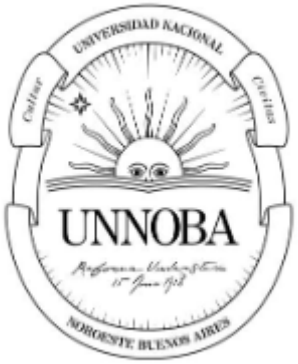
UNNOBA
Universidad Nacional
Noroeste • Buenos Aires

Anexo V

Informe Final de la PPS

claroLab



Universidad Nacional del Noroeste de la Provincia de Buenos Aires

Título: Imágenes V2

Carrera: Ingeniería en Informática

Práctica Profesional Supervisada

Luisina Santos

Tutor Docente: Federico Naso

Tutor de Empresa/Institución/Organización: Francisco Vives

Fecha de presentación: 24 de Abril de 2024

Indice

Presentación del Informe

Lineamientos Generales

1. Introducción

Clarolab es una empresa situada en la ciudad de Junín, provincia de Buenos Aires. Su actividad principal consiste en la terciarización de servicios informáticos, tales como desarrollo de software y testing de aplicaciones, a clientes cuyas actividades tienen un alcance internacional.

Khoros es uno de los actuales clientes de Clarolab , con oficina central en Austin, Texas. Actualmente tiene más de 1500 empleados y trabaja con más de 2000 marcas en todo el mundo, algunas de las cuales forman parte del “Fortune 500”. Se trata de una compañía conocida mundialmente por ofrecer un producto basado en la administración de comunidades en línea [Anexo A], marketing y análisis en redes sociales y administración de contenido para marcas y agencias empresariales. Cada comunidad tiene la particularidad de ser totalmente personalizable por quien contrate dicho producto.

Actualmente la empresa se encuentra en un proceso de actualización de su producto, llevado a cabo con el lanzamiento del Proyecto Aurora. Manteniendo características básicas existentes, y mejorando el producto a nivel performance, el proyecto en curso utiliza tecnologías en auge: GraphQL como lenguaje de consultas de API y React para el desarrollo de componentes reutilizables y completamente personalizables.

Entre las mejoras propuestas como parte del Proyecto Aurora, se encuentra la actualización en el sistema de almacenamiento y servicio de imágenes a nivel usuario de la comunidad.
Hasta el momento, para LIA Classic [Anexo A], el concepto de imágen es considerado un elemento dentro de un Álbum; cada usuario tiene la posibilidad de subir imágenes a su cuenta y categorizarlas en diferentes álbumes, para utilizarlas luego en cualquier aspecto de la comunidad -como avatar de usuario, como avatar de grupo, como parte del cuerpo de un post, o en un mensaje privado-.

La actualización busca eliminar el concepto de álbum, que permite el almacenamiento de imágenes no utilizadas en la comunidad, para de esta forma almacenar sólo aquellas imágenes asociadas a mensajes, usuarios o nodos [Anexo A].

Así aparece la necesidad de una nueva versión de imágenes, a la que llamaremos *ImagesV2*, que cambie conceptualmente el ciclo de vida de una imágen y además permita la migración de imágenes en su versión original, así toda comunidad preexistente que quiera utilizar Aurora en el futuro podrá mantener su contenido multimedia y éste será consistente con los requerimientos de dicha actualización.

2. Objetivos

El objetivo general de la presente Práctica Profesional Supervisada consiste en la adaptación del módulo existente encargado de la manipulación de Imágenes para que sea posible su utilización en una nueva versión del producto denominado Proyecto Aurora, cumpliendo con las necesidades del mismo.

El alcance de este trabajo se limita a exponer la utilización de la nueva versión de imágenes solo para mensajes, refactorizando para Aurora únicamente la lógica de negocio. En etapas posteriores, utilizando el trabajo realizado como parte de esta etapa inicial como patrón, se realizará la correspondiente adaptación para implementar *ImagesV2* en mensajes privados y avatares de nodos y usuarios.

Se plantean como objetivos específicos:

1. La interfaz de usuario debe ser capaz de subir una imágen al sistema a través de GraphQL y la misma debe ser tratada como es esperado para la nueva versión.
2. La interfaz de usuario debe ser capaz de utilizar la imágen previamente almacenada como parte del cuerpo de un mensaje a través de GraphQL y la API REST.
3. La interfaz de usuario debe ser capaz de visualizar la imágen subida a partir de una URL generada por la aplicación.
4. Una comunidad existente que quiera utilizar Aurora, debe tener la capacidad de migrar todas las imágenes almacenadas en el sistema de su versión anterior a la nueva versión.

3. Plan de Trabajo y Carga Horaria

3. 1. Metodología de trabajo utilizada

El siguiente plan de trabajo se enmarca en el desarrollo incremental propuesto por Scrum, una metodología de trabajo ágil que permite y facilita el trabajo colaborativo entre equipos al mismo tiempo que se entregan productos o prototipos.

El cumplimiento de los objetivos se llevará a cabo siguiendo con todos los "eventos" definidos por dicha metodología.

- Sprint: En Scrum un proyecto se ejecuta en ciclos de tiempo de duración fija. En este caso, dos semanas. Se busca que cada iteración o Sprint proporcione un resultado completo, un incremento del producto final que pueda ser entregado al cliente.[1]
- Sprint Planning: El listado de requisitos ordenados por prioridad es presentado al equipo de trabajo y este tiene la posibilidad de evacuar sus dudas.

  Luego, el equipo elabora la lista de tareas necesarias para alcanzar el cumplimiento de dichos requisitos y se realiza una estimación del esfuerzo que implica cada una de estas.
- Daily Scrum: Cada día durante la iteración o Sprint tiene lugar una reunión de estado del proyecto, conocida como Daily Standup, con una duración fija de entre 5 y 15 minutos [2]. El objetivo es que todos los miembros del equipo brinden una actualización de estado enfocándose en tres puntos clave:
  - En qué trabajó cada uno desde el último Standup
  - Qué problemas encontraron o prevén encontrar
  - Qué planes tienen hasta el siguiente Standup

  Estas reuniones mejoran la comunicación, eliminan la necesidad de otras reuniones y ayudan a identificar y eliminar los impedimentos para el desarrollo. Promueve la toma de decisiones rápida y mejora el nivel de transparencia y conocimiento del equipo de desarrollo.
- Sprint Review: El equipo presenta los avances alcanzados durante la última iteración. En función de estos resultados, el cliente tiene la posibilidad de volver a planificar el proyecto, o lo que espera de la siguiente iteración.
- Sprint Retrospective: Esta fase permite al equipo alcanzar un estado de mejora continua, ya que el mismo tiene la posibilidad de evaluar cómo ha sido su manera de trabajar y qué problemas se presentaron que podrían impedir el progreso esperado.

3.2. Indice de tareas

1. Subir una imágen al sistema a través de GraphQL
    a. Diseño de Base de Datos
    b. Creación del módulo de persistencia
    c. Versionado de API
    d. Implementación de la nueva funcionalidad
    e. Exposición de *mutation* en GraphQL
    f. Revisión y mejora de código
    g. Testing
2. Visualizar la imágen subida utilizando una URL generada por el sistema
    a. Diseño de URL
    b. Implementación de un Servidor Web
    c. Revisión y mejora de código
    d. Testing
3. Migrar imágenes a la nueva versión desarrollada
    a. Análisis de la estructura de Base de Datos utilizada en la versión 1
    b. Implementación de un servicio de migración de imágenes
    c. Revisión y mejora de código
    d. Testing
4. Eliminar imágenes no utilizadas
    a. Implementación de un servicio de limpieza
    b. Revisión y mejora de código
    c. Testing

		Semana 1					Semana 2					Semana 3					Semana 4					Semana 5				
Tarea	Duración	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V	L	M	M	J	V
1	8 días																									
1. a	1 día																									
1. b	1 día																									
1. c	4 días																									
1. d	1 día																									
1. e	2 días																									
1. f	3 días																									
1. g	2 días																									
2	5 días																									
2. a	1 día																									
2. b	3 días																									
2. c	2 días																									
2. d	1 día																									
3	7 días																									
3. a	1 día																									
3. b	5 días																									
3. c	2 días																									
3. d	1 día																									
4	4 días																									
4. a	3 días																									
4. b	2 días																									
4. c	2 días																									

4. Descripción de la Práctica Profesional Efectuada

4. 1. Contexto

Sabemos que el Proyecto Aurora cuenta con algunas actualizaciones a nivel conceptual y lógico y que tanto la versión original, utilizada por LIA Classic [Anexo A] y lanzada mensualmente para proveer actualizaciones a los actuales clientes, como la nueva versión de imágenes que se utilizará para el proyecto aún no lanzado, comparten a nivel código y que deben coexistir sin afectar la una a la otra en aquellos aspectos que se espera sean actualizados en la segunda versión. Para hacerlo, en base a configuraciones del sistema el comportamiento se separa en caso de ser necesario.

En la actualidad, cada usuario tiene la posibilidad de crear múltiples álbumes e insertar en ellos múltiples imágenes. Las mismas pueden ser utilizadas como avatar de usuario, avatar de nodos creados por el usuario, o insertadas en el cuerpo de un post, así como también ser “olvidadas” en su álbum.

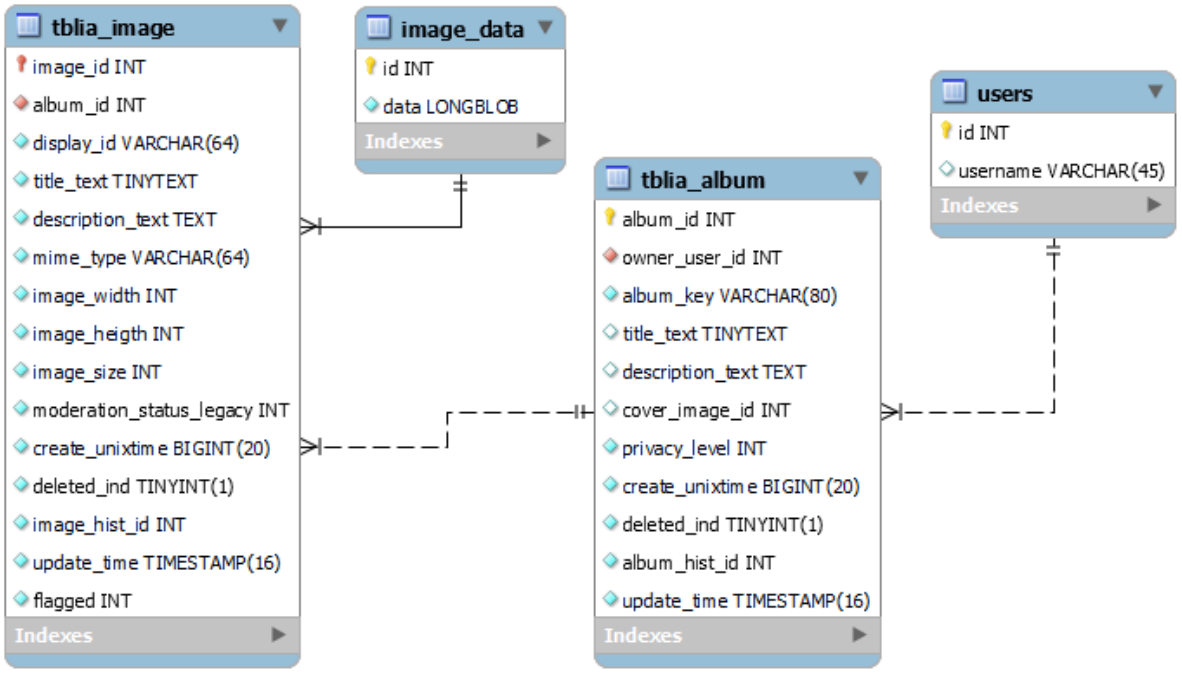


Figura 1. Diseño de tablas de base de datos preexistente

Con el fin de evitar que la comunidad se transforme en un almacen de imágenes para los usuarios, para el Proyecto Aurora se busca eliminar por completo el concepto de álbum, y sólo almacenar aquellas imágenes que están “asociadas” a otro componente de la comunidad, es decir, siendo utilizadas como contenido público y no sólo a nivel usuario.

4. 2. Subir una imagen al sistema a través de GraphQL

4.2.1 Qué es GraphQL?

GraphQL es un lenguaje de consulta y un entorno de ejecución para APIs, originalmente desarrollado por Facebook, que proporciona una forma más eficiente y flexible de interactuar con servicios web y obtener datos [3][4].
El entorno de ejecución es el contexto en el cual se ejecuta y procesa un programa o código. En el caso de GraphQL, el entorno de ejecución se encarga de tomar las consultas enviadas por los clientes y llevar a cabo la lógica necesaria para recuperar y proporcionar los datos solicitados.
Cuando un cliente envía una consulta GraphQL al servidor, el entorno de ejecución analiza la consulta, verifica su validez y luego ejecuta las operaciones para recuperar la información solicitada.

Algunos aspectos clave que hacen que GraphQL sea el lenguaje de consulta elegido para exponer la API utilizada para el Proyecto Aurora son:

- Consulta precisa: En lugar de recibir datos fijos de una API, la interfaz de usuario puede especificar exactamente los datos que el usuario necesita. Esto evita la sobrecarga de información y mejora el rendimiento al reducir el exceso de datos transferidos.
- Flexibilidad: La interfaz tiene el control total sobre la información que recibe. Es posible anidar consultas, solicitar datos específicos de múltiples recursos en una sola solicitud y recibir respuestas estructuradas según sea necesario.
- Versionamiento simplificado: GraphQL simplifica el versionamiento de la API al permitir que se soliciten solo los campos y operaciones específicas que necesitan. Esto facilita la introducción de cambios sin afectar a los clientes que no requieren esas modificaciones.
- Adaptabilidad: Puede ser implementado en diversas tecnologías y lenguajes de programación. No está vinculado a una base de datos o motor específico, lo que brinda flexibilidad en la construcción y evolución de servicios.

- Documentación autogenerada: La interfaz de Graphql, que incluye esquemas y tipos, tiene la capacidad de autogenerar documentación a partir de los mismos, lo que garantiza que siempre esté actualizada y sincronizada con los cambios en la API.[5][6]

En resumen, GraphQL permite redefinir la forma en que la nueva interfaz de usuario interactúa con las API al proporcionar un enfoque más personalizado y eficiente para la transferencia de datos.

4.2.2 Cómo se define una imágen en términos de *ImagesV2*?

Para almacenar en el sistema una imagen subida a través de graphQL y luego de evaluar los requisitos, se realizó el siguiente diagrama de base de datos que contiene toda la información necesaria para renderizar dicha imagen utilizando un servidor de imagenes posteriormente desarrollado:

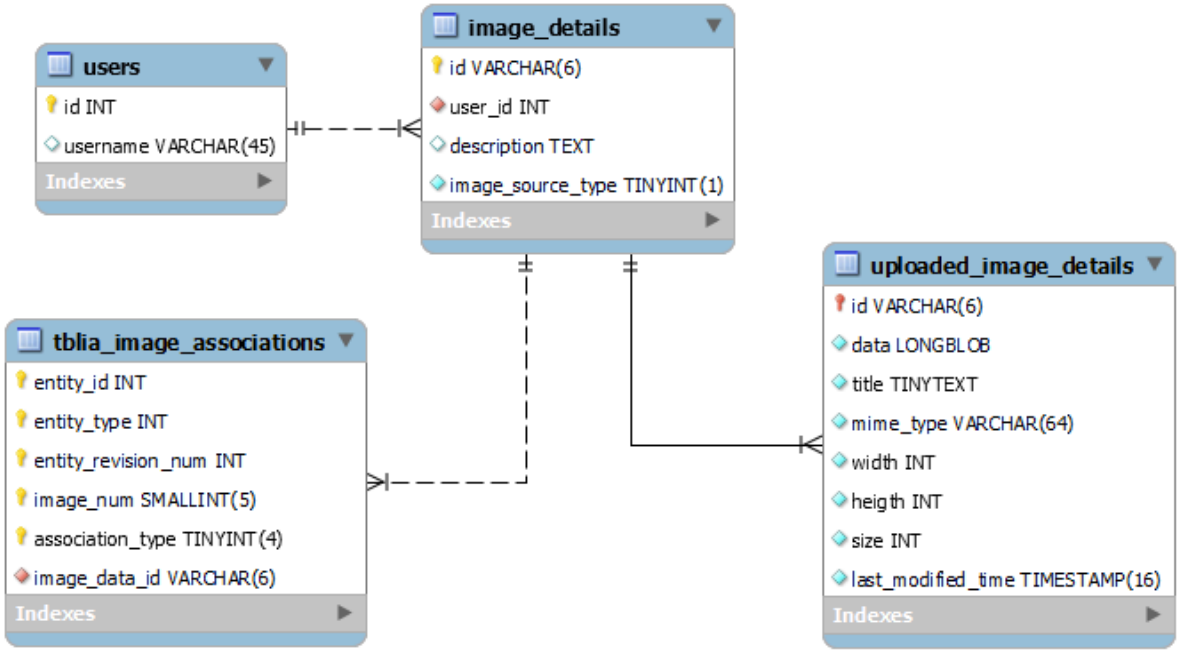


Figura 2. Diseño de tablas de base de datos para la versión 2

Nota: La tabla de usuarios no es relevante para la implementación por lo que no se reflejan en ella la totalidad de las columnas que el sistema almacena para cada entrada.

Con el nuevo modelo de base de datos, la misma imagen puede ser utilizada en múltiples asociaciones y desaparece el concepto de álbum sin perder los datos valiosos que se estaban almacenando previamente para cada imágen.

Una vez que dichas tablas fueron añadidas en la base de datos actual y dado que este comportamiento para imágenes no es el esperado para las versiones del producto que no utilizan Aurora como interfaz de usuario, es necesario hacer una separación de código bajo una configuración con el fin de no afectar las características esperadas para imagenes version 1, tal como se explicó en la sección 4.1.

4.2.3 Como está definido el modelo expuesto por la API?

Se diseña luego un modelo de esquema para GraphQL con el fin de recibir todos los parámetros necesarios para el almacenamiento de la imagen y devolver una respuesta valiosa para el usuario, que incluya por ejemplo la identificación de imagen asignada para esa carga (id). Esto facilita la inserción de la misma en el cuerpo de un post, y de esta manera no necesita realizar otra consulta para conocer en su totalidad la información acerca de la imagen que acaba de subir.

Por un lado, la *mutation* para crear lo que llamaremos “uploaded image” (una imágen temporal en el sistema que no está asociada a otra entidad), recibe como input (entrada) la descripción de la imagen, el tipo de entidad a la que se pretende asociar y la definición de la imagen, ya sea como archivo, url, o utilizando datos de Unsplash[1].

```
extend type Mutation {
 "Sube una imagen utilizando la carga de archivos."
 createUploadedImage(
   "Crea o actualiza la imagen subida."
   createInput: CreateUploadedImageInput!
 ): CreateUploadedImageResponse!
}
```

[1] Unsplash es una plataforma de búsqueda de fotografías de uso gratuito y alta definición. [7]

```
input CreateUploadedImageInput {
 "Descripción de la imagen a subir."
 description: String

 "Imagen para subir. Utilice file, imageUrl o unsplashData."
 file: Upload

 "Imagen para subir. Utilice file, imageUrl o unsplashData."
 imageUrl: Url

 "Datos de unsplash consistentes en la url de unsplash y el ID de la imagen
a subir. Utilice file, imageUrl o unsplashData."
 unsplashData: UnsplashInput

 "El tipo de entidad al que se asociará la imagen."
 associationEntityType: UploadedImageAssociatedEntityType!
}

input UnsplashInput {
 "La url de la imagen de unsplash a subir."
 unsplashUrl: String!

 "El ID de la imagen de unsplash a subir."
 unsplashId: String!

 "Url de descarga de Unsplash para la imagen subida usando unsplashUrl."
 unsplashDownloadBeaconUrl: String!
}

enum UploadedImageAssociatedEntityType {
 MESSAGE
 PRIVATE_NOTE
}
scalar Url
scalar Upload
```

Cuadro 1. Definición de mutation en GraphQL para subir una imágen

Como respuesta a la *mutation* recibiremos un resultado y una lista de errores tipados, dado que este es el patrón esperado por el esquema de GraphQL definido para el proyecto. El resultado contiene data referente a la imágen que acabamos de subir y cada error en la lista un mensaje y un campo que indica a qué parámetro del input corresponde el mismo en el caso de corresponder a alguno de ellos.

```
"Respuesta a la mutation de imagen createUploaded".
type CreateUploadedImageResponse {
 "Imagen subida."
 result: UploadedImage
 errors: [CreateUploadedImageError!]
}

interface Image {
 "El título de la imagen."
 title: String

 "La descripción de la imagen."
 description: String

 "La referencia completa a la versión original de la imagen."
 url: String!

 "La anchura de la imagen."
 width: Int!

 "La altura de la imagen."
 height: Int!

 "El tipo MIME de la imagen."
 mimeType: String!
}

type UploadedImage implements Image {
 "El ID que define de forma única esta imagen en la comunidad."
```

```
 id: String!

 "El título de la imagen."
 title: String

 "La descripción de la imagen."
 description: String

 "La referencia completa a la versión original de la imagen."
 url: String!

 "La anchura de la imagen."
 width: Int!

 "La altura de la imagen."
 height: Int!

 "El tipo MIME de la imagen."
 mimeType: String!
}

"Unión de Errores que pueden ser devueltos por la mutation createUpload."
union CreateUploadedImageError =
 UserNotRegistered
 | MalformedUrl
 | UploadLimitReached
 | TooManyUploadedImages
 | DisallowedImageFormat
 | UploadTooLarge
 | UnsplashDataNotProvided
 | NoFileUploaded
 | UploadFailed
 | InvalidUriError
 | NotValidPossibleValueError
```

Cuadro 2. Tipos definidos como respuesta de la mutation en GraphQL para subir una imágen

4.2.4 Cuál es el resultado de subir una imagen al sistema con la nueva versión?

Cualquier usuario que utilice esta *mutation* para subir al sistema una imágen, obtendrá un id de imágen autogenerado que podrá utilizar luego como referencia en el cuerpo de un mensaje, por ejemplo, ya sea a través de GraphQL o utilizando la API Rest, dado que ambas están integradas una con la otra.


```
{
   "status": "success",
   "message": "",
   "http_code": 200,
   "data": {
       "type": "uploaded_image",
       "id": "upbZCW",
       "title": "file_example_PNG_500kB.png",
       "description": "this image description",
       "url": "http://localhost:8080/lia/uploaded_images/MS11cGJaQ1c"
   },
   "metadata": {}
}
```


Cuadro 3. Respuesta obtenida luego de subir una imagen al sistema


```
{
  "data": {
    "type": "message",
    "subject": "root topic with image",
    "body": "This is Xbox 360 test and there is not much other than <li-image id=\"upbZCW\" size=\"medium\" />",
    "board": {
      "id": "forum"
    }
  }
}
```


Cuadro 4. Definición de JSON para asociar la imagen subida al cuerpo de un mensaje

4.3. Visualizar la imágen subida utilizando una URL generada por el sistema

Aquellas imágenes subidas al sistema que aún no están asociadas a otra entidad, siguen un patrón de url definido como: */uploaded_images/<encodeBase64><userId/>-<imageId/></encodeBase64>.*
Estas solo serán visibles para el usuario que las haya subido al sistema, por lo que es necesario incluir del lado del servidor el usuario que está realizando la petición para determinar si es el propietario de la imagen o no, es decir, si tiene acceso a ella o no.

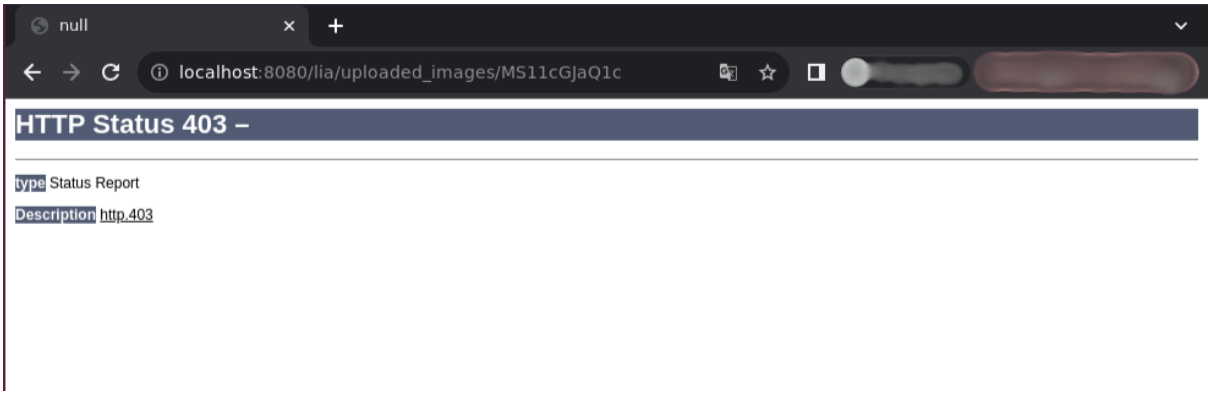


Figura 3. Respuesta del sistema para un usuario sin permisos de acceso

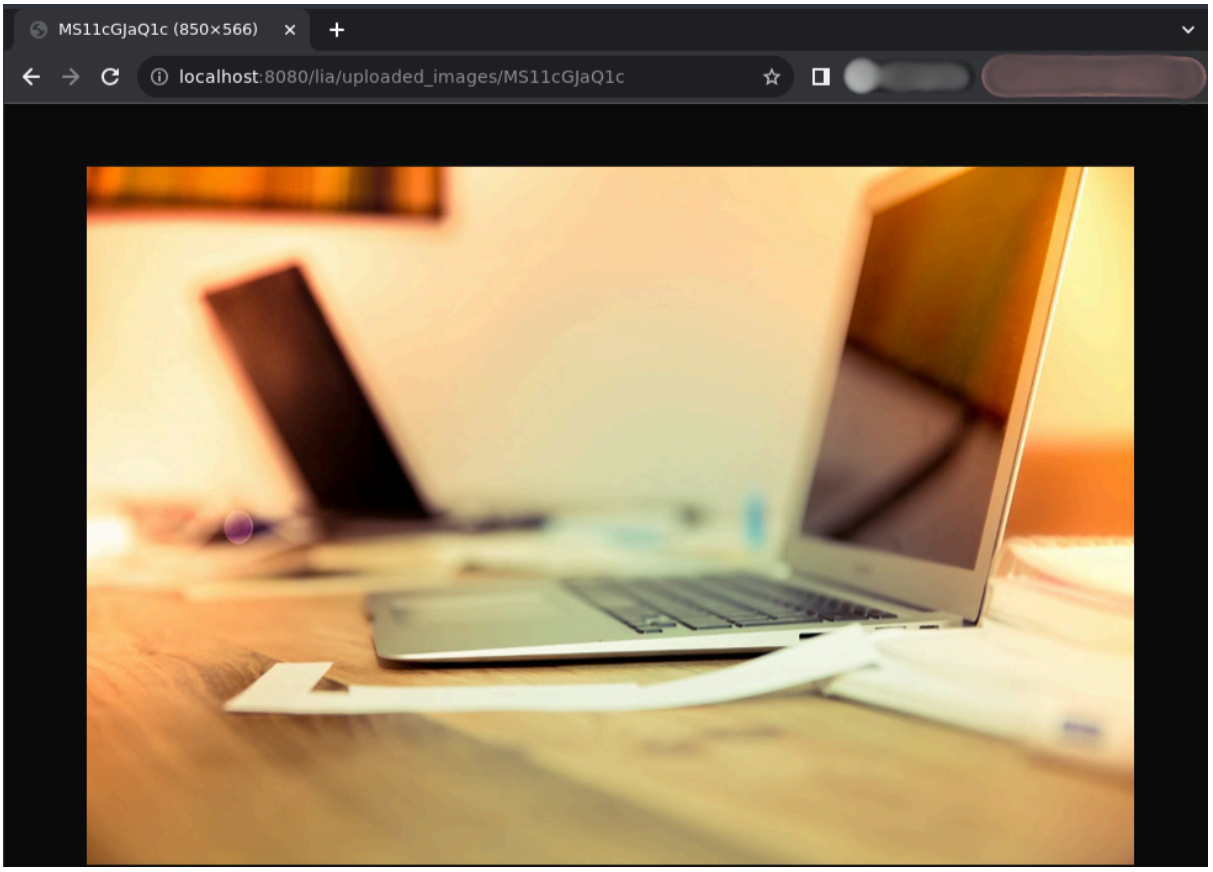


Figura 4. Respuesta del sistema para el usuario que subió la imágen

A su vez, todas las imágenes subidas al sistema que están siendo utilizadas como imagen asociada a una entidad siguen el mismo patrón de URL definido como:
/images/<encodeBase64><entityType/>-<entityId/>-<imageId/></encodeBase64>.
Tanto las imágenes pre-existentes como aquellas que se suban utilizando imágenes en su versión 2 serán accesibles mediante dicha URL.
Dado que no todas las imágenes son públicas, ya que algunas de ellas son utilizadas en mensajes privados o nodos con acceso restringido a ciertos usuarios, es necesario obtener los datos de sesión del usuario que está realizando el Request al servidor para determinar si el mismo tiene acceso o no a esta imagen.

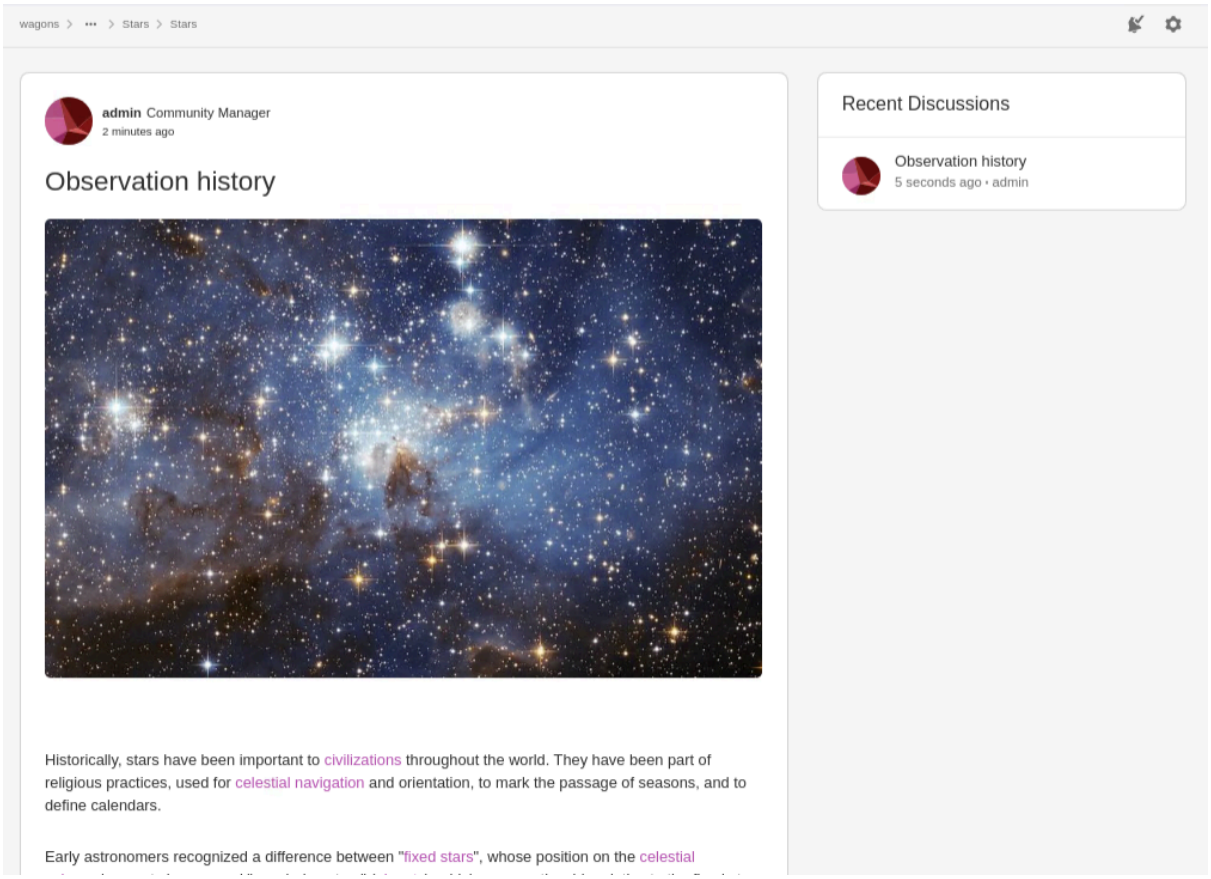


Figura 5. Imágen subida utilizada en el cuerpo de un mensaje

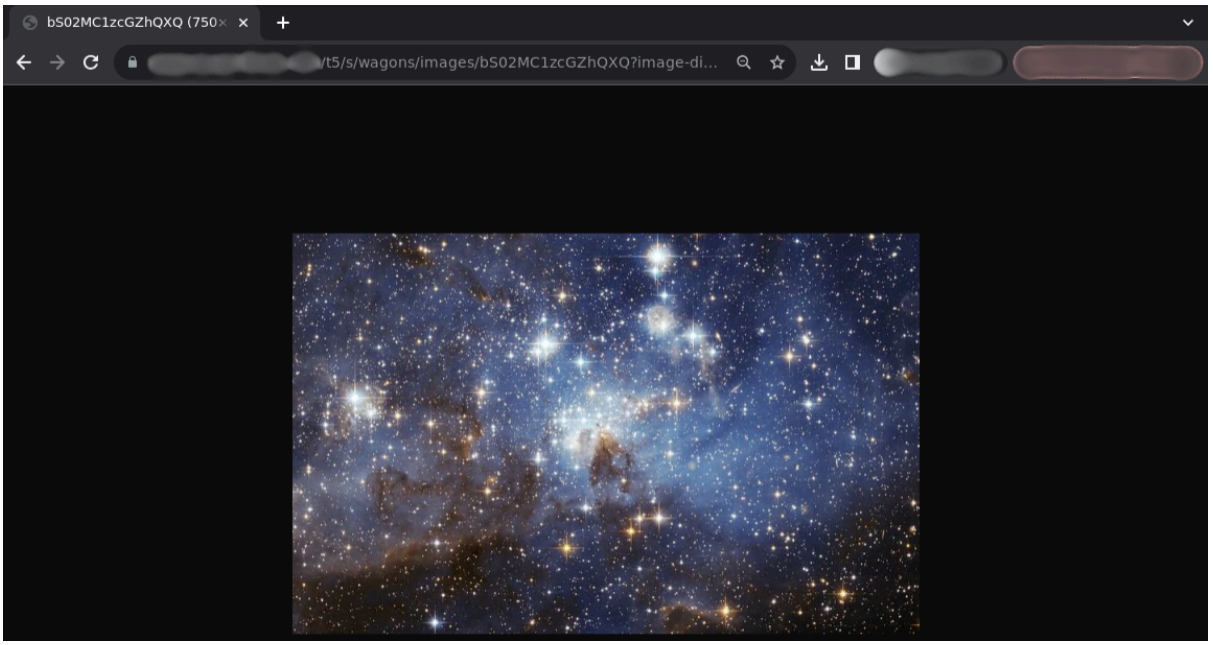


Figura 6. URL de la imagen utilizada en el cuerpo de un mensaje para un usuario con permisos

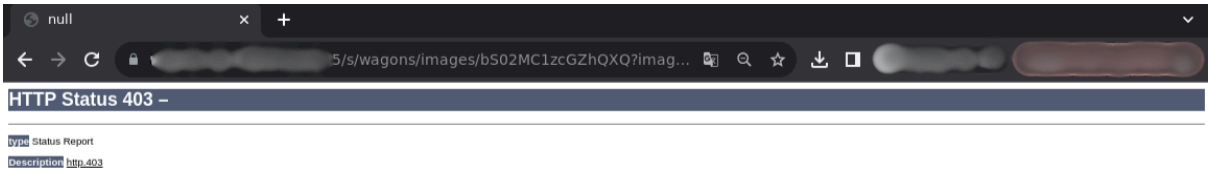


Figura 7. URL de la imagen utilizada en el cuerpo de un mensaje para un usuario sin permisos

4.4. Migrar imágenes a la nueva versión desarrollada

Todas las imágenes existentes en el sistema que se encuentren asociadas a alguna entidad (posteo, mensaje privado, avatar de usuario o avatar de nodo) serán migradas a la nueva version con su correspondiente asociación, y todas aquellas que no estén siendo utilizadas como “contenido” para la comunidad serán ignoradas, dado que como se mencionó anteriormente, ya no son relevantes al no existir en la nueva versión el concepto de álbum.

Para ello, principalmente es necesario conocer acerca de todas las tablas involucradas, en este caso, y dado el alcance del trabajo, en la composición de lo que llamamos un mensaje dentro de la comunidad.

```
INSERT INTO tblia_media_associations (media_data_id, entity_id, entity_type,
media_num, association_type, entity_revision_num, associated_description)
    SELECT image.display_id, ip.post_id, 1, ip.image_order, ip.association_type,
    MAX(mh.revision), image.description_text
        FROM image_posts ip
        INNER JOIN tblia_image image ON ip.image_id = image.image_id
        INNER JOIN message_history mh ON mh.unique_id = ip.post_id
            WHERE mh.publish_time > 0 AND image.moderation_status_legacy = 2 AND
            image.deleted_ind = 0
    GROUP BY image.display_id, ip.post_id, ip.image_order, ip.association_type
    UNION
    SELECT image.display_id, ip.post_id, 1, ip.image_order, ip.association_type,
    1, image.description_text
        FROM image_posts ip
        INNER JOIN tblia_image image ON ip.image_id = image.image_id
        LEFT JOIN message_history2 mh ON mh.unique_id = ip.post_id
            WHERE mh.unique_id IS NULL AND image.moderation_status_legacy = 2
            AND image.deleted_ind = 0
ON DUPLICATE KEY UPDATE entity_type = 1;
```

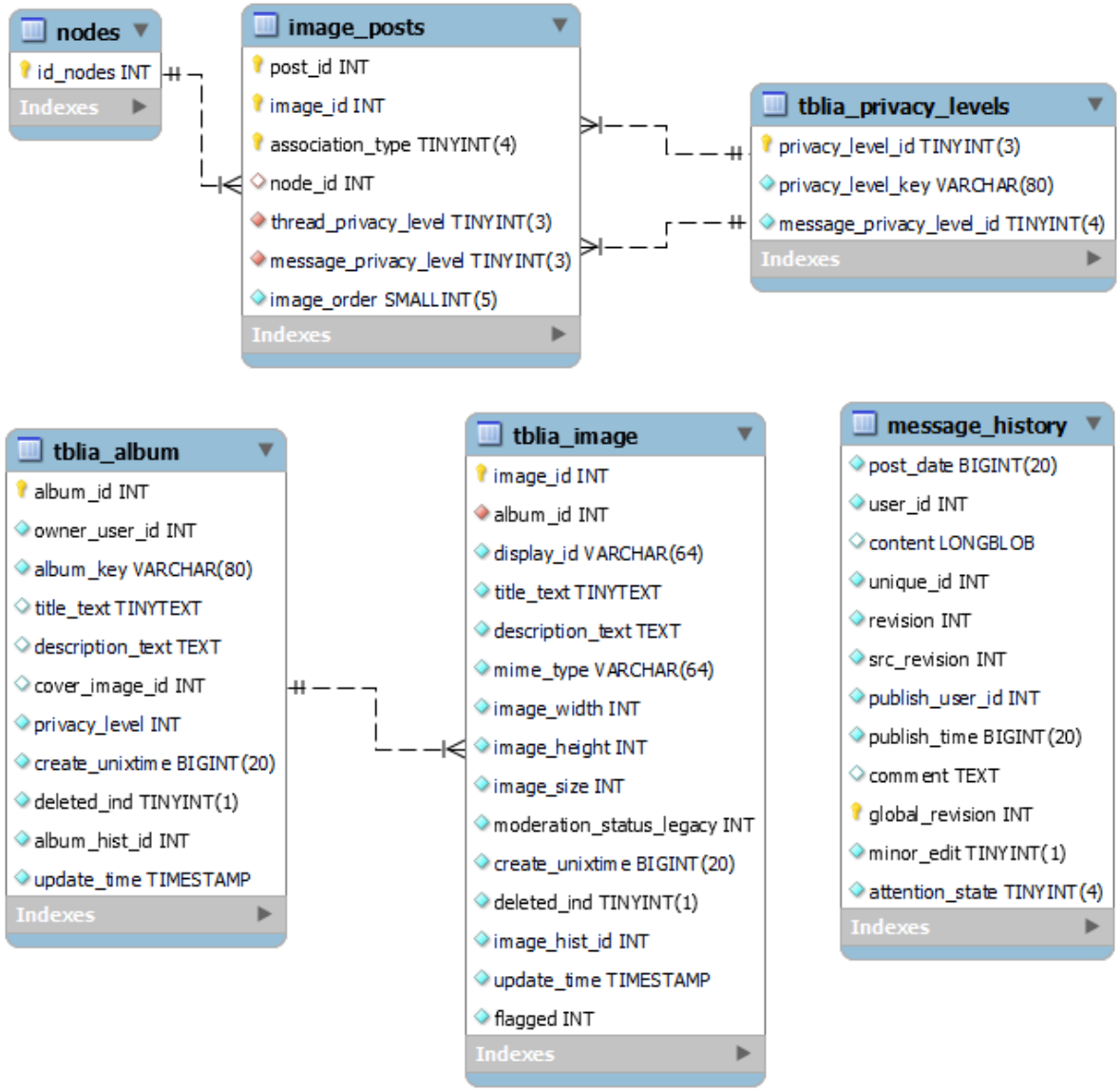


Figura 8. Diseño de tablas de base de datos utilizada para obtener la lista de ids asociados a mensajes que contienen imágenes

Se define entonces un servicio de migración de imágenes que corre en segundo plano al momento en que la instancia se actualiza para utilizar la nueva versión de imágenes (Images v2). El mismo se desarrolla en Java creando un hilo en segundo plano que sirve de servicio.

4.5. Eliminar imágenes no utilizadas

Dado que el usuario tiene la posibilidad ahora de subir una imágen al sistema utilizando la *mutation* expuesta en GraphQL a través de la interfaz de usuario, y posteriormente no utilizarla en la comunidad, es decir, no asociarla a ninguna entidad dentro de la comunidad porque se canceló la acción, se requiere de un servicio que corra en segundo plano con la finalidad de eliminar por completo las mismas del sistema 24 horas después de haberse subido si no están siendo utilizadas en la comunidad, esto es, si no tienen una asociación definida en la tabla `tblia_image_associations` dentro de la base de datos.

Esto permite que la base de datos no se encuentre almacenando imágenes que no están siendo referenciadas, por lo que no serán accesibles desde ninguna página en la comunidad.

El desarrollo de servicios en segundo plano para mantener una base de datos limpia en términos de data obsoleta o no referenciada es una buena práctica, dado que, al estar involucrada a consultas SQL altamente costosas, por el volumen de datos que implica, hace que mejore la eficiencia, y en consecuencia, la experiencia de usuario.

5. Planes a Futuro

Se espera que siguiendo el formato de trabajo realizado como parte de este proyecto para imágenes que pertenecen al cuerpo de un mensaje, la cual llamamos *MessageAssociatedImage*, todos los tipos de imágenes presentes en la aplicación que se encuentren asociadas a una entidad sean migradas y consumidas como imágenes en su versión 2 por Aurora y que la versión 1 de imágenes sea totalmente incompatible a partir de entonces.

Asociaciones de imágenes que restan por migrar:

- *PrivateNoteAssociatedImage*: Son aquellas imágenes enviadas como mensajes privados dentro de la aplicación
- *NodeAssociatedImage*: Son aquellas imágenes utilizadas como avatar de un nodo
- *UserAssociatedImage*: Son aquellas imágenes utilizadas como avatar de usuario
- *BadgeAssociatedImage*: Se denomina badge a aquellas insignias que los usuarios obtienen como “premios” por participar en la comunidad o lograr objetivos en la misma, por lo que son aquellas imágenes asociadas a cada insignia.
- *RankAssociatedImage*: Son aquellas imágenes asociadas al rango que se le otorga a los usuarios cuando cumplen cierto criterio, ya sea de participación o de permisos en la comunidad.

6. Conclusiones

El proceso de adaptación del módulo de manipulación de imágenes para el Proyecto Aurora no sólo constituye un hito en la evolución tecnológica de Khoros, sino que también establece un estándar significativo en la mejora continua del producto. La transición de la antigua lógica de almacenamiento basada en álbumes hacia la innovadora *ImagesV2* responde no solo a la demanda actual de eficiencia y personalización, sino que también sienta las bases para futuras expansiones y mejoras en la plataforma.

La capacidad de subir, manipular y utilizar imágenes a través de GraphQL no solo simplifica el proceso para los usuarios finales, sino que también brinda una mayor flexibilidad y control a los administradores de comunidades. La interoperabilidad con la API REST permite una integración fluida, asegurando una experiencia de usuario cohesiva y eficiente.

La eliminación del concepto de álbumes en favor de una gestión más centrada en los mensajes, usuarios y nodos representa un avance considerable en la optimización de recursos. Esta decisión no solo alinea la plataforma con las tendencias actuales de desarrollo de comunidades en línea, sino que también responde a una demanda creciente de simplificación y focalización en el contenido relevante.

La capacidad de migrar imágenes desde la versión anterior a la nueva *ImagesV2* demuestra un compromiso con la retrocompatibilidad, así como también destaca el enfoque proactivo de la organización hacia la satisfacción del cliente. Esta funcionalidad, además de preservar el valioso contenido multimedia existente, facilita la transición sin inconvenientes para las comunidades preexistentes que deseen adoptar el Proyecto Aurora.

En resumen, este trabajo de adaptación no solo cumple con los objetivos establecidos, sino que también sienta las bases para futuras innovaciones y mejoras. *ImagesV2* es una respuesta a las demandas actuales de sus clientes, y este proyecto no solo es una contribución técnica, sino también una declaración de compromiso con la excelencia y la evolución constante en el panorama de la tecnología de comunidades en línea.

7. Bibliografía

[1] Qué es SCRUM. (2021, 20 septiembre). Proyectos Ágiles.
https://proyectosagiles.org/que-es-scrum/

[2] What is Scrum: a guide to the most popular agile framework. (s. f.).
https://www.scrumalliance.org/about-scrum/events

[3] ¿Qué es GraphQL? (s. f.-b). https://www.redhat.com/es/topics/api/what-is-graphql

[4] GraphQL | a query language for your API. (s. f.). https://graphql.org/

[5] Jesús. (2023, 6 noviembre). GraphQL: ¿Qué es y cómo está cambiando el desarrollo web?
Tutoriales Dongee. https://www.dongee.com/tutoriales/graphql-que-es/

[6] Academia del Editor GraphQL: Aprende para qué sirve GraphQL. (s. f.). GraphQL Editor.
https://graphqleditor.com/es/graphql/

[7] Unsplash. (s. f.). Términos y condiciones | Unsplash. Unsplash.
https://unsplash.com/es/t%C3%A9rminos

[8] Entities - The Java EE 6 tutorial. (2013, 1 enero).
https://docs.oracle.com/javaee/6/tutorial/doc/bnbqa.html

8. Anexos

Anexo A: Glosario

C

Comunidad en línea: Se refiere a un grupo de personas que interactúan y se conectan a través de Internet con el objetivo de compartir intereses, información, ideas o experiencias comunes. Estas comunidades pueden adoptar diversas formas y tamaños, abarcando desde foros y redes sociales hasta plataformas especializadas y aplicaciones de mensajería.
Algunas características comunes de las comunidades en línea incluyen la posibilidad de publicar mensajes, comentarios o contenido multimedia, así como la interacción entre los miembros a través de discusiones, grupos temáticos o chats. Estas comunidades pueden centrarse en una variedad de temas, como hobbies, profesiones, problemas específicos, o simplemente proporcionar un espacio para la interacción social.

L

LIA: Llamaremos **LIA** a la API Rest preexistente desarrollada en Java 8. Esta fue creada con el propósito de comunicar la lógica de negocios con la interfaz de usuario que actualmente se ofrece a los clientes (para fines de este trabajo la llamaremos LIA Legacy). Dicha API "legacy" es también utilizada como base para exponer entities[2] a través de GraphQL.

N

Nodo: Se denomina **Nodo** en LIA a toda aquella entidad que compone la estructura de árbol de la comunidad, siendo la misma comunidad, el nodo raíz, o el de más alto nivel en la jerarquía de nodos.
Cada **comunidad** de Khoros es única y las categorías, foros, mensajes y usuarios existen dentro del contexto de esa comunidad.
Las **categorías** son nodos contenedores que almacenan tableros y/o otras categorías.
Un **grupo** es otro tipo de nodo contenedor. Puede tener un tipo de membresía abierta, cerrada u oculta y puede contener uno de cada estilo de interacción (foro, idea, base de conocimiento, preguntas y respuestas, blog).

[2] En java, una entidad es un objeto de persistencia. Normalmente, una entidad representa una tabla en una base de datos relacional, y cada instancia de dicha entidad corresponde con una columna en dicha tabla. [8]

Los **tableros** son otro tipo de contenedor, pero en este caso no de nodos, sino de discusiones, también conocidas como mensajes o conversaciones. Un tablero puede ser un **foro**, **blog**, **base de conocimientos**, **evento**, **idea**, **concurso**, **preguntas y respuestas**.

Anexo B: Interfaz de usuario

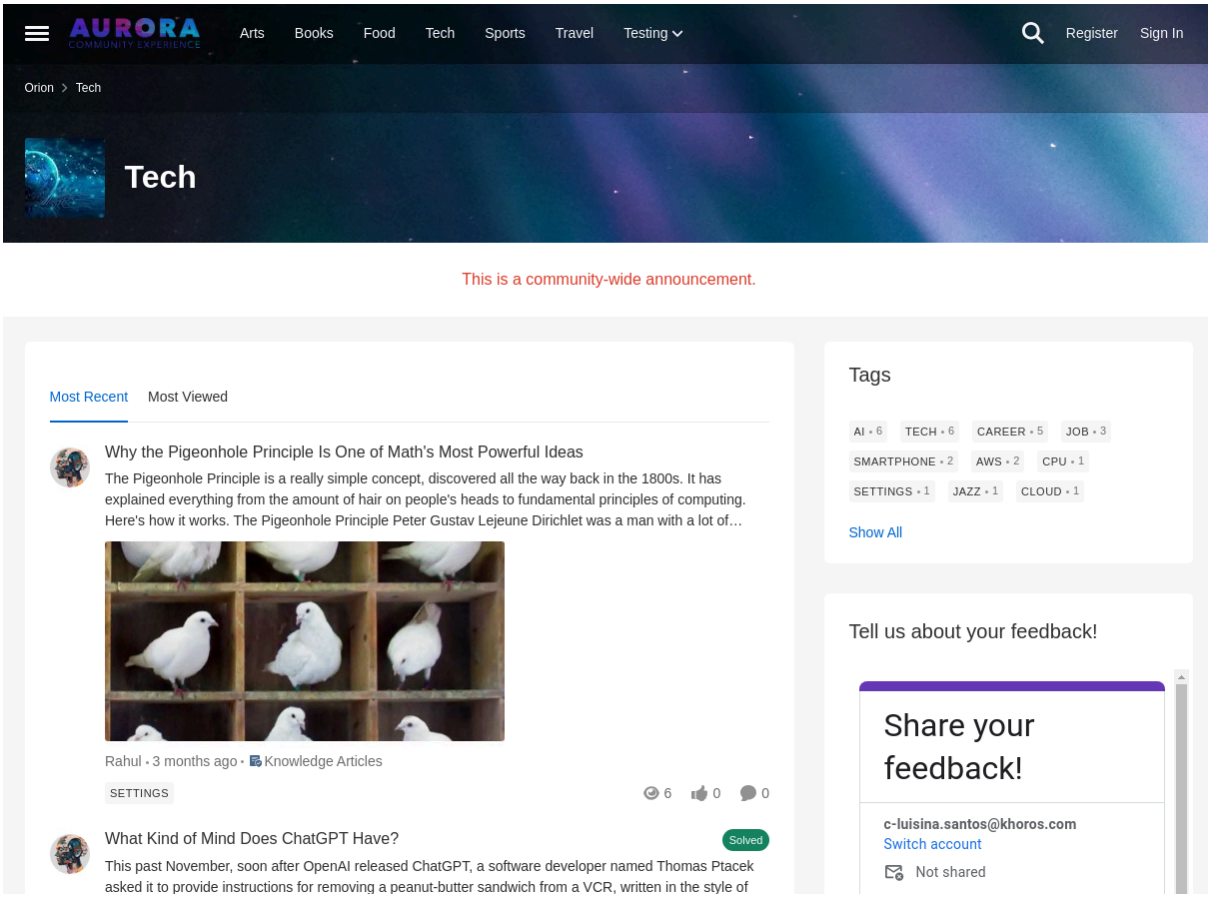


Figura 9: Vista de un mensaje que contiene imágenes en una página

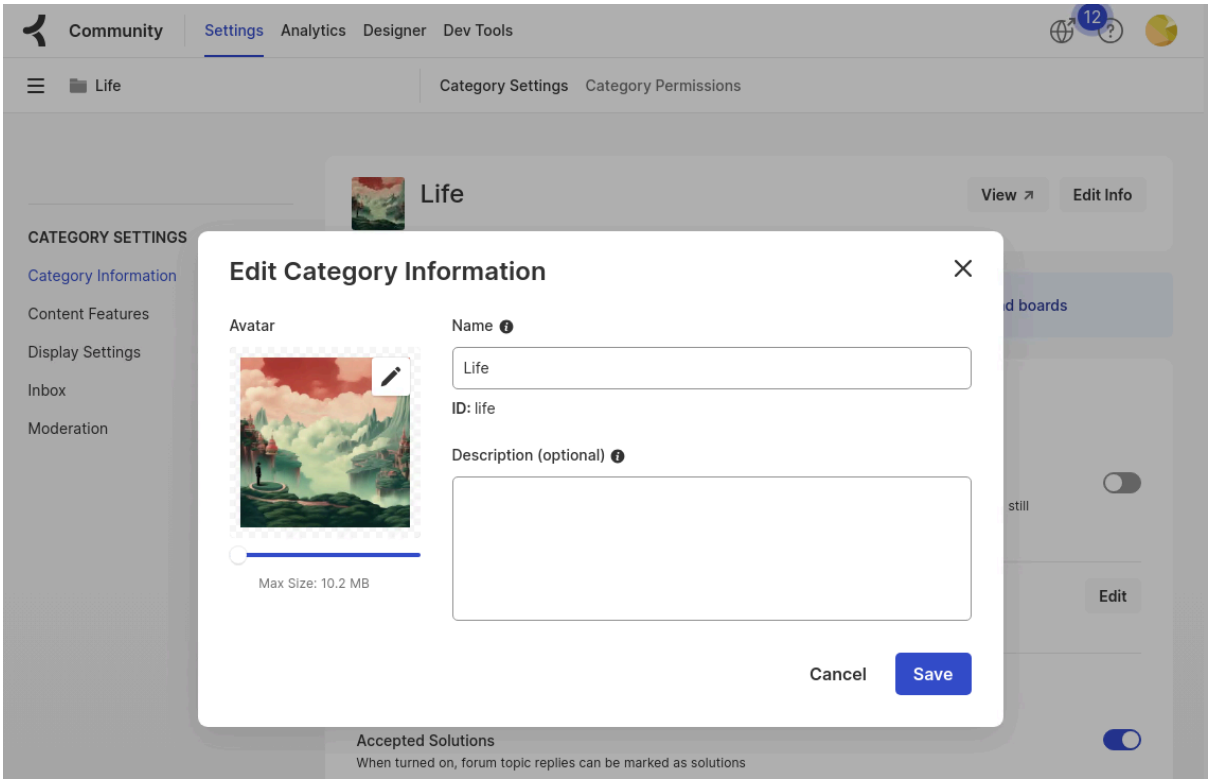


Figura 10: Vista de la página de administración de nodos

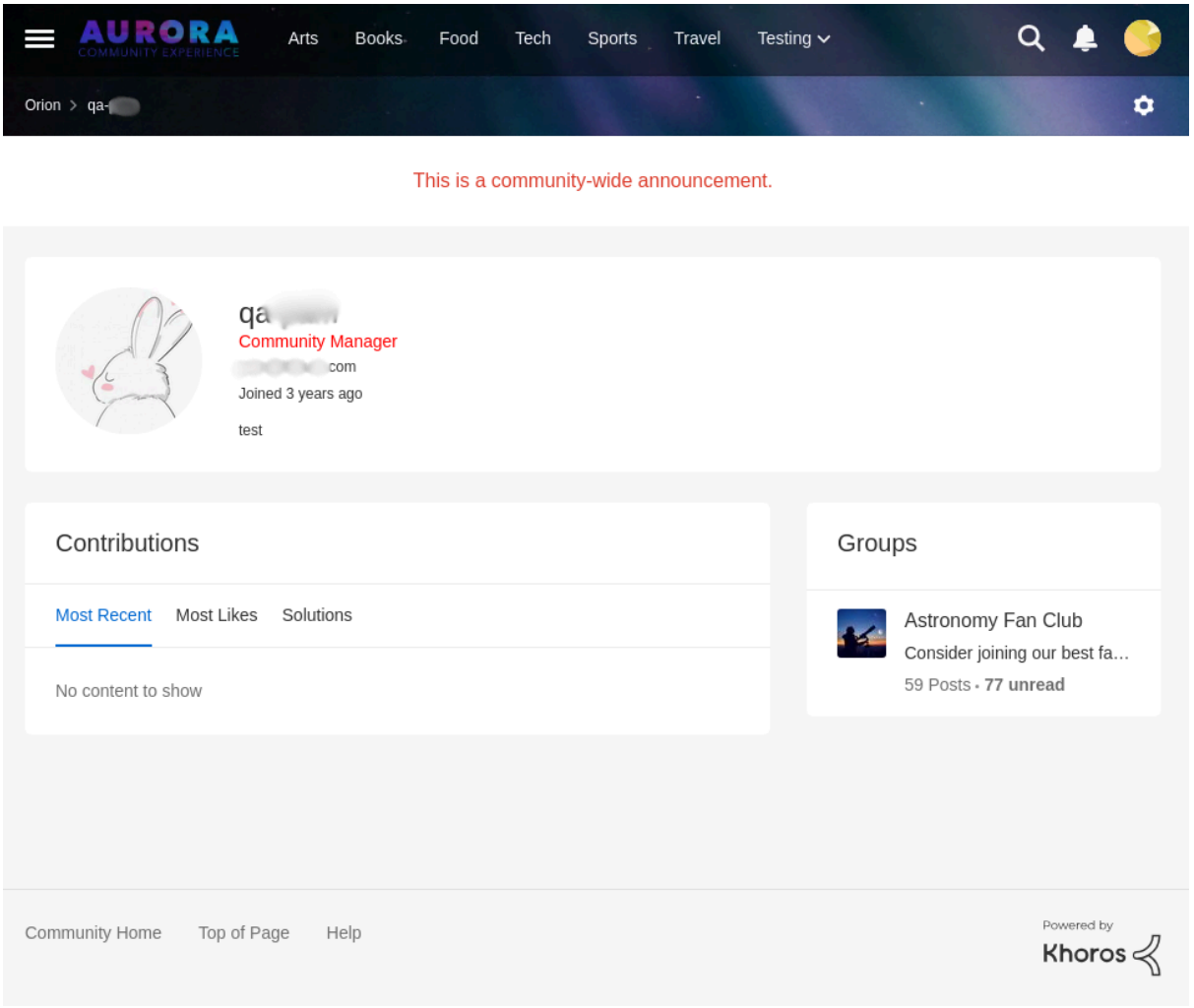


Figura 11: Vista de la página de usuario

9. Agradecimientos

A mis padres, Lorena y Claudio.