

República Bolivariana de Venezuela  
Universidad Central de Venezuela  
Facultad de Ciencias  
Escuela de Computación



## **Desarrollo de un editor de visualizaciones de propiedades de historiales de wikis**

Trabajo Especial de Grado presentado ante la ilustre  
Universidad Central de Venezuela por el Br. Leonardo Testa  
para optar al título de Licenciado en Computación.

Tutor Prof. Eugenio Scalise

abril, 2018

# Resumen

**Título:**

Desarrollo de un editor de visualizaciones de propiedades de historiales de wikis.

**Autor:**

Leonardo Testa.

**Tutor:**

Prof. Eugenio Scalise.

Un Wiki es un sitio web, generalmente de carácter informativo (como lo es Wikipedia), que puede ser modificado por múltiples personas. Cada una de estas modificaciones son almacenadas, y en conjunto conforman un historial de versiones, en donde cada versión representa una modificación y los efectos que causó en el artículo wiki. Siendo Wikipedia un caso real con bastante popularidad, es normal que el historial de versiones de un artículo sea suficientemente extenso y complejo, por lo tanto las personas interesadas en mantener el artículo “*sano*” perderán una gran suma de tiempo revisando las modificaciones. En este documento, presentaremos la investigación y la realización de una herramienta web que facilita la lectura de propiedades del historial a aquellas personas interesadas, en donde se optará por visualización de datos como estrategia, de esta forma, mediante una interfaz capaz de manipular gráficas el usuario podrá proyectar distintas propiedades y conseguir fácilmente información más completa y concretar patrones.

**Palabras claves:**

Visualización de datos, wiki, propiedades de historiales, gráficas, herramienta web, editor de visualizaciones, wikipedia.

# Índice general

Índice de figuras	4
<b>1. Introducción</b>	<b>7</b>
1.1. Objetivo general . . . . .	8
1.2. Objetivos específicos . . . . .	8
1.3. Justificación . . . . .	8
1.4. Distribución del documento . . . . .	9
<b>2. Marco Teórico</b>	<b>10</b>
2.1. Wikis e historiales de wikis . . . . .	10
2.2. Visualización de datos . . . . .	12
2.2.1. Enfoque explicativo y exploratorio . . . . .	13
2.2.2. Preparación de los datos . . . . .	15
2.2.3. Tipos de gráficas según método . . . . .	16
<b>3. Marco Tecnológico</b>	<b>22</b>
3.1. Tecnologías para la visualización de datos en web . . . . .	22
3.1.1. SVG vs Canvas . . . . .	22
3.1.2. Bibliotecas basadas en Canvas . . . . .	23
3.1.3. Bibliotecas basadas en SVG . . . . .	24
3.1.4. D3 (Data-Driven Documents) . . . . .	25
3.1.5. Evaluación de Bibliotecas . . . . .	26
3.2. Tecnologías para el desarrollo web . . . . .	28
3.2.1. Arquitectura . . . . .	29
3.2.2. Diseño . . . . .	33
3.2.3. Utilidades . . . . .	37

<b>4. Marco Aplicativo</b>	<b>41</b>
4.1. Metodología . . . . .	41
4.2. Realización de tareas . . . . .	42
4.3. Arquitectura . . . . .	71
4.3.1. Front-end . . . . .	72
4.3.2. Back-end . . . . .	74
<b>5. Conclusiones</b>	<b>77</b>
5.1. Limitaciones . . . . .	78
5.2. Trabajos futuros . . . . .	78
5.3. Contribuciones . . . . .	79
<b>Bibliografía</b>	<b>80</b>

# Índice de figuras

2.1. Historial del artículo “Venezuela” en inglés. . . . .	12
2.2. Propiedades de una versión del historial del artículo “Venezuela” en inglés. . . . .	12
2.3. Diagrama que refleja los actores y acciones de una comunicación visual . . . . .	13
2.4. Usuarios activos usando “Historias” de Snapchat vs Instagram	14
2.5. Mediante una gráfica Scatter Plot, podemos analizar patrones y relaciones . . . . .	15
2.6. Gráfica de puntos . . . . .	17
2.7. Gráfica de barras . . . . .	17
2.8. Gráfica de barras flotantes . . . . .	18
2.9. Histograma . . . . .	18
2.10. Gráfica de líneas . . . . .	19
2.11. Gráfica de áreas . . . . .	19
2.12. Gráfica de áreas apiladas . . . . .	20
2.13. Gráfica de dispersión . . . . .	20
2.14. Gráfica de burbujas . . . . .	21
3.1. Comparación de bibliotecas de visualización . . . . .	28
3.2. Ciclo de vida de una aplicación SPA . . . . .	30
4.1. Pizarra de Github . . . . .	42
4.2. Lista de artículos . . . . .	44
4.3. Caja de sugerencias de artículos de Wikipedia . . . . .	48

4.4. Flujo del agregar artículo. En el punto (1) se accede al API de Wikipedia para obtener los artículos sugeridos, luego al presionar agregar sucede el punto (2) que hace un request al API nuestra de usuarios para agregar el artículo y luego en el punto (3) se hace un request en el API de Wikimetrics 2.0 para activar el proceso de extracción del artículo . . . . .	48
4.5. Estilo de un artículo en estado pendiente de extracción . . . . .	50
4.6. Estilo de un artículo en estado exitoso de extracción . . . . .	50
4.7. Barra superior . . . . .	52
4.8. Visualización e información específica del artículo . . . . .	54
4.9. Explicación del mecanismo usado en la visualización de History Flow . . . . .	55
4.10. Herramienta History Flow Visualization . . . . .	55
4.11. Visualización History Graph . . . . .	56
4.12. Visualización Wiki History Flow del artículo 'Programación dirigida por eventos' . . . . .	58
4.13. Información principal de detalle de artículo incluyendo vínculo a Wikipedia . . . . .	58
4.14. Información principal para crear una visualización . . . . .	59
4.15. Vista de componente para seleccionar query de la visualización . . . . .	60
4.16. Componente de visualización y selector de tipo en el editor de visualizaciones . . . . .	62
4.17. Visualización tipo número . . . . .	62
4.18. Visualización tipo línea . . . . .	63
4.19. Visualización tipo barra . . . . .	63
4.20. Visualización tipo torta . . . . .	64
4.21. Visualización tipo dispersión . . . . .	64
4.22. Wiki History Flow con filtro de rango de fecha. . . . .	66
4.23. Toggle para habilitar/deshabilitar <i>preview</i> . . . . .	66
4.24. Componente <i>preview</i> de visualización en detalle de artículo. . . . .	67
4.25. Visualización predefinida de total de ediciones menores vs. no menores del artículo 'Programación dirigida por eventos' . . . . .	69
4.26. Visualización predefinida de total de usuarios anónimos vs. no anónimos del artículo 'Programación dirigida por eventos' . . . . .	70
4.27. Visualización predefinida de Top 10 de usuarios con más ediciones del artículo 'Programación dirigida por eventos' . . . . .	70
4.28. Visualización predefinida de total de ediciones agrupadas por mes y año del artículo 'Programación dirigida por eventos' . . . . .	71

4.29. Arquitectura general . . . . .	72
4.30. Componentes de ruta . . . . .	73
4.31. Modelo de base de datos . . . . .	75

# Capítulo 1

## Introducción

Un Wiki es un sitio web que puede ser modificado por múltiples personas. Generalmente los wikis son de carácter informativo, un claro ejemplo es Wikipedia, que es la enciclopedia online más popular del mundo basada en el concepto wiki. Al ser un sitio colaborativo es necesario llevar un historial de los cambios realizados por los usuarios, logrando un control en las ediciones de estos. De una edición se pueden sacar algunas propiedades importantes como: el autor de la edición, que se representa con una dirección IP cuando es anónimo y con un nombre de usuario en caso contrario, la cantidad de texto editado y la fecha y hora en que realizó la edición. Cabe destacar, que dicho historial de ediciones es de suma importancia para aquellas personas que le hacen seguimiento o que de alguna forma les interesa el estado del artículo, estas personas son consideradas *watchers* del artículo.

Hoy en día existen cantidades de personas colaborando en estos sitios que hacen que el historial de ediciones se haga suficientemente extenso y difícil de comprender. La abundancia de datos provoca complejidad en su búsqueda e interpretación, lo que da lugar a la necesidad de un mecanismo que permita facilitar la transmisión y comprensión de la información, llamado visualización de datos.

La visualización de datos logra transmitir un conjunto inmenso de datos de manera clara y lo hace como su nombre indica, a través de elementos visuales, es decir, gráficas que combinan variedad de colores, figuras y texto. Es importante destacar que la visualización de datos necesita un estudio



previo para la preparación, transformación y análisis de los datos. Debido a lo expuesto anteriormente, en este trabajo se propone la implementación de una herramienta web que consta de un editor de visualizaciones de propiedades de historiales de wikis, en donde los datos necesarios para las visualizaciones serán surtidos principalmente por un servicio (API) llamado Wikimetrics 2.0.

## 1.1. Objetivo general

Desarrollar una aplicación web que permita construir y editar visualizaciones de propiedades de historiales de wikis.

## 1.2. Objetivos específicos

- Diseñar visualizaciones generales basadas en la información de los historiales de artículos de wikis provista por el API de Wikimetrics 2.0.
- Definir los requerimientos de la aplicación.
- Implementar una interfaz SPA adaptativa que ofrezca las funcionalidades requeridas por un *watcher* de un wiki.
- Implementar un servicio API que delegue los requerimientos de la aplicación en cuanto a persistencia de datos.
- Utilizar un método ágil para el desarrollo de la aplicación.
- Realizar el despliegue y puesta en producción de la aplicación.

## 1.3. Justificación

La justificación de este trabajo recae en la posibilidad de hacer investigación en un campo que está siendo cada vez más explorado que es la visualización de datos, que desencadena el área de analistas de datos, y por otro lado un área sumamente amplia que es el desarrollo en tecnologías de internet.

Este trabajo va dirigido especialmente para aquellas personas que le hacen seguimiento a artículos de wikis y quieren informarse rápidamente de anomalías, cambios e información de interés sobre dichos artículos, con el resultado de este trabajo se facilitará mucho más su trabajo, logrando así un artículo de mayor calidad.

Adicionalmente este trabajo puede servir como base para futuros Trabajos Especiales de Grados en la Escuela de Computación de la Facultad de Ciencias en la Universidad Central de Venezuela relacionados con visualización de datos y tecnologías en el área web.

## **1.4. Distribución del documento**

El presente trabajo se encuentra dividido en cinco (5) capítulos. En donde, el capítulo 1, introduce el contexto, el problema, los objetivos planteados (general y específicos), la justificación de la investigación y la distribución del documento. El capítulo 2, presenta las bases teóricas sobre Wiki y su entorno, y la visualización de datos, en donde dichos conceptos son necesarios para lograr el entendimiento de capítulos posteriores. El capítulo 3 presenta la investigación y evaluación de herramientas de apoyo para el desarrollo del proyecto. El capítulo 4 constituye el análisis e interpretación de los resultados presentados en las actividades aplicadas para alcanzar los objetivos planteados. Por último, el Capítulo 5 presenta las conclusiones del trabajo realizado, describiendo los aportes logrados, limitaciones encontradas y planteamiento de trabajos futuros.

# Capítulo 2

## Marco Teórico

Este capítulo cubre los conceptos teóricos necesarios para lograr el entendimiento de asuntos a tratar a lo largo del documento. El capítulo inicia dando contexto sobre Wiki y conceptos que lo rodean. Por último, el capítulo introduce bases teóricas sobre la visualización de datos.

### 2.1. Wikis e historiales de wikis

La idea de un Wiki, que es un término hawaiano que significa “*rápido*” o “*super-rápido*”, fue acuñada por Ward Cunningham en el año 1994 [1]. Esta idea de Cunningham, que consistía en compartir información, fue iterada y hoy en día llamamos Wiki a un sistema manejador de contenido, lo cual, representa un sitio web, cuyas páginas pueden ser editadas directamente desde el navegador, donde los usuarios crean, modifican, y/o eliminan contenido de la misma.

En la actualidad, existen bastantes herramientas o softwares que implementan el concepto de wiki: MediaWiki, UseModWiki, PhpWiki, TikiWiki, DokuWiki, WikkaWiki, entre otros. Tienen la misma finalidad, pero se distinguen en su destino de uso (uso personal, para intranets, para la web) y su funcionalidad (mantener historial, seguridad, editores visuales, etc.)

En el documento, se hará énfasis en el sistema MediaWiki, debido a que

se trabajará con artículos de Wikipedia <sup>1</sup>, donde la plataforma hace uso específico de este.

MediaWiki, como se había definido anteriormente, es una implementación del concepto wiki, adicionalmente, es un software de código libre, esto dice que el código fuente puede ser copiado y mejorado por cualquier persona. Está construido en el lenguaje de programación PHP y apoyado sobre un sistema manejador de base de datos llamado MySQL. MediaWiki consta de las siguientes funcionalidades que son sumamente importante para la realización de este trabajo, que son:

- **Perfil de Usuario:** posibilidad de contener y gestionar una cuenta personal, identificado por un nombre de usuario y contraseña, en donde se pueden tener acciones adicionales, como realizar votaciones, seguir un artículo y otros privilegios.
- **Watchlist:** representa una lista de artículos a los que se le hace seguimiento, de esta manera, será avisado cualquier cambio sobre estos artículos. Como se mencionó en el punto anterior, esta funcionalidad está disponible solo para usuarios registrados.
- **Historial de ediciones de artículos:** bitácora que almacena todos los cambios que ha recibido un artículo con ciertas propiedades respectivas al cambio.

De las funcionalidades relevantes, la más significativa es el historial de ediciones de artículos debido a que nuestro trabajo se basará principalmente en este. Un historial de ediciones, véase en la **Figura 2.1**, representa una serie de cambios o versiones por la que sufrió un artículo, por lo general suelen ser una lista extensa.

---

<sup>1</sup>Wikipedia es una enciclopedia online, creada y editada por voluntarios de distintos lados del mundo

- (cur | prev) 08:45, 13 August 2017 **Freddiem** (talk | contribs) **m** . . (193,228 bytes) (-2) . . *(corrected link)* (undo | thank)
- (cur | prev) 22:31, 12 August 2017 66.87.73.101 (talk) . . (193,230 bytes) (+4) . . (undo) *(Tags: Mobile edit, Mobile web edit)*
- (cur | prev) 05:39, 10 August 2017 Mojoworker (talk | contribs) . . (193,226 bytes) (-582) . . *(Template: Geographic location documentation states it is a "navigational aid for articles about communities" & "is redundant for continents, countries and other large geographical areas like states, provinces and islands.")* (undo | thank)
- (cur | prev) 19:52, 9 August 2017 Mr.choppers (talk | contribs) . . (193,808 bytes) (+40) . . *(→Bolivarian government: 1999–present: they at least pay lip service to socialist ideals)* (undo | thank)
- (cur | prev) 19:50, 9 August 2017 190.202.94.238 (talk) . . (193,768 bytes) (-2) . . *(→Bolivarian government: 1999–present)* (undo)
- (cur | prev) 16:30, 9 August 2017 MusikBot (talk | contribs) **m** . . (193,770 bytes) (-55) . . *(removing {{pp-vandalism}} as page is not edit-protected)* (undo)

Figura 2.1: Historial del artículo “Venezuela” en inglés.

Cabe destacar que cada versión o cambio provee una serie de propiedades, ilustradas en la **Figura 2.2**, como: Fecha y hora de la edición, autor de la edición (si el usuario está registrado se refleja su nombre, si no, la dirección IP de su conexión), tamaño del artículo (bytes), bytes modificados, descripción de la edición, selector para identificar si el cambio es menor, y acciones para ejecutar sobre una edición (agradecer o revertir).

08:45, 13 August 2017 **Freddiem** (talk | contribs) **m** . . (193,228 bytes) (-2) . . *(corrected link)* (undo | thank)

Figura 2.2: Propiedades de una versión del historial del artículo “Venezuela” en inglés.

Adicionalmente, MediaWiki provee un API Web<sup>2</sup>, esto significa que la mayoría de sus funcionalidades están expuestas en la Web y pueden ser accedidas y usadas mediante peticiones HTTP (Hypertext Transfer Protocol). De esta forma, será posible enlazar nuestro trabajo con información de Wikipedia.

## 2.2. Visualización de datos

La visualización de datos es un medio efectivo y eficiente para comunicar una gran cantidad de información [2], en donde la comunicación está conformada por elementos visuales, contando con barras, puntos, líneas, colores, figuras, sombras, entre otras. La agrupación de estos elementos visuales se conoce como gráfica.

<sup>2</sup>Página Principal del API de MediaWiki

Como en toda comunicación, es necesario un mensajero, un mensaje y un receptor. En la visualización de datos, el papel del mensajero lo protagoniza un diseñador que codifica la información de manera visual y el receptor es el decodificador del mensaje [3], véase la **Figura 2.3**.

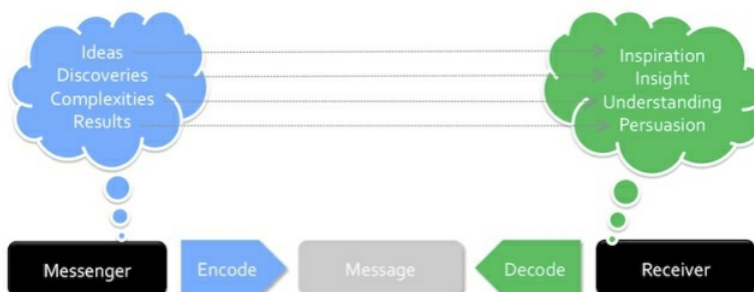


Figura 2.3: Diagrama que refleja los actores y acciones de una comunicación visual

El trabajo del mensajero, que en su defecto es el diseñador, desempeña el papel más importante, debido a que tiene que transmitir una información densa usando elementos visuales, por lo tanto tiene que codificar el mensaje lo más claro y simple posible, lo que significa, que tiene que hacer uso correcto de las gráficas y lograr una buena representación de los datos.

### 2.2.1. Enfoque explicativo y exploratorio

En la visualización de datos se pueden tomar dos enfoques: explicativo y exploratorio.

**Explicativo:** consta de transmitir una información de manera específica, generalmente, el punto de vista del diseñador de la visualización.

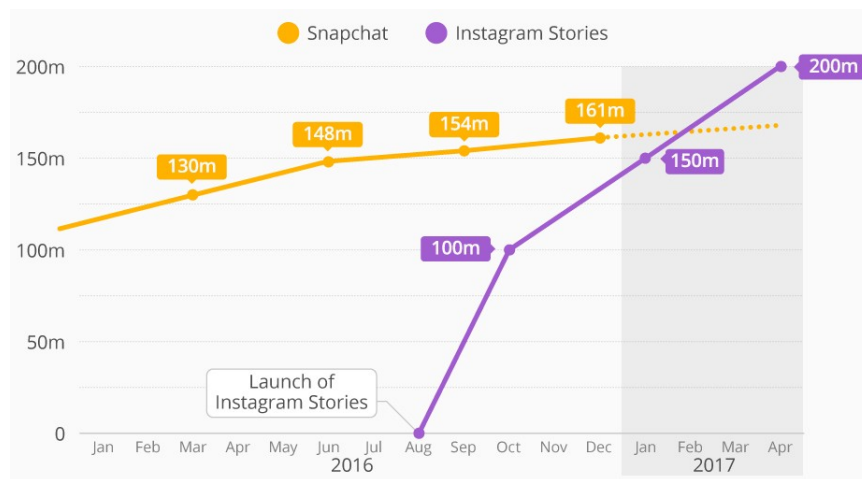


Figura 2.4: Usuarios activos usando “Historias” de Snapchat vs Instagram

En la **Figura 2.4**, se contempla un enfoque explicativo, en donde el diseñador quiere transmitir una información concreta, que es una comparativa en el crecimiento de usuarios activos haciendo uso de la funcionalidad “*Historias*”<sup>3</sup> entre la mensajería de imágenes Snapchat y la red social Instagram.

**Exploratorio:** es un enfoque en donde la gráfica está adaptada para que el receptor pueda analizar y explorar en ella, y así detectar patrones y relaciones en los datos. Este tipo de gráficas por lo general no suelen transmitir una historia como el enfoque explicativo.

<sup>3</sup>Historias o Stories, es una funcionalidad en donde una persona puede publicar una foto durante un período de tiempo, suele ser de 24 horas.

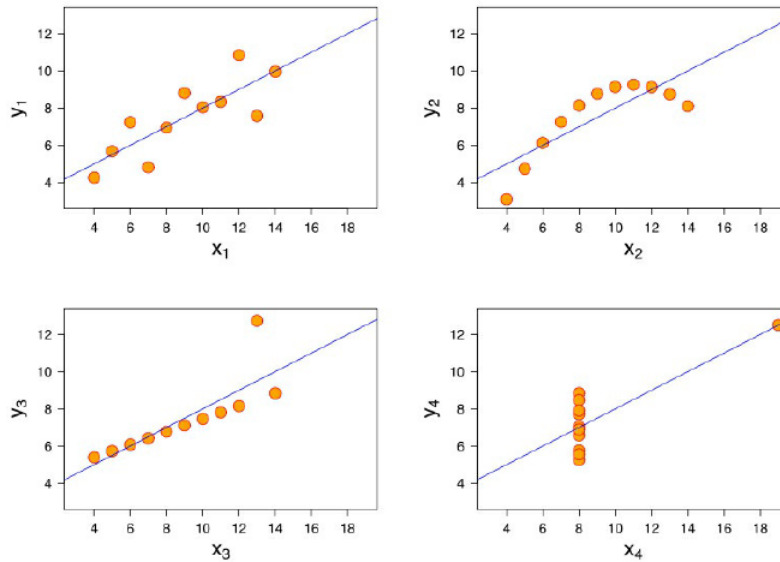


Figura 2.5: Mediante una gráfica Scatter Plot, podemos analizar patrones y relaciones

En la **Figura 2.5**, cada sub-gráfica representa una relación que existe entre dos (2) atributos. No existe una información concreta a transmitir, por lo tanto queda como tarea de la audiencia analizar y explorar relaciones entre atributos. Por ejemplo, el eje Y3 representa precio y eje X3 representa calidad de un producto, entonces según el comportamiento de la gráfica se interpreta que mientras más costoso es un producto mayor es su calidad.

### 2.2.2. Preparación de los datos

Visualizar datos no es tan sencillo como parece, en su mayoría estos datos tienen que pasar por una limpieza y procesamiento antes de ser visualizados. Si los datos a visualizar son incorrectos o están incompletos, la visualización transmitirá una información errónea. A continuación se presentarán los pasos recomendados [3] para preparar los datos:

- **Adquisición:** lo principal es encontrar la fuente que nos va proveer los datos (Excel, base de datos, etc). Sin los datos es imposible continuar.
- **Examinación:** suelen existir datos incorrectos o incompletos, por lo



tanto se debe realizar una verificación de ellos, como eliminar los duplicados, completar con otra fuente los incompletos, acomodar los datos erróneos o en el peor caso removerlos.

- **Estructurar los datos:** dependiendo del tipo de los datos con que contamos la visualización puede variar. Los tipos de datos se pueden dividir en: categórica nominal, categórica ordinal y cuantitativo.
  - Categórica nominal: se distinguen por ser un dato que representa un valor textual. Por ejemplo: un país, un género, etc.
  - Categórica ordinal: es un dato nominal que puede representar un valor. Por ejemplo: medallas olímpicas (oro, plata, bronce), calor o frío, etc.
  - Cuantitativo: es un dato que representa un valor numérico, en donde algunos son de escala de intervalo y otros de proporción. Por ejemplo: fechas, temperatura, precio, edad, etc.
- **Limpiar:** eventualmente algunos datos pueden ser atípicos al resto, esto no dice que sean erróneos, pero pueden hacer ruido en la visualización, es buena opción eliminarlos si ese es el caso.
- **Transformar:** para simplificar la visualización y el análisis de la misma es posible que los datos tengan que sufrir una transformación de tipo o pre-calcular ciertas operaciones en los datos. Por ejemplo: promedio de los precios, categorizar edades (niño: 0-12, adolescente: 13- 19, adulto: 20-50, anciano: +50).

### 2.2.3. Tipos de gráficas según método

Toda visualización está soportada por un método de clasificación, es decir, tiene un motivo y función. En esta sección se presentarán algunos tipos de gráficas correspondientes al método o función [3]:

- **Comparar categorías:** gráfica de puntos (**Figura 2.6**), gráfica de barras (**Figura 2.7**), gráfica de barras flotantes (**Figura 2.8**) e histogramas (**Figura 2.9**).

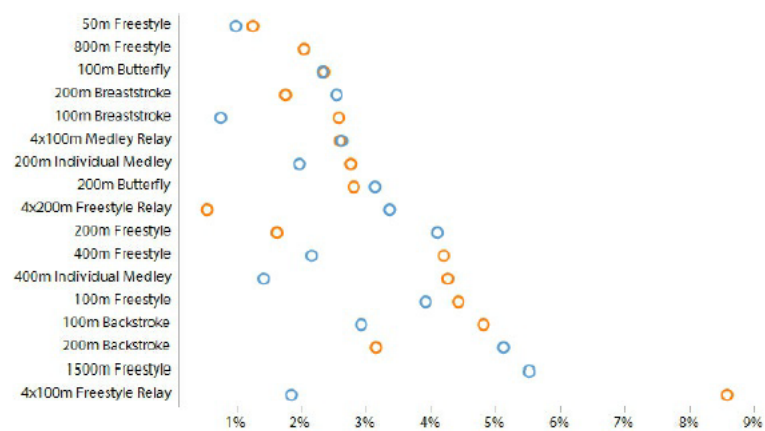


Figura 2.6: Gráfica de puntos

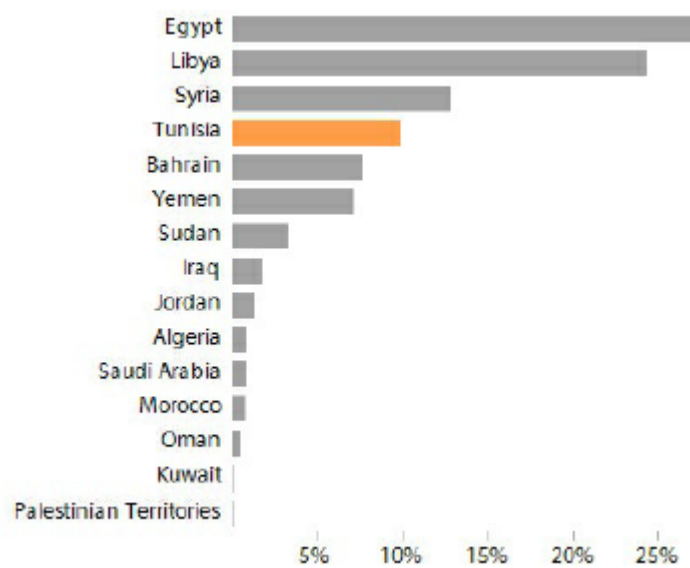


Figura 2.7: Gráfica de barras

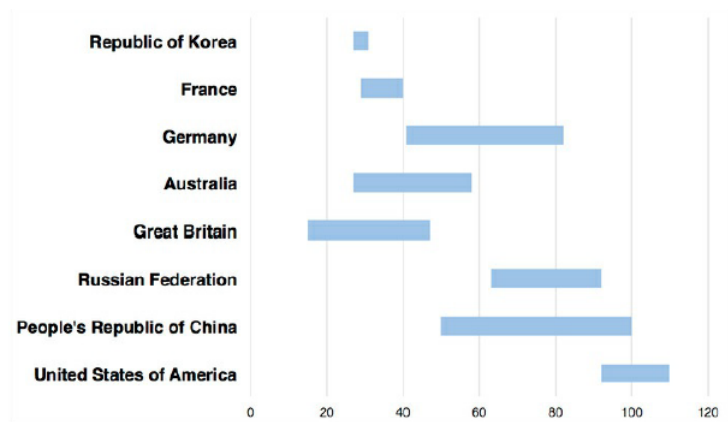


Figura 2.8: Gráfica de barras flotantes

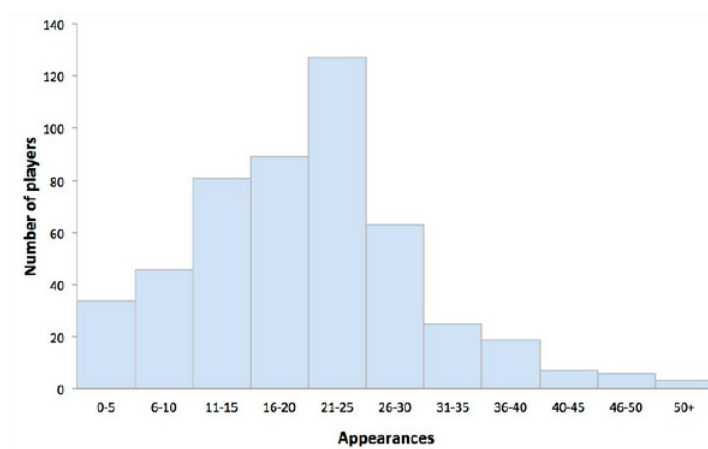


Figura 2.9: Histograma

- **Cambios en el tiempo:** gráfica de línea (**Figura 2.10**), gráfica de área (**Figura 2.11**) y gráfica de áreas apiladas (**Figura 2.12**).

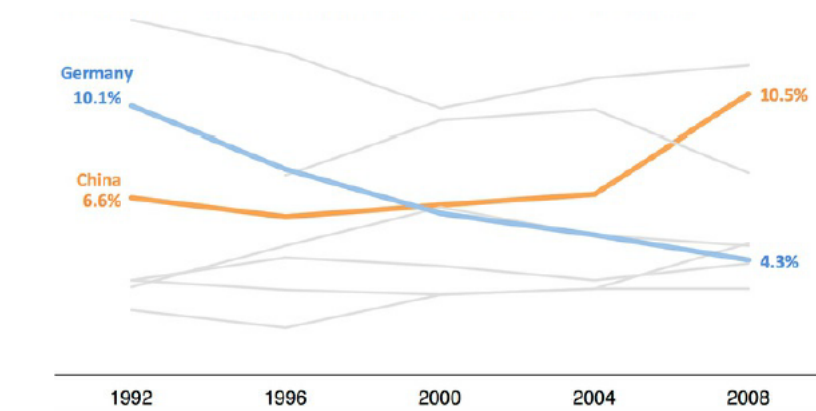


Figura 2.10: Gráfica de líneas



Figura 2.11: Gráfica de áreas

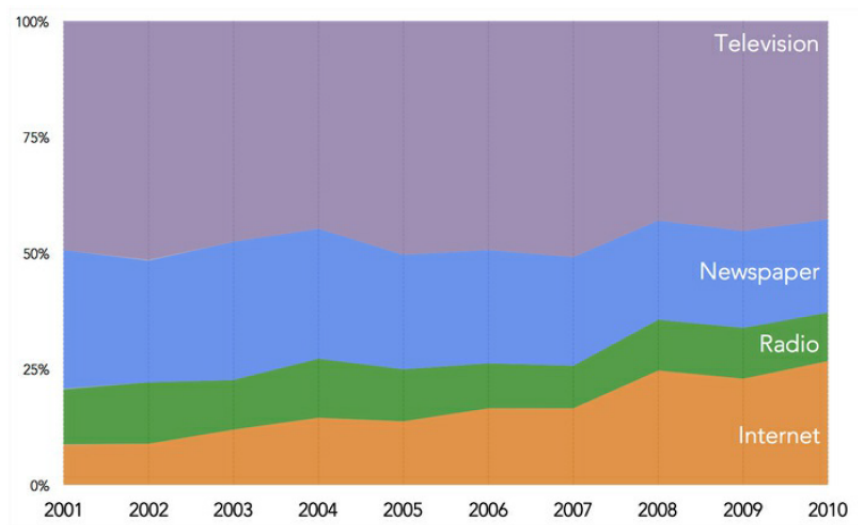


Figura 2.12: Gráfica de áreas apiladas

- **Conexiones y relaciones:** gráfica de dispersión (**Figura 2.13**) y gráfica de burbujas (**Figura 2.14**).

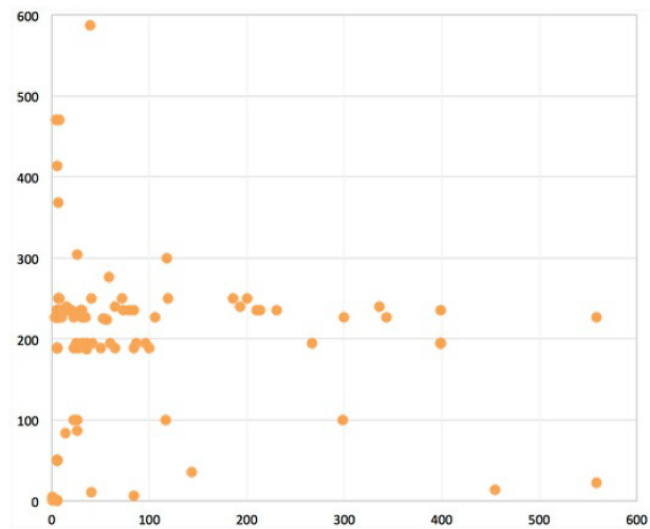


Figura 2.13: Gráfica de dispersión

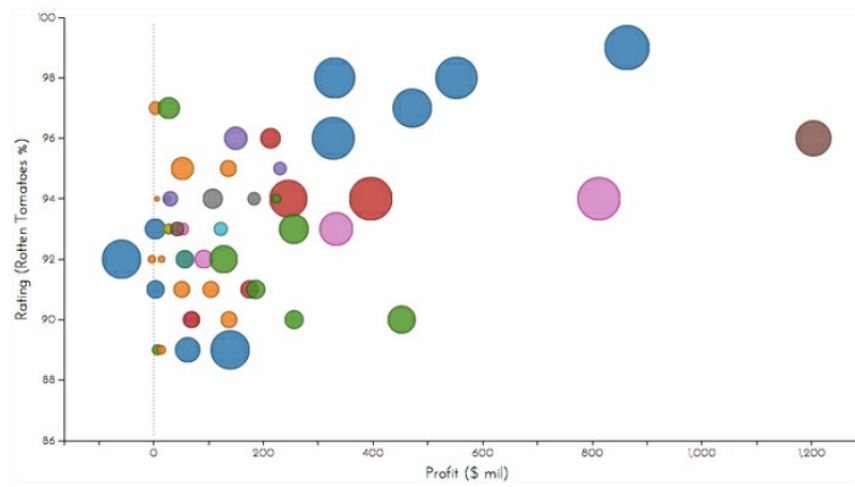


Figura 2.14: Gráfica de burbujas

# Capítulo 3

## Marco Tecnológico

Este capítulo cubre la investigación y evaluación dada sobre herramientas de apoyo posibles a usar en el trabajo para simplificar el desarrollo. El capítulo inicia introduciendo aquellas herramientas que facilitan la construcción de las visualizaciones sobre los datos. Por último, el capítulo presenta las herramientas que soportan, optimizan y simplifican el desarrollo general de la aplicación web.

### 3.1. Tecnologías para la visualización de datos en web

Para comenzar, hay que saber que las visualizaciones para este caso se construirán en el ámbito web, haciendo uso de HTML5 [4] y JavaScript [5]. De las bibliotecas que nos apoyaremos, algunas construyen la gráfica utilizando el componente Canvas [6] de HTML5 y otras utilizando el estándar SVG [7] (Scalable Vector Graphics).

#### 3.1.1. SVG vs Canvas [8]

El elemento Canvas es literalmente, un lienzo donde se va a pintar la gráfica, el proceso de construcción es más complejo y manual ya que la manera

para dibujar consiste en pintar pixel por pixel, por lo que el cambio de resoluciones afecta lo dibujado. El elemento SVG (Scalable Vector Graphics) representa un vector escalable, donde cada elemento de la gráfica, ya sea una caja, círculo, texto, o imagen representa un sub-elemento del SVG general. Adicionalmente SVG permite el manejo de eventos y los cambios de resoluciones no afectan la visualización, esto hace que sea más fácil trabajar y personalizar los elementos de la gráfica a dibujar.

Como usaremos bibliotecas para la creación de las gráficas no debemos preocuparnos por la dificultad que toma la construcción con Canvas comparado con SVG, ya que la biblioteca lo hará por nosotros. El punto importante es que Canvas tiene un rendimiento mejor que el SVG cuando la gráfica maneja muchos objetos, debido a que cada objeto de la gráfica SVG es un elemento que impacta en el DOM [9] del HTML y esto hace que más memoria RAM sea consumida.

### 3.1.2. Bibliotecas basadas en Canvas

A continuación se presentará una lista de bibliotecas que construyen las gráficas haciendo uso de Canvas, acompañada de una lista de las gráficas principales que incluyen:

- **Processing.js**<sup>4</sup>: Es un lenguaje de programación visual. Al ser un lenguaje de programación visual queda claro que su objetivo es general y no únicamente gráficas, por lo que nos permite elaborar desde animaciones hasta juegos. Basta con aprender sus definiciones propias para poder usarlo.
- **Chartjs**<sup>5</sup>: Biblioteca con aporte de 5 tipo de gráficas: gráfica de línea, barra, área polar, circular (dona), dispersión. En particular, las gráficas de esta biblioteca presentan un buen diseño adaptativo a distintos tipos de pantalla que pueden ser personalizado, incluyendo animaciones. Adicionalmente, podemos extender las funcionalidades descargando y configurando *plugins* elaborados por otras personas.
- **Echarts**<sup>6</sup>: Posee una gran variedad de gráficas personalizables y dando

---

<sup>4</sup><http://processingjs.org/>

<sup>5</sup><http://www.chartjs.org/>

<sup>6</sup><https://ecomfe.github.io/echarts-doc/public/en/index.html>



la posibilidad de habilitar animaciones. Ofrece soporte para la mayoría de los navegadores web y buena usabilidad para los dispositivos móviles, tanto rendimiento como adaptabilidad a la pantalla. La biblioteca con todas las gráficas y componentes incluidos ocupa alrededor de 500KB, pero es posible solo descargar algunos tipos de gráficas para reducir el tamaño de la biblioteca. Entre las gráficas disponibles se tienen: gráfica de línea, barra, área, mapa, circular (dona), dispersión, velas, grafos, boxplot, paralela, embudo y *themeriver* (variación temática sobre el tiempo).

### 3.1.3. Bibliotecas basadas en SVG

A continuación se presentará una lista de bibliotecas que construyen las gráficas haciendo uso de SVG, acompañada de una lista de las gráficas principales que incluyen:

- **Raphael.js**<sup>7</sup>: Es una pequeña biblioteca *cross-browser*, es decir, soportada por la mayoría de los navegadores web, con la capacidad de ofrecernos herramientas para elaborar cualquier visualización con vectores. No está orientada solo a la elaboración de gráficas, por lo que podemos crear cualquier visualización, ya sea juegos, alguna especie de arte o animación.
- **Google Chart**<sup>8</sup>: Biblioteca elaborada por Google, donde ofrecen aproximadamente 29 tipos de gráficas, animadas, con un estilo minimalista y soporte para muchos navegadores, incluyendo versiones viejas. Además, nos permite configurar y personalizar las gráficas a nuestro gusto. La biblioteca solo ocupa 70 KB. Entre los tipos de gráficas disponibles destacan: gráfica de línea, barra, área, mapa, circular (dona), dispersión, intervalos, boxplot, velas, treemap y línea en el tiempo.
- **Plotly.js**<sup>9</sup>: Es una biblioteca que está basada (construida) con ayuda de la biblioteca D3 y stackgl<sup>10</sup>. Tiene 20 tipos de gráficas, incluyendo en 3D (tres dimensiones) con un diseño agradable y con una cómoda

---

<sup>7</sup><http://dmitrybaranovskiy.github.io/raphael/>

<sup>8</sup><https://developers.google.com/chart/>

<sup>9</sup><https://github.com/plotly/plotly.js/>

<sup>10</sup>Es un ecosistema para WebGL <http://stack.gl/>

caja de herramientas flotante para interactuar con la gráfica. Entre los tipos gráficas disponibles se tienen: gráfica de línea, barra, área, circular (dona), mapa, dispersión, boxplot, velas, treemap e histogramas (2D y 3D).

### 3.1.4. D3 (Data-Driven Documents)

D3 (Data-Driven Documents)<sup>11</sup> es considerada una de las bibliotecas más potentes para la manipulación de datos, con varios años en desarrollo y con una comunidad bastante activa. Con esta biblioteca se puede llegar a construir casi cualquier tipo de gráfica deseable en SVG o HTML Canvas, gracias a que tenemos un control total sobre la construcción y diseño de la gráfica. También se dispone de *plugins* y gran cantidad de ejemplos aportados por otras personas que pueden ayudarnos a facilitar la programación de la gráfica. La biblioteca tiene un tamaño base de 230 KB aproximadamente en su versión actual (v4.7.3). Para hacer posible la construcción de una gráfica, D3 incluye los siguientes elementos claves:

- **Selecciones:** modificar elementos de manera imperativa, siendo menos tediosa a la tradicional

```
1 d3.selectAll('p').style('color', 'white');
```

Podemos modificar atributos o estilos, registrar eventos, agregar, eliminar nodos logrando cambiar HTML o texto contenido.

- **Propiedades dinámicas:** los estilos, atributos y otras propiedades pueden ser especificadas como funciones de datos, es decir no siempre reciben constantes.

```
1 d3.selectAll("p").style("color", function() {  
2   return "hsl(" + Math.random() * 360 + ", 100%, 50%)" ;  
3 });
```

- **Entrar y salir:** facilita agregar y/o remover elementos en un grupo de datos.

```
1 var p = d3.select("body")  
2   .selectAll("p")
```

---

<sup>11</sup><https://d3js.org/>

```

3     .data([4, 8, 15, 16, 23, 42])
4     .text(function(d) { return d; });
5
6 // Entrar
7 p.enter().append("p")
8     .text(function(d) { return d; });
9
10 // Salir
11 p.exit().remove();

```

En casos donde se busca optimizar esta sección es útil debido a que podemos establecer una navegación en la visualización y solo mostrar un grupo de elementos donde el resto se elimina, de tal forma que se vayan agregando y eliminando elementos a medida que se realicen acciones sobre la gráfica.

- **Transiciones:** existen controles para las animaciones, como la duración y tiempo de aplazo.

```

1 d3.selectAll("circle").transition()
2     .duration(750)
3     .delay(function(d, i) { return i * 10; })
4     .attr("r", function(d) { return Math.sqrt(d * scale); });

```

La biblioteca D3 no introduce una nueva representación visual como hace Raphael.js y Processing.js (bibliotecas mencionadas anteriormente), debido a que se trabaja directamente con estándares web (HTML, CSS [10], SVG), por ejemplo, podemos crear una gráfica en SVG y luego darle estilo con un archivo externo CSS. Sin embargo, estas tres (3) bibliotecas por el hecho de darnos una libertad total al construir visualizaciones requieren un tiempo considerado para aprender y poder usarlas debidamente.

### 3.1.5. Evaluación de Bibliotecas

Es importante saber que para nuestro caso todas las bibliotecas mencionadas anteriormente son las que mejor se adaptan según las necesidades, aún así se realizaron ciertas evaluaciones para decidir cuál utilizar. Para la evaluación se consideró lo siguiente:

## ■ Variedad de gráficas

Para nuestro trabajo es necesario varios tipos de gráficas, ya que se realizarán visualizaciones con cantidad y tipo de datos distintos, por lo tanto es indispensable que existan diferentes tipos de gráficas para cada situación a visualizar en particular. Se revisó todos los tipos de gráficas que podían ofrecer cada una de las bibliotecas. Si es necesario un diseño o tipo de gráfica bastante particular probablemente para ese caso la mejor opción sería D3, Raphael.js o Processing.js.

## ■ Rendimiento

Definimos rendimiento como el comportamiento que toma la biblioteca al construir una gráfica considerablemente pesada (con inmensa cantidad de datos). El problema que podemos presentar al visualizar grandes cantidades de datos es que la vista donde está la gráfica se perciba con cierta lentitud y en el peor de los casos el navegador (browser) se detenga, es decir, deje de trabajar. Afortunadamente, hay algunas bibliotecas que optimizan las gráficas y aplican un estilo de paginación, donde no dibujan todos los puntos a la vez, sino a medida que nos profundizamos en la gráfica.

Se sometieron algunas de las bibliotecas a la creación de una gráfica desde 10.000 hasta 1.000.000 de datos. En la **Figura 3.1** podemos apreciar una aproximación del resultado basado en la cantidad de objetos que pudieron soportar, en donde el eje 'y' corresponde al número de objetos, donde M representa el millón. Cabe destacar que D3, Processing y Raphael no fueron considerados ya que su rendimiento es totalmente relativo a como se programe la gráfica.

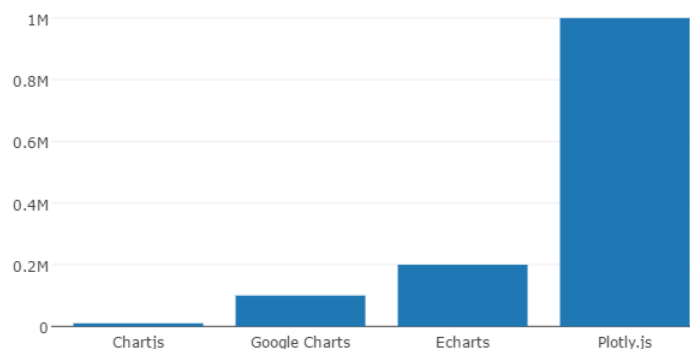


Figura 3.1: Comparación de bibliotecas de visualización

Al analizar la evaluación, quedamos con tres (3) candidatos: D3, Echarts y Plotly. Donde cada uno predomina en distintos escenarios necesarios para nuestro trabajo. Adicionalmente, D3 al ser una biblioteca que nos ofrece la mayor libertad para elaborar gráficas a nuestro gusto dará aquel soporte en gráficas sumamente específicas, también viene acompañada de muchos ejemplos elaborados por la comunidad que pueden servir de apoyo. Plotly a pesar de construir las visualizaciones en SVG logra un excelente rendimiento, aún mejor que el resto de las bibliotecas, además su diseño es agradable y nos ofrece una caja de herramientas para interactuar con la gráfica. Como último candidato a Echarts, por su gran variedad de ejemplos y los posibles tipos de gráficas que nos puede ofrecer, agregando que, dan soporte con gráficas que son usables en dispositivos móviles logrando además un buen rendimiento para aquellos sistemas de bajos recursos.

## 3.2. Tecnologías para el desarrollo web

Brindar una buena experiencia de usuario en una aplicación puede llegar a ser un gran desafío donde existen varios elementos que afectan directamente, como el **diseño**, ya que es importante jugar un buen estilo, de carácter

agradable, y con animaciones amigables. El **rendimiento**, donde la aplicación debe desempeñar una buena fluidez de respuesta y **correctitud**, que se contemple una interfaz que tenga sentido con la funcionalidad de la misma.

Cuando entramos en el área web, nuestro caso una aplicación web, llegan algunos desafíos adicionales como aplicar soporte de diseño para variedad de densidades de pantallas y versiones de navegadores web, esto se debe a que nuestra aplicación estará expuesta en la web, donde puede ser accedida desde tabletas, móviles, computadores de escritorio, televisores, y cualquier dispositivo que tenga conexión a internet y un navegador. También el peso de la aplicación, incluyendo códigos, fuentes e imágenes impacta considerablemente al tiempo de espera para conexiones lentas a internet.

En este capítulo presentaremos herramientas que nos facilitarán el trabajo para lidiar las problemáticas mencionadas segmentando la aplicación por: arquitectura, diseño y utilidades extras a emplear. Cabe acotar que para el manejo de estas herramientas es necesario tener conocimiento sobre JavaScript, TypeScript<sup>12</sup> (subconjunto de JavaScript con un sistema de tipos más robusto), CSS y HTML.

### 3.2.1. Arquitectura

La aplicación adoptará una arquitectura llamada *Single-page Application* [11] (SPA) que significa aplicación de una página, donde se busca englobar toda la aplicación en una vista, logrando cargar la base de la aplicación completa y luego dinámicamente mediante JavaScript hacer la transición de las vistas. La gran ventaja que podemos sacar de SPA es la fluidez de la aplicación entre cambios de vistas, ya que todos estos elementos ya fueron cargados previamente. Por contraparte, la carga inicial suele ser pesada por traerse elementos de más, pero de igual forma esta puede optimizarse para traer solo los recursos necesarios.

Mayormente nuestra aplicación tendrá comunicación casi constante con un API que nos ofrecerá cantidades de datos para nosotros transmitir visualmente, por lo tanto manejaremos peticiones HTTP<sup>13</sup> (Hypertext Transfer

---

<sup>12</sup>TypeScript <https://www.typescriptlang.org/>

<sup>13</sup>HTTP: es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.

Protocol) con AJAX [12] (Asynchronous JavaScript And XML), cabe destacar que al ser asíncrono evitamos que afecte el flujo principal de la aplicación.

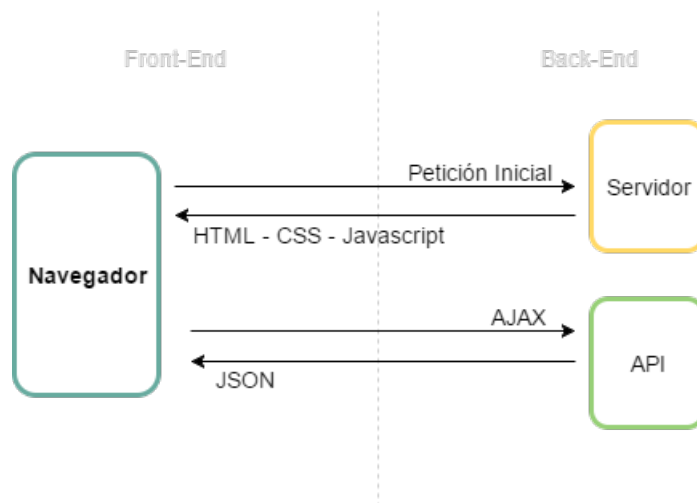


Figura 3.2: Ciclo de vida de una aplicación SPA

El ciclo de una aplicación SPA como se muestra en la **Figura 3.2** requiere por lo general una primera petición para cargar las vistas, funciones y diseño, el resto de las peticiones se dirigen al API a través de AJAX para pedir los datos a visualizar.

A continuación presentaremos dos (2) bibliotecas que nos permiten definir la arquitectura de la aplicación: Angular y jQuery.

## Angular

Angular<sup>14</sup> es un framework muy popular creado por Google destinado a construir aplicaciones web, donde se maneja la lógica de la aplicación con JavaScript. Para la versión 2 de Angular, que es la que se contempla para este proyecto, se logra una estructura modular en la aplicación, donde cada elemento es considerado un componente que se construye de manera aislada, por lo tanto no se ve afectado por el resto de los mismos. Claramente, el enfoque de Angular sigue la arquitectura SPA, donde logra optimizar y resolver posibles problemas con los estados de transiciones, carga inicial, caching y

<sup>14</sup>Angular 2 <https://angular.io/>

mejor manejo de peticiones HTTP.

Es importante saber que Angular es una biblioteca con un catálogo inmenso de funcionalidades soportado por la mayoría de los navegadores, donde ofrece desde mecanismos de seguridad para evitar *Cross-site scripting*<sup>15</sup> (XSS) hasta un paquete completo de animaciones.

A continuación, se muestra un ejemplo de código representando un componente en Angular:

```
1 // app.component
2 import { Component } from '@angular/core';
3
4 @Component({
5   selector: 'my-app',
6   template: `<h1>Hola Mundo</h1>`
7 })
8 export class AppComponent { name = 'Angular'; }
```

Podemos observar que para declarar un componente es necesario importar primero el paquete de Angular y luego llamar al decorador `@Component()`, en donde es pasado un objeto identificado por un **selector**, que corresponde a un alias del componente y el **template** que puede ser una ruta de un archivo HTML o este caso una pieza de código HTML.

```
1 // app.module
2 import { NgModule }      from '@angular/core';
3 import { BrowserModule } from '@angular/platform-browser';
4 import { AppComponent }  from './app.component';
5
6 @NgModule({
7   imports:      [ BrowserModule ],
8   declarations: [ AppComponent ],
9   bootstrap:   [ AppComponent ]
10 })
11 export class AppModule { }
```

Es indispensable al menos un módulo en Angular por lo tanto en el código anterior estamos definiendo un módulo con el decorador `@NgModule()`,

---

<sup>15</sup>Cross-site scripting [https://es.wikipedia.org/wiki/Cross-site\\_scripting](https://es.wikipedia.org/wiki/Cross-site_scripting)



lo importante de aquí es que en dicho módulo declaramos el componente anteriormente creado.

Otras características que destacan a Angular son:

### ■ Directivas

Es una manera de lograr que las vistas sean dinámicas, es decir, los componentes HTML pueden tener atributos especiales de Angular para poder modificar elementos del DOM.

```
1 <li *ngFor="let item of arrayItem"></li>
2 <app-detail *ngIf="selectedItem"></app-detail>
```

La directiva **\*ngFor** nos permite iterar en un ciclo, originando un **<li>** por cada elemento de **arrayItem** y **\*ngIf** nos permite mostrar o no elementos mediante una condición.

### ■ Servicios

Pueden representar cualquier función, valor o características que la aplicación necesite. Mayormente los componentes son consumidores de estos servicios, por ejemplo: una configuración de la aplicación, un servicio que hace peticiones HTTP a un API, un servicio encargado de actualizar una barra de navegación, un servicio para mostrar mensajes estilo Logs, etc.

```
1 // logger.service.ts
2 export class Logger {
3   log(msg: any) { console.log(msg); }
4   error(msg: any) { console.error(msg); }
5   warn(msg: any) { console.warn(msg); }
6 }
```

El código de arriba representa un ejemplo de un servicio de Logs, en donde nos ofrece una clase con tres (3) variedades de avisos a consola.

## jQuery

jQuery<sup>16</sup> es una biblioteca ligera y sencilla, que lleva bastante tiempo siendo usada. Su finalidad es ofrecer soluciones a funciones complejas de manera sencilla principalmente para la manipulación de elementos HTML y eventos, animaciones y AJAX. jQuery es considerado una alternativa manual si se quiere construir una aplicación SPA, ya que directamente no ofrece una solución completa, pero existen *plugins* y/o guías para lograr dicha arquitectura.

### ■ Manipulación de elementos

```
1 $( "#button.green" ).html( "Next" );
```

### ■ Manejo de Eventos

```
1 var hiddenBox = $( "#banner-message" );
2 $( "#button-container button" ).on( "click", function( event ) {
3     hiddenBox.show();
4 } );
```

### ■ Petición HTTP Asíncrona con AJAX

```
1 $.ajax({
2     url: "/api/getWeather",
3     data: {
4         zipcode: 97201
5     },
6     success: function( result ) {
7         $( "#weather-temp" )
8         .html( "<strong>" + result + "</strong> degrees" );
9     }
10 } );
```

## 3.2.2. Diseño

Es importante que nuestra aplicación presente un buen diseño, donde se simplifique la finalidad de la aplicación, además de ser capaz a adaptarse a distintos tamaños de pantallas. No hay un guion o estilo definido para el

---

<sup>16</sup>jQuery <https://jquery.com/>

diseño, la idea es aplicar colores que combinen, componentes sencillos como: botones, selectores, barra de menú, diálogos, iconos, *inputs* (entradas de información), texto y componentes no tan sencillos como las gráficas para visualizar los datos. Todos estos componentes harán posible una interfaz que sea capaz de representar un editor de visualizaciones, en donde recibiremos apoyo de bibliotecas que nos ofrecen estos componentes bien formados y listos para usar.

## Angular Material

Angular Material<sup>17</sup> es una biblioteca que ofrece componentes siguiendo un diseño llamado Material (Material Design)<sup>18</sup> creado por Google. Para poder integrar esta biblioteca es requerido Angular 2, por lo tanto, esta opción se ve atada a usar dicho framework. Lo que destaca de Angular Material es la variedad de componentes, que además son totalmente adaptativos gracias a una biblioteca incluida llamada *Flex Layout*<sup>19</sup> que nos ofrece propiedades sencillas de usar para corresponder el tamaño de los componentes en distintos escenarios. Adicionalmente, es soportado por la mayoría de los navegadores modernos.

Angular Material al estar atado al framework nos da la posibilidad de exportar sólo los componentes que necesitemos y no todos los de la biblioteca, además nos permite añadir soporte de gestos a los componentes como *toggle* o *slider*.

Entre tantos componentes, se mostrará ejemplos de algunos a continuación:

### ■ Botones

```
1 <button md-raised-button>Raised button</button>
2 <button md-fab><md-icon>check</md-icon></button>
```

En donde el atributo **md-raised-button** representa un botón con elevación, **md-fab** es un botón mayormente flotante con forma circular. Podemos observar que existe otro elemento **md-icon** que nos permite

---

<sup>17</sup>Angular Material <https://material.angular.io/>

<sup>18</sup>Material Design <https://material.io/>

<sup>19</sup>Flex Layout <https://github.com/angular/flex-layout>

colocar un ícono a través de una fuente especial llamada *Material Design Icons*<sup>20</sup>, obteniéndola con el siguiente código desde nuestro archivo HTML.

```
1 <link
2 href="https://fonts.googleapis.com/icon?family=Material+Icons"
3 rel="stylesheet">
```

### ■ Spinner de Progreso

```
1 <md-progress-spinner
2     class="example-margin"
3     [attr.color]="color"
4     [mode]="mode"
5     [value]="value">
6 </md-progress-spinner>
```

El *spinner*, que hace referencia a un elemento circular con acción de progreso, se puede lograr usando **md-progress-spinner** en donde puede tener como atributo un color, un modo para representar si girará finitamente o infinitamente y un valor que representa el porcentaje de progreso actual.

### ■ Barra de Herramientas

```
1 <md-toolbar>My App</md-toolbar>
```

Con **md-toolbar** logramos crear una barra en donde podremos colocar texto e iconos con acciones, por ejemplo, la barra principal superior de la aplicación.

## Bootstrap

Bootstrap<sup>21</sup> ofrece un gran catálogo de componentes, desde distintos tipos de botones, formularios, barras de menú, hasta diálogos. Recientemente lanzaron la versión 4, en donde se reescribió la mayoría del proyecto corrigiendo bastantes errores, sin embargo esta versión se encuentra en fase alfa, esto quiere decir que no es estable. Bootstrap es una biblioteca elaborada por

---

<sup>20</sup>Material Design Icons <https://material.io/icons/>

<sup>21</sup>Bootstrap <http://getbootstrap.com/>

Twitter, donde es posible la construcción rápida de una interfaz tanto para escritorio como para móvil, ofreciendo gran soporte para la mayoría de los navegadores y también posibilidad de adaptarse a distintas dimensiones de pantallas. Para que Bootstrap funcione es necesario la biblioteca jQuery que anteriormente mencionamos.

Bootstrap tienen su propio sistema *grid* para escalar los componentes según cambie el tamaño de la pantalla, en donde se distinguen tres medidas: **lg** (largo), **md** (mediano), **sm** (pequeño) y **xs** (extra pequeño). Estas medidas pueden aplicarse a columnas o filas, representadas como **col** y **row**. Por cada fila puede haber doce (12) columnas, si este es superado el elemento faltante irá posicionado abajo.

```
1 <div class="row">
2   <div class="col-md-4">.col-md-4</div>
3   <div class="col-md-4">.col-md-4</div>
4   <div class="col-md-4">.col-md-4</div>
5 </div>
6 <div class="row">
7   <div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8</div>
8   <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
9 </div>
```

Entre los componentes considerados a usar:

#### ■ Botones

```
1 <button type="button" class="btn btn-default">Default</button>
2 <button type="button" class="btn btn-primary">Primary</button>
3 <button type="button" class="btn btn-success">Success</button>
4 <button type="button" class="btn btn-info">Info</button>
5 <button type="button" class="btn btn-warning">Warning</button>
6 <button type="button" class="btn btn-danger">Danger</button>
```

Bootstrap nos proporciona un abanico de botones con estilos definidos, en donde cada uno juega un color de fondo distinto.

#### ■ Tipografía

```
1 <span class="glyphicon glyphicon-star"
2   aria-hidden="true"></span> Star
```

Es posible incluir íconos gracias a que Bootstrap trae su propia tipografía.

#### ■ Barra de Progreso

```
1 <div class="progress">
2   <div class="progress-bar" role="progressbar"
3     aria-valuenow="60" aria-valuemin="0"
4     aria-valuemax="100" style="width: 60%;">
5     <span class="sr-only">60% Complete</span>
6   </div>
7 </div>
```

#### ■ Barra de Herramientas

```
1 <nav class="navbar navbar-default">
2   <div class="container-fluid">
3     <div class="navbar-header">
4       <a class="navbar-brand" href="#">
5         
6         <h1>Title</h1>
7       </a>
8     </div>
9   </div>
10 </nav>
```

Es posible posicionar una barra superior de manera fija, con cualquier elemento adentro de ella, pero para ellos debes cumplir con los 3 primeros elementos padres: el primer **nav** y los dos siguientes **div**.

### 3.2.3. Utilidades

Existen distintas bibliotecas de propósito más específico cuyo uso se ha vuelto cotidiano, donde pueden solucionar tareas complejas o engorrosas de implementar en JavaScript, específicamente para nuestro proyecto existe la necesidad de preparar ciertos conjuntos de datos complejos para poder construir una gráfica, al igual que manejar varios formatos de fechas.

## Lodash<sup>22</sup>

La manipulación de ciertos objetos, arreglos y strings complejos pueden contemplar soluciones un tanto ilegible, por lo tanto la biblioteca Lodash nos otorga utilidades que resuelven dichos problemas de una manera simple. Adicionalmente, cuenta con funciones matemáticas y soluciones para manejar valores null, NaN y undefined. Todas sus funcionalidades son soportadas por la mayoría de los exploradores.

A continuación, se mostrarán unos ejemplos de las ventajas que puede traer lodash a nivel de implementación:

### ■ Verificar variable (no sea null ni undefined)

```
1 // no-lodash
2 if(a != null && a != undefined) {...}
3
4 // lodash
5 if(!_.isNull(a)) {...}
```

### ■ Ciclos

```
1 // no-lodash
2 for(var i = 0; i < 5; i++) {
3   ...
4 }
5
6 // lodash
7 _.times(5, function(){
8   ...
9 });
```

### ■ Números aleatorios

```
1 // no-lodash
2 Math.floor(Math.random() * (max - min + 1)) + min;
3
4 // lodash
5 _.random(min, max);
6 });
```

---

<sup>22</sup>Lodash <https://lodash.com/>

### ■ Seleccionar elementos de un arreglo

```
1 // no-lodash
2 function pick(arr) {
3     var _this = this;
4     var obj = {};
5     arr.forEach(function(key){
6         obj[key] = _this[key];
7     });
8
9     return obj;
10 };
11 objA.pick(['x', 'y'])
12
13 // lodash
14 _.pick(objA, ['x', 'y']);
15 });
```

## Moment<sup>23</sup>

La manipulación de fecha y tiempo en JavaScript suele ser confusa y consume muchas líneas de código, es probable que en nuestra aplicación tengamos que recibir fechas con un formato específico y de la misma forma representar este visualmente de otra. Moment es una biblioteca que tiene como finalidad analizar sintácticamente, validar, manipular y mostrar fechas/horas (tiempo) de la forma más simple posible. Adicionalmente, esta biblioteca ofrece extensiones para poder realizar ciertas funcionalidades más específicas.

A continuación mostraremos ejemplos de cosas interesantes que podemos lograr con dicha biblioteca:

### ■ Obtener la fecha/hora de hoy con formato

```
1 // date
2 moment().format("DD/MM/YYYY");
3 // time
4 moment().format("HH:mm:ss");
```

---

<sup>23</sup>Momentjs <https://momentjs.com/>



- **Agregar/Quitar tiempo**

```
1 // add
2 moment().add(7, 'days');
3 moment().add(1, 'week')
4 // subtract
5 moment().subtract(7, 'days');
6 moment().subtract(1, 'week')
```

- **Obtener unidad de tiempo específica**

```
1 // hours
2 moment().hours();
3 // month
4 moment().month();
```

Presentadas estas alternativas, se tomará Angular para la arquitectura y Angular Material para el diseño respectivamente. Esta selección se justifica debido a que Angular como framework para el front-end tiene un enfoque total SPA, además tiene ciertas herramientas integradas, como un manejador de conexiones HTTP que es de bastante utilidad para nuestro caso. También, Angular orienta a programar de manera modular, en donde se logra un código más organizado y reutilizable. Por el lado del diseño, Angular Material es una buena opción, debido a su compatibilidad con el framework, adicionalmente, posee una gran cantidad de vistas adaptativas totalmente controlables mediante directivas de Angular.

# Capítulo 4

## Marco Aplicativo

Este capítulo describe la metodología utilizada para el desarrollo de la solución y la aplicación de la misma, explicando cada una de las tareas en orden cronológico de manera detallada y su resultado.

### 4.1. Metodología

Los objetivos específicos serán desglosados de manera técnica en pequeñas tareas ordenadas por prioridad. El control de estas asignaciones se manejará mediante la plataforma GitHub (aplicación web para alojar repositorios Git) en donde cada tarea será un *issue* a resolver. Github permite proyectar estos *issues* en una pizarra, con el fin de visualizar el estado de cada asignación, los estados definidos son: Por hacer, En progreso, Terminado. Puede verse un ejemplo en la **Figura 4.1**

- **Por hacer:** representa aquellas tareas especificadas, que por el momento son *issues* sin resolver.
- **En progreso:** representa las tareas que están siendo desarrolladas. Cabe destacar que cada asignación se resuelve en una rama distinta del repositorio git haciendo uso del *pull request* para llevar un mayor control del desarrollo de la asignación.
- **Terminado:** representa aquellas tareas culminadas. Cuando se consi-

dera que una tarea está lista, esta tiene que ser mezclada a la rama principal del repositorio, llamada *master*, luego cerrar el *pull request* y el *issue* asociado.

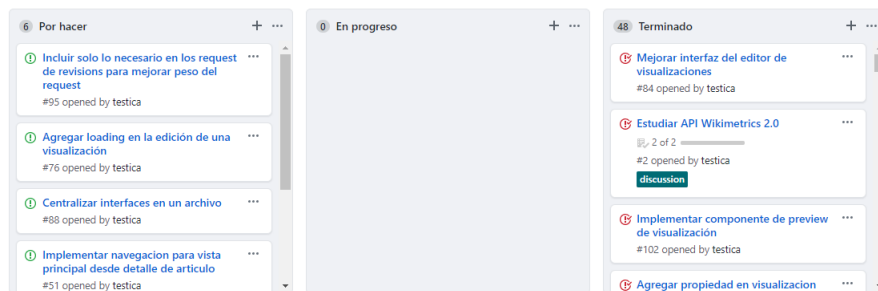


Figura 4.1: Pizarra de Github

La pizarra de Github, véase la **Figura 4.1**, es una herramienta de la metodología Kanban [13], en donde visualizar el flujo de trabajo y hacerlo visible es la base para comprender cómo avanza el trabajo. Sin comprender el flujo de trabajo, realizar los cambios adecuados es más difícil. Una forma común de visualizar el flujo de trabajo es el uso de columnas. Las columnas representan los diferentes estados o pasos en el flujo de trabajo. Kanban logra un desarrollo evolutivo e incremental, donde las soluciones de las asignaciones puede que no sean las mejores comenzando, pero a medida que se itera se va perfeccionando.

## 4.2. Realización de tareas

A continuación, de listarán las tareas de mayor relevancia ordenadas cronológicamente, con la descripción del problema y el resultado.

### 1. Preparar entorno de desarrollo

Antes de empezar el desarrollo se necesita instalar o preparar el ambiente para el uso de las herramientas:

- Instalar Node<sup>24</sup> para servir la aplicación web local.

<sup>24</sup><https://nodejs.org>

- Instalar NPM<sup>25</sup> (Node Package Manager) necesaria para instalar paquetes JavaScript como Angular y dependencias.
- Instalar editor de texto inteligente, en preferencia personal, Visual Studio Code.
- Instalar navegadores modernos, principalmente Google Chrome y Firefox, para reproducir la aplicación web y asegurar el soporte de la misma.

## 2. Elaborar estructura base de la aplicación

Lo principal es tener la estructura base del proyecto en código y tener la configuración del framework Angular lista.

Afortunadamente, existe una herramienta llamada **angular-cli**, que facilita la creación de un proyecto en Angular poniendo lo siguiente en el terminal:

```
1      # instalar angular-cli
2      npm install -g @angular/cli
3      # crear proyecto
4      ng new wiki-history-client
```

## 3. Investigar API de WikiMedia

Es necesario ofrecer los artículos del *watchlist* de los usuarios para que posteriormente puedan analizarlos con la aplicación, dado esto se necesita saber cómo autenticar un usuario y obtener su *watchlist* haciendo uso del API.

En el momento cuando se desempeñó esta tarea hubo un problema con el API de autenticación, debido a que estaban abandonando la manera tradicional y migrando a una más segura usando OAuth2<sup>26</sup>, por motivos de privilegios no fue posible registrar la aplicación con el método nuevo de autenticación.

Debido a esto, no iba a ser posible extraer el *watchlist* sin credenciales de usuarios, por lo que se optó la decisión de construir un API pro-

---

<sup>25</sup><https://www.npmjs.com/>

<sup>26</sup><https://oauth.net/2/>

pia donde se gestionaran usuarios y ellos manualmente agregarían los artículos de interés.

#### 4. Implementar vista de Artículos

La vista principal de la aplicación sería una lista de artículos de interés.

En la versión actual, los datos de los artículos se simularon en una variable, para probar que la vista funcionaba.



Figura 4.2: Lista de artículos

Se usó un elemento de Angular Material llamado `md-card`, para formar el estilo de un artículo. La propiedad `*ngFor` es una directiva de angular que nos permite replicar ese elemento tanta veces como iteraciones tenga. Además hacemos uso de `fxFlex` que nos permite ajustar el tamaño del elemento dependiendo de las dimensiones de la pantalla, véase en la **Figura 4.2** el resultado.

```
1      <md-card
2          *ngFor="let article of articles"
3          fxFlex="30" fxFlex.sm="50" fxFlex.xs="100">
4          {{article.title}}
5      </md-card>
```

#### 5. Implementar vista de detalle de Artículo

Al presionar un artículo de la lista nos tiene que dirigir a un detalle para acceder a más información del mismo.

En la versión actual simplemente se mostrará una vista blanca que representa el detalle. De esta forma, se programó que al pisar un artículo direcciona a la vista del detalle. La ruta del detalle se representa por:

```
/articles/<titulo_artículo>
```

## 6. Implementar servicio (API) para manejar usuarios y configuraciones

Es necesario un servicio que delegue la autenticación y gestión de los recursos persistentes como usuarios y artículos.

Se desarrolló un API usando el micro framework Python Flask<sup>27</sup> y para la persistencia de los datos se usó MongoDB, se consideró una base de datos NoSQL debido a que los datos no están relacionados, son simplemente usuarios con configuraciones personales.

Se implementaron las siguientes rutas en el API:

```
POST /sign-up
POST /sign-in
GET /articles
POST /articles
DELETE /articles/<titulo>
```

Para la autorización de recursos, se usó el mecanismo JWT<sup>28</sup>(JSON Web Token), que consiste en la generación de un token resultado de datos cifrado con una clave privada. De esta forma podemos extraer del token el usuario y corroborar si la solicitud del recurso es válida.

En la versión actual, el modelo de cada usuario se verá representado de la siguiente manera:

```
{
  "username": "admin",
  "password": "202cb962ac59075b964b07152d234b70",
  "articles": [
    {
      "title": "Titulo 1",
      "locale": "es"
    }
  ]
}
```

---

<sup>27</sup>[flask.pocoo.org/](http://flask.pocoo.org/)

<sup>28</sup><https://jwt.io/>

Es importante acotar que las contraseñas son almacenadas usando la función hash MD5<sup>29</sup>.

## 7. Agregar documentación y dependencias de API

A nivel de desarrollo es importante tener instrucciones de cómo hacer funcionar las cosas por si otro desarrollador continúa el trabajo, dado esto, se le agrego documentación y se fijaron las versiones de las dependencias para hacerla funcionar en cualquier momento.

## 8. Implementar vista para iniciar sesión

Se requiere una vista para poder iniciar sesión con un usuario y contraseña.

Se implementó un formulario pidiendo ambos requerimientos, que puede ser accedido bajo la ruta:

`/sign-in`

Se creó un servicio de Angular para abstraer la comunicación con el API para hacer el inicio de sesión:

```
1  import { Injectable } from '@angular/core';
2  import { Http } from '@angular/http';
3  import { environment } from '../environments/environment';
4  import { ISignIn } from './resource';
5
6  @Injectable()
7  export class AuthService {
8
9      private loggedIn = false;
10
11      constructor(private http: Http) {
12          this.loggedIn = !!window.localStorage.getItem('auth_token');
13      }
14
15      signIn(username: string, password: string) {
16          return this.http.post(
```

---

<sup>29</sup><https://en.wikipedia.org/wiki/MD5>

```

17     `${environment.API_URL}/sign-in`,
18     {username, password}
19   )
20   .toPromise()
21   .then(res => {
22     const obj: ISignIn = res.json();
23     // store token
24     window.localStorage.setItem('auth_token', obj.access_token);
25     return obj;
26   });
27   }
28
29   }

```

Un servicio es una instancia *singleton*, que puede ser inyectada y usada en cualquier parte de la aplicación, luego de crear el servicio podemos hacer inicio de sesión con:

```

1   signIn("admin", "1234").then();

```

## 9. Implementar vista para registrar un usuario

Es indispensable poder registrar un usuario para luego poder acceder a él.

Por lo tanto, se creó la vista usando un componente que proyecta un formulario similar al iniciar sesión.

## 10. Implementar componente de sugerencia de artículos de Wikipedia

Una vez creado un usuario, lo siguiente es preparar los artículos que queremos examinar. Como no hay forma de obtener el watchlist se tiene que ofrecerle una manera de buscar los artículos de Wikipedia a desear.

Se implementó un input que sugiere artículos de Wikipedia a medida que escribes cualquier texto, además se consideró el idioma, véase la **Figura 4.3**.





Figura 4.3: Caja de sugerencias de artículos de Wikipedia

## 11. Implementar componente y servicio para agregar artículo

Surge la necesidad de persistir los artículos agregados por los usuarios, por lo que se tiene que habilitar la opción para crear artículos y extraerlo asincrónamente usando el API de Wikimetrics 2.0.

Se abstrae la petición al API para crear un artículo a través de un servicio, y se encapsula el componente de la tarea anterior en otro componente que interactuó con el servicio, véase la **Figura 4.4**.

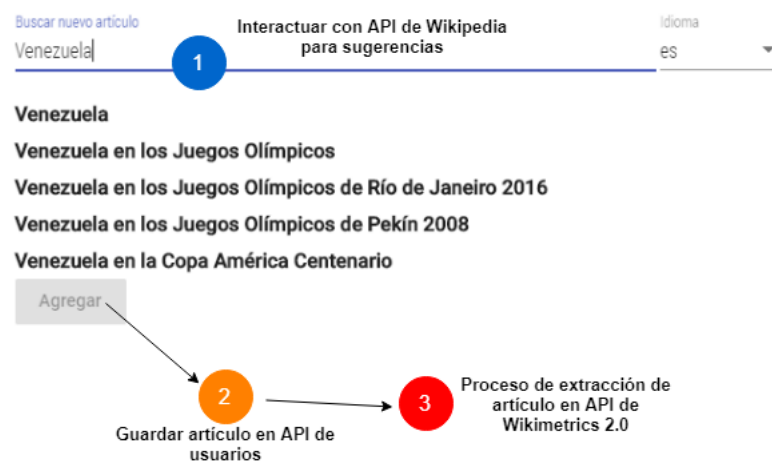


Figura 4.4: Flujo del agregar artículo. En el punto (1) se accede al API de Wikipedia para obtener los artículos sugeridos, luego al presionar agregar sucede el punto (2) que hace un request al API nuestra de usuarios para agregar el artículo y luego en el punto (3) se hace un request en el API de Wikimetrics 2.0 para activar el proceso de extracción del artículo

## 12. Implementar servicio para obtener lista de artículos

Para la versión actual, se estaba trabajando con una lista de artículos falsa. Por lo tanto surge la necesidad de poder pedir la lista real de artículos asociada a un usuario.

Se implementa un servicio que abstrae la lógica de realizar la petición a la ruta para pedir los artículos.

## 13. Crear componente dedicado para los artículos en el listado

El estilo de los artículos en el listado es hasta ahora muy simple y vacío.

Se encapsula la lógica de la carta de un artículo en un componente, para abstraer funcionalidades complejas y se muestra más información del mismo, con un mejor estilo.

## 14. Actualizar estado del artículo cada cierto tiempo

Se sabe que se tiene que hacer un proceso de extracción del artículo para que el API de Wikimetrics pueda ofrecer datos a visualizar, pero actualmente no se tiene forma de saber si el artículo ya fue extraído o no, de alguna forma se tiene que comunicar esa información al usuario para que esté al tanto del proceso. Un artículo puede tardar varios minutos en extraerse, todo esto depende del tamaño y cantidad de ediciones.

Cuando un artículo es mandado a extraer, este responde con un código que nos servirá para preguntar por su estado. Replicamos esta lógica en el lado del API de usuarios, guardando el código y su estado actual:

```
1      {  
2          "locale": "es",  
3          "extract": {
```

```

4         "status": "pending",
5         "id": "665d387e-e547-4275-8abf-1076eacf8f92"
6     },
7     "title": "Universidad Central de Venezuela"
8 }

```

El proceso de extracción puede pasar por 4 estados: *pending* (**Figura 4.5**), *success* (**Figura 4.6**), *failure*, *in progress*.

Para dar una interacción mas precisa al usuario, cada 10 segundos se hace un request para comprobar el estado y se actualiza en nuestra API, la condición de parada es que este *success* o *failure*

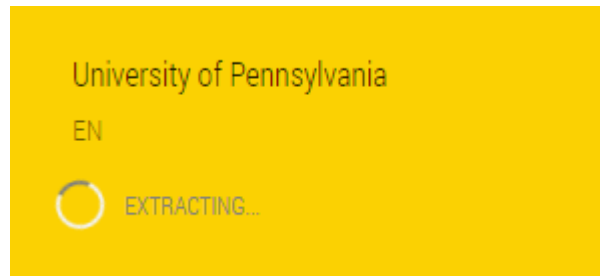


Figura 4.5: Estilo de un artículo en estado pendiente de extracción

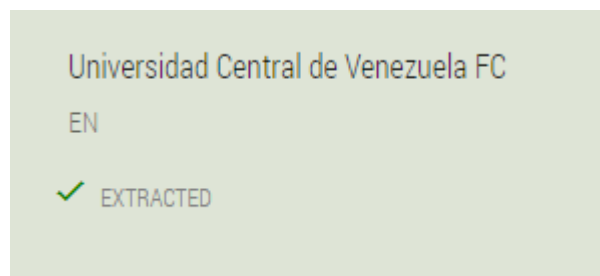


Figura 4.6: Estilo de un artículo en estado exitoso de extracción

## 15. Manejar autorización en las vistas

Hasta este punto del desarrollo no hay ningún método que proteja las vistas que requieran autenticación, es decir anteriormente se puede intentar acceder a la vista del listado de artículos sin haber iniciado

sesión, queda claro que al entrar a esta vista la petición daría error por seguridad del API.

Se agrega una capa más de seguridad a nivel de cliente, para que el usuario no puede entrar sin autenticación a dichas vistas. Para esto usaremos guardias de rutas, a continuación la definición del guardia:

```
1   import { Injectable } from '@angular/core';
2   import { CanActivate, Router } from '@angular/router';
3   import { AuthService } from '../auth.service';
4
5   @Injectable()
6   export class AuthGuard implements CanActivate {
7
8       constructor(
9           private authSvc: AuthService,
10          private router: Router) {}
11
12       canActivate() {
13           if (!this.authSvc.isSigned()) {
14               this.router.navigate(['/sign-in']);
15           }
16
17           return this.authSvc.isSigned();
18       }
19   }
```

Podemos observar que antes de navegar a una ruta la función **canActivate()** se llamará, esto verifica si hay una sesión de usuario válida, en caso contrario redirige al inicio de sesión.

## 16. Implementar barra superior (navbar)

Es importante desplegar una barra superior que muestre información extra y despliegue acciones, principalmente navegación a vista principal y cerrar sesión.

Se implementó un componente para la barra y un servicio para cambiar sus configuraciones, como título, acciones y cierre de sesión, véase la **Figura 4.7**.



Figura 4.7: Barra superior

**17. Implementar ruta en API para obtener un artículo en específico:**

Al entrar al detalle de un artículo se necesita solicitar la información de ese artículo en particular.

Se implementó en API la nueva ruta:

```
GET /articles/<idioma>/título
```

Luego se adapta al componente del detalle de artículo.

**18. Implementar servicio para pedir información sobre revisiones de un artículo**

En el detalle de artículo se mostrará cierta información básica sobre las ediciones, como el número total de ediciones y las últimas 20.

Para tener el número total de ediciones se tiene que hacer uso de una ruta que Wikimetrics provee:

```
GET /api/v1/count?locale=en&title=Venezuela
```

Para tener la información de las últimas 20 ediciones ordenadas por fecha descendientemente:

```
GET /api/v1/revisions?locale=en&title=Venezuela  
&page_size=20&sort=desc
```

Se encapsularon en el servicio de wikimetrics para abstraer su uso.

**19. Implementar componente de visualización**

Algo sumamente importante es la posibilidad de desplegar gráficas, en distintos casos, la aplicación necesitará de un componente que muestre gráficas variables.

Se abstraerá la visualización de un gráfica en un componente, apoyándose de la biblioteca Plotly. El componente acepta las siguientes entradas para variar su configuración:

```
1  @Input() chartType: 'pie' | 'bar' | 'scatter'
2    | 'scattergl' | 'line' | 'area';
3  @Input() chartX: string[] | number[];
4  @Input() chartY: string[] | number[];
5  @Input() chartXTitle = '';
6  @Input() chartYTitle = '';
7  @Input() chartTitle = '';
```

En donde **chartType** es para pasarle el tipo de gráfica que queremos desplegar, **chartX** y **chartY** corresponde el set de datos para ambos ejes, **chartXTitle** y **chartYTitle** describe el título de ambos ejes y **chartTitle** hace referencia al título de la visualización.

Se consideró hacer el componente de tal forma que fuera reactivo al cambio dimensión de la pantalla, es decir, las visualizaciones son *responsive*.

```
1  this.resizeService.onResize$.subscribe(() =>
2    Plotly.Plots.resize(this.gd)
3  );
```

Haciendo uso del patrón *observer*, con ayuda de la biblioteca `ReactiveX`<sup>30</sup>, cada vez que la pantalla cambia su posición se notifica mediante un servicio y se ajusta el tamaño de la visualización.

## 20. Mostrar información específica en el detalle del artículo

Al entrar al detalle del artículo hay que encontrar información específica y que sume valor.

Por lo tanto, se considera colocar el número de ediciones, fecha de última edición, autor de la última edición, tamaño del artículo y una visualización que considere el tamaño y fecha de las ultimas 20 ediciones, véase la **Figura 4.8**.

---

<sup>30</sup><http://reactivex.io/>

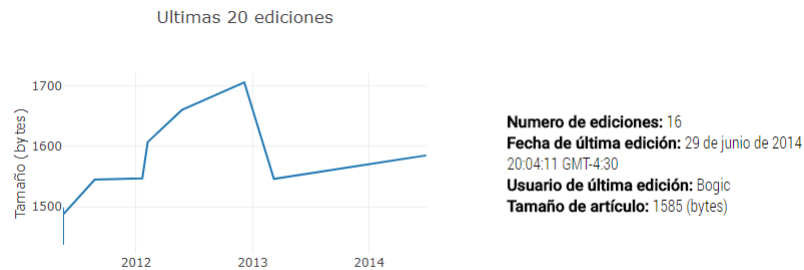


Figura 4.8: Visualización e información específica del artículo

De la siguiente manera quedaría el código para mostrar la gráfica, haciendo uso del componente de visualización :

```

1  <app-visualization
2    chartTitle="Ultimas 20 ediciones"
3    chartYTitle="Tamaño (bytes)"
4    chartType="scatter"
5    [chartX]="timestamps(revisions)"
6    [chartY]="sizes(revisions)"
7    fxFlex="60" fxLayout="column">
8  </app-visualization>

```

Las función **timestamps(revisions)** extrae la fecha de edición y la función **sizes(revisions)** extrae el tamaño de cada edición

## 21. Implementar gráfica WikiHistoryFlow

A nivel de visualizaciones, ofrecer la gráfica WikiHistoryFlow era un requerimiento principal. Con esta gráfica se puede identificar el comportamiento de un usuario, el estado del artículo en el tiempo y detectar patrones de vandalismo.

La visualización proviene principalmente de la herramienta de History Flow Visualization [14], que consta de barras laterales, donde cada barra representa una edición. La altura de la barra representa el tamaño, el color representa un usuario único y el ancho representa la distancia del texto del artículo de la versión anterior y la actual, véase la **Figura 4.9** y **Figura 4.10**.

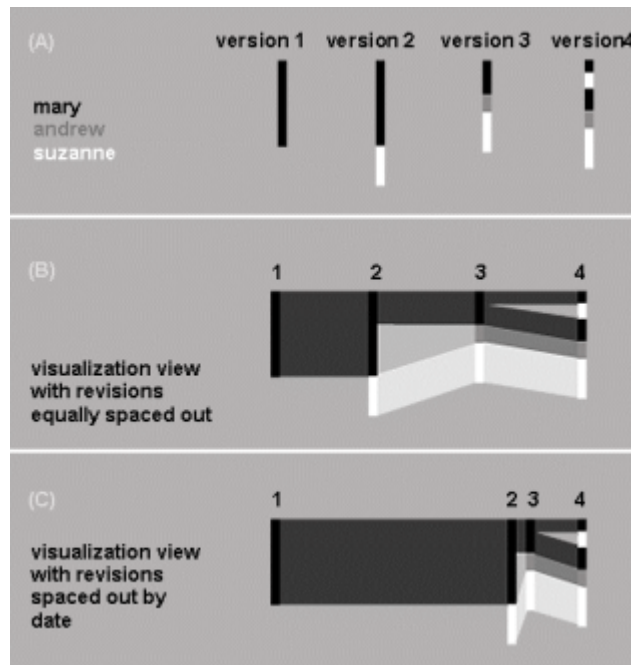


Figura 4.9: Explicación del mecanismo usado en la visualización de History Flow

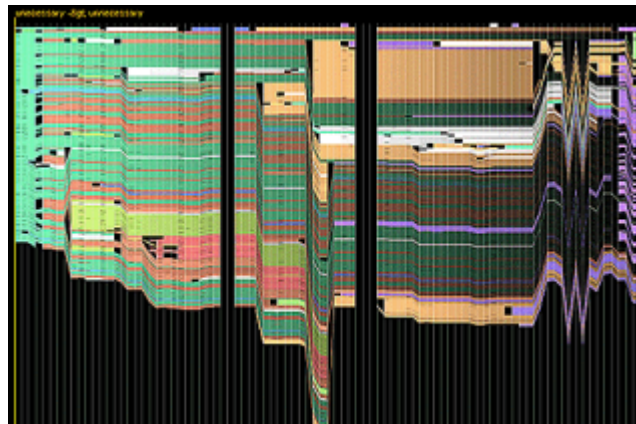


Figura 4.10: Herramienta History Flow Visualization

Para este trabajo por motivos de cómputos, se adaptara la visualización a una versión más simple, llamada History Graph [15], véase la **Figura**



#### 4.11.



Figura 4.11: Visualización History Graph

No hay manera de desplegar esta gráfica con la biblioteca Plotly dado que es bastante específica, por lo tanto para este caso se optó por la biblioteca D3 que nos permite libertad a la hora de construir visualizaciones.

Se encapsuló la visualización en un componente, en donde la primera tarea fue ordenar las ediciones por orden de creación de manera cronológica, luego se almacenó el valor de tamaño de edición máximo y se declaró un arreglo que contiene la distancia de cada edición contra la anterior (se asignó 0 para la primera edición). Para calcular la distancia entre dos textos se usó una biblioteca que aplica la distancia de Levenshtein.

Calculados los datos anteriores se tiene todo listo para graficar, en donde cada elemento del arreglo de revisiones ordenado cronológicamente representa una barra usando el elemento **div** de html.

```
1 d3.select(this.elemRef.nativeElement)
2   .selectAll('div.bar')
3   .data(revsOrderByDate)
4   .enter().append('div').attr('class', 'bar')
```

Para asignar el ancho, se usó del arreglo de distancias antes calculado. Adicionalmente, se tiene que conservar el tamaño de manera proporcional a la pantalla, por lo que se le asignó a cada barra un ancho base predeterminado. Para que no ocurrieran desbordamientos de píxeles se usó una función de CSS llamada **calc** para ajustar el ancho base más la distancia.

```
1 .style('width', (_, i) => `calc(
2   ${((1 / totalRev) * 100)}% + ${distanceRevs[i]}px`)
```

```

3     )`
4   )

```

Para asignar el alto basta con dividir el tamaño de la edición entre el tamaño máximo anteriormente calculado.

```

1   .style('height', rev => `${(rev.size / maxSize) * 100}%`)

```

Finalmente, para asignar el color de la barra se usó una función hash que recibe un nombre de usuario (string) y retorna un color hexadecimal.

```

1   .style('background', rev =>
2     `${this.stringToColour(rev.user)}`
3   )

1   stringToColour(str: string) {
2     let hash = 0;
3     for (let i = 0; i < str.length; i++) {
4       // tslint:disable-next-line:no-bitwise
5       hash = str.charCodeAt(i) + ((hash << 5) - hash);
6     }
7     let colour = '#';
8     for (let i = 0; i < 3; i++) {
9       // tslint:disable-next-line:no-bitwise
10      const value = (hash >> (i * 8)) & 0xFF;
11      colour += ('00' + value.toString(16)).substr(-2);
12    }
13    return colour;
14  }

```

Como funcionalidad adicional, se agregó un *tooltip* que muestra nombre del usuario, fecha y tamaño de la edición cuando nos apoyamos sobre cualquier barra, véase la **Figura 4.12**.

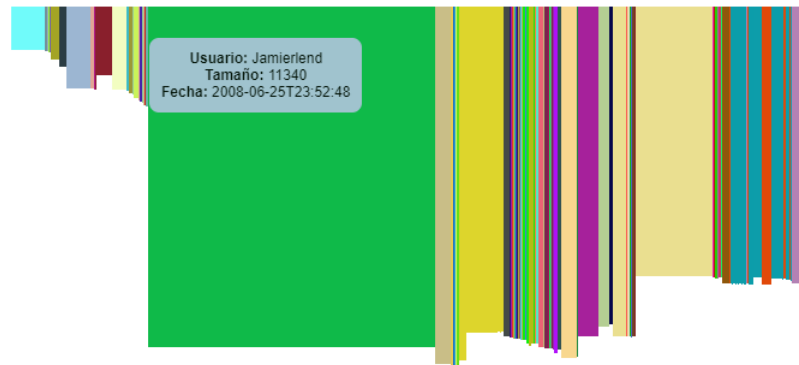


Figura 4.12: Visualización Wiki History Flow del artículo 'Programación dirigida por eventos'

## 22. Agregar enlace a wikipedia del artículo

En el detalle del artículo puede ser de gran utilidad tener un enlace que dirija al artículo en Wikipedia, véase la **Figura 4.13**.

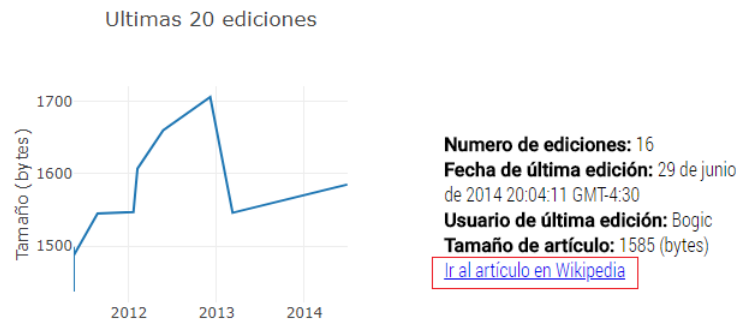
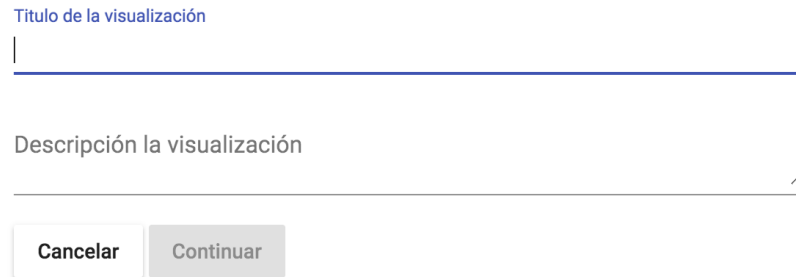


Figura 4.13: Información principal de detalle de artículo incluyendo vínculo a Wikipedia

## 23. Implementar componente y servicio para crear nueva visualización

Un requerimiento principal es poder crear nuestras visualizaciones sobre un artículo y poder editarlas, pero como primer paso necesitamos la creación de la misma preguntando información básica.

Se posicionó un botón en la barra de navegación capaz de crear la visualización. Al pisarse mostrará un dialogo preguntando el título y descripción que tendrá la visualización, véase la **Figura 4.14**.

El formulario tiene un campo de texto superior con el placeholder "Título de la visualización" y un campo inferior con el placeholder "Descripción la visualización". Debajo de los campos hay dos botones: "Cancelar" y "Continuar".

Título de la visualización

Descripción la visualización

Cancelar Continuar

Figura 4.14: Información principal para crear una visualización

Dado esto, se manejó en base de datos un objeto de visualizaciones donde internamente cada visualización es referenciada por el título:

```
1  {
2    "title": "Universidad Central de Venezuela",
3    "visualizations": {
4      "Visualización de usuarios": {
5        "query": "",
6        "type": "",
7        "description": "Torta"
8      },
9      "Visualización de ediciones menores": {
10       "query": "",
11       "type": "",
12       "description": ""
13     }
14   }
15 }
```

#### 24. Implementar componente y servicio para crear nueva visualización

Es necesario listar las visualizaciones creadas en el detalle de artículo, para luego ser accedidas.

Se crearon dos listas, una de las visualizaciones creadas por el usuario y otra de visualizaciones predefinidas por la aplicación.

## 25. Implementar componente para seleccionar el query de la visualización

Creada una visualización es necesario empezar a editarla para construir la gráfica respectiva, por lo que necesitamos ofrecer la herramienta para seleccionar lo que se desea visualizar.

La herramienta constará de tres (3) selectores, el primero permite filtrar, el segundo ofrece operaciones de agrupaciones y el tercero permite agrupar, véase la **Figura 4.15**.

El selector de **filtro** ofrece filtrar por edición anónima, tamaño de la edición, id de usuario, nombre de usuario, edición menor y fecha de edición.

El selector de **vista** ofrece operadores sobre agrupaciones como contar, sumar tamaño de ediciones, promediar tamaño de ediciones, máximo tamaño de ediciones y mínimo tamaño de ediciones.

El selector de **agrupar** ofrece la posibilidad de agrupar según edición anónima, id de edición, tamaño de edición, id de usuario, nombre de usuario, edición menor, fecha (mes), fecha (mes y año), fecha (día, mes y año).



Figura 4.15: Vista de componente para seleccionar query de la visualización

El valor de cada opción de los selectores referencia la parte del query de MongoDB necesario. Por lo tanto se realizó una función para construir el query final a enviar al API de Wikimetrics:

```
1 buildQuery(): WikimetricsQuery[] {  
2   // setup filters
```

```

3     let obj = {};
4     this.selectedFilters.forEach(i => {
5         obj = merge(obj, i.value);
6     });
7
8     // build query
9     const newQuery = [
10        {
11            $match: obj
12        },
13        {
14            $group: {
15                _id: this.selectedGroup ?
16                    this.selectedGroup.value : null ,
17                ... this.selectedView && this.selectedView.value ?
18                    { result: this.selectedView.value } : {}
19            }
20        }
21    ];
22
23     return newQuery;
24 }

```

En donde **obj** es el objeto con todos los filtros, **selectedGroup.value** el campo u objeto de la agrupación y **selectedView.value** el nombre del operador de agrupaciones.

Por último, si se tiene un query formado, se necesita la función inversa que sería extraer del query los valores para rellenar los selectores. Esta funcionalidad es extensa por lo que se puede revisar en el código fuente del proyecto en el componente **query-selector.component.ts**.

## 26. Implementar componente para editar visualizaciones

Cuando se obtiene la respuesta del query, se tiene los datos pero no listos para visualizarlos, tiene que darse un formato entendible para usarlo con el componente de visualización anteriormente creado, además se tiene que definir el tipo de visualización a mostrar.

Se tuvo que condicionar cada uno de los casos posibles de las opciones,

en el caso de edición menor, interpretar a 'No menor' y 'Menor', para usuario anónimo se interpretó como 'Anónimo' y 'No Anónimo'. En el caso de agrupación por fecha se usó la biblioteca luxon<sup>31</sup> (alternativa ligera de Momentjs) para darle un mejor formato, véase la **Figura 4.16**.

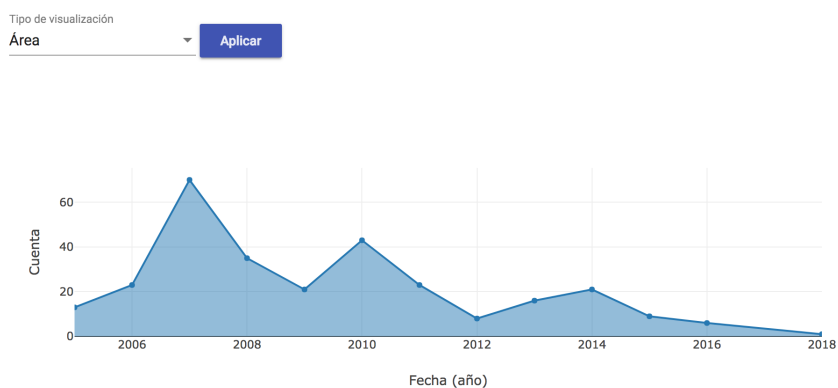


Figura 4.16: Componente de visualización y selector de tipo en el editor de visualizaciones

El selector de tipo visualizaciones ofrece las siguientes opciones:

- **Número:** muestra un número total, solo es válido cuando no se aplica agrupación, véase la **Figura 4.17**.

289

Figura 4.17: Visualización tipo número

- **Línea:** gráfica que consiste en trazar un línea entre cada par de puntos, véase la **Figura 4.18**.

<sup>31</sup><https://moment.github.io/luxon/>

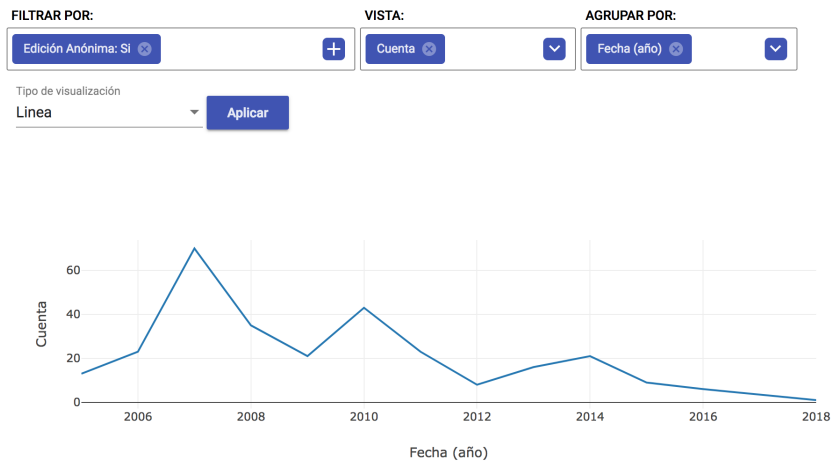


Figura 4.18: Visualización tipo línea

- **Barra:** gráfica que consiste en proyectar barras, véase la **Figura 4.19**.

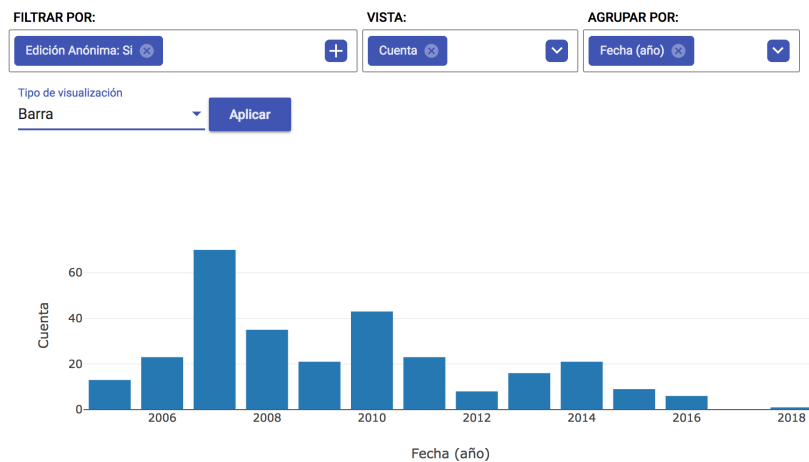


Figura 4.19: Visualización tipo barra

- **Área:** gráfica similar a la de línea pero con el área pintada. Se puede observar en la **Figura 4.16**.
- **Torta:** gráfica circular en donde los datos son representados por proporción, véase la **Figura 4.20**.



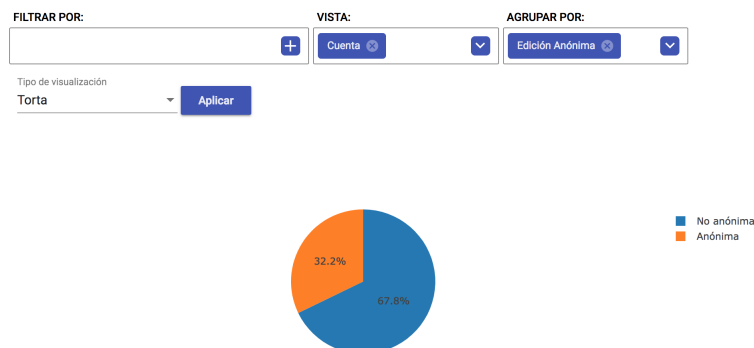


Figura 4.20: Visualización tipo torta

- **Dispersión:** gráfica donde cada dato es representado por un punto, véase la **Figura 4.21**.

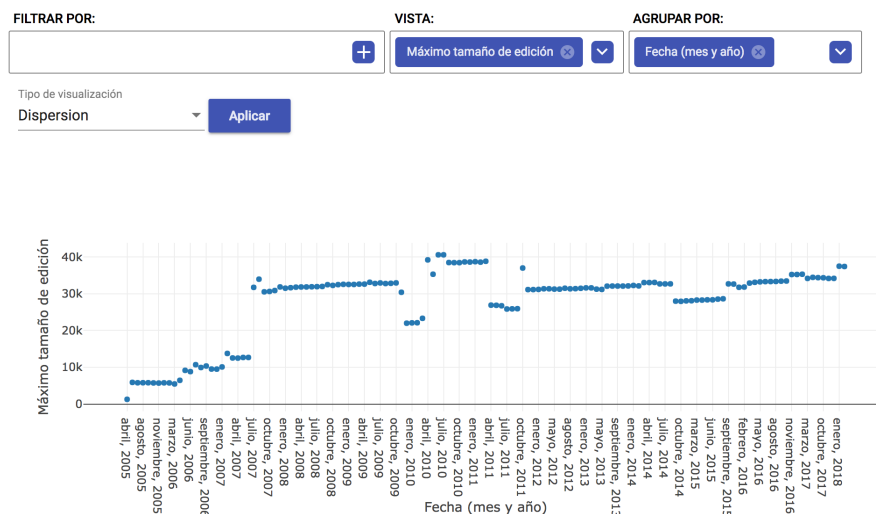


Figura 4.21: Visualización tipo dispersión

## 27. Implementar borrar artículo y visualización

Hasta ahora podemos crear artículos y visualizaciones, pero no borrar.

Se implementaron las rutas en el API usando el método DELETE de HTTP para ambos recursos, cada servicio, el de visualizaciones y artículos encapsulando la petición. Por último, se habilitó la acción de eliminar en la interfaz para ambos casos.

## 28. Agregar zoom a gráfica del editor de visualizaciones

En el editor de visualizaciones puede que una visualización esté tan cargada de datos que la información se resuma y no se muestre toda, por lo que no será posible ver todos los datos.

Plotly, automáticamente esconde la información para no sobrecargar la visualización, pero nos provee la funcionalidad de navegar en la visualización haciendo zoom.

```
1  const layout = {  
2    xaxis: { title: this.chartXTitle, fixedrange: !this.chartZoom},  
3    yaxis: { title: this.chartYTitle, fixedrange: true},  
4  };
```

La biblioteca permite pasarle un objeto con configuraciones al graficar. El atributo **xaxis** hace referencia al eje 'x' y el atributo **yaxis** al eje 'y'. Se habilitará el zoom en el eje 'x' solamente y se manejará con un input llamado **chartZoom**.

## 29. Agregar filtro de fecha en gráfica de Wiki History Flow

La gráfica Wiki History Flow suele ser muy pesada, en artículos con muchas ediciones puede tomarse un tiempo considerable en cargar debido a la petición de la información y el algoritmo para extraer la distancia del contenido.

Como mecanismo para evitar larga espera, se implementó un filtro por rango de fecha, en donde se cargará solo las ediciones realizadas en ese rango.

En la primera carga se definió un rango de fecha que involucren las 200 primeras ediciones del artículo, véase la **Figura 4.22**.



Figura 4.22: Wiki History Flow con filtro de rango de fecha.

### 30. Agregar componente de *preview* de visualización

Para poder observar una visualización es necesario entrar al detalle del artículo y seleccionar la que se quiere editar en el listado. Surge la necesidad de observar la gráfica de la visualización sin necesidad de acceder al editor.

Se implementó una propiedad llamada *preview*, donde cada elemento de la lista de visualizaciones proveerá la acción de habilitar o no el *preview* de dicha visualización, véase la **Figura 4.23**.

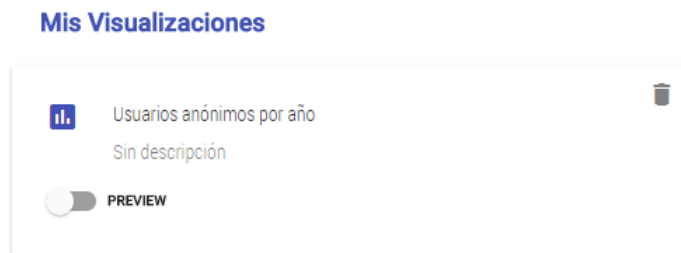


Figura 4.23: Toggle para habilitar/deshabilitar *preview*.

Al habilitar el *preview* la gráfica de la visualización será mostrada en una sección del detalle de actividad haciendo uso del componente de visualización, observe la **Figura 4.24**.



Figura 4.24: Componente *preview* de visualización en detalle de artículo.

El *preview* al ser pisado dirige al editor de la misma.

### 31. Incluir número de ediciones menores (con porcentaje) y número de editores en detalle de artículo

Se consideró agregar más información en el detalle del artículo, en donde el número de ediciones menores y número de editores pueden ser de utilidad.

Apoyándose de la flexibilidad del API, para obtener el número de ediciones del artículo se envía el siguiente query:

```

1  [
2    {
3      "$match": {
4        "title": "Programación dirigida por eventos",

```

```

5         "locale": "es",
6         "minor": {
7             "$exists": true
8         }
9     },
10 },
11 {
12     "$group": {
13         "_id": null,
14         "result": {
15             "$sum": 1
16         }
17     }
18 }
19 ]

```

Para obtener el número de editores se envía el siguiente query y se calcula el tamaño de la respuesta:

```

1     [
2     {
3         "$match": {
4             "title": "Programación dirigida por eventos",
5             "locale": "es"
6         }
7     },
8     {
9         "$group": {
10             "_id": "$userid",
11             "result": {
12                 "$sum": 1
13             }
14         }
15     }
16 ]

```

### 32. Definir e implementar gráficas generales en el detalle del artículo

Existen ciertas visualizaciones que seguramente sean de interés, por lo que se puede ofrecer de manera predeterminada y evitar que el usuario pierda tiempo en la creación de las mismas.

En la lista de visualizaciones predeterminadas, en el detalle del artículo, se agregaron las siguientes visualizaciones: ediciones menores (**Figura 4.25**), usuario anónimos (**Figura 4.26**), top 10 editores (**Figura 4.27**) y número de ediciones por mes y año (**Figura 4.28**).

La información de estas visualizaciones se encuentran de manera estática en la versión actual dentro del código.

Se implementó un componente capaz de visualizarlas en una vista nueva, similar al *preview* y este fue el resultado:

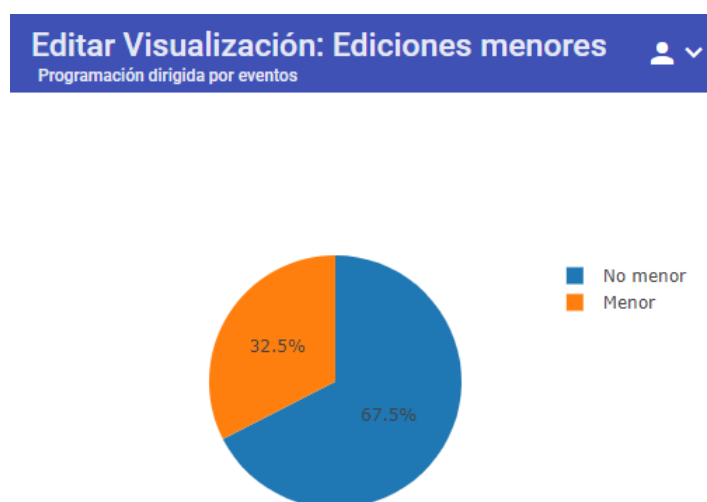


Figura 4.25: Visualización predefinida de total de ediciones menores vs. no menores del artículo 'Programación dirigida por eventos'

### Editar Visualización: Usuarios anónimos Programación dirigida por eventos

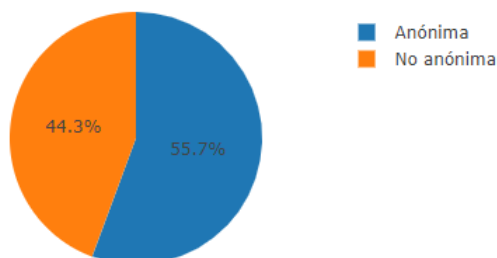


Figura 4.26: Visualización predefinida de total de usuarios anónimos vs. no anónimos del artículo 'Programación dirigida por eventos'

### Editar Visualización: Top 10 editores Programación dirigida por eventos

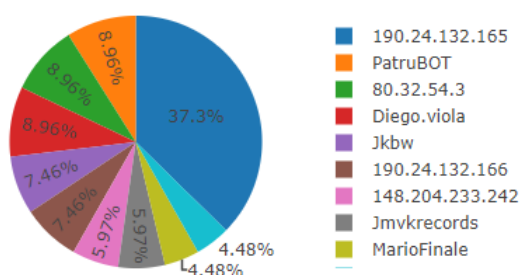


Figura 4.27: Visualización predefinida de Top 10 de usuarios con más ediciones del artículo 'Programación dirigida por eventos'



Figura 4.28: Visualización predefinida de total de ediciones agrupadas por mes y año del artículo 'Programación dirigida por eventos'

### 4.3. Arquitectura

Anteriormente se había mencionado que la aplicación web está construida con el framework Angular, que hace uso de distintos servicios web: API de Wikimetrics, API de Usuarios y API de Wikipedia. Todo este ecosistema es indispensable para que la aplicación funcione correctamente, véase la **Figura 4.29**.



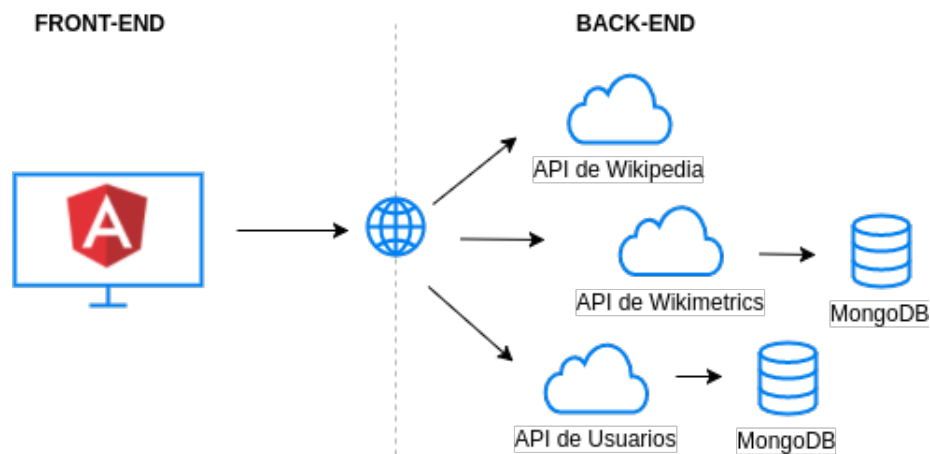


Figura 4.29: Arquitectura general

#### 4.3.1. Front-end

Angular está formado por una arquitectura basada en componentes, los componentes son piezas que conforman una vista. Algo importante en la arquitectura de Angular son los servicios, que son los encargados de proveer funcionalidades que no están relacionadas directamente con la vista. Para que los componentes puedan usar otros componentes y servicios es necesario contenerlos en un módulo. Los módulos permiten agrupar artefactos de Angular (incluyendo componentes y servicios) para que los componentes tengan el acceso a los mismos.

La aplicación desarrollada, tiene seis (6) componentes de ruta, que representan aquellos componentes que se *renderizan* acorde a una ruta especificada en el navegador, véase la **Figura 4.30**. Cabe destacar que Angular necesita de un componente raíz que sea el encargado de encapsular los demás componentes, llamado **app-component**.

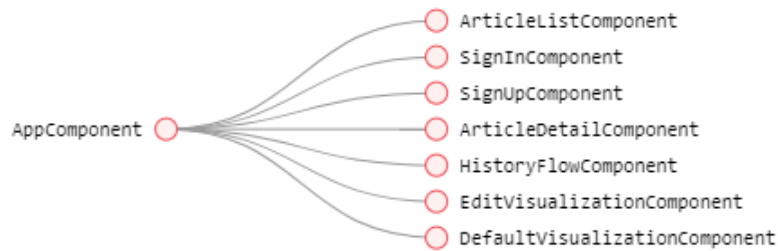


Figura 4.30: Componentes de ruta

Luego se tienen los otros componentes que son usados internamente en los componentes de ruta, en total tenemos los siguientes componentes:

```
1 article-card
2 article-detail (componente ruta)
3 article-list (componente ruta)
4 default-visualization (componente ruta)
5 edit-visualization (componente ruta)
6 history-flow (componente ruta)
7 loading
8 navbar
9 new-visualization
10 preview-visualization
11 query-selector
12 search-suggest
13 sign-in (componente ruta)
14 sign-up (componente ruta)
15 visualization
```

A parte de los componentes propios, se usaron algunos componentes de Angular Material mencionados en ciertas tareas de la sección anterior.

Los componentes requieren el apoyo de los siguientes servicios:

```
1 article.service
2 auth-guard.service
3 auth.service
4 navbar.service
5 resize.service
6 visualization.service
```

```
7 wikimetrics.service
8 wikipedia.service
```

Las dependencias del proyecto de angular se pueden encontrar en un archivo llamado **package.json** bajo el atributo *dependencies*. Entre las dependencias se puede resaltar: D3, Luxon, Lodash y Text-diff. Cabe acotar que Plotly es una dependencia asíncrona que se descarga cuando el **index.html** se ejecuta por lo que no viene pre-cargada en el código de la aplicación.

### 4.3.2. Back-end

Para persistir la información de los usuarios es necesario implementar un servicio que provea y almacene los datos. De esta forma se implementó un servicio web RESTful, el cual trabaja en la existencia de recursos. Este servicio se construyó apoyándose del micro-framework Python Flask, que cubre las necesidades básicas, como recibir peticiones y manejar respuestas bajo los distintos métodos (GET, POST, PATCH y DELETE).

En el caso de este proyecto, se tuvieron que usar algunas extensiones de Flask para habilitar CORS y manejar autorización de recursos en la sesión con JWT.

Se implementaron las siguientes rutas en el API:

```
1 POST /sign-up
2 POST /sign-in
3 GET /articles
4 GET /articles/<locale>/<title>
5 POST /articles
6 PATCH /articles/<locale>/<title>/status/<status>
7 DELETE /articles/<locale>/<title>
8 POST /articles/<locale>/<title>/visualizations
9 PATCH /articles/<locale>/<title>/visualizations
10 DELETE /articles/<locale>/<title>/visualizations/<title_vis>
```

Para persistir los datos se usó MongoDB, que es un manejador de sistema de base de datos no relacional. Al ser información única por usuario no era necesario relacionarla con la información de los otros usuarios, por lo que se

trató cada usuario y configuración como un objeto separado en un documento JSON. La **Figura 4.31** refleja el modelo de la base de datos.

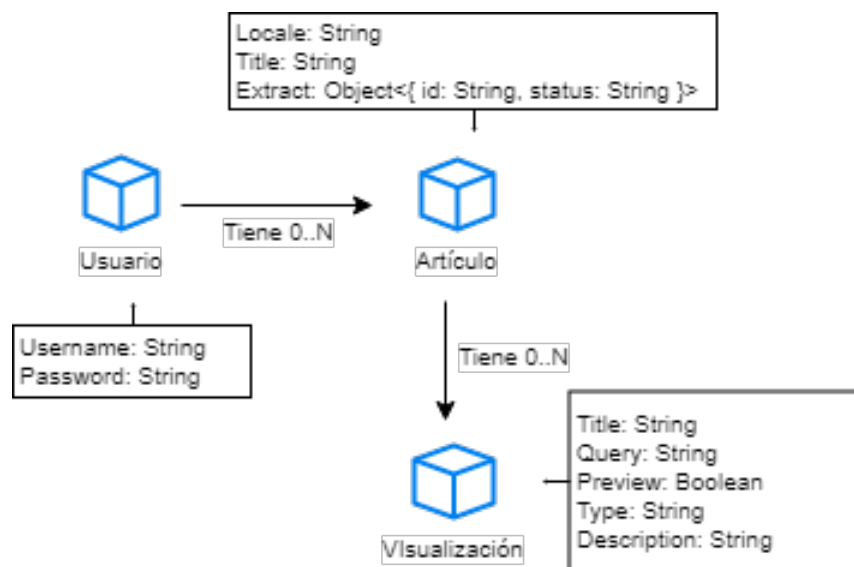


Figura 4.31: Modelo de base de datos

Un ejemplo de los datos crudos en MongoDB:

```

1 {
2   "user": {
3     "username": "user1",
4     "password": "106a6c241b8797f52e1e77317b96a201"
5   },
6   "articles": [
7     {
8       "locale": "en",
9       "extract": {
10        "status": "success",
11        "id": "665d387e-e547-4275-8abf-1076eacf8f92"
12      },
13       "title": "Article 1",
14       "visualizations": {
15         "vis1": {
16           "query": "...",

```

```

17         "preview": true,
18         "type": "number",
19         "description": ""
20     }
21 },
22 {
23     "locale": "en",
24     "extract": {
25         "status": "success",
26         "id": "e7f48aa8-39e0-46c4-a23a-39485b65e0b4"
27     },
28     "title": "Article 2"
29 }
30 ]
31 }
32

```

Se usó un driver llamado **PyMongo** para establecer la comunicación entre el código Python y la instancia de base de datos de MongoDB.

# Capítulo 5

## Conclusiones

En el presente trabajo luego de analizar y diseñar, se concluyó satisfactoriamente la implementación de una aplicación web capaz de construir y editar visualizaciones de historiales de wikis. Gracias a la flexibilidad de Angular, se logró cumplir una arquitectura SPA que con ayuda del *layout* de Angular Material se implementaron componentes visuales bastante adaptativos a distintas densidades de pantallas. Se diseñaron y ofrecieron visualizaciones generales para cada artículo extraído haciendo uso del API de Wikimetrics, entre varias la más relevante es la gráfica de Wiki History Flow. Es importante persistir el progreso y acciones de los usuarios, por lo tanto se diseñó e implementó un API de Usuarios capaz de proveer los recursos necesarios, apoyándose de una base de datos no relacional.

Angular a su vez dispone de una herramienta llamada Angular CLI que ofrece diversas funcionalidades para facilitar la construcción, reproducción y despliegue de la aplicación para distintos ambientes de trabajos (desarrollo o producción). Por lo tanto, se pudo servir en un ambiente de desarrollo exitosamente.

Se aplicó correctamente una metodología ágil, que fue llevada totalmente en la plataforma Github, en donde la pizarra (como artefacto de la metodología Kanban) jugó un papel importante, la cual se encargaba de proyectar el estado de las asignaciones o tareas. Además, los *issues* prestaban un espacio para abrir discusión y documentar toma de decisiones.

## 5.1. Limitaciones

- Principalmente se iba a usar el API de Wikipedia para realizar la autenticación de usuarios y extraer los artículos de su *watchlist* como primera instancia, pero Wikipedia dejó obsoleta la manera vieja de autenticación y la nueva forma es mediante OAuth2, por lo que pedían una serie de requerimientos que estaban fuera del alcance. Por tal motivo se optó por gestionar los usuarios con nuestro propio sistema.
- Para la realización de visualizaciones se tuvo que diseñar e implementar un nuevo end-point en el API de Wikimetrics debido a que los otros eran pocos flexibles. Por lo tanto se tomó tiempo del trabajo para el análisis de la misma. Con el nuevo end-point por falta de tiempo los queries en MongoDB los tuvo que definir el front-end en base a los requerimientos.

## 5.2. Trabajos futuros

- Dejar de manejar los usuarios internamente y manejar la autenticación con el API de Wikipedia. Lo principal sería cumplir con los requerimientos que pide MediaWiki como API para crear la aplicación usando OAuth2. De esta forma solo manejaremos un inicio de sesión usando el usuario de Wikipedia.
- Habilitar una opción para poder sincronizar los artículos del watchlist del usuario de Wikipedia con los artículos de esta plataforma.
- Analizar qué visualizaciones pueden ser frecuentes en los usuarios y ofrecerlas como visualizaciones predeterminadas.
- Implementar la gráfica original de History Flow. Esta visualización requiere de mucho cómputo, por lo que calcularlas en el front-end es inviable. La idea es delegarle esta funcionalidad a un servicio en el back-end que envíe los datos preparados para la visualización.
- Alguna funcionalidad para compartir *dashboard* de visualizaciones o visualizaciones individuales en modo de solo lectura

- Implementar un servicio en el backend que se encargue de actualizar el estado de los artículos extraídos en el API de Usuarios. De esta manera, se evitan constantes peticiones HTTP en la aplicación web (HTTP Polling).

### 5.3. Contribuciones

- Se realizó una investigación de bibliotecas de visualización que puede servir como apoyo para la decisión de futuros trabajos dependiendo de los requerimientos.
- Se realizó investigación en el área de Visualización de Datos, refrescando conceptos teóricos claves y planteamientos de como resolver problemas haciendo uso de diferentes técnicas.
- Se implementó un componente en Angular que facilita la creación de visualizaciones responsives usando Plotly, en donde será de apoyo para aquellos trabajos que involucren las mismas tecnologías.
- Se elaboró un componente de Angular que construye la gráfica Wiki History Flow haciendo uso de D3. Para trabajos futuros puede servir de apoyo para la mejora o construcción de la misma.
- Al ser una aplicación web que implica tecnologías nuevas e interesantes puede aportar contenido y ejemplos para la materia de Aplicaciones en Internet.



# Bibliografía

- [1] Eugene Barsky y Dean Giustini. “Introducing Web 2.0: wikis for health librarians”. En: (2007).
- [2] Noah Iliinsky y Julie Steele. *Designing Data Visualizations*. 2011.
- [3] Andy Kirk y col. *Data Visualization: Representing Information on Modern Web*. 2016.
- [4] Mozilla Developer Network. *Introduction to HTML*. URL: <https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Introduction>.
- [5] Mozilla Developer Network. *JavaScript*. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [6] Mozilla Developer Network. *Canvas API*. URL: [https://developer.mozilla.org/en-US/docs/Web/API/Canvas\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API).
- [7] SVG Working Group. *Scalable Vector Graphics (SVG)*. World Wide Web Consortium. 2010. URL: <http://www.w3.org/Graphics/SVG/>.
- [8] Mihai Sucan. *SVG or Canvas, choosing between the two*. Opera Software. 2010. URL: <https://dev.opera.com/articles/svg-or-canvas-choose/>.
- [9] Texcel Research Jonathan Robie. *What is the Document Object Model*. W3. URL: <https://www.w3.org/TR/WD-DOM/introduction.html>.
- [10] Mozilla Developer Network. *CSS*. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS>.
- [11] the free encyclopedia Wikipedia. *Single-page application*. URL: [https://en.wikipedia.org/wiki/Single-page\\_application](https://en.wikipedia.org/wiki/Single-page_application).

- [12] Jesse James Garrett. “Ajax: A New Approach to Web Applications”. En: (2005).
- [13] David J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Con pról.de Donald G Reinertsen. 2010.
- [14] Fernanda B Viégas, Martin Wattenberg y Kushal Dave. “Studying cooperation and conflict between authors with history flow visualizations”. En: (2004).
- [15] Eugenio Scalise, Nancy Zambrano y Jean-Marie Favre. “Visualización de Propiedades del Historial de los Artículos de un Manejador de Contenidos Basado en Wiki Aplicando Ingeniería Dirigida por Modelos”. En: (2008).