TABLA DE CONTENIDO

INTRODUCCIÓN

El presente documento expone la propuesta de desarrollo para la plataforma de noticias INFILE, diseñada para cubrir necesidades de gestión, publicación y consumo de contenido informativo, así como la administración de usuarios y membresías. Dada la complejidad funcional y técnica del sistema, se plantea una estrategia de implementación dividida en fases, con el objetivo de asegurar la viabilidad, escalabilidad y mantenibilidad del proyecto.

La solución contempla la integración de tecnologías modernas y probadas, como FastAPI para el backend, Django para el frontend, PostgreSQL como base de datos principal, Redis para el manejo eficiente de visitas y Celery para la sincronización de procesos. Además, se consideran aspectos de seguridad, usabilidad y reportería corporativa, alineados con los requerimientos del negocio.

La división por fases responde a la necesidad de mitigar riesgos, facilitar la validación temprana de funcionalidades críticas y permitir ajustes progresivos según la retroalimentación de los usuarios y stakeholders. Cada fase se define con entregables claros y medibles, priorizando la construcción de una base sólida antes de abordar módulos avanzados.

Este documento detalla el alcance, objetivos y entregables de cada fase, así como los criterios de éxito y consideraciones técnicas relevantes para la ejecución del proyecto.

ALCANCES

- **Autenticación y gestión de usuarios:**

  Registro, login con JWT, recuperación y cambio de contraseña, gestión de roles (Administrador/Lector) y municipios, con validaciones y seguridad en el manejo de credenciales.

- **Gestión de noticias y categorías:**

  CRUD completo para noticias y categorías, con soporte de rich text editor (tinymce) e integración de imágenes, recomendación por autor y categoría, y asociación de cada noticia a una única categoría.

- **Lectura y visualización de noticias:**

  Listado paginado y filtrable, visualización de noticias solo para usuarios autenticados, sin apertura en pestañas externas.

- **Despliegue y entorno local:**

  Dockerización de todos los servicios (FastAPI, Django, PostgreSQL, Redis, Celery), documentación técnica y scripts para instalación y levantamiento del entorno.

- **Gestión de membresías y pagos:**

  CRUD de tipos de membresía, compra y asignación de membresías (incluyendo membresías gratuitas), integración con pasarela de pagos y reportería sobre membresías y pagos.

- **Reportería corporativa:**

  Consultas sobre usuarios con membresía, membresías próximas a vencer, categorías con mayor número de noticias, pagos realizados y demanda por municipio.

- **OCR y contador de visitas:**

  Integración de lector OCR para extracción de texto desde imágenes en noticias, implementación de contador de visitas con Redis y sincronización a PostgreSQL mediante Celery.

LÍMITES

- **Integración de pagos:**

  Se limita a la integración con una sola pasarela de pagos definida en la fase de análisis (ejemplo: Stripe o MercadoPago). No se contempla la gestión de reembolsos ni conciliación bancaria avanzada.

- **Gestión de imágenes:**

  El almacenamiento de imágenes se realizará en el sistema de archivos local, en la carpeta de files del proyecto de la API. No se incluye procesamiento avanzado ni CDN.

- **OCR:**
  El reconocimiento de texto se limita a imágenes en formatos estándar (jpg, png) y a la calidad soportada por Tesseract OCR. No se garantiza precisión en textos manuscritos o imágenes de baja calidad.

- **Contador de visitas:**

  El contador se sincroniza periódicamente, pero no se garantiza exactitud en tiempo real ante caídas de Redis o Celery. El reporte es orientativo y no apto para auditoría legal.

- **Escalabilidad y alta disponibilidad:**

  El entorno está preparado para desarrollo y pruebas. La escalabilidad horizontal, balanceo de carga y alta disponibilidad quedan fuera del alcance inicial.

- **Soporte y mantenimiento:**

  El proyecto contempla únicamente la entrega técnica y documentación. Soporte post-entrega, mantenimiento evolutivo y capacitación quedan fuera del alcance.

- **Internacionalización y accesibilidad:**

  La plataforma se entrega en idioma español y con accesibilidad básica. No se incluye soporte multilenguaje ni certificación de accesibilidad.

REQUISITOS FUNCIONALES

1. **Registro y autenticación de usuarios**
    - Permitir registro de usuarios con correo electrónico, nombre, teléfono, municipio y contraseña.
    - Validar unicidad de correo y enviar correo de confirmación.
    - Login mediante JWT y recuperación/cambio de contraseña por correo.
2. **Gestión de roles**
    - Asignación y gestión de roles: Administrador y Lector.
    - Control de acceso a funcionalidades según rol.
3. **Gestión de noticias**
    - CRUD completo de noticias para administradores.
    - Asociación de noticia a una única categoría y autor.
    - Edición de cuerpo de noticia con rich text editor (tinymce) y soporte de imágenes.
4. **Gestión de categorías**
    - CRUD completo de categorías para administradores.
    - Validación de unicidad de nombre y estado activo/inactivo.
5. **Lectura y visualización de noticias**
    - Listado paginado y filtrable de noticias por categoría y autor.
    - Visualización de detalle de noticia solo para usuarios autenticados.
6. **Gestión de usuarios**
    - CRUD de usuarios por administradores, con asignación de roles y municipios.
7. **Gestión de membresías**
    - CRUD de tipos de membresía.
    - Compra y asignación de membresías (incluyendo membresías gratuitas).
    - Integración con pasarela de pagos.
8. **Reportería corporativa**
    - Consultas sobre usuarios con membresía, membresías próximas a vencer, categorías con más noticias, pagos realizados y demanda por municipio.

9. **OCR en noticias**
    - Carga de imágenes y extracción de texto mediante OCR para agilizar la publicación.
10. **Contador de visitas**
    - Registro de visitas a noticias usando Redis y sincronización periódica con PostgreSQL mediante Celery.
11. **Despliegue y documentación**
    - Dockerización del entorno local.
    - Documentación de la API con Swagger.

REQUISITOS NO FUNCIONALES

1. **Seguridad**
    - Almacenamiento seguro de contraseñas (hash).
    - Autenticación y autorización basada en JWT.
    - Validación de datos en todos los endpoints.
2. **Performance**
    - Uso de Redis para manejo eficiente de visitas.
    - Paginación en listados y consultas.
3. **Escalabilidad**
    - Arquitectura modular y dockerizada para facilitar despliegue y escalabilidad futura.
4. **Disponibilidad**
    - Tolerancia básica a fallos en sincronización de visitas (Redis/Celery).
    - Recuperación ante errores en OCR y carga de imágenes.
5. **Usabilidad**
    - Interfaz amigable en Django, con gama de colores INFILE.
    - Rich text editor para edición de noticias.
6. **Mantenibilidad**
    - Código documentado y estructurado.
    - Pruebas unitarias y de integración para endpoints críticos.
7. **Compatibilidad**
    - Soporte para navegadores modernos.
    - Imágenes en formatos estándar (jpg, png).
8. **Portabilidad**
    - Entorno local completamente dockerizado.
    - Scripts y documentación para instalación y levantamiento.
9. **Auditoría básica**
    - Registro de acciones críticas (alta, edición, eliminación) en base de datos.

10. **Idioma**

- Plataforma en español.

DIAGRAMA DE ARQUITECTURA

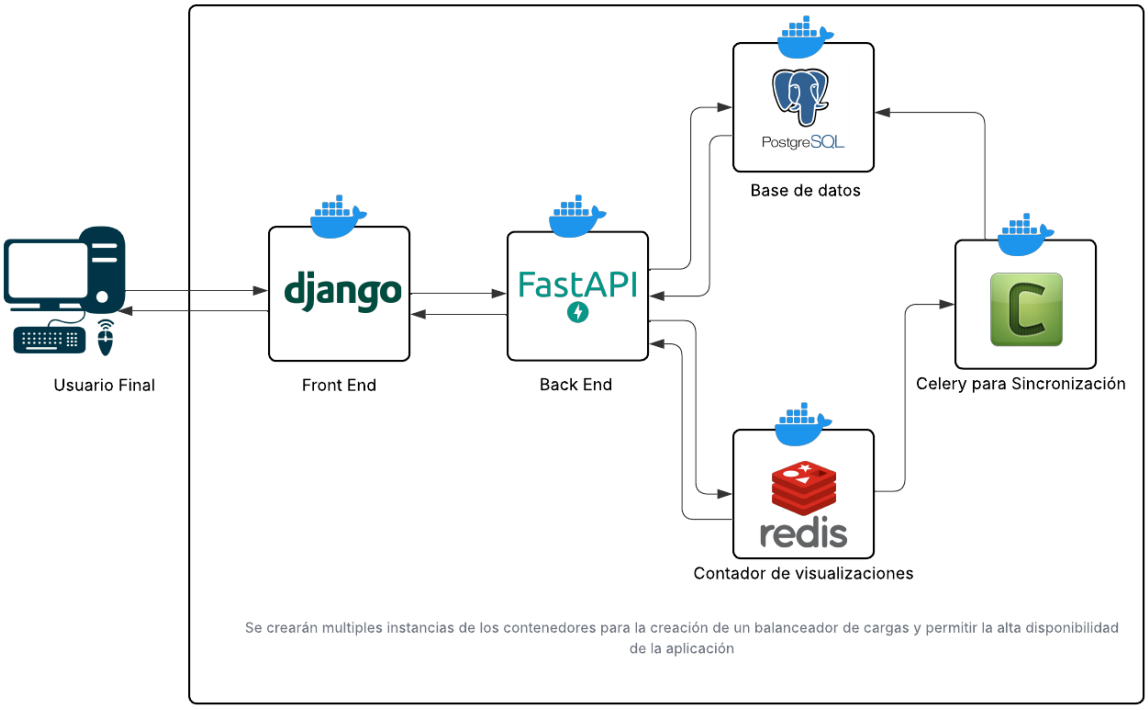

DIAGRAMA DE ENTIDAD RELACION

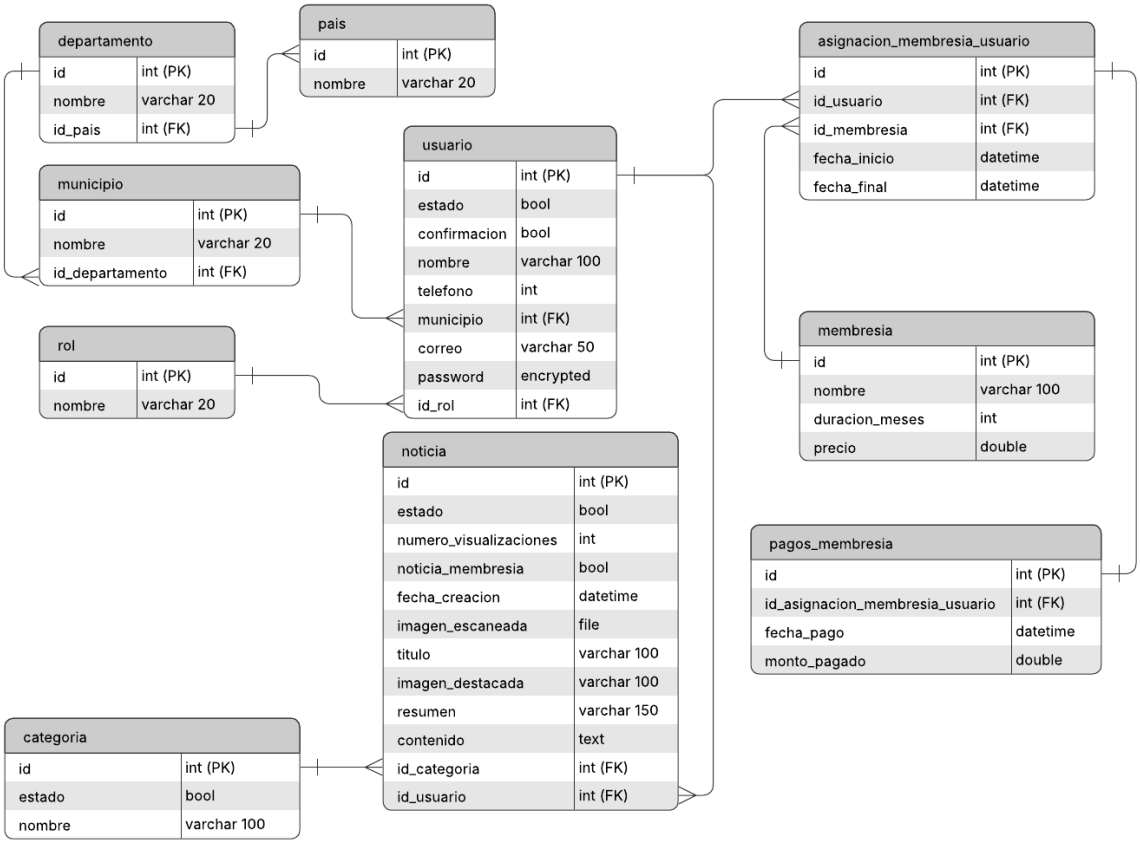

DIAGRAMA DE PROCESOS

Debido a lo extenso que son los procesos, el diagrama se visualiza desde la plataforma de draw.io en el siguiente enlace → Click aquí

DIVISIÓN EN FASES

La decisión de dividir el desarrollo de la plataforma en tres fases responde a la complejidad funcional, la integración de múltiples tecnologías y la necesidad de asegurar entregas viables en tiempos acotados. A continuación, se detallan los motivos principales:

1. **Limitación de tiempo y recursos:**

   La primera fase cuenta con un máximo de 8 horas de desarrollo, lo que obliga a priorizar funcionalidades críticas y asegurar un producto mínimo viable (MVP) que permita validar los flujos esenciales de autenticación, gestión y visualización de noticias.

2. **Reducción de riesgos técnicos:**

   La arquitectura propuesta involucra FastAPI, Django, PostgreSQL, Redis y Celery, además de integración con servicios externos (correo, pagos, OCR). Dividir el proyecto permite abordar primero los módulos base y luego, en fases posteriores, integrar componentes avanzados y procesos asíncronos, minimizando riesgos de integración y permitiendo pruebas controladas.

3. **Validación incremental:**

   La entrega por fases facilita la validación temprana de funcionalidades, la obtención de retroalimentación de usuarios y stakeholders, y la corrección de posibles desviaciones antes de avanzar a módulos más complejos como membresías, reportería, OCR y contador de visitas.

4. **Gestión eficiente del alcance:**

   La segmentación permite definir entregables claros y medibles en cada etapa, evitando la dispersión de esfuerzos y asegurando que cada fase cumpla con los criterios de aceptación y calidad definidos.

5. **Facilidad de despliegue y mantenimiento:**

   Al dockerizar el entorno desde la primera fase y documentar la API con Swagger, se garantiza que el equipo pueda instalar, probar y mantener el sistema de forma modular, facilitando la incorporación de nuevas funcionalidades en fases posteriores sin afectar la estabilidad del núcleo.

6. **Escalabilidad y evolución:**

   La división en fases permite que el sistema evolucione de forma ordenada, integrando reportería corporativa, gestión avanzada de usuarios y membresías, y funcionalidades especializadas (OCR, contador de visitas) solo cuando la base está probada y estable.

En resumen, la estrategia de desarrollo por fases es la opción más realista y viable para asegurar calidad, minimizar riesgos y cumplir con los tiempos y recursos disponibles, sin comprometer la robustez ni la escalabilidad futura del sistema.

METODOLOGÍA DEL DESARROLLO

El desarrollo de la plataforma se realizará bajo un enfoque ágil, priorizando la entrega incremental y la validación continua de funcionalidades. Se utilizarán los siguientes principios y prácticas:

1. Iteraciones cortas y entregables por fase

- El proyecto se divide en tres fases, cada una con objetivos y entregables claros.
- Al final de cada fase se realiza una revisión técnica y funcional, permitiendo ajustes antes de avanzar.

2. Gestión de historias de usuario

- Las historias de usuario se documentan y priorizan según valor de negocio y dependencia técnica.
- Cada historia incluye criterios de aceptación, endpoints, tecnologías y parámetros DoD (Definition of Done).

3. Desarrollo guiado por pruebas

- Se implementan pruebas unitarias y de integración para endpoints y procesos críticos.
- La cobertura de pruebas es revisada en cada iteración para asegurar calidad y evitar regresiones.

4. Control de versiones y CI/CD

- El código fuente se gestiona en un sistema de control de versiones (Git).
- Se configura integración continua para ejecutar pruebas y validar builds en cada commit.
- El despliegue local se realiza mediante Docker y docker-compose, asegurando portabilidad y reproducibilidad.

5. Documentación técnica

- La API se documenta con Swagger/OpenAPI.
- Se generan manuales de instalación, uso y despliegue para cada fase.

6. Revisión y retroalimentación

- Al cierre de cada fase se realiza una demo funcional y se recopila retroalimentación de usuarios y stakeholders.
- Los cambios y mejoras se priorizan para la siguiente iteración.

7. Gestión de riesgos

- Se identifican riesgos técnicos y de integración en cada fase.

- Se aplican pruebas de estrés y validaciones de seguridad en módulos críticos.

8. Comunicación continua

- Se mantienen reuniones breves de seguimiento para resolver bloqueos y ajustar prioridades.
- El avance y los impedimentos se documentan y comunican oportunamente.

FASE 1

Epic: Autenticación y acceso de usuarios

Implementación de los mecanismos de registro, inicio de sesión, recuperación y cambio de contraseña, asegurando la protección de datos y el acceso seguro a la plataforma.

Historia 1: Registro de usuarios

Descripción:
Como usuario, quiero poder registrarme en la plataforma proporcionando mi nombre, correo electrónico, teléfono, municipio y contraseña, para acceder a las funcionalidades de la aplicación.

Endpoints:

- POST /api/v1/auth/register (FastAPI)

Librerías/Tecnologías:

- FastAPI (backend)
- Django (frontend)
- PostgreSQL (persistencia)
- JWT (autenticación)
- Passlib/Bcrypt (hash de contraseñas)
- Swagger (documentación API)

Criterios de aceptación:

- El usuario debe recibir un correo de confirmación.
- El registro debe validar unicidad de correo.
- El password debe almacenarse encriptado.
- El usuario se asocia a un municipio y rol por defecto (Lector).

DoD:

- Endpoint documentado en Swagger.

- Pruebas unitarias y de integración.
- Validaciones de datos y errores.
- Registro exitoso crea usuario en la tabla usuario (ver ERD).
- Email de confirmación enviado.

Historia 2: Login con JWT

Descripción:
Como usuario registrado, quiero iniciar sesión con mi correo y contraseña para obtener un token JWT y acceder a las noticias.

Endpoints:

- POST /api/v1/auth/login (FastAPI)

Librerías/Tecnologías:

- FastAPI
- JWT (PyJWT)
- PostgreSQL

Criterios de aceptación:

- El usuario recibe un JWT válido.
- El JWT incluye el id, rol y municipio.
- El login falla si el usuario está inactivo o la contraseña es incorrecta.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Validación de credenciales y estado.
- Token JWT generado y retornado.

Historia 3: Recuperar Contraseña

Descripción:
Como usuario, quiero recuperar mi contraseña mediante correo electrónico para restablecer el acceso a mi cuenta.

Endpoints:

- POST /api/v1/auth/forgot-password
- POST /api/v1/auth/reset-password

Librerías/Tecnologías:

- FastAPI
- Email backend (SMTP)
- PostgreSQL

Criterios de aceptación:

- El usuario recibe un correo con enlace de recuperación.
- El enlace permite establecer una nueva contraseña.
- El password se almacena encriptado.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Flujo completo de recuperación y cambio de contraseña.
- Email enviado correctamente.

Historia 4: Cambiar Contraseña

Descripción:
Como usuario autenticado, quiero cambiar mi contraseña desde mi perfil para mantener la seguridad de mi cuenta.

Endpoints:

- POST /api/v1/auth/change-password

Librerías/Tecnologías:

- FastAPI
- JWT
- PostgreSQL

Criterios de aceptación:

- El usuario debe estar autenticado.
- La nueva contraseña se almacena encriptada.
- Se valida la contraseña actual antes de permitir el cambio.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Validación de contraseña actual.
- Cambio exitoso reflejado en la base de datos.

Epic: Noticias y categorías

Desarrollo de los módulos para la gestión (alta, edición, eliminación) de noticias y categorías, permitiendo la organización y publicación eficiente del contenido.

Historia 5: CRUD de Noticias

Descripción:
Como administrador, quiero crear, editar, eliminar y restaurar noticias, asociándolas a una categoría y usuario autor, para gestionar el contenido publicado.

Endpoints:

- GET /api/v1/news
- POST /api/v1/news
- PUT /api/v1/news/{id}
- DELETE /api/v1/news/{id}
- POST /api/v1/news/{id}/restore
- POST /api/v1/news/upload-image

Librerías/Tecnologías:

- FastAPI
- Django (tinymce para rich text e imágenes)
- PostgreSQL

Criterios de aceptación:

- Solo administradores pueden acceder al CRUD.
- Cada noticia tiene una categoría y autor.
- Cada noticia tiene una imagen destacada que debe ser subida al EP upload-image.
- El contenido soporta texto enriquecido e imágenes.
- Estado de la noticia (activo/inactivo) gestionado.
- Validación de unicidad de título.

DoD:

- Endpoints documentados en Swagger.
- Pruebas unitarias y de integración.
- Validaciones de datos y errores.
- Noticias reflejadas en la tabla noticia (ver ERD).

- Imágenes almacenadas correctamente.

Historia 6: CRUD de Categorías

Descripción:
Como administrador, quiero crear, editar, eliminar y restaurar categorías para organizar las noticias.

Endpoints:

- GET /api/v1/categories
- POST /api/v1/categories
- PUT /api/v1/categories/{id}
- DELETE /api/v1/categories/{id}
- POST /api/v1/categories/{id}/restore

Librerías/Tecnologías:

- FastAPI
- PostgreSQL

Criterios de aceptación:

- Solo administradores pueden acceder al CRUD.
- Validación de unicidad de nombre.
- Estado de la categoría (activo/inactivo) gestionado.

DoD:

- Endpoints documentados en Swagger.
- Pruebas unitarias y de integración.
- Categorías reflejadas en la tabla categoria (ver ERD).

Epic: Lectura de Noticias

Construcción de las vistas y funcionalidades para el listado y visualización de noticias, garantizando la experiencia de usuario y el cumplimiento de los requisitos de autenticación.

Historia 7: Listado de Noticias

Descripción:
Como usuario autenticado, quiero ver un listado paginado de noticias, filtradas por categoría y autor.

Endpoints:

- GET /api/v1/news?category={id}&author={id}&page={n}

Librerías/Tecnologías:

- FastAPI
- Django (frontend)
- PostgreSQL

Criterios de aceptación:

- Solo usuarios autenticados pueden acceder.
- Listado paginado y filtrable.
- Solo noticias activas se muestran.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Listado funcional en frontend.

Historia 8: Visualización de Noticias

Descripción:
Como usuario autenticado, quiero visualizar el detalle de una noticia, incluyendo contenido enriquecido e imágenes.

Endpoints:

- GET /api/v1/news/{id}

Librerías/Tecnologías:

- FastAPI
- Django (tinymce)
- PostgreSQL

Criterios de aceptación:

- Solo usuarios autenticados pueden acceder.
- Visualización completa del contenido y metadatos.
- No se requiere abrir en otra pestaña.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Visualización funcional en frontend.

Epic: Despliegue

Configuración del entorno local mediante Docker, asegurando la portabilidad y facilidad de instalación para el equipo de desarrollo.

Historia 9: Dockerización del entorno local

Descripción:
Como desarrollador, quiero contar con archivos Docker y docker-compose para levantar el entorno local de la plataforma, incluyendo backend, frontend, base de datos y servicios auxiliares.

Endpoints:

- N/A (configuración)

Librerías/Tecnologías:

- Docker
- Docker Compose
- FastAPI
- Django
- PostgreSQL

Criterios de aceptación:

- El entorno se levanta con un solo comando.
- Todos los servicios se comunican correctamente.
- Documentación clara de uso.

DoD:

- Archivos Docker y docker-compose en el repositorio.
- Documentación de instalación y uso.
- Pruebas de levantamiento exitoso.

FASE 2

Epic: Gestión de usuarios

Incorporación de herramientas administrativas para la gestión integral de usuarios, incluyendo edición, eliminación y asignación de roles.

Historia 10: CRUD de Usuarios

Descripción:
Como administrador, quiero poder crear, editar, eliminar y consultar usuarios, asignando roles y municipios, para gestionar el acceso y permisos en la plataforma.

Endpoints:

- GET /api/v1/users
- POST /api/v1/users
- PUT /api/v1/users/{id}
- DELETE /api/v1/users/{id}

Librerías/Tecnologías:

- FastAPI
- PostgreSQL
- JWT (autorización por rol)
- Swagger

Criterios de aceptación:

- Solo administradores pueden acceder al CRUD.
- Validación de unicidad de correo.
- Asignación de roles (Administrador/Lector) y municipio.
- Eliminación lógica (estado activo/inactivo).

DoD:

- Endpoints documentados en Swagger.
- Pruebas unitarias y de integración.
- Cambios reflejados en la tabla usuario (ver ERD).
- Validaciones de datos y errores.

Epic: Gestión de membresías

Implementación de la compra, asignación y administración de membresías, así como la gestión de los tipos de membresía disponibles.

Historia 11: CRUD de Tipo de Membresías

Descripción:
Como administrador, quiero crear, editar y eliminar tipos de membresía, definiendo duración, precio y beneficios, para ofrecer opciones a los usuarios.

Endpoints:

- GET /api/v1/memberships/types
- POST /api/v1/memberships/types
- PUT /api/v1/memberships/types/{id}
- DELETE /api/v1/memberships/types/{id}

Librerías/Tecnologías:

- FastAPI
- PostgreSQL
- Swagger

Criterios de aceptación:

- Solo administradores pueden acceder al CRUD.
- Validación de unicidad de nombre.
- Definición de duración y precio.

DoD:

- Endpoints documentados en Swagger.
- Pruebas unitarias y de integración.
- Cambios reflejados en la tabla tipo_membresia.

Historia 12: Compra de membresías

Descripción:
Como usuario, quiero poder comprar una membresía mediante el sistema de pagos, para acceder a contenido exclusivo.

Endpoints:

- POST /api/v1/memberships/purchase

Librerías/Tecnologías:

- FastAPI
- Integración con pasarela de pagos (ej. Stripe, MercadoPago)
- PostgreSQL

Criterios de aceptación:

- El pago debe ser validado y registrado.
- La membresía se asocia al usuario y se actualiza la fecha de vencimiento.
- El usuario recibe confirmación por correo.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Registro en la tabla membresia y pago.
- Email de confirmación enviado.

Historia 13: Asignación de membresías gratis

Descripción:
Como administrador, quiero asignar membresías gratuitas a usuarios específicos, para promociones o cortesías.

Endpoints:

- POST /api/v1/memberships/assign-free

Librerías/Tecnologías:

- FastAPI
- PostgreSQL

Criterios de aceptación:

- Solo administradores pueden asignar membresías gratis.
- Únicamente pueden ser asignadas membresías a usuarios activos.
- Registro de la membresía en la base de datos.
- El usuario recibe notificación por correo.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Registro en la tabla membresia.
- Email de notificación enviado.

Epic: Repotería corporativa

Desarrollo de módulos de reportería para visualizar métricas clave, como usuarios con membresía, pagos realizados y categorías más activas.

Historia 14: Cantidad de usuarios con membresía

Descripción:
Como administrador, quiero consultar el número total de usuarios con membresía activa, para análisis corporativo.

Endpoints:

- GET /api/v1/reports/memberships/active-users

Librerías/Tecnologías:

- FastAPI
- PostgreSQL
- Swagger

Criterios de aceptación:

- Solo administradores pueden acceder.
- El reporte muestra cantidad y detalle por tipo de membresía.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Datos extraídos de la tabla membresia.

Historia 15: Usuarios con membresía pronta a vencer

Descripción:
Como administrador, quiero ver el listado de usuarios cuyas membresías están próximas a vencer, para gestionar renovaciones.

Endpoints:

- GET /api/v1/reports/memberships/expiring-soon

Librerías/Tecnologías:

- FastAPI
- PostgreSQL

Criterios de aceptación:

- Solo administradores pueden acceder.
- El reporte muestra usuarios y fechas de vencimiento.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Datos extraídos de la tabla membresia.

Historia 16: Categorías con mayor número de noticias

Descripción:
Como administrador, quiero consultar las categorías con mayor cantidad de noticias publicadas, para análisis de contenido.

Endpoints:

- GET /api/v1/reports/categories/top

Librerías/Tecnologías:

- FastAPI
- PostgreSQL

Criterios de aceptación:

- Solo administradores pueden acceder.
- El reporte muestra categorías y conteo de noticias.
- Debe de mostrar únicamente el top 10 de categorías

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Datos extraídos de las tablas categoria y noticia.

Historia 17: Pagos realizados en el tiempo

Descripción:
Como administrador, quiero visualizar los pagos realizados en un periodo determinado, para control financiero.

Endpoints:

- GET /api/v1/reports/payments?start={date}&end={date}

Librerías/Tecnologías:

- FastAPI
- PostgreSQL

Criterios de aceptación:

- Solo administradores pueden acceder.
- El reporte muestra pagos filtrados por fecha.
- Los pagos son visibles en la tabla pagos_membresia.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Datos extraídos de la tabla pago.

Historia 18: Lugares con mayor demanda de membresías

Descripción:
Como administrador, quiero consultar los municipios con mayor cantidad de membresías adquiridas, para análisis de demanda geográfica.

Endpoints:

- GET /api/v1/reports/memberships/top-municipalities

Librerías/Tecnologías:

- FastAPI
- PostgreSQL

Criterios de aceptación:

- Solo administradores pueden acceder.
- El reporte muestra municipios y cantidad de membresías.
- Los municipios se encuentran en la tabla usuarios como clave foránea.

DoD:

- Endpoint documentado en Swagger.
- Pruebas unitarias y de integración.
- Datos extraídos de las tablas membresia y municipio.

FASE 3

Epic: Lector OCR

Integración de la funcionalidad de reconocimiento óptico de caracteres para procesar y extraer texto de imágenes en las noticias.

Historia 19: Lector OCR de noticias

Descripción:
Como administrador, quiero poder cargar imágenes en las noticias y extraer automáticamente el texto mediante OCR, para agilizar la publicación de contenido.

Endpoints:

- POST /api/v1/news/{id}/ocr-upload
- GET /api/v1/news/{id}/ocr-result

Librerías/Tecnologías:

- FastAPI
- Tesseract OCR (via pytesseract)
- Django (frontend para carga y visualización)
- PostgreSQL (almacenamiento de resultados)
- Swagger

Criterios de aceptación:

- Solo administradores pueden acceder a la funcionalidad.
- El sistema debe aceptar imágenes en formatos estándar (jpg, png).
- La imagen se carga en el atributo “imagen_escaneada” de la tabla noticia.
- El texto extraído se asocia a la noticia y puede ser editado antes de publicar.
- El proceso debe manejar errores de reconocimiento y notificar al usuario.

DoD:

- Endpoints documentados en Swagger.
- Pruebas unitarias y de integración.
- Resultados almacenados en la base de datos.
- Flujo de carga y edición funcional en frontend.

Epic: Contador de visitas

Implementación y sincronización del contador de visitas utilizando Redis y PostgreSQL, con Celery para la gestión de procesos asíncronos y reporte de visitas.

Historia 20: Implementación y sincronización del contador de visitas con Redis y PostgreSQL

Descripción:
Como administrador, quiero contar y visualizar el número de visitas a cada noticia, utilizando Redis para alta concurrencia y sincronizando los datos con PostgreSQL mediante Celery.

Endpoints:

- GET /api/v1/news/{id}/visits
- (Sincronización interna vía Celery, sin endpoint público)

Librerías/Tecnologías:

- FastAPI
- Redis (contador en memoria)
- Celery (tareas de sincronización)
- PostgreSQL
- Django (visualización en frontend)
- Swagger

Criterios de aceptación:

- El contador incrementa en Redis cada vez que una noticia es visualizada.
- Celery sincroniza los datos periódicamente con PostgreSQL (podría ser de forma diaria).
- El administrador puede consultar el número de visitas por noticia.
- El sistema debe manejar concurrencia y evitar pérdida de datos.

DoD:

- Endpoints documentados en Swagger.
- Pruebas unitarias y de integración.
- Sincronización funcional y tolerante a fallos.
- Visualización de visitas en frontend.