

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**

**Jeferson Vargas 6473**

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

**2025/07/15 2025/07/15**

1. **OBJETIVO(S):**

**2.1. GENERAL**

* Realizar el análisis de una aplicación (Sistema de facturación)para realizar el proceso de ingeniería inversa.

**2.2. ESPECÍFÍCOS**

* Analizar los archivos principales de el sistema para entender us funcionamiento
* Realizar el proceso de ingeniería inversa basándonos en los archivos que identificamos como relevantes

1. **METODOLOGÍA**

**Selección del Aplicativo Informático:**

* El primer paso consiste en identificar y seleccionar un aplicativo informático existente sobre el cual se aplicarán las técnicas de ingeniería inversa.
* Se recomienda elegir un software cuya complejidad sea manejable para el alcance de la práctica, pero que a su vez permita la aplicación efectiva de las técnicas.

**Aplicación de la Ingeniería Inversa:**

* Se deben utilizar técnicas de **análisis estático** (sin ejecutar el código) para examinar la estructura del software, como la revisión del código fuente (si está disponible), el uso de desensambladores (como IDA Pro o Ghidra para binarios) y decompiladores (como JD-GUI para Java o dnSpy para .NET) para obtener representaciones de alto nivel del código.
* Adicionalmente, se puede emplear el **análisis dinámico** (observando el comportamiento del software durante su ejecución) utilizando depuradores (como OllyDbg o GDB) para entender los flujos de ejecución, el uso de memoria y las interacciones con el sistema operativo o la red (mediante herramientas como Wireshark).
* El objetivo es comprender a fondo los componentes del aplicativo, sus interrelaciones, los flujos de datos, la lógica de negocio y la arquitectura inferida.

1. **EQUIPOS Y MATERIALES:**

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

1. **ANALISIS DEL SISTEMA**

El sistema analizado, del cual se ha examinado un componente clave, tiene como **propósito principal facilitar la emisión y gestión de comprobantes de pago electrónicos, específicamente facturas, de acuerdo con la normativa de la Superintendencia Nacional de Aduanas y de Administración Tributaria (SUNAT) de Perú**. Su función central es automatizar el proceso de creación de documentos fiscales digitales, su firma electrónica y su envío a la autoridad tributaria, garantizando así la validez y el cumplimiento fiscal de las transacciones comerciales.

En esencia, sirve como un **intermediario programático** entre las operaciones de negocio internas y los exigentes requisitos técnicos y legales de la facturación electrónica peruana, transformando datos comerciales en documentos fiscales válidos y comunicándolos eficientemente con SUNAT.

**Funcionalidades Principales**

Basándonos en el código y la estructura observada, las funcionalidades principales de este sistema son:

**Modelado de Documentos Electrónicos:**

* + Permite la **estructuración programática de una factura electrónica (Invoice)**, incluyendo todos los campos requeridos por SUNAT como la versión UBL, fechas de emisión y vencimiento, tipo de operación, tipo de documento (01 para factura), serie y número correlativo, moneda (PEN), y campos de observación.
  + Capacidad para **definir los detalles de venta (SaleDetail)** de cada ítem de la factura, especificando datos como código de producto, descripción, unidad de medida, cantidad, valores unitarios (con y sin IGV), montos de IGV, y el tipo de afectación al impuesto.
  + Soporte para la **inclusión de leyendas (Legend)** en la factura, como la representación textual del monto total.

**Gestión de Datos Maestros (Empresa y Cliente):**

* + A través de la clase auxiliar Util y su componente shared, el sistema abstrae la obtención de los **datos de la empresa emisora** (Razon Social, RUC, Dirección, etc.) y los **datos del cliente receptor** (Tipo de Documento, Número de Documento, Nombre/Razón Social, etc.). Esta centralización sugiere una funcionalidad de gestión de entidades previamente cargadas o configuradas.

**Generación y Envío a la Autoridad Tributaria:**

* + Orquesta la **conexión con los servicios web de la SUNAT** (o un intermediario como beatose.herokuapp.com) mediante la configuración de un servicio de envío (Greenter\See).
  + Realiza el **envío automatizado de la factura** al servicio de SUNAT, lo que implica la generación interna del XML bajo el estándar UBL, la aplicación de la firma digital (gestionada por Greenter), y la transmisión segura del documento.

**Manejo Básico de Respuestas Fiscales:**

* + **Verificación del Estado de Envío:** Capacidad para determinar si el documento fue recibido y validado exitosamente por SUNAT.
  + **Gestión de Errores:** En caso de fallos en el envío o validación por parte de SUNAT, el sistema captura y muestra el mensaje de error.
  + **Procesamiento del CDR:** Si el envío es exitoso, el sistema es capaz de obtener y manejar el Comprobante de Recepción (CDR) emitido por SUNAT, el cual es la constancia oficial de la validez del comprobante.

**Persistencia de Documentos (Auditoría):**

* + Utiliza la clase Util para **guardar los archivos XML** de las facturas generadas y los archivos **ZIP que contienen el CDR** recibido de SUNAT. Esta funcionalidad es crucial para el cumplimiento y la auditoría, permitiendo almacenar una copia local de los documentos fiscales emitidos y sus respuestas.

1. **ARQUITECTURA**

El análisis de ingeniería inversa del sistema de facturación, basado en el código fuente proporcionado (especialmente el script de emisión de facturas y las referencias a la clase Util), revela una arquitectura pragmática que combina elementos de un diseño en **Capas** con la aplicación parcial del patrón **Modelo-Vista-Controlador (MVC)**, y la utilización de patrones de diseño específicos como el **Singleton**.

**Enfoque Arquitectónico Principal: Arquitectura en Capas**

El sistema demuestra una clara separación de responsabilidades a través de una arquitectura en capas lógicas:

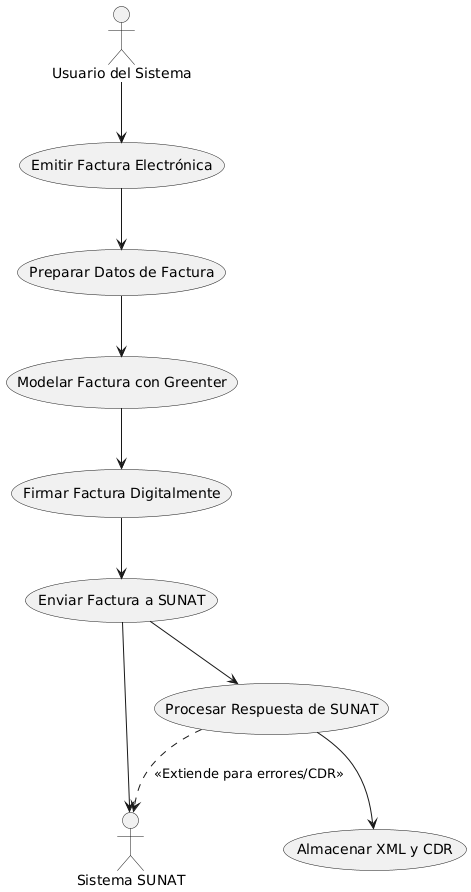
* **Capa de Orquestación/Controlador:** El script factura-beatose.php se posiciona como una capa de control y orquestación. Su función es recibir la solicitud de emisión de una factura, coordinar la preparación de los datos, interactuar con la lógica de negocio y los servicios de integración, y gestionar la respuesta final. En un sistema más amplio, esta capa podría ser un controlador de una aplicación web o un procesador de un servicio de fondo.
* **Capa de Lógica de Negocio/Dominio:** Esta capa contiene las reglas y el procesamiento central relacionados con la facturación.
  + Los **Modelos de Greenter** (Greenter\Model\Sale\Invoice, Greenter\Model\Sale\SaleDetail, Greenter\Model\Sale\Legend) actúan como representaciones de las entidades de dominio (la factura y sus ítems) y encapsulan la estructura de datos conforme a las especificaciones UBL y SUNAT.
  + La **Clase Util** se identifica como un componente clave en esta capa. Aunque su implementación completa no fue proporcionada, su rol inferido abarca lógica auxiliar crítica para el negocio, como la obtención de datos maestros de la empresa y el cliente, la gestión de la configuración del servicio SUNAT, y las operaciones de E/S de archivos para documentos fiscales.
* **Capa de Acceso a Datos (Inferida):** Se infiere la existencia de una capa de acceso a datos que interactúa con la **Clase Util**. Esta capa sería responsable de la persistencia y recuperación de datos maestros (empresa, clientes) de una base de datos subyacente. Adicionalmente, Util gestiona el acceso a datos relacionados con el almacenamiento de los archivos XML de las facturas generadas y los ZIP de los Comprobantes de Recepción (CDR) de SUNAT en el sistema de archivos.
* **Capa de Integración con Servicios Externos:** Esta capa gestiona la comunicación con entidades externas y servicios especializados.
  + La **Librería Greenter** constituye el componente principal de esta capa, encapsulando la complejidad de la comunicación con los servicios web de la SUNAT (firma digital, protocolos de comunicación).
  + El **Endpoint de SUNAT** (o del intermediario beatose.herokuapp.com) representa el servicio externo final con el que el sistema interactúa para la validación y registro de los comprobantes electrónicos.

**Patrones de Diseño Aplicados**

El análisis revela la aplicación de patrones de diseño que contribuyen a la estructura y funcionalidad del sistema:

* **Modelo-Vista-Controlador (MVC) - Aplicación Parcial:** Si bien el sistema no se presenta como un *framework* MVC completo, se observan elementos clave:
  + **Modelos:** Claramente representados por las clases de Greenter\Model (Invoice, SaleDetail, etc.), que encapsulan la información estructurada del documento fiscal.
  + **Controlador:** El script factura-beatose.php asume el rol de un controlador, orquestando el flujo desde la preparación de la factura hasta la gestión de la respuesta del servicio externo. La ausencia de una "Vista" explícita en el fragmento sugiere que la salida (errores, confirmaciones) podría manejarse en la consola o ser integrada en una capa de presentación superior.
* **Singleton:** La clase Util implementa el patrón **Singleton** (Util::getInstance()). Este patrón garantiza que solo exista una única instancia de Util a lo largo de la ejecución del programa. Esta elección de diseño es adecuada para componentes que gestionan recursos compartidos o globales, como configuraciones del sistema, credenciales o conexiones a servicios, asegurando una gestión centralizada y consistente.
* **Fluent Interface (Method Chaining):** El uso extensivo de encadenamiento de métodos (ej., $invoice->setUblVersion(...)->setFecVencimiento(...)->...) es un patrón que mejora la legibilidad del código al permitir que múltiples operaciones sobre un mismo objeto se realicen de manera concisa y consecutiva.

1. **MODELO DE CASOS DE USO**



1. **Análisis Estatico**

**Como primer paso realizamos la configuración del entorno en el que se puede desplegar el sistema pa poder utilizarlo en modo local, basándonos en el requerimiento del codigo alojado en el reposito rio de GitHub comprobamos que necesitamos php en sus versiones actualizadas y configuradas con las extensiones soao, y openssl activadas para lo caul es modifican en el archivo php init de nuestro servido en este caso XAMPP**

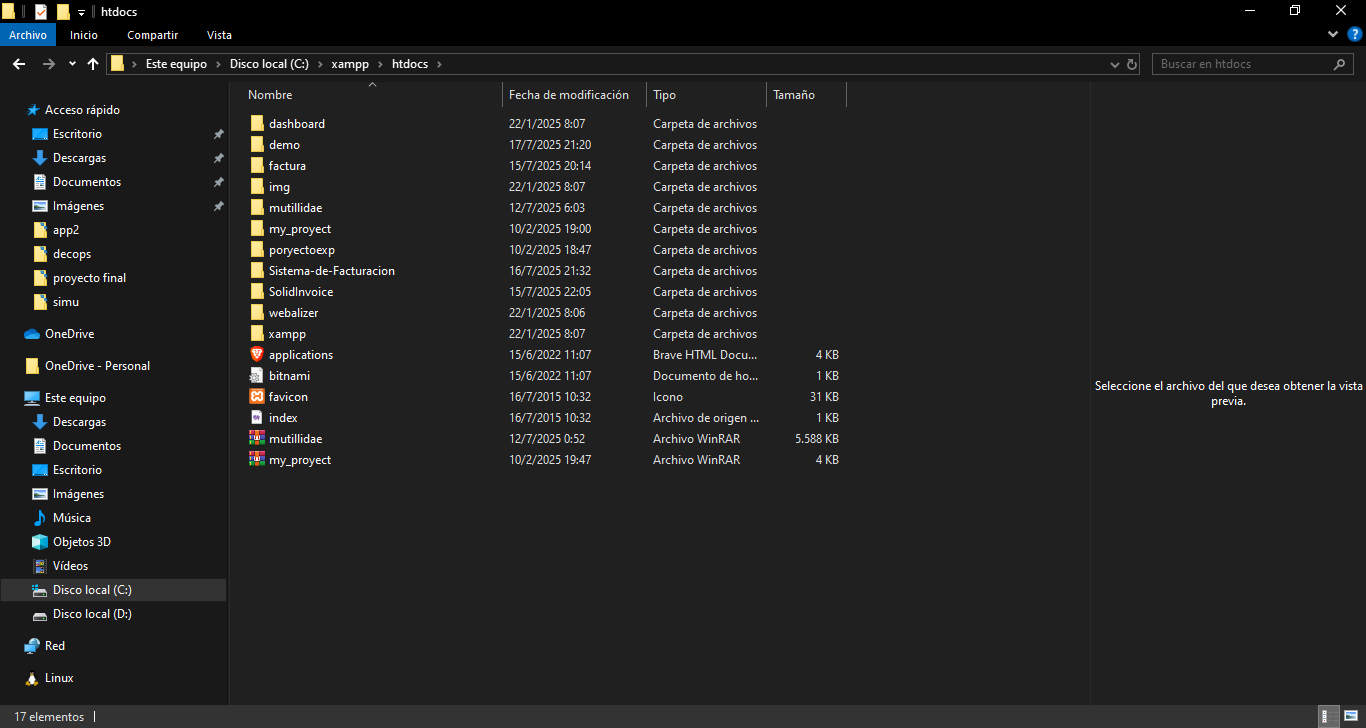
Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

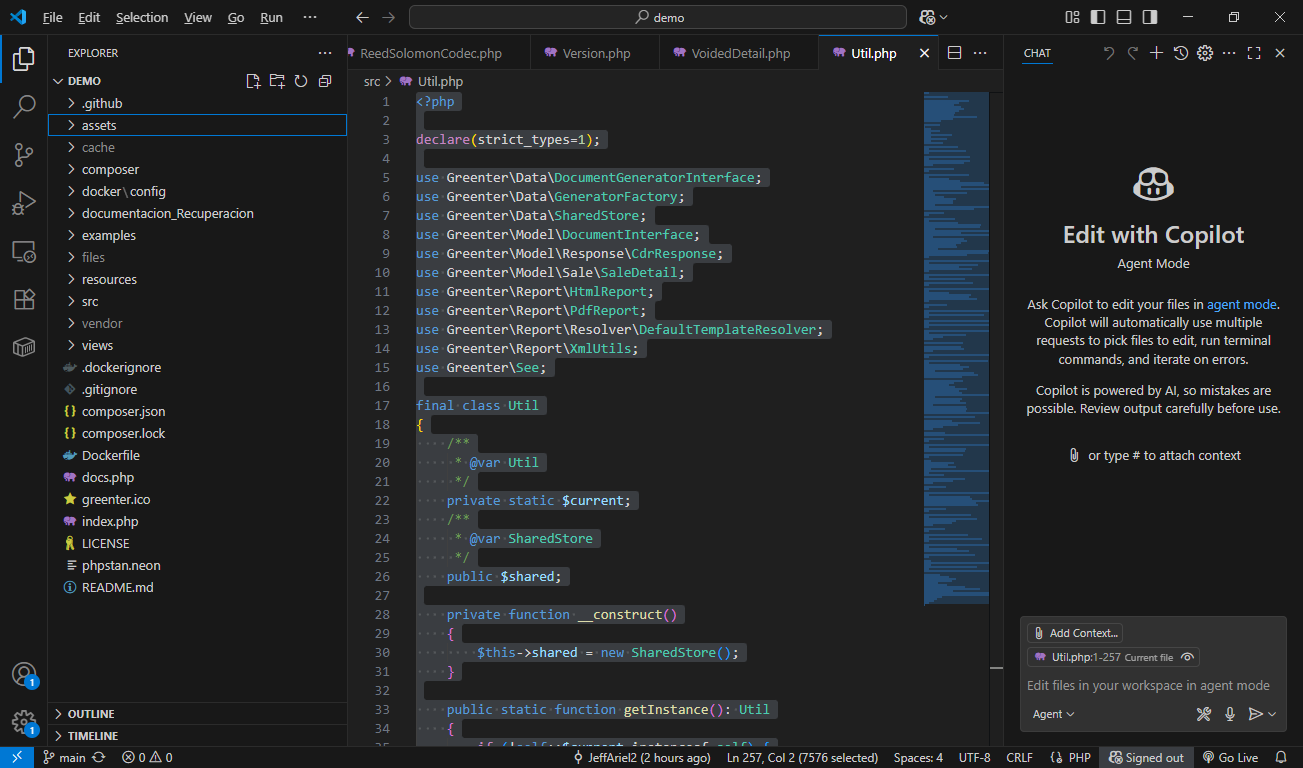
Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

**Una vez configurado el servido de XAAMP en el que uqeremos correr el aplicativo procedemos a cloanr nuestro repositorio en la carpeta del servidor . nuestro directorio clonado se llamara demo**



Para el análisis del código fuente y también de emos elegido el IDE Visual studio Code para lo cual copiamos nuestra carpeta a nuestro IDE para poder analizar de esta manera el sistema en un análisis estático.



Observando la estructura del directorio nos podemos fijar que tiene muchos documentos de diferente formato lo cual podemos deducir que el sistema tiene lenguajes de JavaScript, PHP, Html,CSS, entre otros