Proyecto II
Journal Search Platform (JSP)

Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Bases de Datos (IC 4302)
Segundo Semestre 2022

---

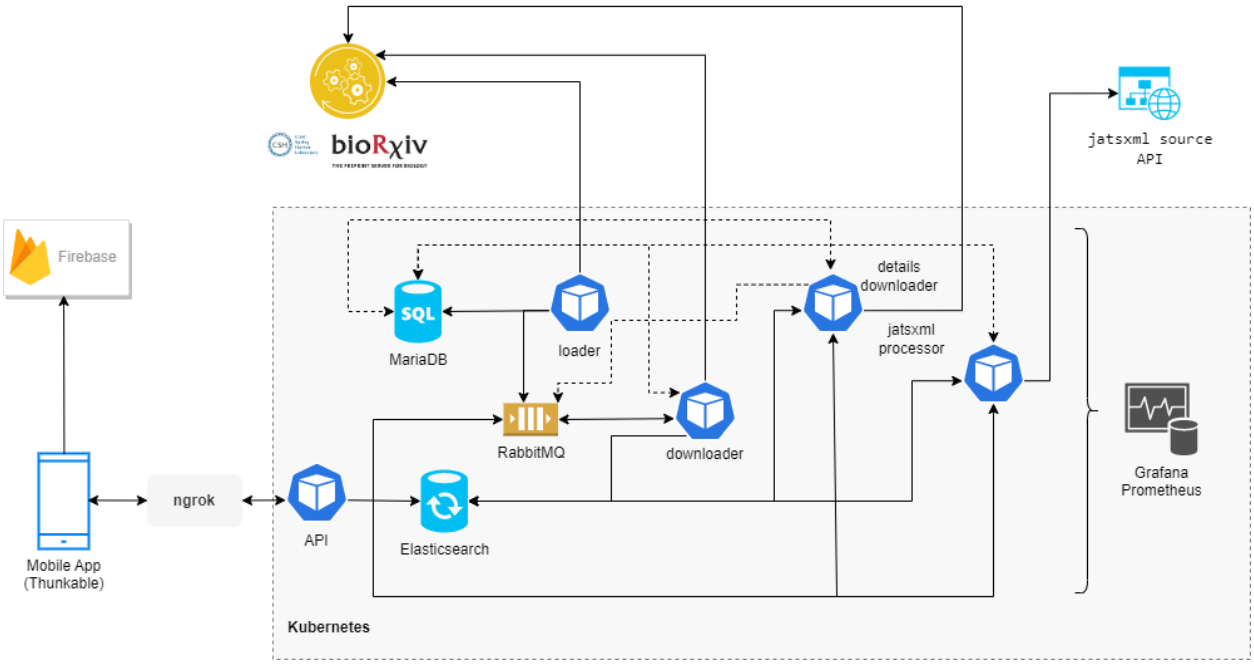
1. **Objetivo General**
   - ➔ Desarrollar un prototipo de plataforma para realizar búsquedas de artículos científicos mediante bases de datos NoSQL y SQL.
2. **Objetivos Específicos**
   - ➔ Implementar una aplicación que interactúe con bases de datos SQL, así como NoSQL.
   - ➔ Implementar un ingestion pipeline de un RestAPI.
   - ➔ Desarrollar habilidades en lenguaje de programación Python.
   - ➔ Instrumentar aplicaciones Python para ser monitoreadas con Prometheus.
   - ➔ Desarrollar aplicaciones con patrón productor consumidor mediante la utilización de servicios de mensajería.
   - ➔ Diseñar e implementar Dashboards para Grafana.
   - ➔ Automatizar una solución mediante el uso de Docker, Docker Compose, Kubernetes y Helm Charts.
   - ➔ Desarrollar arquitecturas de microservicios en Kubernetes.
   - ➔ Instalar y configurar servicios de bases de datos relacionales.
   - ➔ Instalar y configurar servicios de bases de datos NoSQL.
   - ➔ Instalar y configurar servicios de monitoreo con bases de datos de series de tiempo.
3. **Datos Generales**
   - ➔ El valor del proyecto: 30%
   - ➔ Nombre del proyecto: Journal Search Platform (JSP).
   - ➔ La tarea debe ser implementada en grupos de máximo 5 personas.
   - ➔ La **fecha de entrega**:
     - ◆ Documentación: 18 de noviembre del 2022 antes de las 9:53 am
     - ◆ Proyecto: 19 de noviembre del 2022, la hora de entrega es contra cita de revisión.
   - ➔ Cualquier indicio de copia será calificado con una nota de 0 y será procesado de acuerdo con el reglamento. La copia incluye código/configuraciones que se puede encontrar en Internet y que sea utilizado parcial o totalmente sin el debido reconocimiento al autor.
   - ➔ La revisión es realizada de forma virtual mediante citas en las cuáles todos los y las integrantes del grupo deben estar presentes, el proyecto debe estar completamente automatizado, el profesor decide en que computadora probar.
   - ➔ Se debe incluir documentación con instrucciones claras para ejecutar el proyecto.
   - ➔ Se deben incluir pruebas unitarias para verificar los componentes individuales de su proyecto, si estas no están presentes se obtendrá una nota de 0.

- ➔ Se espera que todos y todas las integrantes del grupo entiendan la implementación suministrada.
- ➔ Se deben seguir buenas prácticas de programación en el caso de que apliquen, entre ellas:
  - ◆ Documentación interna y externa.
  - ◆ Estándares de código.
  - ◆ Diagramas de arquitectura.
  - ◆ Diagramas de flujo
  - ◆ Pruebas unitarias.
- ➔ Toda documentación debe ser implementada en Markdown.
- ➔ Si el proyecto no se encuentra automatizado obtendrá una nota de 0. Si el proyecto no se entrega completo, la porción que sea entregada debe estar completamente automatizada, de lo contrario se obtendrá una nota de 0.
- ➔ En caso de no entregar documentación se obtendrá una nota de 0.
- ➔ El email de entrega debe contener una copia del proyecto en formato tar.gz y un enlace al repositorio donde se encuentra almacenado, debe seguir los lineamientos en el programa de curso.
- ➔ Los grupos de trabajo se deben autogestionar, la persona facilitadora del curso no es responsable si un miembro del equipo no realiza su trabajo.
- ➔ En consulta o en mensajes de correo electrónico o Telegram, como mínimo se debe haber leído del tema y haber tratado de entender sobre el mismo, las consultas deben ser concretas y se debe mostrar que se intentó resolver el problema en cuestión.
- ➔ Como parte de la evaluación, se revisarán los aportes de código en el repositorio compartido con el profesor, esto con el fin de determinar que el trabajo en equipo este siendo realizado equitativamente por parte de los integrantes del grupo, no basta con cantidad de commits por persona, se evaluara calidad de los aportes.

4. **Descripción**

Journal Search Platform (JSP) es un servicio que permite a investigadores realizar búsqueda de artículos científicos mediante el uso de una aplicación para telefonía móvil, para este prototipo nos enfocaremos en cerca de 25 mil artículos científicos relacionados con la Covid19 que se pueden encontrar en el sitio bioRXiv.

Desde un punto de vista técnico, se utilizarán las diferentes herramientas y bases de datos que se han utilizado a lo largo del curso esto permitirá el desarrollo de una aplicación con una arquitectura moderna y tecnologías estado del arte. El siguiente diagrama conceptualiza el funcionamiento de JPS.



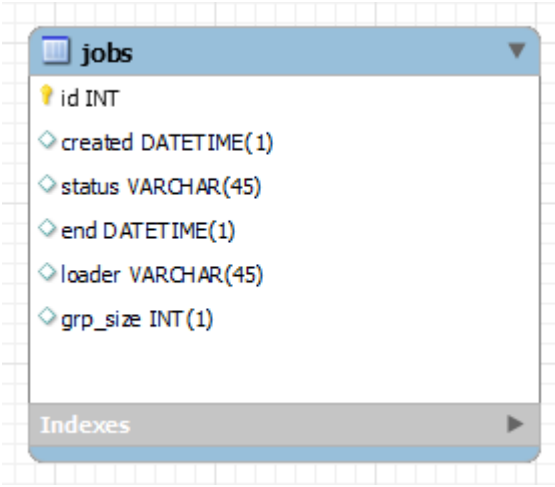
Dentro de Kubernetes se deberán implementar los siguientes componentes:

- Bases de datos y mensajería: Se deberán instalar los siguientes motores de bases de datos utilizando Helm Charts:
    - MariaDB
    - RabbitMQ
    - Elasticsearch

  Todos los servicios deberán ser monitoreados con Prometheus y Grafana, la configuración de los motores queda a discreción de los grupos de trabajo, pero deberán estar debidamente documentados en una sección de la documentación.
- Grafana y Prometheus: Se debe instalar Grafana y Prometheus como se ha realizado con otros proyectos y tareas, los mismos deber ser utilizados para monitorear todos los componentes en la solución que sean ejecutados dentro de Kubernetes.
    - Se pueden utilizar los Helm Charts que consideren adecuados, los de Bitnami permiten exponer para todos los motores ServiceMonitors.
    - Para los motores de bases de datos, se pueden utilizar Dashboards públicos.
    - Para las aplicaciones API, loader, downloader, details downloader y jatsxml processor, se deben generar métricas personalizadas y además se deben elaborar Dashboards personalizados.
    - Se deben recolectar las siguientes métricas personalizadas:
        - API:
            - Número de requests (cuantas veces se ha llamado el API)
            - Número de errores en esas llamadas.
            - Número de documentos retornados en las búsquedas.
        - Loader:

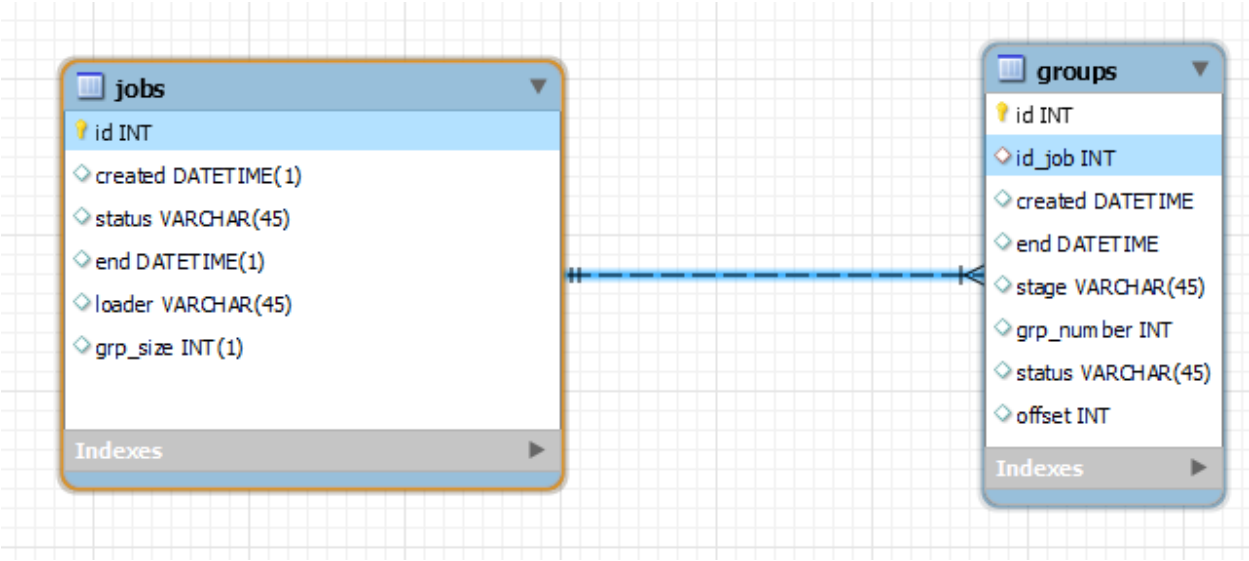
        - Cantidad de grupos que han sido procesados.
        - Cantidad de errores
        - Tiempo tardado en procesar cada grupo.
        - Jobs completados
    - Downloader:
        - Cantidad de documentos descargados.
        - Grupos procesados
        - Cantidad de errores.
        - Tiempo tardado en procesar cada grupo.
    - Details Downloader:
        - Cantidad de documentos a los cuales se les ha descargado detalles.
        - Cantidad de documentos que no se les ha descargado detalles (no existen).
        - Grupos procesados
        - Cantidad de errores.
        - Tiempo tardado en procesar cada grupo.
    - Jatsxml Processor:
        - Cantidad de documentos a los cuales se les ha procesado Jatsxml.
        - Cantidad de documentos que no se les ha procesado Jatsxml (no existen).
        - Grupos procesados
        - Cantidad de errores.
        - Tiempo tardado en procesar cada grupo.

- Loader: Este componente se conecta a una base de datos MariaDB, en la tabla ***jobs*** y mediante una transacción toma algún ***job*** que este pendiente de ejecución, una vez que obtiene este lo marca en progreso (columna ***status***) y agrega la información del loader que se encuentra procesándolo (columna ***loader*** = IP/Identificador del ***pod***), la tabla ***jobs*** tiene el siguiente formato:



Cuando el pod obtiene el ***job,*** deberá realizar una llamada al endpoint https://api.biorxiv.org/covid19/0, este retornara los siguientes valores (pueden cambiar):

Utilizando el campo ***grp_size*** de la base de datos, se generarán grupos de documentos a procesar, tomando como ejemplo un ***grp_size*** = 100, se deberían de generar 254 grupos (25335/100), estos serán almacenados en la tabla ***groups***:

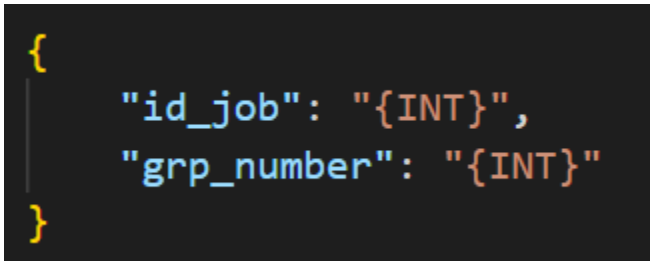


En términos de los valores que deberán almacenarse:

- id: auto generado
- id_job: llave foránea a la tabla jobs.
- created: momento en que los datos fueron almacenado.
- end: cuando el group está completamente procesado, siempre null.
- stage: Corresponde al nombre del componente que está procesando en este momento.

- grp_number: numero del grupo iniciando en 0.
- status: null para iniciar.
- offset: el desplazamiento en el API.

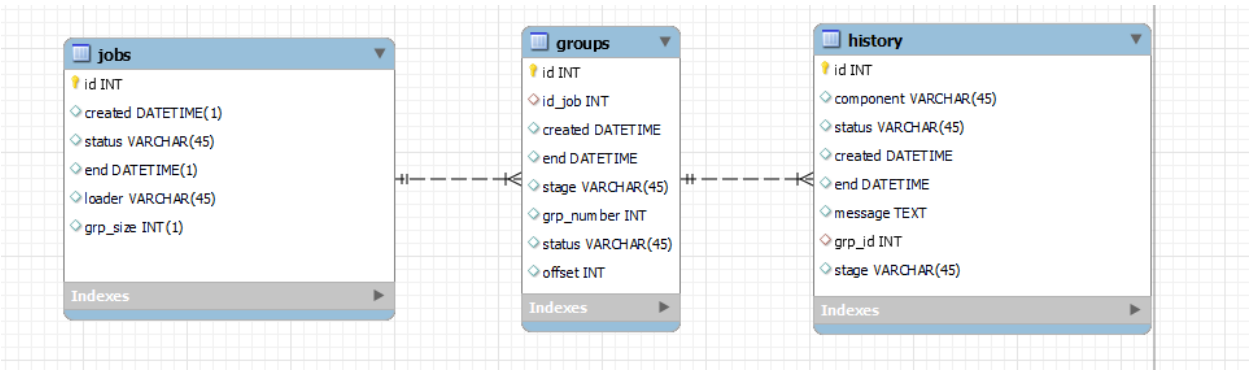
Una vez que se el grupo ha sido almacenado en MariaDB, se debe agregar al nombre de la cola de salía especificada en la variable de entorno configurada en el Deployment, un documento con el siguiente formato será:



Otros aspectos de este componente:

- Kubernetes Deployment, puede tener múltiples replicas.
- Los pods que forman parte del Deployment deben estar detrás de un servicio.
- Implementado en Python.
- Recibe las siguientes variables de entorno:
  - URL del API de bioRXiv
  - URL de RabbitMQ.
  - URL de MariaDB.
  - URL de Elasticsearch.
  - Password y usuario de RabbitMQ.
  - Password y usuario de MariaDB.
  - Password y usuario de Elasticsearch.
  - Nombre del índice de Elasticsearch.
  - Nombre de la base de datos de MariaDB.
  - Nombre de la cola de entrada.
  - Nombre de la cola de salida.

- Downloader: Este componente se activa cuando recibe mensajes en la cola de entrada configurada en las variables de entorno del Deployment, el mismo toma el mensaje y mediante los campos ***id_job*** y ***grp_number***, obtiene la información relevante de la base de datos, realiza las siguientes acciones:
  - Actualiza el grupo para que su ***stage*** sea igual a ***downloader*** y que ***status*** sea in-progress.
  - Agrega un record a la tabla de ***history***:



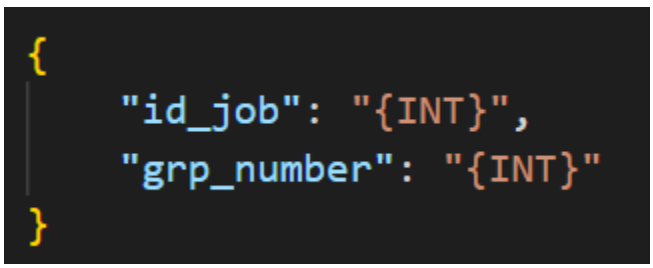
        - stage = dowloader
        - status= in-progress
        - created= tiempo actual
        - end: null
        - message: null
        - grp_id: llave foránea para referenciar el grupo fuente.
        - component: identificador del pod.
    - Descarga los documentos del grupo y los almacena en el índice groups en Elasticsearch (los grupos de trabajo deben definir y documentar bien la información que deben almacenar para que su proyecto funcione).
    - Actualiza el record en la tabla ***history***:
        - status: completed/error
        - end: tiempo actual
        - message: si se dio un error se guarda aquí
    - Actualizar el grupo para que su ***status*** sea ***completed***.
    - Se agrega un mensaje en la cola de salida especificada en la variable de entorno con el siguiente formato:

```
{
    "id_job": "{INT}",
    "grp_number": "{INT}"
}
```

Otros aspectos de este componente:

- Kubernetes Deployment, puede tener múltiples replicas.
- Los pods que forman parte del Deployment deben estar detrás de un servicio.
- Implementado en Python
- Recibe las siguientes variables de entorno:
    - URL del API de bioRXiv
    - URL de RabbitMQ.
    - URL de MariaDB.

        - URL de Elasticsearch.
        - Password y usuario de RabbitMQ.
        - Password y usuario de MariaDB.
        - Password y usuario de Elasticsearch.
        - Nombre del índice de Elasticsearch.
        - Nombre de la base de datos de MariaDB.
        - Nombre de la cola de entrada.
        - Nombre de la cola de salida.
- Details Downloader: Este componente se activa cuando recibe mensajes en la cola de entrada configurada en las variables de entorno del Deployment, el mismo toma el mensaje y obtiene el grupo de Elasticsearch y realiza las siguientes acciones:
    - Actualiza el grupo para que su ***stage*** sea igual a ***details-downloader*** y que ***status*** sea in-progress.
    - Agrega un record a la tabla de ***history***:
        - stage = details-dowloader
        - status= in-progress
        - created= tiempo actual
        - end: null
        - message: null
        - grp_id: llave foránea para referenciar el grupo fuente.
        - component: identificador del pod.
    - Para cada documento en el grupo obtenido de Elasticsearch, se deben descargar los detalles y agregarlos a cada documento (si existen) siguiendo los lineamientos en https://api.biorxiv.org/ (sección Content Details). Por ejemplo, si el grupo tiene el documento con ***rel_doi*** igual a "10.1101/2022.10.21.22281363" y este tiene ***rel_site*** igual a medRxiv, se deberían obtener los detalles de https://api.biorxiv.org/details/medrxiv/10.1101/2022.10.21.22281363.
    - Actualiza el record en la tabla ***history***:
        - status: completed/error
        - end: tiempo actual
        - message: si se dio un error se guarda aquí
    - Actualizar el grupo para que su ***status*** sea ***completed***.
    - Se agrega un mensaje en la cola de salida especificada en la variable de entorno con el siguiente formato:



Otros aspectos de este componente:

  - Kubernetes Deployment, puede tener múltiples replicas.
  - Los pods que forman parte del Deployment deben estar detrás de un servicio.
  - Implementado en Python
  - Recibe las siguientes variables de entorno:
    - URL del API de bioRXiv
    - URL de RabbitMQ.
    - URL de MariaDB.
    - URL de Elasticsearch.
    - Password y usuario de RabbitMQ.
    - Password y usuario de MariaDB.
    - Password y usuario de Elasticsearch.
    - Nombre del índice de Elasticsearch.
    - Nombre de la base de datos de MariaDB.
    - Nombre de la cola de entrada.
    - Nombre de la cola de salida.

- Jatsxml Processor: Este componente se activa cuando recibe mensajes en la cola de entrada configurada en las variables de entorno del Deployment, el mismo toma el mensaje y obtiene el grupo de Elasticsearch y realiza las siguientes acciones:
  - Actualiza el grupo para que su ***stage*** sea igual a ***jatsxml-processor*** y que ***status*** sea in-progress.
  - Agrega un record a la tabla de ***history***:
    - stage = jatsxml-processor
    - status= in-progress
    - created= tiempo actual
    - end: null
    - message: null
    - grp_id: llave foránea para referenciar el grupo fuente.
    - component: identificador del pod.
  - Para cada documento en el grupo obtenido de Elasticsearch, se debe verificar si el componente previo (details-downloader) encontró el campo ***jatsxml***, si ese es el caso, se debe seguir este link (es XML), se debe convertir a un JSON valido y agregarlo a cada documento (actualización).
  - Cada documento se debe guardar en el índice indicado en la variable de entorno en el Deployment, para este momento el documento se considera listo.
  - Actualiza el record en la tabla ***history***:
    - status: completed/error
    - end: tiempo actual
    - message: si se dio un error se guarda aquí
  - Actualizar el grupo para que su ***status*** sea ***completed***.
  - Remueve el grupo de Elasticsearch.

Otros aspectos de este componente:

- Kubernetes Deployment, puede tener múltiples replicas.
- Los pods que forman parte del Deployment deben estar detrás de un servicio.
- Implementado en Python
- Recibe las siguientes variables de entorno:
  - URL del API de bioRXiv
  - URL de RabbitMQ.
  - URL de MariaDB.
  - URL de Elasticsearch.
  - Password y usuario de RabbitMQ.
  - Password y usuario de MariaDB.
  - Password y usuario de Elasticsearch.
  - Nombre del índice de Elasticsearch.
  - Nombre de la base de datos de MariaDB.
  - Nombre de la cola de entrada.
  - Nombre de la cola de salida.

- API: Este componente implementa todos los endpoints requeridos por la aplicación (queda a discreción de cada grupo). Algunos aspectos importantes de este componente son:
  - Kubernetes Deployment, puede tener múltiples replicas.
  - Los pods que forman parte del Deployment deben estar detrás de un servicio, este se debe exponer mediante un NodePort a la maquina host.
  - Implementado en Python y Flask.
  - Recibe las siguientes variables de entorno:
    - URL de Elasticsearch.
    - Password y usuario de Elasticsearch.
    - Nombre del índice de Elasticsearch.

Fuera de Kubernetes se debe implementar una aplicación en Thunkable X que permita:

- Autenticar y autorizar usuarios (Firebase).
- Agregar un nuevo job a MariaDB (tabla jobs).
- Realizar una búsqueda de artículos científicos y mostrar los primeros 100 resultados (si no hay suficientes se muestran los enviados por Elastic), se recomienda el uso de Data Viewer List y se debe paginar con un máximo de 10 resultados por página.
- Si se le da clic a un resultado se deberá mostrar título, lista de autores y abstract.
- El usuario podrá darle like a un artículo, los likes se almacenan en Firebase.
- El usuario podrá ver los artículos a los cuales les ha dado like.
- Toda comunicación con el API se hace mediante ngrok

Documentación

Se deberá entregar una documentación que al menos debe incluir:

- Instrucciones claras de como ejecutar su proyecto.
- Pruebas realizadas, con pasos para reproducirlas.
- Pruebas unitarias y resultados de estas.
- Recomendaciones y conclusiones (al menos 10 de cada una).
- La documentación debe cubrir todos los componentes implementados o instalados/configurados, en caso de que algún componente no se encuentre implementado, no se podrá documentar y tendrá un impacto en la completitud de la documentación.
- Referencias bibliográficas donde aplique.

Imaginen que la documentación está siendo creada para una persona con conocimientos básicos en el área de computación y ustedes esperan que esta persona pueda ejecutar su proyecto y probarlo sin ningún problema, el uso de imágenes, diagramas, code snippets, videos, etc. son recursos que pueden ser útiles.
La documentación debe estar implementada en Markdown y compilada en un PDF.

5. **Recomendaciones**
   - Utilizar Docker Desktop para instalar Kubernetes en Windows.
   - Utilizar Minikube para ejecutar Kubernetes en Linux
   - Realicen peer-review/checkpoints semanales y no esperen hasta el último momento para integrar su aplicación.
   - Crear un repositorio de GitHub e invitar al profesor.
   - Realizar commits y push regulares.

6. **Entregables**

   - Documentación.
   - Docker files, Docker Compose files y Helm Charts junto con todos los archivos/scripts requeridos para ejecutar su proyecto.

7. **Evaluación**

Funcionalidad / Requerimiento	Porcentaje
Documentación (**)	20%
Implementación (*): ● loader (5%) ● downloader (10%) ● jatsxml processor (15%) ● API (10%)	80%

● details downloader (10%) ● Mobile App + Firebase (10%) ● Instalación RabbitMQ, MariaDB, Elasticsearch (10%) ● Monitoreo Grafana y Prometheus (10%)	
	100%

(*) Todo tiene que estar debidamente automatizado utilizando Helm Charts, de lo contrario se calificará con una nota de 0.
(**) Incluye pruebas unitarias.