UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1

CATEDRÁTICO: ING. WILLIAM ESTUARDO ESCOBAR ARGUETA

TUTOR ACADÉMICO: JOSUÉ RODOLFO MORALES CASTILLO

MANUAL TÉCNICO

Alex Ricardo Castañeda Rodríguez

CARNÉ: 202300476

SECCIÓN: B

GUATEMALA, 01 DE ABRIL DEL 2,024

ÍNDICE

INTRODUCCIÓN

El objetivo de este manual técnico es proporcionar una guía detallada sobre la implementación y funcionamiento interno del sistema de gestión de viajes en carro. Este documento está dirigido principalmente a desarrolladores y programadores que necesitan comprender la estructura del código, las tecnologías utilizadas y los procesos de desarrollo asociados con el sistema.

OBJETIVOS

1. GENERAL

1.1. El objetivo general de este manual técnico es explicar la arquitectura, el diseño y la implementación del sistema de gestión de viajes en carro, proporcionando una visión completa de su funcionamiento interno y sus componentes principales.

2. ESPECÍFICOS

2.1. Objetivo 1: Detallar la estructura del código y la lógica de programación utilizada en el desarrollo del sistema.

2.2. Objetivo 2: Explicar los requisitos de hardware y software necesarios para trabajar con la aplicación y realizar un desarrollo futuro.

ALCANCES DEL SISTEMA

El alcance de este manual técnico proporciona una visión general de las funcionalidades y características principales del sistema, así como los procedimientos de instalación, configuración y mantenimiento recomendados.

También se detallan las interfaces de usuario, los flujos de trabajo y los protocolos de comunicación utilizados dentro del sistema.

Se incluyen también las pautas y estándares de desarrollo seguidos durante la implementación del sistema, así como cualquier consideración de seguridad relevante. Además, se proporciona información sobre la escalabilidad del sistema y las posibles mejoras futuras que podrían ser implementadas.

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**
  - Procesador compatible con arquitectura x86.
  - Memoria RAM mínima de 2 GB.
  - Espacio de almacenamiento disponible de al menos 100 MB.
  - Conexión a Internet para acceder a recursos externos (opcional).
- **REQUISITOS DE SOFTWARE**
  - Sistema operativo compatible (Windows 7/8/10, macOS, Linux).
  - Java Development Kit (JDK) versión 8 o superior.
  - IDE de desarrollo Java (IntelliJ IDEA, Eclipse, NetBeans).

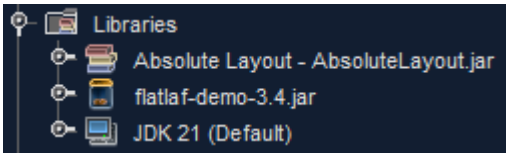
DESCRIPCIÓN DE LA SOLUCIÓN

El sistema de gestión de viajes en carro ha sido meticulosamente concebido y desarrollado tras un exhaustivo análisis de los requisitos establecidos en el enunciado. Para ello, se ha implementado una arquitectura modular que garantiza una escalabilidad sencilla y un mantenimiento eficiente del sistema a lo largo del tiempo. Esta solución se ha enfocado en brindar una interfaz de usuario intuitiva y amigable, facilitando así la experiencia del usuario en la gestión de sus viajes en carro. Además, en el backend se ha diseñado una estructura sólida y eficaz, asegurando un rendimiento óptimo del sistema en todo momento.

Este enfoque multidimensional no solo asegura la satisfacción del usuario, sino también la robustez y fiabilidad del sistema en su conjunto. La arquitectura modular permite la fácil incorporación de nuevas funcionalidades y adaptaciones según las necesidades cambiantes del mercado y de los usuarios. Asimismo, se ha puesto especial énfasis en la seguridad de los datos y la protección de la información sensible, implementando medidas de cifrado y protocolos de seguridad avanzados para garantizar la confidencialidad y la integridad de los datos del usuario. En resumen, el sistema de gestión de viajes en carro representa una solución completa y bien estructurada que satisface las necesidades actuales y futuras de los usuarios, ofreciendo una experiencia de usuario excepcional y un funcionamiento óptimo en todo momento.

LÓGICA DEL PROGRAMA

❖ Main



➢ Librerías

Librerías Utilizadas

El desarrollo de esta aplicación hace uso de diversas librerías de Java para facilitar y potenciar su funcionalidad. A continuación, se detallan las librerías utilizadas junto con sus funciones principales:

- **java.util:** Esta librería es fundamental para el manejo eficiente de colecciones de datos y otras utilidades relacionadas. Proporciona clases y interfaces para trabajar con listas, conjuntos, mapas, entre otros, lo que facilita el almacenamiento y manipulación de datos de manera estructurada.

- **javax.swing:** Utilizada específicamente para la creación de la interfaz gráfica de usuario (GUI). Esta librería ofrece un conjunto robusto de componentes gráficos, como botones, campos de texto, paneles, entre otros, que permiten diseñar interfaces interactivas y atractivas para los usuarios.

- **java.io:** Esta librería es esencial para el manejo de entrada y salida de archivos. Proporciona clases y métodos para leer y

escribir datos desde y hacia archivos en el sistema de archivos local. Esto resulta fundamental para la persistencia de datos y la manipulación de archivos en la aplicación.

- **java.util.concurrent:** Utilizada para la gestión de hilos de ejecución. Esta librería ofrece herramientas para trabajar con concurrencia en aplicaciones Java, permitiendo la ejecución simultánea de múltiples tareas de forma eficiente y segura. Facilita la creación y coordinación de hilos, así como el control de la concurrencia en la aplicación.

- **com.formdev.flatlaf:** Esta librería se utiliza para el diseño y estilo de la interfaz de usuario. Proporciona un conjunto de componentes gráficos con un estilo moderno y plano, que pueden integrarse fácilmente en la aplicación para mejorar su apariencia visual. Esto contribuye a una experiencia de usuario más agradable y actualizada.

➢ **Variables Globales de la clase (Main)**

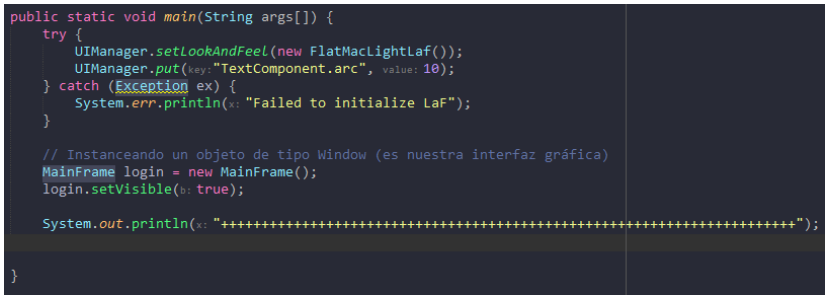
- **counterRoutes:** Contador de rutas cargadas en el sistema.
- **counterTrips:** Contador de viajes realizados.

- **fileChooser:** Objeto JFileChooser para seleccionar archivos.
- **finList:** ArrayList para almacenar destinos finales de viajes.
- **inicioList:** ArrayList para almacenar puntos de inicio de viajes.
- **routes:** ArrayList para almacenar rutas de viaje.

Breve descripción generalizada del uso de las variables globales:
Las variables globales se utilizan para almacenar datos relevantes para el funcionamiento del programa, como el número de rutas cargadas, el número de viajes realizados y listas para almacenar puntos de inicio y destinos finales de viajes. Estas variables se acceden y modifican en diferentes partes del programa según sea necesario para realizar diversas operaciones.

➢ **Función Main**

La función main se utiliza como punto de entrada del programa. En esta función, se realiza la configuración del aspecto y la sensación (Look and Feel) de la interfaz gráfica utilizando el tema FlatMacLightLaf y se establece un radio de esquina de 10 para los componentes de texto. Luego, se instancia un objeto de tipo MainFrame, que representa la ventana principal de la interfaz gráfica, y se hace visible para el usuario. Además, se imprime un mensaje en la consola para indicar que la inicialización del programa ha sido exitosa.



➢ **Métodos y Funciones utilizadas**

La función main se utiliza como punto de entrada del programa. En esta función:

- Se establece el aspecto y la sensación (Look and Feel) de la interfaz gráfica como FlatMacLightLaf.
- Se configura el radio de esquina de los componentes de texto en 10.
- Se instancia un objeto de tipo MainFrame, que representa la ventana principal de la interfaz gráfica.
- Se hace visible la ventana principal para el usuario.
- Se imprime un mensaje en la consola para indicar que la inicialización del programa ha sido exitosa.

MainFrame.java

Clase MainFrame

La clase MainFrame representa el marco principal de la interfaz gráfica. Es responsable de contener todos los elementos de la interfaz, incluido el menú y todas las funcionalidades del sistema.

Breve descripción del uso de la función o para qué sirve

Funciones utilizadas en MainFrame

Constructor **MainFrame()**

- Se configura la ventana como no decorada (**setUndecorated(true)**).
- Se inicializan los componentes de la interfaz (**initComponents()**).
- Se establece la ubicación de la ventana en el centro de la pantalla (**setLocationRelativeTo(null)**).
- Se configura la apariencia del JLabel **noPilotsLabel**.

- Se define la forma redondeada de la ventana.
- Se agrega un listener para permitir arrastrar la ventana.

Método **getTransport1()**

- Retorna el JLabel **lblTransport1**, utilizado para obtener la etiqueta de transporte 1.

Método **getBarrera()**

- Retorna el JLabel **lblBarrera**, utilizado para obtener la etiqueta de la barrera.

Método **refreshRoutesTable()**

- Actualiza la tabla de rutas con la información almacenada en **Main.routes**.

Método **chooseCSVFile()**

- Permite al usuario seleccionar un archivo CSV y luego carga y procesa el archivo seleccionado.

Método **readCSV()**

- Lee y procesa un archivo CSV seleccionado previamente.

Método **routeExists(String start, String end, int distance)**

- Verifica si una ruta ya existe en la lista de rutas.

Método **checkLists()**

- Verifica si las listas **Main.inicioList** y **Main.finList** están vacías y las imprime en la consola.

Método **loadRoutes()**

- Carga las rutas en los ComboBoxes después de leer el archivo CSV.

Método **getTableModel()**

- Retorna el modelo de tabla de la jTable1.

Método **findDistance(String startLocation, String endLocation)**

- Busca la distancia entre dos ubicaciones en la tabla de rutas.

Método **checkPilotsAvailability()**

- Verifica si hay pilotos disponibles.

Método **updatePilotsAvailability()**

- Actualiza el estado de los pilotos disponibles después de cada viaje generado.

Método **getVehicleIcon(String vehicleType)**

- Retorna el ícono correspondiente al tipo de vehículo especificado.

Método **updateVehicleAvailability(String vehicleType)**

- Actualiza la disponibilidad de los vehículos después de seleccionar uno para un viaje.

Clase **jPanelGradient**

- Clase interna que extiende JPanel y se utiliza para dibujar un fondo con gradiente.

```
public MainFrame() {
    this.setUndecorated(undecorated:true);
    initComponents();
    this.setLocationRelativeTo(c: null);
    // Configuración inicial del JLabel de pilotos no disponibles
    noPilotsLabel.setForeground(fg: Color.RED); // Texto en color rojo
    noPilotsLabel.setText(text:"No hay pilotos disponibles por el momento");
    noPilotsLabel.setVisible(aFlag: false); // Inicialmente invisible

    RoundRectangle2D.Float shape = new RoundRectangle2D.Float(x: 0, y: 0, w: getWidth(), h: getHeight(), arcw: 20, arch: 20);
    setShape(shape);

    // Agregar listener para arrastrar la ventana
    this.addMouseListener(new MouseAdapter() {
        public void mousePressed(MouseEvent evt) {
            mouseX = evt.getX();
            mouseY = evt.getY();
        }
    });
    this.addMouseMotionListener(new MouseAdapter() {
        public void mouseDragged(MouseEvent evt) {
            // Obtener la posición actual del mouse en la pantalla
            int x = evt.getXOnScreen();
            int y = evt.getYOnScreen();

            // Calcular la nueva posición de la ventana
            setLocation(x - mouseX, y - mouseY);
        }
    });

    panelTripStart.repaint();
     panelTripStart.setFocusable(focusable: true);
}
```

Distance.java

Clase Distance

La clase Distance se utiliza para editar la distancia de una ruta específica. Proporciona métodos para modificar la distancia entre dos puntos en la ruta.

```java
public class Distance extends javax.swing.JFrame {
private MainFrame mainFrame;
    private int mouseX, mouseY;
    public Distance(MainFrame mainFrame) {
        this.mainFrame = mainFrame;
        this.setUndecorated(undecorated:true);
        initComponents();
        this.setLocationRelativeTo(c: null);

        RoundRectangle2D.Float shape = new RoundRectangle2D.Float(x: 0, y: 0, w: getWidth(), h: getHeight(), arcw: 20, arch: 20);
        setShape(shape);

        // Agregar listener para arrastrar la ventana
        this.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent evt) {
                mouseX = evt.getX();
                mouseY = evt.getY();
            }
        });
        this.addMouseMotionListener(new MouseAdapter() {
            public void mouseDragged(MouseEvent evt) {
                // Obtener la posición actual del mouse en la pantalla
                int x = evt.getXOnScreen();
                int y = evt.getYOnScreen();

                // Calcular la nueva posición de la ventana
                setLocation(x - mouseX, y - mouseY);
            }
        });
    }
```

Journey.java

Clase Journey

La clase Journey es un hilo utilizado para mover los vehículos (carros o motos) a lo largo de una ruta durante un viaje. Gestiona la animación y actualización de la posición de los vehículos durante el recorrido.

```java
public Journey(JLabel labelToMove, JLabel destinationLabel) {
    this.labelToMove = labelToMove;
    this.destinationLabel = destinationLabel;

    this.startX = labelToMove.getX();
    this.endX = destinationLabel.getX() - labelToMove.getWidth(); // Ajuste para detenerse justo antes del destino
}

public void run() {
    try {
        // Calcula el desplazamiento necesario en el eje X
        int deltaX = endX - startX;
        // Calcula la distancia total que se necesita para llegar justo antes del destino
        double distance = Math.abs(a: deltaX);
        // Calcula el número total de pasos necesarios para alcanzar el destino
        int numSteps = (int) (distance / stepSize);
        // Calcula el tamaño del paso en el eje X
        double stepX = deltaX / (double) numSteps;

        // Mueve gradualmente el JLabel hacia el destino en el eje X
        for (int i = 0; i < numSteps && running; i++) {
            int newX = (int) Math.round(startX + i * stepX);
            SwingUtilities.invokeLater(() -> {
                labelToMove.setLocation(x: newX, y: labelToMove.getY());
                this.labelToMove.repaint();

            });
            Thread.sleep(millis: 140); // Pausa el hilo para suavizar el movimiento
        }

        SwingUtilities.invokeLater(() -> {
            labelToMove.setLocation(x: endX, y: labelToMove.getY());
            this.labelToMove.repaint();

        });

    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Route.java

Clase Route

La clase Route define un constructor para crear objetos de ruta. Almacena información sobre el punto de inicio, el punto final y la distancia entre ellos para un viaje.

```
class Route {
    private int id;
    private String start;
    private String end;
    private int distance;

    public Route(int id, String start, String end, int distance) {
        this.id = id;
        this.start = start;
        this.end = end;
        this.distance = distance;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getStart() {
        return start;
    }

    public void setStart(String start) {
        this.start = start;
    }

    public String getEnd() {
        return end;
    }

    public void setEnd(String end) {
        this.end = end;
    }

    public int getDistance() {
        return distance;
    }

    public void setDistance(int distance) {
        this.distance = distance;
    }

        // Por medio de esta funcion se retorna en un string los datos del objeto actual
    @Override
    public String toString() {
        return "Route{" + "id=" + id + ", start=" + start + ", end=" + end + ", distance=" + distance + '}';
    }
```

vehicles

Se refiere a un conjunto de imágenes de vehículos utilizadas en la interfaz gráfica para representar los diferentes tipos de transporte disponibles en el sistema. Estas imágenes se utilizan para proporcionar una representación visual de los vehículos durante la planificación y ejecución de viajes.

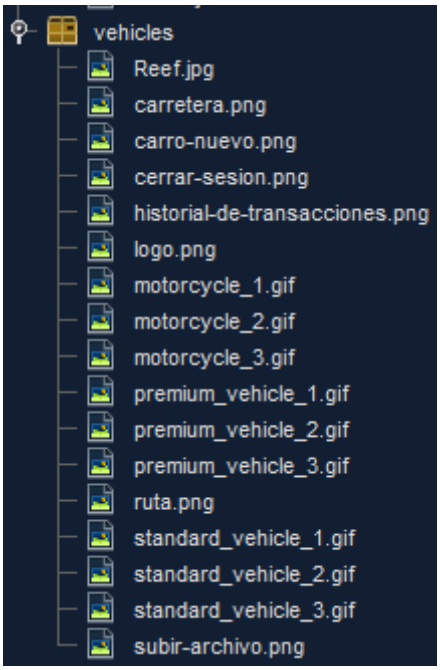


Diagrama de Clase