Metodología de trabajo

1. Introducción

Este documento describe la **metodología de trabajo** seguida durante el desarrollo del proyecto, con el objetivo de garantizar una gestión ordenada y trazable de los cambios, incidencias y nuevas funcionalidades.

Para ello, se ha definido una estrategia clara basada en el uso de **issues, branches, commits, pull requests y GitHub Projects**, siguiendo buenas prácticas de desarrollo colaborativo y control de versiones.

2. Gestión de issues

La gestión del trabajo se ha realizado mediante **GitHub Issues**, utilizando distintas plantillas según el tipo de tarea a realizar. Esto ha permitido una correcta planificación, priorización y seguimiento del desarrollo.

2.1. Historias de usuario

Las funcionalidades principales se han definido mediante **historias de usuario**, creadas a partir de una plantilla específica.

Cada historia de usuario incluye:

- **Título** con un identificador único.
- **Descripción funcional** de la tarea.

Estas historias de usuario se han añadido al **tablero Kanban del proyecto**.

2.2. Issues generales

Para tareas que no encajan directamente en una historia de usuario (refactorizaciones, mejoras técnicas, tareas de mantenimiento, etc.), se ha utilizado una **plantilla de issue genérica**, que incluye una breve descripción del trabajo a realizar.

2.3. Bug reports

La gestión de errores se ha realizado mediante una plantilla específica de **bug report**, que incluye:

- Descripción del problema.
- Pasos para reproducir el error.
- Resultado esperado y resultado actual.
- Prioridad del bug.
- Información del entorno (si aplica).
- Posibilidad de adjuntar evidencias (logs, capturas, etc.).

3. GitHub Projects (Kanban)

Para el seguimiento visual del progreso del proyecto, se ha utilizado **GitHub Projects** con un tablero Kanban.

El tablero incluye las siguientes columnas de estado:

- `To Do`
- `In Progress`
- `In Review`
- `Integration`
- `Done`

Este tablero ha permitido tener una visión clara del estado de cada issue y del avance general del proyecto.

4. Estructura de ramas (Branching)

Se ha seguido una estrategia basada en **GitFlow adaptado**, simplificada para ajustarse al tamaño y necesidades del proyecto.

4.1. Ramas principales

Las ramas principales del repositorio son:

- `main`: Rama estable de producción. Contiene únicamente código probado y listo para su despliegue.
- `develop`: Rama de desarrollo. Integra las funcionalidades finalizadas antes de su paso a producción.

4.2. Ramas de trabajo

A partir de la rama `develop`, se han creado **ramas de trabajo** para cada issue, funcionalidad o corrección de bug.

Estas ramas se utilizan para desarrollar de forma aislada cada tarea, facilitando la revisión y evitando conflictos. Cuando se mergean sus cambios a develop en algunas ocasiones son borradas.

Convención de nombres

- Nombres en minúsculas.
- Uso de guiones (`-`) para separar palabras.
- Referencia al tipo de tarea.

Formatos utilizados:

- `feature/descripcion`
- `bug/descripcion`

Ejemplos:

```
feature/playlist-crud
bug/fix-rating-validation
```

5. Conventional Commits

Este proyecto sigue la especificación **Conventional Commits**, la cual proporciona una forma estándar de estructurar los mensajes de commit.

Existen **hooks** configurados en el repositorio para evitar realizar commits con mensajes incorrectos o contenido no deseado.

5.1. Convención de commits

Para mantener un historial de Git limpio y consistente, **todos los commits deben seguir esta convención**:

```
<type>: <short description>
```

5.2. Tipos permitidos

- **feat**: nueva funcionalidad
- **fix**: corrección de errores
- **docs** / **doc**: documentación
- **style**: cambios de formato, sin modificaciones en la lógica
- **refactor**: refactorización de código
- **perf**: mejora de rendimiento
- **test** / **tests**: añadir o modificar tests
- **build**: cambios en el sistema de build o dependencias
- **ci**: integración continua
- **chore**: tareas de mantenimiento
- **sec**: mejoras de seguridad

5.3. Ejemplos de commits válidos

```
feat: add login endpoint
fix: correct user password validation
docs: update README with new instructions
style: format code with prettier
refactor: optimize database queries
perf: improve response time
test: add unit tests for auth
ci: update GitHub Actions workflow
chore: remove deprecated package
sec: hash passwords with bcrypt
```

6. Pull Requests

Las **Pull Requests (PRs)** se utilizan para integrar el trabajo de las ramas de desarrollo en `develop` y, posteriormente, de `develop` en `main`.

Cada PR:

- Está asociada a una issue concreta.
- Sigue la misma convención de nombres que la rama.
- Permite la revisión del código antes de su integración.

Ejemplos de títulos de PR:

```
feature/playlist-crud
bug/fix-rating-validation
```

Antes de realizar el merge, se verifica que:

- El código compila correctamente.
- Los tests pasan satisfactoriamente.
- Se cumplen las normas de estilo y validación del proyecto.

7. Conclusión

La metodología de trabajo adoptada ha permitido:

- Mantener un desarrollo ordenado y trazable.
- Facilitar la colaboración entre los miembros del equipo.
- Reducir errores mediante revisiones y pruebas continuas.
- Garantizar la estabilidad del código en producción.

El uso de una estrategia de branching clara, junto con buenas prácticas en la gestión de issues, commits y pull requests, ha sido clave para el correcto desarrollo del proyecto.