Nuevo +117-0 # Manual de Desarrollador

Este documento resume la estructura y arquitectura del proyecto **Gestor de Videojuegos**.

1. Visión general

El proyecto es una plataforma para la gestión de videojuegos y la interacción de usuarios. Consta de dos grandes componentes:

1. **Backend**: API REST desarrollada con **Django** y **Django REST Framework**.
2. **Frontend**: Aplicación web realizada en **React** con **Vite** y Tailwind.

Ambos módulos se comunican por HTTP. El backend expone endpoints para operaciones de juegos, usuarios, comentarios y actividad. El frontend consume estas APIs para presentar la interfaz al usuario.

2. Organización del repositorio

```
/gestor_videojuegos      # Proyecto Django principal
/frontend                # Aplicación React
/juegos, /usuarios, ...  # Apps Django con su lógica propia
/media                   # Archivos subidos (avatares)
/avatares                # Recursos de ejemplo
/docs                    # Documentación
```

Los ficheros `manage.py` y `requirements.txt` se encuentran en la raíz del repositorio.

Apps principales

- **usuarios**: Gestión de cuentas, perfiles, seguidores, bloqueos y favoritos.
- **juegos**: Integración con la API de IGDB, bibliotecas de usuarios y valoraciones.
- **comentarios**: Sistema de comentarios sobre juegos o noticias.
- **actividad**: Registro de logros y acciones de los usuarios.
- **diario**: Funciones de diario personal (por ejemplo registro de horas de juego).

Cada app sigue la estructura típica de Django: `models.py`, `views.py`, `serializers.py`, `urls.py` y un paquete `migrations`.

3. Backend

Configuración

El archivo `gestor_videojuegos/settings.py` define parámetros esenciales:

- Conexión a base de datos MySQL.

- Configuración de **Redis** como sistema de caché, usada para almacenar juegos y tokens de IGDB.
- Lista de aplicaciones instaladas y middleware.
- Permisos de CORS para permitir el acceso desde el frontend (Puerto 5173).

Rutas principales

`gestor_videojuegos/urls.py` agrupa las URLs de cada app:

```
urlpatterns = [
    path('api/usuarios/', include('usuarios.urls')),
    path('api/juegos/', include('juegos.urls')),
    path('api/comentarios/', include('comentarios.urls')),
    path('api/actividad/', include('actividad.urls')),
    path('api/diario/', include('diario.urls')),
]
```

Además se expone `forzar_cache_juegos` para lanzar manualmente la recopilación de juegos desde IGDB.

Integración con IGDB

El módulo `gestor_videojuegos/recopilar.py` contiene la lógica para descargar masivamente información de IGDB. Utiliza la caché de Redis para almacenar el progreso y reanudar descargas en caso de fallo. El proceso se ejecuta en un hilo aparte, activado desde `gestor_videojuegos/views.py`.

La app `juegos` ofrece endpoints para listar, buscar y obtener detalles de juegos. También permite gestionar bibliotecas de usuario y valoraciones.

Usuarios y perfiles

La app `usuarios` extiende el modelo de usuario mediante `Perfil`. Incluye avatar, biografía, seguidores y bloqueados. Se proveen vistas para registro, login, actualización de perfil, favoritos, seguir/bloquear usuarios, etc.

Comentarios y actividad

- **comentarios**: Permite a los usuarios dejar comentarios asociados a distintos objetos (por ejemplo juegos).
- **actividad**: Registra eventos y logros desbloqueados, permitiendo mostrar un historial de acciones en el frontend.

4. Frontend

Dentro de `frontend/` se encuentra la aplicación React. Usa Vite para el bundling y dispone de configuración de ESLint y Tailwind. El código fuente se organiza en `src/`, que contiene componentes, páginas y contexto global para la autenticación. Se conecta a la API del backend para mostrar juegos, perfiles y demás datos.

No se detallan todos los componentes por ser un tema más propio del desarrollo del cliente, pero su estructura sigue la convención de React y Vite.

5. Flujo de ejecución resumido

1. El usuario interactúa con el frontend (React) y realiza peticiones a la API.
2. Django procesa la solicitud mediante las vistas de cada app.
3. Si es necesario, el backend consulta IGDB (o la caché Redis) para obtener datos de juegos.
4. Las respuestas se envían en formato JSON al frontend para su visualización.

6. Requisitos y puesta en marcha

1. Instalar dependencias de Python:

   ```
   pip install -r requirements.txt
   ```

2. Configurar la base de datos MySQL y actualizar las credenciales en `settings.py`.

3. (Opcional) Instalar Redis para aprovechar la caché según las instrucciones del `README.md`.

4. Ejecutar migraciones:

   ```
   python manage.py migrate
   ```

5. Iniciar el servidor de desarrollo:

   ```
   python manage.py runserver
   ```

6. Para el frontend, desde `frontend/`:

   ```
   npm install
   npm run dev
   ```

7. Próximos pasos

Este manual es una referencia rápida. Cada app contiene lógica específica (modelos, vistas y tests) que conviene revisar para comprender por completo el flujo interno. En tareas futuras se documentará el código de forma más detallada. ## 8. Buenas prácticas de desarrollo

Entorno de desarrollo

Indica las versiones de Python y Node recomendadas en `README.md`. Utiliza un entorno virtual (`python -m venv venv`) y activa tailwind y eslint en el frontend.

Proceso de revisión de código

1. Crea una rama descriptiva por funcionalidad.
2. Abre un Pull Request en GitHub solicitando revisión.
3. Otro desarrollador revisará lint, pruebas y claridad del código antes de aprobar.

Refactorización

Refactoriza cuando el código sea difícil de mantener o duplicado. Asegura que las pruebas sigan pasando y realiza commits atómicos.

Manejo de errores y solicitudes

Registra los errores en la sección *issues* del repositorio. Etiqueta cada incidencia con su prioridad. Las nuevas características siguen el mismo flujo antes de su desarrollo.

Estándares de código

Sigue el PEP8 para Python y la guía de estilo de Airbnb para React. Mantén los comentarios breves y significativos.

Pruebas

Ejecuta `python manage.py test` para las apps de Django y `npm test` para el frontend. Añade pruebas unitarias y de integración por cada módulo nuevo.

Documentación

Actualiza este manual y los comentarios del código en cada cambio relevante. Usa docstrings en Python y JSDoc en JavaScript.

Implementación

El despliegue se realiza mediante contenedores Docker. Revisa los Dockerfiles del proyecto y sigue el entorno de producción indicado en `docs/GestorVideojuegos.pdf`.

Mantenimiento

Planifica actualizaciones de dependencias cada mes y revisa vulnerabilidades. Limpia la caché de Redis periódicamente.

Prácticas de desarrollo ágil

Se trabaja con iteraciones semanales utilizando un tablero Kanban en GitHub Projects. Las tareas se definen como *issues* y se priorizan en conjunto con el equipo.