**Tienda de mascotas Proyecto**



**Plan de Pruebas**

**Preparado por**: Ismael Carvajal

**Historial de versiones**

|  |  |  |
| --- | --- | --- |
| Fecha | Autor | Comentario Versión |
| 03/03/2022 | Mario Hernández | El plan de pruebas se basa en los requerimientos Creación  iniciales |

**Índice**

1. Introducción

2. Contextualización

3. Recursos

4. Alcance

5. Fuera de alcance

6. Pruebas de rendimiento

7. Pruebas de aceptación (UAT)

8. Infraestructura

9. Suposiciones

10. Riesgos

**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 cuáles son los riesgos y suposiciones.

En este plan de pruebas se va a evaluar las funcionalidades del módulo ‘**Pet-Todo acerca de las mascotas’** para verificar y validar que el producto satisfaga las expectativas del cliente y de los usuarios. Así como también reducir la probabilidad de que aparezcan defectos en producción.

**Contextualización**

La empresa Pet Store quiere construir el backend de un e-comerce para ofertar la adopción de mascotas.

La aplicación tiene 3 módulos principales, pero en esta oportunidad se probará el módulo 1 que se describe a continuación:

1. Pet – Todo acerca de las mascotas:

* Crear una mascota
* Añadir imágenes de la mascota
* Actualizar información de una mascota
* Filtrar por mascota (estado – id)
* Eliminar una mascota

**Recursos**

|  |  |
| --- | --- |
| **Tester** | **% Participación** |
| Karina Ramos (QA de performance) | 50 % |
| Mario Hernández (QA manual) | 100% |
| Ismael Carvajal (QA automation) | 50% |

**Alcance**

Las pruebas incluyen todas las funcionalidades del módulo ‘**Pet-Todo acerca de las mascotas’.**  En este módulo se realizarán pruebas funcionales y no funcionales para los distintos niveles de prueba y además en cuanto a pruebas no funcionales se aplicarán pruebas para validar el rendimiento y usabilidad de dicho componente.

Los EndPoints que se validarán en dicho modulo son:

* POST: {{URL}} [/pet/{petId}/uploadImage](https://petstore.swagger.io/#/pet/uploadFile)
* POST: {{URL}} [/pet](https://petstore.swagger.io/#/pet/uploadFile)
* PUT: {{URL}} [/pet](https://petstore.swagger.io/#/pet/uploadFile)
* GET: {{URL}}/[pet/findByStatus](https://petstore.swagger.io/" \l "/pet/findPetsByStatus)
* GET: {{URL}}[/pet/{petId}](https://petstore.swagger.io/#/pet/getPetById)
* POST: {{URL}}[/pet/{petId}](https://petstore.swagger.io/#/pet/getPetById)
* DELETE: {{URL}}[/pet/{petId}](https://petstore.swagger.io/#/pet/getPetById)

Además, se verificarán los Status code que se encuentran descritos en el Swagger por cada Endpoints que se pruebe en Postman que en este caso será el ambiente de pruebas.

Las pruebas de rendimiento serán desarrolladas por el QA de performance, la prueba de usabilidad será desarrollada por el QA manual y las pruebas funcionales serán repartidas entre el QA manual y el QA automation.

**Fuera del Alcance**

No se encuentra dentro del alcance de este plan de pruebas los componentes de **‘Store-Órdenes de compra’** ni tampoco el de ‘**User-Operaciones de usuario’**.

Por otro lado, no se aplicarán pruebas de seguridad de acuerdo a las directrices que fueron entregadas al equipo y que se pueden confirmar en los requerimientos.

**Estrategia**

El equipo de desarrollo implementará pruebas unitarias y de integración como parte de su pipeline.

El equipo de testing realizará pruebas funcionales y no funcionales de forma manual y automatizada. Cada vez que una nueva versión llegue a testing se hará una prueba de regresión sobre aquellos escenarios que se consideren de prioridad alta.

Es importante mencionar que las pruebas manuales serán desarrolladas en Postman y las pruebas automatizadas usarán Java y Cucumber para escribirlas.

Dentro de las pruebas no funcionales se ejecutarán pruebas de performance, portabilidad, compatibilidad y usabilidad.

Al finalizar cada sprint se realizarán pruebas de aceptación de usuario. Al mismo tiempo, se aprovecharán esas reuniones para hacer pruebas de usabilidad.

Las pruebas de aceptación serán realizadas y coordinadas con las oficinas centrales. Los usuarios se escogerán según su nivel de experiencia en las áreas del sistema.

**Criterios**

Las pruebas del equipo de testing comenzarán cuando hayan pasado satisfactoriamente las pruebas unitarias.

El criterio de finalización de las pruebas será cuando se hayan ejecutado todas las pruebas planificadas o, si el tiempo apremia, al menos se hayan ejecutado los casos de prueba de prioridad alta y media siempre y cuando se le haya comunicado al product Owner.

El criterio para pasar a producción será que no haya errores bloqueantes o críticos sin resolver.

**Infraestructura**

Nuestros ambientes de prueba tendrán las siguientes configuraciones de Hardware:

* 1 portátil MAC para el probador que hará las pruebas manuales
* 1 computador de escritorio HP con Windows 10 para el probador que realizará las pruebas automatizadas
* 1 notebook Samsung con ubuntu 22.04 LTS para el probador que realizará las pruebas de performance.

Por otro lado, los ambientes de prueba tendrán las siguientes configuraciones de Software:

* Maven con una versión 3.8.6: Es un gestor de dependencias.
* Git con una versión 2.36.1: Es un sistema de versionamiento
* Java con la versión 1.8.0\_333: Es un lenguaje de programación
* Aplicación Web de Pet Store que funciona en la siguiente dirección URL/pet.
* Postman con la versión 9.4: Es un probador de API’s
* Karate con la versión 2.2.3: Es un framework de automatización de pruebas
* Rest Assured con la versión 3.0.2.3: Es un framework escrito en Java y diseñado para simplificar las pruebas sobre servicios basados en REST.

**Entregables**

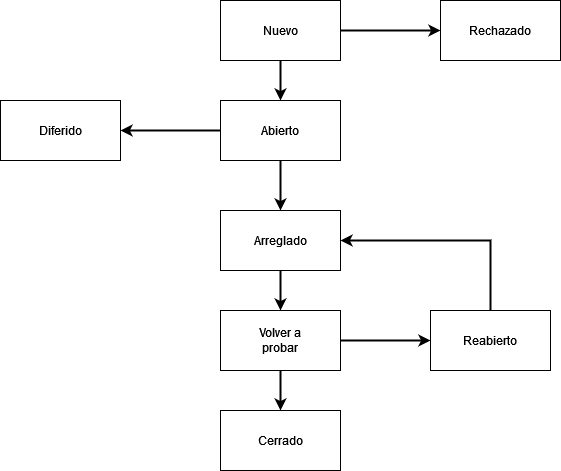
Los entregables que se generarán son:

* Plan de pruebas
* Lista de escenarios de pruebas ejecutadas
* Informe de estado al finalizar cada sprint
* Reporte de defectos

**Gestión de incidentes**

**(Se va a reportar por tal herramienta y se va a reportar al desarrollador)**

Se utilizará el siguiente workflow para la gestión de defectos:



**Suposiciones**

La severidad será asignada por el tester y la prioridad por el product owner.

**El core review de los desarrolladores:** La pruebas unitarias deben abarcar más del 90%.

Las severidades que se utilizarán serán:

• ***Bloqueante:*** cuando una funcionalidad no se pueda utilizar y no haya otra forma de realizar esa misma acción.

***• Crítica:*** cuando una funcionalidad no se pueda utilizar pero exista un camino alternativo para realizar eso mismo.

***• Alta:*** cualquier defecto sobre una funcionalidad prioritaria del sistema o que tiene impacto secundario en una funcionalidad prioritaria del mismo.

***• Medio:*** cualquier defecto que impacte en una funcionalidad secundaria.

• ***Bajo:*** cualquier defecto cosmético o errores de ortografía.

**Riesgos**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Riesgos** | **Probabilidad**  **(1-5)** | **Impacto**  **(1-5)** | **Severidad**  **(Prob\*Impct)** | **Plan de Mitigación** |
| 1 | Retrasos en la implementación de las funcionalidades. | 2 | 5 | 10 | Evaluar el avance del desarrollo de las funcionalidades y re-planificar acorde al avance de ser necesario. |
| 2 | Los usuarios no están listos para las pruebas de aceptación (UAT) | 1 | 5 | 5 | Coordinar con las oficinas centrales la selección temprana de los usuarios. |
| 3 | La arquitectura del sistema podría no ser adecuada para algunos requisitos no  funcionales. | 2 | 4 | 8 | Evaluar los requisitos funcionales constantemente dentro de la arquitectura del sistema para su mayor control. |
| 4 | Los tiempos de respuestas de la aplicación podrían ser inadecuados | 3 | 3 | 9 | Hacer un control por parte del probador de performance para mantener tiempos de respuesta optimos. |