Universidad Tecnológica del Perú

Sistema de Análisis de Consumo Eléctrico

*Docente: Palomino Vidal Carlos Efrain

*Integrantes:

-Serrano Arias Alexander

-Ramos Hilario Adriel

*Carrera y ciclo:

-Ingeniería de sistemas - 3er ciclo

-Ingeniería de software - 3er ciclo

*Sede: UTP-LIMA NORTE

*Año:

2025

Índice:

Aspectos generales:

*Descripción del problema: Detectamos que muchas organizaciones almacenan datos de consumo eléctrico en archivos planos, pero no cuentan con una herramienta sencilla para analizarlos. Esto dificulta la obtención de promedios, comparaciones y reportes útiles para la toma de decisiones. Por ello, decidimos desarrollar un sistema que procese automáticamente estos datos, facilite su interpretación y permita generar reportes exportables de manera rápida y clara.

*Objetivos del Sistema:

°Objetivo general: Desarrollar un sistema que permita autenticar usuarios, procesar datos de consumo eléctrico y generar reportes estadísticos útiles de forma automática y accesible. El sistema deberá facilitar el análisis de la información mediante una interfaz interactiva, brindando opciones para visualizar o exportar los resultados obtenidos.

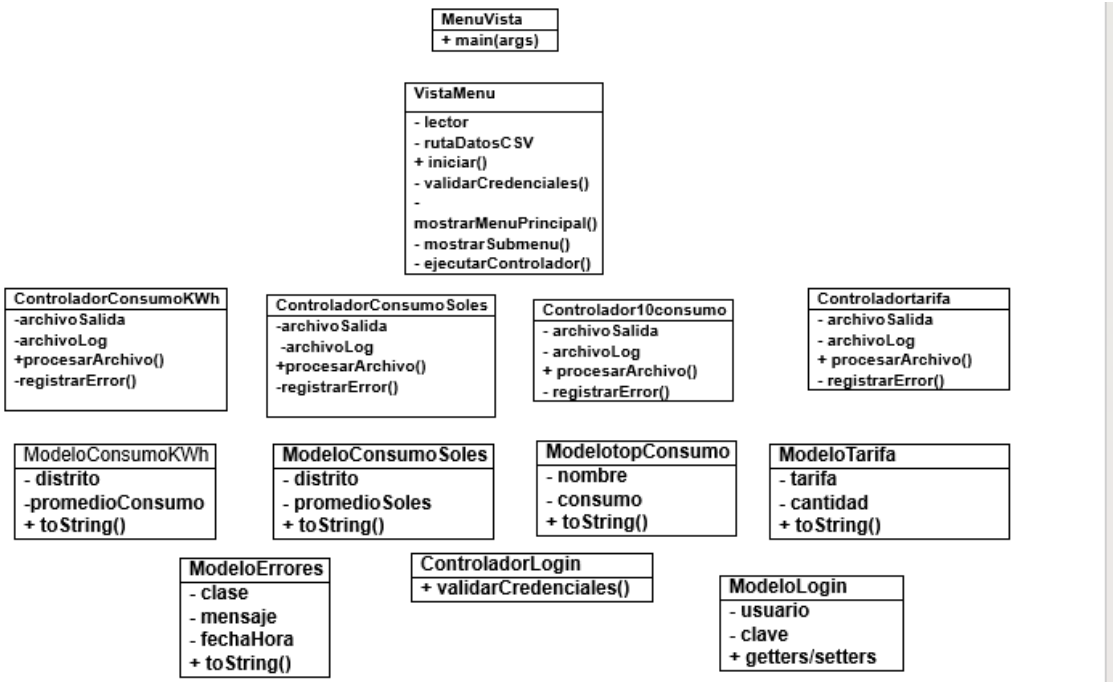
°Objetivos específicos:

-Validar el acceso de usuarios mediante un sistema de autenticación con archivo de texto.

-Procesar datos de consumo eléctrico desde un archivo CSV y extraer información relevante.

-Generar reportes estadísticos y permitir su visualización o exportación en archivos de texto.

Diseño de la Aplicación:



-Descripción de módulos:

Nombre del Módulo	Funcionalidad del módulo
Módulo de Ranking de Consumo Eléctrico (Controlador10Consumo)	Identifica a los 10 usuarios con mayor consumo de energía (kWh) y genera un reporte.
Módulo de Promedio de Consumo por Distrito en kWh (ControladorConsumoKWh)	Calcula el consumo promedio de electricidad por distrito a partir de los registros.
Módulo de Promedio de Consumo en Soles por Distrito (ControladorConsumoSoles)	Calcula cuánto se paga en promedio por consumo eléctrico en cada distrito.
Módulo de Autenticación de Usuarios (ControladorLogin)	Verifica usuario y contraseña desde un archivo y gestiona los intentos fallidos.
Módulo de Estadísticas de Tarifas Eléctricas (ControladorTarifa)	Cuenta cuántos usuarios están asociados a cada tipo de tarifa eléctrica.

Nombre del Módulo	Funcionalidad del Módulo
Modelo de Consumo por Distrito en kWh (ModeloConsumoKWh)	Representa y muestra el consumo promedio de energía (kWh) por distrito.
Modelo de Consumo en Soles por Distrito (ModeloConsumoSoles)	Representa y muestra el promedio del costo en soles por consumo eléctrico por distrito.
Modelo de Registro de Errores (ModeloErrores)	Registra errores indicando clase, mensaje y fecha/hora del suceso.
Modelo de Autenticación de Usuario (ModeloLogin)	Contiene los datos de usuario y contraseña para validar el acceso al sistema.
Modelo de Tarifa Eléctrica (ModeloTarifa)	Representa cada tipo de tarifa eléctrica y su número de usuarios asociados.
Modelo de Top de Consumo por Usuario (ModeloTopConsumo)	Almacena los usuarios con mayor consumo eléctrico para generar el ranking.

Nombre del Módulo	Funcionalidad del Módulo
Módulo de Ejecución del Programa (MenuVista)	Inicia el sistema y llama a la vista principal donde se gestiona el acceso y los reportes.
Módulo de Interfaz de Usuario Principal (VistaMenu)	Inicia el sistema y llama a la vista principal donde se gestiona el acceso y los reportes.

-Opciones del menú principal y lo que realiza cada una:

Opción del Menú	Descripción de la Funcionalidad
Promedio de consumo por distrito (kWh)	Calcula y muestra o exporta el promedio de consumo eléctrico por distrito en kilovatios hora.
Promedio de consumo en soles por distrito	Calcula y muestra o exporta el promedio del costo en soles del consumo eléctrico por distrito.
Top 10 usuarios con mayor consumo	Muestra o exporta un ranking con los 10 usuarios que más consumieron energía.
Total de consumo por tarifa	Genera un reporte con el número de usuarios por tipo de tarifa eléctrica.
Salir	Finaliza la ejecución del programa.

Opción del Submenú	Descripción de la Funcionalidad
Mostrar en pantalla	Ejecuta el reporte seleccionado y lo muestra directamente en la consola.
Exportar a archivo plano	Genera el reporte y lo guarda como archivo de texto plano en el proyecto.
Volver al menú principal	Retorna al menú principal sin ejecutar ninguna acción adicional.

```
String menuPrincipal = """

        --------------- MENÚ PRINCIPAL ----------------
        1. Promedio de consumo por distrito (kWh)
        2. Promedio de consumo en soles por distrito
        3. Top 10 usuarios con mayor consumo
        4. Total de consumo por tarifa
        5. Salir
        """;
```

```
String submenu = """

        -------- SUBMENÚ --------
        1. Mostrar en pantalla
        2. Exportar a archivo plano
        3. Volver al menú principal
        """;
```

Paquete: controlador

• Clase: Controlador10Consumo

Atributo	Tipo	Descripción
archivoSalida	String	Ruta del archivo donde se guardará el reporte del Top 10 de consumo.
archivoLog	String	Ruta del archivo donde se registran los errores.
mayores	double	Arreglo que guarda los 10 consumos más altos encontrados.
usuarios	String	Arreglo que guarda los nombres de usuarios correspondientes al top 10 de consumo.

Método	Tipo	Descripción
procesarArchivo()	void	Procesa el archivo CSV para obtener el Top 10 de consumo.
registrarError()	void	Registra errores en un archivo log usando la clase ModeloErrores.

Paquete: controlador

• Clase: ControladorConsumoKWh

Atributo	Tipo	Descripción
archivoSalida	String	Ruta del archivo donde se guarda el promedio de consumo por distrito.
archivoLog	String	Ruta del archivo para registrar errores.
distritosProcesados	String	Cadena acumuladora para evitar duplicar el procesamiento de distritos.

Método	Tipo	Descripción
procesarArchivo()	void	Lee el archivo CSV, agrupa por distrito y calcula el promedio.
registrarError()	void	Registra errores en un archivo log con la clase ModeloErrores.

Paquete: controlador

• Clase: ControladorConsumoSoles

Atributo	Tipo	Descripción
archivoSalida	String	Ruta del archivo donde se guarda el promedio en soles por distrito.
archivoLog	String	Ruta del archivo donde se registra cualquier error.
distritosProcesados	String	Acumulador que evita procesar varias veces el mismo distrito.

Método	Tipo	Descripción
procesarArchivo()	void	Lee el archivo de datos y calcula el promedio de consumo en soles por distrito.
registrarError()	void	Registra errores en un archivo log con la clase ModeloErrores.

Paquete: controlador

• Clase: ControladorLogin

Atributo	Tipo	Descripción
modeloLogin	ModeloLogin	Contiene los datos del usuario ingresados en la vista.
intento	int	Lleva el conteo de intentos de ingreso fallidos.
BufferedReader br	BufferedReader	Permite leer línea por línea el archivo usuarios.txt
partes	String	Arreglo que contiene usuario y clave separados por coma desde el archivo.

Método	Tipo	Descripción
validarCredenciales()	boolean	Verifica si usuario y clave coinciden con los registrado

Paquete: controlador

• Clase: ControladorTarifa

Atributo	Tipo	Descripción
archivoSalida	String	Ruta y nombre del archivo donde se exporta el reporte generado.
archivoLog	String	Ruta del archivo donde se registran errores en caso de fallos.
procesarArchivo()	void	Procesa los datos desde el archivo de entrada para contar usuarios por tipo de tarifa.
BufferedReader	BufferedReader	Permite leer línea por línea el archivo de entrada.
tarifas	String	Arreglo para almacenar los nombres únicos de tarifas.
conteos	int	Arreglo para contar cuántos usuarios hay por cada tipo de tarifa.
registrarError	void	Registra los errores en un archivo log.

Método	Tipo	Descripción
procesarArchivo()	void	Procesa el archivo para contar usuarios por tipo de tarifa.
registrarError()	void	Registra errores en un archivo log con ayuda del modelo ModeloErrores.

Paquete: Modelo

• Clase: ModeloConsumoKWh

Atributo	**Tipo**	**Descripción**
distrito	String	Nombre del distrito al que pertenece el consumo.
promedioConsumo	double	Promedio de consumo eléctrico en kWh del distrito.

Método	Tipo	Descripción
Constructor	Constructor	Inicializa distrito y promedioConsumo.
toString()	String	Devuelve el objeto como cadena formateada para mostrar.

Paquete: Modelo

• Clase: ModeloConsumoSoles

Atributo	**Tipo**	**Descripción**
distrito	String	Nombre del distrito al que pertenece el consumo.
promedioSoles	double	Promedio de consumo en S/. del distrito.

Método	Tipo	Descripción
Constructor	Constructor	Inicializa distrito y promedioSoles.
toString()	String	Devuelve el objeto como cadena formateada.

Paquete: Modelo

• Clase: ModeloErrores

Atributo	Tipo	Descripción
clase	String	Nombre de la clase donde ocurrió el error.
mensaje	String	Mensaje o descripción del error.
fechaHora	String	Fecha y hora exacta en que ocurrió el error. .

Método	Tipo	Descripción
Constructor	Constructor	Inicializa clase, mensaje y fecha/hora actual.
obtenerFechaHoraActual()	String	Obtiene la fecha y hora actual en formato "yyyy-MM-dd HH:mm:ss".
toString()	String	Retorna el error como cadena con formato de log.

Paquete: Modelo

• Clase: ModeloLogin

Atributo	**Tipo**	**Descripción**
usuario	String	Nombre del usuario usado para autenticación.
clave	String	Contraseña correspondiente al usuario.

Método	Tipo	Descripción
Constructor	Constructor	Inicializa usuario y clave.
setUsuario()	void	Asigna valor al atributo usuario.
setClave()	void	Asigna valor al atributo clave.
getUsuario()	String	Devuelve el nombre de usuario.
getClave()	String	Devuelve la contraseña.

Paquete: Modelo

• Clase: ModeloTarifa

Atributo	**Tipo**	**Descripción**
Tarifa	String	Categoría o tipo de tarifa registrada.
cantidad	int	Número de usuarios asociados a esa tarifa.

Método	Tipo	Descripción
Constructor	Constructor	Inicializa los valores de tarifa y cantidad.
toString()	String	Devuelve el objeto como una cadena formateada.

Paquete: Modelo

• Clase: ModeloTopConsumo

Atributo	**Tipo**	**Descripción**
nombre	String	Nombre del usuario, cliente.
consumo	double	Cantidad de consumo registrado (en kWh o S/).

Método	Tipo	Descripción
Constructor	Constructor	Inicializa nombre y consumo.
toString()	String	Devuelve el objeto como cadena formateada.

Paquete: vista

Clase: MenuVista

Atributo	Tipo	Descripción
menu	VistaMenu	Objeto que representa el menú principal de la aplicación.

Método	Tipo	Descripción
main()	static void	Método principal que inicia la ejecución del programa.

Paquete: vista

Clase: VistaMenu

Atributo	Tipo	Descripción
lector	Scanner	Objeto para la lectura de datos desde consola.
rutaDatosCSV	String	Ruta al archivo CSV que contiene los datos de entrada.

Método	Tipo	Descripción
VistaMenu	constructor	Inicializa el lector y define la ruta del archivo CSV.
iniciar	void	Controla el flujo de inicio de sesión e intenta autenticar al usuario.
validarCredenciales	boolean	Verifica si las credenciales ingresadas coinciden con las del archivo.
mostrarMenuPrincipal	void	Muestra el menú principal y permite al usuario elegir una funcionalidad.
mostrarSubmenu	void	Muestra un submenú con opciones para mostrar o exportar los datos.
ejecutarControlador	void	Ejecuta el controlador correspondiente según la opción seleccionada.

Conclusiones:

- A lo largo del proyecto optamos por una estructura basada en el patrón MVC, lo cual fue clave para mantener una buena organización del código. Esta separación entre la lógica del sistema, los datos y la interfaz nos ayudó a entender mejor cada parte del programa y facilitó el desarrollo en equipo.
- Supimos manejar correctamente la lectura y escritura de archivos, especialmente al trabajar con datos reales en formato CSV y al generar logs de errores. Esto le dio al sistema mayor realismo y robustez.
- Logramos implementar una interfaz por consola clara y funcional, permitiendo al usuario acceder fácilmente a reportes como promedios por distrito, top de consumos y distribución de tarifas. Además, la opción de exportar resultados fue un plus importante en cuanto a utilidad.

Recomendaciones:

- Para mejorar la confiabilidad del sistema, sería bueno reforzar las validaciones tanto en las entradas del usuario como en la lectura de archivos, ya que errores de formato o datos faltantes podrían afectar los resultados.
- Una mejora significativa sería implementar una base de datos en lugar de trabajar solo con archivos planos. Esto permitiría una gestión más eficiente de la información y facilitaría la escalabilidad del sistema.
- Finalmente, se podría migrar la aplicación a una plataforma web, lo cual mejoraría la experiencia del usuario y abriría la posibilidad de llegar a más personas con una interfaz visualmente más amigable.

BIBLIOGRAFIA:

Tutoriales y documentación:

• Video sobre lectura de archivos CSV en Java
https://www.youtube.com/watch?v=_GpN3GQFgP4

• Curso Java desde cero (Píldoras Informáticas) –

https://www.youtube.com/playlist?list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B

• Uso del patrón MVC en Java –
https://www.baeldung.com/mvcpattern

• Gestión de errores y archivos log en Java –

https://www.youtube.com/watch?v=eybg9K2OJnE

• Scanner en Java –

https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

Foros y apoyo técnico:

• Stack Overflow – https://stackoverflow.com

https://youtu.be/Z0yLerU0g-Q?si=m1YdPcjZW9r2_Q9y