Universidad Minuto de Dios

Facultad de Ingeniería y Arquitectura

Carrera de Ingeniería de Sistemas



**MERCADO DE ENERGÍA DE COLOMBIA**

Trabajo de investigación de la asignatura

**participantes**

[**Samuel Vega Mojica**](https://presencial.aulasuniminuto.edu.co/user/view.php?id=29253&course=4246), **Nicolas Andres Angarita Perez**, [**Juan Diego Chaparro Vargas**](https://presencial.aulasuniminuto.edu.co/user/view.php?id=2276&course=4246), [**Juan Sebastian Blanco Barbosa**](https://presencial.aulasuniminuto.edu.co/user/view.php?id=31009&course=4246)

**Código** 1002989, 1016004, 1006401

[**ALONSO GUEVARA PEREZ**](https://presencial.aulasuniminuto.edu.co/user/view.php?id=1957&course=4246) **Asesor**

Bogotá – Colombia

Mayo de 2025

ANALISIS DEL PROYECTO

**1. Propósito del Proyecto**

El proyecto es un sistema de gestión de consumo eléctrico para clientes. Permite registrar clientes, asociar dispositivos de medición (registradores) a cada cliente, generar datos de consumo eléctrico automáticamente y realizar modificaciones manuales. También incluye la funcionalidad para generar facturas en formato PDF utilizando la biblioteca iText.

**2. Estructura del Proyecto**

El proyecto sigue una estructura estándar de Maven para aplicaciones Java:

Proyecto-electricidad/

├── pom.xml

├── src/

│ └── main/

│ └── java/

│ └── com/

│ └── mycompany/

│ └── proyecto/

│ ├── controller/

│ │ ├── ConsumoControlador.java

│ │ └── SistemaEnergia.java

│ ├── model/

│ │ ├── Cliente.java

│ │ ├── Consumo.java

│ │ ├── FacturaGenerator.java

│ │ └── Registrador.java

│ └── view/

│ └── Main.java

│ └── MainView.java

**3. Funcionalidades Principales**

**3.1. Gestión de Clientes**

* Crear clientes con un ID único, nombre y dirección.
* Editar los datos de un cliente existente.
* Buscar clientes por su ID.

**3.2. Gestión de Registradores**

* Asociar registradores a un cliente.
* Editar la ubicación de un registrador.
* Buscar registradores por su ID dentro de un cliente.

**3.3. Gestión de Consumo**

* Generar datos de consumo eléctrico automáticamente para un cliente o todos los clientes.
* Modificar manualmente el consumo eléctrico de una hora específica.

**3.4. Generación de Facturas**

* Generar facturas en formato PDF para un cliente específico y un mes determinado utilizando la clase FacturaGenerator.

**4. Análisis de Clases**

**4.1. Cliente**

* Representa a un cliente con atributos como [id](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [nombre](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [direccion](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ") y una lista de registradores.
* Métodos clave:
  + [agregarRegistrador](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Permite asociar un registrador al cliente.
  + [buscarRegistrador](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Busca un registrador por su ID.

**4.2. Registrador**

* Representa un dispositivo de medición con atributos como [id](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [ubicacion](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ") y un objeto de tipo [Consumo](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html).
* Métodos clave:
  + Getters y setters para acceder y modificar los atributos.

**4.3. Consumo**

* Maneja datos de consumo eléctrico y costos organizados por mes, día y hora.
* Métodos clave:
  + [generarDatos](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Genera datos de consumo aleatorios.
  + [modificarConsumoHora](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Permite modificar manualmente el consumo de una hora específica.
  + Métodos para calcular el consumo y costo diario, mensual y anual.

**4.4. FacturaGenerator**

* Genera facturas en formato PDF utilizando la biblioteca iText.
* Métodos clave:
  + [generarFactura](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Crea un archivo PDF con los datos del cliente, consumo y costos.

**4.5. SistemaEnergia**

* Gestiona la lista de clientes y sus registradores.
* Métodos clave:
  + [agregarCliente](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [buscarCliente](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "), [editarCliente](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "): Gestión de clientes.
  + [agregarRegistradorACliente](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [editarRegistrador](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Gestión de registradores.

**4.6. ConsumoControlador**

* Controla la interacción con el usuario mediante un menú en consola.
* Métodos clave:
  + [crearCliente](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [editarCliente](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "): Gestión de clientes.
  + [crearRegistrador](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [editarRegistrador](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "): Gestión de registradores.
  + [cargarConsumoCliente](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html), [cargarConsumoTodos](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o "): Generación automática de datos de consumo.
  + [modificarConsumoHora](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Modificación manual del consumo.
  + [generarFactura](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Generación de facturas en PDF.

**4.7. Main**

* Punto de entrada de la aplicación. Inicia el controlador principal.

4.8. MainView

• Proporciona una interfaz gráfica (GUI) para interactuar con el sistema.

• Utiliza bibliotecas como Swing o JavaFX (dependiendo de cómo lo implementaste).

• Permite al usuario realizar todas las operaciones principales (crear/editar/buscar clientes, generar consumo, generar facturas) de manera visual e intuitiva.

• Métodos clave: inicializarComponentes, mostrarPantallaPrincipal, conectarConControlador.

5. Dependencias

El proyecto utiliza Maven para la gestión de dependencias. En el archivo (pom.xml) se incluye la biblioteca ¡Text para generación de PDFs:

<dependency>

<groupId>com.itextpdf</groupId>

<artifactId>itextpdf</artifactId>

<version>5.5.13.3</version>

</dependency>

**6. Puntos Fuertes**

* **Organización del Código**: El proyecto está bien estructurado con una separación clara entre las capas de modelo, controlador y vista.
* **Funcionalidad Completa**: Cubre todas las operaciones necesarias para gestionar clientes, registradores y consumo eléctrico.
* **Uso de Maven**: Facilita la gestión de dependencias y la compilación del proyecto.
* **Generación de Facturas**: La integración con iText permite crear facturas en formato PDF de manera profesional.

**7. Áreas de Mejora**

1. **Validación de Entrada**:
   * Actualmente, la validación de entrada en el controlador es básica. Podría mejorarse para manejar errores como entradas no numéricas o valores fuera de rango.
2. **Pruebas Unitarias**:
   * No se incluyen pruebas unitarias en el proyecto. Sería útil agregar pruebas para validar la funcionalidad de las clases principales.
3. **Documentación**:
   * Aunque el código es legible, sería útil agregar comentarios más detallados en los métodos para explicar su propósito y funcionamiento.
4. **Interfaz de Usuario**:
   * La interacción actual es mediante consola. Considerar implementar una interfaz gráfica (GUI) para mejorar la experiencia del usuario.

**Optimización del Cálculo de Costos**:

* El método calcularCosto  en Consumo utiliza un generador aleatorio para determinar el precio por kWh. Esto podría reemplazarse por una lógica más realista basada en tarifas predefinidas.

**8. Recomendaciones**

* Implementar pruebas unitarias utilizando JUnit para garantizar la calidad del código.
* Mejorar la validación de entrada en el controlador para evitar errores en tiempo de ejecución.
* Considerar el uso de una base de datos para almacenar los datos de clientes, registradores y consumo, en lugar de mantenerlos en memoria.
* Agregar un archivo de configuración para definir tarifas eléctricas y otros parámetros del sistema.

**DISEÑO DE PROYECTO:**

**1. Diseño Basado en Arquitectura en Capas**

El proyecto está dividido en las siguientes capas principales:

**1.1. Capa de Modelo (Model)**

* **Ubicación:** com.mycompany.proyecto.model
* **Responsabilidad:**
  + Contiene las clases que representan los datos y la lógica de negocio del sistema.
  + Ejemplos: Cliente, Registrador, Consumo, FacturaGenerator.
  + Estas clases encapsulan la lógica relacionada con los clientes, los dispositivos de medición y el consumo eléctrico.

**1.2. Capa de Controlador (Controller)**

* **Ubicación:** com.mycompany.proyecto.controller
* **Responsabilidad:**
  + Contiene la lógica de control que conecta la capa de modelo con la vista.
  + Ejemplos: ConsumoControlador, SistemaEnergia.
  + Gestiona las operaciones principales, como la creación de clientes, la generación de datos de consumo y la generación de facturas.

**1.3. Capa de Vista (View)**

* **Ubicación:** com.mycompany.proyecto.view
* **Responsabilidad:**
  + Contiene la interfaz de usuario, que en este caso es una interacción basada en consola.
  + Ejemplo: Main.
  + Sirve como punto de entrada para el usuario y delega las operaciones al controlador.

**2. Principios de Diseño Aplicados**

El diseño del proyecto refleja varios principios de diseño de software:

**2.1. Separación de Preocupaciones (SoC)**

* Cada capa tiene una responsabilidad específica:
  + La capa de modelo se encarga de los datos y la lógica de negocio.
  + La capa de controlador gestiona la interacción entre la vista y el modelo.
  + La capa de vista se encarga de la interacción con el usuario.

**2.2. Encapsulación**

* Las clases del modelo encapsulan los datos y exponen métodos para interactuar con ellos, como agregarRegistrador en Cliente o modificarConsumoHora en Consumo.

**2.3. Reutilización de Código**

* Las clases y métodos están diseñados para ser reutilizables. Por ejemplo, FacturaGenerator puede ser reutilizado para generar facturas en diferentes contextos.

**2.4. Bajo Acoplamiento**

* Las capas están desacopladas entre sí. Por ejemplo, la vista no interactúa directamente con el modelo, sino a través del controlador.

**3. Patrón de Diseño Identificado**

El proyecto implementa el patrón **MVC (Modelo-Vista-Controlador)**, que es común en aplicaciones organizadas en capas. Este patrón divide la aplicación en tres componentes principales:

**3.1. Modelo**

* Representa los datos y la lógica de negocio.
* Ejemplo: Las clases Cliente, Registrador y Consumo.

**3.2. Vista**

* Representa la interfaz de usuario.
* Ejemplo: La clase Main, que interactúa con el usuario a través de la consola.

**3.3. Controlador**

* Gestiona la lógica de la aplicación y actúa como intermediario entre el modelo y la vista.
* Ejemplo: La clase ConsumoControlador.

**4. Beneficios del Diseño**

1. **Mantenibilidad:** La separación en capas facilita la modificación y el mantenimiento del código. Por ejemplo, si se necesita modificar la forma de calcular el consumo eléctrico, solo se debe intervenir en las clases del modelo, sin afectar la vista o el controlador.
2. **Escalabilidad:** Es fácil agregar nuevas funcionalidades (como una interfaz gráfica usando JavaFX) sin alterar las capas existentes.
3. **Reutilización:** Las clases del modelo y el controlador pueden reutilizarse en diferentes contextos. Por ejemplo, la clase FacturaGenerator podría usarse tanto en consola como en una aplicación web.
4. **Pruebas** Unitarias: La separación de lógica facilita la creación de pruebas unitarias para cada capa, asegurando que cada componente se comporte como se espera.
5. **Desacoplamiento**: La vista y el modelo no interactúan directamente, lo cual permite cambios independientes en cada capa sin afectar a las demás.

**INFORME ACERCA DE LA CODIFICACION:**

**1. Introducción**

Este informe describe la codificación del proyecto, su estructura, las prácticas de programación implementadas y las áreas de mejora. El proyecto es un sistema de gestión de consumo eléctrico que permite registrar clientes, asociar dispositivos de medición, generar datos de consumo y emitir facturas en formato PDF.

**2. Organización del Código**

El proyecto está organizado siguiendo una arquitectura en capas, lo que facilita la separación de responsabilidades y el mantenimiento del código. La estructura principal del proyecto es la siguiente:

Proyecto-electricidad/

├── pom.xml

├── src/

│ └── main/

│ └── java/

│ └── com/

│ └── mycompany/

│ └── proyecto/

│ ├── controller/

│ │ ├── ConsumoControlador.java

│ │ └── SistemaEnergia.java

│ ├── model/

│ │ ├── Cliente.java

│ │ ├── Consumo.java

│ │ ├── FacturaGenerator.java

│ │ └── Registrador.java

│ └── view/

│ └── Main.java

│ └── MainView.java

**2.1. Paquete model**

* Contiene las clases que representan los datos y la lógica de negocio.
* Ejemplo: Cliente, Registrador, Consumo, FacturaGenerator.

**2.2. Paquete controller**

* Contiene las clases que gestionan la lógica de control y la interacción entre la vista y el modelo.
* Ejemplo: ConsumoControlador, SistemaEnergia.

**2.3. Paquete view**

* Contiene la clase principal que sirve como punto de entrada para la aplicación.
* Ejemplo: Main.

**3. Prácticas de Codificación**

**3.1. Uso de Clases y Métodos**

* El proyecto utiliza clases bien definidas para encapsular datos y lógica de negocio.
* Los métodos están diseñados para realizar tareas específicas, como agregarRegistrador en Cliente o generarFactura en FacturaGenerator.

**3.2. Modularidad**

* El código está dividido en módulos (paquetes) que agrupan clases relacionadas, lo que mejora la organización y la legibilidad.

**3.3. Uso de Maven**

* El proyecto utiliza Maven para la gestión de dependencias, lo que facilita la integración de bibliotecas externas como iText para la generación de PDFs.

**3.4. Separación de Preocupaciones**

* La lógica de negocio, la lógica de control y la interfaz de usuario están separadas en diferentes capas.

**4. Análisis de la Codificación**

**4.1. Puntos Fuertes**

1. **Estructura Clara:** La arquitectura en capas facilita la comprensión y el mantenimiento del código.
2. **Uso de Buenas Prácticas:** Se aplican principios como la encapsulación, la reutilización de código y la separación de preocupaciones.
3. **Gestión de Dependencias:** El uso de Maven simplifica la integración de bibliotecas externas.
4. **Generación de Facturas:** La integración con iText permite generar facturas en formato PDF de manera profesional.

**4.2. Áreas de Mejora**

1. **Validación de Entrada:**
   * La validación de datos ingresados por el usuario es básica y podría mejorarse para evitar errores en tiempo de ejecución.
2. **Pruebas Unitarias:**
   * No se incluyen pruebas unitarias en el proyecto. Sería útil agregar pruebas para validar la funcionalidad de las clases principales.
3. **Documentación:**
   * Aunque el código es legible, sería útil agregar comentarios más detallados en los métodos para explicar su propósito y funcionamiento.
4. **Interfaz de Usuario:**
   * La interacción actual es mediante consola. Considerar implementar una interfaz gráfica (GUI) para mejorar la experiencia del usuario.

**5. Recomendaciones**

1. **Implementar Pruebas Unitarias:**
   * Utilizar JUnit para crear pruebas que validen la funcionalidad de las clases principales.
2. **Mejorar la Validación de Entrada:**
   * Agregar validaciones más robustas para manejar errores como entradas no numéricas o valores fuera de rango.
3. **Agregar Documentación:**
   * Incluir comentarios detallados en el código y un archivo [README.md](vscode-file://vscode-app/c:/Users/svega/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) para describir el propósito del proyecto y cómo ejecutarlo.
4. **Optimizar el Cálculo de Costos:**
   * Reemplazar el uso de valores aleatorios para el precio por kWh con una lógica basada en tarifas predefinidas.
5. **Considerar el Uso de una Base de Datos:**
   * Implementar una base de datos para almacenar los datos de clientes, registradores y consumo, en lugar de mantenerlos en memoria.