se está construyendo.
- *atributtes_dict*: es un diccionario que funciona como el *namespace* de la clase, agregando los atributos que indique con la especificación proporcionada.

Ambas ideas pueden combinarse para provocar el comportamiento que se deseaba lograr; obtener un modelo por cada tabla auditable. Con la *signal* “*class_prepared*” se puede detectar cuando un nuevo modelo auditable es agregado y con la función “*type*” se puede crear una la tabla de auditoría con los atributos necesarios. En la figura 1, se muestra un ejemplo del código en cuestión.

```

@receiver(class_prepared)
def handler_class_prepared(sender, **kwargs):
    """Create an audit model for the specific class."""
    if issubclass(sender, AuditableMixin):
        name = str(sender.__name__ + 'Audit')

        class Meta:
            db_table = '%s_audit' % sender._meta.db_table
            app_label = 'auditor_core'

        # Set up a dictionary to simulate declarations within a class
        attrs = {
            '__module__': 'turpial_auditor.auditor_core.models',
            'Meta': Meta,
            'author': models.ForeignKey(settings.AUTH_USER_MODEL, null=True),
            'action': models.CharField(max_length=1, choices=ACTIONS_CHOICES),
            'datetime': models.DateTimeField(auto_now_add=True),
            'model_json_old': JSONField(),
            'model_json_new': JSONField(),
            '__str__': lambda self: auditToString(self)
        }

        model = type(name, (models.Model,), attrs)

```

Figura 1. Extracto del código para crear una tabla por modelo auditable.

Al nombre del modelo a auditar, se le agrega el sufijo “Audit” y al nombre de la base de datos se le añade “_audit” para diferenciarlo del que ya existe. Para este objeto, se agrega la superclase “models.Model” que es la que permite a Django considerarla un nuevo modelo y migrarlo. También, se agregan los mismos campos que poseen la tabla centralizada, exceptuando el “model_name” puesto que el nombre escogido para la clase ya indica a qué modelo pertenece.

El principal problema encontrado con esta solución, es una advertencia de que el modelo ya fue registrado.

Observaciones

- Si bien esta solución posee el comportamiento deseado, es necesario realizar una investigación más profunda para evitar la advertencia mencionada.
- A pesar de que descentralizar la información de las auditorías permite que la tabla no se sature tan rápidamente, generar varias tablas de auditoría puede tener una incidencia sobre el desempeño del sistema.

APÉNDICE L

Módulo Permisología

Creación de permisos

Django, crea nativamente una tabla de Permisos en la cual el programador puede agregar objetos cuando necesite nuevos permisos. Para esto, debe crear un objeto con al menos un código, una descripción y a que modelo pertenece.

El módulo Permisología, debe disponer del código que permita generar nuevos permisos en esta base de datos y que estén asociados con la tabla “AuditableAction”. Para facilitar la inclusión de los mismo, debe crearse una migración que automatice su inclusión al momento de instalar el módulo.

El manejo de la inclusión de un permiso particular a un usuario, puede realizarse a través del sitio administrador de Django. Adicionalmente, se puede incluir un Group, Auditor, que incluya ciertos permisos por defecto.

Tomando en cuenta las Historias de Usuario, algunos de los permisos de usuario podrían ser:

- Puede ver listados de auditoría.
- Puede generar reportes CSV.
- Puede generar reportes PDF.
- Puede visualizar las estadísticas.
- Puede visualizar el usuario que realizó alguna acción.
- Puede visualizar el modelo sobre el cual se realizó la acción.
- Puede inspeccionar los registros creados.
- Puede inspeccionar los registros actualizados.
- Puede inspeccionar los registros eliminados.

Mixin de para el control de los permisos.

Django posee una serie de *mixins* que permiten de alguna manera, restringir las funcionalidades sobre una vista particular. El “PermissionRequiredMixin” verifica si un usuario cuenta con todos los permisos definidos en la variable “permissions_required”, la cual es un iterable (lista o tupla). Si el usuario no posee alguno de estos, se envía un *response Forbidden* (403). Si no está autenticado, redirige a la página de “iniciar sesión”.

Se pueden generar dos *mixins* que hereden “PermissionRequiredMixin” para manejar los casos en los que se desean restringir los listados y las estadísticas, puesto que estas prohíben el acceso al URL.

Existe otro *mixin* de Django que proporciona parte de la lógica requerida para el resto de las funcionalidades, “AccessMixin”. Este puede ser usado para redirigir a la página de autenticación o levantar una excepción indicando que el usuario no posee los permisos si el atributo “raise_exception” es igual a True.

Para los permisos de generar reportes, se puede heredar el *mixin* anterior, colocando False el atributo “raise_exception” puesto que la intención no es enviar una respuesta *Forbidden* sino agregar en el contexto de la vista una variable que oculte el botón para descargar el reporte, según sea el caso.

El “AccessMixin” puede ser heredado para manejar el resto de los permisos. Ese *mixin* procesaría el *queryset*, eliminando la información que está restringida o evitando que ciertas variables sean agregadas al contexto de la vista que lo hereden.

APÉNDICE M

Comparación de Librería Auditorías Turpial con otras

Nombre de paquete	Django-reversion	Django-auditor	Django-simple-history	turpial-auditor (Pasantía)
Descripción	Es una extensión de Django que provee control de versiones para los modelos.	Una aplicación de Django que guarda logs con los cambios hechos a los objetos de una manera simple y granular.	Almacena el historial de un modelo. Permite observar y revertir los cambios desde el sitio administrador de Django.	Una extensión de Django que mantiene un historial de operaciones realizadas sobre ciertos modelos.
Estado	Producción/ Estable	Estable	Producción/ Estable	Desarrollo
Mecanismo de Trigger	Middleware/ Decorador/ Manejo de contexto	No posee. Se debe colocar el código en cada vista.	Señales (Save).	Sobreescribiendo el método save() y con señales
Soporte de Rollback/Revert	Si.	No.	Si.	No está contemplado.
Almacenamiento de historial	Otro modelo.	Otro modelo.	Misma tabla.	Otro modelo.
Rastreo de relaciones	Si.	No.	No especifica.	Clave foránea