Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería de Software

Desarrollo de aplicación web utilizando ASP.NET para la gestión de reglas y sanciones en F1

Alumno: Jorge Rebé Martín

Tutora: Yania Crespo González-Carvajal

A los que han estado y sido

Agradecimientos

A mi familia y a mis amigos por ayudarme a seguir cuerdo durante estos últimos cuatro años.

A Yania por enseñar.

A todas las personas que han provocado en mí un deseo por seguir mejorando en cualquier aspecto.

A mí por seguir y no ceder al tercer intento.

Resumen

El objetivo de este proyecto es desarrollar una aplicación web para gestionar los reglamentos y las sanciones dentro de la Fórmula 1, así como el transcurso de las temporadas, competiciones y sesiones en las que se puedan aplicar dichas sanciones.

La aplicación ha sido desarrollada utilizando ASP.NET Core MVC, siguiendo Scrum como marco de trabajo ágil y utilizando DDD y BDD como enfoques de desarrollo de software. Se ha construido sobre una arquitectura Clean, aplicando buenas prácticas en el desarrollo de software de calidad y siguiendo principios y patrones de diseño.

Abstract

The aim of this project is to develop a web application to manage the regulations and penalties in Formula One, as well as the course of the seasons, competitions, and sessions in which those penalties can be imposed.

The app has been developed using ASP.NET Core MVC, with Scrum as agile framework and using DDD and BDD as software development approaches. It's been built using a Clean architecture, applying good practices in the development of quality software following design principles and patterns.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

La Fórmula 1 (en adelante "F1") es una categoría deportiva de automovilismo. Actualmente, en cada temporada compiten 20 pilotos repartidos en 10 equipos. En ocasiones el comportamiento en la pista de los pilotos no es el adecuado, y es entonces cuando el reglamento deportivo de la F1 [1] y el código deportivo internacional [2], publicados por la Federación Internacional del Automovilismo (en adelante "FIA"), ha de ser aplicado. Estos reglamentos contienen una gran cantidad de artículos donde se detallan las normas que se han de seguir, tanto fuera como dentro de la pista, y las sanciones a aplicar en caso de que se incumplan. Estas sanciones pueden tener consecuencias económicas o deportivas.

En la propia página web de la FIA se publican documentos informativos donde se indican las sanciones a los pilotos que ocurren en cada evento [3], pero no existe un sitio dedicado a poder consultar las sanciones que tiene un piloto, que han ocurrido en un evento o similares. Para realizar estas consultas habría que visualizar de manera manual centenares de documentos en un tiempo prohibitivo.

Por esto nace la idea de desarrollar una aplicación web que gestione las temporadas de F1 y lo relacionado con ellas: su reglamento, el transcurso de las competiciones, competidores, participaciones de pilotos e incidentes en cada sesión de las competiciones.

Este proyecto se desarrolla en el marco de la asignatura 'Trabajo de Fin de Grado' del Grado de Ingeniería Informática, en la mención de Ingeniería de Software [4]. Como proyecto de software que es, pasará por las fases de planificación, análisis, diseño, implementación de funcionalidad y pruebas, y despliegue.

1.2. Motivación

La idea de este TFG parte del estudiante y surge tras detectar la necesidad de que exista un sitio donde poder consultar sin esfuerzo el histórico de penalizaciones y reglamentos en Fórmula 1 a lo largo de cada temporada.

Se busca crear una plataforma única donde los comisarios de los Grandes Premios puedan imponer sanciones a los pilotos, los aficionados puedan consultar el histórico de los incidentes ocurridos, y los equipos puedan inscribir a los pilotos en los eventos correspondientes.

Por otro lado, el estudiante también tiene la motivación de aprender a desarrollar una aplicación web en ASP.NET, con el objetivo de mejorar en la programación en C# y en el uso de la plataforma .NET.

Adicionalmente, se busca mejorar con respecto a la toma de decisiones en la arquitectura de la aplicación y su despliegue, eligiendo la mejor opción teniendo en cuenta el presupuesto disponible y las necesidades de la aplicación.

1.3. Objetivos

Al tratarse de un proyecto académico, podemos distinguir dos tipos de objetivos: 1) objetivos de desarrollo y 2) objetivos personales.

1.3.1. Objetivos de desarrollo

Los objetivos de desarrollo son aquellas funcionalidades que el sistema debería tener, expresadas de manera general.

- Gestionar los reglamentos de F1.
- Gestionar los pilotos, equipos y la pertenencia de los pilotos a un equipo dentro de la categoría.
- Gestionar las temporadas, sus competiciones, el reglamento y equipos participantes.
- Gestionar las participaciones de los pilotos en cada competición.
- Gestionar el transcurso de las competiciones y los incidentes ocurridos en cada sesión.

1.3.2. Objetivos personales

Por otro lado, estarían los objetivos personales de aprendizaje y formación personal que se desean alcanzar.

- Aprender a utilizar ASP.NET, adoptando las mejores prácticas posibles en el desarrollo de una aplicación web en la que se usa esa tecnología.
- Mejorar las capacidades de programación en C#.
- Tomar decisiones de diseño de acuerdo a la tecnología y tiempo disponible.
- Realizar tests unitarios y aprender a especificar, diseñar e implementar tests automatizados de aceptación.
- Elegir una plataforma adecuada para el despliegue de la aplicación.

1.4. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 1 - Introducción: En este capítulo se explicará la idea del proyecto, el qué se va a desarrollar y los objetivos que se pretenden cumplir con el proyecto.

Capítulo 2 - Requisitos y planificación: En este capítulo se explica el marco de trabajo utilizado en el desarrollo de este proyecto, así como los enfoques de desarrollo de software usados. También se expondrán los planes de riesgos, de presupuestos y los requisitos de la aplicación.

Capítulo 3 - Análisis: En este capítulo se mostrarán diversas tareas de análisis llevas a cabo a lo largo de todo el proyecto, a partir de los requisitos recogidos en el capítulo 2.

Capítulo 4 - Tecnologías utilizadas: En este capítulo se explican las tecnologías utilizadas para el desarrollo de todo el proyecto.

Capítulo 5 - Diseño: En este capítulo se explican las decisiones de diseño, así como diversos diagramas que representan el sistema desarrollado como consecuencia de esas decisiones.

Capítulo 6 - Implementación y pruebas: En este capítulo se explicarán detalles relevantes sobre la implementación de la aplicación, así como una explicación de las pruebas realizadas para asegurar su correcto funcionamiento.

Capítulo 7 - Seguimiento del proyecto: En este capítulo se hará una descripción detallada de lo realizado en el proyecto explicando planificación, incidencias sucedidas y respuesta a esas incidencias.

Capítulo 8 - Conclusiones y líneas futuras: En este capítulo se expondrán las conclusiones obtenidas tras finalizar el proyecto y se explicarán algunas ideas para futuros desarrollos del proyecto.

Anexo A - Manuales: Incluye manuales de despliegue en local, en Azure, de mantenimiento y de uso.

Anexo B - Resumen de enlaces adicionales: Incluye enlaces de interés sobre el proyecto, como el repositorio de código y el enlace al sistema desplegado.

Capítulo 2

Requisitos y Planificación

El manifiesto para el desarrollo de software Agile [5] expone los siguientes cuatro valores en los que se basa la mentalidad Agile:

- Individuos e interacciones por encima de procesos y herramientas.
- Software funcional por encima de documentación completa.
- Colaboración con el cliente por encima de negociación contractual.
- Responder al cambio por encima de seguir un plan.

Estos cuatro valores junto con los doce principios del manifiesto se unen perfectamente con el proyecto a desarrollar en este TFG, ya que los requisitos no están perfectamente definidos y serán cambiantes durante el desarrollo del mismo.

2.1. Scrum

Scrum [6] es una implementación de la mentalidad ágil, un marco de referencia. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y controlar el riesgo. En resumen, Scrum requiere un *Scrum Master* (ver Subsección 2.1.1) que promueva un ambiente donde:

1. Un *Product Owner* ordena el trabajo para un problema complejo en un *Product Backlog* (ver Subsecciones 2.1.1 y 2.1.3).
2. El equipo de desarrollo transforma una selección del trabajo en un Incremento de valor durante un Sprint (ver Subsecciones 2.1.1, 2.1.2 y 2.1.3).

3. El equipo de desarrollo y los participantes inspeccionan los resultados y ajustan para el siguiente Sprint (ver Subsecciones 2.1.1, 2.1.2 y 2.1.3).

4. Se vuelve al paso 1.

Scrum combina cuatro eventos formales para inspección y adaptación dentro de otro evento contenedor, el Sprint. Estos eventos implementan los pilares de Scrum de **transparencia**, **inspección** y **adaptación**:

- Transparencia: El trabajo debe ser visible tanto para los que lo realizan como para los que lo reciben, dado que las decisiones importantes en Scrum se toman en base al estado percibido de sus tres artefactos formales. Los artefactos con poca transparencia pueden provocar decisiones que reducen el valor y aumentan el riesgo.
- Inspección: Los artefactos de Scrum y el progreso hacia los objetivos deben ser inspeccionados frecuentemente para detectar problemas o variaciones no deseadas. Para facilitar esta inspección existen los cinco eventos de Scrum. La inspección permite que haya adaptación, sin la cual la inspección carece de sentido.
- Adaptación: Si alguna parte del proceso se desvía de los límites aceptables, entonces hay que ajustarlo tan pronto como sea posible para evitar una desviación mayor.

2.1.1. Roles

Un equipo de Scrum está formado por un pequeño grupo de personas, formado por un *Scrum Master*, un *Product Owner* y los desarrolladores. Todos los miembros del equipo están centrados en un objetivo a la vez: el objetivo del producto.

Un equipo Scrum se autogestiona, decide internamente quién hace qué, cuándo y cómo. Generalmente está formado por diez personas o menos. Todo el equipo es responsable de crear un incremento que aporte valor y sea útil en cada Sprint.

A continuación se explican cada uno de los tres roles dentro de un equipo Scrum.

- **Product Owner**: Es el responsable de maximizar el valor del producto resultante del trabajo del equipo Scrum. Sus decisiones se reflejan en el contenido y orden de los elementos del *Product Backlog*, artefacto que se describirá posteriormente (Sección 2.1.3).
- **Scrum Master**: Es el responsable de impulsar todas las prácticas de Scrum dentro del equipo y de la eficacia del mismo. Es un líder que sirve al equipo Scrum y al resto de la organización de diversas maneras.
- **Desarrolladores**: Son las personas dentro del equipo comprometidas con la creación de cualquier aspecto de un Incremento (artefacto que será explicado en la sección 2.1.3) que aporte valor cada Sprint.

2.1.2. Eventos

Los eventos de Scrum permiten cumplir con los tres pilares descritos anteriormente: es una oportunidad para **1)** inspeccionar y **2)** adaptar los artefactos, con la **3)** transparencia requerida. Los eventos de Scrum también tienen como objetivo crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Estos son los eventos de Scrum:

- **Sprint**: Es un contenedor para el resto de eventos. Tienen una duración fija de un mes o menos, y un Sprint nuevo comienza inmediatamente después de la finalización del anterior.

- **Sprint Planning**: Es el evento que da inicio al Sprint, en el que se planifica el trabajo a realizar. En él participan todos los componentes del equipo Scrum, y se tratan los siguientes temas: **1)** El *Product Owner* propone cómo el producto podría incrementar su valor y utilidad en el Sprint actual, y todo el equipo colabora para definir un objetivo del Sprint (*Sprint Goal*). **2)** los desarrolladores eligen qué elementos del *Product Backlog* (Subsección 2.1.3) se van a incluir en el Sprint y, **3)** para cada elemento elegido, los desarrolladores estiman el esfuerzo que necesitarán para crear un incremento que cumple con la *Definition of Done* (Definición de Hecho). Para esto último, los desarrolladores descomponen los elementos del *Product Backlog* en tareas de una duración de un día o menos. El *Sprint Goal*, los elementos del *Product Backlog* elegidos para el Sprint y el plan para entregarlos forman el *Sprint Backlog*.

- **Daily Scrum**: Es una reunión diaria con una duración de aproximadamente 15 minutos para los desarrolladores del equipo Scrum. En ella, cada miembro del equipo de desarrollo informa de su progreso desde la última *Daily Scrum* y de su plan de trabajo hasta la siguiente.

- **Sprint Review**: Su objetivo es inspeccionar el resultado del Sprint y determinar las adaptaciones que hay que realizar. En esta reunión están presentes los clientes o sus representantes, se revisa lo logrado durante el Sprint y se habla sobre qué hacer en el siguiente. Si es necesario, se modifica el *Product Backlog* (Subsección 2.1.3).

- **Sprint Retrospective**: Es la reunión con la que termina el Sprint, en la que el equipo Scrum evalúa lo que ha ido bien durante el Sprint, problemas encontrados, y cómo se resolvieron (o no) esos problemas.

2.1.3. Artefactos

Los artefactos Scrum representan trabajo o valor, y están diseñados para maximizar la transparencia de la información clave. Cada artefacto contiene un compromiso contra el que se puede medir el progreso en cada uno de los artefactos. Para el artefacto *Product Backlog*, el compromiso es el *Product Goal*; para el *Sprint Backlog*, el compromiso es el *Sprint Goal*; y para el Incremento, el compromiso es la *Definition of Done*.

- **Product Backlog**: Es una lista ordenada por prioridad de lo que se necesita para mejorar el producto. Los elementos del *Product Backlog* que se pueden completar en un Sprint forman parte de los que se seleccionan durante el *Sprint Planning*.

  Product Goal: Define un estado futuro del producto que sirve como objetivo para el equipo Scrum. El *Product Goal* está en el *Product Backlog*, y el resto de componentes del mismo emergen para definir qué completará el *Product Goal*.

- **Sprint Backlog**: Es un plan hecho por y para los desarrolladores, una imagen que los desarrolladores planean completar para conseguir el *Sprint Goal*. Está compuesto del *Sprint Goal* (el por qué del Sprint), los elementos del *Product Backlog* seleccionados (qué se realizará), y el plan para la entrega del incremento (el cómo).

  Sprint Goal: Es el objetivo a completar en el Sprint, un compromiso hecho por los desarrolladores. Sin embargo, es flexible en cuanto al trabajo necesario para conseguirlo. Si el trabajo acaba siendo diferente del esperado, los desarrolladores colaboran con el *Product Owner* para negociar el alcance del *Sprint Backlog*, pero sin afectar al *Sprint Goal*.

- **Incremento**: Es un paso concreto hacia el Product Goal. Es posible que haya varios incrementos dentro del mismo Sprint, y el total de los incrementos es presentado durante la *Sprint Review*. Sin embargo, un incremento puede ser entregado a los stakeholders antes del final del Sprint. El trabajo realizado no puede ser considerado parte de un incremento hasta que no alcance la *Definition of Done*.

  Definition of Done: Es una descripción formal del estado del incremento cuando satisface las medidas de calidad requeridas por el producto. En el momento en el que un elemento del *Product Backlog* alcanza la *Definition of Done* (o Definición de Hecho), aparece un incremento.

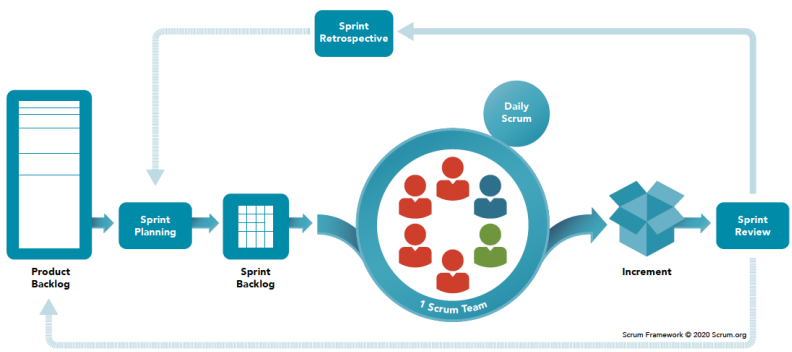


Figura 2.1: Eventos y artefactos en Scrum [7]

2.2. Adaptación de Scrum al proyecto

El marco de trabajo Scrum no está completo, define sólo las partes requeridas para implementar la teoría Scrum. Esto hace que Scrum haya sido elegido para este proyecto, debido a varias razones: primero, la idea del proyecto es del estudiante, y los requisitos del mismo serán cambiantes y no van a estar completamente definidos desde el principio. También el seguimiento de la tutora será semanal, permitiendo inspeccionar el progreso y adaptarse para mejorar según se avance en el proyecto.

Debido a las características especiales del proyecto, como la limitación del personal, se ha tenido que realizar una adaptación de Scrum.

En cuanto a roles, en este proyecto sólo habrá dos miembros involucrados en el proyecto: el estudiante, autor del TFG, y la tutora. El estudiante será a la vez *Product Owner* y equipo de desarrollo, mientras que la tutora cumplirá el papel del *Scrum Master*.

En cuanto a los eventos, los sprints tendrán una duración de dos semanas. Los eventos *Sprint Planning*, *Sprint Review* y *Sprint Retrospective* tendrán lugar cada dos semanas, el mismo día que finaliza el sprint y comienza el siguiente. Las *Daily Scrum* se han convertido en *Weekly Scrum*, ya que estas reuniones serán semanales. El día elegido para estas reuniones ha sido el martes.

2.3. Stakeholders

Dentro de este proyecto, podemos identificar tres grupos de usuarios directos del sistema que se va a desarrollar.

- Por un lado, tenemos a los **comisarios**. Son las personas encargadas de crear los reglamentos de la F1, del transcurso de las temporadas y de anotar los incidentes y sus sanciones (si las hubiera) a los pilotos.
- Los **jefes de equipo** son los responsables de declarar los pilotos que forman parte de su equipo que van a participar en una competición.
- Los **aficionados** son las personas que podrán visualizar la información de las temporadas y los incidentes.

2.4. Enfoques de desarrollo de Software

A continuación se explican los dos enfoques de desarrollo de software que se usarán en este proyecto:

2.4.1. Domain Driven Design

Domain Driven Design (en adelante 'DDD') [8] es un enfoque de desarrollo de software que centra el desarrollo en un dominio que tiene un gran conocimiento de las reglas del negocio que se está modelando. Es particularmente útil para dominios complejos, y a su alrededor existe un amplio catálogo de patrones, algunos de los cuales se usarán en este proyecto (ver Sección 5.5).

2.4.2. Behaviour Driven Development

Behaviour Driven Development (en adelante 'BDD') [9] es un enfoque de desarrollo de software que busca la colaboración entre los desarrolladores y los clientes para que exista una mejor comprensión de cómo se debe comportar el sistema a desarrollar.

Esencialmente, BDD es un proceso iterativo de tres pasos:

1. A partir de una historia de usuario se exploran ejemplos completos de la funcionalidad que tiene que tener el sistema para que se considere que está hecha.
2. Después, documentar esos ejemplos de una manera que puedan ser automatizados (Gherkin).
3. Finalmente, implementar el comportamiento descrito por el ejemplo documentado, comenzando por un test automatizado que guíe el desarrollo del código.

2.5. Riesgos

Un riesgo [10] es un suceso relacionado con algo que puede ocurrir en el futuro y que involucra una causa y un efecto. Por un lado, tiene que referirse a algo que ocurriría en el futuro porque es incierto, y si no lo fuera no habría riesgo. Por otro lado, un riesgo tiene una causa que hace que ocurra y un efecto negativo en el desarrollo del proyecto.

Debido a que todo proyecto existen riesgos, es esencial desarrollar un plan de riesgos en el que se sigan los siguientes pasos:

1. Identificación de riesgos
2. Análisis y priorización de riesgos
3. Planificación de riesgos: Una vez se han identificado y priorizado los riesgos, hay que decidir cómo se lidiará con ellos. Se elegirá una de las siguientes acciones:
    a) Aceptación: Esta es la opción de no hacer nada, la elegida para los riesgos cuyo daño que podrían infligir sería menor que el coste de la reducción de la probabilidad de que el riesgo ocurra.

b) Evitación: Consiste en evitar aquellas acciones que aumentarían la probabilidad de que un cierto riesgo ocurra.

c) Reducción: Aquí se decide seguir hacia delante a pesar de los riesgos, pero tomando algunas acciones que reducen la probabilidad de que ocurran.

d) Mitigación: Consiste en realizar acciones antes de que suceda el riesgo para reducir su impacto.

e) Plan de contingencia: Es una acción planeada que se lleva a cabo una vez el riesgo se ha materializado.

f) Derivación: el riesgo es transferido a otra persona u organización.

4. Monitorización de riesgos: Una vez se ha decidido que se hará con cada riesgo, hay que monitorizar el estado en el que se encuentran para poder tomar las acciones pertinentes en el caso de que se materialice.

En el caso de este proyecto, se han identificado los riesgos, asignado probabilidad e impacto. Se ha calculado nivel de cada riesgo en base a la matriz de la Figura 2.2, y se han asignado acciones de mitigación y un plan de contingencia.

PROB/IMP	BAJA	MEDIA	ALTA
BAJO	BAJO	BAJO	MEDIO
MEDIO	BAJO	MEDIO	ALTO
ALTO	MEDIO	ALTO	ALTO

Figura 2.2: Matriz de nivel de riesgo por probabilidad e impacto. (Basada en [11]).

Se han clasificado los riesgos identificados en dos grupos. Por un lado, riesgos generales y comunes a prácticamente todos los proyectos de desarrollo de software. Por otro, riesgos particulares que podrían suceder debido a las características de este proyecto.

2.5.1. Riesgos generales

Desde la Tabla 2.1 a la Tabla 2.7 se muestran los riesgos generales que podrían afectar al proyecto.

Riesgo RSK01					
Nombre	Ausencia del estudiante por enfermedad o lesión				
Descripción	El estudiante cae enfermo por una cantidad indeterminada de tiempo, durante el cual es imposible continuar con el desarrollo del proyecto				
Probabilidad	BAJA	Impacto	ALTO	Nivel de riesgo	MEDIO
Reducción y mitigación	1. Evitar realizar actividades con alta probabilidad de acabar en lesión. 2. Llevar una vida saludable dentro de lo posible, realizando actividad física, descansando lo necesario y manteniendo una alimentación correcta para evitar caer enfermo (tanto mental como físicamente) y, en el caso de caer enfermo, que el tiempo de recuperación sea el menor posible				
Plan de contingencia	1. Hacer uso de los sprints de refuerzo si fuera necesario. 2. Intensificar el trabajo durante los días siguientes a la ausencia para tratar de no afectar a la planificación inicial. 3. Desplazar tareas a los sprints siguientes si no se pueden realizar en el sprint en el que ha ocurrido la ausencia.				

Tabla 2.1: Riesgo 01

Riesgo RSK02					
Nombre	Desconocimiento de la tecnología de desarrollo				
Descripción	Retrasos en el desarrollo del proyecto debidos al desconocimiento de la tecnología utilizada para el desarrollo del proyecto				
Probabilidad	BAJA	Impacto	BAJO	Nivel de riesgo	BAJO
Reducción y mitigación	1. Familiarizarse con la tecnología con la que se va a trabajar antes de comenzar el proyecto.				
Plan de contingencia	1. Consultar la documentación de la tecnología para buscar cómo solucionar los problemas que ocurren. 2. Realizar búsquedas en Stack Overflow, en los foros de Microsoft o consultar tutoriales que resuelvan el problema ocurrido.				

Tabla 2.2: Riesgo 02

2.5.2. Riesgos particulares

Desde la Tabla 2.8 a la Tabla 2.9 se muestran los riesgos particulares que podrían afectar al proyecto.

Riesgo RSK03					
Nombre	Subestimación del esfuerzo de algunas historias de usuario				
Descripción	Retrasos en el desarrollo del proyecto debido a la subestimación del esfuerzo que requerirán algunas historias de usuario				
Probabilidad	BAJA	Impacto	BAJO	Nivel de riesgo	BAJO
Reducción y mitigación	1. Tratar de ser conservador en la estimación de las primeras historias de usuario. 2. Utilizar la experiencia para realizar estimaciones realistas. 3. Desarrollar sólo la funcionalidad estrictamente necesaria de las historias de usuario que podrían causar un retraso en el desarrollo.				
Plan de contingencia	1. Si una historia no puede ser completada en un sprint, moverla al siguiente. 2. Si fuera necesario, hacer uso de los dos sprints de refuerzo.				

Tabla 2.3: Riesgo 03

Riesgo RSK04					
Nombre	Problemas técnicos en el equipo de trabajo				
Descripción	Incapacidad de seguir desarrollando el proyecto debido a problemas técnicos que han dejado el equipo de trabajo inutilizado				
Probabilidad	BAJA	Impacto	ALTO	Nivel de riesgo	BAJO
Reducción y mitigación	1. Depositar el código en un repositorio online, utilizando un control de versiones para poder trabajar en otro equipo. 2. Trabajar en la memoria sobre un documento online.				
Plan de contingencia	1. Coger prestado otro equipo para poder retomar el trabajo en el proyecto lo antes posible. 2. Asumir el gasto de las reparaciones del equipo o, si no fuera posible repararlo, adquirir otro nuevo.				

Tabla 2.4: Riesgo 04

Riesgo RSK05					
Nombre	Pérdida de comunicación con la tutora.				
Descripción	La plataforma de comunicación se cae y la comunicación con la tutora se interrumpe durante un tiempo indeterminado.				
Probabilidad	BAJA	Impacto	BAJO	Nivel de riesgo	BAJO
Reducción y mitigación	1. Tener planeadas vías de comunicación alternativas, como el correo electrónico, para poder mantener la comunicación.				
Plan de contingencia	1. Notificar el problema ocurrido para que se pueda solucionar lo antes posible. 2. Seguir trabajando en el proyecto a pesar de estos problemas.				

Tabla 2.5: Riesgo 05

Riesgo RSK06					
Nombre	Tiempo empleado en el sprint insuficiente.				
Descripción	El tiempo empleado en el sprint es insuficiente y no corresponde con lo planeado en la reunión de planificación, debido a otros eventos académicos (prácticas en empresa, preparación de exámenes o entregas, etc.).				
Probabilidad	MEDIA	Impacto	MEDIO	Nivel de riesgo	MEDIO
Reducción y mitigación	1. Tratar de prever estos eventos en la planificación y reducir la carga de trabajo para ese sprint.				
Plan de contingencia	1. Mover las tareas que se planificaron pero no se acabaron al siguiente sprint. 2. Tratar de recuperar el tiempo en los siguientes sprints en los que haya más tiempo disponible.				

Tabla 2.6: Riesgo 06

Riesgo RSK07					
Nombre	Mejoras excesivas en la implementación de los requisitos.				
Descripción	Demasiado tiempo implementando algunas partes de algunas historias de usuario, que no son necesarias para obtener un producto mínimo viable, provocando retrasos en el desarrollo del proyecto.				
Probabilidad	BAJA	Impacto	ALTO	Nivel de riesgo	MEDIO
Reducción y mitigación	1. Definir muy bien la implementación a realizar en cada historia de usuario para obtener un producto mínimo viable.				
Plan de contingencia	1. Si es necesario, utilizar el sprint extra para recuperar el tiempo perdido.				

Tabla 2.7: Riesgo 07

Riesgo RSK08					
Nombre	Deshabilitación de los servicios de Azure por finalización del crédito.				
Descripción	Los 100$ de crédito de Azure se agotan y por tanto la suscripción de Azure se inhabilita [12], imposibilitando tener la aplicación desplegada.				
Probabilidad	BAJA	Impacto	MEDIO	Nivel de riesgo	BAJO
Reducción y mitigación	1. Vigilar los costes mensuales de Azure para evitar que se disparen los gastos y se agote el crédito de 100$.				
Plan de contingencia	1. Actualizar la cuenta de Azure y pagar lo necesario para tener la aplicación desplegada. 2. Buscar una alternativa gratuita a Azure donde poder tener desplegada la aplicación contenerizada y tener una base de datos SQL.				

Tabla 2.8: Riesgo 08

2.6. Planificación y calendarización

Según la guía docente [4], el proyecto se corresponde con 12 ECTS, es decir, una duración de 300 horas. Se ha establecido un plan inicial de 8 sprints, cada uno con una duración de dos

Riesgo RSK09					
Nombre	Imposibilidad de realizar despliegue continuo automatizado				
Descripción	Incapacidad de realizar despliegue continuo mediante una pipeline de Azure DevOps por falta de acceso a Azure AD.				
Probabilidad	BAJA	Impacto	ALTO	Nivel de riesgo	BAJO
Reducción y mitigación	1. Ponerse en contacto con los gestores de las licencias 'Azure Students' de la universidad para que garanticen el acceso.				
Plan de contingencia	1. Búsqueda de plataformas alternativas donde realizar el despliegue continuo y que se pueda materializar mediante una pipeline en Azure DevOps. 2. Renunciar al despliegue continuo, y desplegar la aplicación mediante Visual Studio.				

Tabla 2.9: Riesgo 09

semanas. Se espera también que la cantidad de horas utilizadas por semana en el proyecto sea aproximadamente de 20, haciendo un total de **40 horas por Sprint**. Este número de horas por semana y por sprint podrá variar dependiendo de eventos académicos o la manifestación de alguno de los riesgos recogidos en la sección anterior que limiten el tiempo que se pueda emplear en el proyecto.

La cantidad de trabajo a realizar en cada Sprint se decidirá en el Sprint Planning, evento en el que se elegirán las historias de usuario del Product Backlog que se van a desarrollar durante ese Sprint. Cada historia de usuario llevará asociado un tamaño, correspondiente con el esfuerzo que supondrá realizarla. Inicialmente se ha establecido una *velocity* de 20 puntos de historia de usuario por Sprint. La *velocity* [13] es una métrica ágil que indica la cantidad de esfuerzo que puede realizar el equipo Scrum durante un sprint, medido por puntos de historia de usuario.

Para la estimación del esfuerzo requerido para completar una historia de usuario, se utilizarán los puntos de historia. Esos puntos se asignarán según la siguiente secuencia Fibonacci: 1, 2, 3, 5, 8, 13. Es decir, la estimación más baja del esfuerzo de una historia de usuario sería de 1 punto, y la más alta 13. Sin embargo, se considerará que una historia de usuario con 13 puntos sería demasiado grande, y, por tanto, de estimar una historia con 13 puntos, se desglosará en otras más pequeñas.

El primer Sprint ha sido el denominado *Sprint Zero* [14]. Este sprint es utilizado para generar una primera lista de historias de usuario, poner a punto el entorno de desarrollo, estimar las primeras historias y establecer un plan inicial. La duración de este Sprint será de varias semanas, hasta que se hayan completado esas tareas iniciales, pero con un número de horas por semana menor que cuando se empiece a desarrollar el proyecto.

En la Tabla 2.10 se observa una calendarización inicial con los eventos que ocurrirán durante el proyecto y sus fechas.

Evento	Fecha inicio	Fecha fin
Sprint 0	17/01/2023	14/02/2023
Scrum Weekly, Sprint Review, Sprint Retrospective y Sprint Planning	14/02/2023	
Sprint 1	14/02/2023	28/02/2023
Scrum Weekly	21/02/2023	
Scrum Weekly, Sprint Review, Sprint Retrospective y Sprint Planning	28/02/2023	
Sprint 2	28/02/2023	14/03/2023
Scrum Weekly	07/03/2023	
Scrum Weekly, Sprint Review, Sprint Retrospective y Sprint Planning	14/03/2023	
Sprint 3	14/03/2023	28/03/2023
Scrum Weekly	21/03/2023	
Scrum Weekly, Sprint Review, Sprint Retrospective y Sprint Planning	28/03/2023	
Sprint 4	28/03/2023	11/04/2023
Scrum Weekly	04/04/2023	
Scrum Weekly, Sprint Review, Sprint Retrospective y Sprint Planning	11/04/2023	
Sprint 5	11/04/2023	25/04/2023
Scrum Weekly	18/04/2023	
Scrum Weekly, Sprint Review, Sprint Retrospective y Sprint Planning	25/04/2023	
Sprint 6	25/04/2023	09/05/2023
Scrum Weekly	02/05/2023	
Scrum Weekly, Sprint Review, Sprint Retrospective y Sprint Planning	09/05/2023	
Sprint 7	09/05/2023	23/05/2023
Scrum Weekly	16/05/2023	
Scrum Weekly, Sprint Review, Sprint Retrospective y Sprint Planning	23/05/2023	
Sprint 8	23/05/2023	06/06/2023
Inicio solicitud defensa - Primera Convocatoria	29/05/2023	
Scrum Weekly	30/05/2023	
Scrum Weekly, Sprint Review, Sprint Retrospective (Sprint Planning si se hace uso del Sprint extra 1)	06/06/2023	
Sprint extra 1	06/06/2023	20/06/2023
Scrum Weekly	13/06/2023	
Scrum Weekly, Sprint Review, Sprint Retrospective	20/06/2023	
Límite solicitud defensa - Primera Convocatoria	23/06/2023	

Tabla 2.10: Planificación inicial

2.7. Presupuesto

A continuación se exponen dos planes presupuestarios elaborados para este proyecto. Primero, se mostrarán los costes que habría tenido el proyecto de no haberse realizado en un contexto académico. Esto es, en un contexto en el que, por ejemplo, no se tendrían disponibles licencias académicas gratuitas para estudiantes de distintos programas.

Después, se mostrará el segundo plan presupuestario, que serán los costes reales que se han necesitado para la realización de este proyecto.

2.7.1. Presupuesto simulado

Lo primero que se va a tener en cuenta en el presupuesto simulado va a ser el sueldo del trabajador. Según *es.talent.com*, el sueldo medio de un Desarrollador Junior de ASP.NET [15] es de 13,08€ por hora. Dado que el Trabajo de Fin de Grado debe tener una duración de 300 horas, el gasto total en el sueldo bruto del desarrollador será 3924€. A este sueldo bruto hay que sumar las cotizaciones a la seguridad social por el trabajador, una cifra que corresponde aproximadamente con el 30 % del sueldo bruto [16]. Esto implica que el gasto en dirigido a la seguridad social es de 1177.2€.

Siguiendo ahora por el equipo del trabajador, se ha decidido contar con un Dell Latitude 5440 [17], con la configuración presente en el enlace. Su precio de venta es de 1257,18€. Suponiendo una vida útil del equipo de 4 años, y teniendo en cuenta que la duración del proyecto será de aproximadamente 6 meses, el coste del equipo durante los meses de trabajo será de 157.15€.

Para el espacio de trabajo se ha escogido un espacio de coworking en Palencia [18], con una tarifa mensual de 60.00€ al mes. Para cubrir los 6 meses de trabajo, el precio total a pagar será de 360€.

En cuanto a las licencias de los programas que se van a utilizar, tenemos la del programa Visual Paradigm y la de Visual Studio Professional. En cuanto a Visual Paradigm, el precio por una suscripción de 3 meses para la versión Standard es de 52.10€ aproximadamente [19], por lo que el precio total para los 6 meses de utilización sería de 104.20€.
En cuanto a Visual Studio, se utilizará la versión de Visual Studio Professional, que tiene un precio aproximado de 41€ [20] al mes, e incluye también un plan básico de Azure DevOps. Por tanto, el precio total por el uso de Visual Studio Professional será de 246€.

Por último, tenemos los servicios en la nube que vamos a utilizar. En este proyecto se usarán tres productos de Azure: una base de datos SQL, *Azure SQL Database*; un registro de contenedores donde poder almacenar la imagen Docker de nuestra aplicación, *Container Registry* y una servicio de contenedores para poder desplegar la aplicación, *Azure Container App*.
Para calcular los precios utilizaremos la calculadora de precios de Azure [21]. En cuanto a la base de datos SQL, se utilizará una base de datos con nivel de servicio standard, basado en DTU. Se dispondrá de 250GB de almacenamiento y 10 DTUs por segundo. Todo esto

supondrá un gasto de 13.33€ al mes. Para el contenedor de imágenes Docker de nuestra aplicación, se utilizará un registro básico, disponible todos los días, por un precio de 4.68€ al mes. Azure ofrece un servicio de contenedores gratuito, con hasta 2 millones de peticiones gratuitas cada mes, permitiendo 20 peticiones concurrentes por aplicación y con un tiempo de ejecución por petición de 100 ms. Todo esto hace un total de 18.00€ al mes por los tres productos (los precios están en dólares originalmente, y luego Azure lo convierte a euros, de ahí la diferencia de un céntimo en la suma de cada producto individualmente). Por tanto, para los 6 meses que durará el proyecto se necesitará 108€ para pagar los servicios de Azure.

Sumando todo lo anterior, tenemos un presupuesto para los 6 meses de 4899.35€. Para poder hacer frente a algún gasto imprevisto, como algún retraso en el desarrollo del proyecto o algún programa que se necesite y requiera de licencia, se incrementará ese presupuesto en un 25 %. Por tanto, el presupuesto final para el proyecto será de 6076.55€.

En la Tabla 2.11 se resumen los detalles del presupuesto simulado.

Concepto	Precio por unidad	Cantidad	Total
Trabajador y espacio de trabajo			
Sueldo Desarrollador Junior ASP.NET	13.08€/hora	300 horas	3924€
Seguridad Social del empleado	3.924€/hora	300 horas	1177.2€
Equipo del empleado (Dell Latitude 5430)	26.19€/mes	6 meses	157.15€
Espacio de trabajo coworking	60.00€/mes	6 meses	360€
Licencias			
Visual Paradigm Standard	52.10€/3 meses	6 meses	104.20€
Visual Studio Professional	41€/mes	6 meses	246€
Productos Azure			
Azure SQL Database	13.33€/mes	6 meses	79.98€
Container Registry	4.68€/mes	6 meses	28.08€
Azure Container App	0.00€/mes	6 meses	0€
TOTAL			**6076.55€**
+ 25 % de incremento			**7595.68€**

Tabla 2.11: Presupuesto simulado

Hay que tener en cuenta que este presupuesto sería un cálculo interno, no se entregaría así al cliente. Ese 25 % de incremento se tendría en cuenta en la preparación de presupuesto para cliente en concepto de personal principalmente, en el caso de que se necesitaran horas extra o no se pudiera desarrollar el producto completo en los 6 meses planificados. Se han incluido sólo los costes de desarrollo y producción durante el tiempo que se espera que dure el desarrollo del proyecto, pero habría que añadir el coste del producto en producción (en explotación) una vez desarrollado y desplegado el proyecto.

2.7.2. Presupuesto real

En el presupuesto real no van a existir la mayoría de costes presentes en el presupuesto simulado, ya que se trata de un proyecto académico realizado en el marco de la asignatura

Trabajo de Fin de Grado. Por un lado, no existen gastos relacionados con el sueldo ni la seguridad social. Tampoco generará un gasto el espacio de trabajo, ya que el proyecto se realizará en casa la mayor parte, y puntualmente en la Escuela de Ingeniería Informática de Valladolid.

Sin embargo, al trabajar desde casa sí que se tendrá en cuenta el consumo de electricidad. El consumo medio de un portátil es de 65Wh cada hora aproximadamente [22], y el precio de la luz de media en enero fue de 0.07€ por kWh [23]. Por tanto, en 300 horas que durará el proyecto tenemos 19.5kWh de consumo eléctrico y un gasto de 1.37€.

El equipo de trabajo, un MSI Creator 15 A10SDT-065ES, tiene un precio de 1600€. Suponiendo que su vida útil es de 4 años, el precio del equipo es de 33.33€ al mes. En los 6 meses que durará el proyecto, el coste total asciende a 199.98€.

Tampoco ha habido ningún coste relacionado con las licencias de los programas utilizados. Por un lado, se utilizará una licencia académica de Visual Paradigm. por otro, se utilizará la versión Community de Visual Studio, que es gratuita.

En referencia a los productos de Azure, dado que se cuenta con una suscripción para estudiantes y algunos servicios gratuitos, tampoco supondrán ningún coste real. Todos los productos de Azure que se utilizarán en el proyecto son gratuitos para estudiantes, respetando ciertos límites de uso. En el caso de que se sobrepasaran en algún momento, se cuenta con 100€ de crédito de Azure para hacer frente a ellos, de forma que en ningún caso el uso de los productos de Azure supondrá un coste real.

En la Tabla 2.12 se resumen los detalles del presupuesto real.

Concepto	Precio por unidad	Cantidad	Total
Electricidad	0.00455€/hora	300 horas	1.37€
Equipo de trabajo	33.33€/mes	6 meses	199.98€
TOTAL			**201.35€**

Tabla 2.12: Presupuesto real

2.8. Product Backlog inicial

Tal y como se ha explicado en la Sección 2.2, el alumno asume el papel de Product Owner y, por tanto, es el encargado del Product Backlog. Mediante el análisis de los reglamentos deportivos de varios años de la F1 se ha elaborado el Product Backlog inicial que se muestra en la Tabla 2.13

Los requisitos en Scrum tienen forma de historias de usuario, que tienen la siguiente estructura: *"Como «stakeholder» quiero «objetivo» para «razón que genere un valor para el stakeholder»"*. Los requisitos en esta versión inicial del Product Backlog están escritos en forma de épicas, que son historias de usuario de gran tamaño que tendrán que ser separadas en otras más pequeñas, de manera que puedan abordarse en un sprint.

Número	Épica
1	Como comisario quiero gestionar el reglamento deportivo de la F1 y poder añadir y modificar artículos antes del comienzo de cada temporada para asegurar el correcto funcionamiento del campeonato.
2	Como comisario quiero crear temporadas, con su reglamento, sus competiciones y sus equipos para poder contener el funcionamiento normal del campeonato.
3	Como comisario quiero gestionar los pilotos y el transcurso de una temporada.
4	Como jefe de equipo y como comisario queremos gestionar las participaciones de los pilotos en competiciones y el transcurso de las competiciones en la temporada.
5	Como comisario quiero sancionar a pilotos cuando incumplan algún artículo del reglamento.
6	Como aficionado quiero poder visualizar el histórico de reglamentos, sanciones y eventos de la categoría, así como consultar las sanciones recientes y sus consecuencias en cada piloto de manera rápida y eficiente.
7	Como equipo de desarrollo quiero utilizar la arquitectura de referencia y documentar el proyecto correctamente para que sea mantenible en el futuro.

Tabla 2.13: Product Backlog Inicial

2.9. Requisitos

La descomposición de las épicas en historias de usuario más pequeñas y abordables en un sprint se ha realizado a lo largo de todo el proyecto, y en esta sección se detalla el resultado de ese trabajo. Se detallarán los requisitos funcionales, requisitos funcionales de información y, por último, los requisitos no funcionales y restricciones, todos ellos expresados como historias de usuario.

2.9.1. Requisitos funcionales

Los requisitos funcionales se han obtenido descomponiendo las épicas del Product Backlog (Tabla 2.13) en historias de usuario. A cada historia de usuario resultante de esta descomposición se le ha asignado unos puntos de historia de usuario, correspondientes con la estimación del esfuerzo que supondrá completar el desarrollo de esa historia de usuario.

En las tablas 2.14 a 2.19 se muestran las historias de usuario que descomponen cada épica del Product Backlog inicial.

Épica 1: Como comisario quiero gestionar el reglamento deportivo de la F1 y poder añadir y modificar artículos antes del comienzo de cada temporada para asegurar el correcto funcionamiento del campeonato.

Código	Historia de Usuario	Puntos
HU01	Como comisario quiero añadir artículos antes del inicio de cada temporada para mantener el reglamento actualizado.	3
HU02	Como comisario quiero poder asignar penalizaciones que no puedan eliminar puntos de licencia a un reglamento para que su incumplimiento tenga consecuencias.	2
HU03	Como comisario quiero poder crear un reglamento antes del inicio de una temporada, asignándole los artículos y las sanciones que lo formarán.	3
HU04	Como comisario quiero poder editar sólo los artículos que no forman parte de un reglamento asignado a una temporada.	3
HU05	Como comisario quiero poder eliminar sólo los artículos que no forman parte de un reglamento asignado a una temporada.	3
HU06	Como comisario quiero poder iniciar sesión en la aplicación.	5

Tabla 2.14: Historias de usuario de la épica 1

Épica 2: Como comisario quiero crear temporadas, con su reglamento, sus competiciones y sus equipos para poder contener el funcionamiento normal del campeonato.

Código	Historia de Usuario	Puntos
HU08	Como comisario quiero asignar penalizaciones que eliminen puntos de licencia a un reglamento para que su incumplimiento tenga consecuencias.	3
HU09	Como comisario quiero poder añadir circuitos en los que se podrán disputar competiciones.	2
HU10	Como comisario quiero poder inscribir a nuevos equipos que vayan a participar en la competición.	3
HU11	Como comisario, quiero poder crear una temporada en la que estén definidos desde el comienzo el reglamento, las competiciones y los equipos que van a participar, y que no sean editables a partir de ese momento.	5
HU12	Como comisario quiero poder registrar nuevos comisarios y jefes de equipo en el sistema.	2

Tabla 2.15: Historias de usuario de la épica 2

Épica 3: Como comisario quiero gestionar los pilotos y el transcurso de una temporada.

Código	Historia de Usuario	Puntos
HU15	Como comisario quiero inscribir a un piloto en la competición y asignarle un equipo.	3
HU16	Como comisario quiero poder eliminar la asignación de un equipo a un piloto, y también poder reasignarle a otro equipo.	2
HU17	Como comisario quiero que no se puedan crear temporadas si existe alguna que no se ha terminado.	2
HU18	Como comisario quiero poder arrancar la siguiente competición de una temporada, siempre que la anterior se haya finalizado ya.	5

Tabla 2.16: Historias de usuario de la épica 3

Épica 4: Como jefe de equipo y como comisario queremos gestionar las participaciones de los pilotos en competiciones y el transcurso de las competiciones en la temporada.

Código	Historia de Usuario	Puntos
HU19	Como jefe de equipo quiero poder añadir la participación de dos pilotos que tenga asignados antes de que comience la primera sesión de una competición.	3
HU20	Como comisario quiero poder dar por finalizadas las sesiones de una competición e iniciar las siguientes.	3

Tabla 2.17: Historias de usuario de la épica 4

Épica 5: Como comisario quiero sancionar a pilotos cuando incumplan algún artículo del reglamento.

Código	Historia de Usuario	Puntos
HU21	Como comisario quiero poder registrar un incidente, una investigación sobre el incumplimiento de un artículo por parte de un piloto en una sesión que está en curso y asignar una sanción.	5
HU22	Como comisario quiero que al añadir un incidente se actualicen los puntos de licencia del piloto involucrado.	3
HU23	Como comisario quiero que al empezar una competición se actualicen los puntos de licencia de los pilotos.	3
HU24	Como comisario quiero que un piloto con 12 puntos de licencia o con una suspensión del Gran Premio anterior no pueda participar en la siguiente competición.	2

Tabla 2.18: Historias de usuario de la épica 5

Épica 6: Como aficionado quiero poder visualizar el histórico de reglamentos, sanciones y eventos de la categoría, así como consultar las sanciones recientes y sus consecuencias en cada piloto de manera rápida y eficiente.

Código	Historia de Usuario	Puntos
HU25	Como aficionado quiero poder ver el histórico de los incidentes de la categoría y poder filtrarlos por piloto y por sesión.	3

Tabla 2.19: Historias de usuario de la épica 6

2.9.2. Requisitos funcionales de información

Los requisitos funcionales de información indican el tipo de información que deberá almacenar el sistema, a partir de los cuales se elaborará el modelo de dominio inicial.

En la Tabla 2.20 se observan los requisitos funcionales de información en forma de historias de usuario.

2.9.3. Requisitos no funcionales y restricciones

Para la obtención de los requisitos no funcionales y de las restricciones que afectan al proyecto, se ha utilizado la guía FURPS [24], que es un acrónimo inglés que significa lo siguiente: funcionalidad (ya cubierto en los requisitos funcionales), usabilidad, fiabilidad, rendimiento y soporte (en inglés: Functionality, Usability, Reliability, Performance, Supportability). Requisitos no funcionales expresados como historias de usuario.

En la Tabla 2.21 se muestra una primera parte de los requisitos no funcionales, especialmente aquellos que tienen que ver con soporte, en concreto mantenibilidad de la aplicación, expresadas como historias de usuario. Este primer grupo forma parte del Product Backlog, en concreto del desglose de la épica 7, y tiene asociada una estimación en forma de puntos de historia del esfuerzo que conllevará realizar cada historia.

Historia de usuario
Como equipo de desarrollo quiero guardar la siguiente información sobre un artículo: contenido y sus subartículos, para poder procesar la información necesaria para el funcionamiento de la aplicación.
Como equipo de desarrollo quiero guardar la siguiente información sobre una penalización: su nombre, su tipo, donde se incluye su nombre, descripción, si se pueden aplicar puntos de licencia o una multa económica. Si es una penalización de posiciones en parrilla, el número de posiciones. Si es una descalificación, se guardará si es descalificación de la competición actual o suspensión en la siguiente. Si es una penalziación de tiempo, se anotará de cuántos segundos. Si es un Drive Through, se anotará el tiempo que se tarda en cumplir la sanción. Si es un Stop&Go, se guardará el número de segundos que ha de parar el sancionado. Si es una reprimenda, se indicará si es por conducción o no.
Como equipo de desarrollo quiero guardar la siguiente información sobre un reglamento: sus artículos, que serán un subconjunto de los artículos existentes, y sus sanciones, que serán un subconjunto de las penalizaciones existentes, para poder procesar la información necesaria para el funcionamiento de la aplicación.
Como equipo de desarrollo quiero almacenar la siguiente información sobre un usuario del sistema: rol, nombre de usuario, email, contraseña y nombre completo, para poder procesar la información necesaria para el funcionamiento de la aplicación.
Como equipo de desarrollo quiero almacenar la siguiente información sobre un competidor: nombre, unidad de potencia, estado y jefe de equipo, para poder procesar la información necesaria para el funcionamiento de la aplicación.
Como equipo de desarrollo quiero almacenar la siguiente información sobre un piloto: nombre, fecha de nacimiento, puntos de licencia y competidor para el que compite, para poder procesar la información necesaria para el funcionamiento de la aplicación.
Como equipo de desarrollo quiero almacenar la siguiente información sobre un circuito: nombre, longitud, número de vueltas en el Gran Premio, año del primer Gran Premio, tiempo de vuelta rápida, piloto que hizo la vuelta rápida y año de la vuelta rápida, para poder procesar la información necesaria para el funcionamiento de la aplicación.
Como equipo de desarrollo quiero almacenar la siguiente información sobre una temporada: año, competidores y reglamento, para poder procesar la información necesaria para el funcionamiento de la aplicación.
Como equipo de desarrollo quiero almacenar la siguiente información sobre una competición: nombre, fecha, si es sprint, temporada, sesiones, participaciones (piloto y competidor de cada piloto) y estado, para poder procesar la información necesaria para el funcionamiento de la aplicación.
Como equipo de desarrollo quiero almacenar la siguiente información sobre una sesión: tipo de sesión, estado e incidentes que se han producido en ella.
Como equipo de desarrollo quiero almacenar la siguiente información sobre un incidente: fecha y hora de creación, piloto involucrado, hechos ocurridos, sesión, artículo investigado, penalización aplicada, razón y, si aplica, multa y puntos de licencia añadidos, para poder procesar la información necesaria para el funcionamiento de la aplicación.

Tabla 2.20: Requisitos funcionales de información como historias de usuario

Épica 7: Como equipo de desarrollo quiero utilizar la arquitectura de referencia y documentar el proyecto correctamente para que sea mantenible en el futuro.

Código	Historia de Usuario	Puntos
HU07	Como equipo de desarrollo quiero crear el proyecto utilizando desde el principio arquitectura de referencia en el desarrollo correctamente para que sea mantenible.	5
HU13	Como equipo de desarrollo quiero revisar que la arquitectura de referencia se sigue usando correctamente en el desarrollo.	15
HU14	Como equipo de desarrollo quiero documentar el proyecto correctamente para que el proyecto sea mantenible.	30

Tabla 2.21: Historias de usuario de la épica 7

Por otro lado, tenemos los requisitos no funcionales y restricciones que no van a formar parte del Product Backlog, sino que se tomarán decisiones de diseño sobre ellos en la Sección 5.1. A las historias de usuario de esta segunda parte se les ha asignado un código distinto, 'HUNF<número>', que se utilizará para referirse a ellas en el resto del documento.

En la Tabla 2.22 se observan el resto de requisitos no funcionales y restricciones expresados como historias de usuario.

Código	Historia de Usuario
HUNF01	Como participante y como jefe de equipo quiero que el sistema sea una aplicación web para poder acceder a ella desde cualquier dispositivo.
HUNF02	Como participante quiero ser capaz de utilizar la aplicación de manera rápida con dos horas de entenamiento.
HUNF03	Como equipo de desarrollo quiero que los secretos de la aplicación sean almacenados de la forma más segura posible.
HUNF04	Como equipo de desarrollo quiero que la arquitectura utilizada en la aplicación haga que sea lo más mantenible, reusable y extensible posible.
HUNF05	Como equipo de desarrollo quiero que la aplicación sea escalable para garantizar un buen rendimiento cuando el número de usuarios crezca.
HUNF06	Como equipo de desarrollo quiero el idioma del código y la documentación interna del proyecto sea el inglés, para que sea traspasable internacionalmente.
HUNF07	Como usuario quiero que la interfaz de la aplicación esté en inglés, para que usuarios de todo el mundo puedan utilizarla sin problemas.
HUNF09	Como equipo de desarrollo quiero que la aplicación funcione en un navegador Chrome para Windows 10
HUNF10	Como equipo de desarrollo quiero que la aplicación haya sido probada para asegurar su buen funcionamiento antes de desplegarla en un entorno productivo.

Tabla 2.22: Requisitos No Funcionales y Restricciones expresados como historias de usuario

Capítulo 3

Análisis

A lo largo de este capítulo se ha descrito el análisis del sistema a desarrollar. Esta tarea de análisis se ha realizado en dos momentos en el desarrollo del proyecto: 1) durante el sprint 0 se realizó un primer análisis del sistema en general a partir de los requisitos recogidos en el Capítulo 2 y 2), en el desarrollo de cada historia de usuario existirá una tarea dedicada a revisar y mejorar el análisis.

3.1. Modelo de dominio

Teniendo en cuenta los requisitos funcionales de información (Tabla 2.20) se ha realizado un modelo de dominio inicial a partir del cual se ha ido iterando hasta obtener el modelo del dominio representado mediante un diagrama de clases en la Figura 3.1.

En él se pueden observar las entidades principales de la aplicación y sus relaciones entre ellas, así como la información que caracteriza a cada una de ellas.

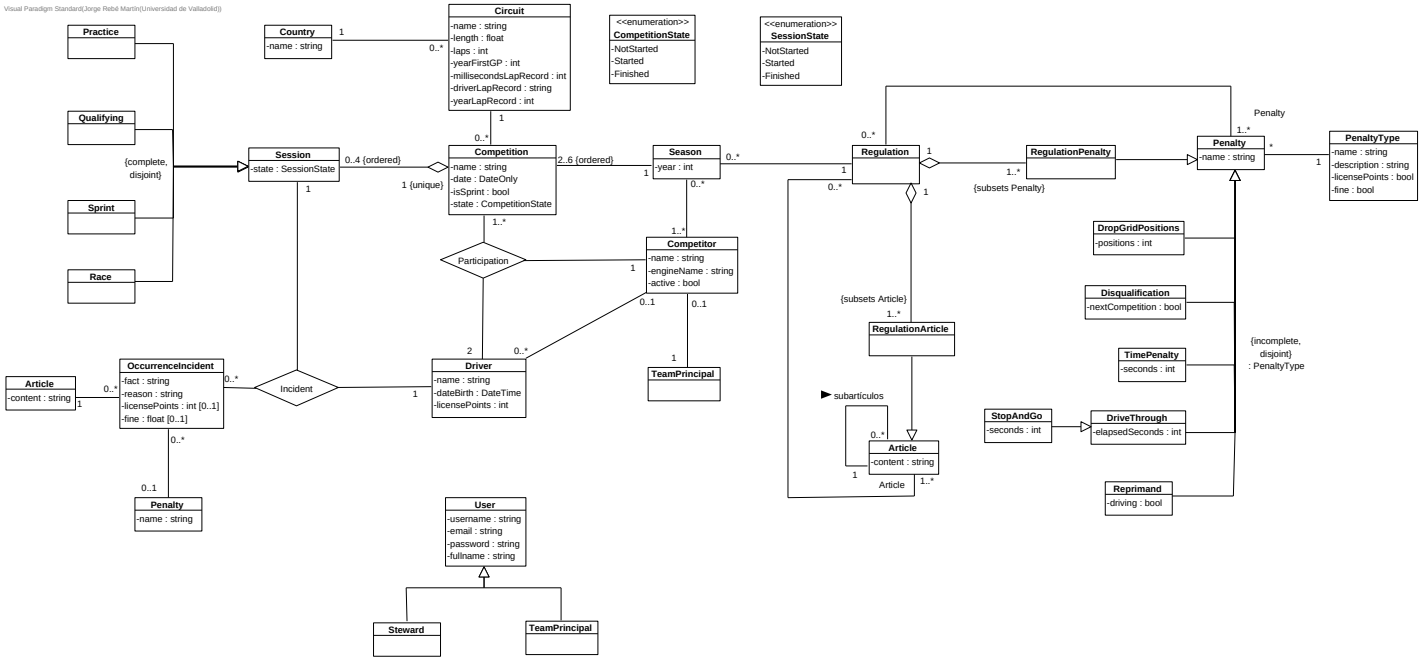


Figura 3.1: Modelo de dominio

3.2. Restricciones sobre el modelo de dominio

Sobre el modelo de dominio inicial existen algunas restricciones adicionales que se van a expresar utilizando el lenguaje de restricción de objetos (Object Constraint Language, OCL) (ver Subsección 4.3.1).

A continuación se describen algunas restricciones sobre el modelo de dominio, expresadas primero en lenguaje natural, y luego utilizando OCL.

- Un piloto no puede tener menos de cero puntos de licencia ni más de doce.

  context Driver
  inv: self.licensePoints >= 0 and self.licensePoints <= 12

- En la ocurrencia de un incidente no se pueden aplicar más de seis puntos de licencia de golpe ni el piloto se puede quedar con más de doce tras aplicar la sanción.

  context OccurrenceIncident
  inv: self.licensePoints ->notEmpty() implies self.licensePoints <= 6 and (self.Driver.licensePoints + self.licensePoints) <= 12

- La ocurrencia de un incidente sólo puede tener puntos de licencia si se ha aplicado una penalización, y el tipo de esa penalización permite añadir puntos de licencia.

  context Incident
  inv: (self.licensePoints ->notEmpty() implies self.Penalty ->notEmpty() and self.Penalty.PenaltyType.licensePoints = true) and ((self.Penalty ->notEmpty() and self.Penalty.PenaltyType.licensePoints = true) implies self.licensePoints ->notEmpty())

- La ocurrencia de un incidente sólo puede tener una multa si se ha aplicado una penalización, y el tipo de esa penalización permite añadir una multa.

  context Incident
  inv: (self.fine ->notEmpty() implies self.Penalty ->notEmpty() and self.Penalty.PenaltyType.licensePoints = true) and ((self.Penalty ->notEmpty() and self.Penalty.PenaltyType.fine = true) implies self.fine ->notEmpty())

- Un equipo puede participar con un máximo de cuatro pilotos distintos en una temporada.

  context Season
  inv: self.Competitors ->forAll(c : Competitor | c.Participation.Driver->asSet()->size() <= 4)

3.3. Modelo de proceso de negocio

Para finalizar con el análisis del sistema a desarrollar, se ha realizado un diagrama de actividades que muestra el proceso de negocio principal: el transcurso de una sesión y la asignación de incidentes mientras está en marcha. En la Figura 3.2 puede observarse el diagrama.

Se puede observar que hay un único actor implicado en el diagrama. En el momento que una sesión da comienzo, el comisario la marca como comenzada. A partir de ese momento, cada vez que ocurra un incidente, el comisario asigna el incidente al piloto que corresponda, escribe sobre lo ocurrido y añade el artículo quebrantado y la penalización elegida.

Una vez la sesión termina, no se pueden añadir nuevos incidentes, la sesión se marca como terminada y la actividad termina.

3.4. Máquinas de estados

En el modelo de proceso anterior se aprecia la sesión cambiando de estado. Es por ello que se hace necesario analizar la sesión como máquina de estados y determinar si algún otro elemento del dominio se comporta como tal para modelarlo también.

En las Figuras 3.3 y 3.4 se muestran dos diagramas de máquinas de estados, representando los estados y transiciones de una competición y una sesión, respectivamente.

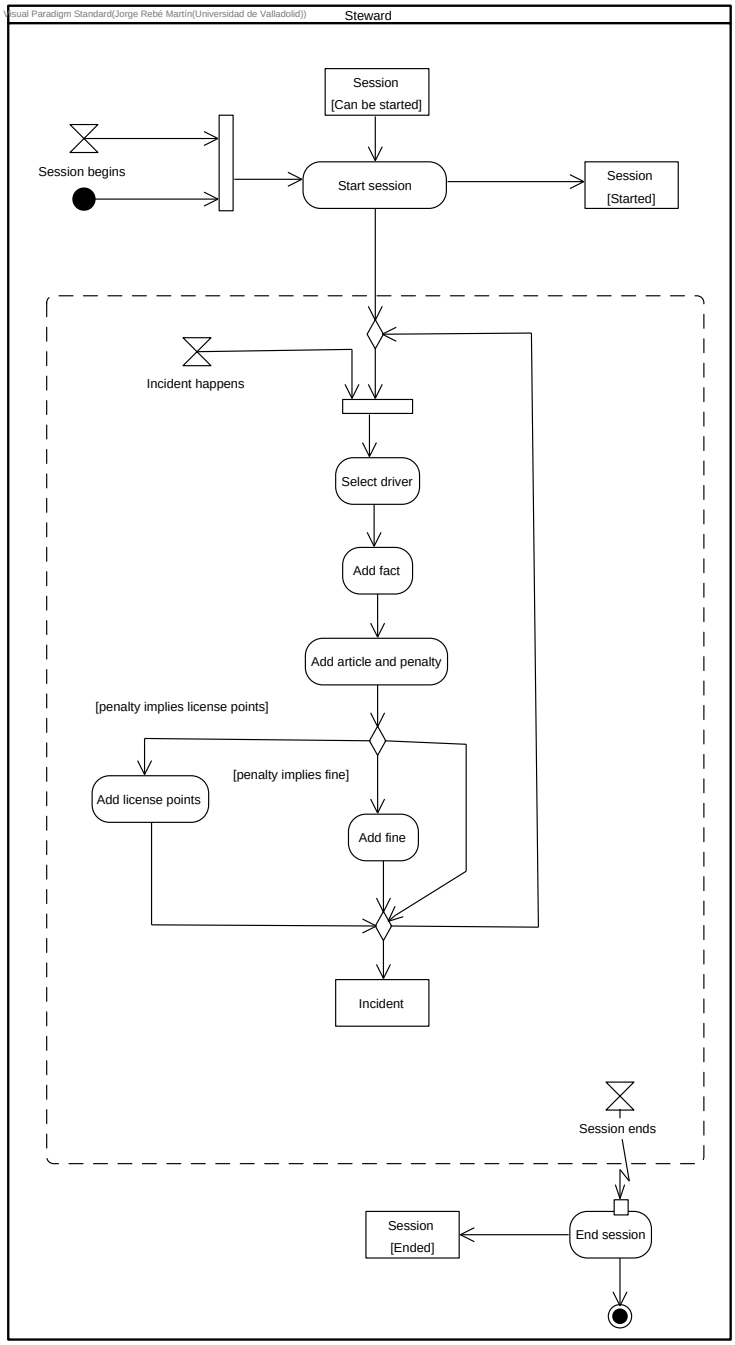


Figura 3.2: Modelo de proceso de negocio

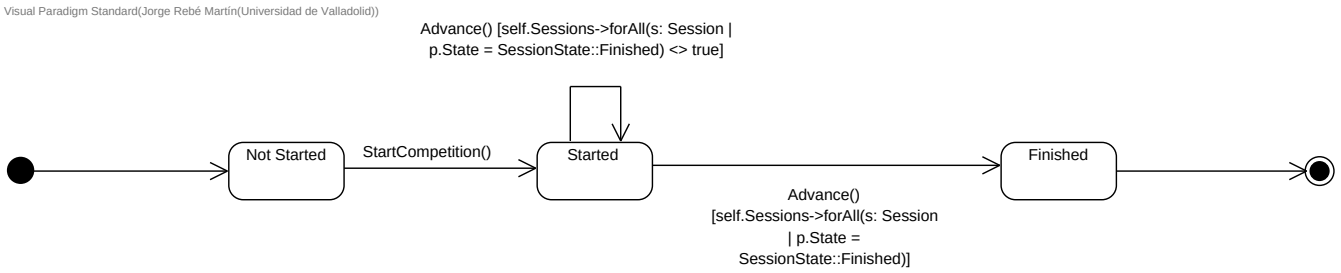


Figura 3.3: Diagrama de máquina de estados de una competición.

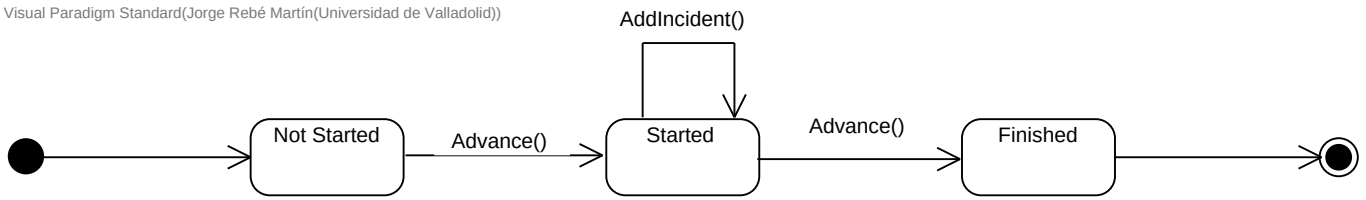


Figura 3.4: Diagrama de máquina de estados de una sesión.

Capítulo 4

Tecnologías utilizadas

En este Capítulo se presenta un resumen y pequeña descripción de las tecnologías que han sido utilizadas en el proyecto, divididas en tres categorías: **1)** herramientas para el desarrollo del proyecto, **2)** herramientas para la gestión del proyecto y **3)** herramientas para la documentación del proyecto.

4.1. Tecnologías de desarrollo

4.1.1. .NET

.NET [25] es una plataforma de desarrollo de código abierto, creada por Microsoft, que puede ser utilizada para construir aplicaciones web, de escritorio, móviles y videojuegos, entre otros tipos. Estas aplicaciones se pueden desarrollar utilizando tres lenguajes de programación: C#, F# y Visual Basic.

Además, .NET cuenta con bibliotecas y APIs que son comunes para todas las aplicaciones .NET. Existe un buscador de estas APIs de .NET [26], y ya sea por espacio de nombres o por el nombre de la clase, se puede encontrar la clase que estamos buscando, documentación sobre ella y diversos ejemplos de cómo usarlas.

Como extensión de la funcionalidad, se ofrece un ecosistemas de paquetes denominado NuGet [27], que también permite a los desarrolladores publicar sus propios paquetes para que sean utilizados por la comunidad.

En cuanto a usos conocidos de .NET, tenemos varios [28]. En particular, destaca el de la plataforma de preguntas y respuestas Stack Overflow [29], que se desarrolló utilizando ASP.NET.

.NET 7

.NET 7 [30] es la versión de .NET utilizada para el desarrollo de este proyecto. Se lanzó el 8 de noviembre de 2022, como una *'standard term support (STS) release´*, lo que significa que tendrá soporte de Microsoft durante 18 meses, hasta el 14 de mayo de 2024. Antes de esa fecha, habría que migrar la aplicación a la versión .NET 8, que se lanzará a finales de 2023 y tendrá soporte durante 3 años desde su lanzamiento al ser una *'long term support"*

En .NET 7 se han realizado diversas mejoras en rendimiento, soporte para la publicación de aplicaciones contenerizadas, gestor central de dependencias entre los proyectos de una solución, etcétera.

ASP.NET Core

ASP.NET Core [31] es un framework open source de desarrollo web, que permite construir aplicaciones web modernas y servicios.

ASP.NET Core extiende la plataforma .NET con herramientas y bibliotecas para la construcción de aplicaciones web. Proporciona un framework base para el procesamiento de peticiones web en C# o F#, sintaxis de plantillas de páginas web, bibliotecas para patrones web comunes (como MVC), un sistema de autenticación y extensiones al editor (como el autocompletado de código).

Para el código Backend, se puede utilizar C#, F# o Visual Basic, aunque en el caso de este proyecto se utilizará únicamente C#.
En la parte de Front, se utilizarán páginas dinámicas utilizando C# y HTML. El código C# en las páginas es evaluado en el servidor, y el HTML resultante se envía al cliente. Por supuesto también se utilizará CSS para el estilo de las páginas HTML y JavaScript para el código ejecutado en el cliente que permite realizar algunas validaciones y poder modificar de forma dinámica los formularios.

C#

C# [32] es un lenguaje de programación orientado a objetos y orientado a componentes, de propósito general.

Los programas C# se ejecutan en .NET, un sistema de ejecución virtual CLR y un conjunto de liberías de clases. El código fuente C# se compila en un lengua intermedio, que, junto con sus recursos, se almacenan en un ensamblado que típicamente tiene la extensión .dll.

4.1.2. Entity Framework

Entity Framework [33] es un mapeador entre objectos y su modelo de datos relacional (*ORM*) que permite construir una capa de acceso a datos limpia, portable y de alto nivel en .NET usando C#. Tiene soporte para varias bases de datos, incluyendo SQL Database, SQLite, MySQL y PostgreSQL, entre otras. Permite consultas utilizando LINQ, seguimiento de cambios, actualizaciones y migraciones del modelo de datos.

Para realizar las operaciones sobre la base de datos, existe la clase *DbContext* [34]. Una instancia de esta clase representa una sesión con la base de datos que se utiliza para realizar consultas y guardar instancias de entidades. Típicamente se crea una clase que hereda de *DbContext*, y en ella se añaden las propiedades y configuraciones de la base de datos particulares de cada proyecto.

4.1.3. Visual Studio 2022

Visual Studio [35] es un entorno de desarrollo integrado (IDE, por sus siglas en inglés), desarrollado por Microsoft, que ofrece una versión gratuita (Community Edition) para el uso personal. Ha sido el IDE elegido para el desarrollo de este proyecto, debido a la familiaridad con él por proyectos anteriores, por la facilidad con la que se puede desplegar una aplicación desde él y por sus herramientas de Debug, entre otras razones.

4.1.4. SQL Server

SQL Server [36] es un sistema gestor de bases de datos (SGDB) relacional desarrollado por Microsoft. Es el elegido para trabajar en este proyecto, aunque podría haberse utilizado cualquier otro, ya que no existen limitaciones en .NET sobre esto. Cuando la aplicación se despliega en local, se tendrá una instancia de un servidor SQL Server, y en cuando se despliegue en el entorno de producción se tendrá en Azure también una instancia de SQL Server.

Azure Data Studio

Azure Data Studio [37] es una herramienta para gestionar bases de datos que permite la conexión tanto a bases de datos en local (la utilizada en el entorno de desarrollo), como a bases de datos en la nube (la utilizada en el entorno de producción). Permite ver rápidamente los datos de las tablas, ejecutar scripts, ver el diseño de las tablas, editar los datos, crear scripts de borrado de tablas, de creación a partir de una tabla existente, etcétera.

4.1.5. Azure

Azure [38] es una plataforma de computación en la nube de Microsoft. Ofrece infraestructura, plataforma y software como servicios de computación (*IaaS*, *PaaS* y *SaaS*).

En el caso de este proyecto se usarán los siguientes productos de Azure: una base de datos (*IaaS*), un sistema para almacenar los secretos de la aplicación (*PaaS*) y una plataforma en la que desplegar la aplicación (*PaaS*).

SQL Database

Azure SQL Database [39] es un motor de base de datos PaaS completamente administrado. Se ejecuta siempre en la última versión estable del motor de base de datos SQL Server

La versión gratuita en la suscripción de estudiantes utilizada para este proyecto consiste en una instancia S0 de 250GB, con 10 unidades de transacción de base de datos (DTUs).

Azure Key Vault

Azure Key Vault [40] es una forma de almacenar secretos, tales como connection strings para realizar la conexión con una base de datos de producción. La idea era utilizar este servicio proporcionado por Azure para esto mismo, pero debido a que no se me dio acceso al Azure Active Directory (Azure AD), no fue posible utilizarlo.

Con una aplicación no desplegada y ejecutada desde Visual Studio se hizo funcionar, ya que Visual Studio está conectado con la cuenta de Azure, pero para conectarlo en producción es necesario registrarla en el Azure AD.

Azure Container Apps

Azure Container Apps [41] es un entorno que permite ejecutar microservicios contenerizados (o aplicaciones monolíticas, como en el caso de este proyecto) en una plataforma sin servidor.

Dado que el despliegue de este proyecto se realizará mediante un contenedor Docker, se ha elegido este entorno para ejecutarlo.

4.1.6. xUnit.net

En el caso de este proyecto, el framework de testing elegido ha sido xUnit [42]. Es una herramienta gratuita de código abierto de testing, que se utiliza tanto para escribr tests como para ejecutarlos. En este proyecto se utiliza individualmente para ejecutar tests unitarios, y junto con SpecFlow para ejecutar las pruebas de aceptación.

4.1.7. SpecFlow

SpecFlow [43] es una herramienta para la automatización de tests construida para .NET en el paradigma BDD (ver Sección 2.4.2. Es usado para definir, gestionar y ejecutar tests de aceptación que son entendibles por el cliente.

Los tests de SpecFlow se escriben utilizando Gherkin [44], que es un lenguaje que permite escribir las pruebas en lenguaje natural, siguiendo la sintaxis básica: 'Dada una situación', 'Cuando se realiza una acción', 'Entonces'. Después se genera código que va vinculado a esos pasos definidos con Gherkin, que se utiliza para ejecutar los tests utilizando un framework de testing de nuestra elección (xUnit en este caso).

4.1.8. Selenium

Selenium [45] es una herramienta de automatización de navegadores, utilizada principalmente para la automatización de tests de aplicaciones web. En este proyecto se usará conjuntamente con SpecFlow para automatizar la interacción con la interfaz de usuario en la aplicación web en los tests de aceptación automatizados.

4.1.9. Autofac

Autofac [46] es un contenedor de inyección de dependencias, utilizado para gestionar dependencias entre clases, de manera que las aplicaciones se mantengan fáciles de cambiar mientras se incrementa su tamaño y complejidad. En la sección 5.3 se explicará el principio de inversión de dependencias y su relación con la inyección de dependencias.

Al iniciar la aplicación, se configurará el contenedor de dependencias indicando cada una de las interfaces deseadas, la clase que la implementa implementación y el tiempo de vida de la instancia de esa clase.

4.1.10. Bootstrap

Bootstrap [47] es un framework utilizado para construir sitios web que se adapten a varios tamaños de pantalla (*responsive*). Contiene estilos propios en CSS y plugins en Javascript que son necesarios para que funcionen muchos de sus componentes.

Además, está perfectamente integrado con ASP.NET. Desde ASP.NET Core 3.0 y hasta ASP.NET 5.0 se usó Bootstrap 4, y a partir de ASP.NET Core 6 se utiliza Bootstrap 5.

4.1.11. ChatGPT

ChatGPT [48] es una inteligencia artificial de procesamiento del lenguaje natural desarrollada por OpenAI. Se lanzó de manera gratuita al público el 30 de noviembre de 2022, y se ha convertido en una herramienta de asistencia a la programación muy potente. En particular, en este proyecto ha permitido, entre otras cosas, aumentar la productividad en tareas como la generación de interfaces de usuario.

Por ejemplo, la interfaz de usuario para la creación de artículos se pensó, se diseñó y se pidió a ChatGPT que la realizara. Una vez hecha, se revisó y se le añadió detalles para mejorarla. De esta manera el trabajo más sencillo pero a la vez más laborioso lo realizó la herramienta, consiguiendo un pequeño ahorro de tiempo y esfuerzo en tareas repetitivas no creativas.

4.2. Herramientas para la gestión del proyecto

4.2.1. Git

Git [49] es un sistema de control de versiones distribuido, gratuito y de código abierto que se utiliza para gestionar tanto proyectos pequeños como grandes de manera rápida y eficiente.

Git se basa en un sistema de repositorios, cada uno con una o varias ramas. Podemos pensar en Git como un sistema de ficheros en miniatura [50], y en un repositorio de Git como un conjunto de ficheros y su historial de versiones y cambios. Dentro de un mismo repositorio habrá una o varias ramas (típicamente la primera rama creada en un repositorio Git se llama 'master' o 'main'). Suponiendo un repositorio Git con sólo una rama master, cuando se hace un commit, Git toma un *snapshot* (imagen instantánea) de los ficheros del repositorio y almacena una referencia a ese *snapshot*. Para ser eficiente, si algún fichero no ha cambiado, no se vuelve a almacenar de nuevo, sólo se almacena un enlace al fichero ya almacenado.

Un **commit** es ese *snapshot* y algunos metadatos como el autor, su email, la fecha y un mensaje descriptivo del commit. Una **rama** es un puntero movible a un commit. En nuestro ejemplo, una vez realizado el commit, el puntero de la rama master se movería a ese último commit. Cuando se crea una rama nueva, se crea un puntero nuevo que referencia al commit al que apunta la rama a partir de la cual se está creando esa nueva rama. Cuando se ha terminado de trabajar con esa rama nueva que se ha creado y se quieren añadir los cambios realizados en ella a otra rama, se realiza la operación **merge**, con origen la rama con la que se ha terminado de trabajar y destino la rama a la que se quieren añadir los cambios.

En la Figura 4.1 se observa el estado de un repositorio Git con tres ramas y algunos commits por cada rama.

Al ser Git un sistema distribuido, lo cual significa que la copia local del código es una versión completa del repositorio que se tiene en remoto. Para mantener sincronizados los dos

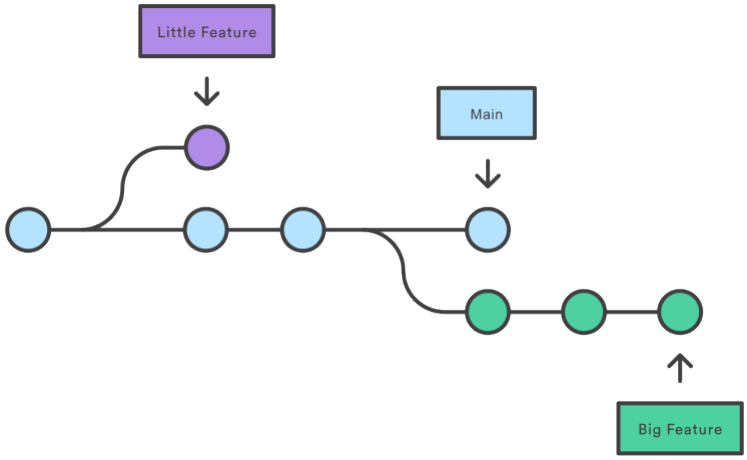


Figura 4.1: Vista gráfica del funcionamiento de Git. Los círculos representan commits y los rectángulos ramas. [51]

repositorios existen algunas operaciones: *pull*, que trae los cambios producidos en el repositorio remoto al local; *commit*, como se ha visto antes, genera un snapshot en el repositorio local con los cambios realizados; y *push*, que actualiza el repositorio remoto con los últimos commits realizados en local.

Una de las características especiales de Git es que, cuando se realiza un *commit*, no es necesario incluir los cambios de todos los archivos. Git cuenta con una zona de preparación o *staging area*, donde se encuentra una lista de los ficheros modificados que van a formar parte del commit. Para añadir los ficheros que van a formar parte del commit, se utiliza la operación *add*, y tras esto, pasan a la *staging area* y puede realizarse el commit con los ficheros existentes en el *staging area*. En la Figura 4.2 se muestra de forma gráfica el proceso de guardado de los cambios en los ficheros en un repositorio Git.

Las distintas ramas que se han utilizado en este proyecto, su nomenclatura y su propósito son las siguientes:

- `master`: rama original, en la que se encuentra una versión estable de la aplicación. Hasta que no se finalice con el desarrollo del TFG no se considerará que la aplicación está en una versión suficientemente estable como para entrar en esta rama.
- `develop`: rama creada a partir de máster, donde se encuentra una versión estable de la aplicación, todavía en desarrollo.
- `sprint<número>`: rama creada al comienzo de cada sprint, que se utilizará para hacer los desarrollos correspondientes al mismo. Al finalizar el sprint, se fusionará con la rama `develop`.
- `HU<número>_<descripción historia de usuario>`: rama que se utilizará para añadir nueva funcionalidad, creada a partir de la rama que corresponda con el sprint del

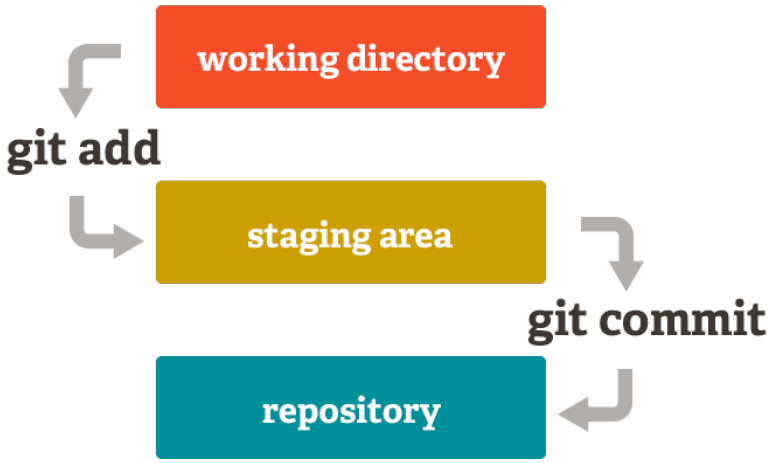


Figura 4.2: Guardado de cambios en ficheros en Git [49].

momento de su desarrollo. Se creará una rama por historia de usuario con el formato indicado.

- `BUG<número>_<descripción de bug>`: si se encuentra algún fallo o bug en la aplicación, se creará una rama a partir de la rama del sprint del momento de la detección (o de develop, en el caso de que se haya finalizado con el desarrollo), y se solucionará.

4.2.2. Azure DevOps

Azure DevOps [52] es un servicio de Microsoft utilizado para gestionar proyectos software. Permite el desarrollo de una aplicación durante todo el ciclo de vida de la misma, integrando varios servicios como control de versiones colaborativo, tableros para gestionar el seguimiento del trabajo hecho y por hacer, y pipelines para realizar integración continua y despliegue automatizado. Azure DevOps proporciona otros servicios, pero los tres mencionados son los que se van a utilizar en este proyecto.

Repos

Azure Repos [53] es un conjunto de herramientas de control de versiones que se utiliza para gestionar el código de proyectos software. Azure Repos ofrece dos tipos de control de versiones: Git y TFVC. Como se ha visto al principio de esta sección, el sistema de control de versiones elegido para este proyecto es Git.

Al ser Git un sistema de control de versiones distribuido, la copia local del código es un repositorio de control de versiones completo, lo cual permite trabajar offline o de forma remota, y sincronizar los cambios que se realicen con la versión del repositorio en el servidor cuando se desee.

Antes de hacer la operación *merge*, es conveniente revisar el código con el resto del equipo y asegurarse de que la aplicación junto con los cambios realizados compila y pasan los tests. Para ello, se crea un *pull request*, en la que se indica la rama origen y la rama destino del *merge*. Se revisan los cambios, se dejan comentarios si es necesario modificar algo y, finalmente, se aprueba la *pull request* y se realiza la operación *merge* entre las dos ramas especificadas.

Boards

Azure Boards [54] es un servicio que ayuda a planificar, realizar seguimiento en todo el proceso de desarrollo de software. Ofrece una plataforma flexible y personalizable para gestionar elementos de trabajo, tales como historias de usuario, bugs, tareas, además de para seguir el progreso de cada uno de ellos.

Para gestionar los elementos de trabajo, Azure Boards cuenta con varios centros de trabajo. A continuación aparecen los que se han utilizado en este proyecto, junto con sus funciones principales:

- ***Work items***: Una lista de elementos de trabajo que se puede filtrar por tipo, por asignados, estado, etcétera.
- ***Boards***: Se ven los elementos de trabajo como tarjetas, y se puede actualizar su estado arrastrándolos, simulando una pizarra física con tarjetas que representa cada elemento de trabajo. Se pueden visualizar todos los elementos de trabajo, pero suele ser más útil tenerlos filtrados para que aparezcan solo los del sprint actual.

  En la Figura 4.3 se observa el estado del Board durante el Sprint 8.
- ***Backlogs***: Aquí se puede ver, planear, ordenar y organizar los elementos de trabajo, moverlos a un Product Backlog o al Board de un Sprint concreto.
- ***Sprints***: Es una vista filtrada de los elementos de trabajo basado en un sprint específico. Se pueden asignar elementos de trabajo a un sprint arrastrándolos desde el Backlog.

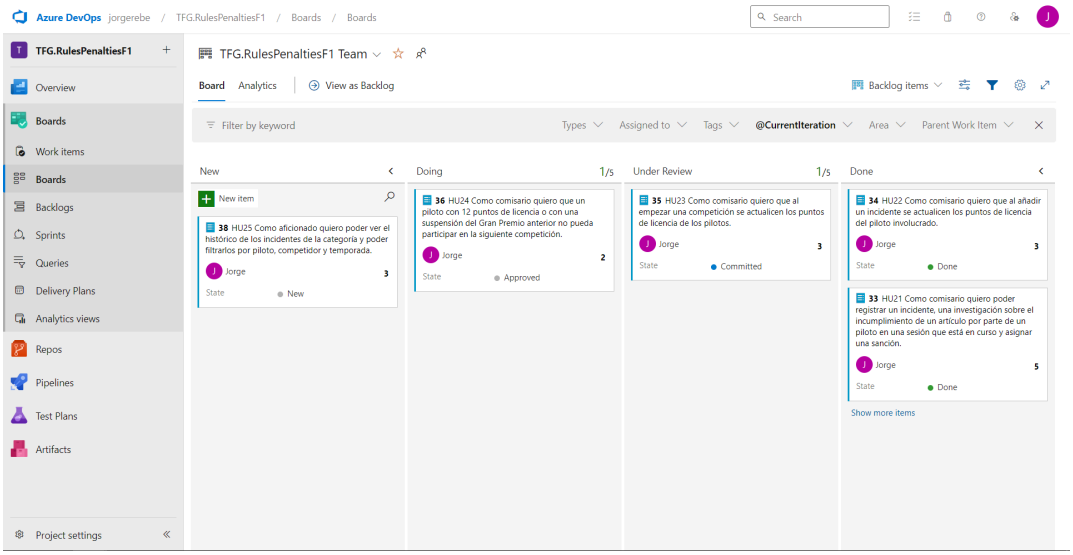


Figura 4.3: Estado del Board durante el Sprint 8. En él se encuentran los elementos de trabajo que se preveían realizar durante el sprint, y el estado de cada uno de ellos: *New* por comenzar, *Doing* en progreso, *Under Review* bajo revisión y *Done* terminados.

Pipelines

Azure Pipelines [55] es una herramienta utilizada para compilar y ejecutar automáticamente las pruebas del código de un proyecto. Soporta la mayoría de lenguajes de programación y tipos de proyecto, y combina integración continua (CI), entrega continua (CD) y pruebas continuas (CT) para compilar, probar y entregar el código en cualquier destino.

La pipeline se define en un fichero de texto en formato *.yml*, realiza unas acciones especificadas cuando se realiza un *push* en la rama o ramas determinadas. En la Figura 4.4 se muestra un ejemplo de una pipeline que se dispara cuando se realiza un *push* en la rama master, compila la aplicación y ejecuta los tests que encuentre.

4.3. Herramientas de documentación y comunicación

4.3.1. Lenguajes de modelado

UML

El lenguaje unificado de modelado (UML, por sus siglas en inglés) [56], un estándar de representación que ha sido usado para el modelado estructural, del comportamiento y adicional de este proyecto. La versión de UML utilizada ha sido la 2.5.1, publicada en diciembre de 2017 y que sigue vigente a día de hoy.

```yaml
trigger:
- master

pool:
  vmImage: 'windows-latest'

variables:
  solution: '**/*.sln'
  buildPlatform: 'Any CPU'
  buildConfiguration: 'Release'

steps:
Settings
- task: NuGetToolInstaller@1

Settings
- task: NuGetCommand@2
  inputs:
    restoreSolution: '$(solution)'

Settings
- task: VSBuild@1
  inputs:
    solution: '$(solution)'
    msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:PackageLocation="$(build.artifactStagingDirectory)"'
    platform: '$(buildPlatform)'
    configuration: '$(buildConfiguration)'

Settings
- task: VSTest@2
  inputs:
    platform: '$(buildPlatform)'
    configuration: '$(buildConfiguration)'
```

Figura 4.4: Fichero de una pipeline de Azure DevOps que compila y ejecuta los tests de una aplicación ASP.NET

A lo largo de todo el documento se muestran diversos diagramas que se han realizado utilizando UML: diagramas de clases, de actividades y de secuencia.

OCL

OCL [57] es un lenguaje formal utilizado para describir expresiones en modelos UML. Típicamente esas expresiones son condiciones invariantes que se deben cumplir para el sistema que está siendo modelado (aunque también puede utilizarse para declarar restricciones sobre operaciones mediante precondiciones y postcondiciones). Hay que aclarar que OCL es un lenguaje sin efectos colaterales, es decir, cuando se evalúan sus expresiones no se puede alterar el estado del sistema en ejecución.

OCL tiene su origen en el hecho de que un diagrama UML no es suficiente para expresar algunas restricciones sobre el sistema que se está modelando. Por tanto, se terminan expresando en lenguaje natural, y esto desemboca en ambigüedades. Para solucionar esto, se podrían expresar utilizando algún lenguaje formal, pero éstos tienen la desventaja de que no son adecuados para personas que no tienen una formación matemática alta. Por todo esto, nació OCL, como un lenguaje formal fácil de leer y escribir, que además está integrado con UML.

4.3.2. Visual Paradigm

Visual Paradigm [58] es una herramienta de modelado y diseño de software, que da soporte al lenguaje de modelado UML. Se ha utilizado para documentar técnicamente este proyecto, a través de diagramas UML (de actividades, de clases, de paquetes, despliegue, secuencia,

etc.), además de para el diseño de las interfaces de usuario. La versión utilizada es la 16.2, ya que se cuenta con una licencia académica para ella.

4.3.3. Overleaf

Overleaf [59] es un editor LaTeX online y colaborativo, que se ha utilizado para redactar este documento. Permite la edición del mismo por varios usuarios de manera simultánea, así como un historial de los cambios realizados. Ha sido una herramienta muy útil para realizar las reuniones de seguimiento del proyecto, ya que permite añadir comentarios al documento, lo cual hace que no se pierdan las mejoras sugeridas por la tutora.

4.3.4. Clockify

Debido a que hay que realizar un conteo de las horas invertidas en el proyecto, se ha utilizado la herramienta gratuita de gestión de tiempo Clockify [60]. Cada vez que trabajaba en el proyecto, activaba un cronómetro asociado a este proyecto y con una etiqueta (según en qué estaba trabajando), y cuando acababa lo paraba de nuevo. Al final del sprint se realizaba el conteo de horas y se anota en el capítulo de seguimiento.

4.3.5. Rocket.Chat

Rocket.Chat [61] es una herramienta de mensajería que ha sido la principal vía de comunicación con la tutora durante todo el proyecto. En particular, se ha utilizado el espacio de trabajo de Rocket.Chat exclusivo de la Escuela de Ingeniería Informática que se pone a disposición de los estudiantes. Esta vía de comunicación ha sido utilizada para tratar asuntos más urgentes de resolver que no podían esperar a la siguiente reunión semanal.

Capítulo 5

Diseño

En este capítulo se exponen las decisiones tomadas con respecto al diseño y a la arquitectura de la aplicación.

5.1. Decisiones de diseño

A lo largo del proyecto se han tenido que tomar decisiones de diseño para poder cumplir con los requisitos no funcionales y restricciones del mismo. En esta sección se muestran las decisiones de diseño tomadas para satisfacer los requisitos y restricciones de la Tabla 2.22.

- **HUNF01**: Para la construcción de la aplicación Web se va a utilizar ASP.NET Core MVC, con la versión .NET 7, como se menciona en la Subsección 4.1.1.
- **HUNF02**: Para que la aplicación sea fácilmente usable se desarollará un manual de usuario completo (ver Apéndice A) y para el desarrollo de la interfaz de usuario se utilizará Bootstrap (Ver Subsección 4.1.10).
- **HUNF03**: Para el almacenamiento de los secretos de la aplicación, como el *connection string* con la base de datos del entorno de producción, se utilizara Azure Key Vault (Ver Subsección 4.1.5).
- **HUNF04**: Se utilizará *Clean Architecture* como arquitectura de referencia para que la aplicación sea lo más mantenible y usable posible.
- **HUNF05**: Para asegurar que la aplicación sea escalable, se desplegará mediante un contenedor Docker de manera que se pueda escalar el número de contenedores fácilmente para distribuir la carga.
- **HUNF06**: Para que el proyecto sea traspasable internacionalmente, el código y la documentación del proyecto se ha realizado en inglés.

- **HUNF07**: La interfaz de usuario estará completamente en inglés para que pueda utilizar la aplicación el mayor número de usuarios posible.
- **HUNF08**: Para asegurar que la aplicación satisface las necesidades del cliente y que las historias de usuario desarrolladas cumplen con la *Definition of Done*, se realizarán tests automatizados unitarios y de aceptación.
- **HUNF09**: Con el objetivo de que el final del proyecto sea un producto mínimo viable funcional, solamente se asegurará que el sistema funciona correctamente en un navegador Chrome para el sistema operativo Windows 10. A pesar de que se desarrollará intentando que el producto sea multiplataforma, la cantidad de pruebas necesarias para comprobar que funciona perfectamente en todos los navegadores y sistemas operativos es inabordable por cuestiones de tiempo.
- **HUNF10**: Antes de cada despliegue de la aplicación se realizarán pruebas automatizadas y sin automatizar para asegurar el correcto funcionamiento del sistema.

5.2. Arquitectura de referencia

Tradicionalmente, la mayoría de las aplicaciones optaban por una arquitectura 'N-Layer', un enfoque de organización de las capas de la aplicación donde las dependencias entre ellas van de arriba hacia abajo.

Típicamente, se utilizan tres capas con esta arquitectura: la de interfaz de usuario (UI), la de lógica de negocio (BLL) y la de acceso a datos (DAL). La de UI sólo interactua con la de BLL, y la de BLL hace llamadas a la DAL.

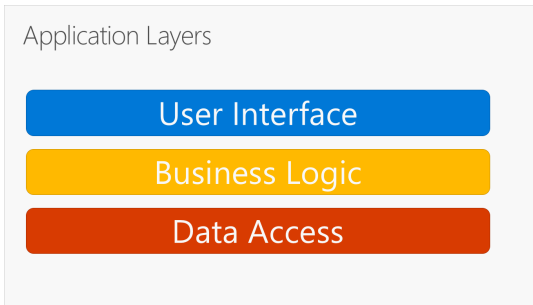


Figura 5.1: N-Layer Architecture [62]

Una desventaja de este enfoque tradicional es que las dependencias van de arriba hacia abajo, lo que significa que la capa UI depende de la BLL, y la BLL depende del DAL. Esto implica que la BLL, que normalmente contiene la lógica más importante de la aplicación depende de los detalles de implementación de acceso a datos (y a veces en la existencia de una base de datos). Hacer pruebas de la lógica de negocio en una arquitectura así es difícil. Sin embargo, el principio de inversión de dependencias (Ver Sección 5.3) se puede utilizar para resolver este problema.

La solución a este problema es una nueva arquitectura, que ha tenido varios nombres a lo largo de los años. Últimamente se la ha llamado *'Onion Architecture'* o *'Clean Architecture'* (este último nombre se utilizará para referirse a ella en este documento).

Clean Architecture pone la lógica de negocio en el centro de la aplicación y, en lugar de que la lógica de negocio dependa del acceso a datos u otros detalles de la infraestructura, se invierte la dependencia: la infraestructura y los detalles de implementación dependen de la lógica de negocio (Application Core). Esto se consigue definiendo abstracciones, interfaces, en el Application Core, las cuales se implementan después en la capa de infraestructura. Una forma de representar esta arquitectura es mediante círculos concéntricos, de forma similar a una cebolla (de ahí el nombre de 'Onion Architecture').

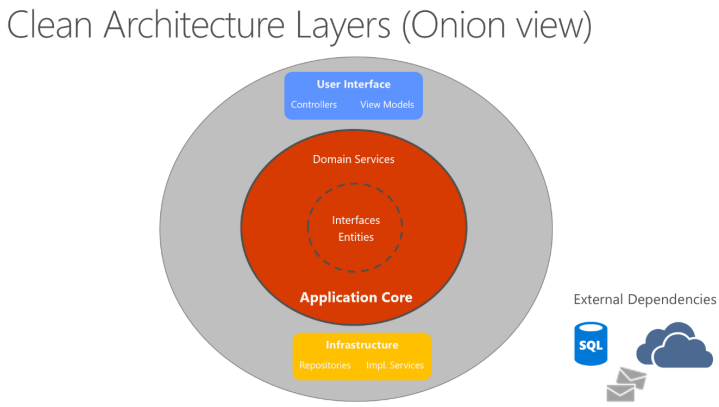


Figura 5.2: Clean Architecture; vista de 'cebolla' [62]

En este diagrama, las dependencias van de fuera hacia dentro. La capa de infraestructura y la de interfaz de usuario dependen de la capa 'Application Core', mientras que ésta última no depende de ninguna. A su vez, la capa de interfaz de usuario y la de infraestructura no dependen una en la otra (necesariamente).

A continuación se muestra una vista más tradicional horizontal de las capas de Clean Architecture.

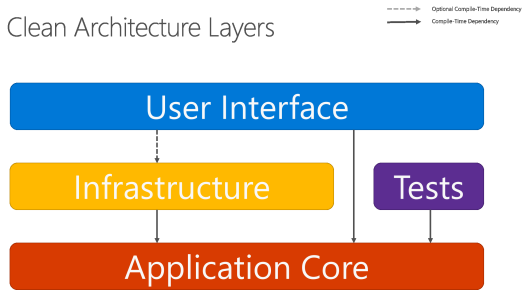


Figura 5.3: Clean Architecture; horizontal layer view [62]

5.3. Principios de Diseño

En esta sección se van a explicar algunos principios de diseño y su uso en el proyecto. Se han tenido en cuenta y aplicado otros muchos principios de diseño durante el desarrollo del proyecto, pero aquí se explican dos relativos a las dependencias que se consideran especialmente relevantes.

5.3.1. Dependencias explícitas

El principio de dependencias explícitas [62] indica que las clases y los métodos deberían de exponer los objetos que necesitan para su correcto funcionamiento de forma clara.

Los constructores de las clases son una oportunidad para que las clases especifiquen los componentes que necesitan para funcionar correctamente. Si una clase puede ser construida sin pasarle ningún parámetro al constructor y permite llamadas a un método que solo funciona si algún componente global está disponible, entonces esa clase está siendo deshonesta con sus clientes.

En este proyecto, los controladores de MVC, los servicios de aplicación y los servicios de dominio especifican en cada uno de sus constructores los servicios o repositorios que necesitarán para funcionar, cumpliendo así con el principio de dependencias explícitas.

5.3.2. Inversión de dependencias

El principio de inversión de dependencias [62] dice que los sistemas más flexibles son aquellos que dependen de abstracciones, en lugar de en concreciones.

El hecho de que las clases dependan de interfaces en lugar de en clases concretas permite que diferentes implementaciones de esas interfaces se puedan utilizar según se necesiten.

Las aplicaciones que utilizan este principio de diseño son más fáciles de probar, más modulares y más mantenibles. El uso de este principio permite la utilización de inyección de dependencias. En este proyecto la inyección de dependencias se realiza utilizando Autofac (Ver Subsección 4.1.9).

Como se ha explicado en el principio anterior, en este proyecto las dependencias se hacen explícitas en los constructores de las clases. Estas dependencias, siguiendo el principio de inversión de dependencias, son interfaces. Por tanto, en el momento de instanciación de la clase, Autofac inyecta la implementación correspondiente de cada interfaz declarada como dependencia.

5.4. Patrones arquitectónicos

A continuación se describen patrones arquitectónicos utilizados en el proyecto, y su uso específico en el mismo.

5.4.1. Cliente Servidor

La arquitectura utilizada para el desarrollo de este proyecto es una arquitectura Cliente-Servidor de tres niveles.

En este estilo arquitectónico contamos con tres niveles [63] lógicos y físicos de computación: el nivel de presentación, el de aplicación y el de datos.

- Nivel de presentación: es el nivel en el que se encuentra la interfaz de usuario, en el que el usuario final interactúa con la aplicación. Su objetivo es mostrar datos al usuario y permitir al usuario introducirlos.
  En el caso de este proyecto, este nivel se ejecuta en un navegador web.
- Nivel de aplicación: en este nivel la información introducida por el usuario se procesa. Se realizan validaciones contra el nivel de datos o reglas de negocio definidas en el nivel de aplicación.
- Nivel de datos: es el nivel donde la información procesada por el nivel de aplicación es almacenado y gestionado. En este proyecto, el encargado de esa tarea sistema gestor de bases de datos SQL Server.

Antes de pasar a describir la arquitectura utilizando otro estilo arquitectónico, es conveniente recordar la diferencia entre nivel y capa. Una capa se refiere a una división funcional del software, mientras que un nivel se refiere a una división funcional del software que se ejecuta en una infraestructura diferente del resto de divisiones.

Por ejemplo, cuando se ejecuta este proyecto en local, la aplicación tiene tres capas, pero un solo nivel. Sin embargo, cuando se utiliza la aplicación en el entorno de producción sigue teniendo tres capas pero pasa a tener tres niveles.

5.4.2. MVC

El patrón arquitectónico Model-View-Controller (MVC) [64] busca conseguir una separación de responsabilidades: en este caso, separar la interfaz de usuario de la lógica de negocio de la aplicación. Para conseguirlo, se divide la aplicación en tres grupos de componentes: Modelos (Model), Vistas (View) y Controladores (Controller).

Estas son las responsabilidades de cada uno de los componentes del patrón:

- Modelo: Representa la información del sistema trabaja, la lógica de negocio.
- Vista: Presenta la información del modelo a través de la interfaz de usuario. La lógica aquí presente debería ser mínima y que sólo realice tareas de presentación del contenido.
- Controlador: Los controladores gestionan la interacción con el usuario, trabajan con el modelo y seleccionan la vista a mostrar

Existen dos variantes para el patrón MVC: la pasiva y la activa. Por un lado, la pasiva se utiliza cuando el controlador modifica el modelo de forma exclusiva, y la vista sólo se actualiza cuando el controlador informa a la vista de un cambio en el modelo. No existe forma de que el modelo notifique cambios en su estado.

Sin embargo, hay ocasiones en las que el modelo cambia de estado sin implicación del controlador. En ese caso, el modelo tendría que notificar a la vista, haciendo uso del patrón observador para evitar que aparezca una dependencia del modelo con la vista.

En la Figura 5.4 se muestran las dependencias de cada uno de los componentes de MVC en su versión pasiva, la utilizada en este proyecto.

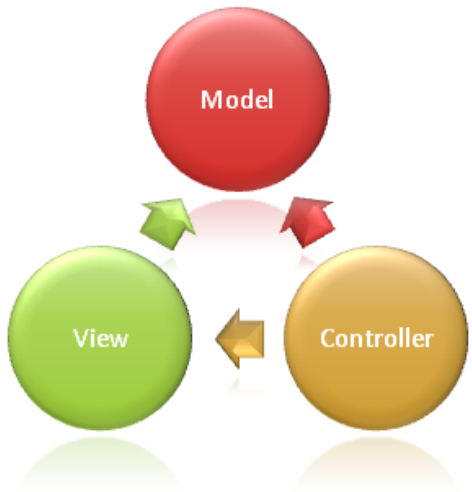


Figura 5.4: Referencias entre los componentes de MVC pasivo [64]

Uso de MVC en este proyecto

MVC es el patrón arquitectónico utilizado para construir aplicaciones con el framework ASP.NET Core MVC, como en el caso de este proyecto. Dado que se está desarrollando una aplicación web cliente-servidor, se utilizará la variante pasiva de MVC.

En ASP.NET Core MVC, las vistas [65] son ficheros con extensión *.cshtml*, cuyo contenido es HTML con código en C# para producir páginas dinámicas. Típicamente, a las vistas se

les pasa un modelo fuertemente tipado (Model), para poder mostrar los datos. A este modelo se le denomina *viewmodel*, que no hay que confundir con el viewmodel del patrón MVVM [66].

Lo recomendado es que estos viewmodels que se pasan a las vistas sean viewmodels *Plain Old CLR Object* (POCO)[67] , casi sin lógica definida (sólo el mapeo de viewmodel a modelo de la lógica de negocio).

Estos viewmodels que se pasan a las vistas podrían ser las mismas clases utilizadas para la lógica de negocio, pero normalmente se utilizan clases distintas por lo siguiente:

- Al utilizar modelos distintos, las vistas pueden cambiar independientemente de la lógica de negocio y de la de acceso a datos.
- Por beneficios en la seguridad, al utilizar *model binding* [68] y validación para los datos enviados a la aplicación [69].

5.5. Patrones de diseño

A continuación se explicarán los patrones de diseño utilizados y su aplicación en este proyecto.

5.5.1. Repository

El patrón *'Repository'* [70] [71] es un patrón DDD que busca mantener los asuntos relacionados con la persistencia fuera del modelo de dominio del sistema. Para ello, se definen abstracciones (interfaces) en la capa del dominio, que serán luego implementadas en la capa de infraestructura. De esta manera, las clases del dominio son independientes de los detalles de implementación de la persistencia.

Un repositorio encapsula una serie de objetos almacenados en una base de datos junto con operaciones que se pueden ejecutar sobre ellos, además de la separación clara y en una dirección, de la dependencia entre el dominio y el alojamiento de los datos.

Uso de Repository en este proyecto

En el caso de este proyecto, esas implementaciones de acceso a datos se realizan utilizando el *DbContext* de *Entity Framework* (Ver Subsección 4.1.2). En el constructor de cada repositorio existente en el proyecto se inyecta una instancia de *DbContext*, a través de la cual se realizarán las operaciones CRUD (Create, Read, Update, Delete) que se necesiten.

Para reducir al máximo el número de repositorios particulares por *aggregate* (Ver Subsección 5.5.4), se utilizará genericidad. Existirá un repositorio genérico que implementará las

operaciones CRUD típicas, y del que heredarán repositorios particulares de algunos *aggregate*, que se crearán cuando se necesiten operaciones más particulares.

5.5.2. Factory Method

El Método Factoría [72] es un patrón de diseño creacional, utilizado para crear un objeto mediante un método de fábrica en lugar de utilizando el operador new. Se provee una interfaz a través de la cual se puede crear un tipo de objeto cuando la clase que quiere crearlo no puede anticipar de qué tipo es el objeto que se quiere crear.

En la Figura 5.5 se muestra la estructura general del patrón. La clase *Creator* es la factoría, y cada una de sus subclases es la encargada de crear un tipo de producto. En este caso, *ConcreteCreator* es la encargada de crear *ConcreteProduct*.

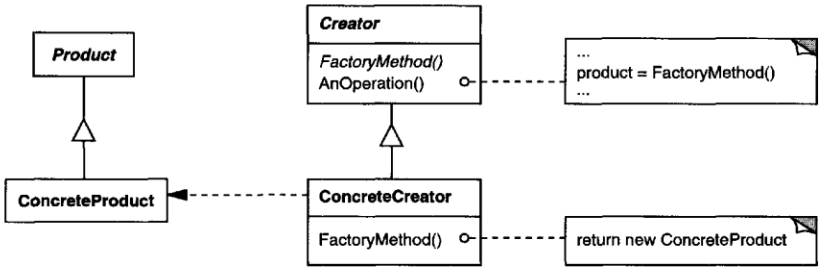


Figura 5.5: Patrón Factoría [72]

Uso de Factory Method en este proyecto

Cada tipo de penalización (Descalificación, Suspensión, Drive Through, etcétera) tiene su propio viewmodel asociado para mostrarlo en la vista. Sin embargo, cuando se traen una penalización de la base de datos su tipo es 'Penalización'. Es decir, no se sabe qué penalización es. Por tanto, no se sabe que viewmodel construir para pasarlo a la vista.

Para ello se ha construido una factoría simplificada, que primero evalúa cuál es el tipo de penalización que se ha traído de la base de datos. Después, según ese tipo, se construye el viewmodel correspondiente y se le asignan sus propiedades para poder mostrar su información en la vista posteriormente.

Esa evaluación de tipo es posible gracias a la existencia de una columna en la tabla de la base de datos que guarda la información de las penalizaciones. Esa columna se llama *Discriminator* (Ver Sección 5.6), y lo que guarda es el tipo de la penalización de una determinada fila de la tabla.

Gracias a esa columna, es posible conocer el tipo de la penalización y por tanto construir el viewmodel correcto para pasar a la vista. La clase con el método factoría aquí explicado es 'PenaltyViewModelFactory' (Ver repositorio de código en Anexo B).

5.5.3. Domain Event

Domain Event (Evento del Dominio) [73] [74] es un patrón de diseño DDD utilizado para representar algo que ha ocurrido en el pasado, junto con una acción a tomar cuando dicho evento ha ocurrido.

Cuando se crea un evento del dominio, se pasa a un despachador de eventos que buscara el manejador (o manejadores, puede haber más de uno) correspondiente al evento, que se encargarán de realizar las acciones necesarias.

Uso de Domain Events en este proyecto

En este proyecto, existirán dos eventos del dominio:

- Cuando un comisario registre un incidente se creará un evento, para el que existirá un manejador de eventos que deberá actualizar los puntos de licencia del piloto involucrado. Ese manejador de eventos comprobará si se ha impuesto una sanción en el incidente, y si es así, si esa sanción incluye puntos de licencia se sumarán a los que tenga en ese momento el piloto.
- Cuando comience una competición se lanzará un evento, con un manejador de ese evento cuya responsabilidad será actualizar los puntos de licencia de todos los pilotos. Por un lado, restaurará a 0 los puntos de licencia de todos los pilotos con 12 puntos en ese momento que no hayan sido sancionados en la competición anterior. Por otro, recalculará los puntos de todos los pilotos que tuvieran entre 1 y 11 puntos (ambos incluidos) en ese momento, teniendo solo en cuenta los puntos ocurridos en los últimos 12 meses.

5.5.4. Aggregate Pattern

Aggregate (agregado) [75] es un patrón de diseño DDD. Un agregado es un grupo de objetos que se puede tratar como si fuera uno solo. Por ejemplo, un pedido y sus líneas de productos son objetos separados, pero es útil tratarlos como un único agregado.

Cada agregado [76] tiene una entidad como su raíz, y se nombran de acuerdo a ella. Los hijos de los agregados no se persisten solos, sino como parte del agregado. Además, las raíces de los agregados son las encargadas de la validez del agregado completo.

Uso de Aggregate Pattern en esta proyecto

En la Sección 5.7 se puede ver como las entidades del dominio de la aplicación se han agrupado en agregados.

5.5.5. Page Object Model

Page Object Model [77] es un patrón de diseño utilizado para reducir la duplicidad de código y mejorar la mantenibilidad de los tests automatizados. Un Page Object es una clase que sirve como una interfaz de una página de la aplicación que se está probando. Contiene sus elementos, botones y acciones que se pueden realizar con ellos. Una vez definido, los tests pueden hacer uso de esa clase Page Object cuando necesiten interactuar con la interfaz de usuario de esa página. Si esa interfaz de usuario cambia en algún momento, los tests no tendrán que cambiar, sino que solo lo hará el Page Object correspondiente.

Existen dos ventajas principales en el uso de este patrón:

- Hay una separación limpia entre el código del test y el código específico de la página, de manera que los tests no tengan conocimiento sobre los elementos específicos de la interfaz de usuario.
- Hay un único sitio donde se encuentran los servicios y operaciones de una página, en lugar de tener esos servicios dispersos por los tests, e incluso duplicados.

Uso de Page Object Model en este proyecto

En el repositorio del código de la aplicación (Ver Anexo B) se puede ver el uso del patrón Page Object Model en el proyecto.

5.5.6. Driver Pattern

El patrón *Driver Pattern* [78] ofrece una capa adicional de abstracción entre la definición de los pasos de los tests de aceptación y el código de automatización de los tests.

Tener el código de la automatización de los tests hace que sea más fácil de mantener y más fácil de entender, incluso para gente sin conocimiento técnico sobre el desarrollo de software. En lugar de ver un gran número de líneas de código, se ve una llamada a una clase Driver que ejecutará el código necesario para el test.

Uso de Driver Pattern en este proyecto

En el repositorio del código de la aplicación (Ver Anexo B) se puede ver el uso del patrón Driver Pattern en el proyecto.

5.6. Modelo de Datos

En la Figura 5.6 se puede ver el modelo de datos desarrollado para la aplicación. Algunas restricciones como claves alternativas o columnas que admiten valores nulos se han omitido, pero se pueden consultar en el archivo *RulesPenaltiesF1DbContextModelSnapshot.cs*, clase autogenerada por Entity Framework con el modelo de datos utilizado en el sistema (Ver repositorio en el Anexo B).

En cuanto al modelo de datos mostrado en el diagrama, lo más relevante a comentar es la decisión de la implementación de la herencia de las penalizaciones. Dado que los incidentes van a tener una asociación con una penalización, ha de haber una tabla para la superclase. En cuanto a tablas para las subclases, aunque es cierto que por cada registro va a haber varias columnas nulas, habrá un número de tuplas muy reducido en la tabla por lo que no se considera necesario tener una tabla para cada subclase.

Por tanto, la jerarquía de penalizaciones se implementará mediante el patrón tabla por jerarquía (Table Per Hierarchy - TPH), y para distinguir al tipo que cada fila representa se utilizará una columna adicional, denominada discriminador.

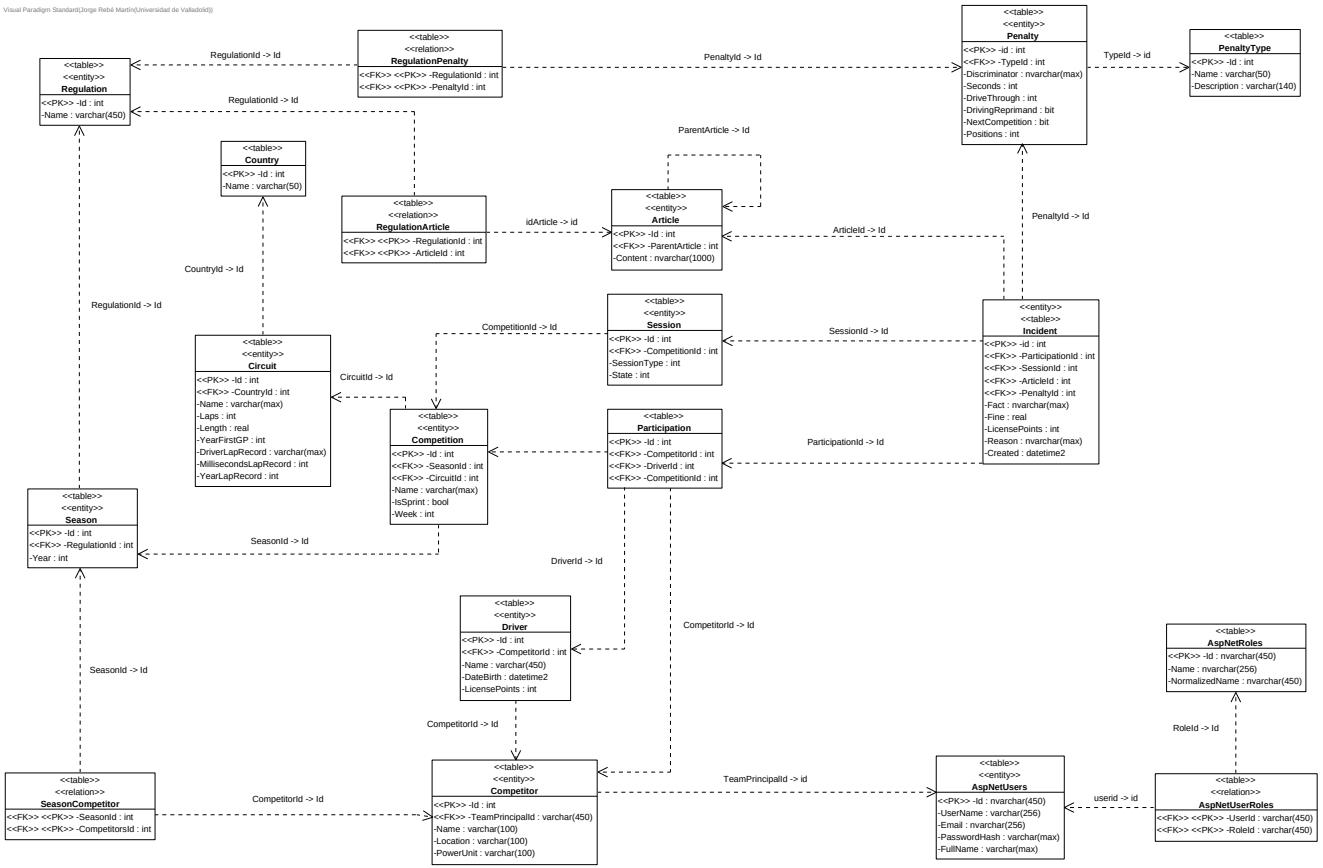


Figura 5.6: Modelo de datos

5.7. Diseño Arquitectónico

Como se ha mencionado en la Seccción 5.2, se va a utilizar *Clean Architecture* como arquitectura de referencia. En esta sección se mostrará la arquitectura de la aplicación mediante diagramas que representan la descomposición modular y las dependencias entre módulos (*Decomposition & Uses Style*), explicando cada capa, sus componentes, dependencias entre ellos y su propósito.

En la Figura 5.7 se muestra la arquitectura general de la aplicación, compuesta de las siguientes tres capas.

- **Core**: Es la capa de la lógica de negocio (*Application Core* en *Clean Architecture*), donde residen las entidades, algunos servicios del dominio y eventos del dominio. También están presentes las interfaces de los repositorios y los servicios del dominio.
- **Infrastructure**: Es la capa de acceso a datos (*Infrastructure* en *Clean Architecture*), donde estará la implementación de las interfaces de los repositorios de la capa *Core*, junto con algunas configuraciones de la base de datos, etcétera.
- **Web**: Es la capa de interfaz de usuario, (la capa *User Interface* en *Clean Architecture*), donde residen las vistas, controladores, viewmodels y servicios de la aplicación.

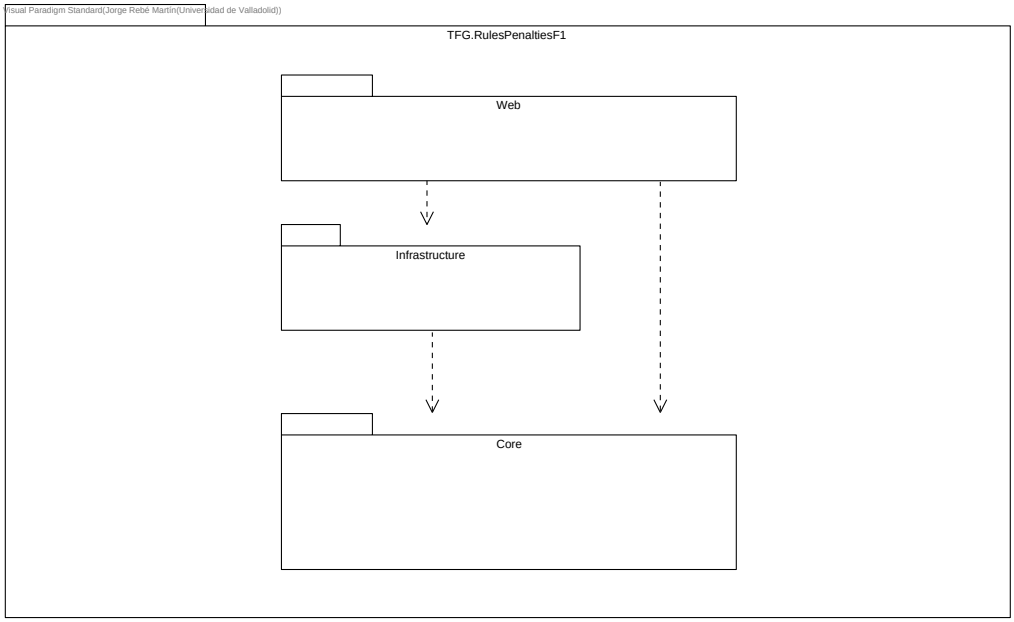


Figura 5.7: Arquitectura general de la aplicación

Dentro de la capa *Core*, tenemos las entidades, interfaces y servicios del dominio. A continuación se explica cada uno de estos componentes de la capa *Core*:

- **Entities**: En este paquete estarán todas las clases del dominio, donde se encuentra la lógica de negocio. Dado que se está siguiendo DDD, las clases del dominio estarán agrupadas en agregados. Las raíces de los agregados heredarán de *EntityBase* e implementarán la interfaz *IAggregateRoot*.
- **Interfaces**: En este paquete estarán las interfaces que deberán implementar los repositorios en la capa de infraestructura, así como las interfaces de los servicios de dominio. También estará la interfaz que implementará el despachador de eventos de dominio y la interfaz *IAggregateRoot*.
- **Services**: En este paquete estarán todos los servicios de dominio [79], cuyo objetivo es realizar modificaciones sobre el dominio que no son responsabilidad de una entidad.

En la Figura 5.9 se puede ver el *Decomposition* & *Uses Style* de las entidades del dominio, así como su agrupación en agregados representados por paquetes en el diagrama. Las clases que no están dentro de un paquete, son un agregado con una única clase. No se las ha metido dentro de un paquete por espacio y claridad en el diagrama.

Dentro de la capa *Infrastructure*, tenemos la implementación del acceso a datos, configuración de la base de datos y migraciones. A continuación se explica cada uno de estos componentes de la capa:

- **Data**: En este paquete está toda la implementación de acceso a datos, que está en repositorios que implementan las interfaces declaradas en la capa *Core*. Estos repositorios heredan de la clase *EfRepository* y utilizan *RulesPenaltiesF1DbContext*.

  En este paquete también hay configuraciones de la base de datos sobre cada una de las tablas (restricciones de unicidad, declaración de claves primarias, columnas requeridas, etcétera). También hay convertidores de valores [80], cuya función es mapear propiedades de una clase a columnas de una tabla cuando no son del mismo tipo. Por ejemplo, el mapeo de un tipo de datos 'sólo fecha' en una propiedad de una clase a un tipo de datos 'fecha y hora' en la base de datos.
- **Identity**: En este paquete se encuentra la clase *ApplicationUser*, que representa a un usuario del sistema. Hereda de *IdentityUser* para añadir nuevas propiedades, como el nombre completo del usuario.
- **Migrations**: En este paquete se encuentran las clases donde se ven las variaciones del modelo de datos según ha ido evolucionando la aplicación.

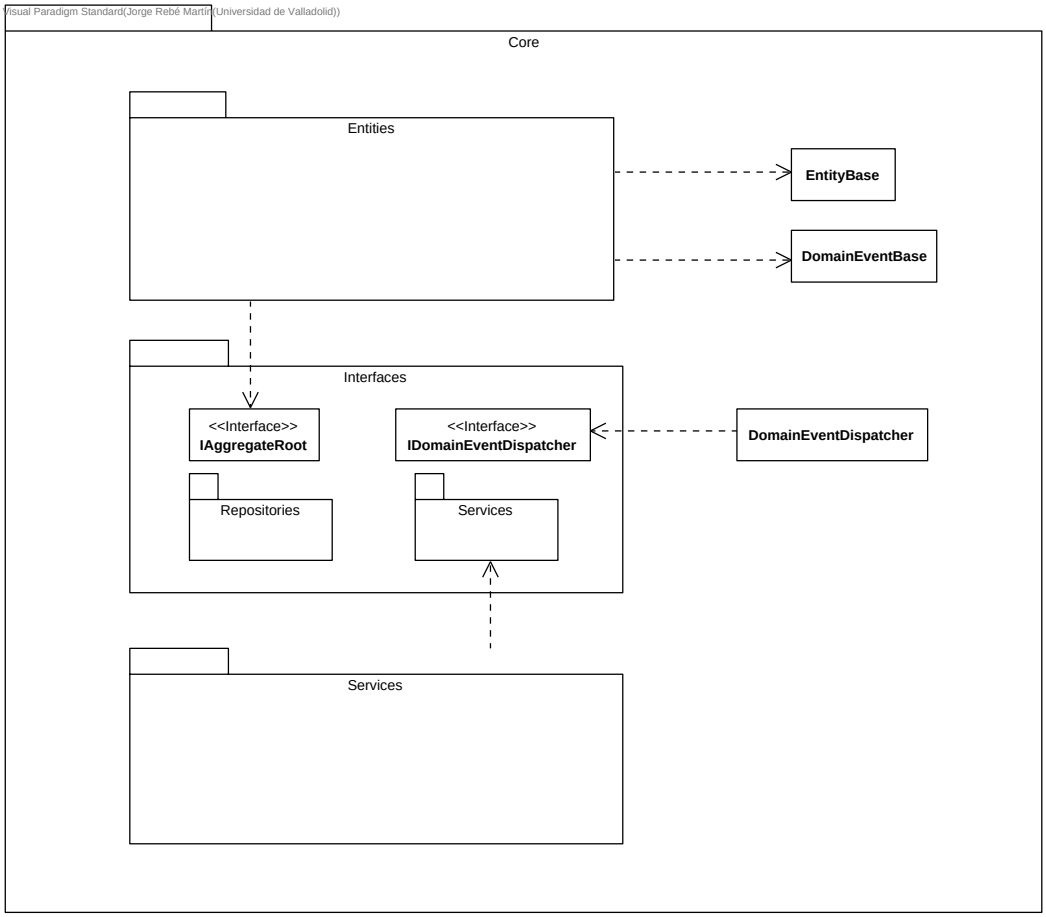


Figura 5.8: *Decomposition & Uses Style* de la capa 'Core'

Dentro de la capa *Web* tenemos la interfaz de usuario, con todos sus componentes que se explican a continuación:

- **Controllers, Views y ViewModels**: En estos paquetes están los componentes de la implementación de MVC en este proyecto.
- **Interfaces y Services**: En el paquete *Interfaces* están las interfaces de los servicios de aplicación que se implementarán en las clases de la capa *Services*. El objetivo de estos servicios es que los controladores dependan solo de los viewmodels, y deleguen en estos servicios la interacción con los repositorios y la conversión de objetos del dominio a su viewmodel correspondiente.
- **Areas e Identity**: Las *Areas* en ASP.NET son estructuras de organización para agrupar funcionalidad común. Dentro de Identity hay componentes de interfaz de usuario

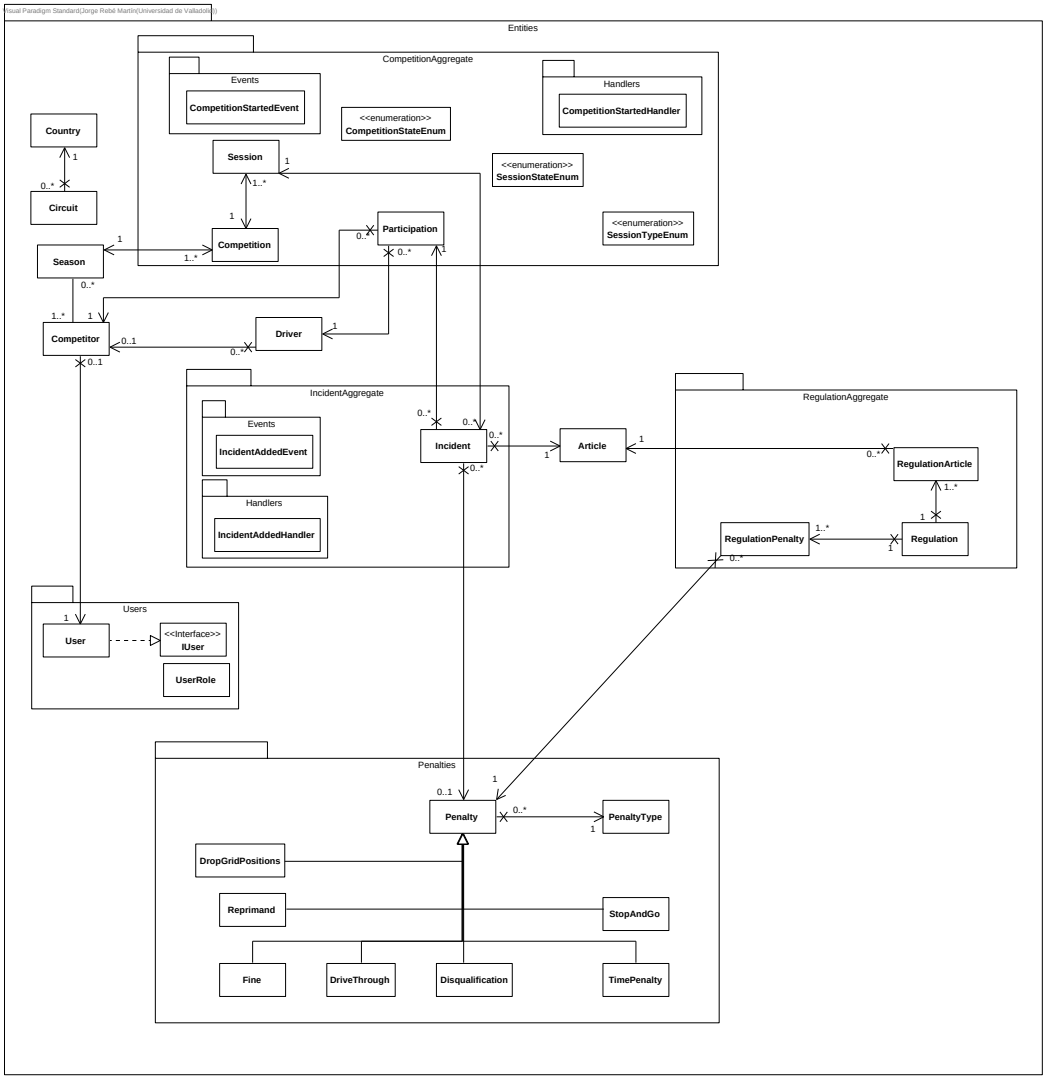


Figura 5.9: *Decomposition & Uses Style* de la capa 'Core'

para la gestión de usuario, generados automáticamente al importar al proyecto el paquete Identity.

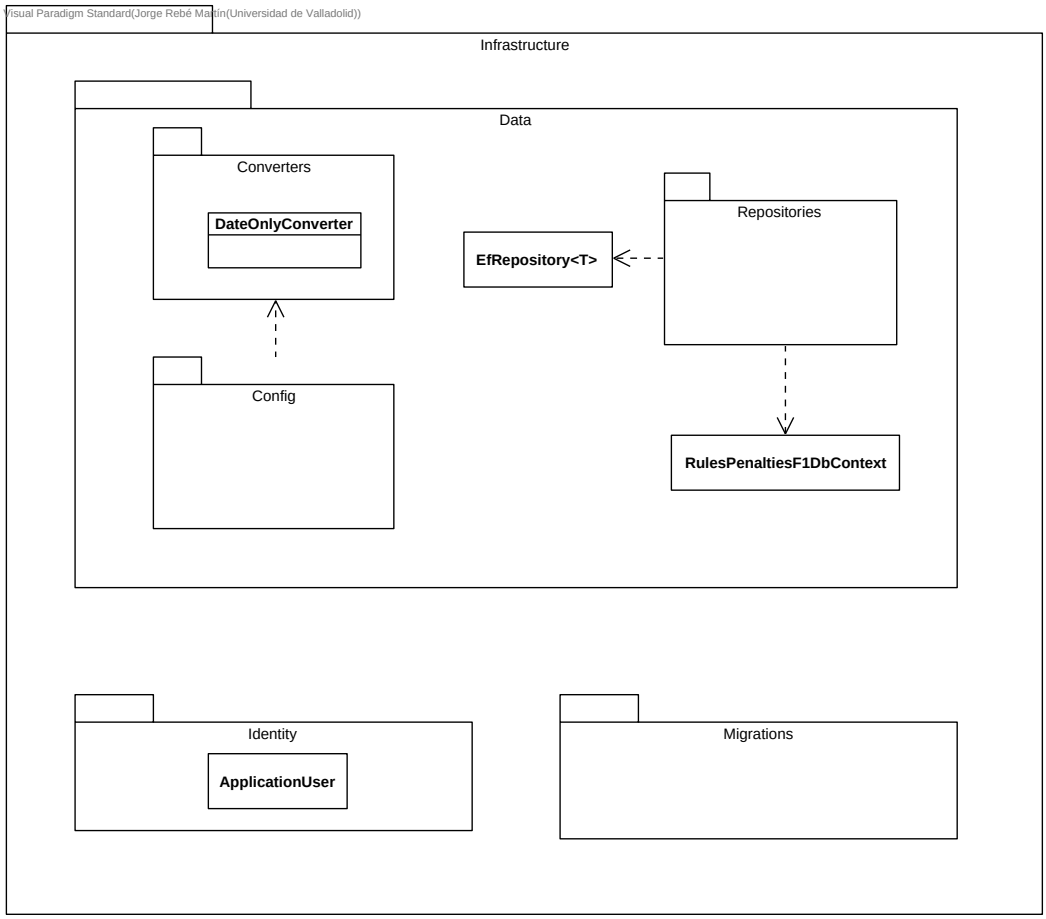


Figura 5.10: *Decomposition & Uses Style* de la capa 'Infrastructure'

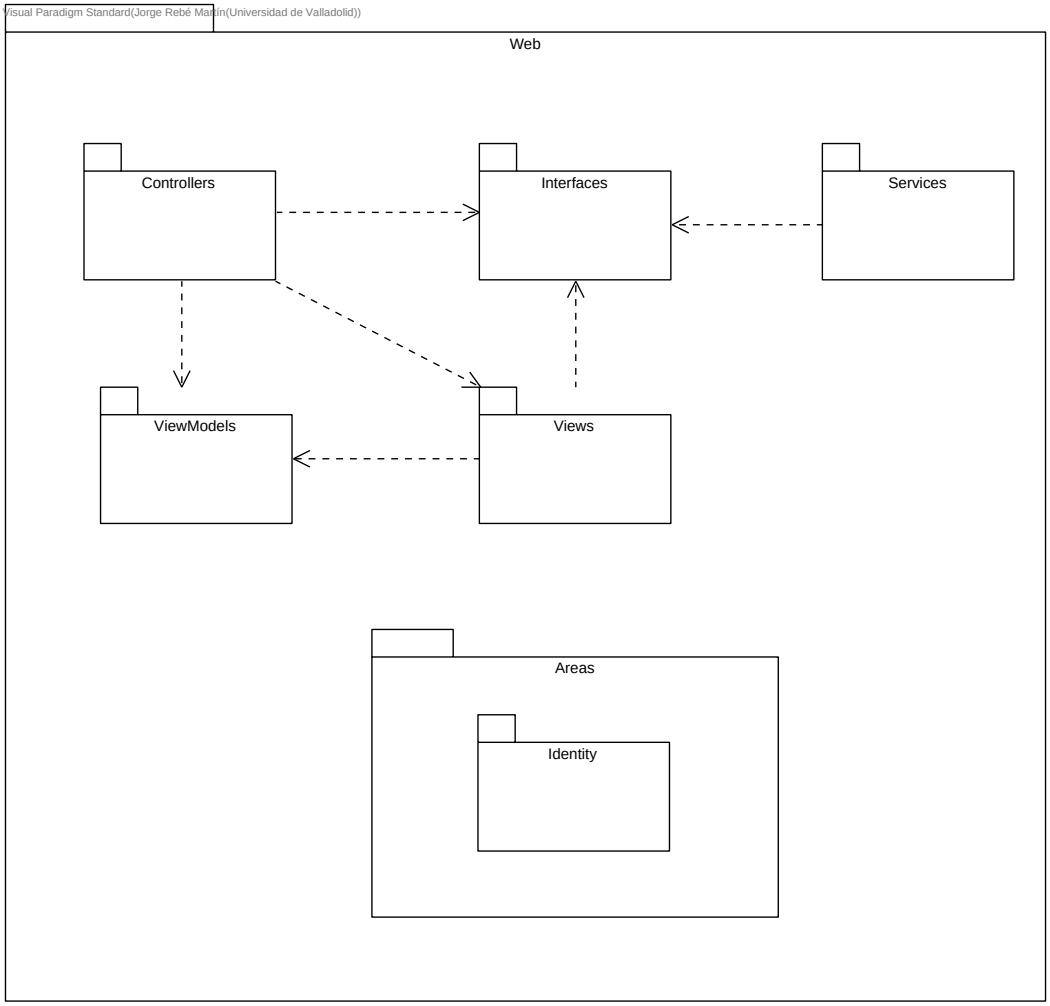


Figura 5.11: *Decomposition & Uses Style* de la capa 'Web'

5.8. Diseño del despliegue

Durante el transcurso del proyecto se trabajará principalmente con dos despliegues: primero, un despliegue en local, en el que la aplicación se ejecutará en un servidor kestrel levantado en la máquina local, que se conectará a un servidor de base de datos SQL Server local, y será accedido por un navegador web en local (Chrome).

En la Figura 5.12 se muestra el diagrama de despliegue correspondiente al entorno de desarrollo en local.

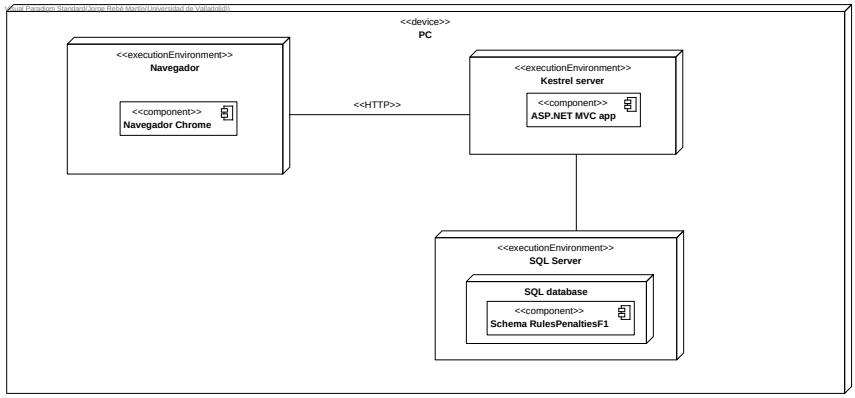


Figura 5.12: Despliegue en el entorno local

Por otro lado, tenemos el despliegue del entorno productivo. Por una parte está el entorno de Azure que aloja nuestra 'Azure Container App', que es la aplicación desplegada como un contenedor en Azure. La aplicación se comunicará con una 'SQL Database' también alojada en Azure, en un 'SQL Server'. Por último, tenemos al cliente, que será un dispositivo con sistema operativo Windows 10, que accederá a través de un navegor Chrome a la aplicación alojada en el contenedor.

En la Figura 5.13 se muestra el diagrama de despliegue correspondiente al entorno de producción.

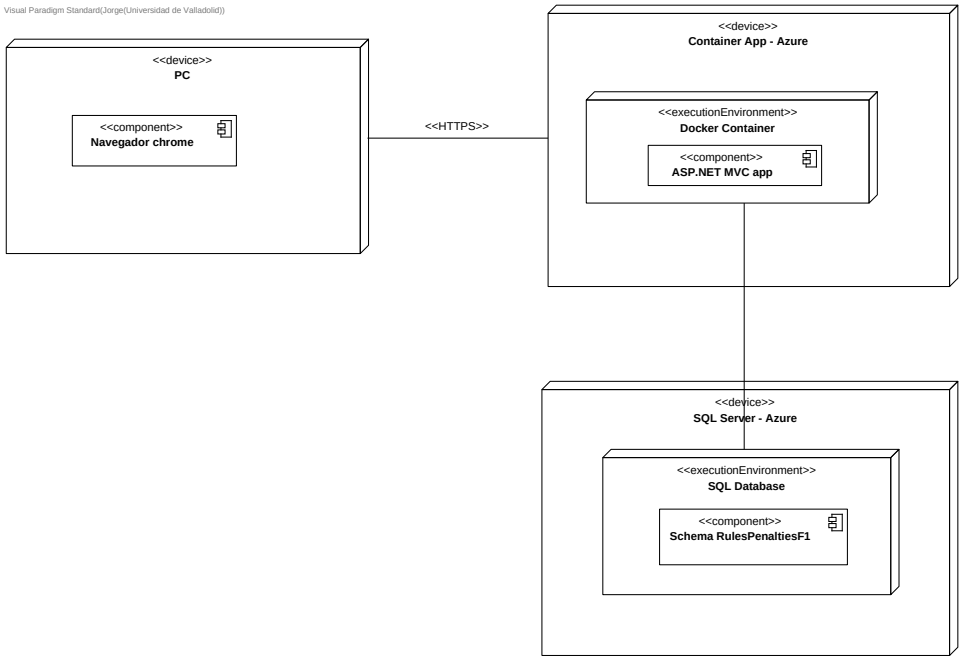


Figura 5.13: Despliegue en el entorno de producción

5.9. Diseño de la comunicación entre objetos

Para explicar cómo se comunican los objetos dentro de la aplicación, se van a mostrar y explicar los diagramas de secuencia correspondientes a la historia de usuario **HU03: Como comisario quiero poder crear un reglamento antes del inicio de una temporada, asignándole los artículos y las sanciones que lo formarán.** Para el resto de historias de usuario no se mostrarán diagramas de secuencia, pero la comunicación entre los objetos es análoga.

El diagrama de secuencia correspondiente a la secuencia principal de la historia de usuario se encuentra en la Figura 5.14. Se puede ver cómo es necesario que el comisario haya iniciado sesión en la aplicación. Se solicita al controlador la página para la creación de reglamentos, y lo primero que hace el controlador al recibir la llamada es recoger los artículos y penalizaciones que puedan formar parte de un reglamento, y devuelve una vista conteniendo los campos necesarios para crear un reglamento y los artículos y penalizaciones previamente recogidos. Esta recuperación de artículos y penalizaciones se explicará posteriormente en otro diagrama de secuencia.

Cuando el comisario ha completado el formulario, se realizan comprobaciones para ver si el reglamento introducido es válido, y si es así, se comprueba que no exista ya un reglamento con el nombre especificado (la comprobación de unicidad del nombre se explicará en otro diagrama de secuencia). Si alguna de estas dos condiciones falla, se muestran los errores correspondientes y la secuencia termina.

Si la entrada del comisario se ha validado correctamente, se convierte el viewmodel en una entidad gracias a un método estático de la clase *RegulationViewModel*, y se le pide al servicio correspondiente que se persista el nuevo reglamento (se explicará en otro diagrama de secuencia).

En la Figura 5.15 se muestra el diagrama de secuencia correspondiente al interaction use *PopulateListArticlesAndPenalties* de la secuencia principal. El servicio de aplicación *RegulationViewModelService* recibe una llamada a la función *ExistsRegulationWithName*, que toma como parámetro el nombre de un reglamento (en este caso el nombre del reglamento que quiere crear el comisario). Lo que hace entonces este servicio es preguntarle al repositorio correspondiente, *RegulationRepository* por la existencia de un reglamento con el nombre especificado. La interacción del repositorio con la base de datos no se expone en este diagrama, pero se explicará en uno posterior.

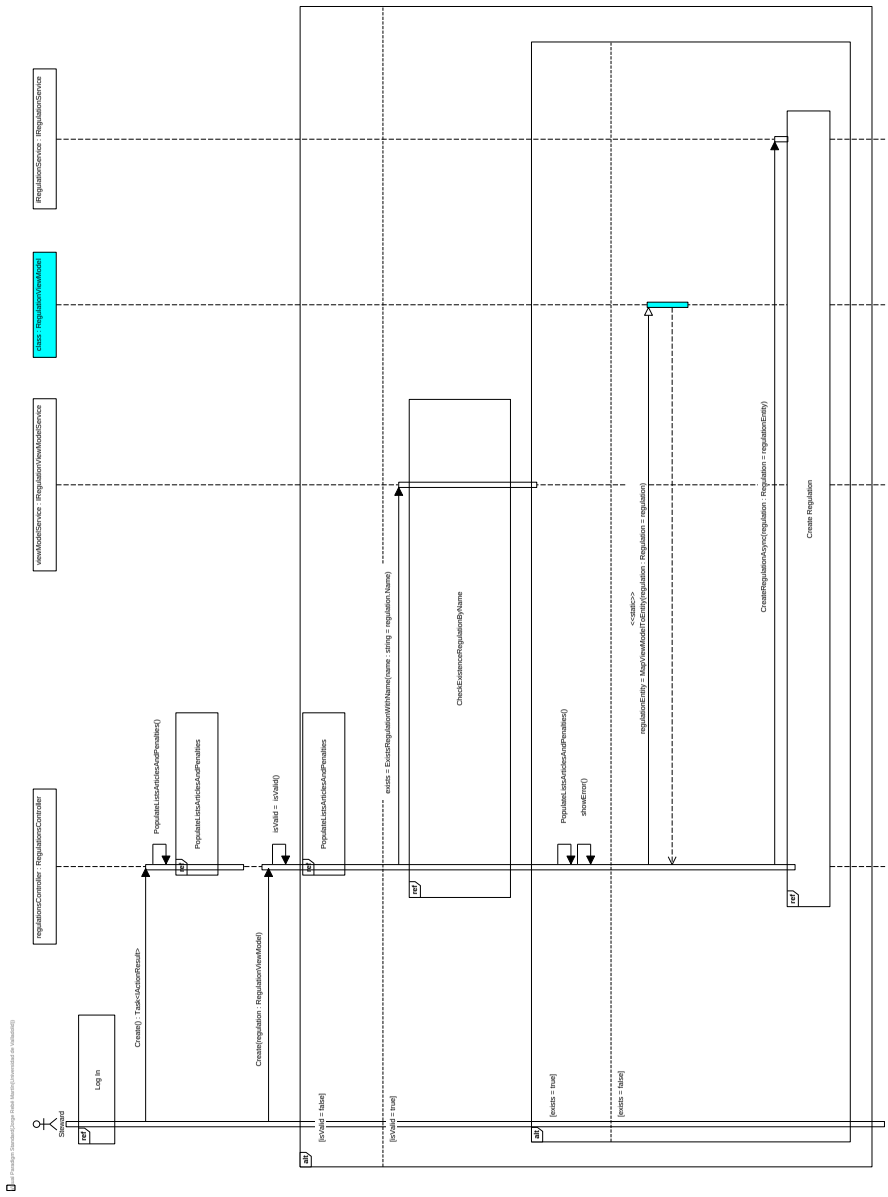


Figura 5.14: Diagrama de Secuencia Principal - HU03

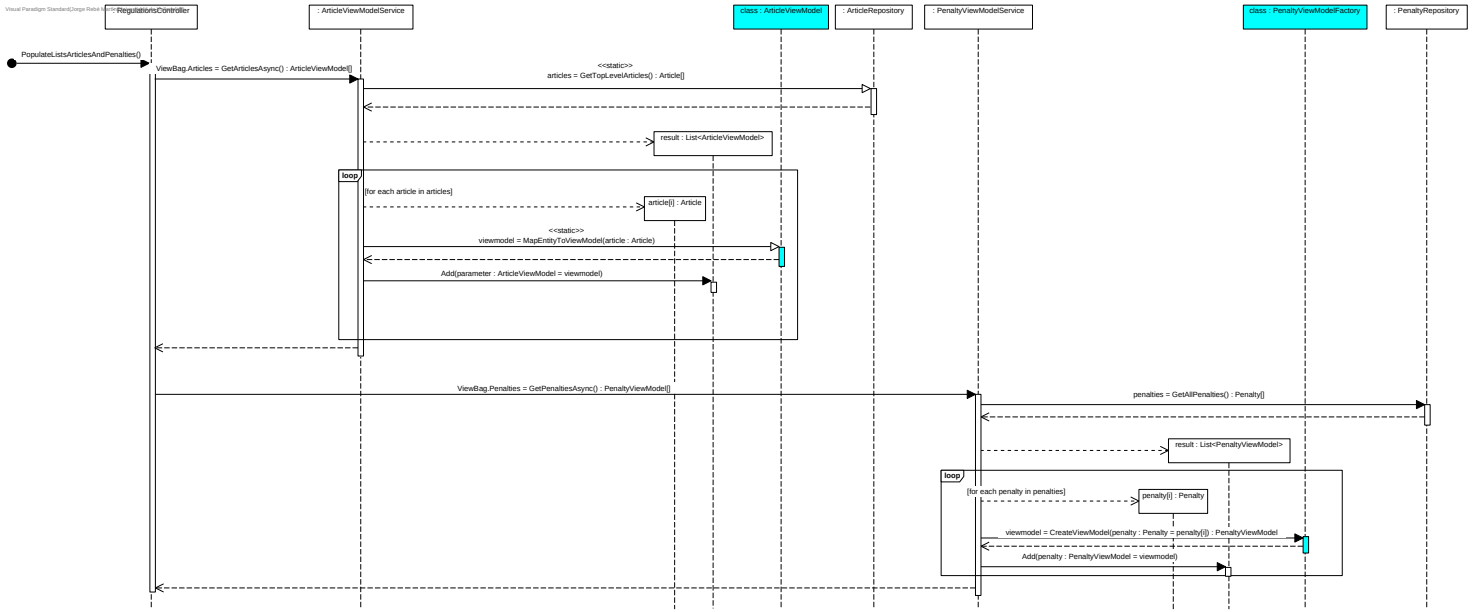


Figura 5.15: Diagrama de Secuencia de 'Poblar artículos y penalizaciones' - HU03

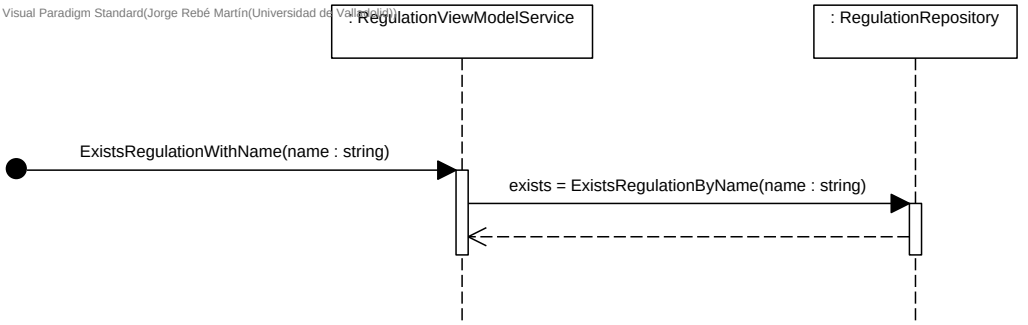


Figura 5.16: Diagrama de Secuencia de 'Comprobar existencia de reglamento por nombre' - HU03

Por último, se va a explicar el diagrama de secuencia correspondiente al interaction use de la secuencia principal *Create Regulation*. El controlador realiza una llamada al servicio de dominio que implementa la interfaz *IRegulationService*, pasándole como argumento el reglamento creado tras las validaciones realizadas. Cuando este servicio recibe el reglamento, le pide al repositorio genérico que lo persista.

Tras esto, el repositorio hace dos llamadas a la clase *RulesPenaltiesF1DbContext*, que mantiene la sesión con la base de datos (Ver Sección 5.7). Se añade la entidad al registro de entidades de las que mantiene el registro la clase *RulesPenaltiesF1DbContext* (método Add), y después se hace la llamada a *SaveChangesAsync* para que las entidades con cambios sin guardar se persistan en la base de datos.

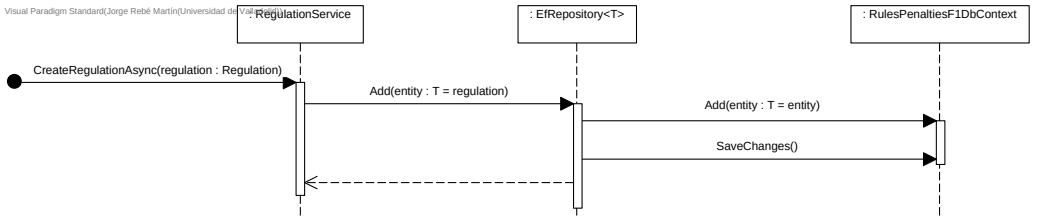


Figura 5.17: Diagrama de Secuencia de 'Crear Reglamento' - HU03

Para mostrar mejor las relaciones y las dependencias entre las distintas clases, interfaces y paquetes, en la Figura 5.18 se puede ver el diagrama *Decomposition&Uses Style* asociado a la historia de usuario HU03.

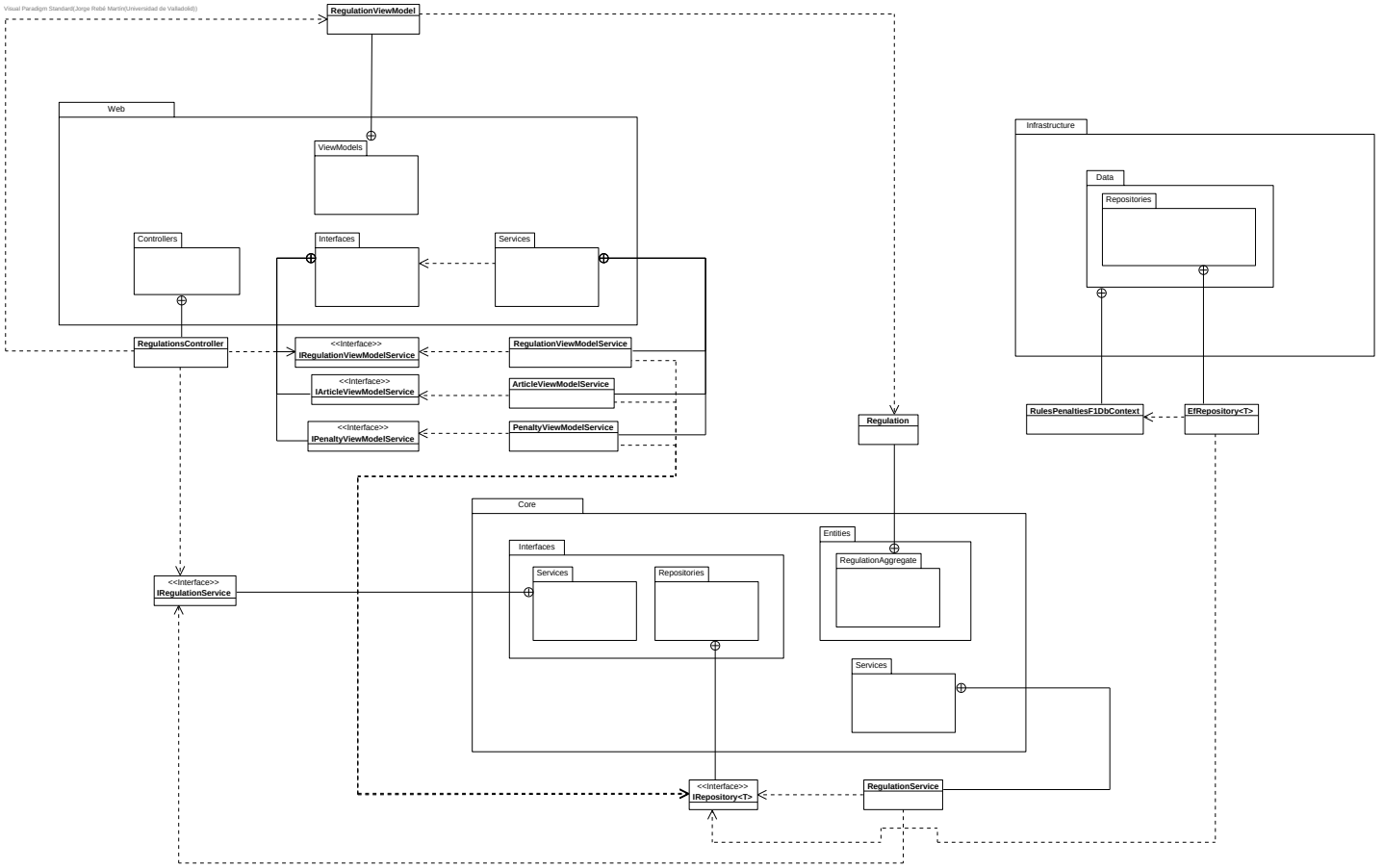


Figura 5.18: *Decomposition & Uses Style* - HU03

5.10. Diseño de la interfaz de usuario

A continuación se mostrará el diseño de las interfaces de usuario para cada historia de usuario, realizadas utilizando la herramienta 'Web Wireframe' de Visual Paradigm. De la Figura 5.19 a la Figura 5.32 se muestran estos diseños.

En la Subsección 6.3.2 se explicará el uso de ChatGPT para la generación y refinamiento de las interfaces de usuario mostradas en esta sección.

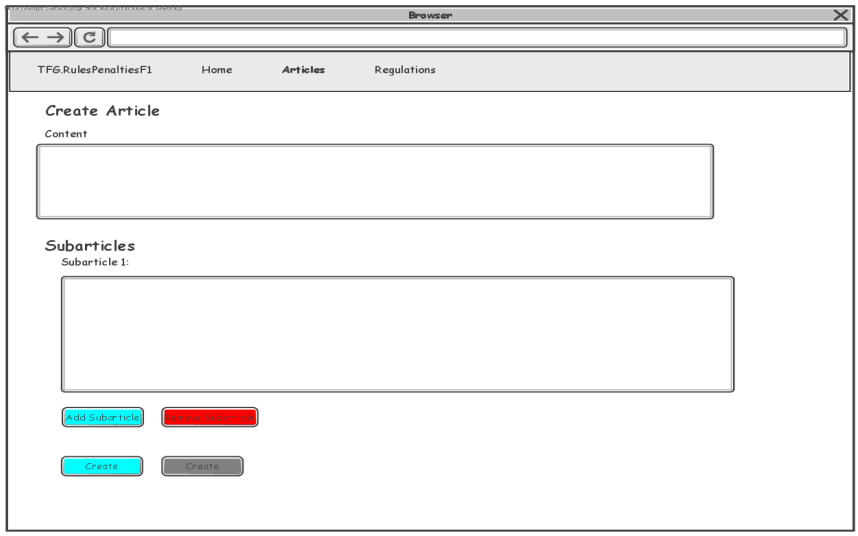


Figura 5.19: Interfaz de usuario HU01

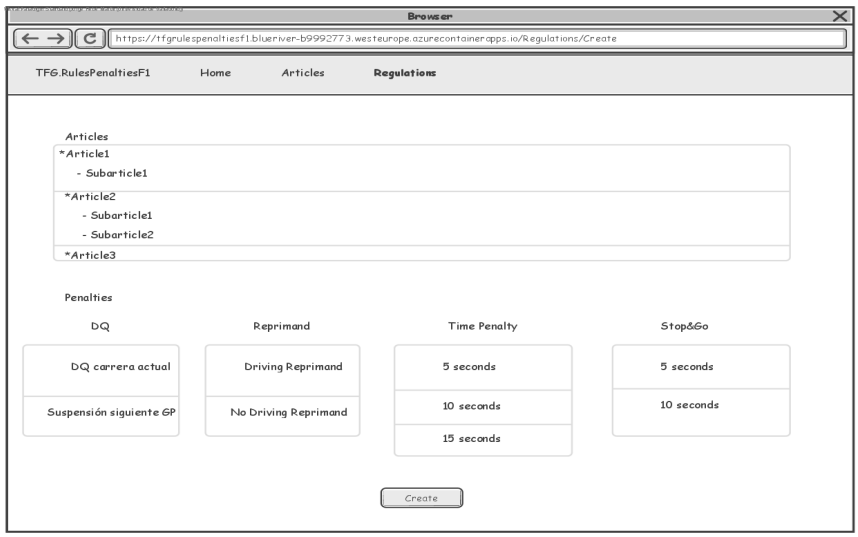


Figura 5.20: Interfaz de usuario HU03

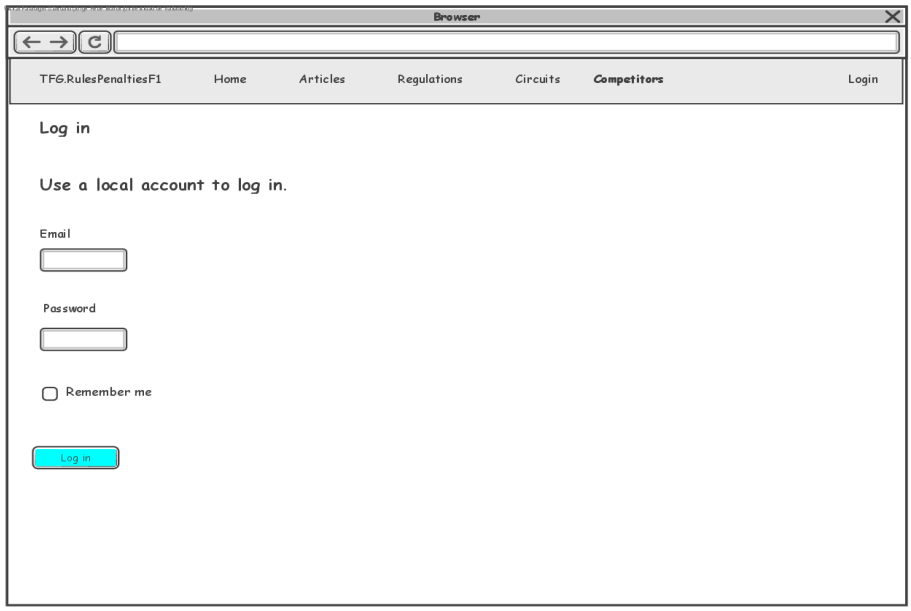


Figura 5.21: Interfaz de usuario H06

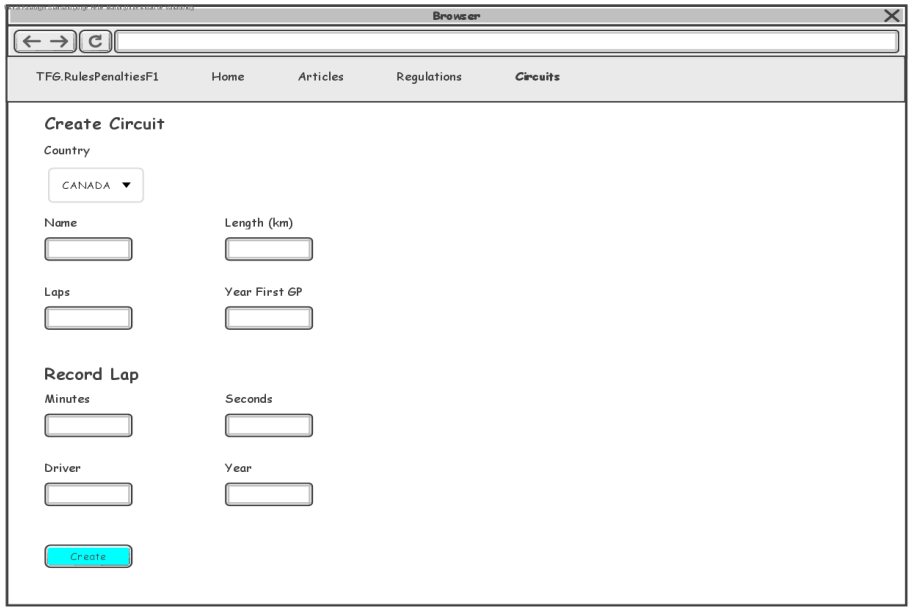


Figura 5.22: Interfaz de usuario HU09

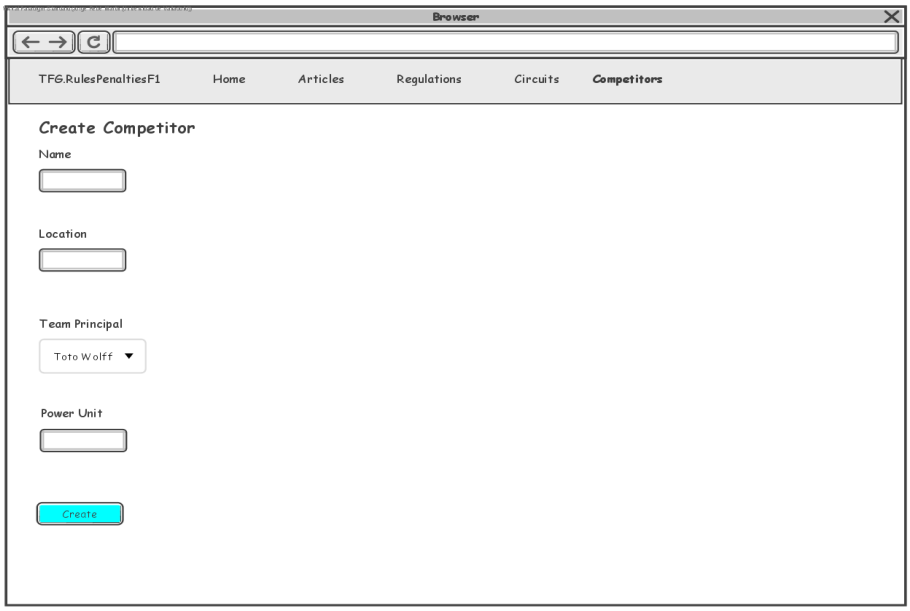


Figura 5.23: Interfaz de usuario HU10

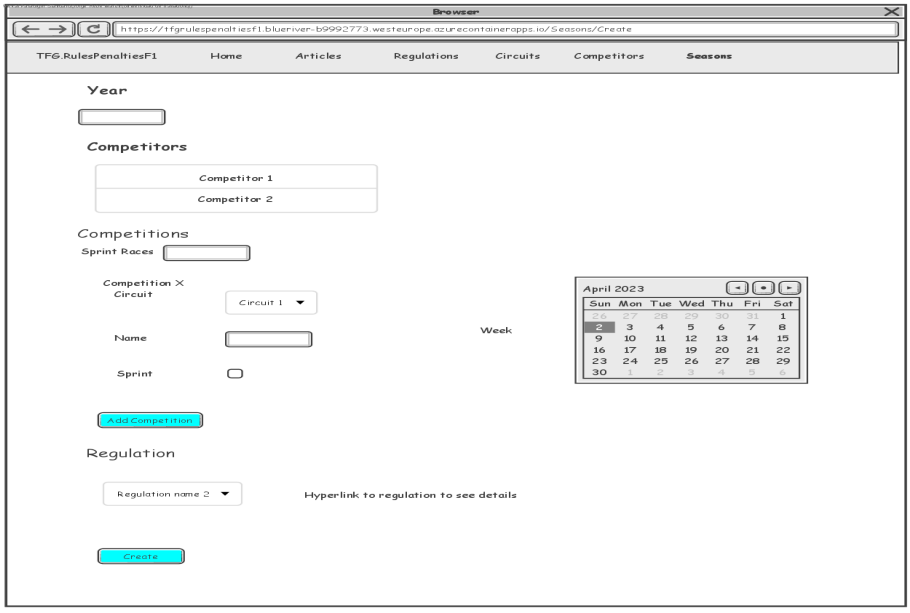


Figura 5.24: Interfaz de usuario HU11

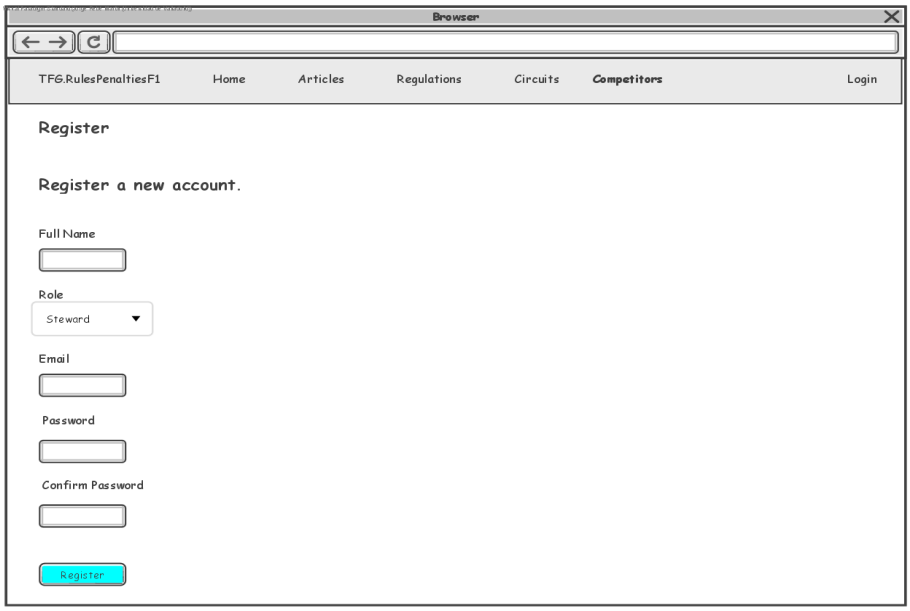


Figura 5.25: Interfaz de usuario HU12

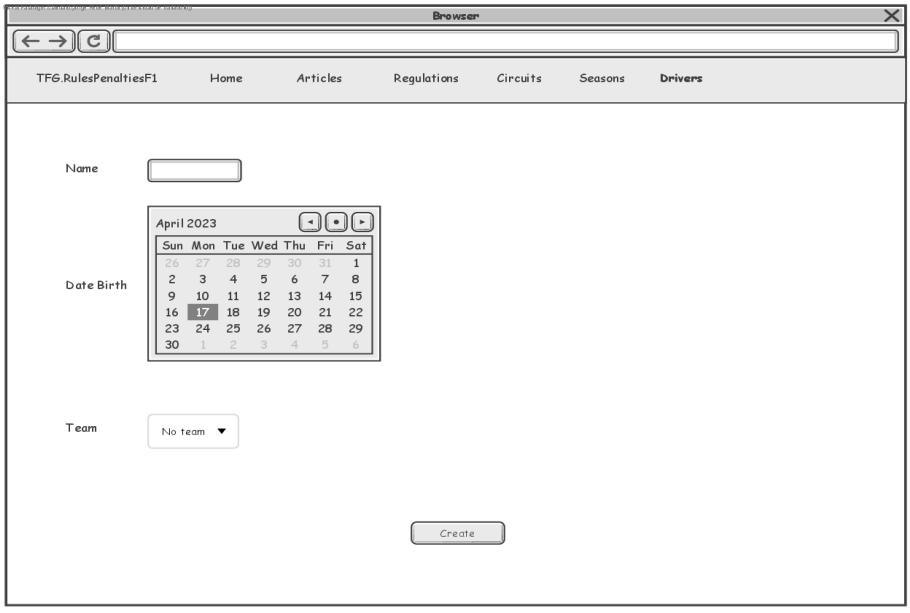


Figura 5.26: Interfaz de usuario HU15

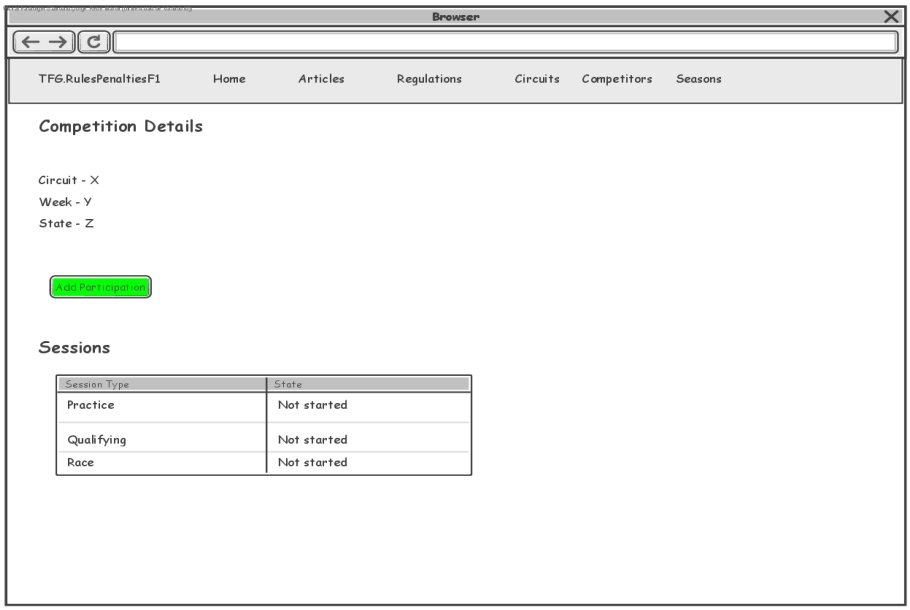


Figura 5.27: Interfaz de usuario HU19 - Parte 1

Figura 5.28: Interfaz de usuario HU19 - Parte 2

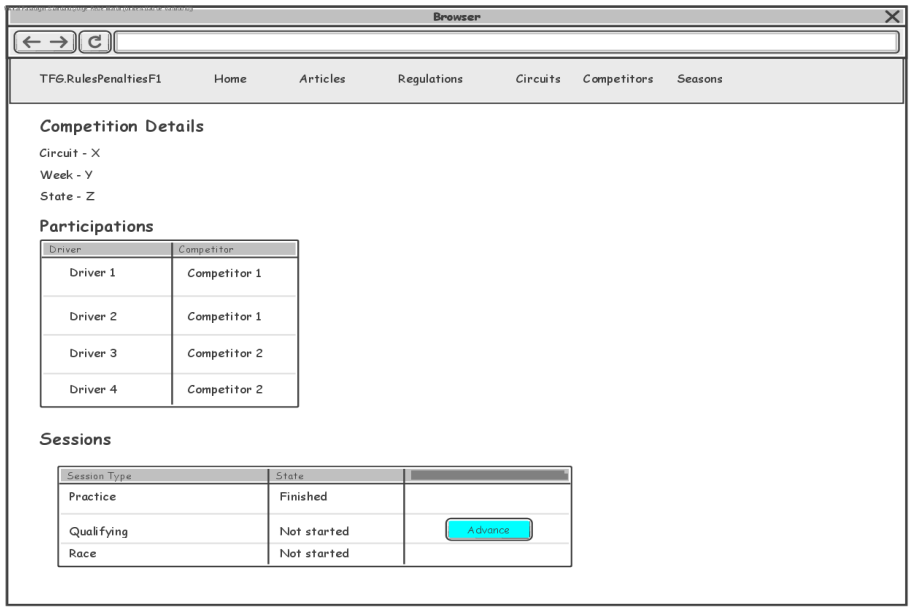


Figura 5.29: Interfaz de usuario HU20

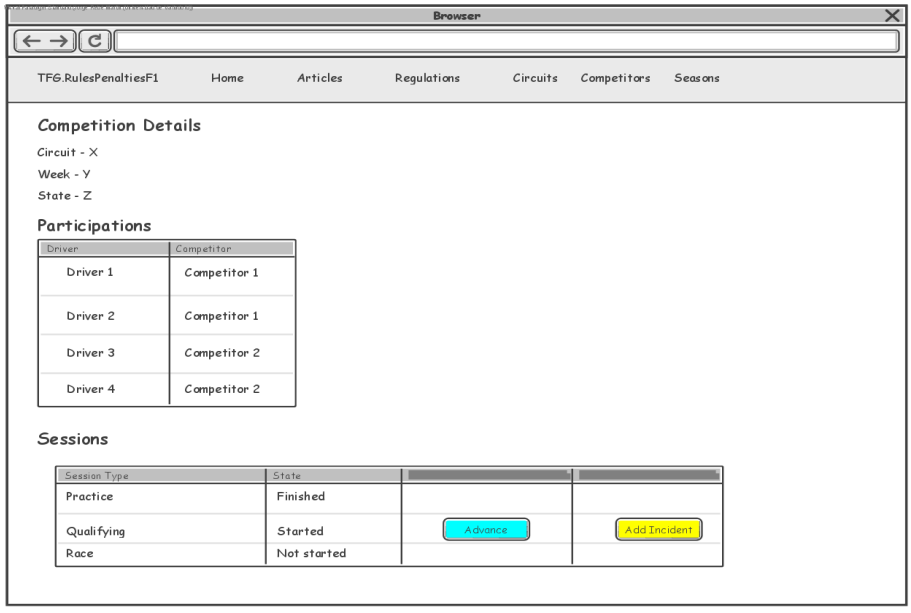


Figura 5.30: Interfaz de usuario HU21 - Parte 1

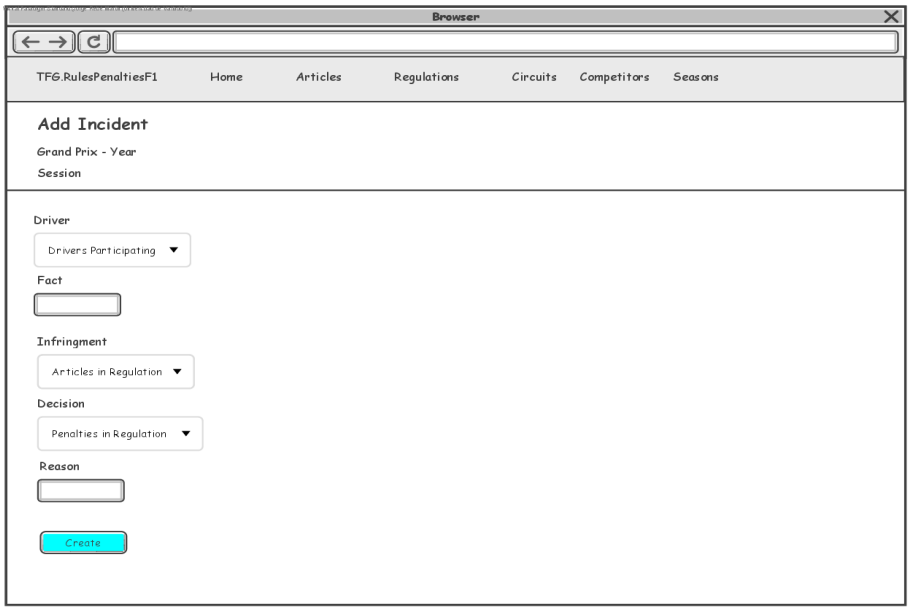


Figura 5.31: Interfaz de usuario HU21 - Parte 2

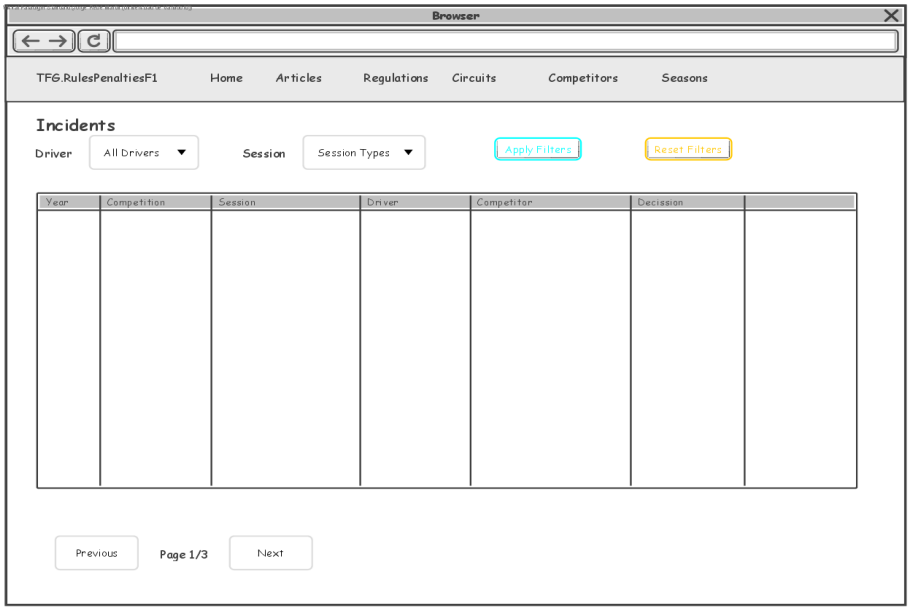


Figura 5.32: Interfaz de usuario HU25

Capítulo 6

Implementación y pruebas

6.1. Plantilla del proyecto: punto de partida

Como punto de partida para el desarrollo de la aplicación, se ha tomado una plantilla de una arquitectura *Clean Architecture* con ASP.NET [81], desarrollada por Steve Smith. En la siguiente sección se explicará la estructura del proyecto que se construyó partiendo de la plantilla.

6.2. Organización del proyecto en Visual Studio

Dentro de Visual Studio, una solución es un contenedor de uno o más proyectos relacionados. En este caso existirá una solución dentro de la cual habrá tres carpetas: 1) *'Solution Items'* para almacenar un fichero de configuración general del editor y las dependencias de todos los proyectos gestionadas de manera centralizada; 2) *'src'*, que contiene los proyectos en los que se encuentra la implementación de la funcionalidad de la aplicación , y 3) *'tests'*, que contiene los proyectos de los tests automatizados del proyecto (en este caso, tests unitarios y tests de aceptación).

Como se ha explicado en la Sección 5.2, la aplicación tiene una arquitectura compuesta por tres capas. Cada una de ellas se implementará por un proyecto diferente que se encontrará dentro de la carpeta `src/`. Así, la capa de interfaz de usuario se encontrará en el proyecto `TFG.RulesPenaltiesF1.Web`, la capa de infraestructura en el proyecto `TFG.RulesPenaltiesF1.Infrastructure` y la capa de lógica del negocio en el proyecto `TFG.RulesPenaltiesF1.Core`.

En la Figura 6.1 se muestra la estructura de la aplicación en Visual Studio.

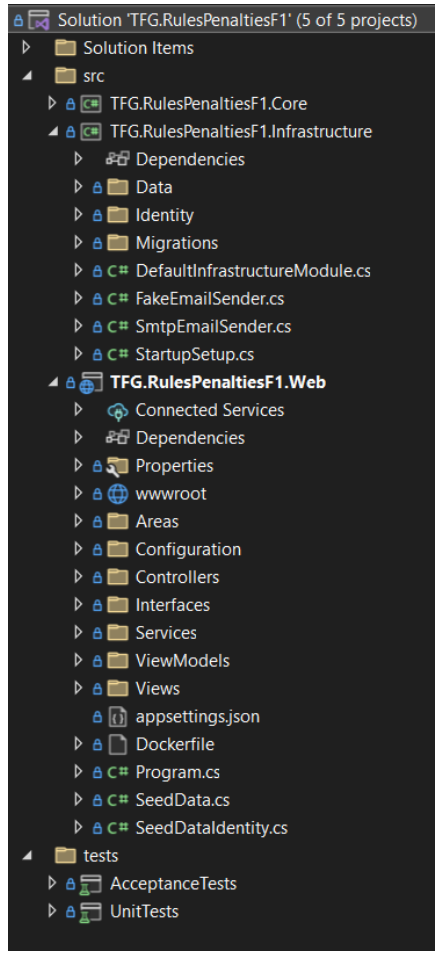


Figura 6.1: Estructura del proyecto en Visual Studio

6.3. Implementación

En esta sección se explicarán detalles relativos a la implementación de la gestión de usuarios y a la generación de interfaces de usuario.

6.3.1. Gestión de usuarios

Para toda la gestión de usuarios (inicio de sesión, registro, almacenamiento seguro de contraseñas, roles, etcétera) se ha utilizado una solución ofrecida por Microsoft, llamada ASP.NET Core Identity [82].

Identity es una API que da soporte a la funcionalidad de login de la interfaz de usuario,

y que administra usuarios, contraseñas, datos de perfil, roles, etcétera.

Soporta cuentas de usuario almacenadas directamente en Identity, o cuentas utilizando un proveedor de inicio de sesión externo (como Google). En este proyecto se va utilizar la primera opción.

6.3.2. Generación de interfaces de usuario

Para la generación y refinamiento de algunas interfaces de usuario se ha utilizado la inteligencia artificial generativa ChatGPT (Ver Subsección 4.1.11).

Aquí se va a explicar a modo de ejemplo cómo se generó la interfaz de usuario para la historia de usuario **HU01**: (Como comisario quiero añadir artículos antes del inicio de cada temporada para mantener el reglamento actualizado.).

Se le pasó a ChatGPT el código correspondiente al viewmodel que representa al artículo (`ArticleViewModel`), junto al siguiente *prompt* en inglés: 'I want you to create a Create view, where content of an article is added in a text-area. Furthermore, I want a button to add a subarticle and another to remove it Those buttons will generate text areas to add the content of those subarticles'.

ChatGPT devuelve una vista para ASP.NET. denominada '*Create*' debido a que la acción que se va a realizar dentro de ella es crear un registro. La vista resultante se parece de primeras bastante a la utilizada que se encuentra en el fichero `Create.cshtml` dentro de la carpeta `src/TFG.RulesPenaltiesF1.Web/Views/Articles/`. Una vez devuelta la vista, se leía, se probaba, y se realizaban pequeños refinamientos de forma manual o pidiendo a ChatGPT añadidos sobre lo generado anteriormente.

6.3.3. Problemas encontrados

Azure Key Vault

Idealmente, algunos elementos como los *connection strings*, que permiten la conexión con la base de datos, no deberían estar incrustadas en el código (*hard-coded*). En su lugar, Azure da una solución para almacenar este tipo de secretos, denominada Azure Key Vault 4.1.5. Se consiguió tener funcionando este sistema en local, pero para que funcionara con la aplicación desplegada, era necesario que estuviera registrada en el Azure Active Directory. Sin embargo, dado que no se ha permitido el acceso a esta parte de Azure, la aplicación no se ha podido registrar y por tanto el uso de Azure Key Vault no ha sido posible.

Por tanto, para el almacenamiento de los distintos *connection strings* se han utilizado distintos ficheros de configuración `appsettings.json` [83] (uno para cada entorno) donde se ha especificado ese *connection string*, y se obtiene de manera única al arrancar la aplicación dependiendo del entorno.

Pipelines

La idea con el proyecto era desarrollarlo aplicando despliegue continuo (*CD*), pero debido a la manifestación del RSK09 (Ver Tabla 2.9), se ha tenido que renunciar a ello. La razón de esto es que la aplicación necesitaría estar registrada en Azure Active Directory, y no se ha podido obtener el acceso a esta parte de Azure.

El despliegue de la aplicación ha tenido que hacerse de forma manual a través de las opciones ofrecidas por Visual Studio, en lugar de que en cada operación *push* en una determinada rama del repositorio Git se ejecutaran los tests de forma automática y se desplegara el sistema con lo actualizado.

6.4. Pruebas

Junto a la implementación de la funcionalidad de la aplicación, se han incluido pruebas para asegurar el correcto funcionamiento de la aplicación. Se han realizado tests automatizados unitarios, tests de aceptación automatizados y tests de aceptación no automatizados.

6.4.1. Tests unitarios

Los tests unitarios son un tipo de test utilizado para verificar algún componente de la aplicación en aislamiento, con independencia de cualquier otro componente del que pueda depender el componente que se está probando. El framework utilizado para realizar los tests unitarios es xUnit.net (Ver Subsección 4.1.6).

En este proyecto, los tests unitarios se han utilizado para probar las entidades del dominio con una funcionalidad más importante. En concreto, estas clases han sido las clases *Article*, *Circuit*, *Competition*, *Driver* y *Session*.

El proyecto en el que se encuentran los tests unitarios se llama `UnitTests`, que se encuentra dentro de la carpeta `/tests` en la solución. En total, se han realizado 75 tests unitarios de las clases mencionadas, con una cobertura del 100 % del código.

En la Figura 6.2 se puede ver el resultado de la ejecución de los tests unitarios, donde se ve el tiempo total de ejecución de los mismos (145 ms).

6.4.2. Tests de aceptación

Los tests de aceptación son aquellos que describen cómo se tiene que comportar el software, los que forman la *Definition of Done* de cada historia de usuario.

Se han realizado varios tests de aceptación automatizados pero, debido al tiempo invertido en la configuración de Specflow, en aprender sobre cómo escribir estos tests de aceptación y

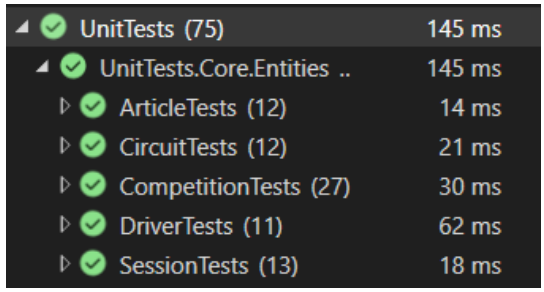


Figura 6.2: Resultado de la ejecución de los tests unitarios.

en solucionar problemas que han surgido (Ver Subsección 6.4.3), no se han podido cubrir los tests de todas las historias de usuario de forma automatizada.

Por tanto, se han realizado unos pocos tests de aceptación automatizados, y el resto han sido tests de aceptación manuales para completar las pruebas realizadas sobre el sistema y asegurar que su funcionamiento es correcto.

Tests de aceptación automatizados

Se han desarrollado tests de aceptación automatizados para tres historias de usuario. Para la HU01, consistente en crear artículos para su posterior introducción en un reglamento, HU03, creación de un reglamento compuesto por un nombre, artículos y penalizaciones, y la HU06, consistente en el inicio de sesión en el sistema por parte de los usuarios.

Se va a explicar a continuación la implementación del test de aceptación automatizado para la HU03. En la Figura 6.3 se muestra la especificación en lenguaje natural utilizando la sintaxis Gherkin para la especificación del test.

El tag *@stewardlogin* se utiliza para anotar aquellos tests en los que sea precondición que el comisario esté identificado en el sistema. Todos los tests con ese tag realizarán unas operaciones en el sistema antes para realizar ese inicio de sesión.

El escenario de test descrito en Gherkin después se transforma a código, que se va a explicar a continuación. Specflow genera una clase automáticamente a partir de la descripción con Gherkin, donde se escribe la lógica de los tests de aceptación.

A continuación se va a explicar esta clase con el código de automatización de pruebas. Primero, la definición de la clase, sus dependencias, y luego cada uno de los métodos de la clase, que representan cada uno un paso del escenario.

En primer lugar, se ve la declaración de la clase junto con sus dependencias necesarias, declaradas explícitamente en el constructor (ver Listing 6.1). Serán inyectadas automáticamente por Autofac.

El Page Object (ver Subsección 5.5.5) corresponde con la página de creación de reglamen-

```
Feature: HU03-Regulations

A short summary of the feature

@stewardlogin
Scenario: [Create regulation with one article and two penalties]
    Given [The steward is creating a regulation]
    When [The steward enters the name of the regulation]
    And [The steward selects the article part of the regulation]
    And [The steward selects two penalties for the regulation]
    And [The steward submits the regulation]
    Then [The regulation is created]
```

Figura 6.3: Especificación con Gherkin de los tests de aceptación de la HU03.

tos, y se utiliza para la interacción con la interfaz de usuario de creación de reglamento. El Driver (Ver Subsección 5.5.6) se utilizará en la verificación de la poscondición de la historia de usuario.

Listing 6.1: Declaración de la clase donde se encuentra el código automatizado para el test de aceptación de la HU03.

```
public class HU03_RegulationsStepDefinitions
{
    private readonly RegulationPageObjectModel _regulationPageObjectModel;
    private readonly RegulationPageDriver _regulationPageDriver;

    public HU03_RegulationsStepDefinitions(BrowserDriver browserDriver,
        RegulationPageDriver regulationPageDriver)
    {
        _regulationPageObjectModel = new
            RegulationPageObjectModel(browserDriver.Current);
        _regulationPageDriver = regulationPageDriver;
    }
```

El primer método (ver Listing 6.2) corresponde al primer paso definido anteriormente con Gherkin, y consiste en la precondición de la historia de usuario: el comisario ha de estar creando un reglamento.

Para la verificación de esto, se pide al Page Object que se asegure de que se está en la página correcta.

Listing 6.2: Código de automatización correspondiente al primer paso del test de la HU03.

```
    [Given(@"\[The steward is creating a regulation]")]
    public void GivenTheStewardIsCreatingARegulation()
    {
        _regulationPageObjectModel.EnsureRegulationPageIsOpenAndReset();
    }
```

El siguiente paso (ver Listing 6.3) es la entrada por parte del comisario del nombre del reglamento que se va a crear. Se le pide al Page Object la introducción de dicho nombre, que se pasa por parámetro.

Listing 6.3: Código de automatización correspondiente al segundo paso del test de la HU03.

```
[When(@"\[The steward enters the name of the regulation]")]
public void WhenTheStewardEntersTheNameOfTheRegulation()
{
   _regulationPageObjectModel.EnterNameRegulation("testing regulation
       1");
}
```

Tras esto, el comisario selecciona los artículos que quiere que formen parte del reglamento (ver Listing 6.4). En este caso es sólo uno, y correspondería con al primer artículo que se verá en el selector de la página de creación del reglamento.

Listing 6.4: Código de automatización correspondiente al tercer paso del test de la HU03.

```
[When(@"\[The steward selects the article part of the regulation]")]
public void WhenTheStewardSelectsTheArticlePartOfTheRegulation()
{
   _regulationPageObjectModel.SelectArticle(0);
}
```

El siguiente paso (ver Listing 6.5) es la elección por parte del comisario de las penalizaciones que se podrán aplicar cuando el reglamento esté vigente. En este caso se le pide al Page Object que seleccione dos.

Listing 6.5: Código de automatización correspondiente al cuarto paso del test de la HU03.

```
[When(@"\[The steward selects two penalties for the regulation]")]
public void WhenTheStewardSelectsTwoPenaltiesForTheRegulation()
{
   _regulationPageObjectModel.SelectPenalty(2);
   _regulationPageObjectModel.SelectPenalty(4);
}
```

Una vez se ha rellenado el formulario correspondiente a la creación del reglamento, el comisario pulsa el botón para subirlo y crear de esta manera el reglamento (ver Listing 6.6).

Listing 6.6: Código de automatización correspondiente al quinto paso del test de la HU03.

```
[When(@"\[The steward submits the regulation]")]
public void WhenTheStewardSubmitsTheRegulation()
{
   _regulationPageObjectModel.ClickSubmitRegulation();
}
```

Una vez creado el reglamento, se le pide al Page Driver que compruebe la existencia de un reglamento en la base de datos con un nombre que sea igual al que se ha creado ahora, y si

se ha podido realizar con éxito el test, la comprobación debería tener un resultado positivo. Esto correspondería con la clausula Then de la especificación Gherkin del test (ver Listing 6.7)

Listing 6.7: Código de automatización para la comprobación de la postcondicción del test de la HU03.

```
    [Then(@"\[The regulation is created]")]
    public void ThenTheRegulationIsCreated()
    {
        _regulationPageDriver.existsRegulationWithName("testing regulation
            1").Should().BeTrue();
    }
}
```

Tests de aceptación no automatizados

Desde la Tabla 6.1 a la Tabla 6.17 se muestran los casos de prueba de los tests de aceptación para cada historia de usuario. Antes de la ejecución manual de cada caso de prueba, el estado de la base de datos se resetea al especificado por las clase *SeedData* y *SeedDataIdentity* (Ver en `/src/TFG.RulesPenaltiesF1.Web/` en el repositorio de código en el Anexo B).

HU01	
Caso de prueba	**Salida esperada**
Intentar crear un artículo con un contenido vacío.	Error indicando que el campo es requerido.
Intentar crear un artículo con menos de 10 caracteres.	Error indicando longitud mínima del campo.
Intentar crear un artículo con subartículo cuyo contenido está vacío.	Error indicando que el campo es requerido.
Intentar crear un artículo con subartículo cuyo contenido tenga menos de 10 caracteres.	Error indicando longitud mínima del campo.
Creación de un artículo cuyo contenido tiene 10 caracteres y un subartículo cuyo contenido es de 10 caracteres	Artículo creado con éxito.

Tabla 6.1: Tests de aceptación de la historia de usuario HU01

HU03	
Caso de prueba	**Salida esperada**
Intentar crear un reglamento con un nombre vacío.	Error indicando que el campo es requerido.
Intentar crear un reglamento sin artículos.	Error indicando que es necesario seleccionar al menos 1.
Intentar crear un reglamento sin penalizaciones.	Error indicando que es necesario seleccionar al menos 1.
Intentar crear un reglamento con un nombre ya existente.	Error indicando que ya existe un reglamento con ese nombre.
Creación de un reglamento con un nombre único formado por un artículo y una penalización.	Reglamento creado con éxito.

Tabla 6.2: Tests de aceptación de la historia de usuario HU03

HU06	
Caso de prueba	**Salida esperada**
Inicio de sesión de un usuario con el rol 'Steward'.	El nuevo usuario puede realizar las acciones de un comisario.
Inicio de sesión de un usuario con el rol 'TeamPrincipal'.	El nuevo usuario puede realizar las acciones de un jefe de equipo.

Tabla 6.3: Tests de aceptación de la historia de usuario HU06

HU10	
Caso de prueba	**Salida esperada**
Intentar crear un competidor con un nombre vacío.	Error indicando que el campo es requerido.
Intentar crear un competidor con un nombre ya existente	Error indicando que ya existe un competidor con ese nombre.
Intentar crear un competidor con el nombre de la ubicación vacío.	Error indicando que el campo es requerido.
Intentar crear un competidor con el nombre de la unidad de potencia vacío.	Error indicando que el campo es requerido.
Creación de un competidor con nombre único y todos los campos requeridos rellenos.	Competidor creado con éxito.

Tabla 6.4: Tests de aceptación de la historia de usuario HU10

HU11	
Caso de prueba	**Salida esperada**
Intentar crear una temporada con menos de dos competidores.	Error indicando el número mínimo de competidores.
Intentar crear una temporada con alguna competición con el nombre vacío.	Error indicando que el campo es requerido.
Intentar crear una temporada con alguna competición que ocurra en una semana inválida.	Error indicando el rango de semanas válido.
Intentar crear una temporada con dos competiciones que tengan el mismo nombre.	Error indicando que dos competiciones no pueden tener el mismo nombre.
Intentar crear una temporada con dos competiciones que ocurran en la misma semana.	Error indicando que dos competiciones no pueden ocurrir en la misma semana.
Creación de una temporada con al menos dos competidores, al menos dos competiciones con nombres distintos y que ocurran en semanas distintas.	Temporada creada con éxito.

Tabla 6.5: Tests de aceptación de la historia de usuario HU11

HU12	
Caso de prueba	**Salida esperada**
Creación de un usuario con el rol 'Steward'.	El nuevo usuario puede realizar las acciones de un comisario.
Creación de un usuario con el rol 'TeamPrincipal'.	El nuevo usuario puede realizar las acciones de un jefe de equipo.

Tabla 6.6: Tests de aceptación de la historia de usuario HU12

HU15	
Caso de prueba	**Salida esperada**
Intentar crear un piloto con el nombre vacío.	Error indicando que el campo es requerido.
Intentar crear un piloto con el mismo nombre que otro piloto ya existente.	Error indicando que existe otro piloto con ese nombre.
Intentar crear un piloto con su fecha de nacimiento vacía.	Error indicando que el campo es requerido.
Intentar crear un piloto que no tenga aún 18 años.	Error indicando que la edad mínima de un piloto son 18 años.
Creación de un piloto con nombre único, mayor de 18 años	El piloto ha sido creado con éxito.

Tabla 6.7: Tests de aceptación de la historia de usuario HU15

HU16	
Caso de prueba	**Salida esperada**
No editar el competidor del piloto.	Los datos del piloto no han cambiado.
Modificar el competidor del piloto.	El competidor del piloto ha sido modificado correctamente.
Eliminar el competidor del piloto.	En los datos del piloto se indica que no está asignado a ningún competidor.
Asignar un competidor a un piloto sin él.	En los datos del piloto se indica el nuevo competidor al que pertenece.

Tabla 6.8: Tests de aceptación de la historia de usuario HU16

HU17	
Caso de prueba	**Salida esperada**
Acceder a la página de creación de temporadas cuando hay una en curso.	Error 404 - Página no encontrada.
Acceder a la página de creación de temporadas cuando todas las existentes han terminado.	La página de creación de temporadas aparece normalmente.

Tabla 6.9: Tests de aceptación de la historia de usuario HU17

HU18	
Caso de prueba	**Salida esperada**
Intentar iniciar una competición con todas las anteriores ya finalizadas	La competición pasa a estado 'Started'
Intentar iniciar una competición con una competición anterior no finalizada todavía.	Error 404 - Página no encontrada

Tabla 6.10: Tests de aceptación de la historia de usuario HU18

HU19	
Caso de prueba	**Salida esperada**
Añadir participaciones de dos pilotos del competidor del jefe del equipo a una competición comenzada pero con sesiones sin empezar.	Las participaciones se han añadido a la competición.
Intentar añadir participaciones de pilotos de un competidor que no participa en la temporada.	Error 404 - Página no encontrada.
Intentar añadir participaciones de dos pilotos no pertenecientes al competidor del jefe del equipo que está añadiendo las participaciones.	Error 404 - Página no encontrada.

Tabla 6.11: Tests de aceptación de la historia de usuario HU19

HU20	
Caso de prueba	**Salida esperada**
Intentar avanzar una sesión no existente.	Error 404 - Página no encontrada.
Intentar avanzar una sesión de una competición no comenzada.	Error 404 - Página no encontrada.
Intentar avanzar una sesión de una competición a la que no se han añadido todas las participaciones	Error 404 - Página no encontrada.
Intentar avanzar una sesión de una competición que tiene todas sus participaciones añadidas y no todas las sesiones anteriores han finalizado.	Error 404 - Página no encontrada.
Avanzar una sesión de una competición que tiene todas sus participaciones añadidas y todas las sesiones anteriores han finalizado	La sesión ha avanzado de estado correctamente

Tabla 6.12: Tests de aceptación de la historia de usuario HU20

HU21	
Caso de prueba	**Salida esperada**
Intentar añadir un incidente a una sesión finalizada	Error 404 - Página no encontrada.
Intentar añadir un incidente sin especificar lo ocurrido	Error indicando que el campo es requerido.
Intentar añadir un incidente sin especificar la razón de la decisión	Error indicando que el campo es requerido.
Añadir un incidente con una penalización que involucre puntos de licencia, indicándolos.	Se ha creado un incidente con los puntos de licencia especificados.
Añadir un incidente con una penalización que involucre una multa, indicando la cantidad.	Se ha creado un incidente con la multa especificada.
Añadir un incidente con una penalización que no involucre ni multa ni puntos de licencia.	Se ha creado un incidente con los datos especificados.

Tabla 6.13: Tests de aceptación de la historia de usuario HU21

HU22	
Caso de prueba	**Salida esperada**
Añadir un incidente a un piloto sin especificar puntos de licencia.	Los puntos de licencia del piloto involucrado no se han modificado.
Añadir un incidente a un piloto con 1 punto de licencia.	Los puntos de licencia del piloto involucrado se han incrementado en una unidad.

Tabla 6.14: Tests de aceptación de la historia de usuario HU22

HU23	
Caso de prueba	**Salida esperada**
Un piloto recibe 3 puntos de licencia en la semana 1 y 4 en la semana 4 de la temporada 2023. En la semana 1 de la temporada 2024 pasa a tener 4 en total, y en la semana 5 tendrá 0 (siempre y cuando no haya recibido más sanciones con puntos de licencia entre medias).	Al arrancar la competición de la semana 1 de la temporada 2024 se ve como se actualizan los puntos de los pilotos cuyos puntos han caducado (ídem para la semana 5).

Tabla 6.15: Tests de aceptación de la historia de usuario HU23

HU24	
Caso de prueba	**Salida esperada**
Acumular 12 puntos de sanción para un piloto en una sesión.	En la siguiente competición el piloto no aparecerá como elegible para participar cuando su jefe de equipo añada las participaciones.
Añadir un incidente que tenga como penalización una suspensión a un piloto.	En la siguiente competición el piloto no aparecerá como elegible para participar cuando su jefe de equipo añada las participaciones.

Tabla 6.16: Tests de aceptación de la historia de usuario HU24

HU25	
Caso de prueba	**Salida esperada**
No existen incidentes con los filtros especificados.	Se muestra un mensaje indicando la razón de que no haya incidentes en la página.
Existen incidentes con los filtros especificados.	Los incidentes que aparecen cumplen con los filtros solicitados y aparece la página en la que se encuentran, con opción a ir hacia atrás o hacia delante siempre que se esté dentro del rango posible.

Tabla 6.17: Tests de aceptación de la historia de usuario HU25

Los tests más relevantes, aquellos que han servido para detectar más problemas y que eran siempre los primeros en ejecutarse al desplegar la aplicación son los correspondientes a la HU20, HU21 y HU22. Los tests de estas historias de usuario representan la funcionalidad clave del sistema, y son las historias que unen todos los componentes de la aplicación y, por tanto, son los típicos en los que pueden ocurrir errores tras modificar algo en la aplicación.

6.4.3. Problemas encontrados

A continuación se describen algunos de los problemas encontrados durante el desarrollo de las pruebas. Todos ellos ocurrieron en el desarrollo de las pruebas de aceptación, y pudieron ser solucionados.

TestServer

Para aprender sobre BDD y a desarrollar los tests de aceptación correspondientes se ha utilizado SpecFlow [84], ya que cuenta con una amplía variedad de tutoriales, ejemplos y documentación para aprender a utilizarlo. Además, hace 2 años se realizaron unos tutoriales en los que precisamente realizaban pruebas de aceptación automatizadas utilizando SpecFlow y Selenium [85].

Utilizando los tutoriales como base y la web de Specflow, comencé a preparar la base para los tests. Primero se preparó el servidor de test encargado de alojar la aplicación. Aquí surgió el primer problema, precisamente porque los tutoriales y ejemplos de uso de SpecFlow son de hace 2 años. Hace 2 años, la versión de .NET más nueva utilizada era la 5, que para el lanzamiento de la aplicación usaba dos ficheros: 'Startup.cs' y 'Program.cs'. Por un lado, Program.cs es la clase donde se arranca la aplicación, que crea una instancia de IWebHost que aloja nuestra aplicación. Por otro, Startup.cs es la encargada de configurar los servicios de la aplicación [86].

Sin embargo, a partir de .NET 6, las aplicaciones ASP.NET solo disponen de un fichero: 'Program.cs', desde el que se arranca la aplicación y se configuran los servicios (es decir, Microsoft redujo todo a un único fichero).

El problema es que en este tutorial, se creaba el 'Test Server' de una manera que se utiliza la clase Startup, debido a que la versión de .NET utilizada era inferior a .NET 6, y es imposible de adaptarlo a .NET 7. Por tanto, busqué una manera alternativa para crear el 'Test Server', en este caso WebApplicationFactory [87].
Para configurar WebApplicationFactory, seguí un tutorial de tests de integración [88], adaptando la configuración que se realiza en ese tutorial a la configuración también hecha en los tutoriales de SpecFlow.

Inyección de dependencias

Una vez solventado el problema de lanzar el servidor para ejecutar los tests, surgió un problema al tratar de ejecutarlos: La forma de resolver las dependencias sugerida en el tutorial de SpecFlow no funcionaba correctamente. Como contenedor de inyección de dependencias en la aplicación se usa Autofac [46], mientras que el contenedor de dependencias utilizado por Specflow es BoDi. En un principio se intentó configurar las dependencias utilizando BoDi, tomando como guía el tutorial. Sin embargo, las dependencias seguían sin resolverse y se buscó una solución alternativa, en este caso, utilizar Autofac como contenedor de dependencias también en el Test Server.

Para ello se utilizó del plugin SpecFlow.Autofac [89], de SpecFlow, que permite la utilización de Autofac como inyector de dependencias en lugar de BoDi. Tras varios intentos, consultar varias veces stackoverflow y la documentación, finalmente conseguí que funcionara correctamente y que los tests se ejecutaran como estaba previsto, aunque estos dos problemas supusieron una pérdida bastante grande de horas.

Ejecución de tests en paralelo

Otro problema surgido con los tests de aceptación, es que por defecto se ejecutan en paralelo. Los tests de aceptación son tests de extremo a extremo (*end-to-end* o e2e), por lo que se depende de los datos de la base de datos. Esto significa que no pueden ejecutarse en paralelo, ya que cada test estaría modificando un recurso compartido (la base de datos) al mismo tiempo. Por tanto, se tomó la decisión de no ejecutarlos en paralelo.

Para evitar ejecutar los tests en paralelo [90] se añadió una propiedad al ensamblado del proyecto de tests de aceptación indicando que se deshabilite la paralelización de los tests. Es una propiedad de configuración de xUnit (que es el framework encargado de ejecutar el código de automatización de test, como se ha explicado anteriormente). Esta es la propiedad añadida en la clase *AssemblyInfo.cs* (ver `tests/AcceptanceTests/AssemblyInfo.cs en el repositorio de código en el Anexo B`):

```
[assembly:CollectionBehavior(DisableTestParallelization = true)]
```

Capítulo 7

Seguimiento del proyecto

Con este capítulo se mostrarán las tareas que se han ido realizando a lo largo de los sprints que se han ejecutado en el proyecto. Se explicará el contenido tratado en las *Scrum Weekly*, *Scrum Retrospective* y *Scrum Planning*, y cómo se han ido solucionando los problemas surgidos a lo largo de cada uno de ellos.

Después, se mostrará un resumen con todas esas tareas, donde se podrá ver el tiempo total empleado en el desarrollo del proyecto.

Por último, se realizará una comparación de costes y calendarización con lo planificado en el Capítulo 2, así como los riesgos que se han manifestado en el transcurso del proyecto.

7.1. Seguimiento de los sprints realizados

7.1.1. Sprint 0 [17/01/2023 - 14/02/2023]

Este sprint 0 va a ser completamente diferente a los demás, ya que se va a dedicar a la preparación de todo lo necesario para poder empezar a desarrollar las historias de usuario en el sprint 1. La duración de este primer sprint también será distinta, habiéndole dedicado un total de 4 semanas.

A continuación se listan las tareas realizadas durante el *Sprint Zero*.

- **Redacción de la memoria**: Preparación y redacción de parte de los capítulos iniciales de la memoria. En concreto, se trabajó en los capítulos 1, 2 y 3.
- **Elaboración del *Product Backlog* inicial**: A partir del análisis de varios reglamentos deportivos de la F1 se realizó una versión inicial del *Product Backlog*, que se irá refinando

a lo largo del desarrollo del proyecto. Se escribieron algunas épicas y se descompusieron en historias de usuario, describiendo el comportamiento que el sistema debe tener.

- **Modelo de dominio**: Se realizó una versión inicial del modelo de dominio de la aplicación, para poder ir viendo cómo se va a estructurar la aplicación. También pasará por varias mejoras en el desarrollo del proyecto.
- **Formación en ASP.NET Core**: Se realizaron algunos tutoriales [91] para familiarizarse con el framework y coger agilidad para cuando se empiecen a desarrollar las historias de usuario.
- **Exploración de soluciones para el despliegue de la aplicación**: El servicio en la nube para desplegar la aplicación fue Azure, pero dentro de los productos de Azure se evaluaron dos: Azure App Service y Azure Container App. Se eligió la última opción debido a que es totalmente gratuita para estudiantes.
- **Plan de riesgos**: Se realizó el plan de riesgos presentado en la Sección 2.5.
- **Calendarización inicial**: Se realizó el calendario inicial presentado en la Sección 2.6.
- **Plan de presupuestos**: Se realizó el plan de presupuestos presentado en la Sección 2.7.
- **Arquitectura de referencia**: Se eligió *Clean Architecture* como la arquitectura de referencia para la aplicación, como se ha explicado en la Sección 5.2.

El tiempo total empleado en el desarrollo de estas tareas ha sido de aproximadamente 40 horas.

7.1.2. Sprint 1 [14/02/2023 - 28/02/2023]

Para esté primer sprint se planificó inicialmente la realización de todas las historias de usuario correspondientes a la épica 1. Sin embargo, durante el desarrollo del mismo se vio que la estimación inicial era demasiado optimista, ya que surgieron problemas que no se tuvieron en cuenta inicialmente.

En la HU07, que es la primera que se realizó, se adaptó la plantilla de la arquitectura de referencia a las necesidades de este proyecto. Se tomaron decisiones generales sobre la utilización o no de algunos patrones de diseño cuyo uso es común en DDD, pero que se salen del alcance del proyecto. Por ejemplo, se descartó el patrón Especificación [92] para encapsular consultas.

En cuanto a la HU01, la revisión del análisis, la realización del diseño, el diseño de las pruebas y la implementación en si mismas sí que se podían haber adaptado a la estimación realizada. Sin embargo, hubo que dedicar varias horas a la formación con Specflow y Selenium, herramientas utilizadas para las pruebas de aceptación. También se utilizaron muchas horas en la puesta en marcha del Host para las pruebas de aceptación (en la máquina local), debido a que la documentación de Specflow no está actualizada para .NET 7.

Historia de usuario	Tareas	Estimación (puntos)	Tiempo empleado	Estado
HU01	Como comisario quiero añadir artículos antes del inicio de cada temporada para mantener el reglamento actualizado.			
	Análisis Diseño Pruebas Implementación	3	25 horas	Pruebas no terminadas
HU07	Como equipo de desarrollo quiero crear el proyecto utilizando desde el principio arquitectura de referencia en el desarrollo correctamente para que sea mantenible.			
	Preparación proyecto.	5	12 horas	Terminada
Resumen		8	37 horas	1/2 terminadas

Tabla 7.1: Tareas realizadas en el Sprint 1

En la Tabla 7.1 se puede ver un resumen de las tareas realizadas en el sprint 1.

En el sprint 2 se terminarán las pruebas de la historia de usuario HU01, y se trabajará en la HU02 (2 puntos), HU03 (3 puntos) y HU06 (5 puntos). También se avanzará en las historias de usuario dedicadas al mantenimiento de la arquitectura y documentación del proyecto (HU13 y HU14).

7.1.3. Sprint 2 [28/02/2023 - 14/03/2023]

En este sprint también se hizo una estimación optimista, pero mucho más realista que en el sprint anterior. Se han conseguido realizar casi todas las historias de usuario que se habían previsto, y de una manera mucho más rápida que en el anterior.

En la Tabla 7.2 se puede ver un resumen de las tareas realizadas en el sprint 2.

En el siguiente sprint se planifica terminar con la historia de usuario HU06, así como trabajar en la HU09 (2 puntos), HU10 (3 puntos) y HU12 (2 puntos). También se continuará trabajando en las dos historias de mantenimiento de la arquitectura y documentación del proyecto (HU13 y HU14).

Historia de usuario	Tareas	Estimación (puntos)	Tiempo empleado	Estado
HU01	Como comisario quiero añadir artículos antes del inicio de cada temporada para mantener el reglamento actualizado.			
	Implementación	3	3 horas	Terminada
HU02	Como comisario quiero poder asignar penalizaciones que no puedan eliminar puntos de licencia a un reglamento para que su incumplimiento tenga consecuencias.			
	Análisis Diseño Implementación	2	8 horas	Terminada
HU03	Como comisario quiero poder crear un reglamento antes del inicio de una temporada, asignándole los artículos y las sanciones que lo formarán.			
	Análisis Diseño Implementación	3	16 horas	Terminada
HU06	Como comisario quiero poder iniciar sesión en la aplicación.			
	Análisis Diseño Implementación	5	1 hora	Análisis comenzado.
HU13	Como equipo de desarrollo quiero revisar que la arquitectura de referencia se sigue usando correctamente en el desarrollo.			
	-	15	4 horas	En progreso
HU14	Como equipo de desarrollo quiero documentar el proyecto correctamente para que el proyecto sea mantenible.			
	-	30	3 horas	En progreso
Resumen		13	35 horas	3/4 completadas

Tabla 7.2: Tareas realizadas en el Sprint 2

7.1.4. Sprint 3 [14/03/2023 - 28/03/2023]

En total, se ha completado un trabajo casi correspondiente a 13 puntos completos de historia de usuario, lo cual indica que se ha conseguido mejorar en este sprint también la precisión de la planificación.

En este sprint se ha podido comprobar que existe un mayor dominio de la tecnología utilizada, al haber aumentado la *velocity*.

En los problemas surgidos en este sprint se debe destacar que no se ha podido hacer el despliegue debido a unos conflictos de dependencias con paquetes externos, detectados solo a la hora de hacer el despliegue al final del sprint. Esos problemas no estaba presentes en el despliegue de pruebas local y se manifestaron al realizar el despliegue de producción.

Para solucionarlo, se añadió una línea al Dockerfile que copiaba un fichero con las dependencias de paquetes y sus versiones.

En la Tabla 7.3 se puede ver un resumen de las tareas realizadas en el sprint 3.

Historia de usuario	Tareas	Estimación (puntos)	Tiempo empleado	Estado
HU06	Como comisario quiero poder iniciar sesión en la aplicación.			
	Implementación	5	12 horas	Terminada
HU09	Como comisario quiero poder añadir circuitos en los que se podrán disputar competiciones.			
	Análisis Diseño Implementación	2	8 horas	Terminada
HU10	Como comisario quiero poder inscribir a nuevos equipos que vayan a participar en la competición.			
	Análisis Diseño Implementación	3	8 horas	Implementación por terminar
HU12	Como comisario quiero poder registrar nuevos comisarios y jefes de equipo en el sistema.			
	Análisis Diseño Implementación	2	2 horas	Terminada
HU13	Como equipo de desarrollo quiero revisar que la arquitectura de referencia se sigue usando correctamente en el desarrollo.			
	-	15	3 horas	En progreso
HU14	Como equipo de desarrollo quiero documentar el proyecto correctamente para que el proyecto sea mantenible.			
	-	30	3 horas	En progreso
Resumen		12	36 horas	2/3 completadas

Tabla 7.3: Tareas realizadas en el Sprint 3

Se detectaron durante la sprint review varios errores en relación a la historia de usuario HU09, que serán corregidos posteriormente. Para empezar, los nombres de algunos campos eran poco intuitivos y estaban escritos tal cual el nombre de la propiedad del ViewModel, y alguna parte del formulario de creación de circuitos era poco intuitiva. Además, las validaciones no eran muy robustas: era posible insertar un circuito con 0km, vuelta rápida de duración 0, un año de lap record previo al del primer GP, y creación de circuitos con el mismo nombre.

Para el siguiente sprint se acabará la historia de usuario HU10, y se realizarán las historias HU11 (5 puntos), HU15 (3 puntos) y HU16 (2 puntos). Además, también se planea avanzar de forma significativa con la HU13 y la HU14, correspondientes al mantenimiento de la arquitectura y la documentación del proyecto.

7.1.5. Sprint 4 [28/03/2023 - 11/04/2023]

La planificación para este sprint fue excesivamente optimista, ya que acabar la historia de usuario 10, pendiente de acabar del anterior, y la HU11 que se ha empezado y terminado en este, han llevado más de la mitad del tiempo del sprint. Se han presentado algunas situaciones que han requerido de investigación extra, de ahí el retraso.

Por otro lado, se ha avanzado bastante en la redacción de la memoria y se ha mantenido usando la arquitectura de manera correcta.

Durante la sprint review se iba a realizar una demo de las HU10 y HU11, como de costumbre, pero al cargar cualquier página relacionada con los competidores o al crear una temporada se recibía un error 500. Se debía a que hay dos paquetes distintos que dan soporte a algunas anotaciones utilizadas para la verificación del input de los usuarios, y el paquete importado en el ViewModel de competidor era el equivocado, lo cual provocaba en producción comportamientos inesperados.

El problema fue difícil de diagnosticar, ya que en local funcionaba correctamente, incluso ejecutado en Docker en local no había ningún problema, y los logs de Azure no daban demasiada información. Esto no supuso un retraso para el siguiente sprint.

En la Tabla 7.4 se puede ver un resumen de las tareas realizadas en el sprint 4.

Para el siguiente sprint se planifica realizar las historias de usuario H15, HU16 y avanzar con las historias HU13 y HU14. Es una planificación mas conservadora que en el sprint anterior, con el objetivo de avanzar bastante en las historias de mantenimiento de la arquitectura y documentación del proyecto.

Historia de usuario	Tareas	Estimación (puntos)	Tiempo empleado	Estado
HU10	Como comisario quiero poder inscribir a nuevos equipos que vayan a participar en la competición.			
	Implementación	3	5 horas	Terminada
HU11	Como comisario, quiero poder crear una temporada en la que estén definidos desde el comienzo el reglamento, las competiciones y los equipos que van a participar, y que no sean editables a partir de ese momento.			
	Análisis Diseño Implementación	5	19 horas	Terminada
HU15	Como comisario quiero inscribir a un piloto en la competición y asignarle un equipo.			
	-	3	-	No comenzada
HU16	Como comisario quiero poder eliminar la asignación de un equipo a un piloto, y también poder reasignarle a otro equipo.			
	-	2	-	No comenzada
HU13	Como equipo de desarrollo quiero revisar que la arquitectura de referencia se sigue usando correctamente en el desarrollo.			
	-	15	4 horas	En progreso
HU14	Como equipo de desarrollo quiero documentar el proyecto correctamente para que el proyecto sea mantenible.			
	-	30	10 horas	En progreso
Resumen		13	38 horas	2/4 completadas

Tabla 7.4: Tareas realizadas en el Sprint 4

7.1.6. Sprint 5 [11/04/2023 - 25/04/2023]

Durante este sprint no fue posible realizar las típicas 40 horas que se venían realizando en los anteriores sprints, llegando a 20 en este. La razón de esto es que se ha materializado el riesgo RSK06. Debido a la carga de trabajo de entregas académicas y las prácticas en empresa que se acumularon durante este sprint, ha sido imposible completar las historias de usuario planeadas.

Sin embargo, se consiguieron completar las 2 historias planeadas (HU15 y HU16) y avanzar con las dos historias de documentación del proyecto y revisión de la arquitectura (HU13 y HU14).

En la Tabla 7.5 se puede ver un resumen de las tareas realizadas en el sprint 5.

Para el siguiente sprint, se planea realizar las siguientes historias de usuario: HU17 (2 puntos), HU18 (4 puntos), HU19 (3 puntos) y HU20 (3 puntos), además de continuar con las historias de revisión de la arquitectura y documentación del proyecto (HU13 y HU14).

Historia de usuario	Tareas	Estimación (puntos)	Tiempo empleado	Estado
HU15	Como comisario quiero inscribir a un piloto en la competición y asignarle un equipo.			
	Análisis Diseño Implementación	3	6 horas	Terminada
HU16	Como comisario quiero poder eliminar la asignación de un equipo a un piloto, y también poder reasignarle a otro equipo.			
	Análisis Diseño Implementación	2	4 horas	Terminada
HU13	Como equipo de desarrollo quiero revisar que la arquitectura de referencia se sigue usando correctamente en el desarrollo.			
	-	15	4 horas	En progreso
HU14	Como equipo de desarrollo quiero documentar el proyecto correctamente para que el proyecto sea mantenible.			
	-	30	10 horas	En progreso
Resumen		5	24 horas	2/2 completadas

Tabla 7.5: Tareas realizadas en el Sprint 5

7.1.7. Sprint 6 [25/04/2023 - 09/05/2023]

Durante el sexto sprint no hubo ningún problema reseñable, el trabajo se completó sin ningún imprevisto. Se completaron a tiempo las 4 historias de usuario planeadas para el sprint, y también se utilizó el tiempo esperado en las historias correspondientes a revisión de la arquitectura y documentación del proyecto.

En la Tabla 7.6 se puede ver un resumen de las tareas realizadas en el sprint 6.

Para el siguiente sprint se planifica continuar con el trabajo en la documentación del proyecto, intentando hacer uso aproximadamente de la mitad del trabajo del sprint. También se tratarán de realizar las historias de usuario HU21 (3 puntos) y HU22 (3 puntos), y completar alguna pendiente de sprints anteriores si hubiera tiempo suficiente para ello.

Hasta ahora, las historias de usuario no completadas, en orden de prioridad descendente, son las siguientes:

- HU08 Como comisario quiero asignar penalizaciones que eliminen puntos de licencia a un reglamento para que su incumplimiento tenga consecuencias.
- HU12 Como comisario quiero que una vez identificado pueda realizar las acciones que tengo permitidas.
- HU04 Como comisario quiero poder editar sólo los artículos que no forman parte de un reglamento asignado a una temporada.

Historia de usuario	Tareas	Estimación (puntos)	Tiempo empleado	Estado
HU17	Como comisario quiero que no se puedan crear temporadas si existe alguna que no se ha terminado.			
	Análisis Diseño Implementación	2	5 horas	Terminada
HU18	Como comisario quiero poder arrancar la siguiente competición de una temporada, siempre que la anterior se haya finalizado ya.			
	Análisis Diseño Implementación	5	10 horas	Terminada
HU19	Como jefe de equipo quiero poder añadir la participación de dos pilotos que tenga asignados antes de que comience la primera sesión de una competición.			
	Análisis Diseño Implementación	3	9 horas	Terminada
HU20	Como comisario quiero poder dar por finalizadas las sesiones de una competición e iniciar las siguientes.			
	Análisis Diseño Implementación	3	5 horas	Terminada
HU13	Como equipo de desarrollo quiero revisar que la arquitectura de referencia se sigue usando correctamente en el desarrollo.			
	-	15	4 horas	En progreso
HU14	Como equipo de desarrollo quiero documentar el proyecto correctamente para que el proyecto sea mantenible.			
	-	30	7 horas	En progreso
Resumen		13	40 horas	4/4 completadas

Tabla 7.6: Tareas realizadas en el Sprint 6

- HU05 Como comisario quiero poder eliminar sólo los artículos que no forman parte de un reglamento asignado a una temporada.

7.1.8. Sprint 7 [10/05/2023 - 23/05/2023]

Durante este sprint no se pudieron utilizar tantas horas como se había planeado, debido a varios eventos académicos que limitaron el tiempo. Sin embargo, se consiguieron realizar casi todas las historias de usuario, y se corrigió un bug descubierto durante el sprint.

Se completó las historia de usuario pendiente HU08 y se corrigió el bug BUG01. En cuanto a la HU21, se consiguió dejar prácticamente terminada, aunque habrá que completarla en el siguiente sprint.

En cuanto a la documentación del proyecto, se avanzó durante la primera semana del sprint, dejando la segunda para los avances en funcionalidad ya comentados.

En la Tabla 7.7 se puede ver un resumen de las tareas realizadas en el sprint 7.

Historia de usuario	Tareas	Estimación (puntos)	Tiempo empleado	Estado
HU08	Como comisario quiero asignar penalizaciones que eliminen puntos de licencia a un reglamento para que su incumplimiento tenga consecuencias.			
	Análisis Diseño Implementación	2	2 horas	Terminada
BUG01	Desasignar de un equipo a un piloto			
	Revisión y correción del bug	1	1 hora	Terminada
HU21	Como comisario quiero poder registrar un incidente, una investigación sobre el incumplimiento de un artículo por parte de un piloto en una sesión que está en curso y asignar una sanción.			
	Análisis Diseño Implementación	5	10 horas	Implementación por terminar
HU13	Como equipo de desarrollo quiero revisar que la arquitectura de referencia se sigue usando correctamente en el desarrollo.			
	-	15	2 horas	En progreso
HU14	Como equipo de desarrollo quiero documentar el proyecto correctamente para que el proyecto sea mantenible.			
	-	30	15 horas	En progreso
Resumen		8	30 horas	3/4 completadas

Tabla 7.7: Tareas realizadas en el Sprint 7

Para el siguiente sprint se completará la HU21, y se realizarán HU22, HU23 y H24. También se tendrá lista para revisar por la tutora una versión inicial completa de la documentación del proyecto.

7.1.9. Sprint 8 [24/05/2023 - 06/06/2023]

A lo largo de este sprint se han podido realizar todas las historias de usuario previstas, y además se ha comenzado con la HU25 que no se había planificado en el *Sprint Planning*.

En la Tabla 7.8 se puede ver un resumen de las tareas realizadas en el sprint 8.

Historia de usuario	Tareas	Estimación (puntos)	Tiempo empleado	Estado
HU21	Como comisario quiero poder registrar un incidente, una investigación sobre el incumplimiento de un artículo por parte de un piloto en una sesión que está en curso y asignar una sanción.			
	Implementación	5	2 horas	Terminada
HU22	Como comisario quiero que al añadir un incidente se actualicen los puntos de licencia del piloto involucrado.			
	Análisis Diseño Implementación	3	7 horas	Terminada
HU23	Como comisario quiero que al empezar una competición se actualicen los puntos de licencia de los pilotos.			
	Análisis Diseño Implementación	3	6 horas	Terminada
HU24	Como comisario quiero que un piloto con 12 puntos de licencia o con una suspensión del Gran Premio anterior no pueda participar en la siguiente competición.			
	Análisis Diseño Implementación	2	3 horas	Terminada
HU25	Como aficionado quiero poder ver el histórico de los incidentes de la categoría y poder filtrarlos por piloto y por sesión.			
	Análisis Diseño Implementación	3	5 horas	Implementación por terminar
HU13	Como equipo de desarrollo quiero revisar que la arquitectura de referencia se sigue usando correctamente en el desarrollo.			
	-	15	2 horas	En progreso
HU14	Como equipo de desarrollo quiero documentar el proyecto correctamente para que el proyecto sea mantenible.			
	-	30	15 horas	En progreso
Resumen		16	40 horas	4/5 completadas

Tabla 7.8: Tareas realizadas en el Sprint 8

Para el siguiente sprint, que será el sprint extra de la planificación, se terminará con la implementación de la HU25, se realizarán mejoras en los tests unitarios y de aceptación (HU26) y se realizarán mejoras en la interfaz de usuario (HU27). También se finalizará con

la redacción de la memoria del TFG.

7.1.10. Sprint 9 [Extra] [06/06/2023 - 20/06/2023]

En este sprint extra se han completado sin contratiempos las historias de usuario planificadas, y se han terminado las historias de usuario HU13 y HU14, correspondientes al mantenimiento de la arquitectura y documentación del proyecto, en las que se ha trabajado desde el sprint 2 hasta este momento.

También se ha solucionado el BUG02, detectado durante el sprint. Este bug consistía en un error a la hora de recalcular los puntos de sanción de los pilotos al comienzo de una competición.

En la Tabla 7.9 se puede ver un resumen de las tareas realizadas en el sprint 9.

Historia de usuario	Tareas	Estimación (puntos)	Tiempo empleado	Estado
HU25	Como aficionado quiero poder ver el histórico de los incidentes de la categoría y poder filtrarlos por piloto y por sesión.			
	Análisis Diseño Implementación	3	4 horas	Terminada
HU26	Como equipo de desarrollo quiero realizar mejoras en los tests unitarios y de aceptación para que la aplicación sea más mantenible.			
	Mejoras completadas.	3	10 horas	Terminada
HU27	Como equipo de desarrollo quiero realizar mejoras en la interfaz de usuario para que la aplicación sea más fácil de utilizar para los usuarios.			
	Mejoras completadas.	3	8 horas	Terminada
BUG02	Arreglo de actualización de puntos de licencia después de haber llegado a los 12 y acumular más, sin llegar a 12			
	Revisión y corrección del bug	2	2 horas	Terminada
HU13	Como equipo de desarrollo quiero revisar que la arquitectura de referencia se sigue usando correctamente en el desarrollo.			
	-	15	2 horas	Terminada
HU14	Como equipo de desarrollo quiero documentar el proyecto correctamente para que el proyecto sea mantenible.			
	-	30	33 horas	Terminada
Resumen		11	59 horas	6/6 completadas

Tabla 7.9: Tareas realizadas en el Sprint 9

7.2. Resumen de la ejecución del proyecto

En esta sección se va a realizar un análisis sobre lo ejecutado en el proyecto, resumen del tiempo total empleado para su completación, riesgos que se han manifestado y los costes (simulados y reales).

7.2.1. Tareas Realizadas

En la Tabla 7.10 se muestra un resumen de las tareas realizadas durante el proyecto, sprints en los que se han realizado, estimación y tiempo empleado por cada una de ellas.

7.2.2. Riesgos que se han manifestado

En esta subsección se van a listar los riesgos que se han manifestado junto con el plan de contingencia adoptado, siguiendo el plan de riesgos desarrollado en la Sección 2.5.

- **RSK02**: El desconocimiento de la tecnología de desarrollo ha provocado varios retrasos en el desarrollo de las historias de usuario, tanto en la parte de implementación de la funcionalidad como en la parte de desarrollo de los tests. Esto se ha notado sobre todo en los primeros sprints, ya que la familiarización con las tecnologías utilizadas ha ido creciendo y con ello la facilidad para desarrollar las historias de usuario de manera rápida y correcta.
- **RSK03**: La subestimación en el esfuerzo requerido para completar algunas historias de usuario ha provocado que haya habido algunos retrasos en el proyecto, aunque con el paso de los sprints las estimaciones han sido cada vez más precisas, reduciendo la aparición de retrasos.
- **RSK06**: Durante el Sprint 5 hubo una serie de eventos académicos que significaron un menor número de horas dedicadas al desarrollo del proyecto, lo cual significó tener que aumentar el esfuerzo realizado en los siguientes sprints y, al final, tener que hacer uso del sprint de refuerzo.
- **RSK09**: La imposibilidad de utilizar el despliegue automatizado proporcionado por Azure DevOps ha provocado que se tenga que hacer de forma manual a través de Visual Studio. En su lugar, habría sido mejor que se realizara el despliegue de forma automática tras pasar los tests al realizar un push o un merge sobre la rama develop (al final de los sprints).

7.2.3. Costes finales

Teniendo en cuenta que hay una diferencia de 79 horas en el tiempo invertido en el desarrollo del proyecto y algunos costes imprevistos en los productos de Azure, se ha recalculado

Tarea	Sprints	Estimación (puntos)	Tiempo empleado
Tareas iniciales	0	-	40 horas
HU01	1 y 2	3	28 horas
HU02	2	2	8 horas
HU03	2	3	16 horas
HU04	X	X	X
HU05	X	X	X
HU06	2 y 3	5	13 horas
HU07	1	5	12 horas
HU08	7	2	2 horas
HU09	3	2	8 horas
HU10	3 y 4	3	13 horas
HU11	4	5	19 horas
HU12	3	2	2 horas
HU13	Todos	15	25 horas
HU14	Todos	30	96 horas
HU15	5	3	6 horas
HU16	5	2	4 horas
HU17	6	2	5 horas
HU18	6	5	10 horas
HU19	6	3	9 horas
HU20	6	3	5 horas
HU21	7 y 8	5	12 horas
HU22	8	3	7 horas
HU23	8	3	6 horas
HU24	8	2	3 horas
HU25	8 y 9	3	9 horas
HU26	9	3	10 horas
HU27	9	3	8 horas
BUG01	7	1	1 hora
BUG02	9	2	2 horas
TOTAL		120	379

Tabla 7.10: Resumen de las tareas realizadas.

el coste del proyecto total, tanto de forma simulada como real, y se compararán con los presupuestos calculados en la Sección 2.7.

Coste simulado final

Los elementos son los mismos que los del presupuesto simulado. El nuevo coste total debido al aumento de número de horas trabajadas en el proyecto y a otros costes de Azure se ve en la Tabla 7.11.

Se puede ver que con la normalización del 25 % aplicada en el presupuesto simulado

Concepto	Precio por unidad	Cantidad	Total
Trabajador y espacio de trabajo			
Sueldo Desarrollador Junior ASP.NET	13.08€/hora	379 horas	4957.32€
Seguridad Social del empleado	3.924€/hora	379 horas	1487.20€
Equipo del empleado (Dell Latitude 5430)	26.19€/mes	6 meses	157.15€
Espacio de trabajo coworking	60.00€/mes	6 meses	360€
Licencias			
Visual Paradigm Standard	52.10€/3 meses	6 meses	104.20€
Visual Studio Professional	41€/mes	6 meses	246€
Productos Azure			
Azure SQL Database	13.33€/mes	6 meses	79.98€
Container Registry	4.68€/mes	6 meses	28.08€
Azure Container App	0.00€/mes	6 meses	0€
Costes Azure	-	-	28.64€
TOTAL			**7448.57€**

Tabla 7.11: Coste simulado final

(7595.68€) se habrían podido hacer frente a los costes simulados del proyecto (7448.57€).

Coste real final

Los elementos son los mismos que los del presupuesto real. Añadiendo el sobrecoste de los productos de Azure y el incremento en horas, el coste real final es el que se ve en la Tabla 7.12.

Concepto	Precio por unidad	Cantidad	Total
Container Registry	0.00455€/hora	379 horas	1.72€
Equipo de trabajo	33.33€/mes	6 meses	199.98€
Costes de Azure	-	-	28.64€
TOTAL			**230.34€**

Tabla 7.12: Coste real final

El total estimado en el presupuesto real era de 201.35€, pero debido a los incrementos comentados el coste final ha sido de 230.34€.

Capítulo 8

Conclusiones y líneas futuras

8.1. Conclusiones

Tras la finalización del proyecto, se puede concluir que ha sido exitoso a pesar de las numerosas dificultades que han ocurrido durante el desarrollo del mismo. Todas las horas de trabajo invertidas han resultado en un producto mínimo viable que podría tener un valor importante de ser utilizado por usuarios reales.

Se ha hecho uso de un gran porcentaje de los conocimientos adquiridos durante el Grado en Ingeniería Informática, especialmente en aquellos pertenecientes a la mención en Ingeniería de Software.

En cuanto a los objetivos de desarrollo establecidos en la Subsección 1.3.1:

- Se ha conseguido gestionar los reglamentos de la F1 de una forma sencilla y efectiva, seleccionando los artículos y penalizaciones que los forman.
- Se ha conseguido una forma mediante la cual se pueden traspasar pilotos entre equipos de una manera rápida, efectiva y útil para poder realizar modificaciones y traspasos entre competiciones.
- Se ha conseguido poder gestionar el transcurso de temporadas, creando competiciones, su tipo, seleccionando el reglamento que se tendrá en cuenta.
- Se ha conseguido que cada equipo pueda seleccionar los pilotos que van a participar en una competición.
- Se ha conseguido que exista un sistema de estados entre las distintas sesiones de una competición, pudiendo añadir incidentes a los pilotos mientras se está celebrando una sesión.

En cuanto a los objetivos personales establecidos en la Subsección 1.3.2:

- Se ha conseguido aprender ASP.NET. Aprender a utilizar ASP.NET, adoptando las mejores prácticas posibles en el desarrollo de una aplicación web en la que se usa esa tecnología.
- Se ha conseguido mejorar notabemente la soltura en el desarrollo de software en C#, conseguido conocimiento de características avanzadas y de las novedades de las últimas versiones del lenguaje de programación.
- Se ha elegido una arquitectura y un enfoque de desarrollo de software adecuados al tiempo disponible, evitando adoptar soluciones excesivamente complejas que alargarían el desarrollo del proyecto varios meses.
- Se han realizado varios tests unitarios para lograr cobertura total de las clases más importantes del dominio y se ha aprendido como desarrollar tests automatizados de aceptación de manera muy satisfactoria.
- Evaluando todas las opciones disponibles, se eligió Azure como la plataforma más adecuada para el despliegue de la aplicación, teniendo en cuenta su gratuidad para estudiantes y la integración con Visual Studio.

En cuanto a los tests automatizados de aceptación contra los tests manuales, se puede concluir que a pesar de que la configuración es costosa y la curva de aprendizaje para los tests automatizados es muy grande, a largo plazo, el esfuerzo de desarrollar tests de aceptación automatizados es mucho menor que hacerlos manuales. Primero, porque con cada cambio realizado en la aplicación es necesario volver a ejecutar los tests (pruebas de regresión) para asegurar que ninguna parte del sistema se ha visto afectada por el nuevo desarrollo. Los tests automatizados tardan segundos (minutos como mucho) en ejecutarse, mientras que una batería larga de pruebas manuales podría tomar horas.

En cuanto al modelo de proceso elegido, se puede concluir que ha sido adecuado. Al no tener totalmente clara la funcionalidad del sistema desde el principio ni todos los requisitos del mismo, el uso de una metodología ágil ha hecho posible que realizar tantos cambios sea factible.

El análisis de riesgos realizado en la Sección 2.5 se puede concluir que ha sido suficiente y satisfactorio, ya que no se han manifestado más problemas en el desarrollo del proyecto que los ahí especificados.

8.2. Líneas de trabajo futuras

En cuanto a las distintas líneas de trabajo que podría seguir la aplicación destacan las siguientes:

- **Migración a .NET 8**: Migrar la aplicación a la versión 8 de .NET en cuanto se haya lanzado, debido a que será una versión con soporte para varios años más que .NET 7.
- **Integración con el sistema de la FIA**: Integración con el sistema de la FIA de gestión de sanciones, que ahora mismo es oculto a los usuarios. De esta manera los usuarios podrían disfrutar de una mejor visualización de las sanciones, mientras que el sistema de la FIA sigue estando oculto.
- **Implementación de historias de usuario pendientes**: Implementación de las historias de usuario HU04 y HU05, cuya baja prioridad ha provocado que no se hayan implementado, pero que aún así serían interesantes.
- **Mejoras de interfaz de usuario**: Mejorar algunos elementos de la interfaz de usuario para hacerla más fácil de usar y más intuitiva.
- **Conversión de la aplicación en una API**: Debido a la facilidad de convertir una aplicación web completa en una API, este podría ser un camino a tomar. De esta forma se podría construir un cliente web con Angular, lo cual permitiría tener una mejor interfaz de usuario, construir una aplicación Android, etcétera.

Bibliografía

[1] Federation Internationale de l'Automobile. 2023 FORMULA ONE SPORTING REGULATIONS. https://www.fia.com/sites/default/files/fia_2023_formula_1_sporting_regulations_-_issue_4_-_2023-02-22.pdf. Último acceso el 02/03/2023.

[2] Federation Internationale de l'Automobile. International sporting code. https://www.fia.com/regulation/category/123. Último acceso el 02/03/2023.

[3] Federation Internationale de l'Automobile. F1 Official Documents. https://www.fia.com/documents/championships/fia-formula-one-world-championship-14/. Último acceso el 28/01/2023.

[4] Universidad de Valladolid. Proyecto docente del trabajo de fin de grado 2022-2023 (Mención Ingeniería de Software). https://albergueweb1.uva.es/guia_docente/uploads/2022/545/46976/1/Documento.pdf. Último acceso el 07/06/2023.

[5] Beck, Beedle, Bennekum, Cockburn y 13 autores más. Manifesto for agile software development. https://agilemanifesto.org/. Último acceso el 05/02/2023.

[6] Schwaber, Sutherland. The scrum guide - the definitive guide to scrum: The rules of the game. https://scrumguides.org/scrum-guide. Último acceso el 05/02/2023.

[7] Scrum.org. What is scrum? https://www.scrum.org/resources/what-scrum-module. Último acceso el 25/05/2023.

[8] Martin Fowler. DomainDrivenDesign. https://martinfowler.com/bliki/DomainDrivenDesign.html. Último acceso el 11/06/2023.

[9] Cucumber. Behaviour-Driven Development. https://cucumber.io/docs/bdd/. Último acceso el 11/06/2023.

[10] Mike Cotterell Bob Hughes. *Software Project Management*. Robert C. Martin Series. McCraw-Hill Education, 2009.

[11] Wrike. What is a risk matrix? https://www.wrike.com/blog/what-is-risk-matrix/#What-is-a-risk-assessment-matrix-in-project-management. Último acceso el 11/06/2023.

[12] Microsoft. Why is my Azure for Students subscription disabled and how do I reactivate it? https://learn.microsoft.com/en-us/azure/cost-management-billing/manage/azurestudents-subscription-disabled. Último acceso el 30/05/2023.

[13] Scrum.org. Agile Metrics: Velocity. https://www.scrum.org/resources/blog/agile-metrics-velocity. Último acceso el 21/06/2023.

[14] R.C. Martin. *Clean Agile*. Robert C. Martin Series. Pearson Education, 2019.

[15] Talent.com. Salario medio para Junior ASP.NET en España, 2023. https://es.talent.com/salary?job=Junior+ASP.NET. Último acceso el 11/05/2023.

[16] Quipu Blog. ¿Cuánto cuesta contratar a un trabajador? https://getquipu.com/blog/cuanto-cuesta-contratar-un-trabajador/. Último acceso el 11/05/2023.

[17] DELL Technologies. Portátil Latitude 5440. https://www.dell.com/es-es/shop/port%C3%A1tiles-dell/nuevo-latitude-5440-port%C3%A1til/spd/latitude-14-5440-laptop/s0051544014es_vp. Último acceso el 11/05/2023.

[18] Cámara Oficial de Comercio, Industria y Servicios de Palencia. PÍO XII Coworking Center. https://cocipa.es/coworking/. Último acceso el 11/05/2023.

[19] Visual Paradigm. Visual Paradigm Complete Price List. https://www.visual-paradigm.com/shop/pricelist.jsp. Último acceso el 13/05/2023.

[20] Microsoft. Buy Visual Studio. https://visualstudio.microsoft.com/vs/pricing/?tab=business. Último acceso el 13/05/2023.

[21] Microsoft Azure. Pricing calculator. https://azure.microsoft.com/en-gb/pricing/calculator/. Último acceso el 13/05/2023.

[22] Eco Cost Savings. How Many Watts Does A Laptop Use? [Actual Usage & Costs Revealed – 1,084 Studied]. https://ecocostsavings.com/how-many-watts-does-a-laptop-use/. Último acceso el 15/05/2023.

[23] Fotocasa. Evolución del precio de la luz en 2023. https://www.fotocasa.es/fotocasa-life/hogar/energia/subida-precio-de-la-luz/. Último acceso el 15/05/2023.

[24] Wikipedia. FURPS. https://en.wikipedia.org/wiki/FURPS. Último acceso el 29/05/2023.

[25] Microsoft. What is .NET? https://learn.microsoft.com/en-us/dotnet/core/introduction. Último acceso el 11/03/2023.

[26] Microsoft. .NET API browser. https://learn.microsoft.com/en-gb/dotnet/api/. Último acceso el 31/03/2023.

[27] NuGet. Create .NET apps faster with NuGet. https://www.nuget.org/. Último acceso el 31/03/2023.

[28] Microsoft. .NET customers showcase. https://dotnet.microsoft.com/en-us/platform/customers. Último acceso el 31/03/2023.

[29] Stack Overflow. Stack Overflow. `https://stackoverflow.com/`. Último acceso el 31/03/2023.

[30] Microsoft. What's new in .NET 7. `https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-7`. Último acceso el 01/04/2023.

[31] Microsoft. What is ASP.NET Core? `https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core`. Último acceso el 01/04/2023.

[32] Microsoft. A tour of the C# language. `https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/`. Último acceso el 01/04/2023.

[33] Microsoft. Entity Framework documentation hub. `https://learn.microsoft.com/en-us/ef/`. Último acceso el 07/06/2023.

[34] Microsoft. DbContext Class. `https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext?source=recommendations&view=efcore-7.0`. Último acceso el 10/06/2023.

[35] Microsoft. Visual Studio 2022 IDE - Programming Tool for Software Developers. `https://visualstudio.microsoft.com/vs/`. Último acceso el 01/04/2023.

[36] Wikipedia. Microsoft SQL Server. `https://en.wikipedia.org/wiki/Microsoft_SQL_Server`. Último acceso el 01/04/2023.

[37] Microsoft. What is Azure Data Studio? `https://learn.microsoft.com/en-us/sql/azure-data-studio/what-is-azure-data-studio?view=sql-server-ver16`. Último acceso el 16/04/2023.

[38] Microsoft. What is Azure? `https://learn.microsoft.com/en-us/training/modules/intro-to-azure-fundamentals/what-is-microsoft-azure`. Último acceso el 07/06/2023.

[39] Microsoft. What is Azure SQL Database? `https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql`. Último acceso el 16/04/2023.

[40] Microsoft. Azure Key Vault basic concepts. `https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts`. Último acceso el 31/03/2023.

[41] Microsoft. Azure Container Apps overview. `https://learn.microsoft.com/en-us/azure/container-apps/overview`. Último acceso el 16/04/2023.

[42] .NET Foundation. xUnit.net. `https://xunit.net/`. Último acceso el 06/04/2023.

[43] SpecFlow by Tricentis. BDD Framework for .NET - SpecFlow - Enhance Your Automated Tests. `https://specflow.org/`. Último acceso el 06/04/2023.

[44] Cucumber. Gherkin Syntax. `https://cucumber.io/docs/gherkin/`. Último acceso el 06/04/2023.

[45] Selenium. Selenium. `https://www.selenium.dev/`. Último acceso el 06/06/2023.

[46] Autofac. Autofac. `https://autofac.readthedocs.io/en/latest/`. Último acceso el 14/03/2023.

[47] Bootstrap. Bootstrap · The most popular HTML, CSS, and JS library in the world. `https://getbootstrap.com/`. Último acceso el 06/06/2023.

[48] OpenAI. ChatGPT. `https://chat.openai.com/chat`. Último acceso el 01/04/2023.

[49] Git. Git. `https://git-scm.com/`. Último acceso el 27/04/2023.

[50] Scott Chacon, Ben Straub. *Pro Git*. Apress, 2021. Último acceso el 29/05/2023.

[51] Atlassian. Git Branch. `https://www.atlassian.com/git/tutorials/using-branches`. Último acceso el 29/05/2023.

[52] Microsoft. What is Azure DevOps? `https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops`. Último acceso el 29/05/2023.

[53] Microsoft. What is Azure Repos? `https://learn.microsoft.com/en-us/azure/devops/repos/get-started/what-is-repos`. Último acceso el 06/06/2023.

[54] Microsoft. What is Azure Boards? `https://learn.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards`. Último acceso el 06/06/2023.

[55] Microsoft. What is Azure Pipelines? `https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines`. Último acceso el 06/06/2023.

[56] Object Management Group (OMG). Unified Modeling Language, v2.5.1. `https://www.omg.org/spec/UML/2.5.1/PDF`. Último acceso el 01/04/2023.

[57] Object Management Group® (OMG®). Object Constraint Language. `https://www.omg.org/spec/OCL/`. Último acceso el 02/06/2023.

[58] Visual Paradigm. Visual Paradigm Product Overview. `https://www.visual-paradigm.com/support/documents/vpuserguide/12/13/5963_visualparadi.html`. Último acceso el 01/04/2023.

[59] Overleaf. Overleaf. `https://www.overleaf.com/`. Último acceso el 01/04/2023.

[60] Clockify. Clockify - FREE Time Tracking Software. `https://clockify.me/`. Último acceso el 01/04/2023.

[61] Rocket.Chat. Rocket.Chat: Communications Platform You Can Fully Trust. `https://www.rocket.chat/`. Último acceso el 01/04/2023.

[62] Steve "ardalis" Smith. *Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure*. Microsoft Developer Division, .NET, and Visual Studio product teams, 2021. Último acceso el 12/02/2023.

[63] IBM. What is three-tier architecture? `https://www.ibm.com/topics/three-tier-architecture`. Último acceso el 01/05/2023.

[64] Microsoft. Overview of ASP.NET Core MVC. https://learn.microsoft.com/en-gb/aspnet/core/mvc/overview?WT.mc_id=dotnet-35129-website&view=aspnetcore-7.0. Último acceso el 08/05/2023.

[65] Microsoft. Views in ASP.NET Core MVC. https://learn.microsoft.com/en-gb/aspnet/core/mvc/views/overview?view=aspnetcore-7.0. Último acceso el 22/05/2023.

[66] Microsoft. Model-View-ViewModel (MVVM). https://learn.microsoft.com/en-gb/dotnet/architecture/maui/mvvm. Último acceso el 23/05/2023.

[67] Ardalis. What is the difference between a DTO and a POCO (or POJO). https://ardalis.com/dto-or-poco/. Último acceso el 23/05/2023.

[68] Microsoft. Model Binding in ASP.NET Core. https://learn.microsoft.com/en-gb/aspnet/core/mvc/models/model-binding?view=aspnetcore-7.0. Último acceso el 19/06/2023.

[69] Microsoft. Model validation in ASP.NET Core MVC and Razor Pages. https://learn.microsoft.com/en-gb/aspnet/core/mvc/models/validation?view=aspnetcore-7.0. Último acceso el 19/06/2023.

[70] Microsoft. Design the infrastructure persistence layer. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design. Último acceso el 24/04/2023.

[71] Martin Fowler. Repository. https://martinfowler.com/eaaCatalog/repository.html. Último acceso el 10/06/2023.

[72] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[73] Martin Fowler. Domain Event. https://www.martinfowler.com/eaaDev/DomainEvent.html. Último acceso el 07/06/2023.

[74] Microsoft. Domain events: Design and implementation. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/domain-events-design-implementation. Último acceso el 07/06/2023.

[75] Martin Fowler. DDD_Aggregate. https://martinfowler.com/bliki/DDD_Aggregate.html. Último acceso el 11/06/2023.

[76] Ardalis. Aggregate Responsibility Design. https://ardalis.com/aggregate-responsibility-design/. Último acceso el 11/06/2023.

[77] Selenium. Page object models. https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/. Último acceso el 13/05/2023.

[78] Specflow. Driver Pattern. https://docs.specflow.org/projects/specflow/en/latest/Guides/DriverPattern.html. Último acceso el 09/06/2023.

[79] Lev Gorodinski. Services in Domain-Driven Design (DDD). http://gorodinski.com/blog/2012/04/14/services-in-domain-driven-design-ddd/. Último acceso el 11/06/2023.

[80] Microsoft. Value Conversions - EF Core. https://learn.microsoft.com/en-us/ef/core/modeling/value-conversions?tabs=data-annotations. Último acceso el 11/06/2023.

[81] Steve Smith - Ardalis. Clean architecture. https://github.com/ardalis/CleanArchitecture. Último acceso el 15/06/2023.

[82] Microsoft. Introduction to Identity on ASP.NET Core. https://learn.microsoft.com/en-gb/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio. Último acceso el 14/03/2023.

[83] Microsoft. Configuration in ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-7.0. Último acceso el 19/06/2023.

[84] Specflow by Tricendis. Specflow. https://specflow.org/tools/specflow/. Último acceso el 14/03/2023.

[85] Specflow. Automate an ASP.NET Core Application with Selenium and SpecFlow. https://www.youtube.com/playlist?list=PL5lX-Y5KVGPrYBlVWUgD4DjtJdmCkGjqa. Último acceso el 14/03/2023.

[86] C# Corner. What Is Startup Class And Program.cs In ASP.NET Core. https://www.c-sharpcorner.com/article/what-is-startup-class-and-program-cs-in-asp-net-core/. Último acceso el 14/03/2023.

[87] Microsoft. WebApplicationFactory<TEntryPoint> Class. https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.testing.webapplicationfactory-1?view=aspnetcore-7.0. Último acceso el 14/03/2023.

[88] Microsoft. Integration tests in ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-7.0. Último acceso el 14/03/2023.

[89] SpecFlow. SpecFlow - Autofac. https://docs.specflow.org/projects/specflow/en/latest/Integrations/Autofac.html. Último acceso el 14/03/2023.

[90] StackOverflow. How to run Specflow tests one after another NOT parallel. https://stackoverflow.com/questions/50939235/how-to-run-specflow-tests-one-after-another-not-parallel. Último acceso el 21/06/2023.

[91] Microsoft. ASP.NET Core MVC with EF Core - tutorial series. https://learn.microsoft.com/en-gb/aspnet/core/data/ef-mvc/?view=aspnetcore-7.0. Último acceso el 19/06/2023.

[92] DevIQ. Specification Pattern. https://deviq.com/design-patterns/specification-pattern. Último acceso el 19/06/2023.

Apéndice A

Manuales

A.1. Manual de despliegue en local

Para desplegar la aplicación en una máquina local, son necesarios los siguientes requisitos:

- Tener instalado .NET 7.
- Tener instalado el sistema gestor de bases de datos SQLSERVER, y tenerlo arrancado en el momento de arrancar la aplicación.
- Tener instalado Visual Studio 2022. Esto no es estrictamente necesario, pero el despliegue se realizará desde esta aplicación (aunque es posible hacerlo mediante la línea de comandos de .NET).

Una vez listos estos requisitos, descargar el proyecto del repositorio del código y abrir la solución, el fichero `TFG.RulesPenaltiesF1.sln`.

Una vez abierto, seleccionar el perfil de despliegue 'https', y presionar las teclas `ctrl + F5`. La aplicación se arrancará, poblando la base de datos si estuviera vacía.

A.2. Manual de despliegue en Azure

Para realizar el despliegue de la aplicación en Azure, primero se tendrá abierto el proyecto en Visual Studio. Tras esto, se hará click derecho sobre el proyecto *TFG.RulesPenaltiesF1.Web*, que es el punto de arranque de la aplicación.

Se hará click sobre 'Publicar' y, si no se tiene creado aún, habrá que crear un perfil de publicación. En la Figura A.1 se muestra el menú que aparece al hacer crear el perfil de publicación.

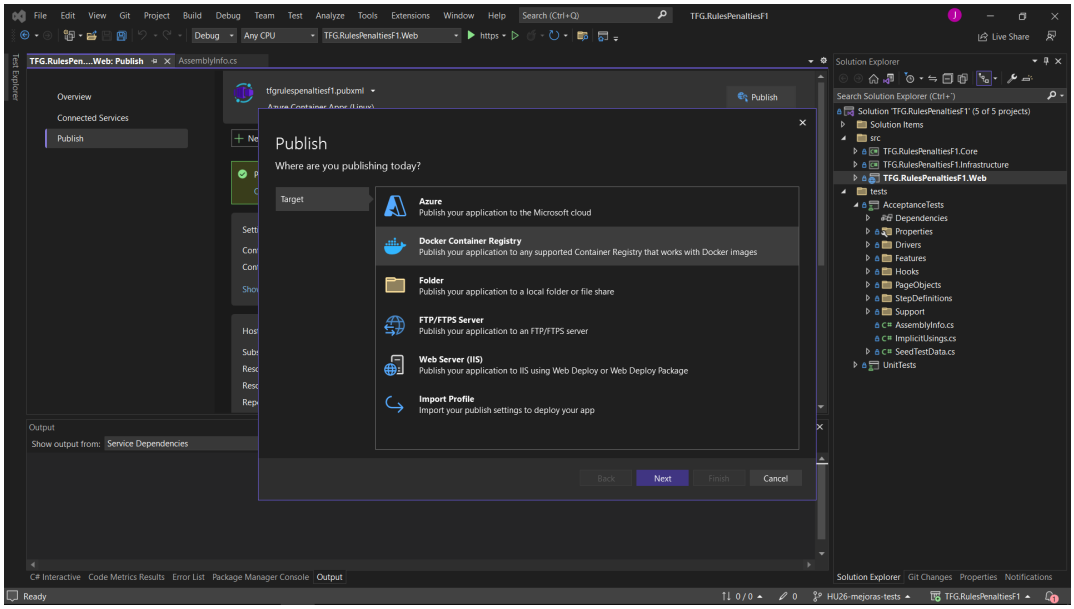


Figura A.1: Creación de perfil de publicación en Visual Studio.

Se elegirá publicarlo en un registro de contenedores Docker, y a continuación habrá que rellenar unos datos que tienen que ver con la cuenta Azure en la que se va a publicar. Una vez hecho este perfil, se arrancará Docker y se hará click sobre publicar con el perfil especificado, y la aplicación será desplegada en Azure.

A.3. Manual de mantenimiento

Para añadir nuevo contenido se creará una rama a partir de develop con un nombre descriptivo de la característica nueva a implementar (Ver Subsección 4.2.1). Si cambia el modelo de datos, se revisarán las configuraciones de la base de datos en la carpeta *Config* dentro del proyecto `src/TFG.RulesPenaltiesF1.Infrastructure`, y se realizará y aplicará una migración según lo indicado en [81].

Una vez terminada la implementación de la nueva característica, se realizarán los tests pertinentes para asegurar que todo lo desarrollado anteriormente sigue funcionando correctamente. Se fusionará la nueva característica con la rama develop y posteriormente con la rama master, y se realizará el despliegue en Azure para actualizar el sistema desplegado.

A.4. Manual de usuario

Nota: es posible que las vistas mostradas en este anexo difieran ligeramente de las vistas de la aplicación desplegada debido a mejoras en la interfaz de usuario.

La primera vez que se accede a la aplicación se hace como usuario no identificado. Esto significa que se puede acceder a todas las partes de la aplicación, pero no es posible realizar operaciones de escritura (creación de registros, avance de competiciones, etcétera).

Para iniciar sesión hay que hacer click sobre el enlace en la parte superior derecha, *Login* que aparece en la Figura A.2.

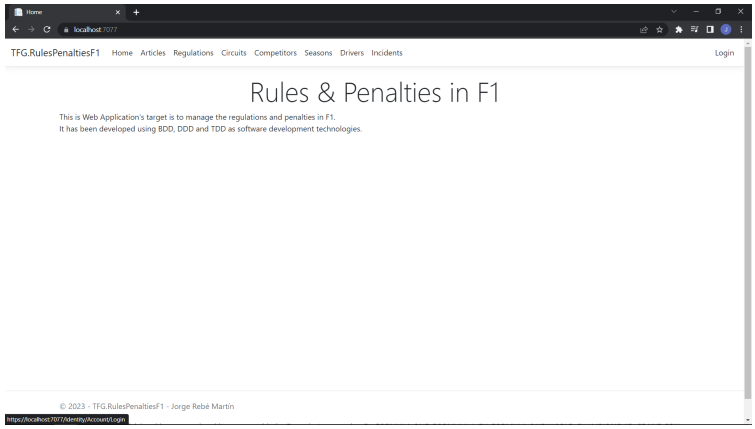


Figura A.2: Página principal de la aplicación.

El navegador irá a la página de incicio de sesión, que aparece en la Figura A.3. Tras esto, el comisario o jefe de equipo iniciará sesión con su cuenta.

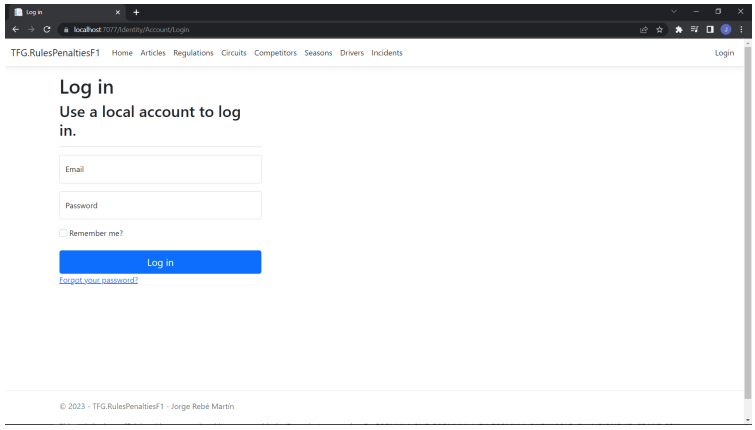


Figura A.3: Página de inicio de sesión.

Dado que en la app existen los dos roles especificados, se explicarán las funciones a las que puede acceder cada uno de ellos por separado.

Comisario

Una vez iniciada sesión como comisario, se pueden acceder a distintas opciones que se van a explicar a continuación.

Primero, los comisarios pueden crear un artículo, indicando contenido del artículo y subartículos si los hubiera. En la Figura A.4 se muestra la página de creación de un artículo.

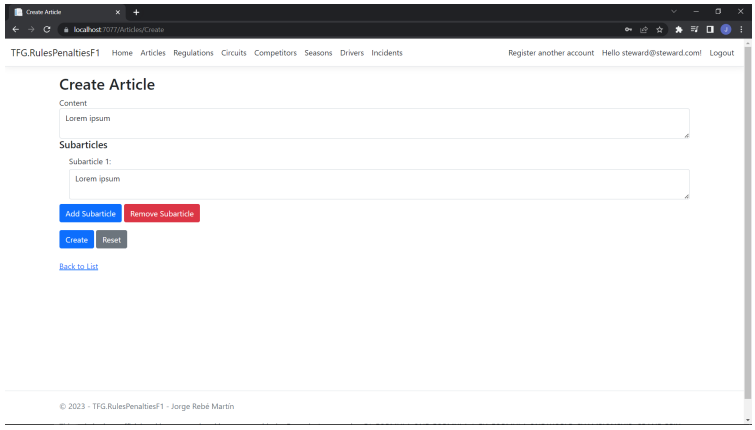


Figura A.4: Página de creación de artículos.

Después, se pueden crear reglamentos, dándoles un nombre y seleccionando los artículos y penalizaciones que lo componen. En la Figura A.5 se muestra la página de creación de reglamento.

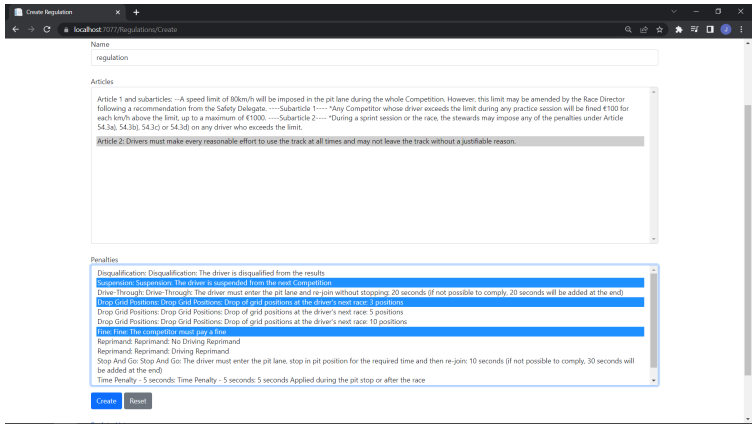


Figura A.5: Página de creación de reglamentos.

Lo siguiente es la creación de circuitos en los que ocurrirán las competiciones. En la Figura A.6 se muestra la página de creación de circuitos.

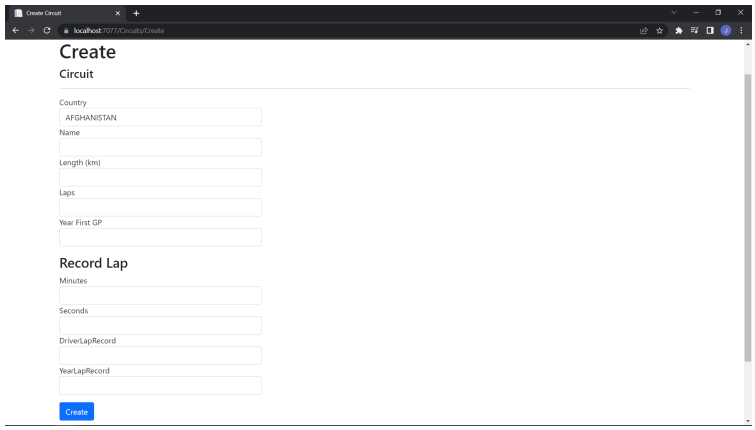


Figura A.6: Página de creación de circuitos.

Tras esto, lo siguiente es la creación de competidores. Para la creación de un competidor es necesario que el jefe de equipo de ese competidor sea un usuario registrado en el sistema. Si no lo está, el comisario deberá registrarlo antes de crear al competidor. En la Figura A.7 se muestra la página de creación de competidores.

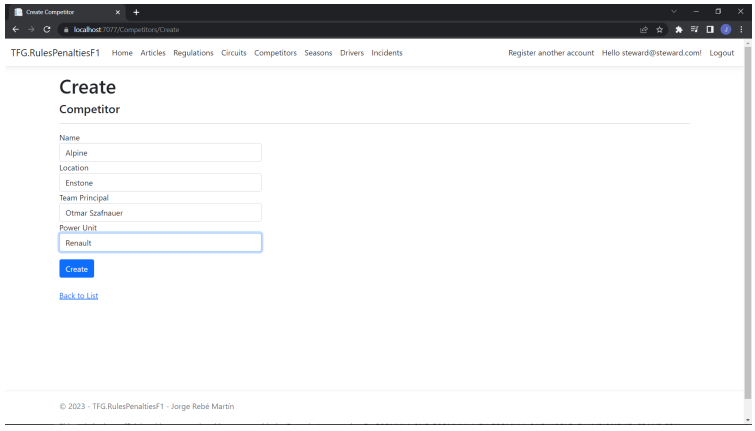
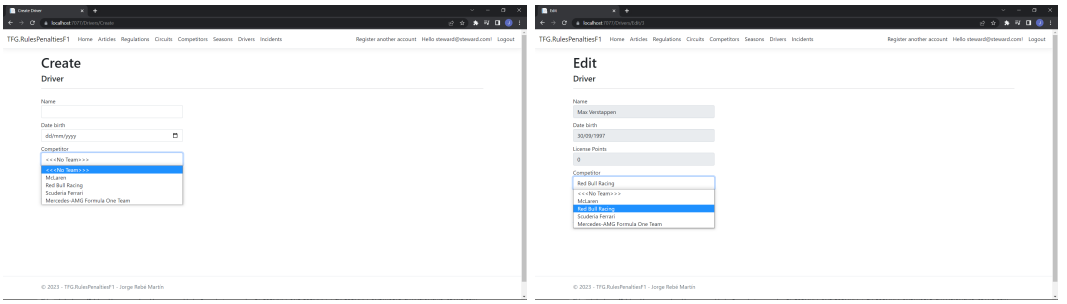


Figura A.7: Página de creación de competidores.

En cuanto a la creación de los pilotos, hay que destacar que el competidor es opcional. Se puede seleccionar en el momento de creación, o editarlo más adelante. En la Figura A.8 se muestra la página de creación de pilotos.



Página de creación de pilotos. Página de edición de pilotos.

Figura A.8: Páginas de creación y edición de pilotos.

Lo siguiente es la creación de una temporada. Se seleccionará un mínimo de dos competidores, de dos competiciones y un reglamento. En la Figura A.9 se muestra la página de creación de temporadas.

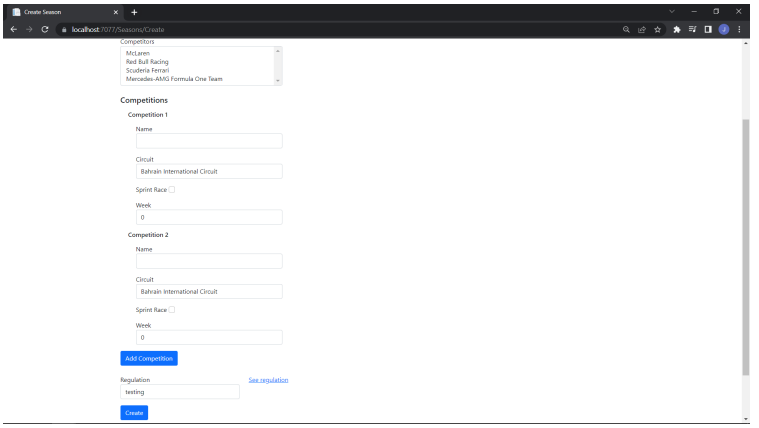
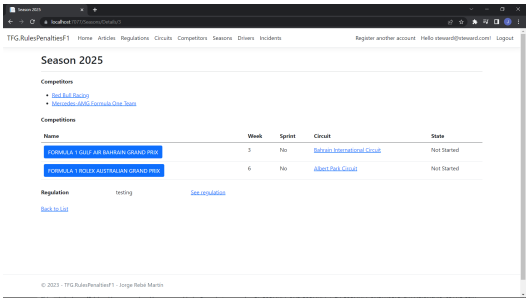
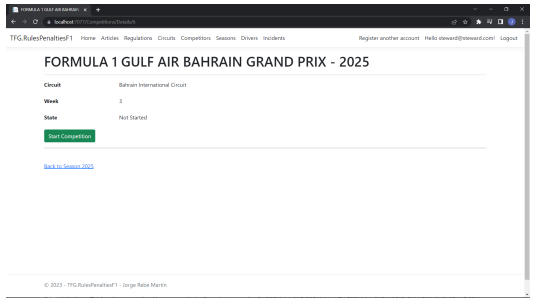


Figura A.9: Página de creación de temporadas.

Una vez creada una temporada, se llega al núcleo de la funcionalidad de la aplicación. En los detalles de una temporada aparece una lista con las competiciones que se han creado. Accediendo a la primera, cuando el estado es no comenzado aparece la opción de comenzarla. En la Figura A.10 se muestra las páginas de detalles de una temporada y los detalles de una competición sin comenzar.



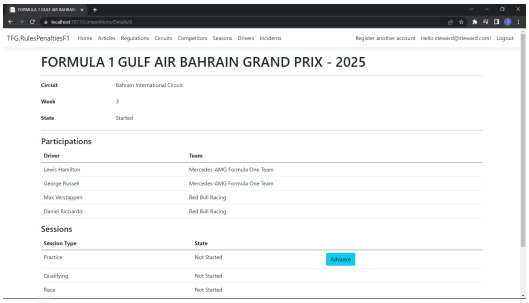
Detalles de una temporada.



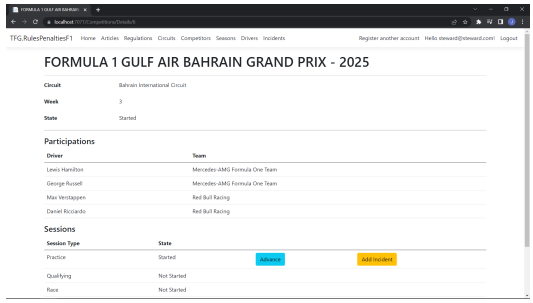
Detalles de una competición no empezada.

Figura A.10: Páginas de detalles de una temporada y detalles de una competición sin comenzar.

Una vez se da comienzo a una competición, los jefes de equipo de los competidores participantes en la temporada deben añadir las participaciones de sus pilotos para esa competición. Una vez añadidas, el comisario tiene el control sobre la competición y sus sesiones. Ahora deberá avanzar las sesiones, y cuando están en curso (estado comenzadas), podrá añadir incidentes. En la Figura A.11 se una página de detalles de competición donde se muestra cómo se cambia de estado y se pueden añadir incidentes.



Competición comenzada con participaciones.



Transcurso sesiones dentro de competición.

Figura A.11: Páginas de competición y transcurso de sesiones.

Cuando un piloto infringe un artículo de la temporada, entonces un comisario añade un incidente a la sesión. Selecciona el artículo y penalización (si procede) del reglamento de la temporada de la sesión, y lo añade. En la Figura A.12.

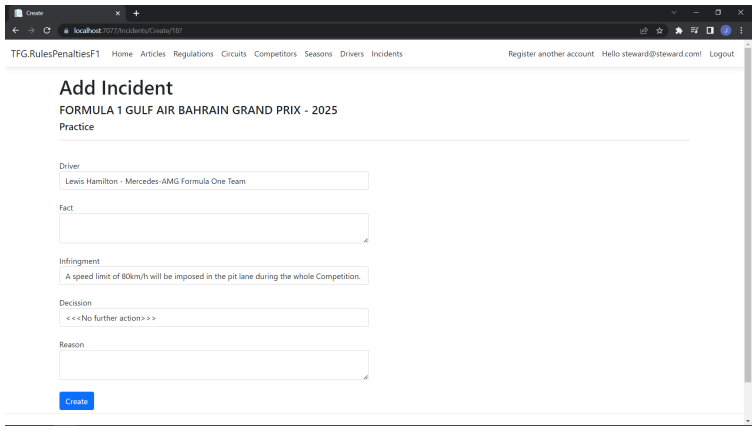


Figura A.12: Página de creación de incidentes.

Por último, un comisario puede registrar nuevos comisarios o jefes de equipo. En la Figura A.13 se muestrá la página de creación de cuentas.

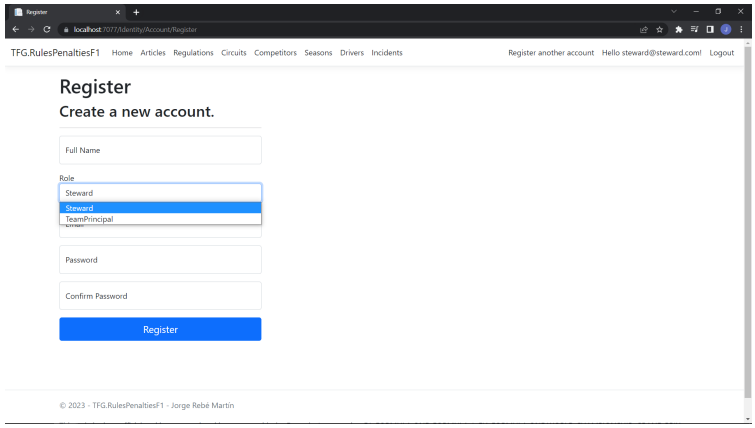
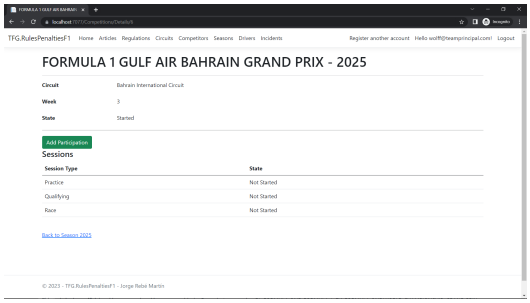


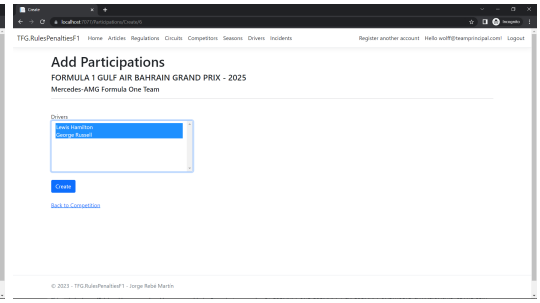
Figura A.13: Página de creación de cuentas.

Jefe de equipo

La operación de la que se encarga un jefe de equipo es añadir las participaciones del competidor al que representa para una determinada competición. En la Figura A.14 se muestra la página de detalles de una competición comenzada, pendiente de añadir participaciones para el jefe de equipo que tiene sesión iniciada, y la página de creación de una participación para ese jefe de equipo.



Competición vista por un jefe de equipo.



Página creación de participación.

Figura A.14: Páginas de detalles de una temporada y detalles de una competición sin comenzar.

Cualquier usuario

Cualquier usuario tiene permisos para ver cualquier tipo de registro. Dentro de los registros más interesantes encontramos a los incidentes. Al listado de incidentes se les puede aplicar algunos filtros como el piloto involucrado o la sesión, además de ordenar por año de ocurrencia. En la Figura A.15 se muestra el listado de incidentes.

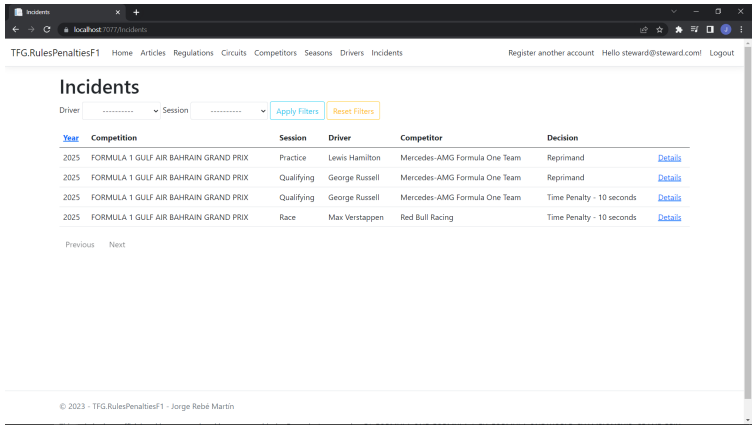


Figura A.15: Página de lista de incidentes.

Apéndice B

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código: `https://dev.azure.com/jorgerebe/_git/TFG.RulesPenaltiesF1`.
- URL del sistema desplegado: `https://tfgrulespenaltiesf1.blueriver-b9992773.westeurope.azurecontainerapps.io/`