

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD: INFORMÁTICA Y ELECTRÓNICA

**ESCUELA DE INGENIERÍA EN SISTEMAS**

**CARRERA: SOFTWARE**

# GUÍA DE LABORATORIO DE APLICACIONES INFORMÁTICAS II

1. **DATOS GENERALES:**

**NOMBRE DEL ESTUDIANTE CODIGO DEL ESTUDIANTE**

Katherin Narváez 6779

**FECHA DE REALIZACIÓN: FECHA DE ENTREGA:**

**2025/07/18 2025/07/18**

1. **OBJETIVO(S):**

**2.1.GENERAL**

Desarrollar la fase de documentación adjunto al proceso de ingeniería inversa sore una aplicación de software ya existente la cual fue analizada y valorada, con el fin de comprender sus componentes, arquitectura, su funcionamiento y el análisis de consideraciones para que faciliten su mantenimiento.

**2.2.ESPECÍFÍCOS**

-Recopilar información del análisis anteriormente realizado y la aplicación de ingeniería inversa.

- Documentar la aplicación de ingeniería inversa sobre el aplicativo informático elegido en base a lo visto en clases.

1. **METODOLOGÍA**

En el presente documento se realizara la documentación de la aplicación de la técnica de ingeniería inversa aplicada anteriormente sobre un software existente. Mediante el análisis estático y dinámico.

1. **EQUIPOS Y MATERIALES:**

* Computador
* Entorno integrado de desarrollo (IDE)
* Aula virtual
* Acceso a internet
* Bibliografía

1. **MARCO TEORICO:**

**Proceso de ingeniería inversa:**

La ingeniería inversa puede ser un proceso complejo, pero generalmente sigue algunos pasos clave:

La ingeniería inversa de software es una disciplina que consiste en analizar un sistema informático existente para identificar sus componentes, estructura y comportamiento, sin tener acceso directo a su documentación original o especificaciones.

Este enfoque es útil en contextos donde se requiere mantenimiento, actualización o documentación de sistemas heredados, facilitando la comprensión de cómo funciona una aplicación desde el código hacia su estructura general.

El objetivo principal de la ingeniería inversa no es rediseñar el software, sino entender cómo fue construido, cómo interactúan sus componentes y qué funcionalidades ofrece. Con ello, se puede generar documentación, modelos y representaciones gráficas que orienten el trabajo de nuevos desarrolladores o equipos de mantenimiento.

**Técnicas principales:**

1. Análisis estático: Consiste en revisar el código fuente sin necesidad de ejecutarlo. Se examinan archivos, estructuras, módulos, funciones, clases, lógica condicional y organización general del proyecto. Esta técnica es muy útil en lenguajes como Python, donde el código suele ser más legible y directo.

2. Análisis dinámico: Se enfoca en observar cómo se comporta el sistema cuando está en ejecución. Permite ver cómo fluye la información, cómo se reciben y procesan los datos, cómo responde el sistema ante errores o entradas inesperadas, y cómo interactúan el frontend y el backend.

Ambos enfoques son complementarios: mientras que el análisis estático ofrece una visión estructural, el dinámico proporciona información sobre el rendimiento y funcionamiento real del sistema.

**Diagramas para documentación**:

Como parte del proceso de documentación de la ingeniería inversa, se recurre a diagramas UML (Unified Modeling Language), los cuales permiten representar gráficamente distintos aspectos del sistema.

• Casos de uso: representan las interacciones del usuario con el sistema.

• Diagramas de secuencia: muestran el flujo de mensajes entre los distintos componentes durante una operación.

• Diagramas de actividades: reflejan los pasos y decisiones dentro de un proceso funcional

- Diagramas de arquitectura: presentan la estructura general del sistema, sus capas o componentes y cómo se comunican.

Estos modelos facilitan la lectura técnica del sistema, la comprensión del flujo de trabajo y la identificación de oportunidades de mejora.

**Importancia de documentar mantenimiento de software:**

• Facilita el mantenimiento y evolución del software.

• Reduce el tiempo necesario para comprender sistemas complejos.

• Mejora la calidad del código mediante la detección de redundancias o errores lógicos.

• Permite la capacitación de nuevos integrantes del equipo de desarrollo.

**MANTENIMIENTO ADAPTATIVO**

Este mantenimiento se encarga de mantener operativo un programa cuando se realizan cambios en el entorno de producción. Es importante mencionar que este tipo de mantenimiento es uno de los más usuales ya que se debe a los cambios que se producen en la tecnología informática, dejando sin operar a los productos software que han sido desarrollados por la competitividad que se da entre empresas, dando como resultado que el software sea más utilizado mientras ocurran más modificaciones.

Objetivo: Modificar un programa en los diferentes cambios en el entorno, es decir, que se realicen cambios en el hardware o software en el que está siendo ejecutado.

1. **PROCEDIMIENTO DOCUMENTACION:**

**Información del Proyecto**

|  |  |
| --- | --- |
| Empresa / Organización | ESPOCH |
| Proyecto | **Sistema de Facturación(Veterinaria)** |
| Fecha de preparación | 07/18/2025 |

1. **HISTORIAL DE CAMBIOS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Versión** | **Organización** | **Descripción** |
| 07/17/2025 | 1.0 | ESPOCH | Análisis del proyecto existente |
| 07/18/2025 | 1.0 | ESPOCH | Aplicación de ingeniería inversa, análisis estático y dinámico. |
| 07/18/2025 | 1.0 | ESPOCH | Elaboración de diagramas |
| 07/18/2025 | 1.0 | ESPOCH | Elaboración del proceso de documentación |
| 07/18/2025 | 1.0 | ESPOCH | Retroalimentación de información, análisis y resultados |

**6.1 Proceso de Análisis**

Como anteriormente realizamos el análisis del dominio mediante la aplicación de ingeniería inversas del programa se pudo comprender que el software que se describe es un sistema de Facturación de una Veterinaria, donde el dominio de este sistema es la administración y operación básica de una veterinaria, con el enfoque en inventarios y facturación, principalmente determinándolo con la lógica de resolver actividades de un software de gestión para negocios veterinarios o comercios de productos para mascotas.  
  
Una vez que se realizo el respectivo proceso de comprensión del programa establecido, se determinó un análisis estático y dinámico, identificamos:

* Que se presentaron varios aspectos que se podrían considerar como deuda técnica, como es el caso de la generación en pdf o impresión de el comprobante.
* Adaptar la facturación a requisitos establecidos por las autoridades tributarias de la facturación nacional Ecutoriana.

1. **CONFIGURACIONES CRITICAS.**

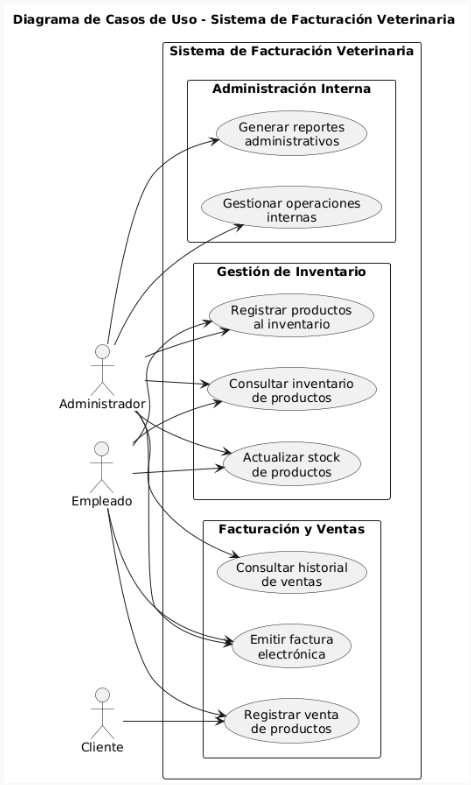
**Detalles de entorno:**

|  |  |
| --- | --- |
| Elemento | Descripción |
| Servidor | **Localhost (localhost) usando servidor embebido de PHP o XAMPP** |
| IDE | **Visual Studio Code (VS Code) con extensiones para PHP, HTML, CSS** |
| Lenguaje/Framework | **PHP 7.x o superior, sin frameworks externos (estructura personalizada)** |
| Base de Datos | **MySQL, definida en archivo conexion.php** |
| Servicios | **Navegador (Chrome/Firefox), ejecución local con conexión a base de datos** |
| Observaciones | **Entorno usado para probar validaciones de formularios, lógica de negocio y flujo básico de facturación e inventario** |

1. **DIAGRAMAS DE FLUJO**

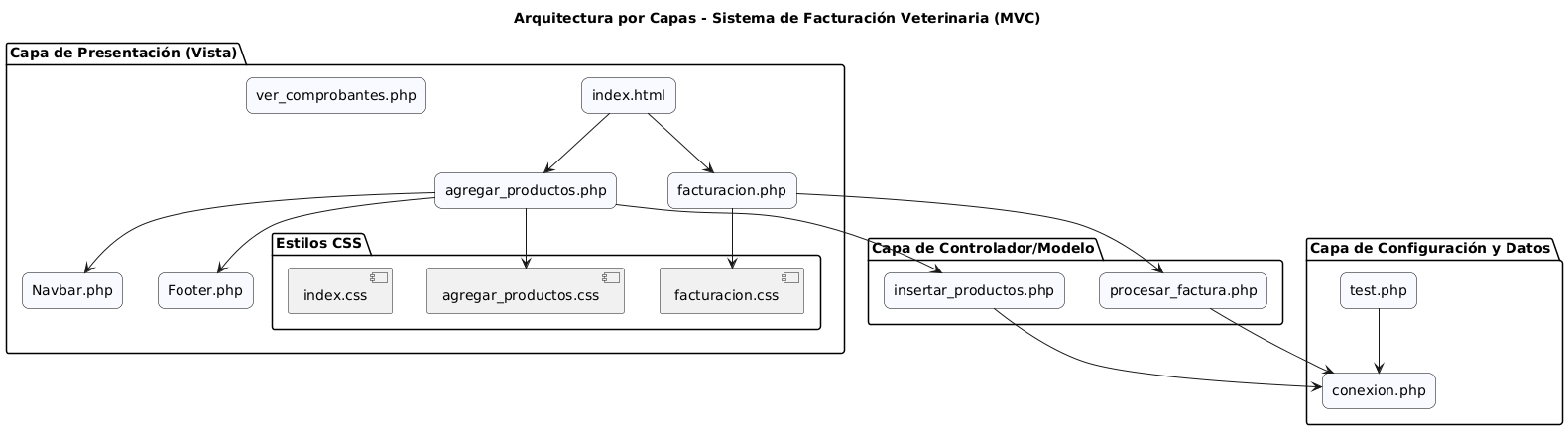
**6.1.1 Reconstrucción de la Arquitectura**

* **Diagrama de Casos de Uso :**

****

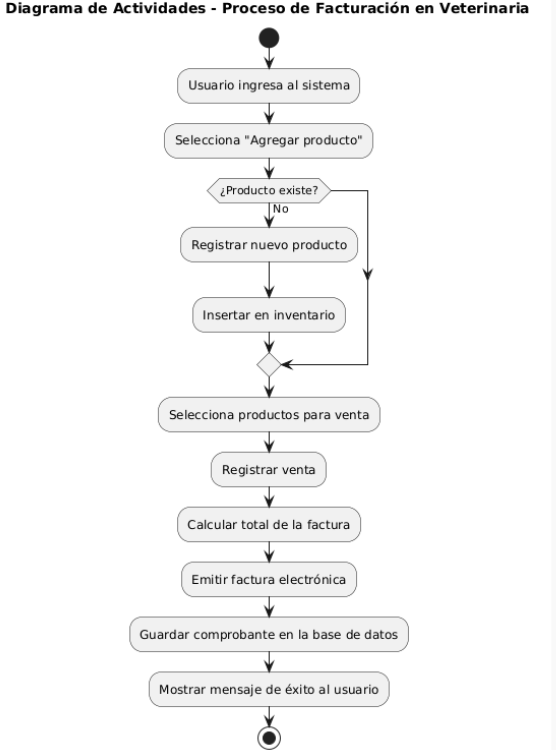
Este diagrama muestra las principales funcionalidades que el usuario puede realizar al interactuar con la aplicación. En este caso, representa cómo el usuario. Este diagrama es útil para entender las necesidades del usuario y los servicios que el sistema debe ofrecer desde su perspectiva.

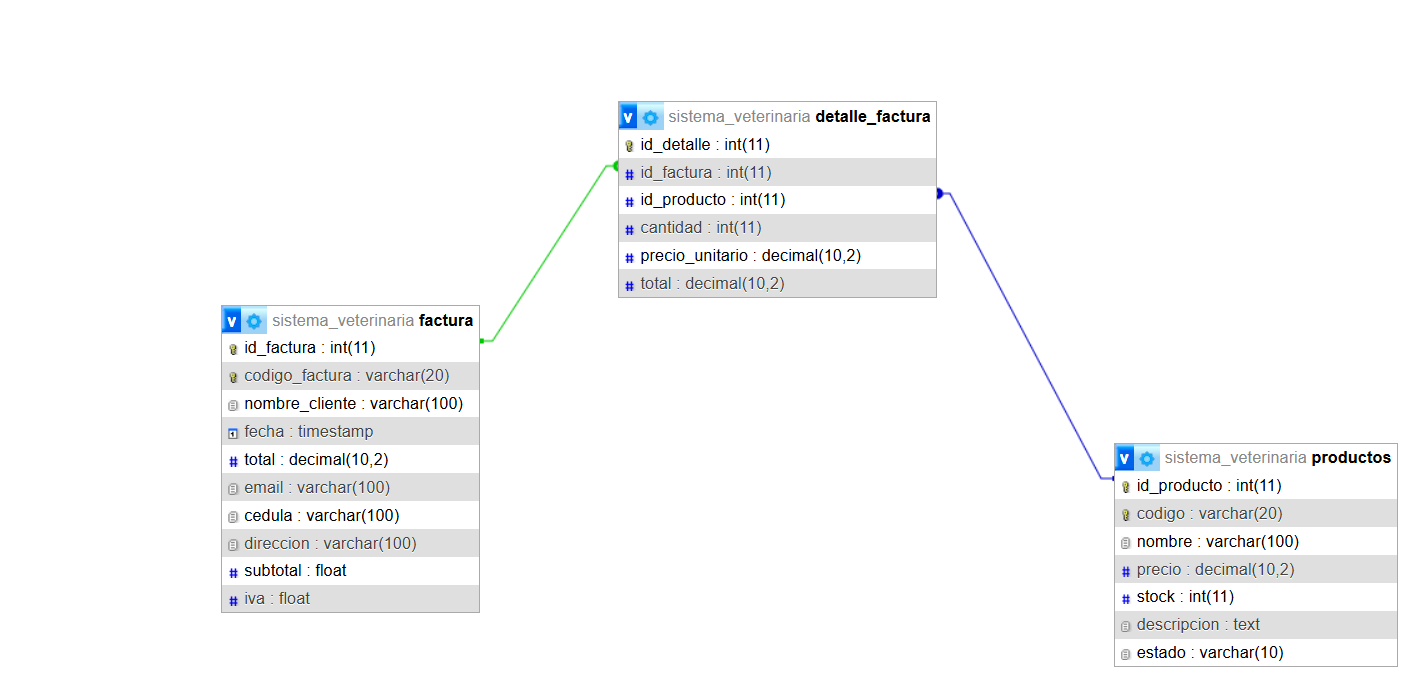
**Diagrama de la Arquitectura**

****

* **Diagrama de Procesos del Sistema**

Estos diagramas describen el flujo funcional del sistema, mostrando paso a paso las acciones que se ejecutan desde que el usuario inicia una operación hasta que se presenta el resultado. Incluye la entrada de datos, selección de la operación, validaciones y la visualización del resultado.

****

****

1. **GLOSARIO DE TÉRMINOS**
2. **PROCEDIMIENTO DE DESPLIEGUE Y RECUPERACIÓN**
3. **CONCLUSIONES Y RECOMENDACIONES:**

**Conclusiones:**

El análisis mediante ingeniería inversa permitió identificar con claridad los componentes clave del sistema, su arquitectura en capas (presentación, lógica y datos) y las dependencias entre ellos. Sin embargo, el proceso reveló limitaciones en la documentación interna, lo que dificultó la comprensión de ciertos módulos complejos, como el manejo de transacciones en la pasarela de pagos. Además, la falta de comentarios descriptivos en el código y la escasa trazabilidad entre los requisitos y su implementación complicaron el mapeo completo de funcionalidades.

Para finalizar podemos concluir que estas mejoras abordan seguridad, rendimiento, usabilidad y escalabilidad, basadas en el análisis y aplicación de la ingeniería inversa. Se recomienda comenzar con las de alta prioridad para reducir riesgos críticos.

**Recomendaciones:**

Para optimizar futuros ejercicios de ingeniería inversa, se recomienda implementar una documentación técnica detallada que incluya diagramas de secuencia y flujo de datos críticos.Estas mejoras no solo agilizarían el proceso de ingeniería inversa, sino que también facilitarían el mantenimiento y la evolución del sistema, asegurando una base más sólida para posteriores actualizaciones.

1. **BIBLIOGRAFÍA:**

* Eilam, E. (2005). *Reversing: Secrets of reverse engineering*. Wiley. <https://doi.org/10.1002/0471756978>
* Pressman, R. S. (2010). Ingeniería del Software: Un Enfoque Práctico (7ª ed.). McGraw-Hill. (Capítulo 29: "Reengineering and Reverse Engineering")
* Sommerville, I. (2011). Software Engineering (9ª ed.). Pearson. (Sección 27.4: "Reverse Engineering")
* Chikofsky, E. J. y Cross, J. H. (1990). Ingeniería inversa y recuperación de diseño: Una taxonomía. *IEEE Software, 7*(1), 13-17. <https://doi.org/10.1109/52.43044>