A otro lado

Víctor Daniel Arango

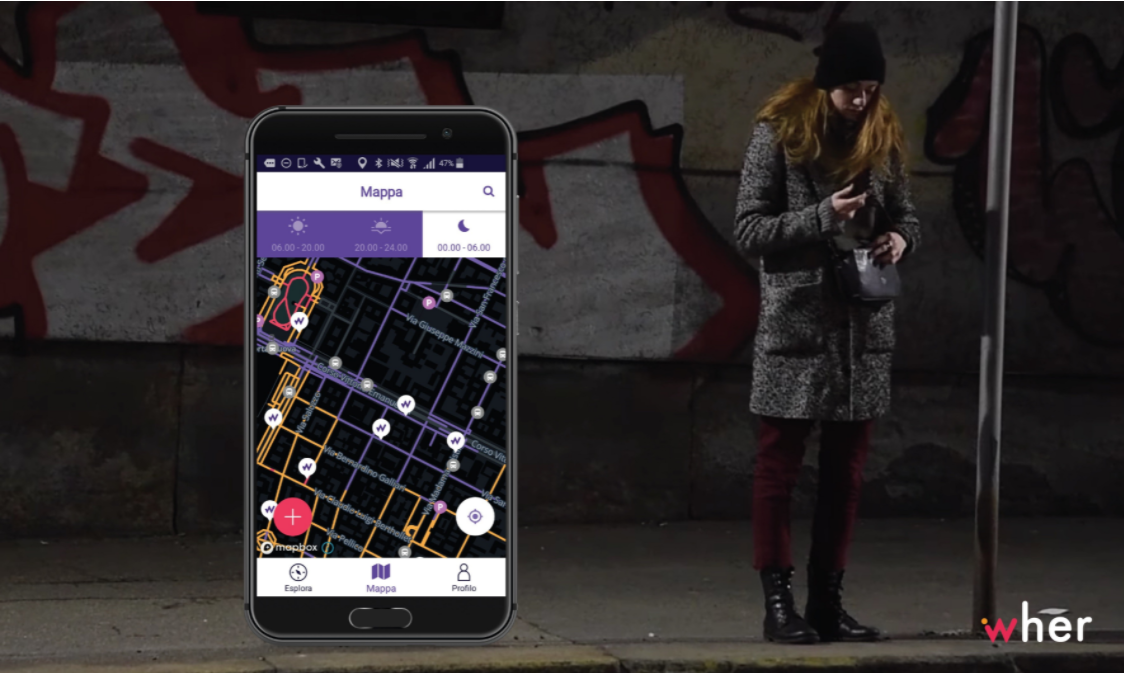
Thomas rivera Fernández



4 alternativas de solución

1. wher

La aplicación consiste en una red de amigas virtuales, que crece para hacer que la plataforma se vuelva cada vez más amplia y precisa.



Para acceder a la aplicación, primero hay que ‘loguearse’ a través de las redes sociales. Luego las usuarias se perfilan a sí mismas ingresando sus hábitos de movilidad y eligiendo la ciudad de interés. Luego aparecen los barrios y las calles sugeridas por la comunidad. Después de una breve descripción del área seleccionada, se muestran las evaluaciones dejadas por otras usuarias. Aparecerán de violeta las zonas seguras, amarillo las que deben cubrirse con cuidado y fucsia las que es mejor evitar. Entre los parámetros considerados, está el grado de aglomeración del área y el nivel de luminosidad.

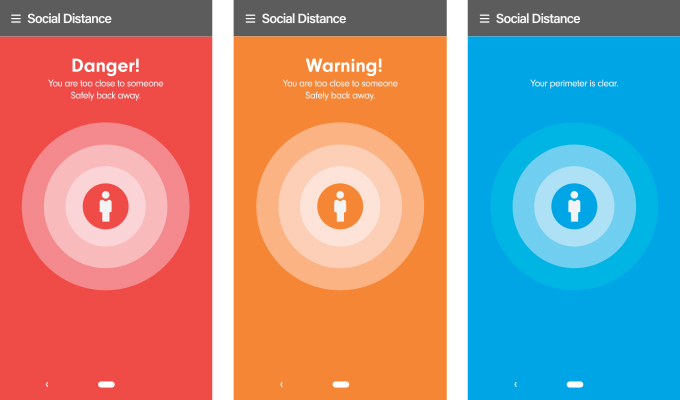
2. Sister

un grupo de mujeres ha desarrollado Sister, una app que te acompaña de camino a casa. Tiina Rinne, CMO de Wave y una de las creadoras de Sister, ha explicado que la aplicación busca “la ruta más segura con comisarías, tiene alarma disuasoria, botón de emergencia que incluye contacto con emergencias y SMS a los contactos guardados en la app con tu localización en tiempo real”.



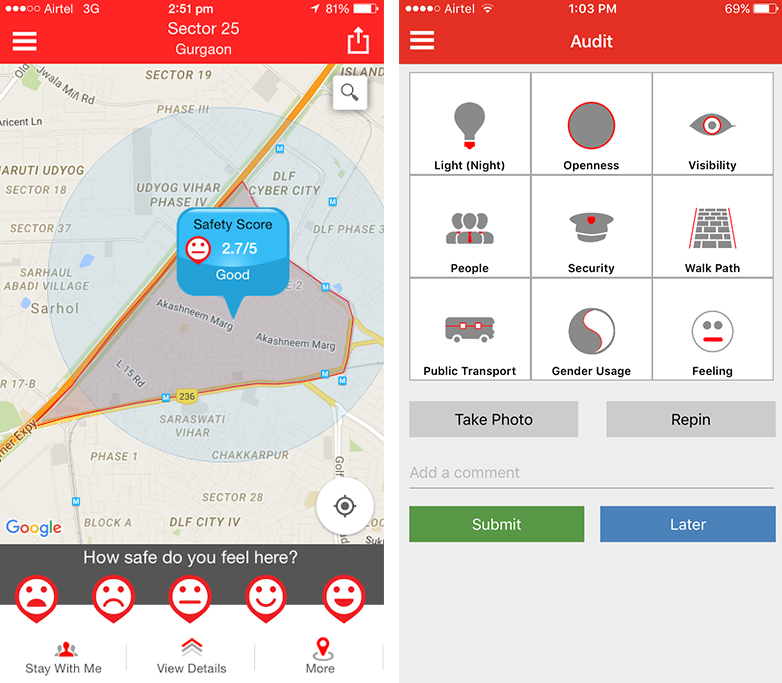
3. Safety

Esta aplicacion envía un mensaje y tu ubicación al contacto de seguridad si presionas un botón. Cuando se activa esta opción, se realizan dos imágenes (cámara frontal y cámara trasera) y se graba un vídeo. Estas pruebas se envían por correo electrónico. Los botones de emergencia son tres y cada uno es de un color (dependiendo de la gravedad del evento): verde para conocer tu posición, naranja para precaución y rojo para emergencia. Pero esta app no tiene la función de recomendar rutas más seguras por lo cual no es tan usada



4.Safetipin

Esta app activa la cámara frontal y la principal las cuales toman una foto cada 30 segundos y con geo localizadores te indican si la ruta es segura o no, luego los usuarios solo tienen que enviar sus informes evaluando los lugares en los que han vivido una situación de acoso, lo que permite evitar ciertas zonas consideradas peligrosas. Este contenido se vuelca en un mapa de información geo localizada que permite visualizar con rapidez las zonas más seguras de la ciudad.



2. 4 tipos de algoritmos

1. Algoritmo de Dijkstra

Este algoritmo utiliza un tipo de estructura de colas llamado cola de prioridad. Una cola de prioridad es una estructura de datos en la que los elementos se atienden en el orden indicado por una prioridad asociada a cada uno.

2. Algoritmo de Bellman-ford

El algoritmo de Bellman-Ford genera los caminos mínimos desde un nodo origen de un grafo al resto de nodos del mismo nodo. Luego soluciona el problema de la ruta más corta o camino mínimo desde un nodo origen, de un modo más general que el Algoritmo de Dijkstra, Y su mayor ventaja es que permite valores negativos en los arcos. El algoritmo devuelve un valor booleano, si encuentra un circuito o lazo de peso negativo. En caso contrario calcula y devuelve el camino mínimo.

3. Algoritmo de Prim

El algoritmo de Prim es un algoritmo que hace parte de la teoría de los grafos para hallar un árbol recubridor mínimo en un grafo conexo, no dirigido y cuyas aristas están etiquetadas.

Partiendo de un nodo del grafo examinamos todas sus aristas, es decir, todos los caminos posibles que tenemos y elegimos el más corto, desde el siguiente nodo tendremos otros caminos posibles y también nos quedarán los caminos posibles que teníamos desde el primer nodo y lo que tenemos que hacer es elegir el más corto, así sucesivamente hasta que hemos llegado a todos los nodos.

4. Algoritmo A\*

Es un algoritmo de búsqueda de rutas y recorrido de gráficos, lo que significa que está formulado en términos de gráficos ponderados, esto significa que a partir de un nodo de inicio específico de un gráfico, el algoritmo tiene como objetivo principal hallar un camino hacia el nodo de destino dado que tiene el menor costo, por consecuencia tiene menor distancia recorrida y menor tiempo. Lo hace manteniendo un árbol de caminos que se originan en el nodo de inicio y extendiendo esos caminos un borde a la vez hasta que se satisface su criterio de terminación.

*Como se planteó la solución:*

Nuestro principal propósito es ganar inventiva o hacer un muy buen proyecto, ya que el problema real que es el acoso en Medellín y este requiere un cambio social para poderse solucionar lo cual no podemos resolver tan fácil, por lo que este proyecto solo atenuara y ayudara un poco con esta problemática social

*Características de la solución:*

Para este proyecto utilizamos flask y pydeck, flask para poder servir archivos HTML, ponerlo en una página web y tenerlo como un servidor para que más y más personas puedan acceder a este sin tener que instalar ningún tipo de aplicación o de archivo para usar de este, y utilizamos pydeck para generar los mapas con los cuales se trabaja el proyecto, este viene en un librería que se llama Deck que es la que utilizan cientos de apps muy reconocidas como Uber con WebGL, en un principio se pensó en usar esta pero aun no se tienen los suficientes conocimientos de JavaScript.

Por lo tanto, con Flask sacamos y emitimos los archivos de HTML en una página y así con pydeck se generan y estos archivos web son los mapas generados. Este proyecto lo estamos trabajando desde un IDE Z para poder poner el proyecto en Python, este setup permite instalar todas las librerías y metadatos con mucha comodidad ,facilidad y felicidad ;-) .

En cuanto a las estructuras de datos estamos utilizando deque que son las listas enlazadas para poder hacer mucho más rápido el proceso de inserción de ciertos datos temporales con los que realizamos el código, además estamos trabajando con NetworkX para el grafo y usar ciertos algoritmos, también se está usando el Dataframe de Pandas para poder leer el archivo, leer los datos, generar más archivos y leerlo de forma remota.

Por otra parte, en cuanto a algoritmos estamos usando Dijkstra ya que es el mas popular,mejor en rendiminento y el que más nos gustó, y el algoritmo de Bellman-ford no lo vimos muy eficiente por es “lo mismo” a un djikstra pero con pesos negativos, en este caso todos los valores eran positivos por lo que este no daba ningun beneficio. Tambien probamos con A\* pero lo usamos por sobrecargava la memoria.

La idea futura es implementar un grafo propio (ya esta hecho para la tercera entrega) y también poderle bajar la complejidad de como procesa los datos y los archivos y los addiciona de nodos al grafo, ya que en este momento el proceso funciona de la siguiente manera, primero los datos son leídos con pandas y esto será con una complejidad + y luego se implementa una ciclo *for* para que convierta esos datos de Pandas a un nodo de NetworkX y de esta manera tendría una complejidad de ; Entonces lo que se va a hacer es que mientras vaya leyendo el archivo con una función típica de Python sin ninguna librería que vaya leyendo el archivo y vaya añadiendo la información al grafo.

*Explicacion de estructuras de datos usadas:*

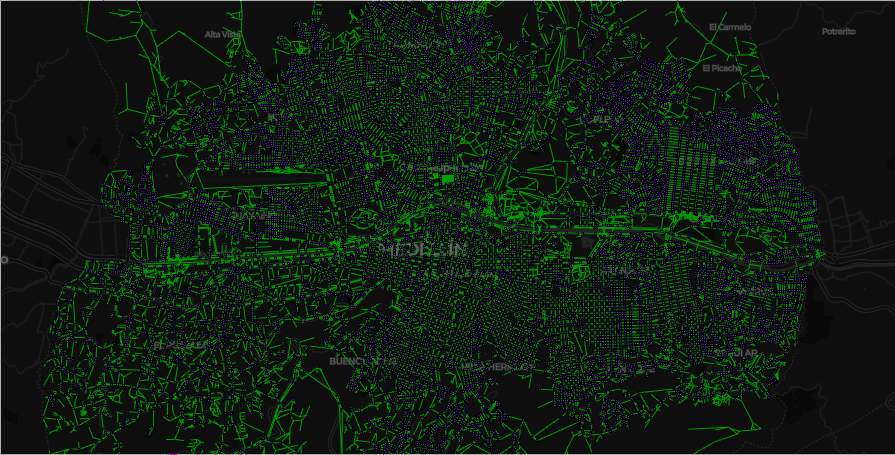
se usaron varias estructuras de datos vistas en el curso como: en un momento se uso listas enlasadas con deque, para hacer mas rapido el proseso de insercion.

se esta usando el dataframe de pandas que es como un diccionario de python, este se usa para leer el archivo csv dado por el docente de donde tenemos los datos y los convertimos json por facilidad para dibujar los datos en pydeck,ya una vez creado el json lo usamos en el codigo principal volviendolo a cargar con padnas como un diccionario, al leerlo con pandas suponemos que tiene una complejidad de O(n) y despues de tener los datos en el dataframe se pasa al grafo, la complejidad de este seria es O(n), el grafo que usamos el del networkx .

Se piensa bajar la complejidad del la insercion al grafo, añadiendo los nodos mientras va leyendo el archivio

nos sirvio de referencia imaginarnos medellin como un grafo para tener mas claro el problema

la imagen muestra las interecciones como nods y las calles como vertices

**

*Explicacion de algoritmos usados:*

se uso A\* , bellman-ford y dijikstra de networkX

**A\*** fue descartado por que se comia la memoria y se demoraba mucho

**bellman-ford** se uso y esta la posibildad de usarlo en el proyecto, lo descartamos por que no tenemos pesos negativos, a razon de esta observacion lo vimos ineficiente

**dijikstra** lo escojimos por popularidad y por que es mas util para este problema y nos funciono.

Dijikstra funciona de la siguente forma:

Antes de empesar hay que aclarar que dijikstra no funciona con pesos negativos, y que los pesos significa un algo que hace mas dificil o costoso, por lo que vamos a escoger lo menos costoso o mas facil.

recorre todos los posibles nodos desde un nodo inicial, observa los nodos vecinos y va guardando el camino con menor peso entre los nodos vecinos, y vuelve a repetir este proceso, al final quedamos con una lista con los caminos del nodo de inicio al nodo final con menos pesos, partir de esto podemos tenemos calculado un conjunto de mejores caminos entre 2 nodos, apartir de estos caminos definimos un camino mas grande y óptimo que sirve para definir la ruta final