

María Díaz, Juan Oviedo, Edwin Sánchez, Julián Sánchez, Ana Goyeneche.

Equipo Trabajo #1

Q&A UNAL

I. INTRODUCCIÓN.

Este documento es un conciso resumen del desarrollo de la aplicación web Q&A UNAL, incluirá información valiosa sobre el avance de la misma, incluyendo la descripción de la problemática que se propone resolver y cómo se alcanzará dicho objetivo.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER.

Existe una gran dificultad para los miembros de la comunidad de la Universidad Nacional de Colombia para encontrar información relevante, concisa y clara en el momento en el que la precisan. Actualmente la UNAL cuenta con muchos canales de información oficiales, como el correo institucional, la página web, la atención telefónica y su directorio, entre otros y no oficiales, como los diversos grupos en Facebook.

Sin embargo, dicha diversidad en fuentes de información, resulta ineficiente y frustrante; se vuelve difícil acceder a los datos deseados pues no resulta claro para los miembros dónde los deben consultar, existe pérdida de información valiosa y la perdurabilidad en el tiempo de la misma es limitada.

En este contexto, proponemos Q&A UNAL como solución; en la que buscamos desarrollar una plataforma centralizada para la información relevante para la comunidad. Mediante el acceso con las credenciales de los miembros se podrán formular preguntas, asignarles tema, contestar a los demás miembros y consultar si otros ya han tenido dicha duda previamente; todo esto con el fin de hacer el proceso más rápido, eficaz y eficiente y construir una comunidad online más unida.

III. USUARIOS DEL PRODUCTO DE SOFTWARE.

Para acceder a Q&A, los miembros de la comunidad usarán sus credenciales que los validan como tales. La clasificación del rol usado por una persona dependerá de la funcionalidad que desea usar; Inquier (para hacer una pregunta), Solver (para resolver una duda mediante un comentario) y Finder (para encontrar la respuesta en las preguntas previamente hechas). Cabe aclarar que ninguno de los roles es excluyente ni permanente, el usuario usará uno u otro de acuerdo a la situación en concreto.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE.

A continuación, se procede a describir las interacciones que tendrán los usuarios con las funcionalidades provistas por Q&A UNAL y las estructuras de datos implementadas.

● Validar de datos para acceder.

Descripción: Control de ingreso a la página web, de tal manera que no-miembros de la comunidad no tengan posibilidad de acceder.

Acciones iniciadoras y comportamiento esperado: El usuario escribirá su nombre de usuario y contraseña asociados a su cuenta institucional. Debe presionar el botón 'acceder' para validar dichos datos.

Requerimientos funcionales:

- i. Tras el cotejo de datos, se dará acceso a la página de bienvenida a los usuarios que hayan escrito un nombre de usuario y contraseña válidos.
- ii. Si los datos de la cuenta no son correctos, se alertará con un mensaje, pidiendo que se reintente el log-in. También se remarcará un ícono con signo de interrogación.
- iii. Al clicar sobre el símbolo de pregunta aparecerá un mensaje explicativo aclarándole al usuario solo podrá acceder si lo intenta con sus credenciales UN.

● Crear pregunta para la comunidad.

Descripción: Redacción y publicación de la pregunta que tiene el usuario y que desea resolver.

Acciones iniciadoras y comportamiento esperado: El Inquier debe llenar los campos correspondientes a *Título*, *Etiqueta(s)*, y *cuerpo* de la pregunta. Al oprimir el botón *OK* será publicada la pregunta.

Requerimientos funcionales:

- i. Esta funcionalidad será la selección por defecto tras la validación de la información de acceso.
- ii. No se permitirá la publicación de preguntas cuando no estén completados todos los campos. Si dicha acción se intenta aparecerá un mensaje alertando sobre el requerimiento.
- iii. Al publicar la pregunta ésta aparecerá con la información dada por el Inquier, más la *fecha*, la *hora* en la que fueron subidas, el *nombre de usuario* y el estado de la misma que por defecto será *-Activo*.

● **Responder preguntas de otros usuarios.**

Descripción: Por medio de comentarios, los Solver podrán dar respuestas y opiniones sobre las preguntas que han sido publicadas.

Acciones iniciadoras y comportamiento esperado: El usuario que funge de Solver debe hacer click sobre la pregunta que quiere responder, escribir su respuesta y oprimir el botón *Enviar* para cargar su respuesta.

Requerimientos funcionales:

- i. Se accederá a esta funcionalidad cuando el usuario oprima sobre el botón responder que se encuentra en la zona izquierda.
- ii. La pestaña responder mostrará a las preguntas que serán organizadas en dos tipos de estructuras de datos; colas y pilas, sugiriendo dar repuestas de manera LIFO o FIFO de acuerdo a la fecha de publicación.
- iii. Tras hacer click sobre la pregunta que quiere contestar, el Solver tendrá una pantalla con el campo disponible para escribir sus comentarios. El comentario sólo se cargará si se oprime el botón *Enviar*.
- iv. Tras cargar su respuesta, será dirigido automáticamente a la pestaña de *Responder*; en caso de que el usuario desee cambiar su rol, debe seleccionar otra pestaña.
- v. Cuando un usuario tipo Inquier haga click sobre una pregunta de su propiedad, podrá, de entre las respuestas dadas, seleccionar aquella que le resultó útil. Al guardar dicha selección, el comentario elegido quedará marcado como *útil* y el estado de la pregunta pasará de *Activo* a *Cerrado*.

● **Consultar información en preguntas previas.**

Descripción: Para los usuarios tipo Finder. Podrán mediante palabras que ellos consideren clave, revisar si previamente se hizo una pregunta de su interés y en caso de que existan, revisar sus respuestas.

Acciones iniciadoras y comportamiento esperado: El usuario debe ingresar mediante la pestaña *Consultar*, escribir el tema que desea buscar y posteriormente oprimir el botón *Buscar*.

Requerimientos funcionales:

- i. Tras clickear en *Buscar*, el usuario tipo Finder debe ver las respuestas que comparten el tema con el de su interés, éstas organizadas de

acuerdo a su fecha de publicación bajo una estructura de datos tipo lista o cola.

- ii. El Finder tendrá la posibilidad de ver los comentarios que han sido hechos al seleccionar la pregunta, no antes. Si la respuesta ya fue Cerrada por el Inquier, aparecerá marcado el comentario que este clasificó como *útil*.
- iii. Mediante el botón *Volver* el usuario podrá regresar a la página que le mostraba las preguntas que coincidían con su búsqueda.

● **Actualizar pregunta.**

Descripción: El inquirer dueño de una pregunta tiene la posibilidad de modificarla cuando esto le parezca pertinente.

Acciones iniciadoras y comportamiento esperado: El usuario debe clickear en la pestaña de preguntas sobre aquella que desea cambiar, posteriormente, con el botón habilitado para esto la podrá modificar, pues re aparecerán los campos de *Título*, *Etiqueta(s)*, y *cuerpo* de la pregunta.

Requerimientos funcionales:

- iv. Sólo se podrán actualizar preguntas si éstas no han cambiado de estado a cerradas.
- v. Tras clickear en su pregunta el usuario tipo Finder verá un botón que le redirigirá a la zona de actualización de su pregunta..
- vi. Tras haber modificado los campos deseados, el usuario guardará la información actualizada apretando sobre el botón *Guardar*.
- vii. La pregunta aparecerá organizada de acuerdo a la estructura de datos adecuada, consistente con la previa elección.

viii. Actualizar comentario.

Descripción: El Solver, quien ha posteoado un comentario en respuesta a una pregunta puede cambiarlo cuando le parezca necesario.

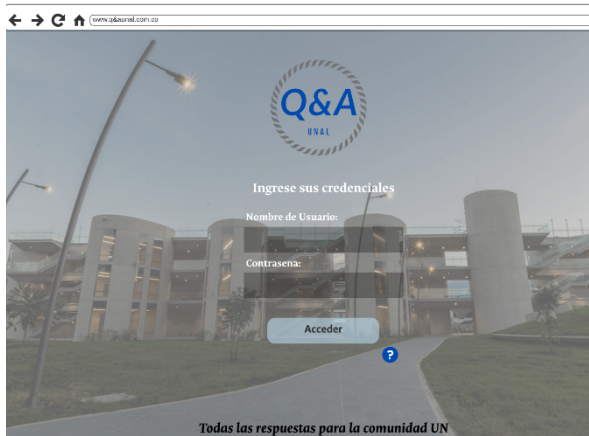
Acciones iniciadoras y comportamiento esperado: El usuario debe clickear en la pestaña de preguntas sobre aquella que respondió, para seguidamente clickear sobre el comentario de su autoría para actualizarlo. Se desplegará la misma sección donde podrá cambiar su respuesta y guardarla seguidamente.

Requerimientos funcionales:

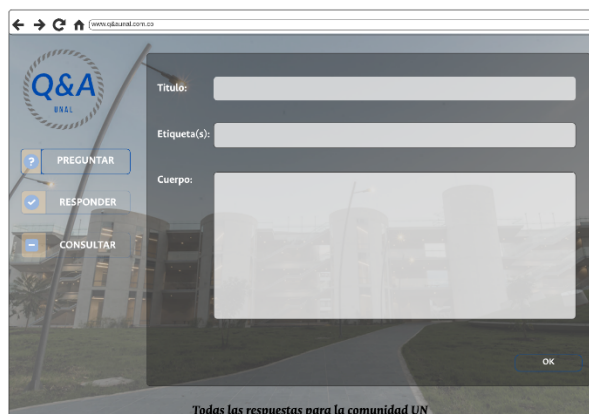
- ix. Sólo se podrán modificar comentarios de preguntas que han NO han cambiado su estado a cerradas.
- x. Tras clicar en la pregunta que lo dirige al comentario que desea cambiar, y también abrir este, el usuario debe editar el texto de acuerdo a su preferencia.
- xi. Tras haber modificado el cuerpo del comentario, el usuario guardará la información actualizada apretando sobre el botón *Guardar*.
- xii. La respuesta o comentario aparecerá de acuerdo a la estructura de datos pertinente.

V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

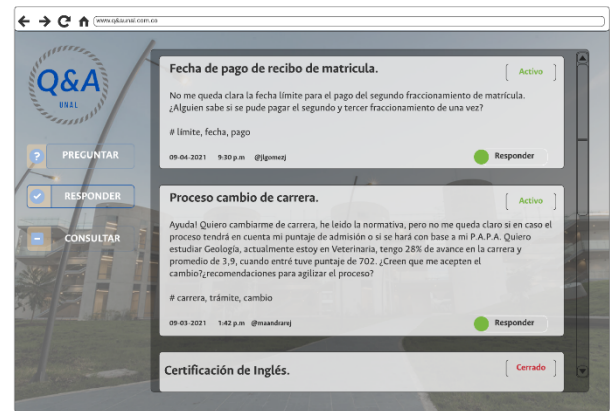
A continuación, se presentan los mockups preliminares de la interfaz con la que el usuario va a interactuar.



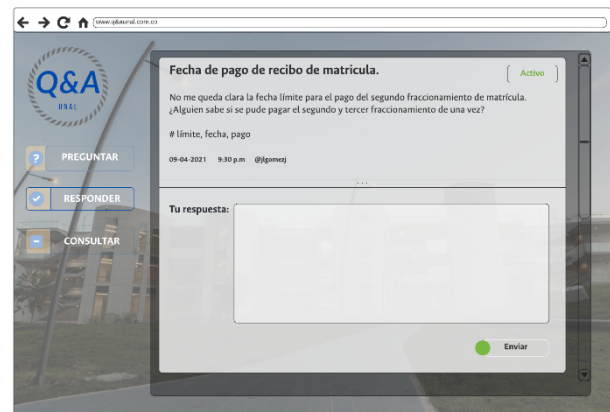
[1] Control de acceso mediante username y password.



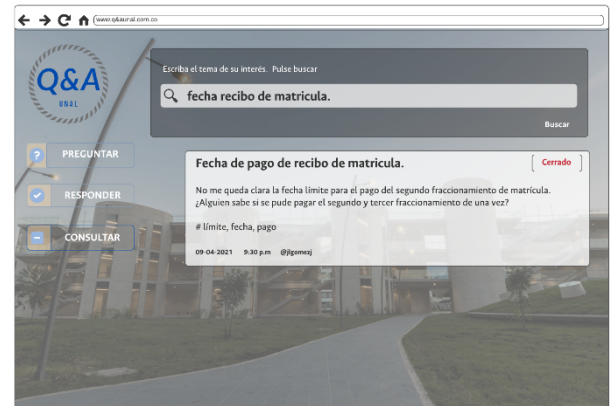
[2] Creación de pregunta (*Título, Etiqueta(s), Cuerpo*).



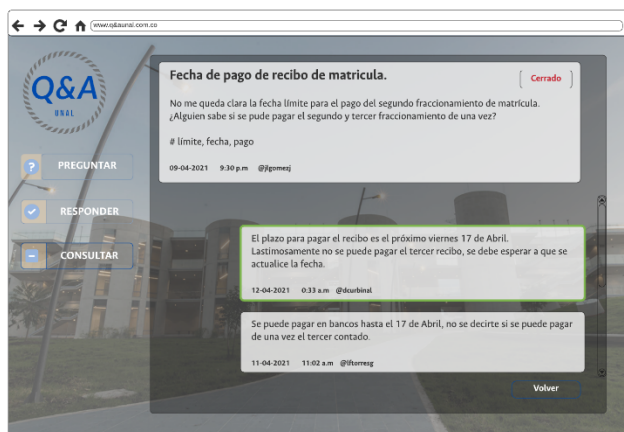
[3] Organización de preguntas a responder.



[4] Responder a una pregunta mediante un comentario.



[3] Consultar información en preguntas ya hechas.



[4] Ver respuestas a preguntas previamente realizadas.

VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN.

El prototipo inicial construido se plantea en base al framework de desarrollo web Flask. Se planteó la construcción de un API, a través del cual se implementarán todas las funcionalidades requeridas, así, el API, se convierte en la única interfaz entre la consola que maneja el usuario, y las estructuras de datos que se encuentran operando en la aplicación.

El despliegue de un aplicativo web, como lo es en este caso el API, puede llegar a ser muy costoso, esto a causa de los recursos que se pueden necesitar para asegurar un funcionamiento continuo. Sin embargo, el prototipo propuesto no requiere recursos de alto nivel, en tanto el mayor consumo de recursos proviene del almacenamiento de los datos.

Así, teniendo en cuenta las necesidades que se tenían para el proyecto, y después de realizar un exhaustivo análisis, se decidió como alternativa ideal para el despliegue la plataforma Heroku. El servicio de plataforma como servicio (PaaS) de Heroku, abre muchas facilidades para un proyecto como este. Inicialmente, por la facilidad que tiene para el despliegue del proyecto directamente desde GitHub, reduciendo, e incluso eliminando, la necesidad del uso de herramientas de línea de comando que pueden llegar a ser confusas por la falta de documentación. En segundo lugar, la plataforma se basa en el uso de contenedores, lo que asegura que, en un futuro, mientras el aplicativo aumenta su base de usuarios, se asegure la escalabilidad del servicio. Finalmente, Heroku, ofrece los recursos técnicos necesarios en su plan gratuito, 512 MB de memoria en un contenedor de sistema operativo Linux con la posibilidad de mantener dos procesos concurrentes. De este modo, se asegura que el aplicativo permanezca en línea y que cuente con la memoria suficiente para manejar los volúmenes de datos propuestos en el proyecto.

VII. PROTOTIPO DE SOFTWARE INICIAL

El prototipo de software inicial, se basa en la construcción de un API (Application Programming Interface) la cual permitirá

al usuario interactuar con los datos, y por consiguiente con las estructuras de datos propuestas, a través de una consola.

El API se encuentra construido usando la librería *flask*, un framework de desarrollo web que se caracteriza por su modularidad y facilidad de extensión. Aunque se consideró la posibilidad de emplear Django como framework de desarrollo, se descartó la opción en tanto su curva de aprendizaje es empinada, y más si se carece de experiencia en el desarrollo web. El framework *flask* facilita el desarrollo gracias a su curva de aprendizaje y a la facilidad que tiene de añadir funcionalidades a través de otras librerías como WTForms, y SQLAlchemy, funcionalidades que a futuro serán fundamentales en el aplicativo web.

La consola, se encuentra basada en la librería *requests* de Python, la cual facilita la realización de solicitudes de tipo HTTP a sitios web. En este caso, el prototipo hace uso de las 4 funcionalidades básicas de un API: GET, POST, PUT, DELETE. Para realizar la obtención, creación, actualización y eliminación de datos respectivamente. Los parámetros adicionales, como por ejemplo los datos para la creación y actualización de preguntas, se envían a través de diccionarios JSON para asegurar el acceso a los datos, tanto para el API como para la librería en la que se basa la consola.

El desarrollo de este prototipo no solo abre muchas posibilidades sino que también da lugar a muchas opciones para el desarrollo del proyecto en futuras versiones. En primer lugar, la construcción del API, pone las bases para la construcción del aplicativo web que se propone como resultado final del proyecto, en tanto permite establecer las rutas que serán las responsables de la ejecución de acciones CRUD. Así mismo, la construcción del API permite asegurar que el usuario, que trabaja con la aplicación a través de una consola, no tendrá que lidiar con la complejidad que existe detrás del manejo de las estructuras y del almacenamiento.

Para el desarrollo de entregas futuras se propone como inclusiones fundamentales, el cambio de la administración del almacenamiento, de archivos de tipo JSON a una base de datos, adicionalmente, se propone la creación de una interfaz visual básica que reemplace la consola. Con estos dos cambios fundamentales, se involucra también el desarrollo de otras funcionalidades, como por ejemplo la validación de los datos ingresados y la automatización de procesos, como la identificación de usuarios.

VIII. IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

Para realizar la implementación de las estructuras de datos, se utilizan como datos básicos los objetos pregunta y comentario. Pregunta se compone de los atributos: IdPregunta, título, texto, fecha y hora a la que se realizó la misma, tema, Id del usuario que realizó la pregunta, estatus (es decir, si la pregunta ya se resolvió o no), número de likes y comentarios asociados a ésta. Comentario se compone de los atributos: IdComentario,

texto, fecha y hora a la que se realizó, Id Usuario que comentó, e IdPregunta a la cual está asociado.

Para este proyecto se utilizaron las estructuras de colas (de tipo FIFO) y pilas (de tipo LIFO). Cada una de estas estructuras se implementa utilizando:

- Listas simplemente enlazadas (SLL): dentro de estas estructuras cada nodo contiene un objeto de tipo pregunta o comentario con sus respectivos atributos, y su nodo sucesor *next*. Esta estructura tiene el atributo *head*, que señala el primer elemento de la lista. En este caso no se tiene un *tail*. Se implementan los siguientes métodos:
 - *pushFront* : agregar al inicio de la lista
 - *pushBack* : agregar al final de la lista
 - *popFront* : devuelve y elimina el inicio de la lista
 - *count*: cantidad de elementos en la lista
 - *find* : Encontrar un elemento por su ID
- Listas doblemente enlazadas (DLL): dentro de estas estructuras cada nodo contiene un objeto tipo pregunta o comentario con sus atributos, junto a las referencias a su nodo antecesor *prev* y a su nodo sucesor *next*. Al igual que las listas simplemente enlazadas, esta estructura cuenta con un atributo llamado *head*, que llama el primer elemento de la lista. Sin embargo, en este caso sí se cuenta con un atributo *tail* que apunta al último elemento de la lista. El uso de los atributos *prev* y *tail* dentro de las listas doblemente enlazadas implica mayor uso de almacenamiento, pero a su vez supone menores tiempos de ejecución de los métodos. Para esta estructura se implementaron los siguientes métodos:
 - *pushFront* : agregar al inicio de la lista
 - *pushBack* : agregar al final de la lista
 - *popFront* : devuelve y elimina el inicio de la lista
 - *popBack*: devuelve y elimina el final de la lista
 - *count*: cantidad de elementos en la lista
 - *find* : Encontrar un elemento por su ID

Las colas se implementan con los métodos: enqueue y dequeue, que agregan y extraen elementos de la cola, implementados utilizando los métodos *pushBack* y *popFront*, respectivamente. Las pilas son implementadas con sus métodos *push* y *pop*, agregando y extrayendo elementos de la pila. Estos son implementados utilizando los métodos *pushFront* y *popFront*. Dentro de las preguntas, los comentarios asociados a esta son construidos como otra estructura igual a la estructura en la que se almacenan las preguntas. Las estructuras soportan las siguientes funcionalidades, respectivamente del tipo de objeto que guarden:

- Creación de estructura de preguntas y de comentarios dentro de cada pregunta. Se implementa creando la

estructura realizando enqueue o push a todas las preguntas.

- Inserción de pregunta o comentario nuevo. Se implementa realizando enqueue o push de la pregunta nueva, o find de la pregunta asociada y enqueue o push del comentario nuevo.
- Actualización del texto o tema de una pregunta o el texto del comentario. Se implementa utilizando el método *find* y cambiando el atributo guardado para la pregunta o comentario
- Eliminación de una pregunta o comentario. Se implementa utilizando el método *find* y quitando de la estructura de datos la pregunta o comentario.
- Consulta de todas las preguntas hechas o de todos los comentarios asociados a una pregunta. Se implementa realizando dequeue o pop a todos los elementos de la estructura.
- Almacenamiento de todos los datos de preguntas y comentarios. Se implementa realizando dequeue o pop a todos los elementos de la estructura y guardando dentro de un archivo .json.
- Búsqueda de una palabra dentro del título o tema de una pregunta. Devuelve todas las preguntas asociadas a esta pregunta. Se utiliza implementando el método *find* de las listas enlazadas.

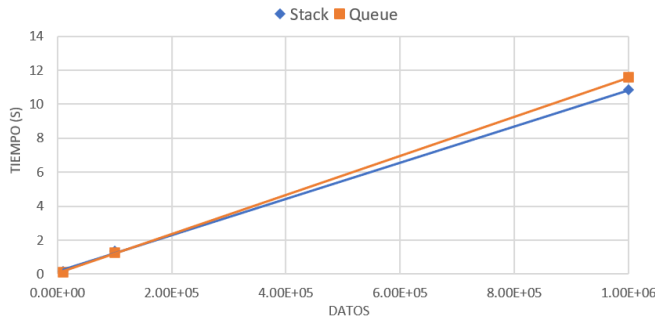
IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

A continuación se presentan los resultados comparativos de la implementación de tres funcionalidades vitales para el aplicativo web Q & A UNAL: buscar pregunta, actualizar una pregunta y eliminar un comentario. En cada caso se confrontará la complejidad de los algoritmos al ser implementados con primero una estructura de cola (queue) basada en una lista simplemente enlazada, y luego con una pila (stack) basada en una doubly linked list.

	# Datos	Buscar pregunta	Actualizar pregunta	Eliminar comentario
Queue	1.00E+04	0.089	0.0019	0
	1.00E+05	1.267	0.0089	0
	1.00E+06	11.57	0.0010088	0
Stack	1.00E+04	0.152	0.002292	0.0498
	1.00E+05	1.347	0.0248	0.0579
	1.00E+06	10.84	0.18	0.15

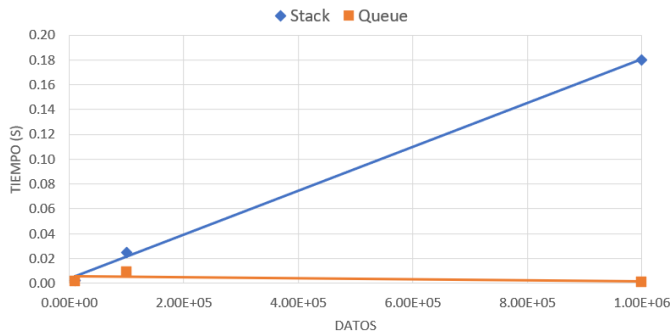
El análisis comparativo y de complejidad resulta más fácil al visualizar en cada caso la información consignada en la anterior tabla.

BUSCAR PREGUNTA POR TEXTO.



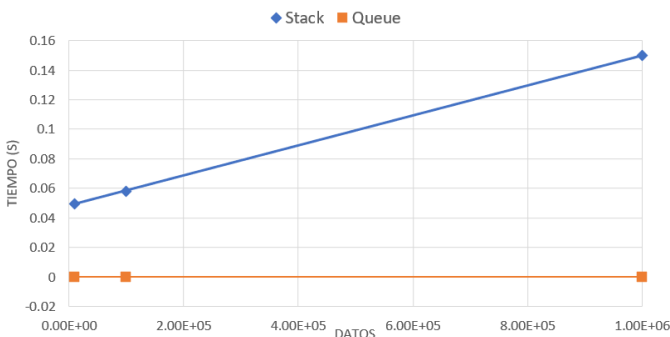
Se puede observar que para esta funcionalidad en ambas implementaciones se tiene una complejidad $O(n)$ y aunque hay algunas diferencias en el tiempo que tarda la ejecución, éstas no son significativas. El comportamiento lineal observado es el esperado pues se deben recorrer n elementos y buscar en ellos las palabras coincidentes con las de la búsqueda.

ACTUALIZAR PREGUNTA.



En este caso es posible abstraer que la pila tiene un comportamiento $O(n)$ para la función mostrada, mientras que la implementación con la cola se obtiene un tiempo de ejecución constante $O(1)$, haciendo más conveniente su uso.

ELIMINAR COMENTARIO.



De manera análoga a la función previamente presentada se puede observar un comportamiento $O(n)$ vs $O(1)$ para la ejecución comparativa de pilas y colas, lo cual sugiere que la implementación para grandes grupos de datos será más eficiente en términos de tiempo al usar colas.

Para completar el previo análisis valdría la pena evaluar cada uno de los algoritmos y los recursos computacionales que estos consumen en términos de memoria, para así ponderar adecuadamente y tomar las mejores decisiones respecto a qué implementación utilizar. Este último factor se vuelve muy importante cuando se manejen recursos limitados y cantidades de datos muy grandes, algo que todavía no está sucediendo, y que se espera no suceda, en tanto, en caso de generarse picos de demanda el despliegue propuesto permitirá que el proyecto se escale de forma horizontal para dar abasto con la demanda.

Se deben realizar y documentar las pruebas del prototipo para algunos ejemplos (casos) de prueba para las funcionalidades que tomen más tiempo y realizar un análisis comparativo así:

- Escoger entre tres y cinco funcionalidades que sean las de mayor costo computacional en tiempo;
- Para cada funcionalidad se deben realizar pruebas para varios tamaños de datos de prueba (n), por lo menos para los siguientes valores de n :
 - 10 mil datos,
 - 100 mil datos,
 - 1 millón de datos,
 - 10 millones de datos, y
 - 100 millones de datos.
- Hacer una tabla comparativa de los tiempos que toma realizar cada una de las funcionalidades consideradas para los diferentes tamaños de los datos de prueba.
- Determine y grafique el correspondiente análisis asintótico comparativo entre las estructuras implementadas y su respectiva complejidad, de acuerdo con las pruebas realizadas. Para esto debe usar, por lo menos, la notación O grande (*Big O*).

X. ROLES Y ACTIVIDADES

Con el fin de trabajar en equipo de la mejor manera posible se hizo la división del trabajo de acuerdo a las fortalezas de cada integrante, el seguimiento de las asignaciones se hizo mediante la herramienta Trello. A continuación, se presenta el esquema de trabajo que se llevó a cabo.

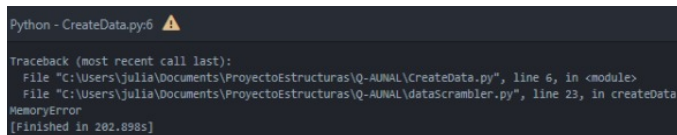
ROL	Actividades fundamentales
Líder(esa)	Consultar a los otros miembros del equipo, atento que la información sea constante para todos. Aportar con la organización y plan de trabajo.
Coordinador(a)	Mantener el contacto entre todos, Programar y agendar y reuniones; ser facilitador para el acceso a los recursos.
Experto(a)	Líder técnico que propende por coordinar funciones y actividades operativas.
Investigador(a)	Consultar otras fuentes. Propender por resolver inquietudes comunes para todo el equipo.
Observador(a)	Siempre está atento en el desarrollo del proyecto y aporta en el momento

	apropiado cuando se requiera apoyo adicional por parte del equipo.
Animador(a)	Energía positiva, motivador en el grupo.
Secretario(a)	Se convierte en un facilitador de la comunicación en el grupo. Documenta (actas) de los acuerdos/compromisos realizados en las reuniones del equipo.
Técnico(a)	Aporta técnicamente en el desarrollo del proyecto.

Integrante	Roles Asignados	Actividades realizadas
María Díaz	Coordinadora, Lideresa	Implementación de estructuras de datos.
Juan Oviedo	Experto, técnico	Desarrollo interfaz consola y API web
Edwin Sánchez		Implementación de estructuras de datos.
Julián Sánchez		Generación aleatoria y almacenamiento de datos para comparación.
Ana Goyeneche		Mockups, redacción documento, diapositivas y video.

XI. DIFICULTADES Y LECCIONES APRENDIDAS

Una de las dificultades encontradas fue la generación de grandes cantidades de datos. No fue posible generar y utilizar 10 millones y 100 millones de preguntas diferentes, en tanto los recursos computacionales requeridos para hacerlo superan la capacidad técnica para abordar el problema. Python enfrentaba un `MemoryError` al intentar realizar la generación de los datos, como se puede ver a continuación.



```
Python - CreateData.py:6
Traceback (most recent call last):
  File "C:\Users\julia\Documents\ProyectoEstructuras\Q-AUNAL\CreateData.py", line 6, in <module>
    File "C:\Users\julia\Documents\ProyectoEstructuras\Q-AUNAL\dataScrambler.py", line 23, in createData
MemoryError
[Finished in 202.898s]
```

Adicionalmente, se evidenciaron problemas al momento de realizar los almacenamientos de los archivos en tipo JSON, debido a que el tiempo de guardado, crecía de forma exponencial ante el aumento de exigencia por crecimiento de números de datos procesados. Con el fin de solucionar esta limitante, a futuro se plantea el traslado a una base de datos.

Para realizar la búsqueda de objetos (tanto preguntas como comentarios) dentro de las estructuras de datos, se planteó inicialmente no utilizar la funcionalidad `find` propia de las listas enlazadas, y utilizar una estructura temporal adicional. Los elementos de la estructura inicial eran agregados a la

estructura temporal por su respectivo método (enqueue o push), mientras se revisa el criterio de búsqueda. Sin embargo, este método resulta muy ineficiente para realizar las búsquedas, ya que los tiempos de búsqueda se comportan como $O(n)$ en la búsqueda de preguntas o $O(n^2)$ en la búsqueda de comentarios. Por este motivo, se tomó la decisión de implementar la búsqueda utilizando el método `find` propio de las listas enlazadas, sin la necesidad de utilizar una estructura temporal adicional.

A lo largo de la presente entrega se pudo evidenciar clara y constantemente la importancia del trabajo en equipo en todo momento. Gracias a las reuniones frecuentes y al apoyo a través de medios de comunicación digitales, los inconvenientes que surgían podían solucionarse rápida y consensuadamente, evitando así retrasos en el proyecto. A futuro como grupo se deben extender los esfuerzos por asegurar una integridad en el nombramiento y retornos de funciones, en tanto en algunos puntos del proyecto, esto llevó a retrasos en el desarrollo.

XII. REFERENCIAS BIBLIOGRÁFICAS

- [1] Weiss, M.A.: *Data Structures and Algorithm Analysis in C++*, 4th Edition, Pearson/Addison Wesley, 2014.
- [2] Hernández, Z.J. y otros: *Fundamentos de Estructuras de Datos. Soluciones en Ada, Java y C++*, Thomson, 2005.
- [3] Shaffer, Clifford A.: *Data Structures and Algorithm Analysis in C++*, Third Edition, Dover Publications, 2013. (En línea.)
- [4] Campos Laclaustra, J.: *Apuntes de Estructuras de Datos y Algoritmos*, segunda edición, 2018. (En línea.)
- [5] Martí Oliet, N., Ortega Mallén, Y., Verdejo López, J.A.: *Estructuras de datos y métodos algorítmicos: 213 ejercicios resueltos*. 2ª Edición, Ed. Garceta, 2013.
- [6] Joyanes, L., Zahonero, I., Fernández, M. y Sánchez, L.: *Estructura de datos*. Libro de problemas, McGraw Hill, 1999.