

Realiza plan pruebas de software GA9-220501096-AA1-EV02

Presentado por:

Ana Karina Yepes Gómez c.c. 43612987
Jheison Fernando Rojas Rendon c.c. 1053846295
Adan de Jesús Restrepo Zapata c.c. 71290603
Jesús Harvey Bernal López. c.c. 1069725992

Presentado a:

Carlos Fuel Tulcan

Ficha: 2627092

TECNOLOGÍA EN ANÁLISIS Y DESARROLLO DE SOFTWARE
SENA REGIONAL QUINDÍO.
2024

Tabla de contenido

1. INTRODUCCIÓN	3
2. Evidencia de conocimiento: GA9-220501096-AA1-EV01 taller sobre codificación de módulos del software.....	4
2.1. ¿Qué tipos de pruebas de software existen? Explique sus características y beneficios.	4
2.1.1. Pruebas Unitarias (Unit Testing)	4
2.1.2. Pruebas de Integración (Integration Testing)	4
2.1.3. Pruebas de Sistema (System Testing)	4
3.1.4. Pruebas de Aceptación (Acceptance Testing)	5
3.1.5. Pruebas de Regresión (Regression Testing)	5
3.1.6. Pruebas de Rendimiento (Performance Testing)	5
3.1.7. Pruebas de Seguridad (Security Testing)	5
3.1.8. Pruebas de Usabilidad (Usability Testing)	6
3.1.9. Pruebas de Compatibilidad (Compatibility Testing)	6
3.2. Según la consulta que realizó, ¿qué tipos de pruebas se adaptan mejor al proyecto de software que está desarrollando?	6
3.2.1. Detección Temprana de Errores	7
3.2.2. Mejora de la Calidad del Código	7
3.2.3. Facilitan el Refactorización del Código	7
3.2.4. Documentación del Código.....	7
3.2.5. Automatización y Eficiencia	7
3.2.6. Reducción de Defectos en Producción	8
3.2.7. Facilitan el Desarrollo Colaborativo	8
3.2.8. Investigue e instale unas herramientas de pruebas de software en su computador, “una de su gusto”	8
1. Conclusiones.....	15

1. INTRODUCCIÓN

En el ámbito del desarrollo de software, la calidad es un factor crucial que influye directamente en la satisfacción del cliente y en la eficacia del producto final. Una de las estrategias fundamentales para garantizar esta calidad es la realización de pruebas de software, un proceso que abarca diversas metodologías y enfoques para detectar y corregir errores o defectos en el software antes de su implementación definitiva.

Existen varios tipos de pruebas de software, cada una con sus propias características y beneficios. Estas pruebas pueden incluir desde la verificación de requisitos funcionales hasta la evaluación de la seguridad y el rendimiento del sistema. Al comprender las diferencias entre estos tipos de pruebas, los equipos de desarrollo pueden diseñar estrategias más efectivas para garantizar la calidad del software.

2. Evidencia de conocimiento: GA9-220501096-AA1-EV01 taller sobre codificación de módulos del software

2.1. ¿Qué tipos de pruebas de software existen? Explique sus características y beneficios.

Las pruebas de software son esenciales para garantizar que una aplicación o sistema funcione correctamente y cumpla con los requisitos especificados. Existen varios tipos de pruebas de software, cada una con sus características y beneficios específicos. A continuación, se describen los tipos principales:

2.1.1. Pruebas Unitarias (Unit Testing)

Características:

Nivel de Código: Se enfocan en probar unidades individuales de código, como funciones o métodos.

Automatización: Generalmente son automatizadas y se ejecutan rápidamente.

Aislamiento: Las unidades se prueban en aislamiento del resto del sistema.

Beneficios:

Detección Temprana de Errores: Permiten identificar y corregir errores en una etapa temprana del desarrollo.

Facilidad de Mantenimiento: Mejoran la mantenibilidad del código al garantizar que las unidades funcionan correctamente de manera independiente.

Documentación: Sirven como documentación del comportamiento esperado del código.

2.1.2. Pruebas de Integración (Integration Testing)

Características:

Combinación de Unidades: Prueban la interacción entre diferentes unidades o módulos de código.

Interfaces y Comunicación: Se centran en verificar que los módulos se comuniquen correctamente entre sí.

Tipos: Pueden ser incrementales (agregando módulos gradualmente) o de "Big Bang" (todos los módulos se integran al mismo tiempo).

Beneficios:

Detección de Problemas de Interfaz: Identifican problemas en las interfaces y en la interacción entre módulos.

Verificación de Comportamiento Conjunto: Aseguran que el sistema funcione como un todo integrado.

2.1.3. Pruebas de Sistema (System Testing)

Características:

Sistema Completo: Evalúan el sistema completo integrado para verificar que cumpla con los requisitos especificados.

Entorno Realista: Se ejecutan en un entorno que simula el entorno de producción.

Pruebas Funcionales y No Funcionales: Incluyen tanto pruebas funcionales (verificación de

funcionalidades) como no funcionales (rendimiento, seguridad, etc.).

Beneficios:

Validación Completa: Proveen una validación completa del sistema frente a los requisitos iniciales.

Identificación de Defectos: Permiten identificar defectos que no se detectaron en pruebas unitarias o de integración.

3.1.4. Pruebas de Aceptación (Acceptance Testing)

Características:

Verificación por el Cliente: Son realizadas por el cliente o el usuario final para verificar que el sistema cumpla con sus expectativas y requisitos.

Criterios de Aceptación: Se basan en criterios de aceptación definidos previamente.

Beneficios:

Aprobación Final: Aseguran que el producto cumple con los requisitos del cliente antes de su implementación.

Satisfacción del Cliente: Mejoran la satisfacción del cliente al garantizar que el sistema funciona según sus expectativas.

3.1.5. Pruebas de Regresión (Regression Testing)

Características:

Cambios en el Software: Se ejecutan después de realizar cambios en el software, como correcciones de errores o nuevas funcionalidades.

Verificación Continua: Aseguran que las modificaciones no hayan introducido nuevos errores.

Beneficios:

Estabilidad del Sistema: Mantienen la estabilidad del sistema al verificar que los cambios no afecten negativamente el funcionamiento existente.

Confianza en el Desarrollo: Aumentan la confianza en el proceso de desarrollo continuo y en las actualizaciones del software.

3.1.6. Pruebas de Rendimiento (Performance Testing)

Características:

Rendimiento y Escalabilidad: Evalúan el rendimiento del sistema bajo diferentes cargas y condiciones de estrés.

Subtipos: Incluyen pruebas de carga, pruebas de estrés, pruebas de escalabilidad y pruebas de estabilidad.

Beneficios:

Optimización del Rendimiento: Ayudan a identificar cuellos de botella y optimizar el rendimiento del sistema.

Preparación para la Escalabilidad: Aseguran que el sistema pueda manejar el crecimiento en el uso y la carga.

3.1.7. Pruebas de Seguridad (Security Testing)

Características:

Vulnerabilidades: Se enfocan en identificar vulnerabilidades de seguridad en el sistema.

Amenazas y Ataques: Simulan posibles ataques para evaluar la robustez de las medidas de seguridad implementadas.

Beneficios:

Protección de Datos: Garantizan la protección de datos sensibles y la integridad del sistema.

Cumplimiento Normativo: Aseguran que el sistema cumpla con las normativas y estándares de seguridad.

3.1.8. Pruebas de Usabilidad (Usability Testing)

Características:

Experiencia del Usuario: Evalúan la facilidad de uso y la experiencia del usuario con el sistema.

Interacción del Usuario: Involucran a usuarios reales para probar la interfaz y la funcionalidad desde una perspectiva de usuario.

Beneficios:

Mejora de la UX: Identifican problemas de usabilidad que pueden ser corregidos para mejorar la experiencia del usuario.

Aumento de la Satisfacción: Aumentan la satisfacción del usuario final al garantizar una interfaz intuitiva y fácil de usar.

3.1.9. Pruebas de Compatibilidad (Compatibility Testing)

Características:

Entornos Diferentes: Verifican que el software funcione correctamente en diferentes entornos, sistemas operativos, navegadores, dispositivos, etc.

Variabilidad de Configuraciones: Aseguran que el sistema sea compatible con diversas configuraciones de hardware y software.

Beneficios:

Cobertura Amplia: Aseguran que el software funcione correctamente para todos los usuarios, independientemente de su entorno.

Reducción de Problemas Post-Implementación: Minimizan los problemas que pueden surgir después de la implementación debido a incompatibilidades.

Conclusión

Cada tipo de prueba de software tiene su propio conjunto de características y beneficios, y todas son esenciales para garantizar la calidad y la fiabilidad de un sistema de software. La combinación adecuada de estos tipos de pruebas puede ayudar a detectar y corregir errores en diferentes etapas del desarrollo, mejorando así la estabilidad y la funcionalidad del producto final.

3.2. Según la consulta que realizó, ¿qué tipos de pruebas se adaptan mejor al proyecto de software que está desarrollando?

El tipo de pruebas mas indicado en nuestro desarrollo son las Pruebas Unitarias (Unit Testing). Las pruebas unitarias son una parte fundamental del proceso de desarrollo de software y ofrecen numerosas ventajas. A continuación, se describen algunas de las principales ventajas de realizar pruebas unitarias:

3.2.1. Detección Temprana de Errores

Ventaja:

Identificación Precoz: Las pruebas unitarias permiten detectar errores en una etapa temprana del desarrollo, cuando son más fáciles y menos costosos de corregir.

Beneficio:

Reducción de Costos: Al corregir errores temprano, se evita la acumulación de problemas que pueden volverse más complejos y costosos de solucionar en etapas posteriores.

3.2.2. Mejora de la Calidad del Código

Ventaja:

Aseguramiento de Funcionalidad: Las pruebas unitarias garantizan que cada unidad de código funcione según lo esperado.

Código Limpio: Fomentan la escritura de código modular y bien estructurado, ya que las unidades deben ser probadas de forma aislada.

Beneficio:

Fiabilidad: Aumentan la fiabilidad del software al asegurar que las unidades individuales funcionen correctamente.

Mantenibilidad: Facilitan la mantenibilidad del código al hacer más sencillo identificar y corregir errores en unidades específicas.

3.2.3. Facilitan el Refactorización del Código

Ventaja:

Seguridad en Cambios: Proporcionan una base segura para realizar cambios en el código, ya que cualquier problema introducido por la refactorización será detectado por las pruebas unitarias.

Beneficio:

Evolución del Código: Permiten mejorar y evolucionar el código con confianza, asegurando que las nuevas implementaciones no afecten negativamente el comportamiento existente.

3.2.4. Documentación del Código

Ventaja:

Ejemplos Prácticos: Las pruebas unitarias actúan como documentación viva del comportamiento esperado del código.

Beneficio:

Claridad: Ayudan a nuevos desarrolladores a entender cómo se supone que debe funcionar el código, proporcionando ejemplos concretos de uso.

3.2.5. Automatización y Eficiencia

Ventaja:

Automatización de Pruebas: Las pruebas unitarias se pueden automatizar y ejecutarse rápidamente como parte del proceso de integración continua.

Beneficio:

Eficiencia: Mejoran la eficiencia del desarrollo al permitir pruebas rápidas y repetitivas sin intervención manual.

Integración Continua: Facilitan la integración continua al permitir la ejecución automática de pruebas cada vez que se realiza un cambio en el código.

3.2.6. Reducción de Defectos en Producción

Ventaja:

Cobertura de Pruebas: Al probar todas las unidades individuales, se reduce la probabilidad de que errores lleguen a producción.

Beneficio:

Confiabilidad del Producto: Aumentan la confiabilidad del producto final y reducen el riesgo de fallos críticos en producción.

Satisfacción del Cliente: Mejoran la satisfacción del cliente al entregar un producto más robusto y confiable.

3.2.7. Facilitan el Desarrollo Colaborativo

Ventaja:

Consistencia del Código: Ayudan a mantener la consistencia del código cuando múltiples desarrolladores trabajan en el mismo proyecto, ya que las pruebas unitarias pueden verificar que los cambios de diferentes personas no interfieran entre sí.

Beneficio:

Coordinación: Facilitan la coordinación y colaboración entre los miembros del equipo de desarrollo.
Prevención de Conflictos: Disminuyen los conflictos al proporcionar una forma clara de verificar que los cambios no rompan el código existente.

Conclusión

Las pruebas unitarias son esenciales para el desarrollo de software de alta calidad. Ofrecen ventajas significativas en términos de detección temprana de errores, mejora de la calidad y mantenibilidad del código, facilitación de refactorización, documentación, automatización, reducción de defectos en producción y apoyo al desarrollo colaborativo. Al implementar pruebas unitarias de manera consistente, los equipos de desarrollo pueden crear software más confiable, eficiente y fácil de mantener, lo que conduce a una mayor satisfacción del cliente y una reducción de costos a largo plazo.

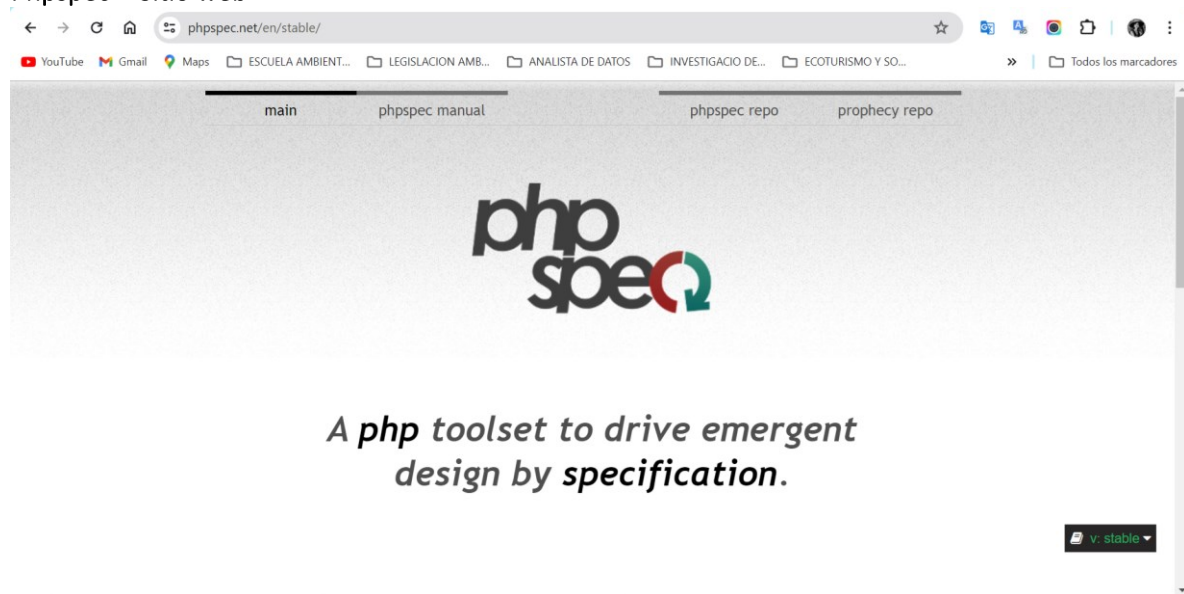
3.2.8. Investigue e instale unas herramientas de pruebas de software en su computador, “una de su gusto”. Con la herramienta instalada, realice unas pruebas básicas de su solución de software, tome capturas de pantalla del proceso y anéxelas al trabajo.

PHPSpec (para pruebas unitarias y de comportamiento)

Investigación

PHPSpec es otra herramienta de pruebas orientada a BDD, diseñada para especificar cómo debería comportarse el código.

Phpspec - Sitio web



Introducción phpspec



Instalación del phpspec



The screenshot shows the 'manual de phpspec' page on the phpspec.net website. The page is titled 'Instalación' and provides instructions on how to install phpspec. It mentions that phpspec is a PHP 5.6+ library and that users should ensure they have PHP 5.6 or 7 installed. The 'Proceso de instalación' section explains that phpspec can be installed via Composer and provides a link to the Composer website. It also notes that the 'autoload' configuration in the 'composer.json' file is crucial for phpspec to detect classes. A sidebar on the right lists the 'CAPÍTULOS' (Chapters) of the manual, including 'Introducción', 'Instalación', 'Empezando', 'Objetos Profetas', 'Deja y deja ir', and 'Actualización a PhpSpec 4'. The 'Instalación' chapter is currently selected.

principal manual de phpspec repositorio phpspec repositorio de profecía

Instalación

phpspec es una biblioteca php 5.6+ que tendrás en el entorno de desarrollo de tu proyecto. Antes de comenzar, asegúrese de tener PHP 5.6 o 7 instalado.

Proceso de instalación:

Puedes instalar phpspec con todas sus dependencias a través de Composer. Siga las instrucciones en [el sitio web del compositor](#) si aún no lo tiene instalado.

Nb: deberá asegurarse de que la `autoload` configuración de Composer sea correcta. phpspec no podrá detectar clases, ni siquiera las que haya creado, a menos que esto esté funcionando. Este es un problema común que causa confusión al instalar phpspec.

La `autoload` sección de su `composer.json` archivo puede verse así:

CAPÍTULOS

- Introducción
- Instalación
- 1. Proceso de instalación:
- 2. Método n.º 1 (comando del compositor):
- 3. Método n.º 2 (archivo de configuración de Composer):
- 4. Resultado:
- Empezando
- Objetos Profetas
- Deja y deja ir
- Actualización a PhpSpec 4

Ejecutando phpspec



The screenshot shows the 'manual de phpspec' page on the phpspec.net website, specifically the 'ejecutando phpspec' section. It explains that the phpspec command-line tool uses the Symfony console component. It provides two methods to run phpspec: using the default configuration or specifying a custom configuration file. The first method is shown as a terminal command: `$ bin/phpspec run --config path/to/different-phpspec.yml`. The second method is shown as a terminal command: `$ bin/phpspec run -c path/to/different-phpspec.yml`. A sidebar on the right lists the 'CAPÍTULOS' (Chapters) of the manual, including 'Introducción', 'Instalación', 'Empezando', 'Objetos Profetas', 'Deja y deja ir', 'Actualización a PhpSpec 4', 'Actualizando a phpspec 3', 'Configuración', 'ejecutando phpspec', and 'v. estable'. The 'ejecutando phpspec' chapter is currently selected.

principal manual de phpspec repositorio phpspec repositorio de profecía

ejecutando phpspec

El comando de consola phpspec utiliza el componente de consola de Symfony. Esto significa que hereda el [comando y las opciones predeterminadas de la consola Symfony](#).

phpspec tiene una opción global adicional que le permite especificar un archivo de configuración distinto de `phpspec.yml`, `.phpspec.yml` o `phpspec.yml.dist`:

```
$ bin/phpspec run --config path/to/different-phpspec.yml
```

o:

```
$ bin/phpspec run -c path/to/different-phpspec.yml
```

Lea más sobre esto en el [Libro de recetas de configuración](#).

CAPÍTULOS

- Introducción
- Instalación
- Empezando
- Objetos Profetas
- Deja y deja ir
- Actualización a PhpSpec 4
- Actualizando a phpspec 3
- Configuración
- ejecutando phpspec
- 1. Describir comando
- 2. Ejecutar comando

1. Instalar PHPSpec usando Composer:

```
bash
```

[Copiar código](#)

```
composer require --dev phpspec/phpspec
```

2. Inicializar PHPSpec en tu proyecto:

```
bash
```

[Copiar código](#)

```
./vendor/bin/phpspec --init
```

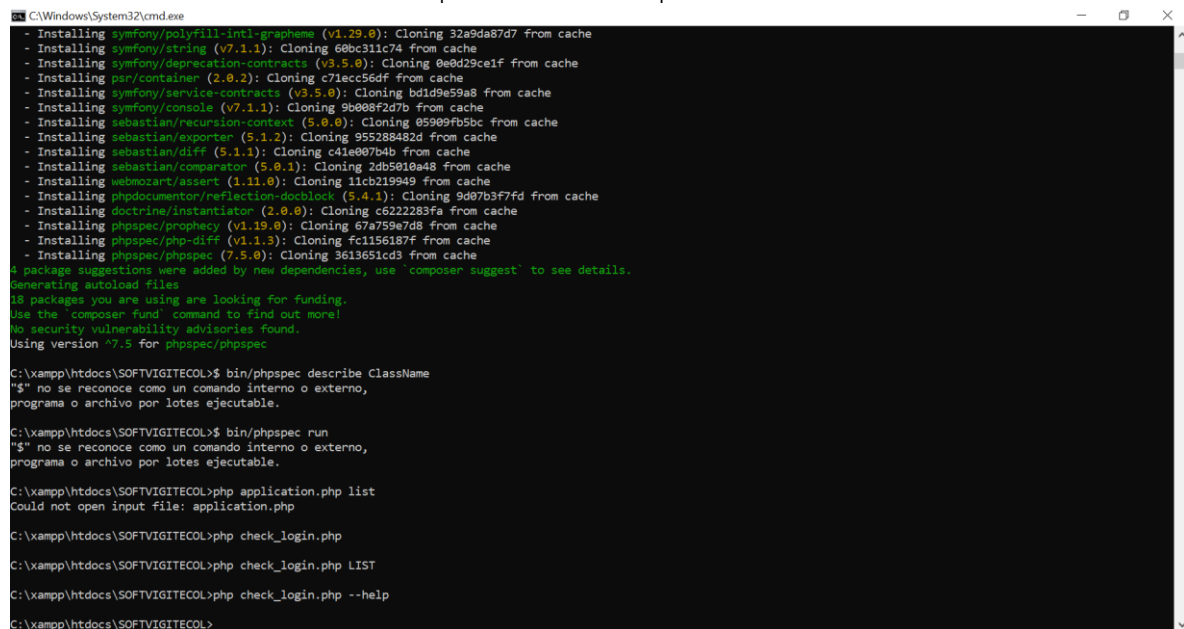
3. Ejecutar pruebas:

```
bash
```

[Copiar código](#)

```
./vendor/bin/phpspec run
```

Pantallazo de la instalación del PHPSpec usando el composer:



```
C:\Windows\System32\cmd.exe
- Installing symfony/polyfill-intl-grapheme (v1.29.0): Cloning 32a9da87d7 from cache
- Installing symfony/string (v7.1.1): Cloning 60bc311c74 from cache
- Installing symfony/deprecation-contracts (v3.5.0): Cloning 0e8d29ce1f from cache
- Installing psr/container (2.0.2): Cloning c71ecc5ddf from cache
- Installing symfony/service-contracts (v3.5.0): Cloning bd1d9e59a8 from cache
- Installing symfony/console (v7.1.1): Cloning 9b008f2d7b from cache
- Installing sebastian/recursion-context (5.0.0): Cloning 05909fb5bc from cache
- Installing sebastian/exporter (5.1.2): Cloning 955288482d from cache
- Installing sebastian/diff (5.1.1): Cloning c41e007b4b from cache
- Installing sebastian/comparator (5.0.1): Cloning 2db5010a48 from cache
- Installing webmozart/assert (1.11.0): Cloning 11cb219949 from cache
- Installing phpdocumentor/reflection-docblock (5.4.1): Cloning 9d07b3f7fd from cache
- Installing doctrine/instantiator (2.0.0): Cloning c6222283fa from cache
- Installing phpspec/prophecy (v1.19.0): Cloning 67a759e7d8 from cache
- Installing phpspec/php-diff (v1.1.3): Cloning fc1156187f from cache
- Installing phpspec/phpspec (7.5.0): Cloning 3613651cd3 from cache
4 package suggestions were added by new dependencies, use 'composer suggest' to see details.
Generating autoload files
18 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
No security vulnerability advisories found.
Using version ^7.5 for phpspec/phpspec

C:\xampp\htdocs\SOFTVIGITECOL> bin/phpspec describe ClassName
"$" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\xampp\htdocs\SOFTVIGITECOL> bin/phpspec run
"$" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

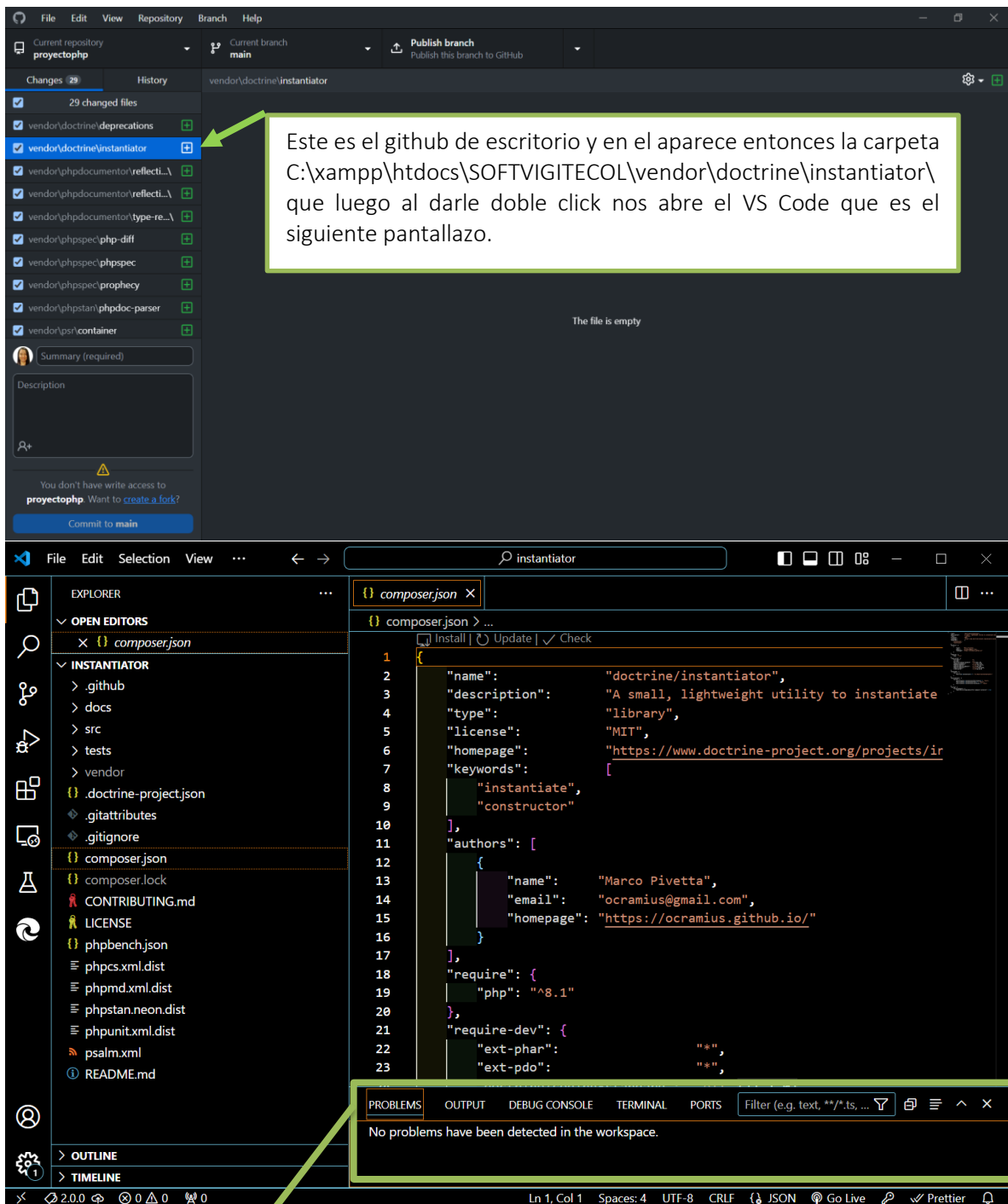
C:\xampp\htdocs\SOFTVIGITECOL> php application.php list
Could not open input file: application.php

C:\xampp\htdocs\SOFTVIGITECOL> php check_login.php

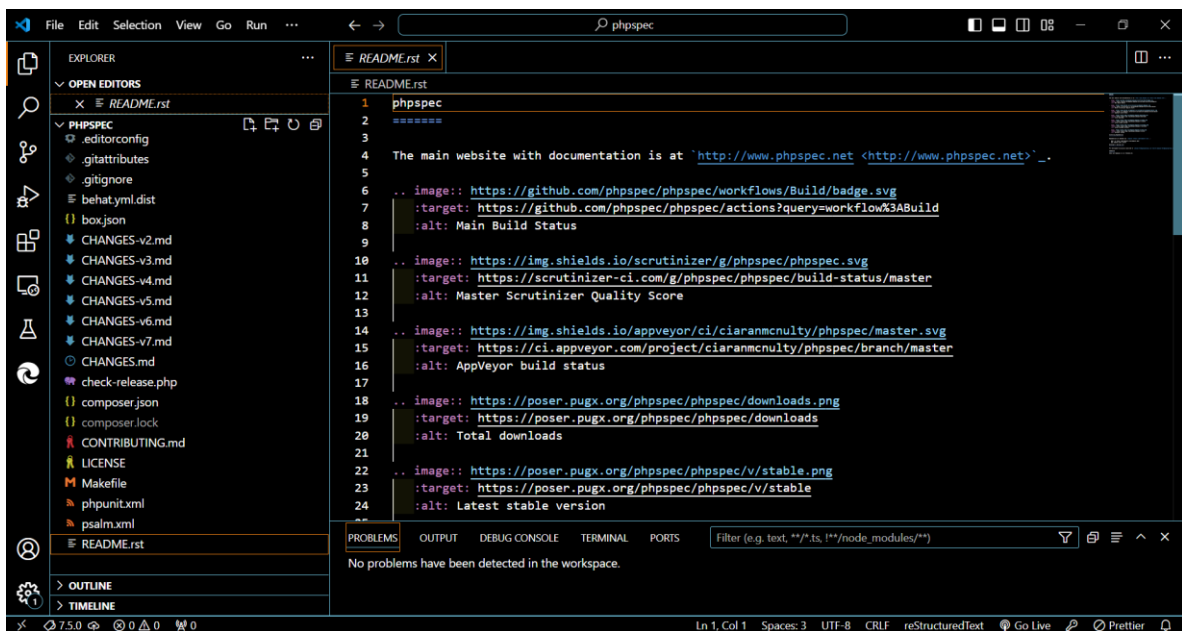
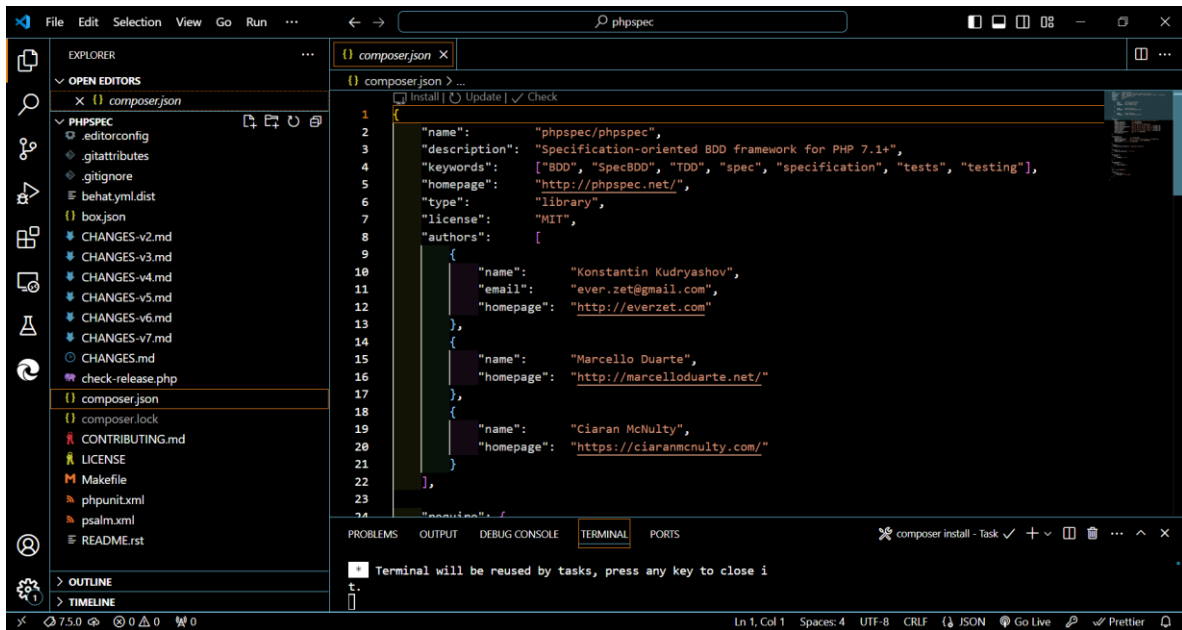
C:\xampp\htdocs\SOFTVIGITECOL> php check_login.php LIST

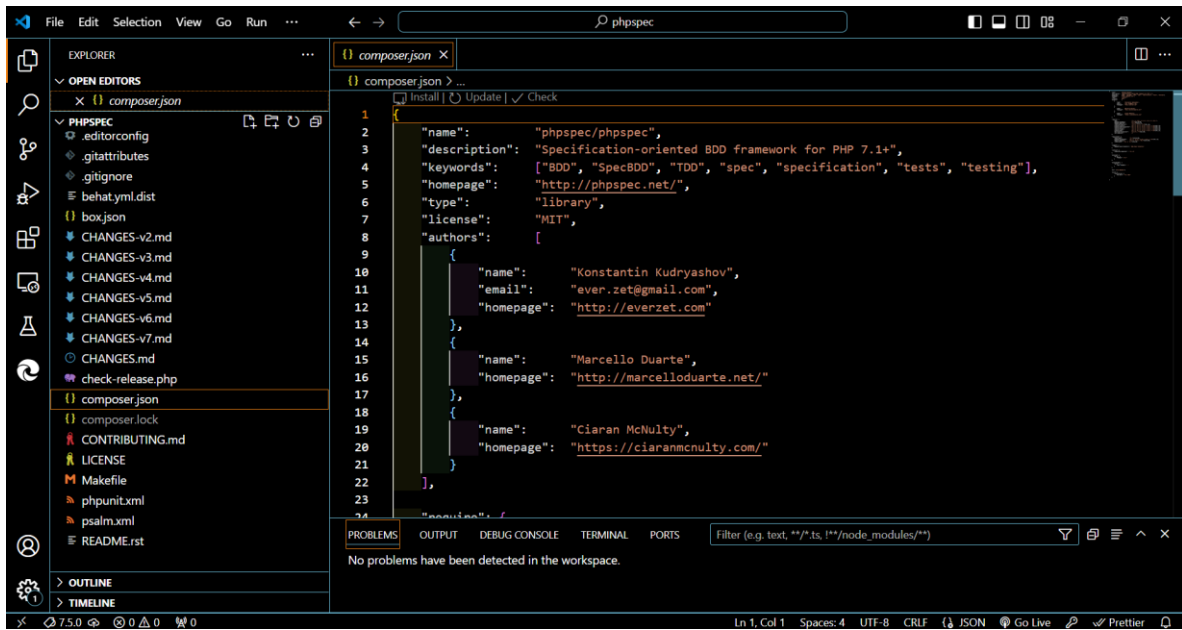
C:\xampp\htdocs\SOFTVIGITECOL> php check_login.php --help

C:\xampp\htdocs\SOFTVIGITECOL>
```



Aquí aparece que no hay ningún error en el código, en el *instantiator*, en el archivo *composer.json*.





1. Conclusiones

- La implementación de buenas prácticas de calidad en el desarrollo de software para SOFVIGITECOL es un paso fundamental para asegurar la entrega de un producto final robusto y confiable. A través de un enfoque estructurado y la utilización de herramientas adecuadas, el equipo de desarrollo puede garantizar la satisfacción tanto de los propietarios del negocio como de sus clientes.
- Al aplicar buenas prácticas de calidad, el equipo puede identificar y mitigar riesgos potenciales, garantizando que el software cumpla con los estándares de rendimiento, seguridad y usabilidad esperados. La adopción de un proceso de desarrollo bien definido, la selección de herramientas tecnológicas adecuadas y la documentación adecuada de todos los aspectos del proyecto contribuyen significativamente a este objetivo.
- Además, al seguir un enfoque iterativo y centrado en el cliente, el equipo puede adaptarse rápidamente a los cambios en los requisitos y las necesidades del restaurante, asegurando que el software desarrollado continúe siendo relevante y útil a lo largo del tiempo.
- La herramienta que escogimos para realizar este plan de pruebas es el phpspec, el cual podemos instalar desde su sitio web oficial <https://www.phpspec.net/en/stable/manual/installation.html>
- La codificación de módulos de software y la realización de pruebas de calidad, como las proporcionadas por herramientas como PhpSpec, son esenciales para asegurar la funcionalidad, fiabilidad y mantenibilidad del software. A continuación, se presentan algunas conclusiones clave acerca de estos aspectos:

1. Importancia de la Codificación Modular

Reusabilidad: La codificación modular permite la reutilización de componentes de software en diferentes partes de la aplicación o en proyectos futuros. Esto reduce el tiempo de desarrollo y los costos a largo plazo.

Mantenibilidad: Al dividir el software en módulos bien definidos, se facilita la identificación y corrección de errores, así como la actualización y mejora de funcionalidades específicas sin afectar el resto del sistema.

Escalabilidad: Los sistemas modulares pueden crecer de manera más eficiente, ya que es más fácil añadir nuevas funcionalidades sin necesidad de rediseñar la arquitectura existente.

Colaboración: Facilita el trabajo en equipo, ya que diferentes desarrolladores pueden trabajar en distintos módulos simultáneamente sin interferencias significativas.

2. Beneficios de las Pruebas de Calidad con PhpSpec

Desarrollo Guiado por Pruebas (TDD): PhpSpec fomenta el desarrollo guiado por pruebas, lo que ayuda a los desarrolladores a escribir código que cumple con los requisitos desde el principio, reduciendo la cantidad de errores y defectos.

Especificación de Comportamiento: PhpSpec permite escribir especificaciones de comportamiento que describen cómo debe funcionar el código, asegurando que los módulos desarrollados cumplan con los requisitos esperados.

Detección Temprana de Errores: Al ejecutar pruebas automáticamente durante el desarrollo, los errores pueden ser identificados y corregidos rápidamente, lo que reduce el costo y el esfuerzo asociados a la corrección de errores en etapas posteriores.

Documentación Viva: Las especificaciones escritas en PhpSpec sirven como documentación viva del comportamiento del sistema, facilitando la comprensión y el mantenimiento del código a largo plazo.

3. Conclusiones Generales

Calidad del Software: La combinación de codificación modular y pruebas de calidad asegura un alto nivel de calidad en el software, proporcionando sistemas más robustos y confiables.

Productividad del Desarrollo: Las pruebas automatizadas, como las de PhpSpec, mejoran la productividad de los desarrolladores al permitirles detectar y corregir errores más rápidamente y con mayor precisión.

Reducción de Costos: Aunque implementar un enfoque basado en módulos y pruebas automatizadas puede requerir una inversión inicial en tiempo y recursos, a largo plazo se traduce en una reducción de costos debido a la disminución de errores y el aumento de la eficiencia del desarrollo.

Adaptabilidad y Flexibilidad: Los sistemas modulares y bien probados son más adaptables a los cambios en los requisitos del cliente o en el entorno tecnológico, lo que permite una mayor flexibilidad en la evolución del software.

Implementación Recomendada

Para aprovechar al máximo los beneficios de la codificación modular y las pruebas de calidad con herramientas como PhpSpec, se recomienda:

Adoptar Buenas Prácticas de Diseño: Utilizar principios de diseño modular como SOLID para estructurar el código de manera eficiente.

Integración Continua: Implementar pipelines de integración y entrega continua (CI/CD) para automatizar la ejecución de pruebas y la integración de cambios en el código.

Formación y Capacitación: Proporcionar formación y recursos a los desarrolladores para que puedan utilizar efectivamente PhpSpec y otros frameworks de prueba.

Revisión y Refactorización: Realizar revisiones de código y refactorizaciones periódicas para mantener la calidad y la cohesión del código a lo largo del tiempo.

En resumen, la adopción de un enfoque modular en la codificación, junto con la implementación de pruebas automatizadas con herramientas como PhpSpec, constituye una estrategia eficaz para desarrollar software de alta calidad que es robusto, mantenible y escalable.