UNIVERSIDAD ANDINA DEL CUSCO
FACULTAD DE INGENIERÍA Y ARQUITECTURA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

UAC

TESIS

"USO DE LA TÉCNICA DE DISEÑO TEST DRIVEN DEVELOPMENT PARA EL DESARROLLO DE SOFTWARE.
CASO DE PRUEBA EXPERIMENTAL: SOFTWARE DE APOYO PARA LA COOPERATIVA DE AHORRO Y CRÉDITO EL AMAUTA LTDA"

PRESENTADO POR:
BR. REDY DEMETRIO DELGADO SEQUEIROS

PARA OPTAR AL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS.

ASESOR:
MGT. ING. WILLIAM ALBERTO CHÁVEZ ESPINOZA

CUSCO – PERÚ

2018

DEDICATORIA

Dedico este trabajo de manera especial a:

El centro de mi vida, a mi pequeña hija Ariana Rafaela, por ser mi motivación día a día para ella con todo el amor que pueda caber en mí.

A mi pareja Patricia. Por el apoyo incondicional que siempre me ha brindado.

A mi hermano Reyner pues fue el principal cimiento para la construcción de mi vida profesional sentó en mí, las bases de responsabilidad y deseos de superación, me ha dado todo lo que soy como persona, mis valores y principios. Él supo ser padre para mí y mi pequeña hermana aun siendo muy joven.

También agradecer a mi Madre, por darme la oportunidad de ser alguien en la vida.

.

AGRADECIMIENTOS

Agradezco a DIOS quien me da lo necesario para seguir adelante y de manera especial a la FACULTAD DE INGENIERIA Y ARQUITECTURA DE LA UNIVERSIDAD ANDINA DEL CUSCO, por inculcarme conocimientos y capacidad de investigación a través de su plana docente, al Ingeniero MG. ING. William Alberto Chávez Espinoza por ser la maravillosa persona que es, por que siempre ha estado dispuesto a ayudarme en cumplir con este objetivo que tengo en mi vida.

A mis dictaminantes Ingeniero Lida Leon Nuñez y al Ingeniero Luis Alberto Sota Orellana, porque gracias a sus aportes y buena voluntad lograron hacer de este trabajo de investigación un mejor trabajo.

A todos ellos GRACIAS GRACIAS GRACIAS.

RESUMEN

La técnica de diseño Test Driven Development es un componente de la metodología eXtreme Programming, esta técnica se enfoca en el desarrollo de software guiado por pruebas antes de la escritura del código, lo cual demuestra una perspectiva de desarrollo de software fuera de lo común.

En este trabajo de investigación, la técnica de diseño Test Driven Development será evaluada para determinar su factibilidad de uso en la creación de un software partiendo de las pruebas hasta conseguir el código necesario que ejecute el funcionamiento deseado; por lo que, se ha de tomar como caso de prueba experimental la creación de un nuevo software para la cooperativa de ahorro y crédito El Amauta Ltda.

La cooperativa de ahorro y crédito El Amauta Ltda. actualmente no cuenta con un mecanismo de control que permita organizar y sistematizar las actividades primigenias que desarrolla cada uno de sus colaboradores, es por lo que se ha de usar como caso de prueba experimental la necesidad de organización que presenta esta institución, con el fin de determinar la factibilidad de uso de esta técnica en un entorno real.

Para poder realizar este trabajo de investigación se usará componentes de código abierto como laravel, bootstrap y phpunit para obtener un entorno de desarrollo completo, y poder realizar las pruebas unitarias necesarias; también se ha de usar la metodología de desarrollo Scrum en combinación con la técnica de diseño Test Driven Development.

ABSTRACT

The design technique Test Driven Development is a component of the eXtreme Programming methodology, this technique focuses on the development of software guided by tests before the writing of the code, which demonstrates a perspective of software development out of the ordinary.

In this research work, the design technique Test Driven Development will be evaluated to determine its feasibility of use in the creation of a software starting from the tests until getting the necessary code that executes the desired operation; Therefore, the creation of a new software for the credit union El Amauta Ltda.

The credit union El Amauta Ltda. Currently does not have a control mechanism that allows organizing and systematizing the original activities developed by each of its collaborators, which is why it has to be used as an experimental test case the need to organization that presents this institution, in order to determine the feasibility of using this technique in a real environment.

In order to carry out this research work we will use open source components such as laravel, bootstrap and phpunit to obtain a complete development environment, and be able to perform the necessary unit tests; The Scrum development methodology must also be used in combination with the Test Driven Development technique.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE IMÁGENES

ÍNDICE DE GRÁFICOS

INTRODUCCIÓN

Es imperativa la necesidad de usar métodos de desarrollo de software ágil para mejorar el proceso de desarrollo de software, puntualmente, el binomio desarrollo - programador, añadiendo un elemento trascendental como es la tecnología.

En esta década, somos testigos de cómo el hardware y el software dan grandes pasos en tiempos relativamente reducidos, debido a su gran demanda e innovación tecnológica.

Partiendo de la premisa anterior, nace la necesidad de entender y experimentar nuevas opciones en cuanto a desarrollo de software, con el propósito de mejorar en calidad y tiempo de desarrollo.

En este trabajo de investigación, haremos uso de la técnica de diseño Test Driven Development enfocado en las pruebas unitarias para crear un nuevo software que organice de mejor manera las actividades diarias realizadas por los colaboradores de la Cooperativa de Ahorro y Crédito El Amauta Ltda.

> ***Es como explorar una casa oscura. Usted va desde habitación a habitación a habitación. Escribir la prueba es como encender la luz. Entonces puedes evitar los muebles y guarda tus espinillas (el diseño limpio resultante de la refactorización). Entonces estás listo para explorar la siguiente habitación.***[1]

Este nuevo software estará compuesto por Laravel que es un framework de desarrollo back-end que posee entre sus ventajas una arquitectura predefinida MVC (modelo vista controlador), que separa la estructura de una aplicación web a desarrollar, separándola de una forma más organizada y simple y que dentro de su estructura tiene inmerso a phpUnit que nos servirá para realizar las pruebas unitarias según vayamos desarrollando el software, con el que estaremos seguros de que cualquier modificación en la escritura del código no nos perjudicara en nuevas incorporaciones del mismo.

> ***Cada proyecto que utiliza el desarrollo impulsado por pruebas sigue tres pasos sencillos repetidamente:***
>
> 1. ***Escriba una prueba para el siguiente bit de funcionalidad que desea agregar.***
> 2. ***Escriba el código funcional hasta que pase la prueba.***

[1] (Beck & Instituto de los Tres Ríos, 2002, pág. 3)

3. ***Refactorizar el código nuevo y viejo para hacerlo bien estructurado.***

Continúe haciendo un ciclo a través de estos tres pasos, una prueba a la vez, aumentando la funcionalidad del sistema. Las pruebas le ayudarán a refactorizar, lo que le permite mejorar su diseño con el tiempo y hace que algunos problemas de diseño sean más obvios.[2]

Es decir, con cada prueba que se escriba y se ejecute, arrojará un error hasta que se supere la prueba; mientras que los test de pruebas ya concluidos nos mostrarán un mensaje "correcto". Permitiéndole al desarrollador tener una trazabilidad del código erróneo.

Las pruebas unitarias para este trabajo de investigación serán alrededor de 36 test de pruebas, en los cuales se ira explicando su proceso de desarrollo, al igual que los inconvenientes encontrados.

Para realizar las pruebas unitarias, primero se estructurará los casos de uso de los usuarios que intervienen en el sistema identificando los requerimientos precisos del usuario. De tal forma el uso de la técnica de diseño Test Driven Development, nos permitirá con antelación, obtener una visión más clara y precisa de lo que se ha de desarrollar.

El uso de esta técnica para el desarrollo de este nuevo software estará acompañado de la metodología ágil SCRUM, en el cual se han de realizarán 2 Sprints.

El objetivo de este trabajo de investigación es determinar la factibilidad de uso de la técnica de diseño Test Driven Development aplicado al desarrollo de software.

Por ello, el presente trabajo de investigación comprende temas de desarrollo web basados en el modelo vista controlador, siguiendo una técnica de diseño ágil y dinámica enfocada en las necesidades del usuario guiada por una metodología ágil como Scrum.

Al termino, se podrá conocer conclusiones obtenidas de este trabajo de investigación.

[2] (Khalili, 2016)

CAPÍTULO I

ASPECTOS GENERALES

1.1.DESCRIPCIÓN DE LA SITUACIÓN ACTUAL

La cooperativa de ahorro y crédito El Amauta Ltda, es una institución sin fines de lucro cuyo fin es servir a sus socios a través de créditos y retribuciones financieras obtenidas por sus aportaciones y ahorros.

Esta institución requiere de un nuevo software sobre los procesos aún no automatizados para el control de su información como: registros de operaciones financieras por los operadores de caja, control de actividades de los analistas de créditos y control de aprendizaje de los colaboradores en las diferentes capacitaciones brindadas por la institución.

Descripción general de la necesidad de procesos a sistematizar en la institución:

a) El actual método de registro de datos financieros de ingresos y salidas, que se realiza en su sistema financiero, no brinda directamente la información requerida a diario por los analistas, operadores de caja y administrativos externos. Lo que ocasiona una carga adicional por parte de los analistas de créditos sobre el único operador que labora en la institución, quien es saturado de consultas sobre el avance del estado crediticio de los socios con tentativa de problemas potenciales de pago; es necesario mencionar que los socios no pueden realizar el seguimiento de sus operaciones financieras realizadas en la caja de forma inmediata.

b) Por otro lado, los analistas no tienen forma alguna para llevar a cabo el registro diario de las actividades realizadas durante el día y por ende no es posible controlar dichas actividades con precisión por parte del administrador.

c) Asimismo, la cooperativa de ahorro y crédito El Amauta Ltda. necesita llevar el control de aprendizaje de sus colaboradores (operador de caja, analistas y administrativos externos), ya que este control es un requisito necesario para la institución como se detalla en el Artículo 12º del reglamento del comité de educación de la institución. Este proceso que aún no se encuentra modelado, ni sistematizado; genera un desorden y una desigualdad ante el ingreso de nuevos

colaboradores a la institución, por ello no se puede llevar a cabo el control deseado del estado de aprendizaje de cada colaborador individualmente.

1.2.FORMULACIÓN DEL PROBLEMA

En el mundo se crean nuevos sistemas informáticos (software) cada vez más complejos, eficientes, escalables y con el uso de menos recursos (tiempo de desarrollo, dinero, horas hombre en pruebas); usando combinaciones de metodologías, técnicas, frameworks que no necesariamente están unidas entre sí. Todo esto con el fin de mejorar en el desarrollo, implementación y producción de software que satisfaga las necesidades y/o requerimientos de los usuarios de este siglo porque hoy en día nuestros equipos tecnológicos son dependientes de algún software para cumplir con su funcionamiento esperado.

Según el instituto de CMMI[3] (Capability Maturity Model Integration) creado por el SEI[4] (Software Engineering Institute). El cual se encarga de evaluar y medir la madurez del proceso de desarrollo de software en una organización. Siendo su escala L1 (caótico) a L5 (mejora continua). Perú como productor de software se ubican en el puesto 22 de 83 países.

[3] Actualmente instituto que maneja un estándar de mejora y evaluación de procesos, cuyas practicas están centras en el desarrollo, mantenimiento y operación de sistemas de software.
[4] Instituto federal estadounidense que se centra en la investigación y desarrollo relacionado con software y ciber seguridad.

País / Nivel	L2	L3	L4	L5	Total
China	20	1843	87	128	**2078**
United States	267	639	12	53	**971**
India	14	381	2	148	**545**
Mexico	110	84	8	22	**224**
Spain	58	62		13	**133**
Korea, Republic Of	29	85	11	6	**131**
Brazil	48	52	1	7	**108**
Colombia	5	64		15	**84**
Japan	13	42	8	6	**69**
France	32	21			**53**
Thailand	9	38		3	**50**
Taiwan	14	23	1		**38**
Turkey	1	32	1	1	**35**
Italy	12	18		1	**31**
Chile	15	10		5	**30**
Germany	11	19			**30**
United Kingdom	6	18		3	**27**
Argentina	4	14		8	**26**
Portugal	13	7		5	**25**
Viet Nam		21		4	**25**
Canada	8	14		3	**25**
Peru	3	17		3	**23**
Philippines		11		6	**17**

Imagen 1: Lista de Empresas certificadas por el CMMI al 10 de septiembre de 2015

En Latinoamérica, Perú está por debajo de México, Brasil, Colombia, Chile, Argentina. Siendo el quinto país, productor de software en Latinoamérica, conteniendo un total de 23 empresas certificadas por él CMMI de un ranking que tiene como primer puesto a México con 224 empresas certificadas por él CMMI. Es decir que Perú tiene el 10.3% de empresas certificadas por el CMMI del total de empresa de México.

Por lo que sucede con las empresas de reciente inicio en el ámbito de desarrollo de software y/o programadores independientes que aún no cuentan con la experiencia en la creación de software, ofrecen a sus clientes herramientas tecnológicas para la sistematización de sus procesos y consumo de su propia información, sistemas informáticos que escasas veces son enfocados en las necesidades propias del cliente especifico siendo estos, sistemas informáticos que contienen más de lo necesario y poco de lo requerido disfrazado por características superfluas para el usuario, pero que ofrecen en un futuro incierto el uso de sus funcionalidades no pedidas pero acomodas dentro del sistema, creando de esta forma

sistemas informáticos con posibles bug's o errores que se han de mostrar con el uso del sistema y que ocasionarían posibles sobre costos en el mantenimiento del mismo no cumpliendo con el ciclo de vida del software esperado(creación, uso esperado y mantenimiento programado).

1.3.FORMULACIÓN DEL PROBLEMA

1.3.1. Formulación Interrogativa del Problema General

¿En qué medida el uso de la técnica de diseño Test Driven Development será factible en el desarrollo de un software?

1.3.2. Formulación Interrogativa de los Problemas Específicos

- ¿La construcción del software a través del uso de la técnica de diseño Test Driven Development satisfará al usuario final?
- ¿En qué medida el software construido con la técnica de Test Driven Development es factible en su proceso de desarrollo?
- ¿Cómo influirá en el desarrollador del software la propuesta brindada por la técnica de diseño Test Driven Development?

1.4.OBJETIVOS

1.4.1. Objetivo General

Desarrollar un software basado en el uso de la técnica de diseño Test Driven Development, centrándose en el comportamiento de la prueba y no en la implementación.

1.4.2. Objetivos Específicos

- Medir el grado de satisfacción del usuario final basado en el uso de la técnica de diseño Test Driven Development que permita entender el desarrollo del software guiado por pruebas.
- Probar la factibilidad de escribir primero las pruebas antes que el código por cada proceso del caso de prueba experimental: procesos de registro financieros, control de actividades y seguimiento de aprendizaje de los colaboradores.
- Determinar los aspectos positivos y negativos del uso de la técnica de diseño Test Driven Development por parte del desarrollador, después de crear un software basado para el caso de prueba experimental.

1.5.HIPÓTESIS

1.5.1. Hipótesis General

El desarrollo de software usando la técnica de diseño Test Driven Development, permite obtener un software enfocado en el requerimiento del usuario y testeado desde la concepción de su desarrollo.

1.5.2. Sub Hipótesis

- El software construido en base a la técnica de diseño Test Driven Development satisface al usuario final.
- En el proceso de desarrollo de la prueba del software, se demostró que la aplicación de la técnica de diseño Test Driven Development es factible.
- El software diseñado a través de la técnica de diseño Test Driven Development posee más aspectos positivos que aspectos negativos para ser usados en el proceso de desarrollo del software.

1.6.VARIABLES

- Factibilidad del uso de la técnica de diseño Test Driven Development.

1.7.JUSTIFICACIÓN DE LA INVESTIGACIÓN

El presente trabajo de investigación tiene como fin de mostrar a la sociedad de desarrolladores de software's principiantes y expertos, el uso de la técnica de diseño Test Driven Development como una buena práctica para el desarrollo de software basado en la creación de test de pruebas antes del código para el desarrollo de software, enfocado en los requerimientos esenciales del usuario final.

1.8.METODOLOGÍA

1.8.1. Tipo de Investigación

Para el desarrollo del presente trabajo de investigación se usará el tipo de investigación aplicada[5] debido a que experimentaremos el uso de la técnica de diseño Test Driven Development que pertenece a la metodología de desarrollo de software ágil XP (eXtreme Programming), pero que para esta oportunidad la combinaremos con SCRUM.

[5] (UC, 2018)

1.8.2. Nivel de Investigación

El nivel de investigación será la explicativa[6] o causal, debido a que conocemos el objeto del análisis, por lo cual relacionaremos los datos obtenidos previamente con el fin de determinar la funcionalidad y complejidad del objeto en estudio en base al uso de una técnica.

1.8.3. Método de Investigación

La metodología que se usará en el presente trabajo de investigación será el método analítico[7]; por lo que iniciaremos con el estudio de cada una de las partes que intervendrán para de esta manera analizar los resultados al término del presente trabajo de investigación.

[6] (Rica, 2017)
[7] (Limón, s.f.)

1.9.MATRIZ DE CONSISTENCIA

FORMULACIÓN DEL PROBLEMA		OBJETIVOS		HIPÓTESIS		METODOLOGIA DE LA TESIS	
Problema General	**Problemas Específicos**	**Objetivo General**	**Objetivos Específicos**	**Hipótesis General**	**Sub Hipótesis**	**Tipo De Investigación**	**Nivel De Investigación**
¿En qué medida el uso de la técnica de diseño Test Driven Development será factible en el desarrollo de un software?	¿La construcción del software a través del uso de la técnica de diseño Test Driven Development satisfará al usuario final?	Desarrollar un software basado en el uso de la técnica de diseño Test Driven Development, centrándose en el comportamiento de la prueba y no en la implementación.	Medir el grado de satisfacción del usuario final basado en el uso de la técnica de diseño Test Driven Development que permita entender el desarrollo del software guiado por pruebas.	El desarrollo de software usando la técnica de diseño Test Driven Development, permite obtener un software enfocado en el requerimiento del usuario y testeado desde la concepción de su desarrollo.	El software construido en base a la técnica de diseño Test Driven Development satisface al usuario final.	Para el desarrollo del presente trabajo de investigación se usará el tipo de investigación aplicada debido a que experimentaremos el uso de la técnica de diseño Test Driven Development que pertenece a la metodología de desarrollo de software ágil XP (eXtreme Programming), pero que para esta oportunidad la	El nivel de investigación será la explicativa o causal, debido a que conocemos el objeto del análisis, por lo cual relacionaremos los datos obtenidos previamente con el fin de determinar la funcionalidad y complejidad del objeto en estudio en base al uso de una técnica.
	¿En qué medida el software construido con la técnica de Test Driven		Probar la factibilidad de escribir primero las pruebas antes que el código por		En el proceso de desarrollo de la prueba del software, se demostró que la		

	Development es factible en su proceso de desarrollo?		cada proceso del caso de prueba experimental: procesos de registro financieros, control de actividades y seguimiento de aprendizaje de los colaboradores.		aplicación de la técnica de diseño Test Driven Development es factible.	combinaremos con SCRUM.	
	¿Cómo influirá en el desarrollador del software la propuesta brindada por la técnica de diseño Test Driven Development?		Determinar los aspectos positivos y negativos del uso de la técnica de diseño Test Driven Development por parte del desarrollador, después de crear		El software diseñado a través de la técnica de diseño Test Driven Development posee más aspectos positivos que aspectos		

			un software basado para el caso de prueba experimental.		negativos para ser usados en el proceso de desarrollo del software.		

Tabla 1: Cuadro de Matriz de Consistencia

Fuente: Elaboración Propia

CAPÍTULO II

MARCO TEÓRICO

2.1. ASPECTOS TEÓRICOS PERTINENTES

Para entender que es Test Driven Development y su forma de aplicación, es necesario comprender algunos conceptos importantes para el entendimiento de este trabajo de investigación.

2.1.1. Test Driven Development

(Carlos Blé Jurado y Colaboradores, 2010) explica en su libro Capitulo 2 ¿Qué es el Desarrollo guiado por Tests?.(TDD):

"...es una técnica de diseño e implementación de software incluida dentro de la metodología XP. Coincido con Peter Provost en que el nombre es un tanto desafortunado; algo como Diseño Dirigido por Ejemplos hubiese sido quizás más apropiado. TDD es una técnica para diseñar software que se centra en tres pilares fundamentales:

La implementación de las funciones justas que el cliente necesita y no más.

La minimización del número de defectos que llegan al software en fase de producción.

La producción de software modular, altamente reutilizable y preparado para el cambio."(p. 48)

También en la siguiente hoja de su libro menciona que TDD es la respuesta a:

"¿Cómo lo hago?, ¿Por dónde empiezo?, ¿Cómo sé qué es lo que hay que implementar y lo que no?, ¿Cómo escribir un código que se pueda modificar sin romper funcionalidad existente?"(p. 49)

Entonces se puede definir a TDD como una técnica de diseño de software, que está orientada a cubrir los requerimientos básicos del usuario final minimizando errores del software. Por qué se ha de iniciar con el desarrollo de software a partir de las pruebas.

2.1.2. Prueba unitaria

El portal web (Rouse, searchsoftwarequality, 2017) explica:

"La prueba unitaria es un proceso de desarrollo de software en el que las partes más pequeñas comprobables de una aplicación, llamadas unidades, se analizan de forma individual e independiente para un funcionamiento adecuado. Las pruebas unitarias pueden realizarse manualmente, pero a menudo son automáticas.

Las pruebas unitarias son un componente del desarrollo impulsado por pruebas (TDD) , una metodología pragmática que adopta un enfoque meticuloso para construir un producto por medio de pruebas y revisiones continuas. El desarrollo basado en pruebas requiere que los desarrolladores primero escriban pruebas de unidades que fallan. Luego escriben código y refactorizan la aplicación hasta que la prueba pase. TDD normalmente da como resultado una base de código explícita y predecible.

Las pruebas unitarias involucran solo aquellas características que son vitales para el rendimiento de la unidad bajo prueba. Esto alienta a los desarrolladores a modificar el código fuente sin preocupaciones inmediatas sobre cómo dichos cambios pueden afectar el funcionamiento de otras unidades o el programa en su conjunto. Una vez que se ha encontrado que todas las unidades de un programa funcionan de la manera más eficiente y sin errores posible, los componentes más grandes del programa se pueden evaluar mediante pruebas de integración.

Las pruebas unitarias tienen una curva de aprendizaje pronunciada. El equipo de desarrollo debe aprender qué es la prueba unitaria, cómo realizar pruebas unitarias, qué pruebas unitarias y cómo usar herramientas de software automáticas para facilitar el proceso de forma continua. El gran beneficio de las pruebas unitarias es que cuanto más temprano se identifica un problema, menos errores compuestos se producen. Un error compuesto es uno que no parece romper nada al principio, pero que finalmente entra en conflicto con algo más adelante y resulta en un problema."

Las pruebas unitarias son test pequeños automatizados listos para ser ejecutados, estas pruebas unitarias pueden contener más de una forma y pueden ser más de uno para un mismo caso. Las pruebas unitarias son componentes fundamentales de TDD

2.1.3. Laravel

En el libro de (Gallego, 2016) define a Laravel como:

"Laravel es un framework de código abierto para el desarrollo de aplicaciones web en PHP 5 que posee una sintaxis simple, expresiva y elegante. Fue creado en 2011 por Taylor Otwell, inspirándose en Ruby on Rails y Symfony, de los cuales ha adoptado sus principales ventajas.

Laravel facilita el desarrollo simplificando el trabajo con tareas comunes como la autenticación, el enrutamiendo, gestión sesiones, el almacenamiento en caché, etc. Algunas de las principales características y ventajas de Laravel son:

- *Esta diseñado para desarrollar bajo el patrón MVC (modelo - vista - controlador), centrándose en la correcta separación y modularización del código. Lo que facilita el trabajo en equipo, así como la claridad, el mantenimiento y la reutilización del código.*
- *Integra un sistema ORM de mapeado de datos relacional llamado Eloquent aunque también permite la construcción de consultas directas a base de datos mediante su Query Builder.*
- *Permite la gestión de bases de datos y la manipulación de tablas desde código, manteniendo un control de versiones de las mismas mediante su sistema de Migraciones.*
- *Utiliza un sistema de plantillas para las vistas llamado Blade, el cual hace uso de la cache para darle mayor velocidad. Blade facilita la creación de vistas mediante el uso de layouts, herencia y secciones.*
- *Facilita la extensión de funcionalidad mediante paquetes o librerías externas. De esta forma es muy sencillo añadir paquetes que nos faciliten el desarrollo de una aplicación y nos ahorren mucho tiempo de programación.*
- *Incorpora un intérprete de línea de comandos llamado Artisan que nos ayudará con un montón de tareas rutinarias como la creación de distintos componentes de código, trabajo con la base de datos y migraciones, gestión de rutas, cachés, colas, tareas programadas, etc."*

Laravel es un framework robusto con particularidades únicas que nos permite construir cualquier sistema de información en menor tiempo y con mas prestaciones para realizar mantenimientos a en un futuro.

2.1.4. MVC

(Gallego, 2016) define el MVC como:

El modelo–vista–controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

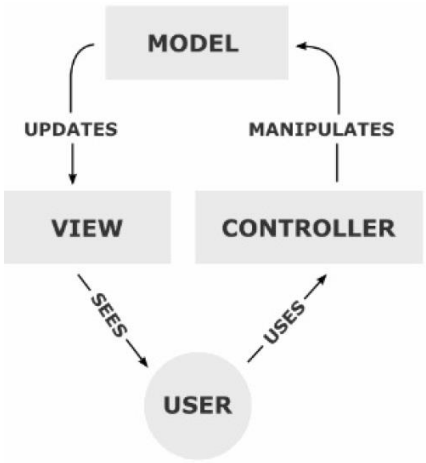


Imagen 2: Estructura de MVC

Elaborado por: (Gallego, 2016)

De manera genérica, los componentes de MVC se podrían definir como sigue:

- ***El Modelo:** Es la representación de la información con la cual el sistema opera, por lo tanto, gestiona todos los accesos a dicha información, tanto consultas como actualizaciones. Las peticiones de acceso o*

manipulación de información llegan al 'modelo' a través del 'controlador'.

- ***El Controlador:*** *Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud de información (por ejemplo, editar un documento o un registro en una base de datos). Por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.*

- ***La Vista:*** *Presenta el 'modelo' y los datos preparados por el controlador al usuario de forma visual. El usuario podrá interactuar con la vista y realizar otras peticiones que se enviarán al controlador.* (p. 7)

Es una estructura de software, donde se divide en tres componentes claves: el modelo, la vista y el controlador cada uno dependiente del otro con el fin de gestionar los archivos no solamente de forma organizada sino mejor diseñada para ofrecer tiempos de respuestas relativamente rápidos.

2.1.5. PhpUnit

En el portal web de PHPUnit (Bergmann, 2001) y el portal web PEAR (Pear, 2011) dan a conocer lo siguiente:

"PHPUnit proporciona un marco simple para crear un conjunto de pruebas para automatizar las pruebas de funciones y clases. PHPUnit está inspirado en JUnit, que fue creado por Kent Beck y Erich Gamma como una herramienta para eXtreme Programming. Una de las reglas de XP es probar componentes de software pequeños tan pronto como sea posible, de esta forma no tendrá que corregir errores en la API mientras configura y prueba aplicaciones más grandes que dependen de la clase. Si bien las pruebas unitarias son una de las reglas fundamentales en XP, no es necesario cambiar a XP para beneficiarse de PHPUnit. PHPUnit se destaca como una buena herramienta para probar clases o un conjunto de funciones y facilitará tu ciclo de desarrollo y te ayudará a evitar interminables sesiones de depuración.

Rutina de trabajo

Normalmente, escribirías una clase, harías algunas pruebas no sistemáticas usando echo () o var_dump (). Después de esto, usa la clase en su aplicación y espera que

todo esté bien. Para beneficiarse de PHPUnit debes replantearte el flujo. La mejor manera es hacer esto:

1. diseña tu clase / API

2. crea un conjunto de prueba

3. implementar la clase / API

4. ejecutar el conjunto de pruebas

5. corrige fallas o errores y ve al # 4 otra vez

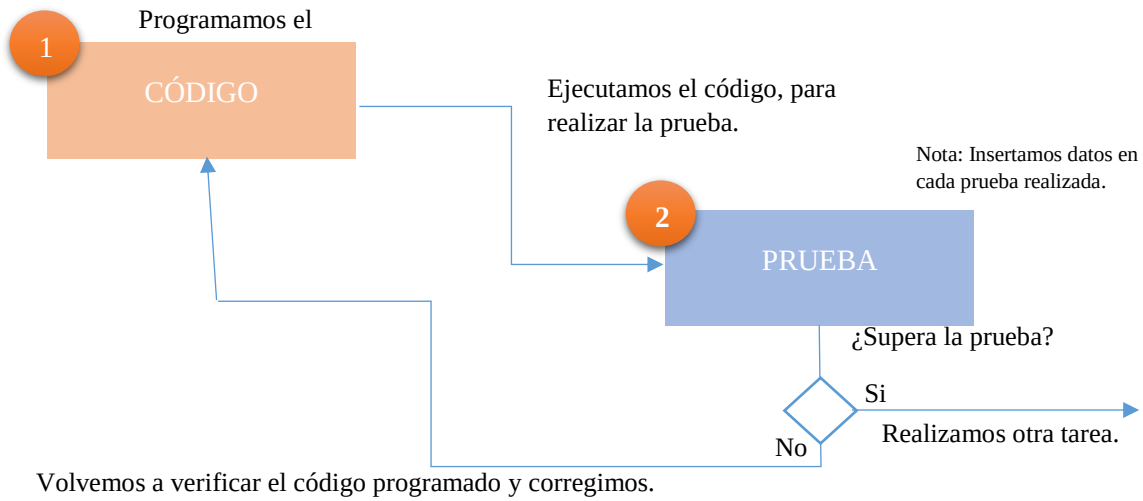
Puede parecer que esto requerirá mucho tiempo, pero esta impresión es incorrecta. La creación del conjunto de pruebas utilizando PHPUnit solo requiere unos minutos y ejecutar el conjunto de pruebas solo en unos segundos."

Esta librería creada por Sebastian Bergmann inspirada por la librería Junit, no necesariamente es aplicada en la metodología XP, pero que si es usada para probar modulo individuales o secuenciales siguiendo una sucesión lógica de pasos.

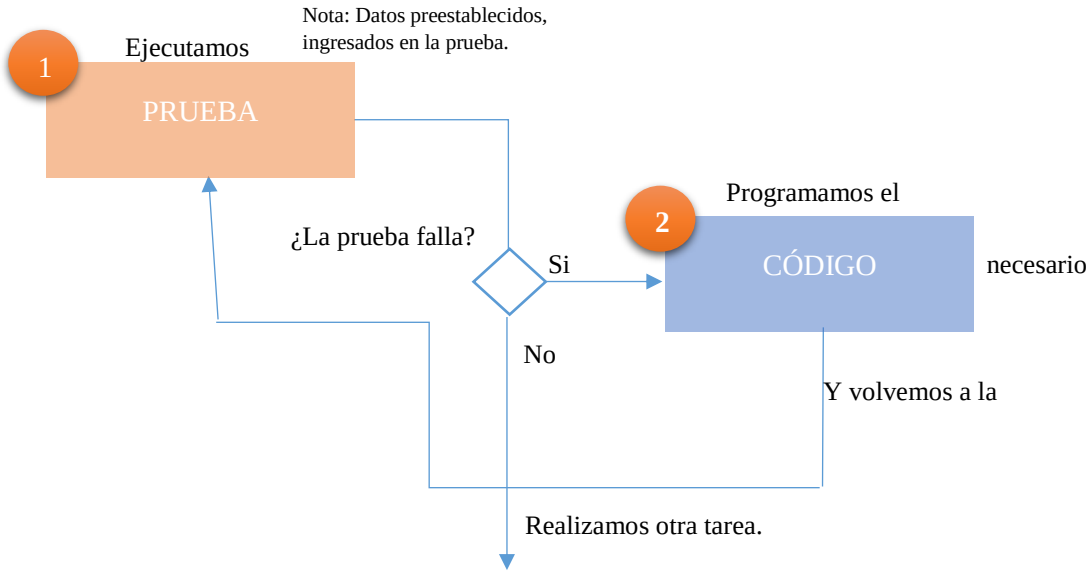
2.1.6. Estructura Invertida en el desarrollo de software

El método tradicional para crear software dentro de la implementación es:

1. Programar el código principal que ejecutara el método deseado.
2. Ejecutar la prueba, hasta que resulte ser lo que se busca.



Pero nos referimos a estructura invertida cuando en vez de primero programar el código principal, primero programamos la prueba que se ha de ejecutar.



Elaboración Propia

2.1.7. Scrum y TDD

Según (Carlos Blé Jurado y Colaboradores, 2010) indica lo siguiente:

"*…la Programación Extrema (eXtreme Programming, XP) como forma de atacar la implementación del producto y hacia Scrum como forma de gestionar el proyecto…"*(p. 36)

En otro párrafo dice:

"*…podemos gestionar el proyecto con Scrum y codificar con técnicas de XP; concretamente TDD…"*(p. 36)

Las combinaciones de metodologías y técnicas son a mejor parecer del equipo de desarrollo y/o desarrollador, con el fin de crear software y minimizar errores en el proceso de desarrollo.

2.2.ANTECEDENTES DE LA INVESTIGACIÓN

Para la realización del presente trabajo, se ha consultado diversas fuentes de información, en el Registro Nacional de Trabajos de Investigación – RENATI – SUNEDU para consultar tesis y trabajos de universidades privadas y públicas, así como páginas web, las mismas que tienen relación directa con el presente trabajo, dentro de ellas se mencionan las siguientes:

2.2.1. Antecedentes a Nivel Nacional

- **UNIVERSIDAD:** PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

  TITULO: UNA EVALUACIÓN EXPERIMENTAL PARA COMPARAR LA CALIDAD DE UN SOFTWARE APLICANDO O NO TDD DENTRO DEL MODELO CASCADA

  AUTOR: ANTONY MAYKOL GOICOCHEA PUERTAS

  RESUMEN: El presente trabajo de investigación brinda un enfoque general de la aplicación de la técnica Test Driven Development (TDD), o Desarrollo Guiado por Pruebas, dentro de la metodología tradicional con enfoque Cascada, y cómo su implicancia proporciona resultados favorables durante el proceso de implementación y en consecuencia la mejora de la calidad del producto. La investigación se llevó a cabo mediante una evaluación experimental en donde se crearon cuatro (4) grupos de desarrollo, cada uno de ellos estaba conformado por once (11) estudiantes del octavo ciclo de la especialidad de Ingeniería Informática. El experimento consistió en que dos (2) grupos apliquen la técnica de TDD dentro de la metodología Cascada y los otros (2) grupos no la apliquen.

  La inclusión de la técnica TDD se llevó a cabo en las primeras fases del modelo Cascada (Definición de requerimientos y Diseño del sistema) a través de la definición de los Casos de Prueba (Test Cases) y mediante ellos se estableció la línea inicial para el comienzo de la implementación del código fuente del sistema a realizar.

  Mediante la aplicación de este experimento se logró obtener resultados estadísticos iniciales que confirman que la inclusión de la técnica TDD en el proceso de implementación y pruebas unitarias permite identificar una

mayor cantidad de errores, lo cual se ve reflejado al final del proceso en un producto de mayor calidad.

Finalmente, al concluir el proceso de desarrollo del software, se aplicó una encuesta para medir la percepción / intención de uso de los participantes respecto a las técnicas TDD y Cascada.

2.2.2. Antecedentes a Nivel Internacional

- **UNIVERSIDAD:** UNIVERSITAT POLITÈCNICA DE VALÈNCIA
- **TITULO:** UNA EXPERIENCIA APLICANDO TEST-DRIVEN DEVELOPMENT (TDD) USANDO UNA HERRAMIENTA JUNIT

  AUTOR: MIGUEL PEÑARROCHA PLANELLS

  RESUMEN: A lo largo de este TFG, se describe la experiencia obtenida al desarrollar unas pruebas unitarias usando JUnit para una aplicación. Esta aplicación, ha sido desarrollada en una empresa del sector médico. Las pruebas unitarias se han realizado sobre una parte de esta aplicación, los mensajes entre médicos y pacientes. La fiabilidad de este apartado de la aplicación es crítica, pues la medicación de muchas personas depende de los avisos o controles programados, vamos a utilizar la técnica Test-Driven Development TDD sobre esta aplicación para reducir el número de fallos que se puedan ocasionar a la hora de programar.

2.2.3. Paper's de Investigación

Dentro del contexto del trabajo de investigación que se pretende desarrollar, fue necesario realizar consultas a diferentes paper's de investigación publicados en la red entre ellas:

ARTICULO DE: INFORMATION AND SOFTWARE TECHNOLOGY

TITULO: THE EFFECTS OF TEST DRIVEN DEVELOPMENT ON INTERNAL QUALITY, EXTERNAL QUALITY AND PRODUCTIVITY: A SYSTEMATIC REVIEW

AUTOR: WILSON BISSI, ADOLFO GUSTAVO SERRA SECA NETO, MARIA CLAUDIA FIGUEIREDO PEREIRA EMER

RESUMEN: context: test driven development (tdd) is an agile practice that has gained popularity when it was defined as a fundamental part in extreme programming (xp). objective: this study analyzed the conclusions of previously published articles on the efects of tdd on internal and external software quality and productivity, comparing tdd with test last development (tld). method: in this study, a systematic literature review has been conducted considering articles published between 1999 and 2014. results: in about 57% of the analyzed studies, the results were validated through experiments and in 32% of them, validation was performed through a case study. the results of this analysis show that 76% of the studies have identified a significant increase in internal software quality while 88% of the studies identified a meaningful increase in external software quality. there was an increase in productivity in the academic environment, while in the industrial scenario there was a decrease in productivity. overall, about 44% of the studies indicated lower productivity when using tdd compared to tld. conclusion: according to our findings, tdd yields more benefits than tld for internal and external software quality, but it results in lower developer productivity than TLD.

ARTICULO DE: FACULTY OF COMPUTER AND INFORMATION SCIENCE, UNIVERSITY OF LJUBLJANA

TITULO: IMPACT OF TEST-DRIVEN DEVELOPMENT ON PRODUCTIVITY, CODE AND TESTS: A CONTROLLED EXPERIMENT

AUTOR: MATJAZ PANCUR, MOJCA CIGLARIC

RESUMEN: Context: Test-driven development is an approach to software development, where automated tests are written before production code in highly iterative cycles. Test-driven development attracts attention as well as followers in professional environment; however empirical evidence of its superiority regarding its effect on productivity, code and tests compared to test-last development is still fairly limited. Moreover, it is not clear if the supposed benefits come from writing tests before code or maybe from high iterativity/short development cycles.

Objective: This paper describes a family of controlled experiments comparing test-driven development to micro iterative test-last development with emphasis on productivity, code properties (external quality and complexity) and tests (code coverage and fault-finding capabilities).

Method: Subjects were randomly assigned to test-driven and test-last groups. Controlled experiments were conducted for two years, in an academic environment and in different developer contexts (pair programming and individual programming contexts). Number of successfully implemented stories, percentage of successful acceptance tests, McCabe's code complexity, code coverage and mutation score indicator were measured.

Conclusion: According to our findings, the benefits of test-driven development compared to iterative test last development are small and thus in practice relatively unimportant, although effects are positive.

There is an indication of test-driven development endorsing better branch coverage, but effect size is considered small.

ARTICULO DE: UNIVERSITY OF HELSINKI, DEPARTMENT OF COMPUTER SCIENCE

TITULO: EFFECTS OF TEST-DRIVEN DEVELOPMENT: A COMPARATIVE ANALYSIS OF EMPIRICAL STUDIES

AUTOR: SIMO MÄKINEN, JÜRGEN MÜNCH

RESUMEN: Test-driven development is a software development practice where small sections of test code are used to direct the development of program units. Writing test code prior to the production code promises several positive effects on the development process itself and on associated products and processes as well. However, there are few comparative studies on the effects of test-driven development. Thus, it is difficult to assess the potential process and product effects when applying test driven

development. In order to get an overview of the observed effects of

test-driven development, an in-depth review of existing empirical studies was carried out. The results for ten different internal and external quality attributes indicate that test-driven development can reduce the amount of introduced defects and lead to more maintainable code. Parts of the implemented code may also be somewhat smaller in size and complexity.

While maintenance of test-driven code can take less time, initial development may last longer. Besides the comparative analysis, this article sketches related work and gives an outlook on future research.

CAPÍTULO III

METODOLOGÍA

Para medir la factibilidad del uso de la técnica de diseño Test Driven Development, se hará uso de la encuesta (Ver Tabla 2) una vez realizada la presentación del módulo al usuario final.

Con el fin de recabar información que nos dé alcances cuantificables sobre el estudio que se pretende probar en este trabajo de investigación basado en el caso de prueba experimental antes mencionado.

Modulo[8]

Puntaje[9]

Pregunta	Si	No	En Parte
El módulo, cumple con las funciones básicas que usted necesita para el desarrollo de sus actividades diarias.			
El módulo, es amigable y entendible para los fines que usted desempeña.			
El módulo, no genera errores de funcionamiento cuando realiza alguna tarea.			
Cuando usted solicitó algún cambio dentro del módulo*, se cumplió con el lapso de tiempo establecido por ambas partes.			
Su participación dentro del desarrollo del módulo* ha sido de forma constante.			
Total:			

Tabla 2: Encuesta de Satisfacción del usuario final

Fuente: Elaboración Propia

[8] Modulo: segmento del software a evaluar.
[9] Puntaje: cada casillero marcado equivale a un punto, para generar igualdad de respuestas.

El valor de las respuestas servirá para comprobar el grado de satisfacción o insatisfacción del usuario final. Al término de la encuesta, se procederá a ingresar la cantidad de "Si", de "No" y de "En Parte" de las respuestas obtenidas para realizar la sumatoria. Los datos de cada encuesta física se consolidarán en la tabla resumen adjunta.

Código Historia de Usuario		
Usuarios encuestados		
Nro. de encuesta		
	Cantidad (#)	Porcentaje (%)
Si		
No		
En parte		
Total:		

3.1.POBLACIÓN

La población encuestada para el caso en estudio, serán los usuarios finales de la institución. Cada persona encuesta, se encuentra en un grupo discriminado de usuario dentro del sistema.

Tipo de usuario	**Nro. de personas**
Administrador del sistema	1 persona
Operador de caja	1 persona
Analista de Créditos	4 personas
Usuarios externos	3 personas

Tabla 3: Población determinada

Fuente: Elaboración Propia

3.2.MUESTRA

Para este trabajo de investigación, se determinó la muestra estratificada; porque la población actual esta subdividida en grupos de usuarios. Por lo que se ha de tener **un encuestado por cada tipo de usuario**, teniendo como **muestra un total de 4 personas**, uno por cada tipo de usuario final, de los cuales determinaran su grado de satisfacción por cada módulo desarrollado del nuevo software.

3.3.FORMULA PARA LA RECOLECCIÓN DE DATOS

Al final de cada iteración se recopilará datos a través de cada encuesta (Ver Tabla 2), la cual brindará un alcance estadístico para nuestro análisis; por lo que se ha de usar las siguientes formulas:

Fórmula para las respuestas (Si, No, En Parte) obtenidas de cada encuesta:

$$\left(\left(\frac{(\sum \boldsymbol{Si})}{\mathbf{0.05}}\right)\%\right)$$

$$\left(\left(\frac{(\sum \boldsymbol{No})}{\mathbf{0.05}}\right)\%\right)$$

$$\left(\left(\frac{(\sum \boldsymbol{En\ Parte})}{\mathbf{0.05}}\right)\%\right)$$

Fórmula para hallar el grado de satisfacción del total de encuestas, aplicado para cada respuesta (Si, No, en Parte):

$$\left(\left(\frac{(\sum \boldsymbol{Total\ de\ "Si"\ de\ todas\ las\ encuestas\ realizadas})}{((\mathbf{0.05}) * \boldsymbol{Número\ total\ de\ encuestas\ realizadas})}\right)\%\right)$$

$$\left(\left(\frac{(\sum \boldsymbol{Total\ de\ "No"\ de\ todas\ las\ encuestas\ realizadas})}{((\mathbf{0.05}) * \boldsymbol{Número\ total\ de\ encuestas\ realizadas})}\right)\%\right)$$

$$\left(\left(\frac{(\sum \boldsymbol{Total\ de\ "En\ Parte"\ de\ todas\ las\ encuestas\ realizadas})}{((\mathbf{0.05}) * \boldsymbol{Número\ total\ de\ encuestas\ realizadas})}\right)\%\right)$$

CAPÍTULO IV

DESARROLLO DEL SOFTWARE

Para el desarrollo de este trabajo de investigación, se ha de basar en la metodología de desarrollo de software ágil Scrum, mi persona como único ejecutor en los roles de Scrum y como cliente, los intervinientes[10] por parte de la cooperativa de ahorro y crédito El Amauta Ltda.

La técnica de diseño Test Driven Development permitirá el desarrollo del software partiendo de las pruebas de ejecución del sistema, hacia la codificación necesaria del proceso. Hasta lograr superar el test de pruebas con éxito para el correcto funcionamiento del cada objetivo de la historia de usuario.

4.1.HISTORIAS DE USUARIO

Las historias de usuario brindaran un alcance aproximado de la prioridad de creación del módulo a implementar; este alcance aproximado lo define el Product Owner basado en los requerimientos mencionados por el administrador del negocio.

* Identificando los requerimientos del usuario final del sistema, estableciendo que la prioridad más importante es la menor unidad numérica.

Código Historia de Usuario	Requerimiento	Prioridad
HU01	El administrador necesita un módulo de registro, edición de datos, cambio de contraseña y habilitación o deshabilitación de usuarios del software.	1
HU02	El operador de caja, el analista de créditos y el administrador, necesitan modulo para registrar y editar socios dentro del software.	2
HU03	El administrador necesita un módulo de configuración para gestionar datos generales del software (Registro de operaciones	3

[10] Adjetivo que interviene

	– operador de caja, registro de actividades – analista de créditos, categoría de publicación - administrador).	
HU04	El operador de caja necesita un módulo para registrar sus operaciones realizadas durante el día.	4
HU05	El analista de créditos necesita un módulo para registrar sus actividades realizas durante el día.	5
HU06	El analista de créditos necesita información del registro del operador de caja, para programar sus labores cobro y otros durante el día.	6
HU07	El operador de caja necesita información diaria del total de ingreso y salida de dinero, para realizar su arqueo de caja correspondiente al cierre del día.	7
HU08	El administrador necesita información de las actividades realizadas durante el día de cada analista de crédito, para el seguimiento preciso de sus labores programadas.	8
HU09	El administrador necesita información consolidada de las actividades del analista de crédito y operador de caja, para la toma de decisiones sobre el rendimiento de los colaboradores y futuros cambios.	9
HU10	El administrador necesita un módulo donde se pueda subir archivos y crear exámenes para el control de aprendizaje de sus colaboradores, de tal forma controlar el estado de aprendizaje de las capacitaciones brindadas por la institución.	10
HU11	El operador de caja, los analistas de créditos necesitan un módulo donde los usuarios puedan visualizar sus labores realizas.	11
HU12	El operador de caja, los analistas de créditos, administrador, usuario externo necesitan un módulo donde puedan visualizar documentación entregada por el administrador.	12

HU13	El operador de caja, los analistas de créditos, administrador, usuario externo necesitan un módulo donde puedan rendir sus exámenes asignados para cumplir con las políticas de la institución.	13

Tabla 4: Historias de Usuarios

Fuente: Elaboración Propia

Es necesario explicar, que solo se está plasmando en este documento casos de uso, diagrama de clases y diagrama de base de datos para tener un mejor entendimiento de que es lo que se pretende crear. Por qué este trabajo busca experimentar la creación de software a partir de contextos creados pero pocas veces combinados entre sí.

4.2. CASOS DE USO

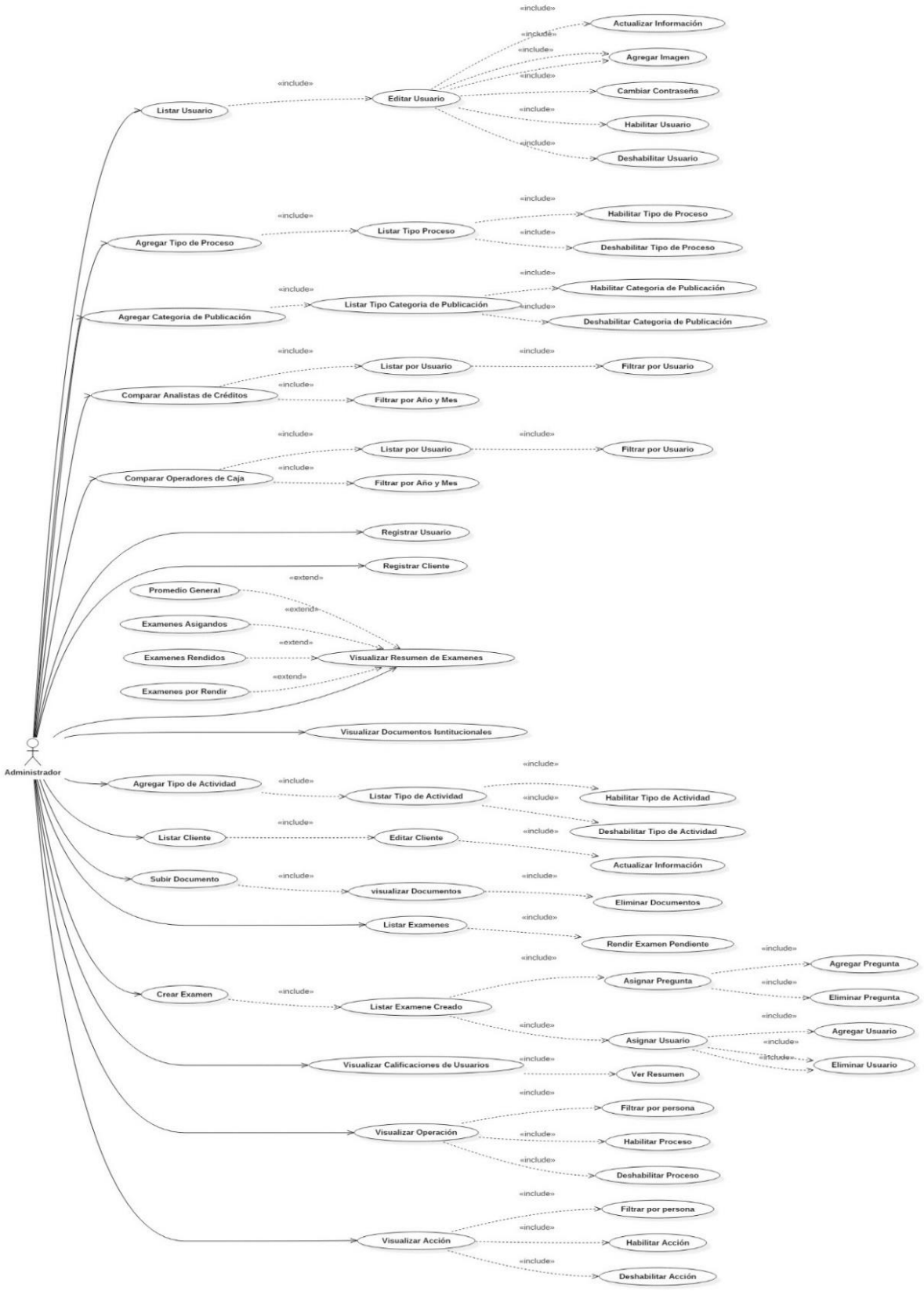


Imagen 3: Caso de Uso del Administrador

Elaboración Propia

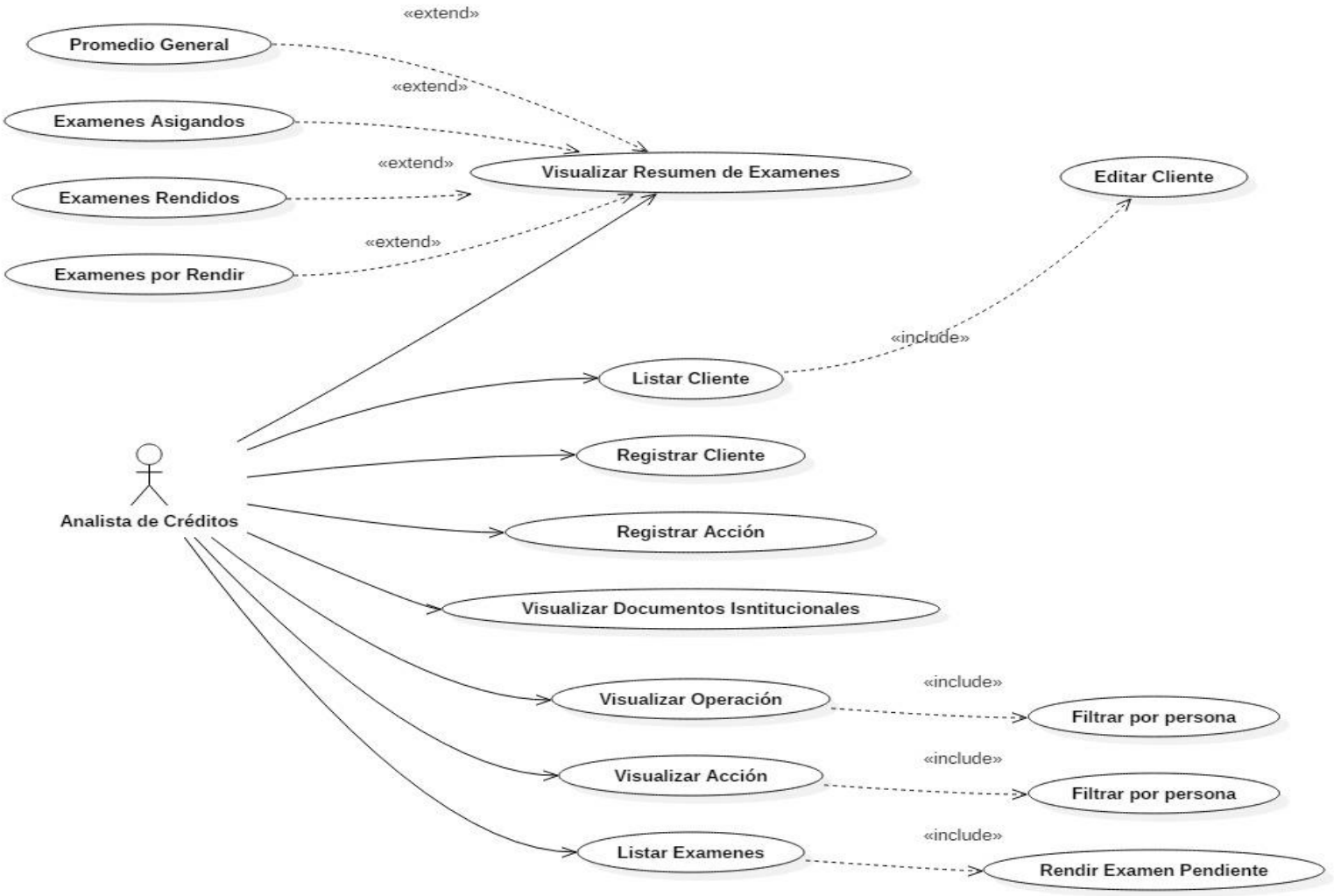


Imagen 4: Caso de Uso del Analista de Créditos

Elaboración Propia

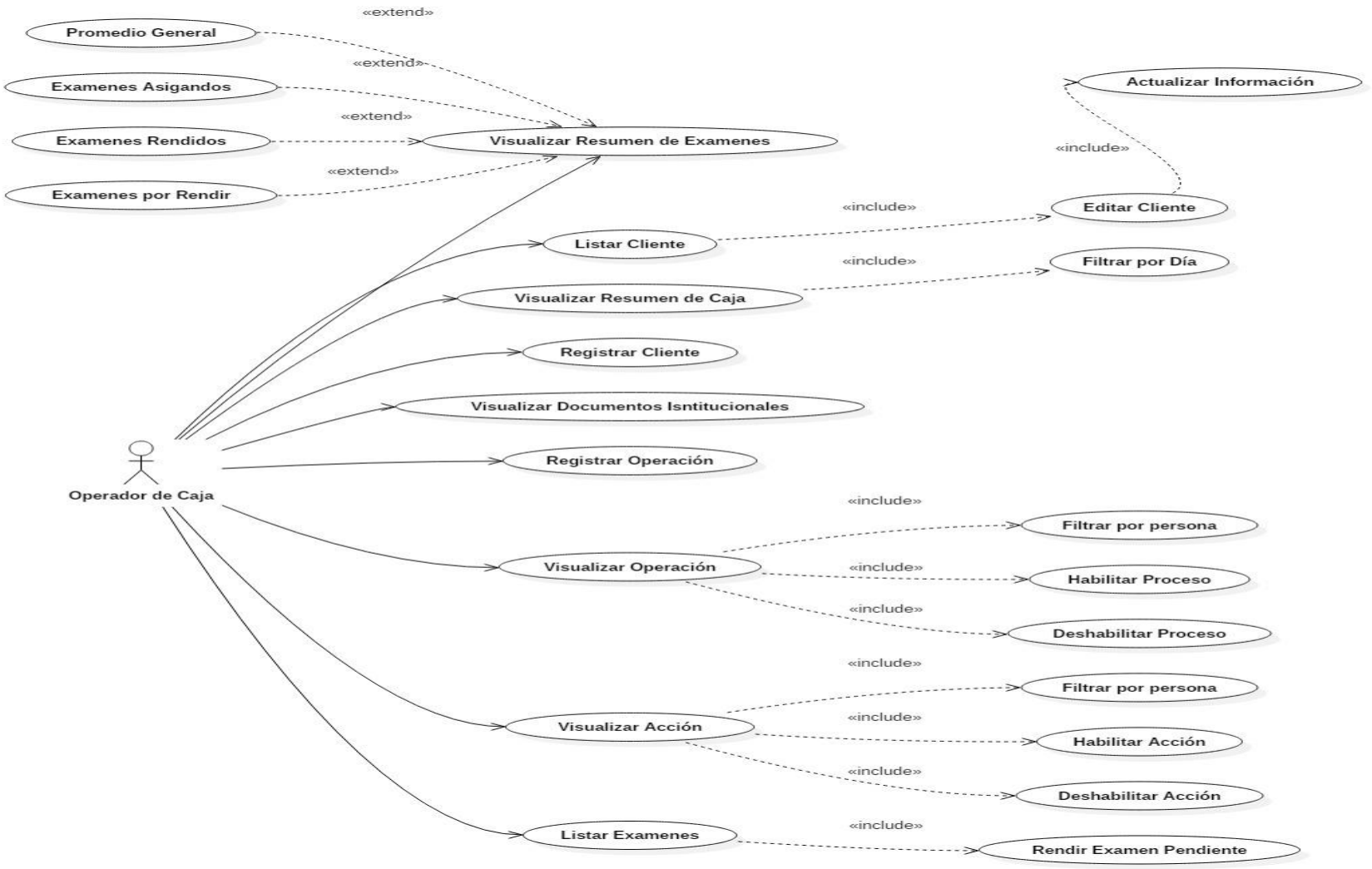


Imagen 5: Caso de Uso del Operador de Caja

Elaboración Propia

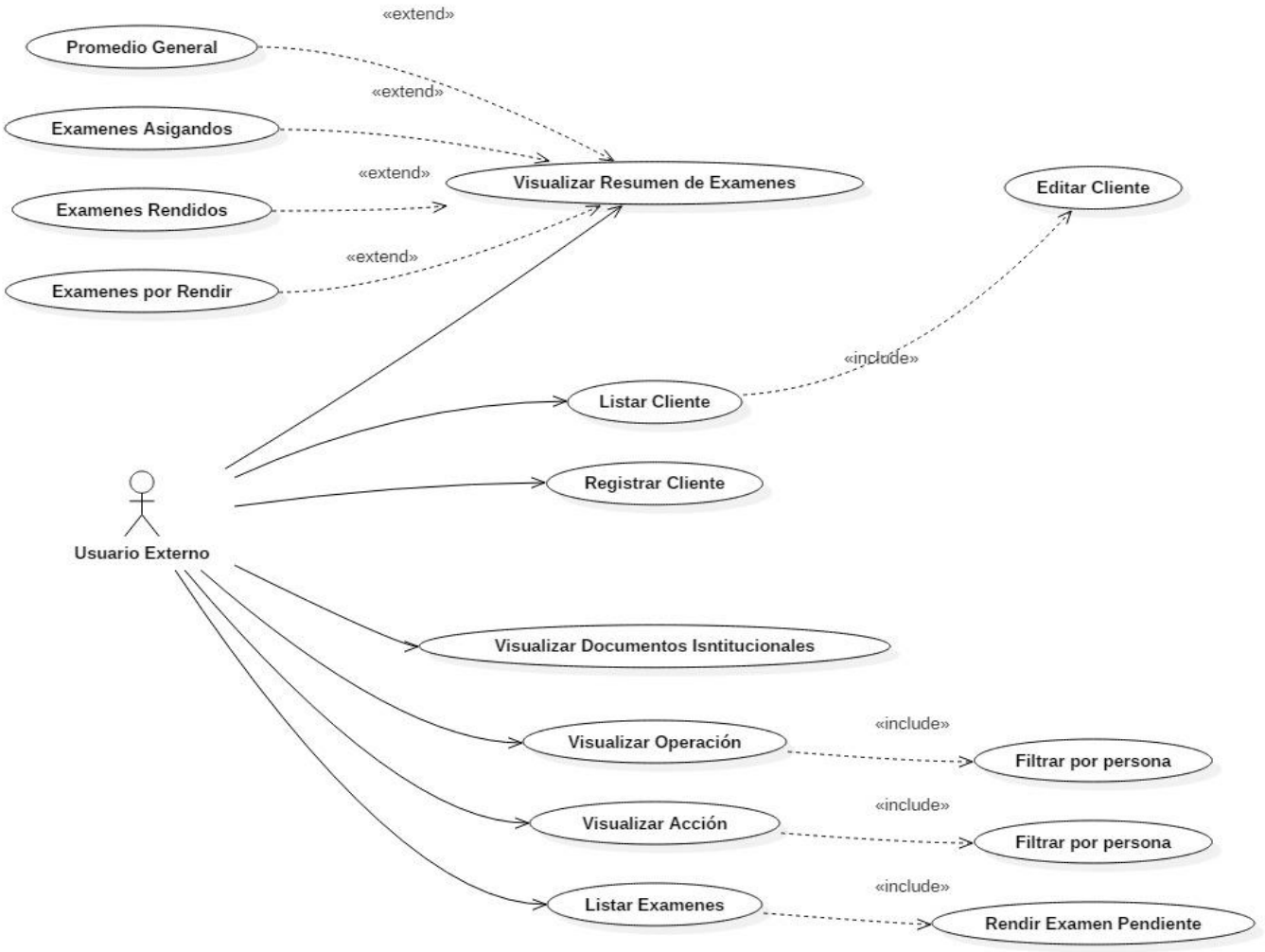


Imagen 6: Caso de Uso del Usuario Externo

Elaboración Propia

4.3.DIAGRAMA DE CLASES

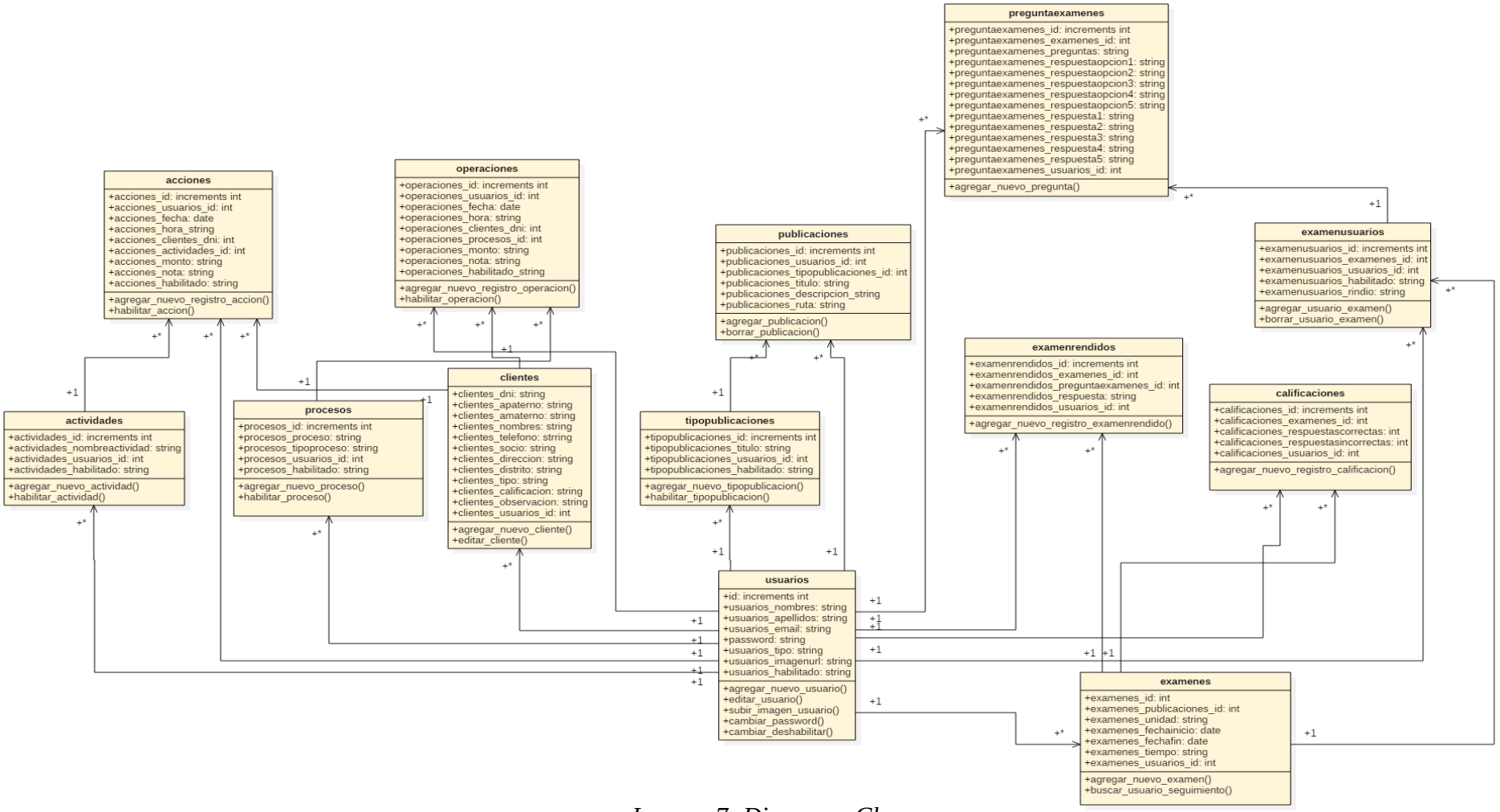


Imagen 7: Diagrama Clases

Elaboración Propia

4.4.DIAGRAMA DE BASE DE DATOS

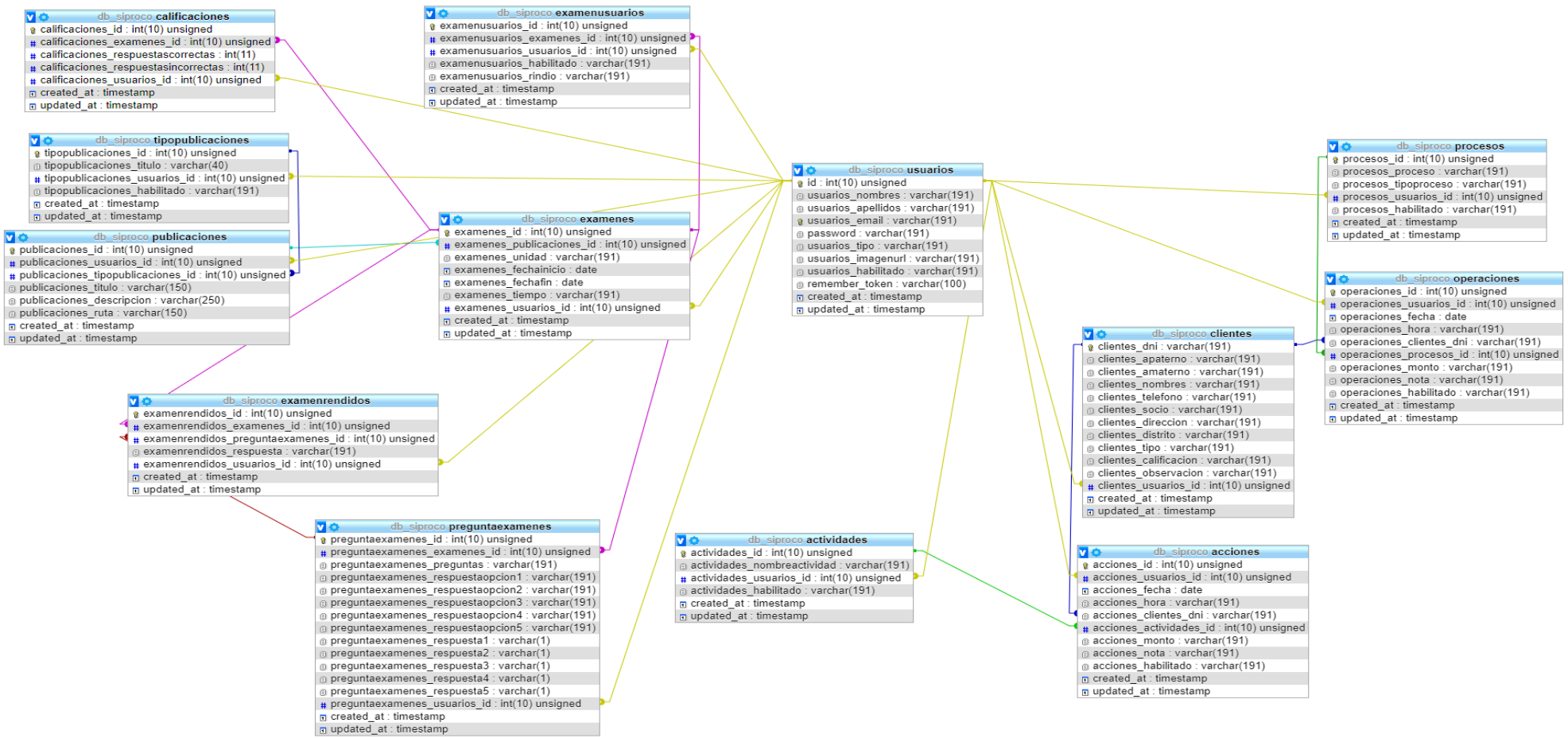


Imagen 8: Diagrama de Base de Datos

Elaboración Propia

4.5.ASPECTOS GENERALES PARA EL INICIO PRUEBAS DE TDD

Para hacer uso de la técnica de diseño Test Driven Development en Laravel, se emplea la librería phpunit que viene incorporada, para lo cual se ha de usar comandos artisan dentro de la consola cmd de Windows como:

Para la instalación del proyecto en Laravel, se escribe la siguiente línea para descargar el proyecto base del repositorio de Laravel dentro de la consola cmd de Windows.

composer create-project --prefer-dist laravel/laravel siproco_tdd "5.5."

Se escoge la versión de Laravel 5.5 porque es la que cuenta más documentación hasta el momento.

Adicionalmente se tiene los comandos necesarios para la ejecución y creación de archivos según se va creando el software.

*php artisan make:model NombreArchivo

Comando con el cual se crea los *modelos* de cada clase dentro del Proyecto.

*php artisan make:migration Tabla_Nombres

Comando con el cual se crea cada *tabla* de la base de datos, antes de ser migrada.

*php artisan migrate

Comando que migra las tablas de base de datos diseñadas en el proyecto.

*php artisan make:controller NombreArchivoController

Comando para crear el *Controlador* dentro del Proyecto. El Controlador servirá para escribir el método de cada función necesaria

*php artisan make:test NombreArchivoTest

Comando para crear el archivo – *test*, que contendrá las pruebas unitarias

Condición en las pruebas unitarias.

Si se recibe la respuesta OK después de ejecutar el test de pruebas, se entiende como prueba superada.

Si se recibe la respuesta FAILURE después de ejecutar el test de pruebas, se entiende como prueba fallida.

Por lo cual hay que identificar el problema que ocasiona la falla y adicionar código o modificarlo hasta que la prueba pase.

4.5.1. CONFIGURACION DE LA BASE DE DATOS

Previamente se crea el nombre de la base de datos en MYSQL y para obtener un acceso más rápido a los datos del test de pruebas, se redirige el almacenamiento de datos directamente en memoria previa configuración en phpunit.

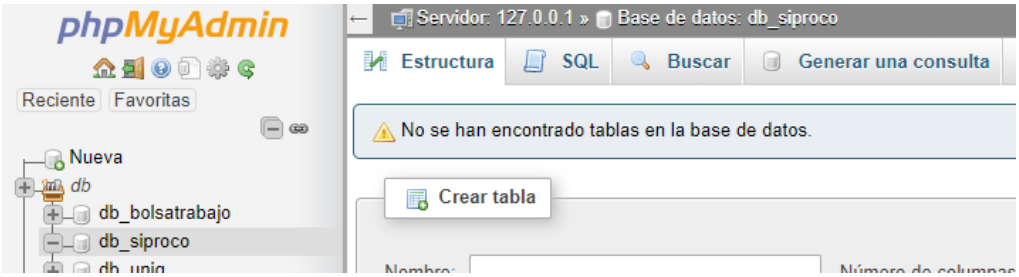


Imagen 9: Creación de la Base de Datos MYSQL

Elaboración propia

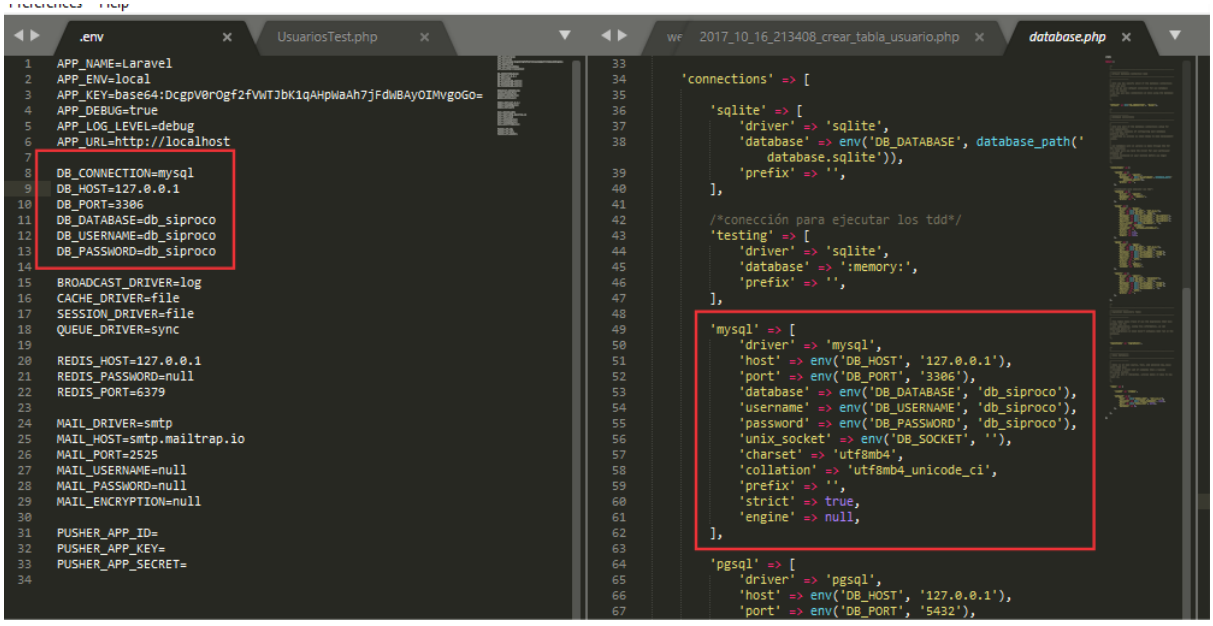


Imagen 10: Configurar la conexión con la base de datos

Elaboración propia

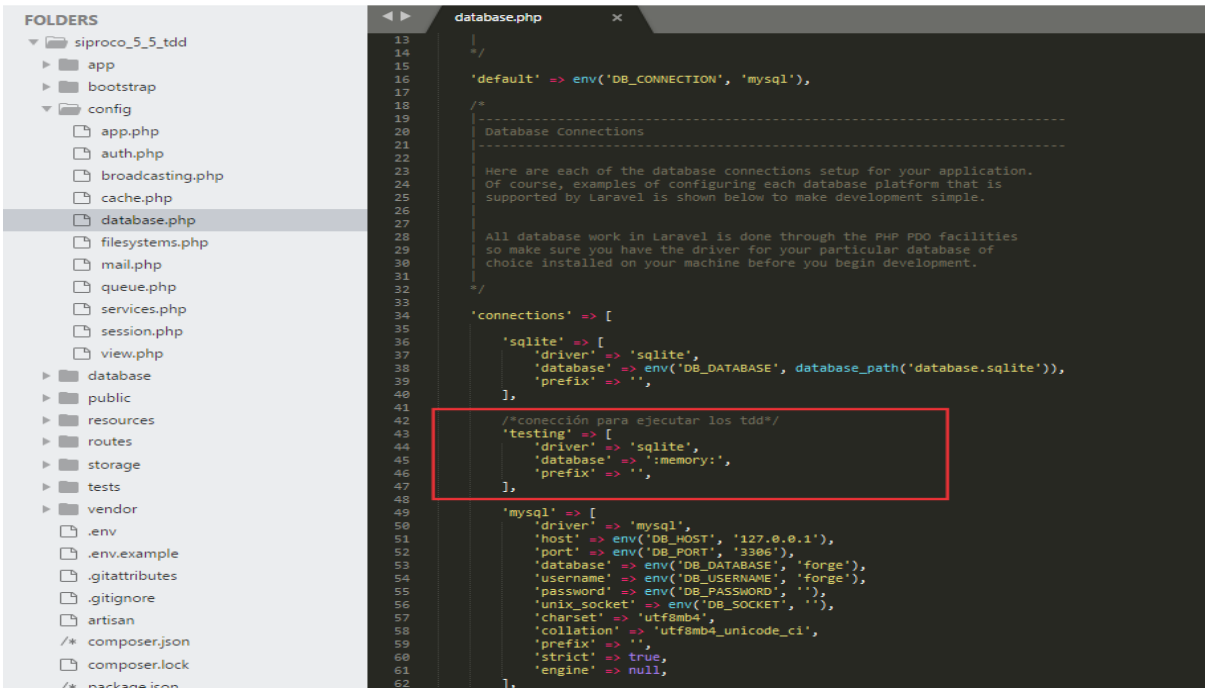


Imagen 11: Configurar Base de Datos en Memoria

Elaboración propia

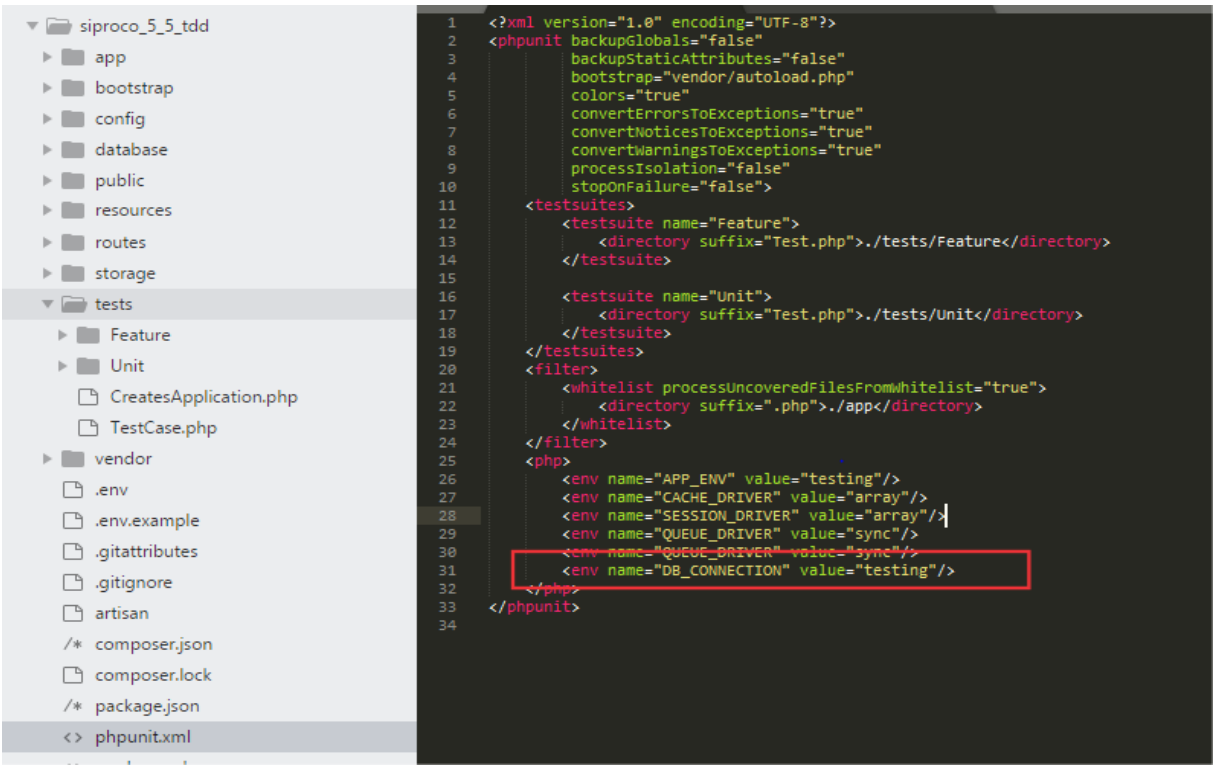


Imagen 12: Configurar la Dirección de la Base de Datos en PHPUnit

Elaboración propia

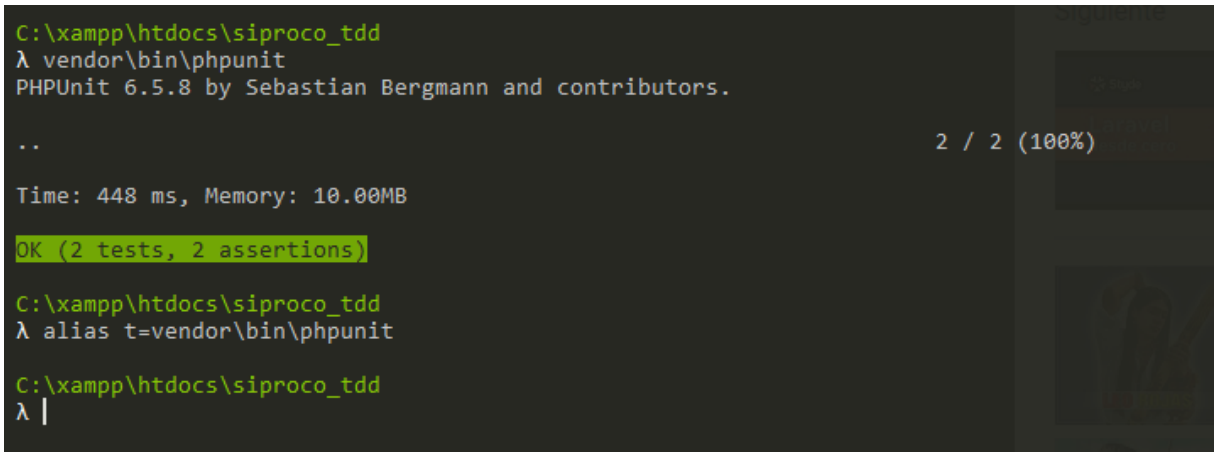


Imagen 13: Configurar Alias a "vendor\bin\phpunit"

Elaboración propia

Se creo la base de datos y se configuro los parámetros de acceso en Laravel con normalidad y sin inconvenientes.

4.6.DESARROLLO DE LA SOLUCION ENFOCADO EN LAS PRUEBAS UNITARIAS

El product backlog es el detalle de: el cómo, el quiero, el para y las condiciones requeridas de cada historia de usuario recogida por el Product Owner en función de las especificaciones del administrador de la institución.

Para demostrar el uso de la técnica de diseño Test Driven Development se desarrollará, todos los test de pruebas.

Las pruebas realizadas están enfocadas en el objetivo principal del requerimiento.

Recordemos que estamos usando Laravel, por lo cual estamos bajo una estructura de framework Modelo Vista Controlador (Ver imagen MVC).

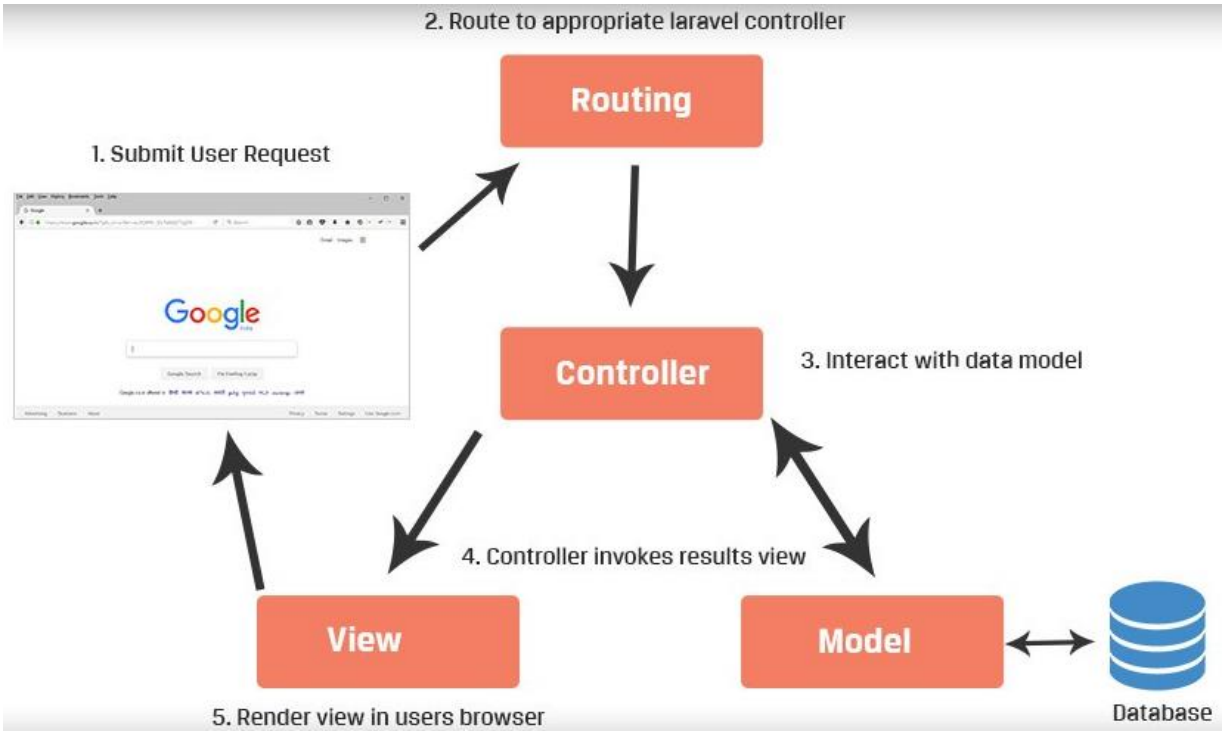


Imagen 14: how controllers work in mvc Laravel

Elaboración: https://medium.com/@dannyhuang_75970/learning-laravel-controllers-101-ad28d2bb5569

4.6.1. Product Backlog – HU01

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU01		**El administrador necesita un módulo de registro, edición de datos, cambio de contraseña y habilitación o deshabilitación de usuarios del software.**	**1**
	HU01-01	Como administrador del sistema, quiero tener el control del registro de usuarios, para tener el control absoluto de los usuarios que deban ingresar al sistema. Condiciones: - El módulo debe registrar a los siguientes roles de	1.1

		usuarios, operador de caja, analista de créditos, administrador y usuario externo.	
	HU01-02	Como administrador del sistema, quiero tener el control de los usuarios registrados, para editar de los usuarios, cambiar contraseña, agregar imagen y habilitar o deshabilitar usuarios. Condiciones: - Debe estar todo en una sola vista.	1.2

Tabla 5: Product Backlog - HU01

Fuente: Elaboración Propia

*Crear el archivo que contendrá los test de pruebas del usuario.

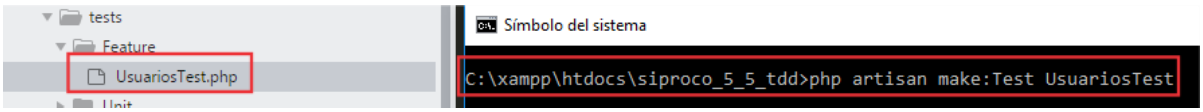


Imagen 15: Creación del Archivo UsuariosTest

Fuente: Elaboración Propia

Imagen 16: Tabla Usuarios desde Laravel

Fuente: Elaboración Propia

Primero, se crea la tabla “Usuarios” en laravel conteniendo los campos definidos en el diagrama de base de datos ya definidos, y se ejecuta la migración de la tabla al gestor de base de datos.

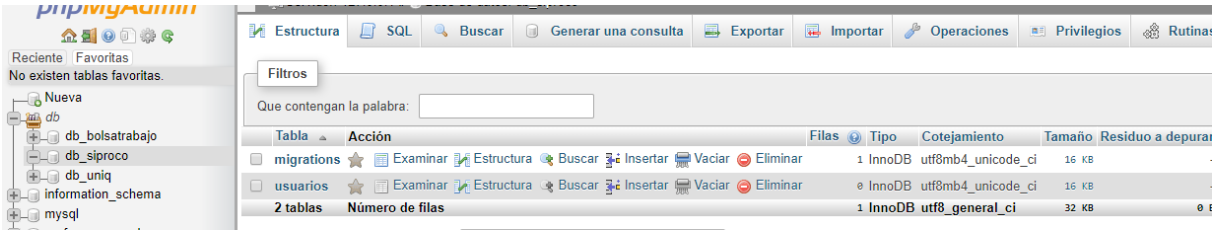


Imagen 17: Tabla Usuarios en MySQL

Fuente: Elaboración Propia

Se puede apreciar que la tabla Usuarios ha sido creada en MySQL desde la ejecución de migración hecha en Laravel.

4.6.1.1.Test de Prueba del Product Backlog HU01-01 – Registro de Usuario

Resultado esperado:

<table>
<tr><td colspan="2">Nombre de proceso:</td><td colspan="2">Registrar Usuario</td><td>Id caso de prueba:</td><td>HU01-01-01</td></tr>
<tr><td colspan="4"></td><td>Id backlog:</td><td>HU01-01</td></tr>
<tr><td colspan="4"></td><td>Id historia de usuario:</td><td>HU01</td></tr>
<tr><td colspan="2">Autor de caso de prueba:</td><td colspan="4">Administrador del sistema</td></tr>
<tr><td colspan="3">Método</td><td colspan="3">Especificación</td></tr>
<tr><td colspan="3">test_Registrar_usuario(ruta, string usuarios_nombres,
string usuarios_apellidos,
string usuarios_email,
string usuarios_tipo,
string password, submit)</td><td colspan="3">Insertar los valores requeridos por el método para el registro de un usuario.</td></tr>
<tr><td colspan="6">Descripción de las acciones y/o condiciones para las pruebas</td></tr>
<tr><td>#</td><td>Método</td><td>Entrada</td><td colspan="3">Salida Esperada</td></tr>
<tr><td>01</td><td>test_Registrar_usuario()</td><td>form_registrar_usuario,
Redy, Delgado,
rdelgado@correo.com,
administrador, 123,
guardar_usuario</td><td colspan="3">Usuario registrado correctamente.</td></tr>
<tr><td colspan="2">Resultado Obtenido:</td><td colspan="4">Usuario registrado correctamente.</td></tr>
<tr><td colspan="2">Dato adicional:</td><td colspan="4">Ninguno</td></tr>
</table>

Se crea el test de pruebas *test_registrar_usuario*(Registrar Usuario), donde le especificaremos al test de pruebas los datos que debe ejecutar:

1. Que se dirija a la vista del formulario *form_registrar_usuario*.
2. Seguidamente de la inserción de datos como: nombres, apellidos, email, contraseña, tipo de usuario en cada campo.
3. Para terminar la prueba, el sistema deberá presionar el botón *guardar_usuario*.

```
class UsuariosTest extends TestCase
{

    use RefreshDatabase;

    public function test_ir_form_registrar_usuario()
    {
        $this->visit('form_registrar_usuario');
    }

    public function test_registrar_usuario()
    {
        $this->visit('form_registrar_usuario')
             ->type('Redy','usuarios_nombres')
             ->type('Delgado','usuarios_apellidos')
             ->type('rdelgado@correo.com','usuarios_email')
             ->type('administrador','usuarios_tipo')
             ->type('123','password')
             ->press('guardar_usuario');
    }
```

```
λ Cmder

Tests\Feature\UsuariosTest))
#32 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpunit\phpunit\src\F
HPUnit\Framework\TestResult))
#33 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpunit\phpunit\src\F
PHPUnit\Framework\TestResult))
#34 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpunit\phpunit\src\F
PHPUnit\Framework\TestResult))
#35 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpunit\phpunit\src\T
PUnit\Framework\TestResult))
#36 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpunit\phpunit\src\T
it\Framework\TestSuite), Array, true)
#37 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpunit\phpunit\src\T
#38 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpunit\phpunit\phpun
#39 {main}
FAILURES!
Tests: 3, Assertions: 3, Failures: 2.

C:\xampp\htdocs\siproco_5_5_tdd
```

Imagen 18: Test de pruebas HU01-01 - FAILURE

Fuente: Elaboración Propia

El test de pruebas de la HU01-01 falla, porque la prueba está buscando la vista *form_registrar_usuario* el cual no existe.

Se crea la ruta *form_registrar_usuario* el cual direccionan al controlador *UsuariosController@form_registrar_usuario*. Para mostrarnos la primera vista

Se ha creado la ruta *agregar_nuevo_usuario* el cual direccionan al controlador *UsuariosController@agregar_nuevo_usuario*. Para ejecutar el método *agregar_nuevo_usuario.*

```
15
16
17  /*Rutas para agregar nuevo usuario*/
18  Route::get('form_registrar_usuario', 'UsuariosController@form_registrar_usuario');
19  Route::post('agregar_nuevo_usuario', 'UsuariosController@agregar_nuevo_usuario');
20
21
22
```

Imagen 19: Ruta del formulario para agregar usuario
Fuente: Elaboración Propia

Creamos el método *agregar_nuevo_usuario*, que permitirá ejecutar el registro del nuevo usuario.

UsuariosController.php

```php
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\Validator;
use Storage;

class UsuariosController extends Controller
{
    public function form_registrar_usuario()
    {
        return view("intranet.formularios.form_registrar_usuario");
    }

    //Metodo para agregar nuevo usuario al sistema.
    public function agregar_nuevo_usuario(Request $request)
    {

        $data = $request->all();

        $reglas = array('usuarios_nombres' => 'required|Alpha',
            'usuarios_apellidos'            => 'required|Alpha',
            'usuarios_email'                => 'required',
            'usuarios_tipo'                 => 'required|Alpha',
            'password'                      => 'required|min:3|max:16',
        );
        $mensajes = array('usuarios_nombres.required' => 'Ingresar los nombres del usuario es obligatorio',
            'usuarios_nombres.alpha'                  => 'El campo de contiene nombres no puede contener números',
            'usuarios_apellidos.required'             => 'Ingresar los apellidos del usuario es obligatorio',
            'usuarios_apellidos.alpha'                => 'El campo de apellidos no puede contener números',
            'usuarios_email.required'                 => 'Ingresar el email insitucional es obligatorio',
            'usuarios_email.email'                    => 'El email debe tener un formato valido ejemplo
                correoinstitucional@elamauta.com.pe',
            'usuarios_tipo.required'                  => 'Ingresar el tipo de usuario es obligatorio',
            'usuarios_tipo.alpha'                     => 'Seleccione un tipo de usuario',
            'password.required'                       => 'La contraseña debe de contener un minio de 3 digitos y un
                maximo de 16 digitos',
        );

        $validacion = Validator::make($data, $reglas, $mensajes);
        if ($validacion->fails()) {
            $errores = $validacion->errors();
            return new JsonResponse($errores, 422);
        }

        $usuario                    = new Usuario;
        $usuario->usuarios_nombres  = $data["usuarios_nombres"];
        $usuario->usuarios_apellidos= $data["usuarios_apellidos"];
        $usuario->usuarios_email    = $data["usuarios_email"];
        $usuario->usuarios_tipo     = $data["usuarios_tipo"];
        $usuario->password          = bcrypt($data["password"]);
        $usuario->usuarios_imagenurl= '';
        $usuario->usuarios_habilitado = 's';
        $usuario->remember_token    = str_random(10);

        $agregar_nuevo_usuario = $usuario->save();
        if ($agregar_nuevo_usuario) {

            return view("intranet.mensajes.msj_correcto")->with("msj", "Usuario registrado correctamente");
        } else {
            return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
        }
    }
```

Imagen 20: Método dentro del controlador agregar usuario

Fuente: Elaboración Propia

Dentro del Controlador *UsuariosControler* en el método *agregar_nuevo_usuario*, se ha escrito el código para almacenar los datos ingresados en la vista *form_registrar_usuario*, y tambien validaciones correspondientes para evitar inconsistencias en la base de datos para el módulo HU01-01 – Agregar Usuario.

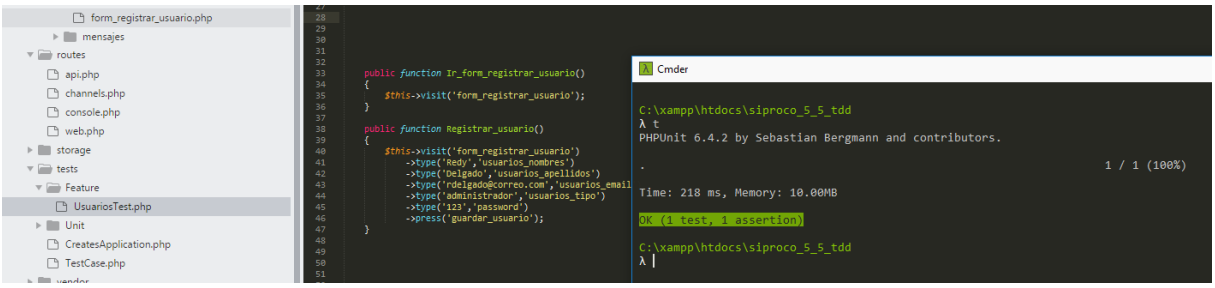


Imagen 21: Test de pruebas HU01-01 - OK

Fuente: Elaboración Propia

Se observa que el test de pruebas, para el registro de un nuevo usuario se ha realizado con éxito, desde ya el sistema puede recibirnos ingresos mecánicos (ingreso de datos de forma manual) de datos para registrar usuarios directamente y todo esto conseguido a partir de las pruebas en una primera instancia, sin necesidad de abrir la ventana del navegador.

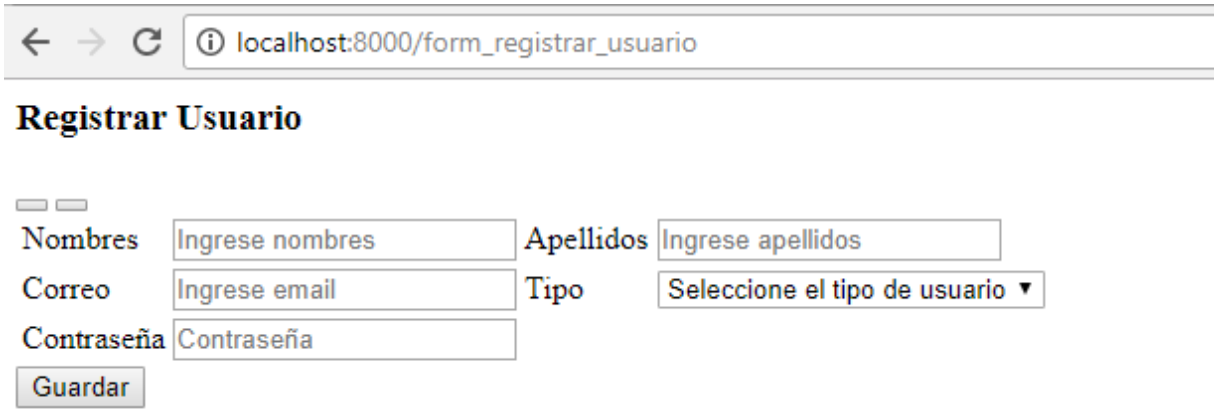


Imagen 22: HU01-01 - Formulario Registrar Usuario - TDD

Fuente: Elaboración Propia

En la imagen 22, se aprecia el formulario resultante de solo escribir las pruebas primero y ejecutar el test.

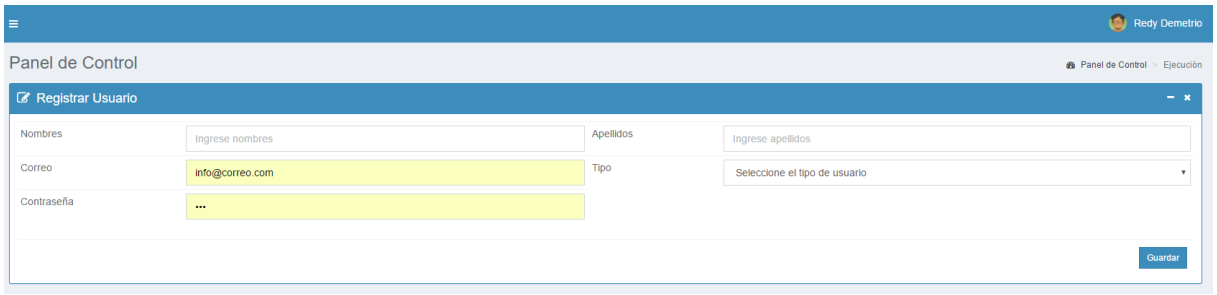


Imagen 23: HU01-01 - Formulario Registrar Usuario

Fuente: Elaboración Propia

En la Imagen 23, se puede ver la vista terminada del formulario Registrar Usuario.

4.6.1.2.Test de Prueba del Product Backlog HU01-02 –Editar Usuario

Dentro del Product Backlog HU01-02, tendremos sub módulos:

1. Cambiar datos del usuario.
2. Cambiar contraseña del usuario.
3. Subir imagen del usuario.

1. Cambiar datos del usuario.

Resultado esperado:

<table>
<tr><td colspan="2" rowspan="3">Nombre de proceso:</td><td colspan="2" rowspan="3">Editar usuario – Actualizar información</td><td>Id caso de prueba:</td><td>HU01-02-01</td></tr>
<tr><td>Id backlog:</td><td>HU01-02</td></tr>
<tr><td>Id historia de usuario:</td><td>HU01</td></tr>
<tr><td colspan="2">Autor de caso de prueba:</td><td colspan="3">Administrador del sistema</td></tr>
<tr><td colspan="3">Método</td><td colspan="3">Especificación</td></tr>
<tr><td colspan="3">test_Editar_usuario(ruta, int id,
string usuarios_nombres,
string usuarios_apellidos,
string usuarios_email, submit)</td><td colspan="3">Insertar los valores requeridos por el método para actualizar los datos del usuario.</td></tr>
<tr><td colspan="6">Descripción de las acciones y/o condiciones para las pruebas</td></tr>
<tr><td>#</td><td>Método</td><td colspan="2">Entrada</td><td colspan="2">Salida Esperada</td></tr>
<tr><td>01</td><td>test_Editar_usuario()</td><td colspan="2">form_editar_usuario/1,
1,
Redy Editado,
Delgado Editado,
rdelgado@correo.com,
actualizar_usuario</td><td colspan="2">Datos actualizados correctamente.</td></tr>
<tr><td colspan="2">Resultado Obtenido:</td><td colspan="4">Datos actualizados correctamente.</td></tr>
<tr><td colspan="2">Dato adicional:</td><td colspan="4">Ninguno</td></tr>
</table>

Creamos el test de pruebas editar usuario, para lo cual se usa datos precargados del primer test de pruebas (Registrar Usuario); en este test de pruebas ingresamos los siguientes valores:

1. Vamos directamente a la ruta form_editar_usuario añadiendo el id del usuario, en este caso “1”. Por qué el id es autoincrementado una vez realizado el registro del usuario desde el test_registrar_usuario.
2. Modificamos los campos de nombres, apellidos, email.
3. Le indicamos a la prueba, presionar actualizar_usuario.

```
    public function test_registrar_usuario()
    {
        $this->visit('form_registrar_usuario')
            ->type('Redy','usuarios_nombres')
            ->type('Delgado','usuarios_apellidos')
            ->type('rdelgado@correo.com','usuarios_email')
            ->type('administrador','usuarios_tipo')
            ->type('123','password')
            ->press('guardar_usuario');
    }

    public function test_ir_form_editar_usuario()
    {
        $this->visit('form_editar_usuario/1');
    }

    public function test_editar_usuario()
    {
        $this->visit('form_editar_usuario/1')
            ->type('1','usuarios_id')
            ->type('RedyEditado','usuarios_nombres')
            ->type('DelgadoEditado','usuarios_apellidos')
            ->type('rdelgado@correo.com','usuarios_email')
            ->press('actualizar_usuario');
    }
}
```

```
Cmder

Tests\Feature\UsuariosTest))
#32 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpu
HPUnit\Framework\TestResult))
#33 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpu
PHPUnit\Framework\TestResult))
#34 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpu
PHPUnit\Framework\TestResult))
#35 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpu
PUnit\Framework\TestResult))
#36 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpu
it\Framework\TestSuite), Array, true)
#37 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpu
#38 C:\xampp\htdocs\siproco_5_5_tdd\vendor\phpu
#39 {main}
FAILURES!
Tests: 5, Assertions: 5, Failures: 4.

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 24: Test de pruebas HU01-02 – Cambiar Datos - FAILURE

Fuente: Elaboración Propia

En el test de pruebas de la HU01-02 modificar datos falla, debido a que está buscando la ruta de la vista *form_editar_usuario* el cual no existe.

```
16
17  /*Rutas para agregar nuevo usuario*/
18  Route::get('form_registrar_usuario', 'UsuariosController@form_registrar_usuario');
19  Route::post('agregar_nuevo_usuario', 'UsuariosController@agregar_nuevo_usuario');
20  Route::get('form_editar_usuario/{id}', 'UsuariosController@form_editar_usuario');
21  Route::post('form_editar_usuario/editar_usuario', 'UsuariosController@editar_usuario')
        ;
22
23
```

Imagen 25: Ruta del formulario para editar usuario – Cambiar datos

Fuente: Elaboración Propia

Se ha creado la ruta *form_editar_usuario/{id}* el cual direcciona al controlador *UsuariosController@form_editar_usuario*. Para mostrarnos la vista del formulario del usuario seleccionado.

Se ha creado la ruta *form_editar_usuario/editar_usuario* el cual direcciona al controlador *UsuariosController@editar_usuario*. Para ejecutar el método *editar_usuario.*

```php
public function form_editar_usuario($usuarios_id)
{
    $usuario  = Usuario::find($usuarios_id);
    $contador = count($usuario);
    if ($contador > 0) {
        return view("intranet.formularios.form_editar_usuario")->with("usuario", $usuario);
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "El usuario con ese id no existe o fue borrad
            ;
    }
}

//Metodo para editar usuario
 public function editar_usuario(Request $request)
{
    $data = $request->all();

    $reglas = array('usuarios_nombres' => 'required|Alpha',
        'usuarios_apellidos'              => 'required|Alpha',
        'usuarios_email'                  => 'required',
    );
    $mensajes = array('usuarios_nombres.required' => 'Ingresar los nombres del usuario es obligatorio',
        'usuarios_nombres.alpha'                  => 'El campo de nombres no puede contener números',
        'usuarios_apellidos.required'             => 'Ingresar los apellidos del usuario es obligatorio',
        'usuarios_apellidos.alpha'                => 'El campo de apellidos no puede contener números',
        'usuarios_email.required'                 => 'Ingresar el correo insitucional es obligatorio',
        'usuarios_email.email'                    => 'El correo insitucional debe tener un formato valido
            ejemplo correoinstitucional@elamauta.com.pe',
        'email.unique'                            => 'El correo insitucional ya existe en la base de datos',
    );

    $validacion = Validator::make($data, $reglas, $mensajes);
    if ($validacion->fails()) {
        $errores = $validacion->errors();
        return new JsonResponse($errores, 422);
    }

    $usuarios_id = $data["usuarios_id"];
    $usuario     = Usuario::find($usuarios_id);

    $usuario->usuarios_nombres   = $data["usuarios_nombres"];
    $usuario->usuarios_apellidos = $data["usuarios_apellidos"];
    $usuario->usuarios_email     = $data["usuarios_email"];

    $editar_usuario = $usuario->save();
    if ($editar_usuario) {
        return view("intranet.mensajes.msj_correcto")->with("msj", "Datos actualizados correctamente");
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
    }
}
```

Imagen 26: Método dentro del controlador editar usuario – Cambiar datos

Fuente: Elaboración Propia

Se vuelve a ejecutar el test de pruebas para HU01-02

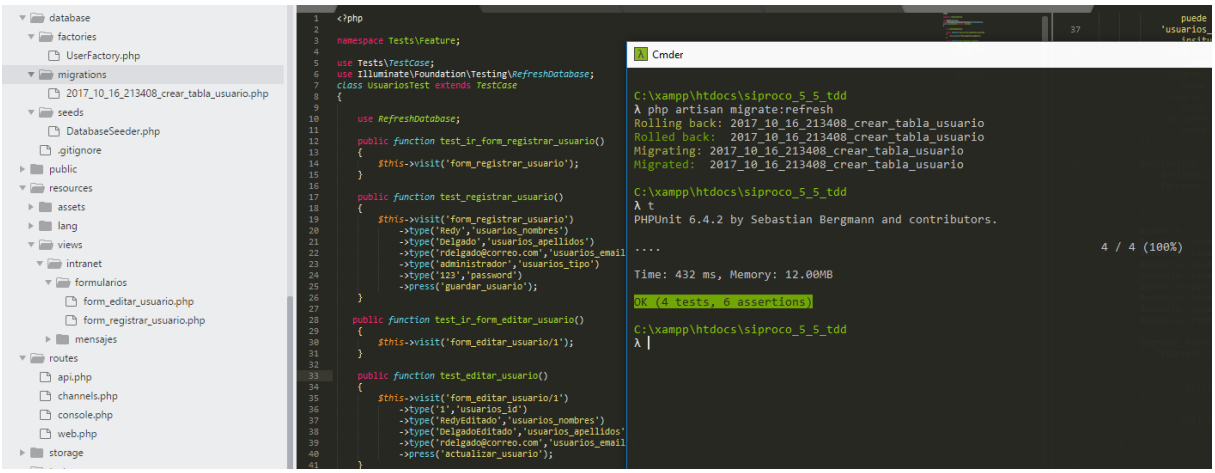


Imagen 27: Test de pruebas - HU01-02 – Cambiar Datos - OK

Fuente: Elaboración Propia

Se observa que ya el cuarto test de pruebas ha sido realizado con éxito.

4. Cambiar contraseña del usuario.

Resultado esperado:

Nombre de proceso:	Editar usuario – Cambiar Contraseña	Id caso de prueba:	HU01-02-02
		Id backlog:	HU01-02
		Id historia de usuario:	HU01
Autor de caso de prueba:	Administrador del sistema		

Método	Especificación
test_Cambiar_contrasena(ruta, int id_usuarios_password, string usuarios_email, string usuarios_password, submit)	Insertar los valores requeridos por el test para cambiar la contraseña del usuario.

Descripción de las acciones y/o condiciones para las pruebas

#	Método	Entrada	Salida Esperada
01	test_Cambiar_contrasena()	form_editar_usuario/1, 1, rdelgado@correo.com, 321, Cambiar_contrasena	Contraseña Actualizada

Resultado Obtenido:	Contraseña Actualizada
Dato adicional:	Ninguno

Creamos el test de pruebas *test_cambiar_password* para cambiar contraseña; para lo cual ingresamos los datos:

1. Indicamos al test dirigirse *al form_editar_usuario/1* es decir, ir directamente a la vista editar usuario y editar el usuario "1". Usuario registrado previamente.
2. Dentro del test de pruebas, le asignamos "1" al id del usuario a modificar en el campo *id_usuario_password.*
3. Identificamos el email del usuario precargándolo en *usuarios_email.*
4. Modificamos la contraseña en el campo *usuarios_contraseña.*
5. El test de pruebas al culminar la inserción de datos deberá presionar la tecla actualizar_contraseña_usuario para ejecutar la prueba

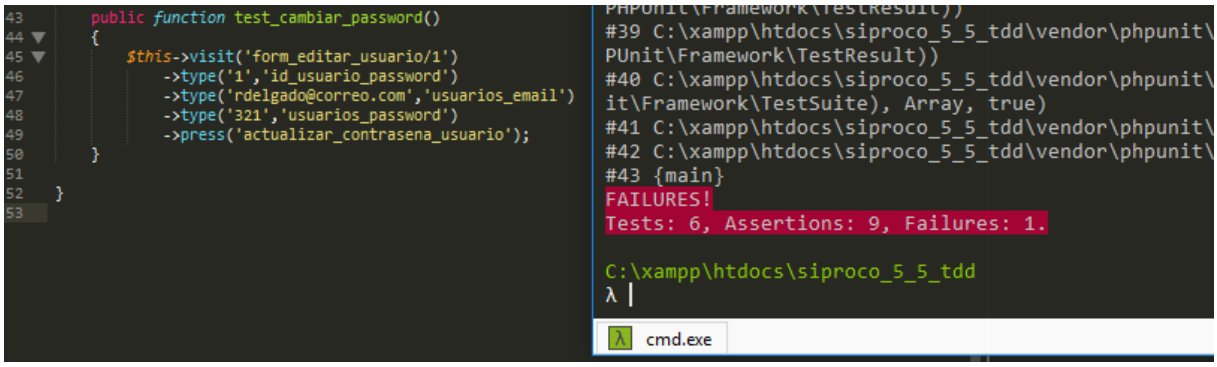


Imagen 28: Test de pruebas HU01-02 - Cambiar Contraseña - FAILURE
Fuente: Elaboración Propia

En el test de pruebas de la HU01-02 para modificar contraseña, falla debido a que está buscando la ruta para cambiar contraseña el cual no existe.

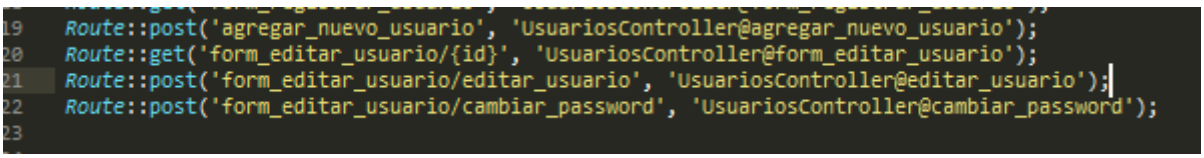


Imagen 29: Ruta del formulario para editar usuario - Cambiar contraseña
Fuente: Elaboración Propia

Se ha creado la ruta *form_editar_usuario@cambiar_password* el cual direcciona al controlador *UsuariosController@cambiar_password*. Para modificar a la contraseña del usuario.

Se crea el método cambiar_password, para modificar la contraseña del usuario seleccionado.

```
web.php    UsuariosController.php
162         }
163     }
164
165     public function cambiar_password(Request $request)
166     {
167         $data        = $request->all();
168         $usuarios_id = $data["id_usuario_password"];
169
170         $usuario                = Usuario::find($usuarios_id);
171         $usuario->usuarios_email = $data["usuarios_email"];
172         $usuario->password       = bcrypt($data["usuarios_password"]);
173
174         $cambiar_password = $usuario->save();
175         if ($cambiar_password) {
176             return view("intranet.mensajes.msj_correcto")->with("msj", "Contraseña actualizada");
177         } else {
178             return view("intranet.mensajes.msj_rechazado")->with("msj", "Error al actualizar la contraseña");
179         }
180
181     }
```

Imagen 30: Método dentro del controlador editar usuario – Cambiar contraseña

Fuente: Elaboración Propia

Se ejecuta el test de pruebas *test_cambiar_password*.

```
{
    $this->visit('form_editar_usuario/1')
         ->type('1','usuarios_id')
         ->type('RedyEditado','usuarios_nombres')
         ->type('DelgadoEditado','usuarios_apellidos')
         ->type('rdelgado@correo.com','usuarios_email')
         ->press('actualizar_usuario');
}

public function test_cambiar_password()
{
    $this->visit('form_editar_usuario/1')
         ->type('1','id_usuario_password')
         ->type('rdelgado@correo.com','usuarios_email')
         ->type('321','usuarios_password')
         ->press('actualizar_contrasena_usuario');
}
```

```
Cmder

C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

......                                                              6 / 6 (100%)

Time: 618 ms, Memory: 12.00MB

OK (6 tests, 9 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 31: Método dentro del controlador editar usuario – Cambiar Contraseña - OK

Fuente: Elaboración Propia

Se observa que el test de pruebas cambiar contraseña, se ha superado y cumplido su propósito con éxito.

5. **Cambiar imagen del usuario.**

Resultado esperado:

Nombre de proceso:	Editar cambiar imagen de usuario	Id caso de prueba:	HU01-02-03
		Id backlog:	HU01-02
		Id historia de usuario:	HU01
Autor de caso de prueba:	Administrador del sistema		

Método	Especificación
test_Cambiar_estado(ruta, int id, string archivo, submit)	Modificar la imagen pre establecida por el sistema.

Descripción de las acciones y/o condiciones para las pruebas

#	Método	Entrada	Salida Esperada
01	test_subir_imagen_usuario ()	form_editar_usuario/1, 1, C:\Users\ARDDS\Desktop\1.png, actualizar_imagen_usuario	Imagen actualizada

Resultado Obtenido:	Imagen actualizada
Dato adicional:	En este test se busca probar la carga de imágenes para distinguir a los usuarios mediante imágenes

Creamos el test de pruebas subir imagen del usuario, para lo cual:

1. Indicamos a la prueba, ir al *form_editar_usuario/1*, el valor "1" es el id del usuario previamente registrado en el primer test de pruebas.
2. Cargamos el id del usuario en el campo *id_usuario_foto*.
3. Escribimos la ruta donde se encuentra el archivo de tipo imagen, en el campo archivo.
4. Indicamos a la prueba, presionar el botón *actualizar_imagen_usuario.*

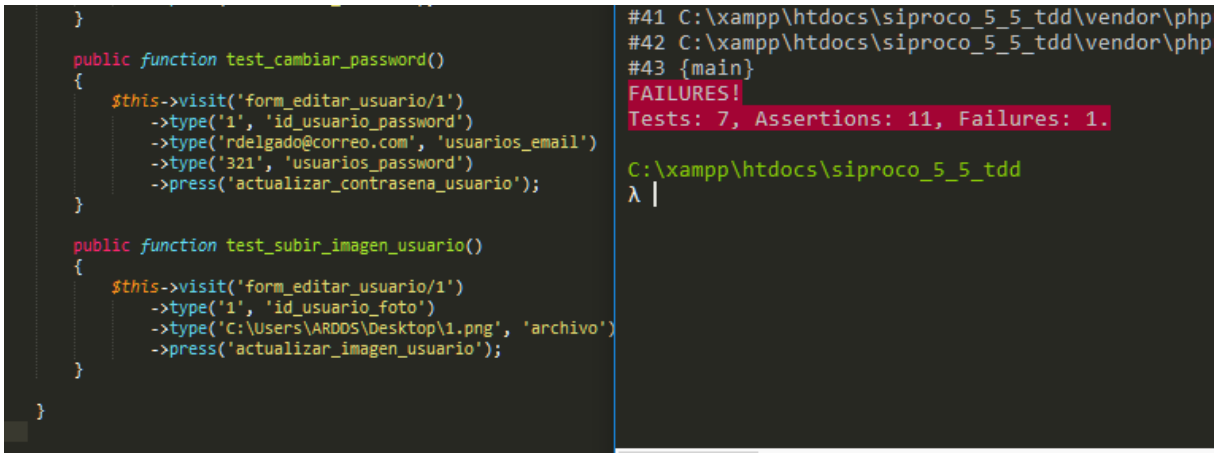


Imagen 32: Test de pruebas HU01-02 - Subir Imagen - FAILURE

Fuente: Elaboración Propia

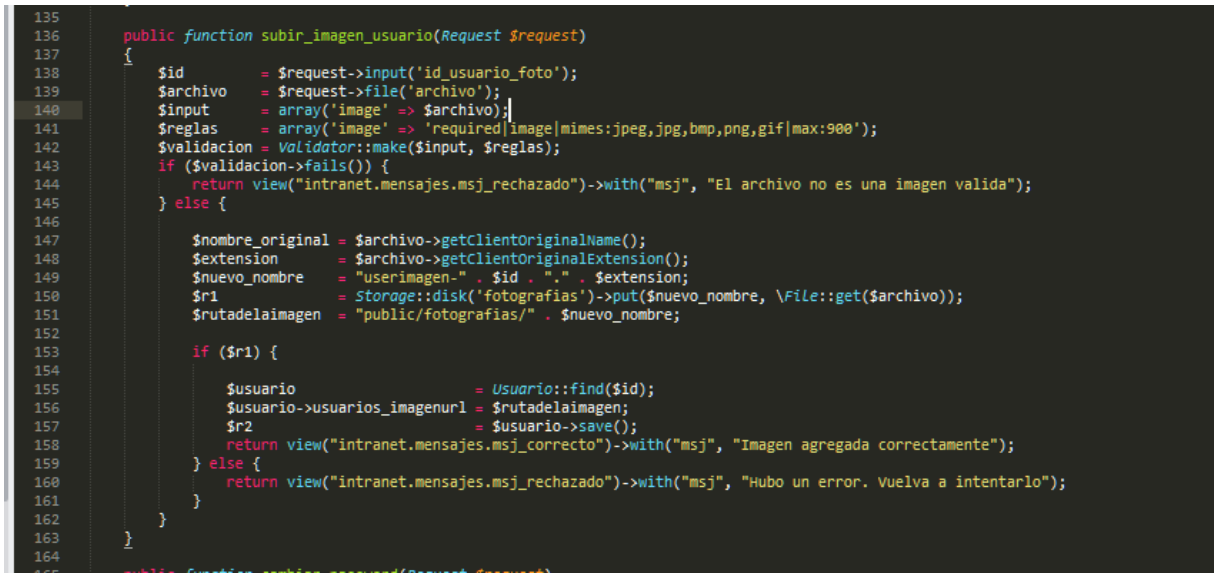
En el test de pruebas de la HU01-02 para modificar la imagen del usuario falla, debido a que está buscando la ruta subir imagen el cual no existe.

```
/*Rutas para agregar nuevo usuario*/
Route::get('form_registrar_usuario', 'UsuariosController@form_registrar_usuario');
Route::post('agregar_nuevo_usuario', 'UsuariosController@agregar_nuevo_usuario');
Route::get('form_editar_usuario/{id}', 'UsuariosController@form_editar_usuario');
Route::post('form_editar_usuario/editar_usuario', 'UsuariosController@editar_usuario');
Route::post('form_editar_usuario/cambiar_password', 'UsuariosController@cambiar_password');
Route::post('form_editar_usuario/subir_imagen_usuario', 'UsuariosController@subir_imagen_usuario');
```

Imagen 33: Ruta del formulario para cambiar imagen - Cambiar contraseña

Fuente: Elaboración Propia

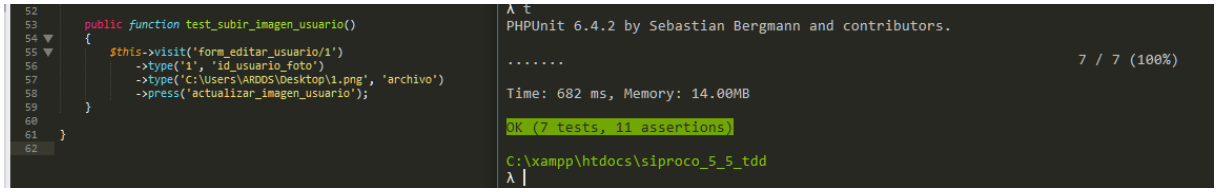
Se ha creado la ruta *form_editar_usuario/subir_imagen_usuario* el cual direcciona al controlador *UsuariosController@subir_imagen_usuario*. Para modificar la imagen del usuario.



```
135
136     public function subir_imagen_usuario(Request $request)
137     {
138         $id         = $request->input('id_usuario_foto');
139         $archivo    = $request->file('archivo');
140         $input      = array('image' => $archivo);
141         $reglas     = array('image' => 'required|image|mimes:jpeg,jpg,bmp,png,gif|max:900');
142         $validacion = Validator::make($input, $reglas);
143         if ($validacion->fails()) {
144             return view("intranet.mensajes.msj_rechazado")->with("msj", "El archivo no es una imagen valida");
145         } else {
146
147             $nombre_original = $archivo->getClientOriginalName();
148             $extension       = $archivo->getClientOriginalExtension();
149             $nuevo_nombre    = "userimagen-" . $id . "." . $extension;
150             $r1              = Storage::disk('fotografias')->put($nuevo_nombre, \File::get($archivo));
151             $rutadelaimagen  = "public/fotografias/" . $nuevo_nombre;
152
153             if ($r1) {
154
155                 $usuario                    = Usuario::find($id);
156                 $usuario->usuarios_imagenurl = $rutadelaimagen;
157                 $r2                         = $usuario->save();
158                 return view("intranet.mensajes.msj_correcto")->with("msj", "Imagen agregada correctamente");
159             } else {
160                 return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
161             }
162         }
163     }
164
```

Imagen 34: Método dentro del controlador editar usuario – Subir imagen

Fuente: Elaboración Propia



```
52
53      public function test_subir_imagen_usuario()
54      {
55          $this->visit('form_editar_usuario/1')
56              ->type('1', 'id_usuario_foto')
57              ->type('C:\Users\ARDOS\Desktop\1.png', 'archivo')
58              ->press('actualizar_imagen_usuario');
59      }
60
61  }
62
```

```
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

.......                                                             7 / 7 (100%)

Time: 682 ms, Memory: 14.00MB

OK (7 tests, 11 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 35: Test de pruebas HU01-02 - Editar Usuario – Subir Imagen - OK

Fuente: Elaboración Propia

En la imagen 35 se observa que el test de pruebas para subir imagen ha culminado con éxito.

Imagen 36: HU01-02 - Formulario Editar Usuario - TDD

Fuente: Elaboración Propia

En la imagen 36, se aprecia el formulario resultante del test de pruebas número 02 y 03.

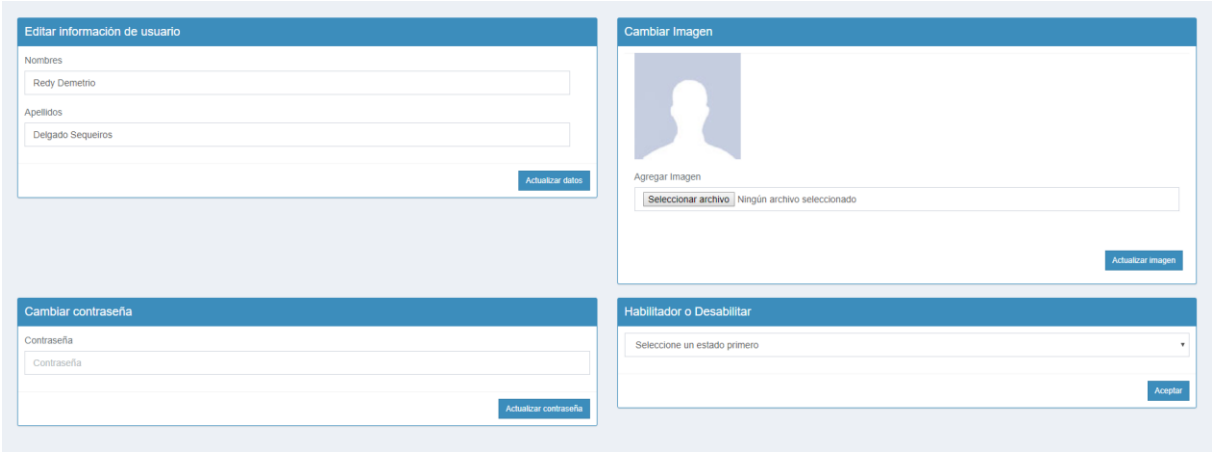


Imagen 37: HU01-02 - Formulario Editar Usuario

Fuente: Elaboración Propia

En la Imagen 37, se puede ver la vista terminada del formulario Editar Usuario.

Del primer Product Backlog - HU01(Historia de Usuario 01), queda claro que esta técnica tiene con un poco de dificultad al inicio y hasta de forma casi concluyente que solo lleva más tiempo el uso de esta técnica.

4.6.2. Product Backlog – HU02

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU02		**El operador de caja, el analista de créditos y el administrador, necesitan modulo para registrar y editar socios dentro del software.**	**1**
	HU02-01	Como usuario administrador, operador de caja, analista de créditos, quiero realizar el registro de socios directamente, para no sobrecargar de trabajo al usuario administrador con cada solicitud de registro. Condiciones: - El módulo debe solicitar datos precisos.	1.1
	HU02-02	Como usuario administrador, operador de caja, analista de créditos, quiero tener el control de los socios registrados, para editar datos de los socios cuando sea necesario. Condiciones: - Debe estar todo en una sola vista.	1.2

Tabla 6: Product Backlog - HU02

Fuente: Elaboración Propia

*Crear el archivo que contendrá los test de pruebas del cliente.

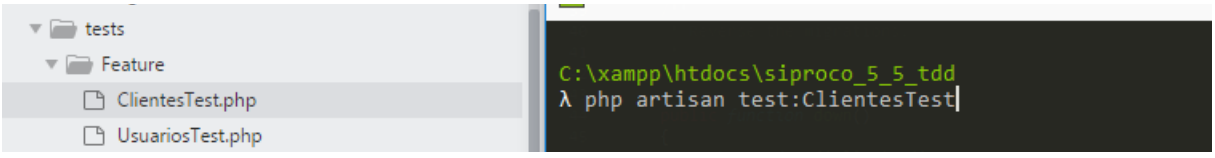


Imagen 38: Creación del Archivo ClientesTest

Fuente: Elaboración Propia

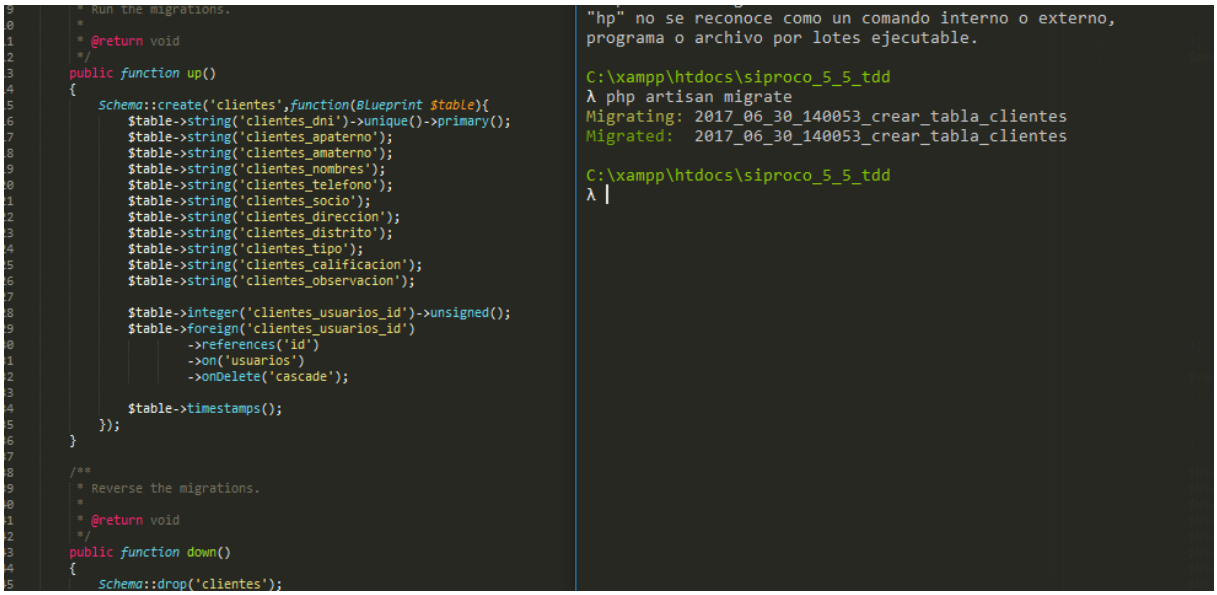


Imagen 39: Tabla Clientes desde Laravel

Fuente: Elaboración Propia

Se crea la tabla “Clientes” en laravel basado en los campos identificados y planteados en el diagrama de clases, y se ejecuta la migración de la tabla al gestor de base de datos.

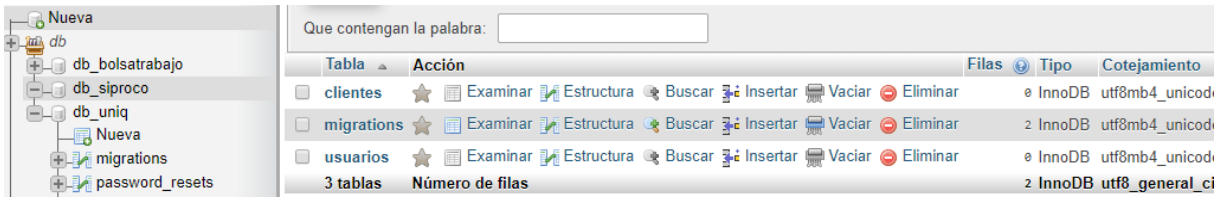


Imagen 40: Tabla Clientes en MySQL

Fuente: Elaboración Propia

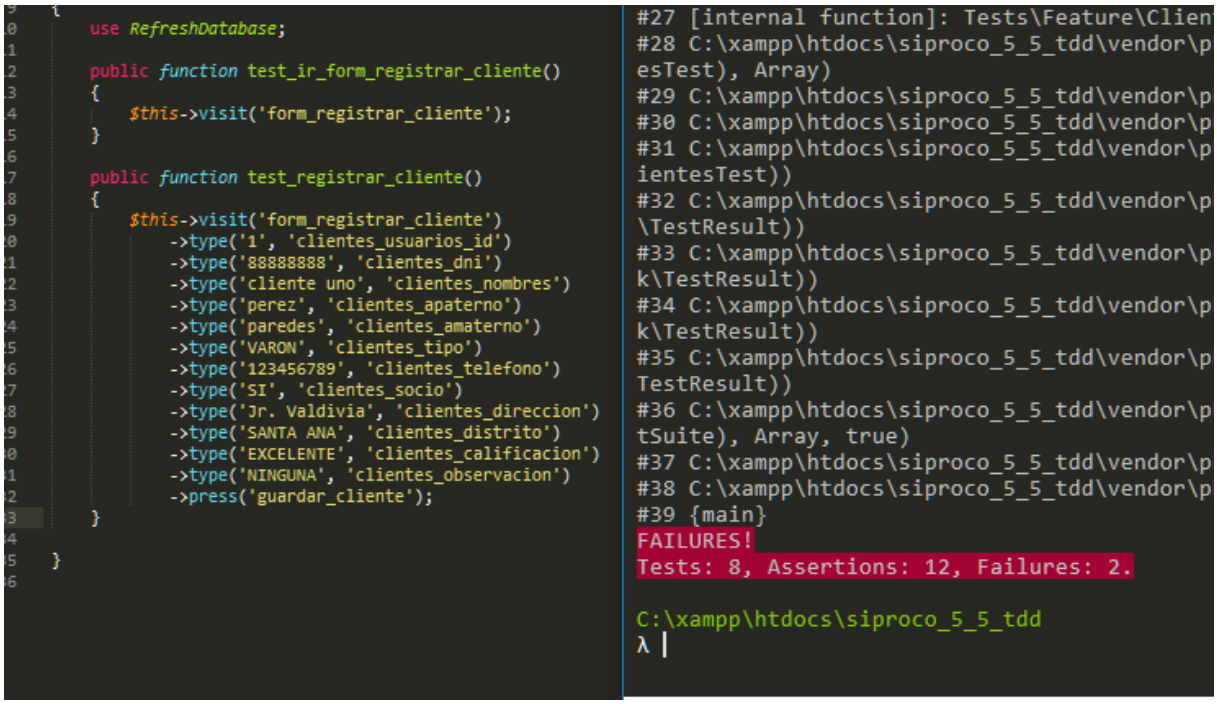
Se puede apreciar que la tabla Clientes ha sido creada en MySQL desde la ejecución de migración hecha en Laravel.

4.6.2.1.Test de Prueba del Product Backlog HU02-01 – Registrar Cliente

<table>
<tr><td>Nombre de proceso:</td><td colspan="2">Registrar cliente</td><td>Id backlog:</td><td>HU02-01</td></tr>
<tr><td></td><td colspan="2"></td><td>Id historia de usuario:</td><td>HU02</td></tr>
<tr><td>Autor de caso de prueba:</td><td colspan="4">Administrador del sistema, Operador de caja, Analista de Créditos</td></tr>
<tr><td colspan="3">Test de prueba</td><td colspan="2">Especificación</td></tr>
<tr><td colspan="3">public function test_registrar_cliente()
{
$this->visit('form_registrar_cliente/1')
->type('12345678', 'clientes_dni')
->type('cliente', 'clientes_nombres')
->type('perez', 'clientes_apaterno')
->type('paredes', 'clientes_amaterno')
->type('VARON', 'clientes_tipo')
->type('555', 'clientes_telefono')
->type('SI', 'clientes_socio')
->type('Jr.Valdivia', 'clientes_direccion')
->type('SANTA ANA', 'clientes_distrito')
->type('EXCELENTE', 'clientes_calificacion')
->type('NINGUNA', 'clientes_observacion')
->type('1', 'clientes_usuarios_id')
->press('guardar_cliente');
}</td><td colspan="2">Registrar al cliente dentro del sistema</td></tr>
<tr><td>Dato adicional:</td><td colspan="4">Los clientes son parte fundamental para el funcionamiento del aplicativo.</td></tr>
</table>

Se crea el test de pruebas test_registrar_cliente, donde le diremos a la prueba ingresar los siguientes datos:

1. Ir a la vista *form_registrar_cliente.*
2. Le asignamos “1” como id de usuario al campo *clientes_usuarios_id* porque “1” es el id del usuario que esta registrando este cliente.
3. Ingresamos datos del cliente: dni, nombres, apellidos, tipo, teléfono, si es socio o no, dirección, distrito, calificación, observación.
4. Al termino la prueba deberá presionar el botón *guardar_cliente.*



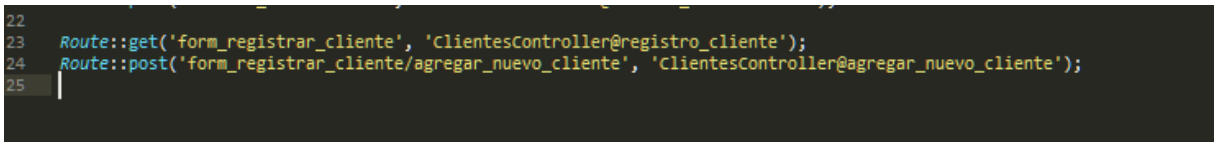
```
    use RefreshDatabase;

    public function test_ir_form_registrar_cliente()
    {
        $this->visit('form_registrar_cliente');
    }

    public function test_registrar_cliente()
    {
        $this->visit('form_registrar_cliente')
            ->type('1', 'clientes_usuarios_id')
            ->type('88888888', 'clientes_dni')
            ->type('cliente uno', 'clientes_nombres')
            ->type('perez', 'clientes_apaterno')
            ->type('paredes', 'clientes_amaterno')
            ->type('VARON', 'clientes_tipo')
            ->type('123456789', 'clientes_telefono')
            ->type('SI', 'clientes_socio')
            ->type('Jr. Valdivia', 'clientes_direccion')
            ->type('SANTA ANA', 'clientes_distrito')
            ->type('EXCELENTE', 'clientes_calificacion')
            ->type('NINGUNA', 'clientes_observacion')
            ->press('guardar_cliente');
    }

}
```

```
#27 [internal function]: Tests\Feature\Clien
#28 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
esTest), Array)
#29 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
#30 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
#31 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
ientesTest))
#32 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
\TestResult))
#33 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
k\TestResult))
#34 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
k\TestResult))
#35 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
TestResult))
#36 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
tSuite), Array, true)
#37 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
#38 C:\xampp\htdocs\siproco_5_5_tdd\vendor\p
#39 {main}
FAILURES!
Tests: 8, Assertions: 12, Failures: 2.

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 41: Test de pruebas HU02-01 -Registrar Cliente - FAILURE

Fuente: Elaboración Propia

Al realizar la ejecución del test de pruebas para registrar un cliente, se observa que nos arroja el error de la no existencia de la ruta.



```
Route::get('form_registrar_cliente', 'ClientesController@registro_cliente');
Route::post('form_registrar_cliente/agregar_nuevo_cliente', 'ClientesController@agregar_nuevo_cliente');
```

Imagen 42: Ruta del formulario para registrar cliente

Fuente: Elaboración Propia

Se ha creado la ruta *form_registrar_cliente* el cual direcciona al controlador *ClientesController@registro_cliente*. Para mostrarnos la vista del formulario.

Se ha creado la ruta *agregar_nuevo_cliente* el cual direccionan al controlador *ClientesController@agregar_nuevo_ cliente*. Para ejecutar el método *agregar_nuevo_cliente.*

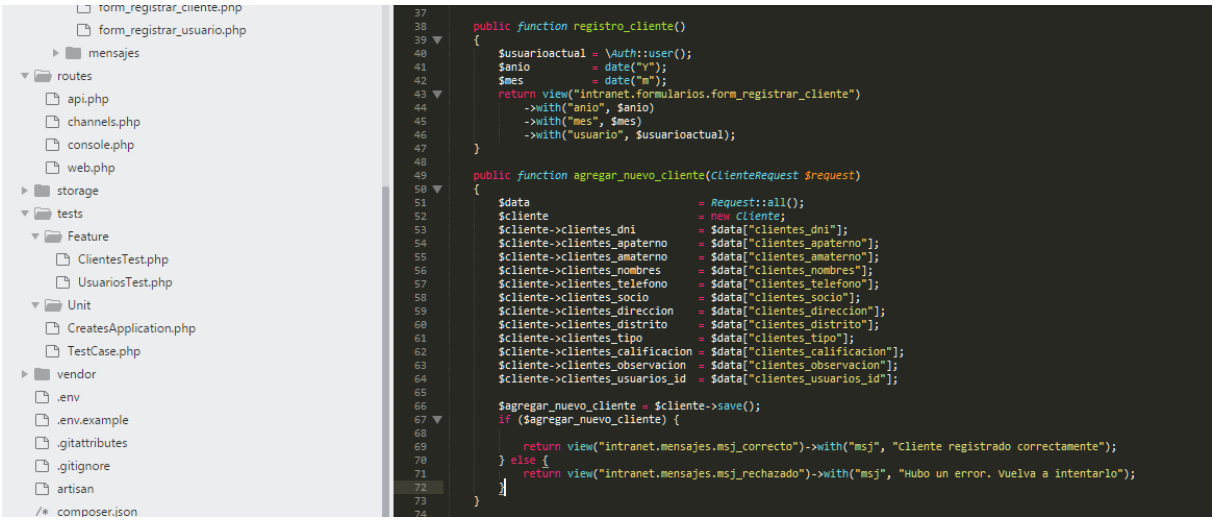


Imagen 43: Método dentro del controlador agregar cliente

Fuente: Elaboración Propia

El método agregar cliente, nos permitirá agregar nuevo cliente.

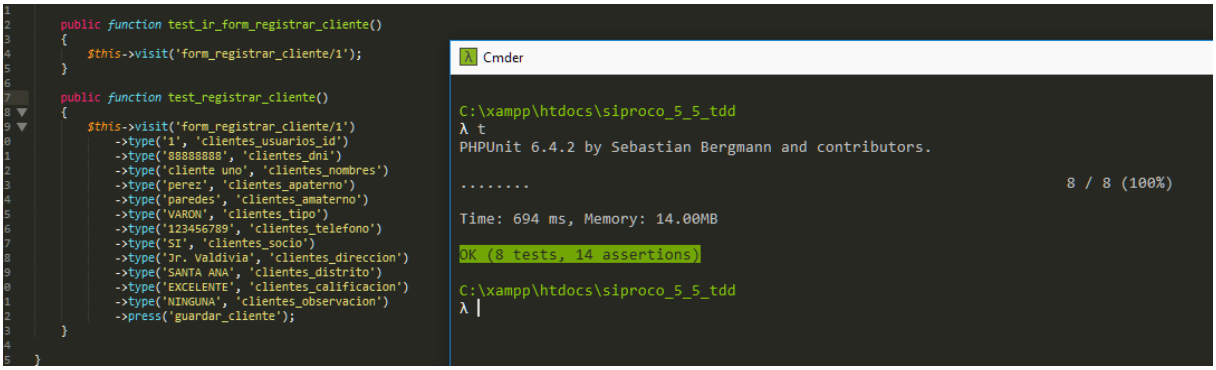


Imagen 44: Test de pruebas HU02-01 -Registrar Cliente - OK

Fuente: Elaboración Propia

El Test de pruebas HU02-01 ha sido ejecutado correctamente, vemos en detalle que ya vamos realizando 8 pruebas correctas.

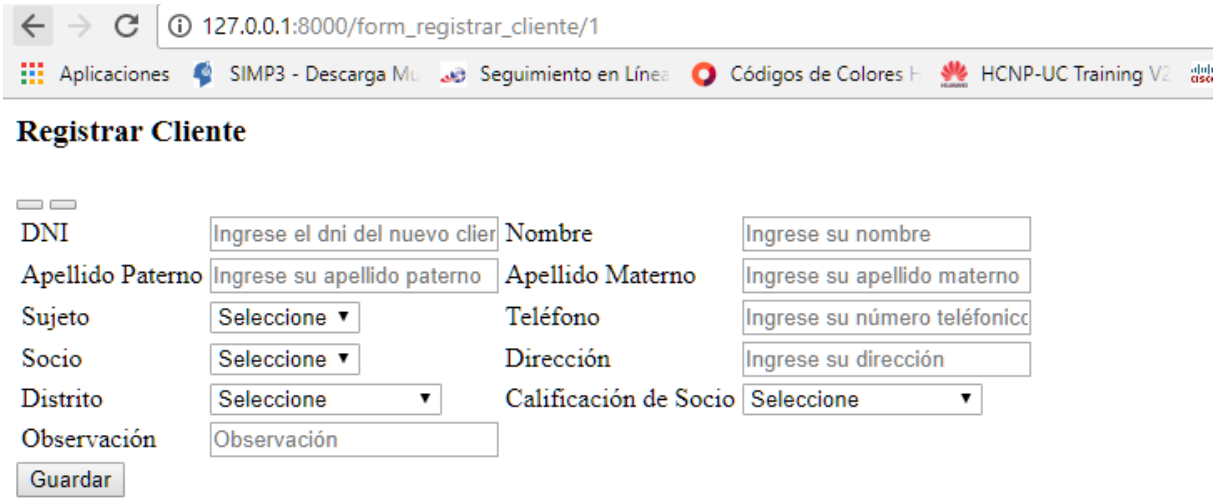


Imagen 45: HU02-01 - Formulario Registrar Cliente – TDD

Fuente: Elaboración Propia

En la imagen 45, se aprecia el formulario de registro de clientes resultante de solo escribir las pruebas primero y ejecutar el test demostrando la funcionalidad deseada.

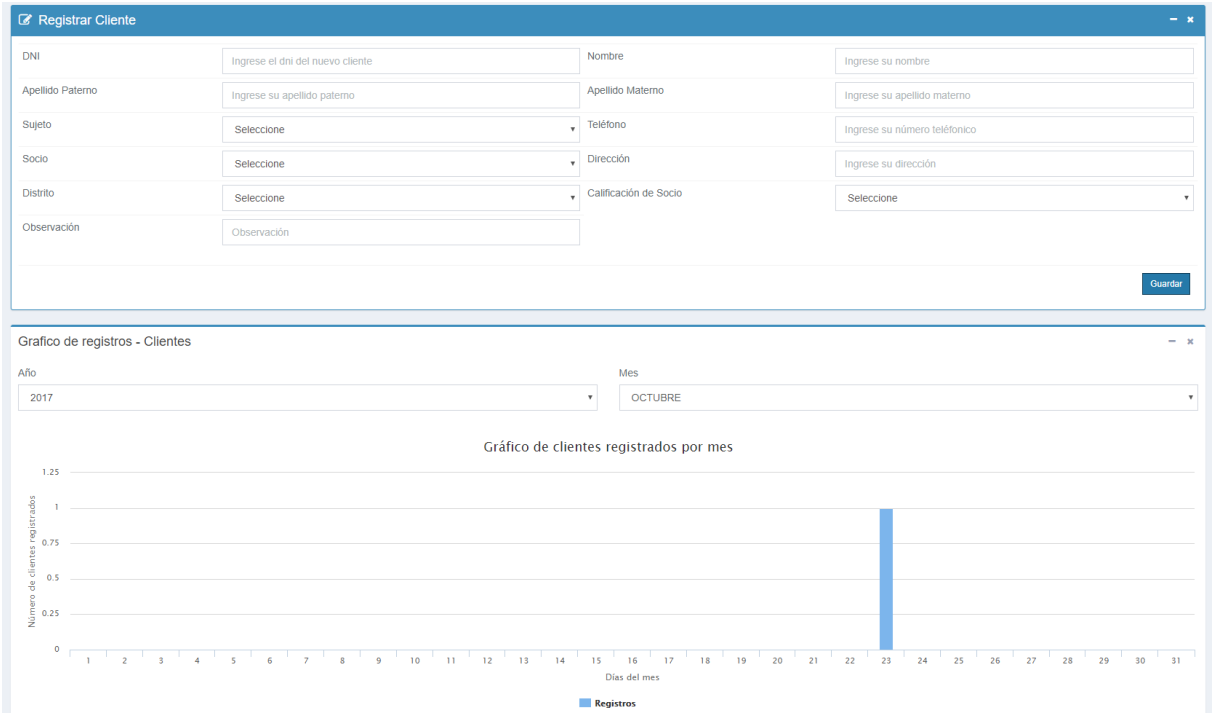


Imagen 46: HU02-01 - Formulario Registrar Cliente

Fuente: Elaboración Propia

La imagen 46, nuestra ya el formulario concluido para registrar el cliente.

4.6.2.2.Test de Prueba del Product Backlog HU02-02 – Editar Cliente

<table>
<tr><td>Nombre de proceso:</td><td>Editar cliente</td><td>Id backlog:</td><td>HU02-02</td></tr>
<tr><td></td><td></td><td>Id historia de usuario:</td><td>HU02</td></tr>
<tr><td>Autor de caso de prueba:</td><td colspan="3">Administrador del sistema, Operador de caja, Analista de Créditos</td></tr>
<tr><td colspan="2">Test de prueba</td><td colspan="2">Especificación</td></tr>
<tr><td colspan="2">public function test_editar_cliente()
{
$this->visit('form_editar_cliente/12345678')
->type('12345678', 'clientes_dni')
->type('clienteEditado', 'clientes_nombres')
->type('perezEditado', 'clientes_apaterno')
->type('paredesEditado', 'clientes_amaterno')
->type('VARON', 'clientes_tipo')
->type('555', 'clientes_telefono')
->type('SI', 'clientes_socio')
->type('Jr.Valdivia', 'clientes_direccion')
->type('SANTA ANA', 'clientes_distrito')
->type('EXCELENTE', 'clientes_calificacion')
->type('NINGUNA', 'clientes_observacion')
->type('1', 'clientes_usuarios_id')
->press('actualizar_cliente');
}</td><td colspan="2">Editar al cliente dentro del sistema.</td></tr>
<tr><td>Dato adicional:</td><td colspan="3">Al editar al cliente, podremos modificar sus datos de ubicación actual.</td></tr>
</table>

Creamos el test de pruebas test_editar_cliente, donde ingresamos los siguientes parámetros para ejecutar la edición de datos:

1. Ir al form_editar_cliente/1, donde “1” es el id del cliente que queremos editar. Nótese que en la base de datos nos generara un id autoincrementado en cada tabla al registrar una nueva fila. Es decir, un id por cada registro.
2. Realizamos la modificación correspondiente en cada campo del formulario editar cliente.
3. Le indicamos a la prueba, presiona r el botón *actualizar_cliente.*

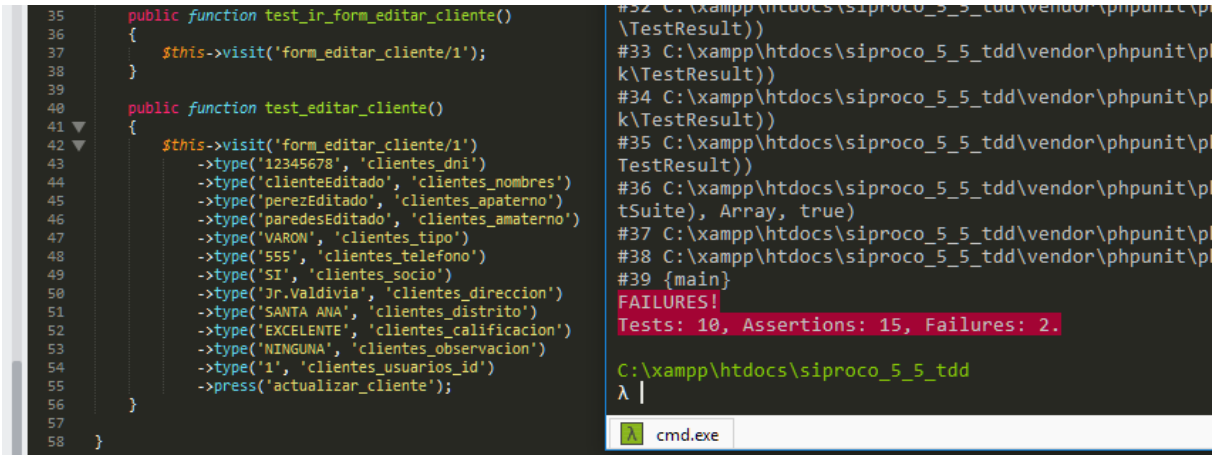


Imagen 47: Test de pruebas HU02-02 - Editar Cliente - FAILURE

Fuente: Elaboración Propia

Como es de procedimiento normal el test de pruebas nos arroja el error esperado, donde indica que falta la ruta para ejecutar la vista y procedimiento indicado.

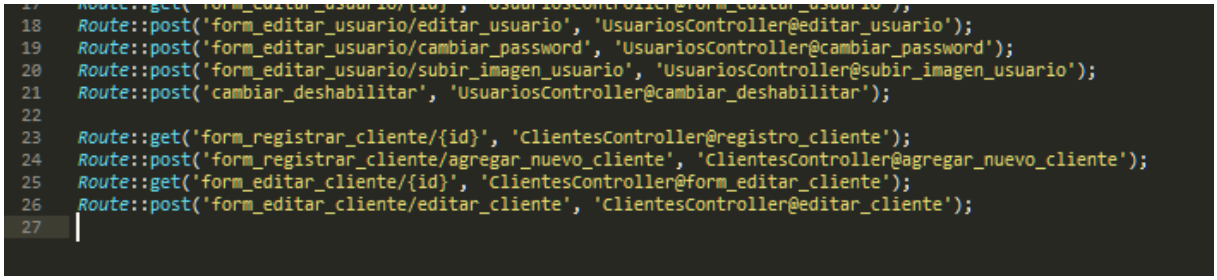
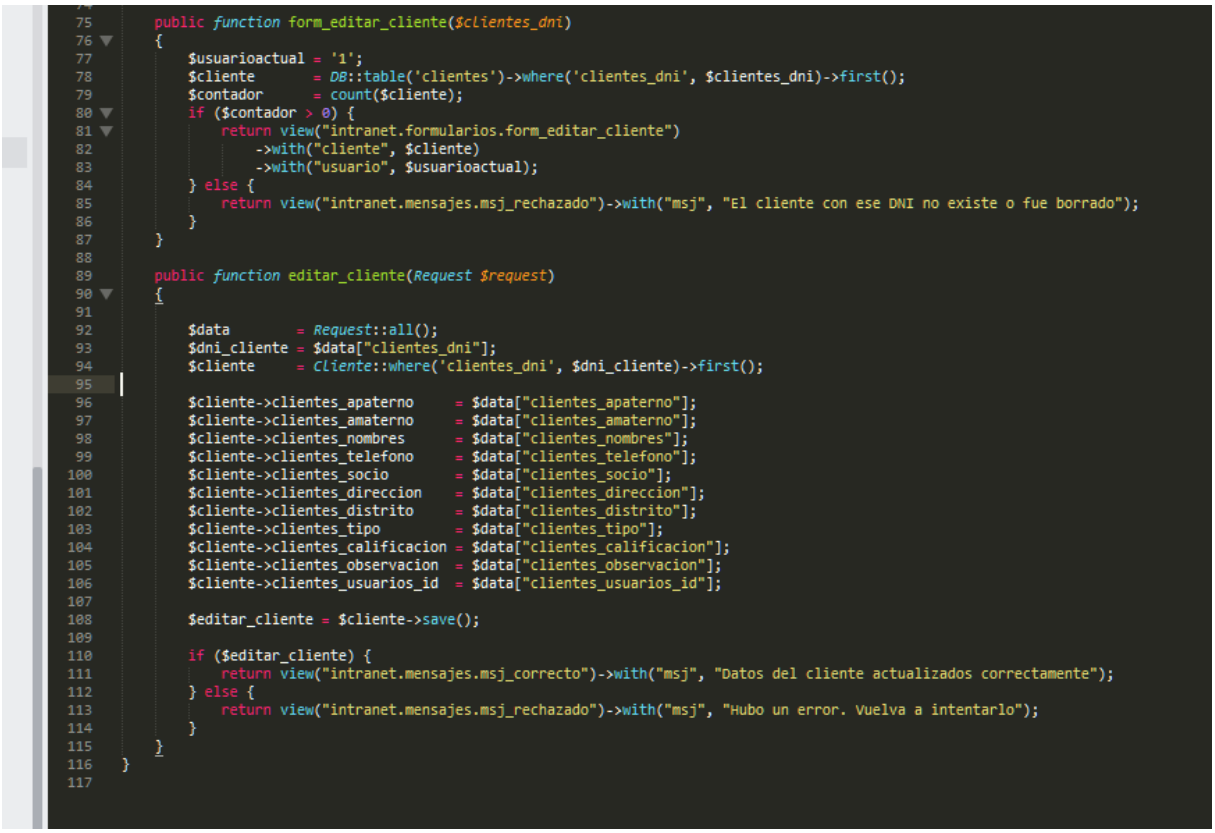


Imagen 48: Ruta del formulario para editar cliente

Fuente: Elaboración Propia

Después del primer test de pruebas de editar cliente, se crea la ruta para ubicar el método.



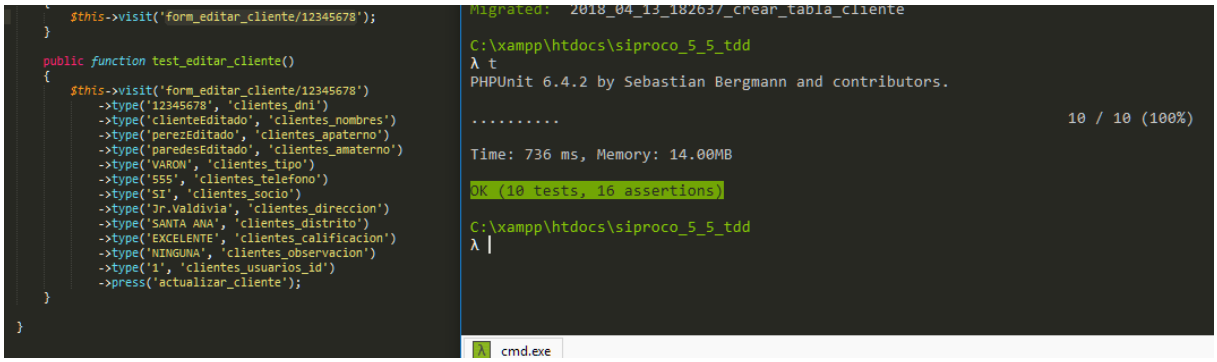
```php
    public function form_editar_cliente($clientes_dni)
    {
        $usuarioactual = '1';
        $cliente       = DB::table('clientes')->where('clientes_dni', $clientes_dni)->first();
        $contador      = count($cliente);
        if ($contador > 0) {
            return view("intranet.formularios.form_editar_cliente")
                ->with("cliente", $cliente)
                ->with("usuario", $usuarioactual);
        } else {
            return view("intranet.mensajes.msj_rechazado")->with("msj", "El cliente con ese DNI no existe o fue borrado");
        }
    }

    public function editar_cliente(Request $request)
    {

        $data        = Request::all();
        $dni_cliente = $data["clientes_dni"];
        $cliente     = Cliente::where('clientes_dni', $dni_cliente)->first();

        $cliente->clientes_apaterno     = $data["clientes_apaterno"];
        $cliente->clientes_amaterno     = $data["clientes_amaterno"];
        $cliente->clientes_nombres      = $data["clientes_nombres"];
        $cliente->clientes_telefono     = $data["clientes_telefono"];
        $cliente->clientes_socio        = $data["clientes_socio"];
        $cliente->clientes_direccion    = $data["clientes_direccion"];
        $cliente->clientes_distrito     = $data["clientes_distrito"];
        $cliente->clientes_tipo         = $data["clientes_tipo"];
        $cliente->clientes_calificacion = $data["clientes_calificacion"];
        $cliente->clientes_observacion  = $data["clientes_observacion"];
        $cliente->clientes_usuarios_id  = $data["clientes_usuarios_id"];

        $editar_cliente = $cliente->save();

        if ($editar_cliente) {
            return view("intranet.mensajes.msj_correcto")->with("msj", "Datos del cliente actualizados correctamente");
        } else {
            return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
        }
    }
}
```

Imagen 49: Método dentro del controlador editar cliente

Fuente: Elaboración Propia

Este método nos permitirá editar los clientes que tengamos registrado en el sistema.

Ejecutamos el test de pruebas test_editar_cliente.



```
    $this->visit('form_editar_cliente/12345678');
}

public function test_editar_cliente()
{
    $this->visit('form_editar_cliente/12345678')
        ->type('12345678', 'clientes_dni')
        ->type('clienteEditado', 'clientes_nombres')
        ->type('perezEditado', 'clientes_apaterno')
        ->type('paredesEditado', 'clientes_amaterno')
        ->type('VARON', 'clientes_tipo')
        ->type('555', 'clientes_telefono')
        ->type('SI', 'clientes_socio')
        ->type('Jr.Valdivia', 'clientes_direccion')
        ->type('SANTA ANA', 'clientes_distrito')
        ->type('EXCELENTE', 'clientes_calificacion')
        ->type('NINGUNA', 'clientes_observacion')
        ->type('1', 'clientes_usuarios_id')
        ->press('actualizar_cliente');
}
}
```

```
Migrated:  2018_04_13_182637_crear_tabla_cliente

C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

.........                                                         10 / 10 (100%)

Time: 736 ms, Memory: 14.00MB

OK (10 tests, 16 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

cmd.exe

Imagen 50: Test de pruebas HU02-02 - Editar Cliente - OK

Fuente: Elaboración Propia

La décima prueba HU02-02 en el desarrollo de este software, ha sido ejecutada y concluida con éxito.

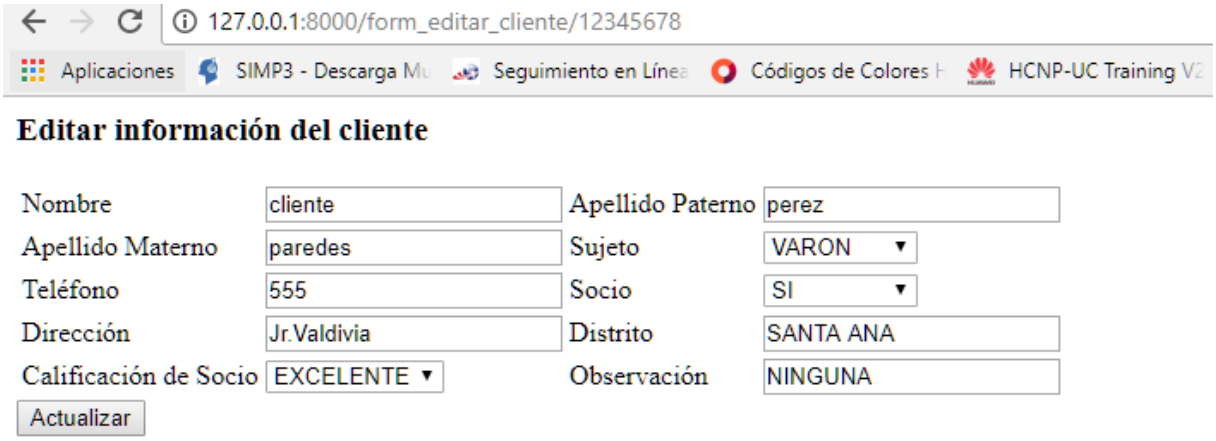


Imagen 51: HU02-02 - Formulario Editar Cliente – TDD

Fuente: Elaboración Propia

En la última imagen, se visualiza el formulario de editar usuario demostrando una vez más la funcionalidad deseada, para este formulario.

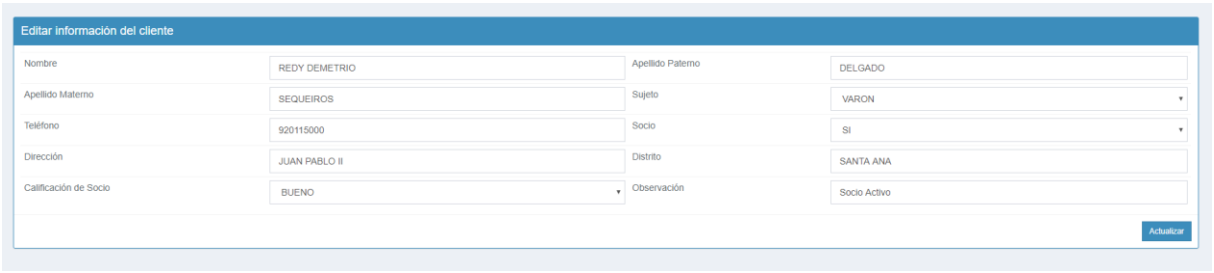


Imagen 52: HU02-02 - Formulario Editar Cliente

Fuente: Elaboración Propia

La imagen 52, ya nos muestra la forma final del formulario editar cliente.

4.6.3. Product Backlog – HU03

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU03		**El administrador necesita un módulo de configuración para gestionar datos generales del software (Registro de operaciones – operador de caja, registro de actividades – analista de créditos, categoría de publicaciones - administrador).**	**1**
	HU03-01	Como usuario administrador del sistema, quiero poder registrar tipos de procesos en caja. Debido a que los procesos del operador de caja son muchas, y deben de ser dividas en dos tipos. Condiciones: - Cada proceso debe de ser o positivo o negativo. - El formulario, debe de tener su listado.	1.1
	HU03-02	Como usuario administrador del sistema, quiero poder registrar tipos de actividades que realizan los analistas de créditos. Debido a que sus acciones son muchas. Condiciones: - El formulario, debe de tener su listado.	1.2
	HU03-03	Como usuario administrador del sistema, quiero poder registrar tipos de las publicaciones personalmente, para clasificar la categoría de la publicación. Condiciones: - El formulario, debe de tener su listado.	1.3

Tabla 7: Product Backlog - HU03

Fuente: Elaboración Propia

*Crear el archivo que contendrá los test de pruebas de los procesos, actividades y publicaciones.

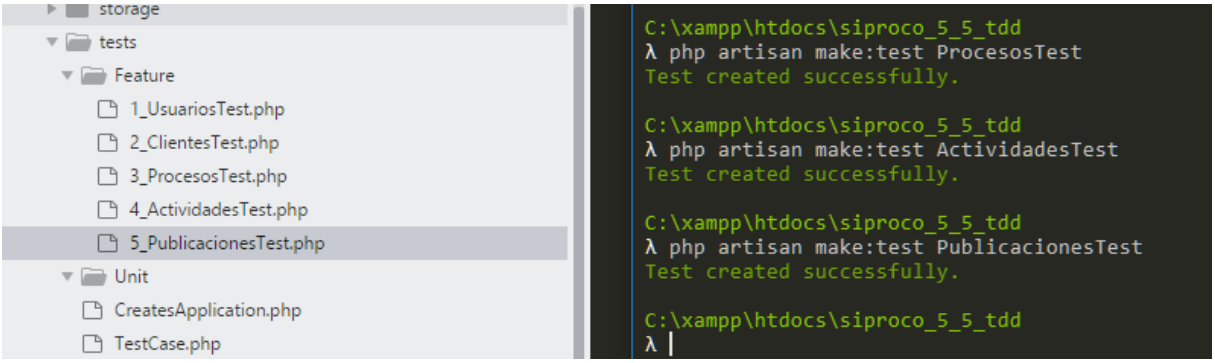


Imagen 53: Creación del Archivo ProcesosTest, ActividadesTest y PublicacionesTest

Fuente: Elaboración Propia

Imagen 54: Tabla Procesos, Tabla Actividades, Tabla TipoPublicaciones desde Laravel

Fuente: Elaboración Propia

Se crea las tablas “Procesos, Actividades, TipoPublicaciones” desde laravel, y se ejecuta la migración de las tablas al gestor de base de datos. Con el fin de que estas tablas nos sirvan como configuración para lograr el requerimiento del usuario.

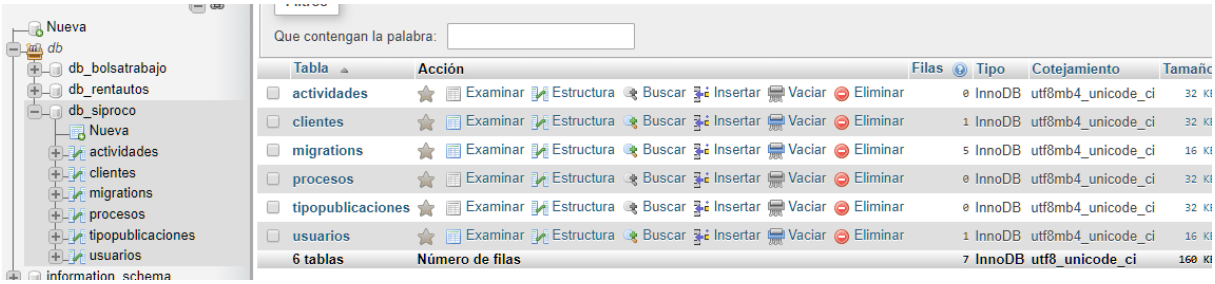


Imagen 55: Tabla Procesos, Actividades y TipoPublicaciones en MySQL

Fuente: Elaboración Propia

Se puede apreciar que las tablas “Procesos, Actividades, *TipoPublicaciones* han sido creadas en MySQL desde la ejecución de migración hecha en Laravel.

4.6.3.1.Test de Prueba del Product Backlog HU03-01 – Registrar Tipos de Procesos

Nombre de proceso:	Registrar Tipos de Procesos	Id backlog:	HU03-01
		Id historia de usuario:	HU03
Autor de caso de prueba:	Administrador del sistema		

Test de prueba	Especificación

public function test_registrar_proceso()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('form_registrar_proceso')
->type('COBRO', 'procesos_proceso')
->type('POSITIVO', 'procesos_tipoproceso')
->type('1', 'procesos_usuarios_id')
->press('agregar_proceso');
}
	Los tipos de procesos, pueden ser positivo o negativo, según su descripción.

Dato adicional:	Los tipos de procesos podrán ayudarnos a determinar si el proceso fue a favor o en contra de la cooperativa.

Creamos el test de pruebas test_registrar_procesos. Ingresamos los datos.

1. Autentificamos al usuario “1”, para ejecutar este test de pruebas.
2. Indicamos a la prueba ir *form_registrar_proceso.*
3. Ingresamos datos del proceso: proceso, tipo de proceso y el id del usuario que realiza el registro.

4. Al termino del ingreso la prueba deberá presionar el botón *agregar_proceso.*

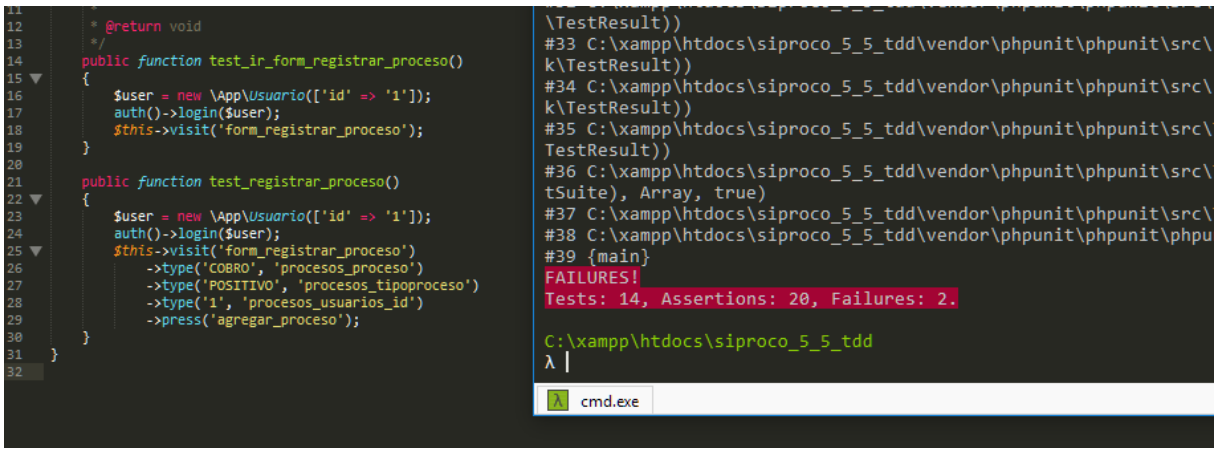


Imagen 56: Test de pruebas HU03-01 – Registrar Proceso - FAILURE

Fuente: Elaboración Propia

El test de pruebas para el registro de procesos falla, una vez más indica que falta la ruta para encontrar el método que permita la inserción de datos en la base de datos.

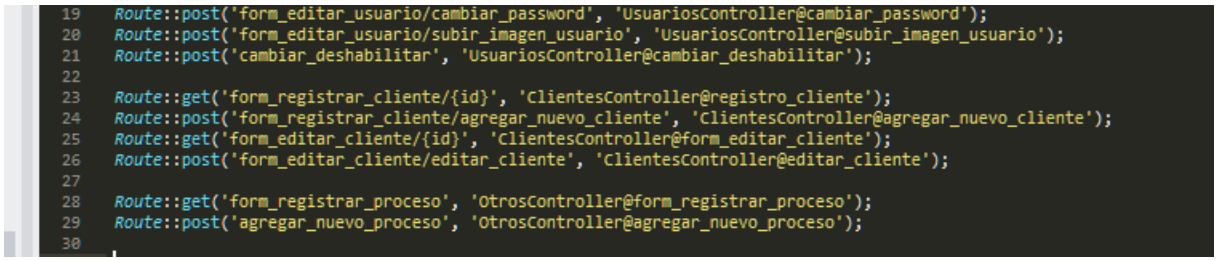


Imagen 57: Ruta del formulario para registrar proceso

Fuente: Elaboración Propia

Se crea el método que permita ingresar los datos a la base de datos, desde el modelo.

```php
public function agregar_nuevo_proceso(Request $request)
{
    $data   = Request::all();
    $reglas = array('procesos_proceso' => 'required|Alpha',
        'procesos_tipoproceso'             => 'required|Alpha',
    );
    $mensajes = array('procesos_proceso.required' => 'Ingresar la denominación del proceso',
        'procesos_proceso.alpha'                  => 'El denominación del proceso no puede contener números',
        'procesos_tipoproceso.required'           => 'Seleccionar el tipo de proceso el cual es obligatorio',
        'procesos_tipoproceso.alpha'              => 'Debe de seleccionar el tipo del proceso',
    );

    $validacion = Validator::make($data, $reglas, $mensajes);
    if ($validacion->fails()) {
        $errores = $validacion->errors();
        return new JsonResponse($errores, 422);
    }
    $proceso                      = new Proceso;
    $proceso->procesos_proceso     = $data["procesos_proceso"];
    $proceso->procesos_tipoproceso = $data["procesos_tipoproceso"];
    $proceso->procesos_usuarios_id = $data["procesos_usuarios_id"];
    $proceso->procesos_habilitado = 's';

    $agregar_nuevo_proceso = $proceso->save();
    if ($agregar_nuevo_proceso) {
        return view("intranet.mensajes.msj_correcto")->with("msj", "Nuevo proceso registrado correctamente.");
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
    }
}
```

Imagen 58: Método Agregar Nuevo Proceso dentro del controlador OtrosController

Fuente: Elaboración Propia

Se ejecuta el test de pruebas *test_registrar_procesos* para el registro de procesos.

```php
     * @return void
     */
    public function test_ir_form_registrar_proceso()
    {
        $user = new \App\Usuario(['id' => '1']);
        auth()->login($user);
        $this->visit('form_registrar_proceso');
    }

    public function test_registrar_proceso()
    {
        $user = new \App\Usuario(['id' => '1']);
        auth()->login($user);
        $this->visit('form_registrar_proceso')
            ->type('COBRO', 'procesos_proceso')
            ->type('POSITIVO', 'procesos_tipoproceso')
            ->type('1', 'procesos_usuarios_id')
            ->press('agregar_proceso');
    }
}
```

```
Migrating: 2018_04_14_045251_crear_tabla_actividades
Migrated:  2018_04_14_045251_crear_tabla_actividades
Migrating: 2018_04_14_045311_crear_tabla_tipopublicaciones
Migrated:  2018_04_14_045311_crear_tabla_tipopublicaciones

C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

..............                                                    14 / 14 (100%)

Time: 879 ms, Memory: 14.00MB

OK (14 tests, 21 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 59: Test de pruebas HU03-01 – Registrar Proceso – OK

Fuente: Elaboración Propia

Se observa que el test de pruebas *test_registrar_procesos* ha superado la prueba con éxito.

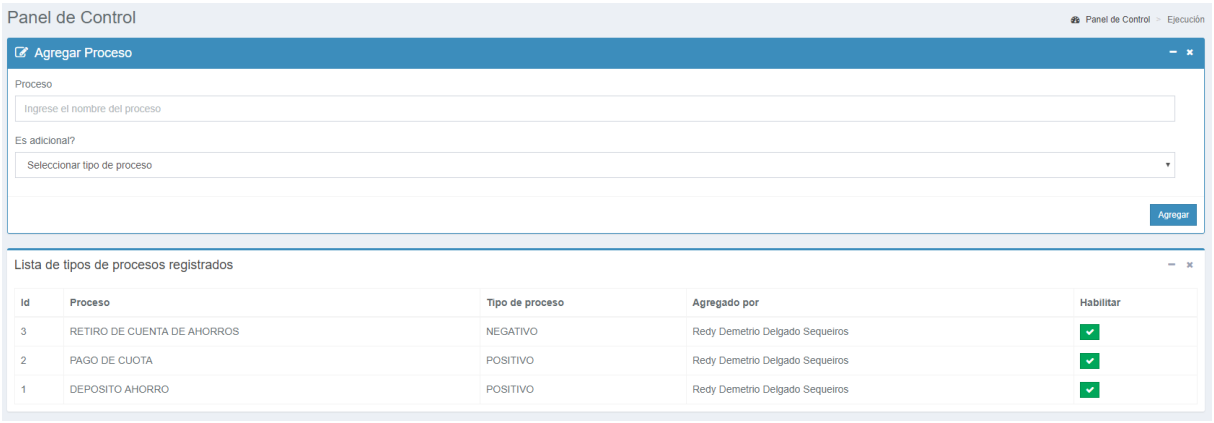


Imagen 60: HU03-01 - Formulario Agregar Proceso

Fuente: Elaboración Propia

La imagen 60, muestra el formulario culminado para el registro de proceso.

4.6.3.2.Test de Prueba del Product Backlog HU03-02 – Registrar Tipos de Actividades

Nombre de proceso:	Registrar Tipos de Actividades	Id backlog:	HU03-02
		Id historia de usuario:	HU03
Autor de caso de prueba:	Administrador del sistema		

Test de prueba	Especificación

public function test_registrar_actividad()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('form_registrar_actividad')
->type('CREDITO', 'actividades_nombreactividad')
->type('1', 'actividades_usuarios_id')
->press('agregar_actividad');
}
	Las actividades, las realizan los analistas de créditos.

Dato adicional:	Establecer la descripción de la actividad es necesario, para que el analista de créditos pueda registrar sus actividades diarias.

Creamos el test de pruebas test_registrar_actividad. Ingresamos los datos.

1. Autentificamos al usuario “1”, para ejecutar este test de pruebas.
2. Indicamos a la prueba ir *form_registrar_actividad.*
3. Ingresamos el nombre de la actividad y el id del usuario que realiza el registro.

4. Al término del ingreso la prueba deberá presionar el botón *agregar_proceso.*

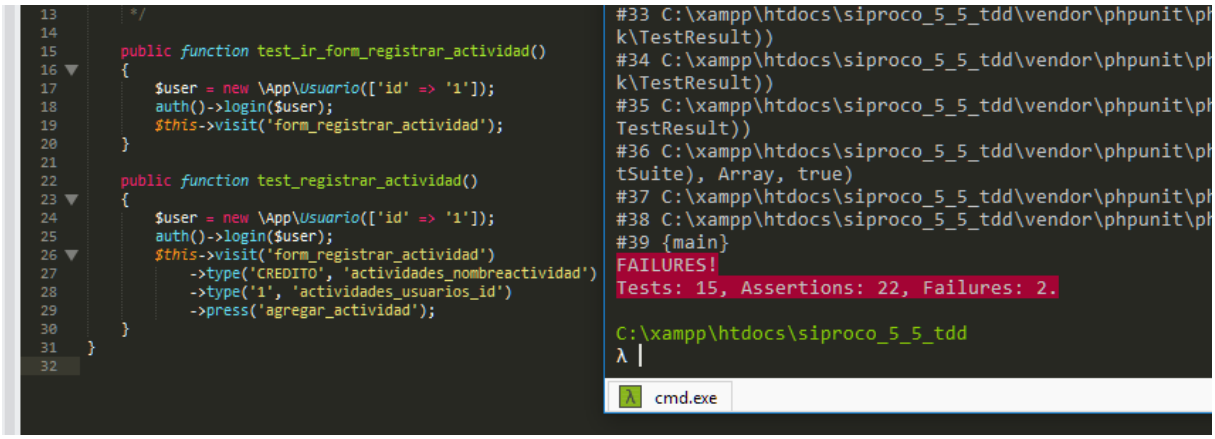


Imagen 61: Test de pruebas HU03-02 – Registrar Actividad - FAILURE

Fuente: Elaboración Propia

Se ejecuta el test de pruebas *test_registra_actividad*, nos arroja error.

Se crea la ruta para la vista y la ruta para direccionar el método.

```
Route::get('form_registrar_actividad', 'OtrosController@form_registrar_actividad');
Route::post('agregar_nuevo_actividad', 'OtrosController@agregar_nuevo_actividad');
```

Imagen 62: Ruta del formulario para registrar actividad

Fuente: Elaboración Propia

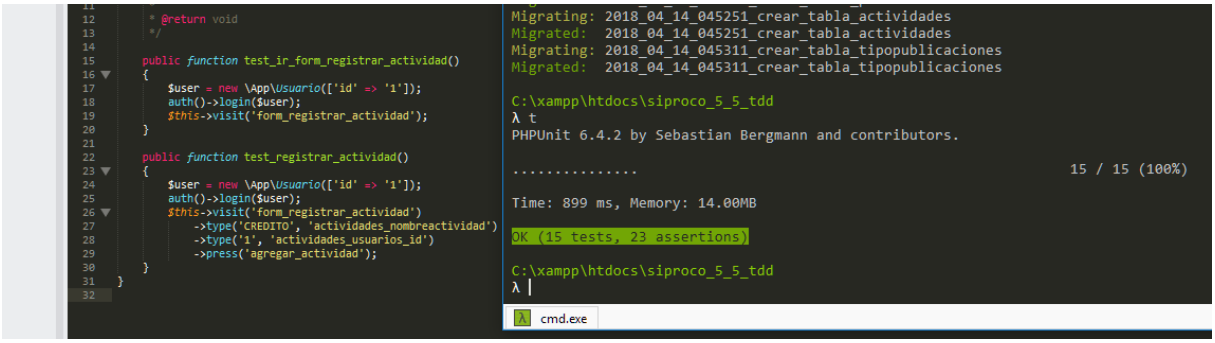
Creamos el método *agregar_nuevo_actividad* que permitirá agregar una nueva actividad

```php
public function agregar_nuevo_actividad(Request $request)
{
    $data    = Request::all();
    $reglas = array('actividades_nombreactividad' => 'required|Alpha',
    );
    $mensajes = array('actividades_nombreactividad.required' => 'Ingresar la denominación de la actividad',
        'actividades_nombreactividad.alpha'                   => 'El denominación de la actividad no puede contener números',
    );

    $validacion = Validator::make($data, $reglas, $mensajes);
    if ($validacion->fails()) {
        $errores = $validacion->errors();
        return new JsonResponse($errores, 422);
    }
    $actividad                          = new Actividad;
    $actividad->actividades_nombreactividad = $data["actividades_nombreactividad"];
    $actividad->actividades_usuarios_id     = $data["actividades_usuarios_id"];
    $actividad->actividades_habilitado      = 's';

    $agregar_nuevo_actividad = $actividad->save();
    if ($agregar_nuevo_actividad) {

        return view("intranet.mensajes.msj_correcto")->with("msj", "Nueva actividad registrada correctamente.");
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
    }
}
```

Imagen 63: Método agregar_nueva_actividad dentro del controlador OtrosController

Fuente: Elaboración Propia

Ejecutamos el test de pruebas, el cual nos devuelve prueba superada.



```php
 * @return void
 */

public function test_ir_form_registrar_actividad()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('form_registrar_actividad');
}

public function test_registrar_actividad()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('form_registrar_actividad')
        ->type('CREDITO', 'actividades_nombreactividad')
        ->type('1', 'actividades_usuarios_id')
        ->press('agregar_actividad');
}
}
```

```
Migrating: 2018_04_14_045251_crear_tabla_actividades
Migrated:  2018_04_14_045251_crear_tabla_actividades
Migrating: 2018_04_14_045311_crear_tabla_tipopublicaciones
Migrated:  2018_04_14_045311_crear_tabla_tipopublicaciones

C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

...............                                                 15 / 15 (100%)

Time: 899 ms, Memory: 14.00MB

OK (15 tests, 23 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 64: Test de pruebas HU03-02 – Registrar Actividad - OK

Fuente: Elaboración Propia

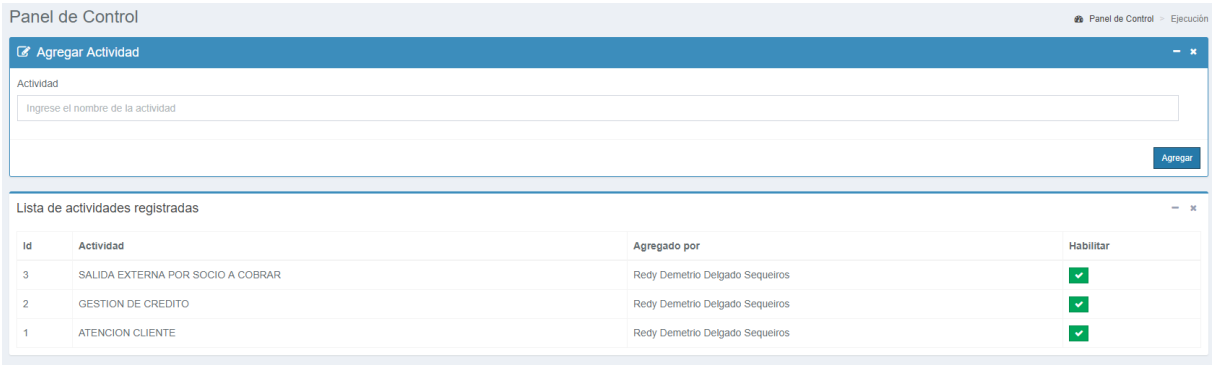


Imagen 65: HU03-02 - Formulario Agregar Actividad

Fuente: Elaboración Propia

En la imagen 65, se puede visualizar el formulario agregar actividad concluido.

4.6.3.3.Test de Prueba del Product Backlog HU03-03 – Registrar Tipos de Publicaciones

Nombre de proceso:	Registrar Tipos de Publicaciones	Id backlog:	HU03-03
		Id historia de usuario:	HU03
Autor de caso de prueba:	Administrador del sistema		

Test de prueba	Especificación

public function test_agregar_nuevo_tipopublicacion()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('form_registrar_publicacion')
->type('REGLAMENTO', 'tipopublicaciones_titulo')
->type('1', 'tipopublicaciones_usuarios_id')
->press('agregar_categoriapublicaicon');
}
	Los tipos de publicaciones son necesarias para categorizar futuras publicaciones.

Dato adicional:	Organiza el tipo de publicación, para los exámenes.

Creamos el test de pruebas test_agregar_nuevo_tipopublicación. Ingresamos los datos.

1. Autentificamos al usuario “1”, para ejecutar este test de pruebas.
2. Indicamos a la prueba ir *form_registrar_publicacion.*
3. Ingresamos el nombre del tipo de publicación y el id del usuario que realiza el registro.
4. Al término del ingreso la prueba deberá presionar el botón *agregar_categoriapublicacion.*

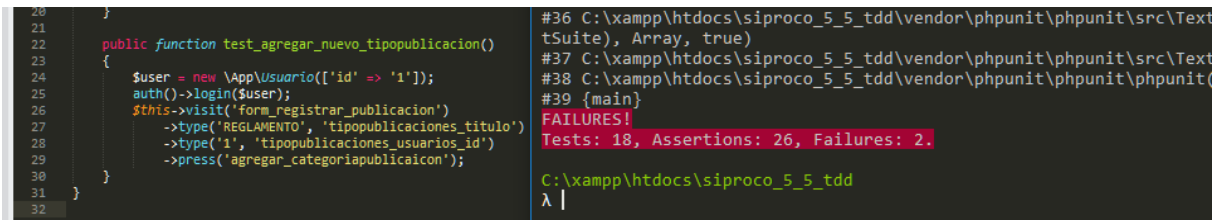


Imagen 66: Test de pruebas HU03-03 – Registrar Tipos de Publicaciones - FAILURE

Fuente: Elaboración Propia

El test de pruebas ejecutado retorna error por la falta de ruta para encontrar la vista y/o método a ejecutar.

```
Route::get('form_registrar_cliente/{id}', 'ClientesController@registro_cliente');
Route::post('form_registrar_cliente/agregar_nuevo_cliente', 'ClientesController@agregar_nuevo_cliente');
Route::get('form_editar_cliente/{id}', 'ClientesController@form_editar_cliente');
Route::post('form_editar_cliente/editar_cliente', 'ClientesController@editar_cliente');

Route::get('form_registrar_proceso', 'OtrosController@form_registrar_proceso');
Route::post('agregar_nuevo_proceso', 'OtrosController@agregar_nuevo_proceso');
Route::get('listado_procesos', 'OtrosController@listadoprocesos');
Route::get('habilitar_proceso/{id_proceso}/{habilitar}', 'OtrosController@habilitar_proceso');

Route::get('form_registrar_actividad', 'OtrosController@form_registrar_actividad');
Route::post('agregar_nuevo_actividad', 'OtrosController@agregar_nuevo_actividad');
Route::get('listado_actividades', 'OtrosController@listadoactividades');
Route::get('habilitar_actividad/{id_proceso}/{habilitar}', 'OtrosController@habilitar_actividad');

Route::get('form_registrar_publicacion', 'OtrosController@form_registrar_publicacion');
Route::post('agregar_nuevo_tipopublicacion', 'OtrosController@agregar_nuevo_tipopublicacion');
```

Imagen 67: Ruta del formulario para registrar tipo de publicación

Fuente: Elaboración Propia

Se crea el método que permita el almacenamiento de la nueva publicación.

```
public function agregar_nuevo_tipopublicacion(Request $request)
{

    $data = Request::all();

    $reglas = array('tipopublicaciones_titulo' => 'required|Alpha|max:150',
    );
    $mensajes = array('tipopublicaciones_titulo.required' => 'Ingresar un nombre para el tipo de categoría documentaria',
        'tipopublicaciones_titulo.alpha'                  => 'El categoría no puede contener números',
    );

    $validacion = Validator::make($data, $reglas, $mensajes);
    if ($validacion->fails()) {
        $errores = $validacion->errors();
        return new JsonResponse($errores, 422);
    }

    $tipopublicacion                                = new TipoPublicacion;
    $tipopublicacion->tipopublicaciones_titulo      = $data["tipopublicaciones_titulo"];
    $tipopublicacion->tipopublicaciones_usuarios_id = $data["tipopublicaciones_usuarios_id"];
    $tipopublicacion->tipopublicaciones_habilitado = 's';

    $agregar_nuevo_tipopublicacion                  = $tipopublicacion->save();
    if ($agregar_nuevo_tipopublicacion) {

        return view("intranet.mensajes.msj_correcto")->with("msj", "Tipo de publicación registrada correctamente");
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
    }

}
```

Imagen 68: Método agregar_nuevo_tipopublicaicon dentro del controlador OtrosController

Fuente: Elaboración Propia

Ejecutamos el test de pruebas test_agregar_nuevo_tipopublicacion con éxito.

```php
class PublicacionesTest extends TestCase
{
    /**
     * A basic test example.
     *
     * @return void
     */

    public function test_ir_form_registrar_publicacion()
    {
        $user = new \App\Usuario(['id' => '1']);
        auth()->login($user);
        $this->visit('form_registrar_publicacion');
    }

    public function test_agregar_nuevo_tipopublicacion()
    {
        $user = new \App\Usuario(['id' => '1']);
        auth()->login($user);
        $this->visit('form_registrar_publicacion')
            ->type('REGLAMENTO', 'tipopublicaciones_titulo')
            ->type('1', 'tipopublicaciones_usuarios_id')
            ->press('agregar_categoriapublicaicon');
    }
}
```

```
Migrated:  2018_04_13_182637_crear_tabla_cliente
Migrating: 2018_04_14_045233_crear_tabla_procesos
Migrated:  2018_04_14_045233_crear_tabla_procesos
Migrating: 2018_04_14_045251_crear_tabla_actividades
Migrated:  2018_04_14_045251_crear_tabla_actividades
Migrating: 2018_04_14_045311_crear_tabla_tipopublicaciones
Migrated:  2018_04_14_045311_crear_tabla_tipopublicaciones

C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

..................                                  18 / 18 (100%)

Time: 1.01 seconds, Memory: 14.00MB

OK (18 tests, 27 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 69: Test de pruebas HU03-03 – Registrar Tipos de Publicaicones - OK

Fuente: Elaboración Propia

Imagen 70: HU03-03 - Formulario Agregar Categoría de Publicación

Fuente: Elaboración Propia

En la Imagen 70 se puede ver, el formulario terminado de agregar nueva categoría de publicación.

4.6.4. Product Backlog – HU04

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU04		**El operador de caja necesita un módulo para registrar sus operaciones realizadas durante el día.**	**1**
	HU04-01	Como usuario operador de caja, quiero poder registrar todas mis transacciones realizadas durante el día. Para llevar un control diario de mis procesos. Condiciones: - El formulario debe de tener la posibilidad de realizar una búsqueda indexada de los socios registrados, para realizar la operación lo más rápida posible. - En la vista del formulario, debe tener un gráfico estadístico de mis operaciones realizadas en caja.	1.1

Tabla 8: Product Backlog - HU03

Fuente: Elaboración Propia

*Crear el archivo que contendrá los test de pruebas de las operaciones de caja.

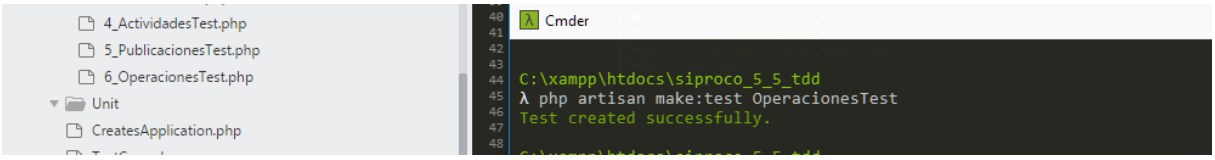


Imagen 71: Creación del Archivo OperacionesTest

Fuente: Elaboración Propia

Imagen 72: Tabla Operaciones desde Laravel

Fuente: Elaboración Propia

Se crea la tabla operaciones desde laravel, y se ejecuta la migración al gestor de base de datos.

Imagen 73: Tabla Operaciones en MySQL

Fuente: Elaboración Propia

Se puede apreciar que la tabla “Operaciones” en MySQL desde la ejecución de migración hecha en Laravel.

4.6.4.1.Test de Prueba del Product Backlog HU04-01 – Registrar Operación

<table>
<tr><td>Nombre de proceso:</td><td colspan="2">Registrar Operación</td><td>Id backlog:</td><td>HU04-01</td></tr>
<tr><td></td><td></td><td></td><td>Id historia de usuario:</td><td>HU04</td></tr>
<tr><td>Autor de caso de prueba:</td><td colspan="4">Operador de caja</td></tr>
<tr><td colspan="3">Test de prueba</td><td colspan="2">Especificación</td></tr>
<tr><td colspan="3"><pre>public function test_registrar_operacion()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('form_registrar_operacion')
        ->select('12345678', 'operaciones_clientes_dni')
        ->type('1', 'operaciones_procesos_id')
        ->type('1000', 'operaciones_monto')
        ->type('ninguna', 'operaciones_nota')
        ->press('agregar_operacion');
}</pre></td><td colspan="2">Al registrar la operación, los usuarios que interactúan con el sistema tienen acceso a la información en tiempo real.</td></tr>
<tr><td>Dato adicional:</td><td colspan="4">Registrar la operación, permitirá al administrador determinar futuras promociones en créditos y/o ahorros. Además de proyectar más personal según la hora y tiempos de transacciones.</td></tr>
</table>

Se crea el test de pruebas test_registrar_operacion, donde ingresaremos en la prueba los siguientes datos:

1. Autentificamos al usuario, que hará el registro de la operación en la caja.
2. Ir a la vista *form_registrar_operacion.*
3. Ingresamos el dni del cliente. Cliente previamente registrado.
4. Seleccionamos el id del proceso. En este caso ya poseemos un id de un proceso anteriormente registrado. “1” proceso que de nombre es “COBRO”.
5. Ingresamos el monto.
6. Si deseamos agregamos una nota.
7. Indicamos a la prueba presionar el botón *agregar_operacion*

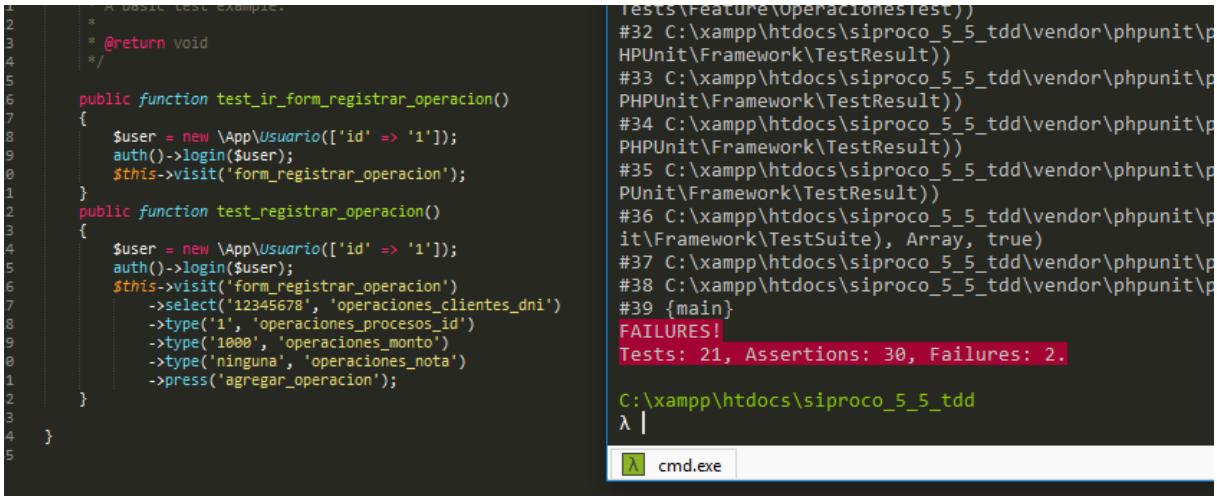


Imagen 74: Test de pruebas HU04-01 – Registrar Operaciones- FAILURE

Fuente: Elaboración Propia

El proceso falla, indicando que no encuentra la ruta del *form_registrar_operacion.*

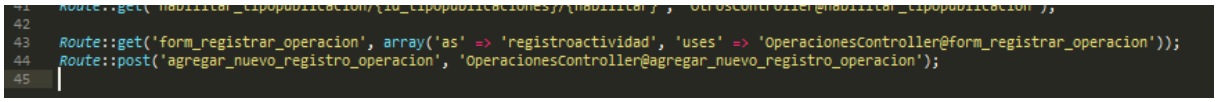


Imagen 75: Ruta del formulario para registrar la operación

Fuente: Elaboración Propia

Creamos el método agregar_nuevo_registro_operacion.

Creamos el método agregar_nuevo_registro_operacion, donde digitaremos el código necesario para agregar el nuevo registro de operación.

```php
public function agregar_nuevo_registro_operacion(Request $request)
{

    $data = $request->all();

    $reglas = array(
        'operaciones_clientes_dni' => 'required|numeric',
        'operaciones_procesos_id'  => 'required|numeric',
        'operaciones_monto'        => 'required|numeric',
        'operaciones_nota'         => 'required|alpha',

    );
    $mensajes = array(
        'operaciones_clientes_dni.required' => 'Es necesario especificar al cliente',
        'operaciones_clientes_dni.numeric'  => 'Ingrese el cliente que realizara esta operación',
        'operaciones_procesos_id.required'  => 'Es necesario especificar el proceso',
        'operaciones_procesos_id.numeric'   => 'Seleccione el tipo de proceso que el cliente realizará',
        'operaciones_monto.required'        => 'Es necesario ingresar un monto',
        'operaciones_monto.numeric'         => 'El monto solo puede contener números',
        'operaciones_nota.required'         => 'Es necesario ingresar una nota',
        'operaciones_nota.alpha'            => 'La nota no puede contener números',
    );

    $validacion = Validator::make($data, $reglas, $mensajes);
    if ($validacion->fails()) {
        $errores = $validacion->errors();
        return new JsonResponse($errores, 422);
    }

    $date  = Carbon::now();
    $fecha = $date->toDateString();
    $hora  = $date->toTimeString();

    $operacion                          = new Operacion;
    $operacion->operaciones_usuarios_id  = $data["operaciones_usuarios_id"];
    $operacion->operaciones_fecha        = $fecha;
    $operacion->operaciones_hora         = $hora;
    $operacion->operaciones_clientes_dni = $data["operaciones_clientes_dni"];
    $operacion->operaciones_procesos_id  = $data["operaciones_procesos_id"];
    $operacion->operaciones_monto        = $data["operaciones_monto"];
    $operacion->operaciones_nota         = $data["operaciones_nota"];
    $operacion->operaciones_habilitado   = 's';

    $agregar_nuevo_registro_operacion = $operacion->save();
    if ($agregar_nuevo_registro_operacion) {

        return view("intranet.mensajes.msj_correcto")->with("msj", "Operación registrada correctamente");
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
    }

}
```

Imagen 76: Método agregar_nuevo_registro_operacion dentro del controlador OperacionesController

Fuente: Elaboración Propia

Ejecutamos el test de pruebas, y observamos que la prueba ha sido superada. Hasta el momento, hay 21 test superados con éxito.

```php
public function test_ir_form_registrar_operacion()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('form_registrar_operacion');
}
public function test_registrar_operacion()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('form_registrar_operacion')
        ->select('12345678', 'operaciones_clientes_dni')
        ->type('1', 'operaciones_procesos_id')
        ->type('1000', 'operaciones_monto')
        ->type('ninguna', 'operaciones_nota')
        ->press('agregar_operacion');
}
```

```
C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

.....................                                          21 / 21 (100%)

Time: 1.62 seconds, Memory: 14.00MB

OK (21 tests, 31 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 77: Test de pruebas HU04-01 – Registrar Operaciones - OK

Fuente: Elaboración Propia

Imagen 78: HU04-01 - Formulario Registrar Operación

Fuente: Elaboración Propia

En la imagen 78, se puede ver el formulario registrar operación ya concluido.

4.6.5. Product Backlog – HU05

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU05		**El analista de créditos necesita un módulo para registrar sus actividades realizas durante el día.**	**1**
	HU05-01	Como usuario analista de creditos, quiero poder registrar todas mis actividades realizadas durante el día. Para llevar un control diario de mis acciones. Condiciones: - El formulario debe de tener la posibilidad de realizar una búsqueda indexada de los socios registrados, para realizar el registro lo más rápido posible. - En la vista del formulario, debe tener un gráfico estadístico de mis operaciones realizadas en caja.	1.1

Tabla 9: Product Backlog - HU05

Fuente: Elaboración Propia

*Crear el archivo que contendrá los test de pruebas de las actividades que realizará el analista de créditos.

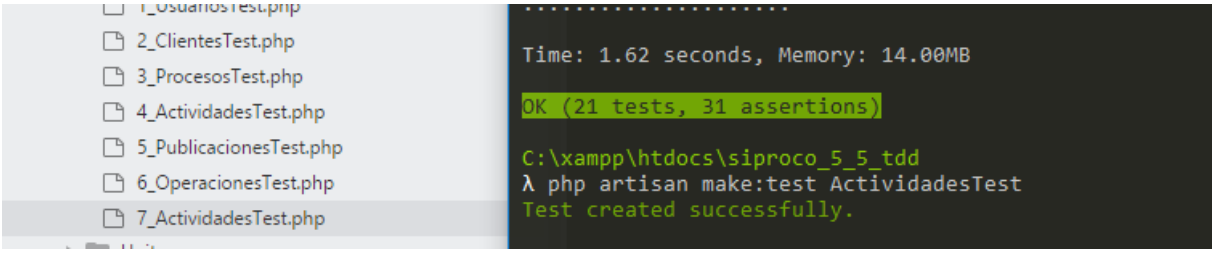


Imagen 79: Creación del Archivo ActividadesTest

Fuente: Elaboración Propia

Imagen 80: Tabla Acciones desde Laravel

Fuente: Elaboración Propia

Se crea la tabla acciones desde laravel, y se ejecuta la migración al gestor de base de datos.

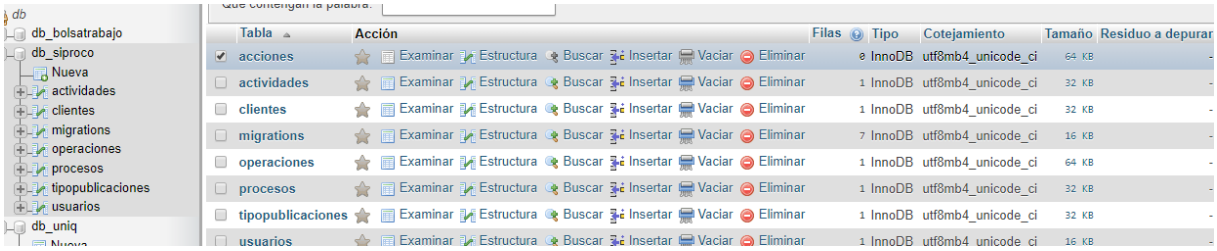


Imagen 81: Tabla Acciones en MySQL

Fuente: Elaboración Propia

Se puede ver que la tabla “Acciones” en MySQL desde la ejecución de migración hecha en Laravel.

4.6.5.1.Test de Prueba del Product Backlog HU05-01 – Registrar Acciones

<table>
<tr><td>Nombre de proceso:</td><td colspan="2">Registrar Acciones</td><td>Id backlog:</td><td>HU05-01</td></tr>
<tr><td></td><td colspan="2"></td><td>Id historia de usuario:</td><td>HU05</td></tr>
<tr><td>Autor de caso de prueba:</td><td colspan="4">Analista de Créditos</td></tr>
<tr><td colspan="2">Test de prueba</td><td colspan="3">Especificación</td></tr>
<tr><td colspan="2"><pre>public function test_registrar_accion()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('form_registrar_accion')
->type('1', 'acciones_usuarios_id')
->select('12345678', 'acciones_clientes_dni')
->type('1', 'acciones_actividades_id')
->type('10000', 'acciones_monto')
->type('prestamo', 'acciones_nota')
->press('guardar_accion');
}</pre></td><td colspan="3">Al registrar las actividades del analista de créditos, se pueden mejorar la atención directa con el socio en función a los productos crediticios.</td></tr>
<tr><td>Dato adicional:</td><td colspan="4">Registrar la acción, permitirá al administrador determinar futuras promociones en créditos y/o ahorros. Además de proyectar más personal según la hora y tiempos determinados para cada acción.</td></tr>
</table>

Se crea el test de pruebas test_registrar_accion, donde le diremos a la prueba ingresar los siguientes datos:

1. Autenticar al usuario “1”, para el registro de la acción.
2. Ir a la vista *form_registrar_accion.*
3. Le asignamos “1” como id de usuario al campo *clientes_usuarios_id* porque “1” es el id del usuario que está registrando esta accion.
4. Ingresamos datos del cliente: dni, identificamos que actividad realizara en este caso “1” por que es el id de la actividad, el monto solo si es necesario y una nota adicional si lo amerita.
5. Al termino la prueba deberá presionar el botón *guardar_accion.*

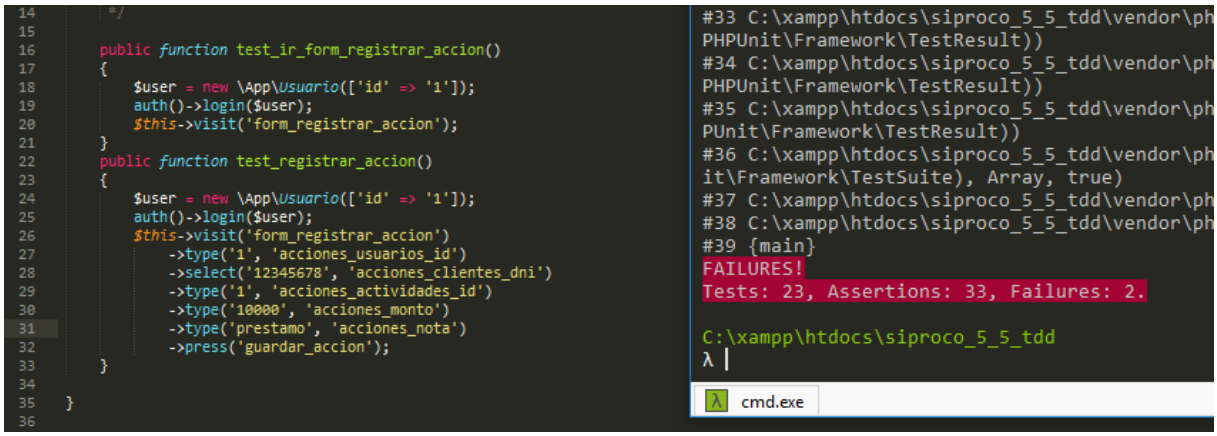


Imagen 82: Test de pruebas HU05-01 – Registrar Acciones- FAILURE

Fuente: Elaboración Propia

Creamos la ruta que direccionará a la vista del formulario y el método que cumplirá la función de registrar el dato.

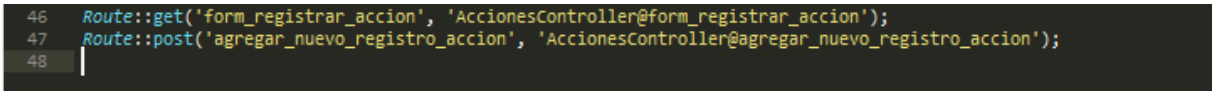
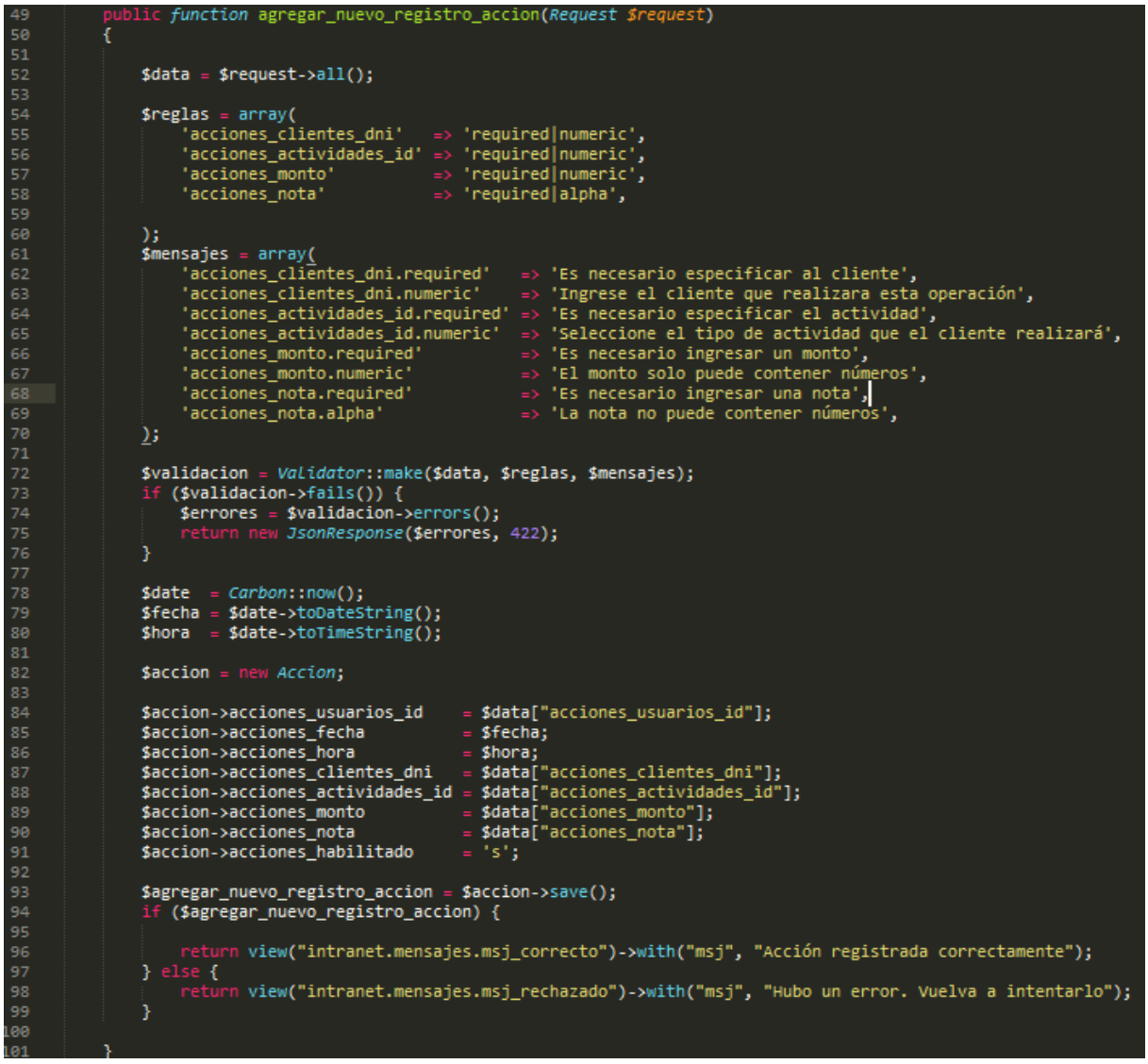


Imagen 83: Ruta del formulario para registrar la accion

Fuente: Elaboración Propia

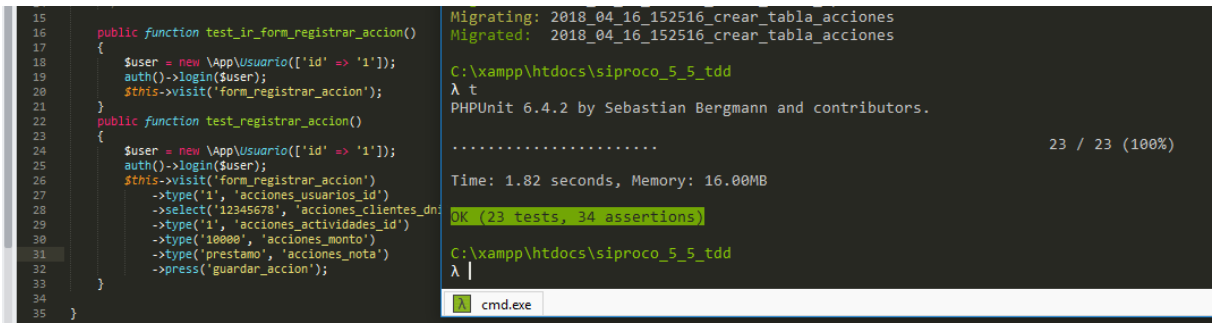
Creamos el contenido del método que permitirá agregar el registro de la acción *agregar_nuevo_registro_accion.*



```php
public function agregar_nuevo_registro_accion(Request $request)
{

    $data = $request->all();

    $reglas = array(
        'acciones_clientes_dni'    => 'required|numeric',
        'acciones_actividades_id' => 'required|numeric',
        'acciones_monto'           => 'required|numeric',
        'acciones_nota'            => 'required|alpha',

    );
    $mensajes = array(
        'acciones_clientes_dni.required'    => 'Es necesario especificar al cliente',
        'acciones_clientes_dni.numeric'     => 'Ingrese el cliente que realizara esta operación',
        'acciones_actividades_id.required' => 'Es necesario especificar el actividad',
        'acciones_actividades_id.numeric'  => 'Seleccione el tipo de actividad que el cliente realizará',
        'acciones_monto.required'           => 'Es necesario ingresar un monto',
        'acciones_monto.numeric'            => 'El monto solo puede contener números',
        'acciones_nota.required'            => 'Es necesario ingresar una nota',
        'acciones_nota.alpha'               => 'La nota no puede contener números',
    );

    $validacion = Validator::make($data, $reglas, $mensajes);
    if ($validacion->fails()) {
        $errores = $validacion->errors();
        return new JsonResponse($errores, 422);
    }

    $date  = Carbon::now();
    $fecha = $date->toDateString();
    $hora  = $date->toTimeString();

    $accion = new Accion;

    $accion->acciones_usuarios_id    = $data["acciones_usuarios_id"];
    $accion->acciones_fecha          = $fecha;
    $accion->acciones_hora           = $hora;
    $accion->acciones_clientes_dni   = $data["acciones_clientes_dni"];
    $accion->acciones_actividades_id = $data["acciones_actividades_id"];
    $accion->acciones_monto          = $data["acciones_monto"];
    $accion->acciones_nota           = $data["acciones_nota"];
    $accion->acciones_habilitado     = 's';

    $agregar_nuevo_registro_accion = $accion->save();
    if ($agregar_nuevo_registro_accion) {

        return view("intranet.mensajes.msj_correcto")->with("msj", "Acción registrada correctamente");
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
    }

}
```

Imagen 84: Método agregar_nuevo_registro_accion dentro del controlador AccionesController

Fuente: Elaboración Propia



```php
public function test_ir_form_registrar_accion()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('form_registrar_accion');
}
public function test_registrar_accion()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('form_registrar_accion')
        ->type('1', 'acciones_usuarios_id')
        ->select('12345678', 'acciones_clientes_dni')
        ->type('1', 'acciones_actividades_id')
        ->type('10000', 'acciones_monto')
        ->type('prestamo', 'acciones_nota')
        ->press('guardar_accion');
}
```

```
Migrating: 2018_04_16_152516_crear_tabla_acciones
Migrated:  2018_04_16_152516_crear_tabla_acciones

C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

.......................                                           23 / 23 (100%)

Time: 1.82 seconds, Memory: 16.00MB

OK (23 tests, 34 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 85: Test de pruebas HU05-01 – Registrar Acciones - OK

Fuente: Elaboración Propia

El test de pruebas test_registrar_accion ha sido ejecutado y como resultado a superado la prueba.

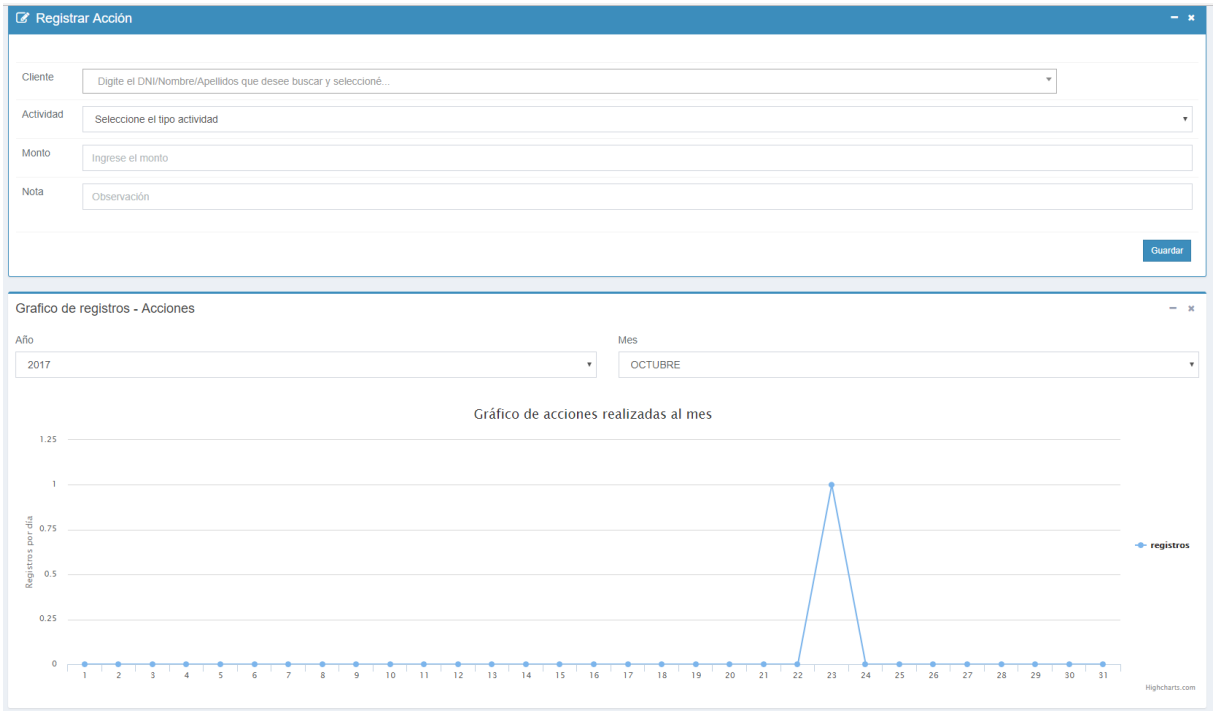


Imagen 86: HU05-01 - Formulario Registrar Actividad

Fuente: Elaboración Propia

En la imagen 86, se puede ver el formulario culminado.

4.6.6. Product Backlog – HU06

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU06		**El analista de créditos necesita información del registro del operador de caja, para programar sus labores de cobro y otros durante el día.**	**1**
	HU06-01	Como usuario analista de créditos, quiero poder ver todas las operaciones registradas por los operadores de caja. Para programar mis acciones durante el día. Condiciones: - Debe de ser un listado, ordenado de forma descendente y con posibilidad de filtrar por dni, nombre, apellido paterno, apellido materno.	1.1

Tabla 10: Product Backlog - HU06

Fuente: Elaboración Propia

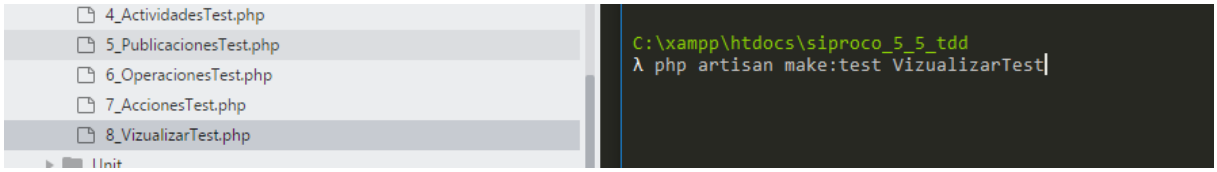


Imagen 87: Creación del Archivo VizualizarTest

Fuente: Elaboración Propia

4.6.6.1.Test de Prueba del Product Backlog HU06-01 –Listar Operaciones

Nombre de proceso:	Listar Operaciones	Id backlog:	HU06-01
		Id historia de usuario:	HU06
Autor de caso de prueba:	Administrador, Operador de Caja, Analista de Créditos		
Test de prueba		Especificación	

public function test_ir_form_listado_operaciones_globales()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('listado_operaciones_globales');
}
		Permite ver el registro de todos los operadores de caja, con el fin de obtener información verídica valida para el analista de créditos.	

Dato adicional:			

Creamos el test de pruebas *test_ir_form_listado_operaciones_globales*, el cual nos mostrar las operaciones ingresadas por todos los operadores de caja. Para lo cual le pedimos al test de pruebas ingresar los siguientes datos:

1. Autenticar al usuario, en este caso el usuario predefinido con id "1".
2. Le indicamos a la prueba dirigirse a la vista *listado_operaciones_globales.*

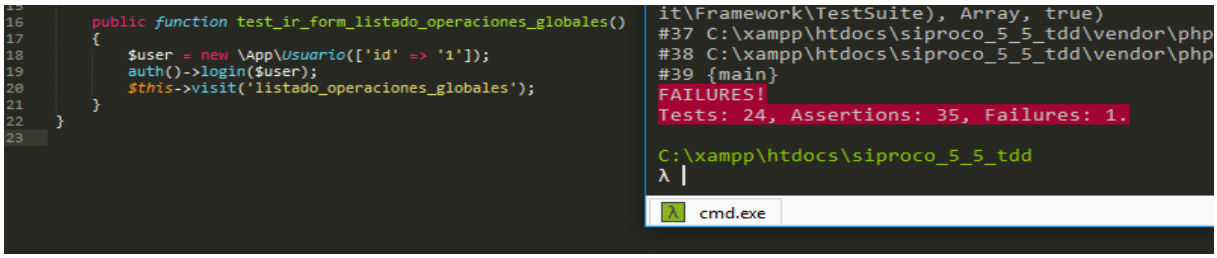


Imagen 88: Test de pruebas HU06-01 – Listado Operaciones- FAILURE

Fuente: Elaboración Propia

Se direcciona la ruta para que el formulario se pueda presentar a la solicitud.

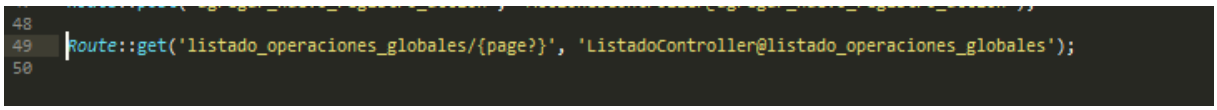


Imagen 89: Ruta del formulario para ver las operaciones registradas

Fuente: Elaboración Propia

Se crea le método listar_operaciones_globales, para lo cual realizamos la consulta por medio de EloquentOrm.

```
34      }
35
36      public function listado_operaciones_globales()
37      {
38          $operaciones = Operacion::orderBy('operaciones_id', 'DESC')->paginate(50);
39          $procesos    = Proceso::all();
40          return view('intranet.formularios.listado_operaciones_globales')
41              ->with("procesos", $procesos)
42              ->with("operaciones", $operaciones);
43      }
44
```

Imagen 90:Método listado_operaciones_globales dentro del controlador ListadoController

Fuente: Elaboración Propia

Ejecutamos el test de pruebas, que resulta prueba superada.

```
11      * A basic test example.
12      *
13      * @return void
14      */
15
16     public function test_ir_form_listado_operaciones_globales()
17     {
18         $user = new \App\Usuario(['id' => '1']);
19         auth()->login($user);
20         $this->visit('listado_operaciones_globales');
21     }
22 }
23
```

```
C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

.......................                                    24 / 24 (100%)

Time: 1.74 seconds, Memory: 16.00MB

OK (24 tests, 35 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
```

Imagen 91: Test de pruebas HU06-01 – Listado Operaciones - OK

Fuente: Elaboración Propia

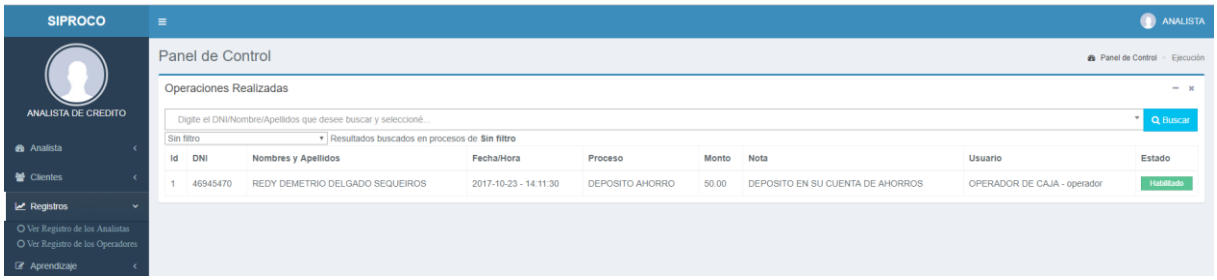


Imagen 92: HU06-01 - Formulario Vista de Operaciones Realizadas

Fuente: Elaboración Propia

En la imagen 92, se aprecia el formulario de la lista de operaciones globales ya culminado.

4.6.7. Product Backlog – HU07

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU07		**El operador de caja necesita información diaria del total de ingreso y salida de dinero, para realizar su arqueo de caja correspondiente al cierre del día.**	**1**
	HU07-01	Como usuario operador de caja, quiero poder ver el resumen de todas las operaciones realizadas durante el día. Para realizar mi arqueo de caja, y verificar mis balances diarios. Condiciones: - El resumen debe ser sencillo y entendible.	1.1

Tabla 11: Product Backlog - HU07

Fuente: Elaboración Propia

*Dentro del mismo archivo de VisualizarTest, creamos el test de pruebas para ver, el movimiento de caja.

4.6.7.1.Test de Prueba del Product Backlog HU07-01 – Mostrar Movimiento de Caja

Nombre de proceso:	Mostrar Movimiento de Caja	Id backlog:	HU07-01
		Id historia de usuario:	HU07
Autor de caso de prueba:	Operador de Caja		
Test de prueba		Especificación	

public function test_ir_form_arqueo_caja()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('arqueo_caja/1');
}
		Permite clasificar el tipo de operación registrada durante el día, si es a favor o en contra de la cooperativa.	

Dato adicional:	Es de gran ayuda al para el cierre de caja diario.		

Creamos el test de pruebas *test_ir_form_arqueo_caja*, el cual nos mostrara los movimientos realizados durante el día. Por lo cual al test de pruebas le enviamos:

1. Autenticar al usuario, en este caso el usuario predefinido con id "1".
2. Le indicamos a la prueba dirigirse a la vista *arqueo_caja/1, el parámetro de control es "1" por ser el id del usuario.*

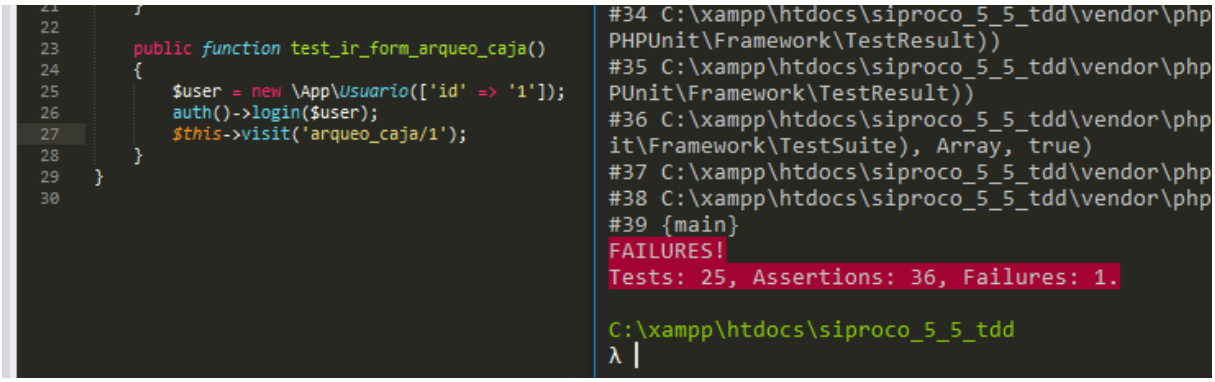


Imagen 93: Test de pruebas HU07-01 – Visualizar Movimiento de Caja- FAILURE

Fuente: Elaboración Propia

Una vez más la prueba falla, indicándonos que no encuentra la ruta del formulario. Creamos la ruta y el método que permita mostrar la información solicitada.

```
50
51  Route::get('arqueo_caja/{id_usuario}/{page?}', 'ListadoController@arqueo_caja');
52  Route::get('arqueo_caja_fecha/{id_usuario}/{fecha}/{page?}', 'ListadoController@arqueo_caja_fecha');
53
```

Imagen 94: Ruta del formulario para ver el movimiento de caja

Fuente: Elaboración Propia

Este método es complejo, porque tiene que identificar variables de tipo negativo y positivo y al final deberá enviar los datos a través de variables.

```
public function arqueo_caja_fecha($id_usuario, $fecha)
{
    $usuario       = \Auth::user();
    $arqueonegativo = DB::table('operaciones')
        ->join('procesos', 'operaciones.operaciones_procesos_id', '=', 'procesos.procesos_id')
        ->where('operaciones.operaciones_usuarios_id', '=', $id_usuario)
        ->where('operaciones.operaciones_fecha', '=', $fecha)
        ->where('procesos.procesos_tipoproceso', '=', 'NEGATIVO')
        ->where('operaciones.operaciones_habilitado', '=', 's')
        ->select('procesos.procesos_proceso as proceso', DB::raw('SUM(operaciones.operaciones_monto) as monto'), DB::raw('COUNT(operaciones.operaciones_monto) as movimientos'),
            operaciones.operaciones_fecha as fecha')
        ->groupBy('operaciones.operaciones_procesos_id', 'procesos.procesos_proceso', 'operaciones.operaciones_fecha')
        ->get();

    $sumanegativo = DB::table('operaciones')
        ->join('procesos', 'operaciones.operaciones_procesos_id', '=', 'procesos.procesos_id')
        ->where('operaciones.operaciones_usuarios_id', '=', $id_usuario)
        ->where('operaciones.operaciones_fecha', '=', $fecha)
        ->where('procesos.procesos_tipoproceso', '=', 'NEGATIVO')
        ->where('operaciones.operaciones_habilitado', '=', 's')
        ->select(DB::raw('SUM(operaciones.operaciones_monto) as monto'))
        ->get();

    $arqueopositivo = DB::table('operaciones')
        ->join('procesos', 'operaciones.operaciones_procesos_id', '=', 'procesos.procesos_id')
        ->where('operaciones.operaciones_usuarios_id', '=', $id_usuario)
        ->where('operaciones.operaciones_fecha', '=', $fecha)
        ->where('procesos.procesos_tipoproceso', '=', 'POSITIVO')
        ->where('operaciones.operaciones_habilitado', '=', 's')
        ->select('procesos.procesos_proceso as proceso', DB::raw('SUM(operaciones.operaciones_monto) as monto'), DB::raw('COUNT(operaciones.operaciones_monto) as movimientos'),
            operaciones.operaciones_fecha as fecha')
        ->groupBy('operaciones.operaciones_procesos_id', 'procesos.procesos_proceso', 'operaciones.operaciones_fecha')
        ->get();

    $sumapositivo = DB::table('operaciones')
        ->join('procesos', 'operaciones.operaciones_procesos_id', '=', 'procesos.procesos_id')
        ->where('operaciones.operaciones_usuarios_id', '=', $id_usuario)
        ->where('operaciones.operaciones_fecha', '=', $fecha)
        ->where('procesos.procesos_tipoproceso', '=', 'POSITIVO')
        ->where('operaciones.operaciones_habilitado', '=', 's')
        ->select(DB::raw('SUM(operaciones.operaciones_monto) as monto'))
        ->get();

    return view('intranet.formularios.listados.listado_arqueo_caja')
        ->with("arqueonegativo", $arqueonegativo)
        ->with("arqueopositivo", $arqueopositivo)
        ->with("sumanegativo", $sumanegativo)
        ->with("sumapositivo", $sumapositivo)
        ->with("usuario", $usuario);
}
```

Imagen 95:Método arqueo_caja dentro del controlador ListadoController

Fuente: Elaboración Propia

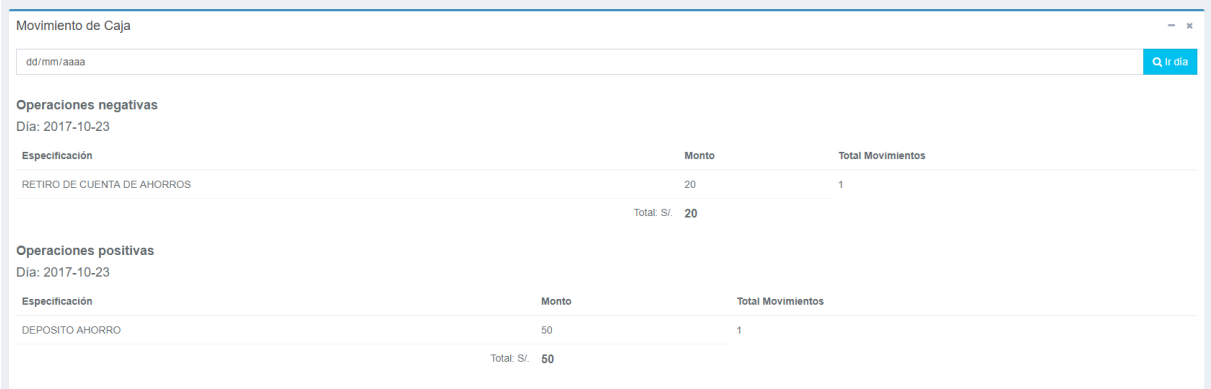


Imagen 96: HU07-01 - Formulario Movimiento de Caja

Fuente: Elaboración Propia

En la imagen 96, se puede ver el formulario concluido.

4.6.8. Product Backlog – HU08

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU08		**El administrador necesita información de las actividades realizadas durante el día de cada analista de crédito, para el seguimiento preciso de sus labores programadas.**	**1**
	HU08-01	Como usuario administrador, quiero poder ver las actividades realizadas durante el día de cada analista de crédito. Para llevar un control sobre el personal un poco más a detalle sobre su rendimiento diario. Condiciones: - La vista de las actividades realizadas diarias, debe tener posibilidad de filtrar por dni, nombre, apellido paterno, apellido materno.	1.1

Tabla 12: Product Backlog - HU08

Fuente: Elaboración Propia

*Dentro del mismo archivo de VisualizarTest, creamos el test de pruebas para ver, las acciones realizadas durante el día por el analista de créditos.

4.6.8.1.Test de Prueba del Product Backlog HU08-01 –Listar Acciones

Nombre de proceso:	Listar Acciones	Id backlog:	HU08-01
		Id historia de usuario:	HU08
Autor de caso de prueba:	Administrador, Operador de Caja, Analista de creditos		
Test de prueba		Especificación	

public function test_ir_form_listado_acciones_globales()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('listado_acciones_globales');
}
		Permite ver el registro de las acciones realizadas por los analistas de créditos, con el fin de ver su rendimiento laboral.	

Dato adicional:	--		

Creamos el test de pruebas *test_ir_form_listado_acciones_globales*, el cual va a mostrar las acciones registradas por todos los analistas de creditos. Para lo cual le pedimos al test de pruebas ejecutar las siguientes ordenes:

1. Autenticar al usuario, en este caso el usuario predefinido con id “1”.
2. Le indicamos a la prueba dirigirse a la vista *listado_ acciones_globales.*

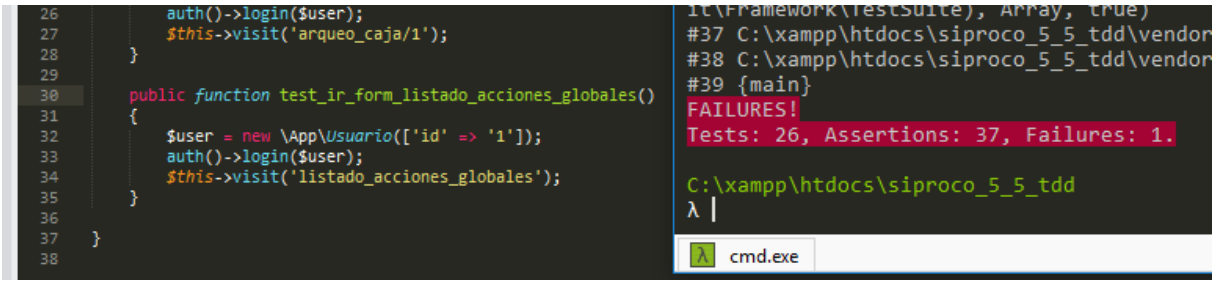


Imagen 97: Test de pruebas HU08-01 – Listado de Acciones - FAILURE

Fuente: Elaboración Propia

Nos arroja el error, de ruta no encontrada y por ende método.

```
54
55  Route::get('listado_acciones_globales/{page?}', 'ListadoController@listado_acciones_globales');
56
```

Imagen 98: Ruta del formulario para ver las acciones realizadas

Fuente: Elaboración Propia

Creamos el método que permita listar las acciones de todos los analistas de créditos.

```
179     public function listado_acciones_globales()
180     {
181         $acciones     = Accion::orderBy('acciones_id', 'DESC')->paginate(50);
182         $actividades = Actividad::all();
183         return view('intranet.formularios.listado_acciones_globales')
184             ->with("actividades", $actividades)
185             ->with("acciones", $acciones);
186     }
187
```

Imagen 99:Método listado_acciones_globales dentro del controlador ListadoController

Fuente: Elaboración Propia

```
29
30      public function test_ir_form_listado_acciones_globales()
31      {
32          $user = new \App\Usuario(['id' => '1']);
33          auth()->login($user);
34          $this->visit('listado_acciones_globales');
35      }
36
37  }
38
```

```
C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

..........................                                        26 / 26 (100%)

Time: 1.64 seconds, Memory: 16.00MB

OK (26 tests, 37 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
```

Imagen 100: Test de pruebas HU08-01 – Listado de Acciones - OK

Fuente: Elaboración Propia

Ejecutamos el test de pruebas *test_ir_form_listado_acciones_globales* con éxito.

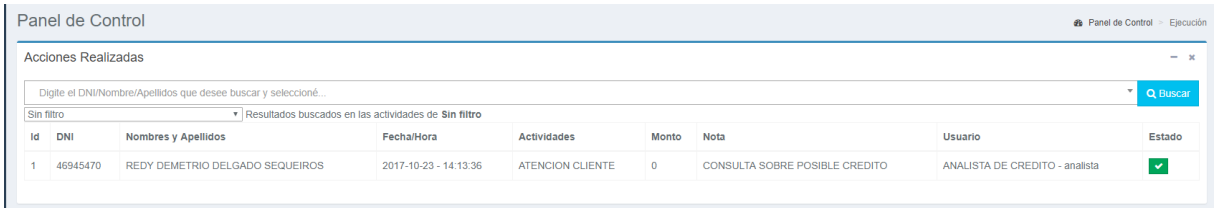


Imagen 101: HU08-01 - Formulario Acciones Realizadas

Fuente: Elaboración Propia

En la imagen 101 podemos observar, el formulario listado de acciones globales concluido.

4.6.9. Product Backlog – HU09

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU09		**El administrador necesita información consolidada de las actividades del analista de crédito y operador de caja, para la toma de decisiones sobre el rendimiento de los colaboradores y futuros cambios.**	**1**
	HU09-01	Como usuario administrador, quiero poder ver el resumen consolidado de las acciones realizadas durante el mes, de cada analista de crédito y con comparación entre todos los analistas de créditos. Para proyectarme en la planificación del próximo año, diciendo así cuales puntos he de reforzar para su mejora continua. Condiciones: - La vista tienen que ser una sola, con diagramas estadísticos y filtro por usuario, además deberá contener un resumen numérico de sus acciones realizadas	1.1
	HU09-02	Como usuario administrador, quiero poder ver el resumen consolidado de las acciones realizadas durante el mes, de cada operador de caja y con comparación entre todos los operadores de caja. Para proyectarme en la planificación del próximo año,	1.2

		diciendo así cuales puntos he de reforzar para su mejora continua. Condiciones: - La vista tienen que ser una sola, con diagramas estadísticos y filtro por usuario, además deberá contener un resumen numérico de sus operaciones realizadas.	

Tabla 13: Product Backlog - HU09

Fuente: Elaboración Propia

*Dentro del mismo archivo de VisualizarTest, creamos el test de pruebas para ver, su índice de rendimiento de los analistas de créditos; comparándolos unos con otros.

4.6.9.1.Test de Prueba del Product Backlog HU09-01 – Seguimiento Labor Analista

Nombre de proceso:	Seguimiento Labor Analista	Id backlog:	HU09-01
		Id historia de usuario:	HU09
Autor de caso de prueba:	Administrador		
Test de prueba		Especificación	

public function test_ir_form_seguimiento_labor_analista()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('seguimiento_labor_analista');
}
		Permite ver su rendimiento mensual del analista de créditos.	

Dato adicional:	--		

Creamos el test de pruebas *test_ir_form_seguimiento_labor_analista*, que permitirá observar las actividades realizadas por el analista de créditos.

1. Autenticar al usuario, en este caso el usuario predefinido con id “1”.
2. Le indicamos a la prueba dirigirse a la vista *seguimiento_labor_analista.*

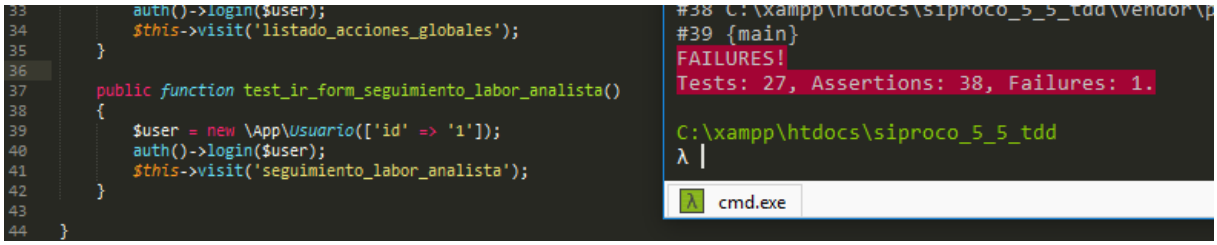


Imagen 102: Test de pruebas HU09-01 –Seguimiento Labor del Analista - FAILURE

Fuente: Elaboración Propia

La prueba falla, y procedemos a crear la ruta y el método.

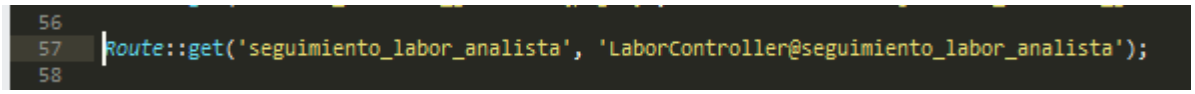


Imagen 103: Ruta del formulario para ver el resumen de rendimiento del analista de créditos

Fuente: Elaboración Propia

Imagen 104: Método seguimiento_labor_analista dentro del controlador LaborController

Fuente: Elaboración Propia

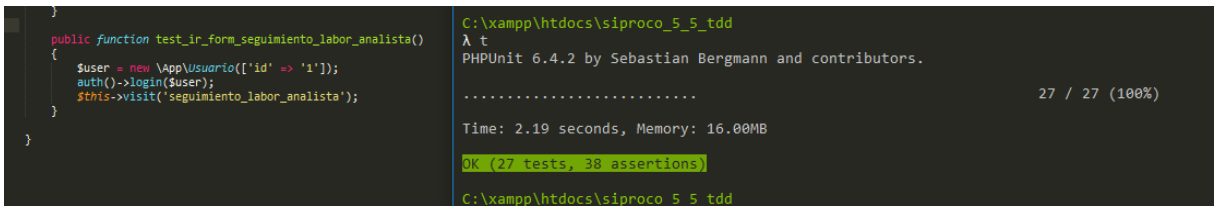


Imagen 105: Test de pruebas HU09-01 –Seguimiento Labor del Analista - OK

Fuente: Elaboración Propia

Ejecutamos el test de pruebas, y vemos que se ejecuta correctamente.

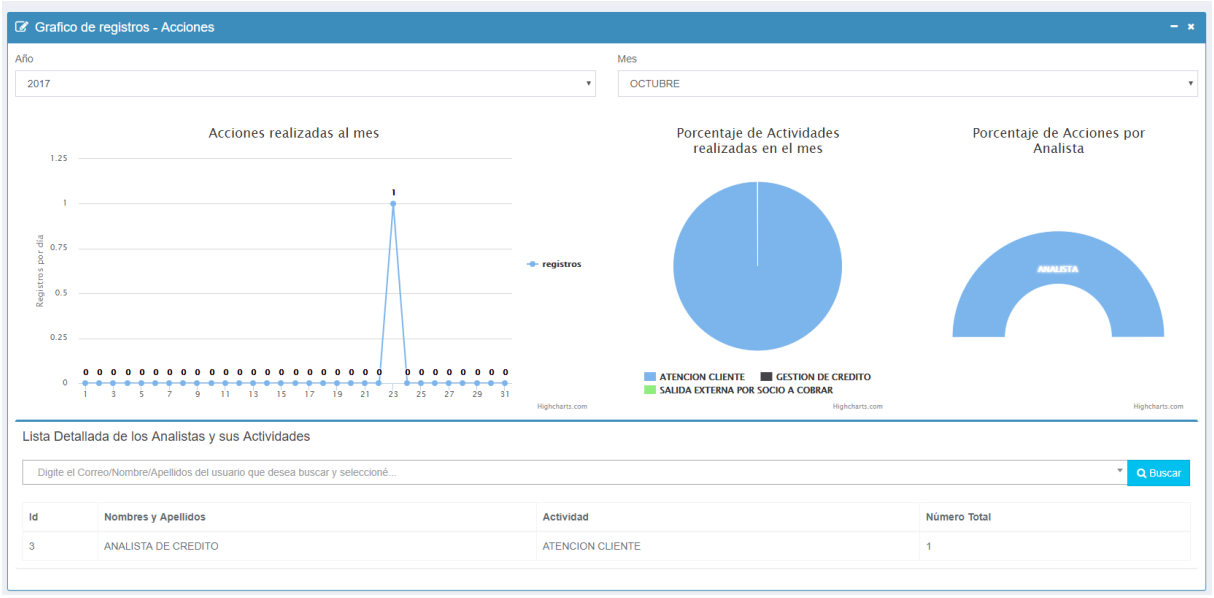


Imagen 106: HU09-01 - Formulario Resumen de Acciones

Fuente: Elaboración Propia

En la imagen 106, se puede observar el formulario resumen de acciones está concluido.

4.6.9.2.Test de Prueba del Product Backlog HU09-02 – Seguimiento Labor Operador

Nombre de proceso:	Seguimiento Labor Operador	Id backlog:	HU09-02
		Id historia de usuario:	HU09
Autor de caso de prueba:	Administrador		
Test de prueba		Especificación	

public function test_ir_form_seguimiento_labor_operador()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('seguimiento_labor_operador');
}
		Permite ver su rendimiento mensual del operador de caja.	

Dato adicional:	--		

Se crea el test de pruebas test_ir_form_seguimiento_labor_operador, para lo cual se ingresa en la prueba los siguientes parámetros.

1. Autenticar al usuario, en este caso el usuario predefinido con id “1”.
2. Le indicamos a la prueba dirigirse a la vista *seguimiento_labor_operador.*

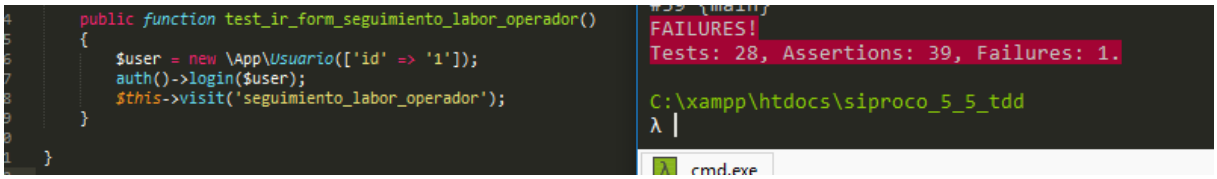


Imagen 107: Test de pruebas HU09-01 –Seguimiento Labor del Operador - FAILURE
Fuente: Elaboración Propia

Se observa que la ruta falla y se procede a direccionarlo creando el método que permita ejecutar la consulta y devolver los datos necesarios para mostrar la información.

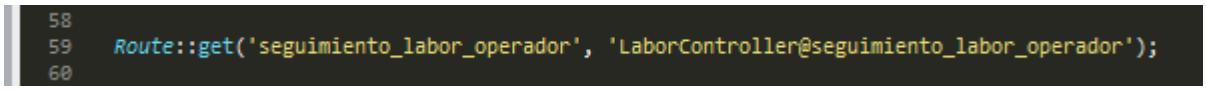


Imagen 108: Ruta del formulario para ver el resumen de rendimiento del operador de caja
Fuente: Elaboración Propia

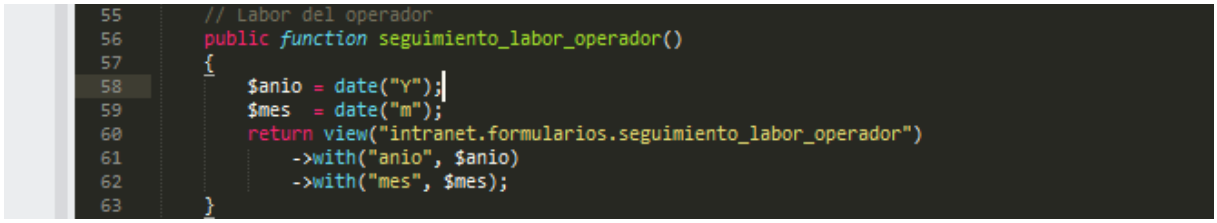


Imagen 109: Método seguimiento_labor_operador dentro del controlador LaborController
Fuente: Elaboración Propia

Ejecutamos el test de pruebas, el cual resulta exitoso.

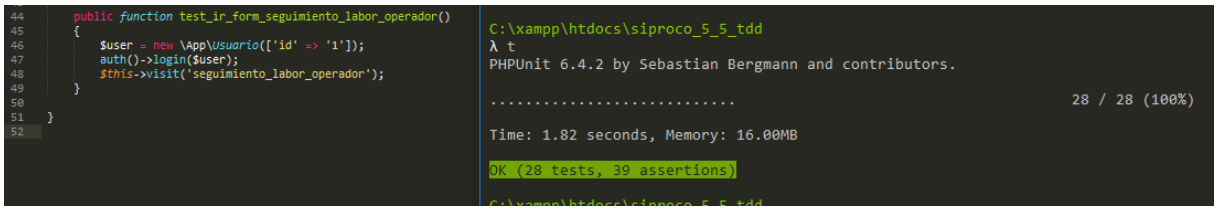


Imagen 110: Test de pruebas HU09-01 –Seguimiento Labor del Operador - OK

Fuente: Elaboración Propia

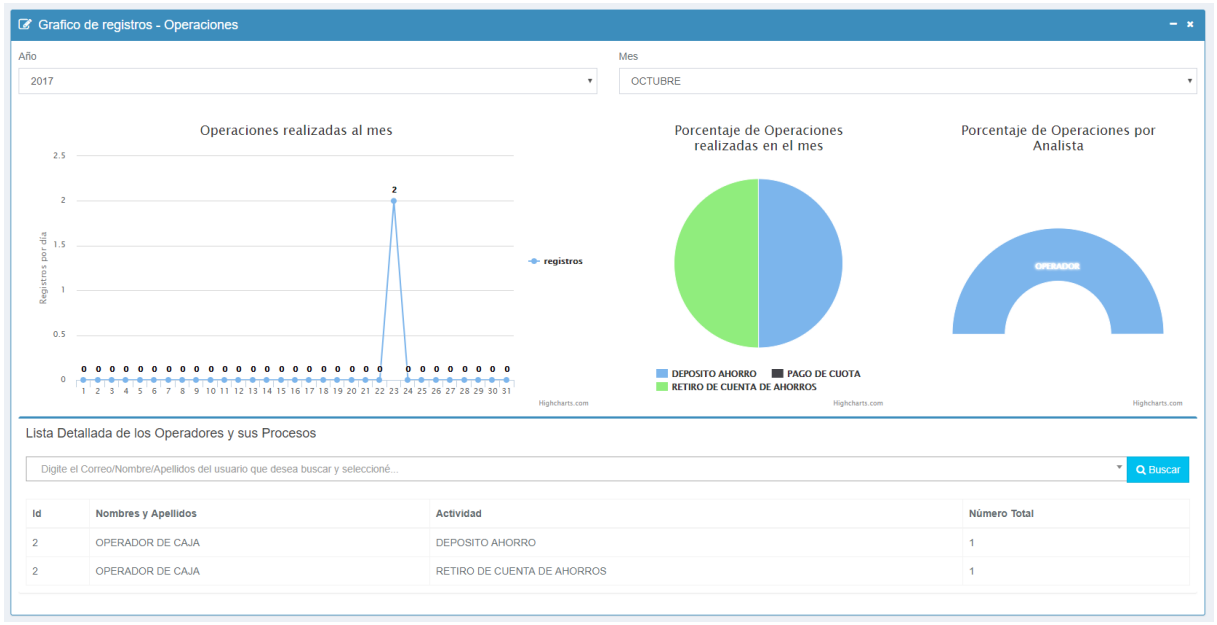


Imagen 111: HU09-02 - Formulario Resumen de Operaciones

Fuente: Elaboración Propia

En la imagen 111, podemos observar el formulario resumen de operaciones culminado.

Ya demostrado y explicado a detalle las pruebas procedemos a continuar con los siguientes test de pruebas, pero de forma más concisa. Por qué el proceso es semejante.

4.6.10. Product Backlog – HU10

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU10		**El administrador necesita un módulo donde se pueda subir archivos y crear exámenes para el control de aprendizaje de sus colaboradores, de tal forma controlar el estado de aprendizaje de las capacitaciones brindadas por la institución.**	**1**
	HU10-01	Como usuario administrador, quiero subir mis documentos institucionales al sistema pero que no se puedan descargar ni imprimir. Para reforzar las capacitaciones brindadas por la institución hacia los colaboradores de la institución. Condiciones: - El método de subir documentos debe ser intuitivo.	1.1
	HU10-02	Como usuario administrador, quiero crear mis propios exámenes, preguntas y asignar a los usuarios que puedan rendir tales exámenes. Para llevar un control del estado de aprendizaje de los colaboradores en cuanto a las capacitaciones brindadas por la institución Condiciones: - Cada examen creado debe tener una fecha de inicio y una fecha final, además de contener un tiempo límite de rendición de examen.	1.2

Tabla 14: Product Backlog - HU10

Fuente: Elaboración Propia

*Crear el archivo que contendrá los test de pruebas de las publicaciones de documentos.

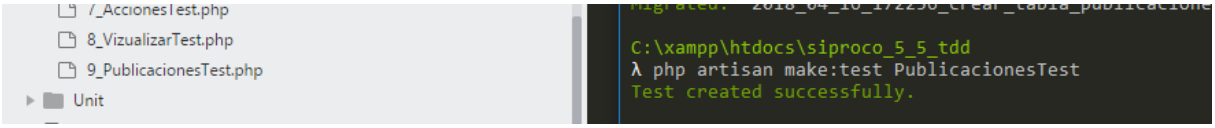


Imagen 112: Creación del Archivo PublicacionesTest

Fuente: Elaboración Propia

Imagen 113: Tabla Publicaciones desde Laravel

Fuente: Elaboración Propia

Se crea la tabla acciones desde laravel, y se ejecuta la migración al gestor de base de datos.

Imagen 114: Tabla Publicaciones en MySQL

Fuente: Elaboración Propia

4.6.10.1. Test de Prueba del Product Backlog HU10-01 – Publicar Documentos

Nombre de proceso:	Publicar Documentos	Id backlog:	HU10-01
		Id historia de usuario:	HU10
Autor de caso de prueba:	Administrador		

Test de prueba	Especificación

public function test_publicar_documentos()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('form_publicar_documentos/1')
->type('1', 'publicaciones_usuarios_id')
->type('1', 'publicaciones_tipopublicaciones_id')
->type('REGLAMENTO', 'publicaciones_titulo')
->type('Reglas', 'publicaciones_descripcion')
->type('C:\Users\ARDDS\Desktop\RHE10469454709E00140.pdf', 'file')
->press('agregar_documento');
}
	Este modulo sirve para subir el archivo al sistema para ser publicado y posteriormente para ser distribuido para los usuarios.

Dato adicional:	Se publica el material del cual se realizaran las preguntas, para el examen.

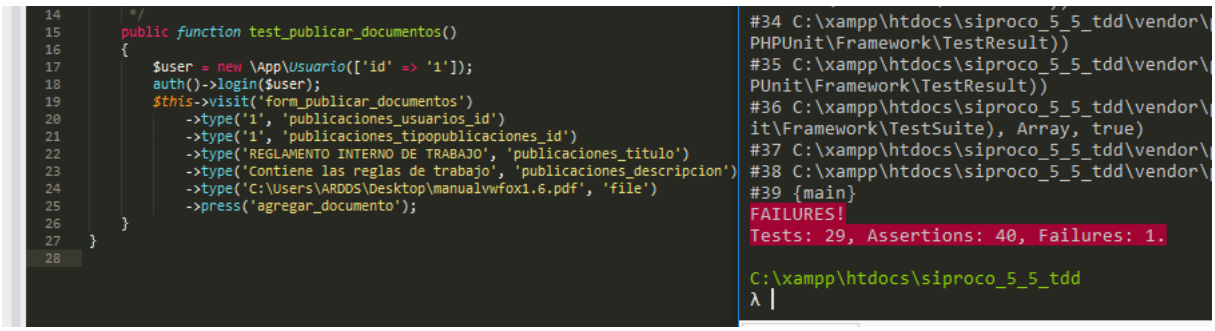


Imagen 115: Test de pruebas HU10-01 –Seguimiento Publicar Documentos - FAILURE

Fuente: Elaboración Propia

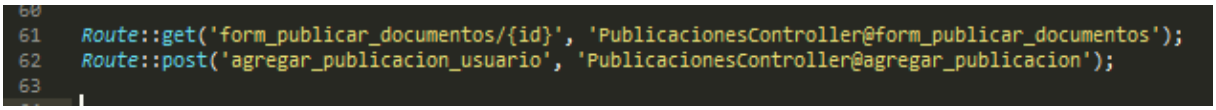


Imagen 116: Ruta del formulario para agregar documentos

Fuente: Elaboración Propia

```php
public function agregar_publicacion(Request $request)
{

    $archivo    = $request->file('file');
    $input      = array('file' => $archivo);
    $reglas     = array('file' => 'required|mimes:pdf|max:50000'); //recordar que para activar mimes se debe descomentar la li
    $validacion = Validator::make($input, $reglas);
    if ($validacion->fails()) {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "El archivo no es un pdf o es demasiado GRANDE para subirl
    } else {
        $data    = $request->all();
        $reglas = array('publicaciones_tipopublicaciones_id' => 'required|numeric',
            'publicaciones_titulo'                           => 'required|alpha|max:150',
            'publicaciones_descripcion'                      => 'required|alpha|max:250',
        );
        $mensajes = array('publicaciones_tipopublicaciones_id.required' => 'Ingresar el tipo de publicación es obligatorio',
            'publicaciones_tipopublicaciones_id.numeric'                 => 'Seleccione la categoría de la publicación',
            'publicaciones_titulo.required'                              => 'Ingresar titulo del documento',
            'publicaciones_titulo.alpha'                                 => 'El campo de titulo no puede contener números',
            'publicaciones_descripcion.required'                         => 'Ingresar la descripción de la publicación es o
            'publicaciones_descripcion.alpha'                            => 'El campo de descripción no puede contener núme
        );

        $validacion = Validator::make($data, $reglas, $mensajes);
        if ($validacion->fails()) {
            $errores = $validacion->errors();
            return new JsonResponse($errores, 422);
        }

        $publicacion                                = new Publicacion;
        $publicacion->publicaciones_usuarios_id     = $request->input("publicaciones_usuarios_id");
        $publicacion->publicaciones_tipopublicaciones_id = $request->input("publicaciones_tipopublicaciones_id");
        $publicacion->publicaciones_titulo          = $request->input("publicaciones_titulo");
        $publicacion->publicaciones_descripcion     = $request->input("publicaciones_descripcion");

        $carpeta                        = $request->input("publicaciones_tipopublicaciones_id");
        $ruta                           = $carpeta . $request->input("publicaciones_usuarios_id") . "_" . $archivo->getClient
        $r1                             = Storage::disk('archivos')->put($ruta, \File::get($archivo));
        $publicacion->publicaciones_ruta = $ruta;
        $agregar_publicacion            = $publicacion->save();

        if ($agregar_publicacion) {
            return view("intranet.mensajes.msj_correcto")->with("msj", "Publicacion agregada correctamente");
        } else {
            return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
        }
    }
}
```

Imagen 117: Método agregar_publicacion dentro del controlador PublicacionesController

Fuente: Elaboración Propia

```php
    */
    public function test_publicar_documentos()
    {
        $user = new \App\Usuario(['id' => '1']);
        auth()->login($user);
        $this->visit('form_publicar_documentos/1')
            ->type('1', 'publicaciones_usuarios_id')
            ->type('1', 'publicaciones_tipopublicaciones_id')
            ->type('REGLAMENTO', 'publicaciones_titulo')
            ->type('Reglas', 'publicaciones_descripcion')
            ->type('C:\Users\DCCRC\Desktop\manualvwfox1.6.pdf', 'file')
            ->press('agregar_documento');
    }
}
```

```
C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

.............................                                      29 / 29 (100%)

Time: 2.25 seconds, Memory: 52.00MB

OK (29 tests, 41 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 118: Test de pruebas HU10-01 –Seguimiento Publicar Documentos - OK

Fuente: Elaboración Propia

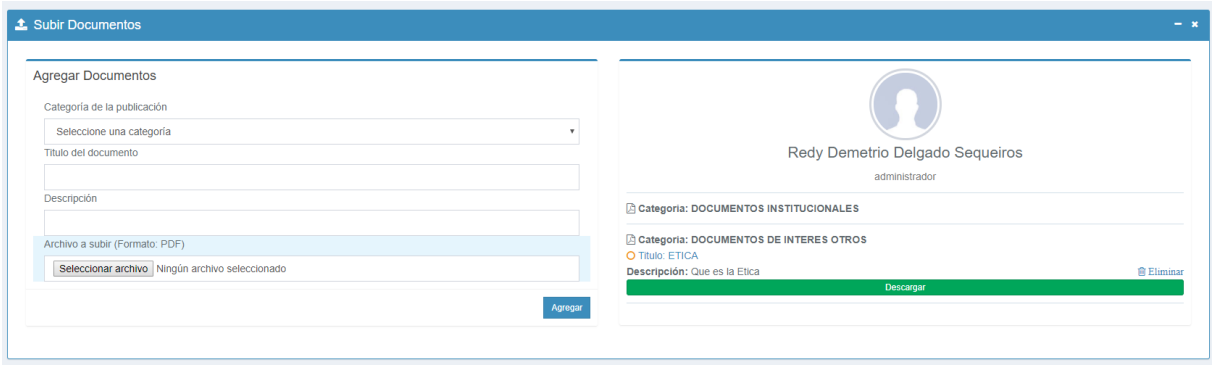


Imagen 119: HU10-01 - Formulario Subir Documentos

Fuente: Elaboración Propia

4.6.10.2. Test de Prueba del Product Backlog HU10-01 – Crear Examen

Nombre de proceso:	Crear Examen	Id backlog:	HU10-01
		Id historia de usuario:	HU10
Autor de caso de prueba:	Administrador		

Test de prueba	Especificación

public function test_Ir_form_registrar_examen()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('form_registrar_examen')
->type('1', 'examenes_publicaciones_id')
->type('Primera', 'examenes_unidad')
->type('2018/09/01', 'examenes_fechainicio')
->type('2018/010/01', 'examenes_fechafin')
->type('15', 'examenes_tiempo')
->press('agregar_examen');
}
	Al crear el examen especificaremos, documento del cual se tomara las preguntas, la unidad, el tiempo que debe estar activo el examen, los minutos disponibles para rendir el examen.

Dato adicional:	--

*Crear el archivo que contendrá los test de pruebas de los exámenes.

Imagen 120: Creación del Archivo ExamenesTest

Fuente: Elaboración Propia

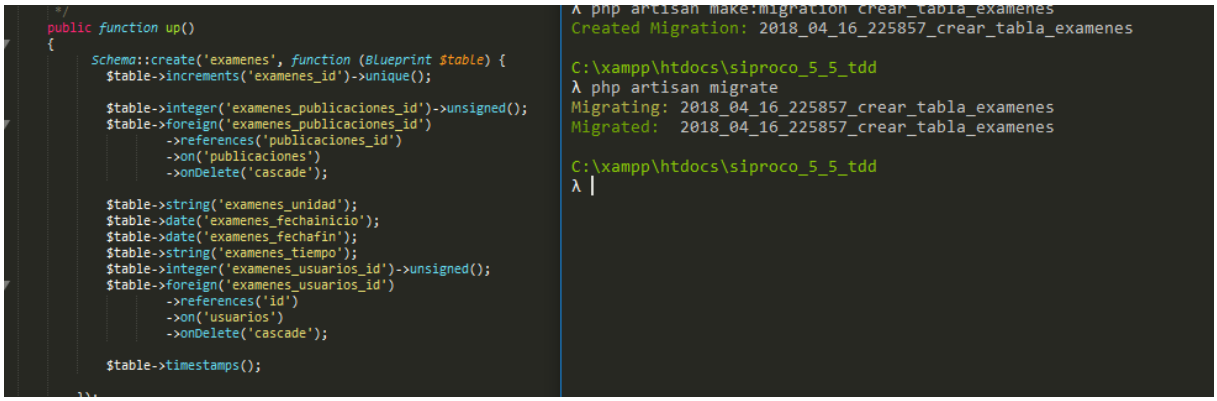


Imagen 121: Tabla Examenes desde Laravel

Fuente: Elaboración Propia

Se crea la tabla exámenes desde laravel, y se ejecuta la migración al gestor de base de datos.

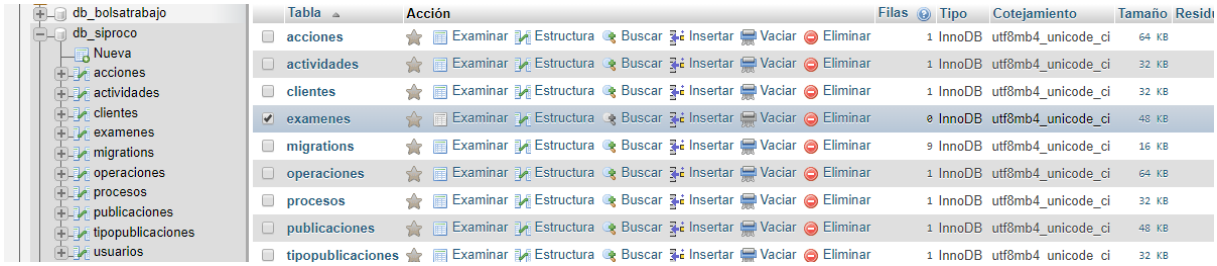


Imagen 122: Tabla Examenes en MySQL

Fuente: Elaboración Propia

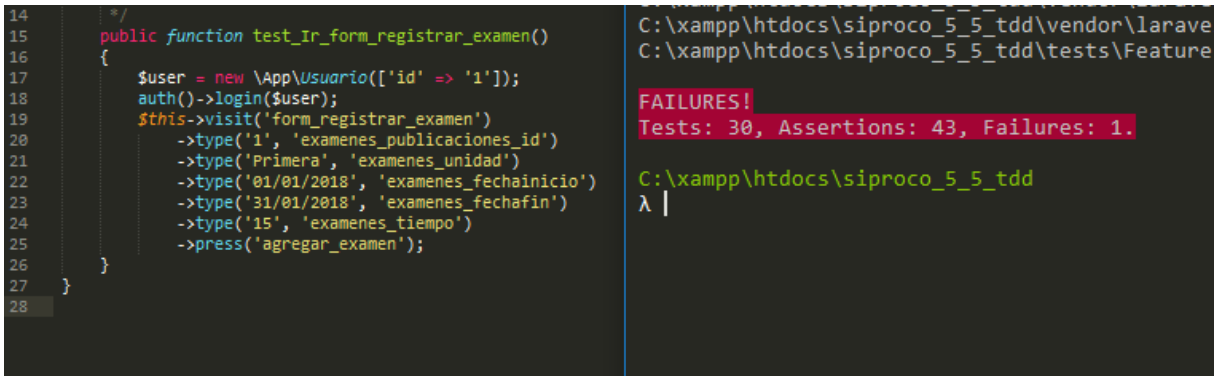


Imagen 123: Test de pruebas HU10-02 - Crear Examen - FAILURE

Fuente: Elaboración Propia

```php
Route::get('form_registrar_examen', 'ExamenController@registro_examen');
Route::post('form_registrar_examen/agregar_nuevo_examen', 'ExamenController@agregar_nuevo_examen');
```

Imagen 124: Ruta del formulario para registrar examen

Fuente: Elaboración Propia

```php
public function agregar_nuevo_examen(Request $request)
{
    $data = $request->all();

    $reglas = array('examenes_publicaciones_id' => 'required|numeric',
        'examenes_unidad'                       => 'required|Alpha',
        'examenes_fechainicio'                  => 'required|date',
        'examenes_fechafin'                     => 'required|date|after:examenes_fechainicio',
        'examenes_tiempo'                       => 'required|numeric',
    );
    $mensajes = array(

        'examenes_publicaciones_id.numeric' => 'Seleccione un tema de examen',
        'examenes_unidad.required'          => 'Ingresar el tema de la unidad',
        'examenes_unidad.alpha'             => 'El campo de nombre de la unidad no puede contener números',
        'examenes_fechafin.after'           => 'La fecha de fin no puede ser menor a la fecha de inicio del examen',
        'examenes_tiempo.numeric'           => 'El tiempo solo es aceptado en minutos totales',
    );

    $validacion = Validator::make($data, $reglas, $mensajes);
    if ($validacion->fails()) {
        $errores = $validacion->errors();
        return new JsonResponse($errores, 422);
    }

    $examen                          = new Examen;
    $examen->examenes_publicaciones_id = $data["examenes_publicaciones_id"];
    $examen->examenes_unidad         = $data["examenes_unidad"];
    $examen->examenes_fechainicio    = $data["examenes_fechainicio"];
    $examen->examenes_fechafin       = $data["examenes_fechafin"];
    $examen->examenes_tiempo         = $data["examenes_tiempo"];
    $examen->examenes_usuarios_id    = $data["examenes_usuarios_id"];

    $agregar_nuevo_examen = $examen->save();
    if ($agregar_nuevo_examen) {
        return view("intranet.mensajes.msj_correcto")->with("msj", "Examen registrado correctamente");
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
    }

}
```

Imagen 125: Método agregar_nuevo_examen dentro del controlador ExamenController

Fuente: Elaboración Propia

```php
public function test_Ir_form_registrar_examen()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('form_registrar_examen')
        ->type('1', 'examenes_publicaciones_id')
        ->type('Primera', 'examenes_unidad')
        ->type('2018/06/01', 'examenes_fechainicio')
        ->type('2018/07/01', 'examenes_fechafin')
        ->type('15', 'examenes_tiempo')
        ->press('agregar_examen');
}
}
```

```
C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

.............................                                     30 / 30 (100%)

Time: 2.27 seconds, Memory: 52.00MB

OK (30 tests, 43 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
```

Imagen 126: Test de pruebas HU10-02 - Crear Examen - OK

Fuente: Elaboración Propia

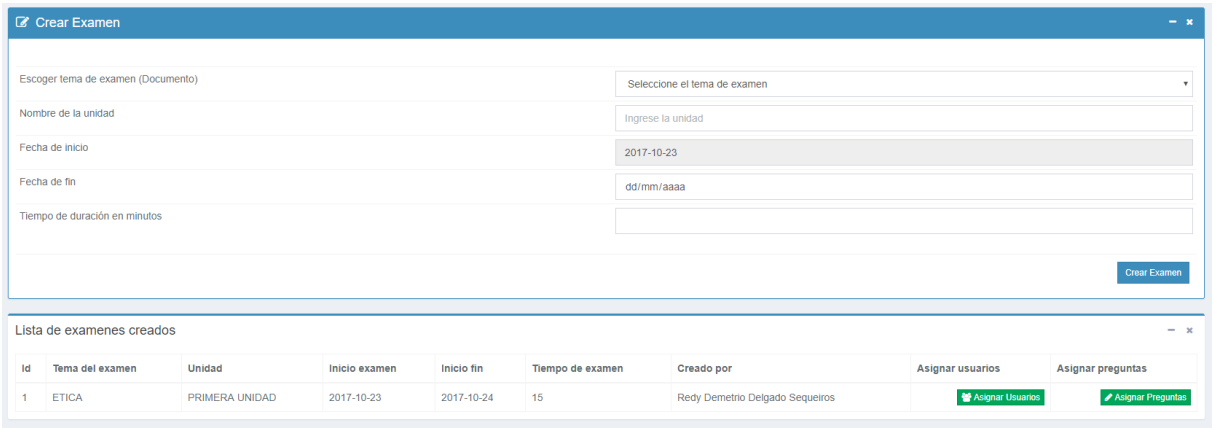


Imagen 127: HU10-02 - Formulario Crear Examen

Fuente: Elaboración Propia

4.6.10.3. Test de Prueba del Product Backlog HU10-02 – Agregar Preguntas al Examen

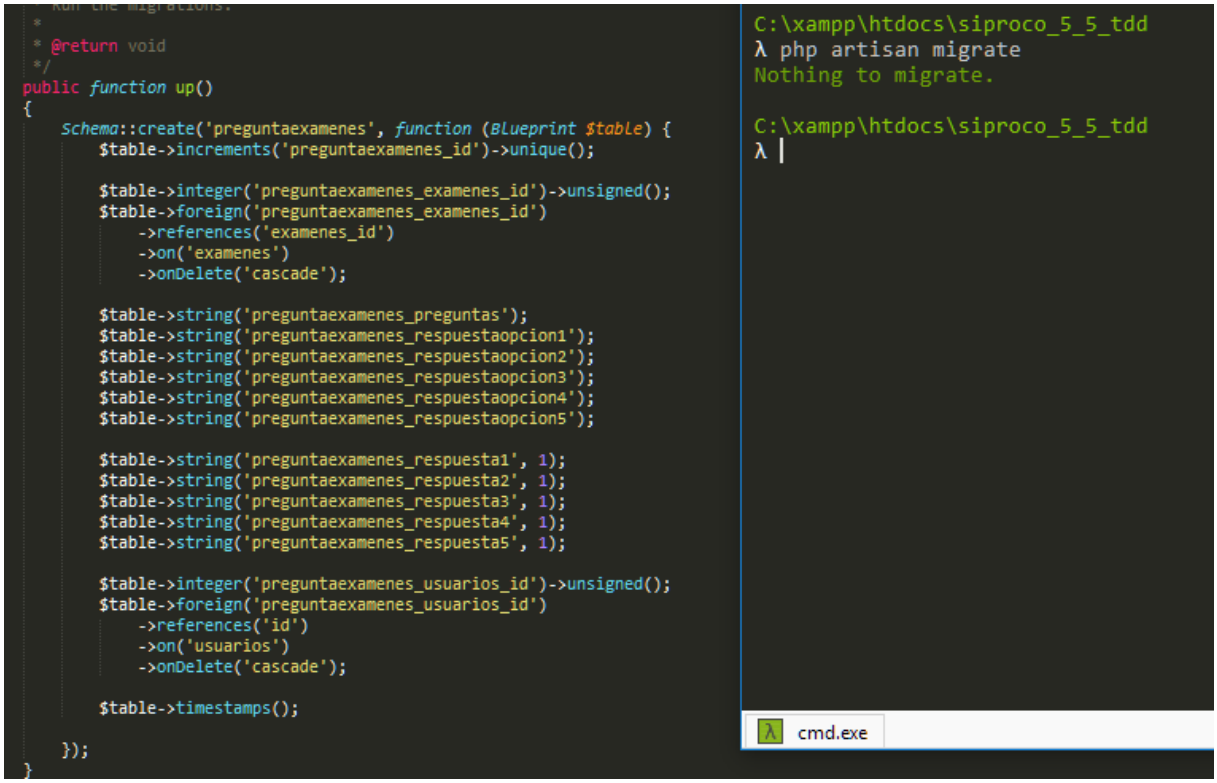
Nombre de proceso:	Agregar Preguntas al Examen	Id backlog:	HU10-02
		Id historia de usuario:	HU10
Autor de caso de prueba:	Administrador		

Test de prueba	Especificación

public function test_agregar_pregunta_al_examen()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('form_registrar_pregunta/1')
->type('1', 'preguntaexamenes_examenes_id')
->type('¿Que es el RIT?', 'preguntaexamenes_preguntas')
->type('Es el Reglamento Interno de Trabajo', 'preguntaexamenes_respuestaopcion1')
->type('Es un reglamento', 'preguntaexamenes_respuestaopcion2')
->type('Es una norma de la institucion', 'preguntaexamenes_respuestaopcion3')
->type('Es un documento importante', 'preguntaexamenes_respuestaopcion4')
->type('Registro Interno de Tareas', 'preguntaexamenes_respuestaopcion5')
->type('c', 'preguntaexamenes_respuesta1')
->type('i', 'preguntaexamenes_respuesta2')
->type('i', 'preguntaexamenes_respuesta3')
->type('i', 'preguntaexamenes_respuesta4')
->type('i', 'preguntaexamenes_respuesta5')
->type('1', 'preguntaexamenes_usuarios_id')
	Una vez creado el examen con sus específicas, se deberá establecer las preguntas que se crean necesarias para demostrar lo aprendido por los demás usuarios.

->press('agregar_pregunta'); }	
Dato adicional:	El sistema nos calcula la nota automáticamente, mediante una fórmula matemática en base al porcentaje de acertacines.

*Se continúa realizando la escritura del test de pruebas en el mismo archivo ExamenesTest.




```
 * @return void
 */
public function up()
{
    Schema::create('preguntaexamenes', function (Blueprint $table) {
        $table->increments('preguntaexamenes_id')->unique();

        $table->integer('preguntaexamenes_examenes_id')->unsigned();
        $table->foreign('preguntaexamenes_examenes_id')
            ->references('examenes_id')
            ->on('examenes')
            ->onDelete('cascade');

        $table->string('preguntaexamenes_preguntas');
        $table->string('preguntaexamenes_respuestaopcion1');
        $table->string('preguntaexamenes_respuestaopcion2');
        $table->string('preguntaexamenes_respuestaopcion3');
        $table->string('preguntaexamenes_respuestaopcion4');
        $table->string('preguntaexamenes_respuestaopcion5');

        $table->string('preguntaexamenes_respuesta1', 1);
        $table->string('preguntaexamenes_respuesta2', 1);
        $table->string('preguntaexamenes_respuesta3', 1);
        $table->string('preguntaexamenes_respuesta4', 1);
        $table->string('preguntaexamenes_respuesta5', 1);

        $table->integer('preguntaexamenes_usuarios_id')->unsigned();
        $table->foreign('preguntaexamenes_usuarios_id')
            ->references('id')
            ->on('usuarios')
            ->onDelete('cascade');

        $table->timestamps();

    });
}
```

Imagen 128: Tabla PreguntaExamenes desde Laravel

Fuente: Elaboración Propia

Imagen 129: Tabla PreguntaExamenes en MySQL

Fuente: Elaboración Propia

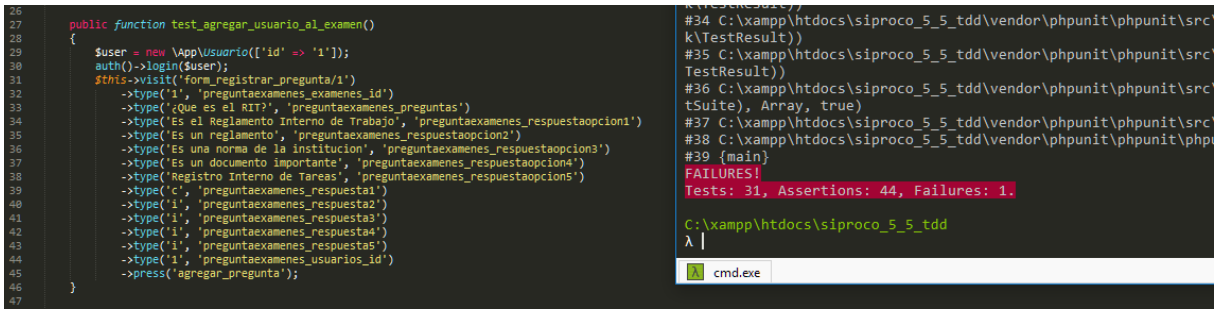


Imagen 130: Test de pruebas HU10-02 - Agregar Pregunta al Examen- FAILURE

Fuente: Elaboración Propia

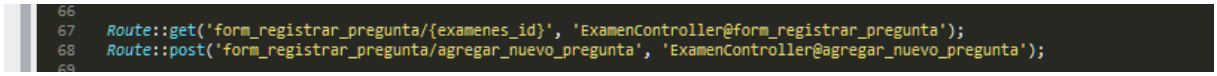
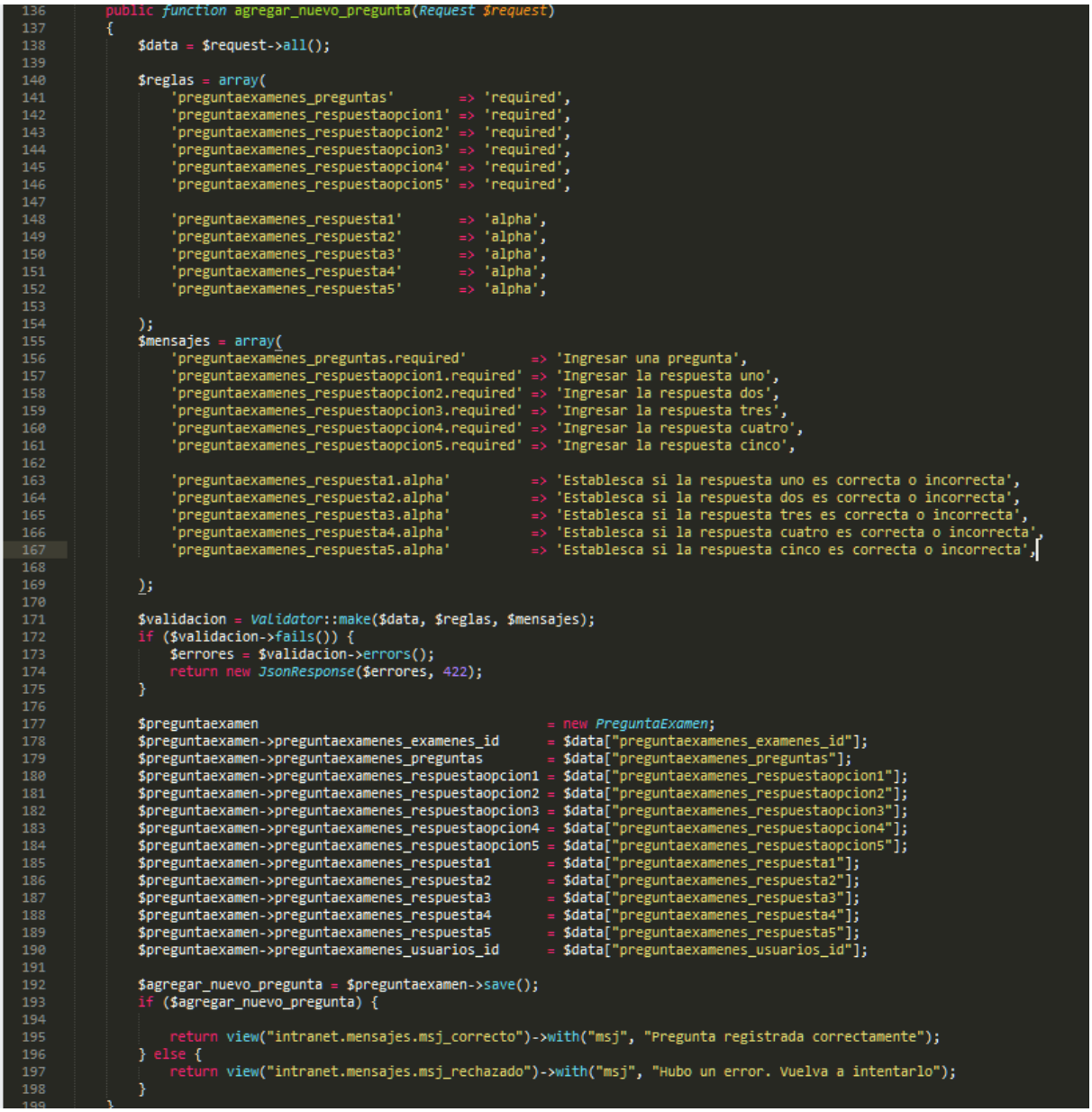


Imagen 131: Ruta del formulario para agregar pregunta al examen

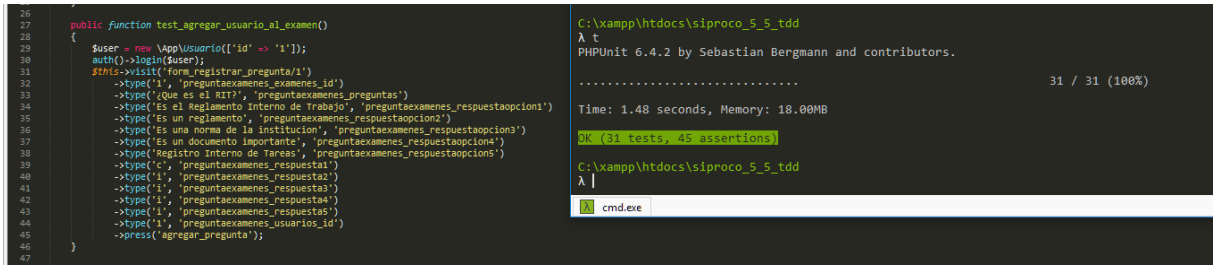
Fuente: Elaboración Propia



```php
public function agregar_nuevo_pregunta(Request $request)
{
    $data = $request->all();

    $reglas = array(
        'preguntaexamenes_preguntas'        => 'required',
        'preguntaexamenes_respuestaopcion1' => 'required',
        'preguntaexamenes_respuestaopcion2' => 'required',
        'preguntaexamenes_respuestaopcion3' => 'required',
        'preguntaexamenes_respuestaopcion4' => 'required',
        'preguntaexamenes_respuestaopcion5' => 'required',

        'preguntaexamenes_respuesta1'       => 'alpha',
        'preguntaexamenes_respuesta2'       => 'alpha',
        'preguntaexamenes_respuesta3'       => 'alpha',
        'preguntaexamenes_respuesta4'       => 'alpha',
        'preguntaexamenes_respuesta5'       => 'alpha',

    );
    $mensajes = array(
        'preguntaexamenes_preguntas.required'        => 'Ingresar una pregunta',
        'preguntaexamenes_respuestaopcion1.required' => 'Ingresar la respuesta uno',
        'preguntaexamenes_respuestaopcion2.required' => 'Ingresar la respuesta dos',
        'preguntaexamenes_respuestaopcion3.required' => 'Ingresar la respuesta tres',
        'preguntaexamenes_respuestaopcion4.required' => 'Ingresar la respuesta cuatro',
        'preguntaexamenes_respuestaopcion5.required' => 'Ingresar la respuesta cinco',

        'preguntaexamenes_respuesta1.alpha'          => 'Establesca si la respuesta uno es correcta o incorrecta',
        'preguntaexamenes_respuesta2.alpha'          => 'Establesca si la respuesta dos es correcta o incorrecta',
        'preguntaexamenes_respuesta3.alpha'          => 'Establesca si la respuesta tres es correcta o incorrecta',
        'preguntaexamenes_respuesta4.alpha'          => 'Establesca si la respuesta cuatro es correcta o incorrecta',
        'preguntaexamenes_respuesta5.alpha'          => 'Establesca si la respuesta cinco es correcta o incorrecta',

    );

    $validacion = Validator::make($data, $reglas, $mensajes);
    if ($validacion->fails()) {
        $errores = $validacion->errors();
        return new JsonResponse($errores, 422);
    }

    $preguntaexamen                                  = new PreguntaExamen;
    $preguntaexamen->preguntaexamenes_examenes_id    = $data["preguntaexamenes_examenes_id"];
    $preguntaexamen->preguntaexamenes_preguntas      = $data["preguntaexamenes_preguntas"];
    $preguntaexamen->preguntaexamenes_respuestaopcion1 = $data["preguntaexamenes_respuestaopcion1"];
    $preguntaexamen->preguntaexamenes_respuestaopcion2 = $data["preguntaexamenes_respuestaopcion2"];
    $preguntaexamen->preguntaexamenes_respuestaopcion3 = $data["preguntaexamenes_respuestaopcion3"];
    $preguntaexamen->preguntaexamenes_respuestaopcion4 = $data["preguntaexamenes_respuestaopcion4"];
    $preguntaexamen->preguntaexamenes_respuestaopcion5 = $data["preguntaexamenes_respuestaopcion5"];
    $preguntaexamen->preguntaexamenes_respuesta1     = $data["preguntaexamenes_respuesta1"];
    $preguntaexamen->preguntaexamenes_respuesta2     = $data["preguntaexamenes_respuesta2"];
    $preguntaexamen->preguntaexamenes_respuesta3     = $data["preguntaexamenes_respuesta3"];
    $preguntaexamen->preguntaexamenes_respuesta4     = $data["preguntaexamenes_respuesta4"];
    $preguntaexamen->preguntaexamenes_respuesta5     = $data["preguntaexamenes_respuesta5"];
    $preguntaexamen->preguntaexamenes_usuarios_id    = $data["preguntaexamenes_usuarios_id"];

    $agregar_nuevo_pregunta = $preguntaexamen->save();
    if ($agregar_nuevo_pregunta) {

        return view("intranet.mensajes.msj_correcto")->with("msj", "Pregunta registrada correctamente");
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
    }
}
```

Imagen 132: Método agregar_nueva_pregunta dentro del controlador ExamenController

Fuente: Elaboración Propia



```php
public function test_agregar_usuario_al_examen()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('form_registrar_pregunta/1')
        ->type('1', 'preguntaexamenes_examenes_id')
        ->type('¿Que es el RIT?', 'preguntaexamenes_preguntas')
        ->type('Es el Reglamento Interno de Trabajo', 'preguntaexamenes_respuestaopcion1')
        ->type('Es un reglamento', 'preguntaexamenes_respuestaopcion2')
        ->type('Es una norma de la institucion', 'preguntaexamenes_respuestaopcion3')
        ->type('Es un documento importante', 'preguntaexamenes_respuestaopcion4')
        ->type('Registro Interno de Tareas', 'preguntaexamenes_respuestaopcion5')
        ->type('c', 'preguntaexamenes_respuesta1')
        ->type('i', 'preguntaexamenes_respuesta2')
        ->type('i', 'preguntaexamenes_respuesta3')
        ->type('i', 'preguntaexamenes_respuesta4')
        ->type('i', 'preguntaexamenes_respuesta5')
        ->type('1', 'preguntaexamenes_usuarios_id')
        ->press('Agregar_pregunta');
}
```

```
C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

.............................                                     31 / 31 (100%)

Time: 1.48 seconds, Memory: 18.00MB

OK (31 tests, 45 assertions)

C:\xampp\htdocs\siproco_5_5_tdd
λ
```

Imagen 133: Test de pruebas HU10-02 - Agregar Pregunta al Examen - OK

Fuente: Elaboración Propia

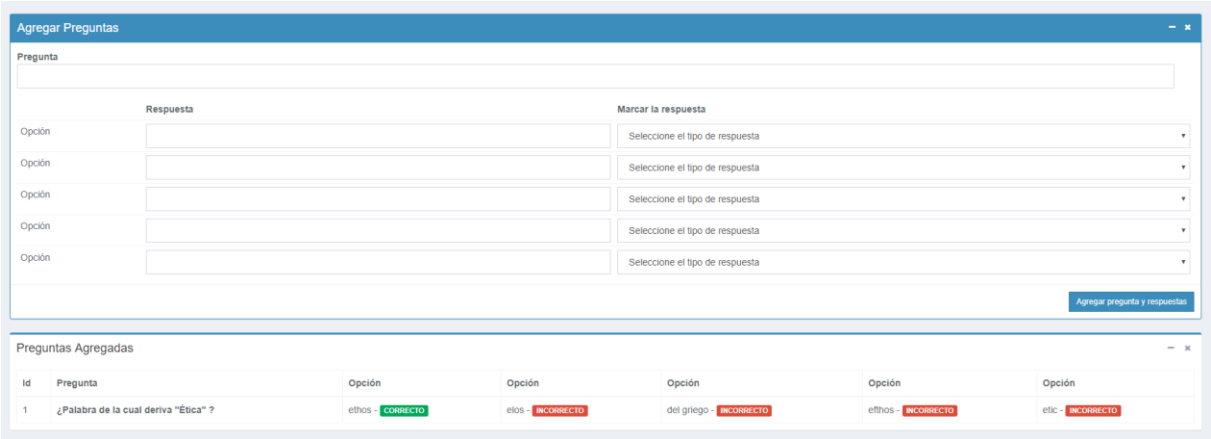


Imagen 134: HU10-02 - Formulario Agregar Preguntas

Fuente: Elaboración Propia

4.6.10.4. Test de Prueba del Product Backlog HU10-03 – Agregar Usuario Examen

Nombre de proceso:	Agregar Usuario Examen	Id backlog:	HU10-03
		Id historia de usuario:	HU10
Autor de caso de prueba:	Administrador		
Test de prueba		Especificación	

public function test_asignar_Usuario()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('agregar_usuario_examen/1/1');
 	
}
		Agregar al usuario para que rinda el examen.	

Dato adicional:	Podremos discriminar los usuarios que deben rendir o no el examen.		

*Se continúa realizando la escritura del test de pruebas, en el mismo archivo ExamenesTest.

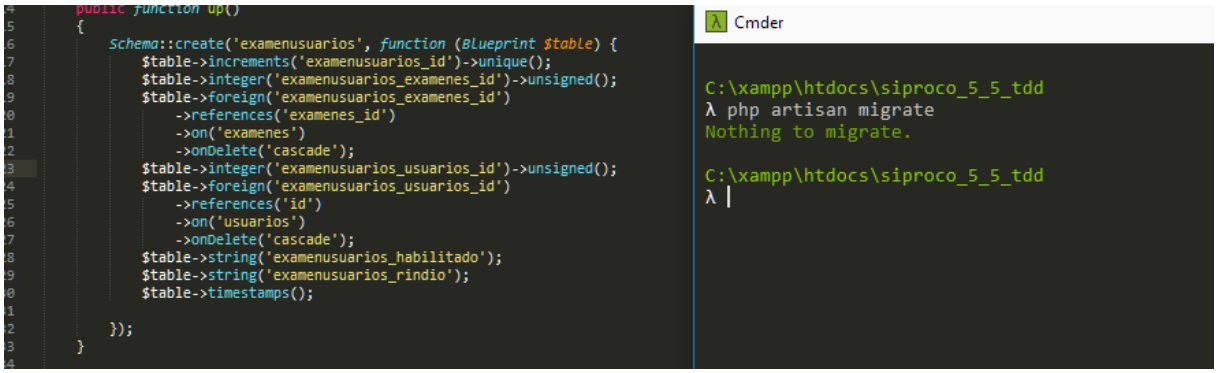


Imagen 135: Tabla ExamenUsuarios desde Laravel

Fuente: Elaboración Propia

Imagen 136: Tabla ExamenUsuarios en MySQL

Fuente: Elaboración Propia

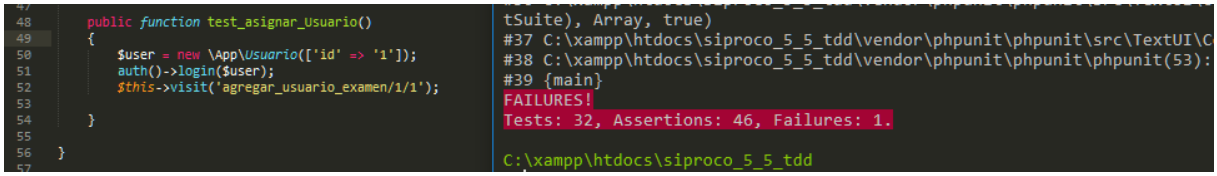


Imagen 137: Test de pruebas HU10-02 - Agregar Agregar Usuario al Examen- FAILURE

Fuente: Elaboración Propia

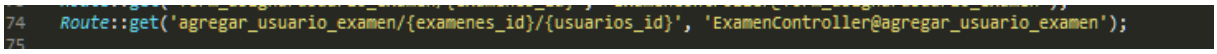


Imagen 138: Ruta del formulario para agregar usuario al examen

Fuente: Elaboración Propia

```php
public function agregar_usuario_examen($examenes_id, $usuarios_id)
{
    $verificar = DB::table('examenusuarios')->where('examenusuarios_habilitado', '=', 's')
        ->where('examenusuarios_usuarios_id', '=', $usuarios_id)
        ->where('examenusuarios_examenes_id', '=', $examenes_id)
        ->join('usuarios', 'usuarios.id', '=', 'examenusuarios.examenusuarios_usuarios_id')
        ->first();
    if ($verificar == null) {
        $examenusuario                          = new ExamenUsuario;
        $examenusuario->examenusuarios_examenes_id = $examenes_id;
        $examenusuario->examenusuarios_usuarios_id = $usuarios_id;
        $examenusuario->examenusuarios_habilitado  = 's';
        $examenusuario->examenusuarios_rindio      = 'n';

        $agregar_a_examen = $examenusuario->save();
        if ($agregar_a_examen) {

            return view("intranet.mensajes.msj_correcto")->with("msj", "Usuario agregado correctamente");
        } else {
            return view("intranet.mensajes.msj_rechazado")->with("msj", "Hubo un error. Vuelva a intentarlo");
        }
    } else {
        return view("intranet.mensajes.msj_rechazado")->with("msj", "Usuario agregado anteriormente a este examen");
    }

}
```

Imagen 139: Método agregar_usuario_examen dentro del controlador ExamenController

Fuente: Elaboración Propia

```php
public function test_asignar_Usuario()
{
    $user = new \App\Usuario(['id' => '1']);
    auth()->login($user);
    $this->visit('agregar_usuario_examen/1/1');

}
}
```

```
C:\xampp\htdocs\siproco_5_5_tdd
λ t
PHPUnit 6.4.2 by Sebastian Bergmann and contributors.

.................................                                 32 / 32 (100%)

Time: 1.47 seconds, Memory: 18.00MB

OK (32 tests, 46 assertions)
```

Imagen 140: Test de pruebas HU10-02 - Agregar Agregar Usuario al Examen - OK

Fuente: Elaboración Propia

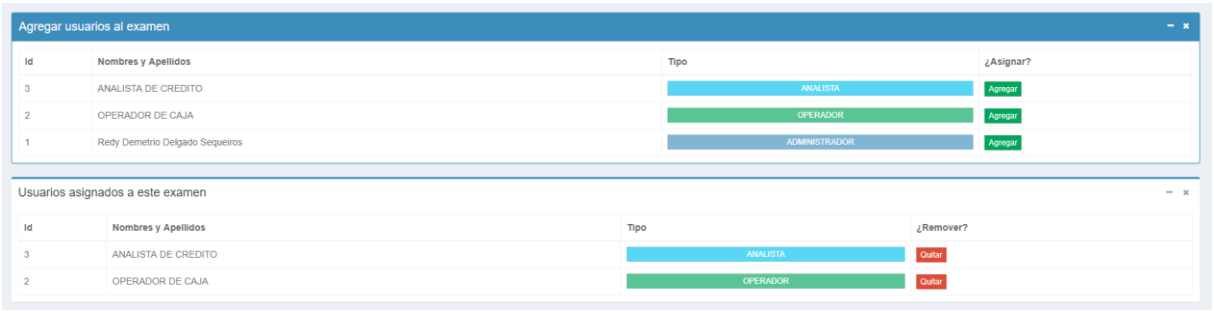


Imagen 141: HU10-02 - Formulario Agregar Usuarios al Examen

Fuente: Elaboración Propia

4.6.11. Product Backlog – HU11

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU11		**El operador de caja, los analistas de créditos necesitan un módulo donde los usuarios puedan visualizar sus labores realizas.**	**1**
	HU11-01	Como usuario del sistema quiero ver un resumen de todos los registros realizados por el operador de caja y analista de créditos. Con el fin de estar enterado sobre las actividades y operaciones que se han estado realizando. Condiciones: - El resumen debe de tener la capacidad de filtrar por actividad o proceso, además de contener un filtro por cliente.	1.1

Tabla 15: Product Backlog - HU11

Fuente: Elaboración Propia

4.6.11.1. Test de Prueba del Product Backlog HU11-01 – Visualizar Labores

Nombre de proceso:	Visualizar Labores	Id backlog:	HU11-01
		Id historia de usuario:	HU11
Autor de caso de prueba:	Operador de caja, analista de créditos		

Test de prueba	Especificación

public function test_ir_form_listado_operaciones()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('listado_operaciones/1');
}
public function test_ir_form_listado_acciones()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
	El usuario puede ver solo sus registros ingresados.

$this->visit('listado_acciones/1'); }	
Dato adicional:	--

*Se vuelve a hacer uso del archivo VisualizarTest para este test de pruebas.

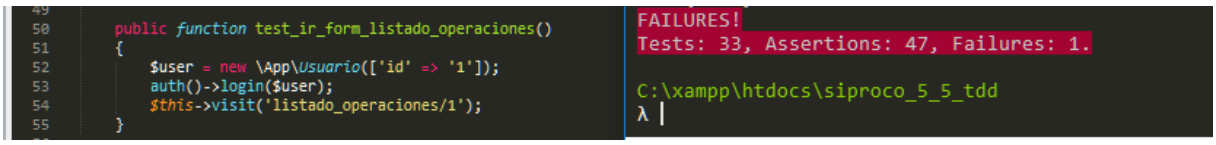


Imagen 142: Test de pruebas HU11-02 – Listado Operaciones - FAILURE

Fuente: Elaboración Propia

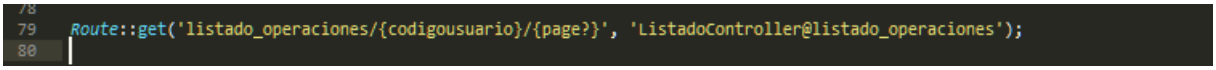


Imagen 143: Ruta del formulario para visualizar sus operaciones realizadas

Fuente: Elaboración Propia

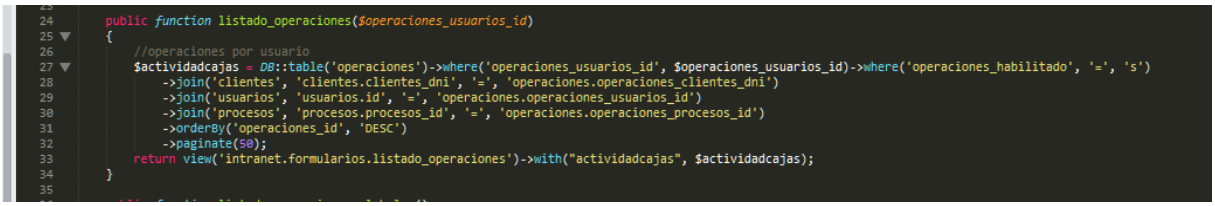


Imagen 144: Método listado_operaciones dentro del controlador ListadoController

Fuente: Elaboración Propia

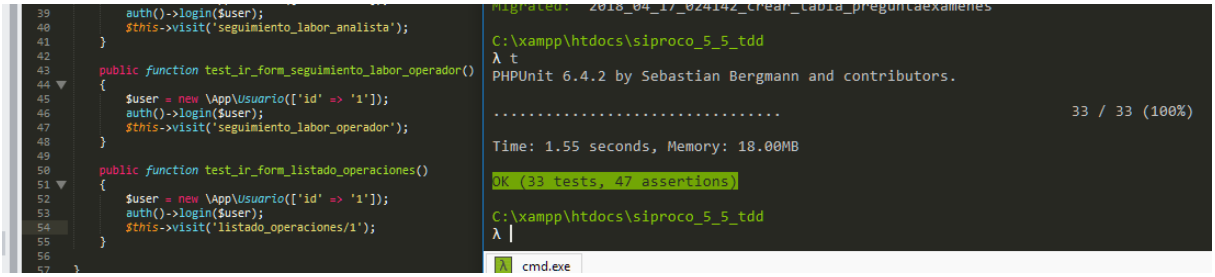


Imagen 145: Test de pruebas HU11-02 – Listado Operaciones - OK

Fuente: Elaboración Propia

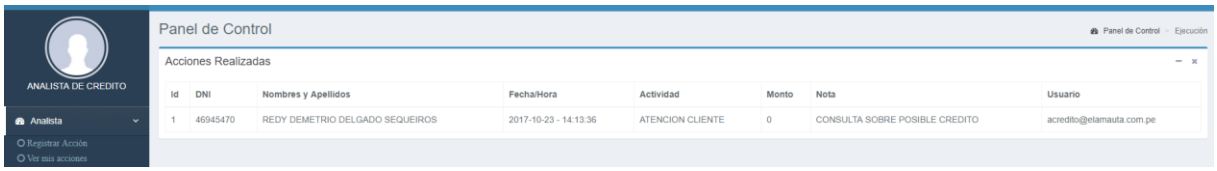


Imagen 146: HU11-01 - Formulario Acciones Realizadas

Fuente: Elaboración Propia

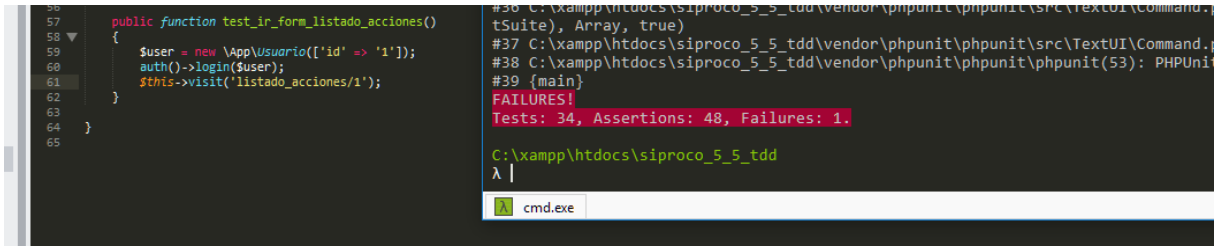


Imagen 147: Test de pruebas HU11-02 – Listado Acciones - FAILURE

Fuente: Elaboración Propia

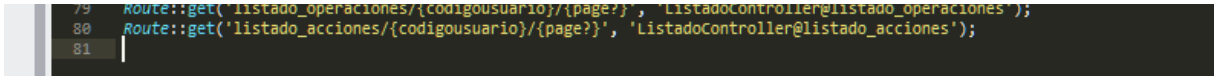


Imagen 148: Ruta del formulario para visualizar sus acciones realizadas

Fuente: Elaboración Propia

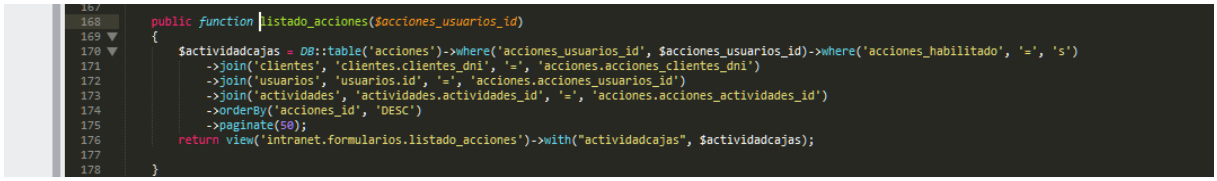


Imagen 149: Método listado_acciones dentro del controlador ListadoController

Fuente: Elaboración Propia

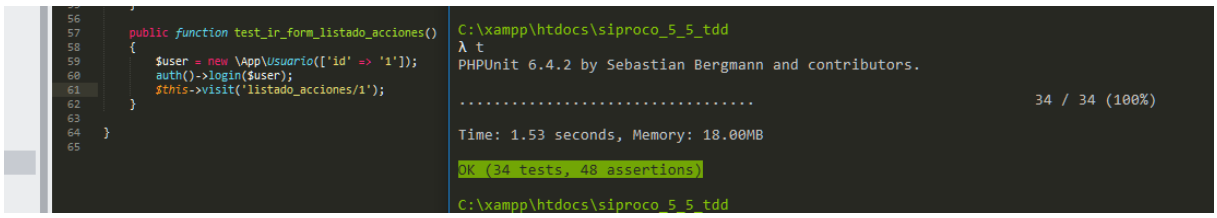


Imagen 150: Test de pruebas HU11-02 – Listado Acciones - OK

Fuente: Elaboración Propia

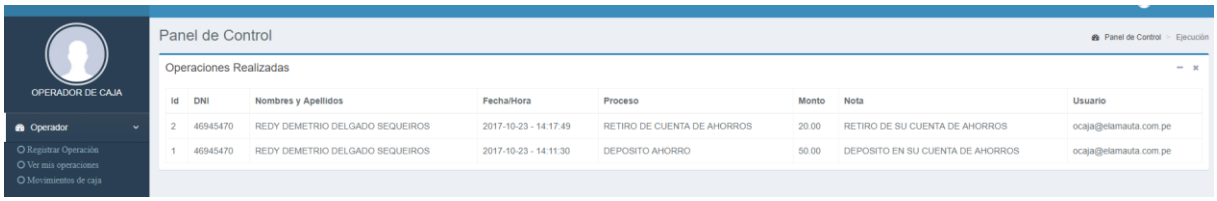


Imagen 151: HU11-01 - Formulario Operaciones Realizadas

Fuente: Elaboración Propia

4.6.12. Product Backlog – HU12

Código Historia de Usuario	Código de backlog	Requerimiento	Prioridad
HU12		**El operador de caja, los analistas de créditos, administrador, usuario externo necesitan un módulo donde puedan visualizar documentación entregada por el administrador.**	**1**
	HU12-01	Como usuario del sistema quiero ver los documentos cargados al sistema por el administrador, para poder realizar un estudio más a profundidad sobre las capacitaciones brindadas por la institución. Condiciones: - Una vista donde tenga las condiciones básicas, para realizar lectura de documento.	1.1

Tabla 16: Product Backlog - HU12

Fuente: Elaboración Propia

*Se vuelve a hacer uso del archivo VisualizarTest para este test de pruebas.

4.6.12.1. Test de Prueba del Product Backlog HU12-01 – Visualizar Publicaciones

Nombre de proceso:	Visualizar Publicaciones	Id backlog:	HU12-01
		Id historia de usuario:	HU12
Autor de caso de prueba:	Administrador, operador de caja, analista de créditos, usuario externo		

Test de prueba	Especificación

public function test_ir_form_listado_publicaciones()
{
$user = new \App\Usuario(['id' => '1']);
auth()->login($user);
$this->visit('listado_publicaciones');
}
	El usuario puede ver los documentos subidos por el administrador.

Dato adicional:	Estos documentos son de carácter privado, por tal motivo no se le permite la descarga.

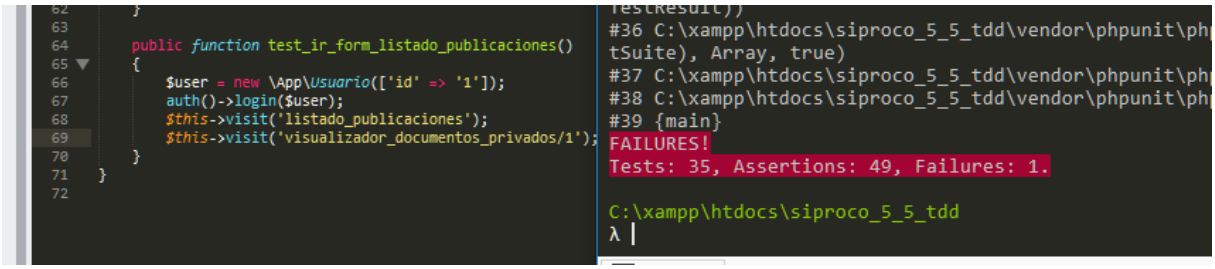


Imagen 152: Test de pruebas HU12-01 – Listado Publicaciones y Visualizador Documentos - FAILURE

Fuente: Elaboración Propia

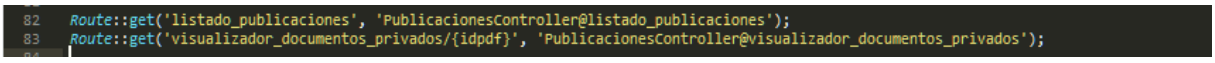


Imagen 153: Ruta del formulario para visualizar documentos

Fuente: Elaboración Propia