

Pontificia Universidad Católica del Ecuador – Sede Ambato



Informe Técnico - Sistema de Rutas con Grafos

Realizado por: Joaquín Esteban Bermeo Quispe

Materia: Programación IV

Docente: Edison Meneses

Fecha de Entrega: 18-06-2025

Contenido

1. Introducción	4
2. Arquitectura del Sistema	5
2.1 Estructura del Proyecto	5
2.2 Patrón de Arquitectura	5
3. Componentes Principales	6
3.1 Modelos de Datos	6
• Ciudad (ciudad.py)	6
• Ruta (ruta.py)	6
• Provincia (provincia.py)	6
3.2 Controladores	6
• Admin Controller (admin_controller.py)	6
• Auth Controller (auth_controller.py)	7
• Grafo Controller (grafo_controller.py)	7
• Home Controller (home_controller.py)	7
3.3 Utilidades de Grafo (grafo_db_utils.py)	8
4. Algoritmo de Dijkstra	8
4.1 Descripción del Algoritmo	8
4.2 Implementación en el Proyecto	8
4.2.1 Construcción del Grafo	8
4.2.2 Cálculo de Camino Óptimo	9
4.2.3 Validación de Entrada	9
4.3 Características del Grafo Implementado	10
4.3.1 Tipo de Grafo	10
4.3.2 Estructura de Datos	10
4.4 Ventajas de Usar NetworkX	10
4.4.1 Eficiencia	10
4.4.2 Funcionalidades Adicionales	10
4.5 Integración con Base de Datos	10
4.5.1 Flujo de Datos	10
4.5.2 Obtención de Conexiones	11
4.6 Manejo de Errores	11

4.6.1 Casos de Error Contemplados.....	11
4.6.2 Respuestas del Sistema	11
4.7 Visualización del Resultado	12
4.7.1 Generación de Imagen con Camino Resaltado	12
4.8 Ejemplo de Uso Completo	12
5. Funcionalidades del Sistema.....	12
5.1 Gestión de Datos	12
5.2 Cálculo de Rutas	13
5.3 Visualización.....	13
6. Flujo de Funcionamiento	13
6.1 Proceso de Cálculo de Rutas.....	13
6.2 Ejemplo de Uso.....	13
7. Base de Datos.....	14
7.1 Esquema de Tablas.....	14
8. Tecnologías Utilizadas	14
9. Consideraciones de Rendimiento.....	14
9.1 Optimizaciones para Dijkstra.....	14
9.2 Escalabilidad	15
10. Conclusiones	15
10.1 Fortalezas	15
10.2 Aplicaciones Prácticas	15
11. Anexos.....	16
11.1 Panel de Inicio.....	16
11.2 Introducción a los Grafos.....	16
11.3 Calculadora de Rutas	17
11.4 Ruta Fija: Ibarra → Loja.....	17
11.5 Administrar Provincias.....	18
11.6 Administrar Ciudades.....	18
11.7 Administración de Rutas	19
11.8 Enlace al repositorio del proyecto.....	19

1. Introducción

El presente informe detalla el desarrollo de un sistema de gestión y cálculo de rutas, implementado en lenguaje Python utilizando el framework Flask para la creación de la interfaz web. El sistema ha sido diseñado con el objetivo de facilitar la administración de una red de ubicaciones geográficas, permitiendo a los usuarios registrar provincias, ciudades y las rutas que las conectan. A través de una interfaz intuitiva desarrollada con HTML y Flask, los usuarios pueden crear, modificar, visualizar o eliminar tanto nodos como conexiones del sistema.

Uno de los componentes centrales del proyecto es la aplicación del algoritmo de Dijkstra, el cual permite calcular la ruta óptima entre dos puntos dentro de la red de ciudades registrada. Además, el sistema incluye la generación automática de gráficos que representan visualmente las rutas establecidas, ofreciendo así una perspectiva clara y organizada del mapa de conexiones.

Como complemento final, se ha integrado una funcionalidad que permite generar un documento en formato PDF, el cual resume detalladamente el cálculo de la ruta óptima, los puntos involucrados y sus respectivas distancias. Este enfoque no solo brinda una herramienta interactiva y funcional, sino que también fortalece el análisis y documentación de las rutas gestionadas dentro del sistema.

2. Arquitectura del Sistema

2.1 Estructura del Proyecto

```
Proyecto_Final_JB/
├── controllers/      # Controladores MVC
│   ├── admin_controller.py
│   ├── auth_controller.py
│   ├── grafo_controller.py
│   └── home_controller.py
├── models/          # Modelos de datos
│   ├── ciudad.py
│   ├── provincia.py
│   ├── ruta.py
│   └── user.py
├── routes/          # Definición de rutas web
│   ├── admin_routes.py
│   ├── auth_routes.py
│   ├── grafos_routes.py
│   └── home_routes.py
├── static/          # Archivos estáticos (CSS, JS, imágenes)
├── templates/       # Plantillas HTML
├── utils/           # Utilidades
│   └── grafo_db_utils.py
└── app.py           # Aplicación principal
```

2.2 Patrón de Arquitectura

El sistema implementa el patrón **MVC (Modelo-Vista-Controlador)**:

- Modelo: Clases en */models/* que representan las entidades de datos
- Vista: Plantillas HTML en */templates/*
- Controlador: Lógica de negocio en */controllers/*

3. Componentes Principales

3.1 Modelos de Datos

- Ciudad (ciudad.py)
 - Representa una ciudad en el grafo con sus atributos:
 - ID único
 - Nombre
 - Provincia asociada
- Ruta (ruta.py)
 - Representa las conexiones entre ciudades:
 - Ciudad origen
 - Ciudad destino
 - Costo
 - Tipo de ruta
- Provincia (provincia.py)
 - Agrupa ciudades por división territorial:
 - ID y nombre de provincia
 - Lista de ciudades asociadas

3.2 Controladores

- Admin Controller (admin_controller.py)
 - Maneja las funcionalidades administrativas del sistema:
 - **Gestión de usuarios:** Crear, editar, eliminar y listar usuarios
 - **Administración de ciudades:** CRUD completo de ciudades con validaciones
 - **Gestión de rutas:** Crear y modificar conexiones entre ciudades
 - **Panel de control:** Dashboard con estadísticas del sistema
 - **Validación de permisos:** Verificación de roles administrativos

- Auth Controller (auth_controller.py)
 - Controla la autenticación y autorización de usuarios:
 - **Login de usuarios:** Validación de credenciales y creación de sesiones
 - **Registro de nuevos usuarios:** Creación de cuentas con validaciones
 - **Logout:** Cierre seguro de sesiones
 - **Recuperación de contraseñas:** Proceso de reset de credenciales
 - **Middleware de autenticación:** Protección de rutas que requieren login
- Grafo Controller (grafo_controller.py)
 - Contiene la lógica principal para el manejo de grafos:
 - **Construcción del grafo:** Generación desde la base de datos
 - **Implementación del algoritmo de Dijkstra:** Cálculo de rutas óptimas
 - **Validación de entrada:** Verificación de ciudades origen y destino
 - **Procesamiento de resultados:** Formateo de caminos y costos
 - **Visualización de grafos:** Generación de imágenes del grafo
 - **Estadísticas del grafo:** Cálculo de métricas y análisis de conectividad
 - **Manejo de ciudades costeras:** Lógica específica para rutas que incluyan costa
- Home Controller (home_controller.py)
 - Gestiona las páginas principales y navegación del sitio:
 - **Página principal:** Renderizado del homepage con información general
 - **Navegación:** Control de menús y enlaces principales
 - **Presentación del sistema:** Información sobre funcionalidades
 - **Dashboard público:** Estadísticas generales sin requerir login
 - **Páginas informativas:** Secciones de ayuda y documentación
 - **Integración de componentes:** Coordinación entre diferentes módulos del sistema

3.3 Utilidades de Grafo (grafo_db_utils.py)

- Funciones auxiliares para:
 - Conexión con base de datos
 - Conversión de datos a estructura de grafo
 - Validación de datos de entrada

4. Algoritmo de Dijkstra

4.1 Descripción del Algoritmo

El algoritmo de Dijkstra encuentra el camino más corto entre dos nodos en un grafo ponderado con pesos no negativos. En este proyecto se utiliza para calcular rutas óptimas entre ciudades donde cada arista representa la distancia o costo de viaje.

4.2 Implementación en el Proyecto

El sistema utiliza la librería NetworkX de Python para implementar el algoritmo de Dijkstra de manera eficiente. La implementación se encuentra principalmente en el archivo *grafo_db_utils.py*.

4.2.1 Construcción del Grafo

```
1  def construir_grafo():
2      """Construye el grafo no dirigido y ponderado desde la base de datos"""
3      G = nx.Graph() # Grafo no dirigido
4
5      # Obtener todas las rutas de la base de datos
6      rutas = Ruta.query.all()
7
8      for ruta in rutas:
9          origen = ruta.ciudad_origen.nombre
10         destino = ruta.ciudad_destino.nombre
11         costo = float(ruta.costo)
12         # En un grafo no dirigido, una arista conecta en ambas direcciones automáticamente
13         G.add_edge(origen, destino, weight=costo)
14
15     return G
```


4.2.2 Cálculo de Camino Óptimo

```
1 def camino_optimo_con_costera(origen='Ibarra', destino='Loja'):
2     """
3     Calcula el camino óptimo entre dos ciudades que pase por al menos una ciudad costera
4     Utiliza el algoritmo de Dijkstra implementado en NetworkX
5     """
6     G = construir_grafo()
7     costeras = obtener_ciudades_costeras()
8
9     try:
10        # Aplicación directa del algoritmo de Dijkstra
11        camino = nx.dijkstra_path(G, origen, destino, weight='weight')
12        costo = nx.dijkstra_path_length(G, origen, destino, weight='weight')
13
14        # Verificar si el camino pasa por al menos una ciudad costera
15        contiene_costera = any(ciudad in costeras for ciudad in camino)
16
17        return {
18            "camino": camino,
19            "costo": costo,
20            "valido": contiene_costera,
21            "ciudades_costeras_en_ruta": [c for c in camino if c in costeras]
22        }
23    except nx.NetworkXNoPath:
24        return {
25            "camino": [],
26            "costo": None,
27            "valido": False,
28            "ciudades_costeras_en_ruta": []
29        }
```

4.2.3 Validación de Entrada

```
1 def validar_ciudades_existen(origen, destino):
2     """Valida que las ciudades de origen y destino existan en la base de datos"""
3     ciudad_origen = Ciudad.buscar_por_nombre(origen)
4     ciudad_destino = Ciudad.buscar_por_nombre(destino)
5
6     if not ciudad_origen:
7         return False, f"La ciudad de origen '{origen}' no existe"
8
9     if not ciudad_destino:
10        return False, f"La ciudad de destino '{destino}' no existe"
11
12    return True, "Ciudades válidas"
```

4.3 Características del Grafo Implementado

4.3.1 Tipo de Grafo

- **No Dirigido (Undirected):** Las rutas son bidireccionales
- **Ponderado:** Cada arista tiene un peso (costo/distancia)
- **Conexo:** Se asume que existe al menos un camino entre cualquier par de ciudades

4.3.2 Estructura de Datos

```
1 # Ejemplo de estructura del grafo en NetworkX:
2 G = {
3     'Quito': {'Ambato': {'weight': 120}, 'Latacunga': {'weight': 90}},
4     'Ambato': {'Quito': {'weight': 120}, 'Riobamba': {'weight': 50}},
5     'Latacunga': {'Quito': {'weight': 90}, 'Riobamba': {'weight': 65}},
6     # ... más conexiones
7 }
```

4.4 Ventajas de Usar NetworkX

4.4.1 Eficiencia

- **Implementación optimizada:** NetworkX usa heap binario internamente
- **Memoria optimizada:** Manejo eficiente de grafos grandes

4.4.2 Funcionalidades Adicionales

```
1 # Funciones adicionales disponibles:
2 camino = nx.dijkstra_path(G, origen, destino, weight='weight') # Camino óptimo
3 costo = nx.dijkstra_path_length(G, origen, destino, weight='weight') # Costo mínimo
4 existe = nx.has_path(G, origen, destino) # Verificar conectividad
```

4.5 Integración con Base de Datos

4.5.1 Flujo de Datos

Base de Datos → Modelos ORM → Construcción Grafo → Dijkstra → Resultados

4.5.2 Obtención de Conexiones

```
1 def obtener_rutas_desde_ciudad(ciudad_nombre):
2     """Obtiene todas las rutas conectadas a una ciudad específica"""
3     ciudad = Ciudad.buscar_por_nombre(ciudad_nombre)
4     if not ciudad:
5         return []
6
7     rutas = Ruta.obtener_por_ciudad(ciudad.id)
8     conexiones = []
9
10    for ruta in rutas:
11        ciudad_conectada = ruta.get_ciudad_conectada(ciudad.id)
12        if ciudad_conectada:
13            conexiones.append((ciudad_conectada.nombre, float(ruta.costos)))
14
15    return conexiones
```

4.6 Manejo de Errores

4.6.1 Casos de Error Contemplados

- **NetworkXNoPath:** No existe camino entre origen y destino
- **Ciudades inexistentes:** Validación previa de entrada
- **Grafo vacío:** Verificación de datos en base de datos

4.6.2 Respuestas del Sistema

```
1 # Respuesta exitosa:
2 {
3     "camino": ["Quito", "Ambato", "Riobamba", "Guayaquil"],
4     "costo": 350.5,
5     "valido": True,
6     "ciudades_costeras_en_ruta": ["Guayaquil"]
7 }
8
9 # Respuesta de error:
10 {
11     "camino": [],
12     "costo": None,
13     "valido": False,
14     "ciudades_costeras_en_ruta": []
15 }
```

4.7 Visualización del Resultado

4.7.1 Generación de Imagen con Camino Resaltado

```
1 def grafo_a_imagen_camino(camino):
2     """Genera la imagen del grafo con el camino resaltado"""
3     G = construir_grafo()
4     pos = nx.spring_layout(G, seed=8)
5
6     # Resaltar el camino encontrado por Dijkstra
7     if camino and len(camino) > 1:
8         edges_camino = list(zip(camino[:-1], camino[1:]))
9         nx.draw_networkx_edges(G, pos, edgelist=edges_camino,
10                               edge_color='red', width=4)
11         nx.draw_networkx_nodes(G, pos, nodelist=camino,
12                                node_color='orange', node_size=2200)
```

4.8 Ejemplo de Uso Completo

```
1 # 1. Usuario selecciona ciudades
2 origen = "Quito"
3 destino = "Guayaquil"
4
5 # 2. Validar entrada
6 valido, mensaje = validar_ciudades_existen(origen, destino)
7 if not valido:
8     return {"error": mensaje}
9
10 # 3. Calcular ruta óptima
11 resultado = camino_optimo_con_costera(origen, destino)
12
13 # 4. Resultado:
14 #{
15 #   "camino": ["Quito", "Latacunga", "Ambato", "Riobamba", "Guayaquil"],
16 #   "costo": 270.0,
17 #   "valido": True,
18 #   "ciudades_costeras_en_ruta": ["Guayaquil"]
19 # }
```

Esta implementación aprovecha la robustez y eficiencia de NetworkX mientras mantiene la flexibilidad para trabajar con datos dinámicos desde la base de datos.

5. Funcionalidades del Sistema

5.1 Gestión de Datos

- **CRUD de Ciudades:** Crear, leer, actualizar y eliminar ciudades
- **CRUD de Rutas:** Gestión de conexiones entre ciudades con sus distancias
- **Validación de Datos:** Verificación de integridad y consistencia

5.2 Cálculo de Rutas

- **Ruta Más Corta:** Utilizando algoritmo de Dijkstra
- **Distancia Total:** Cálculo preciso de kilómetros
- **Ciudades Intermedias:** Lista ordenada de paradas en la ruta óptima

5.3 Visualización

- **Interfaz Web Intuitiva:** Formularios para selección de origen y destino
- **Resultados Detallados:** Distancia total, ciudades intermedias, tiempo estimado
- **Representación del Camino:** Visualización step-by-step de la ruta

6. Flujo de Funcionamiento

6.1 Proceso de Cálculo de Rutas

1. **Entrada del Usuario:** Selección de ciudad origen y destino
2. **Construcción del Grafo:** Carga de ciudades y rutas desde base de datos
3. **Aplicación de Dijkstra:** Ejecución del algoritmo para encontrar camino óptimo
4. **Procesamiento de Resultados:** Formateo de la ruta y cálculo de totales
5. **Presentación:** Muestra de resultados en interfaz web

6.2 Ejemplo de Uso

```
Usuario selecciona: Quito → Guayaquil

Sistema construye grafo:
{
  'Quito': {'Ambato': 120, 'Latacunga': 90},
  'Ambato': {'Quito': 120, 'Riobamba': 50, 'Latacunga': 45},
  'Latacunga': {'Quito': 90, 'Ambato': 45, 'Riobamba': 65},
  'Riobamba': {'Ambato': 50, 'Latacunga': 65, 'Guayaquil': 180},
  'Guayaquil': {'Riobamba': 180}
}

Dijkstra encuentra: Quito → Latacunga → Ambato → Riobamba → Guayaquil
Distancia total: 270 km
```

7. Base de Datos

7.1 Esquema de Tablas

```
1  -- Tabla de provincias
2  CREATE TABLE provincias (
3      id INTEGER PRIMARY KEY,
4      nombre VARCHAR(100) NOT NULL
5  );
6
7  -- Tabla de ciudades
8  CREATE TABLE ciudades (
9      id INTEGER PRIMARY KEY,
10     nombre VARCHAR(100) NOT NULL,
11     provincia_id INTEGER,
12     latitud DECIMAL(10,8),
13     longitud DECIMAL(11,8),
14     FOREIGN KEY (provincia_id) REFERENCES provincias(id)
15 );
16
17 -- Tabla de rutas
18 CREATE TABLE rutas (
19     id INTEGER PRIMARY KEY,
20     ciudad_origen_id INTEGER,
21     ciudad_destino_id INTEGER,
22     distancia DECIMAL(10,2) NOT NULL,
23     tiempo_estimado INTEGER,
24     FOREIGN KEY (ciudad_origen_id) REFERENCES ciudades(id),
25     FOREIGN KEY (ciudad_destino_id) REFERENCES ciudades(id)
26 );
```

8. Tecnologías Utilizadas

- **Backend:** Python 3.13.5, Flask
- **Base de Datos:** MySQL
- **Frontend:** HTML5, CSS3, JavaScript
- **Librerías:**
 - Flask-SQLAlchemy (ORM)
 - Flask-Login (Autenticación)
 - Jinja2 (Plantillas)

9. Consideraciones de Rendimiento

9.1 Optimizaciones para Dijkstra

- **Terminación Temprana:** El algoritmo se detiene al encontrar el destino
- **Caché de Grafos:** Evita reconstrucción innecesaria del grafo
- **Validación Previa:** Verifica que origen y destino existan antes del cálculo

9.2 Escalabilidad

- Para grafos grandes, se puede implementar Dijkstra con heap (priority queue)
- Posibilidad de implementar A* para casos específicos con heurística
- Base de datos indexada para consultas rápidas

10. Conclusiones

En conclusión, el desarrollo de este sistema ha permitido integrar de forma efectiva múltiples componentes tecnológicos para resolver un problema real de planificación de rutas. La combinación de una arquitectura modular basada en el patrón MVC, una interfaz web amigable con Flask y HTML, junto con la robustez del algoritmo de Dijkstra implementado mediante la biblioteca NetworkX, da como resultado una herramienta poderosa y escalable. Además, la generación de representaciones gráficas y reportes PDF aporta valor agregado al usuario final, facilitando la interpretación y documentación de los resultados. Este proyecto no solo representa un ejercicio académico completo, sino también una base sólida para futuras mejoras o aplicaciones reales en ámbitos de transporte, logística o planificación territorial.

10.1 Fortalezas

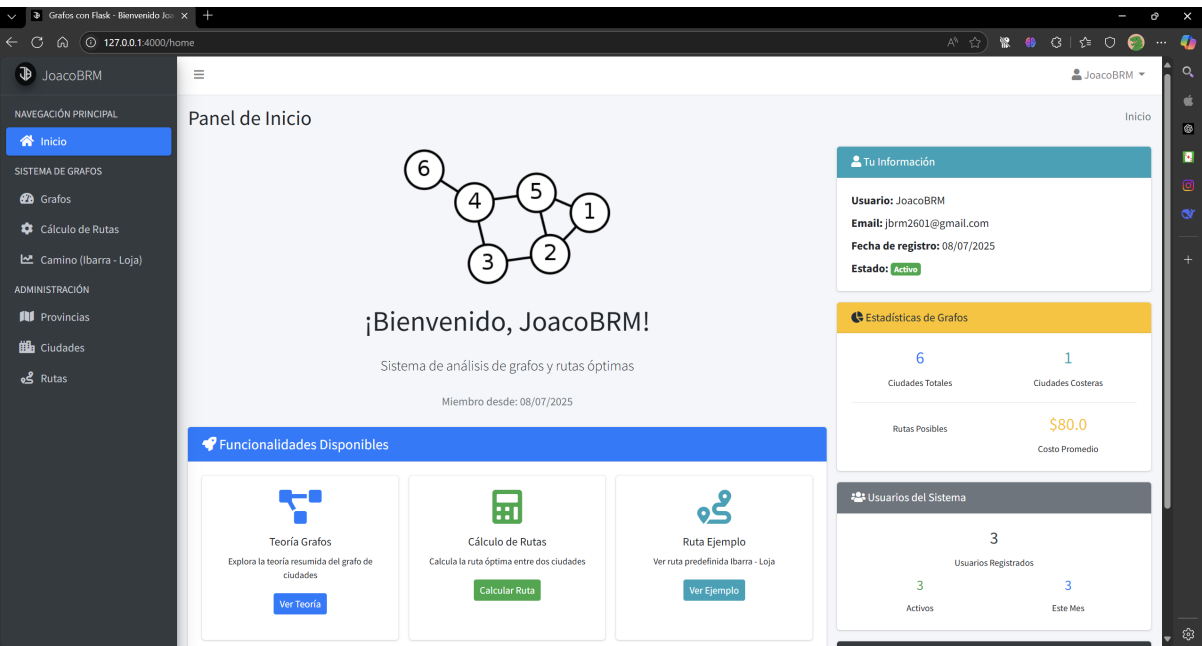
- Implementación correcta y eficiente de Dijkstra
- Garantía de encontrar la ruta más corta
- Interfaz de usuario clara e intuitiva
- Código modular y mantenible

10.2 Aplicaciones Prácticas

- Sistemas de navegación GPS
- Planificación de rutas logísticas
- Optimización de costos de transporte
- Análisis de redes de comunicación

11. Anexos

11.1 Panel de Inicio



11.2 Introducción a los Grafos



11.3 Calculadora de Rutas

The screenshot shows the 'Calculadora de Rutas' web application. The interface is in Spanish and features a sidebar with navigation links: Inicio, Sistema de Grupos, Grupos, Cálculo de Rutas (highlighted), Camino Ibarra - Loja, Administración, Provincias, Ciudades, and Rutas. The main content area is titled 'Calculadora de Rutas' and includes a search bar for the 'camino más económico'. Below the search bar, there are dropdown menus for 'Ciudad de origen' (Ambato) and 'Ciudad de destino' (Guayaquil). A 'Calcular' button is present. The results section, titled 'Resultado del Cálculo', displays the following information: Ruta: Ambato → Salcedo → Guayaquil; Costo total: \$1.180,0; Paradas: 2 conexiones; Tiempo estimado: 200,0 horas; Calculado: 16/07/2025 12:48. A green banner indicates 'El camino incluye una ciudad costera.' and a red button 'Reportar a PDR' is visible. Below the text, a graph shows the route between nodes. At the bottom, there are links for 'Ver Ruta Fija Ibarra - Loja' and 'Aprender sobre Grupos'.

11.4 Ruta Fija: Ibarra → Loja

The screenshot shows the 'Ruta Óptima: Ibarra → Loja' web application. The interface is in Spanish and features a sidebar with navigation links: Inicio, Sistema de Grupos, Grupos, Cálculo de Rutas, Camino Ibarra - Loja (highlighted), Administración, Provincias, Ciudades, and Rutas. The main content area is titled 'Ruta Ejemplo' and includes a search bar for the 'Ruta Óptima: Ibarra → Loja'. Below the search bar, there are dropdown menus for 'Ciudad de origen' (Ibarra) and 'Ciudad de destino' (Loja). A 'Calcular' button is present. The results section, titled 'Resultado del Cálculo', displays the following information: Ruta: Ibarra → Guayaquil → Salcedo → Loja; Costo total: \$350,0; Paradas: 3 conexiones; Tiempo estimado: 700,0 horas; Calculado: 16/07/2025 12:48; Tipo: Ruta óptima predefinida del sistema. A green banner indicates 'El camino incluye una ciudad costera.' and a red button 'Exportar a PDF' is visible. Below the text, a graph shows the route between nodes. At the bottom, there are links for 'Calcular Otra Ruta' and 'Aprender sobre Grupos'.

11.5 Administrar Provincias

JoacoBRM

NAVEGACIÓN PRINCIPAL

Inicio

SISTEMA DE GRAFOS

Grafos

Cálculo de Rutas

Camino (Ibarra - Loja)

ADMINISTRACIÓN

Provincias

Ciudades

Rutas

127.0.0.1:4000/admin/provincias

JoacoBRM

Inicio / Provincias

Administrar Provincias

Agregar Nueva Provincia

Nombre de la Provincia *

Ingrese el nombre de la provincia

+ Agregar Provincia

Solo se permiten letras y espacios

Provincias Registradas

ID	Nombre	Ciudades	Acciones
11	Cotopaxi	2 ciudades	<div>Editar Eliminar</div>
10	Guayas	1 ciudades	<div>Editar Eliminar</div>
12	Imbabura	1 ciudades	<div>Editar Eliminar</div>
13	Loja	1 ciudades	<div>Editar Eliminar</div>
9	Tungurahua	1 ciudades	<div>Editar Eliminar</div>

Administrar Ciudades

Ver Rutas

Volver al Inicio

Copyright © 2014-2025 JoacoBRM. All rights reserved.

PUCESA - Programación IV

11.6 Administrar Ciudades

JoacoBRM

NAVEGACIÓN PRINCIPAL

Inicio

SISTEMA DE GRAFOS

Grafos

Cálculo de Rutas

Camino (Ibarra - Loja)

ADMINISTRACIÓN

Provincias

Ciudades

Rutas

127.0.0.1:4000/admin/ciudades

JoacoBRM

Inicio / Ciudades

Administrar Ciudades

Agregar Nueva Ciudad

Nombre de la Ciudad *

Ingrese el nombre de la ciudad

Provincia *

Seleccione una provincia

Características

☐ Es ciudad costera

Solo se permiten letras y espacios

Conexiones con otras ciudades (No Dirigidas)

Seleccione las ciudades a las que se conectará esta nueva ciudad y especifique el costo de la ruta.

Seleccione una ciudad

Costo

+ -

+ Agregar Ciudad

Ciudades Registradas

ID	Nombre	Provincia	Tipo	Rutas	Acciones
9	Ambato	Tungurahua	Interior	2 conexiones	<div>Editar Eliminar</div>
10	Guayaquil	Guayas	Costera	0 conexiones	<div>Editar Eliminar</div>
14	Ibarra	Imbabura	Interior	1 conexiones	<div>Editar Eliminar</div>
11	Latacunga	Cotopaxi	Interior	1 conexiones	<div>Editar Eliminar</div>
13	Loja	Loja	Interior	1 conexiones	<div>Editar Eliminar</div>
12	Salcedo	Cotopaxi	Interior	2 conexiones	<div>Editar Eliminar</div>

Administrar Provincias

Ver Rutas

Volver al Inicio

11.7 Administración de Rutas

JoacoBRM

NAVEGACIÓN PRINCIPAL

Inicio

SISTEMA DE GRAFOS

Grafos

Cálculo de Rutas

Camino (Ibarra - Loja)

ADMINISTRACIÓN

Provincias

Ciudades

Rutas

Conexiones del Sistema

Inicio / Conexiones

Todas las Conexiones Registradas

+ Añadir Nueva Ruta

Total: 7 conexiones

ID	Ciudad A	Provincia A	Ciudad B	Provincia B	Costo	Tipo	Acciones
28	Ambato	Tungurahua	Guayaquil	Guayas	\$200.00	Mista Conexión no dirigida	
29	Ambato	Tungurahua	Latacunga	Cotopaxi	\$50.00	Interprovincial Conexión no dirigida	
30	Salcedo	Cotopaxi	Ambato	Tungurahua	\$10.00	Interprovincial Conexión no dirigida	
31	Salcedo	Cotopaxi	Guayaquil	Guayas	\$100.00	Mista Conexión no dirigida	
32	Latacunga	Cotopaxi	Salcedo	Cotopaxi	\$30.00	Interprovincial Conexión no dirigida	
33	Loja	Loja	Salcedo	Cotopaxi	\$50.00	Interprovincial Conexión no dirigida	
34	Ibarra	Imbabura	Guayaquil	Guayas	\$200.00	Mista Conexión no dirigida	

Total Conexiones

7

Ciudades Conectadas

6

Costo Promedio

\$91.43

Conexiones Costeras

3

Administrar Provincias

Administrar Ciudades

Volver al Inicio

11.8 Enlace al repositorio del proyecto

https://github.com/JoacoBRM/Proyecto_Final_JB