Análisis, diseño e implementación de un prototipo de sistema de información para un laboratorio de investigación

Grado en Ingeniería Informática
Trabajo de Fin de Grado
Convocatoria diciembre 2017

Autor: Jose Antonio García-Arévalo Acosta
Tutor: Abraham Rodríguez

Fecha de presentación: Enero de 2018

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

TFT04

Lista de comprobación de documentación adjunta (resumida, detalles en el Manual Operativo)

[x] Comprobar que han copiado al repositorio:

[x] Memoria (máximo 100 pág. ~30000 palabras) en formato pdf con logos e identificación del Trabajo en la portada y, tras ella, la primera página de este TFT04 insertada (el pdf resultante debe firmarse digitalmente, ubicando las firmas en el espacio previsto para ello en el TFT04).

[x] Resumen en formato txt en español e inglés (un único fichero txt con límite de 120 palabras por idioma).

[x] Código (en caso de que el TFT lo incluya, un directorio, o bien fichero comprimido, o bien un txt con instrucciones para acceder al repositorio).

- Sólo en el caso de que en este impreso TFT04 se haya indicado que NO está previsto iniciar trámite de registro de la propiedad intelectual/industrial …

[x] Impreso relativo a la Difusión en Abierto del TFT en Biblioteca ULPGC como pdf firmado de forma digital.

NOTA: si se marca el registro de la propiedad intelectual sólo los tutores y el tribunal pueden asistir a la defensa.

Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

1.1 Contexto

El proyecto documentado en este Trabajo Fin de Grado consiste en un sistema de información para un laboratorio de investigación y diagnóstico, en concreto para el laboratorio de diagnóstico de enfermedades infecciosas de la facultad de veterinaria de la ULPGC. Se realiza el análisis, diseño e implementación de un sistema de información que se adapte a las necesidades del laboratorio y que permita gestionar los casos clínicos y los proyectos de investigación que se estén realizando, proporcionando diferentes niveles de acceso dentro de la plataforma dependiendo del rol del usuario. Mediante esta herramienta, se podrá gestionar los diferentes casos clínicos de forma más sencilla y rápida, así como la administración de los diferentes proyectos que se estén llevando a cabo. Este proyecto además de mejorar la productividad del laboratorio mediante funcionalidades exclusivas para las personas implicadas en el trabajo de investigación pretende ayudar a divulgar al público interesado tanto las investigaciones del equipo como ofrecer información sobre la micología veterinaria y de los servicios que ofrece el propio laboratorio.

Todo se ha realizado en contacto directo con el máximo responsable del laboratorio, por lo que se ha evaluado y desarrollado durante todo el proceso teniendo en cuenta las necesidades reales de la organización.

La organización que nos compete se trata de un laboratorio de investigación y diagnóstico de enfermedades infecciosas en veterinaria, englobando principalmente a aquellas patologías de etiología fúngica y bacteriana.

En las últimas décadas las patologías de etiología fúngica en veterinaria han ido en aumento, estas infecciones fúngicas están estrechamente relacionadas con la salud pública, ya que muchas de ellas suponen un riesgo de contagio para los propietarios de los animales y su entorno directo, teniendo un mayor impacto en individuos inmunocomprometidos. El creciente aumento de resistencia a los antifúngicos hace necesario realizar un correcto diagnóstico de estas, lo que justifica la necesidad de existencia de laboratorios especializados en esta especialidad.

El laboratorio tiene una doble vertiente, por un lado, actúa como laboratorio de diagnóstico de casos clínicos (como servicio que ofrece la Universidad a la comunidad) y por otro como laboratorio dedicado a la docencia e investigación.

1.1.1 Diagnóstico, Casos Clínicos

En el área de la salud el concepto de “caso clínico”, según Seco M., Andrés O. & Ramos G. (1999), es la presentación comentada de la situación sanitaria de un paciente, o grupo de pacientes, que se ejemplifica como «caso» al convertirse en la “realización individual de un fenómeno más o menos general”. Es un modelo que ilustra algún componente clínico peculiar con interés docente, o por su singularidad o rareza.

El procedimiento de diagnóstico comienza con la realización de una adecuada anamnesis y recogida de datos clínicos del paciente. El éxito en el diagnóstico micológico partiendo de una sospecha clínica, es fundamental realizar adecuadamente la recogida de la muestra a partir de la lesión, su correcta manipulación para mantener la viabilidad del agente etiológico y evitar posibles contaminaciones en su transporte y procesamiento, la siembra de esta en los medios idóneos y a la temperatura adecuada, así como la identificación e interpretación correcta de los aislamientos. Únicamente con el procesamiento adecuado se puede recuperar el hongo claramente asociado con el proceso infeccioso. A la hora de valorar un crecimiento fúngico hay que tener siempre presente la necesidad de diferenciar un “aislamiento significativo” de otros que no lo son, ya que no es lo mismo identificar una especie fúngica que diagnosticar una micosis (Cuenta y cols., 2006).

El procedimiento se reduce entonces en:

1.- Recogida de la muestra.

2.- Manejo y transporte de la muestra

3.- Recepción de la muestra

4.- Examen directo y procesado laboratorial

1.1.2 Docencia e Investigación

Se trata habitualmente de tesis doctorales, trabajos fin de grado, estudios o publicaciones que realiza el propio personal del laboratorio o alumnos del centro, por lo que existen diferentes tipos de proyectos dentro del laboratorio, algunos colaborativos y de fines educativos, mientras que en otros pueden intervenir una única persona o tener un fin profesional.

Varios son los proyectos de investigación en los que se encuentran incluidos el equipo de Micología Clínica Veterinaria de Enfermedades Infecciosas de la Facultad de Veterinaria de la ULPGC.

1.2 Motivación

La principal motivación de este trabajo fin de grado surge de la necesidad del laboratorio de implantar un sistema de gestión moderno que facilite la realización de las tareas que se llevan a cabo, esto ayudará al equipo de investigación a trabajar de una manera más rápida y eficiente, además de dar a conocer de cara al público tanto información relacionada con el campo de estudio como estudios o publicaciones de los miembros del laboratorio.

Esta necesidad me ofrece la oportunidad de poder aplicar los conocimientos adquiridos durante el Grado de Ingeniería Informática. Analizando, diseñando, implementando una solución a un problema real mediante un sistema de información web creado a medida. Además de ampliar mis conocimientos en el desarrollo web utilizando diferentes tecnologías, esencialmente Laravel que es un Framework de PHP muy potente y poder aplicarlo a un proyecto.

Actualmente todo los relacionado con la web está a la orden del día, con la llegada de las nuevas tecnologías cada vez se enfoca más hacia la compatibilidad entre dispositivos y poder acceder desde cualquier lugar por lo que las aplicaciones web están tomando un rumbo predominante en cuanto a desarrollo de software se refiere, es otro de los puntos por los que he decidido enfocar este proyecto hacia el desarrollo web. Aprender sobre tecnologías web hoy en día lo considero algo básico y es por eso por lo que decidí aprender sobre Laravel y realizar el trabajo con este framework. El desarrollo de TDD para el proyecto es otra de las motivaciones, pues es una práctica que además de efectiva va en aumento en el mundo de la programación.

1.3 Objetivos

El objetivo es la gestión de las principales tareas que se realizan en el laboratorio (casos clínicos y proyectos) mediante un sistema de información, que permita al laboratorio de diagnóstico de enfermedades infecciosas de la facultad de veterinaria de la ULPGC trabajar de manera más eficiente facilitando a los usuarios de la plataforma una manera mucho más rápida respecto a los medios tradicionales que cuenta actualmente la facultad y que además tiene que ser accesible desde cualquier dispositivo móvil u ordenador.

Además de ofrecer al público información sobre el grupo de investigación, los servicios del laboratorio y micología veterinaria.

Todo esto ofreciendo al usuario que vaya a usar tanto la plataforma de gestión como al que vaya a acceder a la información disponible en la web, una manera fácil, interactiva y amigable de interactuar con la interfaz de usuario.

La aplicación contará con un panel de administración donde se podrán modificar todos los registros del sistema de información, casos clínicos, antibióticos, usuarios, roles de usuario etc. de uso exclusivo para el administrador del sistema.

El sistema metodológico elegido es también uno de los objetivos fundamentales del desarrollo de este proyecto, realizar un desarrollo guiado por pruebas ya que es una práctica que hoy en día va en aumento y muchas empresas lo han tomado como desarrollo metodológico fundamental para el desarrollo de sus proyectos, además de ayudar a la depuración del código.

El despliegue de la aplicación es también uno de los objetivos, pues el uso de las conocidas como PaaS (Platform as a Service) o “Plataformas como Servicios” que son la evolución de las IaaS (Infraestructura como Servicio) son cada vez más usadas y aportaría mucho al desarrollo general del proyecto, poder ver como se comportaría el sistema en un entorno que puede ser de producción.

Entre los objetivos también se encuentra la realización de este documento donde se explica todo el proceso de desarrollo de una aplicación web desde cero, aplicando todos los conocimientos adquiridos durante estos años en un framework sencillo y divertido, con el objetivo de seguir un orden profesional y una sintaxis común para todos sus desarrolladores posteriores y su mantenimiento.

1.4 Metodología

La metodología tomada para el desarrollo de este trabajo es el desarrollo guiado por pruebas de software, o Test-driven development (TDD) que es una práctica de ingeniería de software, implica principalmente dos pasos:

1.Escribir las pruebas primero (Test First Development)

2.Refactorización (Refactoring).

Para escribir las pruebas generalmente se utilizan las pruebas unitarias (unit test en inglés). En primer lugar, se escribe una prueba y se verifica que las pruebas fallan. A continuación, se implementa el código que hace que la prueba pase satisfactoriamente y seguidamente se refactoriza el código escrito. El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione.

Para que funcione el desarrollo guiado por pruebas, el sistema que se programa tiene que ser lo suficientemente flexible como para permitir que sea probado automáticamente. Cada prueba será suficientemente pequeña como para que permita determinar unívocamente si el código probado pasa o no la verificación que ésta le impone. El diseño se ve favorecido ya que se evita el indeseado "sobre diseño" de las aplicaciones y se logran interfaces más claras y un código más cohesivo. Frameworks como PHPUnit proveen de un mecanismo para manejar y ejecutar conjuntos de pruebas automatizadas.

1.5 Distribución de la memoria

La estructura a seguir para esta memoria se dividirá en:

- Introducción: Capítulo de introducción donde se detallan nociones básicas sobre el contexto, objetivos, la motivación del proyecto y cómo está organizada la memoria del proyecto.

- Análisis: Comparativa de aplicaciones y webs similares, ¿Por qué Laravel? comparación con otros frameworks, definición de los requisitos del sistema, diagramas de casos de uso y todo lo que rodea al diseño.

- Desarrollo: Alcance de la implementación, diseño arquitectónico, modelo de datos, tests unitarios para el TDD, despliegue del servicio y tecnologías utilizadas.

- Conclusiones: Conclusiones del trabajo y opciones para el futuro.

- Documentación: Listado de los diferentes recursos utilizados para la realización del proyecto.

2. Análisis

Según los objetivos establecidos, previamente hay que comparar los sistemas similares que existen en la actualidad. En la segunda parte del capítulo, se explica la tecnología que sigue una aplicación web y los conceptos básicos que son necesarios.

2.1. Servicios actuales

Hoy en día existen diferentes y numerosos tipos de aplicaciones para la gestión de proyectos, distinto es el caso de algo tan específico como es el caso de nuestro proyecto sobre la gestión de casos clínicos, pues tanto los campos que definen el caso clínico como la gestión del mismo depende tanto de la organización como de las necesidades, además de que por regla general los sistemas que podrían hacer algo similar suelen ser privados por lo que es un caso muy complicado de encontrar y requeriría de un desarrollo totalmente a medida, por tanto no se va a comparar aplicaciones similares para este aspecto y se realizará un gestor de casos clínicos basado completamente en los requisitos que el usuario nos proporcionará. En el caso de la plataforma web en sí como fuente de información y ofrecer el servicio de laboratorio se buscarán plataformas acordes al target de usuario para sacar conclusiones y tener una idea de que es lo que hay en el mercado en este aspecto.

2.1.1 Comparativa sobre la gestión de proyectos

Los sistemas a analizar para la gestión de proyectos son Basecamp y Trello. Para su estudio se han comparado las principales características de estas dos plataformas y así sacar conclusiones sobre el gestor de proyectos para el sistema de información web del laboratorio.

Basecamp, análisis de la aplicación

Descripción

Basecamp es una plataforma que ayuda a organizar y gestionar las tareas para equipos de personas. Aparece en 2004, esta herramienta de gestión de proyectos online cuenta con 5 millones de usuarios. Ofrece herramientas básicas de gestión de proyecto (lista de tareas, compartir archivos, etc.) se basa en una receta simple: la utilidad óptima más que la multiplicación de funcionalidades y la circulación de la información fluida.

Análisis de Fortalezas y debilidades.

Fortalezas

Es una herramienta que dispone de todo el abanico de funcionalidades que se le pueden pedir a una plataforma de este tipo como pueden ser un canal de comunicaciones, herramientas para la coordinación y gestión de los proyectos (calendario de trabajo, asignación de responsabilidades…), registro de documentos y requerimientos etc… pero si hay que destacar los puntos fuertes con respecto a otras plataformas del mismo fin destacaría las siguientes:

- Diseño sencillo, intuitivo y eficaz.
- Mensajería unida al correo electrónico del usuario donde también se pueden compartir archivos. Tiene una interfaz minimalista donde la vista no se pierde y que incluso sin haber usado la plataforma con anterioridad en pocos minutos podemos familiarizarnos. Un detalle importante es que los hilos de conversación se reenvían a nuestro correo, desde donde podemos leer mensajes que nos llegan desde Basecamp y continuar con el hilo de conversación, todo esto quedará registrado además en la propia plataforma, como se aprecia en la *fig.1*.

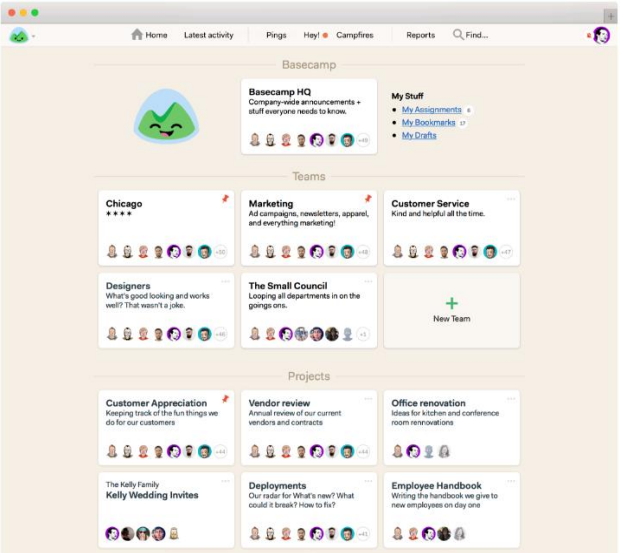


Figura 1. Basecamp

Debilidades

A veces cae en su propia sencillez y hace que sea una plataforma que se queda corta en ciertas situaciones donde la empresa o equipo necesita un mayor control, herramientas de gestión etc… como cuando hablamos de grandes equipos de trabajo o empresas que tienen varios departamentos colaborando dentro de un mismo proyecto, pero es ideal para pymes y proyectos a corto o medio plazo.

Tabla de características

Características Funcionales	**Características No Funcionales**
Mensajería	Minimalista
Compartir archivos	Intuitivo
Planifica calendarios	Eficaz
Gestionar tareas	
Establece fechas de entrega	
Control de trabajo	
Registro de documentos y requerimientos	

Tabla 1. Características Basecamp

Trello, análisis de la aplicación

Descripción

Trello es una herramienta colaborativa que organiza tus proyectos en tablones. Con un solo vistazo, Trello te permite ver en que estas trabajando, que está realizando el resto y en que parte del proceso estas. Esta herramienta sirve indistintamente para el desarrollo de proyectos propios o grupales.

Trello se basa en la metodología Kanban. Una metodología desarrollada por Toyota a principio de los años 40, dentro del sistema de gestión de producción JIT (just in time). Kanban no es un sistema de control o gestión, Es un sistema que te ayuda a determinar que vas a producir en ese momento. No existe un único tipo de Kanban, dependiendo de nuestra forma de procesar las tareas –*tarjetas*-, puede ser recomendable utilizar uno u otro.

Análisis de Fortalezas y debilidades.

Fortalezas

Trabajo colaborativo en grupo fácil basado en Kanban.

Hace la coordinación simple pero suficiente.

Se pueden adjuntar archivos directamente desde Evernote o Dropbox (además de desde el ordenador). Existe aplicación para móvil de Trello que podemos ver en la *fig.2*.

Figura 2. Trello

Debilidades

A pesar de estar un punto por encima de Basecamp en cuanto a nivel de organización, sigue siendo a veces insuficiente.
Los mayores problemas antiguamente venían de que no existía traducción al español, aunque recientemente se ha introducido a la aplicación, por lo que ha dejado de ser un problema.
Tiene una forma de organización algo más compleja que Basecamp.

Tabla de características

Características Funcionales	Características No Funcionales
Mensajería	Intuitivo
Compartir archivos	Eficaz
Gestionar tareas mediante Kanban	
Control de trabajo	
Registro de documentos y requerimientos	

Tabla 2. Características Trello

2.1.2 Comparativa sobre la plataforma web

Vamos a comparar ahora las plataformas web para el objetivo de usuario que buscamos, un perfil profesional en el sector de las enfermedades infecciosas en general, profesionales de la salud pública, estudiantes afines y público interesado, aunque este último es el menos relevante en este caso, por eso compararemos la web del laboratorio http://www.lav-asoria.com que es una web de una laboratorio sobre salud pública en general y http://www.micologiauv.com que es la web de un laboratorio de micología con lo cual acotamos un poco más el target.

Http://www.lav-asoria.com, análisis de la aplicación

Descripción

La web que vamos a analizar es http://www.lav-asoria.com. La web está dedicada la exposición de información sobre los trabajos que ellos realizan en el laboratorio y los servicios que ofrecen. Al entrar en la página se puede observar que su objetivo es abarcar un público muy específico dentro del campo de la veterinaria como son organismos de la salud pública, estudiantes de veterinaria, investigadores etc

Los usuarios podrán encontrar toda la información relacionada con las tareas que se realizan en el laboratorio a forma de informar al público su manera de trabajar, los servicios y productos que ofrece el laboratorio y un área de cliente donde los usuarios del laboratorio pueden acceder a los resultados y facturas de los diferentes pedidos que han realizado.

Análisis de Fortalezas y debilidades.

Fortalezas

Su principal fuerte está en la forma de exponer la información de la web de una manera clara y explicativa, haciendo esta información de gran valor, no solo por su exposición sino también por ofrecer al usuario una información muy concreta y difícil de encontrar. Ofrece también una sección de usuarios donde pueden ver sus facturas o los informes de los servicios que han pedido, podemos entonces resumir los siguientes puntos:

- Exposición de la información, concreta y específica en su nicho de mercado.
- Zona de usuarios del laboratorio.

Figura 3. Lav-Asoria

Debilidades

En la *fig.3* podemos ver como los colores se confunden en el menú del header lo cual quita experiencia de usuario, esto y otros detalles hacen que el diseño sea algo pobre. La organización del menú es algo confusa y la disposición también.

En resumen, son debilidades bastante significantes a nivel de experiencia de usuario y diseño, esto afecta directamente a la experiencia de los usuarios y en definitiva a su opinión sobre la web.

Tabla de características

Características Funcionales	**Características No Funcionales**
Sistema de usuarios	Información específica
Servicio de laboratorio	
Contacto	

Tabla 3. Características Lav-Asoria

Http://www.micologiauv.com, análisis de la aplicación

Descripción

La web a analizar es http://www.micologiauv.com. Se trata de la web del laboratorio de micología de la universidad de Valparaíso de Chile, al igual que el laboratorio que tratamos en el trabajo trabaja en el campo de la micología y también para una universidad por lo que su función principal es para fines formativos, investigación y servicios del laboratorio.

A diferencia de la web anterior no pone a disposición de los usuarios información específica al servicio del público de la web y la difusión que ofrece es únicamente para ofrecer los servicios del laboratorio y difundir los estudios y publicaciones que ahí se realizan.

Análisis de Fortalezas y debilidades.

Fortalezas

Su principal fuerte es que se limitan a ofrecer un servicio a la comunidad, en este caso el laboratorio, además de difundir las publicaciones que el equipo de investigación pública. Al final el usuario que va buscando un laboratorio únicamente va buscando un servicio.

- Web con una estructura coherente con su finalidad.

Figura 4. MigologíaUV

Debilidades

En la *fig.4* se aprecia que es un portal que está únicamente enfocado al servicio de laboratorio y no dispones de información adicional sobre el campo de investigación.

Tabla de características

Características Funcionales	**Características No Funcionales**
Sistema de usuarios	Información específica
Servicio de laboratorio	
Contacto	

Tabla 4. Características MigologíaUV

2.2 Conclusiones

Con los datos obtenidos, tenemos que tomar una decisión sobre que rumbo tomar en cuanto a nuestra plataforma, sobre la gestión de proyectos necesitamos que nuestra plataforma no llegue al nivel de dificultad o detalle que otra creada únicamente con dicha finalidad, buscamos algo simple, eficaz e intuitivo por lo que estos serían las características no funcionales. Sobre las funcionales la principal sería contar con un repositorio de ficheros para cada proyecto de manera individual, únicamente accesible para las personas que trabajan en ese proyecto, existirían otras funcionalidades como mensajería, gestión de tareas o notificaciones que podrían ser útiles para nuestro proyecto pero no es lo que andamos buscando ya que todo el equipo está en contacto directo a diario y no tendría un impacto realmente significativo y lo complicaría todo más o simplemente serían funciones relegadas a un segundo plano.

Sobre la plataforma web las decisiones están más claras, el sistema de usuarios es fundamental para nuestros objetivos, un panel de administración, cosa de la que carecen las aplicaciones analizadas bien sea por inaccesibilidad o desconocimiento, donde podamos controlar y gestionar los roles de nuestros usuarios, además del sistema de gestión de toda la parte clínica. Como características no funcionales el hecho de contar con información específica sobre el campo de acción del laboratorio es fundamental y es algo que encontramos en ambas comparativas, además añadir un diseño limpio e intuitivo en cuanto a organización lo veo también algo fundamental y es algo de lo que carecen las opciones que hemos analizado.

2.3 ¿Por qué Laravel?

A la hora de crear la aplicación web, nos encontramos con la siguiente duda: ¿Sobre qué framework basar nuestro desarrollo? Teniendo en cuenta los requisitos mencionados en el apartado anterior se ha realizado un análisis de los frameworks de desarrollo web más famosos buscando los puntos fuertes que nos pueden interesar y las debilidades que pueden suponer un problema para nuestro interés. Con esta comparativa se pretende esclarecer la decisión de por qué usamos Laravel para el desarrollo del proyecto en detrimento de otros frameworks.

2.3.1 Comparativa de frameworks PHP

CodeIgniter

Ventajas	Inconvenientes
Sencillo	Recursos no necesarios
Buena base de usuarios	Necesarios muchos plugins para tapar carencias
Buena documentación	Demasiado básico
Curva de aprendizaje ligera	
Muy rápido	

Tabla 5. Características Codeigniter

CodeIgniter es un framework PHP que ofrece una base simple para el desarrollo de aplicaciones. El problema es que es que puede quedarse corto para nuestro caso, ya que una vez nos enfrentamos a nuestro proyecto vemos que hay carencias en cuanto a sus modelos de datos y vemos necesario utilizar mucha parte de nuestro código en el controlador de la aplicación debido a que no cuenta con una separación de módulos, haciéndola una súper clase a cargo de casi todo. También es necesario basarnos en demasiados plugins externos para suplir algunas carencias importantes. Además, por otra parte, sigue dando soporte a versiones muy antiguas de PHP5, aunque recomiendan explícitamente no usar versiones tan antiguas, esto implica no poder aprovechar al máximo la potencia de un framework basado o enfocado plenamente en PHP7.

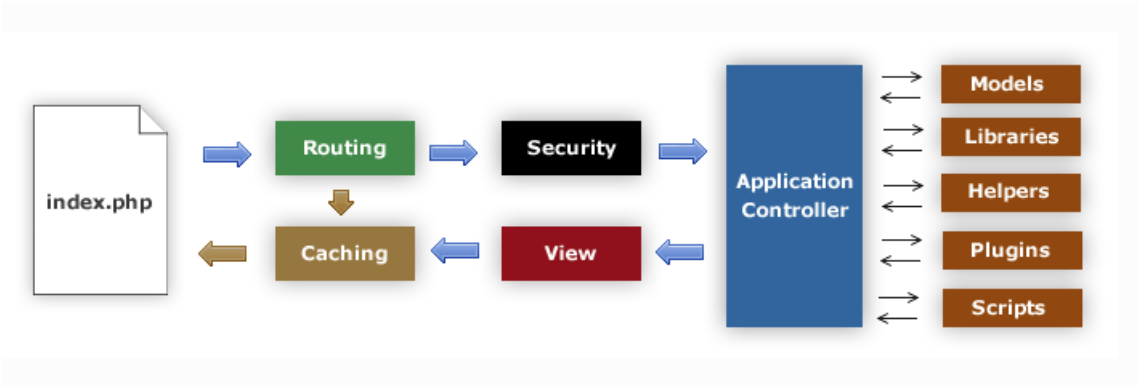


Figura 5. Flujo Codeigniter

Symfony

Cuando analizamos Symfony2 estamos hablando de uno de los framework más robustos y completos que hay actualmente. Cuando se trata de un proyecto a gran escala es una opción a tener en cuenta, pero no es nuestro caso. Para el proyecto en el que nos encontramos, si utilizáramos symfony2 estaríamos desperdiciando una gran cantidad de recursos, pues su curva de aprendizaje y dificultad entorpecería nuestro avance y desarrollo.

Ventajas	Inconvenientes
Estándar de aplicaciones web	Demasiadas opciones de recursos
Muy completo	Complejo
Robusto	Requiere de mucho tiempo para desarrollar una aplicación
Gran base de usuarios	Demasiado grande para un proyecto no corporativo
Excelente documentación	
Contiene todos los componentes	

Tabla 6. Características Symfony

En la *fig.6* podemos ver su flujo, lpotencia de Symfony2 requiere de un servidor potente para tener un rendimiento adecuado. Debido a la escasa disponibilidad de recursos para ofrecer un servicio óptimo en este framework, junto con lo mencionado anteriormente sobre la curva de aprendizaje, no se contempla como buena opción el uso de Symfony.

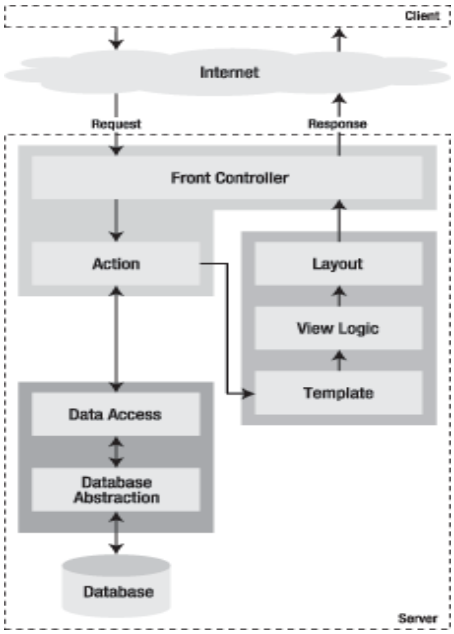


Figura 6. Flujo Symfony

Laravel

Laravel se puede considerar un framework joven que cada vez tiene más aceptación por parte de la comunidad, debido al continuo desarrollo del proyecto Laravel (versión estable cada 6 meses, denominada LTS). Una de las grandes ventajas de utilizar este framework para el desarrollo de aplicaciones es la gran comunidad que lo soporta. Gracias a esto se puede contar con una excelente documentación de respaldo y unos foros (Laracast o Styde) dedicados a ayudar a los usuarios al aprendizaje y entendimiento del framework. La curva de aprendizaje es moderada y anima a seguir trabajando con el framework a medida que se van descubriendo implementar partes de la aplicación que se está construyendo gracias a sus patrones de diseño elegantes y sencillos, en la *tab.7* podemos ver sus ventajas e inconvenientes, así como su flujo en *la fig.7*.

Ventajas	Inconvenientes
Framework moderno	Framework joven
Composer	ORM propio
POO poderna	Plantillas propias
Excelente documentación	
Gran comunidad	
Patrones de diseño elegantes	
Curva de aprendizaje moderada	
Módulos externos fácilmente acoplables	

Tabla 7. Características Laravel

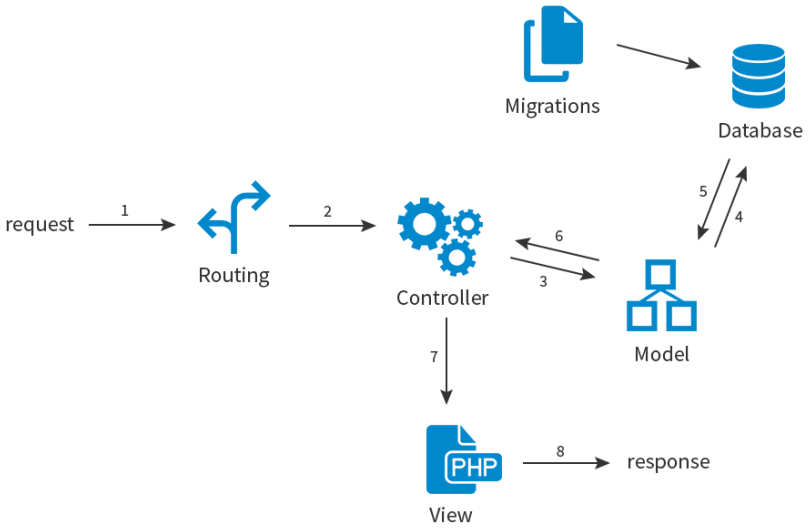


Figura 7. Flujo Laravel

2.3 Plataforma para el Despliegue

En este apartado vamos a hacer una breve comparación entre dos plataformas de hosting como son Google App Engine y Heroku, a fin de elegir una con la que realizar el despliegue de la aplicación en esta plataforma.

2.3.1 Comparativa

Google App Engine

Google App Engine es otro de los servicios que conforman la familia de Google Cloud Platform. Este servicio es del tipo Plataforma como Servicio o Platform as a Service (PaaS), nos permite publicar aplicaciones web en línea sin necesidad de preocuparnos por la parte de la infraestructura y con un enfoque 100% en la construcción de nuestra aplicación y en la posibilidad de correrla directamente sobre la infraestructura de Google, es decir, la que Google usa para sus propios productos.

Como cualquier otra Plataforma como Servicio, App Engine nos facilita construir, mantener y escalar nuestra aplicación en la medida que sea necesario.

Otra de sus características es que las aplicaciones pueden ser escritas en lenguajes como Java, Python, PHP y Go.

Cuando usamos Google App Engine (GAE) no nos tenemos que preocupar por la escalabilidad de nuestra aplicación ya que cuenta con un balanceador de carga y escalamiento automático.

Así nuestra aplicación solamente será atendida por las máquinas necesarias para tener un perfecto comportamiento y para que la respuesta de nuestra aplicación sea la óptima.

Ventajas	**Inconvenientes**
● Acceso al resto de servicios de Google. ● Lidia con tareas asíncronas de manera sólida. ● Usa la propia infraestructura en la nube de Google.	● Falta de flexibilidad en la conexión con la plataforma. ● No usa un estándar de base de datos SQL.

Tabla 8. Características Google App Engine

Heroku

Heroku es uno de los PaaS (Platform as a service) más utilizados en la actualidad en entornos empresariales por su fuerte enfoque en resolver el despliegue de una aplicación. Además, te permite manejar los servidores y sus configuraciones, escalamiento y la administración. A Heroku solo le dices qué lenguaje de backend estás utilizando o qué base de datos vas a utilizar y te preocupas únicamente por el desarrollo de tu aplicación.

Heroku tiene su clientela bien definida: empresas que quieren dejar de preocuparse por cuestiones de infraestructura y sólo enfocarse en el desarrollo. Por lo general estas suelen ser empresas grandes o startups que prefieren no invertir en un equipo de operaciones cuando están en una etapa temprana, y su prioridad debe ser hacer un producto que las personas quieran.

Heroku tiene dos tiers, o niveles, para personas interesadas en aprender: una versión gratuita similar a la de now.sh, que entra en modo “sleep” cada 30 minutos sin tráfico, y otra de pago que compite con el servicio básico que ofrece Digital Ocean u otras plataformas similares, pero agrega las ventajas de que nuestros servidores sean administrados por nosotros.

Ventajas	**Inconvenientes**
● Ofrece estándar SQL. ● Relativamente fácil de desplegar. ● Modelo de precios barato y con versión gratuita.	● No ofrece la cantidad de servicios que ofrece Google App Engine.

Tabla 9. Características Heroku

2.4 Requisitos

Para el desarrollo de este proyecto se analizaron previamente unos requisitos y unas funcionalidades que debía tener el producto final. Estos requisitos y funcionalidades se han ido cumpliendo conforme el proyecto iba avanzando y tomando forma.

2.4.1 Tabla de Dummies

Con el propósito de conseguir sacar nuevos requisitos vamos a definir una tabla de dummies con diferentes perfiles de usuarios potenciales del sistema de información.

Nombre	Nivel de estudios	Ocupación	Motivación	Edad	Tiempo Libre	Frecuencia de consulta
Juan	Bachiller	Estudiante	Buscar información sobre micología.	17	Medio	Anual
Begoña	Universitarios	Veterinaria	Añadir un caso clínico a la plataforma.	54	Poco	Diaria
Soraya	Universitarios	Veterinaria	Buscar una publicación.	52	Poco	Semanal
Fernando	Secundaria	Granjero	Enviar muestra al laboratorio.	59	Poco	Semanal
Jose	Universitarios	Técnico de laboratorio	Contactar con el laboratorio.	36	Poco	Mensual
Lorenzo	Universitarios	Dueño de una clínica veterinaria	Comparar precios de diferentes laboratorios.	28	Poco	Semanal
Davinia	Bachiller	Estudiante	Buscar un proyecto de investigación	20	Medio	Anual

Tabla 10. Dummies

En definitiva, podemos comprobar que el nivel de estudios y la ocupación es un dato relevante pues refleja como bien se ve en la tabla que el cliente potencial de la página web suele ser alguien que maneja conocimientos científicos y esto conlleva a que tengan un nivel de estudios, por lo general, de Bachiller o superior.

La edad es un dato importante pues nos indica que es una página destinada a un público adulto entre los 25 y los 55 años, así mismo el tiempo libre por normal general es poco, lo cual es una información relevante a la hora de construir los cimientos de nuestro portal puesto que esto nos obliga a pensar en una buena usabilidad del sistema y que no sea algo complicado de aprender.

Por último, la frecuencia de consulta es interesante pues nos hace ver que sólo las personas con perfiles más cercanos al campo de la veterinaria, y más concretamente los usuarios del laboratorio, son los que usan de forma recurrente el sistema de información.

2.4.2 Requisitos funcionales

Se expondrán de manera breve los principales requisitos funcionales del sistema, de manera que se puedan ver las principales funcionalidades que se plantean desarrollar:

- El sistema ofrecerá información relevante sobre la micología veterinaria.
- El sistema ofrecerá información asociada a los proyectos que se llevan a cabo en el laboratorio.
- El sistema ofrecerá información y contenidos descargables sobre el servicio de laboratorio.
- El sistema controlará el acceso y lo permitirá tanto a usuarios autorizados como anónimos.
- El sistema sólo permitirá el acceso a las áreas de gestión mediante autenticación y permisos previos.
- El sistema tendrá un área de gestión para los casos clínicos con las funcionalidades de Crear, Ver, Editar y Eliminar.
- El sistema tendrá un área de gestión para los antibióticos con las funcionalidades de Crear, Ver, Editar y Eliminar.
- El sistema tendrá un área de gestión para los proyectos con las funcionalidades de Crear, Ver, Editar y Eliminar.
- El sistema tendrá un panel de administración solamente accesible para el administrador del sistema

- El sistema podrá enviar un caso clínico, al email descrito en el mismo, una vez esté finalizado.
- Desde el panel de administración se podrá asignar permisos y roles a los usuarios ya existentes.
- Se podrá contactar con el administrador del sistema rellenando y enviando previamente un formulario de contacto.

2.4.3 Requisitos no funcionales

Se va a trabajar para que la plataforma sea:

- Accesible: La aplicación tiene que ser accesible a través de diversos dispositivos, tanto móviles o tablets como desde ordenadores. Para ello se utilizará “Responsive web design” que permite adaptar el diseño de la web a cada uno de estos dispositivos sin tener que realizar una aplicación específica para cada uno de ellos.
- Extensibilidad: La plataforma creada tiene que ser ampliable para posibles mejoras de futuro.
- Sencilla: La aplicación tiene que ser fácil de entender y que un usuario se familiarice con las principales funcionalidades de la aplicación en 1 o 2 horas.
- Seguridad: El sistema debe garantizar el control en el acceso, utilizando la autentificación de los usuarios para la administración del mismo, además de un sistema de permisos y roles de usuario.

2.4.4 Casos de uso

Hemos dividido los casos de uso en 4 roles en función de los diferentes tipos de usuarios a los que va enfocada la aplicación web.

Anónimo: Usuario sin identificar que accede a la plataforma sin privilegios, ver en *fig.8*.

Usuario: Usuario identificado sin privilegios, ver en *fig.9*.

Investigador: Usuario identificado como investigador del laboratorio, ver en *fig.9* y *fig.10*.

Administrador: Usuario identificado como administrador en la aplicación web, ver en *fig.10*.

Casos de uso Usuario Anónimo

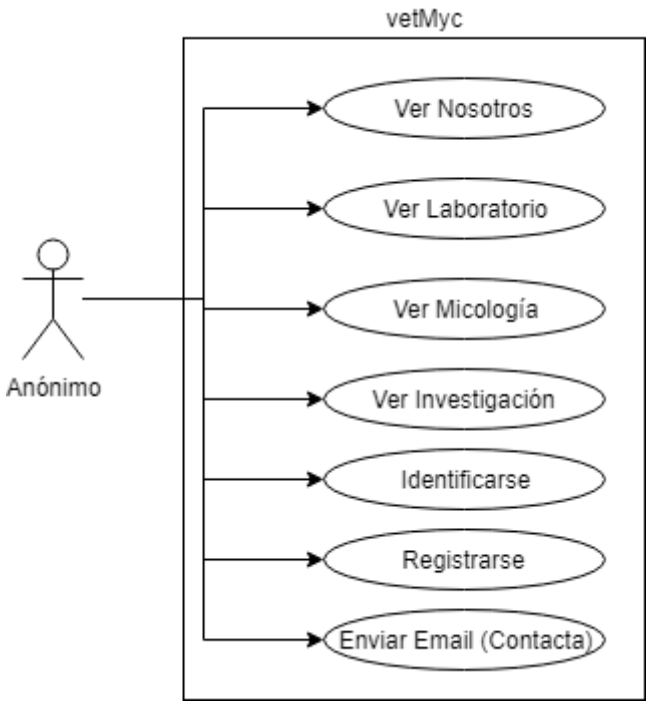


Figura 8.Casos de uso usuario anónimo

Cualquier usuario que fuera a acceder a la aplicación web, sólo podrá acceder al portal web y su información además de identificarse, registrarse y enviar el formulario de contacto.

Casos de uso Usuario e Investigador

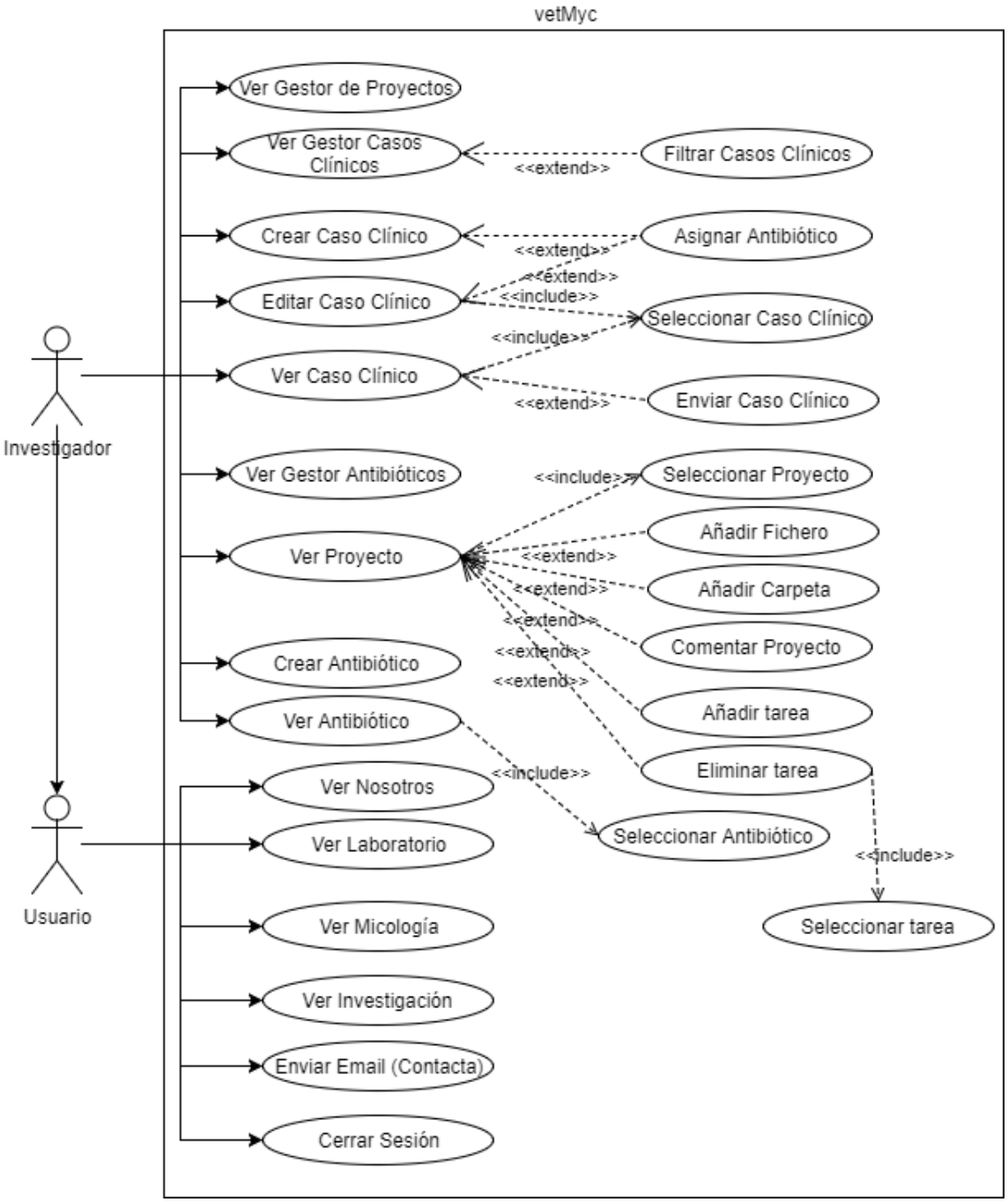


Figura 9. Casos de uso usuario e investigador

Un usuario es aquel que se identifica mediante su usuario y contraseña en la aplicación. Este tipo de usuarios dispone de los mismos privilegios de un usuario anónimo sólo que al estar identificado en lugar de realizar las acciones de identificarse o registrarse puede cerrar sesión. Este tipo de usuarios quedan a disposición del administrador del sistema para asignarles un rol posteriormente.

Un investigador puede acceder a la gestión de los proyectos con los que colabora, dentro de cada proyecto dispone de un gestor de ficheros donde puede subir o crear carpetas, un log de comentarios y un gestor de tareas.

Un investigador puede acceder a todas las funcionalidades de un usuario, pero ya dispone de algunas de las funcionalidades de las herramientas de la plataforma para gestionar casos clínicos, antibióticos o proyectos.

Este rol es el que queda inmediatamente por debajo del administrador dentro de las jerarquías de roles de usuario.

Dispone de la mayoría de funcionalidades a excepción de borrar cualquier registro de la base de datos, editar antibióticos o editar y crear proyectos.

Puede además filtrar casos clínicos por:

-N.º de caso clínico.

-Casos que cumplen la condición de tener “Aislamiento Fúngico”.

-Casos que cumplen la condición de tener “Aislamiento Bacteriano”.

-Casos que están en un estado “Terminado”.

-Casos que están en un estado “En progreso”.

-Localización de la muestra (pelo, ojo, piel etc..).

Y al realizar una edición o creación de un caso clínico se puede asignar si un antibiótico en concreto es resistente, sensible o intermedio para la infección en cuestión.

Casos de uso Administrador

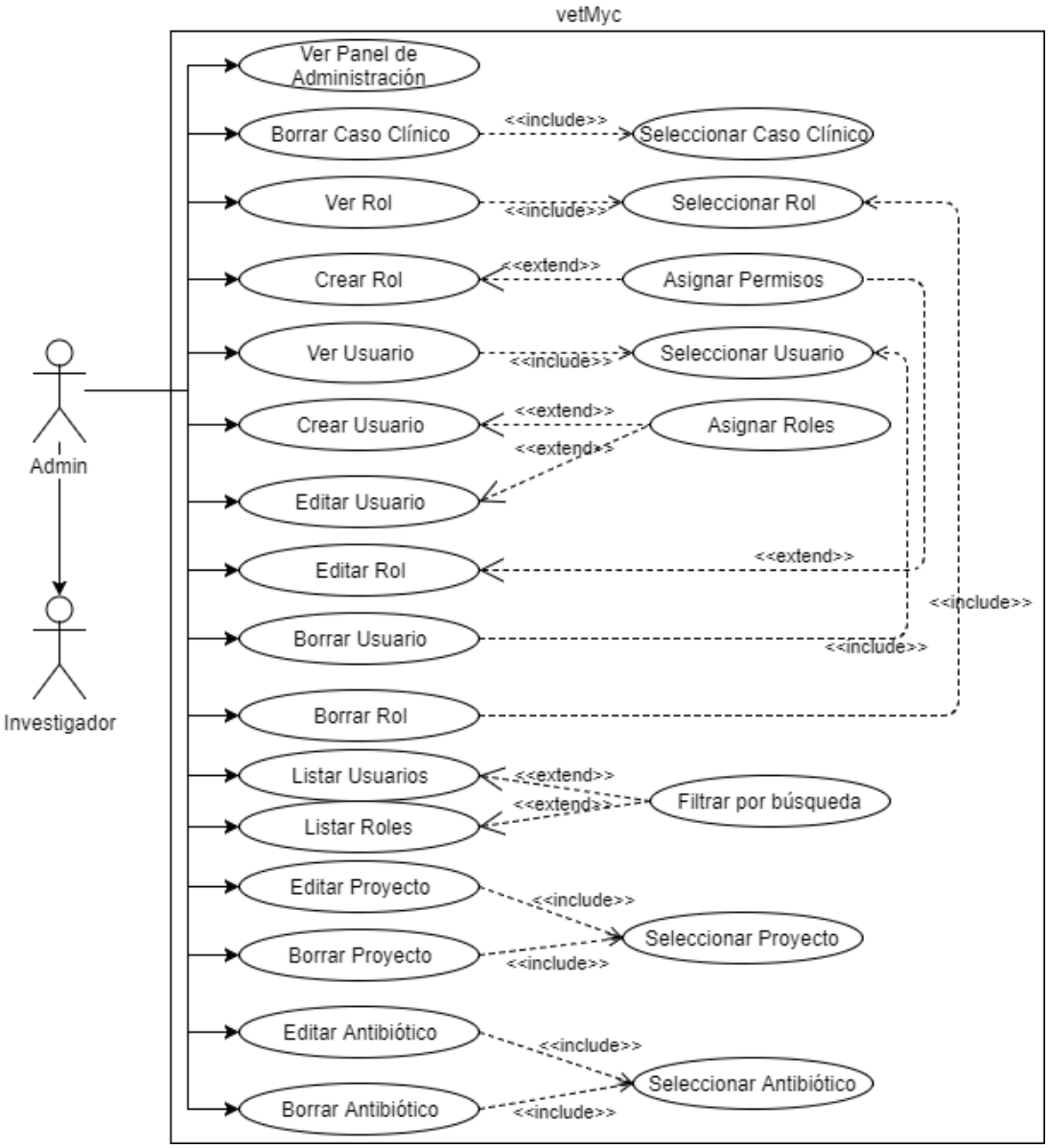


Figura 10. Casos de uso investigador y administrador

El administrador del sistema puede acceder a todas las funcionalidades que dispone la plataforma.

Es el único rol que puede entrar al panel de administración el cual permite acceder a todos los modelos de datos de la aplicación y gestionarlos.

Tiene la flexibilidad de crear roles y asignar los permisos que quiera y de crear usuarios y asignar cualquier rol existente, puede filtrar por búsqueda tanto roles como usuarios. Además de las funcionalidades de editar, ver y eliminar tanto roles como usuarios.

2.4.4 Especificaciones de casos de uso

A continuación, se realizará una especificación de los casos de uso más relevantes dentro del sistema:

26	**Envío de un caso clínico vía email**		
Descripción	El sistema deberá permitir al Administrador e Investigadores en la vista del caso clínico enviarlo mediante email a su destinatario según se describe en el siguiente caso de uso:		
Precondición	Administrador o Investigador autenticado en el sistema Usuario situado en la vista del caso clínico		
Secuencia Normal	**Paso**	**Acción**	
	1	Pulsar el botón 'Enviar Email'	
	2	El sistema procede al envío	
		2a	Si el caso clínico aún no está en estado 'Terminado' el sistema deberá mostrar un mensaje de error mostrando, "El caso clínico debe aún está en progreso."
		2b	Si el caso clínico está en estado 'Terminado' el sistema deberá mostrar un mensaje afirmativo mostrando, "El caso se ha enviado correctamente."
Postcondición	Se envía un correo con todos los datos del caso clínico al usuario correspondiente.		
Excepciones	**Paso**	**Acción**	
	1	En el caso de que un usuario sin permisos fuera a acceder a la ruta para realizar el envío debe ser redirigido a la página principal.	
Rendimiento	El sistema deberá realizar la acción descrita del paso 1 al 2 en un máximo de 1.5 segundos		
Frecuencia	Este caso de uso se espera que se lleve a cabo una media de 2 veces al día		
Importancia	Importante		
Urgencia	-		
Comentarios	-		

Tabla 11. Especificación caso de uso no. 26

11	**Crear Proyecto**		
Descripción	El sistema deberá permitir a un usuario administrador en el gestor de proyectos, crear un proyecto según se describe en el siguiente caso de uso:		
Precondición	Administrador autenticado en el sistema Administrador situado en la vista de creación de proyecto.		
Secuencia Normal	**Paso**	**Acción**	
	1	Rellenar los campos requeridos en el formulario.	
	2	Pulsar en el botón de 'crear' Proyecto.	
	3	El sistema valida los campos enviados.	
		2a	Si los campos introducidos son validados correctamente, el sistema procede al siguiente paso.
		2b	Si los campos introducidos no se validan correctamente, el sistema muestra el error de validación en los campos específicos.
	4	El sistema añade el nuevo proyecto.	
Postcondición	-		
Excepciones	**Paso**	**Acción**	
	1	En el caso de que un usuario sin permisos fuera a acceder a la ruta para realizar el envío debe ser redirigido a la página principal.	
Rendimiento	El sistema deberá realizar la acción descrita del paso 3 al 4 en un máximo de 1.5 segundos		
Frecuencia	Este caso de uso se espera que se lleve a cabo una media de 1 vez por mes		
Importancia	-		
Urgencia	-		
Comentarios	-		

Tabla 12. Especificación caso de uso no. 11

28	Asignar antibióticos a un caso clínico	
Descripción	El sistema deberá permitir a un usuario administrador en momento de crear un caso clínico asignar un antibiótico según se describe en el siguiente caso de uso:	
Precondición	Administrador autenticado en el sistema Administrador situado en la vista de creación o edición de un caso clínico. Deben existir antibióticos en la base de datos.	
Secuencia Normal	**Paso**	**Acción**
	1	Marcar el checkbox de una de las columnas (sensible, intermedio o resistente).
	2	Pulsar en el botón de 'crear' o 'editar' caso clínico.
	3	El sistema guarda las asociaciones correspondientes.
Postcondición	-	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	El sistema deberá realizar la acción descrita del paso 2 al 3 en un máximo de 1.5 segundos	
Frecuencia	Este caso de uso se espera que se lleve a cabo una media de 3 veces al día	
Importancia	-	
Urgencia	-	
Comentarios	-	

Tabla 13. Especificación caso de uso no. 28

2.5 Mockups, diseño de la paleta de colores y logotipo

Mockup es un modelo a escala o de tamaño completo de un diseño, utilizado para la enseñanza, la demostración, la evaluación del diseño, la promoción y otros fines, en nuestro caso las hemos utilizado para el diseño de las vistas principales del proyecto.

2.5.1 Inicio

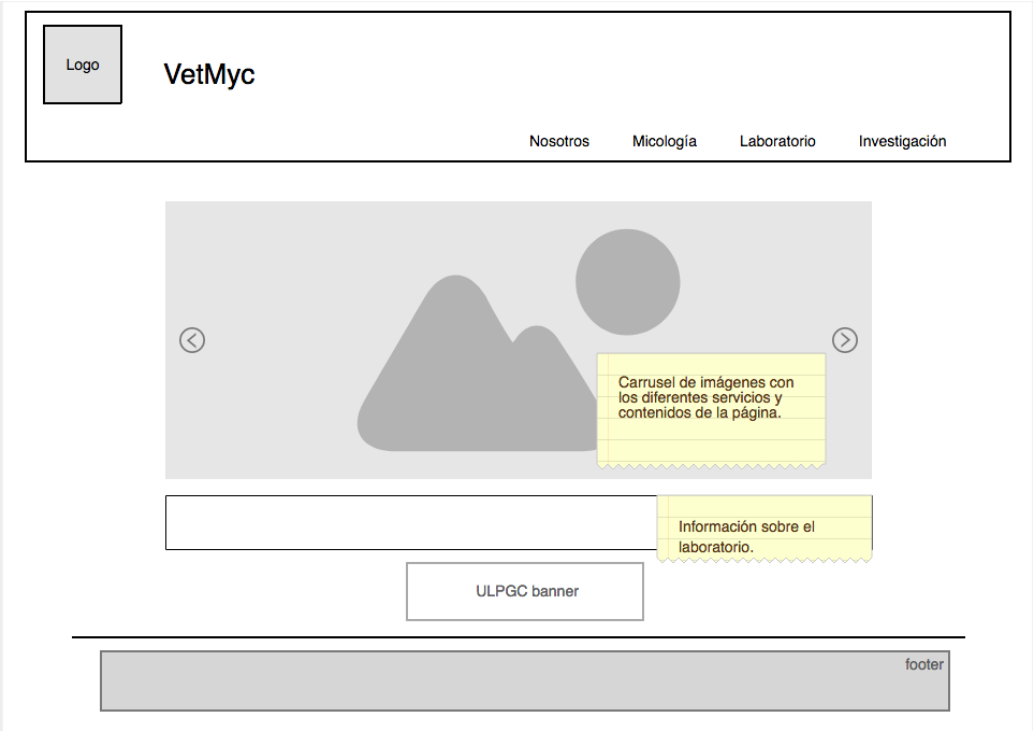


Figura 11. Mockup inicio

2.5.2 Gestor proyectos

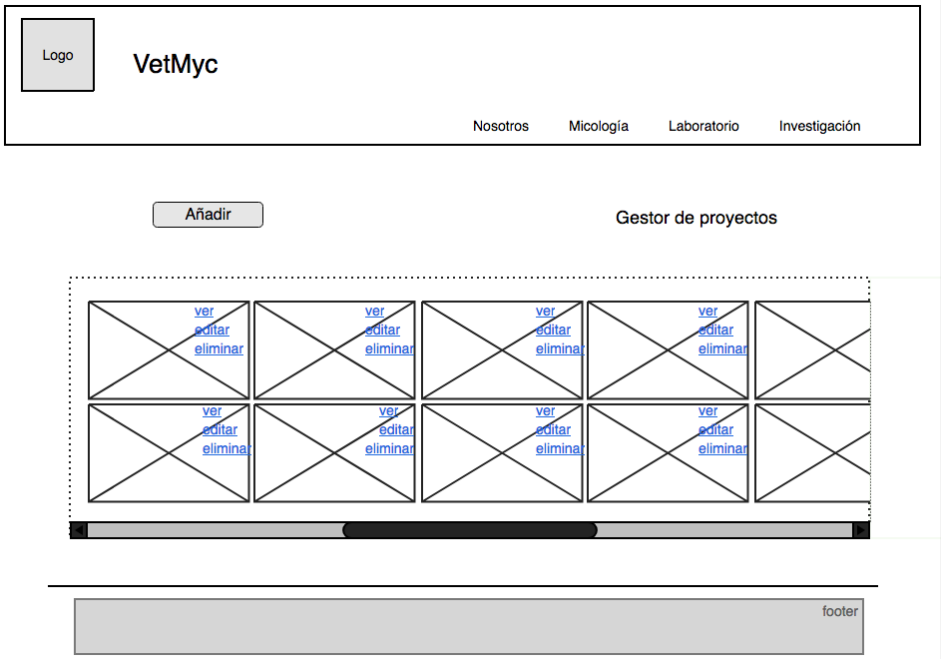


Figura 12. Mockup gestor de proyectos

2.5.3 Laboratorio

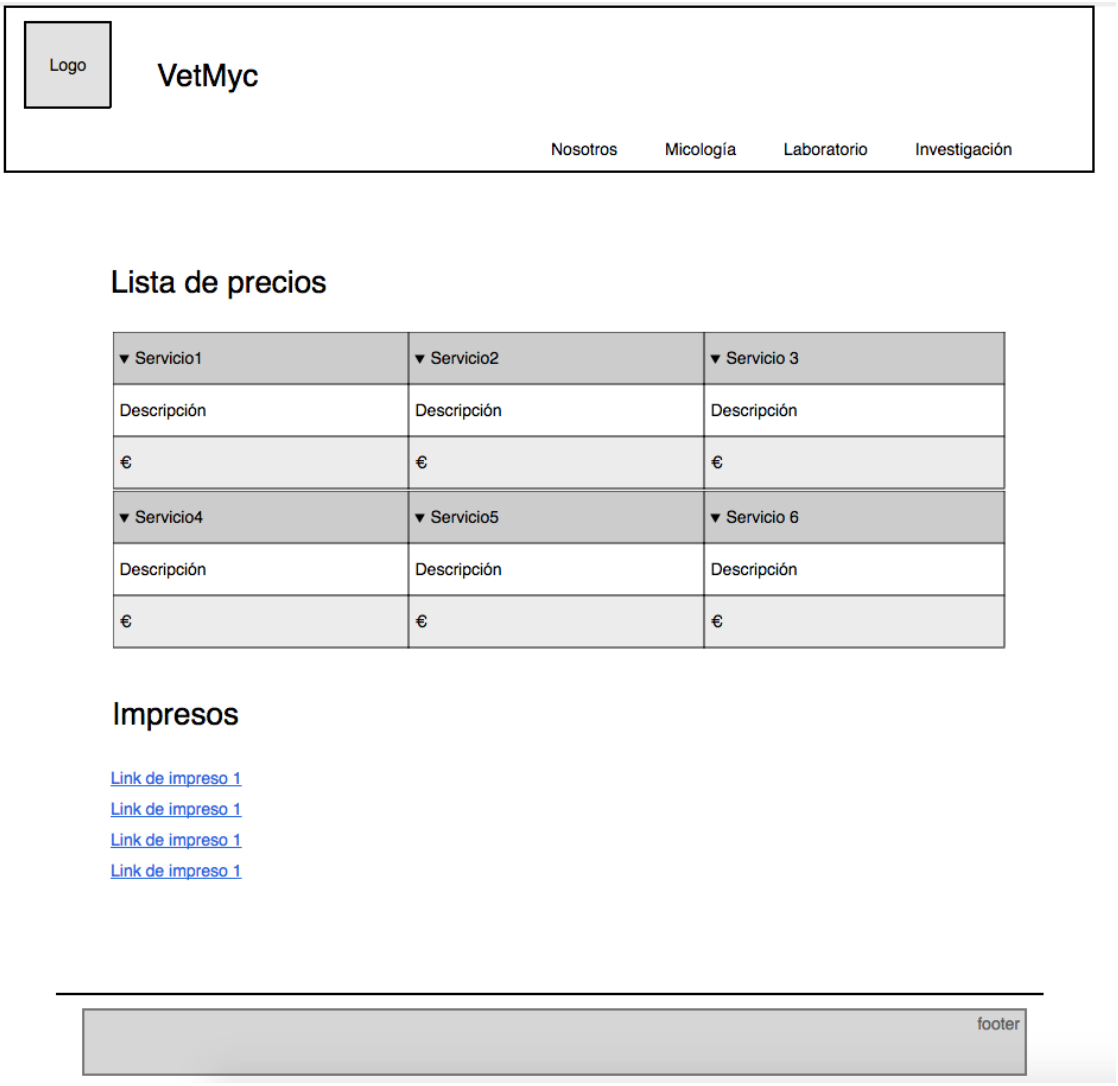


Figura 13. Mockup laboratorio

2.5.4 Gestor de casos clínicos

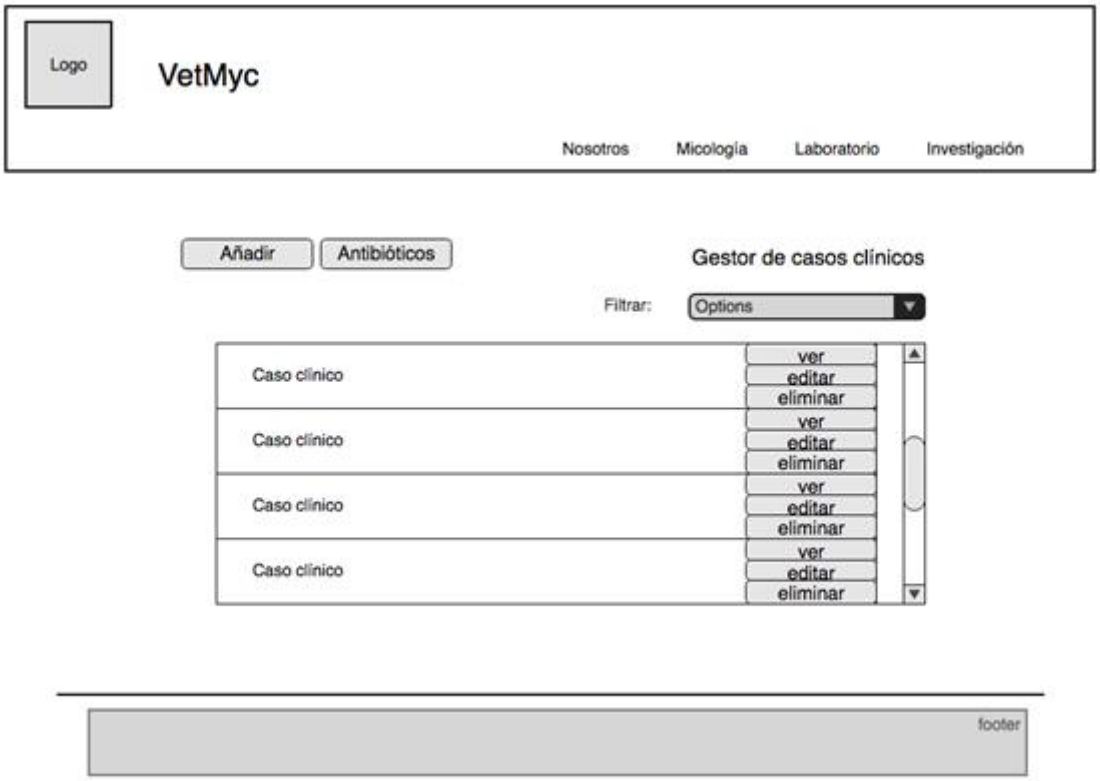


Figura 14. Mockup gestor de casos clínicos

2.5.5 Paleta de colores

Al ser una página cuyo principal activo a ofrecer es la información, consideramos oportuno un esquema de colores monocromático, utilizando tonalidades grises y blancos para complementar el esquema, dándole a la información y fotografías todo el protagonismo también siguiendo un esquema más bien minimalista. Resaltando sobre todo el nombre de la página y a las secciones donde estemos navegando en cuanto a saturación y dotando a las partes que menos protagonismo queremos darle como puede ser el fondo del header una opacidad mezclándolo con el fondo blanco consiguiendo un efecto suave y sintonizado con el fondo.

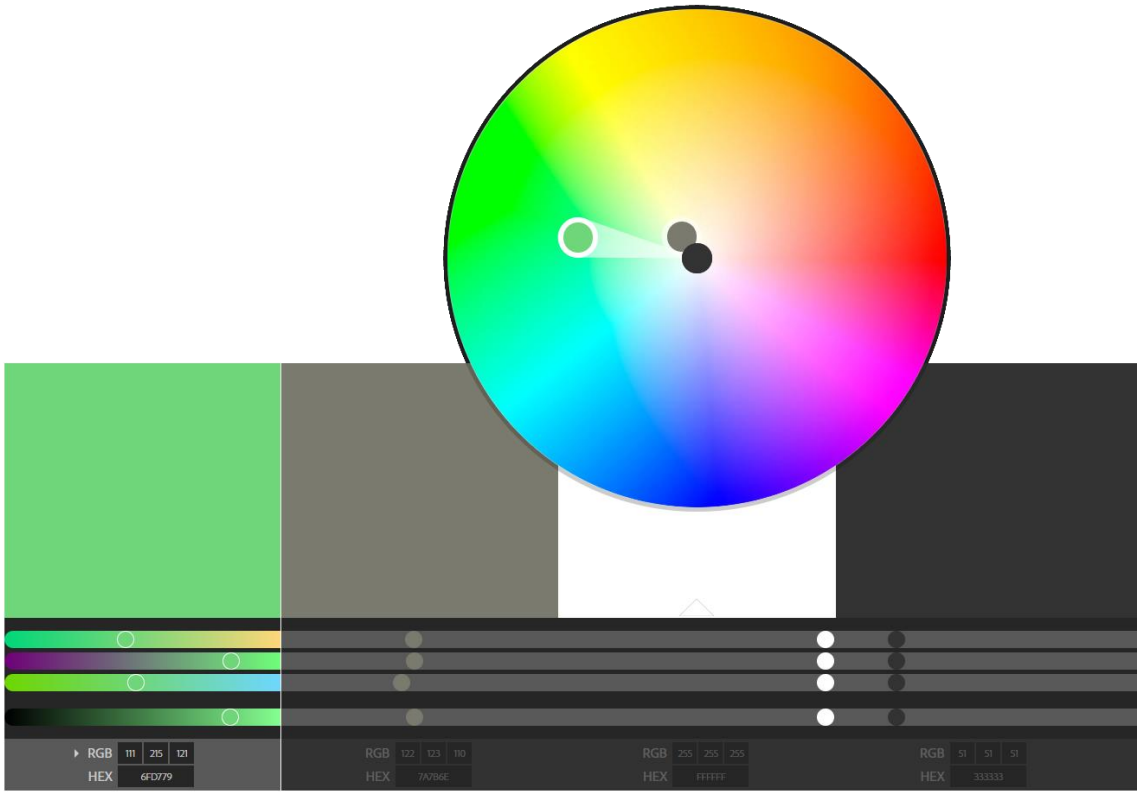


Figura 15. Paleta de colores

En la *fig.15* podemos ver el esquema de colores, donde utilización del verde (#6FD779) creemos que es el mejor color para definir este portal pues es un color asociado a la veterinaria, además de a la salud, vida o la curación, queremos transmitir eso tanto al usuario del día a día como el que no esté acostumbrado a este ámbito (ya que por lo general suelen ser desagradables algunas imágenes para el público menos allegado).En conjunción con negro (#333333), gris (#7A7B6E) y blanco (#FFFFFF), conseguimos simplicidad para el esquema, claridad para la información y profesionalidad. Además de que en la mayoría de webs de este tipo encontramos colores azules bastante intensos que solemos asociarlos a la identidad corporativa, nos hemos querido desmarcar

y proponer algo diferente y con sentido en base al esquema e idea que queremos realizar.

2.5.6 Logotipo

Para el diseño del logotipo se utilizó el esquema de colores básicos de la web, utilizando la idea de usar las placas de medio de cultivo que es un elemento muy común en este tipo de laboratorios y asociarlo con la veterinaria, como se puede observar en la *fig.16*.

Figura 16. Logotipo Vetmyc

2.6 Normativa y legislación

2.6.1 Ley de propiedad intelectual (Real Decreto Legislativo 1/1996, de 12 de abril)

En primer lugar, toda página web tiene que respetar la Ley de Propiedad Intelectual. La página web como tal es una creación del intelecto susceptible de protección como propiedad intelectual. Por ello, nuestro diseño debe ser original y debemos tener la licencia de uso de aquellos complementos (plataformas, plantillas, widgets…) que se hayan usado para su configuración o usar complementos de código abierto. Usar desarrollos de otro sin su autorización o sin licencia es una infracción de su derecho de propiedad intelectual.

Deberíamos registrar nuestra página web en el registro de propiedad intelectual quedando protegidos tanto el diseño y estructura como los contenidos publicados, ya sean texto o gráficos.

Especial atención en este punto deberemos tener con las imágenes que usemos en nuestra web. Si no fuéramos los autores, deberíamos obtener autorización para usarlas o bien asegurarnos de que tienen una licencia de uso abierta (Creative Commons, Copy Left…) ya que la publicación de imágenes o texto en una web constituyen actos de reproducción y comunicación pública.

2.6.2 Ley orgánica de protección de datos (Ley Orgánica 15/1999, de 13 de diciembre)

En la mayoría de este tipo de proyectos se tratan datos: Desde la mención del nombre de una persona hasta un formulario de contacto. La LOPD obliga no sólo a la custodia y seguridad del tratamiento sino también a comunicaciones administrativas, establecimiento de protocolos de actuación y formas de información al ciudadano. Realizar por ejemplo un documento de seguridad de la aplicación sería una buena práctica.

Además, nuestro sistema de información usa cookies por lo que toda persona que navegue a través de la plataforma debería ser informada sobre su uso.

2.6.3 Ley de marcas (Ley 17/2001, de 7 de diciembre)

En materia de propiedad industrial, cuando los elementos distintivos de la página web alcanzan una cierta relevancia es muy aconsejable protegerlos procediendo al registro de la marca y del logotipo para asegurar defensa frente a intromisiones ilegítimas. Más aún cuando la imagen se usa como merchandising y se comercializa. Además, no debe olvidarse desarrollar un adecuado sistema de licencias de los diseños para el uso comercial de los mismos. Además, la protección de marca también protegería nuestro dominio.

3. Desarrollo

3.1 Alcance de la implementación

La implementación del sistema de información se trata de un prototipo, por lo que no deja de ser un modelo del comportamiento del sistema cuya finalidad es clarificar los requerimientos, además de aportar la totalidad o no de las funcionalidades vistas previamente en el análisis y diseño de la aplicación.

3.1.1 Desarrollo de tests unitarios

Para el desarrollo de este proyecto se ha seguido una metodología TDD con tests unitarios realizados con la librería phpUnit. Una prueba unitaria es una forma de comprobar el correcto funcionamiento de una unidad de código. Esto sirve para asegurar que cada unidad funcione correctamente y eficientemente por separado. Además de verificar que el código hace lo que tiene que hacer, verificamos que sean correctas las rutas a la que se accede o se redirecciona y si los códigos o los métodos de petición HTTP son los esperados para la funcionalidad que estamos testando, incluso verificar si en la base datos se encuentran los datos correctamente o no.

La idea es escribir casos de prueba para cada función no trivial o método, de forma que cada caso sea independiente del resto. A continuación, se explicará brevemente como se ha realizado la preparación del entorno y la base de datos para usar el TDD.

Laravel ya incluye la librería phpUnit, podemos comprobar que efectivamente está funcionando tal y como se muestra en la *fig.17*, usando los comandos:

```
phpunit ó vendor/bin/phpunit
```

Figura 17. Inicio phpUnit

Vamos a realizar una separación entre la base de datos que tenemos en producción y la base de datos que vamos a usar para los tests, para ello vamos a utilizar una base de datos sqlite almacenada en memoria, el fichero de configuración de la base de datos de laravel debería contener lo que se muestra en la *fig.18*,

```
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'database' => ':memory:',
        'prefix' => '',
    ],

    'mysql' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', 'eu-cdbr-west-01.cleardb.com'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'XXX'),
        'username' => env('DB_USERNAME', 'XXX'),
        'password' => env('DB_PASSWORD', 'XXX'),
        'charset' => 'utf8mb4',
        'collation' => 'utf8mb4_unicode_ci',
        'prefix' => '',
        'strict' => true,
        'engine' => null,
    ],
],
```

Figura 18. Config/database.php

Y conectar nuestro phpunit a la nueva base de datos sqlite, tal y como se muestra remarcado en la *fig.19*,

Figura 19. phpunit.xml

Ahora todos nuestros tests interactuarán con una base de datos sqlite que se guardará en memoria, la instancia de la base de datos deja de existir tan pronto como se cierre la conexión, por lo tanto, es ideal para realizar cada prueba de forma aislada.

Los siguiente es realizar los tests unitarios, para este caso he tomado como ejemplo los realizados para crear un proyecto tal y como se muestra en la *fig.20*, tenemos 4 tests para comprobar que la funcionalidad de crear proyecto funciona según lo esperado;

- testCreateProjectFailWithoutLoginUser, que como su propio nombre indica testea que un usuario sin estar logeado no puede crear un proyecto.
- testCreateProjectFailWithoutRequiredParameterAsAdmin, que testea que sin un parámetro requerido en el formulario no se puede crear el proyecto.
- testCreateProjectSuccessAsAdmin, se comprueba como siendo un usuario administrador del sistema se puede crear un proyecto, pues es un rol que tiene los permisos necesarios.
- testCreateProjectFailLogedAsUser, en este último se comprueba como un usuario registrado sin permisos no puede crear un proyecto y se le redirige al home de la web.

```php
use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class CreateProjectTest extends TestCase
{
  // use WithoutMiddleware;

  public function testCreateProjectFailWithoutLoginUser()
  {
    $project = factory('App\Project')->make()->toArray();
    $response = $this->call('POST', '/project-manager/create', $project, [], [], []);
    $response->assertStatus(302)->assertRedirect('/');
  }

  public function testCreateProjectFailWithoutRequiredParameterAsAdmin()
  {
    $project = factory('App\Project')->make(['project_name' => ''])->toArray();

    $user = $this->createUserWithAdminPermissions('projects');

    $response = $this
      ->actingAs($user)
      ->post('/project-manager/create', $project);

    $response->assertRedirect('/project-manager/create')
      ->assertSessionHasErrors(['project_name']);
    $this->assertDatabaseMissing('projects', $project);
  }

  public function testCreateProjectSuccessAsAdmin()
  {
    $image = \Illuminate\Http\UploadedFile::fake()->create('image.png', $kilobytes = 1);
    $project = factory('App\Project')->make(['image' => $image])->toArray();
    $user = $this->createUserWithAdminPermissions('projects');

    $response = $this
      ->actingAs($user)
      ->post('/project-manager/create', $project);
    $response->assertRedirect('/project-manager')
      ->assertStatus(302);
    $this->assertDatabaseHas('projects', array_splice($project, 0, 1));
  }

  public function testCreateProjectFailLogedAsUser()
  {
    $project = factory('App\Project')->make()->toArray();

    $user = $this->createUserWithUserPermissions();

    $response = $this
      ->actingAs($user)
      ->post('/project-manager/create', $project);

    $response->assertRedirect('/');
    $this->assertDatabaseMissing('projects', $project);
  }
}
```

Figura 20. createProjectTest.php

en todos estos tests se llama al factory que con la ayuda de una librería llamada faker nos genera una instancia de proyecto de manera aleatoria según la *fig.21*,

```
$factory->define(App\Project::class, function (Faker\Generator $faker) {
        return ['project_name' => $faker->lastname,
         'description' => $faker->paragraph(1),
         'image' => 'settings/March2017/eJo0qPme5R7np2HtdjDT.png',
         'project_type' => $faker->word,
         'research_line' => $faker->word,
         'publication_date' => Carbon::create(2015, 5, 28, 0, 0, 0),
         'entity' => 'ULPGC',
         'author_id' => 1,
         'project_status' => $faker->randomElement($array = array ('inprogress','finished')),
     ];
});
```

Figura 21. Project factory

y llamamos al método post en la ruta /project-manager con el proyecto generado previamente y esperamos obtener una respuesta Http acorde a lo esperado, que en todos los casos va a ser una redirección, lo que cambia es la ruta en cuestión que nos permitirá saber si la funcionalidad realiza los esperado o no.

El procedimiento a seguir para el TDD sería implementar la funcionalidad, que en este caso es crear un proyecto, y realizar una prueba con phpUnit para comprobar que lo que se ha implementado se corresponde con los esperado y si no es así volver al desarrollo y de esta manera hacer un procedimiento cíclico hasta que finalmente los tests se corresponden con la implementación.

Finalmente, el resultado esperado una vez finalizamos los tests unitarios sería el mostrado en la *fig.22*:

```
E:\laragon\www\vetMyc (master){hg}
λ vendor\bin\phpunit
PHPUnit 5.7.19 by Sebastian Bergmann and contributors.

...............................................................  63 / 110 ( 57%)
..................................................               110 / 110 (100%)

Time: 44.86 seconds, Memory: 78.00MB

OK (110 tests, 390 assertions)
```

Figura 22. Final phpUnit

3.1.2 Funcionalidades implementadas.

En la *tab.14* se enumeran las diferentes características implementadas para el prototipo, basándonos en los casos de uso definidos en el análisis del sistema de información realizado previamente y a su lado el fichero que contiene el test de la funcionalidad, todos estos test se pueden encontrar en el Anexo I correspondiente a los tests unitarios de la aplicación.

Funcionalidades	Tests
1. Ver Nosotros (Bienvenido, Servicios, Contacto y Encuéntranos)	tests\Feature\Home\HomeTest.php
2. Ver Laboratorio (Precios e Impresos)	tests\Feature\Home\HomeTest.php
3. Ver Micología (Generalidades, Malassezias, Criptococosis, Candidiasis, Aspergilosis, Dermatofitos y Dimórficos)	tests\Feature\Home\HomeTest.php
4. Ver Investigación (Equipo y Proyectos)	tests\Feature\Home\HomeTest.php
5. Identificarse	tests\Feature\Auth\LoginTest.php
6. Registrarse	tests\Feature\Auth\RegistrationTest.php
7. Cerrar sesión	tests\Feature\Auth\LoginTest.php
8. Contactar (email)	tests\Feature\Home\HomeTest.php
9. Ver gestor de proyectos	tests\Feature\projects\IndexProjectTest.php
10. Ver proyecto	tests\Feature\projects\ShowProjectTest.php
11. Crear proyecto	tests\Feature\projects\CreateProjectTest.php
12. Editar proyecto	tests\Feature\projects\EditProjectTest.php
13. Borrar proyecto	tests\Feature\projects\DeleteProjectTest.php
14. Subir fichero a un proyecto	tests\Feature\projects\CreateProjectTest.php
15. Crear carpeta en un proyecto	tests\Feature\projects\CreateProjectTest.php
16. Ver gestor de casos clínicos	tests\Feature\ClinicCases\IndexClinicCaseTest.php
17. Ver caso clínico	tests\Feature\ClinicCases\ShowClinicCaseTest.php

18. Crear caso clínico	tests\Feature\ClinicCases\CreateClinicCaseTest.php
19. Editar caso clínico	tests\Feature\ClinicCases\EditClinicCaseTest.php
20. Borrar caso clínico	tests\Feature\ClinicCases\DeleteClinicCaseTest.php
21. Ver gestor de antibióticos	tests\Feature\Antibiotics\IndexAntibioticTest.php
22. Ver antibiótico	tests\Feature\Antibiotics\ShowAntibioticTest.php
23. Crear antibiótico	tests\Feature\Antibiotics\CreateAntibioticTest.php
24. Editar antibiótico	tests\Feature\Antibiotics\EditAntibioticTest.php
25. Borrar antibiótico	tests\Feature\Antibiotics\DeleteAntibioticTest.php
26. Enviar caso clínico (email)	tests\Feature\ClinicCases\ShowClinicCaseTest.php
27. Filtrar casos clínicos por:	
a. N.º de caso clínico	tests\Feature\ClinicCases\IndexClinicCaseTest.php
b. Aislamiento fúngico	tests\Feature\ClinicCases\IndexClinicCaseTest.php
c. Aislamiento bacteriano	tests\Feature\ClinicCases\IndexClinicCaseTest.php
d. Localización de la muestra	tests\Feature\ClinicCases\IndexClinicCaseTest.php
e. Terminado	tests\Feature\ClinicCases\IndexClinicCaseTest.php
f. En progreso	tests\Feature\ClinicCases\IndexClinicCaseTest.php
28. Asignar antibióticos a un caso clínico	tests\Feature\ClinicCases\CreateClinicCaseTest.php
29. Ver Panel de administración	tests\Feature\roles\IndexRoleTest.php
30. Listar roles	tests\Feature\roles\IndexRoleTest.php
31. Ver rol	tests\Feature\roles\ShowRoleTest.php
32. Crear rol	tests\Feature\roles\CreateRoleTest.php
33. Editar rol	tests\Feature\roles\EditRoleTest.php
34. Borrar rol	tests\Feature\roles\DeleteRoleTest.php
35. Asignar permisos a un rol	tests\TestCase.php
36. Listar usuarios	tests\Feature\users\IndexUserTest.php

37. Ver usuario	tests\Feature\users\ShowUserTest.php
38. Crear usuario	tests\Feature\users\CreateUserTest.php
39. Editar usuario	tests\Feature\users\EditUserTest.php
40. Borrar usuario	tests\Feature\users\DeleteUserTest.php
41. Asignar rol a un usuario	tests\TestCase.php

Tabla 14. Casos de uso y tests asociados

3.2 Arquitectura de la aplicación

3.2.1 Modelo MVC (Model View Controller)

Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores, o lo que es lo mismo, Models, Views & Controllers, tal y como podemos observar en la *fig.23*.

MVC apareció en los 70 y fue presentado incluso antes de la aparición de la Web. No obstante, en los últimos años ha ganado mucha fuerza y seguidores gracias a la aparición de numerosos frameworks de desarrollo web que utilizan el patrón MVC como modelo para la arquitectura de las aplicaciones web, como es el caso del framework Laravel que hemos utilizado para el prototipo.

Modelos

Es la capa donde se trabaja con los datos, por tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, en el caso de Laravel tiene su propio ORM llamado Eloquent, que nos permite trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

Vistas

Las vistas, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea,

el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite mostrar la salida, en el caso de Laravel al utilizar el motor de plantillas Blade, con el cual podemos entre otras cosas, crear plantillas, extenderlas, integrar o imprimir variables PHP sin perder el buen estilo del código de nuestra aplicación gracias al uso de etiquetas que nos provee.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generará la salida, tal como nuestra aplicación requiera.

Controladores

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad, es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

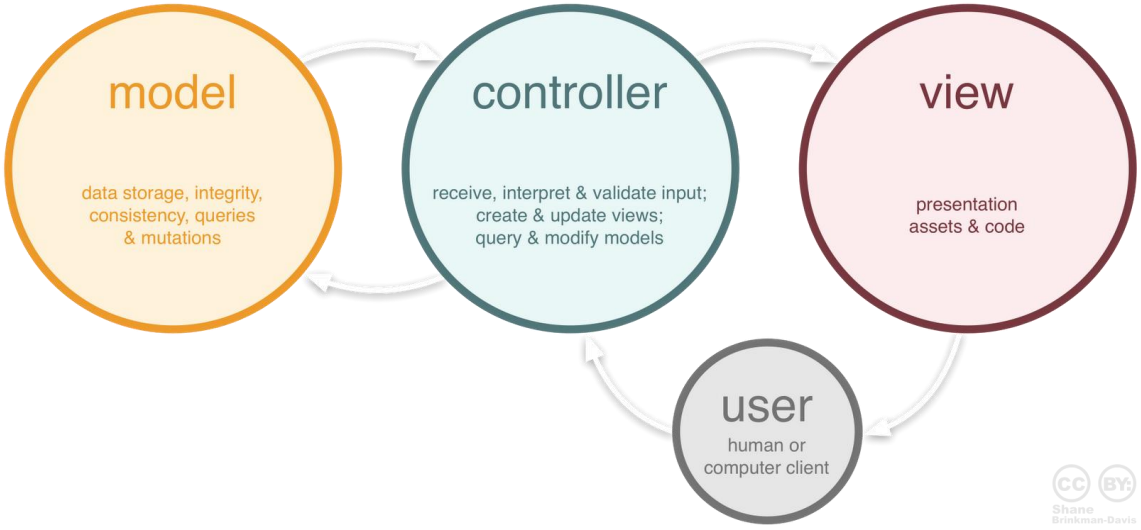


Figura 23.Modelo MVC

3.2.2 MVC en la actualidad

Hoy en día, para desarrollar una aplicación web de forma exitosa, necesitamos mucho más que 3 capas. Por ejemplo, en MVC:

No tenemos una manera de representar las rutas que se encargan de analizar las URLs y asignarlas a un método o función. No tenemos forma de representar un middleware que restringe ciertas áreas de la aplicación basado en el estado del usuario.

Una aplicación moderna requiere mucho más que la representación de datos a través de un modelo. Me imagino que, por eso en Wikipedia, así como en muchas otras partes de la web, han cambiado tantas veces el concepto de MVC, supongo que intentando (y fallando, espectacularmente) que éste refleje las necesidades de una aplicación moderna.

Por ejemplo, en Wikipedia nos dicen que un modelo: “maneja directamente la data, lógica y reglas de la aplicación”, la vista: “es la salida o representación gráfica de información (es decir, ¿Podría ser JSON o un diagrama?), un controlador: “acepta una entrada y la convierte en comandos para el modelo y la vista”.
Estos conceptos son demasiado genéricos, y distan muchísimo de darte las herramientas que necesitas para construir aplicaciones avanzadas.

Por ejemplo, se tiene la creencia muchas veces de que “Modelo” es la “base de datos” pero el 90% de las aplicaciones requieren mucho más lógica que almacenar datos. En la *fig.24* se puede apreciar como el creador de Laravel Taylor Otwell nos lo aclara en un tweet:

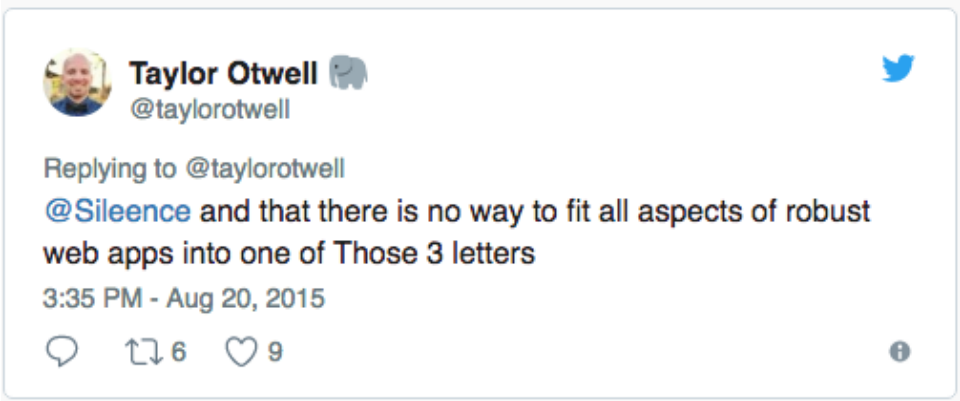


Figura 24. Tweet Taylor Otwell

“No hay manera de encapsular todos los aspectos de aplicaciones web robustas en esas 3 letras.” - Taylor Otwell

Es por ello por lo que frameworks como Symfony o Laravel **no** son MVC.

De hecho, si bien Laravel 4 incluía las 3 famosas carpetas controllers, models, views, en Laravel 5:

Ya no encontramos una carpeta “models”, en vez de eso tienes una carpeta app/ donde puedes estructurar tu aplicación de la forma que tenga más sentido para tu proyecto.

La carpeta Controllers es una pequeña parte de la capa “Http” que se encuentra dentro de app/. Junto a Controllers, tienes Middleware/, tienes el directorio Requests/ donde se albergan los FormRequests y tienes el archivo “routes.php”. La carpeta “views/” se encuentra ahora dentro de resources/ y forma parte de los “recursos” para presentarle los datos al usuario (assets/ lang/).
En la carpeta app/ también encontrarás otras capas como Eventos, Listeners, Excepciones, Jobs, etc.

Aunque la aplicación utilice Modelos, Vistas, y Controladores, no es MVC puesto que se requieren muchas más capas para poder organizar todos los aspectos característicos de tu aplicación, si no quieres terminar escribiendo código espagueti. Por lo que podemos considerar que Laravel tiene una arquitectura de N-capas.

3.2.3 Alternativas

Para un proyecto a mayor escala se podría optar por separar front-end (html, css y js) y back-end (php o java), tal y como se aprecia en la *fig.25*. Alojaríamos en un servidor la parte del front-end que podría ser con cualquier framework de javascript pues es un lenguaje más afín con la parte de cliente, y luego una API Rest en otro servidor, realizada con un framework de php o java, como puede ser Laravel o Spring. Esto seguiría lo que se conoce en ingeniería del software como separation of concerns (SoC) o separación de conceptos en español.

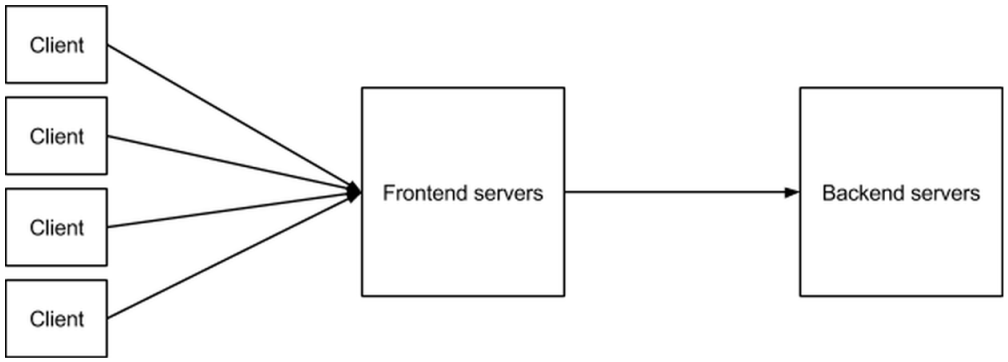


Figura 25. Frontend y backend

A nivel arquitectónico, la separación de conceptos es un componente clave de la construcción de aplicaciones en capas. En una estructura de aplicación de N capas tradicional, las capas pueden incluir acceso a datos, lógica de negocios e interfaz de usuario. Las aplicaciones web, en mayor o menor grado, separan las preocupaciones relacionadas con la estructura, la lógica y el formato mediante el uso de HTML, JavaScript y CSS. En un nivel inferior, el modelo de red utilizado por Internet se divide en una serie de capas, cada una con preocupaciones y responsabilidades específicas, y demuestra cómo la separación de conceptos se puede aplicar efectivamente.

Con esto conseguimos una serie de ventajas a la hora de trabajar y mantener el aplicativo web en cuestión:

-Mayor especialización en el desarrollo de proyectos, lo cual asegura mayor calidad en el producto final.

-División de perfiles en el desarrollo del proyecto, lo cual nos aportaría una mayor productividad.

-Diseño de mejores interfaces para el usuario, al usar javascript que es un lenguaje más amigable e intuitivo para el usuario.

-Una lógica en la aplicación más robusta, pues toda va a estar separada del frontend que lo único que va a hacer es consumir la API Rest.

-Más seguridad, pues no solo tenemos separados los dos conceptos físicamente, sino que validamos tanto desde el cliente y como el servidor.

3.3 Diseño del modelo de datos

3.3.1 Relaciones entre los modelos de la aplicación.

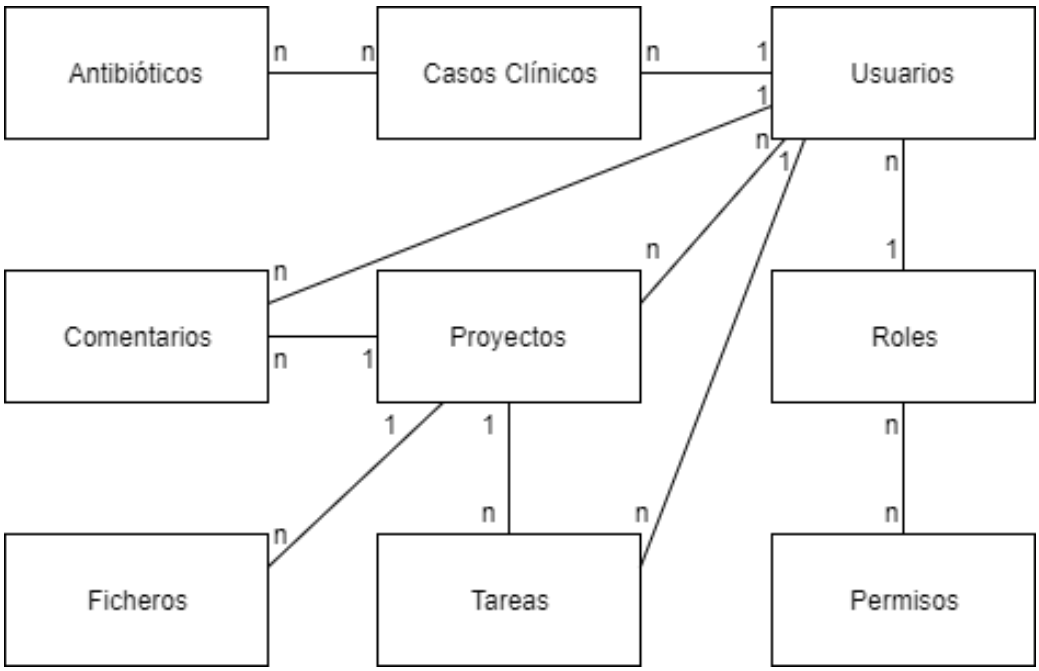


Figura 26. ER base de datos

En la *fig.26* podemos ver las relaciones principales entre los modelos de la base de datos, donde se puede apreciar cómo el usuario está relacionado con los casos clínicos, comentarios, proyectos y tareas de manera que la relación es de uno a varios, sin embargo la relación de los roles a usuarios es a la inversa, todas estas relaciones implican que existirá un campo user_id en las tablas que nombramos con relación de uno a varios y que la tabla usuario deberá contener un campo role_id, de la misma manera que la relación entre proyectos con comentarios y tareas, estas últimas deberán tener un campo project_id que haga la relación 1-n. En los casos de las relaciones de roles-permisos, antibióticos-casos clínicos y usuarios-proyectos son de varios a varios y esto implica la creación de una tercera tabla pivote, que llamaremos permission_role ,clinic_case_antibiotic y project_collaborators respectivamente como se muestra en el siguiente apartado.

3.3.2 Tablas de la base de datos

Podemos ver en las *fig.27* y *fig.28* las tablas que componen la base de datos y los campos de cada una con el tipo de variable.

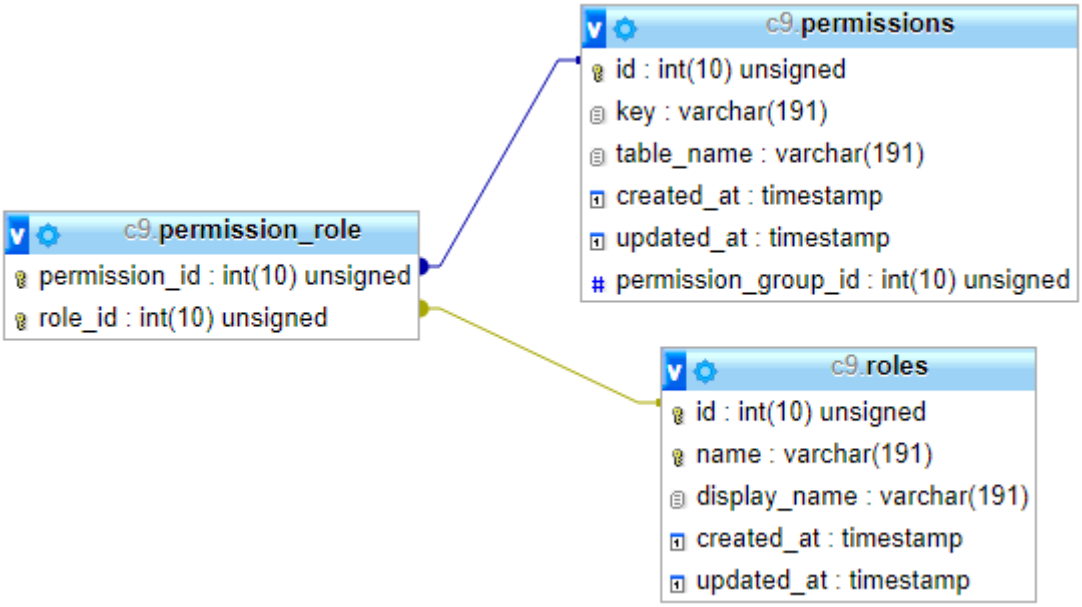


Figura 27. Tablas base de datos 1

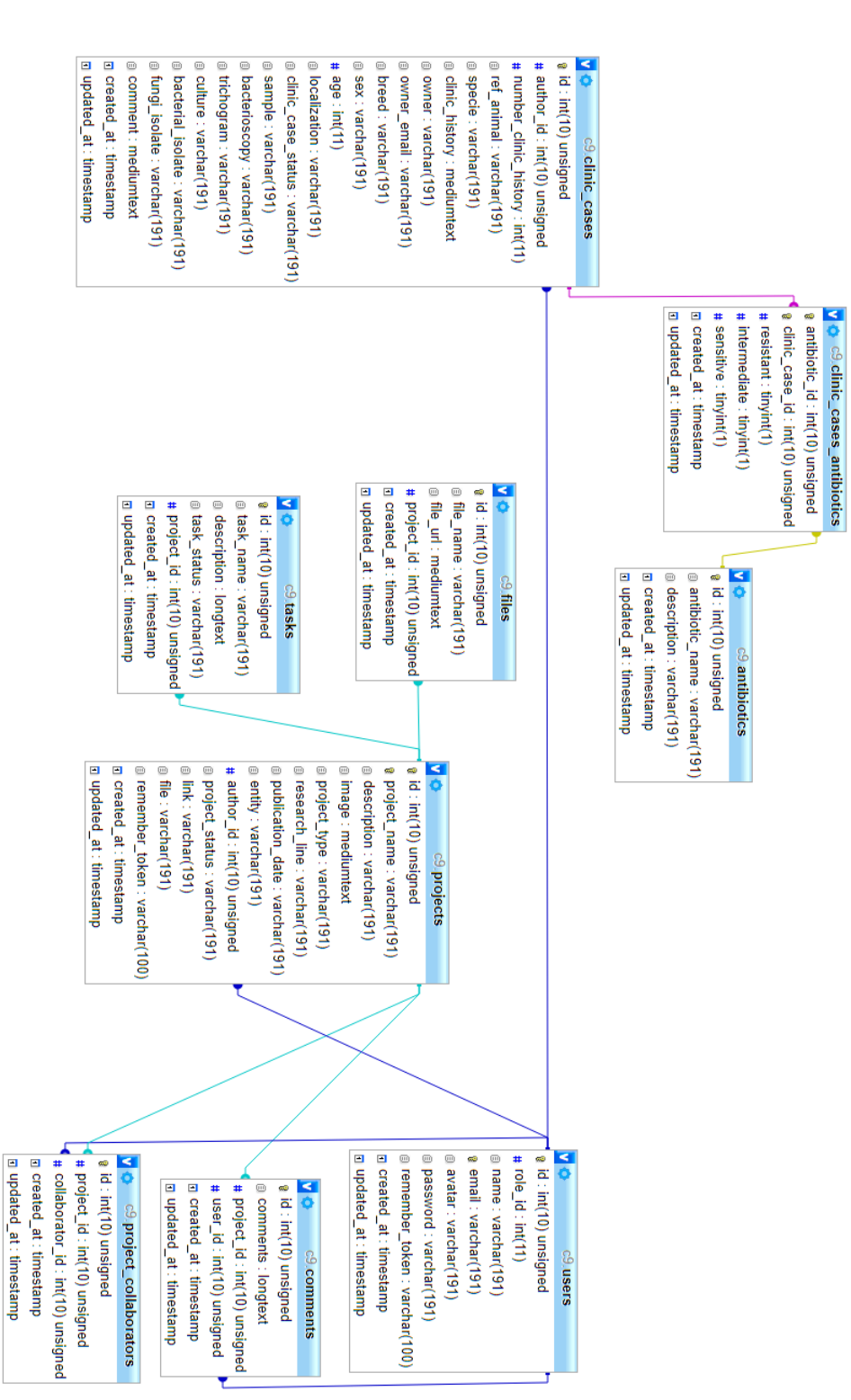


Figura 28. Tablas base de datos 2

Laravel usa el patrón Active Record para la persistencia de los datos. Se trata de una clase que se encarga de implementar todas las operaciones de consulta y modificación de una tabla de la base de datos, que a diferencia de otros patrones como DAO (Data Access Object), va a aportar menor flexibilidad a cambio de un mayor aislamiento para trabajar con el sistema de gestión de la base de datos, en este caso mysql.

El patrón Active Record permite trabajar las tablas como si fueran clases y las filas como objetos. Para ello, Laravel cuenta con un potente ORM (Object-Relational mapping) llamado Eloquent y además tiene un constructor de consultas SQL (query builder) llamado Fluent.

También permite utilizar SQL puro para consultas más complicadas. En primer lugar, Fluent es un constructor de consultas SQL, basado en PDO, encargado de generar cualquier consulta a la base de datos. Las consultas generadas vienen por defecto con los niveles de seguridad para evitar inyecciones SQL de usuarios malintencionados. Las bases de datos relacionales son incompatibles con la forma de trabajar en la programación orientada a objetos, en la que los objetos se relacionan con otros a través de propiedades y métodos. Eloquent permite mapear datos de la base de datos y convertirlos a objetos o, por el contrario, tomar un objeto y almacenarlo como un registro de la base de datos. A continuación se explica la forma de usar las migraciones y seeders, para cargar información aleatoria en las tablas y poder probar la aplicación.

3.3.3 Migraciones y seeders

Iniciamos con, `php artisan make:migration create_clinic_cases_table` en este caso para crear el fichero de migración para la tabla de casos clínicos.

Una vez tenemos el fichero, definimos en él los campos y relaciones, como se muestra en la *fig.29.*

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateClinicCasesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('clinic_cases', function($table) {
            $table->increments('id');
            $table->integer('author_id')->unsigned();
            $table->foreign('author_id')->references('id')->on('users')->onDelete('cascade');
            $table->integer('number_clinic_history');
            $table->string('ref_animal');
            $table->string('specie');
            $table->mediumtext('clinic_history');
            $table->string('owner');
            $table->string('owner_email')->nullable();
            $table->string('breed');
            $table->string('sex');
            $table->integer('age');
            $table->string('localization');
            $table->string('clinic_case_status');
            $table->string('sample')->nullable();
            $table->string('bacterioscopy')->nullable();
            $table->string('trichogram')->nullable();
            $table->string('culture')->nullable();
            $table->string('bacterial_isolate')->nullable();
            $table->string('fungi_isolate')->nullable();
            $table->mediumtext('comment')->nullable();
            $table->timestamps();
        });
    }

```

Figura 29. Migración

Acto seguido vamos a la consola nuevamente y creamos un fichero seeder para los casos clínicos, en la *fig.30* podemos ver la función creada,

`php artisan make:seeder ClinicCasesTableSeeder`

```php
<?php

use App\ClinicCase;
use Illuminate\Database\Seeder;

class ClinicCasesTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        for ($i = 1; $i <= 200; $i++) {
            factory(ClinicCase::class)->create();
        }
    }
}

```

Figura 30. Seeder

De esta manera generamos 200 registros en nuestra nueva tabla, sólo falta definir el factory del caso clínico, podemos ver la defición en la *fig.31*.

```php
$factory->define(App\ClinicCase::class, function (Faker\Generator $faker) {
       return ['number_clinic_history' => $faker->randomNumber(8),
        'author_id' => 1,
        'ref_animal' => $faker->randomNumber(8),
        'specie' => $faker->word,
        'clinic_history' => $faker->paragraph(2),
        'owner' => $faker->name,
        'owner_email' => $faker->email,
        'breed' => $faker->name,
        'sex' => $faker->randomElement($array = array ('male','female')),
        'age' => $faker->randomNumber(1),
        'localization' => $faker->randomElement($array = array (
            'skin',
            'eye',
            'mouth',
            'nose',
            'nail',
            'hair',
            'ear',
            'blood',
            'biopsy',
            'bodyfluids',
            'feces',
            'urine',
            'others')
        ),
        'clinic_case_status' => $faker->randomElement($array = array ('inprogress','finished')),
        'sample' => $faker->word,
        'bacterioscopy' => $faker->word,
        'trichogram' => $faker->word,
        'culture' => $faker->word,
        'bacterial_isolate' => $faker->word,
        'fungi_isolate' => $faker->word,
        'comment' => $faker->paragraph(2),
    ];
});
```

Figura 31. Factory

Ahora solo queda volver a la consola y arrancar los comandos para migrar las tablas y hacer el seed de la base de datos:

```
php artisan migrate
```

```
php artisan db:seed
```

3.4 Despliegue a Heroku

Heroku en un servicio tipo PaaS (plataform as a service) basado en la nube que incluye lenguajes de programación muy variados como php o ruby, incluyendo otras herramientas para el desarrollo. Con todo esto tienes una plataforma con el material necesario para llevar a cabo todas las fases de desarrollo directamente en la web de Heroku, desde la construcción del sitio web hasta el despliegue total de la aplicación.

En Heroku puedes desplegar varias aplicaciones sin ningún costo, siempre y cuando uses su opción de un solo Dyno. Los Dynos son contenedores virtualizados Unix, que proporcionan el entorno necesario para ejecutar una aplicación. La opción gratuita sirve para aplicaciones de mediana capacidad, por lo que es una opción viable para el proyecto desarrollado.

Como prerrequisitos necesitaremos una cuenta de Heroku la cual es totalmente gratuita y Heroku Toolbelt.

Por defecto apache2 apunta a la carpeta web y no a /public. Es por esto que debemos crear un archivo Procfile personalizado para que apunte a la carpeta /public. Este paso es indispensable para que la aplicación corra bien en Heroku.

Una vez tengamos instalado el Heroku Toolbelt, en el directorio principal ejecutamos:

```
heroku create
```

Introducimos los datos de autenticación y el nombre de nuestra aplicación y ya solo nos queda el último paso para hacer pública nuestra aplicación.

```
git push heroku master
```

Una vez ejecutado este comando ya estará disponible en la nube de heroku.

3.5 Tecnologías

Git (https://git-scm.com)

Git es un sistema de control de versiones diseñado por Linus Torvald. Se trata del gestor de versiones moderno más empleado y cuenta con una gran comunidad de desarrolladores a su alrededor lo que favorece la integración de Git con múltiples herramientas. Es distribuido bajo una licencia GNU.

GitHub (https://github.com)

Es un portal que ofrece servicios de hosting para proyectos usando Git. Ofrece cuentas gratuitamente a proyectos de tipo open source y es ampliamente utilizado por la comunidad.

CSS, Cascading Style Sheets (https://www.w3schools.com/css)

Es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C (World Wide Web Consortium) es el encargado de postular las especificaciones de las hojas de estilo que servirán de estándar para los navegadores. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura del documento de su presentación.

StarUML (http://staruml.io)

StarUML es una herramienta UML open source que permite crear todo tipo de diagramas.

GIMP (http://www.gimp.org.es)

GNU Image Manipulation Program es programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito que forma parte del proyecto GNU.

HTML (https://www.w3schools.com/html)

Siglas de HyperText Markup Language, es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para completar el texto con objetos tales como imágenes. HTML se escribe en forma de etiquetas rodeadas por corchetes angulares (<, >).

Javascript (https://www.javascript.com)

JavaScript es un lenguaje de programación interpretado. Se define como orientación a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente del lado del cliente implementado como parte de un navegador web, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Todos los navegadores modernos interpretan el código de JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje de JavaScript de una implementación del DOM.

JQuery (https://jquery.com)

Es una biblioteca o framework de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, general animaciones y agregar interacción con la técnica AJAX a páginas web. JQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menor tiempo y espacio.

Bootstrap (https://getbootstrap.com)

Bootstrap es un framework CSS desarrollado inicialmente (en el año 2011) por Twitter que permite dar forma a un sitio web mediante librerías CSS que incluyen tipografías, botones, cuadros, menús y otros elementos que pueden ser utilizados en cualquier sitio web.

Laragon (https://laragon.org)

Laragon es una suite de desarrollo para PHP que funciona sobre Windows diseñado especialmente para trabajar con Laravel. Similar a otras herramientas como Xampp o Wampp, Laragon nos permite crear un entorno de desarrollo con todas las características que necesitamos.

PHPUnit (https://phpunit.de/documentation.html)

PHPUnit es un framework open source para el desarrollo, orientado a pruebas o TDD para cualquier código PHP. Es decir, es un framework que nos ayuda a probar nuestro código.

PHP (http://php.net)

PHP (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML.

Faker (https://github.com/fzaninotto/Faker)

Faker es una biblioteca PHP que genera datos falsos. Ya sea para arrancar la base de datos, crear documentos XML atractivos, completar su persistencia para ponerla a prueba o anonimizar datos tomados de un servicio de producción.

Voyager (https://voyager.readme.io/docs)

Voyager es una librería de php que nos proporciona un panel de administrador para Laravel. Y proporciona gestión de roles y permisos.

Composer (https://laravel.com/docs/5.5/installation)

Composer es una de las herramientas fundamentales en php a la hora de instalar dependencias de proyectos en Linux para montar un ambiente de desarrollo fácil y de manera correcta.

MySQL (https://www.mysql.com)

Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario. MySQLAB desarrolla MySQL como sofware libre en un esquema de licenciamiento dual.

Atom (https://atom.io)

Atom es un editor de texto, desarrollado por Github que buscaba desarrollar una herramienta que pudiese ser utilizada, por alguien que estuviese dando sus primeros pasos en el mundo de la programación. Para hacer simples las cosas, Github se ha apoyado en el mundo de la web y Atom se ha construido usando los mismos componentes que se usarían para desarrollar una web: HTML y CSS.

Moqups (https://moqups.com)

Moqups es una ingeniosa aplicación HTML5 utilizada para crear wireframes, maquetas o conceptos de interfaz de usuario, o prototipos. Simple y bastante intuitiva.

Heroku (https://heroku.com)

Heroku es una plataforma como servicio de computación en la Nube que soporta distintos lenguajes de programación. Heroku es propiedad de Salesforce.com,

es una de las primeras plataformas de computación en la nube, que fue desarrollada desde junio de 2007, con el objetivo de soportar solamente el lenguaje de programación Ruby, pero posteriormente se ha extendido el soporte a Java, Node.js, Scala, Clojure y Python y (no documentado) PHP. La base del sistema operativo es Debian o, en la nueva plataforma, el sistema basado en Debian Ubuntu.

Cloud9 (https://c9.io)

Cloud 9 o c9 es un Integrated Development Environment (IDE) para desarrollo que tiene como plataforma la nube.

Laravel (https://laravel.com)

Laravel es un nuevo y poderoso Framework PHP desarrollado por Taylor Otwell, que promete llevar al lenguaje PHP a un nuevo nivel. Desarrollar aplicaciones usando Laravel es muy sencillo, fundamentalmente debido a su expresiva sintaxis, sus generadores de código, y su ORM incluido de paquete llamado Eloquent ORM.

Laravel, propone una forma de desarrollar aplicaciones web de un modo mucho más ágil. En Laravel opcionalmente podemos usar el patrón de diseño MVC (Modelo-Vista-Controlador) tradicional.

Laravel Collective (https://laravelcollective.com)

Laravel Collective es una librería que mantiene los componentes que se han eliminado del núcleo del framework de Laravel.

Laravel File-Manager (https://github.com/UniSharp/laravel-filemanager)

Es una librería que ayuda a realizar la subida múltiple de ficheros y organizarlos de manera estructurada en carpetas.

Draw.io (https://www.draw.io)

Es un software de diagrama en línea gratuito para hacer diagramas de flujo, diagramas de proceso, organigramas, UML, ER y diagramas de red.

3.6 Github: Vetmyc

El repositorio en github es accesible desde la url:

https://github.com/Warmeta/vetMyc

4. Conclusiones

Este proyecto ha cumplido lo propuesto inicialmente, desde el punto de vista general del desarrollo de un producto ha sido interesante estar en contacto con la persona encargada del laboratorio para cubrir las necesidades demandadas y el desarrollo en TDD ha supuesto un reto, debido a la gran cantidad de funcionalidades de las que dispone el producto.

Este sistema de información tiene un gran valor para el laboratorio, ahora se pueden realizar algunas de las tareas que allí se llevan a cabo desde cualquier dispositivo (siempre que se disponga de conexión a internet), recabar información de los casos clínicos de una forma más eficiente gracias a los filtros y tener un repositorio de ficheros en cada proyecto.

El desarrollo basado las pruebas ha sido una parte importante dentro del desarrollo, a pesar de los elevados requisitos iniciales de aplicar esta metodología, nos da un gran valor añadido al desarrollo pues cuando se vayan a añadir nuevas funcionalidades, pues va a ser más fácil detectar los posibles errores que vayan apareciendo y en consecuencia la calidad del producto es mayor. Además, nos proporciona una herramienta para saber que todo funciona según lo esperado y durante el desarrollo también ha sido de gran ayuda para seguir un guion sobre que implementar y de esta manera tener un ritmo constante.

También ha supuesto un desafío realizar un producto que tiene un sistema tan flexible de roles de usuario, pues que se puedan definir nuevos roles con los permisos que se desee le da al aplicativo un abanico de posibilidades muy grandes.

El diseño de la interfaz de usuario fue un rompecabezas constante y ha sufrido muchos cambios a lo largo del desarrollo. En este aspecto ha sido de gran ayuda haber usado un framework como Laravel que separa los diferentes aspectos de la aplicación web, en este caso las vistas y los controladores.

A nivel personal he de decir que realizar un producto real para una necesidad real ha sido muy gratificante, aunque siempre la propuesta inicial requiere de cambios constantes para obtener un producto lo más acabado posible, las sensaciones finales han sido buenas y eso se ve reflejado en el producto.

5. Trabajo futuro

El desarrollo de este sistema aún puede mejorarse, aunque vetMyc se trata de un producto completo, es mejorable. Se podría mejorar la gestión de proyectos con algunas funcionalidades como un log de comentarios o un Kanban para la realización de tareas asociadas a un proyecto, son funcionalidades que se pensaron desde un inicio pero que no terminaron de llevarse a cabo por nuevas funcionalidades que aportaban más al producto que las mencionadas anteriormente.

También sería conveniente poder crear de manera dinámica las páginas que ahora mismo son estáticas de la aplicación y que un usuario que tuviera los permisos necesarios pudiera modificar el contenido que se muestra en ellas.

Realizar un despliegue en otra plataforma que ofrezca mejor rendimiento que Heroku, el alojamiento web de la ULPGC en este caso es la opción que probablemente se llevará a cabo a la espera de obtener un dominio y los permisos para poder llevar a cabo el despliegue.

Lo más óptimo para ofrecer un servicio lo más rápido y amigable para el usuario, sería usar javascript para el frontend de la aplicación, ahora mismo se utiliza el motor de plantillas que viene por defecto en Laravel que se llama Blade y al ser html y css puro, cualquier modificación de la vista requiere de que el servidor vuelva a renderizar la vista en cuestión y eso hoy en día no es una buena práctica dentro del frontend, nuestro aplicativo ofrece algunas funciones adicionales hechas con javascript para evitar este comportamiento tan engorroso pero sería conveniente rehacer todo el front de la web con algún framework como Vue.js o Angular para las vistas.

6. Documentación

[1] Bootstrap documentación oficial. https://getbootstrap.com/docs/3.3

[2] Heroku documentación oficial. https://devcenter.heroku.com/categories/reference

[3] Laragon documentación oficial. https://laragon.org/docs/index.html

[4] Laravel Collective documentación. https://laravelcollective.com/docs/master/html

[5] Laravel documentación oficial. https://laravel.com/docs/5.4

[6] Laravel File-Manager. https://github.com/UniSharp/laravel-filemanager

[7] PHP documentación oficial. http://php.net/docs.php

[8] phpUnit documentación oficial. https://phpunit.de/documentation.html

[9] Styde comunidad de Laravel. https://styde.net

[10] Voyager documentación para laravel. https://voyager.readme.io/docs

[11]W3School documentación para Javascript, Html y css. https://www.w3schools.com

[12] Wikipedia. https://es.wikipedia.org

(Última fecha de acceso, lunes 18 de diciembre de 2017.)

Anexos

Anexo I. Manual de usuario

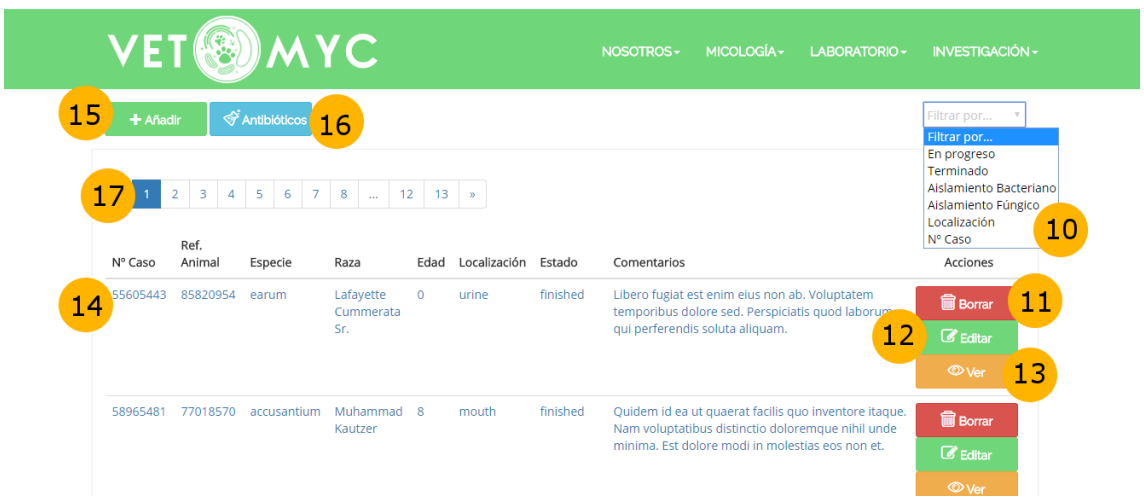
Anexo II. Tests Unitarios

Anexo I. Manual de usuario

Nº	Funcionalidad	Descripción
1	Ver nosotros	Página principal de la aplicación.
2	Ver Micología	Páginas con información sobre la micología veterinaria.
3	Ver Laboratorio	Página con información relacionada al servicio de laboratorio y los diferentes impresos disponibles.
4	Ver Investigación	Página relacionada con los integrantes del grupo de trabajo y los proyectos realizados.
5	Login	Login a la plataforma.
6	Registro	Registro a la plataforma.

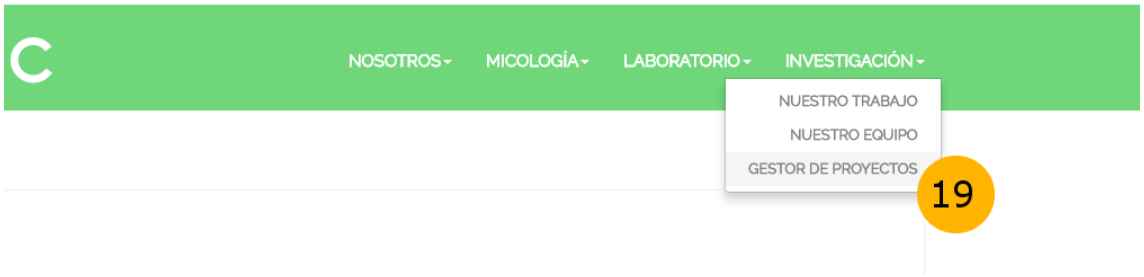
Nº	Funcionalidad	Descripción
7	Cerrar sesión	Cerrar la sesión de la plataforma.
8	Dashboard	Acceso al panel de administración. (sólo visible y accesible para usuarios con permisos)

Nº	Funcionalidad	Descripción
9	Gestor de casos clínicos	Acceso al gestor de casos clínicos. (sólo visible y accesible para usuarios con permisos)

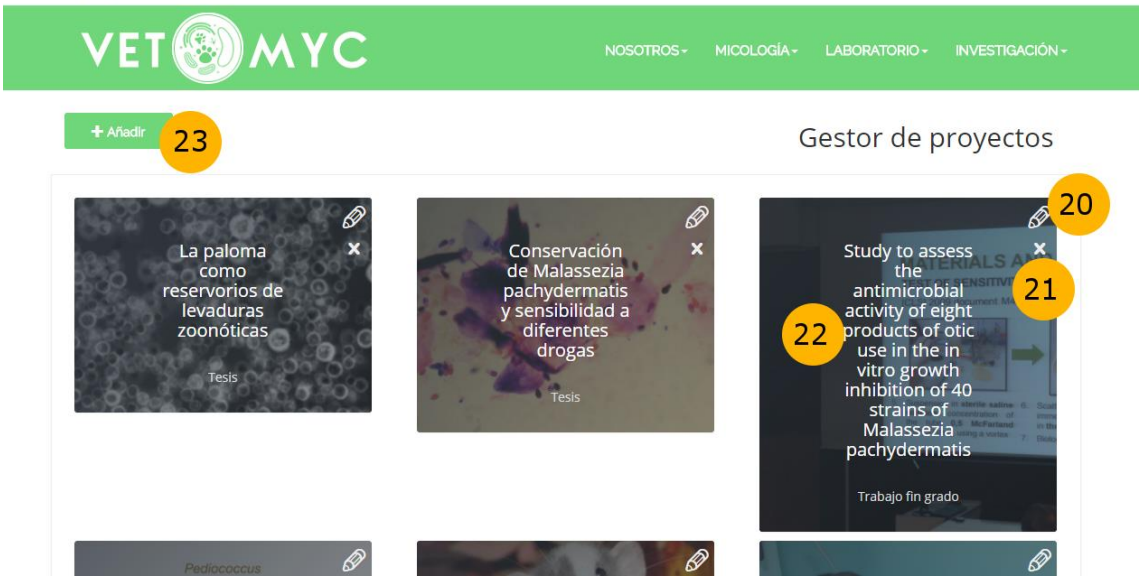


Nº	Funcionalidad	Descripción
10	Filtro de casos clínicos	Filtrado del índice de casos clínicos por, estado, tipo de aislamiento, localización de la muestra y el nº de caso clínico.
11	Borrar caso clínico	Borrado de la base de datos del caso clínico.
12	Editar caso clínico	Editado del caso clínico.
13	Ver caso clínico	Visualización del caso clínico.
14	Ver caso clínico (2)	Visualización del caso clínico clicando directamente encima de la fila.
14.1	Enviar Caso clínico vía email	Dentro de la vista del caso clínico se puede enviar el caso clínico al email correspondiente, clicando el botón enviar email.
15	Añadir caso clínico	Va al formulario para la creación de un nuevo caso clínico.
16	Gestor de antibióticos	Pasa la vista del índice de casos clínicos, a la vista de antibióticos.
17	Paginación	Paginación del índice de casos clínicos.

Nº	Funcionalidad	Descripción
18	Gestor de casos clínicos	Pasa la vista del índice de antibióticos, al índice de casos clínicos.

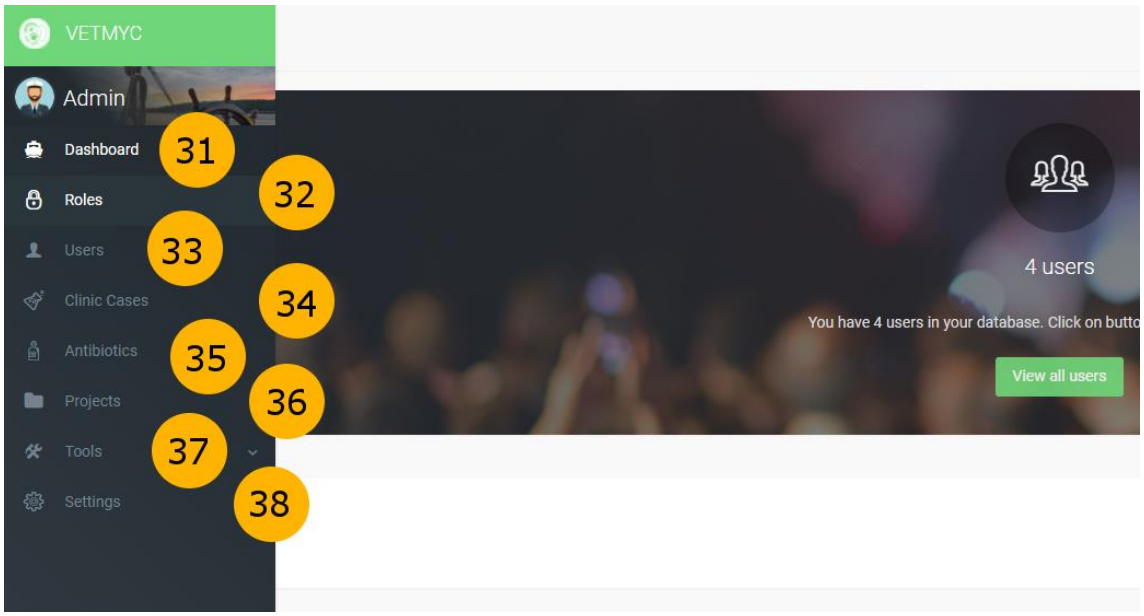


Nº	Funcionalidad	Descripción
19	Gestor de proyectos	Acceso al gestor de proyectos. (sólo visible y accesible a usuarios con permisos)



Nº	Funcionalidad	Descripción
20	Editar proyecto	Abre el formulario de edición del proyecto.
21	Eliminar proyecto	Elimina el proyecto de la base de datos.
22	Ver proyecto	Visualización del proyecto.
23	Añadir proyecto	Abre el formulario de creación de un nuevo proyecto.

Nº	Funcionalidad	Descripción
24	Abrir acciones del File-Manager	Abre las diferentes acciones disponibles dentro del File-Manager.
25	Crear carpeta	Crea una carpeta dentro del repositorio del proyecto.
26	Subir fichero	Subida de un fichero al repositorio previa selección.
27	índice de ficheros	Índice de fichero del File-Manager
28	Opciones del File-Manager	Diferentes opciones que ofrece el File-Manager, como ordenar los ficheros.
29	Link	Enlace externo del proyecto.
30	Investigadores del proyecto	Enlace externo para los diferentes autores del proyecto.



Nº	Funcionalidad	Descripción
31	Dashboard	Acceso a la vista principal del administrador.
32	Roles	Acceso al gestor de roles.
33	Usuarios	Acceso al gestor de usuarios.
34	Casos clínicos	Acceso al gestor de casos clínicos.
35	Antibióticos	Acceso al gestor de antibióticos.
36	Proyectos	Acceso al gestor de proyectos.
37	Herramientas	Acceso a las herramientas del panel de administración. Configuración del menú, base de datos…
38	Ajustes del panel	Ajustes del panel de administración para cambiar el logo de carga y otros ajustes.

Anexo II. Tests Unitarios

Tests Antibióticos

```
<?php

namespace Tests\Feature\Antibiotics;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class CreateAntibioticTest extends TestCase
{
  public function testCreateAntibioticFailWithoutLoginUser()
  {
    $antibiotic = factory('App\Antibiotic')->make()->toArray();
    $response = $this->call('POST', '/lab/antibiotic/create', $antibiotic, [], [], []);
    $response->assertStatus(302)->assertRedirect('/');
  }

  public function testCreateAntibioticFailWithoutRequiredParameterAsAdmin()
  {
    $antibiotic = factory('App\Antibiotic')->make(['antibiotic_name' => ''])->toArray();

    $user = $this->createUserWithAdminPermissions('antibiotics');

    $response = $this
    ->actingAs($user)
    ->post('/lab/antibiotic/create', $antibiotic);
```

```
    $response->assertRedirect('/lab/antibiotic/create')
    ->assertSessionHasErrors(['antibiotic_name']);
    $this->assertDatabaseMissing('antibiotics', $antibiotic);
  }

  public function testCreateAntibioticSuccessAsAdmin()
  {
    $antibiotic = factory('App\Antibiotic')->make()->toArray();

    $user = $this-
>createUserWithAdminPermissions('antibiotics');

    $response = $this
    ->actingAs($user)
    ->post('/lab/antibiotic/create', $antibiotic);

    $response->assertRedirect('/lab/antibiotic')
    ->assertStatus(302);
    $this->assertDatabaseHas('antibiotics', $antibiotic);
  }

  public function testCreateAntibioticFailLogedAsUser()
  {
    $antibiotic = factory('App\Antibiotic')->make()->toArray();

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->post('/lab/antibiotic/create', $antibiotic);
```

```
    $response->assertRedirect('/')
    ->assertStatus(302);
    $this->assertDatabaseMissing('antibiotics', $antibiotic);
  }
}
<?php

namespace Tests\Feature\Antibiotics;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class DeleteAntibioticTest extends TestCase
{
  public function testDeleteAntibioticFailWithoutLoginUser()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);
    $response = $this->delete('/lab/antibiotic/delete/' .
$antibiotic['id']);
    $response->assertStatus(302)->assertRedirect('/');
    $this->assertDatabaseHas('antibiotics', $antibiotic);
  }

  public function testDeleteAntibioticAsAdminSuccess()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);

    $user = $this-
>createUserWithAdminPermissions('antibiotics');
```

```
    $response = $this
    ->actingAs($user)
    ->delete('/lab/antibiotic/delete/' . $antibiotic['id']);

    $response->assertStatus(200);
    $this->assertDatabaseMissing('antibiotics', $antibiotic);
  }

  public function testDeleteAntibioticFailLogedAsUser()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->delete('/lab/antibiotic/delete/' . $antibiotic['id']);

    $response->assertRedirect('/')
    ->assertStatus(302);
    $this->assertDatabaseHas('antibiotics', $antibiotic);
  }
}
<?php

namespace Tests\Feature\Antibiotics;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;
```

```
class IndexAntibioticTest extends TestCase
{
  public function testIndexAntibioticFailWithoutLoginUser()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);
    $response = $this->get('/lab/antibiotic');
    $response->assertStatus(302)->assertRedirect('/');
    $this->assertDatabaseHas('antibiotics', $antibiotic);
  }

  public function testIndexAntibioticAsAdminSuccess()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);

    $user = $this-
>createUserWithAdminPermissions('antibiotics');

    $response = $this
    ->actingAs($user)
    ->get('/lab/antibiotic');
    $response->assertStatus(200)
    ->assertSee((string)$antibiotic['antibiotic_name']);
  }

  public function testIndexAntibioticFailLogedAsUser()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
```

```
    $this->assertDatabaseHas('antibiotics', $antibiotic);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->get('/lab/antibiotic');
    $response->assertStatus(302)
    ->assertRedirect('/');
  }
}
<?php

namespace Tests\Feature\Antibiotics;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class ShowAntibioticTest extends TestCase
{
  public function testShowAntibioticFailWithoutLoginUser()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);

    $response = $this->get('/lab/antibiotic/' .
$antibiotic['id']);
    $response->assertStatus(302)->assertRedirect('/');
    $this->assertDatabaseHas('antibiotics', $antibiotic);
  }
```

```
  public function testShowAntibioticAsAdminSuccess()
  {
    $antibiotic = factory('App\Antibiotic')->create()->toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);

    $user = $this->createUserWithAdminPermissions('antibiotics');

    $response = $this
    ->actingAs($user)
    ->get('/lab/antibiotic/' . $antibiotic['id']);
    $response->assertStatus(200)->assertSee((string)$antibiotic['antibiotic_name']);
  }

  public function testShowAntibioticFailLogedAsUser()
  {
    $antibiotic = factory('App\Antibiotic')->create()->toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->get('/lab/antibiotic/' . $antibiotic['id']);
    $response->assertStatus(302)
    ->assertRedirect('/');
  }
}
<?php
```

```
namespace Tests\Feature\Antibiotics;


use Tests\TestCase;
use Illuminate\Support\Facades\Session;


class UpdateAntibioticTest extends TestCase
{
  public function testUpdateAntibioticFailWithoutLoginUser()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
    $antibiotic2 = factory('App\Antibiotic')->make()->toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);
    $response = $this->put('/lab/antibiotic/' .
$antibiotic['id'] . '/edit', $antibiotic2);
    $response->assertStatus(302)->assertRedirect('/');
    $this->assertDatabaseHas('antibiotics', $antibiotic);
  }


  public function testUpdateAntibioticAsAdminSuccess()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
    $antibiotic2 = factory('App\Antibiotic')->make()->toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);


    $user = $this-
>createUserWithAdminPermissions('antibiotics');


    $response = $this
    ->actingAs($user)
    ->put('/lab/antibiotic/' . $antibiotic['id'] . '/edit',
$antibiotic2);
```

```
    $response->assertStatus(302)-
>assertRedirect('/lab/antibiotic');
    $this->assertDatabaseHas('antibiotics', $antibiotic2);
  }


  public function testUpdateAntibioticFailLogedAsUser()
  {
    $antibiotic = factory('App\Antibiotic')->create()-
>toArray();
    $antibiotic2 = factory('App\Antibiotic')->make()->toArray();
    $this->assertDatabaseHas('antibiotics', $antibiotic);


    $user = $this->createUserWithUserPermissions();


    $response = $this
    ->actingAs($user)
    ->put('/lab/antibiotic/' . $antibiotic['id'] . '/edit',
$antibiotic2);
    $response->assertStatus(302)
    ->assertRedirect('/');
    $this->assertDatabaseHas('antibiotics', $antibiotic);
  }
}
<?php
```

Tests Autenticación de usuario

```
namespace Tests\Feature\Auth;


use Tests\TestCase;


class LoginTest extends TestCase
{
  public function testLoginError()
  {
    $user = factory('App\User')->create();
    $credentials = ['email' => $user->email, 'password' =>
'invalid'];
    $response = $this->call('POST', '/login', $credentials, [],
[], []);
    $response->assertSessionHasErrors();
    $response->assertRedirect('/')
    ->assertStatus(302);;
  }


  public function testLoginSuccess()
  {
    $user = factory('App\User')->create();
    $credentials = ['email' => $user->email, 'password' =>
'secret'];
    $response = $this->call('POST', '/login', $credentials, [],
[], []);
    $response->assertRedirect('/')
    ->assertStatus(302);;
  }
}
<?php
```

```
namespace Tests\Feature\Auth;


use Tests\TestCase;


class RegistrationTest extends TestCase

{

  public function testRegistrationErrorPassword()

  {

    $credentials = ['name' => 'test', 'email' =>
'test@test.com', 'password' => 'invalid',
'password_confirmation' => 'invalid1'];

    $response = $this->call('POST', '/register', $credentials,
[], [], []);

    $response->assertSessionHasErrors();

    $response->assertRedirect('/')

    ->assertStatus(302);

  }


  public function testRegistrationErrorEmail()

  {

    $credentials = ['name' => 'test', 'email' => 'testtest.com',
'password' => 'invalid', 'password_confirmation' => 'invalid'];

    $response = $this->call('POST', '/register', $credentials,
[], [], []);

    $response->assertSessionHasErrors();

    $response->assertRedirect('/')

    ->assertStatus(302);

  }


  public function testRegistrationSuccess()

  {

    $credentials = ['name' => 'test', 'email' =>
'test@test.com', 'password' => 'invalid',
'password_confirmation' => 'invalid'];
```

```
    $response = $this->call('POST', '/register', $credentials,
[], [], []);
    $response->assertRedirect('/')
    ->assertStatus(302);;
  }
}
<?php
```

Tests Casos clínicos

```
namespace Tests\Feature\ClinicCases;


use Tests\TestCase;
use Illuminate\Support\Facades\Session;


class CreateClinicCaseTest extends TestCase
{
  public function testCreateClinicCaseFailWithoutLoginUser()
  {
    $clinicCase = factory('App\ClinicCase')->make()->toArray();
    $response = $this->call('POST', '/lab/clinic-case/create', 
$clinicCase, [], [], []);
    $response->assertStatus(302)->assertRedirect('/');
  }


  public function 
testCreateClinicCaseFailWithoutRequiredParameterAsAdmin()
  {
    $clinicCase = factory('App\ClinicCase')-
>make(['number_clinic_history' => ''])->toArray();


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->post('/lab/clinic-case/create', $clinicCase);


    $response->assertRedirect('/lab/clinic-case/create')
    ->assertSessionHasErrors(['number_clinic_history']);
    $this->assertDatabaseMissing('clinic_cases', $clinicCase);
```

```
  }

  public function testCreateClinicCaseSuccessAsAdmin()
  {
    $clinicCase = factory('App\ClinicCase')->make()->toArray();

    $user = $this-
>createUserWithAdminPermissions('clinic_cases');

    $response = $this
    ->actingAs($user)
    ->post('/lab/clinic-case/create', $clinicCase);

    $response->assertRedirect('/lab/clinic-case')
    ->assertStatus(302);
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
  }

  public function testCreateClinicCaseFailLogedAsUser()
  {
    $clinicCase = factory('App\ClinicCase')->make()->toArray();

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->post('/lab/clinic-case/create', $clinicCase);

    $response->assertRedirect('/')
    ->assertStatus(302);
    $this->assertDatabaseMissing('clinic_cases', $clinicCase);
  }
```

```
}
<?php

namespace Tests\Feature\ClinicCases;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class DeleteClinicCaseTest extends TestCase
{
  public function testDeleteClinicCaseFailWithoutLoginUser()
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
    $response = $this->delete('/lab/clinic-case/delete/' .
$clinicCase['id']);
    $response->assertStatus(302)->assertRedirect('/');
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
  }

  public function testDeleteClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);

    $user = $this-
>createUserWithAdminPermissions('clinic_cases');

    $response = $this
    ->actingAs($user)
    ->delete('/lab/clinic-case/delete/' . $clinicCase['id']);
```

```
    $response->assertStatus(200);
    $this->assertDatabaseMissing('clinic_cases', $clinicCase);
  }

  public function testDeleteClinicCaseFailLogedAsUser()
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->delete('/lab/clinic-case/delete/' . $clinicCase['id']);

    $response->assertRedirect('/')
    ->assertStatus(302);
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
  }
}
<?php

namespace Tests\Feature\ClinicCases;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class IndexClinicCaseTest extends TestCase
{
  public function testIndexClinicCaseFailWithoutLoginUser()
```

```
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
    $response = $this->get('/lab/clinic-case');
    $response->assertStatus(302)->assertRedirect('/');
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
  }


  public function testIndexClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-case');
    $response->assertStatus(200)
    ->assertSee((string)$clinicCase['number_clinic_history']);
  }


  public function testIndexClinicCaseFailLogedAsUser()
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this->createUserWithUserPermissions();
```

```
    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-case');
    $response->assertStatus(302)
    ->assertRedirect('/');
  }


  public function
testIndexFilterStateInProgressClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['clinic_case_status' => 'inprogress'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=&filter=inprogress');
    $response->assertStatus(200)
    ->assertSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterStateInProgressClinicCaseAsAdminFailWithNoClinicC
aseAssociated()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['clinic_case_status' => 'finished'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
```

```
    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=&filter=inprogress');
    $response->assertStatus(200)
    -
>assertDontSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterStateFinishedClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['clinic_case_status' => 'finished'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=&filter=finished');
    $response->assertStatus(200)
    ->assertSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterStateFinishedClinicCaseAsAdminFailWithNoClinicCas
eAssociated()
  {
```

```
    $clinicCase = factory('App\ClinicCase')-
>create(['clinic_case_status' => 'inprogress'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=&filter=finished');
    $response->assertStatus(200)
    -
>assertDontSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterBacterialIsolateClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['bacterial_isolate' => 'test'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=&filter=bacterial_isola
te');
    $response->assertStatus(200)
    ->assertSee((string)$clinicCase['number_clinic_history']);
  }
```

```
  public function
testIndexFilterBacterialIsolateClinicCaseAsAdminFailWithNoClinic
CaseAssociated()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['bacterial_isolate' => null])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=&filter=bacterial_isola
te');
    $response->assertStatus(200)
    -
>assertDontSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterFungiIsolateClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['fungi_isolate' => 'test'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=&filter=fungi_isolate')
;
```

```
    $response->assertStatus(200)
    ->assertSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterFungiIsolateClinicCaseAsAdminFailWithNoClinicCase
Associated()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['fungi_isolate' => null])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=&filter=fungi_isolate')
;
    $response->assertStatus(200)
    -
>assertDontSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterLocalizationClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['localization' => 'ear'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');
```

```
    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=ear&number_clinic_history=&filter=localization
');
    $response->assertStatus(200)
    ->assertSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterLocalizationClinicCaseAsAdminFailWithNoClinicCase
Associated()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['localization' => 'skin'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=ear&number_clinic_history=&filter=localization
');
    $response->assertStatus(200)
    -
>assertDontSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterNumberClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['number_clinic_history' => '321'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
```

```
    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=321&filter=number_clini
c_history');
    $response->assertStatus(200)
    ->assertSee((string)$clinicCase['number_clinic_history']);
  }


  public function
testIndexFilterNumberClinicCaseAsAdminFailWithNoClinicCaseAssoci
ated()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['number_clinic_history' => '123'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-
case?localization=&number_clinic_history=321&filter=number_clini
c_history');
    $response->assertStatus(200)
    -
>assertDontSee((string)$clinicCase['number_clinic_history']);
  }
}
<?php
```

```
namespace Tests\Feature\ClinicCases;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class ShowClinicCaseTest extends TestCase
{
  public function testShowClinicCaseFailWithoutLoginUser()
  {
    $clinicCase = factory('App\ClinicCase')->create()->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);

    $response = $this->get('/lab/clinic-case/' . $clinicCase['id']);
    $response->assertStatus(302)->assertRedirect('/');
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
  }

  public function testShowClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')->create()->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);

    $user = $this->createUserWithAdminPermissions('clinic_cases');

    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-case/' . $clinicCase['id']);
```

```
    $response->assertStatus(200)-
>assertSee((string)$clinicCase['number_clinic_history']);
  }

  public function testShowClinicCaseFailLogedAsUser()
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->get('/lab/clinic-case/' . $clinicCase['id']);
    $response->assertStatus(302)
    ->assertRedirect('/');
  }

  public function testEmailClinicCaseFinishedAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['clinic_case_status' => 'finished'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);

    $user = $this-
>createUserWithAdminPermissions('clinic_cases');

    $response = $this
    ->actingAs($user)
    ->get('/lab/email/' . $clinicCase['id']);
    $response->assertStatus(302)->assertSessionHas('suc');
  }
```

```
  public function testEmailClinicCaseInProgressAsAdminFail()
  {
    $clinicCase = factory('App\ClinicCase')-
>create(['clinic_case_status' => 'inprogress'])->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);


    $user = $this-
>createUserWithAdminPermissions('clinic_cases');


    $response = $this
    ->actingAs($user)
    ->get('/lab/email/' . $clinicCase['id']);
    $response->assertStatus(302)->assertSessionHas('fail');
  }
}
<?php


namespace Tests\Feature\ClinicCases;


use Tests\TestCase;
use Illuminate\Support\Facades\Session;


class UpdateClinicCaseTest extends TestCase
{
  public function testUpdateClinicCaseFailWithoutLoginUser()
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $clinicCase2 = factory('App\ClinicCase')->make()->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
    $response = $this->put('/lab/clinic-case/' .
$clinicCase['id'] . '/edit', $clinicCase2);
```

```
    $response->assertStatus(302)->assertRedirect('/');
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
  }

  public function testUpdateClinicCaseAsAdminSuccess()
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $clinicCase2 = factory('App\ClinicCase')->make()->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);

    $user = $this-
>createUserWithAdminPermissions('clinic_cases');

    $response = $this
    ->actingAs($user)
    ->put('/lab/clinic-case/' . $clinicCase['id'] . '/edit',
$clinicCase2);
    $response->assertStatus(302)->assertRedirect('/lab/clinic-
case');
    $this->assertDatabaseHas('clinic_cases', $clinicCase2);
  }

  public function testUpdateClinicCaseFailLogedAsUser()
  {
    $clinicCase = factory('App\ClinicCase')->create()-
>toArray();
    $clinicCase2 = factory('App\ClinicCase')->make()->toArray();
    $this->assertDatabaseHas('clinic_cases', $clinicCase);

    $user = $this->createUserWithUserPermissions();

    $response = $this
```

```
    ->actingAs($user)
    ->put('/lab/clinic-case/' . $clinicCase['id'] . '/edit',
$clinicCase2);
    $response->assertStatus(302)
    ->assertRedirect('/');
    $this->assertDatabaseHas('clinic_cases', $clinicCase);
  }
}
```

Tests Home

```
<?php

namespace Tests\Feature\Home;

use Tests\TestCase;

class HomeTest extends TestCase
{
  public function testHome()
  {
    $response = $this->call('GET', '/', [], [], [], []);
    $response->assertSee('Bienvenido')
    ->assertStatus(200);
  }

  public function testMycology()
  {
    $response = $this->call('GET', '/mycology/generalidades',
[], [], [], []);
    $response->assertSee('Generalidades')
    ->assertStatus(200);
  }

  public function testLaboratory()
  {
    $response = $this->call('GET', '/lab', [], [], [], []);
    $response->assertSee('Lista de precios')
    ->assertStatus(200);
  }
```

```
  public function testProjectsFailWithoutProjectsInDatabase()
  {
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertDontSee('Publicaciones')
    ->assertStatus(200);
  }


  public function
testProjectsSuccessWithPublicationsInDatabase()
  {
    $project = factory('App\Project')->create(['project_type' =>
'Publicación'])->toArray();
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertSee('Publicaciones')
    ->assertStatus(200);
    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));
  }


  public function testProjectsSuccessWithProjectsInDatabase()
  {
    $project = factory('App\Project')->create(['project_type' =>
'Proyecto de investigación'])->toArray();
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertSee('Proyectos')
    ->assertStatus(200);
    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));
  }


  public function testProjectsSuccessWithTFGInDatabase()
  {
    $project = factory('App\Project')->create(['project_type' =>
'Trabajo fin grado'])->toArray();
```

```
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertSee('Trabajos fin de grado')
    ->assertStatus(200);
    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));
  }

  public function testProjectsSuccessWithTPGInDatabase()
  {
    $project = factory('App\Project')->create(['project_type' =>
'Trabajo Post-grado'])->toArray();
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertSee('Trabajos Post')
    ->assertStatus(200);
    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));
  }

  public function testProjectsSuccessWithTesisInDatabase()
  {
    $project = factory('App\Project')->create(['project_type' =>
'Tesis'])->toArray();
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertSee('Tesis')
    ->assertStatus(200);
    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));
  }

  public function testContactFormFailWithoutCorrectInput()
  {
    $data = ['name' => 'test', 'email' => 'test', 'subject' =>
'test', 'message' => 'test'];
```

```
    $response = $this->call('POST', '/contact', $data, [], [],
[]);
    $response->assertSessionHas('fail')-
>assertRedirect('/#contact')->assertStatus(302);
  }

  public function testContactFormSuccessWithCorrectInput()
  {
    $data = ['name' => 'test', 'email' => 'test@test.com',
'subject' => 'test', 'message' => 'test'];
    $response = $this->call('POST', '/contact', $data, [], [],
[]);
    $response->assertSessionHas('suc')-
>assertRedirect('/#contact')->assertStatus(302);
  }
}
<?php

namespace Tests\Feature\Home;

use Tests\TestCase;

class HomeTest extends TestCase
{
  public function testHome()
  {
    $response = $this->call('GET', '/', [], [], [], []);
    $response->assertSee('Bienvenido')
    ->assertStatus(200);
  }

  public function testMycology()
  {
```

```
    $response = $this->call('GET', '/mycology/generalidades',
[], [], [], []);
    $response->assertSee('Generalidades')
    ->assertStatus(200);
  }

  public function testLaboratory()
  {
    $response = $this->call('GET', '/lab', [], [], [], []);
    $response->assertSee('Lista de precios')
    ->assertStatus(200);
  }

  public function testProjectsFailWithoutProjectsInDatabase()
  {
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertDontSee('Publicaciones')
    ->assertStatus(200);
  }

  public function
testProjectsSuccessWithPublicationsInDatabase()
  {
    $project = factory('App\Project')->create(['project_type' =>
'Publicación'])->toArray();
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertSee('Publicaciones')
    ->assertStatus(200);
    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));
  }

  public function testProjectsSuccessWithProjectsInDatabase()
```

```
  {
    $project = factory('App\Project')->create(['project_type' =>
'Proyecto de investigación'])->toArray();
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertSee('Proyectos')
    ->assertStatus(200);
    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));
  }

  public function testProjectsSuccessWithTFGInDatabase()
  {
    $project = factory('App\Project')->create(['project_type' =>
'Trabajo fin grado'])->toArray();
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertSee('Trabajos fin de grado')
    ->assertStatus(200);
    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));
  }

  public function testProjectsSuccessWithTPGInDatabase()
  {
    $project = factory('App\Project')->create(['project_type' =>
'Trabajo Post-grado'])->toArray();
    $response = $this->call('GET', '/projects', [], [], [], []);
    $response->assertSee('Trabajos Post')
    ->assertStatus(200);
    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));
  }

  public function testProjectsSuccessWithTesisInDatabase()
  {
```

```
    $project = factory('App\Project')->create(['project_type' =>
'Tesis'])->toArray();

    $response = $this->call('GET', '/projects', [], [], [], []);

    $response->assertSee('Tesis')

    ->assertStatus(200);

    $this->assertDatabaseHas('projects', array_splice($project,
0, 1));

  }


  public function testContactFormFailWithoutCorrectInput()

  {

    $data = ['name' => 'test', 'email' => 'test', 'subject' =>
'test', 'message' => 'test'];

    $response = $this->call('POST', '/contact', $data, [], [],
[]);

    $response->assertSessionHas('fail')-
>assertRedirect('/#contact')->assertStatus(302);

  }


  public function testContactFormSuccessWithCorrectInput()

  {

    $data = ['name' => 'test', 'email' => 'test@test.com',
'subject' => 'test', 'message' => 'test'];

    $response = $this->call('POST', '/contact', $data, [], [],
[]);

    $response->assertSessionHas('suc')-
>assertRedirect('/#contact')->assertStatus(302);

  }

}
```

Tests Proyectos

```
<?php

namespace Tests\Feature\Projects;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class DeleteProjectTest extends TestCase
{
  public function testDeleteProjectFailWithoutLoginUser()
  {
    $project = factory('App\Project')->create()->toArray();
    $this->assertDatabaseHas('projects', $project);
    $response = $this->delete('/project-manager/delete/' .
$project['id']);
    $response->assertStatus(302)->assertRedirect('/');
    $this->assertDatabaseHas('projects', $project);
  }

  public function testDeleteProjectAsAdminSuccess()
  {
    $project = factory('App\Project')->create()->toArray();
    $this->assertDatabaseHas('projects', $project);

    $user = $this->createUserWithAdminPermissions('projects');

    $response = $this
    ->actingAs($user)
    ->delete('/project-manager/delete/' . $project['id']);
```

```
    $response->assertStatus(200);
    $this->assertDatabaseMissing('projects', $project);
  }

  public function testDeleteProjectFailLogedAsUser()
  {
    $project = factory('App\Project')->create()->toArray();
    $this->assertDatabaseHas('projects', $project);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->delete('/project-manager/delete/' . $project['id']);

    $response->assertRedirect('/')
    ->assertStatus(302);
    $this->assertDatabaseHas('projects', $project);
  }
}
<?php

namespace Tests\Feature\Projects;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class IndexProjectTest extends TestCase
{
  public function testIndexProjectFailWithoutLoginUser()
  {
    $project = factory('App\Project')->create()->toArray();
```

```
  $this->assertDatabaseHas('projects', $project);
  $response = $this->get('/project-manager');
  $response->assertStatus(302)->assertRedirect('/');
  $this->assertDatabaseHas('projects', $project);
}

public function testIndexProjectAsAdminSuccess()
{
  $project = factory('App\Project')->create()->toArray();
  $this->assertDatabaseHas('projects', $project);

  $user = $this->createUserWithAdminPermissions('projects');

  $response = $this
  ->actingAs($user)
  ->get('/project-manager');
  $response->assertStatus(200)
  ->assertSee((string)$project['project_name']);
}

public function testIndexProjectFailLogedAsUser()
{
  $project = factory('App\Project')->create()->toArray();
  $this->assertDatabaseHas('projects', $project);

  $user = $this->createUserWithUserPermissions();

  $response = $this
  ->actingAs($user)
  ->get('/project-manager');
  $response->assertStatus(302)
  ->assertRedirect('/');
```

```
  }
}
<?php

namespace Tests\Feature\Projects;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class ShowProjectTest extends TestCase
{
  public function testShowProjectAsAnonimateUserSuccess()
  {
    $project = factory('App\Project')->create()->toArray();
    $this->assertDatabaseHas('projects', $project);

    $response = $this->get('/project-manager/' .
$project['id']);
    $response->assertStatus(200)-
>assertSee((string)$project['project_name']);
  }

  public function testShowProjectAsAdminSuccess()
  {
    $project = factory('App\Project')->create()->toArray();
    $this->assertDatabaseHas('projects', $project);

    $user = $this->createUserWithAdminPermissions('projects');

    $response = $this
    ->actingAs($user)
    ->get('/project-manager/' . $project['id']);
```

```
    $response->assertStatus(200)-
>assertSee((string)$project['project_name']);
  }

  public function testShowProjectSuccessLogedAsUser()
  {
    $project = factory('App\Project')->create()->toArray();
    $this->assertDatabaseHas('projects', $project);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->get('/project-manager/' . $project['id']);
    $response->assertStatus(200)-
>assertSee((string)$project['project_name']);
  }
}
<?php

namespace Tests\Feature\Projects;

use Tests\TestCase;
use Illuminate\Support\Facades\Session;

class UpdateProjectTest extends TestCase
{


  public function testUpdateProjectFailWithoutLoginUser()
  {
    $project = factory('App\Project')->create()->toArray();
```

```
    $project2 = factory('App\Project')->make()->toArray();

    $this->assertDatabaseHas('projects', $project);

    $response = $this->put('/project-manager/' . $project['id']
. '/edit', $project2);

    $response->assertStatus(302)->assertRedirect('/');

    $this->assertDatabaseHas('projects', $project);

  }


  public function testUpdateProjectAsAdminSuccess()

  {

    $image = \Illuminate\Http\UploadedFile::fake()-
>create('image.png', $kilobytes = 1);

    $project = factory('App\Project')->create()->toArray();

    $project2 = factory('App\Project')->make(['project_name' =>
'test' ,'image' => $image])->toArray();

    $this->assertDatabaseHas('projects', $project);


    $user = $this->createUserWithAdminPermissions('projects');


    $response = $this

    ->actingAs($user)

    ->put('/project-manager/' . $project['id'] . '/edit',
$project2);

    $response->assertStatus(302)->assertRedirect('/project-
manager');

    $this->assertDatabaseHas('projects', array_splice($project2,
0, 1));

  }


  public function testUpdateProjectFailLogedAsUser()

  {

    $project = factory('App\Project')->create()->toArray();

    $project2 = factory('App\Project')->make()->toArray();

    $this->assertDatabaseHas('projects', $project);
```

```
    $user = $this->createUserWithUserPermissions();


    $response = $this

    ->actingAs($user)

    ->put('/project-manager/' . $project['id'] . '/edit',
$project2);

    $response->assertStatus(302)

    ->assertRedirect('/');

    $this->assertDatabaseHas('projects', $project);

  }

}
```

Tests Roles

```
<?php

namespace Tests\Feature\Roles;

use Tests\TestCase;
use TCG\Voyager\Models\Role;
use Artisan;

class CreateRoleTest extends TestCase
{
  public function testCreateRoleFailWithoutLoginUser()
  {
    $role = Role::firstOrNew(['name' => 'test']);
    if (!$role->exists) {
      $role->fill([
        'display_name' => 'test',
      ]);
    }
    $response = $this->call('POST', '/admin/roles', $role-
>toArray(), [], [], []);
    $response->assertStatus(302)-
>assertRedirect('/admin/login');
    $this->assertDatabaseMissing('roles', $role->toArray());
  }

  public function testCreateRoleSuccessAsAdmin()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $role = Role::firstOrNew(['name' => 'test']);
    if (!$role->exists) {
```

```
      $role->fill([
        'display_name' => 'test',
      ]);
    }
    $role = $role->toArray();
    $user = $this->createUserWithAdminPermissions('roles');

    $response = $this
    ->actingAs($user)
    ->post('/admin/roles', $role);
    $response->assertRedirect('/admin/roles')
    ->assertStatus(302);
    $this->assertDatabaseHas('roles', array_splice($role, 0,
1));
  }

  public function testCreateRoleFailLogedAsUser()
  {
    $role = Role::firstOrNew(['name' => 'test']);
    if (!$role->exists) {
      $role->fill([
        'display_name' => 'test',
      ]);
    }

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->post('/admin/roles', $role->toArray());

    $response->assertRedirect('/')
```

```
    ->assertStatus(302);
    $this->assertDatabaseMissing('roles', $role->toArray());
  }
}
<?php

namespace Tests\Feature\Roles;

use Tests\TestCase;
use TCG\Voyager\Models\Role;
use Artisan;

class DeleteRoleTest extends TestCase
{
  public function testDeleteRoleFailWithoutLoginUser()
  {
    $user = $this->createUserWithAdminPermissions('roles');
    $role = $this->createRole('test')->toArray();
    $this->assertDatabaseHas('roles', $role);
    $response = $this->delete('/admin/roles/' . $role['id']);
    $response->assertStatus(302)-
>assertRedirect('/admin/login');
    $this->assertDatabaseHas('roles', $role);
  }


  public function testDeleteRoleAsAdminSuccess()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $user = $this->createUserWithAdminPermissions('roles');


    $role = $this->createRole('test')->toArray();
```

```
    $response = $this
    ->actingAs($user)
    ->delete('/admin/roles/' . $role['id']);

    $response->assertStatus(302);
    $this->assertDatabaseMissing('roles', $role);
  }

  public function testDeleteRoleFailLogedAsUser()
  {
    $role = $this->createRole('test')->toArray();
    $this->assertDatabaseHas('roles', $role);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->delete('/admin/roles/' . $role['id']);
    $response->assertStatus(302)
    ->assertRedirect('/');
  }
}
<?php

namespace Tests\Feature\Roles;

use Tests\TestCase;
use TCG\Voyager\Models\Role;
use Artisan;

class IndexRoleTest extends TestCase
{
```

```
  public function testIndexRoleFailWithoutLoginUser()
  {
    $user = $this->createUserWithAdminPermissions('roles');
    $role = $this->createRole('test')->toArray();
    $this->assertDatabaseHas('roles', $role);
    $response = $this->get('/admin/roles');
    $response->assertStatus(302)-
>assertRedirect('/admin/login');
    $this->assertDatabaseHas('roles', $role);
  }

  public function testIndexRoleAsAdminSuccess()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $user = $this->createUserWithAdminPermissions('roles');

    $role = $this->createRole('test')->toArray();
    $this->assertDatabaseHas('roles', $role);
    $response = $this
    ->actingAs($user)
    ->get('/admin/roles');
    $response->assertStatus(200)
    ->assertSee((string)$role['name']);
  }

  public function testIndexRoleFailLogedAsUser()
  {
    $role = $this->createRole('test')->toArray();
    $this->assertDatabaseHas('roles', $role);

    $user = $this->createUserWithUserPermissions();
```

```
    $response = $this
    ->actingAs($user)
    ->get('/admin/roles');
    $response->assertStatus(302)
    ->assertRedirect('/');
  }
}
<?php

namespace Tests\Feature\Roles;

use Tests\TestCase;
use TCG\Voyager\Models\Role;
use Artisan;

class ShowRoleTest extends TestCase
{
  public function testShowRoleAsUserSuccess()
  {
    $role = Role::firstOrNew(['name' => 'test']);
    if (!$role->exists) {
      $role->fill([
        'display_name' => 'test',
        ])->save();
      }
      $role = $role->toArray();
      $this->assertDatabaseHas('roles', $role);

      $response = $this->get('/admin/roles/' . $role['id']);
      $response->assertStatus(302)-
>assertRedirect('/admin/login');
    }
```

```
    public function testShowRoleAsAdminSuccess()
    {
      Artisan::call('db:seed', ['--database' => 'sqlite']);
      $role = Role::firstOrNew(['name' => 'test']);
      if (!$role->exists) {
        $role->fill([
          'display_name' => 'test',
          ])->save();
        }
        $role = $role->toArray();
        $this->assertDatabaseHas('roles', $role);

        $user = $this->createUserWithAdminPermissions('roles');

        $response = $this
        ->actingAs($user)
        ->get('/admin/roles/' . $role['id']);
        $response->assertStatus(200)-
>assertSee((string)$role['name']);
      }

      public function testShowRoleFailLogedAsUser()
      {
        $role = Role::firstOrNew(['name' => 'test']);
        if (!$role->exists) {
          $role->fill([
            'display_name' => 'test',
            ])->save();
          }
          $role = $role->toArray();
          $this->assertDatabaseHas('roles', $role);
```

```
            $user = $this->createUserWithUserPermissions();

            $response = $this
            ->actingAs($user)
            ->get('/admin/roles/' . $role['id']);
            $response->assertStatus(302)->assertRedirect('/');
          }
        }
<?php

namespace Tests\Feature\Roles;

use Tests\TestCase;
use TCG\Voyager\Models\Role;
use Artisan;

class ShowRoleTest extends TestCase
{
  public function testShowRoleAsUserSuccess()
  {
    $role = Role::firstOrNew(['name' => 'test']);
    if (!$role->exists) {
      $role->fill([
        'display_name' => 'test',
        ])->save();
      }
      $role = $role->toArray();
      $this->assertDatabaseHas('roles', $role);

      $response = $this->get('/admin/roles/' . $role['id']);
```

```
      $response->assertStatus(302)-
>assertRedirect('/admin/login');
    }

    public function testShowRoleAsAdminSuccess()
    {
      Artisan::call('db:seed', ['--database' => 'sqlite']);
      $role = Role::firstOrNew(['name' => 'test']);
      if (!$role->exists) {
        $role->fill([
          'display_name' => 'test',
          ])->save();
        }
        $role = $role->toArray();
        $this->assertDatabaseHas('roles', $role);

        $user = $this->createUserWithAdminPermissions('roles');

        $response = $this
        ->actingAs($user)
        ->get('/admin/roles/' . $role['id']);
        $response->assertStatus(200)-
>assertSee((string)$role['name']);
      }

      public function testShowRoleFailLogedAsUser()
      {
        $role = Role::firstOrNew(['name' => 'test']);
        if (!$role->exists) {
          $role->fill([
            'display_name' => 'test',
            ])->save();
```

```
    }
    $role = $role->toArray();
    $this->assertDatabaseHas('roles', $role);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->get('/admin/roles/' . $role['id']);
    $response->assertStatus(302)->assertRedirect('/');
  }
}
```

Tests Usuarios

```
<?php

namespace Tests\Feature\Users;

use Tests\TestCase;
use TCG\Voyager\Models\User;
use TCG\Voyager\Models\Role;
use Artisan;

class CreateUserTest extends TestCase
{
  public function testCreateUserFailWithoutLoginUser()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->make()->toArray();
    $response = $this->call('POST', '/admin/users', $u, [], [],
[]);
    $response->assertStatus(302);
    $this->assertDatabaseMissing('users', $u);
  }

  public function testCreateUserSuccessAsAdmin()
  {
    $user = $this->createUserWithAdminPermissions('users');
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $role = Role::where('name', 'admin')->firstOrFail();
    $u = factory('TCG\Voyager\Models\User')->make(['password' =>
'secret'])->toArray();

    $response = $this
```

```
    ->actingAs($user)
    ->post('/admin/users', $u);

    $u = User::where(['name' => $u['name']])->first()-
>toArray();
    $response->assertRedirect('/admin/users/'.$u['id'].'/edit')
    ->assertStatus(302);
    $this->assertDatabaseHas('users', array_splice($u, 0, 1));
  }

  public function testCreateUserFailLogedAsUser()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->make()->toArray();

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->post('/admin/users', $u);

    $response->assertRedirect('/');
    $this->assertDatabaseMissing('users', $u);
  }
}
<?php

namespace Tests\Feature\Users;

use Tests\TestCase;
use TCG\Voyager\Models\User;
use Artisan;
```

```
class DeleteUserTest extends TestCase
{
  public function testDeleteUserFailWithoutLoginUser()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $this->assertDatabaseHas('users', $u);
    $response = $this->call('DELETE', '/admin/users/' .
$u['id']);
    $response->assertStatus(302);
    $this->assertDatabaseHas('users', $u);
  }

  public function testDeleteUserAsAdminSuccess()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $user = $this->createUserWithAdminPermissions('users');

    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();

    $response = $this
    ->actingAs($user)
    ->delete('/admin/users/' . $u['id']);

    $response->assertStatus(302);
    $this->assertDatabaseMissing('users', $u);
  }

  public function testDeleteUserFailLogedAsUser()
  {
```

```
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $this->assertDatabaseHas('users', $u);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->delete('/admin/users/' . $u['id']);
    $response->assertStatus(302)
    ->assertRedirect('/');
  }
}
<?php

namespace Tests\Feature\Users;

use Tests\TestCase;
use TCG\Voyager\Models\User;
use Artisan;

class IndexUserTest extends TestCase
{
  public function testIndexUserFailWithoutLoginUser()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $this->assertDatabaseHas('users', $u);
    $response = $this->call('GET', '/admin/users/');
    $response->assertStatus(302)-
>assertRedirect('/admin/login');
```

```
    $this->assertDatabaseHas('users', $u);
  }

  public function testIndexUserAsAdminSuccess()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $user = $this->createUserWithAdminPermissions('users');

    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $this->assertDatabaseHas('users', $u);
    $response = $this
    ->actingAs($user)
    ->get('/admin/users');
    $response->assertStatus(200)
    ->assertSee((string)$u['name']);
  }

  public function testIndexUserFailLogedAsUser()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $this->assertDatabaseHas('users', $u);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->get('/admin/users');
    $response->assertStatus(302)
    ->assertRedirect('/');
```

```
  }
}
<?php

namespace Tests\Feature\Users;

use Tests\TestCase;
use TCG\Voyager\Models\User;
use Artisan;

class ShowUserTest extends TestCase
{
  public function testShowUserAsUserFail()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $this->assertDatabaseHas('users', $u);

    $response = $this->get('/admin/users/' . $u['id']);
    $response->assertStatus(302)-
>assertRedirect('/admin/login');
  }

  public function testShowUserAsAdminSuccess()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $this->assertDatabaseHas('users', $u);

    $user = $this->createUserWithAdminPermissions('users');
```

```
    $response = $this
    ->actingAs($user)
    ->get('/admin/users/' . $u['id']);
    $response->assertStatus(200)->assertSee((string)$u['name']);
  }

  public function testShowUserFailLogedAsUser()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $this->assertDatabaseHas('users', $u);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->get('/admin/users/' . $u['id']);
    $response->assertStatus(302)->assertRedirect('/');
  }
}
<?php

namespace Tests\Feature\Users;

use Tests\TestCase;
use TCG\Voyager\Models\User;
use Artisan;

class UpdateUserTest extends TestCase
{
  public function testUpdateUserFailWithoutLoginUser()
```

```
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $u2 = factory('TCG\Voyager\Models\User')->make()->toArray();
    $this->assertDatabaseHas('users', $u);
    $response = $this->put('/admin/users/' . $u['id'], $u2);
    $response->assertStatus(302)-
>assertRedirect('/admin/login');
    $this->assertDatabaseHas('users', $u);
  }


  public function testUpdateUserAsAdminSuccess()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $u2 = User::firstOrNew(['name' => 'test']);
    $u2 = factory('TCG\Voyager\Models\User')->make()->toArray();
    $this->assertDatabaseHas('users', $u);


    $user = $this->createUserWithAdminPermissions('users');


    $response = $this
    ->actingAs($user)
    ->put('/admin/users/' . $u['id'], $u2);
    $response->assertStatus(302)-
>assertRedirect('/admin/users/'.$u['id'].'/edit');
    $this->assertDatabaseHas('users', array_splice($u2, 0, 1));
  }


  public function testUpdateUserFailLogedAsUser()
  {
```

```
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $u2 = User::firstOrNew(['name' => 'test']);
    $u2 = factory('TCG\Voyager\Models\User')->make()->toArray();
    $this->assertDatabaseHas('users', $u);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->put('/admin/users/' . $u['id'], $u2);
    $response->assertStatus(302)
    ->assertRedirect('/');
    $this->assertDatabaseHas('users', $u);
  }
}
<?php

namespace Tests\Feature\Users;

use Tests\TestCase;
use TCG\Voyager\Models\User;
use Artisan;

class UpdateUserTest extends TestCase
{
  public function testUpdateUserFailWithoutLoginUser()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
```

```
    $u2 = factory('TCG\Voyager\Models\User')->make()->toArray();
    $this->assertDatabaseHas('users', $u);
    $response = $this->put('/admin/users/' . $u['id'], $u2);
    $response->assertStatus(302)-
>assertRedirect('/admin/login');
    $this->assertDatabaseHas('users', $u);
  }

  public function testUpdateUserAsAdminSuccess()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $u2 = User::firstOrNew(['name' => 'test']);
    $u2 = factory('TCG\Voyager\Models\User')->make()->toArray();
    $this->assertDatabaseHas('users', $u);

    $user = $this->createUserWithAdminPermissions('users');

    $response = $this
    ->actingAs($user)
    ->put('/admin/users/' . $u['id'], $u2);
    $response->assertStatus(302)-
>assertRedirect('/admin/users/'.$u['id'].'/edit');
    $this->assertDatabaseHas('users', array_splice($u2, 0, 1));
  }

  public function testUpdateUserFailLogedAsUser()
  {
    Artisan::call('db:seed', ['--database' => 'sqlite']);
    $u = factory('TCG\Voyager\Models\User')->create()-
>toArray();
    $u2 = User::firstOrNew(['name' => 'test']);
```

```
    $u2 = factory('TCG\Voyager\Models\User')->make()->toArray();
    $this->assertDatabaseHas('users', $u);

    $user = $this->createUserWithUserPermissions();

    $response = $this
    ->actingAs($user)
    ->put('/admin/users/' . $u['id'], $u2);
    $response->assertStatus(302)
    ->assertRedirect('/');
    $this->assertDatabaseHas('users', $u);
  }
}
```