**KarenFlix**

**David Adolfo Gomez Uribe**

**Joan Sebastian Omaña**

**GRUPO U2**

**DOCENTES INVOLUCRADOS:**

**Juan Carlos Mariño Morantes**

**CAMPUSLANDS**

**SALON ARTEMIS  
RUTA BackEnd NODE.js**

**FLORIDABLANCA**

**KarenFlix**

1. **SITUACIÓN PROBLEMA**

En la actualidad, los fanáticos de la cultura geek —incluyendo el cine, los animes, las series y los cómics— carecen de una plataforma especializada que les permita **registrar, calificar y rankear** sus contenidos favoritos de manera confiable y organizada. Aunque existen servicios globales como *IMDb* o *Rotten Tomatoes*, estos carecen de un enfoque específico en la comunidad geek y, en muchos casos, presentan limitaciones en la interacción entre usuarios, la personalización de rankings y el reconocimiento de subcategorías relevantes (por ejemplo: *anime shōnen*, *universos cinematográficos de superhéroes* o *sagas fantásticas*).

La necesidad surge porque los usuarios buscan un espacio donde no solo puedan **acceder a reseñas**, sino también **crear y validar contenido propio**, con un sistema de calificaciones más justo que combine puntuaciones numéricas con la relevancia social (likes/dislikes a reseñas, reputación del usuario y fecha de publicación). A su vez, se requiere que la plataforma diferencie entre **roles de usuario y administrador**, para asegurar un control de calidad sobre las publicaciones y categorías creadas.

Por otro lado, la mayoría de aplicaciones existentes no permiten un **control comunitario dinámico** ni cuentan con un **ranking ponderado que integre múltiples factores** (calificaciones, popularidad, interacciones), lo cual genera rankings sesgados o manipulables. Esto limita la confiabilidad de la información para los usuarios que buscan descubrir nuevas películas, animes o series basados en la experiencia colectiva.

Frente a esta problemática, se identifica la **oportunidad de desarrollar una aplicación full-stack** que integre un backend robusto en **Node.js + Express** con persistencia en **MongoDB** y un frontend accesible desde cualquier navegador mediante **HTML, CSS y JavaScript puro**. Esta solución permitirá a los usuarios registrarse, autenticarse de forma segura, participar en la construcción de reseñas y rankings, y a los administradores gestionar categorías y validar nuevas entradas.

De esta manera, el proyecto no solo responde a una necesidad de la comunidad geek, sino que además fomenta la interacción social, el acceso a información confiable y la creación de un ecosistema participativo, seguro y transparente.

1. **LEVANTAMIENTO DE REQUERIMIENTOS**
2. **REQUERIMIENTOS**
   1. **Requerimientos Funcionales**
   2. **Gestión de usuarios**
      1. RF1. El sistema debe permitir el **registro de nuevos usuarios** con datos básicos (nombre, correo, contraseña).
      2. RF2. El sistema debe permitir el **inicio de sesión** mediante autenticación JWT.
      3. RF3. El sistema debe manejar los roles **de usuario y administrador**.
      4. RF4. El sistema debe permitir a los usuarios editar y eliminar su cuenta.
      5. RF5. Los administradores deben poder **gestionar usuarios** (bloquear, dar de baja, asignar roles).
   3. **Gestión de películas, animes y series**
      1. RF6. Los usuarios podrán **sugerir nuevas películas, animes o series** con atributos mínimos: título, descripción, categoría, año e imagen opcional.
      2. RF7. Solo los administradores podrán **aprobar, editar y eliminar** películas/series.
      3. RF8. El sistema debe **validar que no existan títulos repetidos**.
   4. **Gestión de reseñas y ratings**
      1. RF9. Los usuarios podrán **crear reseñas** con título, comentario y calificación numérica (1–10).
      2. RF10. Los usuarios podrán **editar y eliminar** sus propias reseñas.
      3. RF11. Los usuarios podrán **dar like o dislike** a reseñas de otros (no a las propias).
      4. RF12. El sistema debe calcular un **ranking ponderado** de películas basado en calificaciones, likes/dislikes y fecha de reseña.
   5. **Gestión de categorías**
      1. RF13. El sistema debe permitir el **CRUD de categorías** (crear, leer, actualizar, eliminar).
      2. RF14. Solo los administradores podrán **gestionar categorías**.
   6. **Listados y búsqueda**
      1. RF15. El sistema debe permitir **listar películas por ranking y popularidad**.
      2. RF16. El sistema debe permitir **filtrar películas por categoría**.
      3. RF17. El sistema debe mostrar la **vista de detalle de cada película**, con sus reseñas asociadas.
   7. **Seguridad y control**
      1. RF18. El sistema debe implementar **limitación de peticiones** con express-rate-limit.
      2. RF19. El sistema debe usar **validaciones en todos los endpoints** con express-validator.
      3. RF20. El sistema debe usar **transacciones en MongoDB** en operaciones críticas (ejemplo: creación de reseñas con rating inicial).
3. **Requerimientos No Funcionales** 
   * 1. **Arquitectura y desarrollo**
        1. RNF1. El backend debe estar desarrollado en **Node.js + Express.**
        2. RNF2. El sistema debe usar **MongoDB** con el driver oficial (no mongoose).
        3. RNF3**.** El código debe estar organizado en una arquitectura modular **(/models, /controllers, /routes, /middlewares, /services, /config, /utils).**
        4. RNF4. El sistema debe estar versionado siguiendo semver.
     2. **Rendimiento y escalabilidad**
        1. RNF5. El sistema debe poder manejar al menos **100 peticiones** concurrentes sin degradar significativamente el rendimiento.
        2. RNF6. El sistema debe ser escalable para agregar nuevas funcionalidades sin romper la arquitectura base.
     3. **Seguridad**
        1. RNF7. Todas las contraseñas deben almacenarse usando **bcrypt con salt.**
        2. RNF8. El sistema debe usar **JWT** con expiración configurable.
        3. RNF9. El sistema debe validar todas las entradas para **evitar inyecciones SQL/NoSQL, XSS y CSRF.**
     4. **Interoperabilidad y documentación**
        1. RNF10. El sistema debe exponer la API de forma pública con documentación en **Swagger.**
        2. RNF11**.** El sistema debe permitir **CORS** para que el frontend pueda consumir los endpoints.
     5. **Frontend**
        1. RNF12. El frontend debe estar desarrollado con HTML, CSS y JS puro.
        2. RNF13. La interfaz debe ser responsive y accesible desde dispositivos móviles y escritorio.
        3. RNF14. El frontend debe mostrar mensajes claros de error y validación provenientes del backend.
     6. **Mantenibilidad y pruebas**
        1. RNF15. El sistema debe contar con un manejo centralizado de errores.
        2. RNF16. El sistema debe usar códigos HTTP correctos en todas las respuestas.
        3. RNF17. El sistema debe permitir pruebas unitarias e integrales en endpoints críticos.

Aquí se deben documentar todos los requerimientos no funcionales en algún formato que usted elija.

1. **HISTORIAS DE USUARIO CON CRITERIOS DE ACEPTACIÓN**

|  | | | | |
| --- | --- | --- | --- | --- |
| **HISTORIA DE USUARIO** | | | | |
| **Prioridad: Alta** | | | | |
| **CÓDIGO DEL**  **REQUERIMIENTO:** | | RF03 | **Actor** | Administrador |
| **NOMBRE DEL**  **REQUERIMIENTO** | | **Actualización de la lista de materiales** | | |
| **Descripción** | | | | |
| Como administrador puedo modificar la lista de materiales por persona después de haber seleccionado el tipo de persona con el fin de permitir el ajuste posterior de uno o varios elementos que se consideren realizar. | | | | |
| **Funcionalidad** | | | | |
| Permitir al administrador modificar la lista de materiales asociada a una persona una vez que haya seleccionado el tipo de persona. Esto significa que el administrador tiene la capacidad de agregar, editar o eliminar los materiales disponibles para cada persona en particular. Estas modificaciones bridan flexibilidad al sistema, permitiendo ajustar y personalizar la lista de materiales según las necesidades y características específicas de cada persona registrada. | | | | |
| **Criterios de aceptación** | 1. El sistema debe permitir que el administrador ….  2. El sistema debe permitir que el …  3. El sistema …. | | | |
| **Restricciones** | | | | |
| El administrador solo puede modificar la lista de materiales de una persona si el tipo de persona ya ha sido seleccionado previamente. | | | | |
|  | | | | |

1. **METODOLOGÍA**

Se debe mencionar la metodología o el orden en qué se realizarán las actividades o tareas que permiten dar cumplimiento al proyecto. Si se va a seguir una metodología de desarrollo de Software adicional (por ejemplo, metodologías Ágiles), debe estar especificada en esta sección, junto con los roles y lo que exige la definición de esta.

1. **EVIDENCIA DE PLANTEAMIENTO DE PLATAFORMA DE TRABAJO**

Acá se debe documentar toda la evidencia de trabajo colaborativo con los siguientes elementos:

* Link del repositorio donde se evidencia el trabajo colaborativo con el correcto uso de roles, ramas y conventional commit.
* Link de los videos que se grabaron en las reuniones realizadas (Sprint planning, Daily Stand Up, Sprint Retrospective)
* Evidencia (capturas) del tablero Scrum utilizado en todas las etapas donde se visualicen todas las tareas, requerimientos e historias de usuario documentadas allí con responsables, tiempos, priorización, y demás requerimientos que se indiquen para las tareas.
* Documentación de los resultados y funcionamiento del producto final resaltando tecnologías utilizadas y cumplimiento de trabajo basado en las tareas planteadas.

**7. CONCLUSIONES**

* Conclusiones generales del documento
* Conclusiones de la reunión de retrospectiva del Sprint