# Proyecto Prueba de Tester 1

## Plan de pruebas o ataque

**Preparado por:** Noé Jonatán Gómez Hernández

## Introducción

El plan de prueba está diseñado como una línea de base para identificar lo que se considera dentro y fuera del alcance de las pruebas y cuales son los riesgos y suposiciones.

## Recursos

|  |  |
| --- | --- |
| **Tester** | **% Participación** |
| Noé Jonatán Gómez Hernández | 100% |
| Pablo Joel García Escobar | 50% |

## Alcance

Las pruebas incluyen todas las funcionalidades nuevas adaptadas, las funcionalidades de alto riesgo que pueden ser de regresión, así como también las pruebas de rendimiento.

La localización o adaptación del sistema para un idioma o función numérica en específico podrá ser parte de este proyecto.

Las pruebas de regresión manual de baja prioridad se ejecutarán en tiempo si es permitido.

## Fuera del alcance

Las pruebas de ataque inmersivo directo quedarán como sub-contratación ya que no forman parte de este plan de pruebas, posteriormente se verificará el desarrollo de la aplicación para corroborar si efectivamente son necesarias en pro de la monetización efectiva del proyecto, ya que actualmente el desarrollo del proyecto es básico.

## Criterios de aceptación

1. No se debería aceptar el ingreso de números decimales o fraccionarios.
2. No se debería aceptar el ingreso de letras o símbolos.
3. El único botón deberá funcionar para incrementar intentos y guardar datos en la sesión, no para cualquier otro evento.

## Pruebas de rendimiento

Las pruebas de rendimiento estarán constituidas inicialmente de 4 pruebas manuales según la funcionalidad del proyecto componiéndose de la siguiente manera:

1. Ingreso de números decimales para comprobar el cumplimiento del requerimiento de la prueba inicial. En caso de permitir este fallo, se procederá a realizar una regresión respectiva.
2. Ingreso de caracteres ajenos a números enteros. Todo carácter sea símbolo o letra no debería poderse ingresar al cuadro de texto, por lo que, si se produce este fallo, se procederá a realizar la regresión respectiva.
3. Validación del botón “Ingresar número aleatorio”, ya que dicho botón es de tipo submit, se prevé que no falle al momento de enviar el valor ingresado, por lo que se le pondrá a prueba presionándolo las veces necesarias para provocar un fallo, si este es el caso se procederá a realizar la debida regresión y seguimiento de este, para poder solventarlo completamente.
4. Recargo de la página para llegar a buscar el fallo, si en alguna de las oportunidades se llega a efectuar este evento, se procederá a revisar y enviar el respectivo detalle al departamento de programación.

También se prevé el uso de Herramientas de virtualización: Puede ser útil utilizar herramientas de virtualización, como VirtualBox, Docker, etc. para configurar rápidamente entornos de prueba y hacer pruebas de compatibilidad con diferentes sistemas operativos y navegadores. Así como herramientas de análisis de rendimiento de manera automática: puede ser necesario utilizar herramientas de análisis de rendimiento, como JMeter, LoadRunner, etc.

## Pruebas de Aceptación

Ya que el proyecto de software tiene pocas funcionalidades implementadas, estas se han considerado completamente en las pruebas de rendimiento para no retrasar el desarrollo gradual del equipo.

## Infraestructura

Para poder determinar la infraestructura necesaria para este proyecto en función del plan de pruebas, necesitaríamos más detalles sobre el desarrollo incremental del proyecto y sobre los requisitos del sistema a largo plazo. Sin embargo, a continuación, se proporciona una lista general de las posibles herramientas y tecnologías que podrían utilizarse para establecer una infraestructura de pruebas:

* Framework de pruebas: Se puede elegir entre varios frameworks de pruebas de JavaScript, como Jest, Mocha, Jasmine, etc. Estos frameworks proporcionan una estructura para escribir, organizar y ejecutar pruebas de manera eficiente.
* Herramientas de automatización de pruebas: Selenium WebDriver, Cypress, Puppeteer, permiten simular la interacción del usuario con la aplicación y realizar pruebas automatizadas.
* Servidor de integración continua: Puede ser necesario configurar un servidor de integración continua, como Jenkins, CircleCI, TravisCI, etc. para ejecutar automáticamente las pruebas cada vez que se realiza una nueva compilación.
* Herramientas de análisis de código: Las herramientas de análisis de código, como SonarQube, pueden ser útiles para detectar posibles problemas de calidad de código.
* Herramientas de seguimiento de errores: Es posible que se necesite una herramienta para realizar el seguimiento de errores y problemas encontrados durante las pruebas, como JIRA, Bugzilla, Mantis, etc.
* Herramientas de gestión de pruebas: TestRail, puede ayudar a planificar, ejecutar y rastrear pruebas productivamente.
* Entorno de pruebas: Es necesario un entorno de pruebas separado para evitar que las pruebas afecten al entorno de producción. Esto incluye servidores de bases de datos de prueba como Oracle, servidores de aplicaciones de prueba como los de Google pero esto dependerá de lo que se desee alcanzar profundamente en el proyecto.

## Riesgos

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Descripción del riesgo | Probabilidad (1-5) | Impacto (1-5) | Severidad (1-10) | Plan de mitigación |
| 1 | Errores de sintaxis o lógica en el código | 3 | 3 | 6 | Realizar revisiones y pruebas de código para detectar y corregir errores antes del lanzamiento. |
| 2 | Dificultades en la implementación de las funcionalidades | 2 | 4 | 8 | Realizar pruebas exhaustivas de todas las funcionalidades del juego para asegurarse de que funcionen correctamente. Realizar pruebas de integración para comprobar que todas las partes del juego funcionen juntas correctamente. |
| 3 | Problemas de compatibilidad en diferentes navegadores y dispositivos | 4 | 3 | 9 | Realizar pruebas en diferentes navegadores y dispositivos para asegurarse de que el juego funcione correctamente en todos ellos. Utilizar herramientas y técnicas para la detección y solución de problemas de compatibilidad. |
| 4 | Errores en la validación de entrada de usuario | 2 | 4 | 8 | Validar cuidadosamente la entrada del usuario para evitar errores. Utilizar técnicas como la validación del lado del cliente y del servidor para asegurarse de que los datos sean correctos antes de procesarlos. |
| 5 | Dificultades en la localización y traducción del juego | 3 | 2 | 6 | Planificar la localización y traducción del juego desde el principio. Utilizar herramientas y técnicas para facilitar el proceso de localización y traducción. Realizar pruebas de localización para asegurarse de que todas las traducciones sean correctas. |
| 6 | Problemas de seguridad del juego | 2 | 5 | 10 | Realizar pruebas de seguridad para detectar posibles vulnerabilidades y solucionarlas antes del lanzamiento. Implementar medidas de seguridad adecuadas, como la autenticación y el cifrado de datos sensibles. |
| 7 | Problemas de rendimiento y escalabilidad | 3 | 4 | 12 | Realizar pruebas de rendimiento para asegurarse de que el juego se ejecute sin problemas en diferentes condiciones. Optimizar el código y la arquitectura del juego para mejorar el rendimiento y la escalabilidad. Utilizar técnicas como el almacenamiento en caché y la distribución de carga para mejorar el rendimiento y la escalabilidad. |

Es importante recordar que este plan es solamente una posible aproximación ya que los fallos y riesgos específicos y su gravedad pueden variar dependiendo de las características y requisitos del proyecto. El equipo de prueba y desarrollo debe trabajar juntos para identificar y mitigar los riesgos adecuados para este proyecto en particular.