MOVILÍZATE UN

Natalia Cely Callejas, Edward Aníbal Vásquez Guatapí, Jhohan Contreras, Johan Rodríguez, Felipe Chaves Delgadillo

No. de equipo de trabajo: 8

I. INTRODUCCIÓN

La movilidad dentro del campus de la Universidad Nacional presenta diversos problemas, sobre todo para las personas nuevas, debido a su extensión y la necesidad de movilizarse entre edificios durante las clases; por esto, nace la propuesta de hacer una aplicación que permita a los usuarios establecer un edificio de la universidad como inicio y otro como final, calcular la distancia entre estos puntos y ofrecer el camino más corto entre ellos para que sea recorrido y el viaje sea más sencillo.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

El campus de la Universidad Nacional de Colombia es uno de los más grandes de América Latina y está conformado por más de 150 edificios educativos, administrativos, de servicios en salud, entre otros. Por lo que la movilidad dentro de este es compleja para las personas que no lo conocen o quienes deben ir a un espacio nuevo. De forma adicional, la realización de obras, el cierre de caminos, dificultades en infraestructura por las lluvias y el uso requieren conocer caminos alternativos desde un lugar a otro. Además de los costos de tiempo que no conocer los caminos representa y puede incidir en su desarrollo como parte de la comunidad universitaria (llegar tarde a clase o al trabajo).

III. USUARIOS DEL PRODUCTO DE SOFTWARE

La aplicación web proyectada podrá ser usada por estudiantes, trabajadores y visitantes de la Universidad Nacional de Colombia de la sede Bogotá, quienes tengan acceso a un celular, tablet o computador con internet dentro de las instalaciones. Debido a la seguridad interna del espacio puede ser utilizada en cualquier momento y la temporada de mayor uso será en los primeros días de cada semestre en donde varios estudiantes nuevos desconocen el espacio, los estudiantes antiguos que deben asistir a clase en un edificio nuevo, los visitantes que no habiten el campus frecuentemente y los trabajadores nuevos.

No requiere ningún nivel de experticia más allá del uso básico del celular, puesto que la aplicación será muy sencilla e intuitiva.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

Historial de viajes.

El software almacenará los últimos 10 viajes hechos por el usuario desde el más reciente hasta el más antiguo, permitiendo visualizarlos en cualquier momento dentro de su propio menú. Además, se podrá efectuar una limpieza del mismo, dejándolo vacío.

Cuando un usuario realice la entrada de un dato desconocido (un edificio que no se encuentra en la base de datos o uno inexistente) se mostrará un mensaje de error en el que se pide revisar la escritura del nombre.

Una vez el usuario finalice un viaje, este se guardará como el más reciente dentro del historial y lo modificará en caso de ser necesario, como es explicado en las siguientes condiciones:

- Si ya existen diez viajes almacenados, se eliminará el más antiguo para ser reemplazado por el nuevo dato.
- En cualquier otro caso, es decir, cuando la cantidad de viajes sea menor a diez, simplemente se agregará la información y aumentará en uno el contador que mantiene el tamaño usado del historial.

El manejo de datos dentro de la funcionalidad se realiza de la siguiente manera:

Operación	Descripción
Creación.	Al efectuar un viaje, se crea un nuevo objeto, que es el dato a agregar.
Actualización.	Cada vez que se agrega un dato al historial se debe comprobar si está lleno o no.
Eliminación.	El usuario puede vaciar el historial, eliminando así la información contenida en este.
Consulta.	La información dentro del historial podrá consultarse en cualquier momento al presionar un botón.
Almacenamiento.	Los datos se almacenarán dentro de una estructura de datos secuencial y lineal.

Calcular distancia.

Para determinar la ruta más corta entre dos puntos, uno de los requisitos es conocer cuál es la distancia entre ambos, y para esto, a su vez, es necesario conocer la ubicación de cada uno.

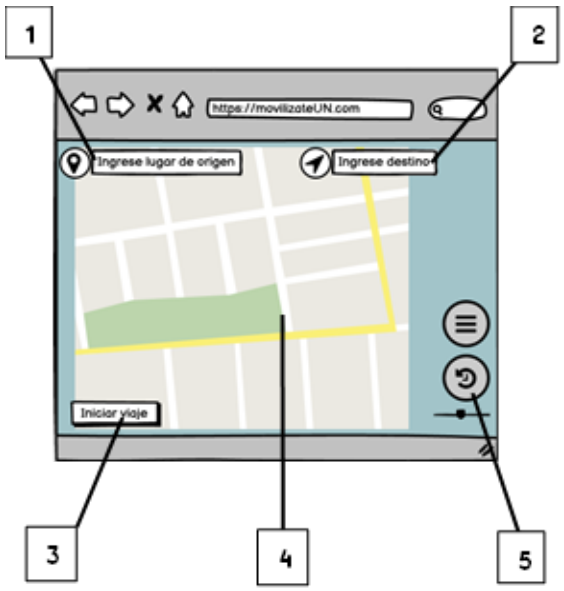
Al momento de realizar un viaje, el usuario deberá ingresar el punto de origen y el destino. Con esta información, el software buscará la ubicación de los puntos indicados en los datos que tendrá almacenados desde la creación del programa, y una vez encontrada, podrá proceder a hacer el cálculo y mostrará al usuario el resultado del mismo.

Dentro de esta funcionalidad, los datos serán usados de la siguiente manera:

Operación	Descripción
Creación.	Tanto el origen como el destino son datos insertados por el usuario
Eliminación	Una vez finalizado el procedimiento, la información de origen y destino ingresada se eliminarán, pasando a ser parte del historial
Búsqueda	Los puntos ingresados se buscarán dentro de la base de datos contenida por el software

Almacenamiento.	Durante el cálculo y su posterior muestra al usuario, tanto los datos ingresados como el resultado serán almacenados

V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR



1. Barra de ingreso de texto. Aquí se ingresará el lugar de inicio del viaje. Se compone de la barra de texto y un icono que permite identificarlo.

2. Barra de ingreso de texto. Aquí se ingresará el sitio de destino. Al igual que la anterior, se compone de la barra de texto y un icono.

3. Botón de inicio de viaje. Únicamente se habilitará al rellenar ambas barras de texto, al presionarlo el software iniciará la búsqueda de la ruta más óptima.

4. Mapa. Mostrará al usuario la mejor ruta posible entre los dos puntos previamente determinados en una vista gráfica

5. Botón de historial. Al presionarlo se desplegará un menú con los últimos 10 viajes realizados

VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

El software se desarrollará en lenguaje Python, para esto se utilizarán los IDE's (Entorno de Desarrollo Integrado) Pycharm de la empresa JetBrains y Visual Studio de la empresa Microsoft, a partir de estos IDE's se realiza una conexión al repositorio en GitHub, donde se registrarán los cambios en el software y se subirán las versiones más recientes del prototipo y el producto final, con el valor agregado de la facilidad en la intervención grupal y el trabajo en equipo.

Se piensa el producto final como una aplicación web, por lo que se programará para funcionar en el navegador web Google Chrome, a dicha aplicación podrán acceder los miembros de la comunidad universitaria, esto quiere decir que podrán acceder desde un celular, tablet o computador.

Para aspectos como el correcto funcionamiento del aplicativo, se tienen como requisito mínimo de los dispositivos:

- Procesador Intel Core i3 de 5.ª generación o superior, o procesador AMD Ryzen 3 serie 1000 o superior
- 2 GB de RAM

Sin embargo, esto puede provocar comportamiento inusual en el programa por la arquitectura del software y algoritmos que utiliza, por lo tanto, se da la especificación recomendada para los dispositivos:

- 4 GB de RAM

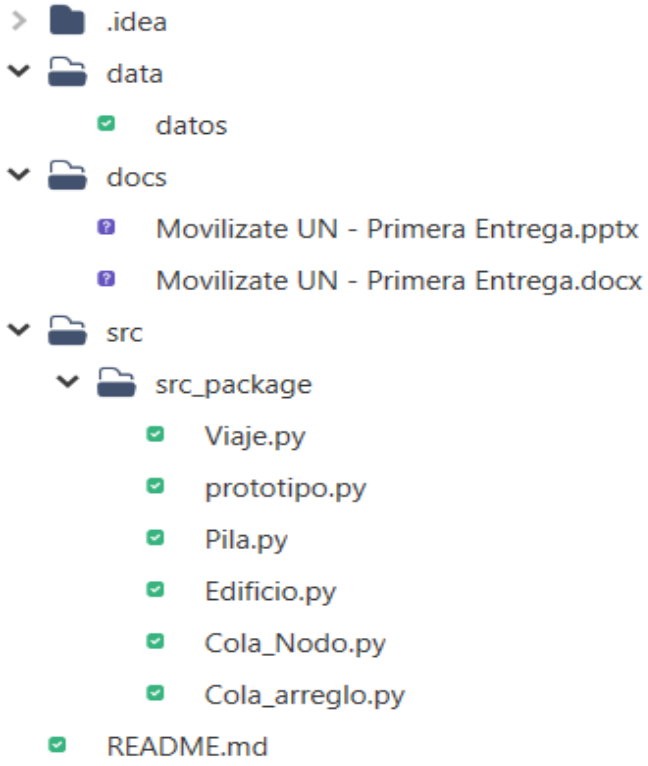
Esto garantizará un correcto funcionamiento y una experiencia agradable para el usuario.

VII. PROTOTIPO DE SOFTWARE INICIAL

El prototipo del software se encuentra en un repositorio de GitHub [1], el cual cuenta con una descripción del proyecto y las funcionalidades hasta ahora desarrolladas.

El código fuente del prototipo se encuentra en la carpeta "src" y se puede ejecutar con un intérprete básico de Python. En la carpeta "data" se encuentra la información con la que se prueba la funcionalidad del prototipo.

La organización del repositorio es el que se muestra a continuación:



VIII. DISEÑO, IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

Para este primer prototipo se diseñaron las funcionalidades "Historial de viajes" y "Calcular Distancia", las cuales se ejecutan mediante la implementación de arreglos, listas encadenadas, pilas y colas.

Para la funcionalidad "Calcular distancia" , se implementó la clase "Edificio", que contiene la información básica necesaria de cada uno de los puntos de origen o destino de los viajes a efectuar, como pueden ser:

- El nombre del edificio.
- El número del edificio.
- La latitud y longitud del edificio como referencia de posicionamiento geográfico.

Este último se puede interpretar como las coordenadas *x* y *y* correspondientes al edificio en el mapa de la universidad, posterior a esto, se implementó la clase "Viaje" que recibe como entradas el punto de origen, el punto marcado como destino que serán de tipo "Edificio" y la distancia entre ellos, la cual se calcula con un método interno de la clase. Este objeto "Viaje" se compone de un arreglo de tres elementos como son: origen, destino y distancia, y este último atributo de la clase posteriormente se retorna como respuesta al usuario.

Seguido de esto, se implementa la funcionalidad "Historial de viajes" en la cual, cada vez que un usuario haga un viaje con la aplicación, sea este creado con la clase "Viaje" anteriormente mencionada, se guardará en una lista enlazada con el fin de que la aplicación muestre al usuario cuando lo requiera desde el viaje más reciente hasta el más antiguo. Para temas de eficiencia en memoria, se limitará la lista enlazada a 10 espacios, es decir, solo se almacenarán los 10 viajes más recientes de cada usuario.

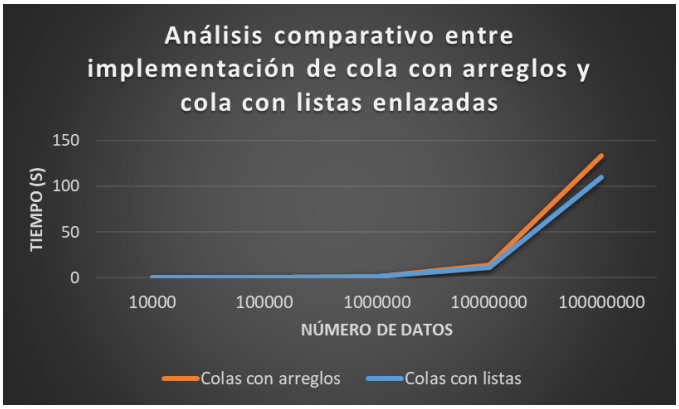
IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

Aunque la estructura de datos principal mediante la cual se espera desarrollar el proyecto es el grafo, debido a que la primera entrega requiere de la implementación de por lo menos una estructura de datos lineal, se optó por hacer uso de colas para el desarrollo de un historial que almacena como máximo los últimos diez viajes realizados por el usuario.

Para ejecutar las pruebas de este prototipo se instanciaron objetos pertenecientes a las clases edificio, viaje e historial. Esta última, como se mencionó anteriormente, implementa colas. Con el fin de determinar cuál era la mejor estructura, se efectuaron pruebas para contrastar la eficiencia al utilizar colas con arreglos y colas con listas. Los resultados obtenidos mediante el uso de la librería *time* fueron los siguientes [2]:

	Tiempo (s)	
Número de datos	**Cola con arreglo**	**Cola con listas**
10000	0.017004251	0.010010481
100000	0.142033339	0.11402607
1000000	1.483698368	1.145257473
10000000	13.61014199	11.21456051
100000000	133.3898618	110.166667

Gráficamente, los resultados se ven reflejados de la siguiente manera:



Si bien en el historial se utilizan fundamentalmente dos métodos de la cola (encolar y desencolar), por lo cual en teoría la complejidad algorítmica de estas acciones es constante, O(1) tanto para su implementación en listas enlazadas como en arreglos, si se tiene en consideración que la clase historial contará con una restricción de máximo diez ítems, lo que significa que por cada vez que se busque encolar y no se cumpla la restricción, se deberá primero desencolar y nuevamente encolar, la complejidad podría variar. De igual manera, el método *enqueue* recibe como parámetro un arreglo que contiene todos los datos de la instancia edificio, lo que significa que se harán operaciones internas que van más allá de las dos ya mencionadas [3].

Por lo anterior, a grandes volúmenes de datos el historial adquiere un comportamiento parabólico $O(n^2)$. Sin embargo, esto solo se presenta con valores superiores al millón de datos.

Por otra parte, la gráfica muestra que no existe una gran diferencia en el tiempo de ejecución entre la implementación de la cola con arreglo o con lista. Esto, sumado al hecho de que el historial almacenará como máximo diez registros, hace

viable el uso de cualquiera de las dos opciones mencionadas como estructura de datos.

X. INFORMACIÓN DE ACCESO AL VIDEO DEMOSTRATIVO DEL PROTOTIPO DE SOFTWARE

El video de la demostración del prototipo del software se encuentra en el siguiente link: https://drive.google.com/file/d/18hx-f8w8OHJM_HxQuGx9o12qWTpfOUry/view?usp=sharing

XI. ROLES Y ACTIVIDADES

Los roles en esta entrega fueron asignados de esta manera:

Integrante	Rol	Descripción de su rol
Felipe Chaves	Experto en tecnología	Seguimiento y revisión de los códigos implementados.
Natalia Cely	Lideresa	Manejo de tareas y funcionalidades.
	Coordinadora	Planeación de reuniones, observación del desarrollo y correcciones.
Jhohan Contreras	Investigador	Encargado de buscar la parte investigativa del código.
Johan Rodríguez	Investigador	Encargado de buscar la parte investigativa del código.
	Técnico	Encargado de resolver los errores finales, por su experiencia.
Edward Vásquez	Experto en tecnología	Seguimiento y revisión de los códigos implementados.
	Técnico	Encargado de resolver los errores finales, por su experiencia.

XII. DIFICULTADES Y LECCIONES APRENDIDAS

Adquisición de información. La información sobre las coordenadas de cada edificio no es de fácil acceso y se tuvo que investigar más a fondo para obtenerlas.

Cálculo de las distancias entre dos puntos. La utilización de las operaciones matemáticas para las distancias entre edificios tampoco fue fácil, ya que los puntos no se indican como distancias, sino como grados, minutos y segundos con respecto a un punto cardinal y una línea imaginaria de referencia en la tierra [4].

La implementación de la cola con arreglos. A pesar de que la cola implementando listas enlazadas se creó, el programa con instancias de colas con arreglos tuvo complicaciones en el desarrollo del código.

XIII. REFERENCIAS BIBLIOGRÁFICAS

[1] https://github.com/FelipeCh18/Proyecto_ED
[2] https://docs.python.org/3/library/time.html
[3] J. T. Streib y T. Soma, Guide to Data Structures. A Concise Introduction Using Java. Springer, 2017.
[4] https://es.wikipedia.org/wiki/F%C3%B3rmula_del_semiverseno