

PROYECTO ALGORITMOS

INTEGRANTES:
SANDY TANICUCHI
DAVID PILATUÑA
RODDIK VILLA
ANDERSON ESTRADA



POLIDEDELIVERY

SISTEMA INTELIGENTE DE
DISTRIBUCION DE PAQUETES



PROBLEMA A RESOLVER

Optimizar entrega de paquetes entre centros
y puntos de entrega para reducir costos y
mejorar tiempos

Explicamos el dolor real: rutas que
aumentan costos y tiempos. Nuestra
meta es minimizar eso



OBJETIVOS DEL PROYECTO

Objetivos principales:

- Implementar un sistema que lea datos de centros y rutas desde archivos.
- Calcular rutas más económicas usando Dijkstra.
- Organizar centros por región (árbol) y mostrar matriz de costos.

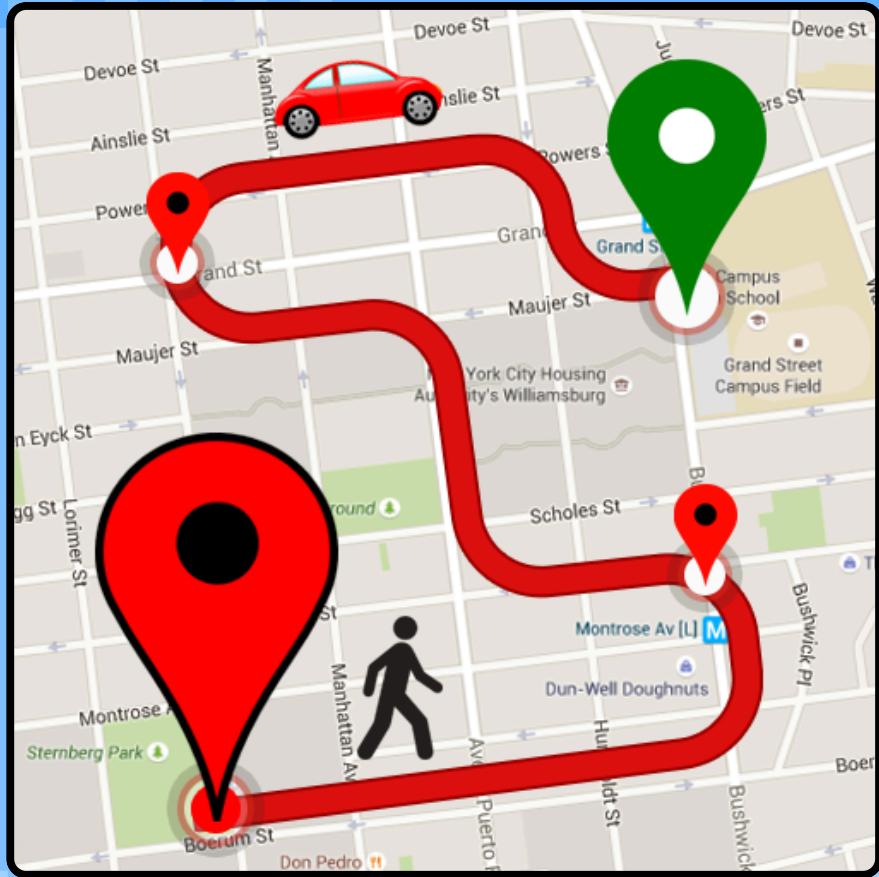


ALCANCE DEL PROYECTO

El proyecto PoliDelivery tiene como alcance el desarrollo de un sistema de gestión logística que permite administrar centros de distribución, rutas de transporte y usuarios mediante una aplicación desarrollada en Python y ejecutada en entorno de consola.

Dentro del alcance funcional, el sistema permite:

- Registrar y almacenar información de centros de distribución y rutas.
- Calcular la ruta más económica entre dos centros utilizando algoritmos de grafos.
- Visualizar recorridos mediante algoritmos BFS y DFS.
- Representar la estructura geográfica mediante un árbol de regiones.
- Mostrar matrices de costos y ordenar o buscar centros de manera eficiente.



TECNOLOGÍAS UTILIZADAS

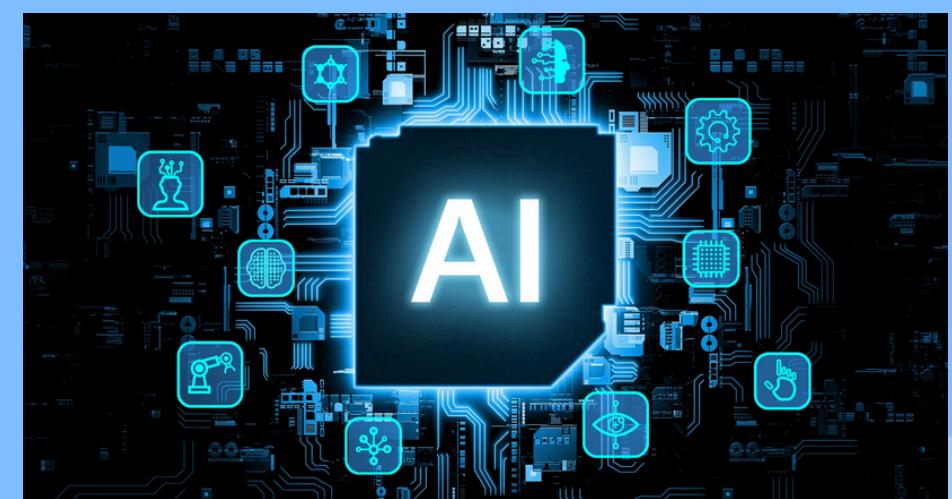
Lenguaje de programación: Python



Internet



Tecnologías tipo IA



REQUISITOS CLAVE:

1 Archivos: `centros.txt`, `rutas.txt`, `usuario.txt`

2 Funcionalidades: registrar/ingresar usuarios, agregar centros y rutas, calcular ruta óptima, mostrar árbol y matriz.

Roles: Administrador - Cliente

```
def registrar_usuario():
    print("\n--- REGISTRO DE USUARIO ---")
    nombre = input("Nombre: ")
    apellido = input("Apellido: ")
    cedula = input("Cédula: ")
    edad = input("Edad: ")
    email = input("Correo (nombre.apellido@gmail.com): ")
    password = input("Contraseña: ")
    if not contraseña_segura(password):
        print("Contraseña insegura")
        return
    with open("usuarios.txt", "a") as f:
        f.write(f"{nombre},{apellido},{cedula},{edad},{email},{password},CLIENTE\n")
    print("Usuario registrado correctamente")
```

ARCHIVOS Y ARQUITECTURA

- `usuarios.txt` ↔ autenticación
- `centros.txt` ↔ datos de nodos
- `rutas.txt` ↔ aristas (distancia/costo)

```
def iniciar_sesion():
    print("\n--- INICIO DE SESIÓN ---")
    email = input("Correo: ")
    password = input("Contraseña: ")
    if not os.path.exists("usuarios.txt"):
        print("No hay usuarios registrados")
        return None
    with open("usuarios.txt", "r") as f:
        for linea in f:
            datos = linea.strip().split(',')
            if datos[4] == email and datos[5] == password:
                print("Sesión iniciada")
                return datos[6]
    print("Credenciales incorrectas")
    return None
```

ESTRUCTURAS DE DATOS USADAS



- Diccionarios (centros)
- Listas/matrices (matriz de costos)
- Grafos (listas de adyacencia)
- Árboles (región → subregión → centro)
- Heap / cola de prioridad

```
def leer_centros():
    centros = {}

    if os.path.exists("centros.txt"):
        with open("centros.txt", "r") as f:
            for linea in f:
                linea = linea.strip()
```



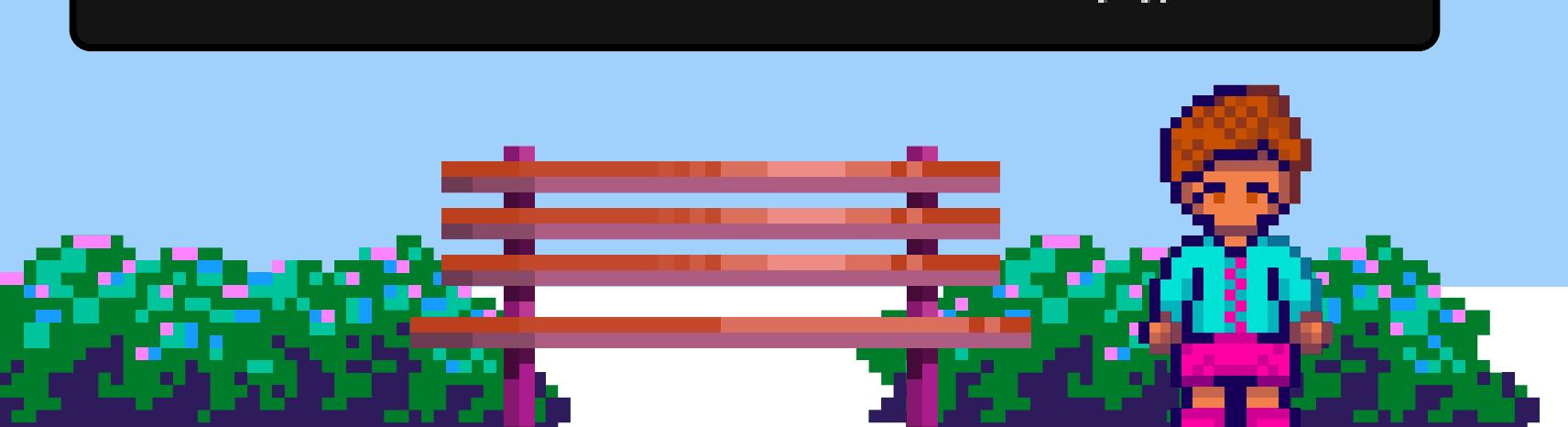
Proyecto Final > Proyecto.py > contraseña_segura

```
1 import os
2 import heapq
3 from collections import deque
4 import re
```

```
def leer_rutas():
    grafo = {}
    if os.path.exists("rutas.txt"):
        with open("rutas.txt", "r") as f:
            for linea in f:
                o, d, dist, costo = linea.strip().split(',')
                o, d = int(o), int(d)
                dist, costo = float(dist), float(costo)

                if o not in grafo:
                    grafo[o] = []
                if d not in grafo:
                    grafo[d] = []
                grafo[o].append((d, dist, costo))
                grafo[d].append((o, dist, costo))

    return grafo
```



ALGORITMO DE RUTA MÁS ECONÓMICA (DIJKSTRA)

Usamos heapq como cola de prioridad para minimizar costo.

```
def Camino_corto(grafo, inicio, fin):
    cola = [(0, inicio, [])]
    visitados = set()

    while cola:
        costo_acum, nodo, camino = heapq.heappop(cola)
        if nodo in visitados:
            continue
        camino = camino + [nodo]
        visitados.add(nodo)
        if nodo == fin:
            return costo_acum, camino
        for vecino, _ in grafo.get(nodo, []):
            if vecino not in visitados:
                heapq.heappush(cola, (costo_acum + costo, vecino, camino))
    return None, None
```

BFS / DFS (RECORRIDOS)



BFS para exploración por niveles; DFS para exploración profunda.

```
def bfs(grafo, inicio):
    visitados = {inicio}
    cola = deque([inicio])
    visitados.add(inicio)
    orden = []
    while cola:
        nodo = cola.popleft()
        orden.append(nodo)

        for vecino, _ in grafo.get(nodo, []):
            if vecino not in visitados:
                visitados.add(vecino)
                cola.append(vecino)
    return orden
```

```
def dfs(grafo, inicio, visitados=None):
    if visitados is None:
        visitados = set()

    visitados.add(inicio)
    orden = [inicio]

    for vecino, _ in grafo.get(inicio, []):
        if vecino not in visitados:
            orden.extend(dfs(grafo, vecino, visitados))
    return orden
```

ÁRBOL DE REGIONES

(JERARQUÍA REGIÓN → SUBREGIÓN → CENTRO)

★ Usamos nodos para construir árbol y mostrar jerarquía.

```
class Nodo:  
    def __init__(self, nombre):  
        self.nombre = nombre  
        self.hijos = []
```

```
def mostrar_arbol(nodo, nivel = 0):  
    print(" " * nivel + "- " + nodo.nombre)  
    for hijo in nodo.hijos:  
        mostrar_arbol(hijo, nivel + 1)
```



★ Este bloque construye y muestra el árbol por región y subregión

```
def arbol_regiones():  
    raiz = Nodo("Ecuador")  
    regiones = {}  
    centros = leer_centros()  
    for id_centro, (nombre, region, subregion) in centros.items():  
        if region not in regiones:  
            regiones[region] = Nodo(region)  
            raiz.hijos.append(regiones[region])  
        sub_nodo = next((h for h in regiones[region].hijos if h.nombre == subregion), None)  
        if not sub_nodo:  
            sub_nodo = Nodo(subregion)  
            regiones[region].hijos.append(sub_nodo)  
            sub_nodo.hijos.append(Nodo(f"{nombre} (ID: {id_centro})))  
    return raiz
```

FUNCIONES DE MANTENIMIENTO

★ **Actualizar_centro()** y
Actualizar_ruta() validan inputs.

```
def actualizar_centro():
    id_actualizar = input("ID del centro a actualizar: ")

    if not os.path.exists("centros.txt"):
        print("No hay centros registrados")
        return

    with open("centros.txt", "r") as f:
        lineas = f.readlines()

    encontrado = False
    nueva_lineas = []
    for linea in lineas:
        partes = linea.strip().split(',')
        if len(partes) != 4:
            nueva_lineas.append(linea)
            continue

        if partes[0] == id_actualizar:
```

Permiten editar sin romper formato;
validan entradas numéricas en rutas

```
def actualizar_ruta():
    if not os.path.exists("rutas.txt"):
        print("No hay rutas registradas")
        return

    o = input("ID centro origen de la ruta a actualizar: ").strip()
    d = input("ID centro destino de la ruta a actualizar: ").strip()

    with open("rutas.txt", "r") as f:
        rutas = f.readlines()

    encontrado = False
    nueva_lineas = []
    for ruta in rutas:
        partes = ruta.strip().split(',')
        if len(partes) != 4:
            nueva_lineas.append(ruta)
            continue

        ro, rd, dist_actual, costo_actual = partes
        if (ro == o and rd == d) or (ro == d and rd == o):
            encontrado = True
            print(f"Valores actuales -> Origen: {ro}, Destino: {rd},\n"
                  f"distancia: {dist_actual}, costo: {costo_actual}\n"
                  "Nuevo valor para distancia: ")
            nueva_dist = input("Nuevo valor para distancia: ")
            nueva_lineas.append(ro + "," + rd + "," + nueva_dist + "," + costo_actual)

    with open("rutas.txt", "w") as f:
        f.writelines(nueva_lineas)
```

PRUEBAS Y RESULTADOS

★ Registro:

```
== POLIDELIVERY ==
1. Registrarse
2. Iniciar sesión
3. Salir
Ingrese una opcion: 1

--- REGISTRO DE USUARIO ---
Nombre: Anakin
Apellido: Skywalker
Cédula: 6969696969
Edad: 20
Correo (nombre.apellido@gmail.com): anakin.skywalker@epn.edu.ec
Contraseña: StarWars06
Usuario registrado correctamente
```

★ Inicio Sesion:

```
== POLIDELIVERY ==
1. Registrarse
2. Iniciar sesión
3. Salir
Ingrese una opcion: 2

--- INICIO DE SESIÓN ---
Correo: anakin.skywalker@epn.edu.ec
Contraseña: StarWars06
Sesión iniciada

--- Menu Cliente ---
1. Ver centros
2. Ruta mas economica
3. Ver recorrido BFS
4. Ver recorrido DFS
5. Salir
Ingrese una opcion: 
```

★ Inicio Sesion(Administrador):

```
== POLIDELIVERY ==
1. Registrarse
2. Iniciar sesión
3. Salir
Ingrese una opcion: 2

--- INICIO DE SESIÓN ---
Correo: admin
Contraseña: admin
Sesión iniciada

--- Menu Administrador ---
1. Agregar centro
2. Agregar ruta
3. Ver centros
4. Ver arbol de regiones
5. Ver matriz de costos
6. Ordenar centros por nombre
7. Buscar centro
8. Actualizar centro
9. Actualizar ruta
10. Eliminar centro
11. Eliminar Ruta
0. Salir
Ingrese una opción: 
```

★ AgregarCentro:

```
Ingrese una opción: 1
ID del centro: 4
Nombre del centro: Centro norte
Región: Costa
Subregión: Esmeraldas
Centro agregado
```

Ingrese una opción: 3

```
--- CENTROS DE DISTRIBUCIÓN ---
ID: 1 | Nombre: Fosh | Región: Sierra | Subregión: Pichincha
ID: 4 | Nombre: Centro norte | Región: Costa | Subregión: Esmeraldas
```

MENÚS Y ROLES

FLUJO ADMIN VS CLIENTE

```
def menu_admin():
    while True:
        print("\n--- Menu Administrador ---")
        print("1. Agregar centro")
        print("2. Agregar ruta")
        print("3. Ver centros")
        print("4. Ver arbol de regiones")
        print("5. Ver matriz de costos")
        print("6. Ordenar centros por nombre")
        print("7. Buscar centro")
        print("8. Actualizar centro")
        print("9. Actualizar ruta")
        print("10. Eliminar centro")
        print("11. Eliminar Ruta")
        print("0. Salir")
        opcion = input("Ingrese una opción: ")
        match opcion:
            case '1':
                agregar_centro()
            case '2':
                agregar_ruta()
            case '3':
                mostrar_centros()
```

```
def menu_cliente():
    while True:
        print("\n--- Menu Cliente ---")
        print("1. Ver centros")
        print("2. Ruta mas economica")
        print("3. Ver recorrido BFS")
        print("4. Ver recorrido DFS")
        print("5. Salir")
        op = input("Ingrese una opcion: ")

        if op == '1':
            mostrar_centros()
        elif op == '2':
            grafo = leer_rutas()
            inicio = int(input("Centro inicio: "))
            fin = int(input("Centro destino: "))
            costo, camino = Camino_corto(grafo, inicio, fin)
            if camino:
                print("Ruta óptima:", camino)
                print("Costo total:", costo)
            else:
                print("No existe ruta")
        elif op == '3':
```

LIMITACIONES Y TRABAJO FUTURO

“Actualmente PoliDelivery guarda la información en archivos .txt, lo cual es suficiente para un proyecto académico, pero no es escalable para sistemas grandes.”

Limitación:

- No hay control de concurrencia
- No hay consultas complejas
- No hay seguridad real

Mejora Futura:

- Migrar de archivos .txt a una base de datos
- Como mejora futura, el sistema podría usar una base de datos como SQLite o MySQL para mayor seguridad y escalabilidad.”

POSIBLES MODIFICACIONES Y MEJORAS

Mejora en Seguridad

- Evitar el almacenamiento de contraseñas en texto plano.
- Validar el formato de correos electrónicos y datos sensibles.

Mejor Gestión de Archivos

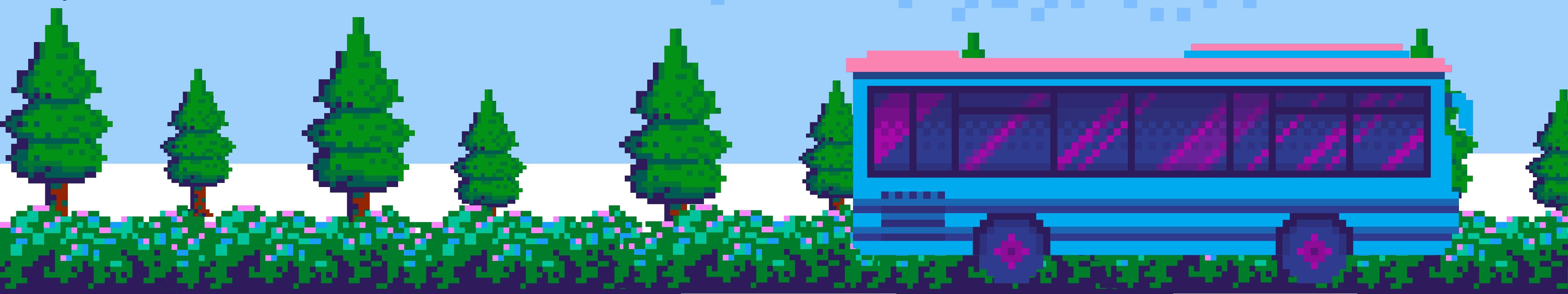
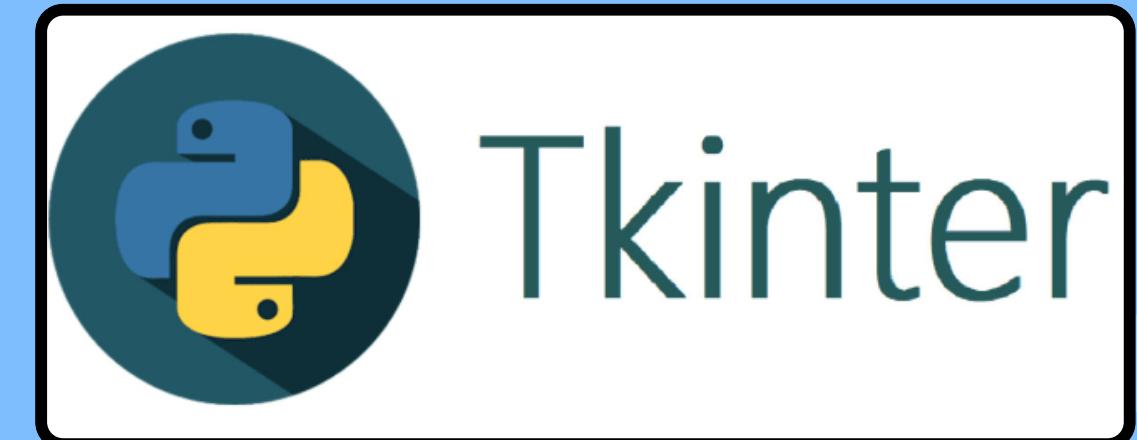
- Separar la información en diferentes archivos por tipo de dato.
- Manejo de errores al leer archivos corruptos o mal formateados.
- Implementar copias de seguridad automáticas.

Control de Usuarios y Roles

- Evitar registros duplicados por correo electrónico.
- Permitir más roles en el sistema (por ejemplo: operador, supervisor).
- Implementar cierre de sesión.

Interfaz Gráfica

- Migrar el sistema de consola a una interfaz gráfica (GUI).
- Uso de frameworks como Qt o Tkinter.
- Visualización gráfica de rutas y centros.

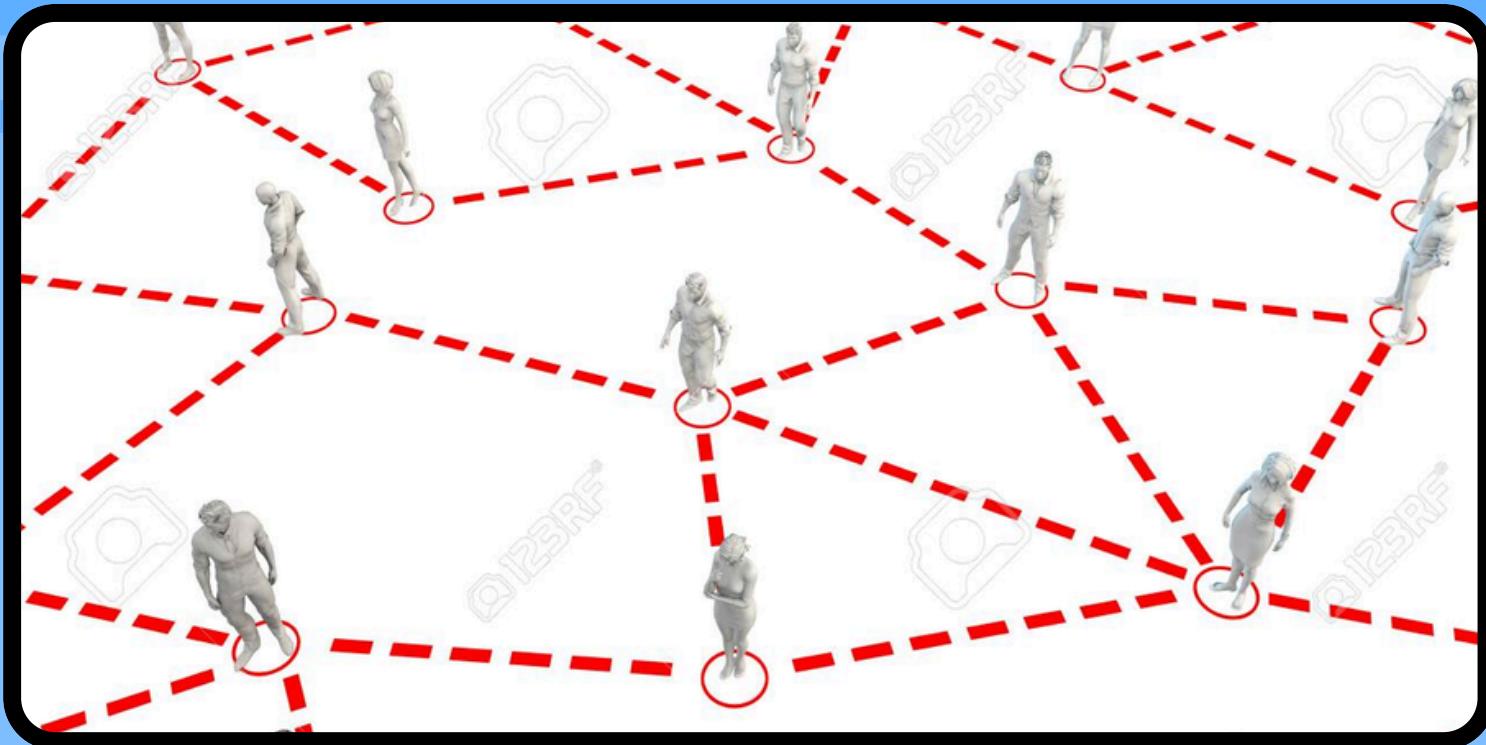


CONCLUSIÓN

El sistema PoliDelivery demuestra cómo los conceptos fundamentales de programación pueden aplicarse de manera práctica para resolver problemas reales del ámbito logístico. A lo largo del desarrollo del proyecto se integraron estructuras de datos, manejo de archivos, validación de usuarios y algoritmos de búsqueda y optimización de rutas, logrando un sistema funcional y organizado.

Este proyecto no solo cumple con los requerimientos académicos planteados, sino que también sienta las bases para una posible evolución hacia un sistema más robusto, incorporando mejoras como el uso de bases de datos, mayor seguridad en la gestión de usuarios, interfaces gráficas y una arquitectura más escalable.

En conclusión, PoliDelivery representa un paso importante en el aprendizaje de programación, demostrando la capacidad de aplicar la lógica computacional para desarrollar soluciones eficientes, estructuradas y con proyección a escenarios reales.





MUCHAS
GRACIAS

The background is a pixelated landscape with green trees, blue clouds, and a dark sky. In the foreground, there's a stone knight statue on the left, a stone castle tower on the right, and a stone chalice in the center. A small glowing yellow orb is visible in the sky.