UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

Proyecto Unidad ll

“Mejoramientos de la Aplicación”

Curso: Calidad y Pruebas de Software

Docente: Ing. Cuadros Quiroga, Patrick Jose

Integrantes

LOPEZ CATUNTA, Brayar Christian	(2021069822)
BRICEÑO DIAZ, Jorge Luis	(2017059611)
CUADROS GARCIA, Mirian	(2021071083)
CASTAÑEDA CENTURION, Jorge Enrique	(2021069822)

Tacna – Perú
2024

INDICE GENERAL

I. INTRODUCCION

El presente trabajo aborda la creación de un sistema automatizado para la gestión de alquiler de vehículos, desarrollado con PHP8 y MySQL. El sistema se basa en la arquitectura MVC (Modelo-Vista-Controlador) para estructurar el código de manera eficiente y mantener una clara separación entre la lógica de negocios, la presentación y el control de la aplicación. Este sistema busca resolver problemas comunes en la gestión de alquiler de vehículos, como errores en las reservas, duplicaciones y una experiencia de usuario deficiente.

El proyecto se centra en la creación de una plataforma en línea que permite a los usuarios registrarse y realizar reservas de vehículos de manera sencilla y segura. Los administradores pueden gestionar el inventario de vehículos, las reservas y los clientes, así como generar reportes y enviar notificaciones por correo electrónico. La aplicación está diseñada para ser intuitiva y permite una fácil personalización para adaptarse a diferentes necesidades empresariales.

El sistema proporcionará una solución centralizada para la gestión de alquiler de vehículos, mejorando la eficiencia operativa y reduciendo los errores humanos. Además, ofrecerá una mejor experiencia al usuario, facilitando el proceso de alquiler y brindando un servicio más confiable y accesible.

II. RESUMEN

El proyecto tiene como objetivo principal desarrollar un sistema automatizado que facilite el proceso de alquiler de vehículos mediante una plataforma en línea. Este sistema permitirá a los usuarios registrarse y realizar reservas de vehículos, mientras que los administradores podrán gestionar el inventario de vehículos, las reservas y los clientes. Además, incluirá la generación de reportes y el envío de notificaciones por correo electrónico.

El sistema está diseñado para ser intuitivo y adaptable a diferentes necesidades empresariales, permitiendo una fácil personalización. La arquitectura MVC utilizada en el desarrollo del sistema garantiza una clara separación entre la lógica de negocios, la presentación y el control, facilitando así el mantenimiento y la escalabilidad del sistema.

- **Abstract**

The main objective of the project is to develop an automated system that facilitates the vehicle rental process through an online platform. This system will allow users to register and make vehicle reservations, while administrators can manage vehicle inventory, bookings, and customers. Additionally, it will include report generation and email notification functionalities.

The system is designed to be intuitive and adaptable to various business needs, enabling easy customization. The MVC architecture used in the system development ensures a clear separation between business logic, presentation, and control, thereby facilitating system maintenance and scalability.

III. OBJETIVOS

1. Objetivo General

Desarrollar un sistema automatizado que optimice la gestión de alquiler de vehículos, mejorando la eficiencia operativa y la experiencia del cliente.

Objetivos Específicos

2. Objetivos Específicos

- Implementar la funcionalidad de registro y autenticación de usuarios: Garantizando la seguridad y privacidad de la información.
- Desarrollar un módulo para la gestión de inventarios de vehículos: Permitiendo agregar, modificar y eliminar vehículos.
- Facilitar el proceso de reserva y alquiler de vehículos: Asegurando que los usuarios puedan realizar estas acciones de manera sencilla y rápida.
- Proveer herramientas administrativas para la gestión eficiente de clientes y reservas: Incluyendo la posibilidad de modificar y cancelar reservas.
- Integrar un sistema de notificaciones por correo electrónico: Para enviar confirmaciones y recordatorios a los usuarios.
- Implementar la generación de reportes detallados: Para análisis y toma de decisiones estratégicas.
- Este proyecto busca

IV. PLANTEAMIENTO DEL PROBLEMA

1. Problema

Las empresas de alquiler de vehículos enfrentan diversos desafíos en la gestión eficiente de sus operaciones. Los sistemas manuales o no automatizados pueden llevar a errores en las reservas, duplicaciones y una experiencia de usuario insatisfactoria. Además, la falta de un sistema centralizado dificulta la gestión efectiva del inventario de vehículos, las reservas y los clientes.

Un sistema manual de gestión puede resultar en una operación ineficiente, pérdida de clientes y una mala reputación para la empresa. Por lo tanto, es necesario un sistema automatizado que permita gestionar estos procesos de manera eficiente, reduciendo los errores humanos y mejorando la satisfacción del cliente.

2. Justificación

La implementación de un sistema automatizado para la gestión de alquiler de vehículos es crucial para aumentar la eficiencia operativa y reducir los errores. Un sistema centralizado proporciona una plataforma para gestionar de manera efectiva las reservas, los clientes y el inventario de vehículos, mejorando así la experiencia del usuario y la operatividad del negocio.

Un sistema automatizado permite realizar tareas de manera más rápida y precisa, reduciendo la carga de trabajo manual y permitiendo a los empleados concentrarse en tareas más estratégicas. Además, mejora la satisfacción del cliente al ofrecer un proceso de reserva sencillo y confiable, y proporciona una mejor visibilidad y control sobre las operaciones del negocio.

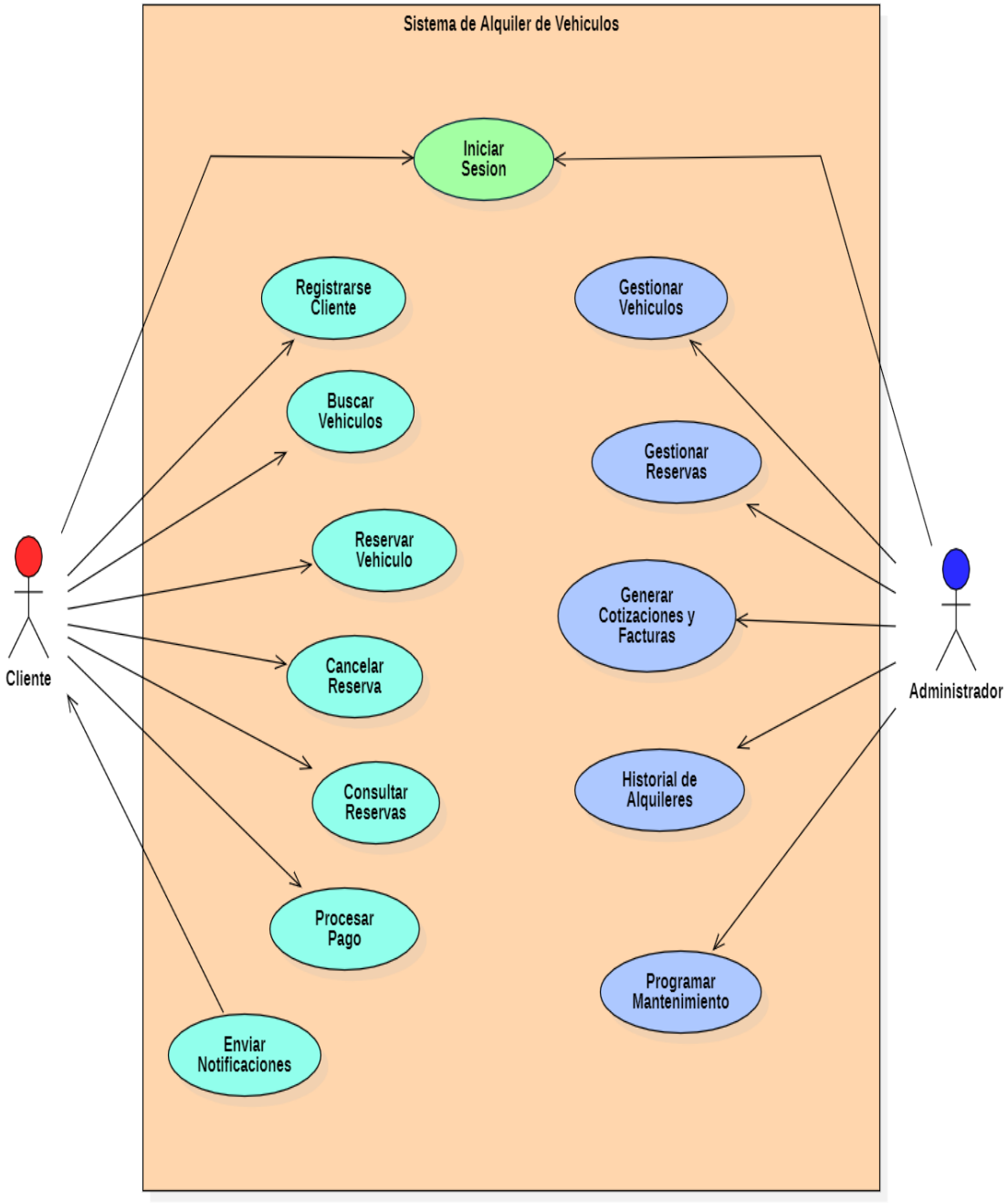
3. Alcance

El sistema abarcará:

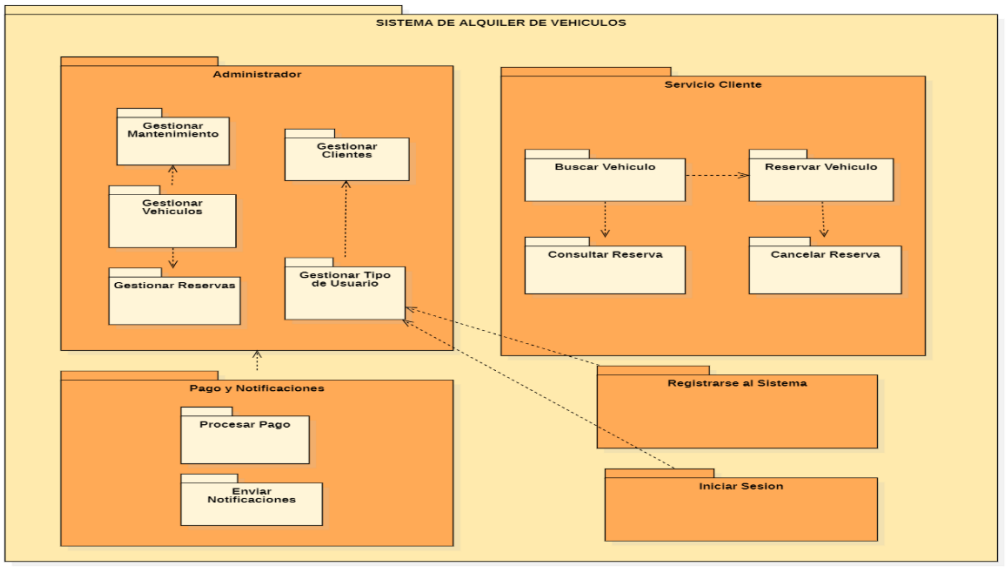
- Gestión de inventarios de vehículos: Permitiendo la alta, baja y modificación de vehículos en el sistema.
- Procesos de reserva y alquiler de vehículos: Los usuarios podrán realizar reservas y alquilar vehículos de manera sencilla y segura.
- Administración de clientes: Gestión de información de los clientes y sus reservas.
- Generación de reportes: Reportes detallados sobre las actividades del sistema para análisis y toma de decisiones estratégicas.
- Envío de notificaciones por correo electrónico: Confirmaciones y recordatorios automáticos para los usuarios.
- Interfaz de usuario intuitiva y personalizable: Adaptable a diferentes tipos de negocios de alquiler de vehículos.

V. DESARROLLO DEL TRABAJO

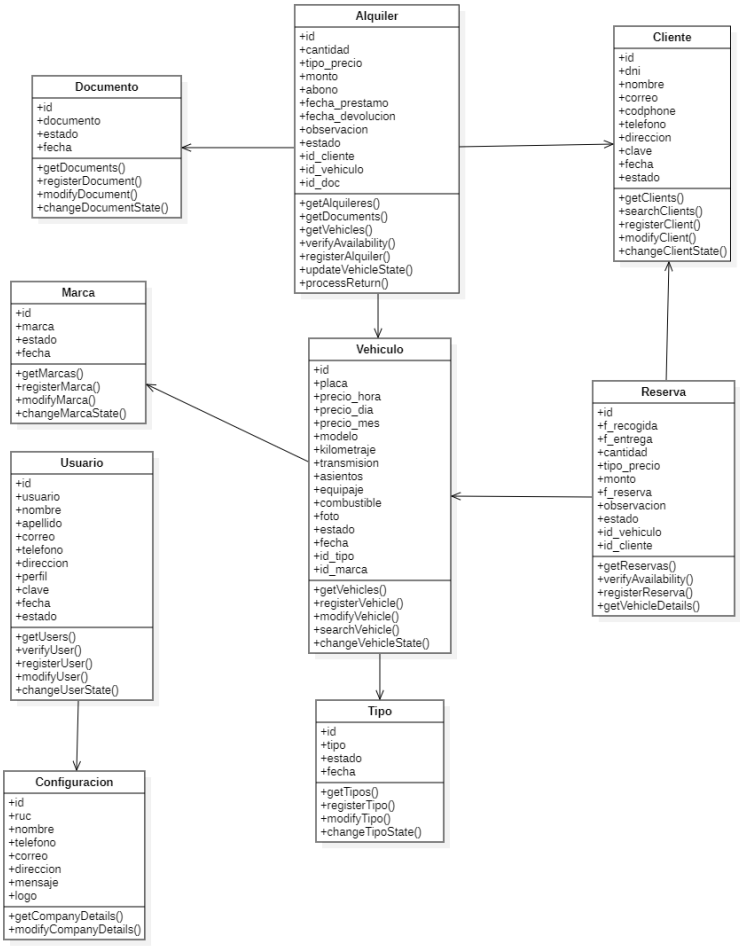
1. Diagrama de Caso de Uso



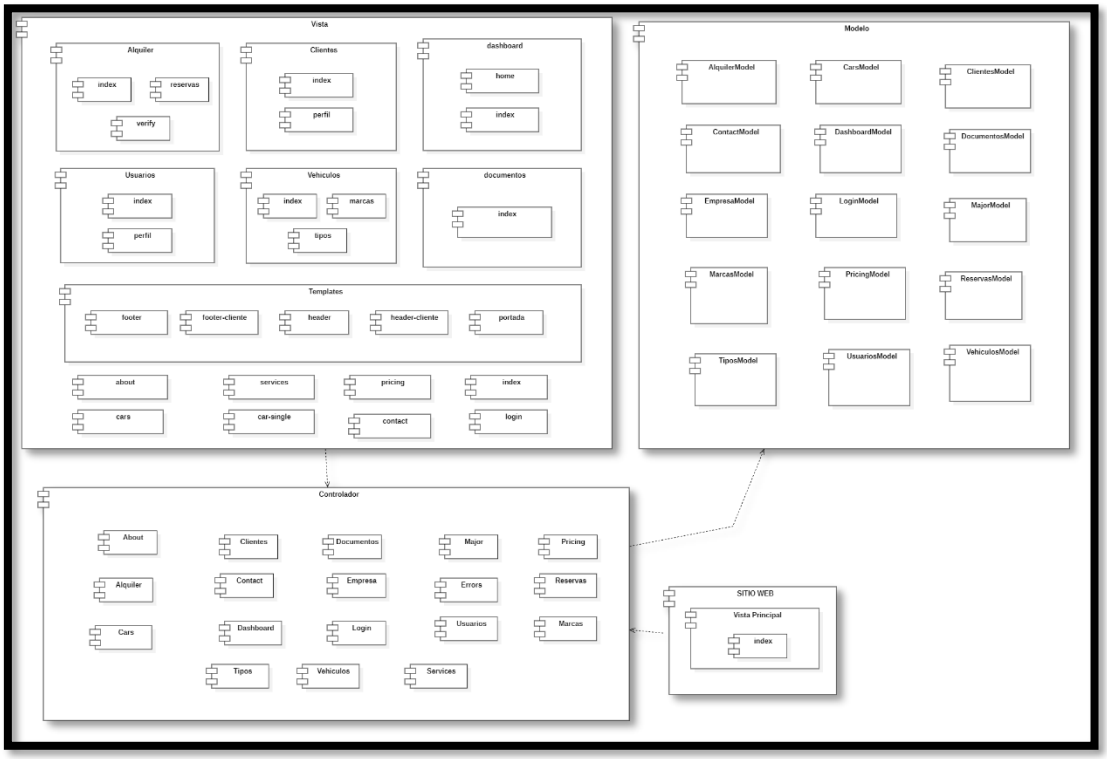
2. Diagrama de Paquetes



3. Diagrama de Clases



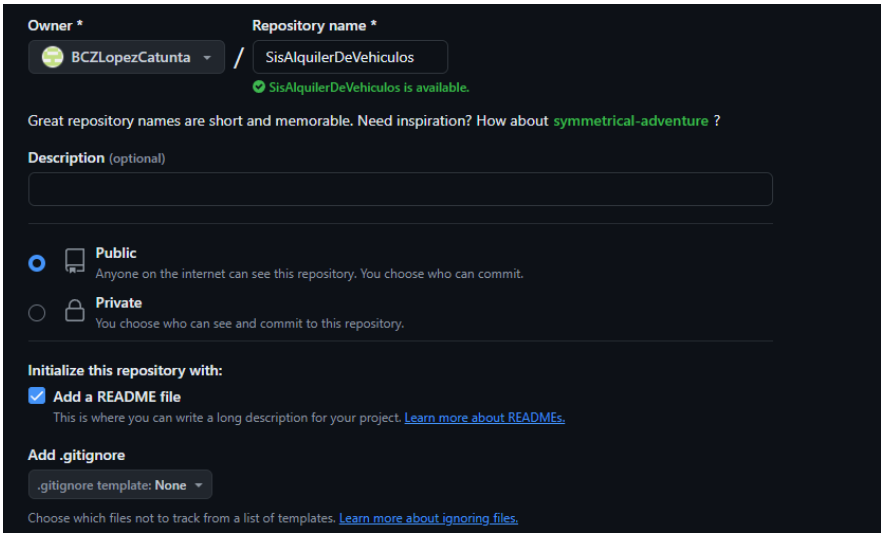
4. Crear Diagrama de Componentes



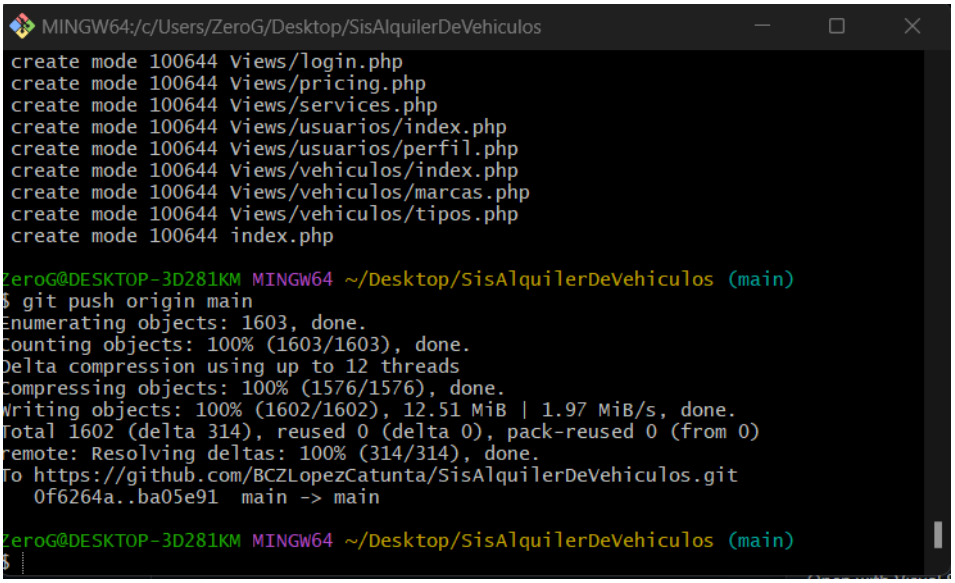
5. Analis de la aplicación con SonarQube y Snyk

- SonarQube

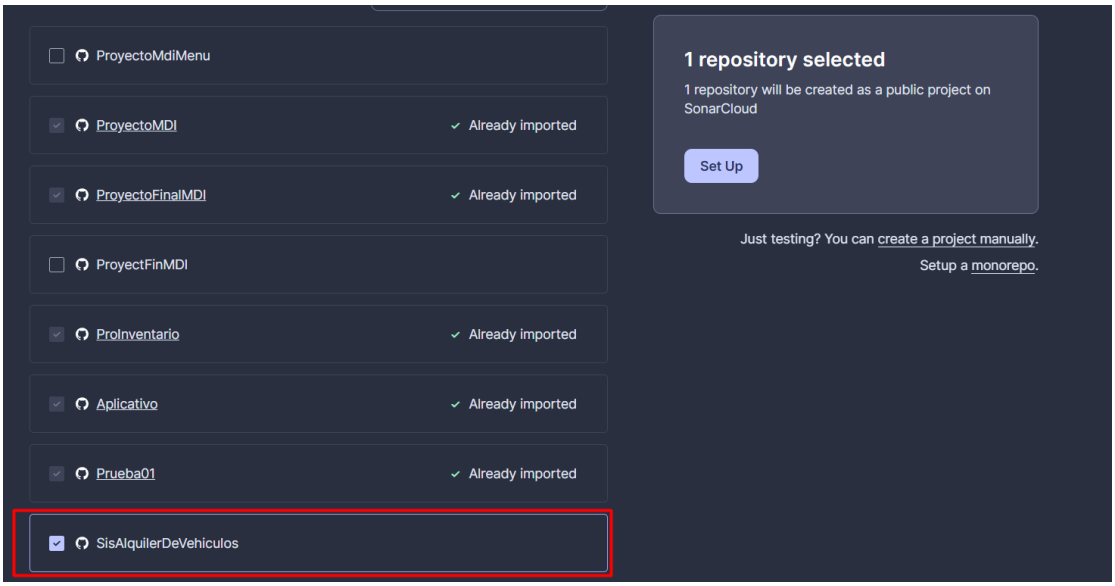
  Paso 01 : Se esta creando un repositorio público en tu cuenta personal llamado "SisAlquilerDeVehiculos", el cual contiene todos los archivos de tu proyecto y su historial de revisiones. Puedes inicializarlo con un archivo README y un archivo .gitignore, y elegir una licencia para tu código.



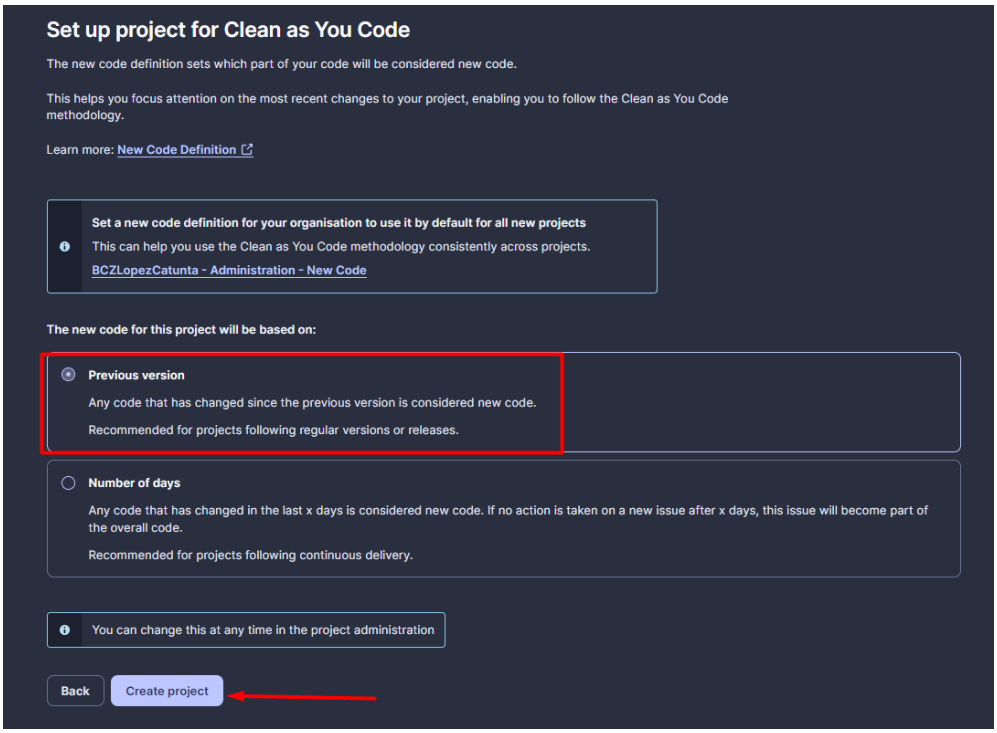
Paso 02: Se ha creado un nuevo repositorio en GitHub con la dirección URL "https://github.com/BCZLopezCatunta/SisAlquilerDeVehiculos.git" y ha cambiado la rama local existente a "main". Luego, ha agregado este repositorio a su sistema local en la carpeta "SisAlquilerDeVehiculos" en el escritorio y ha establecido una conexión remota con el repositorio de GitHub.



Paso 03: Selecciconamos el repositorio " SisAlquilerDeVehiculos" de la organización BCZLopezCatunta en GitHub para configurarlo en SonarCloud. Ahora, después de seleccionar el repositorio deseado, das clic en la opción "Setup" para continuar con la configuración y prepararlo para su análisis en SonarCloud. Este proceso te permitirá evaluar la calidad del código de tu proyecto "SisAlquilerDeVehiculos" utilizando las herramientas y funcionalidades de SonarCloud.



Paso 04: Seleccionamos "Previous Version" en el contexto dado, estás especificando que la nueva codificación para el proyecto se basará en la versión anterior del código. Esto significa que cualquier parte del código que haya sido modificada desde la última versión se considerará como "nuevo código". Esta opción es recomendada para proyectos que siguen versiones o lanzamientos regulares. Una vez que has seleccionado esta opción, al hacer clic en "Create project".



Paso 05: El Análisis de código realizado por SonarSource en el proyecto "SisAlquilerDeVehiculos" de BCZLopezCatunta muestra que el proyecto tiene la calidad del código se evalúa en un 5,2% Duplications, lo que sugiere que hay espacio para mejorar. En cuanto a la seguridad, se han detectado 429 bugs y 4 vulnerabilidades.

Paso 06: Se pasa a resolver las vulnerabilidades

- En el archivo index.php del porque el problema:

Impacto potencial de las inyecciones de inclusión

Si una aplicación es vulnerable a inyecciones de inclusión, puede ser susceptible a ataques dirigidos al servidor subyacente. Un usuario malintencionado puede crear solicitudes que filtren datos valiosos o logren la ejecución remota de código en el sitio web del servidor. El atacante puede llevar a cabo estos ataques sin depender de requisitos previos. El impacto de la vulnerabilidad depende de las medidas de control de acceso adoptadas en el sistema operativo de destino. En el peor de los casos, el proceso se ejecuta con privilegios de root, lo que significa que cualquier comando o programa del sistema operativo puede verse afectado.

Escenarios del mundo real que ilustran el impacto

a. Denegación de servicio y fuga de datos: El ataque tiene como objetivo interrumpir las actividades de la organización y beneficiarse de la filtración de datos. El atacante puede descargar datos del servidor interno para venderlos, modificar datos, enviar malware, detener servicios o agotar recursos. Esta amenaza es especialmente peligrosa si la organización atacada no tiene un plan de recuperación ante desastres.

b. Escalada y pivote de privilegios de root En este caso, el atacante puede elevar sus privilegios a nivel administrativo y atacar otros servidores. El impacto dependerá de la concentración de la empresa objetivo en su Defensa en Profundidad.

- En el archivo principal.php del porque el problema:

Impacto potencial de la falta de contraseña en una base de datos

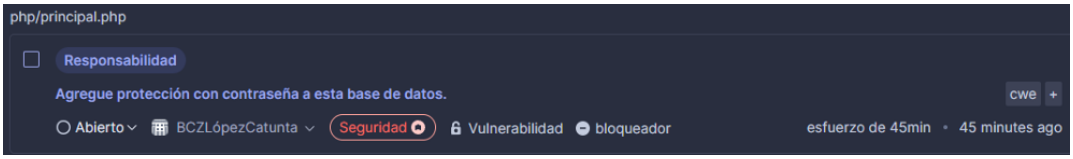
Cuando una base de datos no requiere una contraseña para la autenticación, se abre la posibilidad de acceso y manipulación de los datos almacenados por parte de cualquier persona. Explotar esta vulnerabilidad puede tener varios impactos potenciales:

a. Acceso no autorizado a datos confidenciales: La ausencia de una contraseña para la autenticación permite que personas no autorizadas

obtengan acceso a datos confidenciales, como información de identificación personal, registros financieros y propiedad intelectual. Esto puede provocar robo de identidad, pérdidas financieras y daños a la reputación.

b. Compromiso de la integridad del sistema: Personas no autorizadas pueden obtener acceso ilimitado a la base de datos, lo que compromete la integridad del sistema. Los atacantes pueden inyectar código malicioso, alterar configuraciones o manipular datos, lo que puede provocar fallos en el funcionamiento del sistema y exposición a mayores riesgos de seguridad.

c. Modificaciones o eliminaciones no deseadas: La falta de una contraseña para el acceso a la base de datos permite que cualquier persona realice modificaciones o eliminaciones de los datos almacenados. Esto plantea un riesgo significativo, ya que los cambios no autorizados pueden provocar daños en los datos, pérdida de información crítica o la introducción de contenido malicioso.



- En el archivo categoria_producto.php del porque el problema:

Impacto Potencial de las Inyecciones de Bases de Datos

Las inyecciones de bases de datos, como las inyecciones SQL, representan una seria amenaza para la seguridad de las aplicaciones web. Cuando una aplicación es vulnerable a este tipo de ataques, los atacantes pueden manipular consultas de base de datos para ejecutar comandos maliciosos y comprometer la integridad de los datos. Esta vulnerabilidad expone a la aplicación a una serie de riesgos, que pueden tener impactos significativos en la seguridad y el funcionamiento de las bases de datos afectadas.

Suplantación de Identidad y Manipulación de Datos

La explotación de una consulta maliciosa puede conducir a una escalada de privilegios o a la fuga de datos sensibles almacenados en las bases de datos. Esta amenaza representa uno de los impactos más extendidos, ya que los atacantes pueden obtener acceso no autorizado a información confidencial, lo que a su vez puede resultar en la suplantación de identidad y manipulación de datos.

Eliminación de Datos y Denegación de Servicio

Los atacantes pueden utilizar consultas maliciosas para eliminar datos de las bases de datos afectadas, lo que puede tener consecuencias

devastadoras si la organización no cuenta con un plan de recuperación ante desastres. Esta amenaza es particularmente insidiosa, ya que puede resultar en la pérdida irreversible de información crítica y en la interrupción del servicio.

Encadenamiento de Inyecciones de Bases de Datos con Otras Vulnerabilidades

Los atacantes que explotan las inyecciones SQL suelen buscar maximizar sus ganancias mediante el aprovechamiento de otras vulnerabilidades. Esto puede incluir la filtración de secretos almacenados sin cifrar en las bases de datos, comprometiendo otros componentes del sistema, o incluso la ejecución remota de código si el sistema operativo del servidor o los permisos de la base de datos están mal configurados.

- En el archivo product_img.php del porque el problema:

Las inyecciones de bases de datos, como las inyecciones SQL, representan una seria amenaza para la seguridad de las aplicaciones web. Cuando una aplicación es vulnerable a este tipo de ataques, los atacantes pueden manipular consultas de base de datos para ejecutar comandos maliciosos y comprometer la integridad de los datos. Esta vulnerabilidad expone a la aplicación a una serie de riesgos, que pueden tener impactos significativos en la seguridad y el funcionamiento de las bases de datos afectadas.

Suplantación de Identidad y Manipulación de Datos

La explotación de una consulta maliciosa puede conducir a una escalada de privilegios o a la fuga de datos sensibles almacenados en las bases de datos. Esta amenaza representa uno de los impactos más extendidos, ya que los atacantes pueden obtener acceso no autorizado a información confidencial, lo que a su vez puede resultar en la suplantación de identidad y manipulación de datos.

Eliminación de Datos y Denegación de Servicio

Los atacantes pueden utilizar consultas maliciosas para eliminar datos de las bases de datos afectadas, lo que puede tener consecuencias devastadoras si la organización no cuenta con un plan de recuperación ante desastres. Esta amenaza es particularmente insidiosa, ya que puede resultar en la pérdida irreversible de información crítica y en la interrupción del servicio.

Encadenamiento de Inyecciones de Bases de Datos con Otras Vulnerabilidades

Los atacantes que explotan las inyecciones SQL suelen buscar maximizar sus ganancias mediante el aprovechamiento de otras vulnerabilidades. Esto puede incluir la filtración de secretos almacenados sin cifrar en las bases de datos, comprometiendo otros componentes del sistema, o incluso la ejecución remota de código si el sistema operativo del servidor o los permisos de la base de datos están mal configurados.

- Método y Técnicas Usadas en el SonarQube
  - SonarSource

a)El problema index.php:

Para evitar la vulnerabilidad de inyección en las expresiones de inclusión en PHP, es fundamental validar la entrada del usuario antes de utilizarla en una expresión de inclusión. Esto se logra creando una lista de archivos autorizados y seguros que la aplicación puede cargar con expresiones de inclusión. Si la entrada del usuario no coincide con esta lista, debe ser rechazada para garantizar la seguridad.

En el código proporcionado, se define una lista de archivos permitidos y seguros que la aplicación puede incluir. Luego, se verifica si la vista solicitada está en la lista permitida y si existe como archivo. Si cumple con estas condiciones, se incluye el archivo correspondiente; de lo contrario, se carga una página de error.

Es importante destacar que esta validación debe realizarse en el lado del servidor para garantizar su efectividad. Al implementar esta técnica, se reduce significativamente el riesgo de ataques de inyección en las expresiones de inclusión, lo que contribuye a fortalecer la seguridad de la aplicación.

Solución del Problema:

index.php

```php
<?php
define('VISTAS_PATH', "./vistas/");

$INCLUDE_ALLOW_LIST = [
    'casa.php',
    'tablero.php',
    'perfil.php',
    'configuración.php'
];

// Función para verificar la sesión activa
function verificarSesionActiva() {
    if (empty($_SESSION['identificación']) || empty($_SESSION['us
        require_once VISTAS_PATH . 'cerrar sesión.php';
        exit();
    }
}

// Obtener la vista solicitada
$vista = $_GET["vista"] ?? '';

// Verificar si la vista solicitada está en la lista permitida y
if ($vista && in_array($vista, $INCLUDE_ALLOW_LIST) && is_file(VI
    require_once VISTAS_PATH . $vista . ".php";
} else {
    // Si la vista no está permitida o no existe, cargar la págin
    require_once VISTAS_PATH . '404.php';
}

// Iniciar sesión si la vista solicitada es 'acceso'; de lo contr
if ($vista === 'acceso') {
    require_once VISTAS_PATH . 'iniciar sesión.php';
} else {
    verificarSesionActiva();
}

// Incluir la barra de navegación
require_once "./inc/navbar.php";
?>
```

Se utilizan las siguientes metodologías y técnicas:

- *Lista de Permitidos: Se define una lista de archivos permitidos y seguros que la aplicación puede incluir. Esto se logra mediante la variable $INCLUDE_ALLOW_LIST, que contiene los nombres de los archivos autorizados, como 'casa.php', 'tablero.php', 'perfil.php' y 'configuración.php'.*
- *Validación de Sesión Activa: Se implementa una función llamada verificarSesionActiva() que verifica si la sesión del usuario está activa. Si la identificación del usuario o el usuario mismo están vacíos, se requiere el archivo 'cerrar sesión.php' desde la ruta de vistas y se finaliza la ejecución del script.*
- *Validación de Vista Solicitada: Se verifica si la vista solicitada por el usuario está en la lista permitida y si existe como archivo. Esto se realiza mediante la variable $vista, que obtiene el valor del parámetro "vista" en la URL. Si la vista está en la lista permitida y es un archivo válido, se incluye el archivo correspondiente; de lo contrario, se carga la página '404.php'.*
- *Inclusión Condicional de Archivos: Dependiendo de la vista solicitada, se realizan inclusiones condicionales de archivos. Si la vista solicitada es 'acceso', se requiere el archivo 'iniciar sesión.php'; de lo contrario, se verifica la sesión activa llamando a la función verificarSesionActiva().*
- *Inclusión de Barra de Navegación: Finalmente, se incluye la barra de navegación desde el archivo "./inc/navbar.php".*

b) El problema main.php:

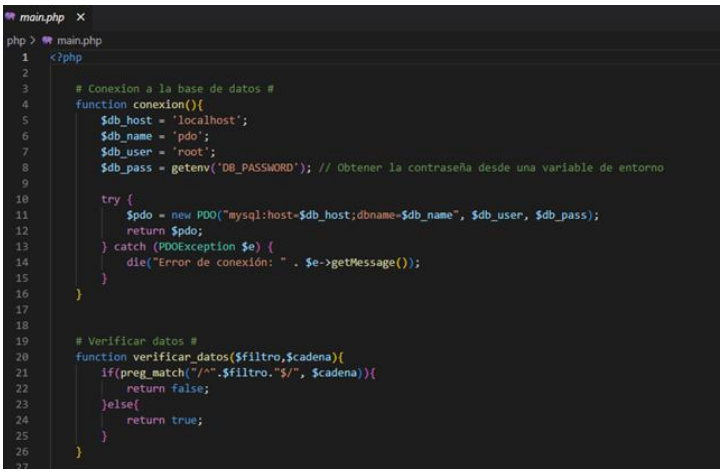
El código presenta una vulnerabilidad al utilizar una contraseña vacía para conectarse a una base de datos MySQL. Para solucionar esta vulnerabilidad, es recomendable utilizar una contraseña segura recuperada de una variable de entorno, como MYSQL_SECURE_PASSWORD, que se establece durante la implementación y debe ser única y robusta para cada base de datos.

Es importante evitar el uso de contraseñas codificadas en el código, ya que esto puede exponer la seguridad de la aplicación a riesgos significativos. Las contraseñas codificadas pueden ser descubiertas fácilmente por desarrolladores o atacantes, lo que podría resultar en acceso no autorizado a la base de datos y posibles violaciones de datos. Además, el uso de contraseñas codificadas dificulta la flexibilidad para cambiar las contraseñas sin modificar el código, lo que puede generar problemas de control de versiones y representar un riesgo para la seguridad.

Para mitigar estos riesgos, se recomienda utilizar métodos seguros para almacenar y recuperar contraseñas, como el uso de variables de entorno, archivos de configuración o sistemas seguros de administración de claves. Estas prácticas permiten una mayor seguridad, flexibilidad y separación

de información confidencial del código base, lo que contribuye a la protección de los datos y a la prevención de posibles vulnerabilidades.

Solución del Problema:



Se utilizan las siguientes metodologías y técnicas:

- Uso de Variables de Entorno: Se utiliza la función getenv() para recuperar la contraseña de la base de datos desde una variable de entorno llamada 'DB_PASSWORD'. Esto asegura que la contraseña no esté codificada en el código y se obtenga de una fuente externa, lo que mejora la seguridad.

- Conexión Segura a la Base de Datos: Al utilizar la contraseña recuperada de la variable de entorno para establecer la conexión a la base de datos, se garantiza que se esté utilizando una contraseña segura al conectarse a la base de datos MySQL. Esto es fundamental para proteger la integridad de los datos y prevenir posibles violaciones de seguridad.

- Manejo de Excepciones: Se utiliza un bloque try-catch para manejar posibles errores de conexión a la base de datos. Esto asegura que cualquier error sea capturado y manejado adecuadamente, lo que contribuye a la robustez y estabilidad del sistema.

c) El problema categoria_producto.php:

El código proporcionado presenta una vulnerabilidad de inyección de SQL, ya que los datos controlados por el usuario se insertan directamente en una cadena de consulta, lo que puede permitir a un atacante realizar consultas maliciosas. Para solucionar esta vulnerabilidad, se recomienda utilizar declaraciones preparadas en lugar de construir consultas SQL directamente a partir de datos controlados por el usuario. La solución compatible implica el uso de declaraciones preparadas, donde los valores de los parámetros se pasan por separado, lo que evita la posibilidad de inyección de SQL. Al utilizar declaraciones preparadas, la lógica de la consulta se compila antes de que se pasen los valores de los parámetros, lo que evita que los datos controlados por el usuario afecten la consulta subyacente.

Solución del Problema:

Se utilizan las siguientes metodologías y técnicas:

- Se utiliza la técnica de declaración preparada para protegerse contra la inyección de SQL. En lugar de construir la consulta SQL directamente a partir de datos controlados por el usuario, se utilizan marcadores de posición en la consulta y luego se enlazan los valores de los parámetros a estos marcadores de posición. Esto evita que los datos controlados por el usuario afecten la lógica de la consulta y previene posibles ataques de inyección de SQL.
  Además, el código utiliza la función isset() para verificar si el parámetro 'category_id' está presente en la URL antes de asignarlo a la variable $categoria_id. Esto ayuda a evitar errores de ejecución si el parámetro no está presente en la URL.

d) El problema product_img.php:

El código proporcionado presenta una vulnerabilidad de inyección de SQL, ya que los datos controlados por el usuario se insertan directamente en una cadena de consulta, lo que puede permitir a un atacante realizar consultas maliciosas. Para solucionar esta vulnerabilidad, se recomienda utilizar declaraciones preparadas en lugar de construir consultas SQL directamente a partir de datos controlados por el usuario.

La solución compatible implica el uso de declaraciones preparadas, donde los valores de los parámetros se pasan por separado, lo que evita la posibilidad de inyección de SQL. Al utilizar declaraciones preparadas, la lógica de la consulta se compila antes de que se pasen los valores de los parámetros, lo que evita que los datos controlados por el usuario afecten la consulta subyacente.

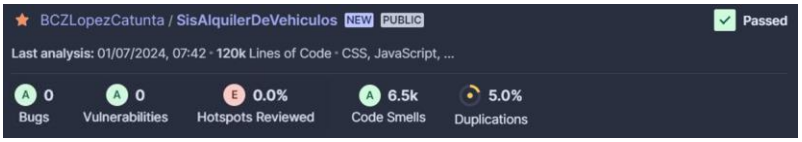
Solución del Problema:

```php
<?php
    include_once "./inc/btn_back.php";
    require_once "./php/main.php";

    $id = (isset($_GET['product_id_up'])) ? $_GET['product_id_up'] : 0;
    /*== Verificando producto ==*/
    $pdo = conexion();
    $statement = $pdo->prepare("SELECT * FROM producto WHERE producto_id = :id");
    $statement->bindParam(':id', $id, PDO::PARAM_INT);
    $statement->execute();
    $result = $statement->fetch(PDO::FETCH_ASSOC);

    if($check_producto->rowCount()>0){
        $datos=$check_producto->fetch();
?>
```

Se utilizan las siguientes metodologías y técnicas:

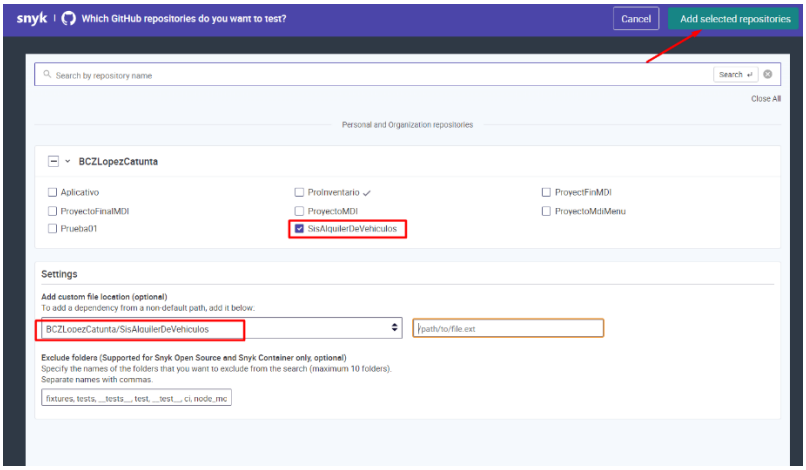
El código proporcionado utiliza la técnica de declaración preparada para protegerse contra la inyección de SQL. En lugar de construir la consulta SQL directamente a partir de datos controlados por el usuario, se utilizan marcadores de posición en la consulta y luego se enlazan los valores de los parámetros a estos marcadores de posición. Esto evita que los datos controlados por el usuario afecten la lógica de la consulta y previene posibles ataques de inyección de SQL. Además, el código utiliza la función isset() para verificar si el parámetro 'product_id_up' está presente en la URL antes de asignarlo a la variable $id. Esto ayuda a evitar errores de ejecución si el parámetro no está presente en la URL.

- Solución del Problema con el SonarQube
  - Existencia de bugs: 0
  - ● Vulnerabilidades: 0.0%
  - ● Hotspots revisados: 0.0%
  - ● Código con olor a bug: 0
  - Codigo Smells: 6.5k
  - ● Duplicaciones: 5.0%
  -

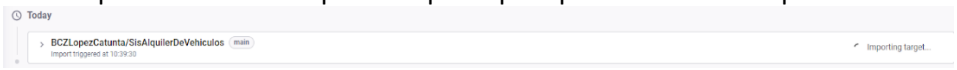


- Snyk

Para subir un proyecto a la plataforma de seguridad de código abierto Snyk. Primero, el usuario busca el repositorio "SisAlquilerDeVehiculos" dentro de su cuenta de GitHub. Luego, selecciona el repositorio y hace clic en "Agregar repositorios seleccionados" para iniciar el proceso de análisis de seguridad.



En este paso observamos que se importa para poder analizar el Aplicativo subido:

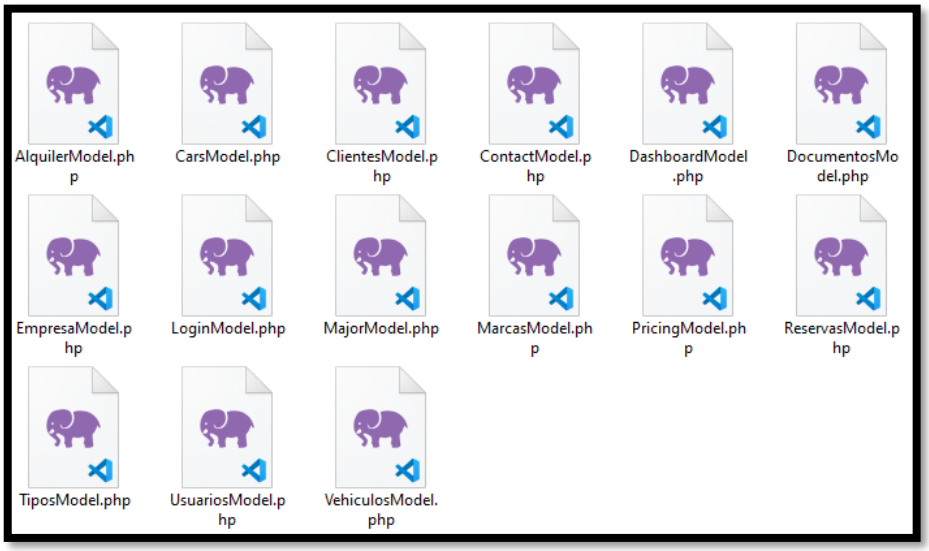


Se han realizado pruebas y se han detectado algunos errores de alta gravedad, incluyendo un error crítico (C), dos errores de alta prioridad (H), y un error de prioridad media (M). También se han encontrado algunos errores de baja prioridad (L) y algunos errores que aún no se han clasificado.

6. Reporte de Cobertura de Pruebas

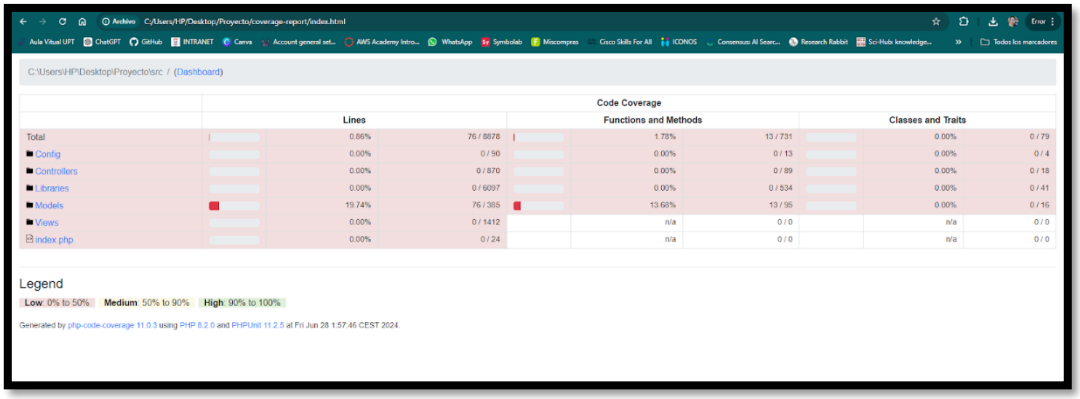
- ***Métodos para hacer las pruebas Tests "Models"***

  Se sugiere usar PHPUnit y Composer para realizar pruebas unitarias de estos modelos. Se puede utilizar Composer para instalar PHPUnit y las dependencias necesarias.



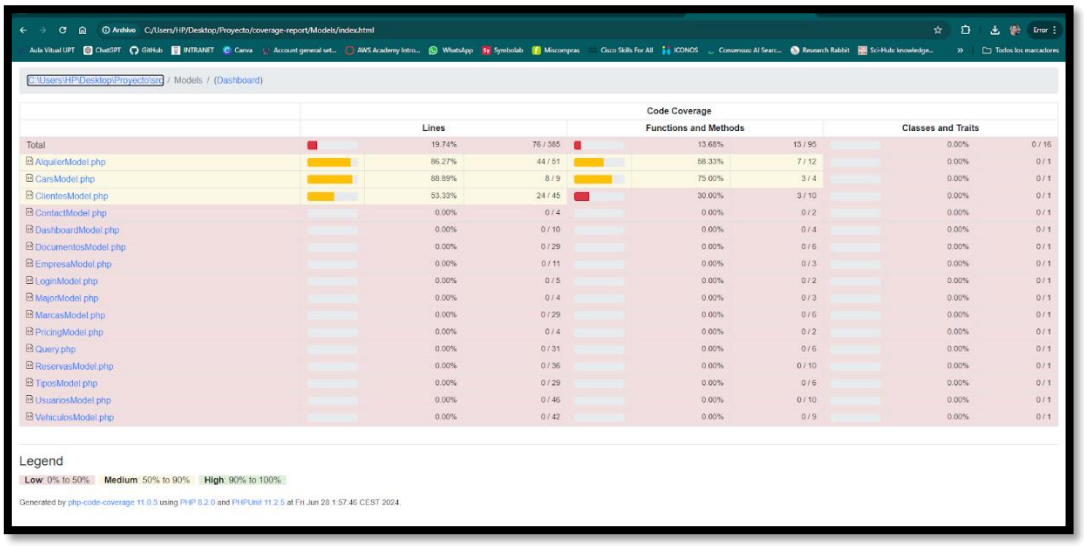
- ***Fase de Inicio para las Pruebas***

  El reporte está dividido en tres secciones: "Líneas", "Funciones y Métodos" y "Clases y Rasgos". Cada sección muestra el porcentaje de cobertura de código para cada elemento de la aplicación, incluyendo el número total de líneas, funciones, métodos y clases. La captura también incluye una leyenda que indica qué colores representan cada rango de cobertura. En general, el reporte indica que la aplicación tiene una baja cobertura de código, con la mayoría de los elementos teniendo una cobertura del 0%. Sin embargo, la sección "Modelos" tiene una cobertura del 19.74%, lo que indica que esta sección de la aplicación ha sido probada más a fondo que las demás.



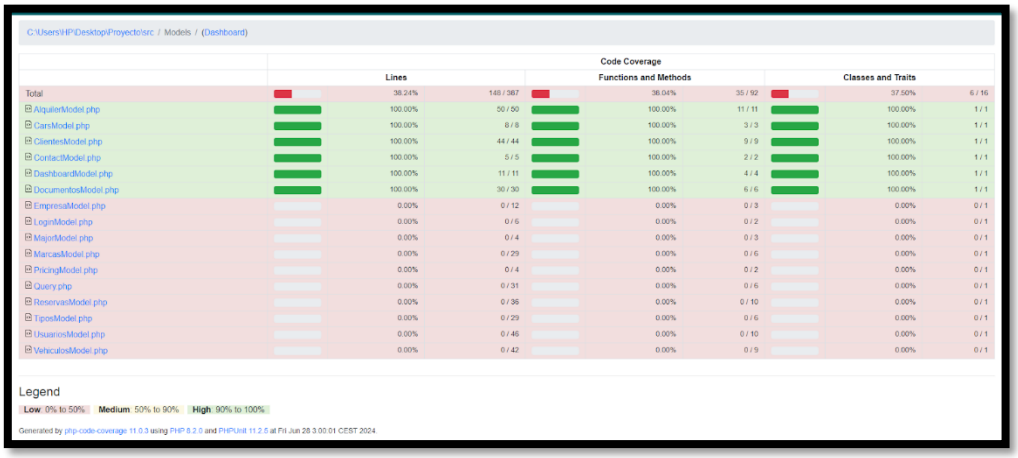
- ***En este apartado podemos observar todos los metodos***

  En la captura se muestra un informe de cobertura de código generado por la herramienta php-code-coverage. El informe muestra el porcentaje de líneas de código, funciones y métodos que están cubiertos por pruebas unitarias. El informe se divide en diferentes secciones, como "Total", "Lines", "Functions and Methods", y "Classes and Traits". Cada sección muestra el nombre del archivo PHP, el porcentaje de cobertura y el número de líneas de código, funciones y métodos que están cubiertos. El informe también muestra una leyenda con los colores que representan los diferentes niveles de cobertura.



- ***En este apartado vemos el progeso de los tests***

  Se puede observar que hay un total de 16 archivos PHP, de los cuales 8 tienen una cobertura de código del 100%, lo que significa que están totalmente cubiertos por pruebas unitarias. Estos archivos son: AlquilerModel.php, CarsModel.php, ClientesModel.php, ContactModel.php, DashboardModel.php, DocumentosModel.php, EmpresaModel.php y LoginModel.php. Los 8 archivos restantes tienen una cobertura de código del 0%.

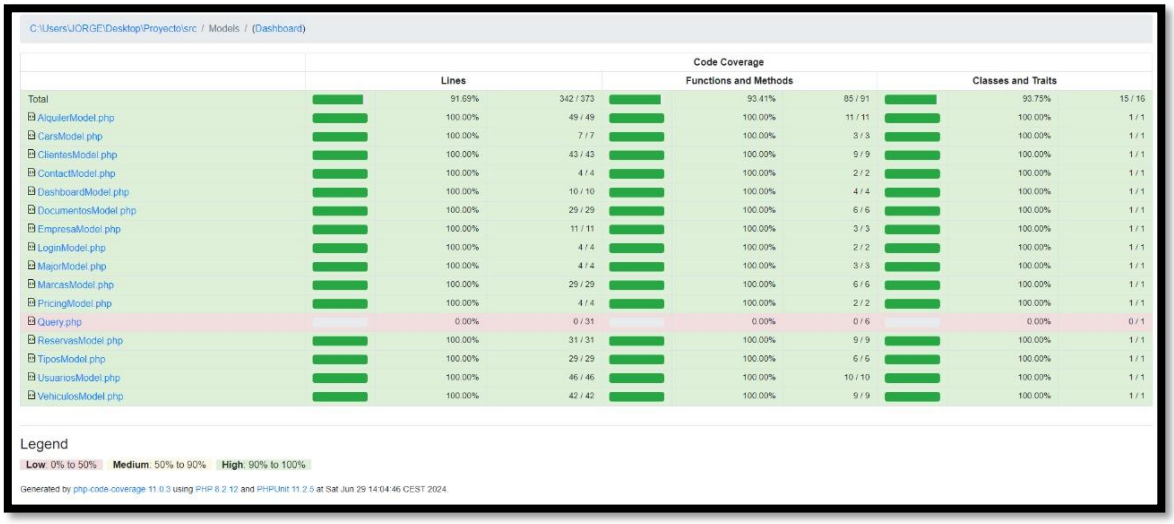


- ***En este otro Punto vemos los siguientes progresos***

La imagen muestra un informe de cobertura de código generado por la herramienta php-code-coverage. El informe muestra la cobertura de código para cada archivo PHP en un proyecto, desglosado por líneas, funciones y métodos, y clases y rasgos. La cobertura de código se muestra como un porcentaje, y el color del indicador de progreso refleja la cobertura: verde para 100%, amarillo para 50% a 90% y rojo para 0% a 50%. El informe también muestra la cantidad de líneas, funciones y métodos, y clases y rasgos que se han probado y la cantidad total.

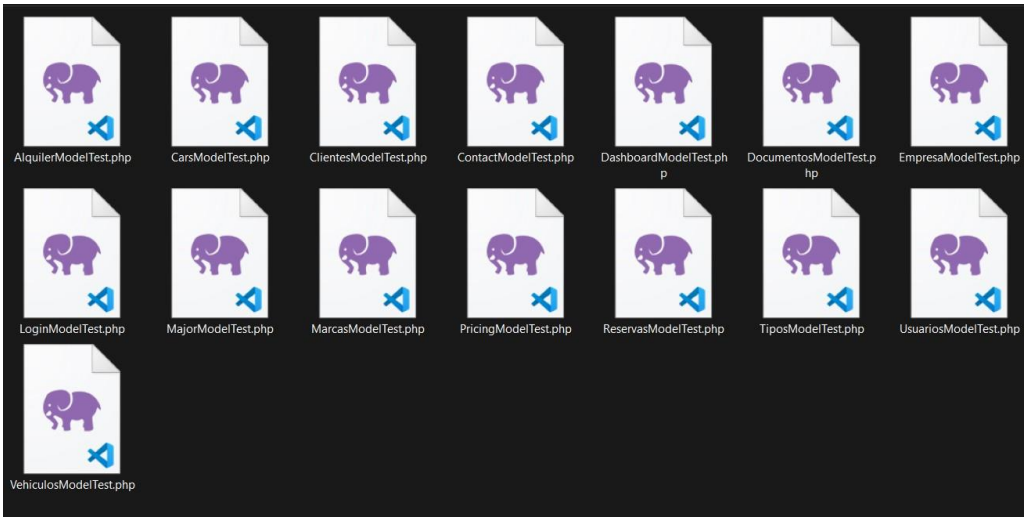
- ***En este otro Punto como parte final podemos los testeos finales de los métodos***

El informe de cobertura de código muestra que la mayoría de los archivos PHP tienen una cobertura del 100%, excepto algunos que tienen una cobertura del 91.69% en líneas, 93.41% en funciones y métodos, y 93.75% en clases y rasgos, lo que indica una alta calidad en la prueba del código.



- ***Pruebas Unitarias***

Se esta utilizando las prácticas de desarrollo de software sólidas, incluyendo pruebas unitarias, con el fin de asegurar la calidad del código de la aplicación de lo cual observamos en estos siguientes archivos.



- ***Herammientas Necesarias para los Test***

Herramienta	Descripción	Requisitos y Pasos a usar
PHPUnit	PHPUnit es un marco de pruebas unitarias para PHP. Se utiliza para escribir y ejecutar pruebas automatizadas que verifican la funcionalidad del código.	1. Instalar PHPUnit: composer require --dev phpunit/phpunit 2. Crear archivos de prueba en la carpeta tests 3. Ejecutar pruebas: vendor/bin/phpunit
Composer	Composer es un gestor de dependencias para PHP. Administra las bibliotecas que tu proyecto necesita y facilita su instalación y actualización.	1. Instalar Composer desde getcomposer.org 2. Crear un archivo composer.json en el proyecto 3. Añadir PHPUnit como dependencia en composer.json 4. Ejecutar composer install para instalar las dependencias

Dentro del proyecto:

Elemento	Descripción
'composer.json'	Archivo para gestionar dependencias del proyecto, asegurando la inclusión de PHPUnit.
'phpunit.xml'	Configuración de PHPUnit para definir cómo y dónde ejecutar las pruebas.
'tests/Unit/'	Directorio donde se alojan todas las pruebas unitarias del proyecto.
'tests/Unit/ModelTest.php'	Ejemplo de archivo de prueba unitario para un modelo específico, como UserTest.php.
'vendor/'	Carpeta generada por Composer que contiene todas las dependencias instaladas, incluido PHPUnit.
'vendor/bin/phpunit'	Comando para ejecutar PHPUnit y realizar las pruebas unitarias definidas en tests/Unit/.

- Archivos para el Test:

- Test de AlquilerModel.php

Titulo del Test	Descripción
testGetAlquiler	Verifica que el método getAlquiler devuelve los datos correctos cuando se llama al método selectAll.

Codigo:

```
public function testGetAlquiler()
    {
        // Preparar los datos esperados
        $mockedData = [
            ['id' => 1, 'nombre' => 'Cliente A', 'placa' => 'ABC123', 'modelo' => 'Modelo X', 'documento' => 'Documento A', 'tipo' => 'Tipo A'],
            ['id' => 2, 'nombre' => 'Cliente B', 'placa' => 'XYZ789', 'modelo' => 'Modelo Y', 'documento' => 'Documento B', 'tipo' => 'Tipo B'],
        ];

        // Configurar el comportamiento esperado del mock de Query para el método selectAll
        $this->mockQuery->shouldReceive('selectAll')
            ->once()
            ->andReturn($mockedData);

        // Instanciar el modelo y llamar al método que queremos probar
        $alquilerModel = new AlquilerModel($this->mockQuery);
        $result = $alquilerModel->getAlquiler();
        // Asegurar que el resultado coincide con los datos esperados
        $this->assertEquals($mockedData, $result);
    }
```

testGetEmpresa	Verifica que el método getEmpresa devuelve los datos correctos cuando se llama al método select con una consulta específica.

Codigo:

```
public function testGetEmpresa()
    {
        // Datos simulados que se esperan retornar del método select
        $mockedData = [
            ['id' => 1, 'nombre' => 'Empresa A', 'direccion' => 'Calle Principal', 'telefono'
=> '123456789'],
        ];

        // Configurar el mock para el método select en la consulta de empresa
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT * FROM configuracion")
            ->andReturn($mockedData);

        // Instanciar AlquilerModel y llamar al método getEmpresa
        $alquilerModel = new AlquilerModel($this->mockQuery);
        $result = $alquilerModel->getEmpresa();

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testRegistrarAlquilerSuccess	Verifica que el método registrarAlquiler registra un alquiler correctamente cuando no hay conflictos y la inserción es exitosa.

Codigo:

```
public function testRegistrarAlquilerSuccess()
    {
        // Preparar los datos para el registro de alquiler
        $cantidad = 1;
        $precios = 'tipo';
        $monto = 5;
        $abono = 15;
        $fecha = '2022-05-10';
        $fecha_devolucion = '2022-05-10';
        $observacion = 'si';
        $id_cli = 1;
        $id_veh = 1;
        $documento = 5;
```

```
            // Configurar el comportamiento esperado del mock de Query para simular que no existe
        un registro previo
            $this->mockQuery->shouldReceive('select')
                ->once()
                ->andReturnNull();

            // Configurar el comportamiento esperado del mock de Query para el método insertar
            // Simulamos que el insertar devuelve 1 para simular una inserción exitosa
            $this->mockQuery->shouldReceive('insertar')
                ->once()
                ->andReturn(1);

            // Instanciar el modelo y llamar al método que queremos probar
            $alquilerModel = new AlquilerModel($this->mockQuery);
              $result = $alquilerModel->registrarAlquiler($cantidad, $precios, $monto, $abono,
        $fecha, $fecha_devolucion, $observacion, $id_cli, $id_veh, $documento);

            // Asegurar que el resultado coincide con el valor retornado por insertar (1 en este
        caso)
            $this->assertEquals(1, $result);
        }
```

testRegistrarAlquilerError	Verifica que el método registrarAlquiler devuelve "error" cuando la inserción falla.

Codigo:

```
    public function testRegistrarAlquilerError()
        {
            // Preparar los datos para el registro de alquiler
            $cantidad = 1;
            $precios = 'tipo';
            $monto = 5;
            $abono = 15;
            $fecha = '2022-05-10';
            $fecha_devolucion = '2022-05-10';
            $observacion = 'si';
            $id_cli = 1;
            $id_veh = 1;
            $documento = 5;

            // Configurar el comportamiento esperado del mock de Query para simular que no existe
        un registro previo
            $this->mockQuery->shouldReceive('select')
```

```
                ->once()
                ->andReturnNull();

            // Configurar el comportamiento esperado del mock de Query para el método insertar
            // Simulamos que el insertar devuelve 0 para simular un fallo en la inserción
            $this->mockQuery->shouldReceive('insertar')
                ->once()
                ->andReturn(0);

            // Instanciar el modelo y llamar al método que queremos probar
            $alquilerModel = new AlquilerModel($this->mockQuery);
             $result = $alquilerModel->registrarAlquiler($cantidad, $precios, $monto, $abono,
    $fecha, $fecha_devolucion, $observacion, $id_cli, $id_veh, $documento);

            // Asegurar que el resultado coincide con el string "error"
            $this->assertEquals('error', $result);
        }
```

testRegistrarAlquilerExistente	Verifica que el método registrarAlquiler devuelve "existe" cuando ya existe un alquiler activo para los mismos datos.

Codigo:

```
    public function testRegistrarAlquilerExistente()
        {
            // Datos simulados para el registro de alquiler
            $cantidad = 1;
            $precios = 'Tipo A';
            $monto = 100;
            $abono = 50;
            $fecha = '2024-06-28';
            $fecha_devolucion = '2024-06-30';
            $observacion = 'Alquiler de prueba';
            $id_cli = 1;
            $id_veh = 1;
            $documento = 1;

             // Configurar el mock para el método select en la verificación de existencia de
    alquiler
            $this->mockQuery->shouldReceive('select')
                 ->with("SELECT * FROM alquiler WHERE id_cliente = $id_cli AND id_vehiculo =
    $id_veh AND id_doc = $documento AND estado = 1")
                ->andReturn([['id' => 1]]); // Alquiler existe
```

```
        // Instanciar AlquilerModel y llamar al método que queremos probar
        $alquilerModel = new AlquilerModel($this->mockQuery);
          $result = $alquilerModel->registrarAlquiler($cantidad, $precios, $monto, $abono,
$fecha, $fecha_devolucion, $observacion, $id_cli, $id_veh, $documento);

        // Verificar que se retorna el resultado esperado en caso de existencia de alquiler
('existe')
        $this->assertEquals('existe', $result);
    }
```

testGetDoc	Verifica que el método getDoc devuelve los datos correctos cuando se llama al método selectAll.

Codigo:

```
    public function testGetDoc()
    {
        // Preparar los datos esperados
        $mockedData = [
            ['id' => 1, 'nombre' => 'Documento A', 'estado' => 1],
            ['id' => 2, 'nombre' => 'Documento B', 'estado' => 1],
        ];

        // Configurar el comportamiento esperado del mock de Query para el método selectAll
        $this->mockQuery->shouldReceive('selectAll')
            ->once()
            ->andReturn($mockedData);

        // Instanciar el modelo y llamar al método que queremos probar
        $alquilerModel = new AlquilerModel($this->mockQuery);
        $result = $alquilerModel->getDoc();

        // Asegurar que el resultado coincide con los datos esperados
        $this->assertEquals($mockedData, $result);
    }
```

testGetVehiculos	Verifica que el método getVehiculos devuelve los datos correctos cuando se llama al método selectAll.

Codigo:

```
    public function testGetVehiculos()
    {
        // Preparar los datos esperados
        $mockedData = [
            ['id' => 1, 'placa' => 'ABC123', 'id_tipo' => 1, 'id_marca' => 1, 'estado' =>
1],
```

```
                ['id' => 2, 'placa' => 'XYZ789', 'id_tipo' => 2, 'id_marca' => 2, 'estado' =>
    1],
            ];

            // Configurar el comportamiento esperado del mock de Query para el método selectAll
            $this->mockQuery->shouldReceive('selectAll')
                ->once()
                ->andReturn($mockedData);

            // Instanciar el modelo y llamar al método que queremos probar
            $alquilerModel = new AlquilerModel($this->mockQuery);
            $result = $alquilerModel->getVehiculos();

            // Asegurar que el resultado coincide con los datos esperados
            $this->assertEquals($mockedData, $result);
        }
```

testGetVehiculo	Verifica que el método getVehiculo devuelve los datos correctos cuando se llama al método select con el ID del vehículo.

Codigo:

```
    public function testGetVehiculo()
        {
            // Preparar los datos esperados
             $mockedData = ['id' => 1, 'placa' => 'ABC123', 'id_tipo' => 1, 'id_marca' => 1,
    'estado' => 1];

            // Configurar el comportamiento esperado del mock de Query para el método select
            $this->mockQuery->shouldReceive('select')
                ->once()
                ->andReturn($mockedData);

            // ID del vehículo a consultar
            $vehiculoId = 1;

            // Instanciar el modelo y llamar al método que queremos probar
            $alquilerModel = new AlquilerModel($this->mockQuery);
            $result = $alquilerModel->getVehiculo($vehiculoId);

            // Asegurar que el resultado coincide con los datos esperados
            $this->assertEquals($mockedData, $result);
        }
```

testVerify	Verifica que el método verify devuelve los datos correctos de una reserva cuando hay superposición de fechas.

Codigo:

```
public function testVerify()
    {
        // Preparar los datos esperados
         $mockedData = ['id' => 1, 'fecha_prestamo' => '2022-05-10', 'fecha_devolucion' =>
'2022-05-15']; // Suponiendo un resultado de reserva superpuesta

        // Configurar el comportamiento esperado del mock de Query para el método select
        $this->mockQuery->shouldReceive('select')
            ->once()
            ->andReturn($mockedData);

        // Parámetros para verificar la reserva
        $desde = '2022-05-12';
        $hasta = '2022-05-14';
        $id_veh = 1;

        // Instanciar el modelo y llamar al método que queremos probar
        $alquilerModel = new AlquilerModel($this->mockQuery);
        $result = $alquilerModel->verify($desde, $hasta, $id_veh);

        // Asegurar que el resultado coincide con los datos esperados
        $this->assertEquals($mockedData, $result);
    }
```

testActualizarVehiculo	Verifica que el método actualizarVehiculo actualiza correctamente el estado de un vehículo y devuelve "ok".

Codigo:

```
public function testActualizarVehiculo()
    {
        // Preparar los datos para la actualización del vehículo
        $estado = 0; // Estado a actualizar
        $vehiculoId = 1; // ID del vehículo

        // Configurar el comportamiento esperado del mock de Query para el método save
        $this->mockQuery->shouldReceive('save')
            ->once()
            ->andReturn(1); // Suponiendo que se actualiza correctamente
```

```
            // Instanciar el modelo y llamar al método que queremos probar
            $alquilerModel = new AlquilerModel($this->mockQuery);
            $result = $alquilerModel->actualizarVehiculo($estado, $vehiculoId);

            // Asegurar que el resultado coincide con 'ok'
            $this->assertEquals('ok', $result);
        }
```

testActualizarVehiculoError	Verifica que el método actualizarVehiculo devuelve "error" cuando la actualización del estado de un vehículo falla.

Codigo:

```
    public function testActualizarVehiculoError()
        {
            // Datos simulados para la actualización del vehículo
            $estado = 1;
            $id = 1;

            // Configurar el mock para el método save en la actualización de vehículo
            $this->mockQuery->shouldReceive('save')
                ->once()
                ->andReturn(0); // Simular que hubo un error en la actualización

            // Instanciar AlquilerModel y llamar al método que queremos probar
            $alquilerModel = new AlquilerModel($this->mockQuery);
            $result = $alquilerModel->actualizarVehiculo($estado, $id);

           // Verificar que se retorna el resultado esperado en caso de error en la actualización
    ('error')
            $this->assertEquals('error', $result);
        }
```

testProcesarEntrega	Verifica que el método procesarEntrega actualiza correctamente el estado de una entrega de alquiler y devuelve "ok".

Codigo:

```
    public function testProcesarEntrega()
        {
            // Preparar los datos para procesar la entrega
            $estado = 1; // Estado a actualizar
            $alquilerId = 1; // ID del alquiler

            // Configurar el comportamiento esperado del mock de Query para el método save
```

```
            $this->mockQuery->shouldReceive('save')
                ->once()
                ->andReturn(1); // Suponiendo que se actualiza correctamente

            // Instanciar el modelo y llamar al método que queremos probar
            $alquilerModel = new AlquilerModel($this->mockQuery);
            $result = $alquilerModel->procesarEntrega($estado, $alquilerId);

            // Asegurar que el resultado coincide con 'ok'
            $this->assertEquals('ok', $result);
        }
```

testProcesarEntregaError	Verifica que el método procesarEntrega devuelve "error" cuando la actualización del estado de una entrega de alquiler falla.

Codigo:

```
    public function testProcesarEntregaError()
        {
            // Datos simulados para procesar la entrega
            $estado = 2; // Nuevo estado
            $id = 1; // ID del registro de alquiler a procesar

            // Configurar el mock para el método save en la actualización del estado de alquiler
            $this->mockQuery->shouldReceive('save')
                ->once()
                ->andReturn(0); // Simular que hubo un error en la actualización

            // Instanciar AlquilerModel y llamar al método que queremos probar
            $alquilerModel = new AlquilerModel($this->mockQuery);
            $result = $alquilerModel->procesarEntrega($estado, $id);

           // Verificar que se retorna el resultado esperado en caso de error en la actualización
    ('error')
            $this->assertEquals('error', $result);
        }
```

testVerPrestamo	Verifica que el método verPrestamo devuelve los datos correctos de un prestamo cuando se llama al método select con el ID del préstamo.

Codigo:

```
    public function testVerPrestamo()
        {
            // Preparar los datos esperados
```

```
            $mockedData = [
                'id' => 1,
                'fecha_prestamo' => '2022-05-10',
                'fecha_devolucion' => '2022-05-15',
                'dni' => '12345678',
                'nombre' => 'Cliente A',
                'telefono' => '123456789',
                'direccion' => 'Calle A',
                'placa' => 'ABC123',
                'modelo' => 'Modelo X',
                'documento' => 'Documento A',
                'tipo' => 'Tipo A',
            ];
```

- Test de CarsModel.php

Titulo del Test	Descripción
testGetVehiculos	Este test verifica que el método getVehiculos del modelo CarsModel devuelve los vehículos activos correctamente, simulando el comportamiento de la consulta SQL esperada.

Codigo:

```
    public function testGetVehiculos()
        {
            // Preparar los datos esperados
            $mockedData = [
                ['id' => 1, 'placa' => 'ABC123', 'id_tipo' => 1, 'id_marca' => 1, 'estado' => 1],
                ['id' => 2, 'placa' => 'XYZ789', 'id_tipo' => 2, 'id_marca' => 2, 'estado' => 1],
            ];

            // Estado para filtrar vehículos activos (estado = 1)
            $estado = 1;

            // Configurar el comportamiento esperado del mock de Query para el método selectAll
            $this->mockQuery->shouldReceive('selectAll')
                ->once()
                 ->with("SELECT v.*, m.marca, t.tipo FROM vehiculos v INNER JOIN marcas m ON v.id_marca = m.id INNER JOIN tipos t ON v.id_tipo = t.id WHERE v.estado != $estado")
                ->andReturn($mockedData);

            // Instanciar el modelo y llamar al método que queremos probar
```

```
        $carsModel = new CarsModel($this->mockQuery);
        $result = $carsModel->getVehiculos($estado);

        // Asegurar que el resultado coincide con los datos esperados
        $this->assertEquals($mockedData, $result);
    }
```

testGetVehiculo	Este test comprueba que el método getVehiculo del modelo CarsModel devuelve la información del vehículo específico solicitado, simulando el comportamiento de la consulta SQL esperada.

Codigo:

```
    public function testGetVehiculo()
        {
            // Preparar los datos esperados
             $mockedData = ['id' => 1, 'placa' => 'ABC123', 'id_tipo' => 1, 'id_marca' => 1,
'estado' => 1];

            // Configurar el comportamiento esperado del mock de Query para el método select
            $this->mockQuery->shouldReceive('select')
                ->once()
                ->andReturn($mockedData);

            // ID del vehículo a consultar
            $vehiculoId = 1;

            // Instanciar el modelo y llamar al método que queremos probar
            $carsModel = new CarsModel($this->mockQuery);
            $result = $carsModel->getVehiculo($vehiculoId);

            // Asegurar que el resultado coincide con los datos esperados
            $this->assertEquals($mockedData, $result);
        }
```

- Test de ClientesModel.php

Titulo del Test	Descripción
testGetClientes	Verifica que el método getClientes del modelo ClientesModel devuelva correctamente los clientes activos.

Codigo:

```
    public function testGetClientes()
        {
```

```
            $mockedData = [
                ['id' => 1, 'nombre' => 'Cliente 1', 'direccion' => 'Dirección 1'],
                ['id' => 2, 'nombre' => 'Cliente 2', 'direccion' => 'Dirección 2'],
            ];

            // Configurar el mock para selectAll en getClientes
            $this->mockQuery->shouldReceive('selectAll')
                ->once()
                ->with("SELECT * FROM clientes WHERE estado = 1")
                ->andReturn($mockedData);

            $clientesModel = new ClientesModel($this->mockQuery);
            $result = $clientesModel->getClientes(1);

            $this->assertEquals($mockedData, $result);
        }
```

testBuscarCliente	Comprueba que el método buscarCliente del modelo ClientesModel retorne los clientes que coincidan con el nombre buscado.

Codigo:

```
    public function testBuscarCliente()
        {
            $mockedData = [
                ['id' => 1, 'nombre' => 'Cliente 1', 'direccion' => 'Dirección 1'],
                ['id' => 2, 'nombre' => 'Cliente 2', 'direccion' => 'Dirección 2'],
            ];

            // Configurar el mock para selectAll en buscarCliente
            $this->mockQuery->shouldReceive('selectAll')
                ->once()
                    ->with("SELECT id, nombre, direccion FROM clientes WHERE nombre LIKE
    '%valor_buscado%' AND estado = 1")
                ->andReturn($mockedData);

            $clientesModel = new ClientesModel($this->mockQuery);
            $result = $clientesModel->buscarCliente('valor_buscado');

            $this->assertEquals($mockedData, $result);
        }
```

testRegistrarCliente_Exito	Confirma que el método registrarCliente del modelo ClientesModel registre un nuevo cliente correctamente cuando no existe previamente.

Codigo:

```
public function testRegistrarCliente_Exito()
    {
         // Configurar el mock para el método select en la verificación de existencia de
cliente
        $this->mockQuery->shouldReceive('select')
            ->andReturn([]); // Cliente no existe

        // Configurar el mock para el método save en el registro de cliente
        $this->mockQuery->shouldReceive('save')
            ->once()
            ->andReturn(1); // Éxito en la inserción

        $nombre = 'Nuevo Cliente';
        $dni = '12345678A';
        $telefono = '987654321';
        $direccion = 'Calle Principal';

        $clientesModel = new ClientesModel($this->mockQuery);
        $result = $clientesModel->registrarCliente($dni, $nombre, $telefono, $direccion);

       $this->assertEquals('ok', $result); // Verificar que se retorna 'ok' en caso de éxito
    }
```

testRegistrarCliente_Error	Verifica que el método registrarCliente del modelo ClientesModel maneje adecuadamente el caso de error al intentar registrar un cliente nuevo.

Codigo:

```
public function testRegistrarCliente_Error()
    {
         // Configurar el mock para el método select en la verificación de existencia de
cliente
        $this->mockQuery->shouldReceive('select')
            ->andReturn([]); // Simulamos que el cliente no existe

        // Configurar el mock para el método save en el registro de cliente
        $this->mockQuery->shouldReceive('save')
            ->once()
            ->andReturn(0); // Simulamos que hubo un error en la inserción

        $nombre = 'Nuevo Cliente';
        $dni = '12345678A';
        $telefono = '987654321';
```

```
            $direccion = 'Calle Principal';

            $clientesModel = new ClientesModel($this->mockQuery);
            $result = $clientesModel->registrarCliente($dni, $nombre, $telefono, $direccion);

             $this->assertEquals('error', $result); // Verificar que se retorna 'error' cuando
    falla la inserción
        }
```

testRegistrarClienteExistente	Comprueba que el método registrarCliente del modelo ClientesModel detecte correctamente cuando se intenta registrar un cliente que ya existe en la base de datos.

Codigo:

```
    public function testRegistrarClienteExistente()
        {
            $nombre = 'Cliente Existente';
            $dni = '12345678A';
            $telefono = '987654321';
            $direccion = 'Calle Principal';

             // Configurar el mock para el método select en la verificación de existencia de
    cliente
            // Simulamos que el cliente ya existe en la base de datos
            $this->mockQuery->shouldReceive('select')
                ->with("SELECT * FROM clientes WHERE nombre = '$nombre'")
                ->andReturn(['id' => 1, 'nombre' => $nombre]);

            $clientesModel = new ClientesModel($this->mockQuery);
            $result = $clientesModel->registrarCliente($dni, $nombre, $telefono, $direccion);

            // Verificar que se retorna 'existe' si el cliente ya existe en la base de datos
            $this->assertEquals('existe', $result);
        }
```

testModificarCliente_success	Asegura que el método modificarCliente del modelo ClientesModel actualice la información de un cliente existente correctamente cuando la operación tiene éxito.

Codigo:

```
    public function testModificarCliente_success()
        {
            $mockedDni = '12345678A';
```

```
            $mockedNombre = 'Nuevo Nombre';
            $mockedTelefono = '987654321';
            $mockedDireccion = 'Calle Principal';
            $mockedId = 1;

            // Configuración del mock para caso de éxito
            $this->mockQuery->shouldReceive('save')
                ->once()
                 ->with("UPDATE clientes SET dni = ?, nombre = ?, telefono = ? ,direccion = ?
    WHERE id = ?",
                    [$mockedDni, $mockedNombre, $mockedTelefono, $mockedDireccion, $mockedId])
                ->andReturn(1);

            $clientesModel = new ClientesModel($this->mockQuery);
                    $result   =   $clientesModel->modificarCliente($mockedDni,   $mockedNombre,
    $mockedTelefono, $mockedDireccion, $mockedId);

            // Verificar que el método devuelve 'modificado' en caso de éxito
            $this->assertEquals('modificado', $result);
        }
```

testModificarCliente_failure	Verifica que el método modificarCliente del modelo ClientesModel maneje adecuadamente el caso de fallo al intentar modificar la información de un cliente existente.

Codigo:

```
    public function testModificarCliente_failure()
        {
            $mockedDni = '12345678A';
            $mockedNombre = 'Nuevo Nombre';
            $mockedTelefono = '987654321';
            $mockedDireccion = 'Calle Principal';
            $mockedId = 1;

            // Configuración del mock para caso de error
            $this->mockQuery->shouldReceive('save')
                ->once()
                 ->with("UPDATE clientes SET dni = ?, nombre = ?, telefono = ? ,direccion = ?
    WHERE id = ?",
                    [$mockedDni, $mockedNombre, $mockedTelefono, $mockedDireccion, $mockedId])
                ->andReturn(0);

            $clientesModel = new ClientesModel($this->mockQuery);
```

```
                $result = $clientesModel->modificarCliente($mockedDni, $mockedNombre,
    $mockedTelefono, $mockedDireccion, $mockedId);

            // Verificar que el método devuelve 'error' en caso de error
            $this->assertEquals('error', $result);
        }
```

testEditarCli	Confirma que el método editarCli del modelo ClientesModel obtenga la información de un cliente específico correctamente.

Codigo:

```
    public function testEditarCli()
        {
            $mockedId = 1;
             $mockedData = ['id' => 1, 'nombre' => 'Cliente Ejemplo', 'direccion' => 'Calle
    Principal'];

            // Configurar el mock para select en editarCli
            $this->mockQuery->shouldReceive('select')
                ->once()
                ->with("SELECT * FROM clientes WHERE id = $mockedId")
                ->andReturn($mockedData);

            $clientesModel = new ClientesModel($this->mockQuery);
            $result = $clientesModel->editarCli($mockedId);

            $this->assertEquals($mockedData, $result);
        }
```

testAccionCli	Asegura que el método accionCli del modelo ClientesModel maneje correctamente la acción de cambiar el estado (activar/desactivar) de un cliente.

Codigo:

```
    public function testAccionCli()
        {
            $mockedEstado = 0;
            $mockedId = 1;

            // Configurar el mock para save en accionCli
            $this->mockQuery->shouldReceive('save')
                ->once()
                ->with("UPDATE clientes SET estado = ? WHERE id = ?", [$mockedEstado, $mockedId])
                ->andReturn(1); // Suponiendo que la actualización tiene éxito
```

```
            $clientesModel = new ClientesModel($this->mockQuery);
            $result = $clientesModel->accionCli($mockedEstado, $mockedId);

            $this->assertEquals(1, $result); // Verificar que el resultado sea 1 (éxito)
        }
```

testModificarPass	Verifica que el método modificarPass del modelo ClientesModel actualice correctamente la contraseña de un cliente.

Codigo:

```
    public function testModificarPass()
        {
            $mockedClave = 'nuevaclave123';
            $mockedId = 1;

            // Configurar el mock para save en modificarPass
            $this->mockQuery->shouldReceive('save')
                ->once()
                ->with("UPDATE clientes SET clave = ? WHERE id = ?", [$mockedClave, $mockedId])
                ->andReturn(1); // Suponiendo que la actualización tiene éxito

            $clientesModel = new ClientesModel($this->mockQuery);
            $result = $clientesModel->modificarPass($mockedClave, $mockedId);

            $this->assertEquals(1, $result); // Verificar que el resultado sea 1 (éxito)
        }
```

testModificarDato_ActualizacionExitosa	Comprueba que el método modificarDato del modelo ClientesModel actualice los datos de un cliente correctamente cuando la operación tiene éxito.

Codigo:

```
    public function testModificarDato_ActualizacionExitosa()
        {
            $nombre = 'Nuevo Nombre';
            $dni = '12345678A';
            $correo = 'nuevo@correo.com';
            $tel = '987654321';
            $dir = 'Calle Principal';
            $id = 1;

            // Configurar el mock para el método save en la modificación de datos del cliente
            $this->mockQuery->shouldReceive('save')
                ->with("UPDATE clientes SET nombre=?, dni=?, correo=?, telefono=?, direccion=?
    WHERE id=?",
```

```
                    [$nombre, $dni, $correo, $tel, $dir, $id])
                ->andReturn(1); // Simulamos que la actualización tiene éxito

            $clientesModel = new ClientesModel($this->mockQuery);
            $result = $clientesModel->modificarDato($nombre, $dni, $correo, $tel, $dir, $id);

            // Verificar que se retorna 1 cuando la actualización tiene éxito
            $this->assertEquals(1, $result);
        }
```

testModificarDato_ActualizacionNoExitosa	Verifica que el método modificarDato del modelo ClientesModel maneje adecuadamente el caso de fallo al intentar actualizar los datos de un cliente.

Codigo:

```
    public function testModificarDato_ActualizacionNoExitosa()
        {
            $nombre = 'Nuevo Nombre';
            $dni = '12345678A';
            $correo = 'nuevo@correo.com';
            $tel = '987654321';
            $dir = 'Calle Principal';
            $id = 1;

            // Configurar el mock para el método save en la modificación de datos del cliente
            $this->mockQuery->shouldReceive('save')
                 ->with("UPDATE clientes SET nombre=?, dni=?, correo=?, telefono=?, direccion=?
    WHERE id=?",
                    [$nombre, $dni, $correo, $tel, $dir, $id])
                ->andReturn(0); // Simulamos que la actualización no tiene éxito

            $clientesModel = new ClientesModel($this->mockQuery);
            $result = $clientesModel->modificarDato($nombre, $dni, $correo, $tel, $dir, $id);

            // Verificar que se retorna 0 cuando la actualización no tiene éxito
            $this->assertEquals(0, $result);
        }
```

- Test de ContactModel.php

Titulo del Test	Descripción
testGetEmpresa	Verifica que el método getEmpresa del modelo ContactModel obtenga correctamente los datos de configuración de la empresa mediante una consulta simulada.

Codigo:

```
public function testGetEmpresa()
    {
        // Datos simulados que se esperan retornar del método select
        $mockedData = [
            ['id' => 1, 'nombre' => 'Empresa A', 'direccion' => 'Calle Principal', 'telefono'
=> '123456789'],
        ];

        // Configurar el mock para el método select en la consulta de empresa
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT * FROM configuracion")
            ->andReturn($mockedData);

        // Instanciar ContactModel con el mock de Query
        $contactModel = new ContactModel($this->mockQuery);

        // Llamar al método getEmpresa y almacenar el resultado
        $result = $contactModel->getEmpresa();

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

- Test de DashboardModel.php

Titulo del Test	Descripción
testGetDatos	Verifica que el método getDatos del modelo DashboardModel obtenga correctamente el número total de usuarios activos mediante una consulta simulada.

Codigo:

```
public function testGetDatos()
    {
        // Datos simulados que se esperan retornar del método select
        $mockedData = [['total' => 10]];

        // Configurar el mock para el método select en la consulta de getDatos
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT COUNT(*) AS total FROM usuarios WHERE estado = 1")
            ->andReturn($mockedData);

        // Instanciar DashboardModel con el mock de Query
        $dashboardModel = new DashboardModel($this->mockQuery);
```

```
        // Llamar al método getDatos y almacenar el resultado
        $result = $dashboardModel->getDatos('usuarios');

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testRentas	Verifica que el método rentas del modelo DashboardModel obtenga correctamente las rentas acumuladas durante un período específico mediante una consulta simulada.

Codigo:

```
    public function testRentas()
    {
        // Datos simulados que se esperan retornar del método select
        $mockedData = [['ene' => 100, 'feb' => 200, 'mar' => 150]];
        // Configurar el mock para el método select en la consulta de rentas
        $this->mockQuery->shouldReceive('select')
            ->with(M::on(function ($sql) {
                // Verificar que el SQL contenga las fechas esperadas
                return strpos($sql, "BETWEEN '2024-01-01' AND '2024-12-31'") !== false;
            }))
            ->andReturn($mockedData);

        // Instanciar DashboardModel con el mock de Query
        $dashboardModel = new DashboardModel($this->mockQuery);

        // Llamar al método rentas y almacenar el resultado
        $result = $dashboardModel->rentas('2024-01-01', '2024-12-31');

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testRentasSemana	Verifica que el método rentasSemana del modelo DashboardModel obtenga correctamente las rentas de la semana actual mediante una consulta simulada.

Codigo:

```
    public function testRentasSemana()
    {
        // Datos simulados que se esperan retornar del método selectAll
        $mockedData = [
            ['fecha' => '2024-06-24', 'total' => 50],
```

```
            ['fecha' => '2024-06-25', 'total' => 100],
            ['fecha' => '2024-06-26', 'total' => 80],
        ];

        // Configurar el mock para el método selectAll en la consulta de rentasSemana
        $this->mockQuery->shouldReceive('selectAll')
            ->with(M::on(function ($sql) {
                // Verificar que el SQL incluya la semana actual
                return strpos($sql, "WEEK(fecha_prestamo) = WEEK(CURDATE())") !== false;
            }))
            ->andReturn($mockedData);

        // Instanciar DashboardModel con el mock de Query
        $dashboardModel = new DashboardModel($this->mockQuery);

        // Llamar al método rentasSemana y almacenar el resultado
        $result = $dashboardModel->rentasSemana();

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

- Test de DocumentosModel.php

Titulo del Test	Descripción
testGetDocumentos	Verifica que el método getDocumentos del modelo DocumentosModel obtenga correctamente todos los documentos activos mediante una consulta simulada.

Codigo:

```
    public function testGetDocumentos()
        {
            // Datos simulados que se esperan retornar del método selectAll
            $mockedData = [
                ['id' => 1, 'documento' => 'Documento A', 'estado' => 1],
                ['id' => 2, 'documento' => 'Documento B', 'estado' => 1],
            ];
            // Configurar el mock para el método selectAll en la consulta de getDocumentos
            $this->mockQuery->shouldReceive('selectAll')
                ->with("SELECT * FROM documentos WHERE estado = 1")
                ->andReturn($mockedData);
            // Instanciar DocumentosModel con el mock de Query
            $documentosModel = new DocumentosModel($this->mockQuery);
```

```
            // Llamar al método getDocumentos y almacenar el resultado
            $result = $documentosModel->getDocumentos(1);

            // Verificar que el resultado retornado sea el mismo que $mockedData
            $this->assertEquals($mockedData, $result);
        }
```

testRegistrarDoc	Verifica que el método registrarDoc del modelo DocumentosModel registre un nuevo documento correctamente mediante consultas simuladas de verificación e inserción.

Codigo:

```
    public function testRegistrarDoc()
        {
            // Documento a registrar
            $documento = 'Nuevo Documento';

            // Configurar el mock para el método select en la consulta de verificar existencia
            $this->mockQuery->shouldReceive('select')
                ->with("SELECT * FROM documentos WHERE documento = '$documento'")
                ->andReturn([]); // Simular que el documento no existe

            // Configurar el mock para el método save en la consulta de inserción
            $this->mockQuery->shouldReceive('save')
                ->once()
                ->andReturn(1); // Éxito en la inserción

            // Instanciar DocumentosModel con el mock de Query
            $documentosModel = new DocumentosModel($this->mockQuery);

            // Llamar al método registrarDoc y almacenar el resultado
            $result = $documentosModel->registrarDoc($documento);

            // Verificar que el resultado sea "ok" (éxito en la inserción)
            $this->assertEquals('ok', $result);
        }
```

testRegistrarDoc_Error	Verifica que el método registrarDoc del modelo DocumentosModel maneje correctamente el escenario de error durante la inserción de un nuevo documento.

Codigo:

```
    public function testRegistrarDoc_Error()
        {
```

```
            // Configurar el mock para el método select en la verificación de existencia de
    documento
            $this->mockQuery->shouldReceive('select')
                ->andReturn([]); // Documento no existe

            // Configurar el mock para el método save en el registro de documento
            $this->mockQuery->shouldReceive('save')
                ->once()
                ->andReturn(0); // Error en la inserción

            $documento = '12345678A';

            $documentosModel = new DocumentosModel($this->mockQuery);
            $result = $documentosModel->registrarDoc($documento);

             $this->assertEquals('error', $result); // Verificar que se retorna 'error' cuando
    falla la inserción
        }
```

testRegistrarDocDocumentoExistente	Verifica que el método registrarDoc del modelo DocumentosModel maneje correctamente el escenario donde se intenta registrar un documento que ya existe.

Codigo:

```
    public function testRegistrarDocDocumentoExistente()
        {
            // Documento que ya existe en la base de datos
            $documentoExistente = 'Documento A';

            // Configurar el mock para el método select en la consulta de verificar existencia
            $this->mockQuery->shouldReceive('select')
                ->with("SELECT * FROM documentos WHERE documento = '$documentoExistente'")
                ->andReturn(['id' => 1, 'documento' => $documentoExistente, 'estado' => 1]); //
    Simular que el documento ya existe

            // Instanciar DocumentosModel con el mock de Query
            $documentosModel = new DocumentosModel($this->mockQuery);

            // Llamar al método registrarDoc con el documento existente y almacenar el resultado
            $result = $documentosModel->registrarDoc($documentoExistente);

            // Verificar que el resultado sea "existe"
            $this->assertEquals('existe', $result);
```

```
    }
```

testModificarDoc	Verifica que el método modificarDoc del modelo DocumentosModel modifique un documento existente correctamente mediante una consulta simulada.

Codigo:

```
    public function testModificarDoc()
        {
            // Documento modificado y su ID
            $documento = 'Documento Modificado';
            $id = 1;

            // Configurar el mock para el método save en la consulta de actualización
            $this->mockQuery->shouldReceive('save')
                ->with("UPDATE documentos SET documento = ? WHERE id = ?", [$documento, $id])
                ->once()
                ->andReturn(1); // Éxito en la modificación

            // Instanciar DocumentosModel con el mock de Query
            $documentosModel = new DocumentosModel($this->mockQuery);

            // Llamar al método modificarDoc y almacenar el resultado
            $result = $documentosModel->modificarDoc($documento, $id);

            // Verificar que el resultado sea "modificado" (éxito en la modificación)
            $this->assertEquals('modificado', $result);
        }
```

testModificarDocError	Verifica que el método modificarDoc del modelo DocumentosModel maneje correctamente el escenario de error durante la modificación de un documento.

Codigo:

```
    public function testModificarDocError()
        {
            // Documento y ID para la modificación
            $documento = 'Documento Modificado';
            $id = 1; // ID existente en la base de datos

            // Configurar el mock para el método save en la actualización del documento
            $this->mockQuery->shouldReceive('save')
                ->once() // Esperamos una sola llamada al método save
                ->with("UPDATE documentos SET documento = ? WHERE id = ?", [$documento, $id])
                ->andReturn(0); // Simular que la actualización del documento falló
```

```
        // Instanciar DocumentosModel con el mock de Query
        $documentosModel = new DocumentosModel($this->mockQuery);
        // Llamar al método modificarDoc y almacenar el resultado
        $result = $documentosModel->modificarDoc($documento, $id);
        // Verificar que el resultado sea "error"
        $this->assertEquals('error', $result);
    }
```

testEditarDoc	Verifica que el método editarDoc del modelo DocumentosModel obtenga correctamente los datos de un documento específico mediante una consulta simulada.

Codigo:

```
    public function testEditarDoc()
    {
        // ID del documento a editar
        $id = 1;

        // Datos simulados que se esperan retornar del método select
        $mockedData = ['id' => 1, 'documento' => 'Documento A', 'estado' => 1];

        // Configurar el mock para el método select en la consulta de editarDoc
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT * FROM documentos WHERE id = $id")
            ->andReturn($mockedData);

        // Instanciar DocumentosModel con el mock de Query
        $documentosModel = new DocumentosModel($this->mockQuery);

        // Llamar al método editarDoc y almacenar el resultado
        $result = $documentosModel->editarDoc($id);

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testAccionDoc	Verifica que el método accionDoc del modelo DocumentosModel modifique el estado de un documento correctamente mediante una consulta simulada.

Codigo:

```
    public function testAccionDoc()
    {
        // Estado y ID del documento a modificar
        $estado = 0;
```

```
            $id = 1;

            // Configurar el mock para el método save en la consulta de accionDoc
            $this->mockQuery->shouldReceive('save')
                ->with("UPDATE documentos SET estado = ? WHERE id = ?", [$estado, $id])
                ->once()
                ->andReturn(1); // Éxito en la actualización del estado
            // Instanciar DocumentosModel con el mock de Query
            $documentosModel = new DocumentosModel($this->mockQuery);
            // Llamar al método accionDoc y almacenar el resultado
            $result = $documentosModel->accionDoc($estado, $id);

            // Verificar que el resultado retornado sea 1 (éxito en la actualización del estado)
            $this->assertEquals(1, $result);
        }
```

- Test de EmpresaModel.php

Titulo del Test	Descripción
testGetEmpresa	Verifica que el método getEmpresa del modelo EmpresaModel obtenga correctamente los datos de la empresa mediante una consulta simulada.

Codigo:

```
    public function testGetEmpresa()
        {
            // Datos simulados que se esperan retornar del método select
            $mockedData = [
                'id' => 1,
                'ruc' => '123456789',
                'nombre' => 'Empresa A',
                'telefono' => '987654321',
                'correo' => 'empresa@example.com',
                'direccion' => 'Calle Principal',
                'mensaje' => 'Bienvenido a nuestra empresa',
                'logo' => 'logo.jpg',
            ];

            // Configurar el mock para el método select en la consulta de empresa
            $this->mockQuery->shouldReceive('select')
                ->once()
                ->with("SELECT * FROM configuracion")
                ->andReturn($mockedData);

            // Instanciar el modelo y llamar al método que queremos probar
```

```
            $empresaModel = new EmpresaModel($this->mockQuery);
            $result = $empresaModel->getEmpresa();

            // Verificar que el resultado retornado sea el mismo que $mockedData
            $this->assertEquals($mockedData, $result);
        }
```

testModificar	Verifica que el método modificar del modelo EmpresaModel actualice correctamente los datos de la empresa mediante una consulta simulada.

Codigo:

```
    public function testModificar()
        {
            // Datos simulados para la modificación
            $ruc = '123456789';
            $nombre = 'Empresa B';
            $tel = '987654321';
            $correo = 'empresa@example.com';
            $dir = 'Calle Secundaria';
            $mensaje = 'Bienvenido a nuestra nueva empresa';
            $img = 'logo.jpg';
            $id = 1;

            // Configurar el mock para el método save en la actualización de configuracion
            $this->mockQuery->shouldReceive('save')
                ->once()
                ->with(
                        "UPDATE configuracion SET ruc=?, nombre = ?, telefono =?, correo=?,
    direccion=?, mensaje=?, logo = ? WHERE id=?",
                    [$ruc, $nombre, $tel, $correo, $dir, $mensaje, $img, $id]
                )
                ->andReturn(1); // Simular que la actualización fue exitosa

            // Instanciar el modelo y llamar al método que queremos probar
            $empresaModel = new EmpresaModel($this->mockQuery);
             $result = $empresaModel->modificar($ruc, $nombre, $tel, $correo, $dir, $mensaje,
    $img, $id);

            // Verificar que el resultado retornado sea "ok"
            $this->assertEquals('ok', $result);
        }
```

testModificarError	Verifica que el método modificar del modelo EmpresaModel maneje correctamente el escenario de error durante la actualización de datos.

Codigo:

```
public function testModificarError()
    {
        // Datos simulados para la modificación
        $ruc = '123456789';
        $nombre = 'Empresa B';
        $tel = '987654321';
        $correo = 'empresa@example.com';
        $dir = 'Calle Secundaria';
        $mensaje = 'Bienvenido a nuestra nueva empresa';
        $img = 'logo.jpg';
        $id = 1;

        // Configurar el mock para el método save en la actualización de configuracion
        $this->mockQuery->shouldReceive('save')
            ->once()
            ->with(
                    "UPDATE configuracion SET ruc=?, nombre = ?, telefono =?, correo=?,
direccion=?, mensaje=?, logo = ? WHERE id=?",
                [$ruc, $nombre, $tel, $correo, $dir, $mensaje, $img, $id]
            )
            ->andReturn(0); // Simular que la actualización falló

        // Instanciar el modelo y llamar al método que queremos probar
        $empresaModel = new EmpresaModel($this->mockQuery);
         $result = $empresaModel->modificar($ruc, $nombre, $tel, $correo, $dir, $mensaje,
$img, $id);

        // Verificar que el resultado retornado sea "error"
        $this->assertEquals('error', $result);
    }
```

- Test de LoginModel.php

Titulo del Test	Descripción
testVerifyWithOtherTable	Verifica que el método verify del modelo LoginModel valide correctamente las credenciales del usuario contra otra tabla mediante una consulta simulada.

Codigo:

```
public function testVerifyWithOtherTable()
    {
        // Datos simulados que se esperan retornar del método select
            $mockedData = [['id' => 1, 'correo' => 'admin@example.com', 'clave' => 'hashed_password', 'estado' => 1]];

        // Configurar el mock para el método select en la consulta de verify
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT * FROM otra_tabla WHERE correo = 'admin@example.com' AND clave = 'hashed_password' AND estado = 1")
            ->andReturn($mockedData);

        // Instanciar LoginModel con el mock de Query
        $loginModel = new LoginModel($this->mockQuery);

        // Llamar al método verify y almacenar el resultado
        $result = $loginModel->verify('otra_tabla', 'admin@example.com', 'hashed_password');

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

- Test de MajorModel.php

Titulo del Test	Descripción
testVerify	Verifica que el método verify del modelo MajorModel obtenga correctamente los datos de un cliente específico mediante una consulta simulada.

Codigo:

```
public function testVerify()
    {
        // Datos simulados que se esperan retornar del método select
        $item = 'correo';
        $valor = 'test@example.com';
       $mockedData = [['id' => 1, 'nombre' => 'Test Cliente', 'correo' => 'test@example.com', 'codphone' => '123', 'telefono' => '456', 'direccion' => 'Calle Principal', 'clave' => 'hashed_password']];

        // Configurar el mock para el método select en la consulta de verify
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT * FROM clientes WHERE correo = 'test@example.com'")
            ->andReturn($mockedData);

        // Instanciar MajorModel con el mock de Query
```

```
        $majorModel = new MajorModel($this->mockQuery);

        // Llamar al método verify y almacenar el resultado
        $result = $majorModel->verify($item, $valor);

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testRegister	Verifica que el método register del modelo MajorModel registre un nuevo cliente correctamente mediante una consulta de inserción simulada.

Codigo:

```
    public function testRegister()
        {
            // Datos simulados para la inserción
            $nombre = 'Nuevo Cliente';
            $correo = 'nuevo@example.com';
            $codphone = '234';
            $telefono = '789';
            $direccion = 'Avenida Principal';
            $clave = 'hashed_password';

            // Configurar el mock para el método insertar en la consulta de register
            $this->mockQuery->shouldReceive('insertar')
                ->with(
                    "INSERT INTO clientes (nombre, correo, codphone, telefono, direccion, clave)
    VALUES (?,?,?,?,?,?)",
                    [$nombre, $correo, $codphone, $telefono, $direccion, $clave]
                )
                ->andReturn(true); // Supongamos que devuelve verdadero (éxito)

            // Instanciar MajorModel con el mock de Query
            $majorModel = new MajorModel($this->mockQuery);

            // Llamar al método register y almacenar el resultado
            $result = $majorModel->register($nombre, $correo, $codphone, $telefono, $direccion,
    $clave);

            // Verificar que el resultado retornado sea verdadero (éxito)
            $this->assertTrue($result);
        }
```

- Test de MarcasModel.php

Titulo del Test	Descripción
testGetMarcas	Verifica que el método getMarcas retorna las marcas activas de la base de datos según el estado proporcionado.

Codigo:

```
public function testGetMarcas()
    {
        // Datos simulados que se esperan retornar del método selectAll
        $estado = 1;
        $mockedData = [
            ['id' => 1, 'marca' => 'Marca A', 'estado' => 1],
            ['id' => 2, 'marca' => 'Marca B', 'estado' => 1],
        ];

        // Configurar el mock para el método selectAll en la consulta de getMarcas
        $this->mockQuery->shouldReceive('selectAll')
            ->with("SELECT * FROM marcas WHERE estado = $estado")
            ->andReturn($mockedData);

        // Instanciar MarcasModel con el mock de Query
        $marcasModel = new MarcasModel();

        // Llamar al método getMarcas y almacenar el resultado
        $result = $marcasModel->getMarcas($estado);

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testRegistrarMarca	Verifica que el método registrarMarca inserta una nueva marca en la base de datos cuando no existe una marca con el mismo nombre.

Codigo:

```
public function testRegistrarMarca()
    {
        // Datos simulados para la inserción
        $marcaNueva = 'Nueva Marca';

        // Configurar el mock para el método select y save en la consulta de registrarMarca
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT * FROM marcas WHERE marca = '$marcaNueva'")
            ->andReturn([]); // Simular que la marca no existe
```

```
            $this->mockQuery->shouldReceive('save')
                ->with("INSERT INTO marcas(marca) VALUES (?)", [$marcaNueva])
                ->andReturn(1); // Supongamos que devuelve 1 (éxito)

            // Instanciar MarcasModel con el mock de Query
            $marcasModel = new MarcasModel();

            // Llamar al método registrarMarca y almacenar el resultado
            $result = $marcasModel->registrarMarca($marcaNueva);

            // Verificar que el resultado retornado sea "ok"
            $this->assertEquals('ok', $result);
        }
```

testRegistrarMarcaExistente	Verifica que el método registrarMarca retorna "existe" cuando se intenta registrar una marca que ya existe en la base de datos.

Codigo:

```
    public function testRegistrarMarcaExistente()
        {
            // Marca que ya existe en la base de datos
            $marcaExistente = 'MarcaExistente';

            // Configurar el mock para el método select en la consulta de registrarMarca
            $this->mockQuery->shouldReceive('select')
                ->with("SELECT * FROM marcas WHERE marca = '$marcaExistente'")
                ->andReturn([['id' => 1, 'marca' => $marcaExistente]]); // Simular que la marca
    existe

            // No se debe llamar a save, así que no configuramos expectativa para save

            // Instanciar MarcasModel con el mock de Query
            $marcasModel = new MarcasModel();

            // Llamar al método registrarMarca con la marca existente y almacenar el resultado
            $result = $marcasModel->registrarMarca($marcaExistente);

            // Verificar que el resultado retornado sea "existe"
            $this->assertEquals('existe', $result);
        }
```

testRegistrarMarcaError	Verifica que el método registrarMarca retorna "error" cuando ocurre un error al intentar insertar una nueva marca en la base de datos.

Codigo:

```
public function testRegistrarMarcaError()
    {
        // Marca a registrar
        $marcaNueva = 'Nueva Marca';

        // Configurar el mock para el método select y save en la consulta de registrarMarca
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT * FROM marcas WHERE marca = '$marcaNueva'")
            ->andReturn([]); // Simular que la marca no existe

        // Configurar el mock para simular un error en la inserción
        $this->mockQuery->shouldReceive('save')
            ->with("INSERT INTO marcas(marca) VALUES (?)", [$marcaNueva])
            ->andReturn(0); // Simular que devuelve 0 (error)

        // Instanciar MarcasModel con el mock de Query
        $marcasModel = new MarcasModel();

        // Llamar al método registrarMarca y almacenar el resultado
        $result = $marcasModel->registrarMarca($marcaNueva);

        // Verificar que el resultado retornado sea "error"
        $this->assertEquals('error', $result);
    }
```

testModificarMarca	Verifica que el método modificarMarca actualiza correctamente una marca existente en la base de datos.

Codigo:

```
public function testModificarMarca()
    {
        // Datos simulados para la modificación
        $marcaModificada = 'Marca Modificada';
        $idMarca = 1;

        // Configurar el mock para el método save en la consulta de modificarMarca
        $this->mockQuery->shouldReceive('save')
            ->with("UPDATE marcas SET marca = ? WHERE id = ?", [$marcaModificada, $idMarca])
            ->andReturn(1); // Supongamos que devuelve 1 (éxito)
```

```
            // Instanciar MarcasModel con el mock de Query
            $marcasModel = new MarcasModel();

            // Llamar al método modificarMarca y almacenar el resultado
            $result = $marcasModel->modificarMarca($marcaModificada, $idMarca);

            // Verificar que el resultado retornado sea "modificado"
            $this->assertEquals('modificado', $result);
        }
```

testModificarMarcaError	Verifica que el método modificarMarca retorna "error" cuando ocurre un error al intentar actualizar una marca existente en la base de datos.

Codigo:

```
    public function testModificarMarcaError()
        {
            // Datos simulados para la modificación
            $marcaModificada = 'Marca Modificada';
            $idMarca = 1;

            // Configurar el mock para el método save en la consulta de modificarMarca
            $this->mockQuery->shouldReceive('save')
                ->with("UPDATE marcas SET marca = ? WHERE id = ?", [$marcaModificada, $idMarca])
                ->andReturn(0); // Supongamos que devuelve 0 (error)

            // Instanciar MarcasModel con el mock de Query
            $marcasModel = new MarcasModel();

            // Llamar al método modificarMarca y almacenar el resultado
            $result = $marcasModel->modificarMarca($marcaModificada, $idMarca);

            // Verificar que el resultado retornado sea "error"
            $this->assertEquals('error', $result);
        }
```

testEditarMarca	Verifica que el método editarMarca retorna los datos correctos de una marca específica según su ID.

Codigo:

```
    public function testEditarMarca()
        {
            // Datos simulados para la edición
            $idMarca = 1;
```

```
        $mockedData = ['id' => 1, 'marca' => 'Marca A', 'estado' => 1];

        // Configurar el mock para el método select en la consulta de editarMarca
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT * FROM marcas WHERE id = $idMarca")
            ->andReturn($mockedData);

        // Instanciar MarcasModel con el mock de Query
        $marcasModel = new MarcasModel();

        // Llamar al método editarMarca y almacenar el resultado
        $result = $marcasModel->editarMarca($idMarca);

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testAccionMarca	Verifica que el método accionMarca cambia correctamente el estado de una marca específica en la base de datos.

Codigo:

```
    public function testAccionMarca()
    {
        // Datos simulados para la acción de cambio de estado
        $estadoNuevo = 0;
        $idMarca = 1;

        $this->mockQuery->shouldReceive('save')
        ->with("UPDATE marcas SET estado = ? WHERE id = ?", [$estadoNuevo, $idMarca])
        ->andReturn(true); // Devolver true en lugar de 1

        // Instanciar MarcasModel con el mock de Query
        $marcasModel = new MarcasModel();

        // Llamar al método accionMarca y almacenar el resultado
        $result = $marcasModel->accionMarca($estadoNuevo, $idMarca);

        // Verificar que el resultado retornado sea verdadero (éxito)
        $this->assertTrue($result);
    }
```

- Test de PricingModel.php

Titulo del Test	Descripción
testGetVehiculos	Verifica que el método getVehiculos retorne los vehículos activos correctamente, incluyendo marcas y tipos asociados.

Codigo:

```
public function testGetVehiculos()
    {
        // Estado simulado
        $estado = 1;

        // Datos simulados que se esperan retornar del método selectAll
        $mockedData = [
            ['id' => 1, 'nombre' => 'Vehículo A', 'marca' => 'Marca A', 'tipo' => 'Tipo A',
'estado' => 1],
            ['id' => 2, 'nombre' => 'Vehículo B', 'marca' => 'Marca B', 'tipo' => 'Tipo B',
'estado' => 1],
        ];

        // Configurar el mock para el método selectAll en la consulta de vehículos
        $this->mockQuery->shouldReceive('selectAll')
            ->once()
              ->with("SELECT v.*, m.marca, t.tipo FROM vehiculos v INNER JOIN marcas m ON
v.id_marca = m.id INNER JOIN tipos t ON v.id_tipo = t.id WHERE v.estado = $estado")
            ->andReturn($mockedData);

        // Instanciar PricingModel y llamar al método getVehiculos
        $pricingModel = new PricingModel();
        $result = $pricingModel->getVehiculos($estado);

        // Verificar que el resultado retornado sea el mismo que $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

- Test de ReservasModel.php

Titulo del Test	Descripción
testGetReservasIdVehiculoVacio	Verifica que el método getReservas retorne todas las reservas cuando id_vehiculo está vacío.

Codigo:

```
public function testGetReservasIdVehiculoVacio()
    {
        // SQL esperado cuando id_vehiculo está vacío
        $sql = "SELECT r.*, c.nombre FROM reservas r INNER JOIN clientes c ON r.id_cliente
= c.id";

        // Datos simulados que se esperan retornar del método selectAll
        $mockedData = [
            ['id' => 1, 'nombre' => 'Reserva A'],
            ['id' => 2, 'nombre' => 'Reserva B']
        ];

         // Configurar el mock para el método selectAll en la consulta de reservas sin
id_vehiculo
        $this->mockQuery->shouldReceive('selectAll')
            ->with($sql)
            ->once() // Esperamos una sola llamada a selectAll con este SQL
            ->andReturn($mockedData);

        // Instanciar ReservasModel y llamar al método getReservas con id_vehiculo vacío
        $reservasModel = new ReservasModel();
        $result = $reservasModel->getReservas('');

        // Verificar que el resultado retornado coincida con $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testGetReservasIdVehiculoEspecifico	Verifica que el método getReservas retorne las reservas específicas para un id_vehiculo dado.

Codigo:

```
public function testGetReservasIdVehiculoEspecifico()
    {
        // Valor específico de id_vehiculo
        $id_vehiculo = 1;

        // SQL esperado cuando id_vehiculo tiene un valor específico
        $sql = "SELECT r.*, c.nombre FROM reservas r INNER JOIN clientes c ON r.id_cliente
= c.id WHERE r.id_vehiculo = $id_vehiculo";
```

```
            // Datos simulados que se esperan retornar del método selectAll
            $mockedData = [
                ['id' => 1, 'nombre' => 'Reserva A']
            ];

             // Configurar el mock para el método selectAll en la consulta de reservas con
    id_vehiculo específico
            $this->mockQuery->shouldReceive('selectAll')
                ->with($sql)
                ->once() // Esperamos una sola llamada a selectAll con este SQL
                ->andReturn($mockedData);

            // Instanciar ReservasModel y llamar al método getReservas con id_vehiculo específico
            $reservasModel = new ReservasModel();
            $result = $reservasModel->getReservas($id_vehiculo);

            // Verificar que el resultado retornado coincida con $mockedData
            $this->assertEquals($mockedData, $result);
        }
```

testRegistrarReserva	Verifica que el método registrarReserva inserte una nueva reserva correctamente y retorne el ID de la reserva.

Codigo:

```
    public function testRegistrarReserva()
        {
            // Datos simulados para la inserción de reserva
            $fecha = '2024-06-10';
            $fecha_devolucion = '2024-06-15';
            $cantidad = 1;
            $tipo = 'tipo';
            $monto = 100.50;
            $fecha_reserva = '2024-06-01';
            $observacion = 'Observación';
            $id_veh = 1;
            $id_cli = 1;

            // Configurar el mock para el método insertar en la inserción de reserva
            $this->mockQuery->shouldReceive('insertar')
                ->once()
                ->andReturn(1); // Simulamos que se insertó correctamente y devolvió un ID
```

```
            // Instanciar ReservasModel y llamar al método registrarReserva con datos simulados
            $reservasModel = new ReservasModel();
              $result = $reservasModel->registrarReserva($fecha, $fecha_devolucion, $cantidad,
    $tipo, $monto, $fecha_reserva, $observacion, $id_veh, $id_cli);

            // Verificar que el ID retornado sea 1
            $this->assertEquals(1, $result);
        }
```

testRegistrarReserva_Fallo	Verifica que el método registrarReserva retorne 0 cuando falla la inserción de una nueva reserva.

Codigo:

```
    public function testRegistrarReserva_Fallo()
        {
            // Datos simulados para la inserción de reserva que falla
            $fecha = '2024-06-10';
            $fecha_devolucion = '2024-06-15';
            $cantidad = 1;
            $tipo = 'tipo';
            $monto = 100.50;
            $fecha_reserva = '2024-06-01';
            $observacion = 'Observación';
            $id_veh = 1;
            $id_cli = 1;

            // Configurar el mock para el método insertar en la inserción de reserva
            $this->mockQuery->shouldReceive('insertar')
                ->once()
                ->andReturn(0); // Simulamos que la inserción falla y devuelve 0
            // Instanciar ReservasModel y llamar al método registrarReserva con datos simulados
            $reservasModel = new ReservasModel();
              $result = $reservasModel->registrarReserva($fecha, $fecha_devolucion, $cantidad,
    $tipo, $monto, $fecha_reserva, $observacion, $id_veh, $id_cli);

            // Verificar que el resultado sea 0, indicando que la inserción falló
            $this->assertEquals(0, $result);
        }
```

testGetVehiculos	Verifica que el método getVehiculos del ReservasModel retorne los vehículos activos correctamente.

Codigo:

```
    public function testGetVehiculos()
        {
            // SQL esperado en el método getVehiculos
```

```
        $sql = "SELECT v.*, m.marca, t.tipo FROM vehiculos v INNER JOIN marcas m ON v.id_marca
    = m.id INNER JOIN tipos t ON v.id_tipo = t.id WHERE v.estado != 0";

        // Datos simulados que se esperan retornar del método selectAll
        $mockedData = [
            ['id' => 1, 'marca' => 'Marca A', 'tipo' => 'Tipo A', 'modelo' => 'Modelo A',
    'estado' => 1],
            ['id' => 2, 'marca' => 'Marca B', 'tipo' => 'Tipo B', 'modelo' => 'Modelo B',
    'estado' => 1],
        ];

        // Configurar el mock para el método selectAll en la consulta de vehículos
        $this->mockQuery->shouldReceive('selectAll')
            ->with($sql)
            ->once() // Esperamos una sola llamada a selectAll con este SQL
            ->andReturn($mockedData);

        // Instanciar ReservasModel y llamar al método getVehiculos
        $reservasModel = new ReservasModel();
        $result = $reservasModel->getVehiculos();

        // Verificar que el resultado retornado coincida con $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testGetVehiculo	Verifica que el método getVehiculo retorne los datos correctos de un vehículo específico por su ID.

Codigo:

```
    public function testGetVehiculo()
    {
        $id_vehiculo = 1;

        // SQL esperado en el método getVehiculo con el ID específico
        $sql = "SELECT v.*, m.marca, t.tipo FROM vehiculos v INNER JOIN marcas m ON v.id_marca
    = m.id INNER JOIN tipos t ON v.id_tipo = t.id WHERE v.id = $id_vehiculo";

        // Datos simulados que se esperan retornar del método select
        $mockedData = [
            'id' => 1,
            'marca' => 'Marca A',
            'tipo' => 'Tipo A',
            'modelo' => 'Modelo A',
            'estado' => 1
```

```
        ];

        // Configurar el mock para el método select en la consulta de vehículo por ID
        $this->mockQuery->shouldReceive('select')
            ->with($sql)
            ->once() // Esperamos una sola llamada a select con este SQL
            ->andReturn($mockedData);

        // Instanciar ReservasModel y llamar al método getVehiculo con el ID específico
        $reservasModel = new ReservasModel();
        $result = $reservasModel->getVehiculo($id_vehiculo);

        // Verificar que el resultado retornado coincida con $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testGetNuevasReservas	Verifica que el método getNuevasReservas retorne las últimas cinco reservas nuevas.

Codigo:

```
    public function testGetNuevasReservas()
    {
        // SQL esperado en el método getNuevasReservas
        $expectedSql = "SELECT r.id, r.f_reserva, c.nombre FROM reservas r INNER JOIN clientes
c ON r.id_cliente = c.id WHERE r.estado = 0 ORDER BY r.id DESC LIMIT 5";

        // Datos simulados que se esperan retornar del método selectAll
        $mockedData = [
            ['id' => 1, 'f_reserva' => '2024-07-01', 'nombre' => 'Cliente A'],
            ['id' => 2, 'f_reserva' => '2024-06-30', 'nombre' => 'Cliente B']
        ];

        // Configurar el mock para el método selectAll en la consulta de nuevas reservas
        $this->mockQuery->shouldReceive('selectAll')
            ->with($expectedSql)
            ->once() // Esperamos una sola llamada a selectAll con este SQL
            ->andReturn($mockedData);

        // Instanciar ReservasModel y llamar al método getNuevasReservas
        $reservasModel = new ReservasModel();
        $result = $reservasModel->getNuevasReservas();
        // Verificar que el resultado retornado coincida con $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testGetReserva	Verifica que el método getReserva retorne los datos correctos de una reserva específica por su ID.

Codigo:

```
public function testGetReserva()
    {
        $id_reserva = 1;

        // SQL esperado en el método getReserva con el ID específico
        $expectedSql = "SELECT r.*, c.nombre, c.correo FROM reservas r INNER JOIN clientes
c ON r.id_cliente = c.id WHERE r.id = $id_reserva";

        // Datos simulados que se esperan retornar del método select
        $mockedData = [
            'id' => 1,
            'f_reserva' => '2024-07-01',
            'nombre' => 'Cliente A',
            'correo' => 'clienteA@example.com'
        ];

        // Configurar el mock para el método select en la consulta de reserva por ID
        $this->mockQuery->shouldReceive('select')
            ->with($expectedSql)
            ->once() // Esperamos una sola llamada a select con este SQL
            ->andReturn($mockedData);

        // Instanciar ReservasModel y llamar al método getReserva con el ID específico
        $reservasModel = new ReservasModel();
        $result = $reservasModel->getReserva($id_reserva);

        // Verificar que el resultado retornado coincida con $mockedData
        $this->assertEquals($mockedData, $result);
    }
```

testActualizarEstado	Verifica que el método actualizarEstado actualice el estado de una reserva correctamente.

Codigo:

```
public function testActualizarEstado()
    {
        $id_reserva = 1;
        $nuevo_estado = 1; // Nuevo estado que se va a actualizar

        // SQL esperado en el método actualizarEstado
        $expectedSql = "UPDATE reservas SET estado = ? WHERE id = ?";
```

```
            // Configurar el mock para el método save en la actualización de estado
            $this->mockQuery->shouldReceive('save')
                ->with($expectedSql, [$nuevo_estado, $id_reserva])
                ->once() // Esperamos una sola llamada a save con este SQL y estos datos
                ->andReturn(1); // Simulamos que la actualización fue exitosa

            // Instanciar ReservasModel y llamar al método actualizarEstado con los parámetros
    simulados
            $reservasModel = new ReservasModel();
            $result = $reservasModel->actualizarEstado($nuevo_estado, $id_reserva);

            // Verificar que el resultado retornado sea 1, indicando que la actualización fue
    exitosa
            $this->assertEquals(1, $result);
        }
```

testGetEmpresa	Verifica que el método getEmpresa retorne los datos de configuración de la empresa correctamente.

Codigo:

```
    public function testGetEmpresa()
        {
            // SQL esperado en el método getEmpresa
            $expectedSql = "SELECT * FROM configuracion";

            // Datos simulados que se esperan retornar del método select
            $mockedData = [
                'nombre' => 'Mi Empresa',
                'direccion' => 'Calle Principal',
                'telefono' => '123456789',
                'email' => 'info@miempresa.com'
            ];
           // Configurar el mock para el método select en la consulta de configuración de empresa
            $this->mockQuery->shouldReceive('select')
                ->with($expectedSql)
                ->once() // Esperamos una sola llamada a select con este SQL
                ->andReturn($mockedData);
            // Instanciar ReservasModel y llamar al método getEmpresa
            $reservasModel = new ReservasModel();
            $result = $reservasModel->getEmpresa();
            // Verificar que el resultado retornado coincida con $mockedData
            $this->assertEquals($mockedData, $result);
        }
```

- Test de TiposModel.php

Titulo del Test	Descripción
testGetTipos	Verifica que el método getTipos retorne los tipos activos correctamente.

Codigo:

```
public function testGetTipos()
    {
        // Estado de los tipos a buscar
        $estado = 1;

        // Datos simulados de tipos que se esperan retornar
        $mockedData = [
            ['id' => 1, 'tipo' => 'Tipo A', 'estado' => 1],
            ['id' => 2, 'tipo' => 'Tipo B', 'estado' => 1],
        ];

        // Configuración de mock para el método selectAll en el modelo
        $sql = "SELECT * FROM tipos WHERE estado = $estado";
        $this->mockQuery->shouldReceive('selectAll')
            ->with($sql)
            ->andReturn($mockedData);

        // Instanciar TiposModel y llamar al método getTipos
        $tiposModel = new TiposModel();
        $result = $tiposModel->getTipos($estado);

        // Verificar que el resultado coincida con los datos esperados
        $this->assertEquals($mockedData, $result);
    }
```

testRegistrarTipo	Verifica que el método registrarTipo inserte un nuevo tipo correctamente cuando no existe previamente.

Codigo:

```
public function testRegistrarTipo()
    {
        // Tipo a registrar
        $tipo = 'Nuevo Tipo';

        // Configuración de mock para verificar si el tipo ya existe (caso no existente)
        $this->mockQuery->shouldReceive('select')
            ->andReturn([]); // Simular que no existe ningún tipo con este nombre

        // Configuración de mock para el método save en el registro de tipo (caso exitoso)
```

```
            $this->mockQuery->shouldReceive('save')
                ->andReturn(1); // Simular que la inserción fue exitosa

            // Instanciar TiposModel y llamar al método registrarTipo
            $tiposModel = new TiposModel();
            $result = $tiposModel->registrarTipo($tipo);

            // Verificar que el resultado sea "ok" (registro exitoso)
            $this->assertEquals('ok', $result);
        }
```

testRegistrarTipoExistente	Verifica que el método registrarTipo detecte y retorne "existe" cuando el tipo ya existe.

Codigo:

```
    public function testRegistrarTipoExistente()
        {
            // Tipo a registrar
            $tipo = 'Nuevo Tipo';

            // Configuración de mock para verificar si el tipo ya existe (caso existente)
            $this->mockQuery->shouldReceive('select')
                ->andReturn(['id' => 1, 'tipo' => 'Nuevo Tipo']); // Simular que ya existe un
    tipo con este nombre

            // Instanciar TiposModel y llamar al método registrarTipo
            $tiposModel = new TiposModel();
            $result = $tiposModel->registrarTipo($tipo);

            // Verificar que el resultado sea "existe" (tipo ya existe)
            $this->assertEquals('existe', $result);
        }
```

testRegistrarTipoError	Verifica que el método registrarTipo retorne "error" cuando falla la inserción de un nuevo tipo.

Codigo:

```
    public function testRegistrarTipoError()
        {
            // Tipo a registrar
            $tipo = 'Nuevo Tipo';

            // Configuración de mock para verificar si el tipo ya existe (caso no existente)
            $this->mockQuery->shouldReceive('select')
```

```
            ->andReturn([]); // Simular que no existe ningún tipo con este nombre

        // Configuración de mock para el método save en el registro de tipo (caso de error)
        $this->mockQuery->shouldReceive('save')
            ->andReturn(0); // Simular que hubo un error en la inserción

        // Instanciar TiposModel y llamar al método registrarTipo
        $tiposModel = new TiposModel();
        $result = $tiposModel->registrarTipo($tipo);

        // Verificar que el resultado sea "error" (error en la inserción)
        $this->assertEquals('error', $result);
    }
```

testModificarTipo	Verifica que el método modificarTipo actualice un tipo correctamente.

Codigo:

```
    public function testModificarTipo()
    {
        // ID del tipo a modificar
        $tipoId = 1;
        $nuevoTipo = 'Tipo Modificado';

        // Configuración de mock para el método save en la modificación de tipo
        $this->mockQuery->shouldReceive('save')
            ->andReturn(1); // Simular que la actualización fue exitosa

        // Instanciar TiposModel y llamar al método modificarTipo
        $tiposModel = new TiposModel();
        $result = $tiposModel->modificarTipo($nuevoTipo, $tipoId);

        // Verificar que el resultado sea "modificado" (actualización exitosa)
        $this->assertEquals('modificado', $result);
    }
```

testModificarTipoError	Verifica que el método modificarTipo retorne "error" cuando falla la actualización de un tipo.

Codigo:

```
    public function testModificarTipoError()
    {
        // ID del tipo a modificar
        $tipoId = 1;
        $nuevoTipo = 'Tipo Modificado';
```

```
        // Configuración de mock para el método save en la modificación de tipo (caso de error)
        $this->mockQuery->shouldReceive('save')
            ->andReturn(0); // Simular que hubo un error en la actualización

        // Instanciar TiposModel y llamar al método modificarTipo
        $tiposModel = new TiposModel();
        $result = $tiposModel->modificarTipo($nuevoTipo, $tipoId);

        // Verificar que el resultado sea "error" (error en la actualización)
        $this->assertEquals('error', $result);
    }
```

testEditarTipo	Verifica que el método editarTipo retorne los datos correctos de un tipo específico por su ID.

Codigo:

```
    public function testEditarTipo()
    {
        // ID del tipo a editar
        $tipoId = 1;

        // Datos simulados del tipo que se espera retornar
        $mockedTipo = [
            'id' => 1,
            'tipo' => 'Tipo A',
            'estado' => 1,
        ];

        // Configuración de mock para el método select en el modelo
        $sql = "SELECT * FROM tipos WHERE id = $tipoId";
        $this->mockQuery->shouldReceive('select')
            ->with($sql)
            ->andReturn($mockedTipo);

        // Instanciar TiposModel y llamar al método editarTipo
        $tiposModel = new TiposModel();
        $result = $tiposModel->editarTipo($tipoId);

        // Verificar que el resultado coincida con los datos esperados
        $this->assertEquals($mockedTipo, $result);
    }
```

testEditarTipoNoExistente	Verifica que el método editarTipo retorne null cuando el tipo especificado no existe.

Codigo:

```
public function testEditarTipoNoExistente()
    {
        // ID del tipo a editar
        $tipoId = 999; // ID que no existe

        // Configuración de mock para el método select en el modelo (caso no existente)
        $sql = "SELECT * FROM tipos WHERE id = $tipoId";
        $this->mockQuery->shouldReceive('select')
            ->with($sql)
            ->andReturn(null); // Simular que no se encontró ningún tipo con ese ID

        // Instanciar TiposModel y llamar al método editarTipo
        $tiposModel = new TiposModel();
        $result = $tiposModel->editarTipo($tipoId);

        // Verificar que el resultado sea null (tipo no encontrado)
        $this->assertNull($result);
    }
```

testAccionTipo	Verifica que el método accionTipo actualice el estado de un tipo correctamente.

Codigo:

```
public function testAccionTipo()
    {
        // ID del tipo a modificar
        $tipoId = 1;
        $nuevoEstado = 0;

        // Configuración de mock para el método save en la acción de tipo
        $this->mockQuery->shouldReceive('save')
            ->andReturn(1); // Simular que la actualización fue exitosa

        // Instanciar TiposModel y llamar al método accionTipo
        $tiposModel = new TiposModel();
        $result = $tiposModel->accionTipo($nuevoEstado, $tipoId);

        // Verificar que el resultado sea 1 (éxito en la acción)
        $this->assertEquals(1, $result);
    }
```

testAccionTipoError	Verifica que el método accionTipo retorne 0 cuando falla la actualización del estado de un tipo.

Codigo:

```
public function testAccionTipoError()
    {
        // ID del tipo a modificar
        $tipoId = 1;
        $nuevoEstado = 0;

        // Configuración de mock para el método save en la acción de tipo (caso de error)
        $this->mockQuery->shouldReceive('save')
            ->andReturn(0); // Simular que hubo un error en la actualización

        // Instanciar TiposModel y llamar al método accionTipo
        $tiposModel = new TiposModel();
        $result = $tiposModel->accionTipo($nuevoEstado, $tipoId);

        // Verificar que el resultado sea 0 (error en la acción)
        $this->assertEquals(0, $result);
    }
```

- Test de UsuariosModel.php

Titulo del Test	Descripción
testGetUsuario	Verifica que el método getUsuario retorne los datos del usuario correctamente al proporcionar el usuario y la clave.

Codigo:

```
public function testGetUsuario()
    {
        // Preparar los datos esperados
        $usuario = 'admin';
        $clave = 'password';
       $mockedData = [['id' => 1, 'usuario' => 'admin', 'nombre' => 'Administrador', 'correo'
=> 'admin@example.com', 'estado' => 1]];

        // Configurar el comportamiento esperado del mock de Query para el método select
        $this->mockQuery->shouldReceive('select')
            ->with("SELECT * FROM usuarios WHERE usuario = '$usuario' AND clave = '$clave'")
            ->once()
            ->andReturn($mockedData);
```

```
            // Instanciar el modelo y llamar al método que queremos probar
            $usuariosModel = new UsuariosModel();
            $result = $usuariosModel->getUsuario($usuario, $clave);

            // Asegurar que el resultado coincide con los datos esperados
            $this->assertEquals($mockedData, $result);
        }
```

testRegistrarUsuarioExitoso	Verifica que el método registrarUsuario inserte un nuevo usuario correctamente cuando no existe previamente.

Codigo:

```
    public function testRegistrarUsuarioExitoso()
        {
            // Datos simulados para el registro de un nuevo usuario
            $usuario = 'nuevo_usuario';
            $nombre = 'Nuevo Usuario';
            $correo = 'nuevo_usuario@example.com';
            $telefono = '123456789';
            $clave = 'password';

            // Configuración de mock para verificar si el usuario ya existe
            $this->mockQuery->shouldReceive('select')
                ->andReturn([]); // Simular que no existe ningún usuario con este nombre o correo

            // Configuración de mock para el método save en el registro de usuario
            $this->mockQuery->shouldReceive('save')
                ->andReturn(1); // Simular que la inserción fue exitosa

            // Instanciar UsuariosModel y llamar al método registrarUsuario
            $usuariosModel = new UsuariosModel();
             $result = $usuariosModel->registrarUsuario($usuario, $nombre, $correo, $telefono,
    $clave);

            // Verificar que el resultado sea "ok" (registro exitoso)
            $this->assertEquals('ok', $result);
        }
```

testRegistrarUsuarioExistente	Verifica que el método registrarUsuario detecte y retorne "existe" cuando el usuario ya existe.

Codigo:

```
    public function testRegistrarUsuarioExistente()
        {
            // Datos simulados para el registro de un nuevo usuario
```

```
            $usuario = 'usuario_existente';
            $nombre = 'Usuario Existente';
            $correo = 'existente@example.com';
            $telefono = '987654321';
            $clave = 'password';

            // Configuración de mock para verificar si el usuario ya existe
            $this->mockQuery->shouldReceive('select')
                ->andReturn([['id' => 1]]); // Simular que el usuario ya existe en la base de
    datos

            // Instanciar UsuariosModel y llamar al método registrarUsuario
            $usuariosModel = new UsuariosModel();
             $result = $usuariosModel->registrarUsuario($usuario, $nombre, $correo, $telefono,
    $clave);

            // Verificar que el resultado sea "existe" (usuario ya registrado)
            $this->assertEquals('existe', $result);
        }
```

testRegistrarUsuarioError	Verifica que el método registrarUsuario retorne "error" cuando falla la inserción de un nuevo usuario.

Codigo:

```
    public function testRegistrarUsuarioError()
        {
            // Datos simulados para el registro de un nuevo usuario
            $usuario = 'nuevo_usuario';
            $nombre = 'Nuevo Usuario';
            $correo = 'nuevo_usuario@example.com';
            $telefono = '123456789';
            $clave = 'password';

            // Configuración de mock para verificar si el usuario ya existe
            $this->mockQuery->shouldReceive('select')
                ->andReturn([]); // Simular que no existe ningún usuario con este nombre o correo

            // Configuración de mock para el método save en el registro de usuario
            $this->mockQuery->shouldReceive('save')
                ->andReturn(0); // Simular que la inserción falló

            // Instanciar UsuariosModel y llamar al método registrarUsuario
            $usuariosModel = new UsuariosModel();
```

```
            $result = $usuariosModel->registrarUsuario($usuario, $nombre, $correo, $telefono,
    $clave);

            // Verificar que el resultado sea "error" (registro fallido)
            $this->assertEquals('error', $result);
        }
```

testGetUsuarios	Verifica que el método getUsuarios retorne los usuarios activos correctamente.

Codigo:

```
    public function testGetUsuarios()
        {
            // Estado del usuario a buscar
            $estado = 1;

            // Datos simulados de usuarios que se esperan retornar
            $mockedData = [
                ['id' => 1, 'usuario' => 'user1', 'nombre' => 'Usuario 1', 'correo' =>
    'user1@example.com', 'estado' => 1],
                ['id' => 2, 'usuario' => 'user2', 'nombre' => 'Usuario 2', 'correo' =>
    'user2@example.com', 'estado' => 1],
            ];

            // Configuración de mock para el método selectAll en el modelo
            $sql = "SELECT id,usuario,nombre,correo,estado FROM usuarios WHERE estado = $estado";
            $this->mockQuery->shouldReceive('selectAll')
                ->with($sql)
                ->andReturn($mockedData);

            // Instanciar UsuariosModel y llamar al método getUsuarios
            $usuariosModel = new UsuariosModel();
            $result = $usuariosModel->getUsuarios($estado);

            // Verificar que el resultado coincida con los datos esperados
            $this->assertEquals($mockedData, $result);
        }
```

testModificarUsuarioExitoso	Verifica que el método modificarUsuario actualice un usuario correctamente.

Codigo:

```
    public function testModificarUsuarioExitoso()
        {
            // Datos simulados para modificar un usuario existente
```

```
            $usuario = 'usuario_modificado';
            $nombre = 'Usuario Modificado';
            $correo = 'modificado@example.com';
            $telefono = '987654321';
            $idUsuario = 1;

            // Configuración de mock para el método save en la modificación de usuario
            $this->mockQuery->shouldReceive('save')
                ->andReturn(1); // Simular que la actualización fue exitosa

            // Instanciar UsuariosModel y llamar al método modificarUsuario
            $usuariosModel = new UsuariosModel();
             $result = $usuariosModel->modificarUsuario($usuario, $nombre, $correo, $telefono,
    $idUsuario);

            // Verificar que el resultado sea "modificado" (actualización exitosa)
            $this->assertEquals('modificado', $result);
        }
```

testModificarUsuarioFallido	Verifica que el método modificarUsuario retorne "error" cuando falla la actualización de un usuario.

Codigo:

```
    public function testModificarUsuarioFallido()
        {
            // Datos simulados para modificar un usuario existente
            $usuario = 'usuario_modificado';
            $nombre = 'Usuario Modificado';
            $correo = 'modificado@example.com';
            $telefono = '987654321';
            $idUsuario = 999; // ID inexistente para forzar un fallo

            // Configuración de mock para el método save en la modificación de usuario
            $this->mockQuery->shouldReceive('save')
                ->andReturn(0); // Simular que la actualización falló

            // Instanciar UsuariosModel y llamar al método modificarUsuario
            $usuariosModel = new UsuariosModel();
             $result = $usuariosModel->modificarUsuario($usuario, $nombre, $correo, $telefono,
    $idUsuario);

            // Verificar que el resultado sea "error" (actualización fallida)
            $this->assertEquals('error', $result);
        }
```

testEditarUser	Verifica que el método editarUser retorne los datos correctos de un usuario específico por su ID.

Codigo:

```
public function testEditarUser()
    {
        // ID del usuario a editar
        $userId = 1;

        // Datos simulados del usuario que se espera retornar
        $mockedUser = [
            'id' => 1,
            'usuario' => 'user1',
            'nombre' => 'Usuario 1',
            'correo' => 'user1@example.com',
            'telefono' => '123456789',
            'estado' => 1,
        ];

        // Configuración de mock para el método select en el modelo
        $sql = "SELECT * FROM usuarios WHERE id = $userId";
        $this->mockQuery->shouldReceive('select')
            ->with($sql)
            ->andReturn($mockedUser);

        // Instanciar UsuariosModel y llamar al método editarUser
        $usuariosModel = new UsuariosModel();
        $result = $usuariosModel->editarUser($userId);

        // Verificar que el resultado coincida con los datos esperados
        $this->assertEquals($mockedUser, $result);
    }
```

testAccionUser	Verifica que el método accionUser actualice el estado de un usuario correctamente.

Codigo:

```
public function testAccionUser()
    {
        // Estado y ID del usuario para cambiar
        $estado = 0; // Estado inactivo
        $idUsuario = 1;

        // Configuración de mock para el método save en la acción de usuario
        $this->mockQuery->shouldReceive('save')
```

```
                ->andReturn(1); // Simular que la actualización fue exitosa

            // Instanciar UsuariosModel y llamar al método accionUser
            $usuariosModel = new UsuariosModel();
            $result = $usuariosModel->accionUser($estado, $idUsuario);

            // Verificar que el resultado sea 1 (indicando éxito en la actualización)
            $this->assertEquals(1, $result);
        }
```

testModificarPass	Verifica que el método modificarPass actualice la contraseña de un usuario correctamente.

Codigo:

```
    public function testModificarPass()
        {
            // Nueva contraseña y ID del usuario
            $nuevaClave = 'nueva_password';
            $idUsuario = 1;

            // Configuración de mock para el método save en la modificación de contraseña
            $this->mockQuery->shouldReceive('save')
                ->andReturn(1); // Simular que la actualización fue exitosa

            // Instanciar UsuariosModel y llamar al método modificarPass
            $usuariosModel = new UsuariosModel();
            $result = $usuariosModel->modificarPass($nuevaClave, $idUsuario);

            // Verificar que el resultado sea 1 (indicando éxito en la actualización)
            $this->assertEquals(1, $result);
        }
```

testGetEmpresa	Verifica que el método getEmpresa retorne los datos de configuración de la empresa correctamente.

Codigo:

```
    public function testGetEmpresa()
        {
            // Datos simulados que se esperan retornar del método select
            $mockedData = [['id' => 1, 'nombre' => 'Empresa A', 'direccion' => 'Calle Principal',
    'telefono' => '123456789']];

            // Configurar el mock para el método select en la consulta de empresa
            $this->mockQuery->shouldReceive('select')
                ->with("SELECT * FROM configuracion")
```

```
                ->andReturn($mockedData);

            // Instanciar UsuariosModel y llamar al método getEmpresa
            $usuariosModel = new UsuariosModel();
            $result = $usuariosModel->getEmpresa();

            // Verificar que el resultado retornado sea el mismo que $mockedData
            $this->assertEquals($mockedData, $result);
        }
```

testModificarDato	Verifica que el método modificarDato actualice los datos de un usuario correctamente.

Codigo:

```
    public function testModificarDato()
        {
            // Datos simulados para modificar los datos de un usuario existente
            $usuario = 'usuario_modificado';
            $nombre = 'Usuario Modificado';
            $apellido = 'Apellido Modificado';
            $correo = 'modificado@example.com';
            $telefono = '987654321';
            $direccion = 'Calle Modificada';
            $perfil = 'imagen_modificada.jpg';
            $idUsuario = 1;

            // Configuración de mock para el método save en la modificación de datos de usuario
            $this->mockQuery->shouldReceive('save')
                ->andReturn(1); // Simular que la actualización fue exitosa

            // Instanciar UsuariosModel y llamar al método modificarDato
            $usuariosModel = new UsuariosModel();
              $result = $usuariosModel->modificarDato($usuario, $nombre, $apellido, $correo,
    $telefono, $direccion, $perfil, $idUsuario);

            // Verificar que el resultado sea 1 (indicando éxito en la actualización)
            $this->assertEquals(1, $result);
        }
```

testModificarDatoFallido	Verifica que el método modificarDato retorne 0 cuando falla la actualización de los datos de un usuario.

Codigo:

```
    public function testModificarDatoFallido()
        {
            // Datos simulados para modificar los datos de un usuario existente
```

```
            $usuario = 'usuario_modificado';
            $nombre = 'Usuario Modificado';
            $apellido = 'Apellido Modificado';
            $correo = 'modificado@example.com';
            $telefono = '987654321';
            $direccion = 'Calle Modificada';
            $perfil = 'imagen_modificada.jpg';
            $idUsuario = 999; // ID inexistente para forzar un fallo

            // Configuración de mock para el método save en la modificación de datos de usuario
            $this->mockQuery->shouldReceive('save')
                ->andReturn(0); // Simular que la actualización falló

            // Instanciar UsuariosModel y llamar al método modificarDato
            $usuariosModel = new UsuariosModel();
              $result = $usuariosModel->modificarDato($usuario, $nombre, $apellido, $correo,
    $telefono, $direccion, $perfil, $idUsuario);

            // Verificar que el resultado sea 0 (indicando fallo en la actualización)
            $this->assertEquals(0, $result);
        }
```

- Test de VehiculosModel.php

Titulo del Test	Descripción
testGetDatos	Verifica que el método getDatos de VehiculosModel devuelve correctamente los datos de una tabla específica.

Codigo:

```
    public function testGetDatos()
        {
            // Configurar el mock para simular la llamada a selectAll con la consulta correcta
            $this->mockQuery->shouldReceive('selectAll')
                            ->with('SELECT * FROM tabla WHERE estado = 1')
                            ->andReturn([
                                ['id' => 1, 'campo1' => 'valor1', 'estado' => 1],
                                ['id' => 2, 'campo1' => 'valor2', 'estado' => 1],
                            ]);

            $vehiculosModel = new VehiculosModel();
            // Llamar al método y almacenar el resultado
            $result = $vehiculosModel->getDatos('tabla');

            // Verificar que el resultado obtenido es el esperado
```

```
            $this->assertEquals([
                ['id' => 1, 'campo1' => 'valor1', 'estado' => 1],
                ['id' => 2, 'campo1' => 'valor2', 'estado' => 1],
            ], $result);
        }
```

testGetVehiculos	Verifica que el método getVehiculos de VehiculosModel devuelve correctamente los vehículos con un estado específico.

Codigo:

```
    public function testGetVehiculos()
        {
            // Estado de vehículos a buscar
            $estado = 1;

            // Configurar el mock para simular la llamada a selectAll con la consulta correcta
            $this->mockQuery->shouldReceive('selectAll')
                            ->with("SELECT v.*, m.marca, t.tipo FROM vehiculos v INNER JOIN
    marcas m ON v.id_marca = m.id INNER JOIN tipos t ON v.id_tipo = t.id WHERE v.estado =
    $estado")
                        ->andReturn([
                            ['id' => 1, 'placa' => 'ABC123', 'marca' => 'Marca A', 'tipo' =>
    'Tipo A', 'estado' => 1],
                            ['id' => 2, 'placa' => 'XYZ789', 'marca' => 'Marca B', 'tipo' =>
    'Tipo B', 'estado' => 1],
                        ]);

            $vehiculosModel = new VehiculosModel();

            // Llamar al método y almacenar el resultado
            $result = $vehiculosModel->getVehiculos($estado);

            // Verificar que el resultado obtenido es el esperado
            $this->assertEquals([
                 ['id' => 1, 'placa' => 'ABC123', 'marca' => 'Marca A', 'tipo' => 'Tipo A',
    'estado' => 1],
                 ['id' => 2, 'placa' => 'XYZ789', 'marca' => 'Marca B', 'tipo' => 'Tipo B',
    'estado' => 1],
            ], $result);
        }
```

testVehiculos	Verifica que el método vehiculos de VehiculosModel devuelve correctamente los vehículos con estado 1 o 2.

Codigo:

```
public function testVehiculos()
    {
        // Configurar el mock para simular la llamada a selectAll con la consulta correcta
        $this->mockQuery->shouldReceive('selectAll')
                        ->with("SELECT v.*, m.marca, t.tipo FROM vehiculos v INNER JOIN
marcas m ON v.id_marca = m.id INNER JOIN tipos t ON v.id_tipo = t.id WHERE v.estado = 1 OR
v.estado = 2")
                    ->andReturn([
                        ['id' => 1, 'placa' => 'ABC123', 'marca' => 'Marca A', 'tipo' =>
'Tipo A', 'estado' => 1],
                        ['id' => 2, 'placa' => 'XYZ789', 'marca' => 'Marca B', 'tipo' =>
'Tipo B', 'estado' => 2],
                    ]);

        $vehiculosModel = new VehiculosModel();

        // Llamar al método y almacenar el resultado
        $result = $vehiculosModel->vehiculos();

        // Verificar que el resultado obtenido es el esperado
        $this->assertEquals([
            ['id' => 1, 'placa' => 'ABC123', 'marca' => 'Marca A', 'tipo' => 'Tipo A',
'estado' => 1],
            ['id' => 2, 'placa' => 'XYZ789', 'marca' => 'Marca B', 'tipo' => 'Tipo B',
'estado' => 2],
        ], $result);
    }
```

testRegistrarVehiculo	Verifica que el método registrarVehiculo de VehiculosModel registra correctamente un vehículo nuevo y devuelve "ok".

Codigo:

```
public function testRegistrarVehiculo()
    {
        // Placa de vehículo que no existe
        $placaNueva = 'XYZ789';

        // Configurar el mock para que select devuelva que el vehículo no existe
        $this->mockQuery->shouldReceive('select')
                        ->with("SELECT * FROM vehiculos WHERE placa = '$placaNueva'")
```

```
                        ->andReturn([]); // Simular que el vehículo no existe

            // Configurar el mock para el método save en el registro de vehículo (caso exitoso)
            $this->mockQuery->shouldReceive('save')
                        ->andReturn(1); // Simular que la inserción fue exitosa

            $vehiculosModel = new VehiculosModel();

            // Llamar al método y almacenar el resultado
            $result = $vehiculosModel->registrarVehiculo(
               $placaNueva, 10, 100, 1000, 'Modelo B', 60000, 'Manual', 5, 'Mediano', 'Gasolina',
'imagen.jpg', 2, 2
           );
            // Verificar que el resultado retornado sea "ok"
            $this->assertEquals('ok', $result);
        }
```

testRegistrarVehiculoExistente	Verifica que el método registrarVehiculo de VehiculosModel devuelve "existe" cuando se intenta registrar un vehículo ya existente.

Codigo:

```
    public function testRegistrarVehiculoExistente()
        {
            // Placa de vehículo que ya existe
            $placaExistente = 'ABC123';

            // Configurar el mock para que select devuelva un vehículo existente
            $this->mockQuery->shouldReceive('select')
                        ->with("SELECT * FROM vehiculos WHERE placa = '$placaExistente'")
                        ->andReturn(['id' => 1, 'placa' => $placaExistente]); // Simular que
el vehículo ya existe

            $vehiculosModel = new VehiculosModel();

            // Llamar al método y almacenar el resultado
            $result = $vehiculosModel->registrarVehiculo(
                 $placaExistente, 10, 100, 1000, 'Modelo A', 50000, 'Automatica', 5, 'Grande',
'Gasolina', 'imagen.jpg', 1, 1
            );

            // Verificar que el resultado retornado sea "existe"
            $this->assertEquals('existe', $result);
        }
```

testRegistrarVehiculoError	Verifica que el método registrarVehiculo de VehiculosModel devuelve "error" cuando falla el registro de un vehículo nuevo.

Codigo:

```
public function testRegistrarVehiculoError()
    {
        // Placa de vehículo que no existe
        $placaNueva = 'XYZ789';

        // Configurar el mock para que select devuelva un vehículo que no existe
        $this->mockQuery->shouldReceive('select')
                        ->with("SELECT * FROM vehiculos WHERE placa = '$placaNueva'")
                        ->andReturn([]); // Simular que el vehículo no existe

        // Configurar el mock para que save devuelva 0, indicando error en la inserción
        $this->mockQuery->shouldReceive('save')
                        ->andReturn(0); // Simular que la inserción falla

        $vehiculosModel = new VehiculosModel();

        // Llamar al método y almacenar el resultado
        $result = $vehiculosModel->registrarVehiculo(
           $placaNueva, 10, 100, 1000, 'Modelo B', 60000, 'Manual', 4, 'Mediano', 'Gasolina',
'imagen.jpg', 2, 2
        );

        // Verificar que el resultado retornado sea "error"
        $this->assertEquals('error', $result);
    }
```

testModificarVehiculo	Verifica que el método modificarVehiculo de VehiculosModel actualiza correctamente los datos de un vehículo y devuelve "modificado".

Codigo:

```
public function testModificarVehiculo()
    {
        $placa = 'ABC123';
        $precio_hora = 15;
        $precio_dia = 150;
        $precio_mes = 1500;
        $modelo = 'Modelo Actualizado';
        $kilometraje = 60000;
        $transmision = 'Manual';
        $asientos = 4;
```

```
            $equipaje = 'Mediano';
            $combustible = 'Gasolina';
            $imgNombre = 'imagen_actualizada.jpg';
            $tipo = 2;
            $marca = 2;
            $id = 1;
            // Configurar mock para el método save en la modificación de vehículo
            $this->mockQuery->shouldReceive('save')
                            ->andReturn(1); // Simular que la actualización fue exitosa

            $vehiculosModel = new VehiculosModel();
              // Llamar al método y almacenar el resultado
            $result = $vehiculosModel->modificarVehiculo(
                    $placa, $precio_hora, $precio_dia, $precio_mes, $modelo, $kilometraje,
    $transmision,
                $asientos, $equipaje, $combustible, $imgNombre, $tipo, $marca, $id
            );
            // Verificar que el resultado retornado sea "modificado"
            $this->assertEquals('modificado', $result);
        }
```

testModificarVehiculoError	Verifica que el método modificarVehiculo de VehiculosModel devuelve "error" cuando falla la actualización de un vehículo.

Codigo:

```
    public function testModificarVehiculoError()
        {
            // Datos del vehículo a modificar
            $placa = 'ABC123';
            $precio_hora = 10;
            $precio_dia = 100;
            $precio_mes = 1000;
            $modelo = 'Modelo A';
            $kilometraje = 50000;
            $transmision = 'Automatica';
            $asientos = 5;
            $equipaje = 'Grande';
            $combustible = 'Gasolina';
            $imgNombre = 'imagen.jpg';
            $tipo = 1;
            $marca = 1;
            $id = 1;

            // Configurar el mock para simular la llamada a save con el SQL y los datos
```

```
            $this->mockQuery->shouldReceive('save')
                            ->andReturn(0); // Simular que la modificación falla y no se realiza

            $vehiculosModel = new VehiculosModel();

            // Llamar al método y almacenar el resultado
            $result = $vehiculosModel->modificarVehiculo(
                $placa, $precio_hora, $precio_dia, $precio_mes,
                $modelo, $kilometraje, $transmision, $asientos, $equipaje,
                $combustible, $imgNombre, $tipo, $marca, $id
            );
            // Verificar que el resultado retornado es "error"
            $this->assertEquals('error', $result);
        }
```

testEditarVehiculo	Verifica que el método editarVeh de VehiculosModel devuelve correctamente los datos de un vehículo específico.

Codigo:

```
    public function testEditarVehiculo()
        {
            $idVehiculo = 1;

            // Datos simulados del vehículo que se espera retornar
            $mockedVehiculo = [
                'id' => $idVehiculo,
                'placa' => 'ABC123',
                'precio_hora' => 10,
                'precio_dia' => 100,
                'precio_mes' => 1000,
                'modelo' => 'Modelo A',
                'kilometraje' => 50000,
                'transmision' => 'Automatica',
                'asientos' => 5,
                'equipaje' => 'Grande',
                'combustible' => 'Gasolina',
                'foto' => 'imagen.jpg',
                'id_tipo' => 1,
                'id_marca' => 1,
                'estado' => 1,
            ];

            // Configuración de mock para el método select en el modelo
            $sql = "SELECT * FROM vehiculos WHERE id = $idVehiculo";
```

```
            $this->mockQuery->shouldReceive('select')
                            ->with($sql)
                            ->andReturn($mockedVehiculo);

            $vehiculosModel = new VehiculosModel();

            // Llamar al método y almacenar el resultado
            $result = $vehiculosModel->editarVeh($idVehiculo);

            // Verificar que el resultado coincida con los datos esperados
            $this->assertEquals($mockedVehiculo, $result);
        }
```

testAccionVehiculo	Verifica que el método accionVeh de VehiculosModel actualiza correctamente el estado de un vehículo y devuelve 1.

Codigo:

```
    public function testAccionVehiculo()
        {
            $estado = 0;
            $idVehiculo = 1;

            // Configurar mock para el método save en la acción de vehículo
            $this->mockQuery->shouldReceive('save')
                            ->andReturn(1); // Simular que la actualización fue exitosa

            $vehiculosModel = new VehiculosModel();

            // Llamar al método y almacenar el resultado
            $result = $vehiculosModel->accionVeh($estado, $idVehiculo);

            // Verificar que el resultado sea 1 (éxito en la acción)
            $this->assertEquals(1, $result);
        }
```

testBuscarVehiculo	Verifica que el método buscarVehiculo de VehiculosModel devuelve correctamente los vehículos que coinciden con un valor buscado.

Codigo:

```
    public function testBuscarVehiculo()
        {
            $valor = 'ABC';

            // Datos simulados de vehículos que se esperan retornar
```

```
            $mockedVehiculos = [
                ['id' => 1, 'placa' => 'ABC123', 'tipo' => 'SUV', 'marca' => 'Toyota', 'estado'
    => 1],
                ['id' => 2, 'placa' => 'XYZ789', 'tipo' => 'Sedan', 'marca' => 'Honda', 'estado'
    => 1],
            ];

            // Configuración del mock para el método selectAll en el modelo
            $this->mockQuery->shouldReceive('selectAll')
                            ->once() // Asegura que se llame exactamente una vez
                            ->andReturnUsing(function ($sql) use ($valor, $mockedVehiculos) {
                                // Verificar que la consulta contenga el valor buscado
                                if (strpos($sql, $valor) !== false) {
                                    return $mockedVehiculos;
                                }
                                return [];
                            });
            // Instanciamos el modelo de vehículos
            $vehiculosModel = new VehiculosModel();

            // Llamamos al método buscarVehiculo y almacenamos el resultado
            $result = $vehiculosModel->buscarVehiculo($valor);

            // Verificamos que el resultado coincida con los datos esperados
            $this->assertEquals($mockedVehiculos, $result);
        }
```

7. Reporte de Pruebas guiadas por el comportamiento (BDD Given When Then)

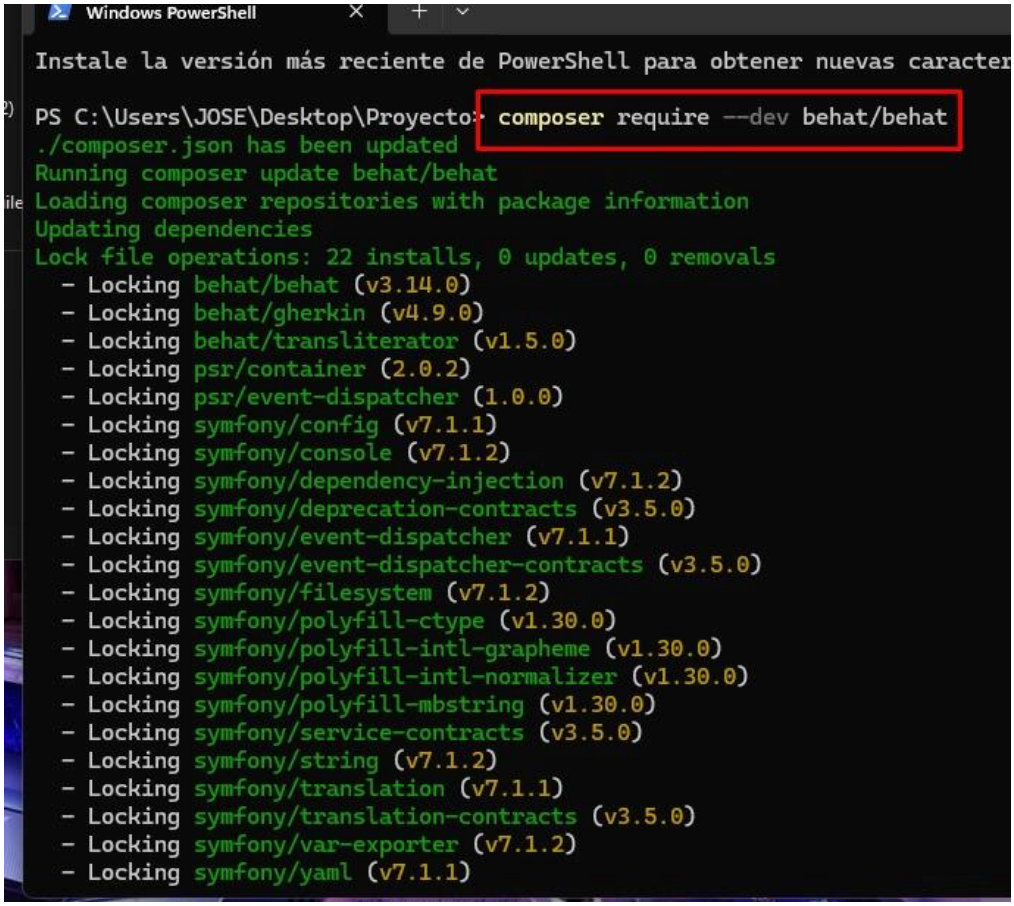
El enfoque de pruebas guiadas por el comportamiento (Behavior Driven Development, BDD) se basa en describir el comportamiento esperado de un sistema utilizando un lenguaje común que sea comprensible tanto para los desarrolladores como para los stakeholders no técnicos. Una de las estructuras más comunes en BDD es el formato Given-When-Then:

Given (Dado)	Describe el contexto inicial del escenario de prueba.
When (Cuando)	Describe el evento o acción que se ejecuta.
Then (Entonces)	Describe el resultado esperado después de la acción.

Este formato ayuda a establecer criterios de aceptación claros y facilita la comunicación entre los diferentes miembros del equipo. Los reportes generados mediante esta metodología permiten una mejor comprensión de los requisitos y su correcta implementación.

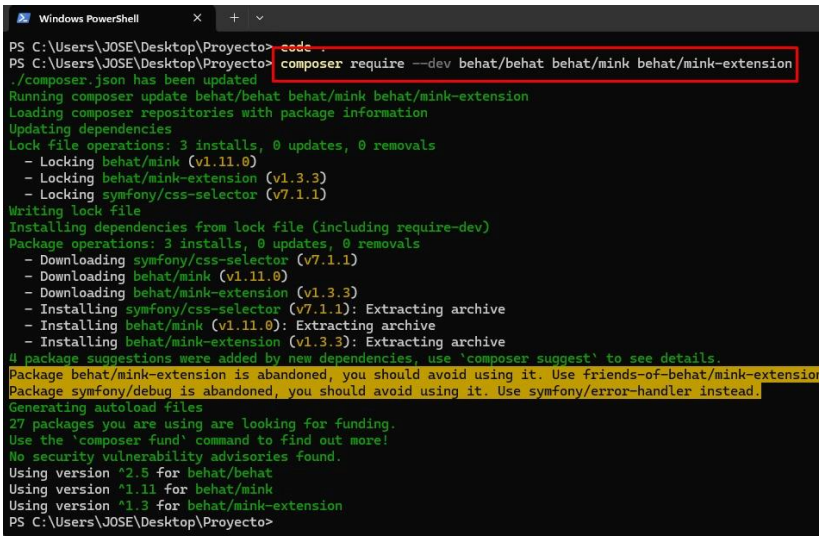
- Instalación de Pruebas Behat

  El comando **composer require --dev behat/behat** indica que se está instalando la biblioteca de pruebas Behat para un proyecto PHP, con la opción **--dev** que la configura como una dependencia de desarrollo.



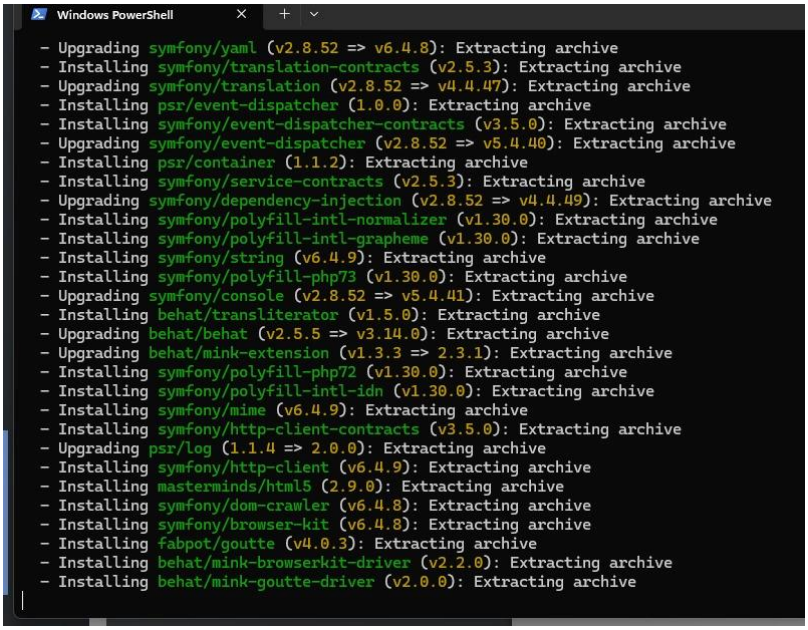
- Instalar las dependencias para el Proyecto

  Se está ejecutando el comando **composer require** para instalar las dependencias de un proyecto de desarrollo. Este comando especifica las dependencias que se necesitan para el proyecto, que en este caso son las librerías **behat/behat**, **behat/mink** y **behat/mink-extension**.

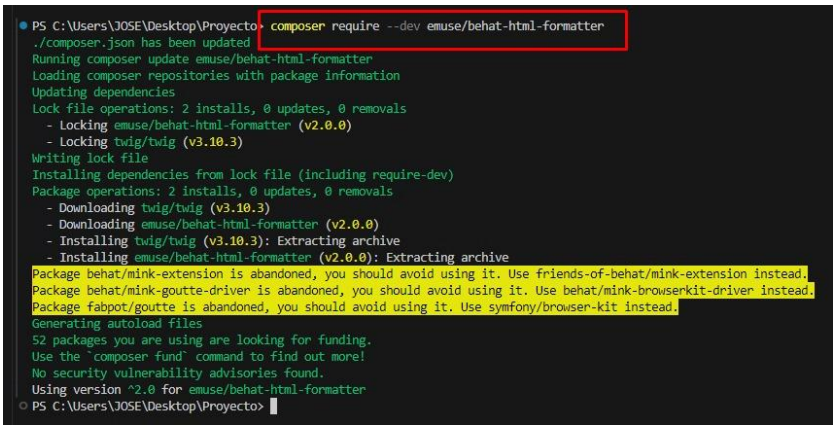


- Se esta Actualizando y instalando “Symfony”

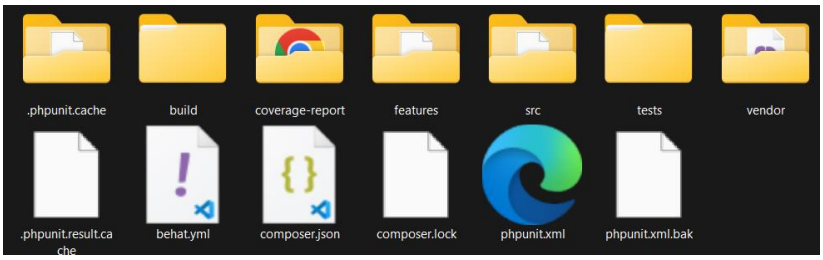
  Se observa que la mayoría de los comandos están relacionados con la librería Symfony, una popular herramienta para el desarrollo web con PHP, con comandos como "Upgrading symfony/yaml" o "Installing symfony/translation-contracts", lo que sugiere que se está actualizando o instalando un proyecto que utiliza Symfony.



- Se usa el comando emuse/behat
  Un comando de consola que está actualizando un proyecto de software usando Composer, una herramienta para gestionar dependencias de proyectos. El comando instala dos paquetes y luego informa que algunos paquetes que se usaban en el proyecto están desactualizados y que deberían ser reemplazados por otros.

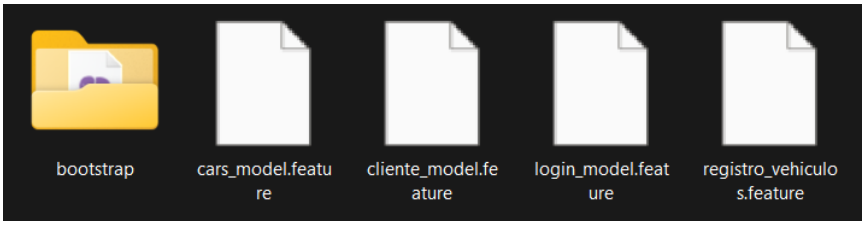


- **Test Para los Escenarios de "features"**

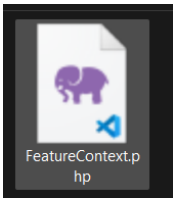


Creación de Features

- En este punto se usa el GIVE WHEN THEN



Carpeta del "bootstrap"



- Escenario para cars_model.feature

CONSULTA DE VEHÍCULOS Y DETALLES DE VEHÍCULO EN EL SISTEMA
Scenario: Consultar lista de vehículos activos
Given que tengo vehículos en el sistema con estado "activo" When consulto la lista de vehículos Then debería obtener una lista de vehículos con estado "activo"
Scenario: Consultar detalles de un vehículo específico
Given que tengo un vehículo con ID "1" When consulto los detalles del vehículo con ID "1" Then debería ver los detalles del vehículo con ID "1"

- Escenario para cliente_model.feature

REGISTRO DE CLIENTE Y VERIFICAR EL PROCESO DE REGISTRO DE UN NUEVO CLIENTE EN EL SISTEMA
Scenario: Registro exitoso de un nuevo cliente
Given que tengo acceso al sistema como administrador When registro un nuevo cliente con los siguientes datos: | DNI | Nombre | Teléfono | Dirección | | 12345678A | Juan Pérez | 987654321 | Calle Principal, 123 | Then debería ver el mensaje "Cliente registrado correctamente."
Scenario: Intento de registro de un cliente que ya existe
Given que tengo acceso al sistema como administrador

When intento registrar un nuevo cliente con los siguientes datos: | DNI | Nombre | Teléfono | Dirección | | 12345678A | Juan Pérez | 987654321 | Calle Principal, 123 | Then debería ver el mensaje "Ya existe un cliente con ese nombre."
Scenario: Falla en el registro de un nuevo cliente
Given que tengo acceso al sistema como administrador When intento registrar un nuevo cliente con datos incorrectos Then debería ver el mensaje "Ocurrió un error al intentar registrar al cliente."

- Escenario para login_model.feature

VERIFICACIÓN DE INICIO DE SESIÓN EN EL SISTEMA
Scenario: Verificar inicio de sesión exitoso con correo electrónico en tabla de usuarios
Given que tengo un usuario con correo "john@example.com" en la tabla "usuarios" When verifico el inicio de sesión con correo electrónico "john@example.com" y contraseña "hashed_password" Then debería obtener un resultado exitoso de verificación de inicio de sesión
Scenario: Verificar inicio de sesión exitoso con correo electrónico en tabla de usuarios
Given que tengo un usuario con correo "admin@example.com" en la tabla "usuarios"

When verifico el inicio de sesión con correo electrónico "admin@example.com" y contraseña "hashed_password" Then debería obtener un resultado exitoso de verificación de inicio de sesión
Scenario: Verificar inicio de sesión fallido con contraseña incorrecta para usuario admin
Given que tengo un usuario con correo "admin@example.com" en la tabla "usuarios" When verifico el inicio de sesión con correo electrónico "admin@example.com" y contraseña "hashed" Then debería obtener un resultado fallido de verificación de inicio de sesión
Scenario: Verificar inicio de sesión fallido con contraseña incorrecta para usuario ana
Given que tengo un usuario con correo "ana@example.com" en la tabla "usuarios" When verifico el inicio de sesión con correo electrónico "ana@example.com" y contraseña "hashed" Then debería obtener un resultado fallido de verificación de inicio de sesión

- Escenario para registro_vehiculos.feature

VERIFICAR EL REGISTRO, MODIFICACIÓN Y BÚSQUEDA DE VEHÍCULOS EN EL SISTEMA.
Scenario: Registrar un nuevo vehículo exitosamente
When registro un nuevo vehículo con los siguientes datos: | Placa | Precio Hora | Precio Día | Precio Mes | Modelo | Kilometraje | Transmisión | Asientos | Equipaje | Combustible | Imagen | Tipo | Marca | | ABC123 | 10 | 50 | 300 | Modelo1 | 10000 | Automática | 5 | Grande | Gasolina | vehiculo.jpg | 1 | 1 | Then debería ver el mensaje "Vehículo registrado correctamente."
Scenario: Intento de registrar un vehículo que ya existe

When intento registrar un nuevo vehículo con los siguientes datos:

Placa	Precio Hora	Precio Día	Precio Mes	Modelo	Kilometraje	Transmisión	Asientos	Equipaje	Combustible	Imagen	Tipo	Marca
XYZ789	15	60	350	Modelo2	8000	Manual	4	Mediano	Diésel	carro.jpg	2	2

Then debería ver el mensaje "Ya existe un vehículo con esa placa."

Scenario: Falla en el registro de un nuevo vehículo

When intento registrar un nuevo vehículo con datos incorrectos

Then debería ver el mensaje "Ocurrió un error al intentar registrar el vehículo."

Scenario: Modificar un vehículo existente

Given existe un vehículo con id 1

When modifico el vehículo con id 1 con los siguientes datos:

Placa	Precio Hora	Precio Día	Precio Mes	Modelo	Kilometraje	Transmisión	Asientos	Equipaje	Combustible	Imagen	Tipo	Marca
ABC123	12	55	320	Modelo1	12000	Automática	5	Grande	Gasolina	vehiculo.jpg	1	1

Then debería ver el mensaje "Vehículo modificado correctamente."

Scenario: Falla al intentar modificar un vehículo

Given existe un vehículo con id 2

When intento modificar el vehículo con id 2 con datos incorrectos

Then debería ver el mensaje "Ocurrió un error al intentar modificar el vehículo."

Scenario: Buscar vehículo por placa, tipo o marca

Given existen vehículos registrados en el sistema

When busco vehículos con el valor "ABC" en la búsqueda

Then debería ver resultados de búsqueda con vehículos que coincidan con "ABC"

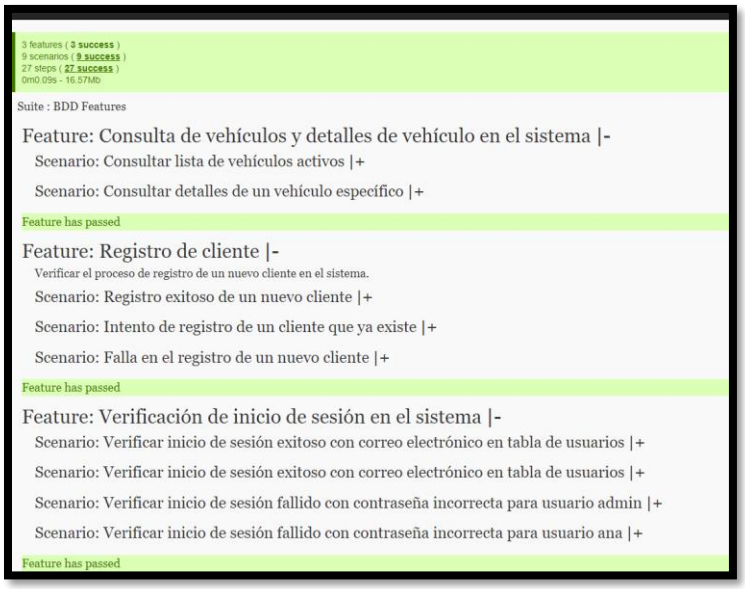
Scenario: Cambiar estado de un vehículo

```
Given existe un vehículo con id 1
When cambio el estado del vehículo con id 1 a "2"
Then debería ver el mensaje "Estado del vehículo cambiado correctamente."
```

- Análisis de los Feature
  - **Feature: Consulta de vehículos y detalles de vehículo en el sistema**
    El sistema permite consultar la lista de vehículos activos.
    El sistema muestra los detalles de un vehículo específico cuando se solicita.

  - **Feature: Registro de cliente**
    El sistema permite registrar un nuevo cliente de manera exitosa.
    El sistema no permite registrar un cliente que ya existe en la base de datos.
    El sistema maneja errores al intentar registrar un nuevo cliente, por ejemplo, cuando falta algún dato obligatorio.

  - **Feature: Verificación de inicio de sesión en el sistema**
    El sistema permite iniciar sesión con éxito utilizando un correo electrónico válido en la tabla de usuarios.
    El sistema también permite iniciar sesión con éxito utilizando un correo electrónico válido en la tabla de usuarios (se repite el escenario).
    El sistema no permite iniciar sesión con una contraseña incorrecta para el usuario administrador.
    El sistema no permite iniciar sesión con una contraseña incorrecta para el usuario Ana.



- **En este punto podemos ver lo que logro pasar**

Se muestran un resumen de pruebas de software realizadas. Se muestra que se han ejecutado tres características diferentes: la consulta de vehículos, el registro de clientes y la verificación de inicio de sesión. Se ha realizado un total de 21 pasos de prueba, 7 escenarios y 3 características.

8. Cronograma

Id	herramientas	Tarea	responsable	aprobado	22	23	24	25	26	27	28	29	30	1	2	3	4	comentarios
1	PHPUnit	Detectar los problemas de cobertura	BD	SI				8am-10am	8am-10am	9am-10am		8am-10am	8am-10am	8am-10am			9am-10am	Los primeros días fue investigar cómo descargar sonarqube y luego realizar análisis con PHPUnit
2	SonarQube	Detectar las vulnerabilidades sonarqube, solucionar vulnerabilidades	LC	SI	8am-10:30am									9am-10am		8am-10am		Solucionamos la vulnerabilidad y los últimos días bajó el porcentaje de código duplicado
3	Snyk	Detectar las vulnerabilidades Snyk, solucionar vulnerabilidades	LC	SI						8am-10:30am				8am-10am				Se realizó el análisis en paralelo pues hay diferencias con sonarqube y Snyk. La nube pasó para mostrar en el aula
4	Snyk	Investigar sobre vulnerabilidades	BD	SI	8am-10:30am						10pm-5am		8am-10:30am	8am-10:30am	8am-10:30am			Se tomó como fuente principal de análisis, el uso de herramientas de GitHub
5	documentación	Diagramas	CC	SI					8am-10:30am					9am-10am		8am-10am		Se encarga de desarrollar la introducción del proyecto y otra parte del documento
6	documentación	Diagramas	CG	SI							8am-10:30am	8am-10:30am	8am-10:30am	8am-10:30am	1pm-2pm			Se encarga de los diagramas de clase y otros diagramas

RESPONSABLE	Iniciales
Lopez Catunta, Brayar Christian	LC
Briceño Diaz, Jorge Luis	BD
Cuadros Garcia, Mirian	CG
Castañeda Centurión, Jorge Enriqu	CC