**Gestor de Portafolio de Proyectos Freelance**

**Brian Fair Suarez Porras – Product Owner**

**Johan Sebastian Omaña Suarez - Developer**

**David Adolfo Gomez Uribe - SRUM Master**

**E3**

**GRUPO # 1**

**DOCENTES INVOLUCRADOS:**

**Juan Carlos Mariño Morantes**

**CAMPUSLANDS**

**SALON ARTEMIS  
RUTA BackEnd NODE.js**

**FLORIDABLANCA**

**2025**

**Gestor de Portafolio de Proyectos Freelance**

1. **SITUACIÓN PROBLEMA**

En el entorno actual del trabajo freelance, los profesionales independientes enfrentan desafíos significativos para gestionar de forma eficiente sus proyectos, clientes, contratos, entregables y finanzas. La mayoría de freelancers deben recurrir a múltiples herramientas dispersas (hojas de cálculo, procesadores de texto, aplicaciones de notas, etc.) para mantener el control de sus actividades diarias. Esto incrementa el riesgo de pérdida de información, duplicidad de tareas, y una visión fragmentada del estado de sus operaciones.

Por esta razón, surge la necesidad de una herramienta centralizada que les permita administrar de forma ordenada y eficiente cada componente de su portafolio de trabajo. Este proyecto busca desarrollar una aplicación de línea de comandos (CLI) construida con Node.js, que integre funcionalidades específicas para registrar clientes, generar propuestas, administrar contratos, hacer seguimiento a entregables y llevar control financiero, todo en un entorno amigable y automatizado.

La solución propuesta responde a una necesidad real del mercado freelance, y se apoya en principios de ingeniería de software moderna, como la Programación Orientada a Objetos, principios SOLID, patrones de diseño y transacciones en bases de datos, garantizando robustez, mantenibilidad y escalabilidad de la aplicación.

1. **LEVANTAMIENTO DE REQUERIMIENTOS**
2. **REQUERIMIENTOS**
   1. **Requerimientos Funcionales**

## REQUERIMIENTOS FUNCIONALES

| **Código** | **Requerimiento Funcional** |
| --- | --- |
| RF01 | El sistema debe permitir crear, listar, actualizar y eliminar clientes. |
| RF02 | El sistema debe permitir crear propuestas para clientes con descripción, precio, plazos y estado (pendiente, aceptada, rechazada). |
| RF03 | Al aceptar una propuesta, se debe generar automáticamente un proyecto. |
| RF04 | El sistema debe permitir la gestión completa (CRUD) de proyectos. |
| RF05 | Cada proyecto debe estar asociado a un cliente, contrato y entregables. |
| RF06 | El sistema debe permitir cambiar el estado de un proyecto (activo, pausado, finalizado, cancelado). |
| RF07 | Se debe permitir registrar avances de cada proyecto. |
| RF08 | El sistema debe generar y asociar contratos a proyectos con condiciones, fechas y valores. |
| RF09 | Cada proyecto debe contener entregables con fechas límite y estados definidos (pendiente, entregado, aprobado, rechazado). |
| RF10 | El sistema debe permitir eliminar o cambiar el estado de un entregable usando transacciones. |
| RF11 | El sistema debe registrar ingresos y egresos financieros asociados a proyectos. |
| RF12 | El sistema debe calcular balances por cliente o por fecha. |
| RF13 | Las transacciones deben evitar la duplicidad y pérdida de consistencia. |

## REQUERIMIENTOS NO FUNCIONALES

| **Código** | **Requerimiento No Funcional** |
| --- | --- |
| RNF01 | La aplicación debe estar desarrollada en Node.js aplicando Programación Orientada a Objetos. |
| RNF02 | Deben aplicarse principios SOLID y al menos dos patrones de diseño. |
| RNF03 | Se deben usar librerías npm para mejorar la experiencia por consola (ej. chalk, inquirer, dotenv). |
| RNF04 | Los datos deben persistirse en MongoDB usando el driver oficial (mongodb), no Mongoose. |
| RNF05 | Se deben utilizar transacciones reales para operaciones críticas. |
| RNF06 | El código debe estar organizado en carpetas como /models, /services, /commands, /config, /utils. |
| RNF07 | El repositorio debe incluir .gitignore, README completo y usar Conventional Commits. |
| RNF08 | La colaboración debe ser evidente si es un equipo (uso de ramas, roles, commits claros). |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HISTORIAS DE USUARIO CON CRITERIOS DE ACEPTACIÓN | | | | |
| **HISTORIA DE USUARIO: S0-HU1** | | | | |
| **Prioridad: Alta** | | | | |
| **CÓDIGO DEL**  **REQUERIMIENTO:** | | RNF06 | **Actor** | Desarrolador |
| **NOMBRE DEL**  **REQUERIMIENTO** | | **Preparar entorno base del proyecto** | | |
| **Descripción** | | | | |
| Como desarrollador quiero establecer la estructura base del proyecto para trabajar ordenadamente desde el primer Sprint, siguiendo buenas prácticas de desarrollo. | | | | |
| **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** | Se crea la estructura base de carpetas: /models, /services, /commands, /config, /utils, /src.  Se crea y configura el archivo .gitignore (excluye node\_modules, .env, y carpetas temporales).  Se inicializa el proyecto con npm init y se configura el archivo package.json.  Se instala y configura dotenv, chalk inquirer boxen lodash mongodb  Se agrega un archivo README.md con una descripción breve del proyecto y cómo ejecutar la app.  Se crea un commit inicial usando convención feat: estructura base del proyecto. | | | |
| **Restricciones** | | | | |
| Seguir la convención de carpetas definida en la documentación del proyecto.  Añadir un script start en package.json para ejecutar el CLI fácilmente.  Los mensajes de consola deben ser claros y estilizados con chalk. | | | | |
|  | | | | |

1. **METODOLOGÍA**

Para el desarrollo del presente proyecto se adoptará la metodología ágil SCRUM, con el objetivo de garantizar una gestión ordenada, iterativa e incremental del producto, favoreciendo la colaboración, la transparencia y la mejora continua.

SCRUM nos permitirá dividir el trabajo en ciclos cortos llamados Sprints, cada uno con objetivos definidos y entregables específicos. Esta metodología nos facilitará adaptarnos a cambios, organizar las tareas con claridad, y asegurar que el equipo mantenga una comunicación constante y una evolución controlada del desarrollo.

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