**David Mora, Diego Romero, Sergio Prieto, Mateo Díaz**

**No. de Equipo Trabajo: 6**

BiciSegura

# [[1]](#footnote-1) INTRODUCCIÓN

# En este documento se explica el desarrollo, funcionalidad y resultados de la aplicación Bici Segura, una aplicación para que los bici-usuarios estén informados e informen a otros bici-usuarios sobre posibles focos de inseguridad en las vías, alimentando una base base de datos que ayudará a los usuarios de la aplicación a elegir el mejor camino para llegar a sus destinos

# DESCRIPCIÓN DEL PROBLEMA A RESOLVER

La problemática que queremos atacar mediante BiciSegura es el creciente número de hurtos de bicicletas en Bogotá en los últimos años, evidentemente bajar los índices de criminalidad esta fuera del alcance de nuestra plataforma. Sin embargo, podemos brindar una herramienta muy útil en la toma de decisiones con respecto a que rutas pueden tomar los biciusuarios hacia sus destinos, ya que no es un secreto que hay ciertos factores influyen en aumentar el riesgo de ser víctima de este delito.

Por esta razón nuestro objetivo es que nuestros usuarios puedan disminuir la probabilidad de ser víctimas de hurto de bicicleta a partir de una toma de decisiones inteligente que permita evitar los principales factores de riesgo proporcionando un trayecto seguro y eficiente.

# Usuario DEL PRODUCTO DE SOFTWARE

El usuario objetivo de BiciSegura son ciclistas de la ciudad de Bogota, especialmente trabajadores que usan la bicicleta como medio de transporte frecuente. No existe distinción de género. Nuestro producto es de uso frecuente ya que el usuario podrá consultar la ruta más adecuada hacia su destino cada vez que quiera movilizarse. Los usuarios también tendrán acceso al registro de hurtos de bicicletas recientes con motivo que servir de medio de información. Las funcionalidades principales a las que tendrá acceso el usuario son realización de reportes, planeación de rutas y grupos de movilización segura.

# REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

**Ingreso de usuario**

*Descripcion:*

La funcionalidad de ingreso le permite al usuario autenticar su identidad mediante un nombre de usuario y contraseña

*Accion iniciadores y comportamiento esperado:*

Al ejecutar la aplicación, primeramente, se pedirá autenticación de acceso al usuario mediante su nombre de usuario y contraseña

Requerimientos funcionales:

1. Mostrar el campo de nombre de usuario y contraseña
   1. En caso de que proporcione una contraseña o usuario invalido. Proporcionará el mensaje en pantalla.
   2. En caso de proporcionar una contraseña y usuario valido, se pasará a la pantalla principal.

**Reportes de hurto.**

*Descripcion:*

La funcionalidad de reporte se encarga principalmente de la operación de añadir un Nuevo reporte, eliminar uno ya existente y mostrar los reportes anteriores.

*Accion inciadores y comportamiento esperado:*

La principal acción iniciadora es acceder a la pestaña de Mis reportes donde se mostrará el historial de reportes del usuario, además de dos botones, “Añadir nuevo reporte” y “Eliminar reporte existente”.

*Requerimientos funcionales*:

1. Mostrar el botón “Añadir una nueva denuncia”
   1. Agregar una nueva denuncia a las denuncias del usuario. En caso de que el usuario intente añadir una denuncia invalidad se le informara por un mensaje en pantalla.
   2. Agregar un nuevo nodo a la pila de denuncias
2. Mostrar el botón “Eliminar una denuncia existente”
   1. Seleccionar la denuncia dentro de las denuncias del usuario.
   2. Remover esa denuncia de las denuncias del usuario.
   3. Buscar y Eliminar la denuncia de la lista de denuncias.

**Consultas de robos**

*Descripcion:*

La funcionalidad de “Recientes” le va a proporcionar al usuario la capacidad de consultar los robos más recientemente agregados, consultar la cantidad de robos y filtrar los robos por hora y lugar.

*Accion inciadores y comportamiento esperado:*

En la pestaña inicial podrá acceder a la pestaña de Consultas mediante un botón, allí se mostrarán las denuncias más recientes por hora, además de una opción para consultar el número total de robos y filtrar los robos por hora y lugar.

*Requerimientos funcionales*:

1. Mostrar las ultimas denunciar añadidas.
   1. Mostrar en pantalla las primeras posiciones del stack de denuncias.
2. Mostrar el botón de consultar el número total de robos.
   1. Retornar la longitud de la lista de denuncias
3. Mostrar un botón de filtrar robos por lugar y hora.
   1. Proporcionar 2 opciones, filtrar por lugar o por hora. En caso de proporcionar una hora o lugar invalida un mensaje en pantalla.
   2. En cualquier caso, buscar en la lista las denuncias coincidentes al parámetro dado por el usuario
   3. Retornar en un mensaje en pantalla los resultados coincidentes y si no hay, retornar acordemente.

**Localidades**

*Descripción: tiene como finalidad visualizar de manera descendente las localidades con mayor cantidad de hurtos.*

*Acción iniciadores y comportamiento esperado:*

En la pestaña inicial podrá acceder a la pestaña de Localidades mediante un botón, allí se mostrarán las denuncias de las localidades más peligrosas, mostrando el nombre de la localidad y el número de robos de cada localidad.

*Requerimientos funcionales*:

1. Mostrar las denuncias de cada localidad.
   1. Mostrar en pantalla el heap de localidades de manera descendente donde la primera será la que tiene mayor número de robos.
2. Mostrar el botón de Localidades.
   1. Mostrar en pantalla un botón que acceda al número de denuncias por la localidad.

# DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

El usuario es recibido por la aplicación con una pantalla, que junto con el logo de la app, le da las opciones de iniciar sesión o de registrarse, la primera opción permite que mediante una combinación adecuada de E-mail y contraseña el usuario tenga acceso a todos las funcionalidades de la aplicación; La segunda opción, el botón de “REGISTRARSE” envía al usuario a una nueva pantalla donde podrá hacer registro creando un nuevo perfil de usuario al ingresar la información de su E-mail y crear una contraseña, acompañada de una confirmación de la contraseña; si este paso es exitoso, la información queda guardada en la aplicación permitiendo a usuario regresar a la primera pantalla para que pueda ingresar con la contraseña y E-mail registrados.

En la pantalla a la cual la aplicación nos envía una vez agregado una combinación correcta de E--mail y contraseña a la pantalla de inicio, encontramos en la parte superior dos opciones, explicaremos la funcionalidad de cada una de las opciones a continuación:

“RECIENTES”, en esta pestaña aparecen todas las denuncias de más reciente a más antigua hechas por los usuarios de la aplicación, en esta pestaña podemos hacer la búsqueda filtrando por hora o lugar las denuncias más recientes que han hecho otros usuarios, lo que nos dará como resultado que en la pantalla solo aparecen las denuncias que cumplen con las características que buscamos.

“MIS DENUNCIAS”, en esta pestaña el usuario podrá observar todas las denuncias que ha hecho organizadas de más reciente a más antigua, además mediante dos botones rojos y circulares en la esquina inferior derecha, puede hacer una denuncia con el botón “!” o eliminar una de sus denuncias con el botón “X”, las interfaces de estos dos botones serán explicadas a continuación.

“LOCALIDADES” en el menú desplegable, el usuario encontrara este botón, este accedera a la lista de localidades con mayores hurtos, cada localidad esta separada por una línea resaltada horizontalmente, cada item muestra el nombre de la localidad y el numero de hurtos en esta.

Botón “!”: al pulsar en el botón “!” la aplicación envía al usuario a otra pantalla donde hay dos campos para llenar con la información de la Hora y Lugar de la nueva denuncia que se quiere hacer, una vez agregada esta información se puede oprimir el botón “DENUNCIAR” para que la aplicación añada la información de la nueva denuncia a la base de datos.

Botón “X”: al pulsar el botón “X” la aplicación envía al usuario a otra pantalla donde muestra la información de la última denuncia que hizo el usuario, muestra la hora y lugar, si el usuario está seguro puede hacer la eliminación haciendo uso del botón “ELIMINAR”, al ser exitosa la operación se imprime un mensaje con la confirmación de que la denuncia ha sido eliminada.

# Entornos de desarrollo y de operación

El software fue desarrollado usando varias aplicaciones, algunas para facilitar el trabajo en grupo como Repl.it, pero últimamente todo el código y los recursos de la aplicación están agrupados y corren en Android Studio. La aplicación está desarrollada para funcionar en dispositivos móviles que utilicen el sistema operativo Android.

# PROTOTIPO DE SOFTWARE INICIAL

El prototipo de software inicial se encuentra en el siguiente repositorio de GitHub:

https://github.com/BiciSegura/BiciSegura.git

# IMPLEMENTACIÓN Y aplicación de las estructuras de datos

**Linked List**

Para la implementación de la Linked List se realizó una lista doblemente enlazada con cola. Esta recibe parámetros genéricos. Los métodos que implementa son los siguientes:

* getFirst()
* getLast()
* deleteFirst()
* deleteLast()
* insertFirst(E)
* insertLast(E)
* isEmpty()
* length()
* setValue(E)
* deleteIndex(E)
* findFirstValue(E)
* findLastValue(E)
* deleteValue()
* insert(Int, E)
* clear()
* contain(E)

La Linked List es nuestra principal estructura de datos ya que nos permite almacenar las denuncias. Así mismo, a través de sus métodos podemos Añadir denuncias, Eliminar denuncias y también buscar/filtrar las denuncias por lugar y hora.

**Pila**

Para la implementación de la pila se heredó de la clase LinkedList y añadieron los métodos de:

* Pop()
* Push()
* Peek()

Esta estructura de datos es muy útil para la funcionalidad de denuncias más recientes, ya que su naturaleza LIFO nos permite llevar esto a cabo de manera sencilla.

**Cola**

Para la implementación se procedió de manera muy similar a la de las Pilas. En esta se heredó de la clase LinkedList y se añadieron los métodos.

* Poll()
* Queue()
* Dequeue()

Esta estructura de datos fue muy útil en el almacenamiento de usuarios ya que nos permite registrarlos de manera cronológica.

**AVLTree**

Se ha hecho la implementación de la estructura de datos de un árbol AVL con el fin insertar las denuncias para optimizar las implementaciones anteriormente hechas, de igual forma se ha estudiado de manera profunda en comparación de la estructura linked list para determinar los niveles de conveniencia según la arquitectura del software. Se ha programado los métodos principales para su posterior uso; de los cuales son:

* insert()
* remove()
* balance()
* find()
* findmax()
* findmin()
* altura()

**Heap**

Se implemento un Heap para nuestra nueva operación e interfaz de usuario llamada Localidad, se ha escogido esta estructura ya que cumple los objetivos de la función Localidad, la cual pretente hacer un Heap de localidades que se clasifican por la cantidad de hurtos, poniendo como raíz principal la localidad con el índice mas alto de robos.

# PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

Se han realizado las pruebas de ejecución en base a las estructura linked list y avlTree, estas estructuras se han programado para que acepten datos genéricos, para este análisis se ha creado un objeto con un atributo entero, el cual es generado aleatoriamente, se utilizó la librería Random interna de java, los resultados fueron los siguientes:

Metodo Insert:

|  |  |  |
| --- | --- | --- |
| **Cantidad de datos** | **AVLTree/Tiempo (s)** | **Linked List/ Tiempo (s)** |
| 10.000 | 0 | 0 |
| 100.000 | 7 | 9 |
| 1´000.000 | 43 | 56 |
| 10´000.000 | 301 | 423 |
| 100´000.000 | 1140 | 1950 |

Encontrar el último dato insertado:

|  |  |  |
| --- | --- | --- |
| **Cantidad de datos** | **AVLTree/Tiempo (s)** | **Linked List/ Tiempo (s)** |
| 10.000 | 0 | 0 |
| 100.000 | 3 | 9 |
| 1´000.000 | 25 | 45 |
| 10´000.000 | 115 | 202 |
| 100´000.000 | 606 | 1120 |

Eliminar valor:

|  |  |  |
| --- | --- | --- |
| **Cantidad de datos** | **AVLTree/Tiempo (s)** | **Linked List/ Tiempo (s)** |
| 10.000 | 0 | 2 |
| 100.000 | 5 | 6 |
| 1´000.000 | 65 | 52 |
| 10´000.000 | 206 | 196 |
| 100´000.000 | 1055 | 1750 |

Como conclusion principal Podemos evidenciar que la estructura AVLTree es mucho más rápida que la Linked List, este resultado era de esperarse ya que la linked list emplea un mayor esfuerzo computacional, recormedos que en el peor de los casos la linked list tiene la complejidad computacional O(n) es lineal por ende tarda mucho más, en el caso de lo AVLTree su complejidad es O(log n) es logarítmica esto hace mucho más eficiente este algoritmo.

Es necesario resaltar que los AVLTree tienen ciertas desventajas en comparación a la linkedList, una ellas es que a medida que la altura del árbol se más alta será mas complejo eliminar algún valor, consideremos que se requieren ciertas rotaciones para equilibrar el árbol, la linked list no posee este problema pero como se menciono anteriormente si requerimos buscar un valor que esta muy lejos del primer indice se tendrán que recorren n índices para encontrar el valor, donde n es el tamaño de la lista.

# roles y actividades

|  |  |  |
| --- | --- | --- |
| **INTEGRANTES** | **ROL(ES)** | **TAREAS REALIZADAS** |
| Mateo Alejandro Diaz Muñoz | Tecnico | Implemetacion de la estructura de datos AVLtree. |
| Analista de estructuras en periodos de tiempo. |
| Secretario | Desarrollo de la documentacion |
|  |
| David Felipe Mora Ramirez | Líder | Planeación de tareas en un calendario con fechas límites. |
| Incentivo al trabajo en equipo. |
|
| Tecnico | Trabajo en implementaciones y métodos del programa. |
| Coordinador | Asegurarse de que todos los integrantes tuviesen un trabajo asignado. |
| Sergio Esteban Prieto Vargas | Tecnico | Trabajo en implementaciones, métodos y miscelanea referente al proyecto. |
| Coordinador | Entrego las tareas al integrante más apto. |
| Investigador | Busco los recursos necesarios para la resolución de problemas. |
| Diego Romero Iregui | Experto | Aportó mucho conocimiento para el desarrollo de la aplicación, además de enseñar a otros. |
| Desarrollo de interfaces. |
| Coordinador | Coordino los tiempos de trabajo de cada miembro para hacer el desarrollo más óptimo. |

# DIFICULTADES Y LECCIONES APRENDIDAS

Entre las dificultades que pudimos encontrar estuvo la implementación de nuestras estructuras y métodos. Esto debido a que recibir datos de tipo genérico se hizo todo un poco más complejo. También tuvimos algunas dificultados al momento de considerar los casos extremos dentro de los métodos. Sin embargo, después de largas horas de “Debugging” logramos tener un manejo más experimentado en este ámbito.

Por otro lado, tuvimos algunas dificultades al inicio del proyecto en lo que concierne a la coordinación de tareas dentro del equipo. No teníamos claro quien se iba a encargar de cual tarea y dentro de que rango de tiempo la iba a completar. No obstante, nos apoyamos de la plataforma Trello, la cual hizo muy más conveniente este aspecto y nos permitió aumentar nuestra productividad.

Otra dificultad importante fue el manejo del entorno de AndroidStudio y en general el desarrollo de aplicaciones móviles, especialmente en lo relacionado con la interfaz gráfica. Hasta el momento logramos implementar una interfaz gráfica básica pero apropiada para nuestro objetivo.

# REFERENCIAS BIBLIOGRÁFICAS

* src/java.base/share/classes/java/util/LinkedList.java. Hg.openjdk.java.net. (2020). Retrieved 17 October 2020, from <https://hg.openjdk.java.net/jdk/jdk11/file/1ddf9a99e4ad/src/java.base/share/classes/java/util/LinkedList.java>.
* *Stack Overflow - Where Developers Learn, Share, & Build Careers*. Stack Overflow. (2020). Retrieved 18 October 2020, from <https://stackoverflow.com/>.
* *Data Structures and Algorithm in java, Wiress*

1. [↑](#footnote-ref-1)