UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Testing Automatizado de Módulo de Promociones

INGENIERÍA DEL SOFTWARE

Memoria del proyecto

Trabajo Final de Grado

Autor: **Muhammad Meraj Ali Anjum**
Director: **David Lema Somoza**
Ponente: **Luis Antonio Belanche Muñoz**
Tutor GEP: **Joan Subirats Soler**
Fecha de la defensa: 26 de enero de 2022

RESUMEN

Gracias al avance de la tecnología cada vez el ser humano utiliza técnicas que ayudan a mejorar el trabajo de forma eficiente y económicamente. En muchos ámbitos y proyectos se están utilizando la automatización de procesos para tener un control mejor y ahorrar tiempo para realizar tareas. En este proyecto de final de grado elaborado con la colaboración de NTT Data (anteriormente Everis) se ha realizado la automatización del módulo de promociones en Drupal con el que se prende minimizar los problemas de *testing* de un módulo de promociones.

Esta memoria analiza y desarrolla un conjunto de pruebas de *testing* automatizado para agilizar los procesos de detectar errores en el módulo de promociones. Concretamente, se han automatizado las pruebas que hasta ahora se realizaban manualmente. La finalidad de este proyecto es ofrecer una herramienta que realice las pruebas de forma automática haciendo uso de la tecnología para solucionar los problemas del *testing*. Este producto de *testing automatizado* se ha desarrollado mediante la tecnología Selenium, un software que permite automatizar *testing*, el lenguaje de Java, TestNG, Maven. Para el desarrollo se ha utilizado la metodología *agile* que ha permitido finalizar el proyecto satisfactoriamente.

RESUM

Gràcies a l'avanç de la tecnologia cada vegada l'ésser humà utilitza tècniques que ajuden a millorar el treball de forma eficient i econòmicament. En molts àmbits i projectes s'estan utilitzant l'automatització de processos per tenir un millor controlar i estalviar temps per realitzar tasques. En aquest projecte de final de grau elaborat amb la col·laboració de NTT Data (anteriorment conegut com Everis) s'ha realitzat l'automatització del mòdul de promocions a Drupal amb què es pren minimitzar els problemes de *testing* del mòdul de promocions.

Aquesta memòria analitza i desenvolupa un conjunt de proves de *testing* automatitzat per agilitzar els processos de detectar errors al mòdul de promocions. Concretament, s'han automatitzat les proves que fins ara es feien manualment. La finalitat d'aquest projecte és oferir una eina que faci les proves de manera automàtica fent ús de la tecnologia per solucionar els problemes del *testing*. Aquest projecte de *testing automatitzat* s'ha desenvolupat mitjançant la tecnologia Selenium, software que permet automatitzar *testing*, el llenguatge de Java, TestNG, Maven. Per al desenvolupament s'ha fet servir la metodologia *agile* que ha permet finalitzar el projecte satisfactòriament.

ABSTRACT

Thanks to the advancement of technology, every time the human being uses techniques that help to improve work efficiently and economically. In many areas and projects, process automation is being used to have better control and save time to complete tasks. In this final degree project elaborated with the collaboration of NTT Data (previously known as Everis), the automation of the promotions module in Drupal has been carried out with which it is possible to minimize the testing problems of a promotions module.

This report analyses and develops a set of automated testing test to streamline the processes of detecting errors in the promotions module. Specifically, tests that until now were carried out manually have been automated. The purpose of this project is to offer a tool that performs tests automatically using technology to solve testing problems. This automated testing product has been developed using Selenium technology, a software that allows automating testing, the Java language, TestNG and Maven. For the development, the agile methodology has been used that has allowed the project to be completed satisfactorily.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Contextualización

El trabajo final de grado (TFG) es una asignatura obligatoria en cualquier carrera de la universidad. Como alumno de ingeniería de la Facultad de Informática de Barcelona (FIB) de la UPC, especializado en ingeniería del software, mi proyecto para este trabajo es el “Testing Automatizado de Módulo de Promociones”.

El proyecto de *testing* se realizará con la colaboración de la empresa NTT Data (anteriormente conocido Everis), una compañía multinacional especializada en consultoría de aplicaciones tecnológicas. Everis abarca diferentes sectores como telecomunicaciones, industria, etc. Consta de más de 27000 empleados en diecisiete países diferentes [1]. De los diferentes clientes que tiene esta compañía, uno de ellos es Damm, una empresa con un gran valor en el mercado de las bebidas, especialmente en cerveza.

La empresa Damm desde su fundación en el año 1876 hasta ahora ha ido creciendo a nivel mundial. Actualmente cuenta con una plantilla de 4707 personas. Además, destaca en las sociedades como bebidas, restauración, distribución, logística y otras actividades en las que sigue expandiendo día a día para diversificar su actividad y ofrecer soluciones en todos los ámbitos de negocio. Por otra parte, la empresa fabrica más de 25 marcas de cerveza, 16 plantas de producción y envasado, y está presente en más de 133 países del mundo [2].

Esta empresa ha requerido el proyecto de *testing* para sus diferentes páginas webs que están conectadas mediante un módulo en Drupal.

1.2. Identificación del problema

Actualmente, el cliente tiene contratado diferentes servicios con Everis. Por ejemplo, mantenimiento de páginas webs, nuevos proyectos para sus páginas webs o aplicaciones móviles. Al ser una empresa que ha ido creciendo en los últimos años, desde la empresa han observado que necesitan ser más eficientes para poder finalizar los proyectos y así mejorar su rendimiento. Concretamente, tanto el equipo de programadores de Everis como del cliente han identificado la pérdida de tiempo a la hora de efectuar el *testing* para comprobar el funcionamiento de la aplicación. Además, hay algunas pruebas que en la comprobación manual no son fáciles de validar, por tanto, valoran positivamente la idea de realizar el *testing* de forma automatizada.

Para entender mejor el problema identificado, se puede decir afirmar que desde la petición que recibe el equipo de Damm de sus usuarios para informar de un error hasta que se corrige el error pueden llegar a pasar un par de días. Esto sucede porque el error pasa por muchas personas hasta llegar al equipo de incidencias que actualizan el código para corregir el error. Otra razón es que algunas pruebas resultan difíciles de validar manualmente porque el flujo de procesos no muestra el resultado esperado, por tanto, el equipo de *testing* muchas de las pruebas terminan como resultado no alcanzado. Por esta razón, se ha propuesto este el proyecto de pruebas automatizadas. Cabe mencionar, como se trata de un módulo con 25 páginas webs, se pueden crear *testing* automático para todas las páginas ya que todas tienen algunos procesos que no necesitan la comprobación manual.

1.3. Definiciones

En este apartado veremos algunos conceptos específicos que pueden tener diferentes interpretaciones y que se considera que deberían definir para no confundir a lo largo del trabajo.

- ***Testing* automático:** Cuando se habla del *testing*, entendemos que se trata de comprobar el funcionamiento de un proceso o desarrollo. En cambio, el *testing* automático es la realización de diferentes pruebas de un proceso de forma automática y repetida, sin la necesidad de la intervención manual [3].

- **Plan de pruebas:** Conjunto de pruebas necesarias para poder validar el funcionamiento del sistema. Estas pruebas suelen tener todas las instrucciones para poder llegar a validar un proceso y confirmar el resultado de las pruebas.

- **Drupal** [4]**:** Es un sistema de gestión que permite crear aplicaciones web de forma eficiente y ahorrar un tiempo reducido del desarrollo.

- **Promociones:** Son campañas publicitarias de actividades que ofrecen la posibilidad de ganar premios o descuentos por participar.

- **Módulo:** Es una parte de un programa que efectúa comúnmente una tarea concreta para cumplir con su funcionamiento.

- **Página web:** Documento que forma parte de un lugar web y es accesible con un navegador con una dirección web.

- ***Landing page* (página de aterrizaje)** [5]**:** Es una página web en la que una persona llega después de pulsar un anuncio, un enlace, un botón. En general, no es posible acceder a una página de aterrizaje a través de sitios webs públicos porque se crean especialmente para optimizar las campañas publicitarias.

1.4. *Stakeholders*

En este apartado se va a presentar los actores implicados (*stakeholders*) que son las personas afectadas, interesadas, participantes y/o beneficiarios del proyecto.

- **Cliente**: Actor importante implicado por ser el principal beneficiario de este proyecto porque en el futuro tendrá un gran beneficio económico y podrá ahorrar mucho tiempo por no encargar a una persona para realizar el *testing*.

- **Equipo de desarrollo:** Consta de dos programadores, un programador experto en la implementación de pruebas automatizadas y el realizador del trabajo final de proyecto.

- **Equipo de desarrollo de páginas webs:** Es un *stakeholder* importante ya que los desarrolladores harán uso de este proyecto para poder validar el *testing*, por tanto, se considera uno de los *stakeholders* principales.

- **Responsable del proyecto:** Persona que se encargará de hacer seguimiento del proyecto, resolver dudas funcionales para poder realizar la implementación del *testing* automático junto con el analista senior.

- **Analista Senior**: Persona con cargo funcional que conoce todos los procesos en detalle del módulo y de las páginas webs del cliente. Se encarga de analizar y especificar nuevos requerimientos para los desarrolladores. En este proyecto, ayuda a entender el funcionamiento del módulo a los desarrolladores para que se pueda implementar de forma correcta las pruebas.

- **Tutor del Trabajo Final de Grado (TFG)**: David Lema es el director del proyecto y asesora el proyecto para que pueda llegar a finalizar con éxito. También es una de las personas más importantes del proyecto, por ser uno de los lideres de los proyectos webs del cliente Damm.

- **Ponente del TFG**: Lluis Belanche es el ponente de este trabajo y tiene una implicación importante para que pueda elaborar este proyecto de forma correcta. Además, también ayudará a orientar el trabajo para poder finalizar la memoria del TFG tal como se espera que un alumno de la FIB finalice un trabajo de este tipo.

- **Realizador del TFG**: El autor de este trabajo, Muhammad Meraj, es un actor principal implicado en este proyecto, con el objetivo de acabar el proyecto con buenos resultados, adquirir nuevos conocimientos y experiencias para su futuro profesional.

2. Justificación

2.1. Análisis del mercado

Una vez presentado el proyecto, en este apartado se va presentar las herramientas que se pueden encontrar en el mercado para poder solucionar el problema. Hoy en día, existen diferentes *frameworks* que proporcionan herramientas para poder realizar pruebas automatizadas de páginas webs. A continuación, veremos algunos de los candidatos que pueden ser de ayuda para este proyecto [6].

- **Selenium**: Es un *framework* que ofrece un entorno para diseñar pruebas automáticas para aplicaciones web. Para escribir el código proporciona la posibilidad de escribir en diferentes lenguajes de programación como Java, C#, Python, Ruby, PHP, etc. Las pruebas con Selenium pueden ser ejecutadas en cualquier navegador moderno de Windows, Linux, Android o IOS. Además, tiene la herramienta Selenium IDE con la que se puede grabar o reproducir pruebas sin necesidad de usar ningún lenguaje de programación [7].

- **Katalon Studio**: Es una herramienta de automatización de *testing* desarrollada por Katalon [8]. El software es de código abierto y está basado en los frameworks Selenium y Appium con una interfaz especializada para crear pruebas automatizadas en navegadores webs, API, móviles y aplicación de escritorio. Katalon Studio está diseñado para que sea fácil de usar.

- **Watir**: *Web Application Test in Ruby* es un conjunto de bibliotecas del lenguaje de programación Ruby que automatiza pruebas para los navegadores web [9]. Watir permite crear pruebas que son fáciles de leer y de mantener. A diferencia de otros entornos, con Watir solo se puede programar pruebas automáticas en Ruby, por ese motivo, la comunidad que usa Watir es muy pequeña.

- **TestComplete**: Es una plataforma de pruebas automatizadas funcional desarrollada. Ofrece la posibilidad creación de pruebas para navegadores webs, aplicaciones en Android e IOS. También permite grabar, reproducir las pruebas. Además, se pueden testear pruebas funcionales y *backend* como testear de las bases de datos [10].

Aparte de estas opciones, existen otras plataformas que ofrecen la posibilidad de crear *testing* automatizado. Sin embargo, las enumeradas anteriormente, se consideran que son las mejores candidatas para nuestro proyecto. Seguidamente, podemos observar en la tabla 1 una comparación resumida de las cuatro tecnologías explicadas anteriormente.

Framework	Navegadores	Lenguajes de programación	Sistema operativo	Grabación de las pruebas
Selenium	Cualquier navegador moderno	Java, C#, Ruby, Groovy, Python, Perl y Php	Windows, Web, Linux, Android, IOS	Sí
Katalon Studio	Cualquier navegador moderno	C#, Java, Python	Windows, Linux, Web, Android, IOS	Sí
Watir	Internet Explore, Google Chrome, Firefox y Safari	Ruby	Windows, Linux, Web, Android, IOS	No
TestComplete	Cualquier navegador moderno	JavaScript, Java, Pyhton, VBScript, JScript	Windows, Web, Linux, Android, IOS	SÍ

Tabla 1: Comparación de frameworks. Fuente: Elaboración propia

2.2. Justificación de la elección

Después de haber realizado el análisis de mercado, en este apartado se seleccionará la tecnología que se utilizará para el desarrollo del proyecto.

En primer lugar, todas las opciones vistas en el apartado anterior tienen características que dan un punto positivo para poder utilizarlas. Sin embargo, consideramos que Selenium es la mejor opción ya que otras plataformas como Katalan Studio están basadas en este. Por otro lado, ofrece escribir el código de las pruebas en diferentes lenguajes de programación. Así, podemos usar el lenguaje que mejor conocimiento tengamos.

En segundo lugar, como este proyecto se desarrolla con la colaboración de Everis y con el cliente Damm, en esta circunstancia la elección también se establece por la aceptación del cliente. Para este proyecto, el cliente Damm ha elegido usar Selenium con el lenguaje de programación Java ya dentro de sus proyectos de automatización tiene otro proyecto finalizado donde se ha usado Selenium. Por ese motivo, prefiere que en este proyecto también se utilice esta tecnología y la solución que se proponga utilice el *testing* automatizado con el navegador Google Chrome en Windows, Móvil y Tablet. Así, cada prueba tendrá que estar validada en las tres plataformas.

En conclusión, para el desarrollo del proyecto se utilizará Selenium con el lenguaje de programación Java ya que es una buena herramienta para crear *testing* automatizado. Además, es como el cliente prefiere que se realicen las pruebas.

3. Alcance

En este apartado se hablará de los objetivos principales y los requisitos del proyecto que se deben conseguir al finalizar el proyecto. También se explicarán los posibles riesgos que se pueden encontrar durante el desarrollo del proyecto.

3.1. Objetivos

A diferencia de otros proyectos, en este proyecto se trabajará sobre un módulo que está funcionando y disponible para los usuarios. Para este proyecto, los objetivos se concentran en el equipo de desarrolladores y responsables del mantenimiento. Sin embargo, se podría decir que si alcanzamos los objetivos del proyecto, uno de los beneficiarios serían los usuarios de las páginas de Damm porque a partir de ese momento muchos de los problemas se podrán detectar de forma rápida usando el *testing* automático.

El objetivo principal del proyecto es conseguir validar de forma eficiente diferentes procesos que proporcionan servicio en el módulo de Drupal, es decir, detectar rápidamente si hay algún bug o error en las diferentes funcionalidades que proporcionan las páginas de Damm y poder solucionarlo lo antes posible. Para cumplir este objetivo principal, a continuación, se han definido subobjetivos que se tienen que alcanzar al finalizar el proyecto:

- **El *testing* automatizado debe ser rápido de validar y finalizar**: Las pruebas automatizadas con Selenium deberían ser rápidas de ejecutar y no deberían tardar mucho tiempo en completarse. Además, el resultado debe crear un informe sobre las pruebas ejecutadas.

- **Mejorar el sistema de validación de procesos que causan problemas**: El nuevo sistema debe ayudar en detectar los errores o *bugs* que pueda haber en los procesos que actualmente son complicados de testear manualmente.

- **Debe ser fácil de modificar y ampliar**: El *testing* automatizado debe ser fácil de actualizar por si hay la necesidad de modificar porqué se ha actualizado alguna parte del módulo. Además, como estamos hablando de un módulo de 25 diferentes páginas webs, en este proyecto no incluimos el *testing* de todo el contenido de estas páginas, pero tampoco se descarta que en un futuro el cliente requiera añadir el *testing* automatizado para todos los servicios que tienen en el módulo de Drupal.

3.2. Requisitos no funcionales

Después de ver los objetivos del proyecto, en esta sección se verán los principales requisitos no funcionales que debe de satisfacer nuestro programa para un funcionamiento correcto:

- **Disponibilidad:** La herramienta de *testing* debe poderse usar en cualquier momento del día, especialmente en el horario laboral cuando trabaja el equipo de programadores del módulo de Drupal.

- **Eficiencia:** El sistema debe ser eficiente en término de tiempo ejecución y debe proporcionar un resultado válido. Además, hay que evitar al máximo el código duplicado.

- **Mantenibilidad:** Debe ser fácil actualizar, mejorar y/o ampliar el sistema de *testing* automatizado.

- **Seguro:** El sistema debe funcionar de forma segura sin que los diferentes entornos de Drupal se vean afectados por el *testing* automatizado.

3.3. Riesgos

En cualquier desarrollo de un proyecto se pueden encontrar algún riesgo u obstáculo. En este proyecto también se pueden identificar algunos posibles riesgos que podrían afectar en el desarrollo del proyecto. A continuación, se listan los riesgos identificados de este proyecto.

- **Inexperiencia en las tecnologías utilizadas**: Este es un riesgo que se prevé en muchos proyectos donde los desarrolladores no tienen experiencia en las tecnologías que usarán en el proyecto. Por tanto, requiere un periodo de aprendizaje para conocer e investigar sobre la tecnología. Esto afecta de tal manera que habrá algunas horas del proyecto que se tengan que invertir en el estudio técnico de la tecnología para poder utilizarlo de forma correcta durante el desarrollo del proyecto.

- **Desvíos del plan del proyecto**: Como este proyecto forma parte del trabajo final de grado, esto significa que el calendario está estrictamente cerrado, ya que el tiempo máximo para finalizar está establecido por la universidad. Por ese motivo, la fecha de entrega está fijada y no puede ser modificada. Es decir, si durante el proyecto tenemos algún desvío debemos intentar resolver lo antes posible para que la fecha de entrega final no se vea afectada.

- **Bloqueos**: Este proyecto se desarrollará en el entorno de pruebas del módulo en Drupal. Durante el *testing* automático se puede encontrar que el entorno de prueba se caiga y no se pueda continuar con el desarrollo hasta que el entorno de prueba vuelva a estar en funcionamiento. Durante este periodo, se debe intentar avanzar en otras partes del proyecto para evitar desvíos u otros obstáculos que puedan afectar al desarrollo del proyecto.

4. Metodología y rigor

En este apartado se explicará la metodología que se usará para el desarrollo del proyecto. Además, también se presentarán las herramientas que se utilizarán para la gestión del proyecto.

4.1. Metodología de Trabajo

Para el desarrollo del proyecto se utilizará la metodología *Agile*, concretamente *Scrum*, metodología que se usa mucho en el ámbito laboral. Por tanto, el proyecto se desarrollará de forma incremental y en iteraciones de dos o tres semanas. Para poder llevar a cabo este proyecto a desarrollar, necesitaremos un equipo con algunos roles imprescindibles:

- ***Product Owner***: Es la persona que se encarga de gestionar el *product backlog* y priorizar las diferentes tareas en sus determinados *sprints*.
- ***Scrum máster***: Se encarga de controlar que el trabajo siga la metodología scrum de forma correcta y ayuda a resolver problemas que podrían surgir durante el desarrollo.
- **Equipo de desarrollo**: Equipo de personas encargadas para implementar el producto.

Para este proyecto, hay a un *product owner* que gestionará el *product backlog*, un equipo formado por un programador a tiempo parcial y el autor del trabajo final de grado para poder desarrollar el proyecto. Por otra parte, no habrá estrictamente el rol de *scrum máster*. Sin embargo, como el *product owner* también tiene experiencia en este rol y ha trabajado en muchos proyectos como *scrum máster*, entonces, a veces este rol será interpretado por él para comprobar que para el desarrollo se está siguiendo de forma correcta la metodología *agile*.

Una vez presentado al equipo, el segundo paso muy importante es la comunicación entre el equipo para poder resolver dudas, problemas o algún obstáculo que podamos encontrar durante la creación del producto final. Para ello, usaremos las diferentes reuniones que existen en la metodología scrum:

- ***Scrum Daily***: Es una reunión diaria de una duración de 15 minutos en la que cada miembro del equipo comenta al resto qué tarea ha finalizado ayer, en qué trabajará hoy y si tiene problemas o bloqueos para poder continuar. En nuestro caso, se realizará la reunión cada día y será una reunión rápida a no ser que algún miembro tenga problema o bloqueo con alguna tarea.

- ***Sprint Planning***: Es la reunión que tendrá lugar al inicio de cada sprint para poder determinar qué tareas se tendrá que realizar en el sprint. En el equipo, se hará esta reunión el lunes cuando empiece el sprint.

- ***Sprint Review***: Es la reunión que se realizará al finalizar una iteración. El objetivo de esta reunión será revisar el sprint y comprobar el trabajo realizado por el equipo.

- ***Sprint Retrospective***: Es la reunión que se llevará a cabo al final de un sprint en el que se reflexionará sobre cómo se ha trabajado y cómo se podría mejorar para los siguientes *sprints*.

4.2. Herramientas de gestión de proyecto

Para poder seguir de forma correcta la metodología *Scrum* y el trabajo es muy importante usar algunas herramientas para tener una buena construcción del producto final. A continuación, se expondrán las herramientas que utilizaremos para la metodología y para el desarrollo del proyecto.

Para la gestión del *product backlog* y asignaciones de tareas en los *sprints* se hará uso del *ServiceNow* [11]. Esta herramienta es en la que trabaja actualmente Everis con el cliente Damm. Aquí es donde se crea el *product backlog* y los *sprints* de los diferentes proyectos que tiene Everis. Además, en *ServiceNow* también gestionan las incidencias que reporta el cliente. Para el proyecto habrá un *product backlog* en esta herramienta con las tareas asignadas en cada sprint.

Finalmente, para la gestión de versiones de git, se utilizará un repositorio de GitHub. Para un entorno seguro, el Gitflow se ha acordado que en primer lugar tendremos la rama máster y seguidamente la rama *develop*. Tanto en la rama *máster* como *develop* no se podrá hacer *commits* directamente. Para cada historia de usuario, se creará una nueva rama *feature* a partir de la rama *develop*. Una vez finalizada y validada la tarea, se bajará todo lo que haya en develop, por si ha habido cambios en la rama y a continuación se hará el *pull request* a la rama *develop*.

5. Descripción de las tareas

Las tareas del proyecto de automatización de *testing* se dividen en tres tipos: gestión del proyecto (GP), desarrollo (DV) y documentación (DC). Las tareas de gestión de proyecto y de documentación se concentran más a tareas de explicación del proyecto. En cambio, las tareas de desarrollo están relacionadas sobre la parte técnica, por ejemplo, las diferentes pruebas de automatización que se quiere implementar para las promociones del módulo de *Drupal*. Para estas tareas, hay que tener en cuenta que la fecha de inicio del proyecto es el 20 de septiembre 2021 y la fecha estimada de finalización es el 7 de enero de 2022, cerca de la fecha de entrega del trabajo final de grado. A continuación, en este apartado se presentan las tareas del desarrollo con una breve explicación, estimación de las horas para finalizar la tarea y si tiene alguna dependencia con otra tarea.

5.1. Gestión del proyecto

En esta sección, se explicarán las tareas principales que corresponden a la gestión del proyecto (GEP) que tiene lugar las primeras 4 semanas del curso.

- **GP1 - Contextualización y alcance**: Se redactará un documento introduciendo el proyecto, el problema que se quiere solucionar, analizando las diferentes soluciones y justificando la solución elegida. Además, también se define el alcance del proyecto y la metodología que se utilizará para el desarrollo del proyecto.
    - **Duración**: 25 horas
    - **Recursos Humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *Google Drive, Word*

- **GP2 - Planificación temporal**: Se elaborará un documento explicando la planificación del proyecto con las tareas a realizar, un diagrama de Gantt con las tareas. Finalmente, se explicará cómo se gestionarán los riegos identificados.
    - **Duración**: 15 horas
    - **Dependencias**: GP1
    - **Recursos Humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *Google Drive, Word*

- **GP3 - Gestión Económica y sostenibilidad**: Se creará otro documento con la estimación del coste y gestión del proyecto. También se elaborará un informe de sostenibilidad.
    - **Duración**: 15 horas
    - **Dependencias**: GP2
    - **Recursos Humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *Google Drive, Word*

- **GP4 - Integración del documento fina**l: Se integrarán los documentos anteriores en un solo documento final.
    - **Duración**: 20 horas
    - **Dependencias**: GP3
    - **Recursos Humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *Google Drive, Word*

- **GP5 – Reuniones de Seguimiento**: Se realizarán reuniones de seguimiento con el tutor, el ponente y reuniones tipo *daily scrum, sprint planning* y *retrospective* con el equipo de desarrollo durante todo el proyecto.
    - **Duración**: 92 horas
    - **Dependencias**: GP4
    - **Recursos humanos**: Responsable de proyecto y equipo de desarrollo
    - **Recursos materiales**: PC, *ServiceNow*, *Word* y *Teams*.

5.2. Desarrollo

Para la parte de desarrollo, las tareas descritas se agrupan en diferentes iteraciones que se realizarán durante la elaboración del proyecto.

5.2.1. Sprint 1

- **DV1 - Preparación del entorno de desarrollo**: Se tiene que configurar todo el entorno para poder empezar con el desarrollo. Descargar los programar y librerías necesarias para crear el proyecto.
    - **Duración**: 12 horas
    - **Dependencias**: GP4
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, *Github*, editor de texto, *Java*, *TestNG* y *Maven*.

- **DV2 – [*Frontend*] Aceptar cookies y restricción de edad:** Implementar la prueba de aceptar cookies y ser mayor de edad en las diferentes páginas del módulo en *Drupal*.
    - **Duración**: 20 horas
    - **Dependencias**: DV1
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, editor de texto, *Java*, *TestNG* y *Maven*.

- **DV3 - [*Frontend*] Comprobación de la información de las promociones existentes:** Se implementará la comprobación de visualización de las promociones existentes en la página web de forma automática.
    - **Duración**: 60 horas
    - **Dependencias**: DV2
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, editor de texto, Java, *TestNG* y *Maven*.

- **DC1 – Documentación *Sprint 1***: Redactar la documentación de las pruebas implementadas del *sprint 1*.
    - **Duración**: 10 horas
    - **Dependencias**: DV1, DV2, DV3
    - **Recursos humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *ServiceNow* y *Word*.

5.2.2. Sprint 2

- **DV4 – [*FrontEnd]* Visualización de la información de premios**: Implementación de pruebas automatizadas para visualizar la información de premios de las promociones desde el *frontend.*
    - **Duración**: 55 horas
    - **Dependencias**: Sprint 2
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, editor de texto, *Java*, *TestNG* y *Maven*.

- **DV5 – [*Frontend*] Visualización de sección e información para poder participar en una promoción**: Comprobar con pruebas automatizadas la visualización sobre la información legal y cómo participar en una promoción.
    - **Duración**: 35 horas
    - **Dependencias**: DV4
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, editor de texto, *Java*, *TestNG* y *Maven*.

- **DC2 – Documentación Sprint 2**: Redactar la documentación de las pruebas implementadas del *sprint 2.*
    - **Duración**: 10 horas
    - **Dependencias**: DV4, DV5
    - **Recursos humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *ServiceNow* y *Word*.

- **DC2 – Documentación Sprint 2**: Redactar la documentación de las pruebas implementadas del *sprint 2.*
    - **Duración**: 10 horas
    - **Dependencias**: DV4, DV5
    - **Recursos humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *ServiceNow* y *Word*.

5.2.3. Sprint 3

- **DV6 – [*Backend*] Creación de promociones**: Implementación de pruebas automatizadas para la creación de promociones desde el *backend*.
    - **Duración**: 60 horas
    - **Dependencias**: Sprint 2
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, editor de texto, *Java*, *TestNG* y *Maven*.

- **DV7 – [*Frontend*] Participar en una promoción**: Implementación de pruebas para comprobar que un usuario registrado puede participar en una promoción.
    - **Duración**: 40 horas
    - **Dependencias**: *Sprint 2*
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, editor de texto, *Java*, *TestNG* y *Maven*.

- **DC3 – Documentación *Sprint 3***: Redactar la documentación de las pruebas implementadas del *sprint 3.*
    - **Duración**: 10 horas
    - **Dependencias**: DV4, DV5
    - **Recursos humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *ServiceNow* y *Word*.

5.2.4. Sprint 4

- **DV8 – [*Backend*] Envío de correo de promociones:** Implementar las pruebas para los envíos de mensajes de las promociones a los usuarios participantes.
    - **Duración**: 40 horas
    - **Dependencias**: *Sprint 3*
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, editor de texto, *Java*, *TestNG* y *Maven*.

- **DV9 – [*Backend*] Verificación del envío del correo**: Implementar las pruebas para poder verificar que el usuario recibe el correo correctamente.
    - **Duración**: 20 horas
    - **Dependencias**: DV8
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, editor de texto, *Java*, *TestNG* y *Maven*.

- **DC4 - Documentación *Sprint 4***: Redactar la documentación de las pruebas implementadas del *sprint 4.*
    - **Duración**: 10 horas
    - **Dependencias**: DV8, DV9
    - **Recursos humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *ServiceNow* y *Word*.

5.2.5. Sprint 5

- **DV10 – Integración de las pruebas automatizadas**: Integrar todas las pruebas y prepararlas para poder ejecutarlas en *Jenkins*.
    - **Duración**: 20 horas
    - **Dependencias**: *Sprint 4*
    - **Recursos humanos**: Equipo de desarrollo
    - **Recursos materiales**: PC, editor de texto, *Java*, *TestNG*, *Jenkins* y *Maven*.

- **DC5 – Documentación Final:** Redactar y mejorar la memoria final del trabajo final de grado.
    - **Duración**: 20 horas
    - **Dependencias**: DV10
    - **Recursos humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *Word*.

- **DC6 – Preparación presentación final**: Elaborar un PowerPoint para la presentación del proyecto ante el tribunal.
    - **Duración**: 20 horas
    - **Dependencias**: DC5
    - **Recursos humanos**: Responsable de proyecto
    - **Recursos materiales**: PC, *PowerPoint*

6. Estimaciones y Diagrama de Grantt

6.1. Estimaciones

En este apartado, en la tabla 2 se puede observar las diferentes tareas listadas con su estimación en horas. Para esta estimación se ha tenido en cuenta la complejidad y las horas estimadas que se puede llegar en finalizar cada tarea. Sin embargo, hay que tener en cuenta que son horas estimadas y después pueden tener llegar a tener un tiempo de diferente dependiendo de la complejidad de cada prueba.

Código	Tarea	Horas	Dependencias	Recursos
GP	**Gestión de Proyecto**	**Total: 167**		
GP1	Contextualización y alcance	25		PC, Word
GP2	Planificación temporal	15	GP1	PC, Word
GP3	Gestión económica y sostenibilidad	15	GP2	PC, Word
GP4	Integración del documento final	20	GP3	PC, Word
GP5	Reuniones de seguimiento	92	GP4	PC, Word,Teams
DV	**Desarrollo**	**Total: 442**		
	Sprint 1	Total: 102		
DV1	Preparación del entorno de desarrollo	12	GP4	PC, GitHub, Java TestNG, Maven
DV2	Aceptar cookies y restricción de edad	20	DV1	PC, GitHub, Java TestNG, Maven
DV3	Comprobación de la información de promociones	60	DV2	PC, GitHub, Java TestNG, Maven
DC1	Documentación Sprint 1	10	DV1, DV2, DV3	PC, Word
	Sprint 2	Total: 100		
DV4	Visualización de la información de premios	55	Sprint 1	PC, GitHub, Java TestNG, Maven
DV5	Visualización sección para poder participar en una promoción	35	DV4	PC, GitHub, Java TestNG, Maven
DC2	Documentación Sprint 2	10	DV4, DV5	PC, Word
	Sprint 3	Total: 110		
DV6	Creación de promociones	60	Sprint 2	PC, GitHub, Java TestNG, Maven
DV7	Participar en una promoción	40	DV6	PC, GitHub, Java TestNG, Maven
DC3	Documentación Sprint 3	10	DV5, DV6	PC, Word
	Sprint 4	Total: 70		
DV8	Envío de correo de promociones	40	Sprint 3	PC, GitHub, Java TestNG, Maven
DV9	Verificación del envío del correo	20	DV8	PC, GitHub, Java TestNG, Maven
DC4	Documentación Sprint 4	10	DV8, DV9	PC. Word
	Sprint 5	Total: 60		
DV10	Integración de las pruebas automatizadas	20	Sprint 4	PC, GitHub, Java TestNG, Maven
DC5	Documentación Final	20	DV10	PC, Word
DC6	Preparación presentación final	20	DC5	PC, PowerPoint
	TOTAL	**609**		

Tabla 2: Estimación de las tareas. Fuente: Elaboración propia

6.2. Diagrama de Gantt

A continuación, en la figura 1 se puede ver el diagrama de Gantt del proyecto. Este diagrama contiene todas las tareas del apartado anterior. El diagrama está dividido en las iteraciones que se realizarán durante el desarrollo del proyecto.

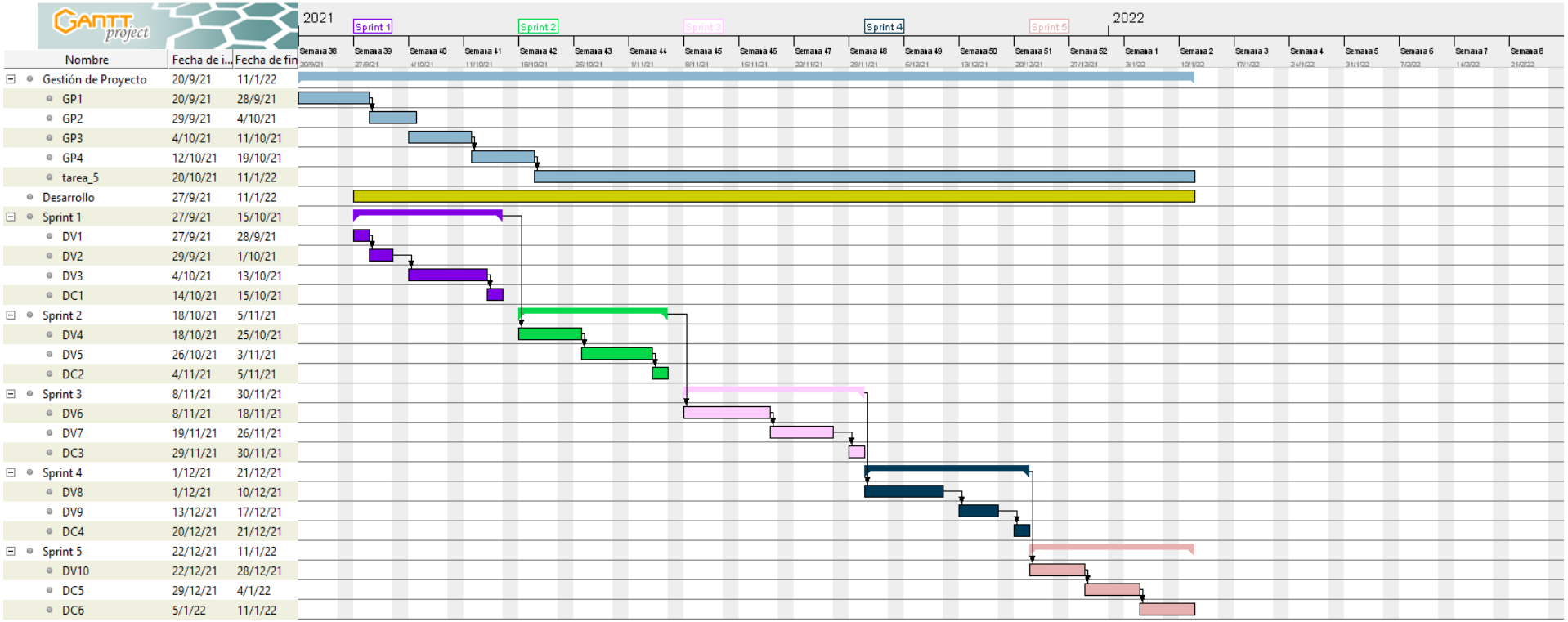

Figura 1: Diagrama de Gantt. Fuente: Elaboración propia.

7. Gestión de riesgos

Para la gestión de riesgos se realizarán algunas acciones o planes alternativos en el caso de que ocurran algunos de los posibles riesgos previstos durante el desarrollo del proyecto en el apartado Alcance.

7.1. Inexperiencia en las tecnologías utilizadas

Debido al avance y a las nuevas tecnologías, a la hora de comenzar un nuevo proyecto cada vez nos encontramos con este tipo de riesgo. En nuestro proyecto utilizaremos una tecnología para realizar pruebas automatizadas en el que no tenemos experiencia necesaria para evitar totalmente este riesgo. Para minimizar el impacto de este tipo de riesgo en nuestro proyecto se han sobrestimado en horas las tareas para poder tener tiempo de investigar la tecnología y así evitar que, por falta de experiencia, el proyecto no esté en riesgo. Por otra parte, en el equipo de desarrollo, el programador experto tiene alguna experiencia con esta tecnología. Para evitar este obstáculo, se podrá solicitar ayuda al compañero del equipo.

7.2. Desvíos de la planificación del proyecto

Desde inicio del proyecto sabemos que tenemos un calendario fijado, así como la entrega final del proyecto. Esto implica que cualquier desvío o problema que tengamos durante el proyecto podría afectar a la fecha de entrega y no poder llegar a cumplirla en plazo planificado. Para poder lidiar con las desviaciones se ha realizado una sobreestimación en horas de las tareas. Además, en caso de que estas sobreestimaciones no sean suficientes, en esta circunstancia también disponemos de cinco días de margen después de la finalización del proyecto. Es decir, antes de entregar la memoria al tribunal tenemos unos cinco días que se podrían utilizar para finalizar el proyecto y después entregarlo al tribunal y al cliente.

7.3. Bloqueos

Otro riesgo u obstáculo que podemos encontrarnos es que el entorno donde ejecutamos el código de las pruebas automatizadas esté caído. Como consecuencia podemos estar bloqueados y no podríamos avanzar. Este riesgo tiene una probabilidad bastante alta ya que en el entorno donde realizaremos el *testing* casi diariamente se está subiendo código del equipo de mantenimiento. Este equipo va resolviendo las incidencias que recibe del cliente. Para solventar este posible problema, el responsable de proyecto está trabajando para tener un entorno seguro donde no tengamos actualizaciones de código que podrían afectar a nuestro proyecto. Mientras tanto, nuestro proyecto realizará las pruebas en el entorno de prueba, hasta que el entorno seguro no esté preparado.

8. Presupuesto

8.1. Identificación y estimación de los costes

Para poder identificar y estimar correctamente los costes de nuestro proyecto, es importante tener en consideración algunos factores como recursos humanos y gastos generales. Además, a la hora de calcular el coste se debe tener presente los costes de contingencia y de los posibles imprevistos que se pueden encontrar durante el desarrollo del proyecto. A continuación, se calculará en cada sección el coste de los diferentes gastos y finalmente el coste estimado del proyecto.

8.1.1. Coste del personal por tarea

Para poder calcular el coste del personal por tarea, se necesita saber el salario por hora que cobra cada uno de los roles del proyecto. Para obtener este dato se ha consultado en una página web para poder tener la media del salario de cada rol. Por otro lado, se puede usar el salario por hora del autor de este trabajo que es de 9€ brutos la hora, salario mínimo establecido por la UPC para los alumnos de grado que realicen prácticas en una compañía mediante un convenio. A continuación, se puede observar el resultado de la consulta en la tabla 3. En esta tabla se puede ver el salario bruto por hora y el salario por hora añadiendo el 30% al salario bruto por hora, el importe que debe pagar la empresa a la seguridad social. Es importante tener en cuenta que la estimación del salario bruto por hora se ha obtenido de la página *Glassdor* [12].

Rol	Salario/hora (bruto)	Salario/hora (bruto) + Seguridad Social
Product Owner (PO)	21€	27.3€
Analista (AN)	14€	18.2€
Desarrollador especialista en *testing (DT)*	13€	16.9€
Desarrollador becario (DB)	9€	11.7€

Tabla 3: Sueldos por hora de cada uno de los roles del equipo. Fuente: Elaboración propia con los datos de Glassdor

Después de tener el salario por hora de cada rol, se puede calcular el coste del personal. Para ello, se va a indicar el número de horas que dedicará cada persona por tarea. De esta forma se podrá saber el coste de cada tarea. Finalmente, se puede calcular el coste total del personal. A continuación, en la tabla 4 se presenta el resultado final del coste del personal del equipo.

ID	TAREA	TOTAL HORAS	HORAS				COSTE (€)
			PO	AN	DT	DB	
GP1	Contextualización y alcance	25	0.5			24.5	460
GP2	Planificación temporal	15	0.5			14.5	184
GP3	Gestión económica y sostenibilidad	15	0.5			14.5	184
GP4	Integración del documento final	20	0.5			19.5	242
GP5	Reuniones de seguimiento	92	22	22	22	26	1677
DV1	Preparación del entorno de desarrollo	12			6	6	172
DV2	Aceptar cookies y restricción de edad	20		1	7	12	277
DV3	Comprobación de la información de promociones	60		2	28	30	861
DC1	Documentación *Sprint* 1	10				10	177
DV4	Visualización de la información de premios	55		1	25	29	780
DV5	Visualización sección para poder participar en una promoción	35		1	15	19	494
DC2	Documentación *Sprint* 2	10				10	177
DV6	Creación promociones	60		4	30	26	884
DV7	Participar en una promoción	40		2	20	18	585
DC3	Documentación *Sprint* 3	10				10	177
DV8	Envío de correo de promociones	40		2	18	20	575
DV9	Verificación del envío del correo	20		1	7	12	277
DC4	Documentación *Sprint* 4	10				10	177
DV10	Integración de las pruebas automatizadas	20			12	8	296
DC5	Documentación final	20				20	234
DC6	Preparación presentación final	20				20	234
TOTAL		**609**					**9124**

Tabla 4: Costes estimados de las tareas. Fuente: Elaboración propia

8.1.2. Costes generales

Los costes generales son aquellos que se calculan teniendo en cuenta el material y otros gastos que se puedan tener durante el desarrollo del proyecto. Para empezar, se utilizará un ordenador portátil corporativo ofrecido por la empresa. Como cada miembro del equipo tiene su propio ordenador portátil corporativo, se debe tener en cuenta este dato para el cálculo del material.

Para poder hacer los cálculos de costes generales explicados, se debe calcular la amortización con la siguiente fórmula:

$$\frac{Coste\ (euros)}{Vida\ útil\ (años) * 220\ (días\ laborales/año) * dedicación/día(h)} * duración\ proyecto\ (h)$$

Donde la vida útil es de 4 años y la dedicación/día, en nuestro caso, es de 7 horas.

RECURSO	COSTE (€) * Nº de personas del equipo	AMORTIZACIÓN (€)
Portátil Dell Latitude 5410	4072	393

Tabla 5: Costes materiales. Fuente: Elaboración propia

En la tabla 5 se puede ver el precio del portátil multiplicado por los miembros del equipo juntamente con el resultado de su amortización. Cabe mencionar que no habrá gastos respecto al *software* ya que se utilizará un *framework* totalmente gratuito para el desarrollo del proyecto.

Por otro lado, otros gastos que hay que añadir es la electricidad que se usa durante las horas de trabajo. También es importante e imprescindible tener acceso a Internet. Como cada miembro trabaja desde su lugar de vivienda, hacer esta estimación puede llegar a ser complicada ya que uno de los miembros del equipo trabaja desde fuera de Cataluña. Por tanto, para estimar este coste vamos a simplificar el cálculo suponiendo que trabajamos en una oficina de trabajo que incluye las necesitades principales para poder realizar la jornada laboral. Es decir, hay acceso a Internet, electricidad, agua, impresora, sala de reuniones, etc. [13]

Recurso	Precio/mes	Precio/mes incluido el IVA	Meses	Total
Oficina	300	363	4	1452

Tabla 6: Costes indirectos. Fuente: Elaboración propia

8.1.3. Costes de contingencia

En uno de los apartados anteriores se identificaron algunos riesgos que pueden afectar en el desarrollo del proyecto. Sin embargo, durante el proyecto puede haber otros imprevistos que no se han anticipado. Para estos tipos de imprevistos existe el coste de contingencia que consiste en añadir dinero en el presupuesto para los imprevistos no anticipados. De esta forma, se intenta que el proyecto no pase del presupuesto estimado. El coste de contingencia se suele calcular como un porcentaje y suele estar entre el 10 y el 20%. Para este proyecto se escoge el 15% de contingencia.

Tipo	Coste (€)	Contingencia (%)	Coste final (€)
Coste del personal	9124	15	10493
Coste material	393	15	452
Coste indirecto	1452	15	1670

Tabla 7: Costes de contingencia. Fuente: Elaboración propia

8.1.4. Costes de imprevistos

En este apartado se va a estimar los riesgos identificados para ver cómo pueden impactar en la parte económica del proyecto. Por tanto, en relación con la probabilidad vamos a añadir el coste a las horas que necesitemos para solucionar los imprevistos durante el proyecto.

Riesgo	Probabilidad	Tiempo (h)	Precio/hora	Coste (€)
Inexperiencia en las tecnologías	40%	40	17	272
Desvíos de la planificación	40%	25	17	170
Bloqueos	66%	35	17	393
TOTAL				**835**

Tabla 8: Costes de imprevistos. Fuente: Elaboración propia

En caso de que durante el desarrollo del proyecto no se produzca ningún riesgo o imprevisto, el coste de imprevistos y contingencias se podrá utilizarlo para mejorar el entorno o para utilizar en algún otro proyecto para automatizar el resto del módulo del *Drupal.*

8.1.5. Estimación del presupuesto final

Después de calcular los diferentes gastos, en la tabla 9 se calcula el presupuesto final estimado.

Tipo	Coste con contingencias (€)
Coste personal	10493
Coste material	452
Coste indirecto	1670
Costes imprevistos	835
Total con imprevistos	**13450**

Tabla 9: Presupuesto final estimado. Fuente: Elaboración propia

8.2. Control de gestión

Para tener un control de las desviaciones que se pueda tener en el presupuesto, se va a utilizar unos mecanismos para tener siempre actualizado el presupuesto gastado. Estos mecanismos ayudarán a detectar las desviaciones que se encuentren durante el proyecto.

- **Desviación coste hora por tarea:**
  (horas estimadas – horas reales) * coste real
- **Desviación de los costes personales por tarea**
  (coste estimado – coste real) * horas reales
- **Desviación total costes personal por tarea**
  (coste total personal por tarea estimado – coste total personal por tarea real)
- **Desviación total costes generales (material + indirecto)**
  (coste general estimado – coste general real)
- **Desviación total costes imprevistos**
  (coste imprevisto estimado – coste imprevisto real)
- **Desviación total de horas**
  (horas totales estimadas – horas totales reales)
- **Desviación total de costos:**
  (Coste total estimado – coste total real)

9. Análisis de requisitos

Un requisito es una condición que tiene que cumplir el software a desarrollar. Este apartado tiene como objetivo definir los requisitos funcionales y no funcionales que debe satisfacer el software. Cabe destacar que los requisitos a definir se concentran en la comprobación de forma automatizada de las diferentes partes del módulo de promociones.

9.1. Requisitos funcionales

Los requisitos funcionales definen las funciones que el software debe tener. Es decir, se especifica qué hará el software a desarrollar. A continuación, se definen brevemente los requisitos funcionales del sistema.

Los requisitos funcionales requeridos por el cliente para este proyecto son principalmente dos: comprobar la visualización de la información de las promociones y crear promociones de forma automatizada.

Para la comprobación de la información de las promociones se ha de verificar que se puede visualizar tanto la información de las promociones como todo el resto de los elementos que tiene las páginas webs. Es decir, visualizar las bases legales de una promoción, la sección de participar en la promoción y en alguna página web la visualización de los restaurantes.

Por otra parte, para la creación de promociones el sistema ha de permitir crear promociones de forma automatizada. Una vez creada una promoción con el *testing* automático se ha poder visualizar la promoción creada para verificar la creación correcta.

9.2. Requisitos no funcionales

Los requisitos no funcionales o requisitos de calidad definen las propiedades de calidad que debe cumplir el software. A continuación, se describen los requisitos no funcionales que el sistema debería incluir.

- **Disponibilidad:** La herramienta de *testing* debe poderse usar en cualquier momento del día, especialmente en el horario laboral cuando trabaja el equipo de programadores del módulo de Drupal.

- **Eficiencia:** El sistema debe ser eficiente en término de tiempo ejecución y debe proporcionar un resultado válido. Además, hay que evitar al máximo el código duplicado. Por otro lado, las pruebas han de poderse ejecutar en paralelo para poder tener un mejor rendimiento.

- **Mantenibilidad:** Debe ser fácil actualizar, mejorar y/o ampliar el sistema de *testing* automatizado. Como el proyecto de automatización se ha iniciado con las promociones del módulo, en el futuro es posible que se amplie el proyecto.

- **Seguro:** El sistema debe funcionar de forma segura sin que los diferentes entornos de Drupal se vean afectados por el *testing* automatizado.

10. Especificación del sistema a testear

El módulo a testear está desarrollado con Drupal y con lenguaje de programación PHP. Actualmente, este módulo contiene aproximadamente 25 *landing pages*. Cada una de las páginas tiene un apartado para las promociones. Además, algunas páginas webs también tienen otras secciones que tratan de diferentes temas, como, por ejemplo, cervezas, eventos, proyectos y un apartado con la explicación del origen de la cerveza principal de la página web. Para el proyecto de *testing automatizado*, se va a enfocar únicamente en la parte de promociones.

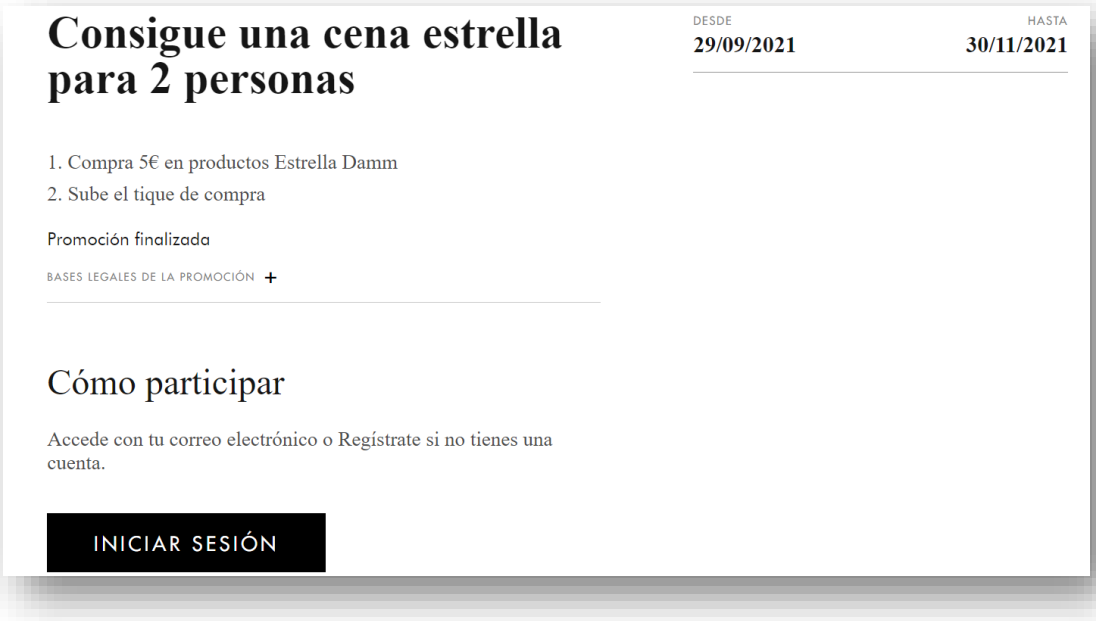

Figura 2: Visualización de una promoción en producción

Todas las promociones del módulo tienen un título descriptivo, una fecha de validez e instrucciones para poder participar. También tienen unas bases legales en las que se definen los términos y condiciones legales de la promoción. En la figura 2 se puede observar la información de una promoción, en este caso se trata de una cena estrella para dos personas. Una vez presentada la promoción en las páginas webs se puede encontrar un apartado que indica como se puede participar en la promoción. Finalmente, en algunas páginas hay una sección en la que se indica los diferentes tipos de premios que un participante puede ganar, tal como de se presenta en la siguiente figura 3.

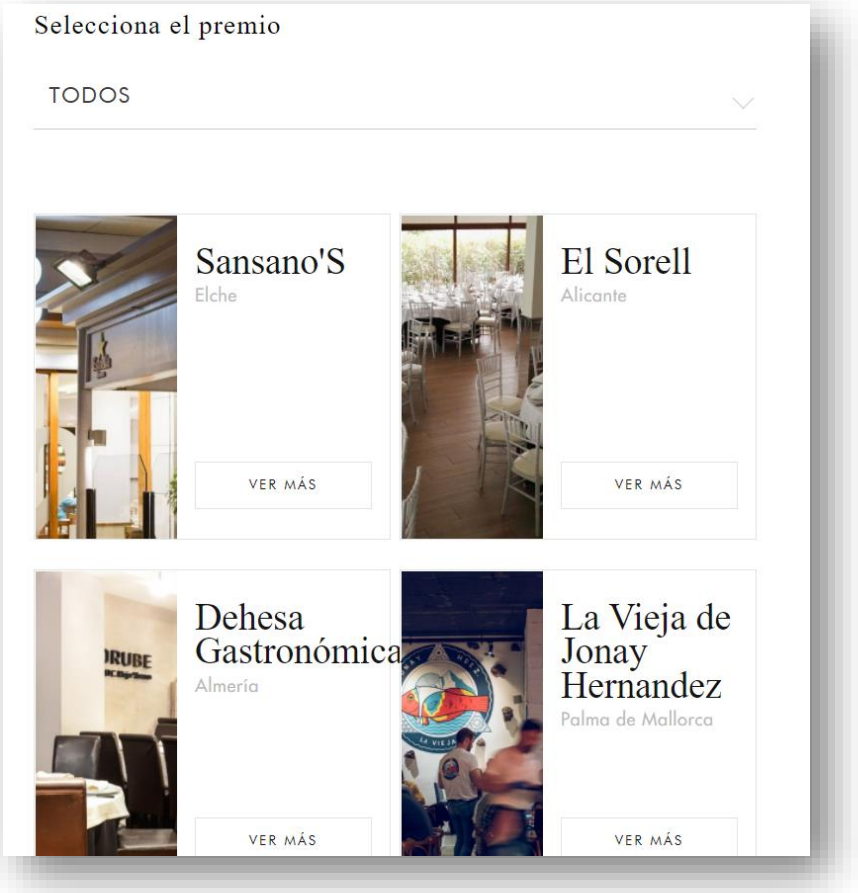

Figura 3: Visualización de la sección de premios de una promoción

Para poder crear una promoción se necesita conectar a un entorno de pruebas. Una vez en este entorno podemos acceder al formulario para poder crear una promoción. Desde el formulario podemos crear cinco tipos de promociones:

- **Concurso con selección de premio**: Promoción en la que el participante puede escoger el premio que desea dentro de un conjunto de regalos.
- **Concurso con comprobante de compra**: Promoción en la que se debe enviar un comprante de compra para poder participar.
- **Descuento con comprobante de compra y selección de premio**: Es una promoción en la que se combinan las dos anteriores promociones.
- **Reembolso de importe**: Promoción en la que se devuelve el dinero al ganador.
- **Momento ganador**: Promociones que se llevan a cabo en un periodo corto de tiempo en la que si un usuario participa puede ganar el premio.

A la hora de crear se ha indicar el tipo de promoción. Después como datos obligatorios hay que indicar el título, dos imágenes de la promoción, descripción, texto para el listado de promoción, fecha de inicio y finalización, mensaje de confirmación para el usuario que desea participar y bases legales de la promoción. Finalmente, existe la opción para enviar un correo automático al usuario una vez que se cree la promoción y así también para la persona que apunta en una promoción. Para el envío de correo, se ha de crear un correo con toda la información, para poder usarlo más tarde. La creación del correo también se realiza con un formulario en el que el principalmente se informa del título del

correo, asunto, tipo de correo y mensaje. De esta información, podemos crear un correo y asociarlo a una promoción para el envío automática de mensajes al correo.

10.1. Diagrama de casos de uso a testear

En este apartado, se va a presentar los casos de uso que se va a tener que testear de forma automatizada. Cabe destacar que estos casos de uso ya están implementados y funcionando en producción. El objetivo de este apartado es describir los casos de uso para poder entender qué tipo de pruebas se necesitará implementar.

A continuación, en la Figura 4 podemos observar los casos de usos. Estos están divididos en dos tipos de usuarios, un usuario administrador y un usuario registrado en el sistema para poder utilizar las funcionalidades que ofrece la página web. Este diagrama se creado usando la herramienta drawio [14].

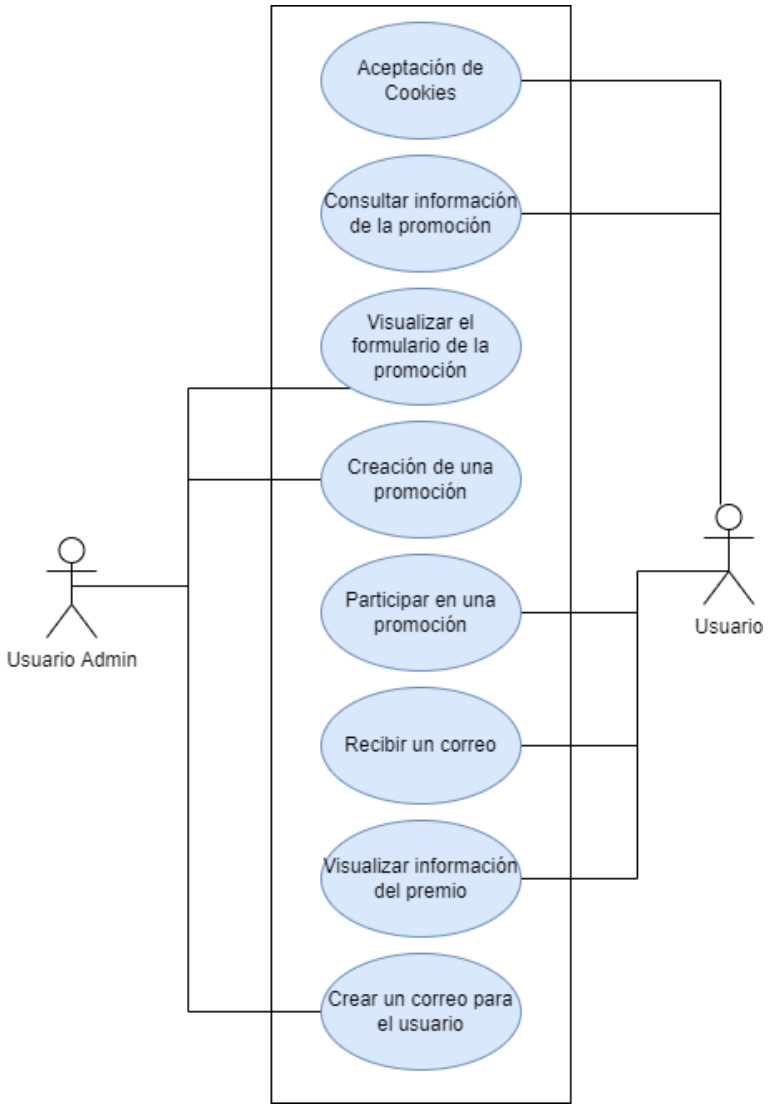

Figura 4. Diagrama de casos de uso. Fuente: Elaboración Propia

A continuación, se especifican los casos de uso presentados en el diagrama de la figura 4.

- **Aceptación de cookies:**

Caso de uso	Aceptación de cookies	**Actor principal**	Usuario
Precondición			
Disparador	El usuario quiere acceder a una página web del módulo.		
Escenario principal de éxito			
1. El usuario accede a una página web del módulo de promociones. 2. El sistema solicitar la aceptación de cookies. 3. El usuario aceptar las cookies pulsando al botón “Aceptar y seguir navegando”. 4. El sistema informe que el usuario solo siga navegando si tiene más de 18 años. 5. El usuario es mayor de edad y pulsa en el botón “Entrar”. 6. El usuario puede visualizar el contenido de la página web.			
Extensiones			
3.a. El usuario decide configurar las cookies. 3.a.1. El sistema muestra todas las cookies aceptar. 3.a.2. El usuario selecciona las cookies que desea aceptar. 3.a.3. El usuario pulsa el botón a ‘guardar mi configuración’.			

Tabla 10: Caso de uso aceptación de cookies. Fuente: Elaboración Propia

- **Consultar información de la promoción:**

Caso de uso	Consultar información de la promoción	**Actor principal**	Usuario
Precondición	El usuario ha aceptado las cookies y la restricción de edad.		
Disparador	El usuario quiere visualizar una promoción.		
Escenario principal de éxito			
1. El usuario accede a una página web del módulo. 2. El usuario acepta las cookies y la restricción de edad. 3. El sistema muestra la promoción activa. 4. El usuario visualiza toda información de la promoción (nombre, instrucciones, bases legales, cómo participar en la promoción, fotos de la promoción, etc.).			

Tabla 11: Caso de uso Consultar información de la promoción. Fuente: Elaboración Propia

- **Visualizar formulario de la promoción:**

Caso de uso	Visualizar formulario de creación de la promoción	**Actor principal**	Usuario *Admin*
Precondición	El usuario administrador tiene la sesión iniciada en el entorno de pruebas.		
Disparador	El usuario quiere visualizar el formulario de crear una promoción		
Escenario principal de éxito			
1. El usuario desde el menú selecciona “contenido” -> ”añadir contenido” -> “Promoción Damm”. 2. El sistema muestra todo el formulario de crear una promoción.			

Tabla 12: Caso de uso visualizar formulario de promoción. Fuente: Elaboración Propia

- **Creación de una promoción:**

Caso de uso	Creación de una promoción	**Actor principal**	Usuario *Admin*
Precondición	El usuario administrador tiene la sesión iniciada en el entorno de pruebas.		
Disparador	El usuario quiere crear una promoción		
Escenario principal de éxito			
1. El usuario desde el menú selecciona “contenido” -> ”añadir contenido” -> “Promoción Damm”. 2. El sistema muestra todo el formulario de crear una promoción. 3. El usuario completa el formulario con la información de la promoción. 4. El usuario valida la información. 5. El sistema acepta la información introducida. 6. El sistema crea la promoción.			
Extensiones			
5.a. El sistema detecta algún campo incompleto o incorrecto. 5.a.1. El usuario corrige el campo. 5.a.2. Se repite el punto 4.			

Tabla 13: Caso de uso Creación de una promoción. Fuente: Elaboración Propia

- **Participar en una promoción:**

Caso de uso	Participar en una promoción	**Actor principal**	Usuario
Precondición	El usuario tiene la sesión iniciada en la página web.		
Disparador	El usuario quiere participar en una promoción.		
Escenario principal de éxito			
1. El usuario accede al apartado de la promoción. 2. El sistema muestra la promoción vigente. 3. El usuario comprueba que cumple los requisitos para participar. 4. El usuario solicita participar en la promoción. 5. El sistema muestra un mensaje informado que su solicitud ha sido enviada.			
Extensiones			
3.a. El usuario no cumple con los requisitos, por tanto, no puede participar.			

Tabla 14: Caso de uso Participar en una promoción. Fuente: Elaboración Propia

- **Recibir un correo:**

Caso de uso	Recibir un correo	**Actor principal**	Usuario
Precondición	El usuario ha solicitado o esta participando en una promoción.		
Disparador	El usuario ha solicitado o ha sido aceptado en una promoción.		
Escenario principal de éxito			
1. El usuario administrador verifica la solicitud del usuario. 2. El usuario administrador aprueba la participación del usuario. 3. El usuario recibe un correo sobre el estado de su participación.			
Extensiones			
1.a. El usuario administrador actualiza el estado de una promoción. 1.a.1. El sistema envía un correo al usuario participante de la promoción actualizada. 1.a.2. Se ejecuta el punto 3.			

Tabla 15: Caso de uso Recibir un correo. Fuente: Elaboración Propia

- **Visualizar la información del premio:**

Caso de uso	Visualizar la información del premio	**Actor principal**	Usuario
Precondición			
Disparador	El usuario quiere mirar la sección de premio de una promoción.		
Escenario principal de éxito			
1. El usuario accede a una página web del módulo. 2. El usuario acepta las cookies y la restricción de edad. 3. El sistema muestra la promoción activa. 4. El usuario visualiza toda la información en la sección de premios de la promoción.			

Tabla 16: Caso de uso visualizar información del premio. Fuente: Elaboración Propia

- **Crear un correo:**

Caso de uso	Crear un correo	**Actor principal**	Usuario *Admin*
Precondición	El usuario administrador tiene la sesión iniciada en el entorno de pruebas.		
Disparador	El usuario administrador quiere crear un correo		
Escenario principal de éxito			
1. El usuario administrador desde el navegador accede al formulario de crear un correo. 2. El sistema muestra el formulario. 3. El usuario administrador completa el formulario introduciendo los datos (titulo, asunto, encabezado, saludos, etc.). 4. El usuario administrador valida y guarda el correo. 5. El sistema confirma que se ha creado un correo automatizado.			
Extensiones			
4.a. El sistema informa que algún dato es incorrecto. 4.a.1. El usuario administrador corrige los campos incorrectos. 4.a.2. Se repite el punto 4.			

Tabla 17: Caso de uso crear un correo para una promoción. Fuente: Elaboración Propia

10.2. Diagrama de clases

En este apartado se presenta un diagrama de clases para poder entender la gestión de información que hace el módulo de promociones. En figura 5 podemos observar el diagrama de clases. Este diagrama se ha elaborado para entender y representar las entidades importantes y necesarias que tiene el módulo de promociones. El siguiente diagrama no representa las clases que tiene la implementación del módulo, sino la estructura que tienen las promociones. De esta forma se puede entender los datos obligatorios que se necesitan para poder realizar el *testing automatizado*.

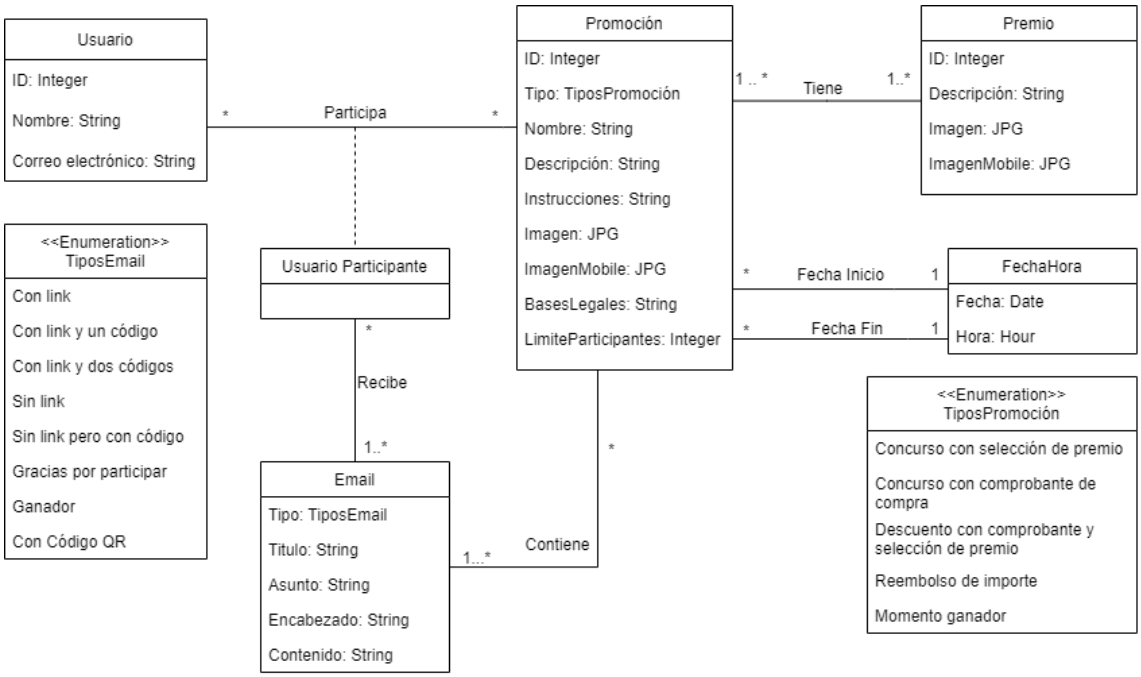

Figura 5. Diagrama de clases. Fuente: Elaboración Propia

Restricciones textuales:

1. Clave externa: (Usuario, ID), (Promoción, ID), (Premio, ID), (FechaHora, Fecha+Hora), (Email, Título)

2. El número de participantes de una promoción no puede superar al número *LimiteParticipantes* de la promoción.

3. Un usuario participante de una promoción puede ser eliminado, si no cumple las bases legales de la promoción.

4. La fecha fin de una promoción tiene que ser posterior a la fecha de inicio.

En el diagrama podemos ver que una promoción se identifica por un identificador (ID) y es siempre de un tipo del listado de *enumeration* de *TiposPromoción*. También tiene otros atributos como nombre, descripción, imágenes de la promoción y *LimiteParticipantes*, un número que se registra como el número máximo de personas que pueden participar en la promoción. Además, cada promoción tiene sus bases legales y normativa que debe cumplir un concursante para poder participar en una promoción. Cada promoción tiene un período de validez para que los cursantes puedan

participar, una vez finalizada la promoción ningún usuario puede solicitar la participación en ella.

Por otro lado, una promoción tiene uno o más premios para los ganadores. Cuando se crea una promoción, se añaden correos automatizados para los diferentes estados de la promoción. Por ejemplo, cuando un usuario solicita participar en una promoción y es aceptado, entonces, el sistema envía un correo informado que ha sido aceptado en la promoción. En caso contrario, también recibe un correo. De esta misma forma se configura el envío de correo automatizado para los usuarios participantes de la promoción.

Finalmente, tenemos una pequeña sección en la que se definen las restricciones que no pueden expresar gráficamente en el esquema conceptual.

Una vez visto el diagrama de clases del módulo de promociones, a continuación, se va a presentar el diagrama de clases del nuestro sistema de pruebas automatizadas. En la figura 6 se puede observar el diagrama de clases. Como se puede notar es un diagrama muy simple y hay muy pocas entidades representadas con algunas clases sin atributos. Esto es así por algunos motivos que se van a explicar a continuación.

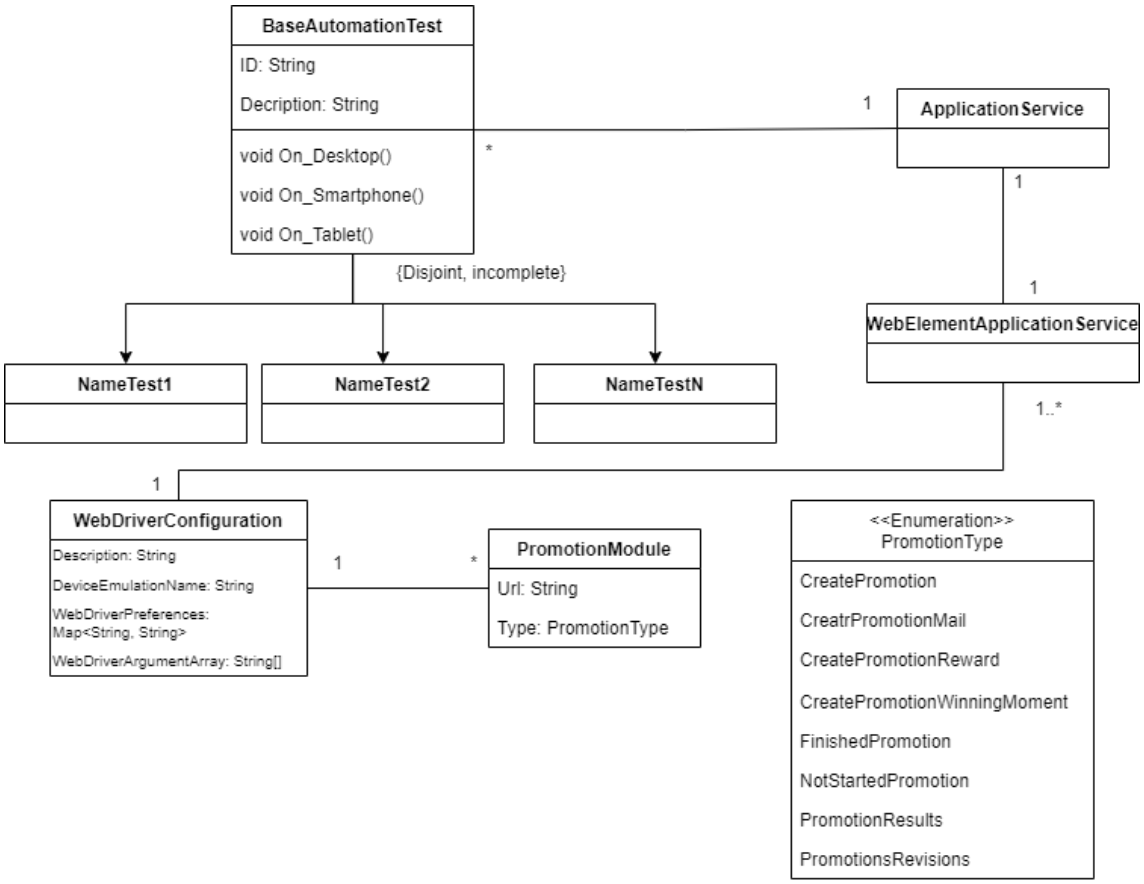

Figura 6: Diagrama de Clases del sistema de Testing Automatizado. Fuente: Elaboración Propia

Primeramente, cabe destacar que las pruebas a automatizar tienen que ver con los elementos webs de las páginas webs, concretamente con los datos del código HTML. Por ese motivo, estas pruebas no necesitan la creación de objetivos como Promoción o Usuario ya que toda esta información se escribe o se verifica directamente en los

elementos webs. Sin embargo, lo que si se crea es la configuración de página web donde se realizan las pruebas.

En el diagrama de la figura 6, hay la clase *BaseAutomateTest* que es abstracte y sirve para controlar los elementos y los métodos que debe tener cada una de las pruebas a automatizar. Por ese motivo, todos los tests son subclases de la clase *BaseAutomateTest*, de esta forma se asegura que cada clase de la prueba contenga los tres métodos para poderlos ejecutar en los tres dispositivos.

Por otra parte, la implementación de validación de las pruebas se realiza en la clase *ApplicationService* que envía el resultado a la prueba para que se verifique si la prueba se ha pasado con éxito. La clase *ApplicationService* para poder comprobar el resultado hace la llamada a la clase *WebElementApplicationService* que se encarga de tratar de las estructuras de datos que se muestran en la página web que se conocen como *WebElement*. Además, esta clase antes de obtener los elementos webs realiza la llamada a la clase *WebDriverConfiguration* pasando todos los parámetros para que inicie el navegador de Google Chrome e ir navegue a la página web en la que se desea realizar la prueba.

Cabe destacar que las clases *ApplicationService* y *WebElementApplicationService* se han separado en diferentes clases de diferentes páginas webs. Es decir, la página web de comprobar elementos de promociones tiene una *ApplicationService* y *WebElementApplicationService* y así se separará para otras páginas webs. Con esto se pretende que no haya mucho código en una misma clase. Si está separado por páginas, es mucho más fácil de encontrar.

11. Diseño

Una vez visto el análisis de requisitos y la especificación del sistema a testear, en este apartado se explicará la arquitectura de software y los patrones de diseño que se han decidido implementar para la automatización de *testing*.

11.1. Arquitectura

La arquitectura del software es la organización de los componentes en paquetes y capas que separan lógicamente la funcionalidad del sistema.

En el proyecto de automatización de *testing* se desarrollan pruebas automatizadas utilizado la arquitectura *Domain Driven Desing* (DDD) [15] [16]. El DDD es una aproximación al diseño de software que pone en el centro el dominio. Ayuda a definir exhaustivamente el dominio y nos permite conocer en detalle el problema de manera que se pueda dividir en subdominios. Con esto se pretende tener el mínimo acoplamiento entre el Dominio y el resto de las capas del sistema y así lograr soluciones que faciliten mantenibilidad y adaptabilidad del sistema.

Este proyecto depende de un sistema que está en funcionamiento y se actualiza para mejorar la experiencia del usuario. Como el proyecto está sujeto a un sistema que puede tener muchos cambios y evoluciones en el futuro, se ha considerado para la arquitectura el *Domain Driven Desing* es una buena opción ya que permite disminuir la cantidad de tiempo que se necesita para actualizar la implementación de las pruebas. De esta forma el sistema a construir sería mantenible y ampliable.

A continuación, se explica cada una de las capas que se va a usar en el sistema:

- **Capa de Aplicación**: Es la capa que contiene los servicios que se conectan a nuestro dominio. En nuestro proyecto, contiene interfaces, servicios de *Selenium* y toda la lógica que tiene que ver con las pruebas automatizadas que necesitan las librerías de *Selenium*.
- **Capa de Dominio o Negocio**: Es la capa que define el comportamiento del sistema y está formada por clases y funciones. En la capa de dominio se encuentran las entidades que representan nuestro sistema para poder la creación de pruebas automatizadas.
- **Capa de Infraestructura**: Es la capa que permite interactuar a un sistema de software con otro sistema externo al recibir, almacenar información cuando se solicitan. La capa de infraestructura gestiona las interacciones del sistema con *Selenium* para realizar la configuración y después proporcionar la información necesaria para ejecutar las pruebas automatizadas.

Todo este proceso depende de cómo esta implementado el código. Para tener una buena arquitectura que se ajuste a nuestras necesitadas, en el diseño se va a tener presente algunos principios conocidos en la ingeniería del software como *Principios SOLID* [17].

El acrónimo *SOLID* hace referencia a:

- **Principio de Responsabilidad Única (*Single Resposability Principle*)**: Se establece que una clase debe tener una única responsabilidad.
- **Principio Abierto Cerrado (*Open Close Principle*)**: Este principio establece que los componentes de software deben estar abiertos para extender, pero cerrados para la modificación. Es decir, se debe ser capaz de extender el comportamiento de las clases sin necesidad de modificar su código.
- **Principio de Sustitución de Liskov (*Liskov Substitution Priniciple*):** Este principio establece que una clase puede ser reemplazable por todas sus subclases sin alterar el funcionamiento de la clase. Es decir, si en alguna parte de código se usa una clase y esta tiene subtipos, entonces se debe poder utilizar cualquiera de las subclases y el programa debería seguir funcionando correctamente.
- **Principio de Segregación de Interfaces (*Interface Segregation Principle*):** Establece que muchas interfaces específicas son mejores que una interfaz de propósito general. Esto es, cada clase debe implementar las interfaces que se va a utilizar.
- **Principio de Inversión de Dependencias (*Dependecy Inversion Principle*):** Establece que las clases de alto nivel no deberían dependen de las clases de bajo nivel. En ambos casos deben depender de las abstracciones.

11.2. Patrones de diseño

Para tener una buena arquitectura e implementación se hará uso de algunos patrones de diseño. En este apartado se explicarán los patrones de diseño que se van a utilizar en el sistema.

11.2.1. Patrón Adaptador (*Wrapper*)

El patrón adaptador permite que dos clases que con diferentes interfaces puedan comunicar a partir de la creación de un objeto que realiza la comunicación entre las dos clases. De esta forma, adapta la información de la primera clase para que la segunda clase pueda entenderla.

En la elaboración de las pruebas se hace uso del *framework* Selenium. Este framework tiene un *webdriver* que permite conectar y controlar un navegador web para poder ejecutar una prueba. El controlador de *Selenium* tiene muchos componentes y funcionalidades. En vez de tener realizar llamadas a los métodos del *webdriver* de *Selenium*, se ha decidido que se creará un controlador web propio en el sistema. Es decir, se creará una clase que adapte los métodos y componentes del *webdriver* de *Selenium,* de manera que, desde cualquier test, en vez de hacer la llamada al webdriver de *Selenium* se hará al *webdriver* adaptador.

El hecho de hacer uso de un controlador adaptado permite tener un campo abierto para poder realizar las pruebas a nuestro gusto y no depender del *webdriver* de *Selenium*. Sin embargo, siempre que queramos utilizar un nuevo método del *webdriver* se tendrá que actualizar el *webdriver* adaptador.

11.2.2. Patrón *Singleton*

El patrón *singleton* es uno de patrones de diseño que existen en la ingeniería de software. Permite restringir la creación de objetos a una clase y solo tiene una única instancia. De esta forma este patrón de diseño garantiza que la clase *singleton* solo tendrá una única instancia en la que se tendrá acceso globalmente.

Como se ha explicado en el apartado anterior, en el sistema se creará una webdriver adaptado para nuestras pruebas. Como este controlador debe ser único y no puede haber más. Para garantizar esta restricción se utilizará el patrón *singleton* para la clase que se encarga de crear y adaptar el *webdriver* de *Selenium*. Con este patrón se controlará que durante la ejecución de las pruebas solo hay un único webdriver que pueda hacer las llamadas necesarias al controlador de Selenium.

12. Implementación

Una vez visto la especificación y el diseño arquitectónica para las pruebas, este apartado tiene como objetivo explicar los aspectos relevantes e importantes sobre la implementación de las pruebas automatizadas. También se presentarán las tecnologías utilizadas para el desarrollo del proyecto.

12.1. Elaboración de las pruebas

Para la creación de pruebas automatizadas se codificarán pruebas unitarias que compruebe el funcionamiento correcto de una parte específica del módulo de promociones. Así, se podrá gestionar rápidamente algún error inesperado en las páginas webs del módulo de promociones.

Para poder realizar y tener bien estructurada cada prueba, se utilizará el patrón *AAA (Arrange, Act, Assert)* [18]. Este patrón tiene como objetivo dividir en tres partes una prueba unitaria: organizar (*arrange*), actuar (*act*) y verificar (*assert*). Estructurar las pruebas de esta forma, facilita mucho entender cada prueba. Sin duda, esto es un beneficio para el proyecto ya que se prevé que en el futuro estas pruebas se tengan que mantener o actualizar. Gracias a este patrón, las pruebas automatizadas tendrán una buena organización que permitirán comprender y actualizar un test de forma rápida.

Las diferentes partes del patrón AAA tiene una función concreta [19]:

- ***Arrange***: Es donde se inicializa los objetos, se preparan los datos y las dependencias necesarias para poder ejecutar la prueba.
- ***Act***: En esta parte se realiza la ejecución de la prueba y se obtiene el resultado para poder verificarlo. Es decir, en este estado tenemos todo el código de la prueba que deseamos comprobar.
- ***Assert***: En esta última parte se realiza la comprobación del resultado obtenido con el esperado. Si el resultado final es igual que el resultado esperado, entonces, la prueba ha pasado con éxito. En caso contrario, la prueba no ha pasado.

Un ejemplo de prueba automatizado usando el patrón *AAA* sería el de la figura 7. En esta imagen podemos identificar claramente las tres partes de una prueba simple que consiste en realizar la aceptación de cookies de una página web. En el *arrange* se prepara la aplicación del servicio, a continuación, en el *act* se realiza la ejecución de la prueba y se obtiene un resultado de tipo booleano. Finalmente, en el *assert* se hace la comprobación de que el resultado obtenido en el *act* era el esperado. Así, se finaliza una prueba.

```java
@Test()
public void on_Desktop() {
    // ARRANGE
    INotStartedPromotionApplicationService applicationService = Container
            .resolve(INotStartedPromotionApplicationService.class);

    // ACT
    Boolean result = applicationService.appearsCookiesAcceptance_Desktop();

    // ASSERT
    Assert.assertTrue(result);
}
```

Figura 7: Ejemplo de Patrón AAA

Para poder entender con profundidad la fase *act* del patrón vamos a seguir el ejemplo anterior y visualizar con detalle cómo se está realizando la prueba de aceptación cookies. En la figura 8 podemos observar la implementación del método *appearsCookiesAcceptance_Desktop()*. En esta función se puede observar una estructura de organización: en primer lugar, se inicia el navegador y se obtiene el elemento web necesario para la prueba, se comprueba que es visible en la página web y finalmente se cierra el navegador. Esta estructura se intenta utilizar en todas las pruebas para que la comprensión de la implementación sea fácil y mantenible de forma eficiente.

```java
public Boolean appearsCookiesAcceptance_Desktop() {
    try {
        Optional<WebElement> optionalWebElement = _cookiesAcceptanceWebElementApplicationService
                .getCookiesAcceptanceWebElement_Desktop();
        Boolean result = optionalWebElement.isPresent();
        if (result == true) {
            _webElementApplicationService.disposeWebDriverByWebElement(optionalWebElement.get());
        }
        return result;
    } catch (Exception exception) {
        exception.printStackTrace();
        return false;
    }
}
```

Figura 8: Ejemplo de estructura de implementación

Como se ha explicado anteriormente, para la automatización de las pruebas se necesita el uso de los elementos de una página web, conocidos como *WebElements*. Estos representan datos que están representados usando el *Hyper Text Markup Language (HTML)*. Usando la información que tiene la página en el *Document Object Model (DOM)* se tiene los elementos webs. Para poder obtener y trabajar con estos elementos se ha creado una aplicación de servicio de elementos webs donde se obtienen y se realiza toda la prueba en realización con estos elementos. Así, se separa la aplicación de servicio que se encarga de la prueba y la otra de iniciar el navegador y obtener los elementos webs necesarios para poder trabajar sobre ellos.

Por otro lado, como se planificó que cada prueba se tenía que ejecutar en el navegador de tres diferentes dispositivos: en Google Chrome de un ordenador de escritorio, de un móvil y de una tableta. La ejecución de las pruebas en un ordenador será realizada en el mismo que se ejecuté la prueba. Sin embargo, la ejecución de las pruebas en móvil y tableta será una simulación. Para que todas las pruebas se lancen en un mismo modelo de dispositivo se ha escogido, como dispositivo móvil y tableta, el iPhone X y el iPad respectivamente. Por tanto, se ha creado una clase abstracta *BaseAutomationTests* que se utiliza para las pruebas a implementar. Esta clase solo contiene tres funciones y son las que aparecen en la siguiente imagen:

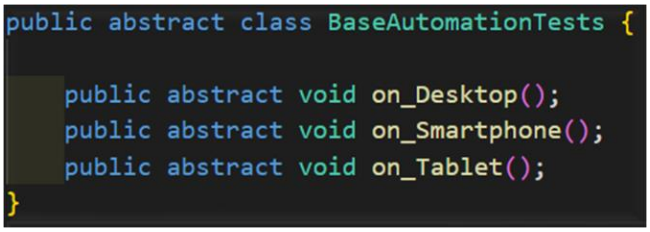

Figura 9: Clase Abstracta base para las pruebas

A continuación, en la figura 10 se puede observar cómo queda estructurada la clase que implementa la prueba de aceptación.

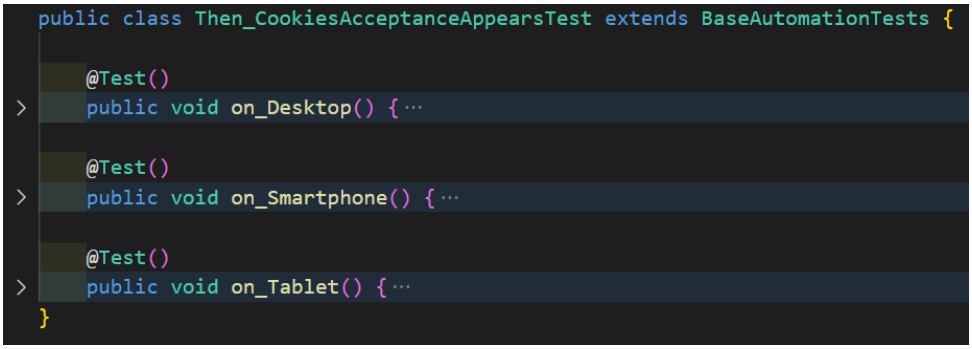

Figura 10: Ejemplo estructura de la prueba automatizada aceptación cookies

Por tanto, una prueba no sé da como completada sin que se haya pasado con éxito en los tres dispositivos. Cabe destacar que algunas pruebas tienen dependencias de otras y para poder procesarlas correctamente se ha implementado esa dependencia en las pruebas. Por ejemplo, la siguiente prueba después de la aceptación de cookies, es la comprobación de restricción de edad. Para la implementación de esta prueba dentro del método que se encarga de la realización, primero tendrá una llamada a un método de la prueba aceptación de cookies donde su papel será únicamente abrir la página web en el navegador y realizar la aceptación de cookies. A continuación, se realizará la prueba de comprobación de restricción de edad. Así, se implementan todas aquellas pruebas que dependan de otra para poder realizar una prueba.

Todas las pruebas automatizadas realizan unos pasos en común: iniciar el *webdriver*, abrir un navegador, a continuación, se realiza la comprobación de la prueba, después para acabar se finaliza el *webdriver* y se cierra el navegador. Al tener estos pasos similares, se ha configurado con un método el arranque del *webdriver* en los diferentes dispositivos. De esta forma, si una prueba que tenga que simular en un dispositivo móvil,

se realizará la llamada a la función encargada de la configuración del webdriver del dispositivo móvil.[1]

12.2 Organización de las pruebas

El proyecto de automatización conlleva a crear muchas pruebas. Al tener una gran variedad de pruebas puede resultar difícil encontrar una prueba específica. Para evitar este tipo de problema y para que resulte fácil encontrar una prueba, se han organizado las pruebas usando la nomenclatura GHERKIN (*Given-When-Then)* [20].

En el primer lugar, cada una de las pruebas están en una carpeta con el nombre de tipo de usuario que puede ser un mánager o un usuario. Es decir, podemos tener un mánager que cree las promociones y un usuario participa en las promociones. Por tanto, podemos tener, *Given_NotRegisterUser* o *Given_PromotionsManager*. A continuación, dentro de cada carpeta podemos encontrar diferentes ficheros con la descripción de *When* y finalmente el nombre de clase que contiene la prueba empieza con *Then*. De esta forma se organiza los tests.

Para entenderlo un poco mejor, podemos ver un ejemplo, en las figuras 11 y 12. En la figura 10 se puede observar cómo estas distribuidas las pruebas en carpetas. En la figura 11 tenemos un ejemplo de una prueba con la historia de usuario "Como nuevo visitante cuando se muestra la restricción de edad y acepto entonces accedo a una promoción finalizada". Esta descripción de la historia usuario se ha representado con (*Given_NotRegisteredUser -> When_FinishedPromotion -> Then_AgeRestriccionsAppears)*. Así se organizan las diferentes pruebas para poder tener una búsqueda rápida.

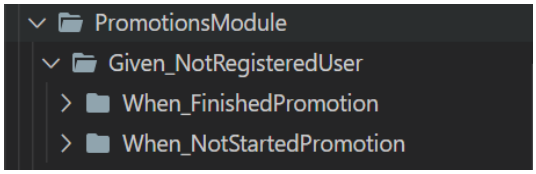

Figura 11: Estructura de ficheros

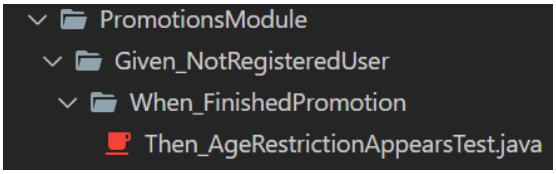

Figura 12: Estructura de ficheros para las pruebas

Por otro parte, aparte de esto cada clase de la prueba contiene la descripción de historia de usuario para poder tener información de la prueba. En la figura 13 se puede observar de la descripción de la prueba vista en el apartado anterior.

[1] En la implementación de las pruebas se ha evitado tener código repetido. En la implementación hay estructuras parecidas, pero en ningún momento código repetido.

```java
@Test(description = "Como nuevo visitante cuando se muestra la restricción de edad y acepto entonces accedo a una promoción finalizada")
public class Then_AgeRestrictionAppearsTest extends BaseAutomationTests{

    @Test()
    public void on_Desktop() {
        // ARRANGE
        IFinishedPromotionApplicationService applicationService = Container
                .resolve(IFinishedPromotionApplicationService.class);

        // ACT
        Boolean result = applicationService.appearsAgeRestriction_Desktop();

        // ASSERT
        Assert.assertTrue(result);
    }
```

Figura 13: Implementación del test aceptación restricción de edad

12.3 Ejecución

Para saber si una prueba se realiza correctamente es necesario ejecutarla. Cada sistema o programa tiene una forma de poder ejecutar. En este apartado se explicará cómo se pueden ejecutar las pruebas implementadas.

Primero de todos existe una prueba forma sencilla de ejecutar solo una prueba, esto se puede realizar abriendo el proyecto en editor de texto que permita ejecutar programas. Sin embargo, esta no es una propuesta de ejecución para estas pruebas automatizadas. En este proyecto, se quiere ejecutar muchas pruebas de forma conjunta, rápida y eficiente. Para ello, se ha configurado la ejecución de las pruebas en paralelo.

La ejecución de las pruebas se realiza con el uso de Maven y TestNG. Se ha configurado en *testng.xml* que se ejecuten con todos los procesadores disponibles en el ordenador. De esta forma utiliza lo máximo que puede, para ejecutar las pruebas paralelamente. Así, las pruebas tienen mejor rendimiento. No obstante, el objetivo de usar todos los procesadores era para que el resultado final se subiera en un servidor y allí utilizar el máximo rendimiento posible.

Para poder ejecutar las pruebas en un ordenador, se puede realizar abriendo el proyecto en una consola en Windows, y ejecutar el siguiente comando: *mvn clean test*. Con esto se ejecutan todas las pruebas paralelamente y la ejecución no finaliza hasta acabar la última prueba. Una vez finalizado, se pueden ver cuántas pruebas han sido finalizadas con éxito y cuántas han fallado. Además, también podremos ver el tiempo que ha tarda en realizar la ejecución de todas las pruebas.

Por otro lado, a la hora de ejecutar las pruebas podemos configurar el *webdriver* de *Selenium* para poder visualizar el navegador de cada prueba. Sin embargo, como este proyecto se requiere la ejecución de muchas pruebas en paralelo y cada una de las pruebas abre un navegador, por parte del cliente se ha solicitado no mostrar el navegador cuando se ejecutan las pruebas. Para poder ofrecer una buena solución se ha creado un fichero de configuración de tipo JSON para que se pueda usar las dos opciones, tal como se puede apreciar en la figura 14. Con este fichero, lo único que hay que hacer es borrar el parámetro *--stateles* para poder visualizar las pruebas durante la ejecución.

```
"WebDriverConfigurationArray": [
    {
        "Description": "Desktop",
        "DeviceEmulationName": "",
        "WebDriverPreferences": {
            "download.default_directory": "./",
            "profile.default_content_settings.popups": "0"
        },
        "WebDriverArgumentArray": [
            "--no-sandbox",
            "--headless",
            "--window-size=1920,1080",
            "--disable-gpu",
            "--disable-dev-shm-usage",
            "--ignore-certificate-errors"
        ]
    },
```

Figura 14: Configuración del WebDriver de Selenium

12.4. Resultados

Después de haber visto como se ejecutan las pruebas automatizadas, en esta sección se va a explicar las distintas formas de visualizar los resultados de las pruebas ejecutadas.

Para empezar, después ejecutar el comando "*mvn clean test*" en la terminal de Windows, se inicia la configuración y empiezan a ejecutarse las pruebas. Una vez que finaliza, podemos encontrar la siguiente información que aparece en la Figura 15. Se puede observar que informa de las pruebas ejecutadas, pruebas falladas, número de errores encontrados durante la ejecución y el tiempo total de la ejecución que en este caso han sido 20 minutos con 21 segundos. Las pruebas han podido ejecutarse a ese tiempo gracias a la ejecución en paralelo. Si se hubiesen ejecutado secuencialmente, el tiempo podría llegar a ser a dos o más horas.

```
[INFO] Tests run: 683, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1,205.053 s - in TestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 683, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  20:21 min
[INFO] Finished at: 2022-01-07T12:41:35+01:00
[INFO] ------------------------------------------------------------------------
```

Figura 15: Resultados de la ejecución de las pruebas

Visualizar los restados de esta forma es una buena opción, pero es mejorable para que incluya alguna otra información. Para ello, se ha configurado la creación del *allure report*, un informe que contiene la información de las pruebas ejecutadas, así como la descripción de la historia de usuario ejecutada. Para poder visualizar este informe una vez obtenido los resultados de las pruebas, se tiene que ejecutar el comando "mvn allure:report" que se encarga de generar y abrir localmente el informe. Un ejemplo del informe se puede ver en la figura 16.

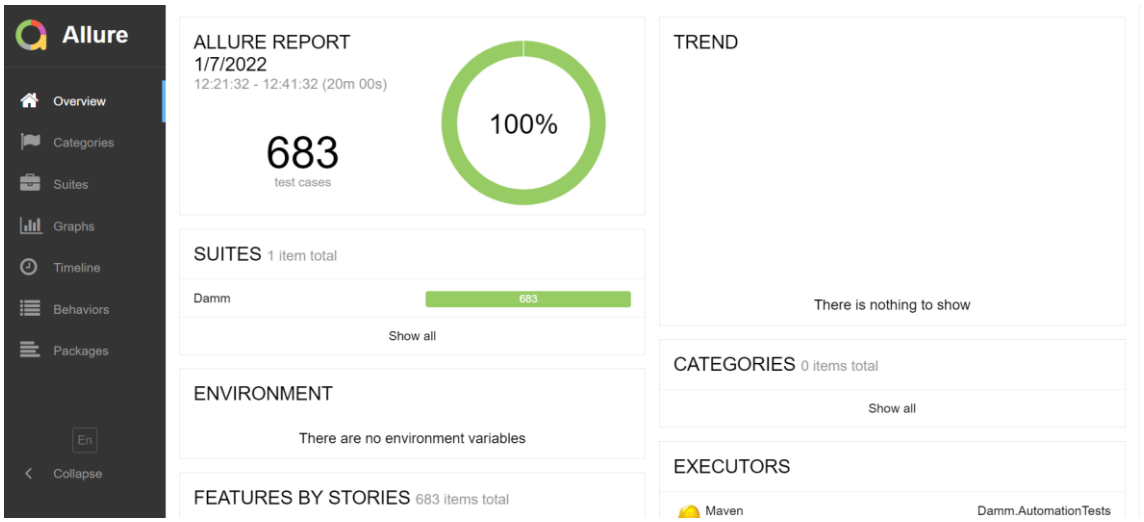

Figura 16: Página principal para la visualización del informe de Allure

En esta ilustración se puede ver la misma información que se mostraba en la terminal, pero con una interfaz mucho mejor. En la parte izquierda de la imagen se puede visualizar el menú. Si accedemos a la sección *Suites* podemos visualizar las pruebas ejecutadas con cada una de la descripción. Por ejemplo, en la figura 17 se muestra por una parte todas las pruebas y a la derecha una prueba ejecutada en el navegador de Google Chrome de escritorio. Se puede ver que contiene una pequeña descripción y también el tiempo que se ha tardado en finalizar esta prueba.

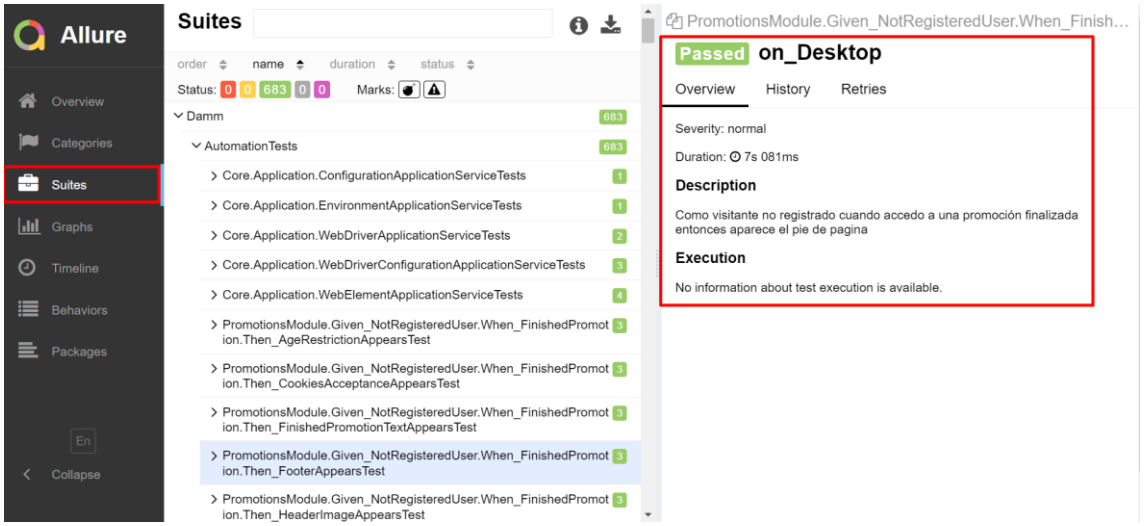

Figura 17: Visualización en Allure de la información de las pruebas ejecutadas

Después de crear este informe *allure*, se consideró mejorar y añadir alguna característica. Se ha decidido crear el historial de ejecución de cada prueba para que tenga un registro de las pruebas para saber cuándo fallan y pasan.

Para poder crear el historial de ejecuciones, se irá guardando el fichero xml que se crea después de la ejecución de las pruebas en paralelo. A continuación, se generará el allure report igual que como se creó anteriormente, con el comando “mvn allure:report”. De esta forma el allure report, contiene la información de las ejecuciones y así el allure

puede leer y mostrar el resultado en su informe. En las siguientes imágenes podemos observar el historial de ejecución de una prueba y algunas gráficas que muestran estadísticas de las pruebas ejecutadas.

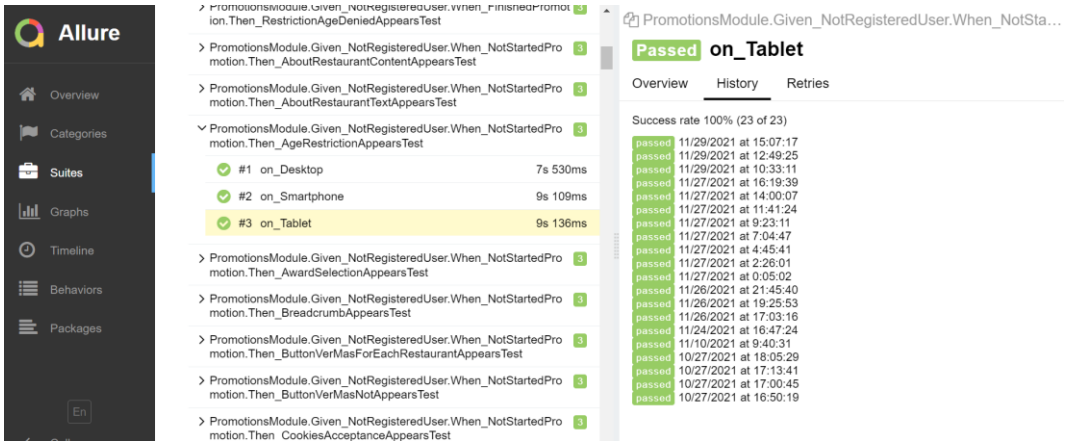

Figura 18: Informe Allure histórico

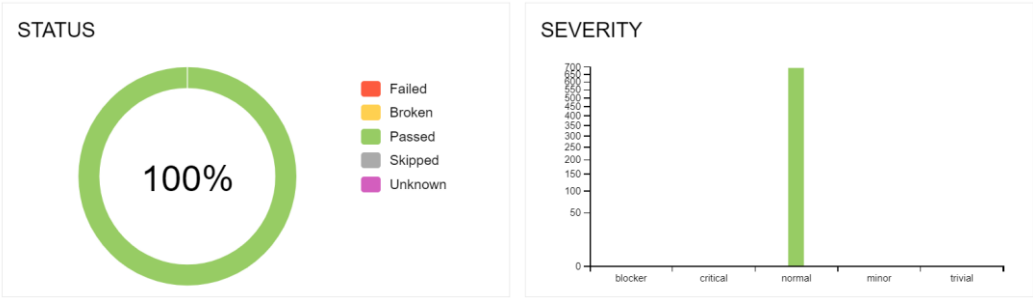

Figura 19: Gráficas de los resultados

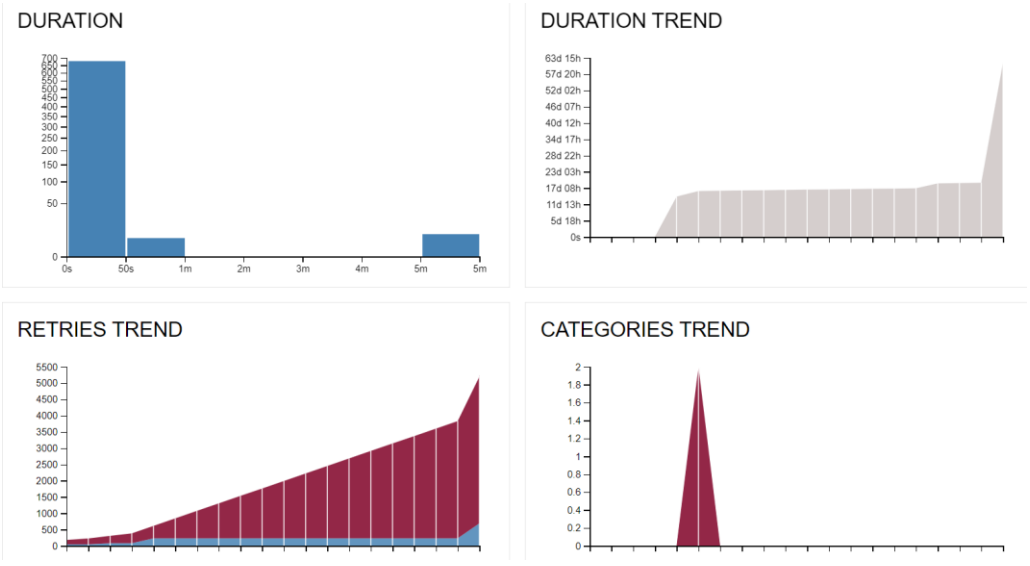

Figura 20: Gráficas del tiempo de ejecución de las pruebas

12.5. Tecnologías utilizadas

En esta sección se presentarán las tecnologías utilizadas para el desarrollo de las pruebas automatizadas. A continuación, se especificación las tecnologías:

- **Selenium** [7]: Es un software que ofrece herramientas y librerías para realizar pruebas automatizas para aplicaciones web haciendo uso de un navegador. Todas las pruebas creadas se han realizado con el uso de Selenium y de los métodos que ofrece. Concretamente, se ha utilizado la versión 4.0.0 que soporta el lenguaje de programación Java.

- **Java** [21]: Es un lenguaje de programación que trabaja como lenguaje de programación orientado a objetos. Para este proyecto se ha usado la versión 1.8 que es compatible con Selenium. Este lenguaje nos ha permitido implementar todas las pruebas y poder utilizar sus librerías.

- **HTML** [22]: Es un lenguaje de etiquetas de hipertexto. Es el elemento básico que define la estructura del contenido de una página web. Se ha utilizado el código HTML para obtener los XPaths del contenido de una página web.

- **XPath** [23]: *XML Path Language e*s un lenguaje que permite construir expresiones regulares que se utilizan para recorrer y procesar un documento XML. Concretamente, se ha utilizado la versión 1.0 que es compatible para poder usarse con el navegador de Google Chrome. Para nuestro proyecto, los XPaths se utilizaban para buscar y procesar elementos de una página web.

- **JavaScript** [24]: Es un lenguaje de programación que generalmente se usa para el *front-end* de una aplicación. Sin embargo, en nuestro proyecto se ha utilizado como alternativa para poder procesar elementos web que no funcionaban con los métodos de Selenium.

- **TestNG** [25]: Es un *framework* para realizar pruebas. Esta basando en JUnit y NUnit y trabaja con el lenguaje de programación Java. A diferencia de JUnit y NUnit introduce nuevas funcionalidades para poder implementar pruebas. En este proyecto, se ha utilizado la versión 7.4.0 que nos ha facilitado la implementación de las pruebas automatizadas.

- **Maven** [26]: Es una herramienta que se usa para la gestión y creación de proyectos principalmente para proyectos en Java. Para este proyecto, se ha utilizado la versión 3.8.1. Con esta herramienta se compila el proyecto antes de ejecutar las pruebas. Además, también se ha usado el *plugin* allure report para crear el informe de los resultados de las pruebas automatizadas.

- **JSON** [27]: Es un formato de texto diseñado para el intercambio de datos para ser legible y fácil de entender para el ser humano. Se ha utilizado este formato para realizar la configuración de los parámetros que se tienen que enviar al controlador de Selenium.

13. Evolución durante el desarrollo del proyecto

En este apartado de la memoria, se ha realizar los informes de seguimiento del desarrollo de la implementación utilizando la metodología agile. A continuación, se informará de los progresos y problemas de cada *sprint*. Para explicarlo, cada *sprint* contará con un *sprint planning,* un *sprint retrospective* y *sprint review.*

Antes de proceder en la explicación del desarrollo de cada *sprint*, se considera importante informar sobre algunas decisiones que se han tomado para llevar a cada *sprint planning, sprint retrospective* y *sprint review.*

En primer lugar, en cada una de las sprints *plannings* se ha realizado la estimación de cada historia de usuario a implementar en el *sprint* con *planningpoker* [28]. Esta estimación ha consistido en realizar la votación entendiendo la historia de usuario y la complejidad que tenía. Para determinar cuál es la estimación adecuada se tenía en cuenta el esfuerzo, la complejidad, la incertidumbre y la experiencia. De esta forma con todo el equipo se asignaba una estimación a las *user stories*.

Por otro lado, para los *sprints restrospectives* también se utilizó una herramienta y una forma de efectuar estas reuniones. Se utilizaba *timboretro* una página web que permitía realizar *sprint retrospective* de forma limitada pero gratuitamente [29]. Con esta herramienta cada miembro del equipo escribía de forma privada los aspectos a discutir en la reunión, después se ponía en común para hablar entre todos los puntos escritos. Finalmente, se escogían entre todos los puntos más importantes, los que se trabajarían en los próximos *sprints*.

13.1. *Sprint* 1

13.1.1. *Sprint Planning*

El objetivo principal de este primer *sprint* era dar los primeros pasos con la tecnología *Selenium* y realizar las principales historias de usuario que tenían más dependencias de otras. En este proyecto cada *prueba* a realizar corresponde a una historia de usuario. Por tanto, habrá algunas pruebas que tengan dependencias de otras y en consecuencia se tengan que implementar antes. De esta forma, en el siguiente *sprint* se podrían escoger más historias de usuario gracias a que las dependencias principales ya estarían implementadas. Otro objetivo que se podría compartirse en todos los *sprints* es que cada iteración se ha entregar código que aporte valor. Como esta es la primera iteración, se intentarán que las pruebas automatizadas tengan un valor añadido al producto final.

Las tareas para este sprint eran realizar toda la instalación y configuración de las tecnologías necesarias para el proyecto. Así como, crear el repositorio y compartirlo con el equipo. Después otras tareas eran empezar con las pruebas de visualización de promociones y de los primeros. Especialmente, se tenía que empezar a realizar las tareas de aceptación de cookies y aceptación de restricciones de edad ya que muchas tenían dependencias de estas.

En resumen, en esta primera iteración se busca automatizar las principales pruebas que tengan que ver con la visibilidad de las promociones ya creadas en producción. Además, familiarizarse con *Selenium* para que en los siguientes *sprints* el ritmo de desarrollo sea más creciente. Por último, este sprint consta de 92 puntos.

13.1.2. *Sprint Retrospective*

El objetivo principal del sprint *retrospective* es intentar mejorar la productividad y la calidad del desarrollo para poder tener un producto de alta calidad. Además, también identificar posibles errores cometidos durante la iteración e intentar no repetir en los siguientes *sprints*. Esto se puede ver mediante observaciones sobre una iteración finalizada, destacando aspectos que han funcionado correctamente, aspectos que se pueden mejorar y aspectos que deben dejar de realizar.

A continuación, se detallan los aspectos puestos en común:

- Aspectos que han funcionado bien:
  - Todas las tareas planificadas para este sprint se han finalizado satisfactoriamente.
  - El resultado del primer sprint ha sido positivo y no se han encontrado dificultades altas que no se puedan resolver durante el sprint.

- Aspectos para mejorar:
  - Se tiene que empezar a realizar *pull request* a la rama delop después de finalizar una tarea en rama *feature*.
  - Empezar a codificar pruebas que se puedan ejecutar en paralelo.
  - En el siguiente *sprint* adaptar y configurar las pruebas automatizadas para que se puedan ejecutar en paralelo.

- Aspectos dejar se tienen que dejar:

No se considera que se tenga que dejar una *actividad* realizada este *sprint*.

13.1.3. *Sprint Review*

Una vez finalizado el *sprint* se ha realizado una reunión con el equipo de desarrollo, en la que ha participado el tutor del trabajo, los responsables de proyectos de *Drupal* y algunos representantes del cliente. El objetivo de esta reunión era verificar los avances que se han realizado durante este *sprint*. Finalmente, se ha realizado una pequeña demo para mostrar las pruebas completadas. Además, el objetivo también era verificar con el cliente si el avance y los planes de prueba realizados iban por un buen camino. Así, al final del *sprint* teníamos una opinión, en primera mano, de las partes interesadas del proyecto.

Una finalizada la reunión, se ha podido llegar a siguientes conclusiones:

- Se ha podido realizar todas las tareas correctamente tal como se planificó en el *sprint planning.*
- La carga de trabajo estimada de cada historia de usuario ha estado correcta.
- Hay que mantener el mismo ritmo de trabajo y seguir avanzado con el desarrollo.
- Las pruebas automatizadas en este *sprint* son aceptadas y completadas por el cliente. Sin embargo, hay que adaptarlas a la ejecución en paralelo.

13.2. *Sprint 2*

13.2.1. *Sprint Planning*

El objetivo principal de este segundo *sprint* es empezar con la ejecución en paralelo de las pruebas automatizadas y después empezar las nuevas pruebas teniendo en cuenta su ejecución en paralelo. De esta forma a partir de este sprint todas las pruebas a implementar no se darán como complementadas hasta que no sé pasen cuando se ejecutan en paralelo. Con esto conseguimos no dejar trabajo pendiente y acumularlo en otros *sprints*.

Las tareas para este *sprint* serán la comprobación de visibilidad de los premios de las promociones existentes en producción. Estas pruebas consisten en verificar la información que se presenta en las páginas webs es visible y correcta. Por otra parte, antes de realizar estas tareas hay que dedicar tiempo para configurar y adaptar las pruebas automatizadas para que se puedan ejecutar en paralelo. Además, este *sprint* consta de 75 *story points*, puntuación estimada con la votación del equipo. En esta iteración se puede observar que hay menos puntos seleccionados, esto es debido a que este *sprint* hay que dedicar tiempo para configurar y adaptar la codificación de las pruebas automatizadas en el sprint anterior. Por ese motivo, se ha planificado un poco a la baja con el objetivo de que se pueda completar todas historias de usuario seleccionadas para este sprint.

En resumen, en esta segunda iteración se intenta configurar la ejecución paralela y a partir de aquí automatizar las pruebas teniendo en cuenta que se tienen que pasar en la ejecución paralela. Después se ha de automatizar las pruebas que tienen que ver la visibilidad de los premios de cada promoción.

13.2.2. *Sprint Retrospective*

Como ya se explicó en el *sprint* anterior, el objetivo de esta reunión consiste en mejorar la productividad y la calidad del producto para las siguientes iteraciones. A continuación, se listan los aspectos que han funcionado bien, qué debería mejorar y se tendría que dejar de realizar para el producto final sea de calidad.

- Aspectos que han funcionado bien:
  - Se ha conseguido configurar y ejecutar las pruebas automatizadas en paralelo.
  - La carga de trabajo para este sprint ha sido elevada aun así se ha podido completar las tareas planificadas.
  - Se ha organizado bien el tiempo durante la iteración evitando mucha carga de trabajo para los últimos días del *sprint*.
- Aspectos para mejorar:
  - Se ha de dedicar más tiempo para realizar la documentación.
  - Consensua nombre de los métodos antes de empezar a implementar.
  - Investigar el resultado de las pruebas automatizadas que a veces dan falsos negativos. Buscar cual puede ser el motivo para que estas pruebas a veces den un resultado negativo.
- Aspectos para dejar de hacer:
  - No crear una *pull request* sin que la rama *feature* esté alineada con la rama *Develop.*

13.2.3. *Sprint Review*

Después de finalizar este segundo *sprint*, se ha realizado el *sprint review* para verificar los avances que se ha habido y después realizar una pequeña demostración. En esta reunión, han participado el equipo de desarrollo, el director del trabajo final y algunos representantes del cliente Damm.

Una vez acabada la reunión, se ha podido llegar a las siguientes conclusiones:

- Se han podido realizar todas las tareas propuestas en esta iteración.
- El ritmo de trabaja ha sido constante durante el *sprint*.
- En general la carga de trabajo ha sido correcta. No obstante, la estimación de las pruebas que tienen que ver con los restaurantes ha sido inferior a la carga real ya que las pruebas a realizar eran más complicadas, especialmente, para los dispositivos móvil y tableta. En estos dispositivos se necesitaba una implementación muy diferente a las pruebas de escritorio ya que el código HTML de las páginas en móvil y tableta era diferente.
- Gracias a la configuración para poder ejecutar las pruebas en paralelo, se ha podido reducir hasta la mitad el tiempo de ejecución.
- Se ha de mejorar y corregir los falsos negativos de las algunas pruebas detectados en las ejecuciones en paralelo.
- Desde el cliente se creará un entorno de pruebas para este proyecto en el que haya una copia de módulo de promociones. De esta forma, este entorno será establece y no tendrá cambio de códigos constantes.

13.3. *Sprint* 3

13.3.1. *Sprint Planning*

El objetivo principal de este tercer *sprint* es poder verificar la creación de promociones y la participación en ellas. Se requiere que de forma automática se acceda al formulario y se registre una promoción. Después una vez creada, cuando se accede a una promoción, se un usuario se pueda registrar en esa promoción.

Por tanto, las tareas de este *sprint* son realizar el *testing* automático con el entorno de pruebas para poder crear promociones y crear pruebas sobre la participación de usuarios en una promoción empezada. Cabe destacar, en la creación de promociones tendremos en primer lugar pruebas de visibilidad de los campos del formulario. Una vez finalizadas estas tareas se procederá a las de creación de promociones. Por tanto, al final de este *sprint* ya tendremos pruebas automatizadas para la creación de promociones y se podrá comprobar si una promoción se crea correctamente. Además, también se ha empezar a configurar para que después de ejecutar las pruebas en paralelo, se cree un informe con la información de las pruebas ejecutadas.

Este *sprint* consta de 105 *story points.* Como esta iteración tiene una complejidad alta por la parte de creación de promociones se ha considerado que repartir estos puntos en este sprint es correcto. Además, finalizando estas pruebas se habrá completado una gran parte de trabajo.

13.3.2. *Sprint Retrospective*

Siguiendo como antes, el objetivo de *sprint retrospective* es mejorar la productividad y la calidad del producto en los próximos *sprints*, observando cómo ha ido esta iteración. A continuación, se listan los aspectos que han funcionado bien, qué debería mejorar y aspectos se tendría que dejar de realizar para el producto final sea de calidad.

- Aspectos que han funcionado bien:
  - A pesar de las dificultades y obstáculos tenidos en este se ha podido completar unos 90 puntos del *sprint.*
  - La carga de trabajo estimada ha sido correcta. Sin embargo, a causa de problemas afectados a la red del cliente han generado que no se puedan completar durante este *sprint.*
  - Se han podido corregir los tests que daban resultados falsos negativos.
  - Se ha podido implementar la configuración para mostrar el informe de resultados de las pruebas ejecutadas.
- Aspectos para mejorar:
  - Se ha de dedicar más tiempo para realizar la documentación.
  - Completar las historias de usuario que no han podido completar en el siguiente *sprint.*
  - Comprobar que cuando si completa la ejecución de los tests, entonces, el navegador se cierra en todas las pruebas y no queda ningún navegador abierto.

- Aspectos para dejar de hacer:
  - Se ha dejar de usar *xpath*[2] extensos para obtener un elemento web. En su lugar se ha de utilizar un *xpath* único y preferentemente el identificador de cada elemento.

13.3.3. *Sprint Review*

Una vez finalizado este tercer *sprint*, se ha realizado el *sprint review* para verificar los avances que se han experimentado y para realizar una pequeña demostración a las partes interesadas. En esta reunión, han participado el equipo de desarrollo, el director del trabajo final y algunos representantes del cliente Damm.

Una vez acabada la reunión, se ha podido llegar a las siguientes conclusiones:

- A consecuencia, del cierre de la VPN de cliente Damm por seguridad, en este *sprint* no se ha podido completar el 100% de las tareas planificadas.
- A pesar de la situación, se ha podido realizar una gran laboral para poder seguir avanzado con el proyecto.
- Se ha recuperar las pruebas que no se han podido completar.
- Se da por entendido que el entorno de pruebas sin conexión VPN no es buena. El equipo de cliente Damm procederá a crear el entorno de pruebas para este proyecto una vez que se restablezca su red y la seguridad de la compañía esta asegurada.
- Se ha de seguir avanzado con el proyecto con las herramientas disponibles en estos momentos, ya que no sé puede desviar más de la planificación.

[2] Xpath es un lenguaje de consulta que permite construir expresiones para obtener objetos de un documento XML.

13.4. *Sprint* 4

13.4.1. *Sprint Planning*

En el *sprint planning* de la esta cuarta iteración se ha establecido como objetivo añadir la nomenclatura *GHERKIN* (*Given, When, Then)* a los tests desarrollados. Además, se ha desarrollar las pruebas de envío de correo. Actualmente, desde el entorno de pruebas se puede configurar el envío de correos a los participantes con toda la información que se desea proporcionar a los usuarios de una promoción. Existen diferentes envíos de correo, pero para las pruebas se utilizarán los más utilizados por el módulo. Por otra parte, también se ha decido factorizar algunas pruebas ya que el tiempo de ejecución de esas pruebas es muy elevado.

Por tanto, las tareas de este *sprint* son factorizar las pruebas con alto tiempo de ejecución, implementar pruebas de envío de correos y verificar que se recibo el correo. Además, también se tendrá de finalizar las tareas que no pudieron acabar en el sprint anterior. De esta forma, intentar casi todo acabado para que el siguiente sprint no tenga mucho trabajo.

En resumen, en este sprint se busca completar las tareas que quedaron pendientes en el *sprint* anterior, automatizar las pruebas de creación de promociones y verificar que se enviar correctamente. Finalmente, este *sprint tiene 152 story points*.

13.4.2. *Sprint Retrospective*

Como en los otros *sprints* el objetivo de *sprint retrospective* es mejorar la productividad y la calidad del producto en las próximas iteraciones mediante la observación de cómo ha ido esta iteración. A continuación, se listan los aspectos que han funcionado bien, qué debería mejorar y aspectos se tendría que dejar de realizar para el producto final sea de calidad.

- Aspectos que han funcionado bien:
    - Se ha podido completar el objetivo planificado, aunque no se haya podido finalizar una historia de usuario de 8 puntos en la que se tiene que optimizar algunas pruebas para reducir su tiempo de ejecución.
    - La carga de trabajo de este *sprint* ha sido alta y la estimación de las historias de usuarios han sido correctas.

- Aspectos para mejorar:
    - En general, el *sprint* ha sido productivo, no sé considera que se haya realizado alguna practica que se tenga que mejorar.

- Aspectos para dejar de hacer:

  No se considera que se haya realizado alguna practica que se tenga que dejar de hacer.

13.4.3. *Sprint Review*

Una vez finalizado este *sprint*, se ha realizado el *sprint review* para comprobar los avances que se han experimentado y ver cómo se encuentra el proyecto para finalizar. En esta reunión, han participado el equipo de desarrollo, el director del trabajo final y algunos representantes del cliente Damm.

Una vez acabada la reunión, se ha podido llegar a las siguientes conclusiones:

- En general, se ha conseguido el objetivo propuesto.
- El ritmo de trabajo ha sido productivo y se ha realizado un buen trabajo de equipo.
- El proyecto sigue avanzado con un ritmo positivo, el cliente está satisfecho del trabajo realizado y espera buenos resultados.
- Después de realizar esta reunión, el cliente informa que está planteando en ampliar el proyecto para realizar *testing automatizado* de otras secciones del módulo y también crear el entorno de pruebas para este proyecto. Hasta ahora no ha podido crear debido a que el trabajo que tenía de otros proyectos.
- A pesar de encontrarse con algunos obstáculos imprevistos se ha podido recuperar el ritmo y el cliente está satisfecho del trabajo.

13.5. *Sprint* 5

13.5.1. *Sprint Planning*

El objetivo principal de este último *sprint* es finalizar todas las pruebas previstas para este proyecto. Es decir, se ha acabar todas las historias de usuario pendientes en el backlog. Además, se ha de dar prioridad en este *sprint* a la documentación ya que en esta última iteración ha de que todo documentado.

Las tareas pendientes que quedan son las tareas de optimizar algunas pruebas, algunas pruebas de visibilidad y se ha de integrar todas las pruebas, repasando cada una para que se pueden ejecutar en Jenkins. Además, se ha decidido añadir el historial de ejecuciones de las pruebas en el informe de resultados. De esta forma, el informe tendrá información sobre qué momentos las pruebas han pasado satisfactoriamente y cuándo han dado un error. Este último *sprint* consta de 76 *story points.*

13.5.2. *Sprint Retrospective*

Como ya se explicó anteriormente, el objetivo de esta reunión es mejorar la productividad y la calidad del producto final. En este último *sprint* se ha realizado esta reunión, especialmente, sabiendo que el proyecto ha sido ampliado porque el cliente requiere más pruebas. A continuación, se listan los aspectos que han funcionado bien, qué debería mejorar y aspectos se tendría que dejar de realizar para los siguientes *sprints*.

- Aspectos que han funcionado bien:
    - Se han podido completar todas las historias de usuario.
    - La carga de trabajo de este *sprint* no ha sido alta y se ha realizado un buen trabajo para conseguir nuestro objetivo final.
    - Se ha podido dedicar más tiempo para la documentación y preparar la entrega final.

- Aspectos para mejorar:

  No se considera que se haya realizado alguna practica que se tenga que dejar de hacer.

- Aspectos para dejar de hacer:

  No se considera que se haya realizado alguna practica que se tenga que dejar de hacer.

13.5.3. *Sprint Review*

Finalmente, se ha realizado el *sprint review* para comprobar los avances que se han experimentado y se ha realizado la demostración con el resultado de todas las pruebas. En esta reunión, han participado el equipo de desarrollo, el director del trabajo final y algunos representantes del cliente Damm.

Una vez acabada la reunión, se ha podido llegar a las siguientes conclusiones:

- Se ha podido completar con el requerimiento de pruebas de manera satisfactoria.
- No se han encontrado problemas de últimos momentos.
- El ritmo de trabajo ha sido muy bueno y se ha podido dedicar bastante tiempo en la documentación.
- La demo ha ido sido corta, pero ha ido bastante bien porque ejecutar todas las pruebas requieren como mínimo media hora, aunque se utilice un ordenador potente.
- El proyecto ha tenido un buen resultado para el cliente y ha decidido ampliarlo para realizar otras pruebas porque considera que es un proyecto y hay que realizar automatizar más pruebas. Además, ha informado que hay un desarrollador encargado de crear el entorno de pruebas para este proyecto. Por tanto, ha decidido no subir las pruebas para ejecutarla en Jenkins porque ha decidido ejecutar el entorno de pruebas que se está preparando.

A continuación, en la siguiente gráfica podemos observar el estado de *story points* del proyecto. Como se puede observar en el primer *sprint* se ha logrado el objetivo puesto. Seguidamente, en el segundo *sprint* se bajo un pongo la carga de trabajo para preparar las pruebas del primer *sprint* para la ejecución en paralelo. En la tercera iteración por la afectación de la desactivación de red no se pudo finalizar el objetivo planificado. Finalmente, en el sprint 4 y 5 se pudo alcanzar el objetivo. En general, teniendo en cuenta los obstáculos que se encontrado, el proyecto ha tenido un rendimiento de crecimiento, aunque en la última iteración solo se tuve que realizar las tareas que quedaban pendientes.

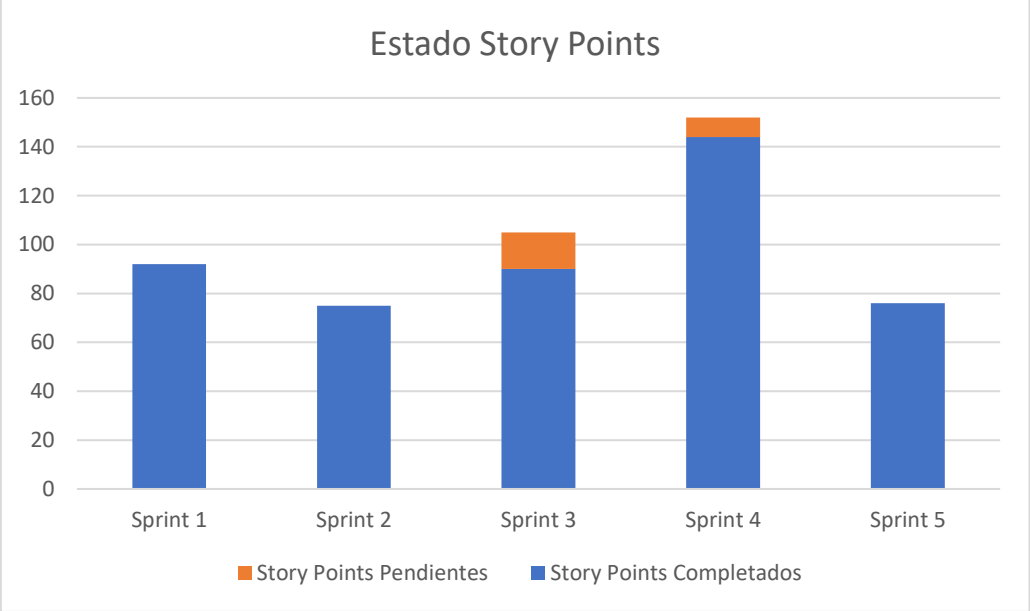

Figura 21: Estado Story Points del Proyecto

14. Discusiones

En este apartado, se analizará el trabajo realizado durante todo el tiempo de desarrollo del proyecto. Se valorará la metodología de trabajo utilizada y se comprará la planificación y el coste del proyecto calculado en la gestión del proyecto (GEP). Finalmente, se expondrán los aspectos de sostenibilidad del proyecto.

14.1 Metodología de trabajo

Utilizar la metodología agile para este proyecto, ha sido una decisión muy acertada para el desarrollo del proyecto, ya que hoy en día se usa en muchos proyectos. Tener un desarrollo incremental ayuda mucho a tener un orden y poder alcanzar los objetivos de manera positiva. También nos ha favorecido tener un buen ambiente de trabajo y en el equipo poder alcanzar a finalizar el producto.

Un aspecto muy positivo y necesario para cualquier proyecto ha sido tener una buena comunicación entre el equipo de desarrollo. Para ello, se han realizado las sesiones de *sprint daily, sprint planing*, *sprint review* y *sprint retrospective*. Gracias a estas reuniones se ha podido aprovechar el tiempo y resolver cualquier imprevisto sin que afecte a la entrega final.

Para el seguimiento de las tareas, ServiceNow ha sido clave para tener todo el proceso de cada sprint al día. Sin embargo, al ser la primera vez que se utilizaba esta herramienta para la gestión proyecto, se ha ido aprendido a lo largo del proyecto. Además, en esta plataforma el cliente puede seguir el avance del proyecto y aportar su valoración, de esta forma el producto se ha elaborado teniendo en cuenta los requisitos del cliente.

Por otra parte, para este proyecto unos de los objetivos establecidos era que el código debía ser mantenible y ampliable. Para poder lograr este objetivo, para el diseño de las pruebas se utilizó los cinco principios básicos de programación, conocidos como SOLID. Gracias a esto, hemos podido garantizar y lograr el objetivo. Además, para poder verificar que cada miembro está cumpliendo los principios SOLID en su implementación, cada uno al finalizar una tarea, en el repositorio de Github creaba un *pull request* para que la resta del equipo revisará el código. Así, el código final tenía una revisión y era aceptado por todo el grupo de desarrollo. Haciendo uso de *pull request*, se estaba usando el GitFlow para tener más organizado y seguro el proceso de crear nuevos features y tener una versión de código.

En definitiva, utilizar la metodología agile para el desarrollo de este proyecto ha sido muy beneficioso y ha dado resultados positivos. Los principios establecidos y las herramientas para la gestión también han sido favorables, gracias a ello, se ha podido finalizar el trabajo.

14.2 Planificación temporal

Durante el curso de GEP, se realizó la planificación temporal para el desarrollo del proyecto, todas las horas asignadas a cada tarea eran una estimación excepto las horas dedicadas a realizar el documento de GEP. Sin embargo, ahora después de finalizar el proyecto, podemos saber las horas reales dedicadas en cada *sprint*. En este apartado, se compararán las horas dedicadas *en* cada *sprint*.

La siguiente tabla que se puede apreciar, muestra aproximadamente las horas reales dedicadas en cada iteración del proyecto. Se ha considero que es mejor comparar las horas dedicadas a cada *sprint* que las horas de cada tarea tal como se realizó en la entrega de GEP. El motivo de esto es porque durante el desarrollo del proyecto se han añadido alguna tarea o se ha tenido que mover la tarea de un *sprint* a otro, por eso es mejor considerar las cifras de cada fase y visualizar las horas totales reales del proyecto.

Fases	Duración (horas)
Gestión del proyecto	150
***Sprint* 1**	100
Sprint 2	97
Sprint 3	125
Sprint 4	85
Sprint 5	64
TOTAL HORAS	**621**

Tabla 18: Horas dedicadas en cada fase del proyecto

Como se puede observar, se han dedicado aproximadamente 621 horas totales para realizar el proyecto. Sin embargo, el cálculo aproximado de las horas que se hizo durante el curso de GEP eran de 609 horas. Podemos observar que hay una diferencia de 20 horas, esto es debido a los riesgos que hemos tenido durante el proyecto. Como podemos acordar, en la planificación del proyecto se consideró tener en cuenta 100 horas para la gestión de riesgos. Es verdad que ahora se puede afirmar que 100 horas eran muchas, pero eso nos ha permitido tener una buena planificación y cumplir con las horas previstas.

14.3 Costes

Una vez calculadas las horas reales dedicadas para la realización del proyecto, a continuación, en este para se ha procesar a calcular el coste del proyecto y comparar si se ha cumplido con el presupuesto estimado en la planificación del proyecto. Primeramente, ya se puede esperar que el coste será diferente al presupuesto calculado en GEP ya que hemos podido observar en el apartado anterior que ha habido una variación en las horas calculadas para la realización del proyecto.

A continuación, en la tabla se muestran costes reales del personal para visualizar las horas dedicadas por cada miembro y obtener el coste.

Fases	Rol (dedicación en horas)				Coste (€)
	PO	**AN**	**DT**	**DB**	
Gestión del proyecto	50	30	20	50	2834
Sprint 1		5	45	50	1437
Sprint 2		3	39	55	1357
Sprint 3		6	56	63	1793
Sprint 4		2	43	40	1231
Sprint 5		2	12	48	801
TOTAL					**9453**

Tabla 19: Tabla con el coste del personal del proyecto

Como se puede observar el coste real del personal corresponde a 9453 euros mientras que en la planificación eran 9124 euros sin el coste de contingencia. Esto es porque durante el desarrollo ha habido desvíos y algunas tareas han sido modificados, a consecuencias las horas dedicadas han sido más de las previstas. Por ese motivo, el coste del personal también ha sido afectado. Sin embargo, como se puso el 15% de contingencia para situaciones como en la que nos hemos visto afectados. Esto significa que el coste del personal estimando con contingencia era 10493 euros, cifra que es menor al coste real del personal. Por tanto, se puede afirmar que a pesar de las situaciones se han visto durante la realización del proyecto el resultado es positivo.

Seguidamente, se calcula el coste final del proyecto. Durante el desarrollo del proyecto no ha habido la necesidad de utilizar el coste de contingencias para el coste del material y costes indirectos.

Tipos de coste	Coste (€)
Coste del personal	9453
Coste del material	393
Costes indirectos	1670
Costes imprevistos	835
Coste Total	**12351**

Tabla 20: Coste final del proyecto

Por tanto, el coste total del proyecto es de 12351 euros que en comparación con el coste estimado con contingencias era de 13450 euros. Esto significa para la realización del proyecto se ha podido ahorrar 1099 euros, esto es porque no se ha utilizado el coste de contingencia que se añadió en el presupuesto estimado para el desarrollo del proyecto.

Como se planificó en la entrega de GEP, el coste de contingencia que no se utilice se reservan para un siguiente proyecto de automatización del módulo del Drupal. Para esto, no hará falta ni esperar, como se comentó anteriormente el proyecto se ha ampliado y el dinero ahorrar en este proyecto se utilizará para la ampliación del proyecto.

14.4 Informe de Sostenibilidad

Después de finalizar el proyecto, una discusión importante es cuánto de sostenible es el proyecto. Para ello, en este apartado se va a analizar y cuantificar el impacto ambiental, económico y social, teniendo en cuenta la matriz de sostenibilidad.

14.4.1. Dimisión ambiental

Para el desarrollo de este proyecto, el recurso más utilizado es la electricidad, ya que se ha utilizado diariamente la luz y el ordenador para construir este software de pruebas automatizadas. Actualmente, las pruebas automatizadas se pueden ejecutar en cualquier ordenador. Sin embargo, el software está preparado para realizar el despliegue a un servidor, cosa que se prevé que se realice en el futuro. Este servidor también necesita electricidad para poderse en funcionamiento.

Respecto al proyecto puesto en producción, se ha intentado reutilizar todos los recursos disponibles y se ha evitado contratar más servicio. En este proyecto no se ha contratar ningún servidor y el uso del recurso de la electricidad ha sido lo necesario. Por tanto, se estima que la parte del proyecto puesto en producción ha estado un 9/10. Si en el futuro, se hiciera de nuevo este proyecto es probable que el recurso de la electricidad fuese

menor ya que el proyecto se podría finalizar con menos tiempo. A consecuencia, se reducirá el uso de la electricidad en el proyecto.

Por otra parte, durante la etapa de la vida útil para la ejecución del proyecto se necesita electricidad. El uso de este recurso es para la ejecución ya sea un ordenador o en servidor propio de la empresa. El impacto de este recurso es mínimo en comparación con la forma que se ha estado realizado las pruebas. En este caso estamos hablando aproximadamente 700 pruebas automatizadas que se ejecutan entre 21-24 minutos. Esto realizarlo manualmente serían muchas horas. De esta forma, globalmente el uso de este software de pruebas mejorará la huella ecológica. Por eso, se estima la nota de la vida útil de este proyecto de 9/10.

Por último, un riesgo que se puede identificar sería un aumento de los servidores para ejecutar las pruebas más eficientemente. Esto se puede producir si se crean más muchas pruebas automatizadas y se dividir para que se ejecuten en diferentes servidores, en ese caso la huella ecológica empeoraría. Sin embargo, la probabilidad de este riesgo es muy baja. Por tanto, la nota para los riesgos es de 9/10

14.4.2. Dimisión económica

Después de finalizar el proyecto, se ha realizado un estudio del cálculo del coste del proyecto y se ha comparado con el presupuesto estimado. Este proyecto económicamente ayudará a la empresa y a medida que se vaya pasando el tiempo el beneficio también irá aumentando.

El proyecto finalizado ha tenido un coste final aproximadamente de 12351€ que respecto el presupuesto estimado en GEP se ha podido ahorrar 1099€. Por eso, se estima que el impacto económico es de una nota de 10/10.

Respecto a la vida útil del proyecto, no tiene ningún coste para estar en funcionamiento. Sin embargo, se puede considerar el mantenimiento que pueden llegar a costar un 30% del coste del proyecto en caso de que se necesiten actualizar muchas pruebas automatizadas. Cabe destacar que este coste sería el mínimo porque solo se necesitaría actualizar las pruebas automatizadas pero la configuración y la ejecución de las pruebas ya están hechas. Por ese motivo, se estima la vida útil con una nota de 9/10.

Por último, no se ha considerado que este proyecto tenga riesgos en el ámbito económico ya que el software creado es únicamente para el equipo de desarrollo del Módulo en Drupal. Sin embargo, se puede identificar un riesgo para un largo plazo que se daría cuando el módulo de promociones tenga una actualización que conlleve a cambiar una gran cantidad de sus páginas webs. En esta circunstancia el proyecto estaría en riesgo porque si las páginas se cambian este proyecto no tendría valor y no aportaría ningún beneficio. Como es un riesgo muy poco probable, se ha estimado una nota de 8/10 a la

14.4.3. Dimisión social

El desarrollo de este proyecto me ha apartado aprender muchos conocimientos y también poner en práctica lo que he ido aprendiendo durante la carrera. He podido aprender nuevas tecnologías como Selenium y Maven para realizar automatización de pruebas. Además, durante el proyecto he podido aprender cómo se trabaja en un equipo cuando se encuentra obstáculos que son difíciles de afrontar y cómo se toman decisiones cuando para poder finalizar el proyecto con éxito. Por tanto, la nota de dimensión referente al proyecto puesto en producción es de 10/10.

Por otra parte, el beneficiario principal es de este proyecto es el Cliente, en este caso, Damm ya que a largo plazo podrá ahorrar mucho tiempo y dinero. Además, también el equipo de *testing* que tendrán que no tendrá que pasar tiempo testeando la aplicación. El proyecto soluciona la pérdida de tiempo que se pasaba testeando. Además, este proyecto no es perjudicable a nadie. Por este motivo, se considera una nota del 10/10 para la vida útil del proyecto teniendo en cuenta el impacto social del proyecto.

Respecto a los riesgos, como este proyecto no tiene ninguna interacción con usuarios, no se considera ningún tipo de riesgo. Por tanto, se valora 10/10 de nota para los riesgos del proyecto.

Para concluir este apartado, en la siguiente tabla se presentan los resultados estimados de sostenibilidad del impacto ambiental, económico y social respecto al proyecto puesto en producción, vida útil y riesgos.

	Proyecto Puesto en Producción	Vida Útil	Riesgos	Total
Ambiental	9/10	9/10	9/10	27/30
Económico	10/10	9/10	8/10	27/30
Social	10/10	10/10	10/10	30/30
Total	29/10	28/10	27/10	84/90

Tabla 21: Matriz de Sostenibilidad del TFG

15. Conclusiones

En este último apartado del trabajo, se presentan las conclusiones finales que se ha podido llegar después de desarrollar el trabajo realizado durante estos meses. Además, en este apartado se explicarán las reflexiones que me han surgido durante el proyecto y las dificultades que me he encontrado durante el desarrollo del trabajo. Por otra parte, también se dedicará un apartado para explicar la integración de conocimientos que se han ayuda de las asignaturas cursadas durante la carrera. A continuación, se justificarán las competencias técnicas logras después de finalizar el trabajo final de grado. Para acabar, se presentará el futuro del proyecto y donde se espera llegar con este proyecto.

15.1. Reflexiones

En este subapartado de la memoria, se pretende poner las reflexiones y valoraciones sobre el proceso de desarrollo del proyecto realizado durante estos últimos cuatros meses. Además, también se expondrán las implicaciones tenidas en el este trabajo.

Realizar el trabajo final de grado en una empresa, con un ambiente real con equipo, ha sido una experiencia positiva en la que se ha podido aprender y adquirir experiencia profesional. Por este motivo, decidí realizar el TFG en la empresa, para poder tener experiencia laboral y así también ver todo el proceso de desarrollo de un proyecto, des de cómo se inicia hasta llegar a finalizarlo. Además, generalmente realizar un proyecto en una empresa implica que el proyecto sea para un cliente, como ha sido en este caso. Esto comporta a tener otros aspectos en cuenta: se ha de realizar el producto tal como quiere el cliente ya que él es el dueño del producto y pone los requisitos. De esta forma, he experimentado la realidad de cómo se trata con el cliente para poder desarrollar el proyecto. Gracias a este proyecto, podré empezar mi vida profesional como ingeniero de software teniendo experiencia laboral.

Por otro lado, si hablamos del tema de este trabajo que se titula “Testing Automatizado de Módulo de Promociones”, por un momento una persona podría llegar a pensar que se trata de verificar el funcionamiento de una aplicación, pero de forma automática y puede resultarle fácil. Sin embargo, la realidad es muy diferente. Si testear manualmente un funcionamiento de una aplicación se tarda 15 minutos, automatizar esto puede darse mucho tiempo dependiendo de la dificultad o el acceso que puede hacer a los datos de la aplicación. Por este motivo, he tenido que realizar algunas pruebas complicadas que han requerido bastante tiempo de investigación para poder finalizar la tarea.

Después de finalizar el proyecto se puede decir que no todo es automatizable. Durante el desarrollo de las pruebas me he encontrado con algunas pruebas que no se podían automatizar y han tenido que modificarlas para que puedan ser automatizables. Como estudiante de ingeniería informática entendía que en la actualidad nos encontramos con muchos procesos automatizados, pero aún no tenemos todos los recursos para poder automatizar todos los procesos. Sin embargo, en el futuro utilizando nuevas tecnologías nos permita automatizar más procesos que ahora.

Otro aspecto importante de este proyecto es que en el futuro llegará a ser muy beneficioso tanto económicamente como para agilizar el proceso de corrección de incidencias. Por ese motivo, para una empresa grande como Damm esto podría llegar a ahorrarle mucho dinero y tiempo, que podría invertir en otros aspectos y de esta forma seguir creciendo a nivel mundial.

Para este trabajo no ha sido necesaria una formación ya que tenía experiencia en el lenguaje de programación utilizado. Sin embargo, nunca había utilizado Selenium y esto ha implicado investigar mucho para poder aprender y utilizar en el desarrollo. Gracias a la dedicación e implicación se ha podido lograr y hacer un buen uso de la tecnología para completar el producto del cliente tal como esperaba.

Por otro lado, Si hablamos de objetivos puesto para alcanzar en este proyecto, se puede decir que se ha logrado. En primer lugar, se ha logrado un sistema que realiza un conjunto de pruebas de forma automática y genera un informa para dar detalle de las pruebas ejecutadas. Además, es eficiente y rápido ya que logra ejecutar unas 700 pruebas en tan solo 22 minutos. Cabe mencionar que este dato es obtenido con un ordenador, pero si se llega a ejecutar en un servidor el resultado es mucho más favorable. También se ha asegurado que sea fácil de comprender y también modificar. Finalmente, se ha conseguido ofrecer un sistema que ofrece una mejora para realizar el *testing* para el módulo de promociones.

Por último, me gustaría destacar que a pesar de la situación y de obstáculos inesperados tenidos durante este largo camino del TFG, valoró positivamente el hecho de finalizar este trabajo de forma exitosa. Esto ha sido posible gracias a la dedicación y ganas de acabar el proyecto con buenos resultados.

15.2. Limitaciones y dificultades

A continuación, en esta sección se detallan algunas limitaciones y dificultades que se han encontrado a lo largo del desarrollo del proyecto.

- **Utilización de métodos de Selenium**: La tecnología utilizada permite acceder de forma automática a un navegador y que realizar peticiones de comprobaciones visualizarles, de escritura, *clicks*, etc. Se ha encontrado una gran dificultad con el uso del *click* que proporciona esta herramienta. Este click de vez en cuando no funciona en los dispositivos móvil y tableta, cosa que no sé acaba de entender de su funcionamiento incorrecto. Sin embargo, después de investigar y buscar una solución alternativa al método *click* de Selenium se decidió utilizar código JavaScript para realizar el click a un elemento web de la página. De esta forma, se pudo solventar este problema.

- **Entorno de pruebas**: El entorno utilizado para realizar pruebas era bastante inestable ya que funcionaba sin la conexión de VPN y porque otros proyectos también estaban utilizando. Cabe destacar que no se puedo utilizar el entorno de pruebas con conexión a VPN de cliente Damm ya que durante el mes de noviembre tuvo un ciberataque a su red que afectó a su producción y a todos sus proyectos. El cliente Damm por seguridad desactivo el acceso a la VPN hasta no tener la red establece. Por ese motivo, este proyecto también fue afectado, pero gracias al equipo de desarrollo y la implicación de mi tutor entendían que este proyecto tenía una fecha límite. Por eso, me proporcionaron otro entorno de pruebas para que este pudiese finalizar el proyecto. Sin embargo, este entorno tenía limitaciones ya que no funcionaba tan bien como el otro entorno de pruebas.

- **Dificultad en la automatización:** Durante el desarrollo del proyecto, muchas veces se ha tenido que replantear una prueba porque no se podía realizar igual que una prueba manual. En estas circunstancias, después de intentar todo al final la única solución era modificar la prueba para que se puede realizar lo más parecido a la prueba manual. Esta decisión se ha tomado con todo el equipo y se ha solventado de esta manera.

15.3. Integración de conocimientos

En esta sección de la memoria se detallan los conocimientos adquiridos en cada asignatura que han sido de ayuda para el desarrollo de este proyecto.

Introducción a la Ingeniería del Software (IES): En esta asignatura se enseñó los conocimientos básicos de la ingeniería del software, se introdujo las diferentes fases por la que se pasa un proyecto de software hasta finalizar. Me han servido de ayuda los conocimientos adquiridos sobre cómo especificar el sistema y también los principios básicos que se enseñaron en esta asignatura.

Proyectos de Programación (PROP): En esta asignatura aprendí a crear una aplicación de escritorio en la que se realizaban todas fases de creación de software. Además, se estudió en profundidad aspectos relacionadas con la programación orientada a objetos. Estos conocimientos me han servido para realizar la implementación del proyecto.

Paralelismo (PAR): En esta asignatura aprendí los principios de la programación en paralelo. Es verdad que para este proyecto lo que he podido utilizar de esta asignatura es saber cómo funciona la programación y la ejecución en paralelo. Sin esta asignatura, probablemente para este proyecto hubiese tenido que aprender qué es la programación en paralelo y cómo funciona.

Ingeniería de Requisitos (ER): En esta asignatura se enseñó a análisis los requisitos de un sistema que se desea crear. Además, también se enseña que hay que tener en cuenta para el análisis de requisitos, como, por ejemplo, sistemas y tecnologías existentes que ya ofrecen una solución al problema que se desea solventar. De esta forma, los conocimientos adquiridos en ER me han sido, especialmente, para la documentación y la parte de análisis de requisitos.

Gestión de Proyectos de Software (GPS): El hecho de poder trabajar con la metodología *agile* para el desarrollo de este proyecto conociendo como se trabaja, es gracias a esta asignatura. Esta fue la primera asignatura donde aprendí y realice la pequeña practica haciendo uso de la metodología agile.

Arquitectura del Software (AS): Es esta la asignatura en la que pude aprender diseño de la arquitectura de software, así como los principios de diseño, patrones de diseño, etc. Además, en la parte práctica hice desarrollo guiado por pruebas (TDD) haciendo uso de los patrones de diseño. Todo lo aprendido AS me ha ayudado a diseñar e implementar código de forma ordenada, usable y mantenible. Si no hubiese hecho esta asignatura el proyecto, tendría un grado de dificultad muy alto.

Proyecto de Ingeniería del Software (PES): Esta es otra asignatura en la que se aprendió a desarrollar un proyecto sobre una aplicación móvil, en la que se integran conocimientos de la mayoría de las asignaturas de la especialidad de software. Gracias a esta asignatura sabía cómo tenía que elaborar la documentación del trabajo final de grado. Además, me sirvió de ayuda haber utilizado la metodología agile en esta asignatura porque gracias a esa experiencia para este proyecto con el equipo profesional se pude realizar una buena labor de metodología agile.

15.4. Justificación de las competencias

En este subapartado se explicará cómo se ha intentar lograr las competencias técnicas previstas realizar durante el desarrollo del proyecto. Finalmente, también se informará del resultado que se ha podido lograr de las competencias.

CES1.1: Desenvolupar, mantenir i avaluar sistemes i serveis software complexos i/o crítics. [Una mica]
Este proyecto consistía en realizar *testing automatizado* para un módulo en Drupal que contiene más de 20 páginas webs y mucha diversidad de contenido. Es decir, que es un sistema bastante complejo. Para poder proceder con la realización de pruebas automatizadas se ha tenido que evaluar el sistema para saber qué tipo de pruebas necesita y cómo enfocar estas para que cumplan los requisitos del cliente. Como solo se ha valorado y testeado una parte de un sistema complejo, por ese motivo se considera que esta competencia se ha logrado satisfactoriamente.

CES1.3: Identificar, avaluar i gestionar els riscos potencials associats a la construcció de software que es poguessin presentar. [Una mica]
El proyecto desarrollado tiene un riesgo durante y después de la construcción. Este riesgo es si se modifican las páginas webs para las que se han implementado pruebas automatizadas, estas dejarían de pasar y necesitaría actualizar o rehacer las pruebas. Para ello, se han implementado pruebas que sean fáciles de comprender y de mantener a un largo plazo. Por tanto, podemos decir que esta competencia se ha logrado con éxito.

CES1.7: Controlar la qualitat i dissenyar proves en la producció de software. [En profunditat]
En este proyecto, como se trataba de realizar pruebas automatizadas tenían que ser de calidad y mantenibles. Para completar esta competencia, desde el principio se decidió que, para controlar la calidad y mantenibilidad de las pruebas, cada código que se finalice tenía que ser aprobado por cada miembro del equipo y de esta forma controlábamos que el código final cumpla los requisitos de calidad y mantenibilidad. Cada miembro cuando validaba tenía en cuenta que no haya código duplicado, se cumplía los principios SOLID, los nombres de los métodos era correcto, el diseño del código no tenía muchas dependencias y también se valoraba la complejidad del código. Estos eran los principales aspectos que cada miembro del equipo valoraba en el *pull request*. Además, también se ha realizado pruebas en la producción de algunas páginas webs. Por tanto, se puede afirmar que se ha logrado satisfactoriamente esta competencia.

CES2.1: Definir i gestionar els requisits d'un sistema software. [Bastant]
Antes de la fase de implementación y de la especificación del proyecto, se ha definido los requisitos del sistema de *testing* a crear. Estos requisitos han sido definidos teniendo en cuenta los requisitos del cliente que solicita la automatización de pruebas. Este análisis ha ayudado a realizar la especificación del sistema a testear y diseñar una arquitectura para poder crear las pruebas automatizadas. Considerando que se ha realizado esta competencia con éxito, por tanto, se da como competencia lograda.

15.5. Futuro del proyecto

Después de ver los resultados de este proyecto, la empresa Damm ha decidido ampliar el proyecto con el NTT Data y ha decidido poner más recursos para automatizar más procesos del módulo de promociones. Se puede decir que antes este proyecto no era tan prioritario ya que se quería valorar los resultados y después tomar una decisión. Ahora, el objetivo ha pasado a ser algo prioritario y necesario no solo por su beneficio económico sino también de tiempo y de poder agilizar la corrección de las incidencias del módulo de promociones.

En la ampliación del proyecto, se espera tener un entorno propio únicamente para el entorno de pruebas y nuevos requerimientos. La planificación consiste en realizar pruebas para otras partes del módulo en Drupal. De esta forma cubrir una gran parte para poder realizarla de forma automática.

Para la ejecución de las pruebas, entre el cliente Damm y la empresa NTT Data se ha decidido que después ejecutar las pruebas en el nuevo entorno de pruebas, se preparará la ejecución automática programada. Es decir, haciendo uso de una tecnología (por ahora no se ha concreado), programar la ejecución de las pruebas para ir validando un par de veces a la semana y así tener controlado el funcionamiento del entorno y las páginas webs. Esto significa no habrá la necesidad de ejecutar ni un comando ni hacer un *click* para probar los test automatizados.

En resumen, se espera realizar un buen trabajo para poder llegar a cumplir los objetivos. Además, ahora que hay más recursos en el proyecto, los resultados finales serán muy beneficiosos para el cliente.

16. Referencias

[1] «Everis an NTT DATA Company,» [En línea]. Available: https://www.everis.com/spain/es/home-spain. [Último acceso: 24 9 2021].

[2] «Damm Corporate,» [En línea]. Available: https://www.dammcorporate.com/es/sobre-damm. [Último acceso: 15 10 2021].

[3] «Getting Started with Website Test Automation,» [En línea]. Available: https://www.browserstack.com/guide/web-automation. [Último acceso: 27 9 2021].

[4] «Drupal,» [En línea]. Available: https://www.drupal.org/. [Último acceso: 12 1 2022].

[5] «Landing Page,» [En línea]. Available: https://en.ryte.com/wiki/Landing_Page. [Último acceso: 30 12 2021].

[6] «The 5 best automation testing tools for web applications that you could use in 2020,» [En línea]. Available: https://medium.com/@OPTASY.com/the-5-best-automation-testing-tools-for-web-applications-that-you-could-use-in-2020-powerful-and-23135826a569. [Último acceso: 26 9 2021].

[7] «Selenium,» [En línea]. Available: https://www.selenium.dev/documentation. [Último acceso: 27 9 2021].

[8] «Katalon Studio,» [En línea]. Available: https://www.katalon.com/. [Último acceso: 27 9 2021].

[9] «Watir,» [En línea]. Available: https://en.wikipedia.org/wiki/Watir. [Último acceso: 27 9 2021].

[10] «TestComplete,» [En línea]. Available: https://smartbear.com/product/testcomplete/overview/. [Último acceso: 27 9 2021].

[11] «ServiceNow,» [En línea]. Available: https://www.servicenow.es. [Último acceso: 18 10 2021].

[12] «Glassdor,» [En línea]. Available: https://www.glassdoor.es/. [Último acceso: 10 10 2021].

[13] «Idealista,» [En línea]. Available: https://www.idealista.com. [Último acceso: 10 10 2021].

[14] «Drawio,» [En línea]. Available: https://app.diagrams.net/. [Último acceso: 30 10 2021].

[15] «Domain Driven Design,» [En línea]. Available: https://en.wikipedia.org/wiki/Domain-driven_design. [Último acceso: 2021 11 3].

[16] «Introducción a Domain Driven Design (DDD),» [En línea]. Available: https://picodotdev.github.io/blog-bitix/2021/02/introduccion-a-ddd-y-arquitectura-hexagonal-con-un-ejemplo-de-aplicacion-en-java/. [Último acceso: 8 11 2021].

[17] «Principios Solid,» [En línea]. Available: https://devexperto.com/principios-solid/. [Último acceso: 2 11 2021].

[18] «The Three A's of Unit Testing,» [En línea]. Available: https://blog.devgenius.io/the-three-as-of-unit-testing-3b8b4bf0d087. [Último acceso: 19 11 2021].

[19] «Pattern AAA,» [En línea]. Available: https://java-design-patterns.com/patterns/arrange-act-assert/. [Último acceso: 22 11 2021].

[20] «¿Qué es Gherkin?,» [En línea]. Available: https://profile.es/blog/que-es-gherkin/. [Último acceso: 13 12 2021].

[21] «JAVA,» [En línea]. Available: https://www.java.com/es/. [Último acceso: 2 10 2021].

[22] «HTML,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTML. [Último acceso: 5 11 2021].

[23] «XPath,» [En línea]. Available: https://www.w3schools.com/xml/xpath_intro.asp. [Último acceso: 10 1 2022].

[24] «JavaScript,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript. [Último acceso: 10 1 2022].

[25] «TestNG Doc,» [En línea]. Available: https://testng.org/doc/. [Último acceso: 12 1 2022].

[26] «Maven,» [En línea]. Available: https://maven.apache.org/. [Último acceso: 12 1 2021].

[27] «Introducing JSON,» [En línea]. Available: https://www.json.org/json-en.html. [Último acceso: 12 1 2022].

[28] «Planning spoker online,» [En línea]. Available: https://planningpokeronline.com/. [Último acceso: 22 12 2021].

[29] «Timboretro,» [En línea]. Available: https://app.timboretro.com/. [Último acceso: 22 12 2021].

[30] «Github,» [En línea]. Available: https://github.com/. [Último acceso: 18 10 2021].

[31] «Drupal,» [En línea]. Available: https://www.drupal.org/. [Último acceso: 12 1 2022].