Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Estudio del control de calidad en procesos de desarrollo de software. Aplicación práctica de pruebas automatizadas a un sitio web de comercio electrónico

---

Alumno: Elia Parra Valverde

Tutor: Miguel Ángel Martínez Prieto

Trabajo Fin de Grado

Estudio del control de calidad en procesos de desarrollo de software. Aplicación práctica de pruebas automatizadas a un sitio web de comercio electrónico

Elia Parra Valverde

20 de julio de 2020

Índice general

Índice de tablas

Índice de figuras

Dedicado a
Jorge, por todo su apoyo,
cariño y ayuda.

Agradecimientos

En este tramo final de mis estudios en el Grado en Ingeniería Informática de Servicios y Aplicaciones, en la Escuela de Ingeniería Informática de Segovia, me gustaría hacer constar mi agradecimiento a todas las personas que me han acompañado en este largo camino, que culmina con la presentación de este proyecto.

En primer lugar, me gustaría agradecer a mis profesores su esfuerzo y su dedicación, pues me han permitido superar los restos que esta titulación plantea, que no son pocos ni pequeños. En particular, me gustaría dar las gracias a mi tutor de TFG, Miguel Ángel Martínez Prieto, por su atención, su paciencia y su continuo interés, pues ha contribuido en gran medida a que este proyecto se haya podido completar con éxito.

También quiero recordar y agradecer a mis compañeros de clase y amigos, por los buenos ratos (y los malos también) que hemos compartido en estos años. A mi familia, especialmente a mis hermanas, por su cariño y su confianza.

Y por último, a Jorge, por su apoyo, siempre.

Resumen

En la actualidad, una de las mayores prioridades en el ámbito del desarrollo de software es entregar al cliente productos de calidad. Es decir, productos que maximicen el valor y la satisfacción del cliente, al tiempo que minimizan la cantidad de fallos que puedan contener. Por ello, el control y aseguramiento de la calidad del software es una actividad que ha ganado peso en los últimos años, estando presente desde el comienzo mismo del proyecto, y suponiendo cerca de la mitad del esfuerzo necesario para llevarlo a cabo. Sin embargo, una gran parte de este esfuerzo, que se realiza de forma manual, podría ser automatizado para mejorar la eficiencia del proceso y proporcionar un mejor resultado al final del mismo.

En este proyecto, se lleva a cabo un acercamiento a la gestión de la calidad en el entorno del desarrollo de software, con el fin de poner de relieve su importancia y caracterizar sus principios, procesos y técnicas más relevantes. Además, se propone un sistema de automatización de pruebas sobre un sitio web de comercio electrónico, con el objetivo de demostrar algunos de los principios expuestos de forma teórica y servir como ejemplo de aplicación de los procesos de aseguramiento de la calidad en un proyecto de estas características. Finalmente, se ha desarrollado una aplicación web para la gestión de las pruebas implementadas, cuyo fin es facilitar la interacción con el sistema de automatización construido a través de una interfaz de usuario adecuada.

Palabras claves: Pruebas de software, Automatización de procesos, Control de calidad, Aseguramiento de calidad

Capítulo 1

Introducción al proyecto.

Durante las últimas décadas, se ha producido un intenso proceso de informatización de la sociedad, especialmente a partir de la segunda mitad de la década del 2000. La explosión tecnológica provocada por la aparición de los *smartphones*, y la estandarización y difusión de modernas y más potentes tecnologías de comunicación han supuesto la democratización del acceso y el uso de aplicaciones informáticas en todos los aspectos de la vida diaria.

Nunca antes se ha utilizado tanto y de maneras tan diversas los recursos software disponibles, y el hardware, a su vez, es un factor cada vez menos limitante, gracias al crecimiento de los servicios en la nube (*Cloud Services*). Del código específico para ser ejecutado en un hardware determinado, hemos pasado a servicios genéricos multiplataforma, gracias a la estandarización, abstracción y establecimiento de capas (también software) que enmascaran la infraestructura subyacente y permiten el acceso a cualquier usuario sin importar los recursos informáticos con que cuente.

De la misma manera, esta diversidad ha supuesto un enorme incremento en el volumen y la complejidad de los productos software, tanto por su propia funcionalidad como por las dependencias que existen entre dicho producto y otros recursos, como *frameworks* y APIs. El desarrollo, inicialmente en manos de una única persona con conocimientos de programación, o de un puñado de ellas en proyectos especialmente importantes, ha pasado a ser llevado a cabo por equipos de trabajo multidisciplinares, en los que cada individuo juega un papel concreto en el complejo ámbito del desarrollo software.

Por otro lado, el software nunca ha estado libre de fallos, y hasta el código más simple es susceptible de contener o provocar algún tipo de error. Uniendo esto a la complejidad y tamaño de los productos software, es imprescindible contar con la capacidad de identificar, y eventualmente corregir, cualquier posible defecto en el código que comprometa su calidad o buen funcionamiento. Estas operaciones son la base del *Testing*, habitualmente traducido como testeo o prueba de software. El *testing* de software es una actividad que consiste en comprobar que los resultados del proceso de desarrollo son los esperados, y que además ese producto está libre de errores.

En ocasiones, las consecuencias de no realizar correctamente estas actividades trascienden al proyecto, y pueden llegar a tener un enorme impacto en el mundo real:

- Un error de software forzó a Starbucks a cerrar casi el 60 % de sus tiendas en EEUU y Canadá, debido a que el sistema servía café gratis porque no se realizaban las transacciones correctamente. [30]
- Algunos proveedores de Amazon vieron cómo sus productos caían de precio en la plataforma. Esas empresas quebraron. [11]
- En Windows 10 había una vulnerabilidad que permitía a los usuarios saltarse la seguridad del SO.
- Un avión Airbus A300 cayó debido a un bug en 1994. Murieron 264 personas. [2]
- En mayo de 2017, la crisis del *ransomware* "*WannaCry*" [6] utilizó una una vulnerabilidad en los sistemas operativos de Microsoft descubierta en marzo de 2017, atacando a más de 230.000 ordenadores de particulares, grandes empresas e incluso instituciones públicas. Esta vulnerabilidad había sido identificada y corregida por Microsoft mediante un parche de seguridad dos meses antes, pero esta actualización no llegó a una gran parte de los equipos vulnerables.

No obstante, actuar de forma reactiva supone un importante coste adicional, y en ocasiones graves consecuencias, razón por la que nace lo que se conoce como Aseguramiento de la Calidad o *Quality Assurance* (QA), que no solo aborda la rectificación de fallos existentes, sino la planificación y ejecución de las medidas necesarias para prevenir su aparición. El QA también requiere la monitorización y supervisión de los procesos, de manera que se asegure la calidad del software en todos los aspectos: minimización de las diferencias entre las expectativas del cliente y el producto entregado, controlar que el software implementado se ajusta a la especificación realizada, asegurar que las condiciones de funcionamiento son las adecuadas (rendimiento y capacidades del producto, etcétera)...

El proceso de QA es altamente complejo, y debe estar presente en cada una de las fases de un proyecto de desarrollo software. De hecho, los denominados probadores de QA tienen perfiles profesionales muy estimados en el mercado de trabajo, y deben poseer unos conocimientos y habilidades concretas para abordar con éxito esta tarea.

Por ello, cobra mayor interés el concepto de automatización de las pruebas. Una proporción importante del *testing* que se lleva a cabo, y especialmente en determinados ámbitos, como el desarrollo web, tiene una naturaleza repetitiva y predecible, y su valor está precisamente en la repetición frecuente de las pruebas (por ejemplo, comprobar que un sitio web está activo y accesible). Por ello, sería posible y deseable automatizar este tipo de tareas, de forma que no requiera de la intervención de un profesional cualificado, que podría dedicarse a cuestiones más productivas.

1.1. Motivación.

Como se ha mencionado anteriormente, es innegable que el testing tiene un papel fundamental en todo el ciclo de vida del desarrollo de software, área en la que se enfoca el Grado de Ingeniería Informática de Servicios y Aplicaciones. Aún así, en el plan de estudios no es una actividad que se trate de forma exhaustiva, aunque sí es cierto que se estudian algunos de sus aspectos clave en algunas asignaturas (pruebas de caja blanca o de caja negra, identificación y prevención de riesgos o gestión de la seguridad en las aplicaciones).

Actualmente, aunque el grueso de la demanda laboral está dedicada al área de desarrollo, el ámbito de las pruebas de software está cobrando más peso, pues las empresas se están empezando a preocupar no solo de entregar los proyectos, sino también de entregarlos bien y con unas garantías de calidad, por lo que es una salida profesional de gran futuro. Las empresas son cada vez más conscientes de la importancia de la fiabilidad y la seguridad del producto, abarcando incluso dimensiones como la protección de datos de carácter personal o la seguridad.

Por todo ello, se hizo la propuesta de este Trabajo Fin de Grado como medio para introducir los conocimientos en el ámbito del testing, y más en concreto en el denominado QA (*Quality Assurance*). Además de todo ello, nuestra propuesta responde a una necesidad que se ha identificado en el mundo real. En muchos ocasiones, las pruebas que se realizan frecuentemente sobre determinadas tipologías muy específicas de sitio web son altamente redundantes, por lo que se podría implementar una herramienta que realice todas esas pruebas de forma automática y genere unos informes, solo configurando algunos parámetros.

Un producto como este permitiría establecer un flujo de trabajo más eficiente y flexible, ya que esas pruebas las podría configurar y ejecutar alguien con un perfil menos especializado que un QA Tester. Un elemento importante de esta herramienta es su sistema de alertas, pues con él nos aseguramos de que cualquier incidencia va a ser comunicada al responsable designado para su pronta resolución.

Por ello, el objetivo será desarrollar una aplicación web sencilla que sirva como base para el desarrollo posterior de una plataforma más completa de automatización de pruebas. El caso de uso elegido es el de un sitio web de comercio electrónico, puesto que nos permitirá ilustrar algunos de los principios más importantes del QA, así como ejemplificar la alta repetitividad de muchas de las pruebas que se suelen llevar a cabo para el aseguramiento de la calidad de un producto software de estas características. Un sitio web de comercio electrónico es un producto web bastante sensible a la calidad puesto que puede haber pérdidas monetarias instantáneas e importantes si no funciona correctamente.

1.2. Visión y alcance del proyecto.

La meta que se persigue con la realización de este Trabajo Fin de Grado es la creación de una plataforma de pruebas que permita monitorizar el correcto funcionamiento de diversas funcionalidades en uno o varios sitios webs. Esa monitorización deberá implementar también un sistema de alertas por correo electrónico para cuando se produzca algún error en alguno de los tests realizados.

Por otra parte, está proyectado construir una herramienta de visualización de los resultados de los tests, que facilite la gestión de los mismos por personal con conocimientos de informática básicos. También se realizarán algunos tests no funcionales con el objetivo de comprobar la respuesta del servidor a una alta carga de trabajo.

Estas dos herramientas facilitarán dar un soporte de calidad a los propietarios de los sitios webs, asegurándoles la disponibilidad y correcto funcionamiento de los mismos.

Para finalizar, se realizará un análisis de los tests ejecutados, tanto funcionales como no funcionales, intentando así proporcionar información útil con la que se puedan tomar decisiones técnicas de optimización.

1.3. Estructura del documento.

Este documento se estructura en diversos capítulos, los cuales se articulan en secciones, en los que se expone de forma gradual todos los aspectos del proyecto. Los capítulos de los que consta son:

1. **Introducción al proyecto:** a lo largo de este capítulo se realiza una breve introducción al proyecto, así como su motivación, visión y alcance total.
2. **Testing de Aplicaciones Web:** en primer lugar se hace una introducción y repaso de qué son las pruebas de software en general, el control de calidad y el aseguramiento de la calidad. También se explican los niveles en los que se puede encuadrar cualquier tipo de test, los tipos más extendidos y las técnicas para diseñar e implementar test. Finalmente, se realiza una explicación de la importancia de la calidad en sitios de venta online.
3. **Estado del arte:** en este capítulo se hace un repaso por algunas de las principales herramientas de testeo funcional y no funcional que hay disponibles en el mercado.
4. **Metodología y planificación del proyecto:** se expone la metodología que se ha seguido durante la realización del proyecto, así como su planificación temporal. También está recogido aquí el presupuesto de todo el material que se entrega, tanto la memoria como el software. Finalmente, se ha realizado un balance para comprobar si las horas y recursos se han ajustado a lo presupuestado inicialmente.

5. **Especificación de tests:** en este capítulo se especifican los tests que se van a realizar como se haría en un entorno de desarrollo real, tanto los funcionales como los no funcionales.
6. **Análisis de requisitos:** en este capítulo se realiza el análisis de todos los requisitos de las dos aplicaciones que se han desarrollado. Se cubren los requisitos de negocio, los de usuario, los funcionales, de información y los no funcionales.
7. **Diseño del sistema:** en primer lugar se diseñan las arquitecturas físicas y lógicas del sistema, para después pasar al diseño lógico de la aplicación, el diseño de interfaces de usuario y el diseño de la base de datos.
8. **Implementación:** en este capítulo se exponen las tecnologías utilizadas en el desarrollo de ambas aplicaciones y finalmente se explica cómo se han implementado los tests funcionales.
9. **Análisis de los tests ejecutados**: aquí se realizará un análisis de todos los tests que se hayan ejecutado hasta el momento, tanto los funcionales como los no funcionales.
10. **Conclusiones:** a lo largo de este capítulo se analizará qué trabajo se esperaba hacer y qué se ha podido realizar finalmente como parte de las conclusiones. Finalmente se expondrá el posible trabajo futuro que tienen las dos aplicaciones que se han desarrollado.
11. **Anexo A. Manual de usuario del sistema:** en este anexo se explica cómo utilizar la aplicación Catón desarrollada.
12. **Anexo B. Manual de usuario de las herramientas utilizadas:** aquí se explica cómo hacer uso de JMeter, Selenium y WooCommerce.

1.4. Entregables del proyecto

En esta sección se enumerarán los productos que se entregarán como resultado del Trabajo.

- **Aplicación de ejecución de tests funcionales con Selenium**: Se entregará un fichero *.jar* con el contenido de todo el código Java que implementa los tests. Está configurado para que los tests se ejecuten sobre el navegador Chrome. Durante su desarrollo, se ha hecho uso de la versión 83.0 del navegador, por lo que es necesario el driver "ChromeDriver.exe" en su versión correspondiente para que Selenium sea capaz de instanciar una ventana del navegador correctamente.
- **Aplicación Catón:** se hará entrega de un fichero *.zip* que contendrá una carpeta donde estarán todos los ficheros que componen el sistema.
- **Memoria:** Se entregará también esta memoria en formato PDF.

- **Driver de Chrome:** se hará entrega del driver para la ejecución de las pruebas con el el navegador Google Chrome en su versión 83.0.
- **Fichero de carga de datos .sql:** Se entregará un archivo que creará y poblará la base de datos necesaria para el correcto funcionamiento de todo el sistema.

Capítulo 2

Proceso de pruebas de Aplicaciones Web

2.1. Testeo de Software

El *testing* es el proceso de comprobar la corrección y calidad de un producto software. El objetivo es verificar que el software satisface los requisitos tomados en la fase de análisis, solventa las necesidades y cumple con las expectativas del cliente. En resumen, se trata de ejecutar un programa para encontrarle *bugs*, errores o defectos. [20]

Conviene resaltar la diferencia entre *bugs*, errores y defectos, ya que se suelen utilizar indistintamente y no son equivalentes:

- Los defectos son irregularidades provocadas por un mal entendimiento de los requisitos funcionales. Suelen ser solventados en la fase de desarrollo.
- Un *bug* es un fallo en el sistema que impacta en los requisitos funcionales o no funcionales. Normalmente los *bugs* se detectan con los tests unitarios (ver Sección 2.5.1). Pueden ser de diferentes tipos, por ejemplo:
  - Errores funcionales
  - Errores de compilación
  - Comandos ausentes
  - Errores en tiempo de ejecución
  - Errores lógicos
  - Mala gestión de excepciones
- Un error es el fallo que se encuentra cuando el sistema ya ha sido entregado al cliente final.

El *tester* o probador es un perfil encargado de planificar y ejecutar las pruebas al software para verificar que el sistema funciona correctamente. Este rol no es necesariamente exclusivo, pues lo pueden realizar diferentes perfiles (desarrollador, diseñador, tester dedicado, o usuario final) y según quién encuentre los fallos se denominará de una forma u otra.

Dependiendo del modelo de ciclo de vida del desarrollo que se haya seleccionado, el proceso de pruebas tendrá presencia en unas etapas u en otras. Tradicionalmente, con el modelo en cascada, se consideraba a las pruebas la última etapa, previa a la entrega del producto al cliente. Según los modelos actuales con un enfoque ágil, las pruebas están presentes desde el inicio hasta el final del desarrollo, y también durante su posterior mantenimiento.

La importancia del testing se deriva del hecho de que los bugs o errores pueden suponer un impacto negativo en múltiples aspectos:

- Pueden afectar a la planificación de los proyectos, ya que identificar y resolver estos errores requiere un tiempo adicional que pudiera no haber sido previsto.
- Pueden afectar a los costes del proyecto, ya que generaría tiempo adicional de trabajo para los desarrolladores.
- Pueden afectar o imposibilitar el correcto funcionamiento del producto o servicio desarrollado
- Puede afectar a la relación con el cliente o los usuarios, debido a un servicio de mala calidad, retrasos o incumplimientos contractuales

Por ello, los beneficios de realizar pruebas se pueden resumir en estos cuatro grandes grupos:

- Seguridad: es el aspecto más vulnerable y sensible de todos en las pruebas de software. Se busca confiar en los productos reduciendo los riesgos y problemas cuanto antes.
- Calidad del producto: es un requisito imprescindible en cualquier producto software. Así se asegura que el consumidor recibe el mejor producto posible.
- Reducción del coste: realizar pruebas desde etapas tempranas del desarrollo disminuye la probabilidad de aparición y prevalencia de fallos. Además, el coste de resolución de un fallo es menor cuanto antes se detecte.
- Satisfacción del cliente: el objetivo principal de cualquier producto es que sea fácil de manejar y entender. Las pruebas de interfaz de usuario y experiencia de usuario (*UI/UX Testing*) aseguran que así sea.

ISTQB [14] es la mayor organización mundial de certificación de probadores del software, creada en 2002 en Bélgica y del que participan diversos países. Su representación en España la realiza el SSTQB [29].

Finalmente, un *test* es un conjunto de casos de prueba, siendo éstos a su vez un conjunto de precondiciones, entradas, acciones (si aplica), resultados esperados y postcondiciones,

desarrolladas en base a las condiciones del test[5]. (ISTQB)

2.2. Control de Calidad

El control de calidad (QC, *Quality Testing*) se ocupa de identificar *bugs* o defectos. Las actividades en las que consiste son el monitoreo y verificación de que los productos entregables cumplen con los estándares de calidad exigidos. Por ello el control de calidad es un proceso reactivo en el que el objetivo principal es la detección de fallos ya existentes. Esa detección se realiza mediante pruebas de software, por lo que el QC engloba a a las pruebas.

Sus actividades principales son:

- Comprobar la integridad de los datos.
- Comprobar la consistencia de los datos.
- Verificar que los flujos de datos a través de los procesos son los correctos.
- Revisar la documentación interna.
- Asegurarse que todo está revisado.
- Comparar los resultados con los resultados esperados.

2.3. Aseguramiento de la Calidad

Por encima del QC, se encuentra el QA (*Quality Assurance*), o aseguramiento de la calidad, que se ocupa de la prevención de los defectos, además de su detección, y por tanto expande el alcance del primero (ver Figura 2.1).

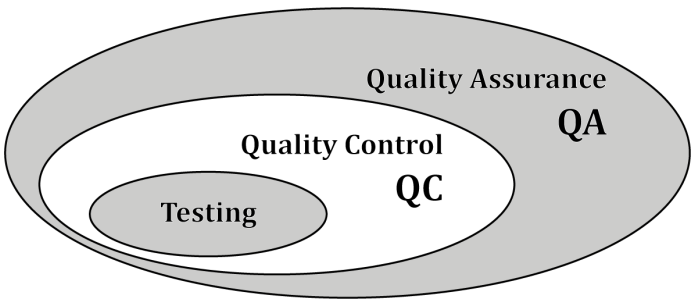


Figura 2.1: Relación entre Testing, QC y QA.

El aseguramiento de la calidad se encarga de asegurar que un proveedor de software está entregando los mejores productos y servicios posibles a sus clientes. Por ello, el QA se enfoca en los procesos de desarrollo y entrega, para que éstos sean eficientes y efectivos, al mismo tiempo que se satisfacen los estándares definidos de calidad.

El responsable de diseñar y aplicar este plan de actuación es el probador de QA. Las actividades de QA consisten en monitorizar y verificar que los procesos para la creación de productos han sido seguidos y son operativos. Su actividad por tanto es transversal a todas las fases del proyecto, pues empieza por identificar el riesgo de la existencia de los errores, los detecta, comunica y sugiere formas de mejorarlo. El proceso de QA es, fundamentalmente, proactivo y de prevención.

Todo proceso de QA consta de un plan, las actividades a realizar, los estándares a seguir, elementos y un conjunto de técnicas que se describirán a continuación.

2.3.1. Plan de QA

El *plan de QA* comprende un conjunto de procedimientos, técnicas y herramientas que permiten asegurar que el sistema final cumple con los requisitos definidos en el Documento de Requisitos Software. El objetivo del plan es especificar las responsabilidad del equipo y enumerar las áreas del sistema que se revisarán.

2.3.2. Actividades de QA

La labor de QA incluye numerosas actividades de muy variada naturaleza, entre las que podemos destacar las siguientes:

- Crear el plan de gestión de QA: es el documento principal, en el que se establece cómo se va a llevar a cabo todo el proceso de QA.
- Establecer *checkpoints*: el equipo de QA debe establecer diferentes hitos a lo largo de todo el ciclo de desarrollo del software para inspeccionar la calidad.
- Revisiones formales técnicas: son reuniones con el equipo de desarrollo para evaluar la calidad y el diseño de un prototipo. Esta actividad ayuda a detectar errores en etapas tempranas y reduce el esfuerzo de corregir más adelante.
- Definir una estrategia de *multi-testing*: el software será considerado correcto si cumple con los requisitos funcionales y no funcionales. Por ello se requiere una estrategia para ejecutar todas las pruebas de cada tipo.
- Controlar el cambio: esta actividad se realiza con una mezcla de procedimientos manuales y automáticos. El objetivo es evaluar la dimensión de los cambios y controlar los efectos, pues solo así se asegura que la calidad se mantiene durante el desarrollo y la fase de mantenimiento.
- Medir el impacto del cambio: cuando se reporta un defecto al equipo de desarrollo y éste lo arregla, el equipo de QA debe determinar el impacto que lleva asociado. Es necesario testear no solo si el defecto se ha corregido, sino si el cambio realizado es compatible con todo el proyecto.

- Realizar auditorías de QA: se evalúa si el actual proceso del ciclo de desarrollo sigue en efecto el proceso establecido.
- Realización de informes: es importante mantener información relacionada con QA para proporcionársela a los diferentes *stakeholders*. Esta documentación puede consistir en resultados de tests, resultados de las auditorías, propuestas de cambios...

2.3.3. Estándares de QA

En general, en QA se pueden seguir uno o varios estándares. Los tres más comunes son:

- **ISO 9000**: [13] este estándar está compuesto por siete principios de gestión de la calidad que ayudan a las organizaciones a asegurar que sus productos o servicios están acorde a las necesidades del cliente. Estos principios son:
  - Enfoque en el cliente.
  - Liderazgo.
  - Implicación y compromiso del personal.
  - Enfoque en el proceso.
  - Mejora continua.
  - Toma de decisiones basadas en evidencias.
  - Gestión de las relaciones.
- **CMMI**: [3] son las siglas de *Capability maturity model Integration*. Este modelo se originó en la ingeniería del software. Se puede usar para mejorar procesos a nivel de proyecto, departamento o incluso una organización entera. Se compone de cinco niveles que determinan el "Perfil de Capacidad": no confiable, informal, estandarizado, monitoreado y optimizado.
- **TMMi**: [32] son las siglas de *Test Maturity Model integration*. Está basado en el CMMI, y pone su foco en la madurez de los niveles la gestión de la calidad y las pruebas de software.

2.3.4. Elementos de QA

A continuación se enumeran los diez elementos básicos que debe tener un proceso de QA:

- Estándares de ingeniería del software.
- Revisiones técnicas y auditorías.
- Pruebas de software para control de calidad (QC).
- Registro de errores y su análisis.

- Gestión del cambio.
- Programas educativos para incluir la calidad como algo ineludible.
- Gestión de proveedores.
- Gestión de la seguridad.
- Protección frente a accidentes.
- Gestión del riesgo.

2.3.5. Técnicas de QA

Aunque la auditoría es la principal técnica, hay otras que conviene destacar:

- Revisión: consiste en una reunión con todos los *stakeholders* para examinar el producto y buscar sus comentarios y aprobación.
- Inspección de código: es una de las técnicas más formales y consiste en la revisión estática del código en busca de defectos que se puedan propagar a otras fases. El revisor no debe ser el autor del código.
- Inspección de diseño: se realiza un *checklist* de diferentes áreas como:
  - Requisitos generales y diseño.
  - Especificaciones funcional y de interfaces.
  - Convenciones.
  - Trazabilidad de requisitos.
  - Estructuras e interfaces.
  - Lógica.
  - Rendimiento.
  - Gestión de errores y recuperación.
  - Acoplamiento y cohesión.
- Análisis estático: consiste en el análisis del software por una herramienta automatizada sin ejecutar el sistema en sí. Esta técnica es muy utilizada en software médico, nuclear o de aviación.
- Simulación: se realiza mediante una herramienta que modela una situación real para examinar virtualmente el comportamiento del sistema.

2.4. ¿Cuál es la diferencia entre QA y QC?

Es un error común pensar que QA y QC son dos términos sinónimos pero no es así. Ambos están unidos y en ocasiones es difícil trazar una línea de delimitación, por ello hay

que ir a ver las diferencias al origen. QA está orientado a la prevención de los defectos a través del proceso de creación, mientras que QC se centra en identificar defectos en el producto ya construido.

En la Tabla 2.1 se muestran las principales diferencias entre estas dos actividades:

QA	**QC**
Se centra en el proceso de construcción.	Se centra en el producto final.
Su objetivo es prevenir los errores.	Su objetivo es identificar y resolver los defectos.
Es una técnica para manejar la calidad.	QC es un método para verificar la calidad.
No implica ejecutar el software.	Implica la ejecución del software.
Todos los miembros del equipo son responsables.	El equipo de pruebas es el responsable.
Verificación.	Validación.
Planificación para realizar un proceso.	Acciones para ejecutar el proceso planeado.

Tabla 2.1: Principales diferencias entre el QC y el QA.

2.5. Niveles de las pruebas

Las pruebas pueden llevarse a cabo a diferentes niveles, dependiendo del momento en que se realicen dentro del ciclo de vida del desarrollo de software. [10] En esta sección, describiremos cada uno de estos niveles y su relación con el punto en que se encuentre el producto en desarrollo.

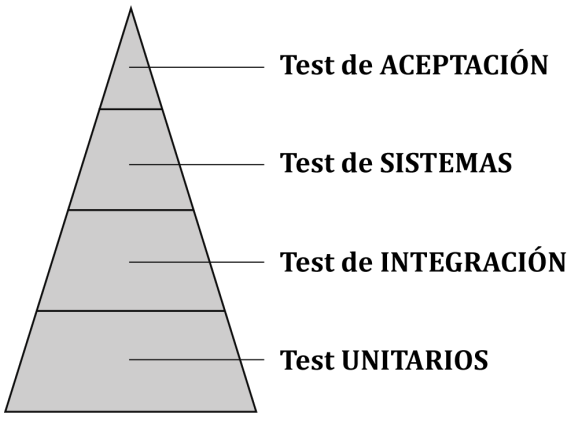


Figura 2.2: Niveles de testing

2.5.1. Test unitario

El test unitario o de componente es el nivel más bajo en que se puede probar la funcionalidad de una aplicación. En él se comprueban las unidades o componentes más fundamentales. El propósito es validar que cada unidad de código se comporta tal y como fue concebida y diseñada. [10] [20] Normalmente, establecen una relación entre una o varias entradas de datos y una salida esperada.

Los elementos a probar dependen del tipo de paradigma de programación que se haya seguido, pudiendo variar entre una función o procedimiento hasta una clase.

Con el objetivo de que una prueba unitaria tenga la calidad suficiente debe cumplir algunos requisitos:

- Reutilizable: en la medida de los posible se deben diseñar las pruebas unitarias de manera que puedan ser utilizadas, más adelante, sobre un producto software distinto.
- Profesionales: los test deben tener la misma consideración que puede tener la documentación de toma de requisitos o el resto de documentos técnicos.
- Automatizables: no debería ser necesaria una intervención manual. Esto es importante porque si seguimos un desarrollo en incrementos, estas pruebas unitarias deberán ser ejecutadas varias veces.
- Completas: deben probar la mayor parte posible de código.

La realización de test unitarios proporciona diferentes ventajas.

- Hacen más fácil la integración: permiten llegar a la fase de integración (ver Sección 2.5.2) con un alto grado de confianza en el código, que permite posteriormente acotar los posibles errores.
- Documenta: las mismas pruebas pueden servir como complemento a la documentación del proyecto, puesto que en ellas se ve qué hace (o se espera que haga) cada parte del proyecto.
- Permiten una refactorización ordenada del código: el programador podrá introducir mejoras en la estructura del código y comprobar con los test unitarios que no se ha perdido funcionalidad o introducido defectos.
- Los errores son más fácilmente localizables.

Por otra parte, los test unitarios no pueden descubrir todos los defectos o errores que pueda tener una aplicación. Debido a su naturaleza, no pueden, por ejemplo, determinar los fallos provocados por la combinación de los elementos básicos probados. Para esos casos, están los test de integración que se explican en la sección siguiente.

2.5.2. Test de integración

En este nivel de las pruebas, el objetivo es comprobar que las partes unitarias funcionan correctamente en conjunto, es decir que la interacción que se establece entre ellas es la adecuada. Por ello, este tipo de test se debe realizar después de las pruebas unitarias y antes de las pruebas de sistema. [10]

Para su implementación se utilizan las principales técnicas habituales en el diseño de pruebas: métodos de caja blanca, negra o gris (que definimos más adelante en la Sección 2.8). Además, al llevar a cabo estos tests de integración se pueden seguir diferentes enfoques:

- *Big Bang*: es un enfoque donde todo el software desarrollado hasta el momento se toma como un todo a ser probado. Es habitual que se utilice este enfoque cuando el equipo de QA recibe el software completo, en lugar de en unidades funcionales.
- *Bottom-up*: se comienza probando las interacciones de las unidades de más bajo nivel. Si éstas se completan satisfactoriamente, se sigue probando la comunicación entre las unidades de más alto nivel.Este enfoque se utiliza cuando los artefactos de software también se están desarrollando de forma *bottom-up*.
- *Top-down*: este modo primero comprueba la integración de módulos de más alto nivel, y progresivamente va disminuyendo hasta llegar a la comunicación entre unidades.
- *Sandwich/Hybrid*: es una mezcla entre los dos enfoques anteriores.

2.5.3. Test de sistema

Este nivel de pruebas corresponde a la fase donde se comprueba todo el software. [10] El objetivo es verificar que todos los requisitos funcionales identificados en la fase de análisis han sido implementados y funcionan correctamente. Se ejecutan después de que hayan concluido los test de integración (Sección 2.5.2) y antes de comenzar con los test de aceptación de usuario (Sección 2.5.4).

Las actividades principales son:

- Probar la comunicación con todas las aplicaciones integradas, incluyendo los posibles periféricos, para verificar cómo interaccionan unos componentes con otros y con el sistema como un todo. A este tipo de pruebas también se le denomina "Escenario *End to end*".
- Comprobar todas las entradas de la aplicación para verificar que las salidas son las esperadas.
- Probar la experiencia de usuario con la aplicación.

Habitualmente se utiliza el método de caja negra.

2.5.4. Test de aceptación

En este nivel de pruebas, se prueba si el sistema es aceptable desde el punto de vista de los requisitos de negocio, y por tanto es posible su puesta en producción. Se realiza después de la conclusión de los test de sistema (Sección 2.5.3), y suelen ser pruebas de caja negra o hechas *ad-hoc*.

Dependiendo del personal que realice estas pruebas se puede diferenciar entre:

- Proceso de pruebas interno (o *Alpha Testing*): es ejecutado por miembros que pertenecen a la organización donde se ha realizado el desarrollo, aunque no por programadores o probadores, sino por jefes de producto, ventas o soporte al cliente.
- Proceso de pruebas externo: se realiza por personas que no pertenecen a la empresa que ha desarrollado el producto software:
  - Pruebas de aceptación de cliente: lo lleva a cabo personal de la empresa que encargó el producto software.
  - Pruebas de aceptación de usuario (o *Beta Testing*): lo realiza el usuario final, que pueden ser los clientes o los clientes de los clientes.

2.6. Pruebas funcionales

Según el ISTQB [14], son las pruebas ejecutadas para evaluar si un componente o sistema satisface los requisitos funcionales. Esto se realiza sometiendo la lógica del software a ejecuciones controladas en las que las entradas están predefinidas y las salidas que deben generarse son conocidas. Hay de diversos subtipos que se especifican a continuación.

2.6.1. Test de humo.

El test de humo o *smoke test* es un tipo de pruebas cuyo objetivo es asegurar que la mayoría de las funciones más importantes de un software funcionan correctamente. [24] Por ello, no se trata de una prueba exhaustiva, pues solo se pretende comprobar que se puede continuar con el trabajo de pruebas porque la versión evaluada es lo suficientemente estable. El término proviene de las pruebas de hardware, cuando se comprueba que el dispositivo no se incendia cuando es encendido la primera vez.

Los test de humo responden preguntas básicas como "*¿El programa es ejecutable?*" o "*¿Se abre la interfaz de usuario?*".

Entre sus ventajas se pueden destacar:

- Muestra problemas de integración.

- Descubre problemas de forma temprana.
- Ofrece cierto nivel de confianza de que los cambios no han afectado a las grandes áreas.

Los test de *smoke* se emplean en los niveles de integración, sistemas y aceptación.

2.6.2. Test sanitario.

El test sanitario o *sanity test* tipo de prueba se realiza cuando se recibe un producto software con cambios menores en el código o la funcionalidad, con el fin de asegurar que los *bugs* encontrados previamente han sido arreglados y no se han visto modificadas otras partes que funcionaban correctamente. Si el *sanity test* o test sanitario falla, se devuelve el producto a los desarrolladores antes de continuar con unas pruebas más rigurosas.

Suele ser habitual la confusión entre el *smoke test* y el *sanity test*, por ello es conveniente resaltar sus diferencias:

Smoke Test	**Sanity Test**
Se ejecuta para comprobar que las principales funcionalidades del programa se ejecutan correctamente.	Se hace para verificar que los bugs han sido corregidos.
Habitualmente se documenta.	Normalmente no se documenta.
Es un subtipo de los test de aceptación (ver Sección 2.5.4).	Es un subtipo de los test de regresión (ver Sección 2.6.4).
Prueba el sistema entero.	Prueba un componente particular.

2.6.3. Test de interfaces

Una prueba de interfaces consiste en comprobar el funcionamiento correcto de cualquier comunicación entre dos sistemas software. La conexión entre ambos sistemas puede ser una API, un servicio web,... Una interfaz consiste en un conjunto de comandos, mensajes y otros atributos. Todo ello se debe verificar.

Dentro de las pruebas de interfaces, se pueden distinguir varios subtipos:

- *Worflow:* Su objetivo es comprobar que el componente encargado de gestionar las interacciones o los flujos de trabajo entre los componentes o sistemas es compatible con el funcionamiento de los mismos.
- *Casos extremos o valores inesperados:* analizar cómo reacciona el sistema ante valores inesperados. Por ejemplo, esto se considera cuando la fecha llega en un formato invertido.

- *Prueba de desempeño, carga y redes:* dependiendo de la carga de trabajo que se prevea tengan las interfaces, será necesario comprobar que soporta bien el tráfico y no comete errores.
- *Sistemas individuales:* cada uno de los sistemas debe funcionar de forma separada. Debe exitir cohesión pero no acoplamiento.

La diferencia principal entre las pruebas de interfaces y las pruebas de integración es que el primero tiene por objetivo probar las comunicaciones entre componentes o sistemas diferentes, mientras que el segundo se enfoca en encontrar los defectos en las interfaces y las interacciones entre componentes integrados.

2.6.4. Test de regresión

Cualquier cambio o evolución del código puede causar efectos en otras partes. Esto se denomina regresión y puede ser de varios tipos:

- Introducción de una funcionalidad nueva.
- Resolución de un defecto.
- Refactorización de código para una mejor eficiencia.
- Cambios en el entorno de *hosting* de la aplicación

Este tipo de prueba se realiza para confirmar que un cambio reciente no ha provocado errores en otras partes del programa. Para ello se ejecutan una selección de test de otros tipos (*sanity*, interfaces...) que ya se hayan probado anteriormente, para así verificar que las funcionalidades existentes antes del cambio siguen funcionando bien. Los test de regresión se pueden ejecutar a cualquiera de los niveles descritos en la Sección 2.5.

2.7. Pruebas no funcionales

Según el ISTQB, la prueba no funcional de un sistema evalúa las características de sistemas y software, como la usabilidad, la eficiencia del desempeño o la seguridad. La prueba no funcional prueba "cómo de bien" se comporta el sistema. [8]

2.7.1. Test de Localización

El objetivo de un test de localización es asegurar que una aplicación se adecúa a la cultura de los usurios finales que la van a utilizar.

La mayor parte de estos test se llevan a cabo en la interfaz gráfica de usuario. En ella se verifica el idioma por defecto, la moneda, formato de fecha y hora para que sea usable en un país o cultura determinado.

2.7.2. Test de Rendimiento

Los test de *performance* o de rendimiento son los que se ejecutan para asegurar que las aplicaciones software se comportan adecuadamente bajo una carga de trabajo determinada. Por ello, se miden los tiempos de respuesta, la confianza en el programa, el uso de recursos y la escalabilidad.

El objetivo de estos test es encontrar cuellos de botella, no fallos o defectos en el código fuente.

Los tres puntos principales que probar son:

- Velocidad: determina que el software responde rápido.
- Escalabilidad: Mide qué carga máxima de trabajo puede soportar.
- Estabilidad: Comprueba si la aplicación es estable bajo varias cargas.

2.7.2.1. Test de Carga

Se somete el sistema a unas condiciones de carga determinadas y constantes (por ejemplo, un cierto número de conexiones de usuario simultáneas), generalmente representativas de una situación realista, una vez que el sistema haya sido desplegado y esté en uso. El objetivo principal es comprobar que el sistema es capaz de absorber dicha carga de trabajo y seguir su funcionamiento con normalidad.

Hay algunos puntos clave que es necesario verificar:

- La capacidad máxima que puede soportar.
- Determinar si la actual infraestructura es suficiente para la ejecución de la aplicación.
- Tiempo que puede estar el software a su máxima capacidad ante una demanda mayor del usuario.
- Número de usuarios concurrentes que una aplicación puede soportar.

2.7.2.2. Test de estrés

Con este test, se evalúa la robustez del sistema y su capacidad de manejo de errores frente a una situación de crecimiento sostenido del volumen de carga que debe procesar.

Este crecimiento se suele aplicar de forma progresiva y escalonada, de manera que pueda caracterizarse el desempeño del sistema a intervalos regulares de volumen de carga.

Por tanto, a diferencia del test de carga, que analiza el comportamiento en condiciones normales, el test de estrés trata de explorar los límites del sistema, así como su respuesta a condiciones inusuales.

2.7.2.3. Test de volumen

Los test de volumen se ejecutan para comprobar el comportamiento del sistema en función del tamaño de su base de datos. En particular, suelen realizarse utilizando una base de datos sobrecargada con el fin de caracterizar el tiempo de respuesta del sistema en estas condiciones. Estos tests presentan algunas diferencias con respecto al test de carga:

Test de volumen	**Test de carga**
Se realiza con la máxima cantidad posible de datos en la base de datos.	La aplicación se ejecuta con una cierta cantidad de carga de ejecución para analizar su comportamiento.
Verifica si el sistema responde como se espera ante una cierta cantidad de datos.	Verifica el comportamiento del sistema cuando el usuario incrementa la carga de ejecución.

2.7.2.4. Test de resistencia

Los test de resistencia someten al sistema a una alta carga de trabajo durante un tiempo prolongado, con el fin de observar si su desempeño se degrada con el tiempo, en unas condiciones de uso intenso y continuado. Este matiz los diferencia de los tests de carga, ya que estos únicamente evalúan si el sistema es capaz de asumir un cierto nivel de carga, sin considerar que este pueda ser sostenido en el tiempo.

Este tipo de prueba suele ser la última en ejecutarse dentro del conjunto de tests de comportamiento. Se asegura así que la aplicación es capaz de manejar una carga de trabajo considerable sin bajar los tiempos de respuesta.

2.7.2.5. *Spike testing*

El *spike testing* es un tipo de prueba de rendimiento donde el sistema recibe un incremento o disminución de carga de forma repentina. El objetivo es determinar el comportamiento de la aplicación con variaciones grandes de carga. Con esta prueba no solo se testea la carga máxima que puede soportar, también se mide el tipo de recuperación entre picos de actividad. Ese intervalo de tiempo debería ser tan corto como sea posible.

2.7.3. Test de seguridad

Los test de seguridad se ejecutan para detectar vulnerabilidades o riesgos que podrían causar un gran daño al propio sistema y/o a la organización propietaria. Dichos daños podrían ser pérdidas de información, pérdidas monetarias o reputación de la marca.

Hay varios tipos de test de seguridad, según se enfoquen en un aspecto u otro:

- Búsqueda de vulnerabilidades: se trata de pruebas automáticas que buscan vulnerabilidades ya conocidas. Un ejemplo de una herramienta que realiza este tipo de pruebas es "w3af" [34].
- *Security Scanning*: consiste en identificar las debilidades del sistema y la red, para posteriormente dar soluciones a esos problemas. Se pueden realizar tanto manual como automáticamente.
- Test de penetración: el objetivo de este tipo de test es simular el ataque de un hacker. Para ello, es necesario realizar un análisis previo similar al que haría un atacante, identificando las características y posibles puntos débiles que aprovechar en el sistema.
- Evaluación de riesgos: en este caso, se realiza un análisis de los riesgos de seguridad observados y se los cataloga, según su impacto potencial, en "Bajo","Medio" o "Alto".
- Auditoría de seguridad: es una inspección del código de la aplicación cuyo objetivo es encontrar puntos débiles que puedan ser susceptibles de ser brechas de seguridad.
- Hacking ético: consiste en hackear una organización con el objetivo de mostrar sus vulnerabilidades, no robar su información.

2.7.4. Test de compatibilidad

En este tipo de test se pretende determinar si el software construido es capaz de funcionar en diferentes entornos.

Existen varios tipos de test de compatibilidad:

- Hardware: verifica que el software sea compatible con diferentes configuraciones de hardware.
- Sistemas operativos: comprueba que la aplicación sea compatible con varios sistemas operativos.
- Software: consiste en verificar que el sistema es compatible con otras aplicaciones. Por ejemplo, MS Word debe ser compatible con MS Outlook, MS Excel...
- Red: evalúa el comportamiento de la aplicación en una red, cambiando diferentes parámetros como el ancho de banda, la velocidad o la capacidad.

- Navegador: verifica la compatibilidad con diferentes navegadores como Internet Explorer, Chrome, Opera o Firefox.
- Dispositivos: comprueba la compatibilidad con dispositivos externos como memorias USB, impresoras, escáners...
- Móvil: prueba que el sistema funcione correctamente sobre sistemas operativos móviles como Android o iOS.
- Versiones de software: evalúa si la aplicación es compatible con diferentes versiones de software. Por ejemplo, MS Word debe ser compatible con Windows 7, Windows 10... Hay dos tipos de pruebas de versionado:
  - Hacia adelante: es compatible con las versiones más nuevas.
  - Hacia atrás: verifica que sea compatible con versiones anteriores.

2.7.5. Test de instalación

Los test de instalación se ejecutan para verificar que el software ha sido instalado correctamente con todas las características requeridas y que funciona según lo esperado. Es la última fase del proceso de pruebas de un producto software, ya que se realiza justo antes de que el usuario tenga su primera interacción con él.

Gracias a estos test, el equipo puede determinar la corrección del proceso de instalación y asegurar que la experiencia de usuario sea la apropiada.

2.7.6. Test de recuperación

Los test de recuperación se encargan de determinar qué operaciones se pueden seguir ejecutando en el caso de que el sistema haya sufrido un fallo o su integridad se haya visto comprometida.

Un punto importante en un test de recuperación en la capacidad del sistema de volver a un punto anterior al fallo, previo a la manifestación del fallo.

2.7.7. Test de usabilidad

Los test de usabilidad se focalizan en ver las formas de mejorar la facilidad de uso de la aplicación, flexibilidad y experiencia de usuario final (*user experience*, UX).

Para ello, el sistema se muestra a un grupo reducido de usuarios finales para que expongan los defectos que puedan encontrar.

2.7.8. Test de conformidad

Los test de conformidad se realizan para determinar si un sistema cumple con los requisitos, condiciones, regulaciones y estándares requeridos según la documentación del proyecto generada en la fase de análisis.

Puede incluir los siguientes aspectos:

- Eficiencia.
- Funcionalidad.
- Robustez.
- Interoperabilidad.

2.8. Técnicas de prueba

2.8.1. Caja Negra

El método de caja negra es una técnica que prueba la funcionalidad de una aplicación sin tener en cuenta o conocer su implementación, diseño o estructura internas, de manera más cercana a la perspectiva de un usuario del sistema. Así, el foco principal está en observar las salidas que genera el sistema ante unas entradas determinadas para determinar si son correctas, independientemente del estado interno o los resultados intermedios que genere el sistema.

Aunque se pueden utilizar tanto para test no funcionales como funcionales (ver Secciones 2.6 y 2.7), es más común en los funcionales.Puede aplicarse en todos los niveles de pruebas, exceptuando el unitario; es decir, en las pruebas de integración, de sistema y de aceptación. Por ejemplo, intenta encontrar errores de las siguientes categorías:

- Errores en las interfaces.
- Errores en las estructuras de datos o en el acceso a bases de datos externas.
- Fallos en el comportamiento debido a la carga.
- Funciones ausentes o incorrectas.

En el diseño de los test de caja negra se pueden seguir algunas pautas:

- Análisis de los valores *border*: hay que determinar el rango de valores que puede admitir una función, para posteriormente pasárselos y ver si se comporta debidamente en los límites.
- Causa - efecto: se identifican las condiciones de entrada y los efectos que a priori deberían tener. Después, se aplican los test con las condiciones de entrada previstas y se comprueba si la salida real del sistema se corresponde con la esperada o deseable.

- Particionado: se seleccionan valores representativos de los grupos de entrada de datos válidos e inválidos y se compruebas las salidas.

Figura 2.3: Caja negra. Diagrama

VENTAJAS:

- Los test se realizan desde el punto de vista del usuario final, esto ayuda a identificar discrepancias con la especificación de requisitos realizada en la fase de análisis.
- El probador no necesita conocimientos de programación o saber cómo ha sido implementado el software para realizar los test correctamente.
- Los test pueden ser ejecutados por personas ajenas al equipo de programadores.
- Los casos de prueba se pueden diseñar tan pronto como la especificación de requisitos haya concluido.

DESVENTAJAS:

- Con este método solo pueden ser probadas un número reducido de entradas, por lo que no es un proceso de pruebas exhaustivo.

2.8.2. Caja Blanca

En la técnica de caja blanca, el probador tiene acceso a la implementación, estructura y diseño del producto que está testando.

Para ello, examina todos los posibles caminos de ejecución y selecciona diferentes valores que se adapten a esas casuísticas. Posteriormente, verifica que las salidas son las apropiadas para cada entrada.

Debido a que se conoce desde un inicio la estructura interna del producto, los tests se pueden ejecutar para asegurarse de que las operaciones internas hacen exactamente lo que pide la especificación y que se han probado todos los componentes adecuadamente.

A diferencia de las pruebas de caja negra, las pruebas de caja blanca son adecuadas para ser aplicadas a nivel de test unitario, de test de integración y de test de sistema. No son aplicables para los tests de aceptación, pues la perspectiva del usuario no incluye conocer la estructura o estados internos del sistema.

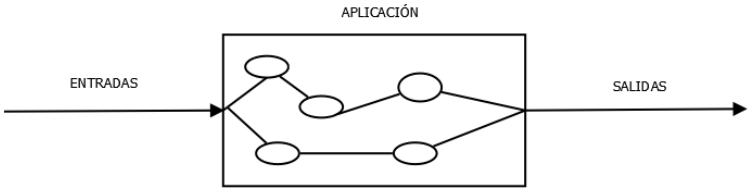


Figura 2.4: Caja blanca. Diagrama

VENTAJAS:

- Se puede comenzar a ejecutar test con esta técnica desde etapas muy tempranas del desarrollo.
- Las pruebas se realizan con más profundidad, por lo que la cantidad de errores final es menor.

DESVENTAJAS:

- Los test resultantes diseñados utilizando esta técnica son muy complejos y requieren conocimientos de programación.
- El mantenimiento de estos test puede ser difícil si la implementación del producto a probar cambia continuamente.

2.8.3. Caja Gris

Los test de caja gris son una combinación de los test de caja negra y los test de caja blanca. La diferencia principal es que se tiene acceso parcial al código y las estructuras de datos para ver los valores borde o posibles caminos de ejecución, pero posteriormente se ejecutan como si fueran test de usuario o de caja negra (solo se comprueban las salidas teniendo en cuenta las entradas).

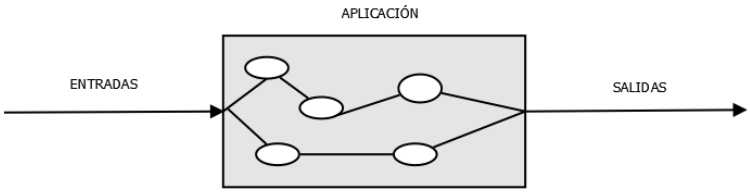


Figura 2.5: Caja gris. Diagrama

Aunque pueden ser utilizados en todos los niveles de testing, son principalmente usados en Integración.

VENTAJAS:

- Unifica las ventajas de la técnica de caja negra y la de caja blanca.

- Combina los conocimientos de los probadores junto con los de los programadores, lo que ayuda a alcanzar una mayor calidad de producto.
- El test se realiza desde el punto de vista del usuario final.

DESVENTAJAS:

- Pueden ser redundantes si ya se han ejecutado casos de prueba similares con alguna de las otras dos técnicas.
- Probar cada una de las posibilidades de datos de entrada consume mucho tiempo y puede dejar zonas del software sin evaluar.

2.9. QA aplicado a comercio electrónico

Un *e-Commerce* es una plataforma software utilizada para comprar o vender productos o servicios a través de una red, habitualmente Internet. Debe constar de un catálogo de productos, unos usuarios registrados y una pasarela de pago.

Por calidad en un sitio de comercio electrónico entendemos principalmente la disponibilidad, la compatibilidad con diferentes dispositivos móviles o la accesibilidad.

Cuando se aborda el desarrollo de un sitio web de estas características generalmente se utilizan plataformas, librerías y frameworks que proporcionan una variedad de componentes muy depurados, en lugar de hacer soluciones *ad-hoc.*Algunas de esas plataformas son WooCommerce, PrestaShop o Magento, que están respaldadas por una gran comunidad.

Recordemos que al ser una tienda, puede haber pérdidas monetarias si algo falla o no funciona del modo correcto. Esa disponibilidad puede referirse a diversos aspectos, tanto funcionales como no funcionales.

Dentro de los funcionales, se pueden citar la respuesta del servidor, que los productos estén visibles y disponibles para su compra o que el login se pueda realizar sin problemas. Respecto a los no funcionales, son importantes la compatibilidad con dispositivos móviles, el nivel de carga que puede soportar el servidor o el estrés al que se le puede someter.

2.10. ¿Qué es WooCommerce?

WooCommerce es un plugin para WordPress de código abierto que fue lanzado en el año 2011. Permite implementar tiendas electrónicas con catálogos limitados. Es una de las soluciones más extendidas en la web debido a su sencillez de instalación y personalización de las tablas de la base de datos de productos vendidos.

WordPress es un sistema CMS o de gestión de contenidos. Inicialmente fue ideado para crear blogs, pero debido a la gran cantidad de plugins y temas, ahora permite crear sitios web más complejos. Actualmente se calcula que alrededor de un tercio de las páginas web funcionan bajo WordPress. El lenguaje que utiliza en servidor es PHP, y está desarrollado bajo licencia GPL para sistemas gestores de bases de datos MySQL y servidores Apache.

WooCommerce fue desarrollado en un principio por la empresa WooThemes.

Principales características de WooCommerce:

- WooCommerce es un plugin de instalación sencilla para WordPress.
- Dentro de los plugins para ecommerce de WordPress es el más popular.
- No son necesarias licencias para su uso. Además es gratuito y de código abierto.
- Se pueden crear tiendas de tamaño pequeño y mediano.
- Se puede instalar sin conocimientos en informática, aunque para su personalización sí se requieren.
- Soporta la mayoría de plantillas gratuitas que hay disponibles para WordPress.
- No afecta a otros plugins que estén instalados en la web.
- Soporta los métodos de pago más conocidos.
- Implementa protocolos seguros como HTTPS.

Para la realización de este proyecto, se ha construido un sitio web de comercio electrónico con WooCommerce y se desplegará en un servidor local para facilitar su instalación, configuración y acceso. Se ha utilizado una plantilla gratuita y unos datos de prueba para que este sitio web sirva de ejemplo en la ejecución de pruebas.

A continuación se muestran dos capturas de dos páginas del sitio que serán importantes en la ejecución de los tests. La primera (Figura 2.6) muestra el formulario de contacto que se ha agregado al sitio; y la segunda (Figura 2.7) la página de inicio de sesión para usuarios registrados.

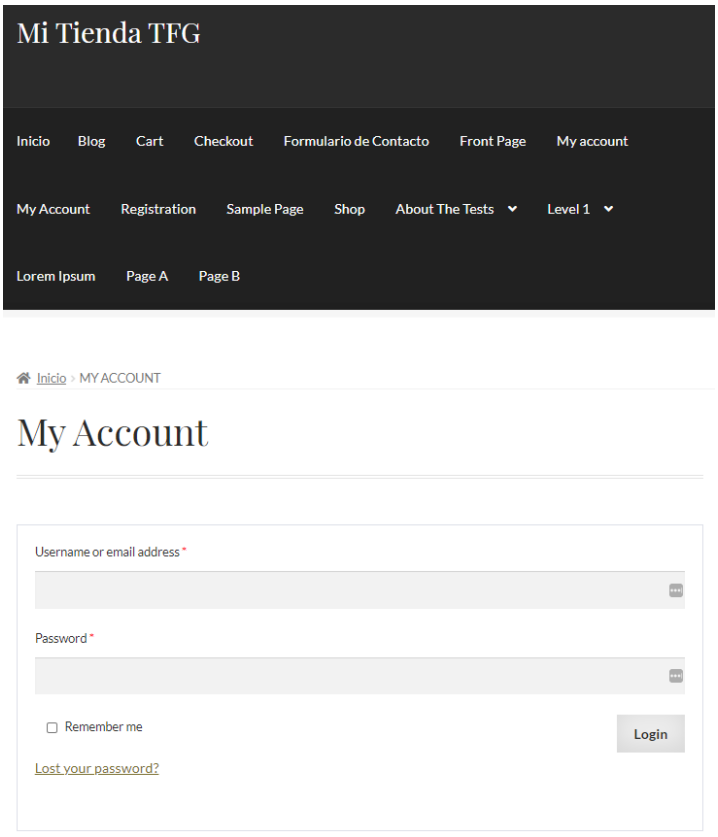


Figura 2.6: Inicio de Sesión. Mi Tienda.

Figura 2.7: Formulario de Contacto. Mi Tienda.

Capítulo 3

Estado del arte

3.1. Herramientas para pruebas funcionales

3.1.1. Selenium

Selenium [26] es la principal herramienta de automatización de pruebas orientada al testing de aplicaciones web. Su característica más importante es su flexibilidad: la librería Selenium está disponible en múltiples lenguajes de programación muy extendidos en la actualidad, como Java o Python, y es compatible con una gran variedad de navegadores y plataformas.

Comparte muchas características con QTP (ver Sección 3.1.5), ya que fue QTP la referencia que tomaron sus creadores para desarrollar Selenium.

Selenium es una suite de herramientas, cada una de las cuales se orienta a satisfacer unas necesidades específicas. Esta suite constaba inicialmente de cuatro componentes, aunque a partir de la versión 2, los componentes Selenium RC y WebDriver se unificaron en un solo *framework*. A continuación, se describe cada uno de estos componentes:

3.1.1.1. Selenium Integrated Development Environment (IDE)

Este componente de Selenium es un plugin disponible para varios navegadores (entre ellos Firefox o Chrome). Permite crear casos de test simples mediante un sencillo método que no requiere conocimientos de programación: se graba una secuencia de interacciones del usuario con la aplicación web (por ejemplo, pulsar botones o hacer *scroll*), y el propio módulo genera el código necesario para replicar esas mismas acciones. Esto lo convierte en el *framework* más simple de aprender de Selenium.

Sin embargo, esa simplicidad se torna pronto limitante. Si se quieren desarrollar casos de prueba más complejos, se debe recurrir a Selenium 2 (la unificación de Selenium RC y WebDriver).

Pros	**Contras**
Instalación sencilla y uso fácil.	Solo se pueden hacer prototipos.
No son necesarios conocimientos de programación, basta con HTML.	No permite iteraciones ni condiciones de control.
Los test se pueden exportar a formato de WebDriver o Selenium RC.	La ejecución es lenta en comparación con otros frameworks.
Tiene un módulo de *reporting* de resultados de test.	No está disponible para todos los navegadores.

3.1.1.2. Selenium Remote Control (RC)

Selenium RC fue el componente principal de la suite de Selenium durante mucho tiempo. Fue la primera herramienta de automatización de pruebas que permitía a los probadores elegir el lenguaje de programación con el que escribir los test.

Desde la versión 2.25.0, RC soporta los siguientes lenguajes:

- Ruby
- Java
- PHP
- Python
- Perl
- C#

Pros	**Contras**
Permite la iteración y las operaciones condicionales	Instalación más compleja.
Soporta pruebas guiadas por datos.	Necesita Selenium RC Server para su ejecución.
Tiene una API muy completa.	La API tiene comandos redundantes y confusos.
Soporta varios navegadores.	Son necesarios conocimientos de programación.
Ejecución más rápida que el IDE.	Ejecución más lenta que WebDriver.
Multi-navegador y multi-plataforma.	A veces devuelve resultados inconsistentes.
	Necesita JavaScript.

3.1.1.3. WebDriver

La principal novedad introducida por Selenium WebDriver es que no es indispensable Selenium RC para ejecutar las pruebas; a cambio, utiliza comandos (bien sea vía API de

cliente o a través de Selenese (el lenguaje de comandos interno de Selenium IDE), un conjunto de comandos predefinidos de Selenium con los que probar diferentes componentes de un sitio web) que envía al navegador y así toma su control. No requiere que JavaScript esté activado para su funcionamiento correcto.

Esto está implementado a través de un controlador específico para cada navegador que manda los comandos y recibe las respuestas. También existe un controlador que simula un navegador (HtmlUnit).Además, permite su utilización en combinación con Selenium Grid, que se describe a continuación.

Los lenguajes soportados son los mismos que tenía Selenium Remote Control.

Pros	**Contras**
Instalación más sencilla que Selenium RC.	Instalación más compleja que Selenium IDE.
La interacción con el navegador es más realista.	Son necesarios conocimientos de programación.
Se comunica directamente con el navegador.	No soporta navegadores nuevos.
No necesita componentes externos como Selenium RC.	No dispone de reporting de resultados de test.
Ejecución más rápida que Selenium IDE y Selenium RC.	

3.1.1.4. Selenium Grid

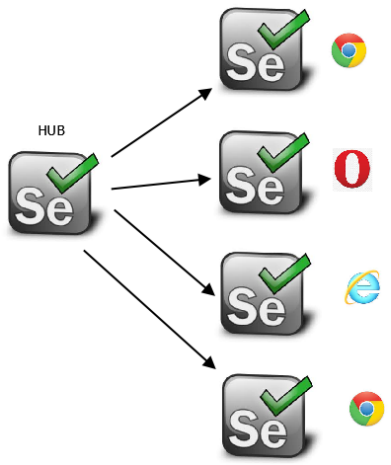


Figura 3.1: Arquitectura de Selenium grid

Selenium Grid es una herramienta de para la ejecución de pruebas de forma automática, que se utiliza en combinación con Selenium RC para ejecutar test de forma pararela en diferentes máquinas y navegadores.

Entre sus características principales se encuentran:

- Permite la ejecución simultánea de múltiples test.
- Supone un enorme ahorro de tiempo
- Se basa en una arquitectura en nodos, de los cuales el más importante es el llamado *hub* que es el encargado de enviar los comandos al resto de nodos, que son los que finalmente ejecutan los test.

3.1.2. JUnit

JUnit [16] es un conjunto de clases programadas por Erich Gamma y Kent Beck para la ejecución de pruebas unitarias en aplicaciones Java. Actualmente se encuentra en su versión 5.

Con JUnit se puede crear un árbol de *suites* de pruebas, compuestas a su vez de casos de prueba. Los casos de prueba se utilizan para probar el correcto funcionamiento de una clase o método implementado en Java. De esta forma, cada clase que se quiera probar deberá llevar una clase de JUnit asociada.

Como se trata de pruebas unitarias, se provee un valor de entrada a cada uno de los métodos de la clase y, en función de los resultados obtenidos, se determina si la clase cumple con los nuevos requisitos. En el caso de que el valor devuelto sea distinto al esperado, JUnit marcará como fallo el método correspondiente.

Gracias a la organización en conjuntos de pruebas y en casos de pruebas se pueden ordenar por funcionalidad y ejecutarlas automáticamente, facilitando así el testing de regresión.

Los resultados se pueden mostrar en modo gráfico o en modo texto.

Actualmente, algunos entornos de desarrollo como Eclipse o Netbeans tienen disponibles plugins que permiten generar las clases de prueba de JUnit automáticamente mediante plantillas, de forma que el desarrollador se pueda concentrar en el test y el resultado esperado.

3.1.3. Cucumber

Cucumber [4] es una herramienta software de pruebas que soporta el desarrollo guiado por comportamiento (*Behaviour Driven Development*, o BDD), siendo BDD un proceso de desarrollo ágil que facilita el entendimiento entre la parte técnica y de negocio estableciendo un lenguaje común que permite entender el comportamiento del sistema que se quiere construir. El lenguaje común con el que Cucumber especifica las pruebas se denomina Gherkin.

Una vez los tests están descritos, Cucumber convierte automáticamente la especificación en Gherkin a lenguaje Ruby para poderlo ejecutar.

Inglés	Español	Italiano	Klingon
feature	Característica	Funzionalità	Qap
background	Antecedentes	Contesto	mo'
scenario	Escenario	Scenario	lut
given	Dado	Dato	DaH ghu' bejlu'

Tabla 3.1: Ejemplos de comandos de Gherkin en diferentes idiomas compatibles.

Es de señalar que Gherkin permite escribir la especificación en más de 60 idiomas, entre los que se encuentran el español, francés, esperanto o klingon, lo que además facilita la portabilidad del código entre idiomas.

Cucumber es a menudo utilizado para la ejecución automática de las pruebas de aceptación.

3.1.4. Ranorex Studio

Ranorex Studio [22] es una herramienta comercial de testing automático propiedad de la empresa autríaca Ranorex GmbH. Salió al mercado en el año 2007.

Ranorex no utiliza un lenguaje específico de *scripting*, sino que está construida sobre la plataforma de Microsoft .NET. Soporta los lenguajes de programación típicos de Windows: C# y Visual Basic .NET para la grabación y creación de test a medida.

Utiliza el lenguaje de consultas XPath para localizar los componentes HTML en el DOM de las aplicaciones web.

Ranorex Studio está disponible para una amplia gama de entornos, especificaciones y configuraciones.

Pros	Contras
Puede ejecutar test para plataformas móviles, web y de escritorio.	Es software propietario, por lo que la licencia tiene un coste alto.
Genera informes automáticamente con los resultados de los test.	Solo tiene soporte C# y Visual Basic .NET.
Fácil de aprender, y no requiere conocimientos de programación avanzados.	No está disponible para su funcionamiento sobre Mac OS.
Detecta automáticamente cambios en la interfaz de usuario.	La comunidad de desarrolladores es limitada.
Permite la colaboración entre equipos de desarrollo y de testing.	Las nuevas *releases* no son muy estables.

3.1.5. QTP/UFT

QTP [21] es un software propietario de HP, quien adquirió y renombró este programa a UFT. Permite el testing automático y de regresión de aplicaciones software.

Utiliza VBScript como lenguaje de *scripting* para especificar los test, tratar los objetos y controlar la aplicación a probar.

QTP permite a los desarrolladores probar desde una única consola las tres capas de una aplicación: la interfaz de usuario, la capa de servicio y la base de datos.

Selenium	**QTP**
Open source y gratuito.	Comercial y propietario.
Altamente extensible.	Dispone de algunos *add-ons*, pero son limitados.
Permite correr test en múltiples navegadores.	Solo soporta Firefox, Internet Explorer y Chrome.
Se ejecuta sobre diferentes sistemas operativos.	Solo está disponible para Windows.
Soporta dispositivos móviles.	Es necesario un módulo específico de pago para que admita las pruebas en dispositivos móviles.
Puede ejecutar los test con el navegador minimizado.	El navegador tiene que estar visible.
Puede ejecutar test en paralelo.	Para una ejecución de los test en paralelo necesita un módulo de pago.
Permite test aplicaciones web.	Se pueden testar aplicaciones web y de escritorio.
Es más lento que QTP.	Es más rápido que Selenium.
No puede acceder a elementos del navegador (barra de favoritos,...).	Sí puede acceder a elementos del navegador.
No dispone de atención al cliente.	Sí dispone de atención al cliente.
No ofrece parametrización a medida.	Ofrece parametrización a medida.
No genera reportes automáticamente.	Sí genera reportes automáticamente.

Tabla 3.2: Comparativa entre Selenium y UFT (antes QTP).

3.1.6. TestingWhiz

TestingWhiz [31] es una plataforma comercial que facilita la automatización de múltiples tipos de testing, asegurando una gran compatibilidad y una amplia cobertura de la infraestructura de una aplicación web:

- Testing de Regresión: Permite automatizar los tests de regresión realizando una grabación de los pasos que se deben seguir.
- Testing para Aplicaciones Web: Su objetivo es automatizar los test funcionales a través de la interfaz de usuario.
- Testing para varios navegadores: Ayuda a automatizar los test de las aplicaciones web en diferentes navegadores como Internet Explorer, Firefox, Google Chrome, Safari u Opera.
- Testing de Bases de Datos: Permite la realización de tests automáticos en la base de datos, como por ejemplo la validación de la configuración del servidor o la ejecución de *benchmarks* de base de datos.
- Servicio Web para operar con la herramienta: Permite verificar que los servicios web tienen una buena comunicación entre el front y el back.
- Testing para aplicaciones móviles: Permite ejecutar test sobre aplicaciones nativas, híbridas o web en dispositivos reales o simulados.

3.1.7. Rapise

Rapise[23] es una plataforma comercial orientada a usuarios con conocimientos técnicos limitados, basando su funcionamiento en la grabación de secuencias de acciones de usuario y sin necesidad de programar. Entre sus principales características se encuentran:

- Automatización sin necesidad de código: Permite crear casos de test de forma rápida gracias a la grabación de los pasos de cada test.
- Microsoft Dynamics y Salesforce: Rapise incluye soporte para la realización de tests sobre plataformas como Dynamics AX, CRM, NAV y 365, así como Salesforce Classic.
- Automatización robusta: Se basa en un enfoque orientado a objetos que permite crear y depurar los test mediante drag and drop.
- Flexible y Extensible: Se basa en Selenium y Appium, además de hacer uso de los principales *frameworks* de pruebas unitario. Así mismo, se integra con herramientas de gestión de pruebas como Microsoft Team System.

3.2. Herramientas para pruebas no funcionales

3.2.1. JMeter

Apache JMeter [15] es una herramienta *open source* escrita en Java, que permite realizar pruebas no funcionales de carga y comportamiento, tanto de aplicaciones web como de escritorio.

Originalmente, JMeter se utilizaba para comprobar sistemas web o aplicaciones FTP. Ahora se utiliza en test no funcionales y en servidores de bases de datos, entre otros.

JMeter simula un grupo de usuarios enviando peticiones a un servidor, y devuelve las estadísticas de carga del sistema. Finalmente las muestra en diagramas.

Algunas de las ventajas de JMeter son:

- Licencia *open source*: JMeter es gratis.
- GUI sencilla: tiene una interfaz de usuario, lo que facilita su uso.
- Multiplataforma: está escrita enteramente en Java, lo que le permite ser ejecutada sobre diferentes plataformas.
- Visualización de resultados: los resultados de los test ejecutados se pueden ver en diversos formatos, como tablas, árboles o archivos de log.
- Ejecución sencilla: solo es necesario ejecutar un archivo *.bat*. No es necesaria una instalación propiamente dicha.
- Altamente extensible: permite escribir los propios test, además de disponer de *plugins* que amplían la funcionalidad nativa de JMeter.
- Múltiples estrategias de prueba: JMeter permite llevar a cabo diferentes tipos de test, como pruebas de carga o funcionales.
- Simulación: puede simular muchos usuarios concurrentes a través de hilos, creando así la carga que soportará la aplicación a testar.
- Múltiples protocolos: JMeter soporta HTTP, JDBC, LDAP, SOAP, JMS, y FTP.
- *Record & Playback*: permite grabar las interacciones del usuario en el navegador y traducirlas automáticamente a código ejecutable.
- *Scripting* de test: puede integrarse con Bean Shell y Selenium para pruebas automáticas.

3.2.2. WebLOAD

WebLOAD es una herramienta para empresas que requieran realizar pruebas de carga exigentes con el sistema. Permite realizar test de carga y estrés sobre cualquier aplicación web, generando dicha carga en la nube.

Los puntos positivos de WebLOAD son su flexibilidad y facilidad de uso, pues permite definir rápidamente test con características determinadas como correlación automática o con JavaScript.

Como salidas a los test, WebLOAD ofrece análisis claros y concisos sobre el comportamiento bajo carga, problemas en los tiempos de respuesta del servidor o cuellos de botella. Esta herramienta soporta muchas tecnologías (desde protocolos web a aplicaciones empresariales), además de tener integración con Selenium o DevOps. Puede ejecutarse sobre Windows y Linux. Sus principales características son:

- IDE: dispone de un entorno de desarrollo integrado que permite grabar, editar y cargar los scripts de test. Los test se generan automáticamente en JavaScript aunque pueden ser mejorados usando herramientas integradas en el IDE.
- Correlación automática: realiza una correlación de los id de sesión para cada uno de los múltiples clientes virtuales.
- Generación de la carga: la carga la puede crear de forma local o desde la nube.
- Análisis: tiene predefinidos una serie de informes que muestran información de *performance*, facilitando así la identificación de cuellos de botella. También están disponibles en un *dashboard* que puede ser visualizado de forma remota.
- PMM: recolecta estadísticas de la ejecución de los test en el lado del servidor, y ofrece información adicional sobre el problema de origen.

3.2.3. Loadstorm

Loadstorm [18] es una herramienta de implementación de pruebas de carga en la nube. Dispone de una versión gratuita y una versión de pago con funcionalidades extra. Al igual que en otras plataformas, se pueden crear los planes de test, los criterios por los que se considerará si un test ha pasado satisfactoriamente y los escenarios de ejecución de cada uno de ellos. Permite crear más de 50.000 usuarios virtuales simultáneos y dirigirlos a la web que se esté evaluando.

No es necesario ningún conocimiento de programación para poder utilizar esta herramienta, que corre sobre sistemas operativos Windows.

Del mismo modo que otras aplicaciones del estilo, ofrece gráficos e informes variados donde muestra información de varias métricas como ratios de error, tiempo medio de respuesta del servidor o el número de usuarios concurrentes.

3.2.4. Rational Performance Tester

Rational Performance Tester (RPT) [12] es una herramienta para pruebas de carga. Permite la creación, ejecución y análisis de los test para validar la escalabilidad y confiabilidad de la web probada durante el desarrollo y su puesta en producción.

Sus principales características son:

- No son necesarios conocimientos de programación.
- Los test se pueden programar a una fecha u hora concretas.
- Provee de informes en tiempo real para la identificación de problemas de forma inmediata.
- Es capaz de realizar los test con múltiples usuarios virtuales.
- Necesita cargas de trabajo de usuario.
- Es posible automatizar la variabilidad de los test.
- Identifica automáticamente las respuestas del servidor.
- Soporte al entorno y la plataforma.
- Admite Java para la mejora y personalización de los test.
- Compatible con los protocolos Citrix, Socket Recording, Web HTTP, SOA, SAP, XML, Websphere y Weblogic.

Capítulo 4

Metodología y planificación del proyecto.

En este capítulo se expone la metodología que se seguirá para realizar el trabajo. La implementación de la metodología dará lugar a una planificación temporal y económica, que se describirán en las respectivas secciones. Finalmente, haremos un balance de la marcha del proyecto a lo largo de su desarrollo, analizando las divergencias que podrían producirse durante el mismo con respecto a la planificación inicial.

4.1. Metodología de desarrollo.

4.1.1. Scrum.

La planificación y organización del proyecto se llevará a cabo mediante el *framework* de desarrollo ágil Scrum [7], muy utilizado actualmente en la empresa para gestionar equipos de trabajo en entornos cambiantes o en los que los requisitos de un proyecto no están claros desde el principio.

Se utilizará este marco de trabajo por tres razones principales:

- El trabajo se enmarca dentro de un área en el que el equipo de trabajo aún se está formando, por lo que es difícil hacer una estimación inicial con fundamento de los esfuerzos y actividades necesarias. Por ello, se necesita una metodología que nos permita adaptar los objetivos del proyecto a medida que se avanza en el aprendizaje.
- Relacionado con el punto anterior y dado que no conocemos bien la complejidad del trabajo que queremos llevar a cabo, Dado que en un TFG el principal factor limitante es el tiempo y debemos ajustarnos a una fecha de entrega, necesitamos una metodología que nos permita adaptar el alcance del proyecto a medida que se vayan definiendo los

objetivos y en función de los recursos disponibles para su finalización.

- Scrum promueve la retroalimentación y la revisión constante de los avances del proyecto, lo que ayudará tanto para la organización del trabajo, como para la realización de una tutorización efectiva por parte del tutor.

Equipo Scrum.

En un equipo de trabajo Scrum se definen tres roles:

- *Product Owner*: No participa en el equipo de desarrollo, pero se encarga de identificar, describir y priorizar las necesidades del proyecto, y también de servir como enlace entre el equipo de desarrollo y los *stakeholders* del proyecto.
- *Scrum Master*: Se encarga de la buena marcha del proceso Scrum y del avance de las tareas por parte del equipo de desarrollo. Asume la función de enlace con el *Product Owner*. Puede tener dedicación exclusiva, o asumir tareas como cualquier otro miembro del equipo de desarrollo, y tiene una naturaleza rotativa entre los miembros del equipo de desarrollo.
- Miembro del equipo de desarrollo: Participa en la planificación de los incrementos del proyecto y lleva a cabo las tareas necesarias para su éxito.

Ciclo de vida.

En Scrum, el desarrollo del proyecto se lleva a cabo de forma incremental, organizando el trabajo en base al concepto de *sprints*. Un *sprint* es un periodo de tiempo de longitud fija a lo largo de todo el proyecto, con duración inferior a 30 días. En cada uno de los *sprints*, el equipo de desarrollo determinará qué objetivos abordará de los identificados en el proyecto, y que se compromete a alcanzarlos dentro de dicho *sprint*. Al final de cada sprint, se deberá alcanzar un *incremento* funcional y *finalizado*.

Al comienzo del proyecto se identifican una serie de ítems necesarias para completarlo, y se describen en el *Product Backlog* o Pila de Producto (inventario de trabajo pendiente); por ejemplo, en forma de historias de usuario. La lista inicial no tiene por qué ser completa, sino que se pueden agregar nuevos ítems en cualquier momento de la vida del proyecto. Los requisitos del proyecto son gestionados por el *Product Owner*, que transmite al equipo de desarrollo los requisitos que establecen los distintos *stakeholders* del proyecto a través del *Product Backlog*.

De manera iterativa, en cada *sprint* el equipo Scrum al completo selecciona ítems del *Product Backlog* para su implementación. Esta selección se lleva a cabo partiendo de los criterios establecidos por el *Product Owner* y la capacidad de trabajo que pueda asumir el equipo de desarrollo. Para cada ítem elegido, se definen varias tareas más pequeñas que lleven a la satisfacción de los objetivos de la primera. Esta lista de sub-tareas da lugar al

Sprint Backlog, que elaboran y gestionan todos los miembros del equipo de desarrollo, y se resumen en un objetivo del *sprint* (*sprint goal*).

La elaboración del *Sprint Backlog* se lleva a cabo al inicio del *sprint*, en una reunión denominada *Sprint Planning*. Posteriormente, la evolución del desarrollo del *sprint* se monitoriza mediante reuniones diarias de sincronización (*daily meetings*), en las que los miembros del equipo expresan sus avances y dificultades a la hora de abordar sus tareas, y otros medios accesorios como un tablero visual.

El cumplimiento de los objetivos del proyecto son evaluados por el *Product Owner*. Al final de cada *sprint*, se lleva a cabo una *Sprint Review* donde se reúnen el *Product Owner* y los *stakeholders* y se analiza la satisfacción de los objetivos elegidos para el *sprint* en base a las tareas realizadas por el equipo de desarrollo.

Por otro lado, el equipo de desarrollo debe realizar una *Retrospectiva*, una reunión donde cada miembro del equipo expone las tareas que ha completado con éxito, los problemas que han existido y las propuestas de mejora que deberían introducirse para el siguiente *sprint*.

4.1.2. Aplicación al proyecto.

Debido a las características del proyecto, será necesario aplicar algunas adaptaciones a la forma en que se aplicará el marco de trabajo Scrum. Por ejemplo, el equipo de trabajo consta de una única persona, y como *stakeholder* tendremos únicamente los requisitos exigibles para un Trabajo de Fin de Grado. Además, será necesario adaptar la organización del proyecto a circunstancias que condicionarán la marcha del mismo, como la dedicación que se pueda asegurar para su desarrollo al tener que compaginarlo con una actividad profesional.

En primer lugar, se utilizará una duración de *sprint* de tres semanas, con una dedicación de 20 horas semanales. En el ámbito del TFG, el tutor actuará en calidad de *Product Owner*, ya que es quien conoce los requisitos para el desarrollo del trabajo y establece su alcance y objetivos.

El equipo de desarrollo estará formado únicamente por la autora del trabajo. En principio esto no es algo contemplado por Scrum, pero sigue siendo aplicable dado que el "equipo" contará igualmente con las habilidades necesarias para llevar a cabo el proyecto planteado. Por ello, se prescindirá de las reuniones diarias de sincronización, y la utilización del tablero visual de equipo se hará de una forma más informal, con fines puramente auto-organizativos.

Los objetivos del proyecto se acordarán con el *Product Owner* en lugar de encargarse el tutor en exclusiva. La definición de los objetivos de proyecto se hará de una manera más formal, ya que nos permitirán valorar el grado de éxito alcanzado en su desarrollo y adaptar el trabajo futuro en consecuencia.

Dada la naturaleza del trabajo, no todos los requisitos exigibles pueden especificarse como historias de usuario, que es una de las formas más habituales para describir las necesidades del cliente en Scrum. En su lugar, describiremos *ítems* que recogen la información esencial de

acuerdo a la guía de Scrum [7]: "Los elementos de la Pila del Producto tienen como atributos la descripción, el orden, la estimación y el valor". Como el proyecto se entregará como un todo, se considerará que todas las tareas aportan el mismo valor. La prioridad se establecerá con una escala de 1 (prioridad alta) a 5 (prioridad baja), siendo este un valor orientativo y sujeto a cambios a lo largo del proyecto.

Para evaluar la complejidad y estimar el tiempo que nos llevará completar cada una de las tareas que conduzcan al éxito del proyecto, utilizaremos un sistema de valoración que asigna puntuaciones de 1 a 5, correspondiendo el 1 a las tareas menos complejas y el 5 a las que requieran un mayor esfuerzo.

Utilizando estos valores de prioridad y complejidad, al comienzo de cada sprint determinaremos qué tareas abordaremos en el mismo. La prioridad nos ayudará a identificar las tareas que se deban completar con mayor urgencia, mientras que la complejidad nos permitirá evaluar la cantidad de trabajo que podemos asumir en un sprint. Si en un sprint hemos planificado un determinado número de puntos de complejidad y nos ha sobrado tiempo, en el siguiente sprint podremos asumir un mayor número de puntos de complejidad. Por el contrario, si no hemos conseguido completar todas las tareas planeadas, en el siguiente sprint deberemos reducir los puntos de complejidad que se abordarán.

Debido a que en cada sprint tenemos un número fijo de horas (60h), asignaremos a cada tarea una parte de este tiempo proporcional a su complejidad. Por lo tanto, la relación entre puntos de complejidad y horas no será fija y cambiará en cada sprint, en función de la complejidad total de las tareas que se asuman.

Este mecanismo de planificación se alinea con el principio de estimación basada en el empirismo que defiende el marco de trabajo Scrum, ya que se basa en la experiencia del equipo y en la evolución del propio proyecto.

Las tareas se clasifican con unos identificadores según el tipo de trabajo que sea. TEO para las actividades puramente teóricas, TESTING se refiere a la aplicación de ejecución de tests automáticos, CATON a la herramienta web de gestión de los tests y visionado de resultados, PROP a las propuestas de trabajo futuro de las dos aplicaciones a desarrollar, META al trabajo que se ha de llevar a cabo en la redacción de la memoria y QC el trabajo de control de calidad del producto software resultante del proyecto.

4.2. Planificación del proyecto.

El proyecto se organizará en cinco *sprints* de tres semanas cada uno, completando 300 horas de trabajo efectivo de acuerdo a lo establecido en la sección anterior. La distribución de tareas prevista se describe en la Tabla 4.1.

A continuación, se describen las tareas necesarias para llevar a cabo el Trabajo Fin de Grado, que después deberán ir asignándose a los diferentes *sprints*. Las tareas con más

ID	Descripción	Dificultad	Prioridad
TEO-01	Introducción al Testing de Software	2	2
TEO-02	Estado del Arte	2	2
TESTING-01	Especificación de los tests	2	1
TESTING-02	Análisis de requisitos técnicos de los tests funcionales	3	1
TESTING-03	Diseño de los tests funcionales	3	1
TESTING-04	Implementación de los tests funcionales	4	2
TESTING-05	Ejecución de los tests no funcionales	1	2
TESTING-06	Análisis de los resultados de todos los tests	3	3
CATON-01	Análisis de requisitos de la aplicación Catón	3	1
CATON-02	Diseño de la aplicación Catón	3	1
CATON-03	Implementación de la aplicación Catón	4	2
PROP-01	Trabajo futuro	1	4
META-01	Describir la metodología de trabajo	1	1
META-02	Definir el alcance y objetivos del trabajo	1	1
META-03	Redactar los contenidos de introducción	1	1
META-04	Consolidar la planificación	1	1
META-05	Elaborar los presupuestos	1	1
META-06	Redactar el balance final	1	5
META-07	Redactar las conclusiones del proyecto	1	5
META-08	Revisión de los contenidos de la memoria	1	5
META-09	Revisión del formato de la memoria	1	5
QC-01	Revisión de control de calidad de ambas aplicaciones	3	5

Tabla 4.1: Tareas identificadas para el proyecto.

prioridad tienen un mayor nivel de concreción, mientras que las que se realizarán más adelante pueden detallarse en el futuro.

TEO-01 Introducción a las pruebas de Software

En esta primera historia se persigue proporcionar una introducción a los contenidos del Trabajo partiendo del contexto general de las pruebas de Software. Se definirá el concepto de pruebas en Software, y se expondrán brevemente los tipos de test más importantes que se realizan.

TEO-02 Estado del Arte

Durante esta historia se expondrán algunas de las principales herramientas de testing que hay disponibles en el mercado, tanto para la realización de testing funcional como no funcional.

TESTING-01 Especificación de los tests

El objetivo de esta historia es hacer una especificación de los tests funcionales y no

funcionales planeados. En el caso de los tests funcionales se deben tener en cuenta los posibles distintos escenarios.

TESTING-02 Análisis de requisitos técnicos de los tests funcionales

A lo largo de esta historia se analizarán los requisitos que deben satisfacer los tests funcionales para su correcta ejecución.

TESTING-03 Diseño de los tests funcionales

A lo largo de esta historia se realizará el diseño de los tests funcionales partiendo del análisis previo realizado en la historia TESTING-02

TESTING-04 Implementación de los tests funcionales

Durante esta historia se codificarán los tests funcionales en el lenguaje de programación seleccionado.

TESTING-05 Ejecución de los tests no funcionales

El objetivo de esta historia es configurar y ejecutar los tests no funcionales que se hayan especificado en la historia TESTING-01

TESTING-06 Análisis de los resultados de todos los tests

En esta historia se analizarán los resultados de los tests funcionales y no funcionales ejecutados hasta el momento.

CATON-01 Análisis de requisitos técnicos de la aplicación Catón

A lo largo de esta historia se analizarán los requisitos que debe satisfacer la aplicación de gestión Catón.

CATON-02 Diseño de la aplicación Catón

A lo largo de esta historia se realizará el diseño de la aplicación Catón partiendo del análisis previo realizado en la historia CATON-01

CATON-03 Implementación de la aplicación Catón

Durante esta historia se implementarán todos los requisitos analizados en la historia CATON-01 según el diseño obtenido en la historia CATON-02 con las tecnologías que hayan sido seleccionadas.

PROP-01 Trabajo futuro

A lo largo de esta historia se expondrá el posible trabajo futuro tanto de la aplicación de los tests funcionales como de Catón.

META-01 Describir la metodología de trabajo

En esta historia se expondrá la metodología de trabajo que se seguirá durante la realización de este Trabajo.

META-02 Definir el alcance y objetivos del trabajo

Durante esta historia se describirá la magnitud total prevista del proyecto y los hitos a alcanzar del mismo.

META-03 Redactar los contenidos de introducción

A lo largo de esta historia se describirán todos los apartados introductorios de todas las secciones de la memoria.

META-04 Consolidar la planificación

En esta historia se realizará la planificación del proyecto en el tiempo disponible para su ejecución.

META-05 Elaborar los presupuestos

En esta historia se elaborará la previsión de costes que conlleva la realización del proyecto.

META-06 Redactar el balance final

Durante esta historia se realizará una revisión del presupuesto por si hubiera habido desvíos. Se verificará también si se ha realizado en el tiempo estipulado.

META-07 Redactar las conclusiones del proyecto

Durante esta historia se realizará una comparativa de los resultados previstos y los obtenidos.

META-08 Revisión de los contenidos de la memoria

En esta historia se repasará la memoria para verificar su completitud.

META-09 Revisión del formato de la memoria

En esta historia se repasará la memoria para verificar que su formato sea el correcto según lo exigido.

QC-01 Revisión de control de calidad de ambas aplicaciones

Durante esta historia se realizará una búsqueda de los errores que hayan podido llegar a la fase final.

4.2.1. Planificación temporal

Las tareas descritas anteriormente deberán abordarse dentro de las 15 semanas previstas para el desarrollo del Trabajo Fin de Grado completo. Para ello, partiremos de la planificación de tareas por sprint que se recoge en la Tabla 4.2, y se describe de manera gráfica en el diagrama de Gantt de la Figura 4.1:

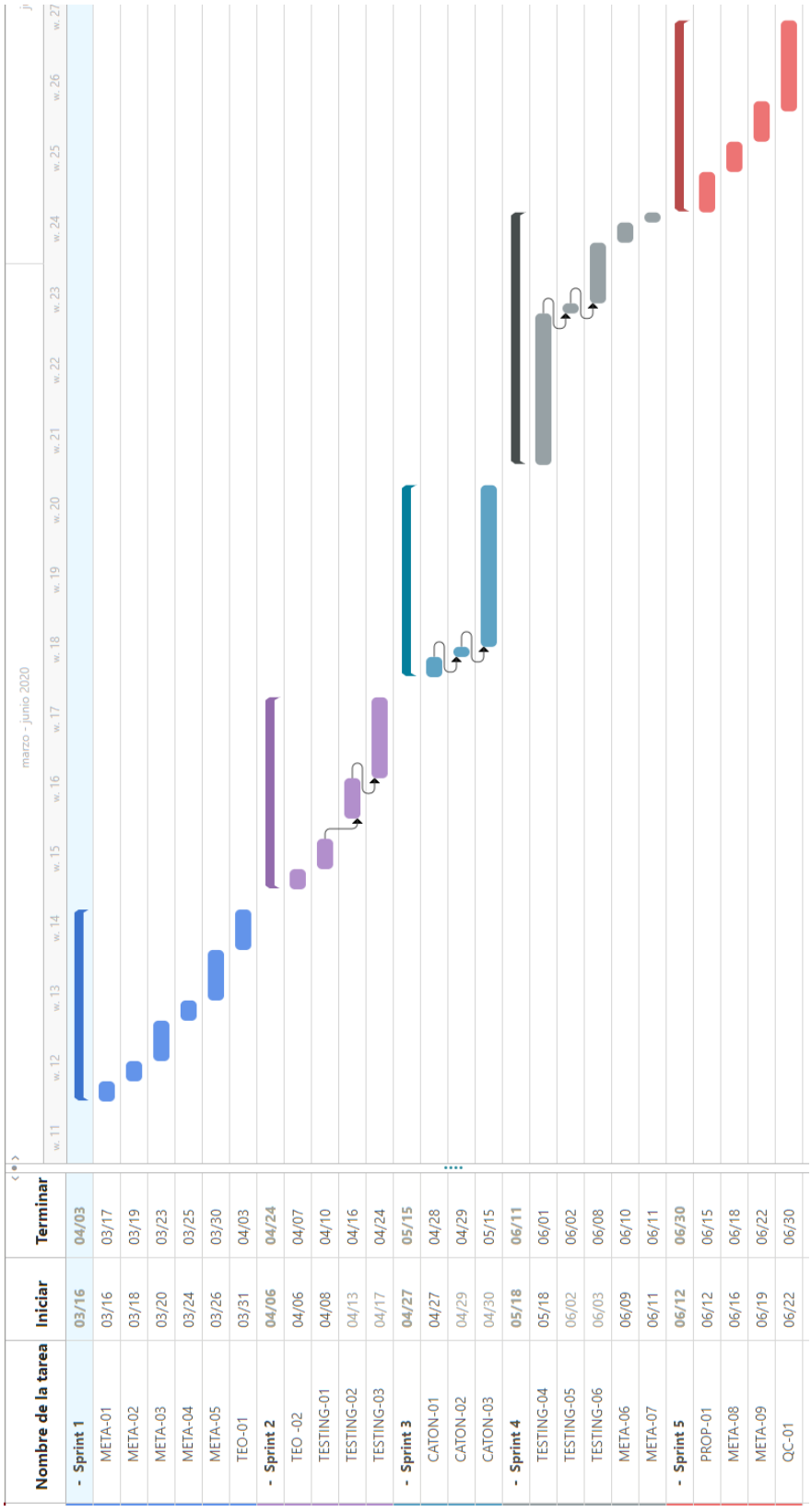


Figura 4.1: Diagrama Gantt

Spr	Tareas	Horas/ tarea	Jefe proyecto	Analista	Desarrollador	QA tester
1	META-01 META-02 META-03 META-04 META-05 TEO-01	5 5 10 10 10 20	60	-	-	-
2	TEO-02 TESTING-01 TESTING-02 TESTING-03	4 4 20 32	4	52	-	4
3	CATON-01 CATON-02 CATON-03	5 5 50	-	10	46	4
4	TESTING-04 TESTING-05 TESTING-06 META-06 META-07	38 2 8 10 2	12	-	38	10
5	PROP-01 META-08 META-09 QC-01	8 8 8 36	24	-	-	36
Total		300 h	100 h	62 h	84 h	54 h

Tabla 4.2: Dedicación temporal por roles en cada uno de los sprints planificados.

4.3. Presupuestos.

En esta sección se recogen todos los costes que conllevará la realización de las actividades previstas, siempre y cuando se cumpla la planificación inicial que hemos llevado a cabo. Para ello, analizaremos las diferentes fuentes de gasto que estarán presentes durante el desarrollo del proyecto, así como la adquisición o desgaste de recursos necesarios.

4.3.1. Costes directos

4.3.1.1. Recursos humanos

Los costes derivados de la contratación de personal para llevar adelante el proyecto dependerán de dos aspectos:

- El rol del trabajador que debe llevar a cabo las tareas: los diferentes perfiles profesio-

nales suponen un coste diferente, en función de que se considere que su labor aporta más o menos valor.

- El tiempo de dedicación: El coste de su trabajo es directamente proporcional al número de horas que inviertan, en lugar de trabajar por objetivos o por obra.

Para ello, se desglosan las horas de trabajo dedicadas al proyecto por el rol del trabajador dentro del proyecto. En el caso de que hubiera tareas compartidas por varios trabajadores con diferentes cargos, se calculará mediante un porcentaje de dedicación.

Teniendo en cuenta que una jornada laboral habitual de 8h al día y el sueldo anual medio[1] para cada uno de los perfiles profesionales considerados, podemos elaborar la relación de costes que se incluye en la Tabla 4.3.

Rol	**Salario anual**	**Horas**	**€/hora**	**Coste (€)**
Jefe de proyecto	42.000 €	100	14,38	1.438,00
Analista SW	32.000 €	62	10,95	678,90
Desarrollador	25.000 €	84	8,56	719,04
QA Tester	25.000 €	54	8,56	462,24
Total				3.298,18

Tabla 4.3: Tabla de costes de recursos humanos previstos.

Los costes por hora calculados se corresponden con el salario bruto por hora, y por tanto no reflejan los costes de contratación para la empresa. Existen diferentes conceptos que debe asumir la empresa por la contratación de un trabajador (contingencias comunes, FOGASA, tipo general de desempleo y formación profesional) que encarecen los costes en alrededor de un 30 % del salario bruto del trabajador [2]. Por tanto, el importe presupuestado para personal alcanza los 4.287,63 €.

4.3.1.2. Software

En la Tabla 4.4 se muestra el desglose de las herramientas utilizadas junto con su amortización. Puesto que solamente se han utilizando herramientas gratuitas, los costes de este tipo son nulos.

4.3.1.3. Hardware

En la Tabla 4.5 se describe el hardware que se ha empleado en el desarrollo de la aplicación y su memoria, su coste y amortización:

[1] Fuente: Linkedin Salary. Actualizado a julio de 2020. `https://www.linkedin.com/salary/`

[2] Fuente: Seguridad social.
`www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537`. Último acceso: 16 de julio de 2020.

Producto	Coste	Amortización
Gantter	-	-
Google Chrome	-	-
Selenium WebDriver	-	-
XAMPP	-	-
WordPress	-	-
WooCommerce	-	-
WooCommerce Template	-	-
Gmail	-	-
Overleaf	-	-
JMeter	-	-
Netbeans	-	-
Windows 10	Inc. portátil	-
Total	0 €	0 €

Tabla 4.4: Relación de costes de software.

Producto	Vida útil	Coste (€)	Amortización (€)
MSI GE72 2QD Apache Pro	4 años	1400	49,58
Total			49,58

Tabla 4.5: Relación de costes de hardware.

Por lo tanto, el total de esta parte de costes del presupuesto es la suma de los recursos humanos, del software y del hardware. Esto da una suma de **4.337,9 €** .

4.3.2. Costes indirectos

En este apartado se calculan los costes que no se pueden atribuir solamente al proyecto, ya que son recursos compartidos con otras actividades y/o personas.

4.3.2.1. Gastos comunes

Producto / Servicio	Periodo	Coste (€)	Amortización (€)
Consumo eléctrico	3,5 meses	33,26	116,41
Acceso a internet	3,5 meses	30	105,00
Total			221,41

Tabla 4.6: Relación de costes indirectos.

4.3.2.2. Contingencias

Con el fin de mitigar los posibles errores en las diferentes fases de análisis, desarrollo, ejecución y redacción de la memoria; es recomendable aplicar un margen del 20 % para tener en cuenta las posibles desviaciones respecto a las cantidades presupuestadas.

En este caso, sería:

$$(4.337{,}90 + 221{,}41) * 0{,}2 = \mathbf{911{,}86\ €}$$

4.3.3. Coste total del proyecto

El coste total del proyecto será la suma de todos los costes calculados anteriormente:

Costes directos	Costes indirectos	Contingencias	Total (€)
4.337,90 €	221,41 €	911,86 €	**5.471,17 €**

Tabla 4.7: Resumen de presupuestos.

4.4. Balance final del proyecto.

En esta sección se realizará un análisis del tiempo finalmente empleado en la consecución de los objetivos del proyecto, así como su distribución temporal. En base a las divergencias identificadas, recalcularemos los costes asociados al proyecto para valorar su rentabilidad.

4.4.1. Distribución final temporal

A continuación se especifica la distribución de tareas a lo largo de los sprints.

Sprint 1 (16/3/2020 - 3/04/2020)

Tareas planificadas:

- META-01: Describir la metodología de trabajo
- META-02: Definir el alcance y objetivos del trabajo
- META-03: Redactar los contenidos de introducción
- META-04: Consolidar la planificación
- META-05: Elaborar los presupuestos
- TEO-01: Introducción al Testing de Software

Sprint 2 (6/4/2020 - 24/4/2020)

Tareas planificadas:

- TESTING-01: Especificación de los tests
- TESTING-02: Análisis de requisitos técnicos de los tests funcionales
- TESTING-03: Diseño de los tests funcionales

Sprint 3 (27/4/2020 - 15/5/2020)

Tareas planificadas:

- TEO-02: Estado del Arte
- CATON-01: Análisis de requisitos de la aplicación Catón
- CATON-02: Diseño de la aplicación Catón
- CATON-03: Implementación de la aplicación Catón

Sprint 4 (18/5/2020 - 5/6/2020)

Tareas planificadas:

- TESTING-04: Implementación de los tests funcionales
- TESTING-05: Ejecución de los tests no funcionales

Sprint 5 (8/6/2020 - 26/6/2020)

Tareas planificadas:

- TESTING-06: Análisis de los resultados de todos los tests
- PROP-01: Trabajo futuro
- META-06: Redactar el balance final

Sprint 6 (29/6/2020 - 17/7/2020)

Tareas planificadas:

- QC-01: Revisión de control de calidad de ambas aplicaciones
- META-07: Redactar las conclusiones del proyecto
- META-08: Revisión de los contenidos de la memoria
- META-09: Revisión del formato de la memoria

4.4.2. Costes finales

Una vez finalizado el trabajo es necesario revisar los presupuestos de los que se partía y contrastarlo con los costes que finalmente se han producido. Esta sección está dedicada a reflejar esas desviaciones en el presupuesto, derivadas del hecho de que hemos tenido que realizar un sprint adicional para completar el proyecto.

Rol	**Precio/hora**	**Previsto (h)**	**Real (h)**	**Real (€)**
Jefe de proyecto	14,38	100	120	1.725,60
Analista	10,95	62	70	766,50
Desarrollador	8,56	84	100	856,00
QA Tester	8,56	54	60	513,60
Total				3.861,70

Tabla 4.8: Tabla de costes finales de recursos humanos.

Producto / Servicio	**Periodo**	**Coste (€)**	**Amortización (€)**
Consumo eléctrico	4 meses	33,26	133,04
Acceso a internet	4 meses	30	120,00
Total			253,04

Tabla 4.9: Relación de costes indirectos finales.

No han existido sobrecostes en las herramientas software ni hardware por lo que se ha cumplido con lo presupuestado.

Los sobrecostes existentes son derivados del hecho de que el proyecto se ha alargado un sprint más de lo presupuestado.

Al igual que hacer el presupuesto, debemos añadir los costes para la empresa derivados de la contratación del personal:

$$3.861{,}70 * 1{,}3 = \mathbf{5.020{,}21\ €}$$

Por ello el coste total final del proyecto ha sido:

Costes directos	**Costes indirectos**	**Total (€)**
5.020,21	253,04	**5.273,25 €**

Por lo tanto, aunque se ha producido un desvío en horas, debido a que se introdujo el componente de Contingencias, el proyecto sigue siendo rentable.

Capítulo 5

Especificación de pruebas

Dentro del alcance del proyecto se contempla la implementación de una serie de pruebas que aseguran funcionalidades críticas para un sitio de comercio electrónico, como la identificación de usuarios o la existencia de productos. En función de la limitación temporal que tiene el Trabajo Fin de Grado, se ha considerado oportuno limitar la implementación a cuatro tests funcionales y tres no funcionales, que ejemplifican la dinámica de aplicación de pruebas a un sitio de estas características.

En este capítulo se detallarán los pasos que han de seguir los tests para ejecutarse, así como los datos que se le han de suministrar (si aplica), las precondiciones que se han de dar para poder llevarlo a cabo y los resultados esperados. Estas especificaciones se utilizarán como guía a la hora de implementar tanto las pruebas funcionales, como las no funcionales.

5.1. Pruebas funcionales

Para cada test se definen diferentes escenarios de prueba. Un escenario es una combinación concreta de precondiciones, datos de entrada, interacciones con el usuario y salidas esperadas, cuyo objetivo es analizar la respuesta del sistema ante estas condiciones y así validar que la funcionalidad implementada es correcta.

No todos los resultados de las pruebas deben ser "en positivo", pues ante determinada combinación de datos, el sistema debe negar una acción, mostrar un mensaje de error o realizar una secuencia de pasos alternativa a la principal. Es decir, *el resultado esperado es el error*.

Veremos esto más claramente con la prueba de *login*: el escenario principal implica que los datos de identificación son correctos, y se introducen de la manera adecuada, con lo que el usuario logra autenticarse en el sistema. Por otro lado, un flujo alternativo se correspondería con la introducción de datos incorrectos, que forzarían a que el sistema reaccionara de una forma diferente.

Para esta versión inicial de nuestra propuesta, únicamente se implementarán los escenarios principales de las funcionalidades consideradas, es decir, los que describen el flujo ideal en el que no se produce ningún error y tanto las entradas como las salidas son las correctas.

El resto de escenarios serán descritos en el lenguaje Gherkin para su integración, en el futuro, en una plataforma de automatización de pruebas basada en Cucumber. Como vimos en el Capítulo 3, Cucumber [4] es una herramienta de software que permite probar el comportamiento de un sistema definiendo las condiciones de prueba mediante su lenguaje Gherkin, con una apariencia similar al lenguaje natural, y que luego se transforma a código de forma automática.

5.1.1. Test: Respuesta HTTP

ID test	**01**
Escenario de test	Comprobar que la página web es accesible
Descripción	La página web debe ser accesible para prestar servicio. Además, este test se ejecuta de forma automática antes de cualquiera del resto de pruebas, puesto que dependen de poder acceder a la página.
Precondiciones	–
Datos de test	– URL: dirección de una página web
Pasos del test	1. Acceder a la página del sitio web. 2. Acceder al título de la página y buscar la palabra “Error”
Resultados esperados	No debe encontrar la palabra “Error”.

Tabla 5.1: Descripción de la prueba de Respuesta HTTP. Escenario 1

5.1.2. Test: Login

ID test	**02**
Escenario de test	Comprobar login de usuario con datos correctos
Descripción	Validar que el usuario hace login cuando se introducen unas credenciales válidas
Precondiciones	El usuario debe estar registrado en la base de datos
Datos de test	– Nombre de usuario: elia.parra.valverde@gmail.com – Contraseña: *mipassword*
Pasos del test	1. Acceder al formulario de acceso de la web. 2. Introducir el nombre / correo de usuario 3. Introducir la contraseña de usuario. 4. Pulsar el botón Iniciar sesión
Resultados esperados	El usuario se loguea en el sitio web

Tabla 5.2: Descripción de la prueba de Login: Escenario 1.

ID test	**03**
Escenario de test	Comprobar login de usuario con datos incorrectos
Descripción	Validar que el usuario no hace login cuando se introducen unas credenciales inválidas
Precondiciones	El usuario no debe estar registrado en la base de datos
Datos de test	– Nombre de usuario: elia.parra.valverde@gmail.com – Contraseña: *mipasswordincorrecta*
Pasos del test	1. Acceder al formulario de acceso de la web. 2. Introducir el nombre / correo de usuario 3. Introducir la contraseña de usuario. 4. Hacer click sobre el botón Iniciar sesión
Resultados esperados	El usuario no se loguea en el sitio web
Sintaxis para Gherkin	`FEATURE`: Login `SCENARIO` : Login con contraseña incorrecta `GIVEN`: el usuario introduce "*elia.parra.valverde@gmail*" en el campo Correo `AND`: el usuario introduce "mipasswordincorrecta" en el campo Contraseña `WHEN`: el usuario pulsa en el botón Iniciar Sesión `THEN` no se produce el login `AND` aparece mensaje de error

Tabla 5.3: Descripción de la prueba de Login: Escenario 2.

5.1.3. Test: Existencia de productos

ID test	**04**
Escenario de test	Comprobar que hay productos en una categoría que sí tiene productos
Descripción	Comprobación de la existencia de productos asociados a una categoría de producto en el *ecommerce*
Precondiciones	La categoría de productos debe existir y tener su página correspondiente.
Datos de test	–
Pasos del test	1. Acceder a la página de categoría de productos. 2. Buscar un elemento ficha de producto.
Resultados esperados	Encuentra algún elemento de productos

Tabla 5.4: Descripción de la prueba de Existencia de productos: Escenario 1.

ID test	**05**
Escenario de test	Comprobar que no hay productos en una categoría que no tiene productos asociados aún.
Descripción	Comprobación de la existencia del mensaje "No hay productos"
Precondiciones	La categoría de productos debe existir y tener su página correspondiente, aunque no debe haber productos asociados a ella
Datos de test	–
Pasos del test	1. Acceder a la página de categoría de productos. 2. Buscar el mensaje “No hay productos”.
Resultados esperados	Encuentra el mensaje de no existencia de productos.
Sintaxis para Gherkin	`FEATURE`: Existencia de productos `SCENARIO`: categoría vacía `GIVEN`: una URL de categoría donde no haya productos `WHEN`: se busca un producto `THEN` no encuentra producto `BUT` encuentra mensaje de error

Tabla 5.5: Descripción de la prueba de Existencia de productos: Escenario 2.

5.1.4. Test: Formulario de contacto

ID test	**07**
Escenario de test	Comprobar que el formulario de contacto no se envía correctamente.
Descripción	El sistema no debe permitir en el envío del formulario de contacto si no se han introducidos los campos requeridos. En este caso se omitirá el correo electrónico.
Precondiciones	–
Datos de test	– Nombre: Marta – Correo electrónico: (vacío) – Asunto: texto asunto – Mensaje: texto del mensaje.
Pasos del test	1. Acceder a la página de contacto del sitio web. 2. Rellenar el campo "Nombre" 3. Rellenar el campo "Correo electrónico". 4. Rellenar el campo "Asunto". 5. Pulsar sobre el botón Enviar.
Resultados esperados	Debe aparecer el mensaje de error "Uno o más campos tienen un error. Por favor revisa e inténtalo de nuevo." y no se debe enviar la consulta.
Sintaxis para Gherkin	`FEATURE`: Formulario de contacto `SCENARIO`: Formulario no se envía con datos incompletos `GIVEN`: el usuario introduce "Marta" en el campo Nombre `AND` el usuario introduce valor vacío en el campo correo `AND` el usuario introduce "texto asunto" en el campo Asunto `AND` el usuario introduce "texto del mensaje" en el campo Mensaje `WHEN`: hace click sobre el botón Enviar `THEN`: No se envía el formulario `AND` encuentra mensaje de error

Tabla 5.6: Descripción de la prueba de Formulario de contacto: Escenario 2.

ID test	**06**
Escenario de test	Comprobar que el formulario de contacto se envía correctamente.
Descripción	El sistema debe permitir en el envío del formulario de contacto si se han introducidos los campos requeridos. En este caso se introducirán todos los datos bien.
Precondiciones	–
Datos de test	– Nombre: Jorge – Correo electrónico: jorge.s@hotmail.com. – Asunto: texto asunto – Mensaje: texto del mensaje.
Pasos del test	1. Acceder a la página de contacto del sitio web. 2. Rellenar el campo "Nombre" 3. Rellenar el campo "Correo electrónico". 4. Rellenar el campo "Asunto". 5. Rellenar el campo "Mensaje". 6. Pulsar sobre el botón Enviar.
Resultados esperados	El mensaje se debe enviar.

Tabla 5.7: Descripción de la prueba de Formulario de contacto: Escenario 1.

5.2. Pruebas no funcionales

ID test	**08**
Tipo de test	Carga
Descripción	Se someterá a la web a una carga de 100 usuarios simultáneos.

ID test	**09**
Tipo de test	Estrés
Descripción	Se someterá a una carga de 100 usuarios en dos iteraciones.

ID test	**10**
Tipo de test	*Spike*
Descripción	Se someterá al sistema a una carga puntual de 1000 usuarios concurrentes durante 5 segundos, teniendo una base de carga de usuarios de 500.

Capítulo 6

Análisis de requisitos

6.1. Requisitos de negocio

6.1.1. Contexto

La prueba del software es una tarea imprescindible y costosa, en particular en lo referente al tiempo que consume. Por ello, la automatización de al menos parte de la carga de trabajo que conlleva proporcionaría un gran valor en el marco del desarrollo de cualquier proyecto software. Este valor presenta dos vertientes principales, con implicaciones tanto en el apartado de negocio, como en el puramente técnico y de gestión de proyectos.

Por una parte, los propietarios de los sitios de comercio electrónico necesitan una monitorización continua de sus webs para asegurar en todo momento tanto su disponibilidad, como un desempeño adecuado de sus funcionalidades principales. En general, esta monitorización se basará en la ejecución de una o varias pruebas de forma periódica, de manera que se verifique frecuentemente este buen funcionamiento. En caso de detectar una situación anómala, sería deseable restablecer el sitio web a su estado de funcionamiento óptimo lo antes posible. En caso contrario, podrían producirse pérdidas económicas o de otro tipo que supongan un perjuicio para el propietario. Para evitar esto, se valorará el establecimiento de un sistema de alerta inmediata para que los encargados del mantenimiento del sitio sean avisados en cuanto se detecte un problema.

Por otra parte, los probadores de software encargados del control de calidad del proyecto valorarían disponer de una herramienta que les permitiera automatizar de forma sencilla algunos de las pruebas básicas necesarias en estos sitios web. Esto no solo facilitaría su tarea, al proporcionarles un medio para ejecutar pruebas sobre diferentes tiendas virtuales con una mínima configuración, sino que les permitiría dedicarse a tareas de mayor calado dentro del proyecto y así proporcionar un mayor valor al cliente.

6.1.2. Objetivos de negocio

Los objetivos de negocio, económicos y empresariales que han sido identificados y que justifican el desarrollo de este proyecto son los descritos a continuación:

- NEG-01: Utilización de las pruebas como aseguramiento de la disponibilidad de la tienda cuando está en producción.
- NEG-02: Disponer de un registro de qué pruebas se han superado y cuáles no.
- NEG-03: Gestión sencilla de los tests.
- NEG-04: Visualización de los resultados, actualizados periódicamente.

6.1.3. Métricas de éxito

Las métricas de éxito permiten evaluar el grado de satisfacción de los requisitos de negocio, una vez puesto en funcionamiento el sistema y después de un periodo de estudio estadístico. Son importantes porque, viendo el nivel de éxito de la plataforma, se podrían considerar dos escenarios: abandonar el proyecto si no se han satisfecho dichos objetivos de negocio, o incrementar las capacidades del sistema si han sido alcanzados.

- ME-01: Desde el punto de vista del equipo técnico, se considerará que el proyecto ha sido un éxito si consigue avisar de cualquier fallo, mediante envío de correo electrónico, al responsable designado en el plazo máximo de 1 min desde que se ha detectado la incidencia.
- ME-02: Desde el punto de vista del cliente dueño de una tienda virtual, se considerará que el programa ha satisfecho sus expectativas si, en caso de caída de la tienda virtual, el tiempo de reacción es inferior a 20 min.
- ME-03: Se considerará un éxito tener a disposición una herramienta que permita obtener una visualización actualizada, con un retraso de como máximo 10 segundos, de los últimos 20 tests de cada tipo.

6.1.4. Riesgos

El principal riesgo de negocio es no poder realizar todos los módulos planteados, pues la sección de gestión de tests y monitorización requiere tiempo y esfuerzo.

6.1.5. Características principales

Las características principales del proyecto son las secciones en las que se divide el sistema de una forma lógica externa. En este caso se han identificado 2:

- CP-01: *Gestión del proceso de automatización: Aplicación Catón.*
  - CP-01.1: *Gestión de Clientes.*
    Con esta característica se podrá dar de alta nuevos clientes junto con sus webs asociadas, así como modificar el nombre o el estado (activo/inactivo) del cliente en el sistema.
  - CP-01.2: *Gestión de Pruebas.*
    Mediante esta característica se podrá ver un listado de todos las pruebas de cada tipo que se encuentran programadas para su ejecución, añadir nuevas pruebas o modificar las ya existentes.
  - CP-01.3: *Visualización de datos.*
    Con esta característica se da al usuario la opción de consultar información agregada de los resultados de las pruebas realizadas a un sitio web, en un rango de fechas de su elección.
  - CP:01.4: *Monitorización de sitios web.*
    Esta característica permite el seguimiento continuo de las pruebas que se están realizando. Se pueden ver los resultados de las últimas pruebas realizadas.
- CP-02: *Automatización de los tests*
  - CP-02.1: *Acceso a URL.*
    Esta característica se corresponde con el test de respuesta HTTP. Su objetivo es acceder a una URL para comprobar si está activa y es accesible.
  - CP-02.2: *Comprobación de existencia de productos.*
    Esta característica implementa el test de existencia de productos. Su objetivo es verificar que, para una URL de página de categoría de producto, existen productos de esa categoría, es decir, un visitante puede visualizarlos.
  - CP-02.3: *Comprobación de funcionamiento de login.*
    Esta característica se corresponde con la prueba de login. Para ello, el sistema simulará la introducción de credenciales de autenticación y verificará que se ha realizado correctamente el login en el sitio web.
  - CP-02.4: *Comprobación de disponibilidad de formulario de contacto.*
    Esta característica implementa la prueba de formulario de contacto. Con ella, se quiere comprobar que el formulario está disponible y funciona correctamente. Para ello, se simulará la introducción de valores de prueba en los campos exigidos y posteriormente se comprobará que el botón de envío se pueda pulsar.

Con el fin de esclarecer las características principales descritas, es útil un Árbol de Características [35]. En él se representan todos los componentes y las jerarquías que hay en cada uno de ellos. La línea central representa el proyecto en su conjunto y, de ella, parten las ramas propias de cada característica.

Para facilitar la legibilidad de estos diagramas, se ha considerado oportuno describir cada característica principal en un árbol aparte. Además, estas características son independientes

y solo interactúan entre sí de forma indirecta, a través de la base de datos, por lo que no tienen funcionalidad en común. En la Figura 6.1 se muestra el árbol del sistema Catón y en la Figura 6.2 el del sistema que ejecutará los tests.

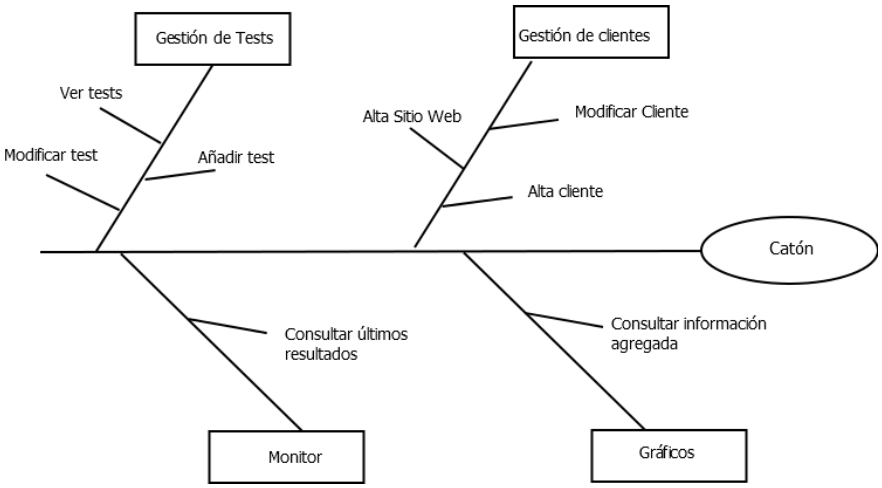


Figura 6.1: Árbol de características. Catón

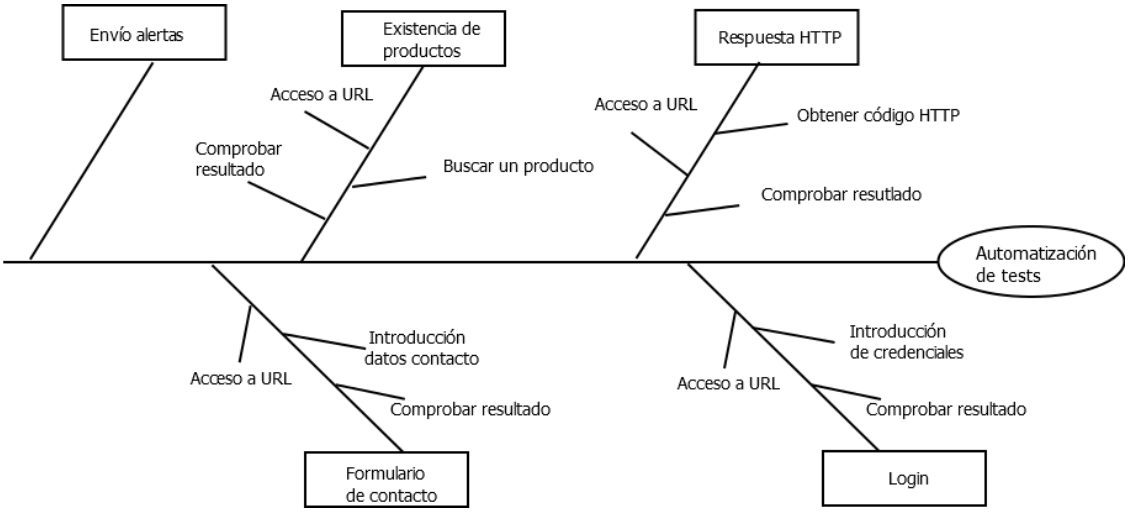


Figura 6.2: Árbol de características. Implementación de los tests

6.1.6. Alcance

La release inicial (o entrega inicial) del proyecto comprenderá las características de configuración de los test y su posterior ejecución junto con el envío de *emails* al equipo de desarrollo, pudiendo quedar excluida, por motivos de limitación temporal, la realización del dashboard de visualización de resultados.

6.1.7. Limitaciones

- No se contempla realizar más test funcionales, ni no funcionales en esta iteración de producto.
- No se contempla la integración con ningún *software* de gestión de *bugs*.

6.1.8. Perfiles de los *stakeholders*

En esta sección se enumerarán los diferentes *stakeholders* participantes en el proyecto y, posteriormente, para cada uno de ellos se especificarán 4 campos: cuál es el mayor valor que proporciona el sistema al *stakeholder*, las actitudes de éste hacia el sistema, sus intereses principales y las restricciones que impone sobre el sistema.

1. **Tester** (usuario): se trata de cualquier perfil técnico encargado de la ejecución de las pruebas funcionales y no funcionales para un sitio web.
2. **Jefe de Calidad del software** (usuario): es la persona encargada de la gestión y reporte al equipo de desarrollo de todos los defectos encontrados en la tienda y la responsable, en última instancia, de la calidad del software que se vaya a hacer entrega al cliente final.
3. **Jefe del Departamento de Desarrollo** (no usuario): es la persona responsable de todo el equipo técnico de la empresa proveedora de servicios, la que se encarga de realizar los presupuestos para nuevos productos.
4. **Jefe del Departamento de Cuentas** (no usuario): este departamento se encarga de la relación con el cliente final y de ir informándole en todo momento de los avances del proyecto, cuando éste está en desarrollo, o de la disponibilidad y rendimiento del sitio web cuando ya ha salido a producción.
5. **Cliente final** (no usuario): se suele tratar el propietario de la tienda virtual. Su objetivo es aumentar las ventas en su sitio web, para lo cual es primordial que el *site* esté disponible.

Stakeholder	**STK-01: Tester**
Mayor valor	Ahorro de tiempo en la ejecución de pruebas.
Actitudes	Realización de pruebas más sencilla
Intereses principales	Ahorro de tiempo en la ejecución de pruebas
Restricciones	Debe cubrir la mayor parte de las excepciones posibles

Tabla 6.1: Descripción del *stakeholder* Tester.

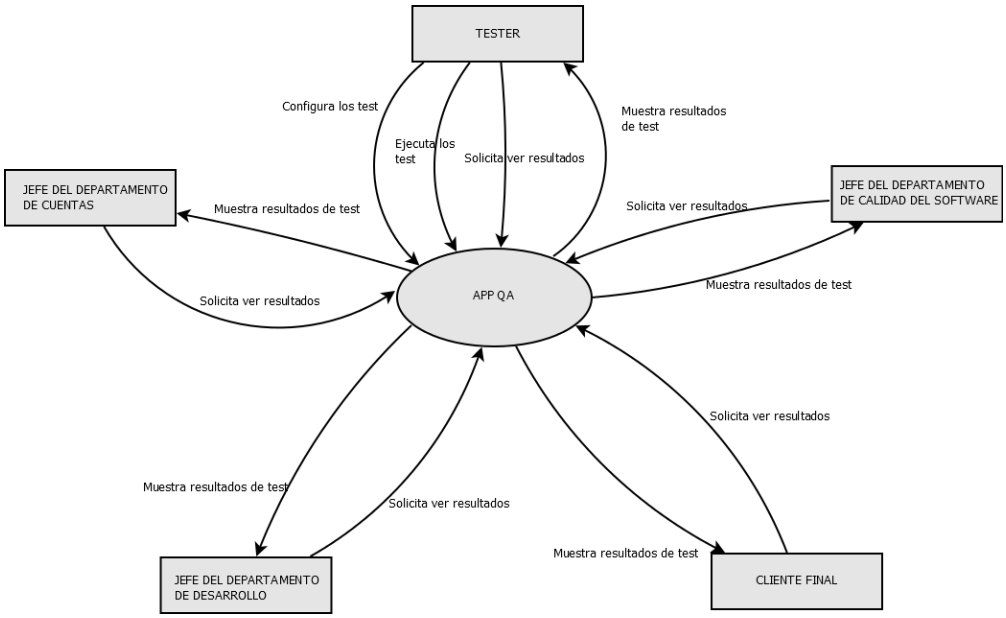


Figura 6.3: Diagrama de contexto

Stakeholder	**STK-02: Jefe de Calidad del Software**
Mayor valor	Gestión más sencilla de los defectos encontrados
Actitudes	Reporte más rápido al equipo de desarrollo
Intereses principales	Ahorro de tiempo y mejor descripción de los defectos encontrados
Restricciones	Debe cubrir la mayor parte de las excepciones posibles

Tabla 6.2: Descripción del *stakeholder* Jefe de Calidad del Software.

Stakeholder	**STK-03: Jefe de Desarrollo**
Mayor valor	Seguridad del correcto funcionamiento de los productos desarrollados
Actitudes	Reporte más rápido al equipo de desarrollo
Intereses principales	Poder incluir en los presupuestos una cantidad referente a la calidad del software entregado
Restricciones	Debe cubrir la mayor parte de las excepciones posibles

Tabla 6.3: Descripción del *stakeholder* Jefe de Desarrollo.

6.1.9. Consideraciones de desarrollo

En esta sección del documento resumimos las consideraciones a tener en cuenta a la hora de desarrollar el sistema. Nos referiremos a ellas mediante un identificador CD-xx (con xx un número incremental):

1. CD-01: El sitio de comercio electrónico de prueba, se desplegará en un servidor Apache.
2. CD-02: La tienda virtual se construirá con la plataforma WooCommerce.

Stakeholder	**STK-04: Jefe de Cuentas**
Mayor valor	Conocimiento de la situación de los sitios web de los clientes
Actitudes	Comunicación proactiva hacia el cliente
Intereses principales	Entregar al cliente información precisa de qué ha ocurrido cuando la tienda se ha caído
Restricciones	Debe cubrir la mayor parte de las excepciones posibles

Tabla 6.4: Descripción del *stakeholder* Jefe de Cuentas.

Stakeholder	**STK-05: Cliente final**
Mayor valor	Conocimiento del estado de su sitio web
Actitudes	Conocer los puntos donde suele fallar su comercio
Intereses principales	Detección precoz de una caída o mal funcionamiento, para así mitigar las pérdidas económicas
Restricciones	Debe cubrir la mayor parte de las excepciones posibles

Tabla 6.5: Descripción del *stakeholder* Cliente final.

Stakeholder	**STK-06: Correo electrónico**
Mayor valor	Envío de avisos al equipo de desarrollo cuando detecte que algo no ha funcionado bien en la web. Interfaz con la que el sistema se comunica
Actitudes	—
Intereses principales	El responsable de testing debe tener unas credenciales para autenticarse en el sistema gestor de correo.
Restricciones	Accesibilidad y usabilidad

Tabla 6.6: Descripción del *stakeholder* Correo electrónico.

3. CD-03: El *dashboard* final, se construirá con HTML, CSS y las librerías JQuery y ChartJS
4. CD-04: Las pruebas se realizarán con el navegador Chrome, por lo que el driver de Selenium será "ChromeDriver".
5. CD-05: La aplicación se realizará en Java 8.
6. CD-06: En la sección de monitorización, debe haber un refresco suficiente de los datos para que se muestren todos los resultados.

6.2. Requisitos de usuario

En este apartado se van a enumerar los requisitos de usuario que se han extraído de la descripción de los casos de uso. Se han agrupado según la característica del sistema a la que pertenezcan.

Identificad. de requisito	Descripción
USR-01	El usuario tester podrá dar de alta un cliente en la plataforma
USR-02	El usuario tester podrá modificar los datos del cliente.
USR-03	El usuario tester podrá dar de alta un nuevo sitio web en la plataforma.
USR-04	El usuario tester podrá añadir un test
USR-05	El usuario tester podrá ver qué tests están registrados en el sistema.
USR-06	El usuario tester podrá modificar los datos de cualquier test
USR-07	El usuario tester podrá consultar información agregada sobre los resultados de los tests.
USR-08	El usuario tester podrá consultar los resultados de los últimos test realizados
USR-09	El sistema podrá acceder a la URL de un sitio Web
USR-10	El sistema cliente podrá comprobar que hay productos disponibles en una URL
USR-11	El sistema cliente podrá comprobar que el login funciona correctamente
USR-12	El sistema cliente podrá comprobar que el formulario de contacto está disponible

Tabla 6.7: Requisitos de usuario.

6.2.1. Especificación. Casos de uso.

Un caso de uso es una descripción de las acciones que puede realizar el usuario con el sistema. Se representan gráficamente mediante un Diagrama de Casos de Uso. En este caso, se ha decidido que cada caso de uso se corresponda con un requisito de usuario.

La Figura 6.4 representa el diagrama de casos de uso definido para el sistema. Recoge los flujos de interacción entre los diferentes componentes del sistema y los *stakeholders* usuarios del mismo. A continuación, se describen los diferentes casos de uso, separados dependiendo del componente del que forman parte.

6.2.1.1. Catón

UC-01	Alta Cliente
Versión	1.0
Requisitos asociados	USR-01: El usuario tester podrá dar de alta un cliente en la plataforma
Descripción	El usuario tester podrá introducir el nombre del cliente y activarlo/desactivarlo
Secuencia normal	1. El usuario introduce el nombre del cliente 2. El sistema guarda el nombre del cliente y su estado Activo / Desactivado.
Postcondición	El sistema tiene guardado el nombre y el estado
Excepciones	El sistema valida que el nombre no sea vacío

Tabla 6.8: Caso de uso UC-01: Alta Cliente.

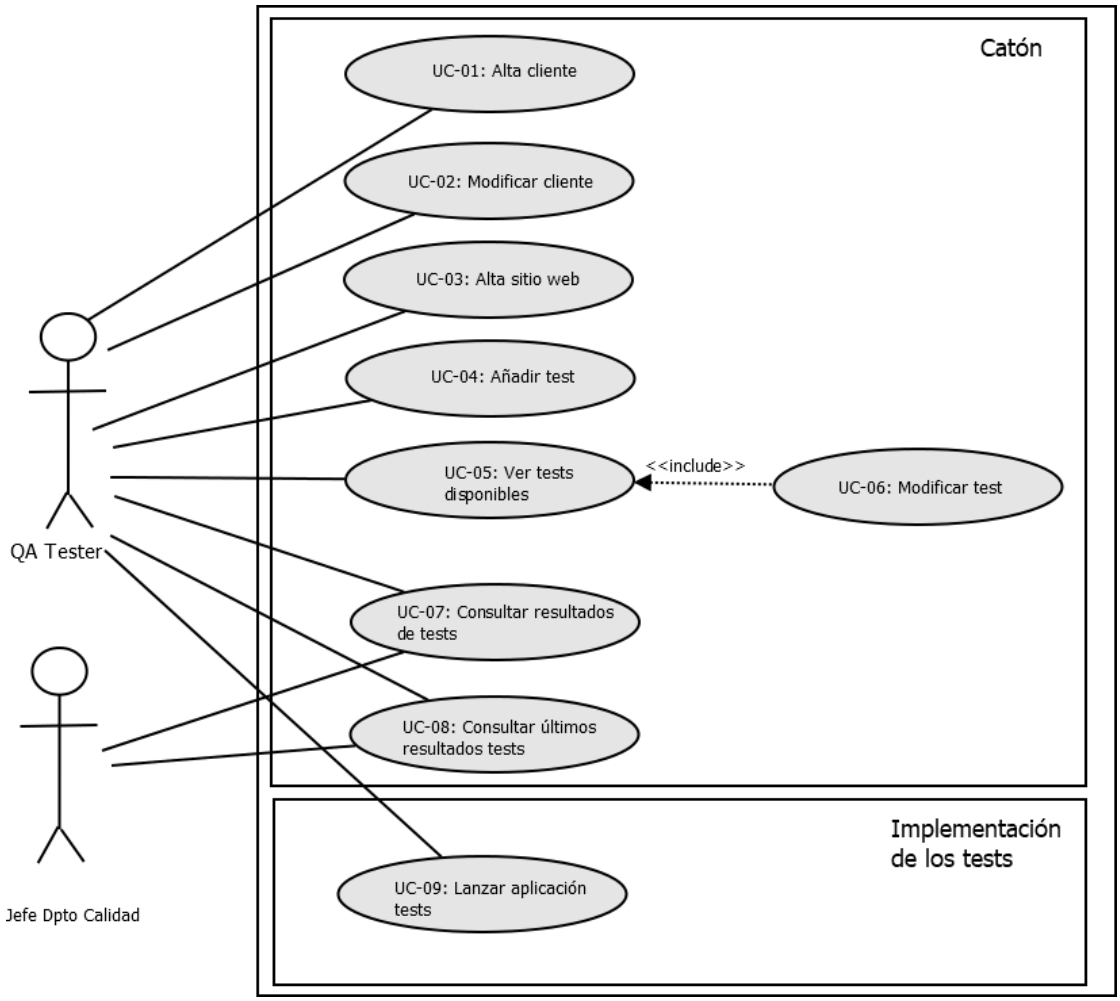


Figura 6.4: Diagrama de Casos de Uso

UC-02	**Modificar Cliente**
Versión	1.0
Requisitos asociados	USR-02: El usuario tester podrá modificar los datos del cliente.
Descripción	El usuario tester podrá modificar el nombre del cliente y/o cambiar el estado.
Precondición	El cliente a modificar ya existe en la base de datos
Secuencia normal	1. El usuario cambia el nombre del cliente o su condición de activo 2. El sistema guarda los datos nuevos.
Postcondición	El sistema tiene actualizado el nombre y/o estado del cliente
Excepciones	El sistema valida que el nombre nuevo no sea vacío.

Tabla 6.9: Caso de uso UC-02: Modificar Cliente.

UC-03	**Alta Sitio Web**
Versión	1.0
Requisitos asociados	USR-03: El usuario tester podrá dar de alta un nuevo sitio web en la plataforma.
Descripción	El usuario tester podrá dar de alta un nuevo sitio web e informa de qué tests se realizarán sobre esa URL
Secuencia normal	1. El usuario introduce la URL del sitio 2. El usuario especifica si activa esa URL 3. El usuario especifica el cliente propietario de esa URL 4. El usuario informa de qué pruebas se van a realizar 5. El sistema guarda los datos.
Postcondición	El sistema tiene un sitio web nuevo guardado
Excepciones	

Tabla 6.10: Caso de uso UC-03: Alta Sitio Web.

UC-04	**Añadir Test**
Versión	1.0
Requisitos asociados	USR-04: El usuario tester podrá añadir un test.
Descripción	El usuario tester podrá añadir una instancia de test perteneciente a uno de los tipos de tests existentes.
Precondición	
Secuencia normal	1. El usuario introduce los datos requeridos para la configuración de cada tipo de test, de acuerdo con los requisitos de información identificados para ese test. 2. El sistema guarda el nuevo test.
Postcondición	El sistema tiene registrado un nuevo test
Excepciones	

Tabla 6.11: Caso de uso UC-04: Añadir Test.

UC-05	**Ver Tests disponibles**
Versión	1.0
Requisitos asociados	USR-05: El usuario tester podrá ver qué tests están registrados en el sistema.
Descripción	El usuario tester podrá ver qué tests están registrados y activos en el sistema, según su tipo
Precondición	Hay tests guardados
Secuencia normal	1. El usuario requiere la lista de test 2. El sistema muestra la lista de tests.
Postcondición	
Excepciones	

Tabla 6.12: Caso de uso UC-05: Ver Tests disponibles.

UC-06	**Modificar Test**
Versión	1.0
Requisitos asociados	USR-06: El usuario tester podrá modificar los datos de cualquier test.
Descripción	El usuario tester podrá modificar los datos necesarios para la ejecución de cualquier test.
Precondición	El test existe en el sistema
Secuencia normal	1. El usuario introduce los nuevos valores de los datos de configuración de cada tipo de test de acuerdo a los requisitos de información. 2. El sistema guarda los datos.
Postcondición	El sistema tiene los datos de ejecución de test actualizados.
Excepciones	

Tabla 6.13: Caso de uso UC-06: Modificar Test.

UC-07	**Consultar información agregada de los resultados de los tests**
Versión	1.0
Requisitos asociados	USR-07: El usuario tester podrá consultar información agregada sobre los resultados de los tests.
Descripción	El usuario tester podrá consultar información agregada sobre los resultados de los tests para cada sitio web registrado y en un rango de fechas de su elección.
Precondición	
Secuencia normal	1. El usuario solicita visualizar los resultados agregados de las pruebas para un sitio web determinado en un rango de fechas. 2. El sistema muestra la información.
Postcondición	
Excepciones	

Tabla 6.14: Caso de uso UC-07: Consultar información agregada de los resultados de los tests.

UC-08	**Consultar últimos resultados de los tests**
Versión	1.0
Requisitos asociados	USR-08: El usuario tester podrá consultar los resultados de los últimos test realizados.
Descripción	El usuario tester podrá visualizar los resultados de los últimos tests realizados
Precondición	
Secuencia normal	1. El usuario solicita los resultados de los últimos tests realizados. 2. El sistema muestra la información.
Postcondición	
Excepciones	

Tabla 6.15: Caso de uso UC-08: Consultar últimos resultados de los tests.

6.2.1.2. Implementación de los tests.

UC-09	**Lanzar ejecución de tests**
Versión	1.0
Requisitos asociados	USR-09: El usuario podrá iniciar la ejecución automática de las pruebas.
Descripción	El usuario podrá ejecutar el programa de manera que el sistema lance de forma periódica las pruebas registradas en su base de datos
Precondición	1.- Existe acceso a la base de datos 2.- Hay pruebas registradas
Secuencia normal	1. El usuario lanza la aplicación de pruebas automáticas. 2. El sistema comienza su ejecución.
Postcondición	
Excepciones	El usuario ordena la interrupción activa de los tests. El sistema detiene la ejecución de las pruebas y se cierra.

Tabla 6.16: Caso de uso UC-09: Lanzar ejecución de tests.

6.3. Requisitos funcionales

6.3.1. Catón.

En esta sección se listan los requisitos funcionales para la aplicación Catón (ver Tabla 6.17). En todos los casos en que se haga mención a una entidad descrita en los requisitos de información, se estará aludiendo a los atributos identificados para dicha entidad exceptuando los identificados internos generados automáticamente por el sistema.

6.3.2. Implementación de los tests.

En esta sección se enumeran los requisitos funcionales que deberá tener la aplicación que ejecute las pruebas, según se muestra en la Tabla 6.18.

ID	Descripción
FUN-01	El sistema permitirá la introducción de los datos de un cliente mediante un formulario
FUN-02	El sistema validará los datos introducidos
FUN-03	El sistema guardará los datos del cliente
FUN-04	El sistema mostrará los datos del cliente
FUN-05	El sistema actualizará los datos del cliente
FUN-06	El sistema permitirá la introducción de los datos de un sitio web
FUN-07	El sistema guardará los datos del sitio web
FUN-08	El sistema permitirá la introducción de los datos necesarios para configurar una prueba
FUN-09	El sistema guardará los datos del nuevo test
FUN-10	El sistema mostrará un listado de todos los tests disponibles
FUN-11	El sistema mostrará los datos de un test
FUN-12	El sistema actualizará los datos del test
FUN-13	El sistema permitirá seleccionar el sitio web y un rango de fechas
FUN-14	El sistema mostrará los resultados agregados de las pruebas
FUN-15	El sistema se refrescará cada 10 segundos
FUN-16	El sistema mostrará un resumen de los últimos resultados correctos y fallidos

Tabla 6.17: Requisitos funcionales de Catón.

ID	Descripción
FUN-17	El sistema hará una petición al servidor del sitio web
FUN-18	El sistema comprobará el código HTTP que devuelve el servidor
FUN-19	El sistema enviará un correo si el test falla
FUN-20	El sistema buscará un producto en la URL
FUN-21	El sistema introducirá unas credenciales válidas
FUN-22	El sistema clicará sobre el botón de inicio de sesión
FUN-23	El sistema buscará el elemento de comprobación de inicio de sesión
FUN-24	El sistema introducirá todos los valores en los campos del formulario
FUN-25	El sistema comprobará que el botón de “Enviar” del formulario sea clicable
FUN-26	El sistema ejecutará los tests con una periodicidad definida
FUN-27	El sistema guardará los resultados de todos los tests.
FUN-28	El sistema consultará los parámetros necesarios para la realización de los test
FUN-29	El sistema respetará los tiempos de espera necesarios para asegurar la correcta carga de los sitios web
FUN-30	El sistema enviará un correo en caso de que ocurra alguna excepción durante la ejecución de los tests

Tabla 6.18: Requisitos funcionales de los tests implementados.

6.4. Requisitos de información

En esta sección se especifican los requisitos de información extraídos de las características exigidas para el sistema. Es importante delimitar estos requisitos para tener una especificación exacta de qué datos y tipos de datos está manejando la plataforma, así como las restricciones que lleven aparejadas y sirvan para mantener la consistencia de la base de datos.

6.4.1. Modelo de datos.

El diagrama Entidad - Relación es una representación del mini-ecosistema que describe los datos que van a ser gestionados por la aplicación y las relaciones entre ellos. La representación de nuestro sistema de información puede verse en el diagrama de la Figura 6.5. Para mejorar la legibilidad, los atributos de cada entidad o relación se definen más adelante, en el diccionario de datos.

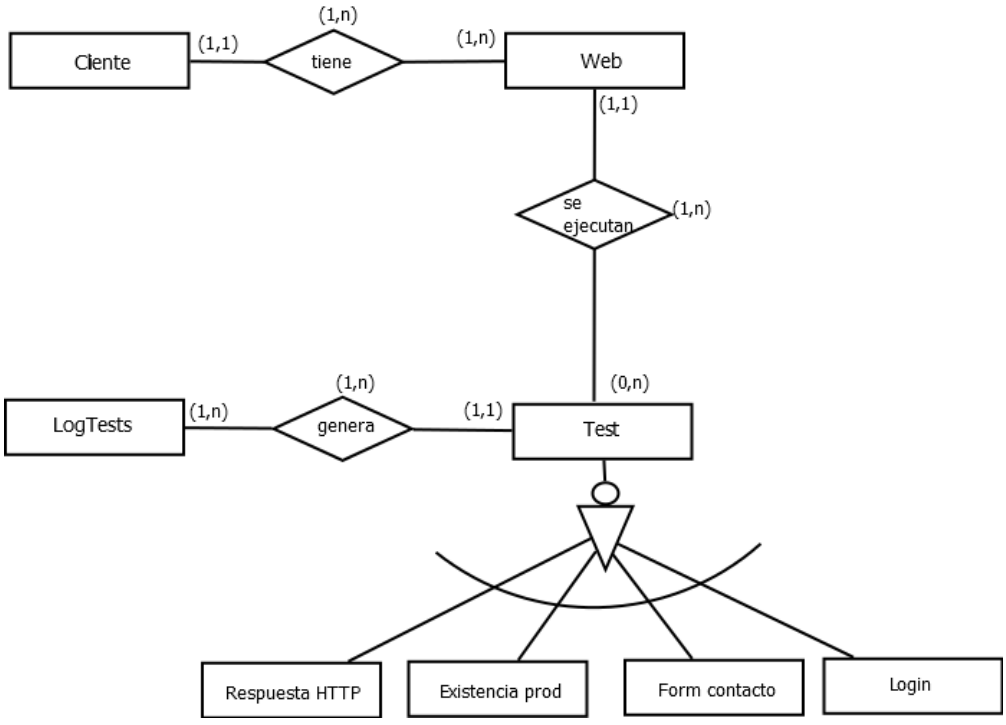


Figura 6.5: Modelo Entidad - Relación

Consideraciones sobre la jerarquía:

La relación de jerarquía es exclusiva. Cada instancia de test solo puede ser de uno de los subtipos previstos (HttpResponse, ExistenciaProductos, FormularioContacto, Login).

Además, es total porque todas las entidades son de alguno de los subtipos especializados (no hay posibilidad de que no pertenezcan a ninguno)

6.4.2. Diccionario de datos.

Un diccionario de datos muestra las características lógicas de los datos que utilizará el sistema final. Se incluye información como el nombre, el tipo de dato, los valores de datos permitidos y sus restricciones y una descripción.

En este caso, se ha optado por dividirlo en tablas que se corresponden con las entidades del esquema Entidad-Relación.

Cliente			
Nombre	**Tipo**	**Dominio**	**Descripción**
Activo	Boolean	(0,1) NOT NULL	El cliente está activo o no en la plataforma
Id	Integer	(>0) NOT NULL	Identificador único del cliente
NombreCliente	Cadena	NOT NULL	Nombre del cliente

Tabla 6.19: Atributos de la entidad Cliente.

Web			
Nombre	**Tipo**	**Dominio**	**Descripción**
Activo	Boolean	(0,1) NOT NULL	La web está activa o no en la plataforma
IdCliente	Integer	(>0) NOT NULL	Identificador del cliente
IdWeb	Integer	(>0) NOT NULL	Identificador de la web
Url	Cadena	NOT NULL	Url de la página web
PruebaLogin	Boolean	(0,1) NOT NULL	Determina si se realiza la prueba de login
PruebaEP	Boolean	(0,1) NOT NULL	Determina si se realiza la prueba de existencia de productos
PruebaFC	Boolean	(0,1) NOT NULL	Determina si se realiza la prueba de formulario de contacto
PruebaHR	Boolean	(0,1) NOT NULL	Determina si se realiza la prueba de respuesta HTTP

Tabla 6.20: Atributos de la entidad Web.

Log Tests			
Nombre	**Tipo**	**Dominio**	**Descripción**
timeStamp	TimeStamp	(>0) NOT NULL	Fecha y hora en la que se realizó el test
resultado	Boolean	(0,1) NOT NULL	Resultado del test
mensaje	Cadena	NOT NULL	Mensaje explicativo del resultado del test.
nombrePrueba	Cadena	Enumerado NOT NULL	Nombre del test realizado Valores: (Respuesta HTTP, Existencia productos, Formulario Contacto, Login)

Tabla 6.21: Atributos de la entidad Log Tests.

Test			
Nombre	**Tipo**	**Dominio**	**Descripción**
Activo	Boolean	(0,1) NOT NULL	El test está activo o no
Url	Cadena	NOT NULL	Url de la página web
htUser	Cadena		Usuario del acceso del HTTP
htPass	Cadena		Contraseña del acceso del HTTP
idTest	Integer	(>0) NOT NULL	Identificador único del test
IdWeb	Integer	(>0) NOT NULL	Identificador de la web
Tipo	Cadena	Enumerado NOT NULL	Indica el tipo de test. Valores: (Respuesta HTTP, Existencia productos, Formulario Contacto, Login)

Tabla 6.22: Atributos de la entidad Test.

Test Existencia de Productos			
Nombre	**Tipo**	**Dominio**	**Descripción**
metodoLocalizador Producto	Cadena	NOT NULL	Elemento o etiqueta HTML utilizado para localizar el producto
localizadorProducto	Cadena	NOT NULL	Valor del elemento o etiqueta que debe tener el elemento de búsqueda del producto

Tabla 6.23: Atributos de la entidad Test Existencia de Productos.

Test Login			
Nombre	**Tipo**	**Dominio**	**Descripción**
correo	Cadena	NOT NULL	Texto que se introducirá en el campo Correo
pass	Cadena	NOT NULL	Texto que se introducirá en el campo Contraseña
metodoLocalizadorCorreo	Cadena	NOT NULL	Método por el que se localizará el campo Correo
localizadorCorreo	Cadena	NOT NULL	Localizador del campo Correo
metodoLocalizadorPass	Cadena	NOT NULL	Método por el que se localizará el campo Contraseña
localizadorPass	Cadena	NOT NULL	Localizador del campo Contraseña
metodoLocalizadorBoton	Cadena	NOT NULL	Método por el que se localizará el botón Iniciar Sesión
localizadorBoton	Cadena	NOT NULL	Localizador del botón Iniciar Sesión
metodoLocalizadorComprobacion	Cadena	NOT NULL	Método por el que se localizará el elemento de comprobación de login
localizadorComprobacion	Cadena	NOT NULL	Localizador del elemento de comprobación de login
textoComprobacion	Cadena	NOT NULL	Texto por el que se comprobará el login

Tabla 6.24: Atributos de la entidad Test Login.

Test Formulario de Contacto (I)			
Nombre	**Tipo**	**Dominio**	**Descripción**
valorNombre	Cadena		Texto a introducir en el campo Nombre
valorCorreo	Cadena		Texto a introducir en el campo Correo
valorTelefono	Cadena		Texto a introducir en el campo Teléfono
valorComentario	Cadena		Texto a introducir en el campo Comentario
valorEmpresa	Cadena		Texto a introducir en el campo Empresa
valorApellido	Cadena		Texto a introducir en el campo Apellido
valorPais	Cadena		Texto a introducir en el campo País

Tabla 6.25: Atributos de la entidad Test Formulario de Contacto (I).

Test Formulario de Contacto (II)			
Nombre	**Tipo**	**Dominio**	**Descripción**
metodoLocalizador Nombre	Cadena		Elemento o etiqueta HTML por la que se localizará el campo Nombre
localizadorNombre	Cadena		Localizador del campo Nombre
metodoLocalizador Apellido	Cadena		Método por el que localizar el campo Apellido
localizadorApellido	Cadena		Localizador del campo Apellido
metodoLocalizador Empresa	Cadena		Método por el que localizar el campo Empresa
localizadorEmpresa	Cadena		Localizador del campo Empresa
metodoLocalizador Correo	Cadena		Método por el que localizar el campo Correo
localizadorCorreo	Cadena		Localizador del campo Correo
metodoLocalizador Pais	Cadena		Método por el que localizar el campo País
localizadorPais	Cadena		Localizador del campo País
metodoLocalizador Departamento	Cadena		Método por el que localizar el campo Departamento
localizador Departamento	Cadena		Localizador del campo Departamento
metodoLocalizador Telefono	Cadena		Método por el que localizar el campo Teléfono
localizadorTelefono	Cadena		Localizador del campo Teléfono
metodoLocalizador Comentario	Cadena		Método por el que localizar el campo Comentario
localizador Comentario	Cadena		Localizador del campo Comentario
metodoLocalizador AceptoPolitica	Cadena		Método por el que localizar el checkbox de Aceptar Política
localizador AceptoPolitica	Cadena		Localizador del checkbox de Aceptar Política
metodoLocalizador Boton	Cadena		Método por el que localizar el botón de Enviar
localizadorBoton	Cadena		Localizador del botón de Enviar

Tabla 6.26: Atributos de la entidad Test Formulario de Contacto (II).

6.5. Requisitos no funcionales

Debido al hecho de que se trata de un prototipo de una herramienta interna, los requisitos no funcionales no son muchos. A medida que se fuera aumentando la funcionalidad y/o número de personas que lo utilizaran del sistema éstos se podrían ver aumentados.

ID	Descripción
NFUN-01	El sistema debe ser sencillo de aprender para un usuario sin experiencia previa con la aplicación, de manera que debería familiarizarse con ella en unos minutos.
NFUN-02	Debe facilitarse la navegabilidad por la aplicación mediante el uso de metáforas visuales adecuadas (p.ej. iconos representativos de las funciones).
NFUN-03	El sistema debe responder visualmente a las acciones de usuario (p. ej. al colapsar el menú), especialmente en las esperas de carga si las hubiera.
NFUN-04	El sistema debe responder de forma ágil a las acciones de usuario (p.ej. usando técnicas asíncronas de actualización de interfaz como AJAX).
NFUN-05	El tamaño de textos en pantalla debe ser adecuado para su visualización en pantallas de diferentes resoluciones.
NFUN-06	El sistema deberá adaptar la interfaz automáticamente para ser funcional en dispositivos móviles (debe ser *responsive*).

Tabla 6.27: Requisitos no funcionales del sistema.

Capítulo 7

Diseño del sistema.

7.1. Arquitectura física

Para el despliegue de todos los elementos que componen el proyecto, se necesita una estructura dispuesta como se indica en la Figura 7.1, en los que podemos encontrar los siguientes elementos:

- Un servidor de aplicaciones web donde instalar la aplicación Catón, a la cual se conectará el usuario final desde la capa de cliente mediante una URL de acceso.
- Un servidor provisto de una máquina virtual de Java (JVM) donde se ejecutará la batería de tests basados en Selenium, y desde el que se realizarán las peticiones correspondientes remotos que alojan los sitios webs a testear.
- Un servidor de bases de datos al que consultarán las dos aplicaciones anteriores y donde se almacenarán tanto los datos de configuración de los tests, como los resultados obtenidos de ejecutar cada uno de ellos.
- Un servidor donde se instalará el sitio web demo de WooCommerce (componente superior en la Figura 7.1). Este elemento es representativo, además, de cualquier otro sitio web que estemos sometiendo a testeo, y al que accedemos por medio de Internet.
- Un cliente que realizará peticiones al servidor de aplicaciones web donde se encuentra desplegado Catón.

7.2. Arquitectura lógica

El diseño lógico del sistema es la estructura y las relaciones existentes entre los diferentes componentes lógicos, de funcionalidad o de prestación de servicios que integran o interactúan

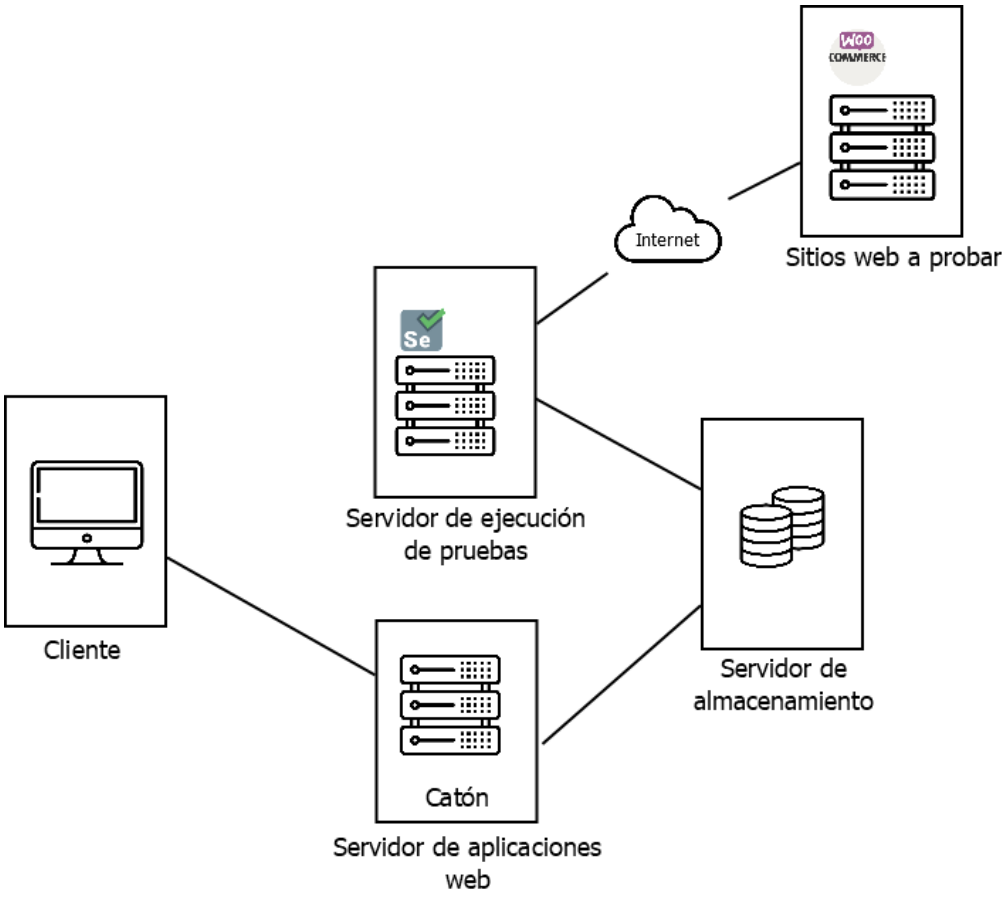


Figura 7.1: Arquitectura física

con nuestro sistema, así como las tecnologías que implementan. Para este proyecto, se ha diseñado la estructura mostrada en la Figura 7.2, con los elementos descritos a continuación.

- Cliente: se utiliza HTML para dar estructura a las páginas webs de las que está compuesta la aplicación Catón y CSS para definir los estilos de las mismas. Además, se hace uso de JavaScript (en concreto JQuery y AJAX) para la realización de los gráficos en la aplicación Catón, mediante la librería CharJS. Para ejecutar e interactuar con la aplicación, el cliente necesitará contar con un navegador compatible que disponga de Javascript.
- Servidor de aplicaciones web: implementa Apache Server con PHP. Se encarga de servir la aplicación para que sea accesible para el cliente. Además, implementa la lógica necesaria para interactuar con la base de datos con el fin de almacenar y consultar tanto los parámetros de configuración de las pruebas, como los resultados de las mismas.
- MySQL: su cometido es guardar la información de parametrización necesaria para ejecutar los test, así como almacenar los logs de resultados generados. También recibe y gestiona las consultas que recibe de los dos servidores del sistema que interactúan

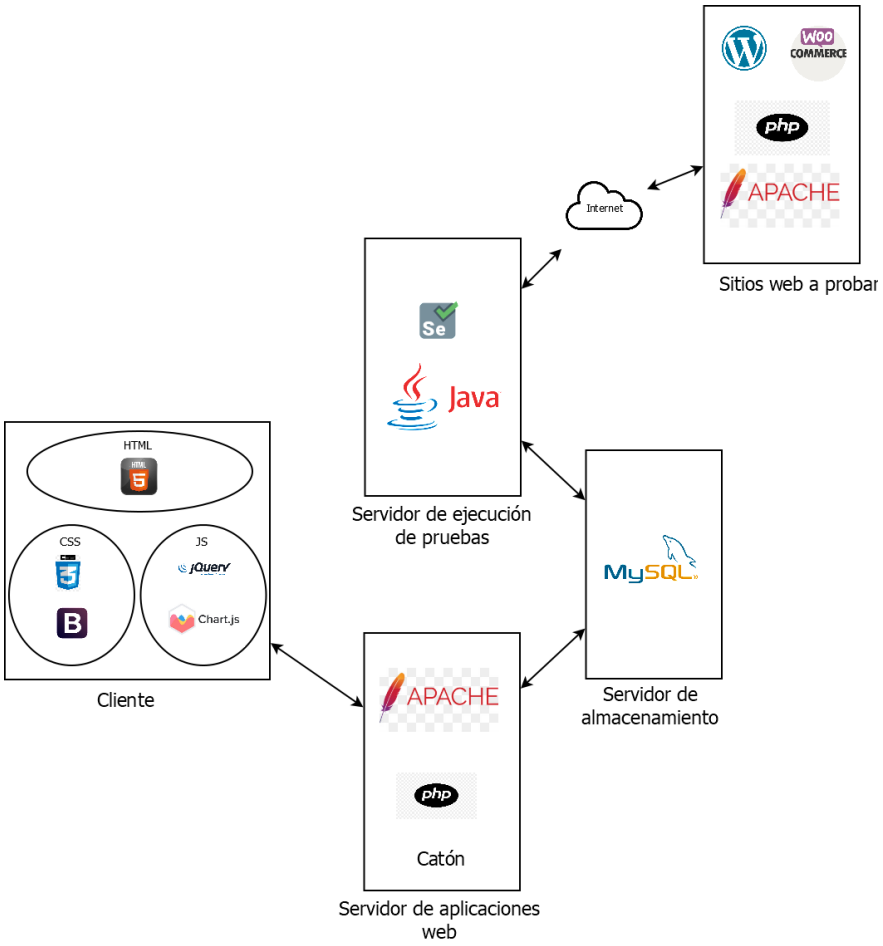


Figura 7.2: Arquitectura lógica

con la base de datos.

- Java con Selenium: el componente desplegado en este servidor es una aplicación Java que utiliza la librería de Selenium para realizar tests funcionales. Esta aplicación requiere la instalación en el servidor del WebDriver compatible con un navegador existente en el servidor (en este caso, Google Chrome).

- WooCommerce: es una plataforma de desarrollo rápido de sitios web y tiendas virtuales. En esta parte están incluidas el resto de páginas que estén siendo testadas.

7.3. Diseño de la base de datos

El diseño lógico de la base de datos pretende construir el esquema de la información que utilizará el sistema, basándose para ello en el modelo conceptual descrito mediante el diagrama Entidad-Relación de la Figura 6.5. A su vez, el diseño lógico sirve de base para realizar el diseño físico y su implementación. Este diseño es importante cuando se alcanza la fase de mantenimiento del producto, pues permite que los cambios que se consideren necesarios se representen correctamente.

A continuación, se describe el modelo relacional diseñado para nuestra base de datos:

CLIENTE (id, nombreCliente, activo)

WEB (idWeb, idCliente, url, activo, PruebaLogin, PruebaEP, PruebaFC, PruebaHR)

- FK: idCliente referencia a CLIENTE (id)

TEST (idPrueba, idWeb, url, activo, htUser, htPass, tipo)

- FK: IdWeb referencia a WEB (idWeb)

RESPUESTA HTTP (idPrueba)

- FK: idPrueba referencia a TEST (idPrueba)

EXISTENCIA PROD (idPrueba, metodoLocalizadorProducto, localizadorProducto)

- FK: idPrueba referencia a TEST (idPrueba)

LOGIN (idPrueba, correo, pass, metodoLocalizadorCorreo, localizadorCorreo, metodoLocalizadorPass,localizadorPass, metodoLocalizadorBoton, localizadorBoton, metodoLocalizadorComprobacion,localizadorComprobacion, textoComprobacion)

- FK: idPrueba referencia a TEST (idPrueba)

FORM CONTACTO (idPrueba, valorNombre, valorCorreo, valorTelefono, valorComentario,valorEmpresa, valorApellido, valorPais, metodoLocalizadorNombre, localizadorNombre, metodoLocalizadorApellido, localizadorApellido, metodoLocalizadorEmpresa, localizadorEmpresa, metodoLocalizadorCorreo, localizadorCorreo, metodoLocalizadorPais, localizadorPais, metodoLocalizadorDepartamento, localizadorDepartamento, metodoLocalizadorTelefono, localizadorTelefono, metodoLocalizadorComentario, localizadorComentario, metodoLocalizadorAceptoPolitica, localizadorAceptoPolitica, metodoLocalizadorBoton, localizadorBoton)

- FK: idPrueba referencia a TEST (idPrueba)

LOG TEST (idPrueba, timestamp, resultado, mensaje, nombrePrueba)

- FK: id referencia a TEST (idPrueba)

7.4. Diseño de la interfaz

La interfaz de la aplicación Catón constará de varias pantallas con un menú de navegación común a todas ellas. La estructura de la aplicación viene dada por la funcionalidad que debe proporcionar, que no es otra que permitir la gestión de los principales aspectos relacionados con la automatización de pruebas con el componente de pruebas en Java. Catón interactuará con la base de datos, introduciendo o modificando los datos que después leerá dicho componente para llevar a cabo su actividad.

Es de destacar que se trata de un sistema de uso interno de la organización, por lo que no es necesario un diseño atractivo, sino funcional. Tampoco es importante en este caso la accesibilidad porque se va a hacer uso de ella en dispositivos de escritorio, por lo que el hecho de que se adapte a dispositivos móviles es deseable pero no imperativo.

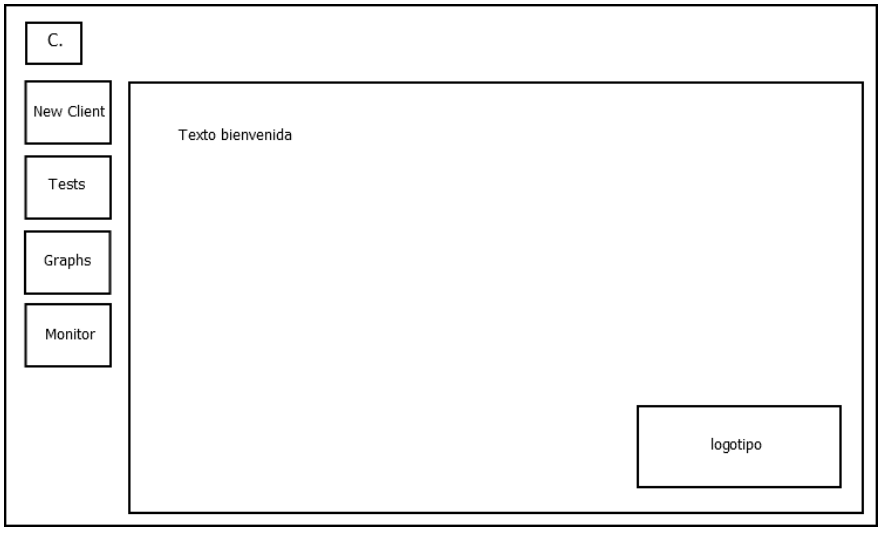


Figura 7.3: Boceto de la pantalla *Home*.

Al inicio de la aplicación nos encontraremos con la pantalla inicial, que denominaremos *Home* (Figura 7.3). Desde esta pantalla, y mediante el menú de navegación de la izquierda, se puede acceder a los cuatro grandes bloques de funcionalidad, que se describen a continuación.

7.4.1. Gestión de clientes y webs

Al acceder a esta parte se mostrará la pantalla descrita en la Figura 7.4, que está compuesta de dos secciones: en la primera, se mostrarán los clientes existentes en el sistema, y en la segunda se dará la posibilidad de agregar nuevos clientes.

Los clientes existentes se mostrarán en forma de listado, indicando su nombre y si están activos o no en el sistema. Además, se agregará un botón asociado al cliente que permita mo-

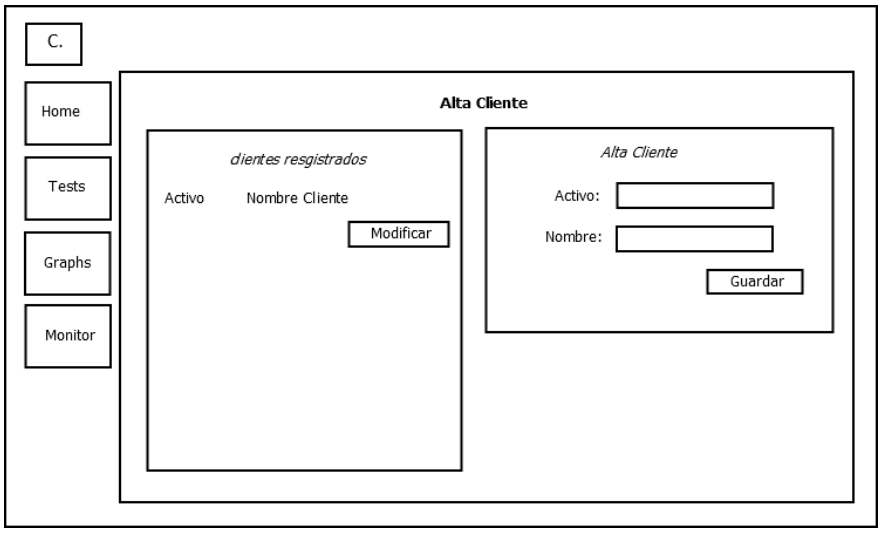


Figura 7.4: Boceto de la pantalla Alta Cliente

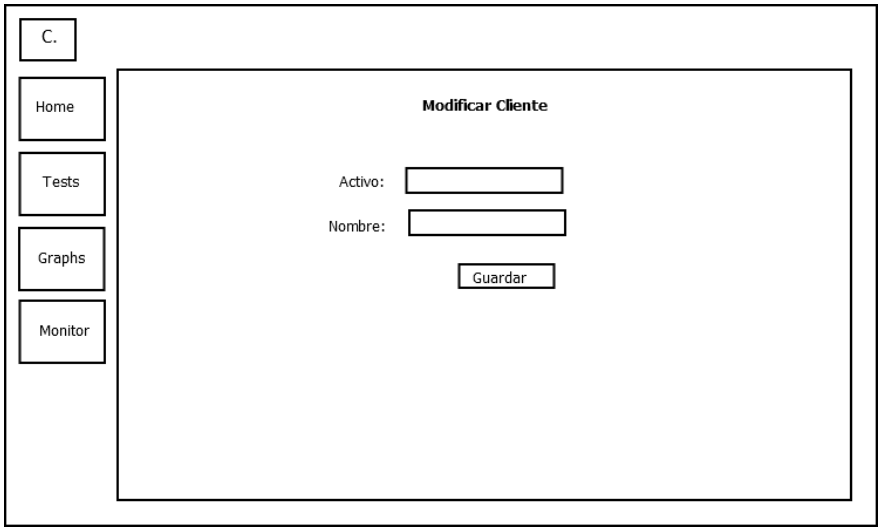


Figura 7.5: Boceto de la pantalla Modificar Cliente

dificar sus datos, de forma que pulsándolo se lleve al usuario a otra pantalla donde introducir los nuevos valores (ver 7.5).

Para el alta de cliente, se requiere la entrada de su nombre y su condición de activo. Al pulsar el botón de "Guardar", la aplicación solicitará asociarle una web, tal y como se muestra en la Figura 7.6. Además en esa página también podremos ver un listado de las webs registradas en el sistema.

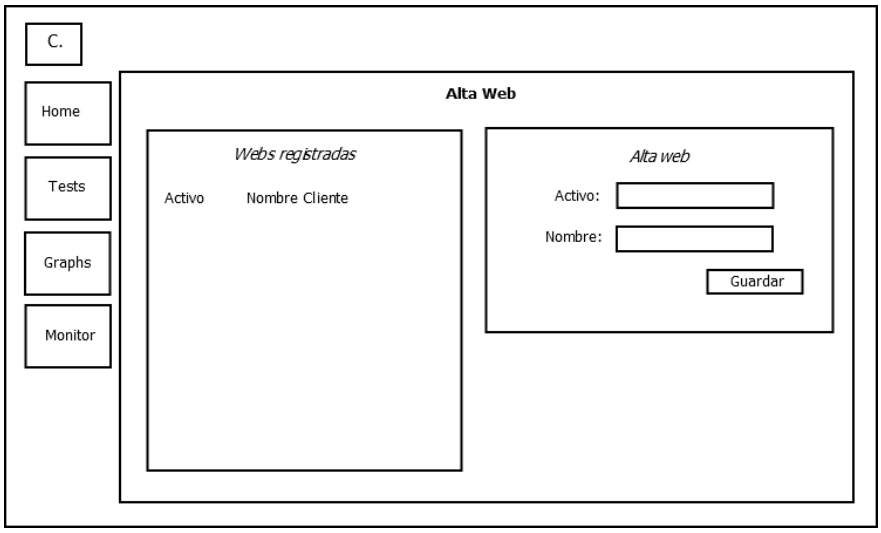


Figura 7.6: Boceto de la pantalla Alta Web

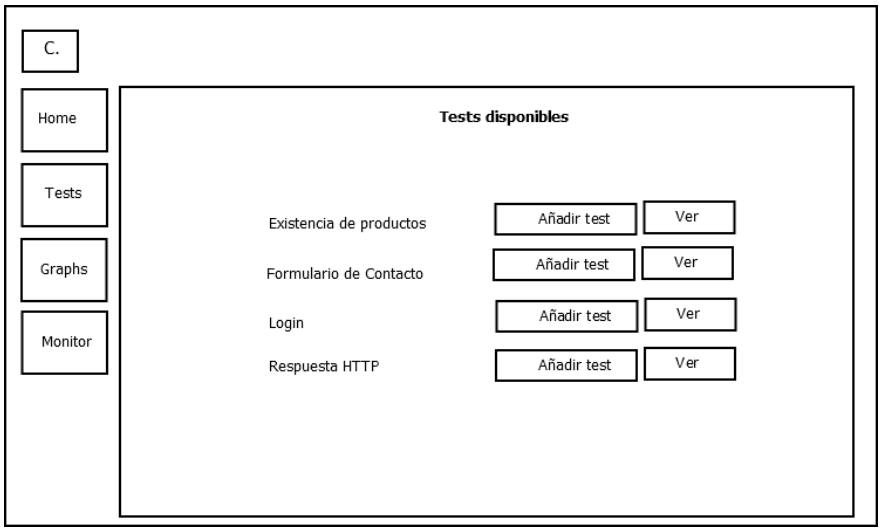


Figura 7.7: Boceto de la pantalla Tests disponibles

7.4.2. Tests

En la sección de Tests, cuya estructura se muestra en la Figura 7.7, podremos gestionar los tests registrados en el sistema y dar de alta nuevas pruebas. Para cada tipo de prueba, se muestran los botones correspondientes que dan acceso a estas funciones ("Ver" y "Añadir test", respectivamente).

El alta de una prueba solicitará los datos necesarios para configurar una prueba del tipo indicado, mediante un formulario de entrada específico. En la Figura 7.8 puede verse un ejemplo de formulario básico.

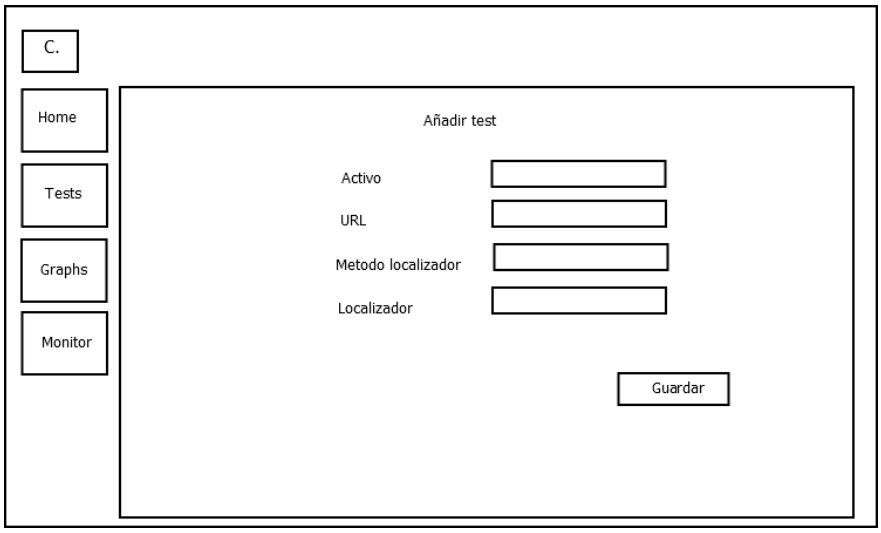


Figura 7.8: Boceto de la pantalla Añadir Test

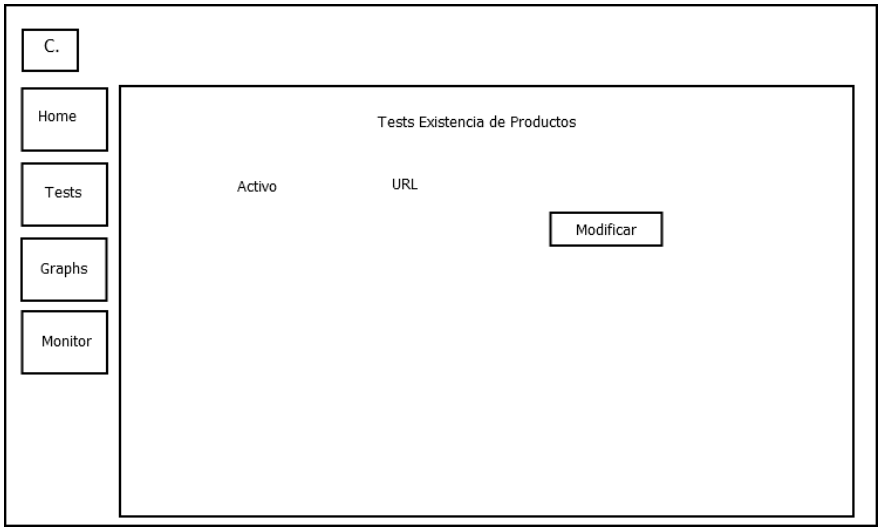


Figura 7.9: Boceto de la pantalla Listado Test

Si se pulsara el botón "Ver" nos llevaría a la pantalla de la Figura 7.9, donde podremos modificar una de las pruebas registradas en la base de datos pulsando el botón "Modificar" asociado. Esto lleva al usuario a un formulario similar al de alta de prueba, donde podrá modificar los datos registrados de un test existente (ver Figura 7.10).

7.4.3. Gráficos

En la sección de Gráficos se puede ver la información agregada de cada tipo de test, filtrando los resultados tanto por sitio web como por fechas. De esta manera, es posible ver

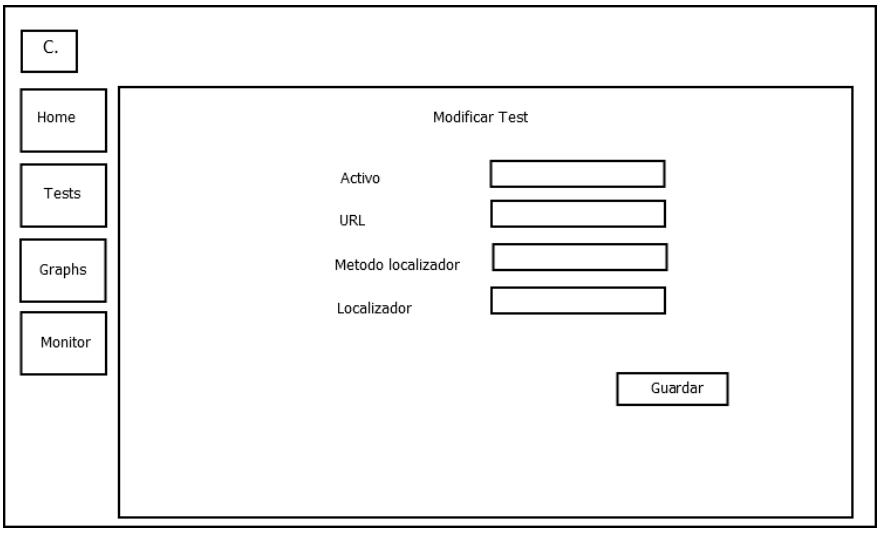


Figura 7.10: Boceto de la pantalla Modificar Test

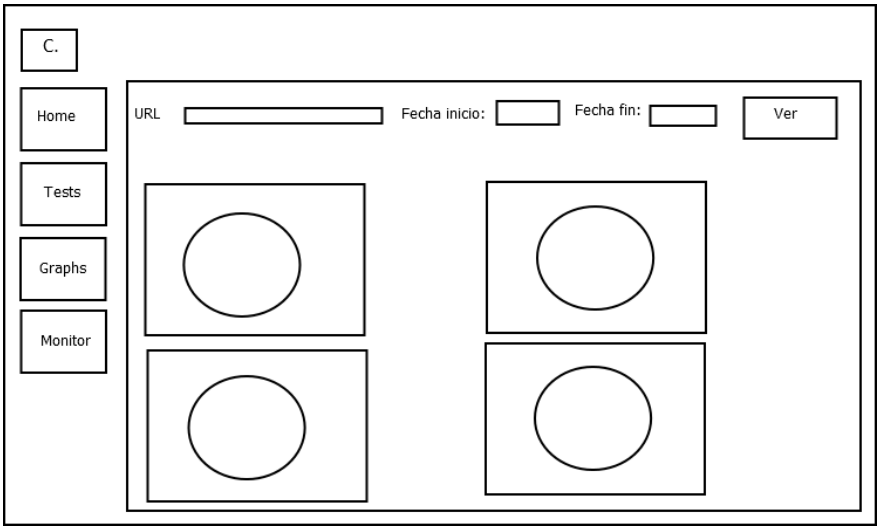


Figura 7.11: Boceto de la pantalla Gráficos

un resumen de las pruebas superadas y fallidas de una web en el período de nuestro interés.

La interfaz contará con elementos de entrada adecuados para recoger estos datos, como se muestra en la Figura 7.11: un selector desplegable que mostrará todos los sitios web registrados, y dos selectores de fecha para las fechas de inicio y fin. Una vez configurada la consulta, el botón "Ver" iniciará el proceso de actualización de los datos visualizados.

Esta petición se lleva a cabo mediante AJAX, por lo que la página no se recargará al completo, dando una sensación de mayor fluidez al usuario. Los datos se mostrarán en forma de gráficos de tarta para una rápida visualización, aunque el usuario podrá obtener un nivel de detalle mayor posando el cursor sobre las distintas áreas definidas en el gráfico.

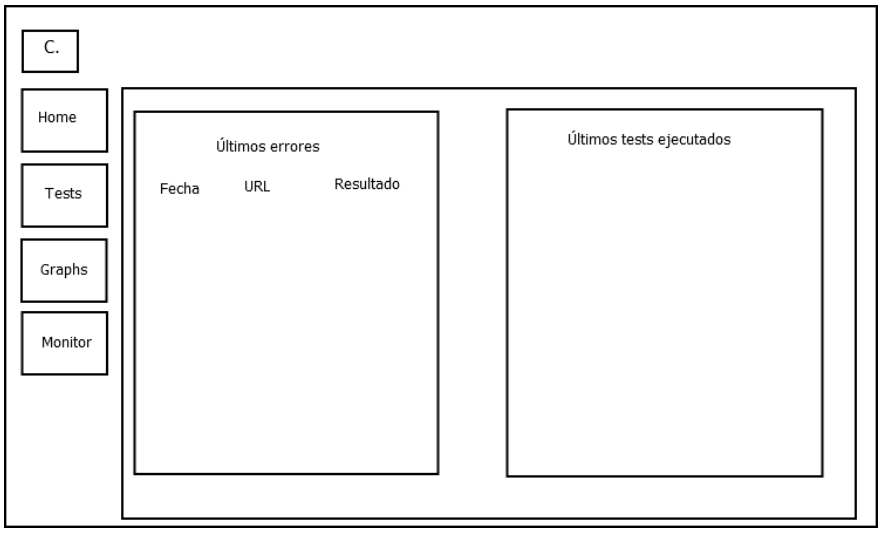


Figura 7.12: Boceto de la pantalla Monitor

7.4.4. Monitor

Mientras que la sección de Gráficos tiene como objetivo proporcionar una visión general y resumida de los resultados, la sección Monitor permite obtener información más detallada de las últimas pruebas ejecutadas. Esta vista debe estar siempre actualizada para facilitar la detección de errores, puntuales o sostenidos, que puedan producirse en las pruebas en curso. Por ello, la aplicación refresca automáticamente los datos mostrados cada 10 segundos, permitiendo un seguimiento ágil de cualquier cambio en los datos de la base de datos.

Esta pantalla se divide en dos áreas con diferente propósito (ver Figura 7.12). A la derecha se sitúa un espacio donde podrán verse las últimas 20 pruebas de cada tipo llevadas a cabo. Para no sobrecargar la interfaz, se ha utilizado un sistema de pestañas (*tabs*) para visualizar las pruebas de uno u otro tipo.

En la parte izquierda tendremos una lista, análoga a cualquiera de las situadas a la derecha, donde se mostrarán los últimos 20 resultados negativos, independientemente de su tipo. Esto nos permitirá descubrir más fácilmente errores persistentes en un sitio web, sin que estos queden rápidamente ocultos por un gran número de resultados positivos del resto de sitios web.

Capítulo 8

Implementación

8.1. Tecnologías utilizadas

Para la implementación de los diferentes módulos se han utilizado las siguientes tecnologías:

- **UBUNTU:** es un sistema operativo basado en Linux que nace en abril de 2004 y está mayoritariamente compuesta de software libre proveniente de una rama de Debian, de la cual parte, pero contiene también software propietario y es distribuido libremente con licencia GNU GPL. Está orientado al uso en ordenadores desktop, pero también presenta variantes para servidor, tablets y otros dispositivos.

  Ubuntu servirá en este caso para instalarlo en el servidor donde se ejecutarán los tests.

- **Apache Server:** Es el un servidor HTTP más utilizado ya que es capaz de operar bajo una gran variedad de sistemas operativos, entre los cuales se encuentran Linux o Windows. Es un software que realiza las funciones de transporte de la información, de conexión y tiene la ventaja de ofrecer funciones de control de la seguridad como las efectuadas por un proxy.

- **HTML:** Es un lenguaje de marcado formado por un conjunto de etiquetas cuyo propósito es dar formato a una página web. Está respaldado por el W3C.

  En este Trabajo se utilizará para formar la estructura de las páginas webs que conforman la aplicación Catón.

- **CSS:** Es un lenguaje usado para definir el formato de los documentos HTML, XHTML y XML. La introducción de CSS se hizo necesaria para separar los contenidos del formato y permitir así una programación más clara y fácil de utilizar.

- **JavaScript:** Es un lenguaje de programación orientado a objetos y a eventos, comúnmente utilizado en la programación web del lado del cliente.

En el proyecto se utilizará para la realización de los gráficos de ratio de errores mediante la técnica AJAX de la librería JQuery.

- **JSON:** es el acrónimo de *JavaScript Object Notation* y es un formato de texto adaptado al intercambio de datos entre las capas de las aplicaciones cliente / servidor.

  JSON será el formato que se utilizará para la comunicación de datos entre capas en la aplicación Catón.

- **Java:** Es un lenguaje de programación de alto nivel, orientado a objetos y fuertemente tipado, que se apoya en la homónima plataforma de ejecución de software. Está proyectado para ser lo más posible independiente de la plataforma hardware de ejecución.

  En Java se desarrollará la aplicación de ejecución de tests.

- **Selenium:** Selenium es un entorno para la realización de test en aplicaciones web. Se compone de varios componentes, aunque en este trabajo se utilizará Selenium WebDriver, el cual instancia un navegador y lo controla por medio de comandos. Esto se lleva a cabo mediante un controlador específico para cada *browser* que envía las instrucciones y obtine los resultados.

  Por tanto, se utilizará Selenium WebDriver con el controlador de Chrome para lanzar los tests parametrizados.

- **Bootstrap:** es una librería para la creación de sitios y aplicaciones Web. Contiene diversos componentes para el desarrollo de la interfaz, como módulos útiles para la navegación o para extender las opciones de JavaScript. Actualmente es compatible con las últimas versiones de todos los navegadores principales y desde la versión 2.0 soporta el diseño *responsive*.

- **ChartJS:** es una librería *open-source* de JavaScript utilizada para la visualización de datos. Actualmente soporta ocho tipos diferentes de gráficos: barras, líneas, área, tarta, burbuja, radar, polar y *scatter*. ChartJS renderiza el elemento canvas de HTML5 para mostrar los elementos en pantalla.

8.2. Implementación de las pruebas automatizadas

La aplicación desarrollada en Java y que hace uso de la librería Selenium WebDriver es la encargada real de la ejecución de los tests. Para ello, inicia la ejecución en la clase *Main*. Esta clase realiza dos actividades:

1. Instanciar un hilo de ejecución por cada tipo de test, con una periodicidad determinada (entre 3 y 5 minutos, dependiendo de la prueba).
2. Verificar que ninguno de los hilos anteriores ha lanzado alguna excepción y se ha detenido. En caso de que sí se haya producido algún error, envía una alerta en forma de correo electrónico. De esa forma se podría reiniciar el sistema y continuar con la ejecución de las pruebas normalmente.

Cada tipo de prueba la ejecuta una clase Java diferente. Al instanciarse, cada una de estas clases realizan una consulta a la base de datos para obtener los datos con los que construir las pruebas (ver Figura 8.1). Además, comprueban que la URL sobre la que va a lanzar el test está disponible (utilizando para ello la prueba de *Respuesta HTTP*), y, si no lo está, envía el aviso correspondiente.

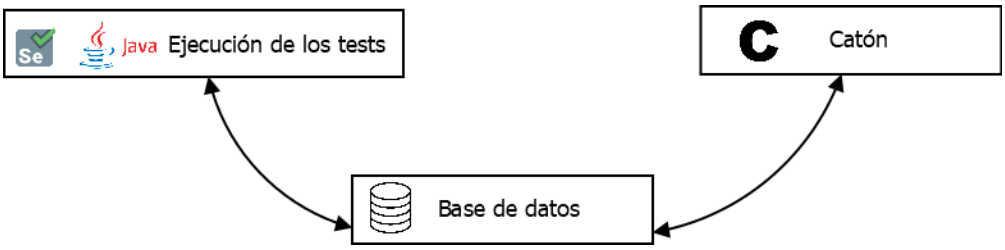


Figura 8.1: Relación componentes

Posteriormente, comienza la lógica de cada prueba, las cuales se describen a continuación:

- El test de Existencia de Productos (Fichero: *ExistenciaProductosParametros.java*) consiste en acceder a la URL de la página de categoría de productos y buscar los productos a través de los identificadores de producto configurados (por ejemplo, pueden estar dispuestos en celdas con un identificador determinado, dentro de una tabla). Si encuentra al menos un producto, el algoritmo del test escribe en la base de datos el resultado; si no encuentra ningún elemento, envía un correo con el nombre de la prueba y la URL exacta donde se ha producido la incidencia.
- El test de Respuesta HTTP (Fichero: *HttpResponseSe.java*) accede a la URL proporcionada y toma el texto del título de la página. Si en esa cadena de caracteres encuentra el literal "Error", significa que el servidor ha lanzado un código diferente a 200, y el algoritmo enviaría una alerta y escribiría en la base de datos este resultado erróneo.
- El test Formulario de Contacto (Fichero: *FormularioContacto.java*) es el más complejo de todos, debido a la cantidad de tipos de datos que se pueden pedir. En primer lugar, accede a la dirección del formulario de contacto, para después rellenar los campos que le correspondan. Finalmente comprueba que el botón de enviar sea clicable, aunque no se ejecuta la acción de pulsación, puesto que eso provocaría *spam* en la bandeja de entrada del correo electrónico donde llegaran esas peticiones. Si el resultado del test es correcto escribe en la base de datos, y si no lo es, además envía una alerta.
- El test Login (Fichero: *Login2.java*) consiste en acceder a la página de login de la aplicación web (que registramos al configurar el test a través de Catón), se introducen los datos de identificación en los campos de entrada correspondientes, y se pulsa el botón "Iniciar sesión". Esto provocará un redireccionamiento a una página de confirmación si el login ha sido exitoso, por lo que se buscará en la nueva página algún elemento indicativo de que esta operación se ha completado correctamente (por ejemplo, un

mensaje de bienvenida). Si el resultado es correcto escribe en la base de datos, si no lo es, además lanzará una alerta.

En todos los casos, cuando se accede a una URL, el sistema tieme implementado el mecanismo de esperas fluidas de Selenium. Esto es útil cuando un sitio web tiene latencias en el proceso de servir una página y mostrarla porque podría llevar a provocar falsos negativos en las pruebas. Con estas esperas se limita la posibilidad de que eso ocurra.

Finalmente, como toda aplicación puede fallar. Por ello cada prueba está implementada en un hilo, y cada hilo está sujeto a un sistema de alertas independiente. Si el algoritmo de una prueba falla, se notificará tal hecho para reiniciar el sistema.

8.3. Implementación de la aplicación de gestión.

Para finalizar, y para ejemplificar el resultado final de la aplicación Catón, se muestran las Figuras 8.2 y 8.3 . En el manual de usuario de la aplicación (ver Anexo A.1) se podrán encontrar capturas de todas las interfaces.

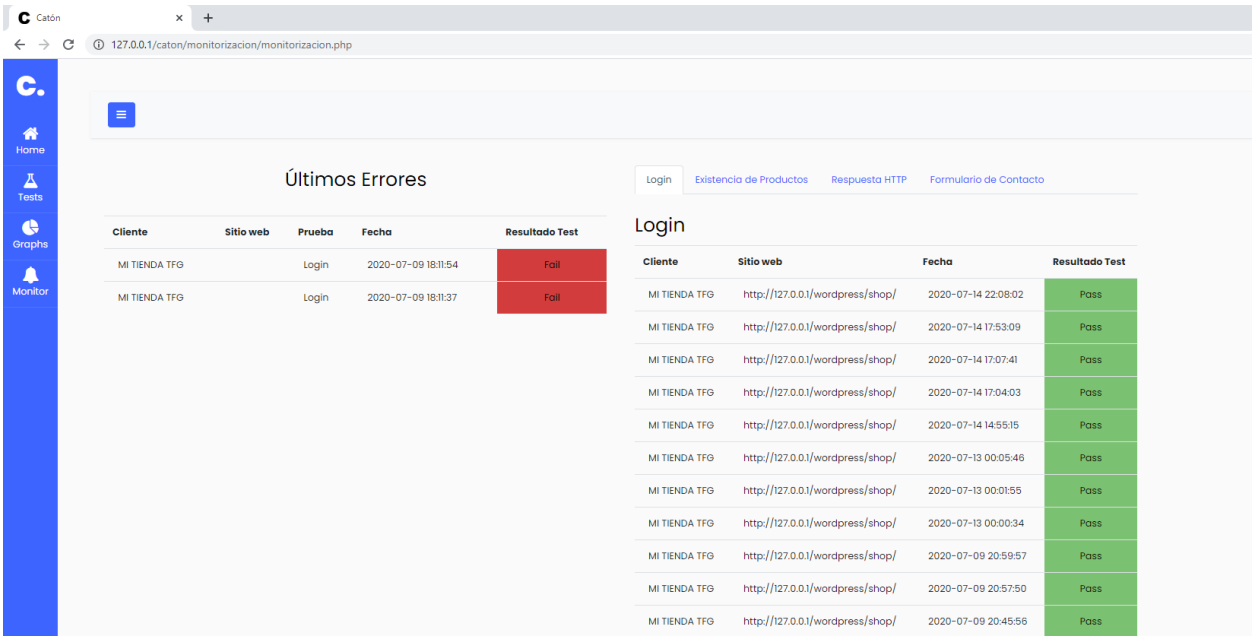


Figura 8.2: Monitorización. Catón

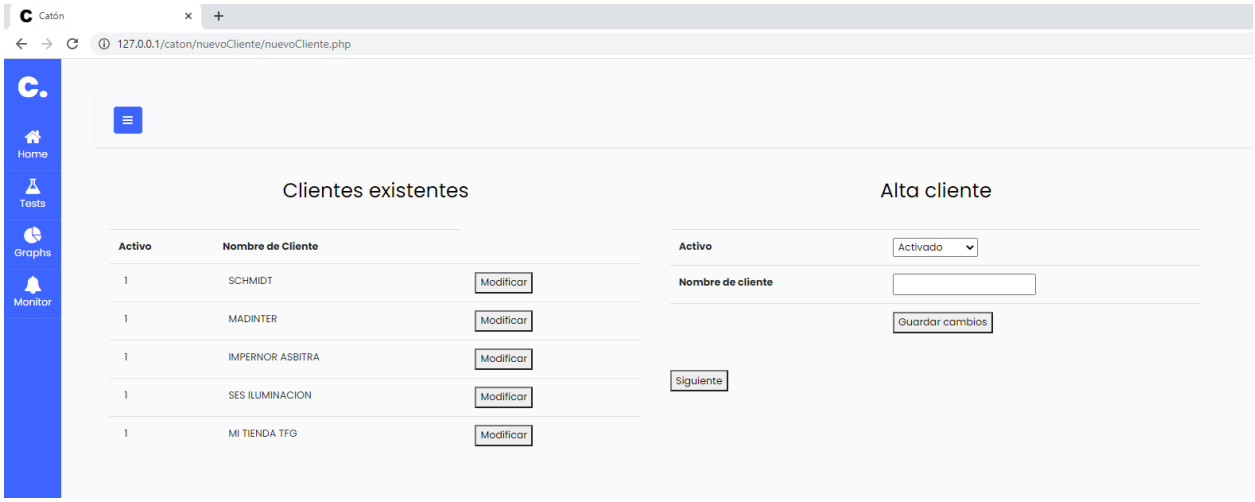


Figura 8.3: Gestión clientes. Catón

Capítulo 9

Análisis de los test ejecutados

En este capítulo se revisarán los resultados obtenidos en las ejecuciones de prueba de nuestro componente de automatización. Este componente se ejecuta de manera continua, lanzando periódicamente las diferentes pruebas registradas para cada uno de los sitios web que se hayan indicado. Al terminar la prueba, tanto si esta ha fallado en algún punto de la ejecución como si se ha completado correctamente, este componente registra en la base de datos el resultado obtenido. En caso de fallo, se registra la traza de la ejecución para facilitar la identificación del error.

9.1. Pruebas funcionales

A continuación se analizarán los resultados de los tests funcionales lanzados contra el sitio web de comercio electrónico construido con WooCommerce en el proyecto. Se han ejecutado los cuatro tipos de prueba varias veces entre los días 6/7/2020 y 13/7/2020 para tener una muestra suficiente con la que realizar el análisis.

9.1.1. Test: Login

La prueba de Login consiste en verificar que un usuario sin autenticar puede hacer login en la tienda virtual. Para ello, es necesario utilizar credenciales válidas de un usuario registrado en la plataforma (en otro caso, nos encontraríamos en un escenario diferente de los descritos en la Capítulo 5).

Como se puede ver en la Figura 9.1, se han realizado varios tests de este tipo (posando el cursor sobre el gráfico podríamos ver que en total son 11). Ha fallado 2 veces, que se corresponden con errores a la hora de configurar el test en la aplicación Catón. El resto del tiempo, el sistema de automatización consigue iniciar sesión sin problemas en la tienda virtual.

(a) Tabla de últimos resultados.

(b) Gráfico de resultados.

Figura 9.1: Resultados del test de Login.

Si se detectara algún error es fundamental ver si es puntual o sostenido, pues esto determina las decisiones o acciones a tomar. Si es un error puntual, se puede deber a una sobrecarga en el servidor, lo que le habría impedido atender nuestra petición, o a que la página no se ha cargado correctamente en esa ejecución del test. Como consecuencia de ello, Selenium no encontraría los elementos HTML que necesita y el resultado del test sería negativo.

Si el problema es sostenido, sería necesario verificar que la prueba está bien configurada, para luego pasar a descartar posibles problemas de conexión a la red del servidor de ejecución, o algún problema en el propio servidor que impida ejecutar las pruebas. En función de dónde esté el problema, habría que avisar al responsable designado para su pronta resolución.

9.1.2. Test: Existencia de productos

Con este test se trata de verificar si el sistema tiene productos en sus páginas de categoría de productos. Dado que una tienda virtual suele tener múltiples categorías de productos, se registra una prueba a través de Catón para cada una de ellas, indicando su URL de acceso.

Este test es especialmente útil como sonda de disponibilidad, ya que a veces las cargas que se hacen en las bases de datos no se indexan bien, y como resultado no se muestran productos en la web del comercio electrónico. En este caso se ha hecho la prueba en una página en la que sí debería haber productos (Figura 9.2).

Según se puede apreciar en la Figura 9.3, en todas las ejecuciones el sistema ha encontrado productos donde se suponía que los debía encontrar. El test ha pasado satisfactoriamente.

En el caso de que se detectara un error, es primordial actuar con rapidez pues la tienda web no tiene ningún valor si no hay productos en ella. La periodicidad de esta prueba en un entorno real sería de varias horas o incluso una vez al día. Los problemas que pueden ocasionar un fallo se pueden localizar en la lógica de la aplicación de acceso a datos, o más

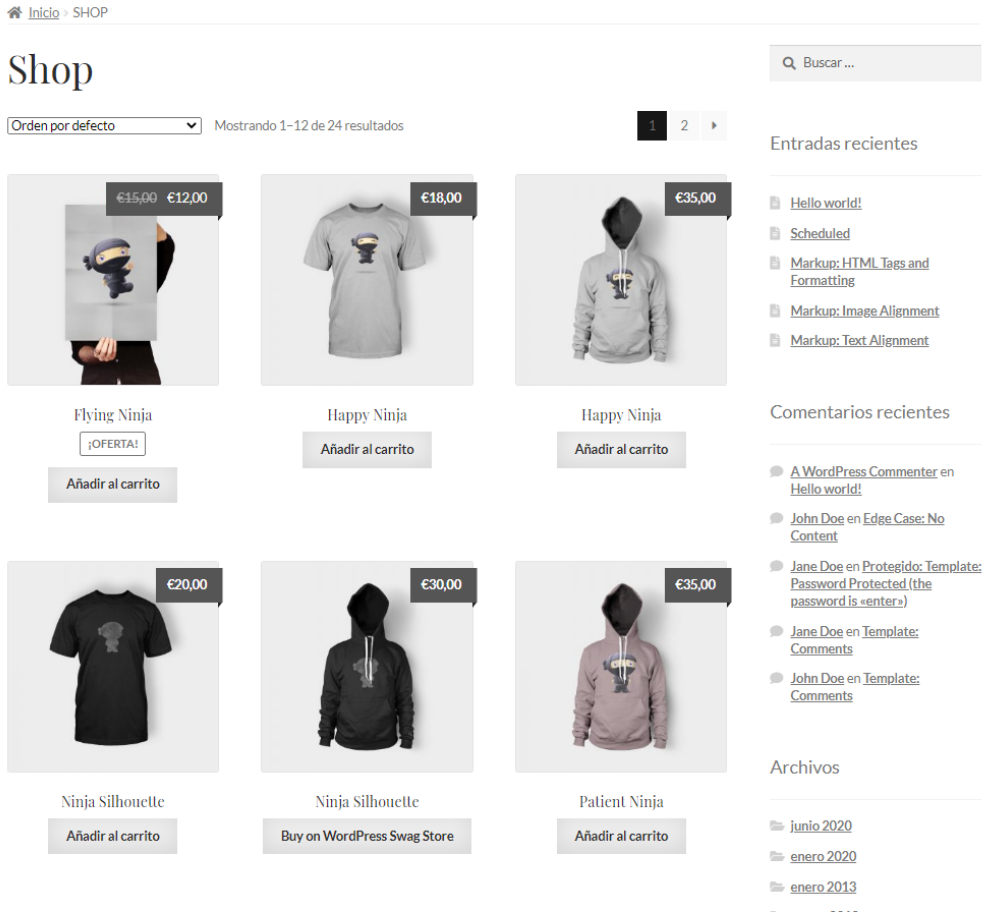
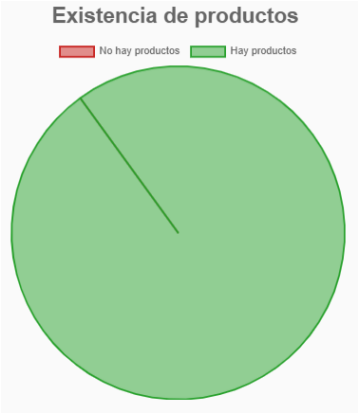


Figura 9.2: Página de categoría de productos WooCommerce

(a) Tabla de últimos resultados.



(b) Gráfico de resultados.

Figura 9.3: Resultados del test de Existencia de Productos.

comúnmente derivados de la sincronización con el ERP (*Enterprise Resource Planning*)[1].

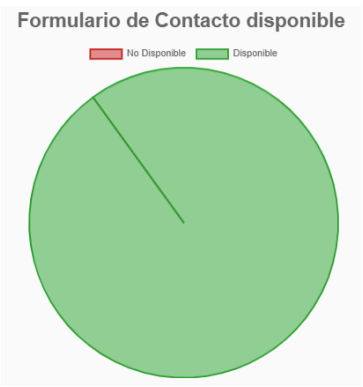
Por ello, si se produjera un error habría que verificarlo y avisar al responsable designado para su resolución inmediata.

9.1.3. Test: Formulario de contacto

Con el test se quería comprobar la funcionalidad de enviar el formulario de contacto si se había introducido todos los datos requeridos. Para ello, se rellenan todos los campos del formulario de contacto con datos de prueba y se comprueba que el botón de "Enviar formulario" se pueda pulsar (no se hace click porque eso provocaría spam en la bandeja de entrada de correo electrónico donde llegan las peticiones).

Tal y como se muestra en la Figura 9.4, el formulario estaba en condiciones de ser enviado todas las veces que se ha realizado la prueba. Podemos concluir que el test ha pasado con éxito.

(a) Tabla de últimos resultados.



(b) Gráfico de resultados.

Figura 9.4: Resultados del test de Existencia de Productos.

En el caso de que produjera un error, también habría que determinar si es un error puntual o sostenido.

Se puede producir un fallo en esta prueba además de por las derivadas de la disponibilidad de la página del formulario de contacto, porque no encuentre algún campo obligatorio de ser informado, o porque existan campos que no han sido previstos o el formulario cuente con mecanismos anti-automatización como los *captchas*.

[1]Es el software que integra y maneja la información relacionada con las actividades de producción, de negocio y de distribución de una empresa

9.1.4. Test: Respuesta HTTP

El objetivo de este test es verificar que el sistema estaba disponible para ser servido sin problemas cada vez que un usuario lo reclamara. Para ello se verifica el código HTTP que envía el servidor cuando se hace una petición. Si responde con un código 200 el test se considera que ha sido superado, en caso contrario se considera que ha fallado (el motivo será especificado en el correo de alerta correspondiente).

Según se puede apreciar en la Figura 9.5, el test ha sido exitoso en todos los casos en que se ha lanzado.

(a) Tabla de últimos resultados. (b) Gráfico de resultados.

Figura 9.5: Resultados del test de Existencia de Productos.

En un entorno real, esta prueba debería ser la que se ejecute más frecuentemente pues si la página no está disponible no se podrá satisfacer su objetivo de canal de comunicación o compra.

En caso de un error, en el mensaje de alerta de fallo enviado automáticamente se especifica el código HTTP que ha devuelto el servidor. Dependiendo del código se determinará a qué responsable avisar.

En cualquier caso, la resolución de esta incidencia debería ser prioritaria.

9.2. Pruebas no funcionales

Para la configuración y ejecución de pruebas no funcionales se ha utilizado JMeter. La configuración necesaria se ha descrito en el manual de usuario incluido en el Anexo B.2).

En todos los tests se simulará la carga de trabajo en el servidor con peticiones HTTP a nuestra tienda virtual, desplegada en un servidor local. El acceso a esta tienda virtual se hará mediante la dirección `http://127.0.0.1/wordpress/shop/`. En consecuencia, todas las pruebas estarán orientadas hacia esta URL, como se muestra en la Figura 9.6.

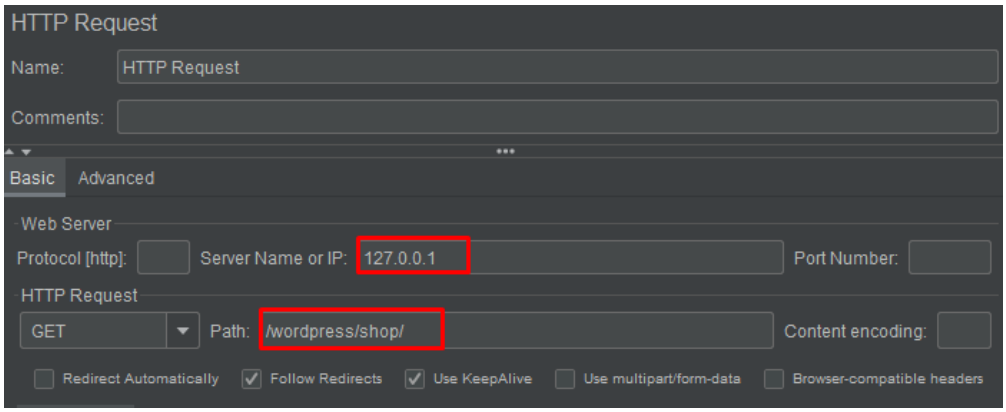


Figura 9.6: Configuración JMeter Peticiones HTTP.

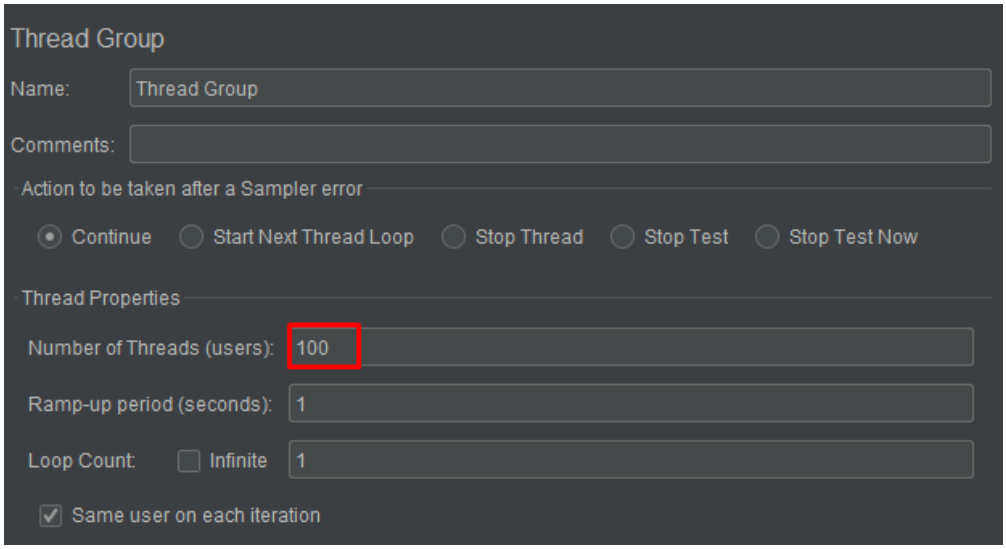


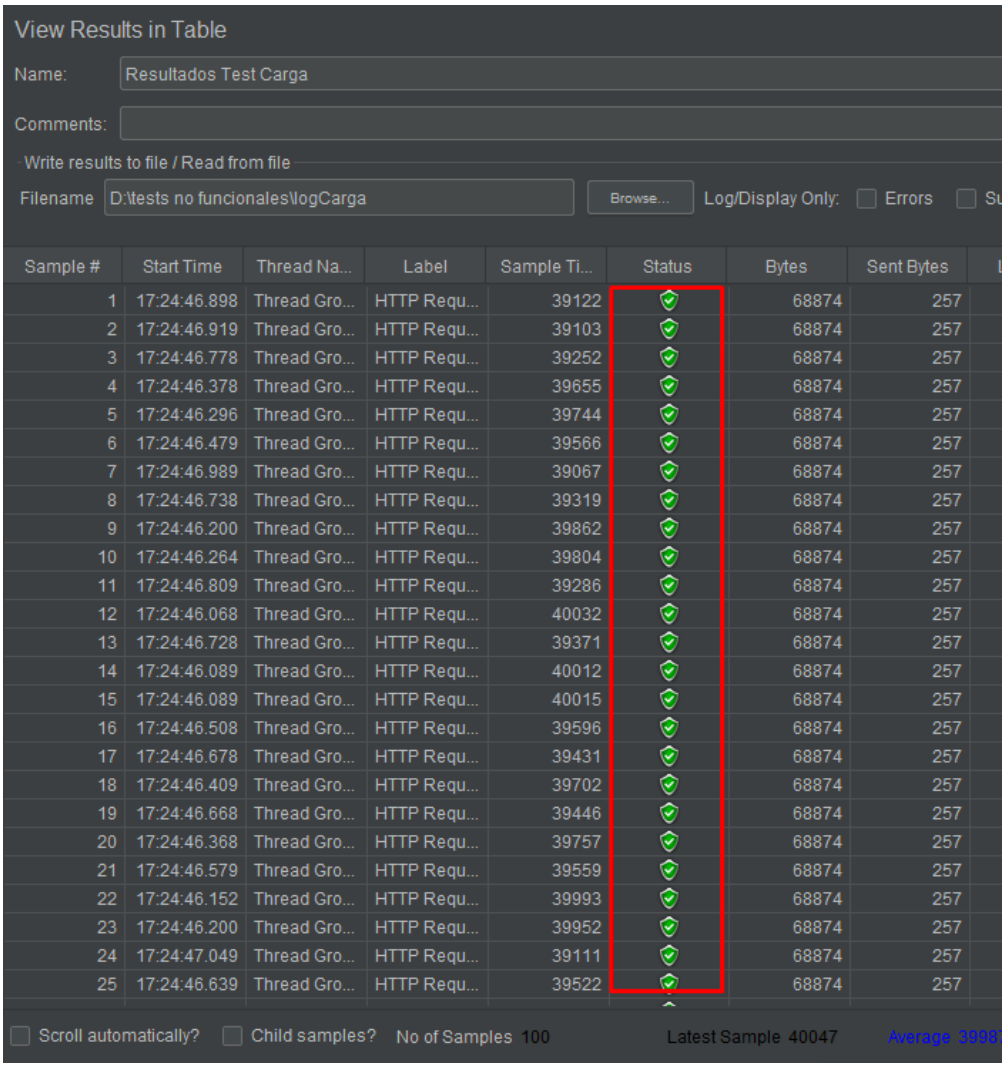
Figura 9.7: Configuración Test Carga

9.2.1. Test: Carga

El objetivo de este test es verificar si el sistema es capaz de soportar 100 usuarios concurrentes realizando peticiones HTTP, de manera que se ponga a prueba la estabilidad y capacidad de carga del servidor con un valor representativo de la carga que podría tener que alcanzar en su funcionamiento habitual.

9.2.1.1. Configuración

Para ello, se configura la prueba para crear este número de peticiones de acceso simultáneas a la tienda virtual mediante el parámetro *Number of Threads (users)*, como se muestra en la Figura 9.7.



Sample #	Start Time	Thread Na...	Label	Sample Ti...	Status	Bytes	Sent Bytes
1	17:24:46.898	Thread Gro...	HTTP Requ...	39122		68874	257
2	17:24:46.919	Thread Gro...	HTTP Requ...	39103		68874	257
3	17:24:46.778	Thread Gro...	HTTP Requ...	39252		68874	257
4	17:24:46.378	Thread Gro...	HTTP Requ...	39655		68874	257
5	17:24:46.296	Thread Gro...	HTTP Requ...	39744		68874	257
6	17:24:46.479	Thread Gro...	HTTP Requ...	39566		68874	257
7	17:24:46.989	Thread Gro...	HTTP Requ...	39067		68874	257
8	17:24:46.738	Thread Gro...	HTTP Requ...	39319		68874	257
9	17:24:46.200	Thread Gro...	HTTP Requ...	39862		68874	257
10	17:24:46.264	Thread Gro...	HTTP Requ...	39804		68874	257
11	17:24:46.809	Thread Gro...	HTTP Requ...	39286		68874	257
12	17:24:46.068	Thread Gro...	HTTP Requ...	40032		68874	257
13	17:24:46.728	Thread Gro...	HTTP Requ...	39371		68874	257
14	17:24:46.089	Thread Gro...	HTTP Requ...	40012		68874	257
15	17:24:46.089	Thread Gro...	HTTP Requ...	40015		68874	257
16	17:24:46.508	Thread Gro...	HTTP Requ...	39596		68874	257
17	17:24:46.678	Thread Gro...	HTTP Requ...	39431		68874	257
18	17:24:46.409	Thread Gro...	HTTP Requ...	39702		68874	257
19	17:24:46.668	Thread Gro...	HTTP Requ...	39446		68874	257
20	17:24:46.368	Thread Gro...	HTTP Requ...	39757		68874	257
21	17:24:46.579	Thread Gro...	HTTP Requ...	39559		68874	257
22	17:24:46.152	Thread Gro...	HTTP Requ...	39993		68874	257
23	17:24:46.200	Thread Gro...	HTTP Requ...	39952		68874	257
24	17:24:47.049	Thread Gro...	HTTP Requ...	39111		68874	257
25	17:24:46.639	Thread Gro...	HTTP Requ...	39522		68874	257

Figura 9.8: Resultados Test Carga

9.2.1.2. Resultados

Según se puede apreciar en la Figura 9.8 el sitio web admite perfectamente 100 usuarios concurrentes: consigue atender todas las peticiones con éxito, sin que se aprecie un incremento en el tiempo de respuesta del servidor a la petición del usuario (columna *Sample Time*). Por tanto, la prueba se da por superada satisfactoriamente.

Si el servidor se hubiera sobrecargado, el tiempo de respuesta habría aumentado progresivamente, hasta el punto de ser incapaz de atender peticiones, que caducarían antes de ser atendidas.

9.2.2. Test: Estrés

En esta prueba se verificará que el servidor

El objetivo de este test es ver si el sistema es capaz de pasar de 100 usuarios concurrentes de base, a 200 de forma escalonada sin que se produzcan caídas del servicio debido al aumento de carga.

9.2.2.1. Configuración

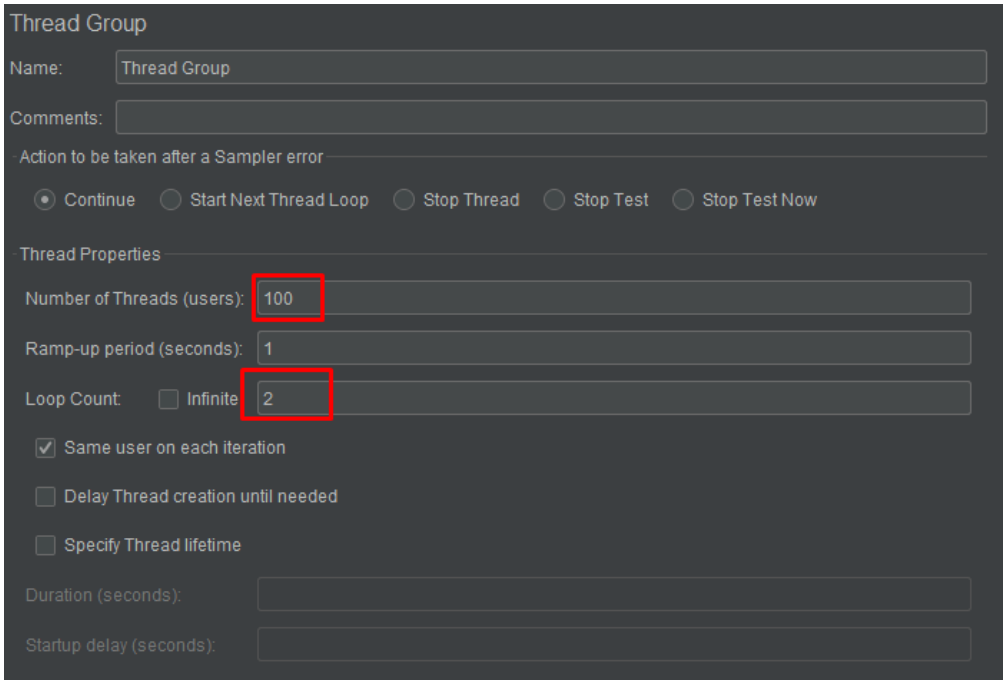


Figura 9.9: Configuración Test EStrés

Para la configuración de este test se han simulado 100 usuarios concurrentes a lo largo de 2 iteraciones. Los 200 usuarios realizarán peticiones HTTP al sitio *ecommerce*.

9.2.2.2. Resultados

El test ha sido superado porque el sitio web ha sido capaz de responder satisfactoriamente a todas las peticiones HTTP, cuando el número total de éstas va en aumento de forma escalonada.

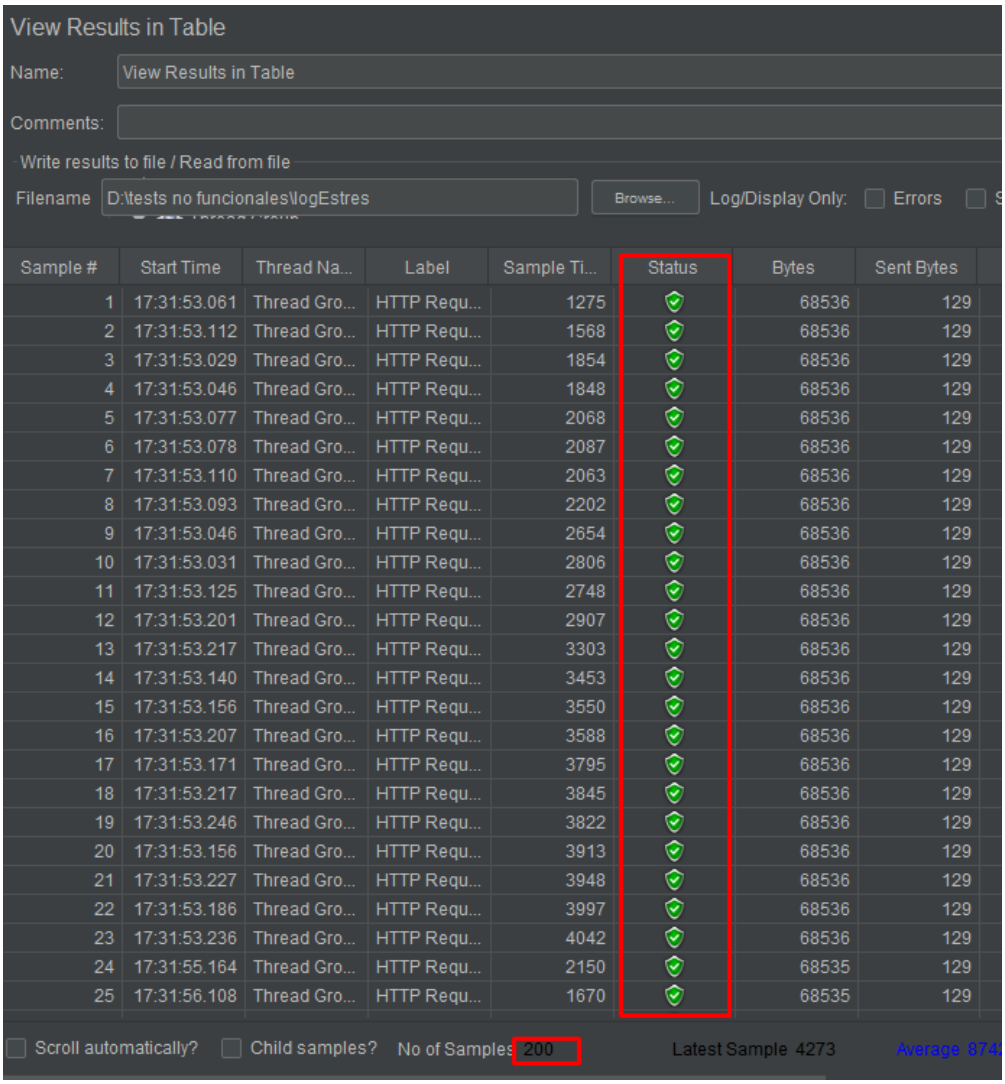


Figura 9.10: Resultados Test Estrés

9.2.3. Test: *Spike*

El objetivo de este test es tener al servidor con 500 de usuarios concurrentes de base y que durante 5 segundos esa carga suba a 1000 usuarios concurrentes. Si es capaz de responder a todas las peticiones incluso con esa cantidad de peticiones HTTP concurrentes tan alta, se considerará que ha pasado el test.

9.2.3.1. Configuración

Se ha configurado el test para que tenga 500 usuarios concurrentes y durante un espacio de 5 segundos suba la carga repentinamente a 1000.

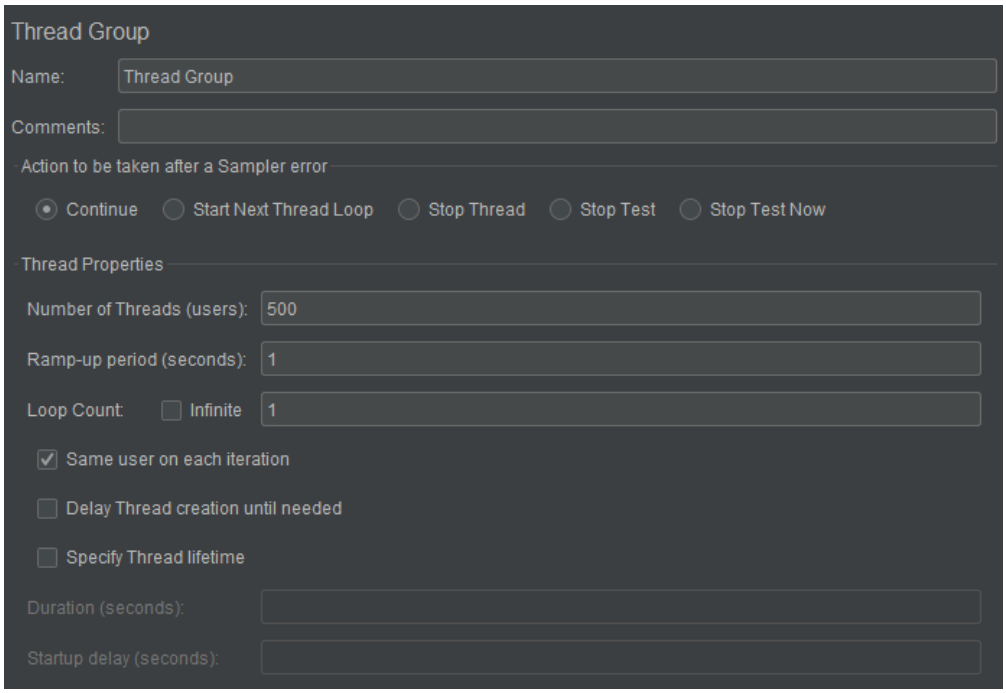


Figura 9.11: Configuración Test Spike (1)

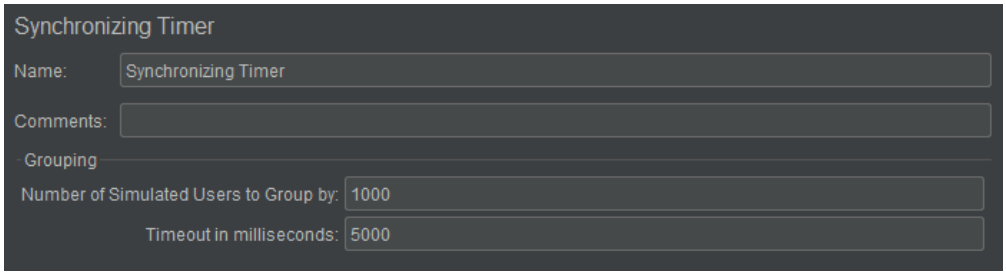


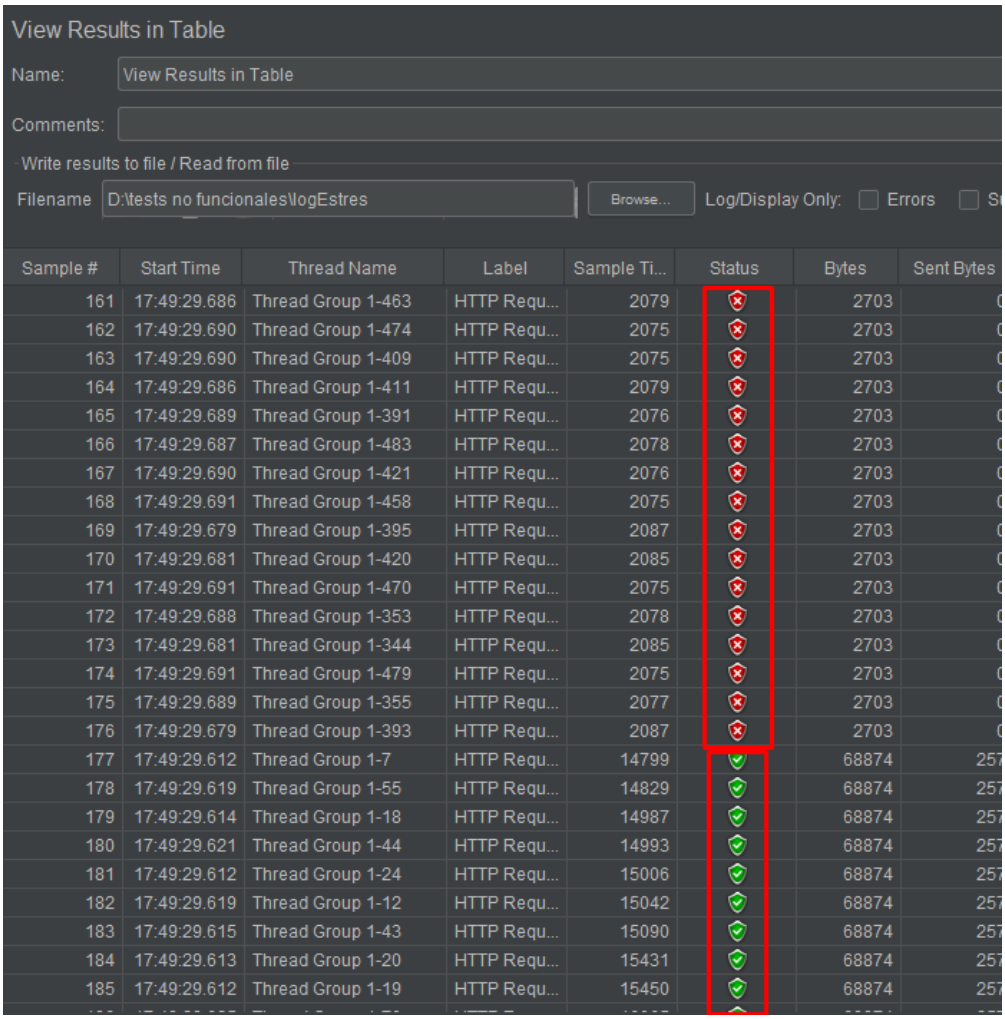
Figura 9.12: Configuración Test Spike (2)

9.2.3.2. Resultados

Como se puede ver en la Figura 9.13, el sistema no puede resolver tantas peticiones HTTP de forma repentina.

El test ha fallado porque no soporta tanta carga, por lo que debería ser comunicado, en caso de estar en un caso real, al equipo de sistemas para que aumenten la capacidad de respuesta del servidor. Los motivos del fallo se pueden deber a ancho de banda insuficiente, o recursos físicos del servidor insuficientes. Puesto que este test se ha lanzado contra un servidor local, lo más probable es que se trate del segundo motivo.

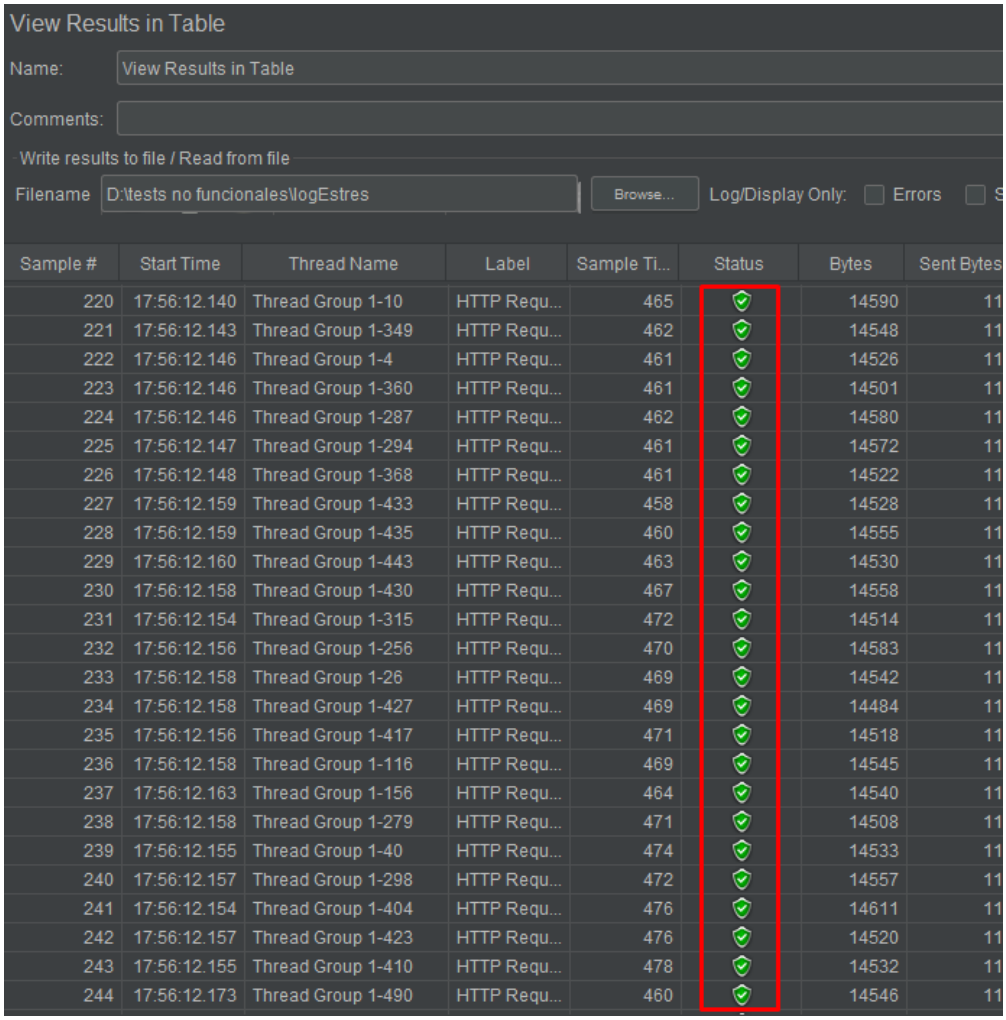
Como demostración de la reacción al pico de carga de mi servidor local, he realizado la misma prueba con la misma configuración pero contra Google España. Estos son los



Sample #	Start Time	Thread Name	Label	Sample Ti...	Status	Bytes	Sent Bytes
161	17:49:29.686	Thread Group 1-463	HTTP Requ...	2079		2703	
162	17:49:29.690	Thread Group 1-474	HTTP Requ...	2075		2703	
163	17:49:29.690	Thread Group 1-409	HTTP Requ...	2075		2703	
164	17:49:29.686	Thread Group 1-411	HTTP Requ...	2079		2703	
165	17:49:29.689	Thread Group 1-391	HTTP Requ...	2076		2703	
166	17:49:29.687	Thread Group 1-483	HTTP Requ...	2078		2703	
167	17:49:29.690	Thread Group 1-421	HTTP Requ...	2076		2703	
168	17:49:29.691	Thread Group 1-458	HTTP Requ...	2075		2703	
169	17:49:29.679	Thread Group 1-395	HTTP Requ...	2087		2703	
170	17:49:29.681	Thread Group 1-420	HTTP Requ...	2085		2703	
171	17:49:29.691	Thread Group 1-470	HTTP Requ...	2075		2703	
172	17:49:29.688	Thread Group 1-353	HTTP Requ...	2078		2703	
173	17:49:29.681	Thread Group 1-344	HTTP Requ...	2085		2703	
174	17:49:29.691	Thread Group 1-479	HTTP Requ...	2075		2703	
175	17:49:29.689	Thread Group 1-355	HTTP Requ...	2077		2703	
176	17:49:29.679	Thread Group 1-393	HTTP Requ...	2087		2703	
177	17:49:29.612	Thread Group 1-7	HTTP Requ...	14799		68874	257
178	17:49:29.619	Thread Group 1-55	HTTP Requ...	14829		68874	257
179	17:49:29.614	Thread Group 1-18	HTTP Requ...	14987		68874	257
180	17:49:29.621	Thread Group 1-44	HTTP Requ...	14993		68874	257
181	17:49:29.612	Thread Group 1-24	HTTP Requ...	15006		68874	257
182	17:49:29.619	Thread Group 1-12	HTTP Requ...	15042		68874	257
183	17:49:29.615	Thread Group 1-43	HTTP Requ...	15090		68874	257
184	17:49:29.613	Thread Group 1-20	HTTP Requ...	15431		68874	257
185	17:49:29.612	Thread Group 1-19	HTTP Requ...	15450		68874	257

Figura 9.13: Resultados Test Spike para Mi Tienda

resultados (ver Figura 9.14):



Sample #	Start Time	Thread Name	Label	Sample Ti...	Status	Bytes	Sent Bytes
220	17:56:12.140	Thread Group 1-10	HTTP Requ...	465		14590	11
221	17:56:12.143	Thread Group 1-349	HTTP Requ...	462		14548	11
222	17:56:12.146	Thread Group 1-4	HTTP Requ...	461		14526	11
223	17:56:12.146	Thread Group 1-360	HTTP Requ...	461		14501	11
224	17:56:12.146	Thread Group 1-287	HTTP Requ...	462		14580	11
225	17:56:12.147	Thread Group 1-294	HTTP Requ...	461		14572	11
226	17:56:12.148	Thread Group 1-368	HTTP Requ...	461		14522	11
227	17:56:12.159	Thread Group 1-433	HTTP Requ...	458		14528	11
228	17:56:12.159	Thread Group 1-435	HTTP Requ...	460		14555	11
229	17:56:12.160	Thread Group 1-443	HTTP Requ...	463		14530	11
230	17:56:12.158	Thread Group 1-430	HTTP Requ...	467		14558	11
231	17:56:12.154	Thread Group 1-315	HTTP Requ...	472		14514	11
232	17:56:12.156	Thread Group 1-256	HTTP Requ...	470		14583	11
233	17:56:12.158	Thread Group 1-26	HTTP Requ...	469		14542	11
234	17:56:12.158	Thread Group 1-427	HTTP Requ...	469		14484	11
235	17:56:12.156	Thread Group 1-417	HTTP Requ...	471		14518	11
236	17:56:12.158	Thread Group 1-116	HTTP Requ...	469		14545	11
237	17:56:12.163	Thread Group 1-156	HTTP Requ...	464		14540	11
238	17:56:12.158	Thread Group 1-279	HTTP Requ...	471		14508	11
239	17:56:12.155	Thread Group 1-40	HTTP Requ...	474		14533	11
240	17:56:12.157	Thread Group 1-298	HTTP Requ...	472		14557	11
241	17:56:12.154	Thread Group 1-404	HTTP Requ...	476		14611	11
242	17:56:12.157	Thread Group 1-423	HTTP Requ...	476		14520	11
243	17:56:12.155	Thread Group 1-410	HTTP Requ...	478		14532	11
244	17:56:12.173	Thread Group 1-490	HTTP Requ...	460		14546	11

Figura 9.14: Resultados Test Spike para Google

Capítulo 10

Conclusiones.

10.1. Conclusiones.

Durante el desarrollo de esta memoria se ha realizado la descripción de todos los aspectos técnicos que conciernen a este trabajo. El objetivo último era la implementación de un sistema que permitiera la ejecución de tests funcionales de forma automática, además de mostrar cómo se llevarían a cabo de forma manual algunos tests no funcionales. Finalmente, se ha hecho un análisis de los tests ejecutados.

En primer lugar se llevó a cabo la implementación encargada de la ejecución de los tests funcionales. Se desarrolló con el lenguaje de programación Java y haciendo uso de la librería de automatización Selenium WebDriver. Los tests se llevan a cabo después de que Selenium instancie una ventana del navegador Chrome. En concreto se han implementado los siguientes tests:

- Test Respuesta HTTP: comprueba que la página web está disponible. Esto lo realiza verificando el código HTTP que devuelve el servidor cuando se le solicita un recurso.
- Test Existencia de Productos: comprueba que en una página de categoría de productos hay realmente productos. Esto es fundamental en cualquier sitio web que se dedique al comercio electrónico y un error común si el SGBD no indexa bien los productos después de una descarga desde el ERP.
- Test Login: comprueba que se puede hacer login en la página web con una credenciales de un usuario registrado. Hay comercios electrónicos (del tipo B2B) en los que es necesario loguearse para poder comprar por lo que esa función es importante.
- Test Formulario de Contacto: Verifica que el formulario de contacto se puede enviar. Solo se comprueba que el botón de "Enviar formulario" sea clicable con el fin de no llenar de spam la bandeja de entrada del correo electrónico que reciba las peticiones.

En segundo término, se han configurado y ejecutado varios tests no funcionales con la

herramienta JMeter. Esos tests son todos de rendimiento, pues varían la carga y la distribución de la misma a lo largo del tiempo, generando así test de carga, de estres, de *spike*,... Estos tests se lanzaron contra el sitio web creado con WooCommerce.

En tercer lugar, se implementó la aplicación Catón. Su objetivo es la configuración y gestión sencilla de los tests que posteriormente ejecutará Java con Selenium; además de la visualización de los resultados mediante gráficas, donde ver el ratio de error o la pantalla de monitorización donde se actualizan los resultados de los últimos tests realizados.

En cuarto lugar, se realizó el análisis de los tests funcionales y no funcionales lanzados contra el *e-commerce* construido con WooCommerce. Todos los funcionales se ejecutan con resultados positivos.Por la parte de los no funcionales, el test de carga se ejecutó sin problemas, el test de estrés se superó igualmente, y fue el test de *spike* el que mostró las limitaciones de un servidor local al no poder gestionar un aumento de los usuarios tan repentino.

Como conclusión, se han conseguido alcanzar satisfactoriamente todos los objetivos que se plantearon al inicio del proyecto.

Además, es de destacar el conocimiento que he adquirido en una parte importante de la rama de pruebas de software.

Más allá del propio proyecto, consideramos cumplidos otros objetivos, menos formales, que se esbozaron en la introducción de esta memoria. A pesar de partir de una experiencia muy limitada en el ámbito del testeo de software, hemos conseguido introducirnos en esta actividad tanto a través de la teoría, como de la práctica. Hemos realizado una revisión de los principales conceptos implicados desde las pruebas de software en general, hasta el caso de uso específico que hemos planteado en el proyecto. Además, hemos diseñado e implementado una variedad de pruebas que, aunque básicas, nos ha permitido descubrir cómo y con qué herramientas se pueden desarrollar las pruebas para el *testing* de software aplicado a sitios web.

Por tanto, a lo largo del proyecto hemos conseguido afianzar un conocimiento teórico y práctico lo suficientemente sólido como para servir de base a un aprendizaje futuro en la materia. Este proyecto nos ha permitido, además, poner en práctica muchos conocimientos adquiridos durante la realización del Grado, desde el análisis software hasta la consolidación de una planificación y unos presupuestos factibles, utilizando una metodología de actualidad como es Scrum.

Sin embargo, hemos tenido que aplicarlos en un proyecto con un alcance mucho mayor que cualquier práctica. Por ello, ha supuesto un gran reto planificar, gestionar, elaborar y revisar un trabajo de estas características, especialmente teniendo que compaginarlo con una actividad laboral a jornada completa. Tomando en cuenta que se han conseguido llevar adelante todos los objetivos que formaban el alcance del proyecto, consideramos el resultado alcanzado como satisfactorio tanto a nivel de aprendizaje como personal, y un final satisfactorio para mi trayectoria en el Grado.

10.2. Trabajo futuro.

La conclusión de este Trabajo ha dado como resultado dos aplicaciones completamente funcionales tal y como se previó en su inicio. No obstante, ambas tienen margen de mejora y ampliación en sus funcionalidades. A continuación enumeraré las líneas de trabajo que considero se podrían seguir en cada una de ellas.

Respecto a la aplicación de ejecución de tests:

- Se podría añadir una prueba más que verificara que se puede seguir el flujo de compra habitual en un *e-commerce*. Es decir, que ejecutara todos los pasos desde la home y comprobara que se puede llegar hasta la pantalla de pago.
- Sería interesante también añadir una prueba que comprobara que se puede realizar un registro de cliente en el sitio web sin problemas. Esta prueba debería llevar aparejado un proceso en el sitio que borrara de la base de datos el usuario cada vez que se realiza.
- El periodo de repetición de las pruebas debería ser configurable por cada test y sitio.
- También debería ser configurable a qué dirección de correo electrónico llegan las alertas.
- Ahora mismo las pruebas se realizan con el navegador Chrome. Se podría ampliar a más navegadores, con ejecución paralela. Esto lo permite Selenium Grid.
- También sería deseable poder configurar (a través de Catón) y lanzar tests no funcionales de forma automática con Selenium. Para ello, Selenium puede hacer llamadas a JMeter.

Por su parte, la aplicación Catón tiene los siguientes puntos de mejora:

- Actualmente los gráficos que se muestran son todos los tarta con el objetivo de ver el ratio de errores. Sería de utilidad poder visualizar un histórico.
- Para finalizar, algunos clientes pueden querer ver un informe de las pruebas que se están realizando a su sitio. Por ello, sería interesante que los informes de gráficas pudieran imprimirse en formato PDF.

Bibliografía

[1] *AlexSoft*.
https://www.altexsoft.com. (Visitado 14-07-2020).

[2] *China Airlines Flight 140*.
https://en.wikipedia.org/wiki/China_Airlines_Flight_140. (Visitado 15-07-2020).

[3] *CMMI para Desarrollo, versión 1.3*.
https://cmmiinstitute.com/getattachment/4439387f-28aa-4f3a-8f2b-a0cc5b449e47/attachment.aspx. (Visitado 14-07-2020).

[4] *Cucumber*.
https://cucumber.io. (Visitado 15-07-2020).

[5] *Definición de test del ISTQB*.
https://glossary.istqb.org/en/search/. (Visitado 14-07-2020).

[6] *Descripción de WannaCry*.
https://www.kaspersky.es/resource-center/threats/ransomware-wannacry. (Visitado 13-07-2020).

[7] *Guía de Scrum*.
https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Spanish-SouthAmerican.pdf. (Visitado 15-07-2020).

[8] *Guía estudio nivel fundamental ISTQB*.
https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html. (Visitado 15-07-2020).

[9] *Guru99*.
https://www.guru99.com. (Visitado 14-07-2020).

[10] Anne Mette Jonassen hass. *Guide to Advance Software Testing*. Artech House, 2008.

[11] *History's Biggest Software Fails*.
https://www.testbirds.com/blog/historys-biggest-software-fails-5/. (Visitado 15-07-2020).

[12] *IBM Rational Performance Tester*.
https://www.ibm.com/developerworks/downloads/r/rpt/index.html. (Visitado 15-07-2020).

[13] *ISO 9000*.
https://www.iso.org/obp/ui/es/#iso:std:iso:9000:ed-4:v1:es. (Visitado 14-07-2020).

[14] *ISTQB*.
https://www.istqb.org. (Visitado 15-07-2020).

[15] *JMeter*.
https://jmeter.apache.org. (Visitado 15-07-2020).

[16] *JUnit*.
https://junit.org/junit5/. (Visitado 15-07-2020).

[17] *Linkedin Salary*.
https://www.linkedin.com/salary. (Visitado 15-07-2020).

[18] *LoadStorm*.
https://loadstorm.com. (Visitado 15-07-2020).

[19] *Manual de usuario de JMeter*.
https://jmeter.apache.org/usermanual/index.html. (Visitado 12-07-2020).

[20] Glenford J. Myers. *The art of software testing*. John Wiley & Sons, Inc, 2004.

[21] *QTP/UFT*.
https://www.microfocus.com/es-es/products/uft-one/overview. (Visitado 15-07-2020).

[22] *Ranorex Studio*.
https://www.ranorex.com. (Visitado 15-07-2020).

[23] *Rapise*.
https://www.inflectra.com/Rapise/. (Visitado 15-07-2020).

[24] Jonathan Rasmusson. *The Way of the Web Tester*. The pragmatic bookshelf, 2016.

[25] Seb Rose, Matt Wynne y Aslak Hellesøy. *The Cucumber for Java book*. The pragmatic Programmers, 2004.

[26] *Selenium*.
https://www.selenium.dev. (Visitado 15-07-2020).

[27] *Software Testing Fundamentals*.
http://softwaretestingfundamentals.com. (Visitado 14-07-2020).

[28] *Software Testing Help*.
https://www.softwaretestinghelp.com. (Visitado 14-07-2020).

[29] *SSTQB*.
http://www.sstqb.es. (Visitado 15-07-2020).

[30] *Starbucks stores reopen after computer glitch led to free coffee for some*.
https://www.theguardian.com/business/2015/apr/25/starbucks-fixes-computer-glitch-that-gave-customers-free-coffee. (Visitado 15-07-2020).

[31] *Testing Whiz*.
`https://www.testing-whiz.com`. (Visitado 15-07-2020).

[32] *TMMI*.
`https://www.tmmi.org/tmmi-model/`. (Visitado 14-07-2020).

[33] *Tools QA*.
`https://www.toolsqa.com`. (Visitado 14-07-2020).

[34] *w3af*.
`http://w3af.org`. (Visitado 15-07-2020).

[35] Karl Wiegers y Joy Beatty. *Software requirements*. Pearson Education, 2013.

[36] *Wikipedia*.
`https://es.wikipedia.org`. (Visitado 14-07-2020).

[37] *Windows 10 security bug leaves your PC vulnerable to attack*.
`https://www.techradar.com/news/the-nsa-has-found-a-serious-windows-10-bug-that-you-need-to-patch-right-now`. (Visitado 15-07-2020).

[38] *WooCommerce*.
`https://woocommerce.com/`. (Visitado 15-07-2020).

[39] *WordPress*.
`https://es.wordpress.com`. (Visitado 15-07-2020).

Anexos

Anexo A

Manual de usuario del sistema.

A.1. Manual de usuario de Catón.

Al entrar a la pantalla *Home* podemos ver cuatro opciones en el menú lateral izquierdo:

- *New Client*: permite la gestión de los clientes y los sitios webs asociados a cada uno de ellos.

  La Figura A.1 muestra las dos partes de que consta este apartado. La izquierda está dedicada a la gestión de los clientes ya existentes, y la derecha al registro de los mismos. Un sitio web tiene que estar asociado a un cliente, y un cliente puede tener una o varias webs.

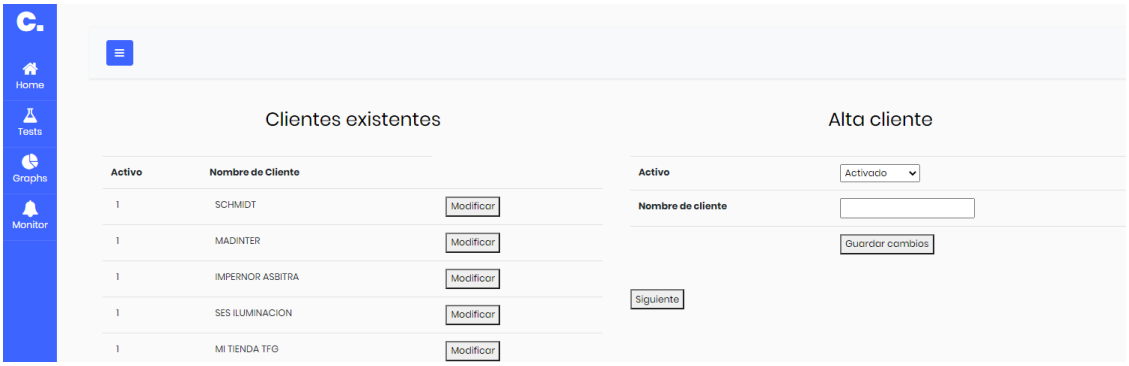


Figura A.1: Gestión de clientes.

  Pulsando en el botón de "Modificar" asociado a un cliente, podemos editar sus datos: su nombre y su condición de activo o inactivo en la plataforma (ver Figura A.2).

  El botón "Siguiente" que aparece en la parte derecha de la Figura A.1 conduce a que se muestre la pantalla de Gestión de sitios web (Figura A.3).

Figura A.2: Modificación de clientes.

Esta interfaz también está dividida en dos secciones: la izquierda donde se muestran todos los sitios web que están registrados en el sistema y la derecha donde se dan de alta nuevas webs y se especifica qué pruebas, de las disponibles, se van a configurar posteriormente.

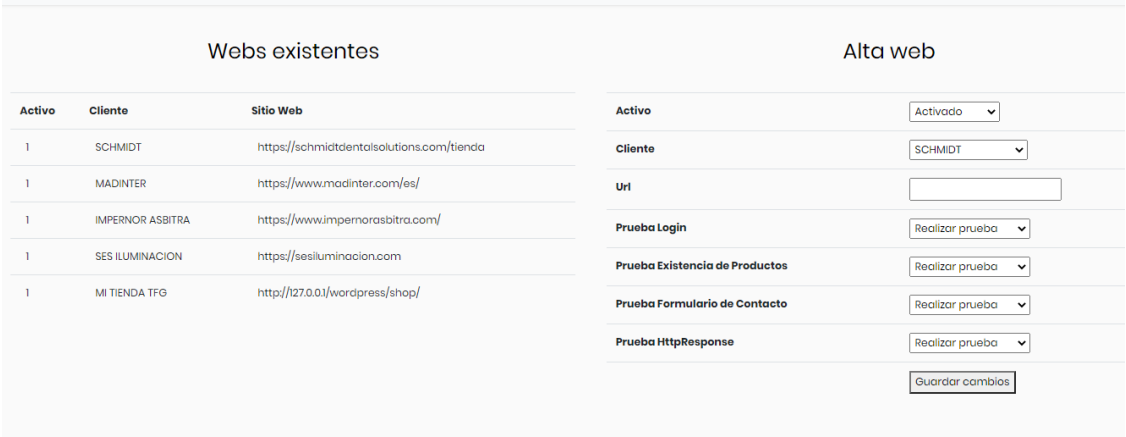


Figura A.3: Gestión de sitios web.

- *Tests*: En esta sección se gestionan los tests que están registrados en el sistema y se dan de alta otros nuevos. En esta pantalla (Figura A.4), para cada tipo de test, hay dos botones: "Ver" y "Añadir test".

  - Pulsando "Ver", se pueden visualizar todos los tests de ese tipo que estén registrados en el sistema en todas las páginas, tal y como se muestra, para el caso particular de las pruebas de Formulario de Contacto, en la Figura A.5.
  - Pulsando "Añadir Test", se puede registrar un test del tipo correspondiente. En la Figura A.6 se muestra el formulario para registrar un test de tipo Existencia de

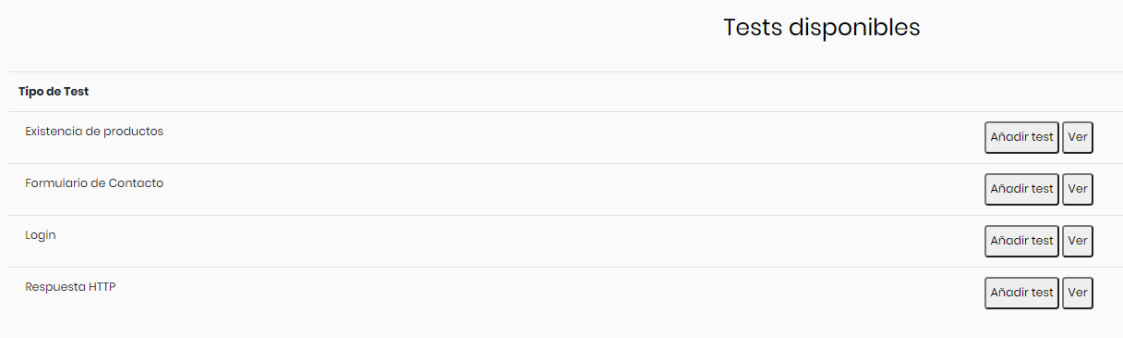


Figura A.4: Gestión de pruebas.

Figura A.5: Listado de pruebas Formulario de Contacto.

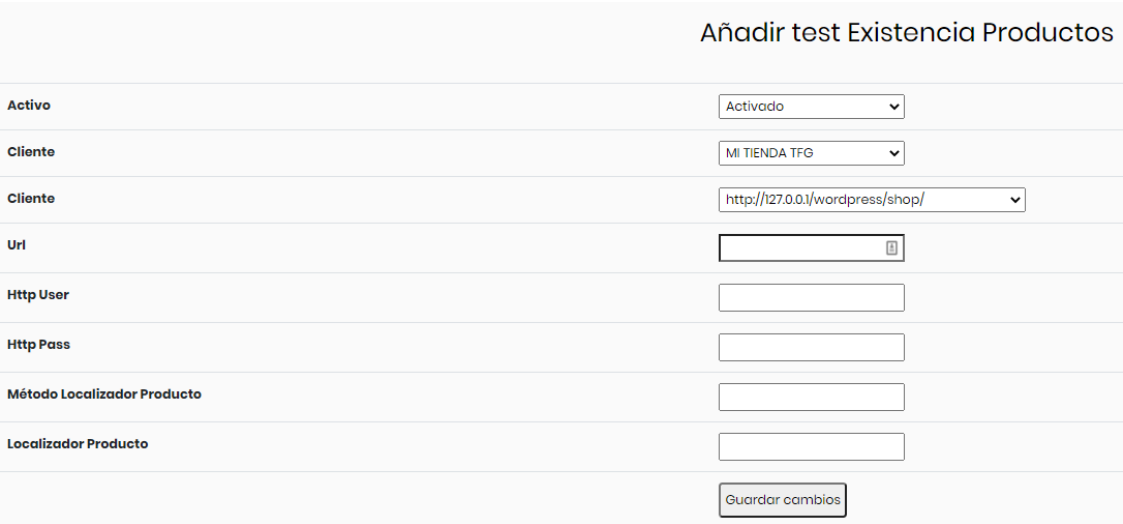


Figura A.6: Añadir Test: Existencia de productos.

Productos. Los campos de este formulario variarán en función del tipo de prueba porque la configuración para cada uno de ellos es diferente.

Desde la pantalla *Listado de tests disponibles* (Figura A.5), se puede acceder a la pantalla de modificación de las pruebas registradas (ver Figura A.7).

- *Graphs*: en esta sección se visualiza el ratio pruebas fallidas y exitosas por cada tipo de test y sitio web.

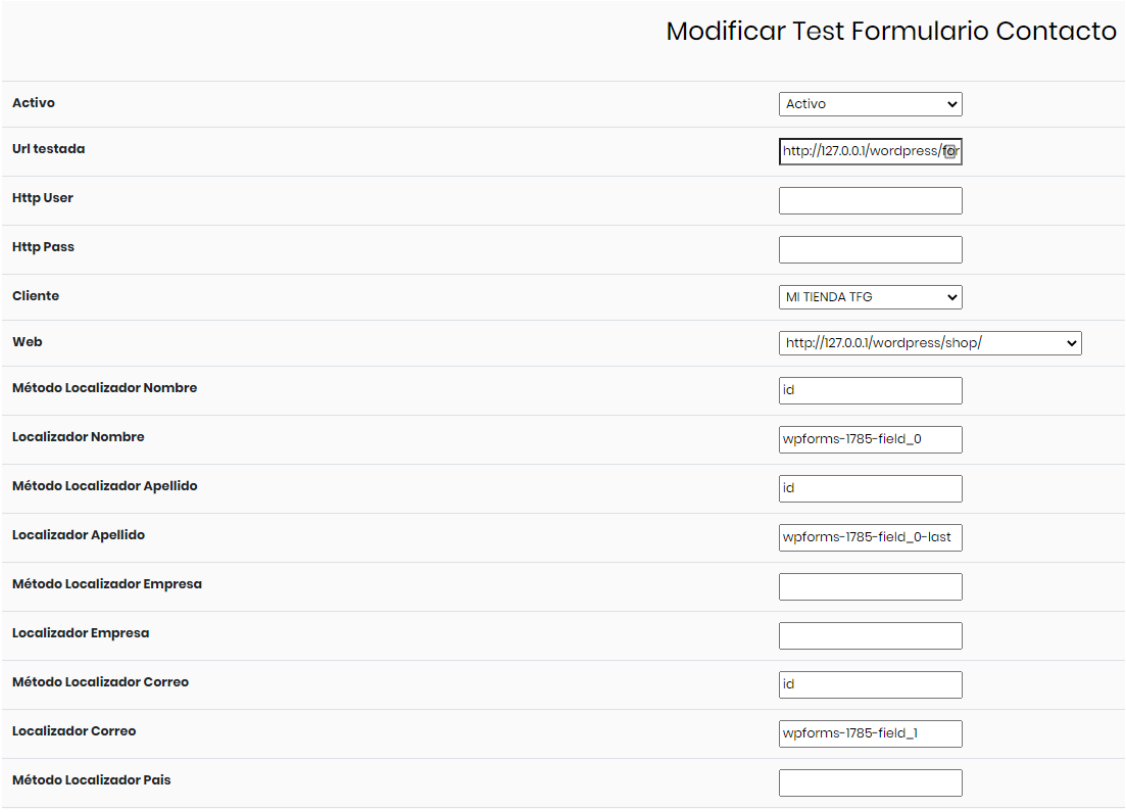


Figura A.7: Modificar Test: Formulario de Contacto.

Figura A.8: Gráficos de resultados.

Para ello, es necesario seleccionar la web y el rango de fechas por el que se quiere filtrar, y se mostrarán los gráficos de tarta correspondientes a los resultados de las pruebas ejecutadas sobre esa URL.

- *Monitor*: En esta sección se muestran los últimos errores registrados. Tiene un refresco automático de 10 segundos para asegurar que los datos disponibles en pantalla representan la situación actual del estado de las pruebas.

  A la izquierda, se reflejan todos los resultados fallidos en cualquiera de los tipos de pruebas, para ofrecer una vista rápida en relación a la ocurrencia de errores. A la derecha, se muestran los últimos resultados de las pruebas de cada uno de los tipos de

test existentes. Puede navegarse entre unos y otros mediante la cabecera de pestañas.

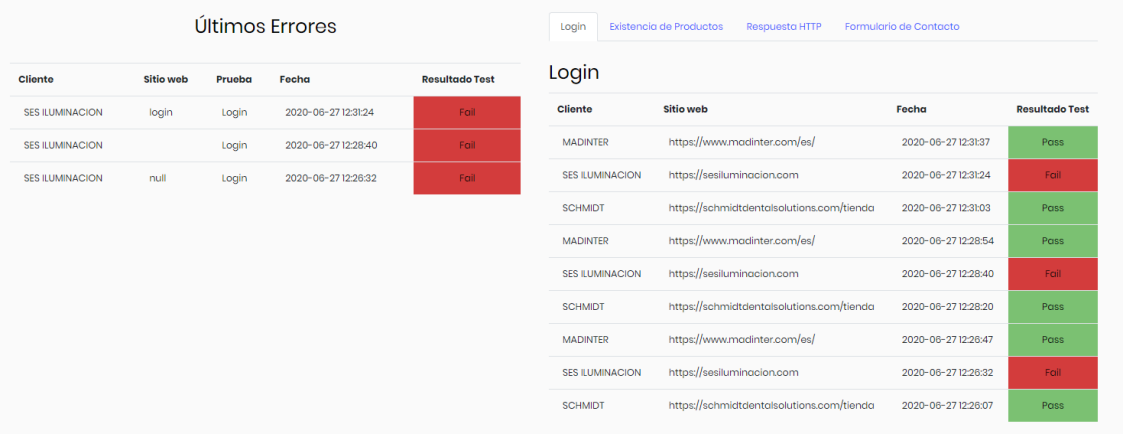


Figura A.9: Monitor.

A.2. Manual de usuario para la ejecución de tests.

El conjunto de tests se entrega empaquetado en un fichero *.jar* ejecutable mediante el comando de consola:

```
> java -jar PruebasDisponibilidad.jar
```

Este paquete contiene todas las dependencias necesarias para el funcionamiento del proyecto.

Además, también se hace entrega del *driver* necesario para instanciar el navegador Chrome (ChromeDriver) en su versión 83.0. Éste debe situarse en la dirección `"C:/"`.

Cuando se ejecuta el fichero *PruebasDisponibilidad.jar* es necesario tener en funcionamiento y accesible la base de datos de Catón, pues es allí donde irá a buscar los parámetros que necesita para ejecutar las pruebas. Una vez inicie su ejecución, Java instanciará cuatro ventanas de Chrome donde se ejecutarán los respectivos tipos de pruebas con la frecuencia establecida.

Anexo B

Manual de usuario de las herramientas utilizadas

B.1. Selenium

B.1.1. Configuración de Selenium WebDriver

B.1.1.1. Descargar Selenium WebDriver

Lo primero que hay que hacer es descargar el Driver Server para el navegador que vayamos a utilizar en nuestro testing. En nuestro caso se usará Chrome, por lo que la dirección donde habrá que dirigirse será: `https://sites.google.com/a/chromium.org/chromedriver/`

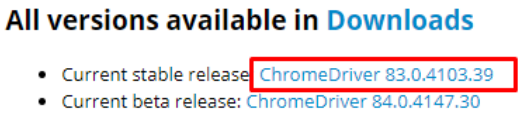


Figura B.1: Selenium Chrome Driver

B.1.1.2. Instanciar WebDriver

Posteriormente, ya en el código de nuestra aplicación será necesario instanciar un objeto de la clase *ChromeDriver*:

```
WebDriver driver = new ChromeDriver();
```

y establecer su localización en el sistema de archivos del mediante:

```
String exePath ="C:/Users/abc/Desktop/Server/chromedriver.exe";
System.setProperty("webdriver.chrome.driver", exePath);
```

B.1.2. Comandos básicos de Selenium WebDriver

B.1.2.1. Abrir una página web

```
driver.get(appUrl);
```

donde appUrl es la dirección de la aplicación web.

B.1.2.2. Cerrar el navegador

```
driver.close();
```

Cierra el navegador si solo hay una ventana abierta.

Para cerrar todo completamente, aunque haya varias pestañas abiertas tendríamos que usar:

```
driver.quit();
```

B.1.2.3. Comandos de navegación

Abrir una nueva pestaña en el mismo navegador:

```
driver.navigate().to(appUrl);
```

Para adelantar una página en la navegación se utilizaría:

```
driver.navigate().forward();
```

Del mismo modo, para ir hacia atrás:

```
driver.navigate().back();
```

B.1.2.4. Comandos para interactuar con elementos HTML

En Selenium cada elemento del DOM de una página puede ser almacenado en un objeto de tipo *WebElement*. A ellos, se accede mediante el método *findElement(By.<criterio>)* o *findElements(By.<criterio>)* si es un conjunto (como podría ser el caso de una lista).

```
WebElement element = driver.findElement(By.id("UserName"));
```

Si quisiéramos escribir en un campo tipo texto, se debería utilizar el método *sendKeys()*:

```
element.sendKeys("text");
```

Para clicar sobre un botón, se dispone del método *click()*:

```
element.click();
```

en el caso de que se quiera comprobar si un elemento HTML está visible:

```
element.isDisplayed();
```

que retorna un valor *booleano*.

Para enviar formularios, Selenium tiene el método *submit()*:

```
element.submit();
```

B.1.3. Localizadores y XPath

En esta subsección se explican las diferentes formas de encontrar un elemento HTML en el DOM.

1. Por ID: si el elemento que buscamos tiene un id único.

   ```
   WebElement elementPassword = driver.findElement(By.id("password"));
   ```

2. Por Nombre: el *name* de un elemento puede no ser único, por lo que hay que tener cuidado con este método de localización.

   ```
   WebElement elementPassword = driver.findElement(By.name("password"));
   ```

3. Por texto de un enlace: el texto que se muestra al usuario en un enlace *<a>* puede servir como parámetro a la función *findElement()*:

   ```
   WebElement element=driver.findElement(By.linkText("Name of the Link"));
   ```

4. XPath: mediante la ruta del elemento a través de todo el árbol DOM también se puede localizar.

   ```
   WebElement element = driver.findElement(By.xpath("//*[@name='loginForm']/input[1]")
   ```

5. Situación en el DOM: otra estrategia parecida a la anterior.

   ```
   document.forms['loginForm'].elements['username']
   ```

B.1.4. Clase Acciones

La clase *Actions* se utiliza para emular un comportamiento complejo del usuario en la GUI, es recomendable hacer uso de ella cuando se requiere hacer uso del teclado o el ratón.

MÉTODOS DE TECLADO:

- *keyDown(tecla)*: Simula el presionado de una tecla.
- *sendKeys(texto a escribir)*: escribe en el elemento de la página web.

- *keyUp(tecla)*: Simula dejar de presionar una tecla.

MÉTODOS DE RATÓN:

- *click()*: Clica sobre el elemento en el que está posicionado en ese instante el ratón.
- *doubleClick()*: Ejecuta un doble click sobre el elemento en que esté posicionado el ratón.
- *contextClick()* : Clica con el botón derecho del ratón sobre el elemento seleccionado.
- *clickAndHold()*: Clica y mantiene el click en el elemento.
- *dragAndDrop(origen, destino)*: Selecciona y mueve el elemento desde la posición origen a la de destino.
- *dragAndDropBy(origen, xOffset, yOffset)*: Mueve el elemento desde la posición origen hasta las coordenadas proporcionadas (xOffset, yOffset).
- *moveByOffset(x-offset, y-offset)*: mueve el ratón desde su posición actual hasta las coordenadas dadas.
- *moveToElement(elementoDestino)*: mueve el ratón hasta el centro del elemento.

B.2. JMeter

B.2.1. Descarga e instalación

JMeter es una aplicación desarrollada enteramente en Java, por lo que el primer paso para utilizarla, es tener descargada la máquina virtual de Java.

Posteriormente, podremos descargar JMeter de este enlace:

`http://jmeter.apache.org/download_jmeter.cgi`

Para finalizar, podremos lanzar JMeter ejecutando el archivo *.bat* que está en su directorio *bin*.

B.2.2. Elementos

Los componentes fundamentales con los que se definen los test en JMeter se llaman Elementos. Cada elemento está diseñado para un propósito determinado, aunque aquí nos quedaremos con los cuatro principales.

La siguiente imagen muestra una relación de una gran parte de ellos:

1. Grupos de hilos: Se trata de un conjunto de hilos. Cada hilo representa un usuario usando la aplicación.

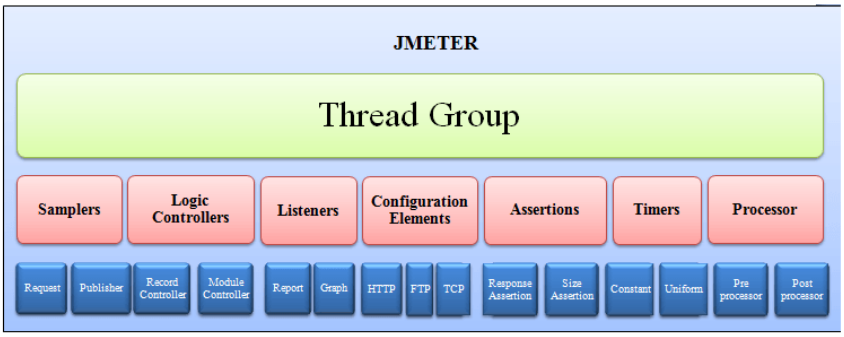


Figura B.2: Elementos JMeter

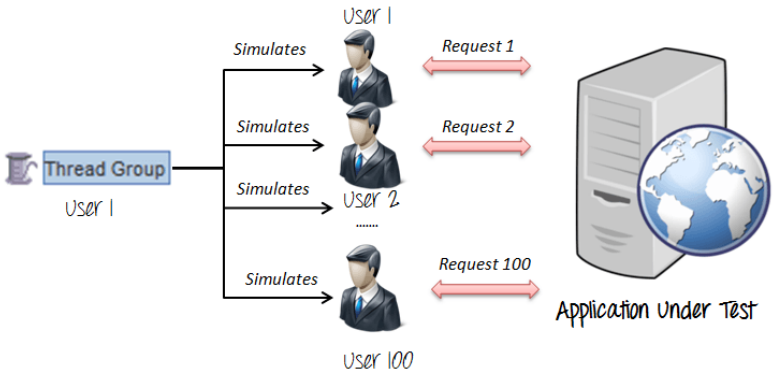


Figura B.3: Thread Group

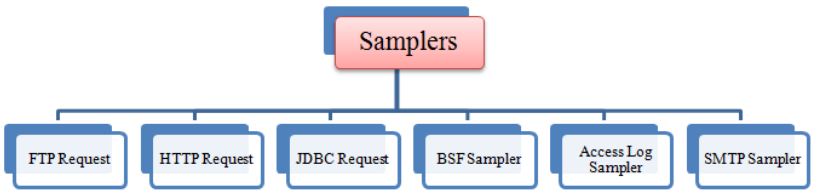


Figura B.4: *Samplers* de JMeter

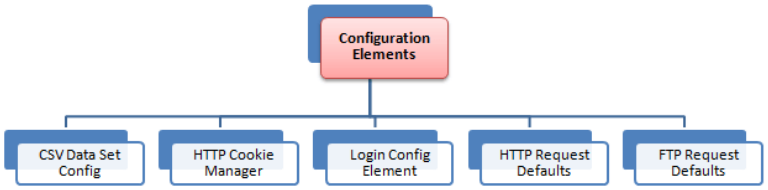


Figura B.5: *Configurations* de JMeter

2. *Samplers:* son conjuntos de otras peticiones que puede hacer el usuario al servidor, como peticiones HTTP, FTP, JDBC...

3. *Listeners:* son los encargados de mostrar los resultados de un test. Los pueden mostrar en forma de tabla, gráfico, fichero log o árbol.

4. Configuración: establecen los valores de las variables que usarán los *samplers*.

B.2.3. Interfaz gráfica

B.2.3.1. Plan de test

El plan de test es donde se almacenan todos los elementos que son necesarios para la ejecución del test, como los temporizadores, los *samplers*...junto con sus configuraciones.

B.2.3.2. *Workbench*

El *workbench* es un almacenamiento temporal de los elementos del test. No tiene relación con el plan de test explicado en el punto anterior y JMeter no guardará el contenido suyo cuando se cierre.

B.2.4. Temporizadores

El modo por defecto de funcionamiento de JMeter es enviar las peticiones al servidor una tras otra, sin pausas entre ellas. Esto puede llevar a una sobrecarga del servidor en un periodo de tiempo relativamente corto, dependiendo de la cantidad de hilos que hayamos configurado.

Para evitar ese problema están los temporizadores, que permiten espaciar las peticiones un tiempo determinado. Hay varias formas de establecer ese intervalo, pero las principales son:

- Temporizador constante: espacia las consultas una cantidad constante de segundos.
- Con aleatoriedad gaussiana: los intervalos tienen una desviación permitida de segundos.
- Con aleatoriedad uniforme: los intervalos aleatorios no tienen una desviación.

B.2.5. Aserciones

Las aserciones son elementos que sirven para comprobar que los resultados obtenidos del test, son los que se esperaban.

Dependiendo del tipo de petición que se haya realizado, las aserciones deberán ser de alguna de las siguientes categorías:

- De tipo de respuesta
- Duración
- Tamaño (por ejemplo el tamaño de un fichero después de una petición FTP)
- XML
- HTML

B.2.6. Controladores

Un hilo debe seguir una secuencia de pasos para que el test se ejecute según lo previsto. De que esos pasos se realicen en el orden correcto se encargan los controladores.

Algunos de los controladores lógicos más usados son:

- Bucles
- Aleatorios (para enviar por ejemplo peticiones HTTP aleatorias a un servidor)
- Simples
- Vídeo: JMeter permite grabar los pasos de un test para luego convertirlo en código. De esa grabación se encarga este tipo de controlador.

B.2.7. *Processors*

Un *processor* se utiliza para modificar el comportamiento de un *sampler*, añadiendo funcionalidad.

Dependiendo de dónde se realice ese cambio, si antes o después de la ejecución del *sampler* distinguiremos entre:

- *Preprocessors*: se añade la funcionalidad justo antes de enviar la petición al servidor.
- *Postprocessors*: la funcionalidad se ejecuta cuando la petición ya ha sido respondida.

B.3. WooCommerce

B.3.1. Instalación de XAMPP

XAMPP es una aplicación de software libre que contiene un servidor Apache, el sistema MySQL e intérpretes para PHP y Perl. Desde su versión 5.6.15 cambió MySQL por MariaDB.

Para su descarga hay que dirigirse a la web: *https://www.apachefriends.org/es/download.html* y ahí descargar el ejecutable adecuado para nuestro sistema operativo y arquitectura.

Una vez descargado, en Windows se puede ejecutar su instalador, que guiará en su proceso de instalación en el equipo.

En esta ocasión se ha instalado de modo local en mi ordenador.

B.3.2. Instalación de WordPress

Para descargar WordPress hay que ir a la URL : *https://es.wordpress.org/download/* y descargar la versión que queramos. En mi caso he descargado la más reciente hasta la fecha, la versión 5.4.1.

Se bajará un archivo comprimido que hay que descomprimir en la carpeta correspondiente del servidor, en este caso el directorio *htdocs* de Apache. Allí crearemos una carpeta con el nombre de nuestro WordPress y en ella situaremos las carpetas descargadas.

Posteriormente, y después de iniciar el servidor y el motor de base de datos, ingresaremos a la dirección de nuestro localhost (127.0.0.1) y a la carpeta con nuestro proyecto. Una vez allí WordPress nos guiará en el proceso de instalación y configuración de la base de datos.

Figura B.6: Instalación de WordPress. Paso 1

Figura B.7: Instalación de WordPress. Paso 2

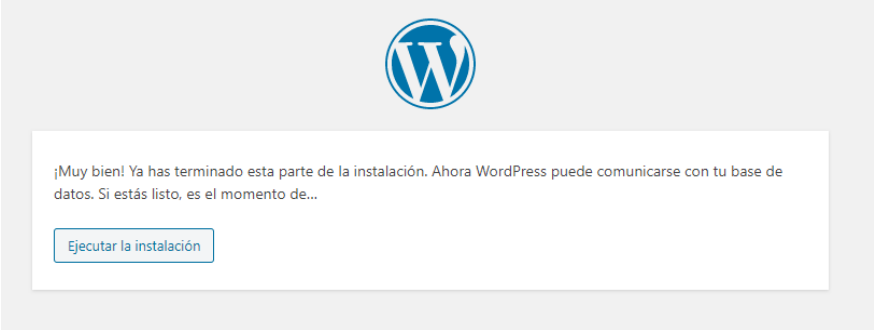


Figura B.8: Instalación de WordPress. Paso 3

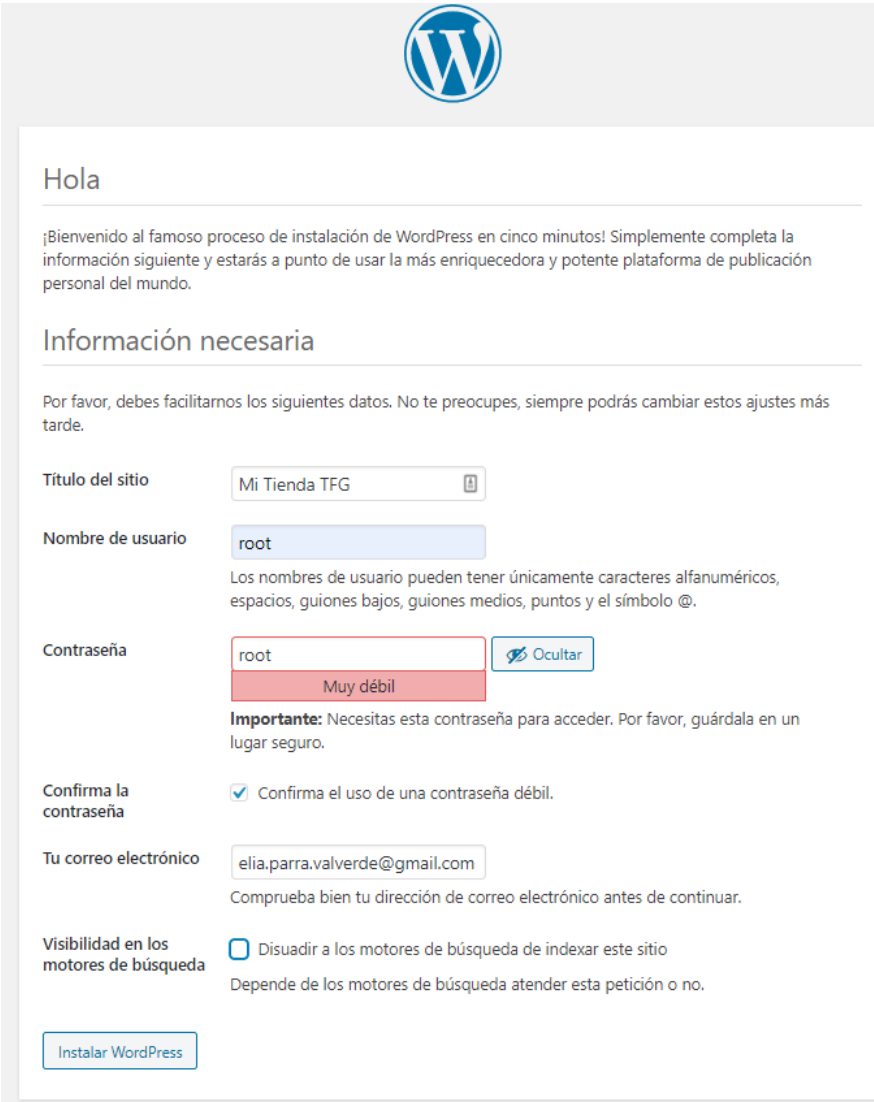


Figura B.9: Instalación de WordPress. Paso 4

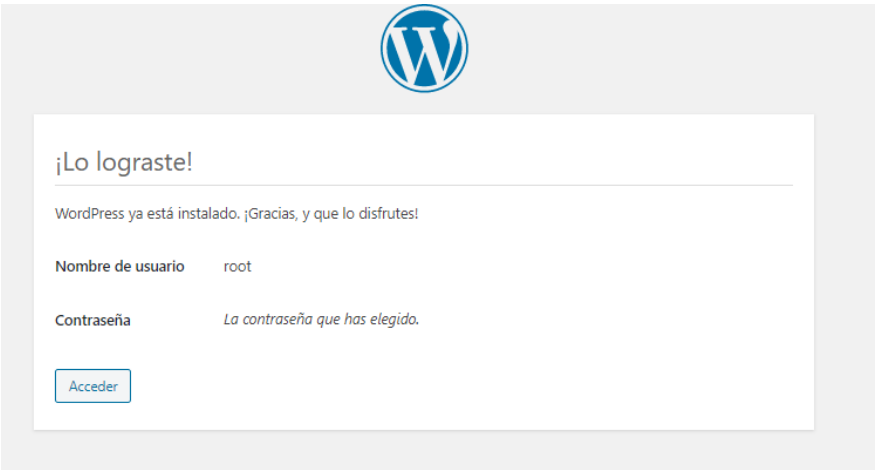


Figura B.10: Instalación de WordPress. Paso 5

Figura B.11: Instalación de WooCommerce.

B.3.3. Instalación de WooCommerce

El primer paso para convertir el WordPress en una tienda virtual es acceder al panel de administración (127.0.0.1/wordpress/wp-admin) con las credenciales adecuadas.

Una vez allí, en la sección de Plugins >Añadir nuevo, se debe buscar el plugin de WooCommerce.

Solo quedará presionar el botón "Instalar ahora" y se añadirán automáticamente todos los componentes de la tienda a nuestro WordPress.

B.3.4. Instalación de la plantilla

Para tener una maquetación apropiada a nuestro *site* debemos cambiar la plantilla que trae por defecto WordPress. Para ello nos dirigiremos a la sección Apariencia >Temas, y ahí presionaremos "Añadir tema". Eso nos cargará todo el catálogo de temas disponibles para WordPress, algunos de los cuales pensados especialmente para WooCommerce.

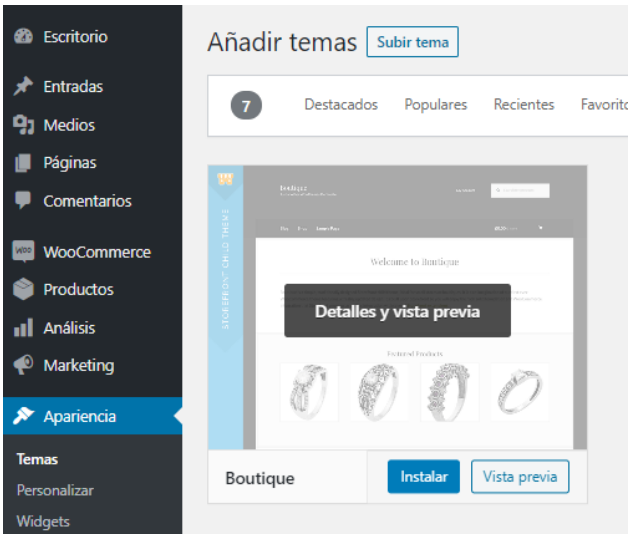


Figura B.12: Instalación de la plantilla.

En este caso se ha elegido la plantilla 'Boutique" en la versión 2.0.17.

Finalmente, habrá que hacer click en el botón "Instalar" y se nos instalará y cambiará la maquetación completa de la web.

B.3.5. Carga de datos de prueba

Lo primero que hay que hacer para hacer una carga de datos de prueba es ir a la página *https://docs.woocommerce.com/document/product-csv-importer-exporter/dummy-data/* y descargar el fichero CSV.

En el panel de control, nos tendremos que dirigir a la sección Productos y clicar el botón "Importar". Se nos mostrará un formulario como el de la imagen, solo tenemos que seleccionar el fichero CSV descargado y WordPress se encargará de rellenar las tablas de la base de datos.

B.3.6. Instalación del formulario de contacto

Se ha instalado el plugin WPForms Lite en su versión 1.6.1, de la misma forma que el de la tienda virtual.

B.3.7. Instalación del formulario de registro

Se ha instalado el plugin *User Registration* en la versión 1.8.4, del mismo modo que se instaló WooCommerce.

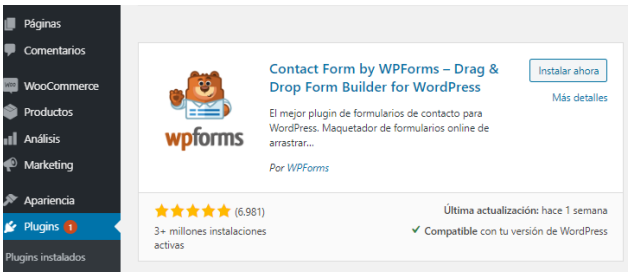


Figura B.13: Instalación del formulario de contacto

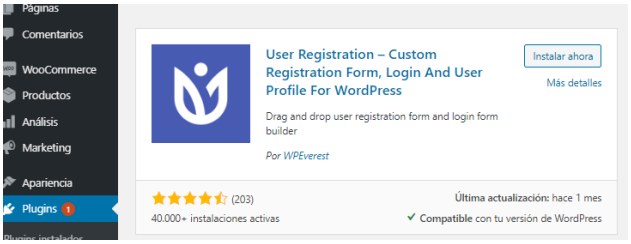


Figura B.14: Instalación del formulario de registro