Universidad Internacional de La Rioja (UNIR)

ESIT

Máster universitario en Dirección e Ingeniería de Sitios Web

RAMA PROFESIONAL

Progressive Web App: Aplicación para la gestión diaria de entrenamientos para equipos de fútbol

Trabajo Fin de Máster

presentado por: Otero Carbón, Noemí

Director/a: Soltero Domingo, Francisco José

Ciudad: Leeds (Reino Unido)
Fecha: 20/07/2020

RESUMEN

En este TFM se ha realizado la planificación de una aplicación web para la programación y gestión de sesiones de entrenamiento de futbolistas. La herramienta ha sido enfocada tanto a la gestión grupal como individual de los jugadores, aportando una solución de trabajo tanto online como offline.

Para ello se ha utilizado una metodología combinada entre Scrum y una más tradicional, Proceso Unificado de Desarrollo (RUP), desarrollando los siguientes pasos: estudio de mercado con la correspondiente definición de los requisitos funcionales y no funcionales, la descripción de los casos de uso, el diseño lógico del sistema (modelo relacional, diagramas de secuencia y de clases) y demás elementos que componen el desarrollo de la planificación de la aplicación web.

Para el desarrollo web se ha considerado utilizar la tecnología PWA según un patrón MVC. Se hará especial mención en la utilización de Service Workers para proporcionar conexión offline.

Palabras Clave: PWA, Preparación física, Futbol, RUP, Scrum.

ABSTRACT

This dissertation addresses the task of planning and conducting a football session planning app. The proposed tool essentially involves managing and reporting progress to improve performance, both at an individual level, and at a group level, offering an offline experience.

In order to do so, we have used a hybrid methodology combining Scrum and the Rational Unified Process (RUP), comprising the following steps: marketing analysis, functional and non-functional requirements, user case definition, design of conceptual model (relational model, sequence diagrams and class diagrams), apart from describing the implementation process.

It has been considered to build a Progressive Web App (PWA) based on the MVC pattern. A special mention must be made of the Service Workers implementation that will enable the offline capabilities.

Keywords: PWA, Fitness coach, Football, RUP, Scrum

ÍNDICE

ÍNDICE DE IMÁGENES

1. INTRODUCCIÓN

El control del entrenamiento en los equipos de fútbol profesional es cada vez un proceso más y más complejo, debido a la gran cantidad de información a la que hoy en día los clubes tienen acceso (partidos, datos físicos, análisis de los procesos pedagógicos, etc.). Esta exigencia en el rendimiento justifica el uso de herramientas de Software para gestionar y controlar sus entrenamientos.

Los entrenadores y preparadores físicos necesitan realizar tanto entrenamientos individuales como grupales, además de sesiones específicas. Los motivos pueden ser tanto el proceso de acondicionamiento o preparación física del jugador, como los procesos de rehabilitación funcional y readaptación físico-deportiva. En muchas ocasiones los entrenadores y preparadores físicos se encuentran en medio del campo de entrenamiento, alejados de las instalaciones y sin acceso a internet. Son habituales también los viajes entre partidos, tanto por carrera, tren o avión, momentos en los cuales las conexiones no son lo suficientemente buenas o incluso inexistentes y, sin embargo, esos son también momentos para preparar las sesiones futuras. Puede que además dispongan de varios dispositivos (ordenadores de sobremesa, portátiles, tableta, o móviles) para realizar su trabajo en función de la localización. Y lo más importante, necesitan realizar ese trabajo lo más rápido y sencillo posible para dedicar tiempo a lo que realmente aporta calidad al equipo: la supervisión de los jugadores, el control y la planificación de las cargas, la planificación de ciclos de trabajo como pretemporadas y la posibilidad de tener tiempo para crear nuevas tareas y ejercicios.

Es por ello por lo que en este trabajo de fin de máster se plantea el desarrollo de una aplicación PWA (Progressive Web App). El usuario debe tener capacidad de diseñar tanto sesiones grupales como individuales según la planificación semanal del equipo. A mayores debe ofrecer capacidad de almacenar datos lesionales y de almacenar ejercicios, tareas y plantillas de sesiones que agilizarán el diseño posterior de las sesiones. Pero la principal característica es que será multiplataforma y la posibilidad de acceso a la aplicación aún sin conexión. La herramienta se diseñará con un “look and feel” de una App, lo que hará más fácil y placentera la experiencia al usuario.

Dado que la gran mayoría de los entrenadores se mueven con su propio equipo de trabajo, entre los cuales destacan las figuras del preparador físico y del readaptador, la finalidad de esta aplicación no es la de realizar un software de gestión integral. El objetivo es diseñar una herramienta específica para el diseño de sesiones a un menor coste, que permita a los preparadores físicos desarrollar su trabajo diario independientemente de las herramientas internas del club y que puedan migrar fácilmente a otros equipos.

2. ESTADO DEL ARTE

A continuación, se presenta el estado del arte del ámbito de aplicación de este trabajo. En primer lugar, es necesario conocer en detalle las necesidades de planificación, control y gestión del entrenamiento en el fútbol profesional, para así poder diseñar una herramienta que dé soporte, facilite y agilice estos procesos a los preparadores físicos profesionales.

El objetivo es ofrecer una solución lo más ajustada posible a las necesidades reales de los usuarios que realmente satisfaga sus expectativas y posibilite la continuidad y desarrollo del proyecto. Una vez comprendidas las necesidades, se continúa con la investigación del estado del arte en el sector para comprobar si ya existe alguna herramienta que dé solución a esas necesidades. En caso de que todavía no exista dicha solución, y con el conocimiento sobre el funcionamiento de las actuales, se procede al estudio de las distintas tecnologías de desarrollo, para analizar las ventajas y desventajas de cada una frente a nuestras necesidades y así poder seleccionar la que sea más conveniente para la aplicación a desarrollar.

2.1. Estudio de la planificación y programación de la preparación física de un equipo de futbol

En el sector del futbol profesional el cuerpo técnico de un equipo de futbol está compuesto por diversos equipos o departamentos. Dichos departamentos pueden depender del club directamente o pueden venir asociados a un entrenador. A su vez, el personal que depende del club puede venir influenciado por las preferencias del entrenador y es común que, en ciclos relativamente cortos varíen.

A menudo, la existencia de estos diferentes grupos de trabajo implica que detrás existen metodologías de trabajo diferentes y como resultado el diseño de un único software interdisciplinar no es viable. En la mayoría de los casos el coste de estas aplicaciones suele ser costoso y no está justificado si no es usado por todos los trabajadores. Como consecuencia de la naturaleza de este trabajo, los entrenadores asociados al entrenador suelen rechazar el uso de este tipo de herramientas, dado que no tienen garantía de poder seguir manteniendo la suscripción al software en el próximo club al que vayan y eso conllevaría a la pérdida del trabajo no confidencial realizado, en cuanto a diseño de tareas y creación de planes de trabajo.

En las últimas décadas del siglo pasado y hasta la actualidad, el fútbol en general y la forma de abordar su entrenamiento han sufrido una gran evolución. En la actualidad los diferentes modelos de planificación del entrenamiento en el fútbol comparten un enfoque estructurado,

a través del que se ordenan temporalmente las estimulaciones que se le van a dar al equipo o al jugador. Estas estrategias de organización de los elementos de entrenamiento pueden ser variadas, no ya solo entre metodologías, sino que la aplicación de cada metodología concreta será a la vez personalizada por cada entrenador, dando mayor o menor relevancia a cada una de ellas. La adaptabilidad y su control, tanto de las variables cualitativas y cuantitativas, tanto a nivel individual como colectivo es esencial para permitir su análisis y valoración.

La competición es el principal determinante para la realización de la programación. Lo común es jugar un partido por semana, en el caso del fútbol profesional, por lo que la unidad temporal básica que se toma como referencia es el microciclo, un ciclo semanal. En el caso de que haya 2 partidos por semana, o incluso 3, se hablará igualmente de microciclos, con 2 o 3 episodios competitivos.

Hoy por hoy, tanto la comunidad científica como los entrenadores hacen uso de sofisticadas herramientas de análisis para tratar de identificar, combatir y contrarrestar aquellos identificadores de rendimiento técnico-tácticos que puedan llevar a conseguir un gol, y por tanto ganar un partido (Santos, Lago-peñas, García-garcía, Santos, & Lago-peñas, 2017; Sarmento et al., 2017). Las estrategias de juego, los comportamientos técnico-tácticos y las variables situacionales y contextuales, son considerados los factores más significativos en el resultado final de un partido en las principales ligas europeas, por delante de la cantidad del desempeño físico que hayan realizado los jugadores, el cual sabemos que se ve totalmente condicionado por las variables contextuales del partido, como por las exigencias de las diferentes posiciones de los jugadores en el campo (Ade, Fitzpatrick, & Bradley, 2016; Lago-Peñas, 2013; Lago, Casais, Dominguez, & Sampaio, 2010; Miñano-Espin, Casáis, Lago-Peñas, & Gómez-Ruano, 2017), es por ello que a pesar de diferencias culturales, de estilos de juego, principios tácticos o de niveles de habilidad de los jugadores, esta evolución en el conocimiento del fútbol está permitiendo a los entrenadores diseñar situaciones de entrenamiento más específicas que les ayuden a mejorar la efectividad en las diferentes situaciones de juego así como a afinar más en la preparación individual del futbolista, las dos vertientes hacia las que se está orientando el entrenamiento actual: el entrenamiento del equipo y el entrenamiento del jugador. El jugador trabaja individualmente para tener su cuerpo a punto, para poder desarrollar posteriormente las exigencias del entrenamiento grupal y así estar en buena disposición para competir de forma óptima.

A su vez, en las últimas décadas, el interés de la comunidad científica ha aumentado exponencialmente, multiplicándose la producción literaria relacionada con la prevención y la readaptación de las lesiones deportivas. Esto está permitiendo a los preparadores físicos

conocer cada vez más sobre epidemiología, mecanismos lesionales, factores de riesgo, procesos de readaptación y reentrenamiento, etc., así como los efectos de las lesiones en el rendimiento colectivo. Esto conlleva un mayor desarrollo y conocimiento en el área del acondicionamiento físico, lo que le permite cada vez más individualizar estos entrenamientos.

Esta mayor especialización por parte de los preparadores físicos implica que a la hora de diseñar las sesiones de trabajo de los deportistas, tanto en campo como en gimnasio, deban crear situaciones con un mayor grado de especificidad, dejando atrás los ejercicios básicos de libro que la mayoría de las aplicaciones web contienen. Supone esto que la aplicación a desarrollar debe permitir al usuario tener la libertad de grabar o subir fotos o pequeños videos explicativos de las tareas con el fin de que puedan crear y desarrollar su propio banco de ejercicios.

Otro de los factores que caracterizan el futbol de alta competición, y de otros muchos deportes en equipo, es la movilidad. Se requiere que tanto jugadores como cuerpo técnico se desplacen a los diversos lugares donde tiene lugar la competición y en ocasiones varias veces por semana. Según las fuentes consultadas esto impide que el cuerpo técnico, en este caso el/los preparadores físicos, puedan aprovechar esos traslados para programar las tareas con sus jugadores.

Alternativamente pueden desarrollar su trabajo con medios tradicionales, como son las anotaciones, pero en muchas ocasiones no tienen acceso a información relevante como los porcentajes de carga totales de la sesión, o incluso sesiones anteriores. Esto implica también que tengan que viajar con un ordenador portátil al cual pasar las sesiones o resultados a posteriori.

2.2. Estudio del sector de las aplicaciones web en el ámbito de la preparación física en el futbol

En el estudio llevado a cabo se han identificado tres tipos de aplicaciones enfocadas a la preparación física o a la planificación de entrenamientos deportivos. Dentro de la clasificación obtenida, se han dejado de lado los asistentes de entrenamiento que el usuario puede descargar para monitorizar su actividad diaria o su propio entrenamiento sin la intervención de un profesional, dado que ese no es el objetivo del proyecto (Byun, Chiu, & Bae, 2018).

Por un lado, se han identificado aplicaciones móviles para la preparación deportiva enfocadas principalmente al entrenamiento individual. El enfoque no está centrado en la alta competición, si no que está pensado en el mercado de la preparación física del fitness. Tienen como principal función el asistir al profesional en el diseño de las sesiones de entrenamiento, pero

tienen como tónica general un enfoque generalista, con bases de datos predefinidas en las que se pueden encontrar ejemplos de entrenamientos, aunque algunas también ofrecen la posibilidad de diseñar ejercicios propios o modelos de sesiones. Otra de las prestaciones que suelen ofrecer estás aplicaciones es la de gestionar clientes, generalmente ofrecen medios para tramitar pagos, realizar seguimiento de las lesiones, hacer entrenamientos en remoto y ofrecer feedback de los entrenamientos. Algunos ejemplos de estas aplicaciones son: TrueCoach, Gain Trainer, The Training Notebook, FitSW, Trainerize, My PT Hub, PTminder, TotalCoaching o Halo, entre otras.

El siguiente grupo, es el de las aplicaciones de rehabilitación. Generalmente de carácter más individual, ideales para clínicas de rehabilitación, que permiten realizar programas de entrenamiento individuales, plantillas de entrenamiento, capacidad de crear ejercicios propios, y a su vez realizar seguimiento individualizado de los pacientes. La mayoría de estas herramientas permiten usarse como aplicaciones de escritorio, pero a su vez ofrecen versión Android y Apple. Aquí se pueden mencionar: BlueJay, RehabTherx, Rehabguru, o MyPhysioRehab.

Un tercer grupo de aplicaciones sería el formado por más específicas al ámbito de fútbol y enfocado a la planificación grupal de sesiones desde un punto de vista táctico. Estas herramientas están centradas en el trabajo en campo, en algunos de los ejemplos permiten incluso el diseño 3D, pero no posibilitan la elaboración de tareas individuales específicas en gimnasio. Algunos de los nombres que destacan tras la realización de la búsqueda son: Soccerlab, Sport Session Planner, Easy2Coach, PlayManager, CoachFX.

Por finalizar, mencionar el último conjunto que se adapta más al objetivo del proyecto. Estas aplicaciones tratan de enfocarse a la mejora y la monitorización del rendimiento grupal del equipo, y están más pensadas en la preparación física. Es por ello que se va a realizar un análisis más detallado de algunas de las más populares para poder analizar las funcionalidades de cada una:

- AthleteMonitoring: Software enfocado al rendimiento grupal. Ofrece cuatro aparatados: monitorización del estado de los atletas, planificación de la carga de trabajo, monitorización de lesiones y análisis de rendimiento. La monitorización de los atletas conlleva la realización de cuestionarios que permiten a los preparadores físicos adaptar las cargas de los entrenamientos. Permite planificar sesiones de entrenamiento individualizado y recoger información de los GPS y demás dispositivos inteligentes usados durante el entrenamiento, el software permite analizar las cargas y los posibles factores de riesgo de lesión aplicando algoritmos que tratan de identificar patrones

característicos en las respuestas a los cuestionarios post-entreno (“Athlete monitoring,” 2020).

- Actimet: Esta plataforma software permite monitorizar, evaluar y asistir en la toma de decisiones a la hora de gestionar el rendimiento de los atletas, manejando el volumen y la intensidad de los entrenamientos (“Actimet,” 2020).

- CoachMePlus: Permite la creación de programas de entrenamiento y la aplicación de algoritmos de control sobre la información registrada en los distintos dispositivos utilizados durante el entrenamiento. Entre las marcas que ofrecen compatibilidad con esta herramienta destacan: Garmin, Polar, Catapult, FirstBeat, entre otros (“CoachMePlus,” 2020).

- FusionSport: Ofrece similitudes con los dos softwares anteriores. Permite la captura y el análisis de los datos de los entrenamientos, así como la planificación de programas. Una de sus características principales es que ofrece la integración con los principales sensores, GPS y dispositivos utilizados para monitorizar entrenamientos (“Fusion Sport,” 2020).

- Soccer SystemPro: Quizás el más enfocado a la figura del preparador físico, permite la planificación de tareas por microciclos, diseñar tareas personalizadas tanto individuales como grupales y la generación de informes sobre las lesiones (“Soccer SystemPro,” 2020).

Se han visto fantásticas herramientas que permiten la captura de datos de los distintos sensores utilizados en los entrenamientos y partidos, y que hacen uso de algoritmos predictivos. Sin embargo, en un club profesional de futbol, la tarea del análisis cuantitativo de las cargas es realizada por otro profesional especializado (analista de rendimiento físico), pero después la interpretación y posterior prescripción de entrenamiento basada en la metodología que ese grupo de entrenadores requiere de otra herramienta que se ajuste a estas necesidades.

De la investigación realizada se ha sacado como conclusión que, excepto Soccer SystemPro, ningún software permite diseñar tareas de forma individual y de forma grupal adaptándose a los requisitos mencionados. A su vez, no existe ninguna aplicación que te permita el diseño de las sesiones utilizando como unidad de medida el microciclo, lo que agiliza la programación de contenidos según los profesionales consultados. Por último, ninguna de estas opciones ofrece la posibilidad de diseñar sesiones de forma offline. De ser posible esta última

característica, el uso de esa aplicación supondría un aumento de productividad con respecto a las opciones disponibles en el mercado.

2.3. Estudio del arte de las tecnologías de desarrollo web

Herramientas y frameworks de trabajo

JavaScript es el lenguaje por excelencia de la web, ya que en casi todas las webs que consultamos está presente en alguna forma: bien en los componentes inteligentes, o bien a la hora de hacerla más responsive o atractiva al usuario (Rozentals, 2017).

El correcto uso de JavaScript puede convertir el uso de una página web o aplicación en una experiencia fácil y placentera para el usuario. A mayores JavaScript está soportado por los navegadores más populares, es compatible con los dispositivos más modernos, y es multiplataforma. Pero sin duda su mayor particularidad es que es el único lenguaje que permite desarrollar una aplicación íntegramente en modo FullStack (cliente y servidor) (Elliott, 2014).

Sin embargo, JavaScript también presenta algunas limitaciones. Entre las más reconocidas están las limitaciones de seguridad, la razón es que en el Front-End el código es visible y requiere de otros lenguajes más específicos para realizar esas tareas. Así mismo y de nuevo relacionado con el Front-End es el hecho de que el usuario tiene capacidad de desactivar JavaScript en su navegador si así lo desea, y eso puede hacer que la página web vea afectado su funcionamiento (Elliott, 2014).

En concreto JavaScript no es un lenguaje difícil de aprender, no obstante, presenta retos cuando se trata de proyectos extensos. JavaScript no necesita compilación, por lo que a usuarios que están más familiarizados con lenguajes compilados, tipados, y con patrones de programación establecidos, JavaScript les puede resultar más complejo.

Es aquí cuando entra en escena TypeScript. Este lenguaje te permite reusar esos conceptos, ya que es un lenguaje compilado, ofrece tipado estático y programación orientada a objetos, lo que lo convierte en un fantástico candidato cuando se viene de lenguajes como C++, Java o C#. El propio compilador de TypeScript se encarga de traducir la sintaxis en puro JavaScript, por lo que garantiza que se pueda ejecutar allá donde JavaScript pueda ejecutarse: tanto en el cliente como en el servidor, y que puedas integrar librerías de JavaScript en tus proyectos a pesar de estar utilizando otro lenguaje (Bierman, Abadi, & Torgersen, 2014; Rozentals, 2017).

JavaScript tiene como origen el estándar ECMAScript. La definición de este estándar, inicialmente publicado en 1999, se realizó para homogeneizar los lenguajes de scripting con la variedad de navegadores y sistemas operativos diferentes que nacieron con el crecimiento de la Web. Existen diferentes versiones de EMACScript, sin embargo, el compilador de TypeScript tiene un parámetro que puede ser modificado para adaptarse a cada una de ellas.

Otra de las ventajas del compilador en TypeScript es que, en JavaScript al ser un lenguaje interpretado, es necesario ejecutar el código para testear que es válido, lo que lo hace más tediosa la tarea de depurar y encontrar errores en el código, como pueden ser nombres erróneos de variables o doble asignación de variables globales, ya que no te advierte de ello y puede dar lugar a errores en la ejecución. En grandes proyectos donde los desarrolladores tienen que realizar un "merge" del código, el uso de TypeScript previene que esos errores pasen inadvertidos antes de ejecutar el proyecto de nuevo. Lo mismo a la hora de escribir código, el propio IDE entiende que tipo de variable se está usando y puede sugerir un mejor autocompletado (Bierman et al., 2014; Rozentals, 2017).

En resumen, TypeScript favorece el desarrollo de aplicaciones más fácilmente escalables y mantenibles que JavaScript, pero a su vez garantiza todas las ventajas y la esencia de JavaScript. Es por todas estas razones aquí mencionadas, que se TypeScript ha sido elegido el lenguaje principal de la aplicación.

Tal y como se ha indicado anteriormente, JavaScript puede ser ejecutado tanto en el lado del cliente como en el lado del servidor. Para facilitar la programación en este lenguaje, existen entornos de trabajo preparados para facilitar el desarrollo.

Los frameworks se definen como una herramienta o conjunto estandarizado de conceptos, prácticas, y criterios sobre el que aplicaciones software pueden ser organizadas y desarrolladas. Generalmente estos entornos emplean un paradigma ya existente, o facilitan su integración, siendo el Modelo-Vista-Controlador (MVC) el más empleado en la actualidad. Esta característica hace que el entorno de trabajo infunda buenas prácticas y ayude a integrar los distintos desarrolladores en el proyecto, haciendo más sencillo el trabajo en equipo y aumentando la productividad. A mayores, los frameworks nos ofrecen un conjunto de funcionalidades ya desarrolladas que se pueden reutilizar y asumen responsabilidad sobre decisiones que, de lo contrario, debería tomar el desarrollador antes de escribir el código de la aplicación, como pueden ser la capa de persistencia para el acceso a una base de datos, gestión de usuarios, gestión de grupos y autorizaciones de acceso (Wohlgethan, 2018).

Es tarea del programador o del equipo evaluar el propósito o el alcance a la hora de elegir un framework adecuado, dado que en ocasiones puede limitar el desarrollo o suponer una carga excesiva el familiarizarse con él cuando se trata de un prototipo o de un proyecto sencillo.

Por otro lado, cabe destacar la existencia de otra herramienta muy útil en programación: las librerías. Una librería de terceros generalmente consiste en un código prescrito: clases, procedimientos, scripts, datos de configuración…, que se puede integrar como parte del proyecto para acortar tiempo de desarrollo. Esto se debe a que muchos problemas ya han sido desarrollados por expertos y han sido compartidos con la comunidad. Sin embargo, esto entraña un riesgo dado que se debe hacer una pequeña investigación sobre las fuentes previamente. Dicha investigación se puede realizar investigando el número de descargas en GitHub o Node Package Manager (NPM), en función del framework y de la librería que vayamos a usar (Wohlgethan, 2018).

Frameworks de Back-End

Desde sus inicios, JavaScript fue concebido como un lenguaje de programación para la parte del navegador. Su finalidad era la de un lenguaje de scripting que permitía modificar detalles estéticos de la página, pero con el tiempo ha crecido hasta convertirse en un complejo lenguaje con multitud de aplicaciones y librerías (Hahn, 2016).

En 2009, apareció Node.js y cambió por completo esa idea original. Node.js es el entorno de programación JavaScript, o descrito de otra forma: una plataforma construida sobre el motor de JavaScript de Chrome , del lado del servidor que permite hacer aplicaciones FullStack con un solo lenguaje de programación tanto en el cliente como en el servidor (“Node js,” 2020) .

Aunque existen otros lenguajes muy populares en el lado del servidor, como lo son PHP, Java o .Net; existen diferentes razones, por las que se ha escogido TypeScript (JavaScript) también para el lado del servidor (Hahn, 2016; Krol, Mithun, & D’mello, 2014):

1. Como bien se ha mencionado, es el único lenguaje que puede ejecutarse tanto en el lado del servidor como en el lado del cliente. Sin duda es una ventaja reducir el número de lenguajes a utilizar, y más en una aplicación pequeña.

2. Para comenzar, Node.js no utiliza hilos para la ejecución de instrucciones, sino que utiliza un sistema no bloqueante basado en eventos llamado Event Loop que permite gestionar concurrencia. Esto permite que interacción con el cliente se realice de forma más rápida e imperceptible para el usuario. Utilizar las solicitudes asíncronas permite apilar tareas y renderizar el contenido al usuario de forma más rápida mientras se espera que el proceso de Back-End realice el trabajo de más carga.

3. Es posible establecer WebSockets para tener comunicación en tiempo real entre cliente y servidor sin gran complejidad. Si bien no es el único lenguaje que las tiene, en Node.js es bastante intuitivo hacerlo por medio de Socket.io

4. Por último, mencionar NPM (Node Package Manager). Node.js cuenta con el mayor repositorio de librerías en comparación con otros lenguajes. Esto da una idea de la gran comunidad que da soporte a este lenguaje.

Al igual que para el Front-End, la variedad de frameworks sobre la que elegir es extensa. A continuación, se describirán algunos de los más populares (Krol et al., 2014):

- Express.js: Express es un framework ligero y flexible que te permite construir aplicaciones en el servidor con muy poco código. Sin duda es el que cuenta con más popularidad, y está basado en el framework web de Ruby Sinatra, al igual que el conocido framework de PHP Laravel. Una buena ventaja de Express es que soporta muchos otros paquetes y motores de plantilla como Mustache, Handlebars, EJS o muchos más.

- Koa.js : Ha sido diseñado por el mismo equipo que creó Express.js. Más ligero, rápido que su predecesor, los creadores han introducido nuevas características de JavaScript y se han liberado de middlewares y de las funciones callback, lo que hace necesario la utilización de promesas (async/await), lo que aligera la lectura.

    - Meteor.js: Meteor se define como un framework web fullstack simple y completo, cuya finalidad es facilitar al máximo el desarrollo de una aplicación robusta en la mínima fracción de tiempo. Su característica más destacable es que permite actualizar en tiempo real los cambios efectuados en las pantallas de los usuarios.

- Sails.js: Es un fantásticos framework basado en el paradigma MVC y ofrece gran compatibilidad tanto con bases de datos (MySQL, Mongo, PostgreSQL...), como con otras tecnologías de frontend. Es una elección acertada para desarrollo de aplicaciones en tiempo real como: desarrollo de chats, cuadros de mando en tiempo real o juegos multijugadores.

    - Loopback.js: Es un framework desarrollado sobre Express.js y enfocado principalmente en el desarrollo de RESTful APIs. De nuevo ofrece gran compatibilidad con una variedad de bases de datos (MySQL, Mongo, PostgreSQL...) y con servicios REST.

A mayores de los mencionados existe un mayor número de frameworks disponibles. Como se puede observar por la selección, para cada problema específico se puede encontrar un framework más adecuado, y una comunidad de apoyo para resolver esas dudas. Como cabe de esperar cada uno tiene sus ventajas y desventajas, para el desarrollo de esta aplicación en concreto se ha optado por elegir Express.js. La razón principal es que es el más popular de todos ellos y la documentación es muy extensa.

Frameworks de Front-End

El Front-End de una aplicación es la parte de software que se encarga del almacenamiento, la presentación y la actualización de los datos recibidos por parte del servidor. La principal finalidad de un framework de Front-End es la de ofrecer una base de código sobre la que construir una aplicación de forma más escalable, proponiendo una estructura de archivos común al proyecto, facilitando el diseño de componentes o la asociación con el DOM.

De este modo, se puede separar de una forma ideal la parte Front-End, haciendo posible la elaboración de una parte del programa que se comunique a través de una interfaz definida e independiente con la parte de Back-End, logrando una efectiva separación de intereses y una estructura modular fácilmente ampliable.

La capa de cliente debe únicamente encargarse de solicitar o enviar datos al servidor, mostrar mensajes de validación o error, decidir qué elementos mostrarse u ocultarse, y tomar decisiones de estilo en función de los datos.

Esta alternativa conlleva el familiarizarse con un nuevo framework de trabajo, y la curva de aprendizaje puede alargarse hasta que se logra una comprensión de la sintaxis, las herramientas y la filosofía.

Dentro de los grandes frameworks de Front-End vamos a destacar los tres más populares del mercado: Angular, React y Vue.js. Los tres son frameworks de código abierto con licencia MIT (Wohlgethan, 2018).

- Angular: Desarrollado por Google. Es el más maduro de los tres, lanzamiento en 2010. Es un marco de JavaScript basado en TypeScript y cuenta con una buena comunidad de respaldo. Se caracteriza por tener una larga curva de aprendizaje dado que es una herramienta muy completa. Y se aconseja para grandes proyectos y desarrolladores que tengan experiencia con TypeScript (Angular, 2019).

- React: Lanzado por Facebook, de nuevo cuenta con una gran comunidad y está sólidamente establecido en el mercado. Es mucho más sencillo de implementar a nivel básico y de adaptar para desarrolladores con experiencia en JavaScript, y permite la utilización de bibliotecas de terceros. El tiempo necesario para familiarizarse con la herramienta es menor, y es el más recomendado para principiantes o startups (Gackenheimer, 2015).

- Vue.js: Es el más reciente y ha sido impulsado por la comunidad de código abierto. Aunque está enfocado a JavaScript, en su versión más reciente plantea pasarse a TypeScript. Presenta excelentes resultados cuando se integra con otras bibliotecas como Bootstrap. Según las fuentes consultadas es más fácil de aprender que los anteriores y es fácil de integrar. Sin embargo, el código resultante puede resultar más tedioso de depurar y mantener (bin Uzayr et al., 2019).

La alternativa opuesta es crear la interfaz partiendo de HTML, CSS, y JavaScript/TypeScript, pero a medida que el proyecto va avanzando, el número de archivos aumentará con él y se dificultará su mantenimiento, el código se volverá repetitivo y la separación de intereses más difusa.

Sin embargo, como alternativa a los grandes frameworks de Front-End que mencionábamos anteriormente, se pueden encontrar otras soluciones, o incluso optar por desarrollar una aplicación monolítica con vistas renderizadas en el lado del servidor. A continuación, se detallan algunas de las opciones:

- Frameworks o bibliotecas de CSS:

    - Bootstrap: Es quizás el framework de CSS más popular. Es de código abierto y multiplataforma que ofrece plantillas de HTML/CSS y clases reutilizables, plugins y widgets de JavaScript y que tiene un enfoque Mobile First (basándose en este concepto se crea primero la versión optimizada para dispositivos móviles y luego se amplía a tablet y ordenador, siguiendo la tendencia de los usuarios navegan cada vez más desde sus smartphones o tabletas (Hesterberg, 2011).
    - Semantic UI: De nuevo se trata de un proyecto de código abierto completamente gratuito. Este framework ofrece elementos prediseñados y permite diseñar interfaces con un diseño responsive. A diferencia de Bootstrap utiliza lenguaje semántico, por lo que es más intuitivo (Khriyenko, 2015).

- Motor de plantillas: Este tipo de herramientas favorece la separación del código por responsabilidades, manteniendo el diseño HTML de nuestro proyecto separado del resto del código de JavaScript.

    - Mustache está considerado como la base de los motores de plantillas de JavaScript y, pero que además proporciona plantillas aceptables para la mayoría de los lenguajes de programación mayoritarios (PHP, JAVA, JS y .NET) (M. Lerner, 2011).
    - Handlebars: Es un sistema de plantillas de JavaScript que permite generar bloques de código HTML partiendo de objetos en formato JSON. En el caso de Handlebars, es una extensión del lenguaje de plantillas Mustache, que ofrece a mayores rutas anidadas y el concepto de "helpers", para registrar nuestras propias funciones y que permite que se puedan referenciar desde cualquier template (Manricks, 2013).
    - EJS: De nuevo es un motor de plantillas que permite trabajar con código de JavaScript incrustado para generar código HTML (Azat Mardan, 2014).

Como se ha visto, los frameworks de Front-End son una poderosa herramienta para desarrollar interfaces de usuario complejas y crear arquitecturas independientes y modulares en la parte de cliente. Dichos frameworks cuentan con grandes comunidades y una amplia documentación. Sin embargo, requieren una familiarización con el entorno y pueden ser excesivos para proyectos pequeños o prototipos, como es el caso de este proyecto.

Es por ello por lo que, si bien es cierto que se ha escogido Bootstrap y Handlebars para el desarrollo de esta aplicación, se ha descartado la utilización de cualquier framework de gran popularidad como React, Angular, o Vue entre otros. La decisión ha venido determinada tras realizar una comparativa entre las distintas soluciones y tener en cuenta el objetivo principal del proyecto.

Base de datos

Toda aplicación web hoy en día, por muy interactiva que sea, no tiene sentido si no es capaz de almacenar datos de forma permanente. Para ello es necesario integrar una base de datos dentro de la aplicación. Dicha base de datos es la encargada de guardar la información que se almacena de forma temporal en el servidor, dado que en el momento en el que este se

apaga, la información se desvanece. La elección de la base de datos dependerá primeramente de la naturaleza de los datos de la aplicación. Existen dos grandes familias: bases de datos relacionales (SQL) y bases de datos no relacionales (NoSQL).

Principalmente las bases de datos relacionales están dirigidas a aplicaciones orientadas a gestión con clientes, o plataformas de comercio electrónico. Cada una de las tablas contiene elementos de una misma entidad, donde cada una de las filas representa un elemento distinto (un usuario, o un producto), y cada columna un atributo que describe esa entidad (edad, o unidades). A su vez, es posible añadir elementos compartidos, de forma que representen las relaciones entre las tablas y faciliten las consultas complejas.

Por otro lado, si su información no está tan fuertemente estructurada, NoSQL puede ser la mejor opción. Ejemplos son aplicaciones para almacenar datos de sensores, contenido de artículos, o publicaciones en redes sociales que no se adaptan completamente a una tabla. En ese caso, los datos se almacenarán en una base de datos sin ningún esquema predefinido, sin embargo, los datos pueden no ser almacenados en una tabla, sino que hay diferentes alternativas: bases de datos orientadas a grafos, documentales, basadas en pares clave-valor, u orientadas a objetos entre otras.

A mayores de la estructura de los datos, existen más diferencias notables entre ambos grupos. Una de las más importantes es la escalabilidad. En las bases de datos relacionales para aumentar la capacidad es necesario un crecimiento vertical (aumentar la CPU, RAM o SSD de un solo servidor. Con las no relacionales, la escalabilidad es horizontal, por lo que solo requiere añadir más servidores para aumentar capacidad.

Para el proyecto en concreto, se requiere diseñar una aplicación de gestión de entrenamientos grupales e individuales dentro de un equipo de futbol. Dado que los datos están muy estructurados y se puede modelar fácilmente mediante tablas relacionadas, se ha decidido modelar el entorno mediante una base de datos SQL. En la mayor parte de la bibliografía se han encontrado ejemplos de Node.js con bases de datos NoSQL, sin embargo, eso no implica que no se puedan utilizar bases de datos SQL.

Como resumen se han detallado algunas de las alternativas que se han tenido en cuenta a la hora de la elección:

- MySQL: La base de datos más popular de Código abierto.
- PostgreSQL: Base de datos de Código abierto y diseñada para el manejo de gran volumen de datos.

- Microsoft SQL server: Sistema de gestión de bases de datos desarrollado por Microsoft.
- Oracle: Sistema de gestión de bases de datos desarrollado por Oracle Corporation.
- MariaDB: Base de datos de código abierto desarrollada sobre MySQL.

También cabe mencionar que en los últimos años el incremento de plataformas que ofrecen bases de datos en la nube ha permitido que se convierta en una solución muy demandada dado que no requiere de infraestructura física para almacenar la base de datos y puede eliminar costes de mantenimiento de esos equipos. A continuación, se detallan las principales opciones de bases de datos relacionales disponibles actualmente en el mercado:

- AWS:
  - RDS: estándar base de datos SQL
  - Aurora: opción basada en el rendimiento y la escalabilidad.
- Microsoft Azure:
  - Azure SQL Database: base de datos totalmente administrada con inteligencia integrada.
  - Bases de datos específicas para MySQL, PostgreSQL y MariaDB.
- Plataforma Google Cloud:
  - Cloud SQL: disponible para MySQL y PostgreSQL.
  - Cloud Spanner: combina elementos de SQL y NoSQL y se define como una base de datos relacional horizontalmente escalable y sin incoherencias.

En el caso concreto de este proyecto se ha optado por utilizar una solución tradicional, y de entre las alternativas anteriormente mencionadas se ha elegido MySQL. La principal razón a parte de su popularidad ha sido que existen herramientas y documentación para su integración con Node.js.

Patrón de diseño

Entre la multitud de opciones, el patrón MVC se ha convertido en un estándar para el desarrollo de aplicaciones web. El MVC permite la separación del código dependiendo su responsabilidad, este reparto se realiza en tres grupos o capas: el modelo, la vista, y el

controlador. Por ello y por la adaptabilidad a las aplicaciones diseñadas con Node.js, esta ha sido la arquitectura elegida para el diseño de nuestra aplicación (Doglio, 2015; A. Mardan, 2014).

MVC

❖ *Modelo*

En esta capa es donde reside la representación de los datos del dominio. Aquí se definirán:

- La lógica de negocio de la aplicación. Esta marca el comportamiento del software, así como las reglas sobre lo que se puede hacer y lo que no con los datos: obtener datos, insertarlos, modificarlos, borrarlos etc. En ella se definen las entidades que nos van a servir para almacenar la información del sistema.

- Los mecanismos de persistencia para gestionar el almacenamiento y la recuperación de los datos. En este punto es frecuente encontrar herramientas de mapeo objeto-relacional (ORM). Permiten trabajar con abstracción sobre la base de datos, de forma que los objectos se convierten a los tipos utilizados por las bases de datos relacionales, evitando tener que usar directamente sentencias SQL y gestionar manualmente la persistencia de los datos.

❖ *Vista*

La vista, como su nombre indica, se encarga de presentar los datos al usuario, de generar la interfaz de nuestra aplicación. Para ello tiene capacidad de acceder a los datos del modelo, sin embargo, esto no se realiza mediante un acceso directo, sino que las vista solicita los datos al modelo a través del controlador y de la misma forma el modelo pone a disposición de la vista los datos. Este código nos permitirá renderizar los estados de nuestra aplicación en HTML.

❖ *Controlador*

El controlador es el intermediario entre la vista y el modelo. Es el encargado de escuchar los eventos desencadenados por la vista, y de ejecutar las operaciones adecuadas a esos eventos, pero en ningún momento manipula los datos directamente, ni muestra ningún tipo de salida. Por lo cual, cuando el usuario realiza una operación, lo hace a través del controlador, el cual actúa sobre el modelo y luego notifica a la vista para que actualice los datos en pantalla.

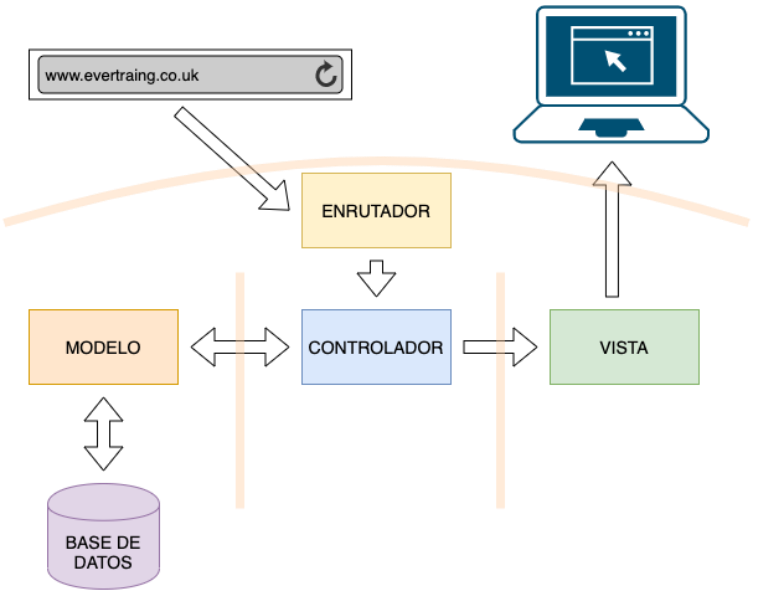


Figura 1. Esquema del patrón MVC (Azat Mardan, 2018)

2.4. Estudio del arte en el desarrollo de las Aplicaciones Web Progresivas

Una PWA (Progressive Web App) es accesible cuando se tiene acceso a Internet mediante una URL, y se ejecuta en el navegador, lo que hace posible que funcione en diferentes dispositivos y sistemas operativos sin la necesidad de realizar una aplicación multiplataforma de forma específica con el esfuerzo que ello requiere. Pero a diferencia de una aplicación web al uso, también se puede ejecutar sin conexión. Sin duda es una metodología en pleno desarrollo, pero sus características resultan altamente atractivas en futuros desarrollos. En este caso, las PWA nos puede ofrecer la solución al problema planteado (Majchrzak, Biørn-Hansen, & Grønli, 2018; Somoza, 2019).

Las aplicaciones web progresivas (PWA) prometen combinar la facilidad de desarrollo de las tecnologías web con la versatilidad de las aplicaciones nativas. A continuación, se detallarán algunas de las ventajas de las PWA frente a otros métodos de diseño:

- Multiplataforma: Aunque las alternativas para el desarrollo de aplicaciones han avanzado significativamente a lo largo de los últimos años, la posibilidad de desarrollar aplicaciones multiplataforma sigue siendo uno de los grandes retos dentro de la industria. Se busca que las aplicaciones sean compatibles con dispositivos Android y iOS, que funcionen sin problema en varios dispositivos hardware y que sean compatibles con una gran cantidad de versiones de plataforma, por no mencionar la aparición de nuevos dispositivos más allá de los smartphones y las tabletas. Esta ardua tarea engloba también el soporte de

futuras versiones de todos los dispositivos, por lo que el problema no se limita tan sólo al desarrollo inicial (Majchrzak et al., 2018). Hasta ahora las PWA podrían no haberse considerado con un enfoque multiplataforma, pero es cierto que de forma inherente buscan admitir múltiples plataformas desde una base de código (Majchrzak et al., 2018; Somoza, 2019).

- Sensación de aplicación nativa: Con la introducción de las PWA, los sitios web pueden en mayor medida que antes, dar la sensación al usuario de que está accediendo a una de las aplicaciones nativas instaladas en el dispositivo móvil (smartphone o Tablet). Sin embargo, para acceder a una PWA, el usuario no es necesario que descargue la aplicación y espere a que esta esté instalada en su dispositivo, sino que el acceso se realizará por medio de la URL de la página. Dicha aplicación puede, por medio de los navegadores compatibles, agregarse a la pantalla de inicio del dispositivo y así acceder de forma más rápida o incluso usarse sin conexión la próxima vez. La aplicación web progresiva se ejecutará entonces dentro de un navegador simplificado, que ocultará varios aspectos de la interfaz para dar esa apariencia de aplicación como la barra de herramientas y de direcciones o los menús (Majchrzak et al., 2018).

- Ausencia de descarga: Relacionado con el punto anterior está el hecho de que las PWA no requieren de instalación. Si bien para una aplicación normal se requiere que el usuario abra la tienda de aplicaciones y que la descargue para su posterior instalación, para las PWA, tal y como se ha enunciado no es así. En el caso de una aplicación web nativa el hecho de tener que descargar la aplicación completa para su uso puede suponer un tiempo extra en comparación con las PWA. Lo mismo sucede en caso de actualización, en el caso de las PWA no se requiere una nueva descarga, si no que la página siempre mostrará los últimos cambios realizados (Majchrzak et al., 2018; Somoza, 2019).

- Conexión offline: En el momento en el que se agrega a la pantalla de inicio, todos los archivos estáticos necesarios para ejecutar la PWA se almacenan en el teléfono del usuario: HTML, CSS, JavaScript, imágenes y fuentes para el estilo web. A su vez, todos los datos dinámicos se pueden almacenar en caché para uso sin conectividad, y actualizados cuando el teléfono se encuentre en un área con acceso a internet (Majchrzak et al., 2018).

- Adaptabilidad: Otra de las características de las PWA es que el abanico de funcionalidades que ofrece una aplicación de este estilo, a diferencia de las aplicaciones web, es que se adaptan al marco en el que se ejecutan. Así en el caso en el que los usuarios, dispositivos o navegadores no ofrezcan disponibilidad con las prestaciones, estas podrán utilizarse de forma reducida, pero de cierta forma seguirán estando disponibles. Un ejemplo podría ser la autorización del acceso a la cámara para subir alguna foto a la aplicación. En caso de que el dispositivo no disponga o que tenga limitado el acceso, todavía podrá ser posible subir las fotos desde otra localización sin ser específicamente la de hardware (Majchrzak et al., 2018; Somoza, 2019).

Así como se han detallado sus ventajas, también es necesario detallar algunos inconvenientes o puntos débiles de esta nueva tecnología.

A través de la búsqueda de bibliografía, de ha identificado una ausencia de información sobre el desarrollo de PWA en el ámbito de iOS, debido a la falta de soporte. Esta falta de apoyo de Apple podría suponer un obstáculo importante para la propagación de las PWA dentro del perfil de aplicaciones multiplataforma.

Esto implica que, a pesar de que se están realizando trabajos y las características básicas ya están garantizadas en iOS, ciertas tecnologías necesarias para que las PWA funcionen con todas sus características no están todavía disponibles. Entre estas destacan: reconocimiento de voz, notificaciones push, almacenamiento de cache limitado, o sincronización en segundo plano.

Esta misma problemática de falta de soporte se manifiesta en el resto de los navegadores, a excepción de Chrome: Safari, Edge, Firefox, Opera…, y en el tipo de dispositivo que se utiliza para acceder a la PWA: móvil o escritorio; siendo Chrome en dispositivos móviles la mejor combinación.

A pesar de que Google es su principal impulsor y su situación es prometedora, la falta de uniformidad entre los distintos navegadores hace entrever que puede deberse a una guerra entre los distintos gigantes tecnológicos (Majchrzak et al., 2018). Esta postura sin duda hace creer que difícilmente a corto plazo las PWA sean una solución universal multiplataforma, pero si una alternativa más conforme esta tecnología va madurando. A su vez, la falta de documentación dentro de la comunidad académica y el rápido desarrollo de esta tecnología denota un gran potencial de investigación.

Recomendaciones de diseño

En la bibliografía consultada se han identificado varios requisitos o recomendaciones que se deben de tener en cuenta a la hora de diseñar una PWA:

- Interfaz de usuario diseñada a partir de elementos comunes y adaptables a diferentes diseños.
- Apariencia de acuerdo con las guías de diseño responsive en cuanto a las pautas de diseño y usabilidad.
- Posibilidad de proporcionar un flujo de control y diversas formas de interacción en una aplicación.
- Posibilidad de definir tipos de datos y proporcionar acceso a las operaciones básicas CRUD (creación, recuperación, actualización y eliminación).
- Validación de datos de entrada y asignación de modelos de datos en formularios.
- Utilización de eventos y cambios de estado.
- Utilización de funciones de hardware comunes: acceso a la cámara, geolocalización, uso de bluetooth…

La tecnología de las PWA

Desde un punto de vista técnico, las Aplicaciones Web Progresivas cuentan con las siguientes características (Biørn-Hansen, Majchrzak, & Grønli, 2017; Electrons, 2018; Majchrzak et al., 2018):

- Manifiesto de aplicación: Documento de metadatos basado en JSON que permite al desarrollador definir ciertas características de las aplicaciones, como son: la ruta de la imagen del logotipo, el nombre de la aplicación o la pantalla de bienvenida.

- App Shell: Interfaz de usuario gráfica mínima (GUI) lógica necesaria para renderizar y visualizar la aplicación sin contenido dinámico dependiente de conexión, o dicho de otra forma, los archivos HTML, CSS y JavaScript mínimos, que son requeridos para representar la interfaz de usuario. Tanto la interfaz como la lógica deben incluir información sobre el enrutamiento, la lógica de navegación y los elementos propios de la interfaz. La idea es separar la aplicación entre contenido y funcionalidad, y tratar ambos de forma independiente. Bien realizado, puede dar la sensación de que el tiempo de carga es menor.

- Service Worker: Script basado en JavaScript responsable de las principales funciones asociadas con las aplicaciones web progresivas. Su funcionalidad es la de gestionar la conectividad de red, la lógica de almacenamiento en caché y ejecutar las sincronizaciones en segundo plano. El Service Worker puede decidir si debe obtenerse nuevo contenido de un servicio web, o si el contenido en caché sigue estando actualizado, evitando de esa forma las descargas innecesarias. Además, el Service Worker solo trabajará con protocolo HTTPS para garantizar la seguridad de la aplicación.

A la hora de auditar la web, Google ha puesto a disposición Lighthouse. Esta herramienta automatizada de código abierto ha sido diseñada para mejorar la calidad de las aplicaciones web. De forma rápida se puede comprobar si se cumple con los requisitos y premisas de las PWA, aparte de medir auditar otras características como la accesibilidad u otras prácticas (Majchrzak et al., 2018).

3. OBJETIVOS Y MOTIVACIONES

El objetivo principal de este proyecto es el de proponer el diseño de una herramienta para la gestión de la preparación física en un equipo de futbol, que aparte de satisfacer en contenido las necesidades profesionales, aporte una solución de trabajo tanto online como offline.

A lo largo del anterior apartado, y más concretamente tras el estudio de mercado, se han ido identificando cuáles serían las necesidades funcionales que debería cumplir nuestra aplicación para que fuese rentable con respecto a sus competidores. Entre ellas destacan las siguientes:

- Diseño de tareas individuales y grupales. Herramienta de gestión de contenidos didácticos del deporte (técnica, táctica, biomecánica, …) para la mejora tanto a nivel individual como colectivo. Su finalidad es la de ayudar en la preparación tanto del jugador como del equipo en la efectividad de las diferentes situaciones de juego. Debe facilitar la programación de contenidos cualitativos de forma que contribuya a la construcción de una progresión pedagógica, que será evaluada de forma observacional.

- Diseño de las sesiones utilizando como unidad de medida el microciclo. El microciclo debe representar un ciclo semanal y debe de estar presente tanto en la vista del diseño de las sesiones individuales, como del equipo. Esta unidad de medida agiliza la programación de contenidos en los deportes de equipo.

- Bajo coste. Herramienta enfocada a un grupo reducido de usuarios. Debe centrarse en el/los preparadores físicos, mejorando su calidad profesional, y dejando a un lado la integración con el resto de los técnicos o departamentos. Así mismo debe facilitar la migración de datos no confidenciales a lo largo de la carrera profesional del preparador físico.

- Diseño de ejercicios y sesiones propias. Permitir al usuario crear sus propios ejercicios a través de grabaciones con el fin de desarrollar un banco de ejercicios. De esta forma el software facilitará el diseño de sesiones de trabajo tanto en campo como gimnasio con un alto grado de especificidad, que pueden ser guardadas como plantilla de trabajo.

- Capacidad de diseño de sesiones offline. De forma que ofrezca un funcionamiento mínimo cuando hay ausencia de conexión a internet. Este

funcionamiento mínimo debe permitir acceder a sesiones anteriores y diseñar otras nuevas, que posteriormente se guarden en la base de datos cuando se vuelva a tener acceso a internet.

Para llevar a cabo todos estos requisitos, se ha decidido diseñar una aplicación Node.js con Express. El lenguaje principal elegido ha sido TypeScript, y a la hora del diseño del cliente, se ha optado por combinar Bootstrap y Handlebars, descartando la utilización de cualquier gran framework de front-end. Para el almacenamiento de datos, se ha elegido integrar una base de datos relacional, más en concreto MySQL.

Por último, cabe mencionar, que para cumplir el objetivo de ofrecer un funcionamiento mínimo cuando no se tiene conexión a internet, y a su vez facilitar el acceso desde distintos dispositivos, se ha optado por una nueva tecnología denominada PWA (Aplicación Web Progresiva), que se presenta como la evolución de las aplicaciones web. Estas nuevas aplicaciones presentan características que solo las aplicaciones nativas ofrecían hasta ahora, como son: el funcionamiento sin conexión, uso de notificaciones, o el acceso al hardware de los dispositivos desde los cuales te conectas.

En los siguientes apartados se detallará el plan de trabajo, y una muestra de su arquitectura.

4. Desarrollo específico de la contribución

4.1. Metodología

La metodología que se utiliza en el proyecto es una convivencia de metodología ágil para el desarrollo de las fases, en este caso Scrum, con una metodología tradicional para la gestión del proyecto, Proceso Unificado de Desarrollo (RUP) (Bashir & Qureshi, 2012).

Por un lado, para el análisis del proyecto, se hace uso de la metodología más tradicional, RUP. Esta metodología tiene especial importancia en las etapas previas para realizar las tareas de análisis de requisitos y diseño del sistema. Como resultado de la aplicación de estos procesos se generarán diversa documentación que asistirá a los desarrolladores en las decisiones de diseño y de implementación a nivel de equipo. A continuación, se detalla los artefactos, o herramientas, generada en este proyecto para cada una de las etapas:

- Inicio: En esta fase se define y se acota el alcance del proyecto, se identifican los riesgos y se detallan los requisitos funcionales y no funcionales.

- Elaboración: En esta fase se seleccionan los casos de uso que permiten definir la arquitectura base del sistema. Como resultado se obtienen los diagramas de caso de uso.

- Desarrollo: El propósito de esta fase es la de refinar el análisis y el diseño, para ello en este proyecto se genera diferente documentación que facilitará la posterior construcción del código: el diagrama de clases, el modelo Entidad-Relación y por último los diagramas de secuencia.

De forma complementaria, la metodología ágil Scrum se centra en gestionar los procesos del proyecto. De esta forma se establece un marco de trabajo para la gestión y el desarrollo del software iterativo e incremental. Las ventajas que ofrece esta metodología es que permiten adaptarse a los cambios en los requerimientos o prioridades de forma rápida, y a su vez incluye a los desarrolladores en la toma de decisiones sobre la definición y organización de las tareas.

El modelo de integración continúa propuesto consta de Sprints o ciclos funcionales cada 3 semanas. De esa forma se pretende realizar un trabajo iterativo en el que se irán realizando pruebas a la par que se describe el resto de las funcionalidades del sistema, para mitigar cualquier incidente e ir generando la documentación. Cabe mencionar que dichos Sprints estarán a su vez divididos en otras subetapas: *Inicio*, *Priorización de tareas*, y *Revisión y cierre*

de Sprint. De esta forma, desde la gestión del proyecto se pretende asegurar que en todo momento la estimación de las tareas en futuros ciclos se adecúe a la realidad y que el orden de ejecución se realice de acuerdo con las prioridades que van surgiendo de la implementación del sistema.

En el siguiente diagrama de Gantt se describen las actividades o tareas necesarias que se han llevado a cabo la ejecución de este proyecto. Cada uno de los bloques del eje temporal representa un período de una semana. Los dos primeros apartados de Investigación (FT1) y Documentación técnica (FT2) se corresponden con el capítulo del Estado del arte, mientras que los siguientes ya coinciden con apartados específicos del proyecto.

TFM	Planificación temporal
	CONCEPCIÓN
FT1	INVESTIGACIÓN
1.1	*Investigación sobre la planificación y programación de la preparación física de un equipo de futbol.*
1.2	*Estado del arte en el sector de las aplicaciones web en el ámbito de la preparación física en el fútbol*
1.3	*Estudio del arte de las tecnologías de desarrollo web*
FT2	DOCUMENTACIÓN TÉCNICA
2.1	*Estudio del arte en el desarrollo de las Aplicaciones Web Progresivas*
FT3	ANÁLISIS
3.1	*Definición de los requisitos funcionales de la aplicación*
	ELABORACIÓN
FT4	ANÁLISIS
4.1	*Diseño de la arquitectura*
4.2	*Diseño de los casos de uso del sistema*
	DESARROLLO - CONSTRUCCIÓN
FT5	DISEÑO
5.1	*Diseño de diagramas de secuencia*
5.2	*Diseño de la base de datos : diagrama E-R*
5.3	*Diseño de diagramas de clase*
5.4	*Diseño de la interfaz de usuario*
5.5	*Diseño y desarrollo de la lógica de negocio*
FT6	IMPLEMENTACIÓN
6.1	*Implementación de las fases de diseño*
	TRANSICIÓN
FT7	INTEGRACIÓN Y VALIDACIÓN
7.1	*Realizaciónde pruebas de casos de uso y tests*
7.2	*Realizaciónde pruebas de casos de uso y test en un entorno real*
FT8	CONCLUSIONES Y LÍNEAS FUTURAS
8.1	*Extracción de conclusiones*
8.2	*Descripción de posibles mejoras o futuras líneas de desarrollo*

Figura 2. Diagrama de Gantt del proyecto

Ciclo de vida

El ciclo de vida del RUP divide el proceso en 4 etapas, las cuales, si es necesario, pueden repetirse de forma iterativa hasta que se da por completada la fase. En la figura adjunta se puede ver un ejemplo de la evolución de las fases en un proyecto se software conforme a esta metodología.

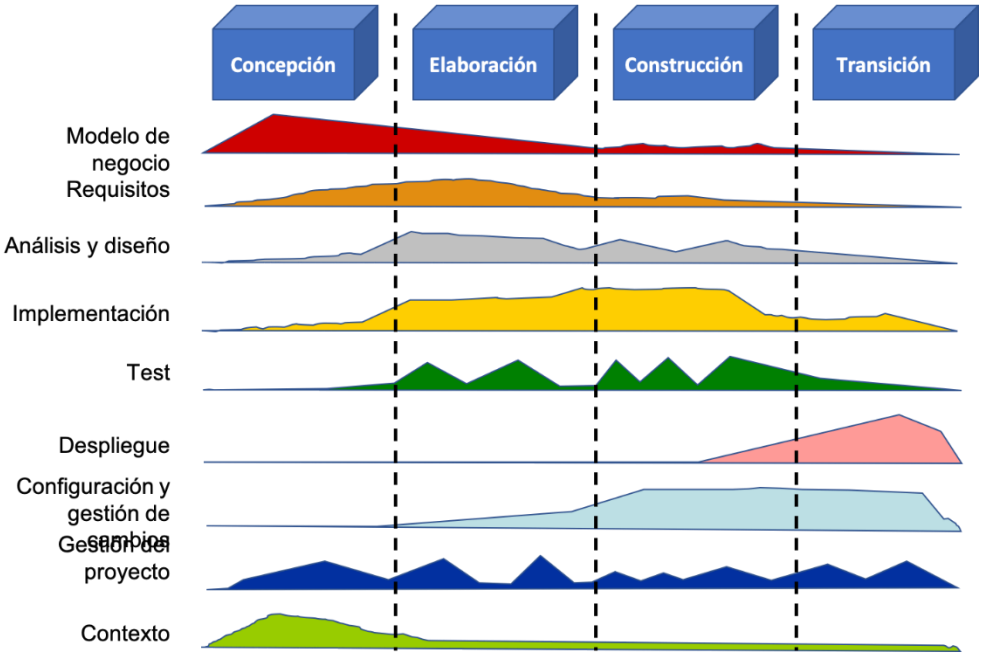


Figura 3. Ciclo de vida de un proyecto software según la metodología RUP (Kruchten, 2000)

Dado que en este proyecto esta metodología convive con Scrum, a continuación, se detallarán las principales actividades llevadas a cabo en cada una de las fases:

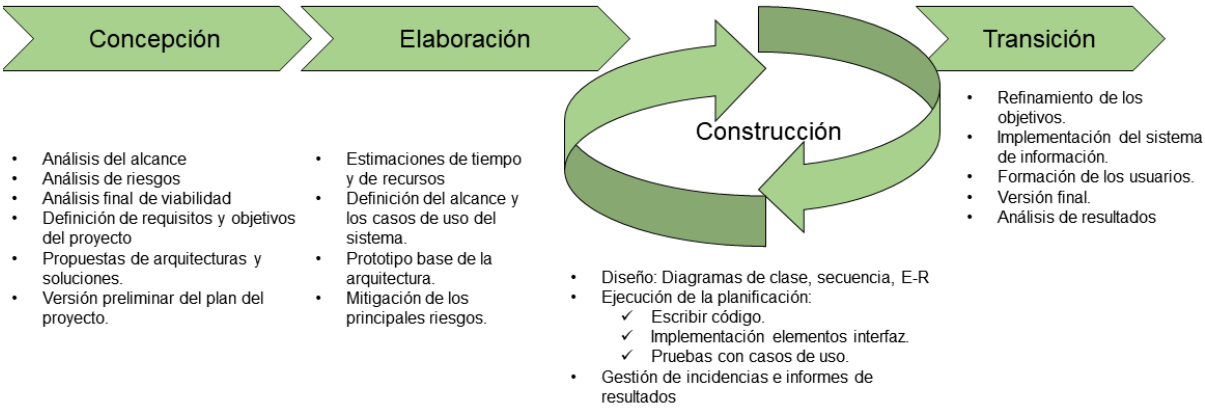


Figura 4. Metodología de trabajo propuesta

Concepción

En el esquema se puede observar que dentro de la etapa de Concepción del sistema la finalidad principal ha sido definir la viabilidad del proyecto, así como el propósito de la web. El resultado de este trabajo se ha visto reflejado en el apartado del Estado del Arte.

Por un lado, se ha identificado una necesidad y se ha analizado el sector profesional al cual iría dirigida la aplicación. Posteriormente se ha realizado un estudio de mercado para poder justificar la viabilidad de la web.

Parte de esa información recogida ha servido para comprender las necesidades de los preparadores físicos, y ha sido utilizada para planificar la estructura y el diseño de la web. Del estudio realizado se ha concluido que existen potentes soluciones de gran coste que cubren un gran abanico de funcionalidades, pero que no ofrecen esa especialización que se ha detallado en el apartado 3 (Objetivos y motivaciones). En este contexto, se ha determinado el diseño de una solución de menor costo que se centre exclusivamente en la tarea del preparador físico y agilice sus necesidades diarias desde un punto de vista novedoso.

Otro de los resultados de este apartado ha sido la elaboración del documento de requisitos funcionales y no funcionales, apartado 4.2.

Se pueden resumir como principales tareas técnicas de esta fase las de modelar el ámbito de negocio y definir los requisitos, así como la definición de las primeras etapas de análisis, diseño e implementación.

Elaboración

Tiene como finalidad principal completar el análisis obtenido en la etapa anterior. Para ello es imprescindible determinar los casos de uso de la aplicación, y de esa forma definir la arquitectura del sistema.

Construcción

Como resultado se ha obtenido un resumen detallado de las tareas que debe llevar a cabo el software, así como el contenido que debe almacenarse en la base de datos, y los informes que deben generarse como resultado del flujo de trabajo. En los apartados siguientes se procederá a explicar con más detalle todos estos aspectos técnicos: diagramas de secuencia, diagrama Entidad-Relación y diagramas de clase.

A la hora de justificar este proyecto, y partiendo de los requisitos definidos en la etapa de Concepción, dentro de los apartados 5 (Implementación) y 6 (Integración y Validación), se ha decidido utilizar una solución simplificada que ofrezca un conjunto mínimo de los requisitos de

forma que se cumpla con la finalidad principal (diseño offline de tareas) y que pueda ser demostrada.

Es en esta etapa donde entra en escena Scrum. Desde una visión global se realiza una previa planificación global en función de los objetivos y de la fecha de entrega al inicio de la etapa. Sin embargo, esa planificación debe de ser revisada cada 3 semanas teniendo en cuenta los recursos de los que se dispone en ese momento, de las horas disponibles o de la complejidad de las tareas para refinar el proceso de forma iterativa. A mayores el código desarrollado será testeado de forma iterativa con cada implementación que se realiza.

Transición

Por último, se propone la etapa de Transición, en la que se termina por definir el sistema adaptándolo a resultados en pruebas de campo o simulacros reales, y se imparte la formación necesaria a los usuarios. En nuestro proyecto esta etapa se corresponde con la sección 6 (Integración y Validación). En el caso de que se realizase la implementación completa de la aplicación, este período debe alargarse hasta que se hayan alcanzado los objetivos acordados y el cliente se muestre satisfecho para poder cerrar el proyecto.

Otra de las tareas importantes de esta etapa es la de realizar un análisis y una interpretación crítica de los resultados, para poder obtener una conclusión y establecer futuras líneas de trabajo (7 Conclusiones y Futuras líneas de trabajo).

4.2. Requisitos funcionales y no funcionales

El proceso de definición de requisitos es un paso imprescindible previo a las etapas de diseño e implementación de la aplicación. En este apartado se recogen los resultados obtenidos a partir de las investigaciones y consultas realizadas a potenciales usuarios de la aplicación.

Para poder obtener una visión clara de estos requisitos, se han creado dos apartados: requisitos funcionales y requisitos no funcionales.

Requisitos funcionales

El primer grupo viene a describir la interacción del sistema con el usuario y con cualquier otro sistema externo con el que se quiere interactuar. Estos requisitos son los que garantizan la funcionalidad de la aplicación.

❖ *Gestión de usuarios*

En la siguiente tabla se recogen los diferentes requisitos relacionados con el acceso de los usuarios al sistema.

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF1	El sistema de acceso a la aplicación estará controlado por un sistema de usuario-contraseña	**1**
RF2	En el sistema existirán los siguientes roles: • Mánager (Manager) • Miembro del equipo (Team member)	**4**
RF3	El usuario de tipo Mánager podrá tener control sobre la creación de las temporadas del equipo y sobre la configuración de los microciclos de toda esa temporada. También serán los únicos que podrán gestionar los datos personales referentes a los jugadores.	**1**
RF4	El usuario de tipo Miembro del equipo podrá introducir nuevos ejercicios y nuevas plantillas de sesiones. A su vez tendrá capacidad de generar las sesiones de los microciclos tanto individuales como grupales y podrán también introducir los datos resultantes de la sesión.	**1**

❖ *Configuración de la temporada*

En este apartado se definen los requerimientos necesarios para configurar la aplicación para gestionar una temporada de competición.

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF5	El sistema debe permitir que el usuario con rol Mánager pueda definir la temporada de un equipo. Como resultado deben crearse automáticamente cada uno de los microciclos individuales y grupales. Como valores de entrada para la definición de los microciclos es necesario que previamente se especifique: Nombre del equipo. Cada uno de los jugadores que forman parte del equipo: en caso de no existir previamente, con añadir el nombre, y número o foto	**1**

	es suficiente, ya que posteriormente se podrá completar la identificación. Las fechas de inicio y final de temporada. Los microciclos generados tendrán inicio en lunes.	

❖ *Gestión de ejercicios*

En este apartado se encuentran detallados los requisitos asociados a las especificaciones de los ejercicios.

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF6	El sistema deberá permitir que todos los usuarios puedan incluir nuevos ejercicios en la base de datos para su posterior utilización en la creación de sesiones.	**1**
RF7	Deben diferenciarse dos tipos de ejercicios: tareas de campo (drills) o ejercicios (exercises)	**1**
RF8	Para definir ambos tipos de ejercicio es necesario especificar los siguientes campos: • Nombre del ejercicio • Número de jugadores implicados • Descripción • Palabras clave • Lugar de ejecución • Imagen • Contenido • Tipo	**1**
RF9	El sistema debe permitir al usuario modificar, eliminar y duplicar el ejercicio.	**1**
RF10	Para consultar los ejercicios disponibles, el sistema proporcionará un sistema de filtrado que permitirá al usuario introducir criterios de búsqueda.	**3**
RF11	El filtrado de los ejercicios debe permitir los siguientes criterios de búsqueda: • Ordenar por nombre • Ordenar por número de jugadores • Ordenar por contenido • Filtrar por palabras clave	**3**

	• Filtrar por nombre • Filtrar por número de jugadores • Filtrar por tipo • Filtrar por contenido • Filtrar por localización	
RF12	Si el dispositivo lo permite la aplicación debe de tener acceso a la cámara para poder subir de forma automática la foto del ejercicio	**3**

❖ *Gestión de plantillas de sesión*

Para poder definir las plantillas de sesión es necesario que se lleven a cabo los siguientes requisitos.

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF13	El sistema deberá permitir que el usuario incluya nuevas plantillas con ejercicios dentro de la base de datos	**1**

RF14	Cada una de las plantillas de sesión debe contener los siguientes campos:
	• Ejercicios incluidos en la sesión. Cada uno de ellos con sus respectivos valores de:
	  ○ series
	  ○ repeticiones
	  ○ carga
	• Nombre
	• Descripción
	• Palabras clave
	• Contenido
	• Tipo
		1

RF15	El sistema debe permitir al usuario modificar, eliminar y duplicar la plantilla.	**1**
RF16	Para consultar las sesiones disponibles, el sistema proporcionará un sistema de filtrado que permitirá al usuario introducir criterios de búsqueda.	**3**

RF17	El filtrado de las plantillas de sesión debe permitir los siguientes criterios de búsqueda: • Ordenar por nombre • Ordenar por número de ejercicios • Ordenar por contenido • Filtrar por palabras clave • Filtrar por nombre • Filtrar por tipo • Filtrar por contenido	**3**

❖ *Gestión de jugadores*

Para poder definir las plantillas de sesión es necesario que se lleven a cabo los siguientes requisitos.

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF18	Los jugadores incluidos en la base de datos estarán disponibles cuando se crea una nueva temporada. (RF5)	**1**
RF19	El usuario con rol Mánager puede añadir, eliminar o suspender cualquier jugador de la plantilla.	**1**
RF20	Dentro del perfil del jugador deben poder accederse a los siguientes apartados: • Microciclos individuales del jugador • Microciclos grupales del equipo • Informe médico del jugador • Informe lesional del jugador • Datos personales	**1**
RF21	El apartado de microciclos individuales permitirá visualizar en una tabla todos los microciclos de una temporada, y cada uno de los microciclos debe tener visibles los campos siguientes: • Número de microciclo • Fecha de inicio • Descripción • Observaciones médicas	**1**

	Por cada microciclo en la lista debe existir la opción de editarlo, copiar su contenido o pegar contenido de otro microciclo.	
RF22	El usuario con rol Mánager debe tener acceso a eliminar o crear nuevos microciclos individuales.	**1**
RF23	El apartado de microciclos grupales permitirá visualizar en una tabla todos los microciclos de una temporada, y cada uno de los microciclos debe tener visibles los campos siguientes: • Número de microciclo • Fecha de inicio • Descripción • Enlace para acceder al microciclo No debe existir ninguna opción para modificar, copiar, pegar o eliminar los microciclos grupales desde la sección del jugador	**1**
RF24	El apartado del Informe médico del jugador debe mostrar en una tabla todas las observaciones médicas que se han ido recogiendo en los microciclos individuales (RF21). Se deben visualizar los siguientes campos: • Número de microciclo • Fecha de inicio • Descripción • Enlace para acceder al microciclo individual No debe existir ninguna opción para modificar los datos ahí visualizados.	**2**
RF25	El apartado del Informe lesional del jugador debe registrar todas las lesiones que el jugador ha tenido desde el inicio del registro, incluyendo temporadas anteriores. Se deben visualizar los siguientes campos: • Número de lesión • Fecha de la lesión • Días transcurridos • Lesión • Tipología • Localización • Tejido afectado	**2**

	• Mecanismo lesional • Actividad • Severidad • Estatus: terminada, en proceso • Enlace para acceder al informe de la lesión Debe existir la opción de añadir o eliminar las lesiones de la lista.	
RF26	Cada uno de los jugadores debe contener los siguientes campos dentro de su perfil personal: • Nombre • Apellido • Apodo • Número • Posición • Pie de preferencia • Equipo • Notas • Imagen	**1**
RF27	El sistema deberá permitir que solo el usuario con rol Mánager pueda modificar los datos personales del jugador.	**1**

❖ *Gestión de microciclos individuales desde el jugador*

Cuando el usuario accede a un microciclo individual la aplicación debe garantizar los siguientes requisitos.

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF28	En la vista de un microciclo deben visualizarse los 7 días de la semana que componen el microciclo.	**1**
RF29	Cada día del microciclo debe mostrar los ejercicios de la sesión o de las sesiones en caso de ser varias.	**1**
RF30	Cada una de las sesiones debe permitir al usuario copiar o pegar contenido previamente copiado, tanto ejercicios como sesiones completas.	**2**
RF31	Debe poderse navegar entre los diferentes microciclos individuales.	**1**

RF32	Cada una de las sesiones visualizadas debe de tener un acceso directo para poder visualizar y editar la sesión a pantalla completa sin la vista del microciclo	**1**
RF33	Cada uno de los ejercicios de la sesión debe mostrar la imagen del ejercicio con un resumen de la carga, series y repeticiones.	**1**
RF34	Cada uno de los ejercicios de la sesión debe permitir al usuario eliminar, o copiar para poder añadir a la sesión de otro día desde esta vista.	**1**

- *Gestión de microciclos grupales desde el jugador*

Cuando el usuario accede a un microciclo grupal desde el jugador el sistema debe comportarse de la siguiente forma.

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF35	La vista del microciclo debe ser análoga a la del microciclo individual. (RF28, RF29, RF31, RF32, RF33)	**1**
RF36	No debe poderse eliminar, copiar o pegar el contenido desde esta vista, tan solo el acceso directo.	**1**

- *Gestión de las lesiones*

Si el usuario accede a una de las lesiones o si desea añadir una nueva a la base de datos, el sistema debe garantizar que.

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF37	Se puedan introducir los valores que describen una lesión (RF25) y un apartado a mayores en el que poder escribir notas o comentarios acerca de la lesión.	**2**
RF38	Debe poder almacenarse diferente dispositivo multimedia: videos, imágenes.	**2**

- *Gestión del equipo*

Debe de existir un acceso a la configuración del equipo desde el cual:

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF39	Dentro del perfil del equipo debe poder accederse a los siguientes apartados:	**1**

	• Microciclos grupales del equipo • Sesiones grupales	
RF40	El apartado de microciclos grupales permitirá visualizar en una tabla todos los microciclos de una temporada, y cada uno de los microciclos debe tener visibles los campos siguientes: • Número de microciclo • Fecha de inicio • Descripción Por cada microciclo en la lista debe existir la opción de editarlo.	**1**
RF41	El usuario con rol Mánager debe poder eliminar o crear nuevos microciclos grupales.	**1**
RF42	El apartado de sesiones grupales permitirá visualizar en una tabla todas las sesiones grupales de una temporada que han sido introducidas en los microciclos. Deben de visualizar los siguientes campos: • Número de sesión • Fecha de sesión • Descripción Por cada sesión en la lista debe existir la opción de editarla y de acceder al microciclo al cual pertenecen.	**1**
RF43	En ningún momento se debe poder eliminar las sesiones desde esta vista.	**1**

❖ *Interacción con otros sistemas*

El sistema debe permitir exportar datos e importarlos desde otros sistemas:

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF44	La aplicación permitirá importar los datos de sesión desde una hoja de Excel dado que la mayoría de los dispositivos GPS trabajan con ese formato.	**3**
RF45	Todas las hojas de sesión, microciclos, tablas de lesiones y observaciones médicas deben poder ser exportadas en formato pdf.	**2**

Requisitos no funcionales

Este otro grupo se especifican aquellos requisitos que no están de forma directa relacionados con las funciones específicas del sistema, sino de propiedades como la eficiencia, la seguridad de los datos, o la usabilidad.

- *Eficiencia*

Los requisitos de eficiencia del sistema son los siguientes:

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF46	La aplicación debe ser construida de forma estructural, de tal forma que se puedan añadir nuevas funcionalidades y requerimientos puedan ser incorporados de la forma más sencilla y rápida posible sin alterar el funcionamiento inicial de la aplicación.	**1**
RF47	La aplicación debe ofrecer una experiencia offline básica que permita crear sesiones de ejercicios que se sincronizarán de forma automática cuando se restaure la conexión.	**1**

- *Usabilidad*

Los requisitos de eficiencia del sistema son los siguientes:

CÓDIGO	DESCRIPCIÓN	PRIORIDAD
RF48	La interfaz de la aplicación deberá ser fácil de usar, agradable e intuitiva. El usuario debe de utilizar el menor número de interacciones posibles para planificar un microciclo o elaborar una sesión de entrenamiento.	**1**
RF49	El sistema debe presentar mensajes de error que permitan identificar el problema y ponerse en contacto con el servicio de técnico.	**1**
RF50	El sistema debe de estar documentado por medio de manuales de usuario o video tutoriales.	**1**
RF51	La interfaz de usuario de la aplicación debe de ser en inglés.	**1**

Como resumen a este apartado, se puede concluir que, de forma inicial la existencia de diferentes roles en el sistema no tiene gran prioridad ante otros requisitos, 4, por lo tanto, es aceptable que las tareas que en un futuro serán específicas para el usuario tipo Manager en un principio estén disponibles para todos los usuarios. El filtrado de ejercicios y plantillas de sesión tiene la siguiente prioridad más baja de entre todos los requisitos, 3, lo mismo que el acceso a la cámara para poder subir imágenes desde los dispositivos móviles o la funcionalidad de importar datos desde una hoja Excel. A mayores se puede observar que la gestión de lesiones y observaciones médicas tampoco tiene una alta prioridad en comparación con las tareas de gestión y creación de sesiones dentro de los microciclos, y puede ser incorporado con posterioridad, su prioridad es 2. De esta misma forma, con prioridad 2 se identifica la tarea de copiado y pegado de sesiones dentro de un microciclo y la exportación de las sesiones a un archivo pdf.

4.3. Arquitectura de la aplicación

A la hora de desarrollar una aplicación, la arquitectura es tan importante como la calidad del código escrito. Una buena organización facilita que se puedan añadir nuevas funcionalidades al código sin incrementar su complejidad, por ello es necesario un patrón de diseño que nos ayude a estructurar el código desde una fase inicial.

Un patrón de diseño no es más que una solución general reutilizable que puede ser aplicada a problemas comunes que se encuentran en el diseño software. No es más que un modelo o guía que expresa como organizar y estructurar los componentes de un sistema software.

Patrón de diseño utilizado

La decisión de llevar a cabo un diseño basado en el patrón Modelo-Vista-Controlador ha sido tomada basándonos en los requisitos funcionales y no funcionales del sistema, pero en especial teniendo en cuenta en el requisito RF46. Este patrón de diseño permite que el código quede definido en módulos claramente definidos entre sí. De esta forma, en caso de necesitar añadir nuevas funcionalidades o requerimientos es más sencillo realizar los cambios en el sistema. De la misma manera en caso de tener que modificar la parte de la vista para extender a más dispositivos la aplicación, no es necesario tener que realizar ningún cambio en el resto de la aplicación.

Más allá de las tres capas del MVC, en la capa de Modelo se ha decidido realizar una diferenciación entre las entidades que definen los datos y por otro lado la capa de acceso a la base de datos. En esta última se ha optado por utilizar una herramienta de mapeo objeto-relacional que permite conseguir mayor nivel de abstracción sobre el acceso a los datos sin

necesidad de realizar consultas que dependan del tipo de base datos. De esta forma, si se opta por migrar el sistema a otro sistema gestor de bases de datos, tan solo es necesario adaptar esa capa de abstracción. Con esta configuración se busca ofrecer una aplicación más robusta y organizada debido al encapsulamiento y a su vez facilita las tareas de mantenimiento y soporte del producto.

Por último, tal y como se indica en el requisito RF47, la aplicación debe garantizar una experiencia offline al usuario, para ello se creará un Service Worker para gestionar el contenido sin conexión. En la siguiente figura se puede ver un esquema de la arquitectura del sistema.

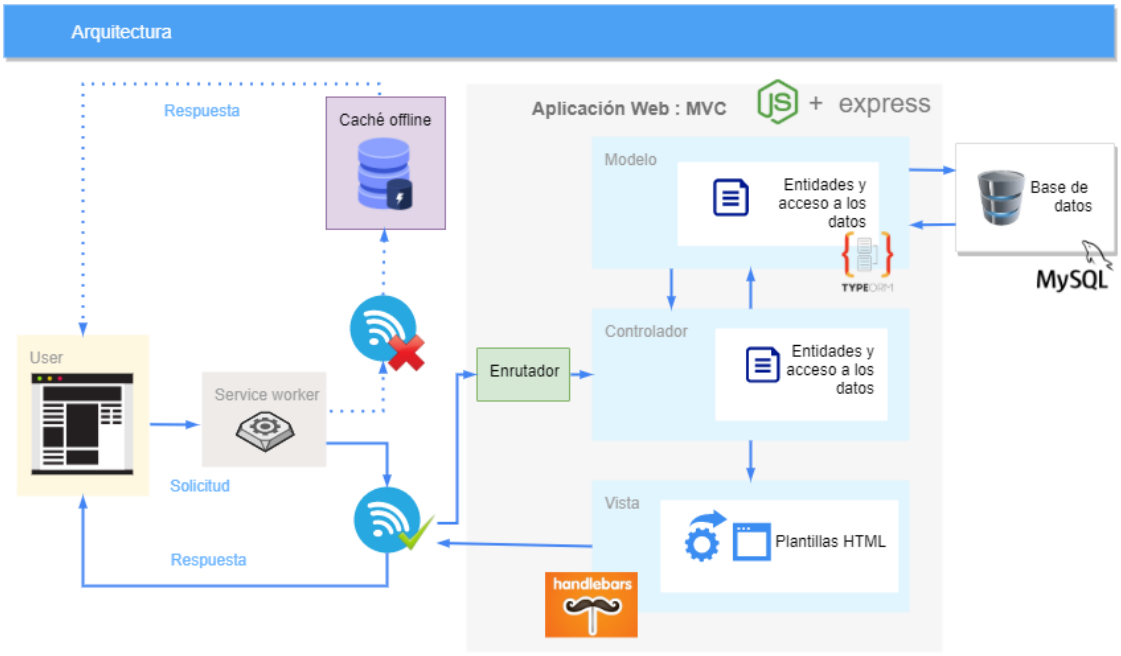


Figura 5. Esquema de la arquitectura del sistema

Cabe mencionar también que para el desarrollo del proyecto se ha decidido incluir una herramienta de gestión de versiones. La razón principal es la de gestionar la trazabilidad del código. El uso de este tipo de herramientas permite realizar un seguimiento de las modificaciones del código, mejorando la eficacia en el desarrollo de nuevas funcionalidades, RF46, y en el soporte técnico del mismo ante errores del sistema, RF49.

4.4. Análisis

Casos de Uso

A continuación, se detallarán os casos de uso de la aplicación. En este apartado se han descrito las posibles interacciones entre el usuario del sistema y la aplicación a desarrollar

con el fin de recoger todos los escenarios y respuestas surgidas de dicha interacción y de los requisitos detallados en el anterior apartado. La especificación de casos de uso facilita la comprensión del sistema y refleja la intencionalidad del usuario, complementándose con los requisitos anteriormente descritos.

Para describir los casos de uso se ha utilizado una notación basada en el Lenguaje Unificado de Modelado (UML) y a mayores en el anexo 8.1 se ha realizado un cuadro más detallado por cada una de las posibles necesidades.

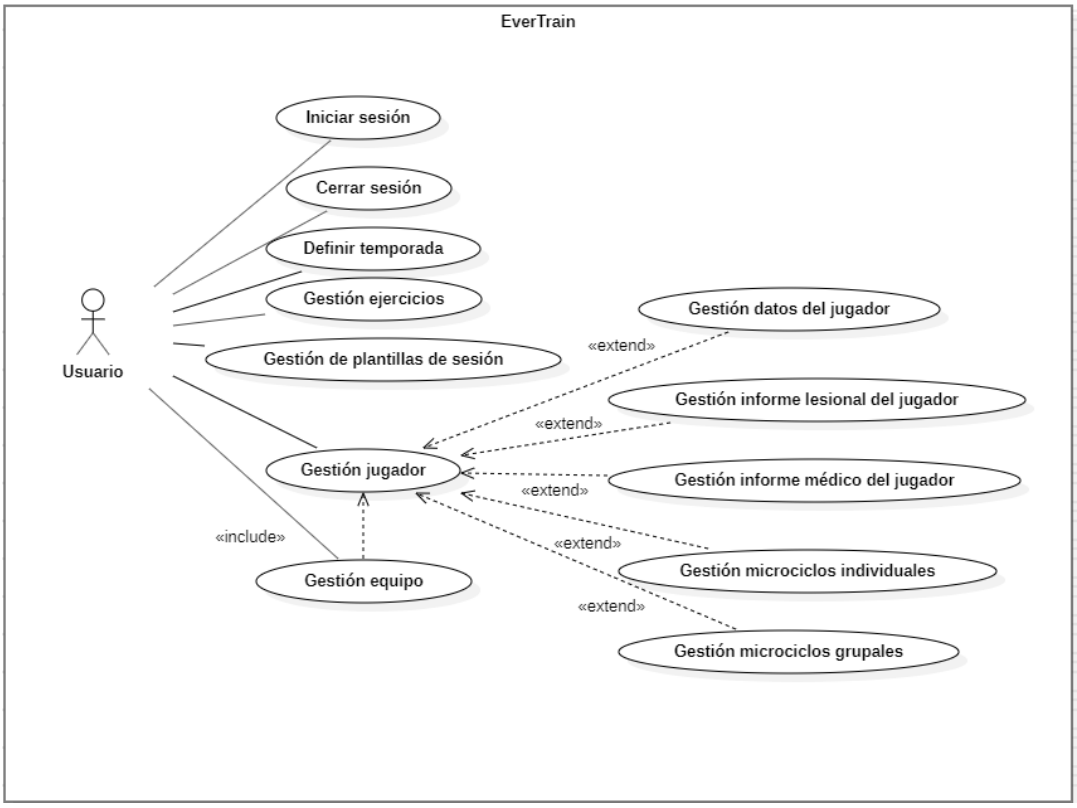


Figura 6. Diagrama de caso de uno general

La Figura 6 representa el diagrama general del sistema. En él se observan un único actor: un usuario. El escenario descrito representa el esquema completo y recoge todos los casos de uso generales que deben llevarse a cabo dentro del sistema como son: la entrada y salida del sistema, la gestión de la temporada, del equipo y de los jugadores.

Los primeros casos de uso definen las acciones de iniciar y abandonar sesión. A continuación, se detallan las tres actividades principales necesarias para configurar la aplicación: Definir temporada, gestión de ejercicios y gestión de plantillas de sesión. La primera de ellas debe realizarse una vez por temporada y por equipo. Las otras dos, sin embargo, pueden realizarse cuando se estime oportuno y sin ningún criterio especifico, más que el de haber iniciado sesión. Su finalidad es la de abastecer la base de datos con ejercicios y sesiones que podrán ser utilizadas posteriormente en el diseño de las sesiones de los jugadores.

Por último, se describen los escenarios que indican cómo el usuario debería interactuar con el sistema para editar el perfil de cada uno de los jugadores. Esta tarea debe repetirse cada vez que se quiera diseñar una nueva tarea o ejercicio dentro de un microciclo, introducir datos lesionales, o actualizar datos personales del jugador.

4.5. Diseño

Diagrama Entidad-Relación

A continuación, se ha detallado el diagrama ER (Entidad-Relación) de la base de datos que necesitamos para desarrollar nuestra aplicación. Este diagrama servirá de guía a la hora de diseñar las entidades (modelos) del sistema.

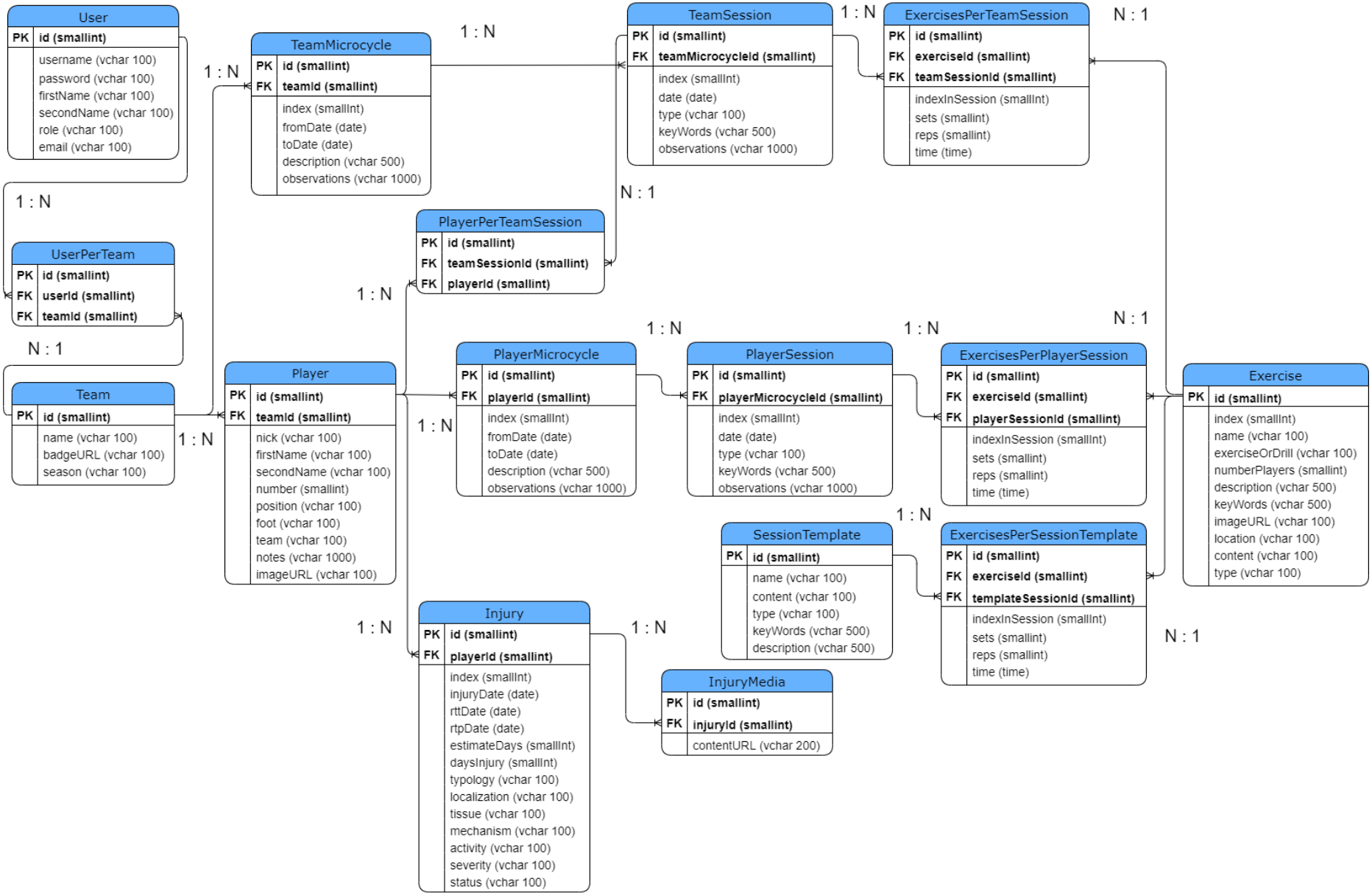


Figura 7. Diagrama Entidad-Relación de la base de datos

Diagramas de secuencia

Los diagramas de secuencia nos permiten diseñar el comportamiento del software. Partiendo de los casos de uso se puede modelar lo que sucede internamente en el software cuando el usuario interacciona con nuestra aplicación.

Una de las ventajas de elaborar diagramas de secuencia para cada caso de uso es que nos permite detectar comportamientos comunes o paralelos entre varios casos de uso de forma temprana. Es por ello por lo que en este apartado se han limitado los casos de uso a aquellos procesos más descriptivos que sirven de ejemplo y de quía para el diseño, sin detallar aquellos con patrón similar. Ejemplos de este paralelismo en el código puede ser: el diseño de un microciclo individual o grupal, filtrar ejercicios o sesiones; u operaciones CRUD (Create Retrieve, Update y Delete en inglés) que no son tan relevantes dentro del proceso: guardar/acceder/modificar/eliminar datos del perfil de jugador, ver historial médico, o almacenar datos lesionales.

Otro de los detalles a mencionar es que entre las especificaciones se indica que la interfaz debe ser realizada en inglés, RF51. Para dar consistencia al software, se utilizará el mismo idioma para el código, por esta razón las funciones descritas en los diagramas contienen descripciones de las tareas en inglés.

El primero de los diagramas de secuencia detallado recoge una de las tareas más importantes del software, la configuración de la temporada RF5. Dicho proceso debe ser llevado a cabo una sola vez por temporada, al inicio. El proceso se ha sintetizado para mostrar los pasos más destacados: creación de microciclos grupales entre las fechas indicadas, vinculación de los jugadores que pertenecen al equipo esa temporada y creación de los microciclos individuales para cada uno de los jugadores.

Este esquema nos sirve también para identificar los pasos que han de seguirse en caso de querer añadir un jugador si la temporada ya ha sido creada, y es que tan sólo sería necesario realizar los pasos contenidos dentro del segundo bucle externo (0, nPlayers).

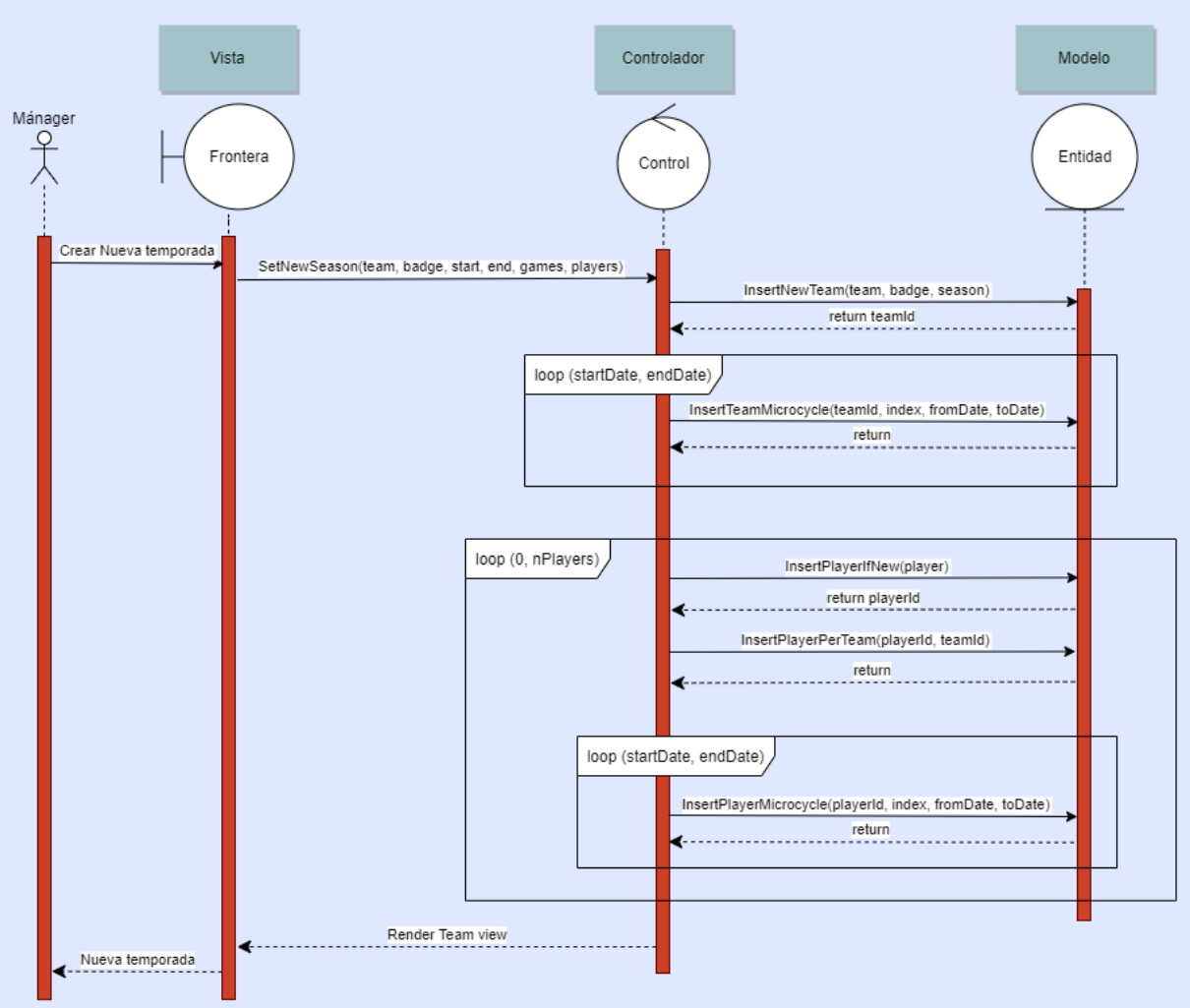


Figura 8. Diagrama de secuencia: configuración de la temporada

Otros de los procesos que se han sido descritos de forma completa son los relacionados con la creación, eliminación, modificación o filtrado de ejercicios. La importancia de detallar cada una de las operaciones es que a pesar de ser operaciones CRUD, forman parte de las operaciones más relevantes de la aplicación. Como se ha indicado anteriormente, dicho patrón puede ser copiado para el resto de las acciones CRUD del sistema.

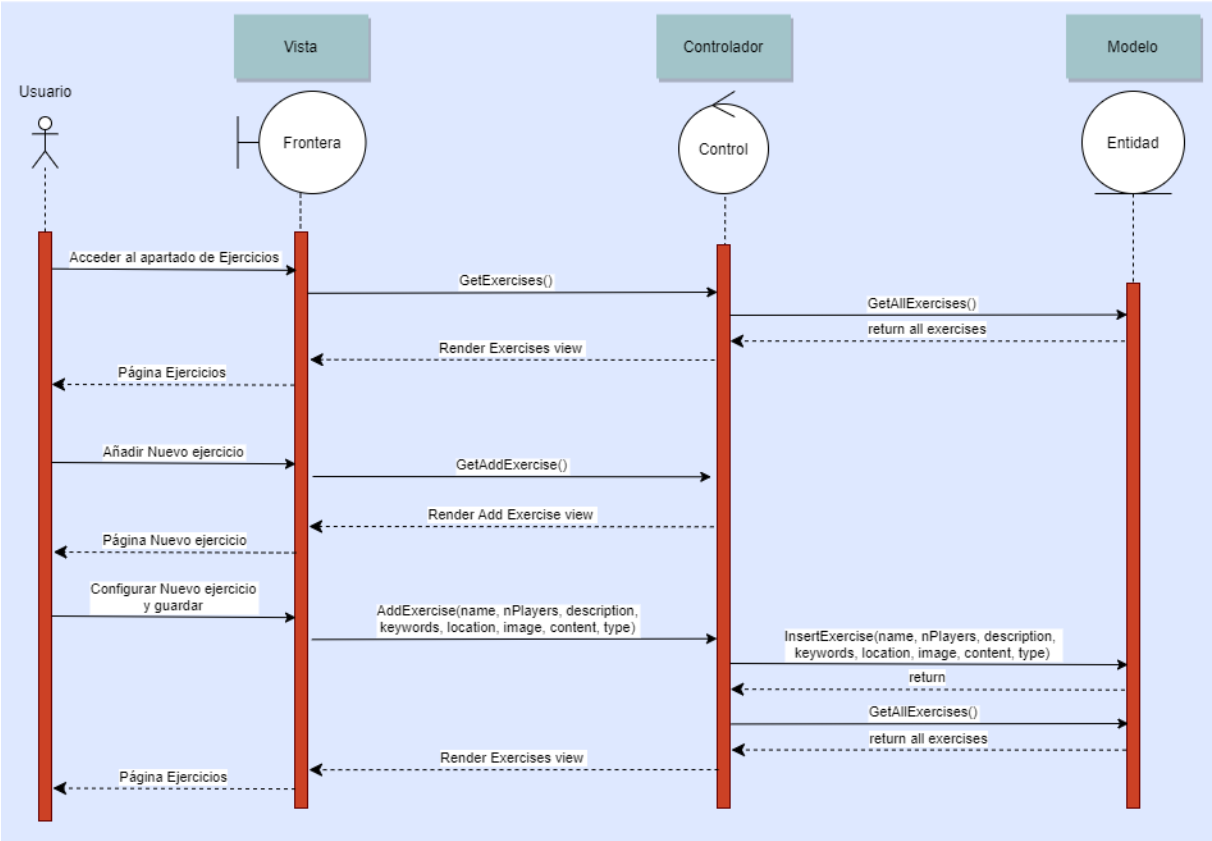


Figura 9. Diagrama secuencia: añadir nuevo ejercicio

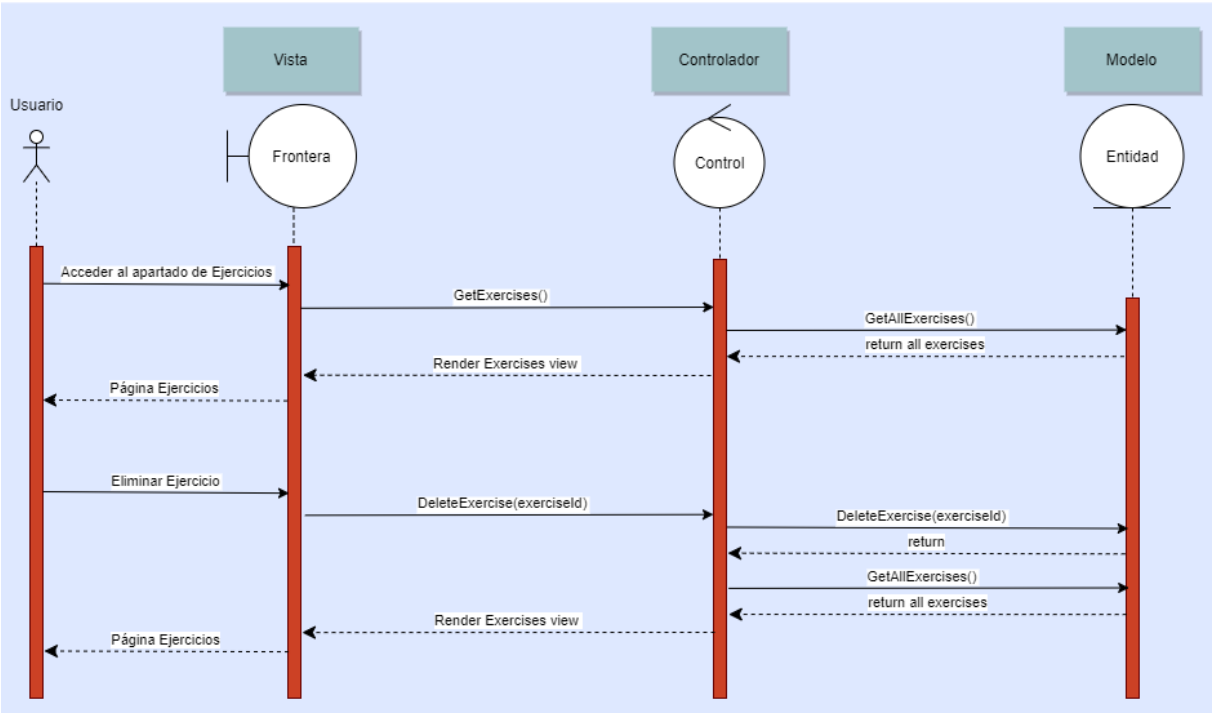


Figura 10. Diagrama de secuencia: eliminar ejercicio

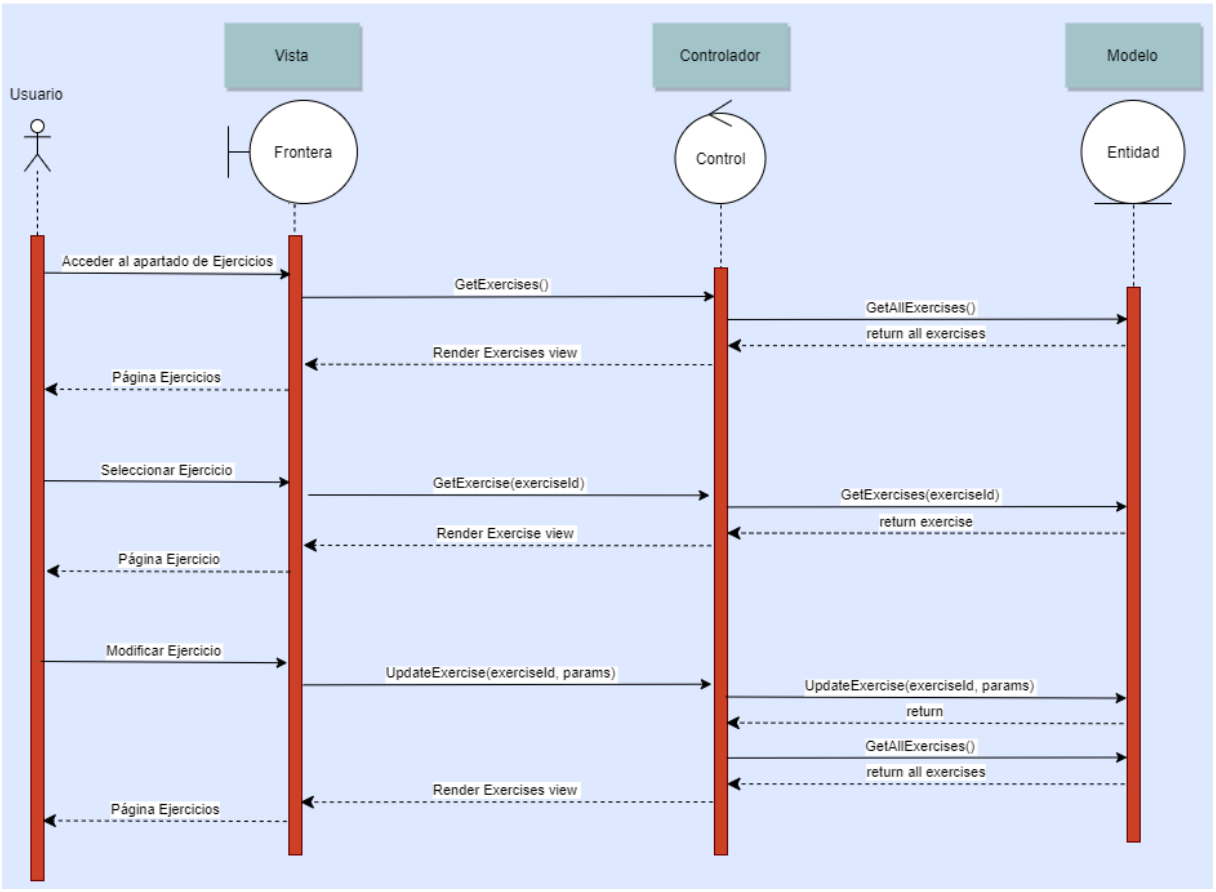


Figura 11. Diagrama de secuencia: modificar ejercicio

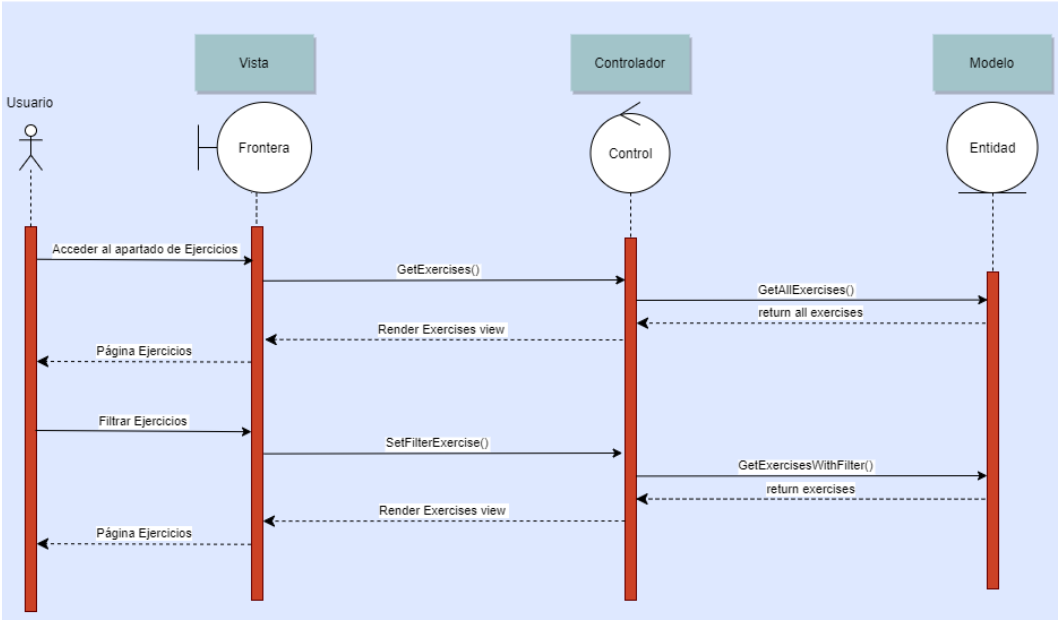


Figura 12. Diagrama de secuencia: filtrar ejercicios

Si bien para la gestión de las plantillas de sesión el proceso es similar, sobre todo el filtrado y la eliminación, es necesario mencionar el proceso de creación porque entrañan alguna

particularidad que nos ayudará a entender cómo diseñar las plantillas de sesión y cómo relacionarlas a los ejercicios.

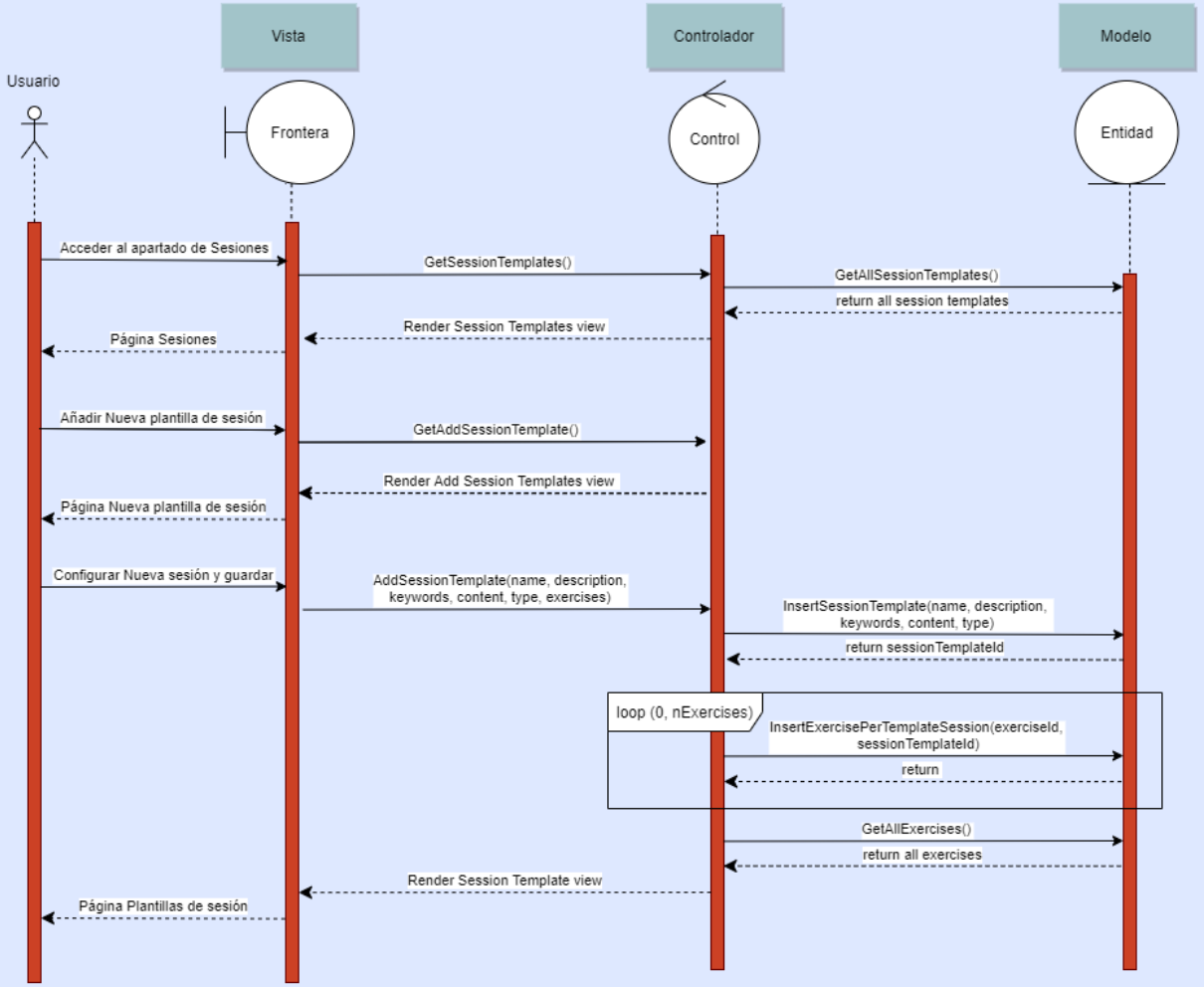


Figura 13. Diagrama de secuencia: añadir nueva plantilla de sesión

Una vez la temporada ha sido creada y disponemos de datos en el sistema, uno de los procesos importantes en el día a día es la creación de microciclos individuales. Para detallar cada una de las partes importantes en el diseño de los microciclos, se ha elegido representar el diagrama de secuencia del acceso al microciclo individual. En dicho esquema se pueden apreciar las partes que interaccionan en el proceso, ofreciendo una idea de cómo serían las demás funcionalidades relacionadas con la entidad del microciclo individual.

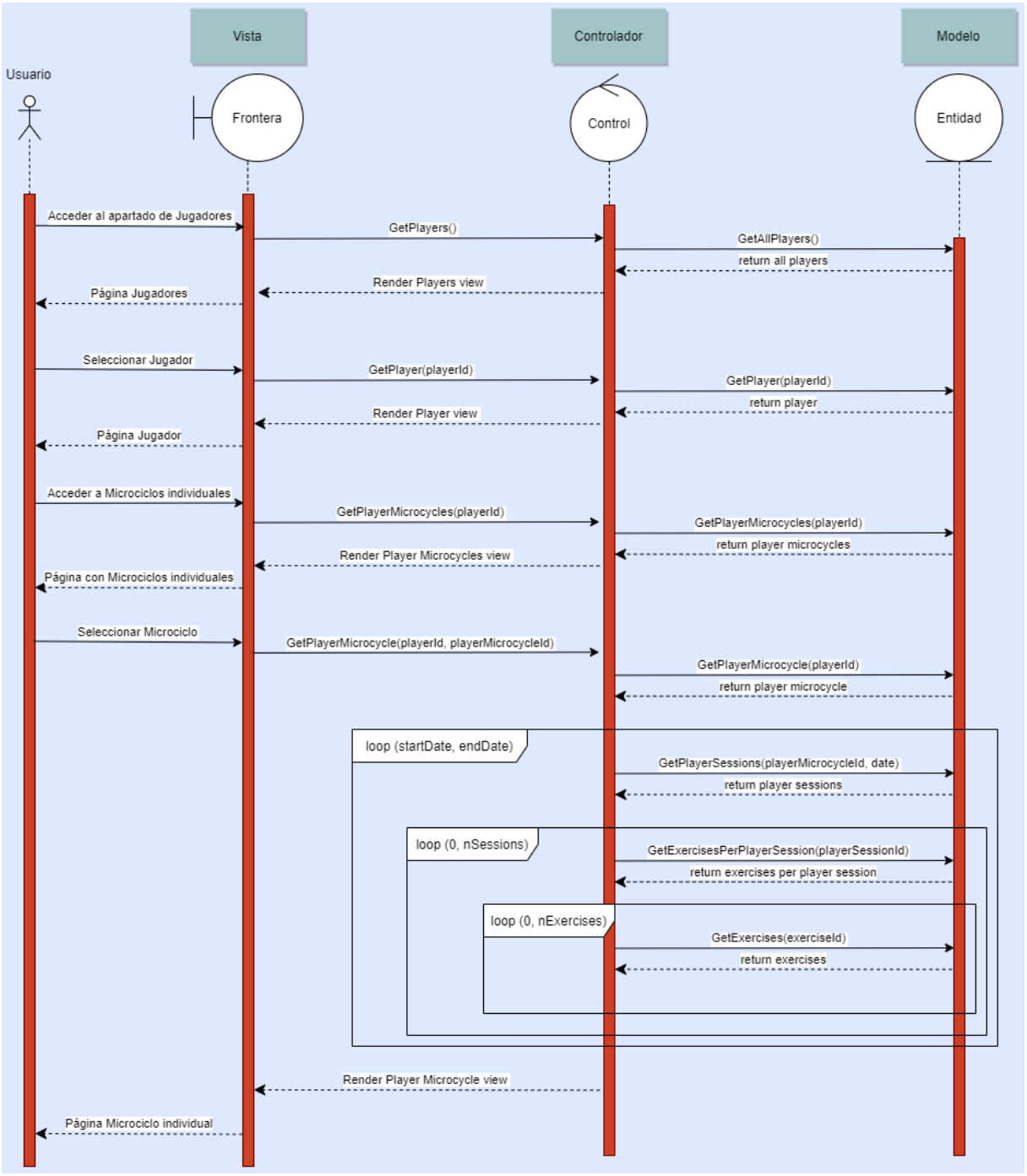


Figura 14. Acceso a microciclo individual

Por último, queda detallar otro proceso muy importante, y es el de añadir una plantilla de sesión a un microciclo, tanto individual como grupal. Una vez definida la sesión por medio de la plantilla, el usuario deberá ajustar los valores de las repeticiones, sesiones, carga de los ejercicios, así como incluir los jugadores participantes en caso de ser un microciclo grupal (operaciones no incluidas en el diagrama de secuencia). En este ejemplo el diagrama de secuencia recoge las acciones para el microciclo grupal.

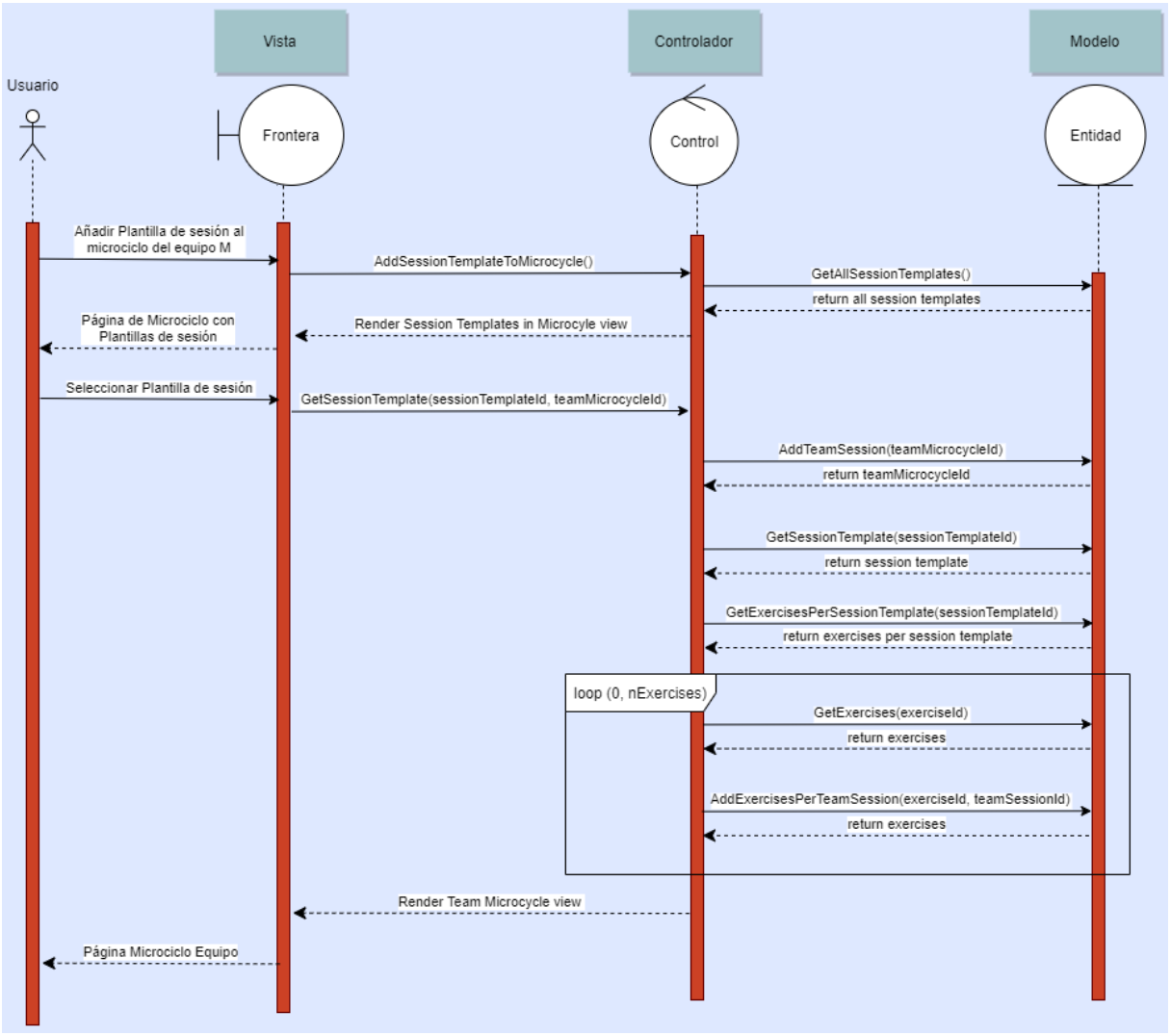


Figura 15. Diagrama de secuencia: añadir plantilla de sesión a microciclo grupal

Diagrama de clases

En la siguiente figura se muestra el diagrama de Clases, que resume las actividades identificadas en los diagramas de secuencia, Anexo 8.3.

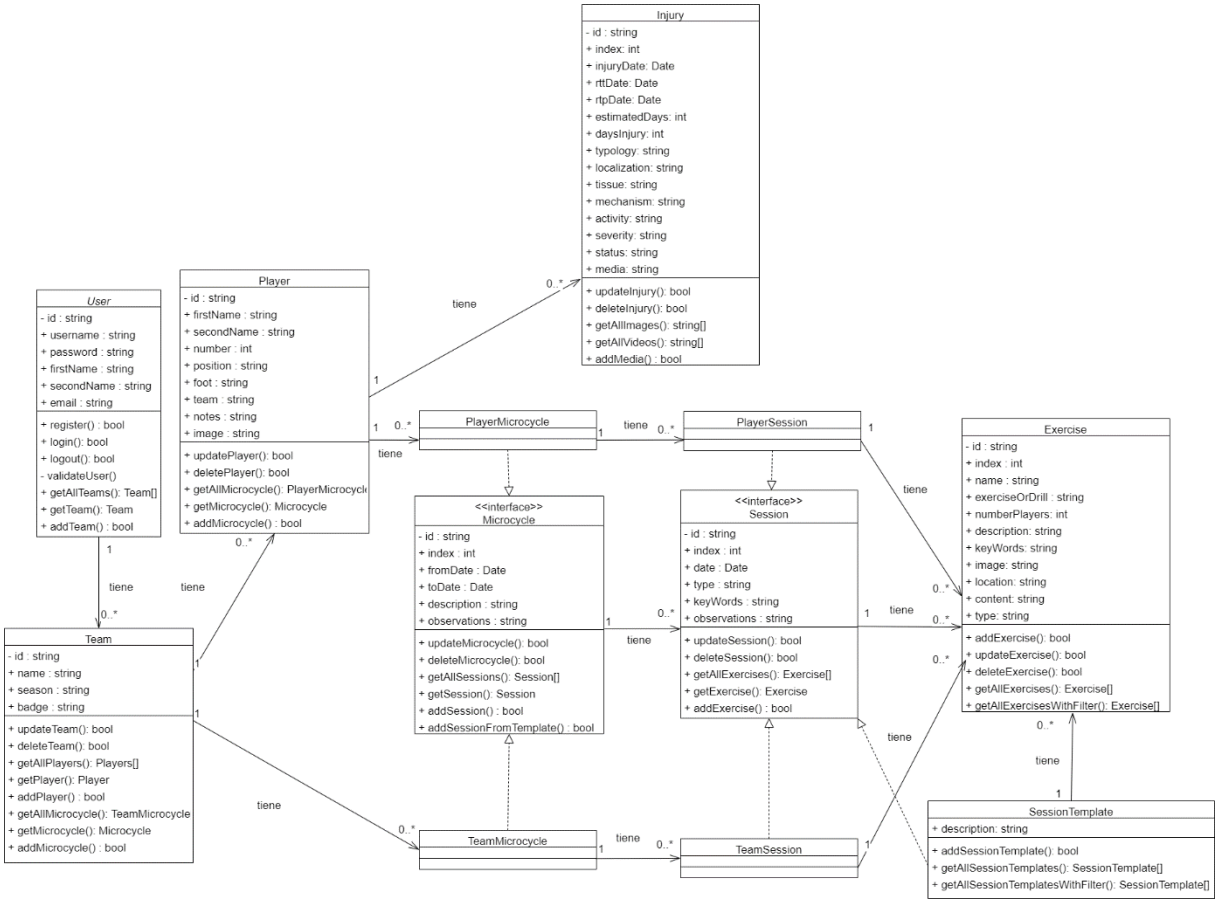


Figura 16. Diagrama de clases

Express.js

Para implementar una arquitectura MVC dentro de nuestra aplicación en Node.js, se ha optado por utilizar el framework Express.js. A pesar de que Express es un framework bastante minimalista por sí mismo, existen multitud de librerías compatibles para abordar muchos problemas del desarrollo web: trabajar con cookies, gestionar sesiones, integración con motores de renderización de "vistas", etc. Otra de las características de Express es que es un framework no dogmático, con esto se quiere decir que no es estricto en la manera de utilizar componentes para alcanzar un objetivo y da libertad al desarrollador en contraposición de aquellos frameworks que menos flexibles (“Introducción a Express/Node,” 2020).

A la hora de diseñar la arquitectura MVC, lo primero que se debe de hacer es determinar qué información queremos almacenar. Eso nos permitirá definir el modelo de los datos que tendrá que interaccionar con la base de datos (este paso ha sido llevado a cabo en el anterior apartado:

4.6. Diseño

Diagrama Entidad-Relación). A continuación, se deben identificar las acciones que queremos llevar a cabo para poder definir los controladores (Diagramas de secuencia) y las URLs apropiadas en las que se ejecutarán esas acciones. Y, por último, se crearán las vistas que en las que se renderizará la información.

En este caso, para realizar el diseño de la arquitectura MVC en Express, se ha utilizado una organización de carpetas en la que se deben añadir los archivos en función de la capa a la que pertenecen: models (modelos), views (vistas) y controllers (controladores). Estas carpetas se han creado en el directorio raíz del proyecto para dar organización al código.

El diagrama siguiente, Figura 17, proporciona un esquema sobre cómo se ha diseñado la aplicación MVC sobre Express. El servidor espera a recibir las peticiones HTTP del navegador o del cliente, y cuando esta sucede, la aplicación determina la acción adecuada de acuerdo con la estructura de la URL y a la información asociada a la petición dentro del enrutador. Dependiendo de dicha acción, puede que se necesite leer o escribir en la base de datos, y como respuesta el controlador crea una página HTML de forma dinámica en la que muestra la información solicitada con ayuda de las plantillas HTML disponibles en la vista.

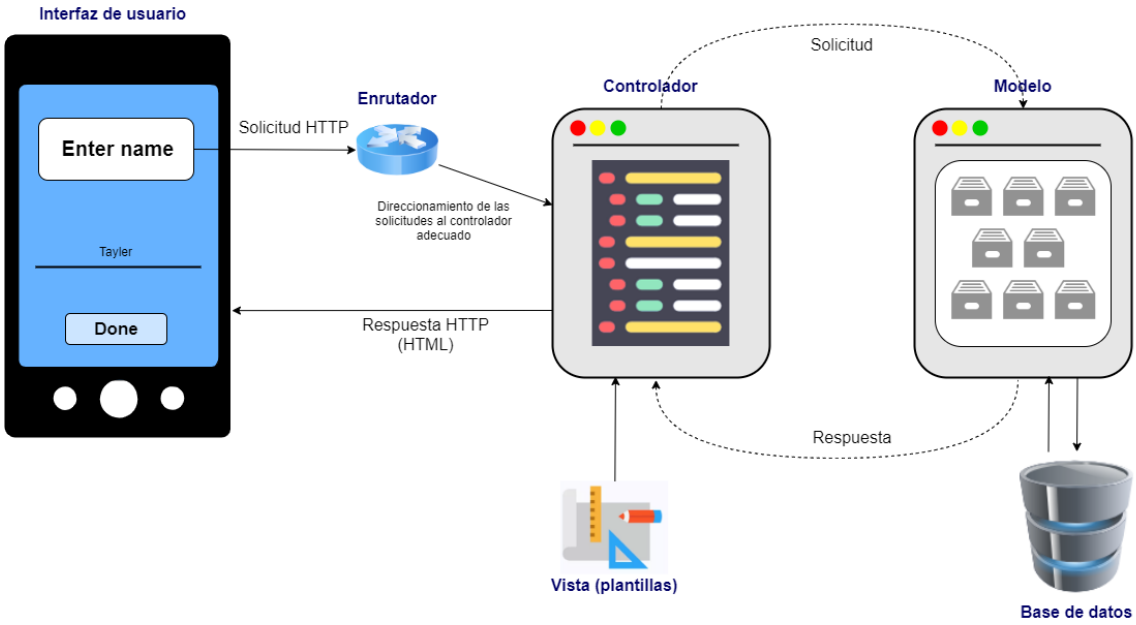


Figura 17. Flujo principal de trabajo en el diseño MVC para Express.js

5. IMPLEMENTACIÓN

5.1. Implementación de la arquitectura MVC con Express

Tal y como se ha adelantado anteriormente, la aplicación por diseñar implicará la visualización de diversas entidades: jugadores, ejercicios, sesiones..., y los controladores permitirán realizar las acciones necesarias descritas en los diagramas de secuencia. Para realizar una demostración de esta sección, se utilizará una representación mínima de la aplicación que nos permita diferenciar cada una de las capas del sistema.

Enrutador

A mayores de esas carpetas descritas en el apartado anterior (models, views, controllers), existe otra más llamada routes (rutas). En esta carpeta es donde encontramos todos los archivos que definen las rutas de la aplicación que redireccionarán las peticiones HTTP al apropiado controlador. A diferencia de otros grandes frameworks, Express no hace asunciones sobre la estructura o sobre los componentes a utilizar: rutas, modelos, vistas, controladores u archivos pueden estar en cualquier parte del directorio, sin embargo, es común una organización como la planteada.

Middlewares

Otro de los conceptos importantes dentro de las aplicaciones en Express es el uso de los middlewares. Los middlewares son bloques de código que se ejecutan entre la petición que hace el usuario hasta que la petición llega a nuestra lógica de la ruta en el servidor.

Su uso habitual es el de editar la petición o la respuesta, o la de terminar la respuesta previamente a que esta sea redireccionada al controlador. A mayores, también existe una multitud de middlewares disponibles en npm: uno de los más populares, y que nos puede servir como ejemplo, es body-parser. Este middleware sirve para validar y analizar el cuerpo de la petición HTTP, de forma que las peticiones que no cumplen esas características son rechazadas antes de llegar a los controladores. Por ello, su ubicación en el código debe ser previa al enrutamiento.

Rutas

Express ofrece la posibilidad de asociar una función de código (callback), con una dirección relativa al directorio raíz que contenga una petición HTTP (get, post, put, delete…). En nuestro caso las funciones callback están definidas dentro del controlador de jugador y son las encargadas de acceder a la información del modelo. En caso de haber más entidades

en la aplicación, se crearán tantos controladores como sean necesarios. A la hora de definir las rutas, existen diferentes alternativas, pero en el caso de este proyecto se ha utilizado el middleware express.Router y se han definido tantas clases router como entidades hay en el modelo. Con esto se puede concluir que el sistema de rutas sirve para invocar a los controladores por medio de una petición a determinada a una ruta.

En los anteriores apartados se han establecido cómo ha de ser la aplicación en cuanto a cómo ha de ser su interacción con el usuario y su estructura. A partir de este punto, el siguiente paso es diseñar la aplicación en sí para poder así formar una base sobre la que implementarla.

En las siguientes secciones se desarrollará con detalle cada capa anteriormente descrita en el diagrama de componentes de la Figura 17.

Clase principal

Para crear nuestro proyecto en node.js con TypeScript se ha optado por crear un archivo en el que definir la clase principal de la aplicación, App (Anexo 1.1). En esta clase se encapsulará nuestro servidor web y relacionará cada uno de los componentes. Dentro de la clase App se podrá configurar el puerto de acceso al servidor, definir los middlewares utilizados dentro de la aplicación, establecer la conexión con la base de datos (modelo), montar las rutas servidas por el servidor (Controlador), o configurar el motor de vistas a utilizar (Vista), entre otras cosas.

La finalidad es construir una clase que pueda ser instanciada para configurar el servidor y crear la conexión de forma rápida y ordenada. En la siguiente figura, se puede observar un ejemplo de cómo sería el archivo principal del proyecto (index.ts), donde la función main tan solo sería responsable de instanciar la clase App, definiendo el puerto 3000, poner el servidor en modo escucha y establecer la conexión con la base de datos.

```
import "reflect-metadata";
import { App } from './app'

async function main() {
    const app = new App(3000);
    await app.listen();
    app.databaseConnection();

 }

main();
```

Figura 18. Ejemplo del archivo index.ts

Base de datos MySQL

La primera de las capas por la que se empezará a implementar nuestra aplicación es por el Modelo. En esta capa se tratará de mantener encapsulada la complejidad de la base de datos para poder interaccionar fácilmente desde la aplicación. Dentro de las tareas a llevar a cabo por esta capa está la de definir el tipo de base de datos con la que vamos a trabajar. Cómo se ha anticipado, se ha optado por elegir una base de datos relacional MySQL, por lo que en este apartado nos centraremos en definir las tablas y sus relaciones.

Si bien es cierto que existen librerías para node.js que permiten hacer consultas a MySQL de forma directa, en esta aplicación se ha decidido utilizar un ORM (Mapeo de Objeto-Relacional) para agilizar el proceso y que a la vez permitieran un diseño sencillo y eficiente. El elegido ha sido TypeORM, que además de cumplir con los requisitos anteriores, se adapta muy bien a TypeScript. El beneficio principal de usar TypeORM es que el desarrollador puede centrarse en la lógica de negocio sin tener que preocuparse de la persistencia de datos, dejando esa preocupación en un segundo plano y además se puede implementar el diagrama de clases de forma directa ya que es un lenguaje que tiene herramientas de programación orientada a objetos. En el Anexo 8.5, se describen los pasos para crear un proyecto en el que se va a emplear la librería TypeORM.

Archivos

- *tsconfig-json*

Como resultado de crear un proyecto de TypeORM se creará un archivo llamado tsconfig.json, Anexo 8.6, en el directorio del proyecto que servirá para configurar TypeScript. El archivo puede tener decenas de configuraciones útiles en función del tipo de proyecto. Una versión elemental podría ser la siguiente:

- *target*: Utilizar estándar “es6” para compilar a código JavaScript.
- *module*: Sistema de módulos CommonJS. API síncrona que permite que los módulos sean cargados en el momento y en el orden que se requieren dentro de un archivo de código fuente.
- *sourceMap*: Genera o no archivos de mapa fuente para depurar directamente los archivos TypeScript de la aplicación en el navegador.
- *outDir*: Directorio donde se van a colocar los archivos Javascript, ".js", una vez transpilados.

- *strict/strictPropertyInitialization*: Permite utilizar TypeScript en modo stricto y obliga a inicializar las propiedades de clase o declarar explícitamente que pueden no estar definidas. Esto ayuda a capturar y prevenir errores comunes en el código.

- *moduleResolution*: Determina como resolver los módulos. Con el enfoque node, se cargan desde la carpeta node_modules como un módulo (require('module-name')).

- *esModuleInterop*: Habilita las importaciones predeterminadas de módulos CommonJS.

- *experimentalDecorators/emitDecoratorMetadata*: Permite utilizar decoradores con TypeORM. Estas dos variables son imprescindibles para poder utilizar TypeORM.

- *include/exclude*: Include determina el directorio y subdirectorios en el que se encuentran los archivos de TypeScript (*.ts o *.tsx) que el compilador incluirá, y exclude, aquellos están excluidos.

❖ *ormconfig.json*

Del proceso de configuración inicial del proyecto se genera también otro fichero que debe adaptarse a nuestra aplicación, ormconfig.json. La finalidad de este archivo es la de configurar la conexión con la base de datos. Los parámetros: username, password y database (nombre de la base de datos) deben ser adaptados a los valores usados cuando se crea la base de datos.

Otro parámetro importante para tener en cuenta es: synchronize. Este parámetro sirve para asegurarse de que las entidades están sincronizadas con la base de datos cada vez que se ejecuta la aplicación.

```
{
   "type": "mysql",
   "host": "localhost",
   "port": 3306,
   "username": "root",
   "password": "******************",
   "database": "ever_train",
   "synchronize": false,
   "logging": false,
   "entities": [
      "src/entity/**/*.ts"
   ],
   "migrations": [
      "src/migration/**/*.ts"
   ],
   "subscribers": [
      "src/subscriber/**/*.ts"
   ],
   "cli": {
      "entitiesDir": "src/entity",
      "migrationsDir": "src/migration",
      "subscribersDir": "src/subscriber"
   }
}
```

Figura 19. Ejemplo del archivo ormconfig.json

Entidades

En TypeORM las tablas son representadas por medio de Entidades. Dentro de la arquitectura propuesta, los modelos de la aplicación están directamente asociados a estos elementos llamados Entidades y a su vez a las tablas de datos. Por lo que para un modelo llamado Player, tendremos en el código una Entidad llamada de la misma forma, y una base de datos player. Para la definición de una Entidad con TypeORM es necesario utilizar decoradores. Estos decoradores son los responsables de que cuando compilemos nuestro proyecto y lo ejecutemos, la base de datos se inicialice y se creen las tablas para las entidades definidas en el código (“TypeORM,” 2020).

A continuación, se detallan algunos de los decoradores más comunes que se han utilizado en el desarrollo:

- @Entity: Por medio de este decorador indicamos que se debe de crear una tabla en la base de datos. Como consecuencia se crea una tabla con el nombre de la entidad y de esa forma es posible utilizarla desde cualquier parte de nuestra aplicación. Bien para insertar, leer, actualizar o borrar datos.

- @Column: Para añadir columnas a la tabla definida con el decorador @Entity simplemente se requiere utilizar @Column en la propiedad para la cual se quiera crear la columna. Los tipos de datos creados en la base de datos son inferidos a partir del

tipo de las propiedades usadas (number - integer, string - vchar, boolean – bool, etc), sin embargo, si se quiere ser más específico se puede utilizar el propio decorador para definir el tipo acorde con los tipos soportados por la base de datos a utilizar.

- @PrimaryColumn: Es la forma de definir aquella columna que va a ser empleada como clave primaria.

- @PrimaryGeneratedColumn: Define una columna como clave primaria auto incremental. Se puede utilizar en lugar del decorador anterior.

En el Anexo 8.7 se muestra un ejemplo de la definición de una entidad.

Relaciones

Como se puede observar en la Figura 7, las relaciones que tenemos en la base de datos son: uno-a-muchos/muchos-a-uno. Para definir estas relaciones con TypeORM es necesario utilizar los decoradores @OneToMany and @ManyToOne por cada par de entidades relacionadas. En el ejemplo de la Figura 21 esto debe ser realizado tanto en la clase Player como en la clase PlayerMicrocycle para que quede perfectamente establecida la relación entre ambas entidades, en las que un jugador (Player) puede tener múltiples microciclos, pero que cada microciclo pertenece a un único jugador (uno-a-muchos o 1:N). El Anexo 8.8 muestra cómo se debe realizar en el código.

Otro decorador que se ha utilizado para definir las relaciones es: @JoinColumn. La finalidad de este elemento de TypeORM es indicar que tabla gobierna la relación. Como consecuencia, en la tabla en la que se utiliza se crea una clave secundaria o foránea (Foreign Key). Esto quiere decir, que en el caso del ejemplo: la tabla del microciclo del jugador (PlayerMicrocycle) tiene información de a qué jugador pertenece, por eso se crea una clave secundaria *player*, pero como un jugador puede tener múltiples microciclos, en el jugador (Player) no existe columna que identifique al microciclo. Las tablas resultantes del código aquí definido pueden verse en la Figura 21.

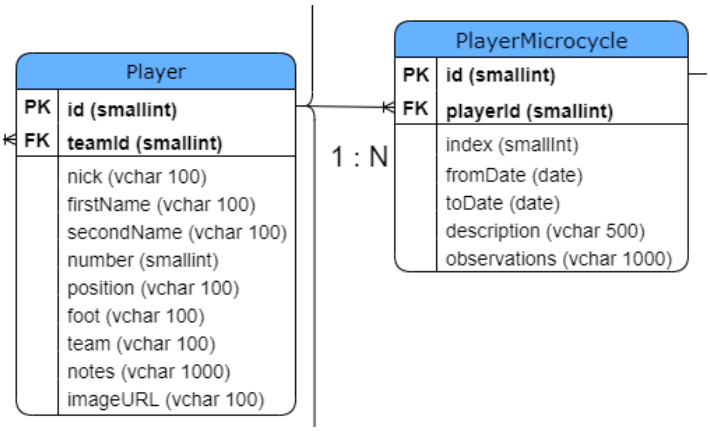


Figura 20. Relación entre las tablas Player y PlayerMicrocycle

```
mysql> SHOW TABLES;
+----------------------------+
| Tables_in_ever_train |
+----------------------------+
| player                     |
| playermicrocycle           |
+----------------------------+
2 rows in set (0.00 sec)

mysql> DESC player;
+-----------------+-----------------------------------------+-------+----------+-----------------+-----------------------+
| Field           | Type                                    | Null | Key      | Default         | Extra                 |
+-----------------+-----------------------------------------+-------+----------+-----------------+-----------------------+
| id              | smallint                                | NO   | PRI      | NULL            | auto_increment |
| nick            | varchar(100)                            | NO   |          | NULL            |                       |
| firstName       | varchar(100)                            | NO   |          | NULL            |                       |
| secondName      | varchar(100)                            | NO   |          | NULL            |                       |
| number          | smallint                                | NO   |          | NULL            |                       |
| role            | enum('Goalkeeper','Central Centre Back','Left Centre Back','Right Centre Back','Left
Fullback','Right Fullback','Left Winger','Right Winger','Defensive Midfielder','Offensive
Midfielder','Forward','Joker')                          | NO   |          | Joker           |                       |
| foot            | enum('Left','Right','Both')             | NO   |          | Right           |                       |
| team            | enum('First Team','U23')                | NO   |          | First Team      |                       |
| notes           | varchar(1000)                           | NO   |          | NULL            |                       |
| imageURL        | varchar(500)                            | NO   |          | NULL            |                       |
+-----------------+-----------------------------------------+-------+----------+-----------------+-----------------------+
10 rows in set (0.00 sec)

mysql> DESC playermicrocycle;
+-----------------+-----------------+--------+--------+-----------------+---------------------+
| Field           | Type            | Null | Key    | Default         | Extra               |
+-----------------+-----------------+--------+--------+-----------------+---------------------+
| id              | smallint        | NO   | PRI    | NULL            | auto_increment |
| index           | smallint        | NO   |        | NULL            |                     |
| startingDate    | date            | NO   |        | NULL            |                     |
| finishingDate   | date            | NO   |        | NULL            |                     |
| description     | varchar(500)    | NO   |        | NULL            |                     |
| playerId        | smallint        | YES  | MUL    | NULL            |                     |
+-----------------+-----------------+--------+--------+-----------------+---------------------+
6 rows in set (0.00 sec)
```

Figura 21. Tablas Player y PlayerMicrocycle en la base de datos

Controladores

Una vez que tenemos nuestras entidades definidas, es momento de crear los controladores que gestionarán las transacciones con la base de datos para cada entidad.

En este punto, para cada una de las entidades, se ha creado una clase con métodos básicos para crear, leer, actualizar o borrar la información de nuestras tablas, también conocidas como operaciones CRUD (create, read, update, delete), Anexo. En caso de querer definir cualquiera otra función específica en la que la aplicación tenga que acceder a la base de datos, o tenga que solicitar una vista distinta, es aquí donde se incluye.

En relación con el apartado siguiente (Vista), es interesante resaltar los métodos en el objeto respuesta *res* que aparecen al final de cada uno de los métodos del controlador. Su funcionalidad es la de enviar una respuesta al cliente y terminar el ciclo de solicitud/respuesta. Sin embargo, aunque estos métodos son invocados desde un controlador, si estos controladores no son llamados, la solicitud no funcionará.

Para terminar de definir cómo se realiza el ciclo de solicitud/respuesta, necesitamos mencionar las rutas (previamente definidas en el apartado Rutas). La instancia Router es un elemento importante dentro de Express (*express.Router)* y sirve para declarar las URIs (identificador uniforme de recursos) de la aplicación. Vienen a resolver el direccionamiento básico entre la solicitud del cliente y la respuesta de la propia aplicación.

La definición genérica de una ruta sería algo así:

```
app.METHOD(PATH,HANDLER)
```

Donde se define un método METHOD de solicitud HTTP (GET,POST,PUT,etc), una dirección relativa PATH que sería por donde se accede al servidor y un HANDLER que especifica la función que se ejecutará como respuesta (métodos anteriormente definidos en el controlador).

Dentro del campo PATH es posible añadir parámetros de ruta. Dichos parámetros son segmentos de URL que se usan para extraer valores especificados en esa posición en concreto de la URL. Los valores obtenidos se pasan como argumento al HANDLER y permiten hacer solicitudes más concretas dentro del controlador.

En el siguiente ejemplo se ilustran las rutas para el controlador Player, acorde al controlador definido en el Anexo 8.9:

```
import {Router} from 'express';
import PlayerController from "../controllers/player.controller";

class PlayerRoutes {

    public router: Router = Router();
    constructor() {
        this.config();
    }
    config(): void {
        this.router.post('/add', PlayerController.createPlayer);
        this.router.get('/', PlayerController.findAllPlayers);
        this.router.get('/edit/:id', PlayerController.findPlayer);
        this.router.post('/update/:id', PlayerController.updatePlayer);
        this.router.get('/delete/:id', PlayerController.deletePlayer);
        /* … Any other method defined in the controller …*/
    }
}

const playerRoutes = new PlayerRoutes();
export default playerRoutes.router;
```

Figura 22. Rutas definidas para el controlador PlayerController

Al mantenerse todas las llamadas a la base de datos desde un mismo código, garantizamos que desde la capa vista se pueda invocar al controlador para acceder a los datos del modelo y de esa forma realizar solicitar los datos y renderizar en una sola operación prácticamente.

Para el desarrollo de esta parte del código, se han utilizado algunos módulos adicionales entre los que destaca Class-transformer.

Class-transformer es un convertidor de tipos para transformar objetos a otras representaciones de clase y viceversa. Permite serializar/deserializar objetos dentro de los controladores, por ejemplo, cuando se recibe la información de un jugador en JSON y se quiere convertir en el objeto Player para así modificar algún campo.

Vistas

A la hora de visualizar la aplicación no sirve con solo añadir el archivo HTML. Las vistas que componen la interfaz de usuario deberán contener los elementos de interacción que permitan al usuario realizar acciones y a su vez visualizar el contenido solicitado al servidor de forma dinámica. Para hacer posible eso, se ha utilizado un sistema de plantillas, llamado Handlebars que se encarga de generar ese HTML.

Tal y como se adelantaba en el apartado anterior, en el propio código del controlador el objeto respuesta *res* es el encargado de enviar la respuesta al usuario. De entre todos los métodos de respuesta, existen uno en concreto, res.render(), que se encargan de realizar la tarea de renderizar la vista con los valores enviados como argumentos.

El método res.render(VISTA, [LOCALES]) se compone de un elemento *vista* que indica el nombre de la vista que se quiere renderizar y en segundo lugar, y por medio de un objeto JSON, de las variables *locales* que se quiere incluir en la vista.

```
public async findPlayer(req: Request, res: Response) {
    let idPlayer = req.params.id;
    let repository = getConnection().getRepository(Player);
    let playerData = await repository.findOne({where: {"id" : idPlayer}});
    res.render("edit", {player: playerData});
}
```

Figura 23. Método en el controlador para llamar a la vista "edit"

A mayores, otro de los beneficios que obtenemos del uso de este motor de plantillas es que es posible fragmentar el código HTML en pequeños ficheros de forma que se convierte en un código modular y reutilizable. De forma simplificada, las plantillas de Handlebars se componen de los siguientes elementos (mencionar que la estructura de carpetas dentro del directorio views sigue la misma estructura):

- Layouts: Son la capa de más alto nivel y son utilizados para definir el diseño de la página general, que a su vez puede estar compuesta por otros elementos como: menús de navegación, el cuerpo de la página web, el pie de página.

- Partials: Sirven para definir apartados como la cabecera, elementos comunes como menús, o el pie de página de la web

- Pages: Define cada una de las páginas de la aplicación. En el ejemplo siguiente se muestra la página “edit”, que cada vez que se renderice la vista edit será compartida, pero mostrará los datos específicos que le pasemos como argumento.

- Context: Es el contenido que se le pasa a las plantillas y que se encarga de actualizar los valores en función de las consultas al servidor.

- Helpers: Permiten añadir lógica a las vistas. Como ejemplo, nos permiten mostrar u omitir información en caso de que una condición se cumpla o no.

En el Anexo 8.10 se indican los pasos necesarios para la instalación y configuración de Handlebars en nuestro proyecto.

El ejemplo siguiente ilustra un ejemplo de una plantilla de layout “main”. En este layout, se puede ver como hay dos elementos principales que lo componen: el partial “navigation” y el “body”. Dentro de la carpeta partials tenemos definido el componente navigation (menú de

navegación) que será insertado en la parte superior, y para el body, Handlebars utilizará las páginas que nosotros queramos renderizar.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Books App - {{title}}</title>
    <link rel="stylesheet" href="https://bootswatch.com/4/cosmo/bootstrap.min.
</head>

<body>
    {{>navigation}}
    <div class="container p-4">
        <div class="row">
            {{{ body }}}
        </div>
    </div>
</body>
</html>
```

Figura 24. Ejemplo de layout "main"

A continuación, se puede ver un ejemplo de una de las páginas de contenido (Pages). En el caso de querer llamar a la vista “edit”, como es el caso de la Figura 23, lo que se visualizará es un formulario al que se le ha pasado un elemento en formato JSON llamado player. Es por ello por lo que para visualizar cada uno de los elementos que componen el objeto JSON tan solo es necesario especificarlo en el atributo “value” del componente del formulario al que se quiera añadir.

```
<div class="col-md-4 mx-auto">
    <div class="card card-body">
        <form action="/update/{{player.id}}" method="POST">
            <div class="form-group">
                <input type="text" name="nick" value={{player.nick}} class="form-
control" placeholder="Nick" autofocus required>
            </div>
            <div class="form-group">
                <input type="text" name="firstName" value={{player.firstName}} class="
form-control" placeholder="First Name" required>
            </div>
            <div class="form-group">
                <input type="text" name="secondName" value={{player.secondName}} class
="form-control" placeholder="Second Name" required>
            </div>
            <div class="form-group">
                <input type="text" name="number" value={{player.number}} class="form-
control" placeholder="Number" required>
            </div>

            /* … */

            <button class="btn btn-primary btn-block">
                edit
            </button>
        </form>
    </div>
</div>
```

Figura 25. Página "edit"

5.2. Implementación PWA

En este apartado se va a detallar cómo ha de llevarse a cabo la implementación de la tecnología PWA en una aplicación descrita con Node.js. La finalidad principal será implementar la funcionalidad offline de la aplicación, RF47.

Para llevar a cabo la implementación de una PWA existen dos requisitos indispensables a tener en cuenta:

1- El archivo Manifest

2- - Los Service Workers

Manifest

El manifest.json es un archivo en formato JSON que sirve para tener control sobre diferentes elementos de la apariencia de la aplicación. A través de este archivo es posible indicar al navegador diferentes elementos de la apariencia de la aplicación. Proporciona información

como el nombre, autor, descripción de la aplicación, tema y color de la pantalla del navegador, pantalla de inicio, o icono de la aplicación.

A mayores de la configuración de la apariencia, el archivo permite al navegador instalar la aplicación web en la pantalla de inicio de un dispositivo para proporcionar a los usuarios un acceso más rápido y una mejor experiencia. Un ejemplo simplificado del contenido del archivo sería el siguiente:

```
{
    "name": "PWA EverTrain",
    "short_name": "PWA_EverTrain",
    "description": "Progressive Web App example to be used in conjuction with
the EverTrain app",
    "scope": ".",
    "start_url": "/",
    "display": "standalone",
    "orientation": "portrait",
    "theme_color": #764ABC",
    "background_color": "#ffffff",
    "icons": [
      {
        "src":"/static/img/icons/splash-screen.png",
        "sizes": "512x512",
        "type": "image/png"
      }
    ]
}
```

Figura 26. Ejemplo del archivo manifest.json

Para la integración del archivo dentro de la aplicación, debemos incluirlo dentro de la cabecera del código HTML. En nuestro caso se colocará en la cabecera del archivo main.hbs y a mayores se incluirán etiquetas para configurar el estilo, el tamaño del viewport, el color del tema, o el icono de inicio en caso de que sea utilizado en un dispositivo iOS.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Books App - {{title}}</title>
    <link rel="manifest" href="/static/manifest.json">
    <link rel="stylesheet" href="https://bootswatch.com/4/cosmo/boot-
strap.min.css">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="apple-touch-icon" href="/static/img/icons/splash-
screen_192x192.png">
    <meta name="theme-color" content="#764ABC"/>
</head>

<body>
      /* … */
</body>

</html>
```

Figura 27. cabecera del archivo main.hbs

Service worker

El Service Worker es el encargado de garantizar que el contenido de la página se pueda ver offline mediante el almacenamiento en caché. A mayores de esa capacidad offline, el uso del Service Worker puede ser justificado también para proporcionar a los usuarios un sitio web más rápido y confiable.

Para realizar una introducción de cómo funciona el Service Worker se debe entender previamente el almacenamiento en caché de HTTP. En la actualidad los navegadores modernos ofrecen esta técnica de almacenamiento de forma que almacenan temporalmente contenido web y multimedia con la finalidad de obtener mayor rendimiento a la hora de cargar las páginas. Pero unido a esta gran capacidad viene la responsabilidad de gestionar adecuadamente este proceso. Cuando se realizan cambios significativos en una web es probable que se modifique desde el código HTML, CSS y cualquier archivo de JavaScript asociado para acomodar los cambios del estilo y el contenido. Una vez se hayan publicado los cambios en un sitio web, si no se ha realizado su almacenamiento en caché de forma adecuada puede dar lugar a que la página se visualice de forma incorrecta (Ater, 2017).

La diferencia entre el almacenamiento en caché de Service Worker con respecto al anteriormente descrito es que el Service Worker brinda mayor capacidad de control sobre los recursos que son almacenados y supone una mejora frente al método convencional de almacenamiento en caché. Con Service Workers, puede tener acceso a cualquier solicitud HTTP y decidir exactamente cómo desea responder. A la hora de escribir la lógica se puede

decidir qué recursos desea almacenar en caché, qué condiciones deben cumplirse y durante cuánto tiempo almacenar un recurso en caché.

En la siguiente figura se puede observar cómo funciona el Service Worker cuando un usuario accede a una página web. La primera vez que esto sucede, el Service Worker comienza a descargarse e instalarse. Y es durante esta etapa de instalación cuando se puede aprovechar para preparar la caché para tener contenidos todos los recursos críticos de la aplicación.

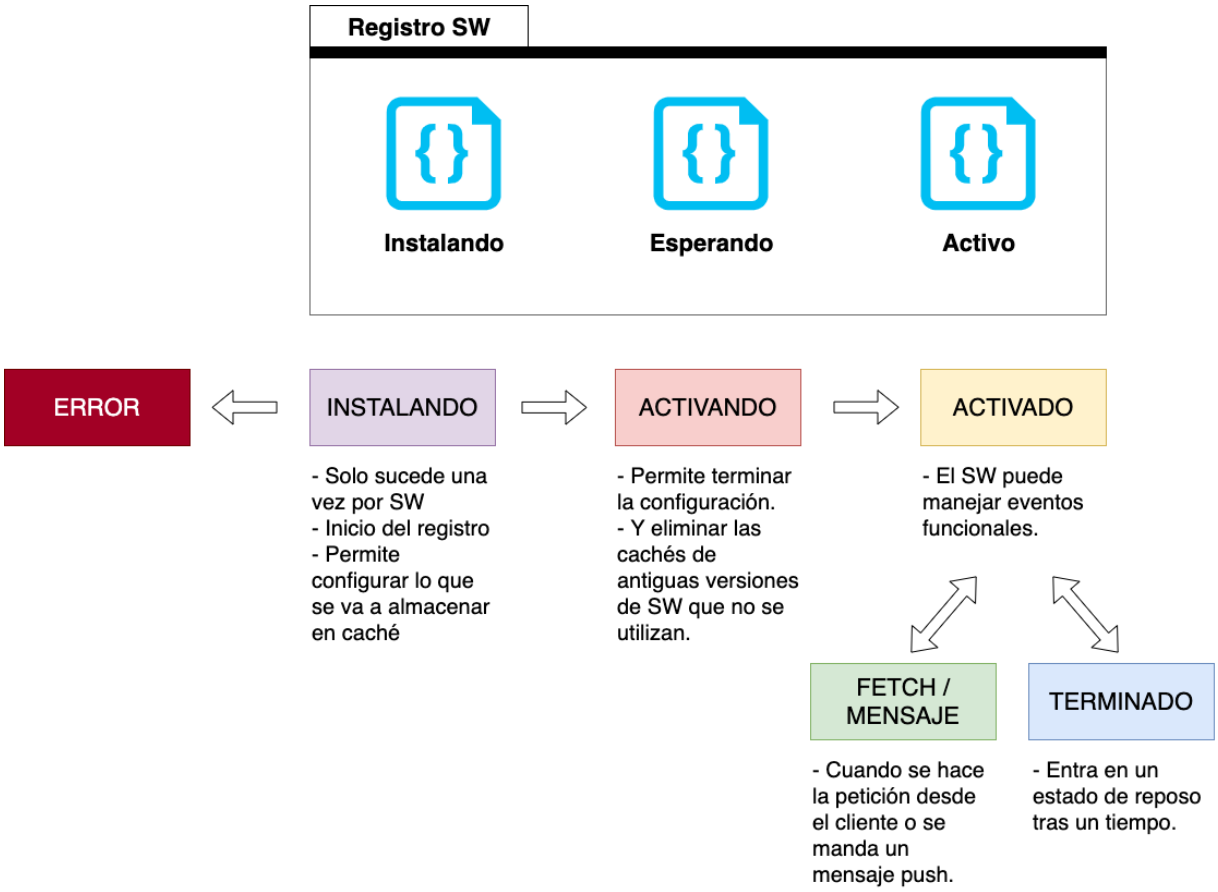


Figura 28. Ciclo de vida del Service Worker (Ater, 2017)

Usando la figura como ejemplo se puede implementar el Service Worker en nuestra aplicación para poder tener una mejor idea de cómo funciona.

1. En primer lugar, es necesario comprobar si el navegador que se está utilizando soporta Service Worker, si es así, intentará registrar un archivo llamado service-worker.js. Este código irá incluido en el archivo pwa.js

```
document.addEventListener('DOMContentLoaded', init, false);
function init() {
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/service-worker.js')
      .then((reg) => {
        console.log('Service worker registered -->', reg);
      }, (err) => {
        console.error('Service worker not registered -->', err);
      });
  }
}
```

Figura 29. Inicialización del service worker

2. A continuación, se debe crear el código que almacenará en caché los recursos, que irá dentro de ese archivo llamado service-worker.js

   Dado que la instalación es lo primero que sucede cuando se intenta registrar un SW, cuando el evento “install” es generado es cuando se deben especificar los archivos que se desean guardar en la caché. A su vez, cada vez que actualizamos el Service Worker, evento “actívate”, deberemos eliminar los antiguos SW en vez de dejarlos en el navegador del cliente y así dejar como activo el que se define en el evento “install”. Por último, nos queda el evento “fetch”. Una vez se detecta, se verifica si la URL coincide con algo que pueda existir en la caché, y en caso de que así sea, se devolverá el recurso almacenado.

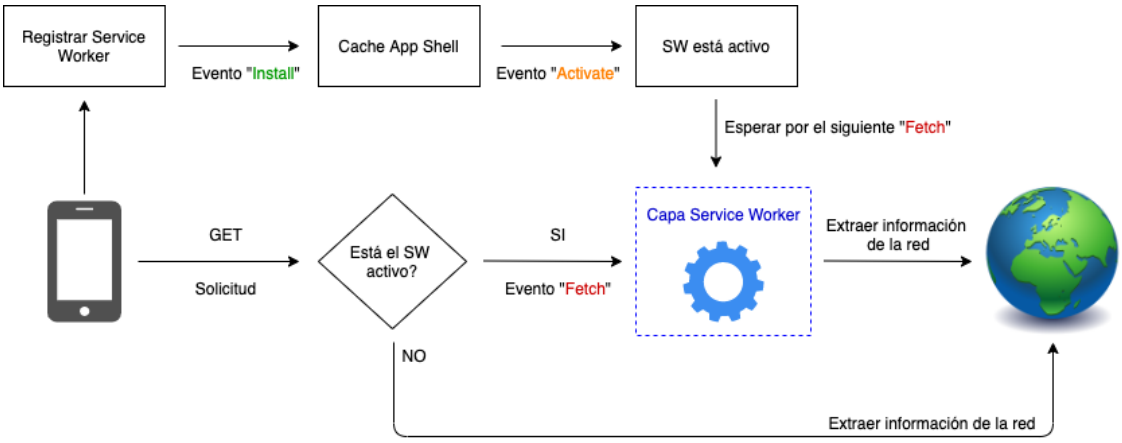


Figura 30. Eventos en el Service Worker (Hume, 2017)

El código de la Figura 31 se puede observar un ejemplo de la implementación del SW. En él se aprovecha el evento de instalación para agregar la plantilla que queremos cargar en caché. También hace referencia a un valor llamado CACHE_NAME en el que se ha guardado el nombre de la caché. Este valor es especialmente útil para el control de versiones. Una vez se ha abierta la caché se empieza a agregar recursos, que en el evento fetch serán utilizados.

```
const CACHE_NAME = 'sw-cache-example';
const toCache = [
  '/static/js/pwa.js',
  '/static/js/status.js',
  '/',
  '/add'
];

self.addEventListener('install', function(event) {
    console.log('used to register the service worker')
    event.waitUntil(
        caches.open(CACHE_NAME)
          .then(function(cache) {
            return cache.addAll(toCache)
          })
          .then(self.skipWaiting())
      )
  })

  self.addEventListener('activate', function(event) {
    console.log('this event triggers when the service worker activates')
    event.waitUntil(
        caches.keys()
          .then((keyList) => {
            return Promise.all(keyList.map((key) => {
              if (key !== CACHE_NAME) {
                console.log('[ServiceWorker] Removing old cache', key)
                return caches.delete(key)
              }
            }))
          })
          .then(() => self.clients.claim())
      )
  })

  self.addEventListener('fetch', function(event) {
    console.log('used to intercept requests to  check for data in the cache')
    event.respondWith(
        fetch(event.request)
          .catch(() => {
            return caches.open(CACHE_NAME)
              .then((cache) => {
                return cache.match(event.request)
              })
          })
      )
  })
```

Figura 31. Ejemplo de la implementación de los eventos del ServiceWorker.

6. INTEGRACIÓN Y VALIDACIÓN

En este apartado se propondrán herramientas para depurar y testear el código de la aplicación PWA en base a la filosofía Agile Testing. Esta metodología de pruebas no hace más que seguir los principios de desarrollo ágil de software, por lo que no se entiende esta fase como una etapa independiente sino como una etapa integrada dentro del desarrollo del software, al igual que la programación.

Algunas de las principales prácticas que se van a utilizar dentro del enfoque Agile Testing son el desarrollo guiado por pruebas o Test Driven Development (TDD) y la automatización de las pruebas.

6.1. Desarrollo guiado por pruebas (TDD)

La principal característica de esta metodología de desarrollo ágil es que unifica el proceso de testeo de una aplicación con el proceso de desarrollo, lo que hace que el producto final sea un software robusto y a prueba de fallos. Aplicada y mantenida a largo plazo esta práctica hace que el código incurra en menos errores por lo que aumenta la productividad de los desarrolladores. A mayores, permite que en cualquier momento del ciclo de vida del proyecto se pueda refactorizar o mejorar partes ya implementadas, puesto que existen pruebas que garantizarán el cumplimiento de los requisitos, y favorecerán una escritura más ordenada y con ejemplos de uso (las propias pruebas).

Para llevarla a cabo se deben de combinar 2 conceptos: Test-First Development (desarrollo de pruebas primero) y Refactoring (refactorización del código), por lo que el ciclo de desarrollo del proceso se divide en tres fases principales:

1. Definir los objetivos de nuestra prueba y escribir su posible implementación por medio de la definición de pruebas. Las pruebas deben generar errores dado que no se ha implementado la funcionalidad en el código.

2. Desarrollar el código conforme a la/s prueba/s definida en el anterior apartado. En este punto las pruebas se ejecutan de forma satisfactoria.

3. Refactorizar o mejorar el código. En este punto es posible también añadir nuevos casos de uso para asegurarse de que se cumplen los diferentes escenarios.

6.2. Automatización de las pruebas

Esta práctica consiste en utilizar marcos de trabajo (frameworks) para poder automatizar las pruebas de forma que se puedan ejecutar fácilmente y de manera integrada con la programación del código.

A la hora de definir las pruebas de validación, el punto de partida es considerar el Cuadrante de Pruebas Ágil, Figura 32, el cual indica que tipos de pruebas deben efectuarse en función de los factores considerados y muestra qué pruebas es recomendable llevar a cabo de manera manual o de manera automatizada.

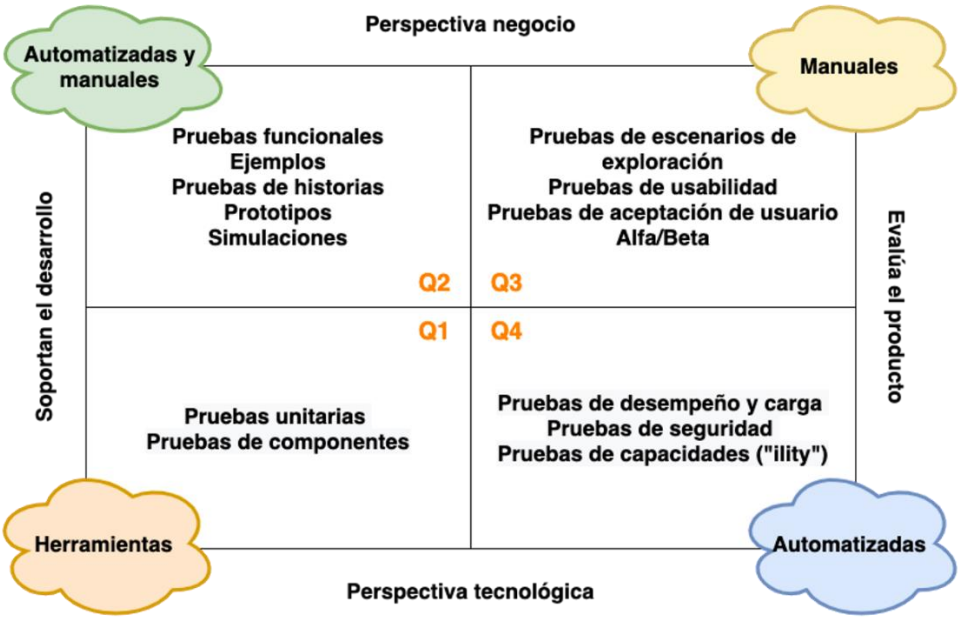


Figura 32. Cuadrante de las pruebas ágiles (Gregory & Crispin, 2014)

El primer cuadrante (Q1) está destinado al equipo de desarrollo desde la perspectiva tecnológica. Dentro de este grupo se engloban las pruebas unitarias y de integración de componentes del sistema de forma automatizada. La finalidad principal de las pruebas unitarias es la de garantizar la funcionalidad de unidades o componentes individuales del software. Con ellas se pretende validar que cada unidad de software producido funcione según lo establecido previamente. La ventaja principal de realizar este tipo de pruebas es que son más fáciles de implementar y no requieren de interfaz de usuario. A mayores y en base a la descripción del sistema y de los diferentes módulos que lo componen, se puede establecer pruebas de integración, estas pruebas sirven para garantizar que los componentes unitarios funcionen y cumplan los requisitos al ser ejecutados de forma conjunta. En principio pueden ser más difíciles de diseñar, pero si se tienen en cuenta ayudan al desarrollador a diseñar un

código más modular. A este nivel se busca asegurar la funcionalidad del código y las interfaces de los módulos definidos.

El segundo cuadrante (Q2) se enfoca también en el equipo del desarrollo, pero desde un punto de vista de negocio. Dentro de este grupo se tienen en cuenta tanto pruebas automáticas como manuales. La finalidad de estas pruebas de validación del sistema es la de garantizar desde un punto de vista funcional que el sistema esté siendo construido acorde a los documentos de especificación definidos en el proyecto, los requisitos funcionales y no funcionales. Generalmente las pruebas van asociadas a cada uno de los casos de uso y deben ser realizadas desde el punto de vista del usuario. Como recomendación es aconsejable automatizar aquellas pruebas funcionales que reflejen el comportamiento esencial de la aplicación desde el punto de vista del usuario, simulando la interacción del usuario con la propia interfaz.

El tercer cuadrante (Q3) engloba las pruebas de producto o servicio desde el punto de vista del negocio. Dichas pruebas de aceptación se realizan de forma manual y no son responsabilidad del equipo de desarrollo. Con estas pruebas se pretende validar el producto conforme a los requisitos del cliente. Aunque se pueden llevar a cabo en el entorno de desarrollo es imprescindible que se lleven a cabo también en el ambiente del cliente para garantizar el cumplimiento de las necesidades establecidas. En este punto es recomendable la participación de los usuarios finales.

Para finalizar, el cuadrante número 4 (Q4). Este último cuadrante se centra en aquellas pruebas que evalúan el producto final desde el punto de vista de operabilidad, como por ejemplo pruebas de carga, de estrés, seguridad… Estas pruebas se llevan a cabo de manera automatizada.

Como conclusión, cada tipo de pruebas cubre un propósito distinto, por lo que de forma independiente no son suficientes para desarrollar un proceso de validación de calidad. Es necesario también tener en cuenta cada uno de estos aspectos desde la fase inicial de su desarrollo e incluso en las etapas tempranas de conceptualización y diseño del producto (Crispin, Gregory, Lead, Mentor, & Services, 2009; Miclea, Stoian, Enyedi, IEEE Computer Society. Technical Council on Test Technology, & Institute of Electrical and Electronics Engineers, 2018).

Siguiendo de nuevo con las prácticas de Agile Testing, y centrándonos en las pruebas que recaen sobre el equipo de desarrollo, Q1 y Q2 principalmente, en este proyecto se ha propuesto una distribución del 70/20/10 en base a la pirámide de automatización de pruebas de Mike Cohn (Cohn, 2010), Figura 33.

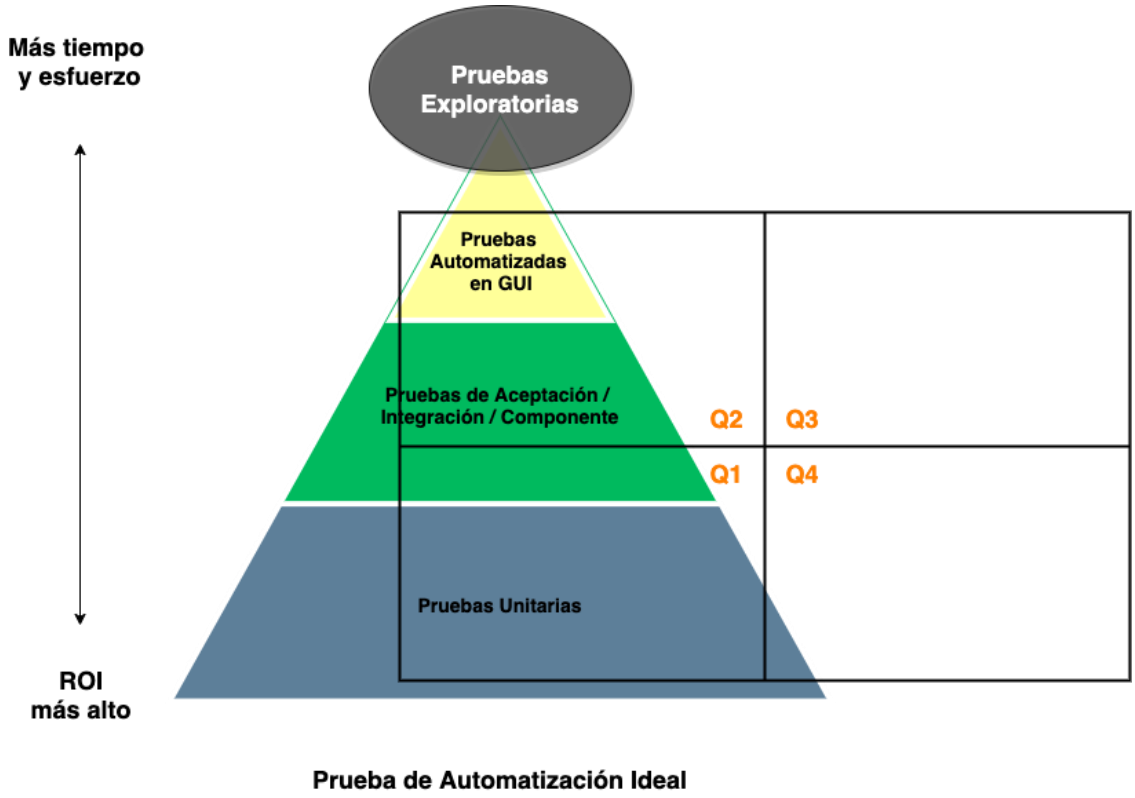


Figura 33. Pirámide de automatización de pruebas de Mike Cohn (Cohn, 2010)

En la base de esta pirámide se representan las pruebas unitarias automáticas, que vendrían a suponer la gran mayoría de las pruebas realizadas al sistema (70% aprox). Este es un punto de vital importancia para detectar fallos a nivel desarrollador y evita propagar esos errores a las capas de arriba. Esta gran cobertura de pruebas unitarios ayuda a documentar mejor nuestro producto.

Estas pruebas deben ser seguidas por las pruebas automatizadas de integración de componentes y aquellas pruebas funcionales que no requieren de intervención del usuario con la interfaz (20%) y por último un menor grupo de test de interfaz gráfica automatizados (10%). La razón por la cual el porcentaje es menor en estos últimos grupos es la dependencia de estas pruebas con respecto a multitud de componentes del sistema, lo que hace que sean más lentos en ejecución y más susceptibles a fallar en caso de modificar alguna parte del código a nivel componente.

Por último, en la cima de la pirámide, se muestran las pruebas exploratorias manuales que el equipo de desarrollo no puede llevar a cabo de forma automática.

6.3. Herramientas propuestas

A continuación, se detallarán las herramientas propuestas para llevar a cabo este proceso de validación de software.

A pesar de que existe en el mercado una multitud de alternativas para llevar a cabo tests unitarios: Jasmine, Ava, Karma, Chai, en este apartado se sugiere la utilización de Jest para llevar a cabo esa tarea. La razón por la que se recomienda la utilización de este marco de pruebas es porque es uno de los más populares y soporta TypeScript. Jest tiene una gran comunidad, es una herramienta de open source creada y mantenida por Facebook y funciona tanto del lado del cliente (frontend) como del lado del servidor (backend).

Sin bien esta herramienta puede también ser utilizada para realizar pruebas de integración y de servicios, puede que no sea suficiente para realizar las pruebas de integración de las APIs. Para cubrir esa necesidad se propone integrar Supertest en nuestra aplicación. Supertest no es más que una librería de pruebas de Node.js para servidores de HTTP que permite llamar a las rutas del servidor por medio de solicitudes HTTP y de esa forma validad el comportamiento del sistema.

Falta por tanto la elección de la herramienta para realizar las pruebas de interfaz gráfica. Una de las principales preocupaciones en el desarrollo web es el correcto funcionamiento en el máximo número de navegadores posibles. La idea es utilizar Selenium para cubrir gran parte de ese proceso de validación. Selenium es un entorno de pruebas que permite grabar, editar y depurar lo que se muestra en el navegador, y que al igual que los anteriores es de código abierto.

Con la elección de estas herramientas y de forma genérica queda descrita la estrategia de validación de una aplicación web. Sin embargo, en nuestro proyecto la aplicación presenta una singularidad, y es que no es una aplicación al uso, sino que debe cumplir unos requisitos a mayores para poder ser considerada PWA por los navegadores. Realizar pruebas de validación a una PWA implica que, a mayores de validar los requisitos del sistema a nivel unitario, de integración y de la propia interfaz como se realiza con las aplicaciones convencionales, debemos ser capaces también de asegurar la calidad de las guías de diseño de PWAs. Es así, que existen algunos pilares clave para garantizar su correcto funcionamiento. En base a estos criterios será necesario implementar una serie de pruebas manuales y basadas en software o automatizadas que en ocasiones es necesario realizar en distintos dispositivos y navegadores:

Validar el documento Manifest

Este archivo es imprescindible para las PWA. Se debe comprobar que los siguientes campos o propiedades están incluidos: name o short_name, start_url, iconos de *192px x 192px* y *512px x 512px,* display: preferiblemente en modo standalone,

Existen también herramientas de software que pueden ser utilizadas para realizar esta validación. Un ejemplo es Web Manifest Validator (https://manifest-validator.appspot.com/). Esta aplicación web se encarga de validar el archivo conforme a la especificación W3C. En caso de que tenga errores el propio software proporciona información al respecto.

Validar los Service Workers

Realizar pruebas a los Service Workers requiere un apartado especial. A la hora de realizar esta tarea pueden crearse tanto pruebas unitarias como de integración para garantizar su correcto funcionamiento.

Un ejemplo de las pruebas de integración sería cargar una página, registrar un Service Worker, parar el servidor para simular que no existe conexión a internet y actualizar la página. En este caso se debería comprobar que al actualizarla la página se carga desde la memoria caché del Service Worker. Por otro lado, un ejemplo práctico de una prueba unitaria sería simplemente el registrar un Service Worker, esperar a que se instale y verificar que se hayan almacenado en caché los archivos indicados.

Hay que tener en cuenta que tanto las pruebas de integración como las pruebas unitarias que requieren ejecutarse en el navegador real no son pruebas especialmente rápidas y pueden dar problemas a la hora de ejecutar. Sin embargo, las pruebas unitarias a funciones específicas del Service Worker pueden ser mucho más sencillas dado que existen herramientas, como puede ser Mocha, que nos permiten simular fácilmente las APIs que estás intentando probar.

Pruebas manuales de comportamiento similar a las apps nativas

Este punto requiere pruebas manuales en navegadores y dispositivos diferentes. Uno de los principales comportamientos es la capacidad de agregar la pantalla de inicio al dispositivo como cualquiera otra aplicación. Una vez la pantalla es agregada a la pantalla de inicio debería iniciarse como una aplicación y no como un sitio web. Aquí se puede comprobar también la capacidad de funcionar Offline, sin conexión.

Pruebas basadas en software de comportamiento similar a las apps nativas

Se recomienda realizar las pruebas manuales para los navegadores y dispositivos más comunes, sin embargo, se puede utilizar la herramienta Lighthouse en Chrome para evaluar algunas de las características nativas.

Entre las propiedades que puede comprobar Lighthouse está la funcionalidad offline. La herramienta examina esa característica buscando los metadatos que permiten a los navegadores saber qué hacer cuando la PWA se inicia en modo sin conexión. A mayores Lighthouse puede probar la capacidad de carga de todas las páginas en modo offline, al igual que para el modo online.

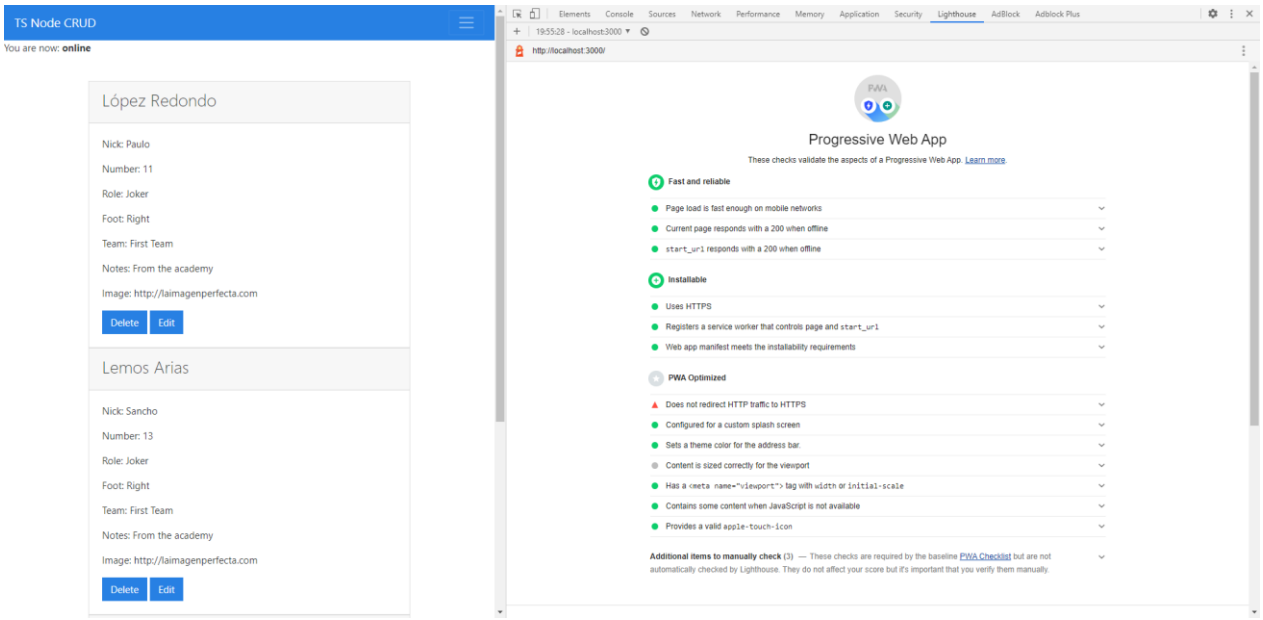


Figura 34. Resultado del análisis PWA de Lighthouse

Pruebas de fiabilidad (reliability)

La fiabilidad de las PWA depende de la capacidad de servir páginas a través de HTTPS. Esta es una de las condiciones indispensables que deben cumplir las PWA para garantizar la seguridad de los datos de los usuarios, por lo que a la hora de llevar una aplicación a producción debe tenerse en cuenta este requisito.

Como se puede observar en la figura anterior, este requisito aparece marcado como rojo dado que no se cumple en el ejemplo desarrollado. Una forma de evitar obtener ese error cuando se realizan las pruebas en local es generar un certificado autofirmado y cambiar el servidor

para redirigirlo a HTTPS para depurar y testear la aplicación. Sin embargo, el certificado autofirmado no sirve para garantizar la identidad de un servidor y llevar nuestra aplicación a producción dado que no es un certificado emitido por entidad autorizada, pero provee un canal de comunicación seguro en el que realizar las pruebas.

Como se puede deducir del anterior apartado, Lighthouse permite garantizar esta propiedad.

Hacer uso correcto de las URLs

A pesar de que las PWAs puedan comportarse como aplicaciones, no nos debemos olvidar que son sitios web. Es por ello por lo que todas las páginas deben tener su URL correspondiente que permita que se puedan compartir y encontrar fácilmente. Esta característica asegurará que los rastreadores de los motores de búsqueda indexen el sitio correctamente.

Pruebas de navegación cruzada

Como se ha visto anteriormente en el Estado del Arte, las funcionalidades de las PWA dependen en gran medida de los navegadores. Esto hace que sea importante asegurarse de que la aplicación web cargue y se comporte como se espera en los navegadores más populares: Chrome, Firefox, Safari, etc. Puede ser posible que un escenario particular tenga distintos comportamientos en función del navegador. Para garantizar la uniformidad se deberá verificar manualmente la carga tanto en dispositivos web como de escritorio. Lo mismo para aquellos navegadores más antiguos en los que no se permitan las PWA, en ellos es crucial comprobar que al menos funcionen como lo haría una aplicación web normal.

Basándonos en la gran dependencia de las PWA con los navegadores modernos, las pruebas de navegadores cruzador tienen una gran importancia a la hora de desarrollar una aplicación PWA de calidad.

7. CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

7.1. Conclusiones

Con el desarrollo de este TFM se ha realizado una planificación de una herramienta web PWA para la planificación de tareas en un equipo de fútbol. Tal y como se indicó en el apartado de Objetivos y motivaciones, se han tenido en cuenta las siguientes características: diseño de tareas individuales y grupales, planificación semanal mediante microciclos, diseño de plantillas de sesiones y de ejercicios y capacidad de diseño offline.

El desarrollo del trabajo aquí planteado permitirá un punto de partida para la simplificación del proceso de diseño y de gestión diaria de las tareas de un equipo a un bajo coste, y permitirá a su vez la incorporación de otras funcionalidades en un futuro.

Se ha tenido también en cuenta en el diseño que el acceso a la información debe ser posible tanto a través de navegadores web como aplicaciones para dispositivos móviles y a su vez en condiciones sin datos (offline), esto permitirá diferenciarse dentro del mercado.

La planificación del software se realizó por medio de una metodología combinada, lo cual permitió un diseño detallado en las etapas previas del proyecto incluyendo el análisis de mercado, la definición de los requisitos y casos de uso, la arquitectura, la interfaz de usuario y el diseño de los diagramas de Entidad-Relación, diagramas de secuencia y diagramas de clase. Esta tarea de especificación favorece la implementación del software por parte de los desarrolladores. A su vez, la incorporación de Scrum añade la capacidad de iteración y revisión de los requisitos durante las etapas de desarrollo y validación principalmente, centrando en todo momento el alcance del proyecto en los requisitos definidos por el usuario.

Análisis del cumplimiento de los objetivos:

- ✓ Se ha tenido en cuenta el diseño de tareas individuales y grupales: Para ello la herramienta se ha diseñado de forma que el usuario tiene acceso a cada uno de los jugadores para poder diseñar las tareas individuales, pero a su vez dentro del apartado del equipo existe la posibilidad de realizar sesiones grupales.

- ✓ Microciclo como unidad de medida: En todo momento la planificación de las sesiones tiene la vista del microciclo por defecto.

- ✓ Bajo coste: Se ha limitado el diseño a los requisitos indispensables para el usuario para asistir en las tareas diarias de planificación y control.

- ✓ Capacidad de elaborar y guardar tareas y sesiones para su posterior utilización en el diseño de las tareas.

- ✓ Capacidad de diseño de sesiones offline: Mediante la utilización de Service Workers.

Con el cumplimiento de estos apartados se concluye que los objetivos planteados al inicio del TFM se han llevado a cabo. Sin embargo, existen funcionalidades que podrán ser añadidas en proyectos futuros, así como pequeñas mejoras o desviaciones en el diseño que pueden ser resultado de las iteraciones en la etapa de desarrollo de software.

7.2. Líneas de trabajo futuro

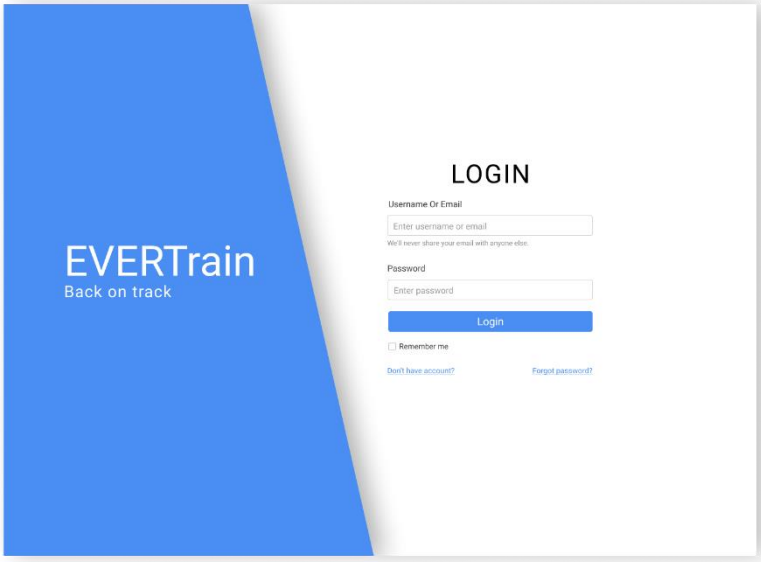
Las líneas de trabajo futuro de este proyecto pasan por complementar el trabajo realizado en el presente TFM. Dichas tareas estarán centradas en el desarrollo e implementación del software, y en la elaboración de las pruebas necesarias acorde a las directrices acordadas a lo largo del proyecto.

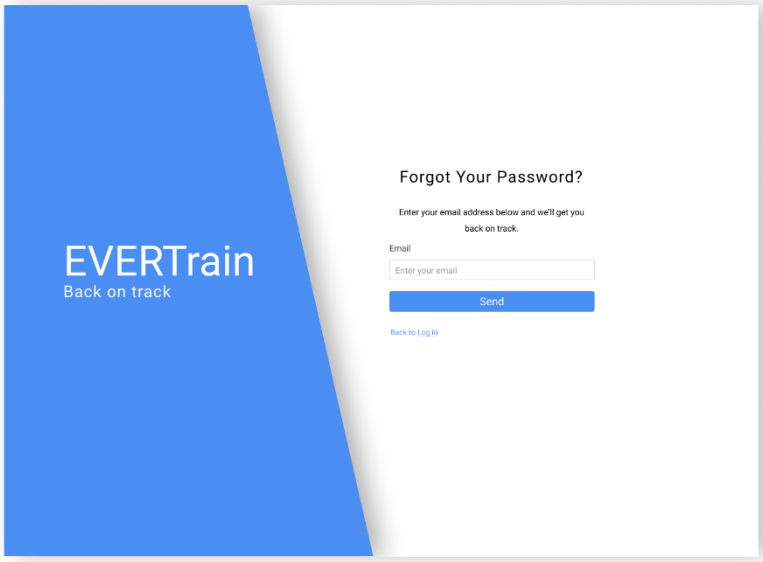
Una vez el software esté funcionando, será de gran utilidad la incorporación de herramientas de análisis para poder elaborar informes, tales como gráficas de contenidos trabajados o la estimación de la carga trabajada en función de las sesiones realizadas. Dichas herramientas facilitan la identificación de patrones de riesgo lesionales en los jugadores y prevendrá recidivas en los lesionados.

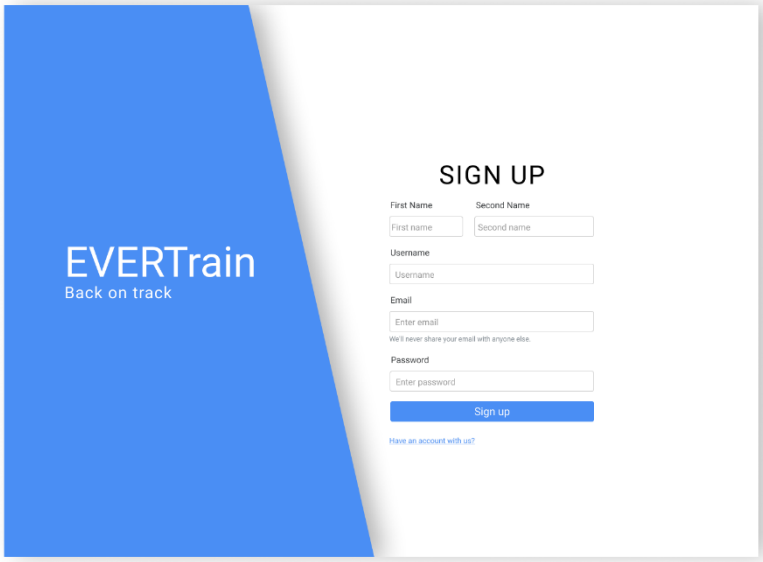
8. ANEXO

8.1. Diseño de la Interfaz de usuario

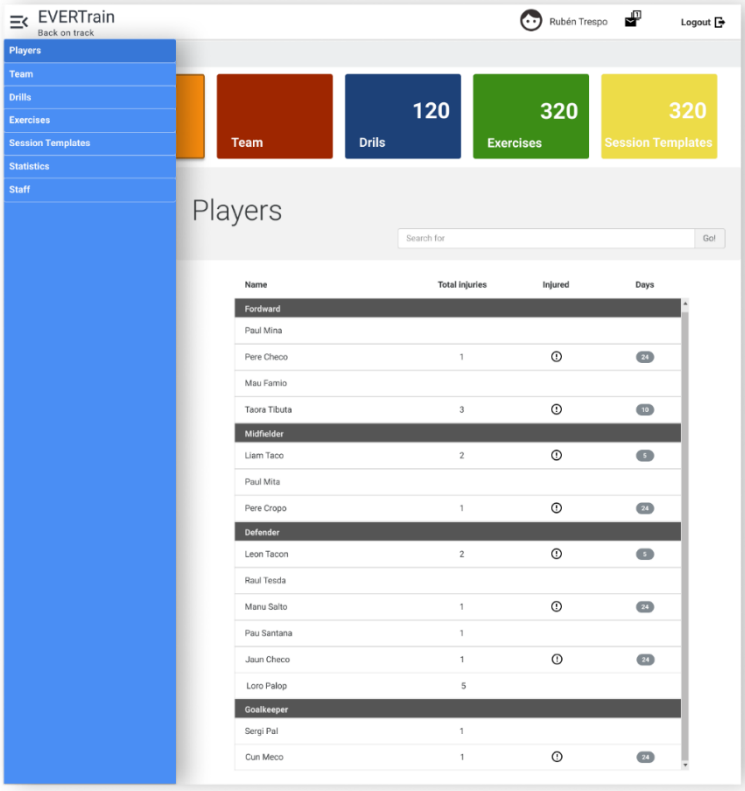
Acceso

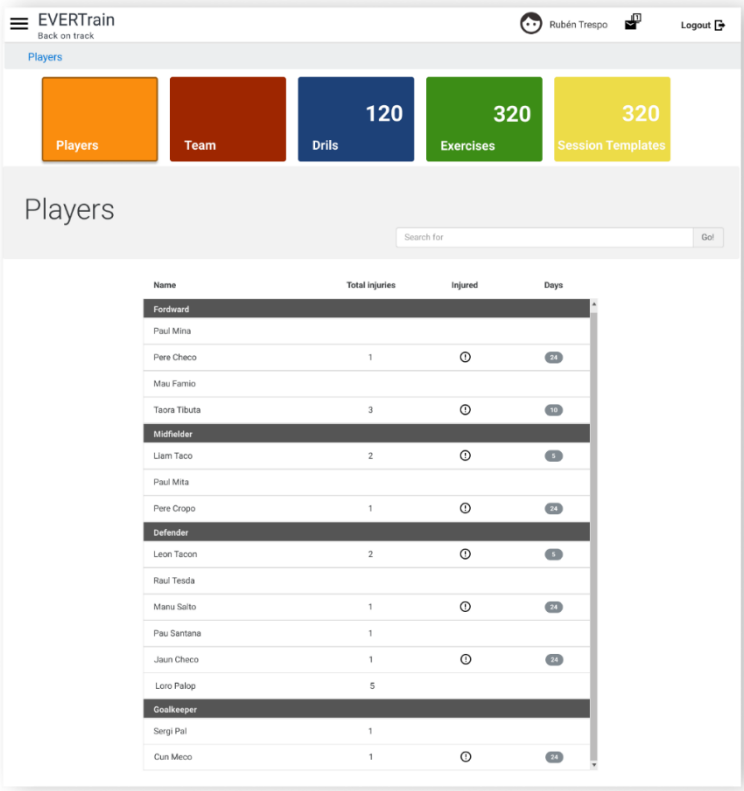


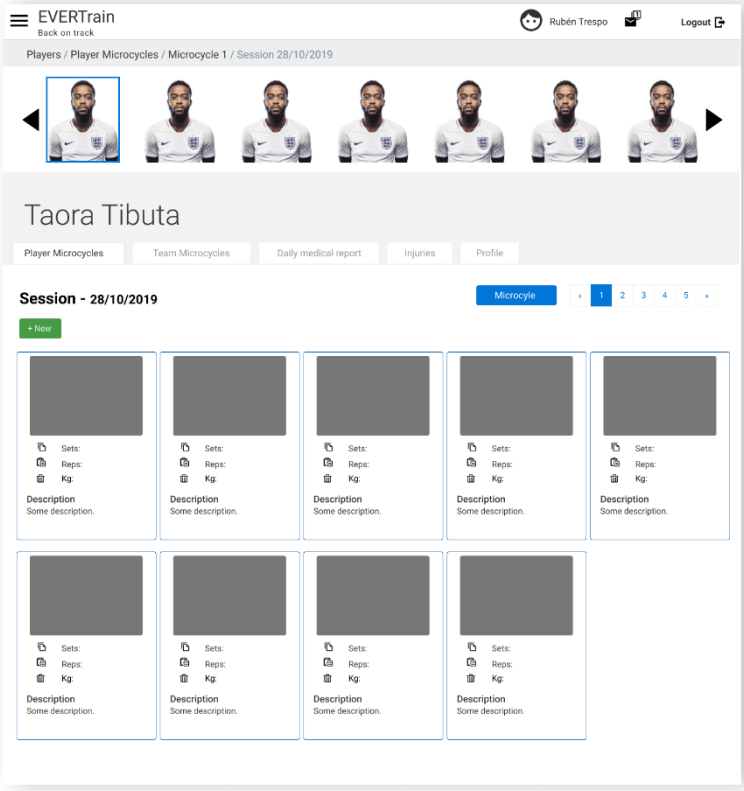


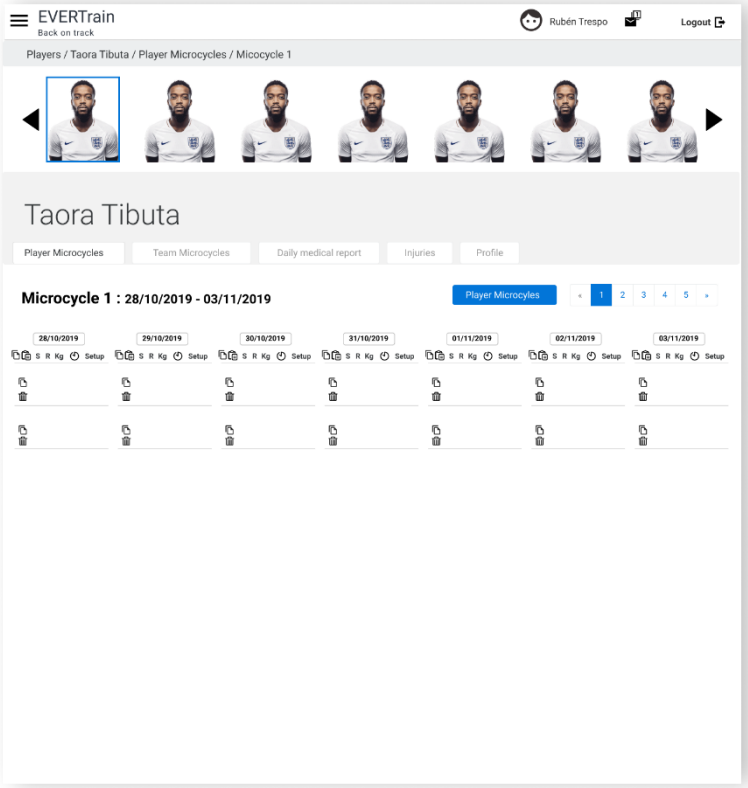

EVERTrain
Back on track
SIGN UP
First Name
Second Name
First name
Second name
Username
Username
Email
Enter email
We'll never share your email with anyone else.
Password
Enter password
Sign up
Have an account with us?

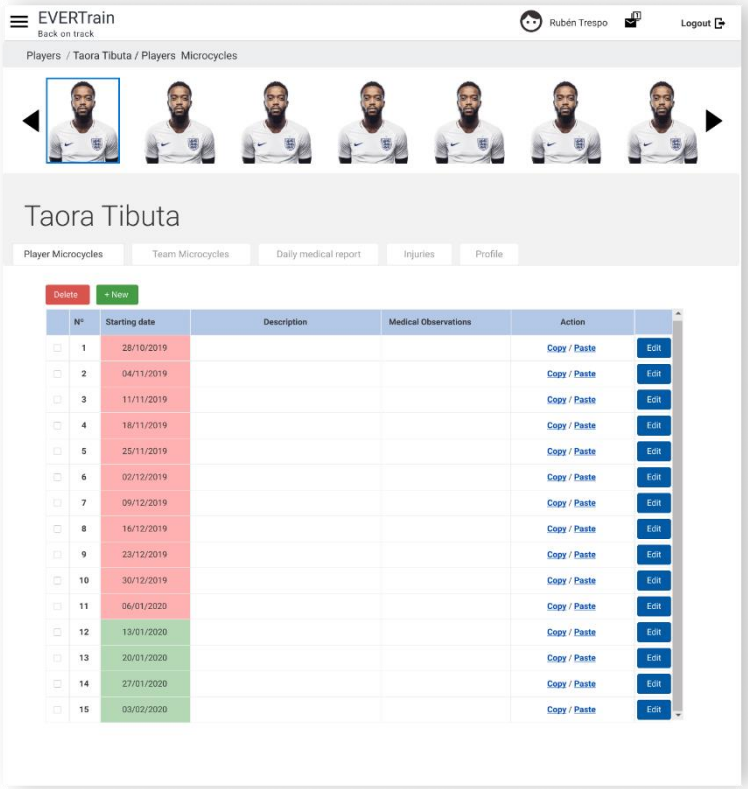
Jugadores

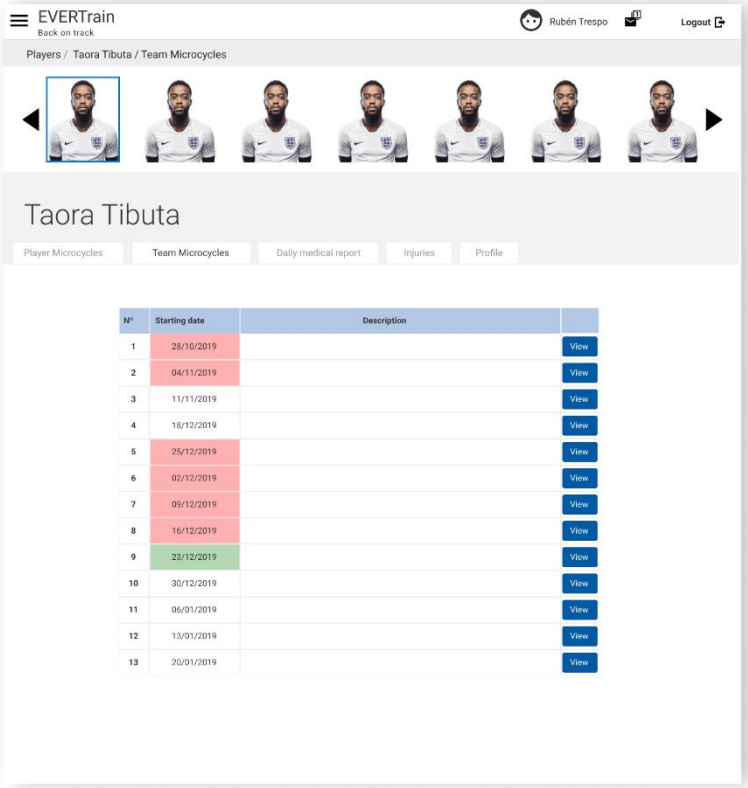

EVERTrain
Back on track
Rubén Trespo
Logout
Players
Team
Drills
Exercises
Session Templates
Statistics
Staff
120
Drills
320
Exercises
320
Session Templates
Players
Search for
Go!
Name
Total injuries
Injured
Days
Fordward
Paul Mina
Pere Checo
1
24
Mau Famio
Taora Tibuta
3
18
Midfielder
Liam Taco
2
5
Paul Mita
Pere Cropo
1
24
Defender
Leon Tacon
2
5
Raul Tesda
Manu Salto
1
24
Pau Santana
1
Jaun Checo
1
24
Loro Palop
5
Goalkeeper
Sergi Pal
1
Cun Meco
1
24

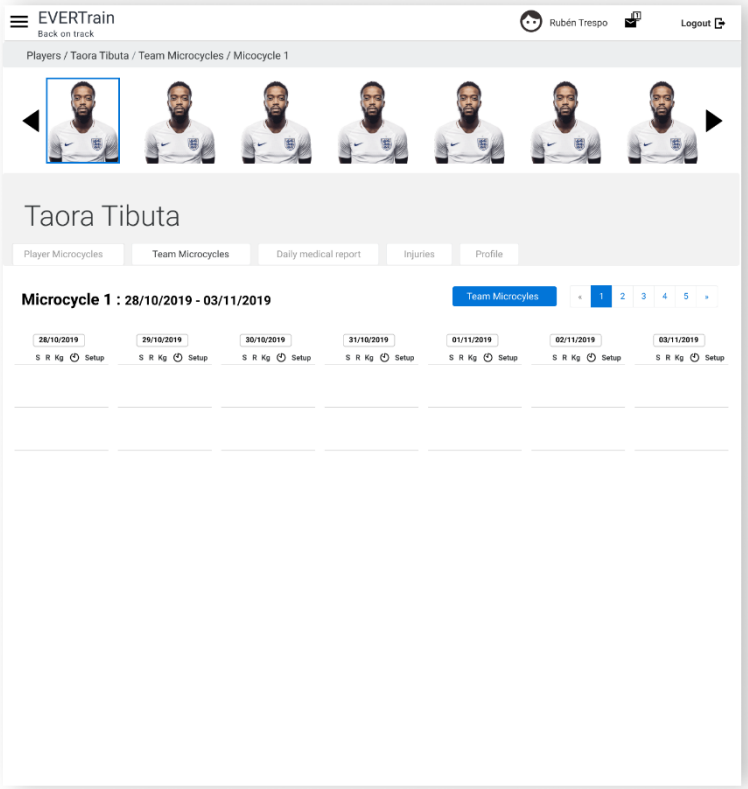

EVERTrain
Back on track
Rubén Trespo
Logout
Players
Players
Team
120
Drils
320
Exercises
320
Session Templates
Players
Search for
Go!
Name
Total injuries
Injured
Days
Fordward
Paul Mina
Pere Checo 1 24
Mau Famio
Taora Tibuta 3 10
Midfielder
Liam Taco 2 5
Paul Mita
Pere Cropo 1 24
Defender
Leon Tacon 2 5
Raul Tesda
Manu Saito 1 24
Pau Santana 1
Jaun Checo 1 24
Loro Palop 5
Goalkeeper
Sergi Pal 1
Cun Meco 1 24

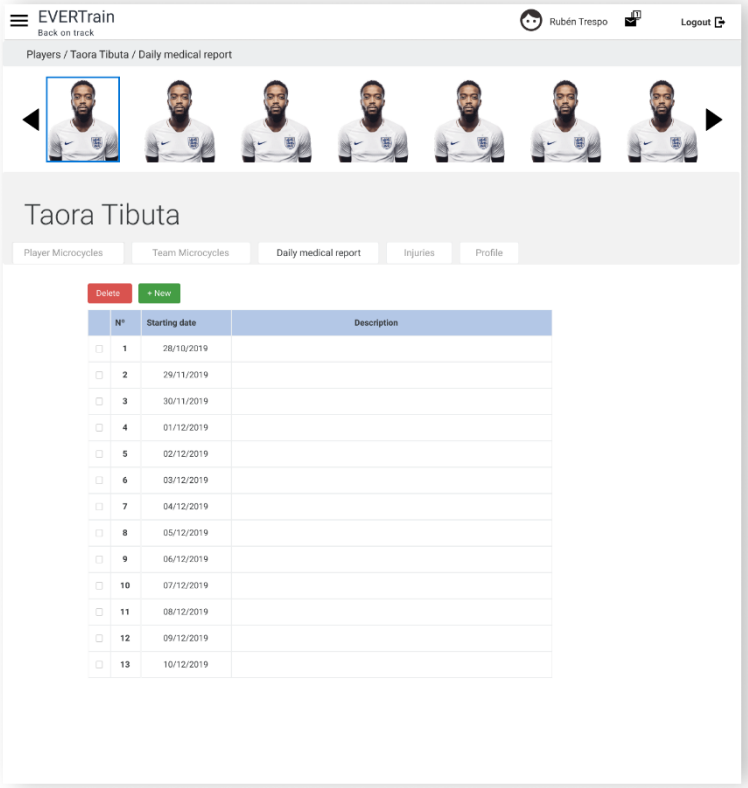

EVERTrain
Back on track
Rubén Trespo
Logout
Players / Player Microcycles / Microcycle 1 / Session 28/10/2019
Taora Tibuta
Player Microcycles
Team Microcycles
Daily medical report
Injuries
Profile
Session - 28/10/2019
Microcyle
+ New
Sets:
Reps:
Kg:
Description
Some description.

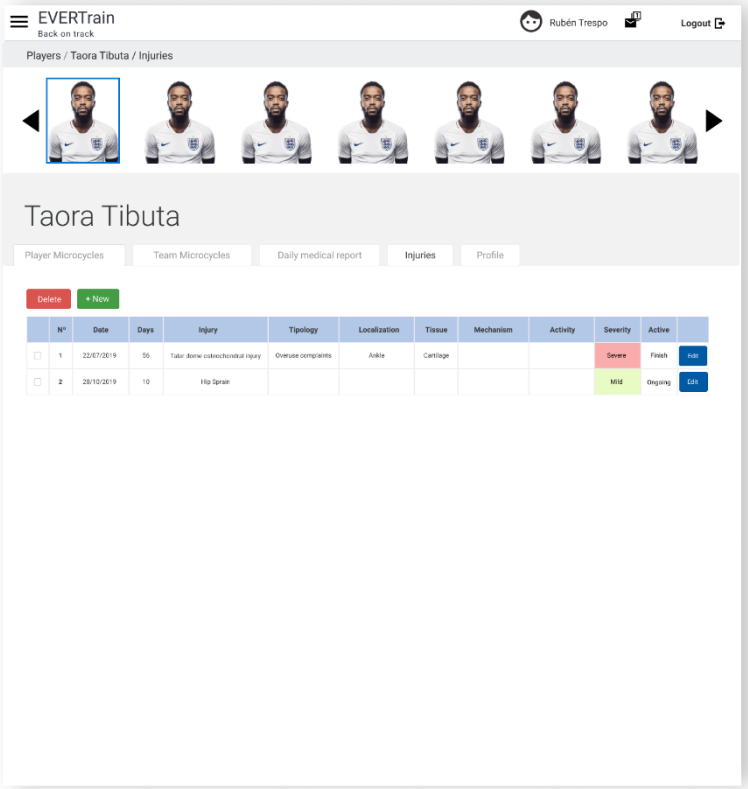

EVERTrain
Back on track
Rubén Trespo
Logout
Players / Taora Tibuta / Player Microcycles / Micocycle 1
Taora Tibuta
Player Microcycles
Team Microcycles
Daily medical report
Injuries
Profile
Microcycle 1 : 28/10/2019 - 03/11/2019
Player Microcycles
28/10/2019
29/10/2019
30/10/2019
31/10/2019
01/11/2019
02/11/2019
03/11/2019
S R Kg Setup

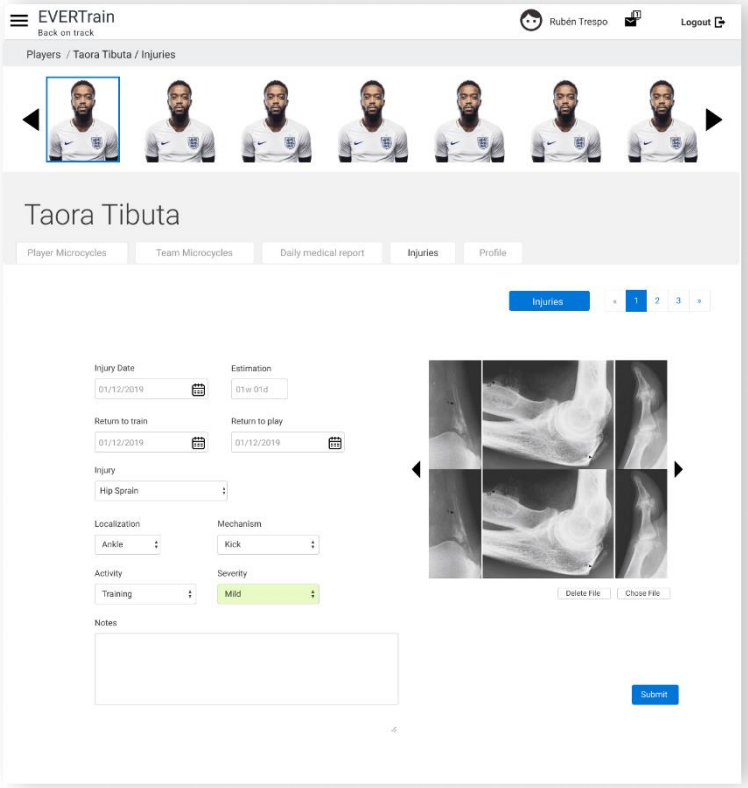

EVERTrain
Back on track
Rubén Trespo
Logout
Players / Taora Tibuta / Players Microcycles
Taora Tibuta
Player Microcycles
Team Microcycles
Daily medical report
Injuries
Profile
Delete
+ New
Nº
Starting date
Description
Medical Observations
Action
1 28/10/2019 Copy / Paste Edit
2 04/11/2019 Copy / Paste Edit
3 11/11/2019 Copy / Paste Edit
4 18/11/2019 Copy / Paste Edit
5 25/11/2019 Copy / Paste Edit
6 02/12/2019 Copy / Paste Edit
7 09/12/2019 Copy / Paste Edit
8 16/12/2019 Copy / Paste Edit
9 23/12/2019 Copy / Paste Edit
10 30/12/2019 Copy / Paste Edit
11 06/01/2020 Copy / Paste Edit
12 13/01/2020 Copy / Paste Edit
13 20/01/2020 Copy / Paste Edit
14 27/01/2020 Copy / Paste Edit
15 03/02/2020 Copy / Paste Edit

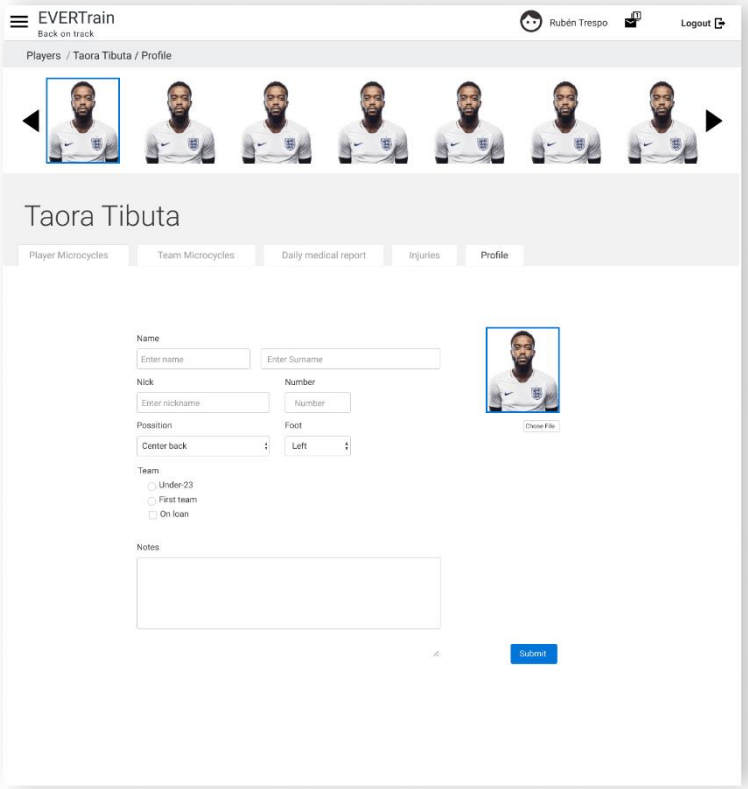

EVERTrain
Back on track
Rubén Trespo
Logout
Players / Taora Tibuta / Team Microcycles
Taora Tibuta
Player Microcycles
Team Microcycles
Daily medical report
Injuries
Profile
Nº
Starting date
Description
1 28/10/2019 View
2 04/11/2019 View
3 11/11/2019 View
4 18/12/2019 View
5 25/12/2019 View
6 02/12/2019 View
7 09/12/2019 View
8 16/12/2019 View
9 23/12/2019 View
10 30/12/2019 View
11 06/01/2019 View
12 13/01/2019 View
13 20/01/2019 View


EVERTrain
Back on track
Rubén Trespo
Logout
Players / Taora Tibuta / Team Microcycles / Micocycle 1
Taora Tibuta
Player Microcycles
Team Microcycles
Daily medical report
Injuries
Profile
Microcycle 1 : 28/10/2019 - 03/11/2019
Team Microcyles
28/10/2019
29/10/2019
30/10/2019
31/10/2019
01/11/2019
02/11/2019
03/11/2019
S R Kg Setup

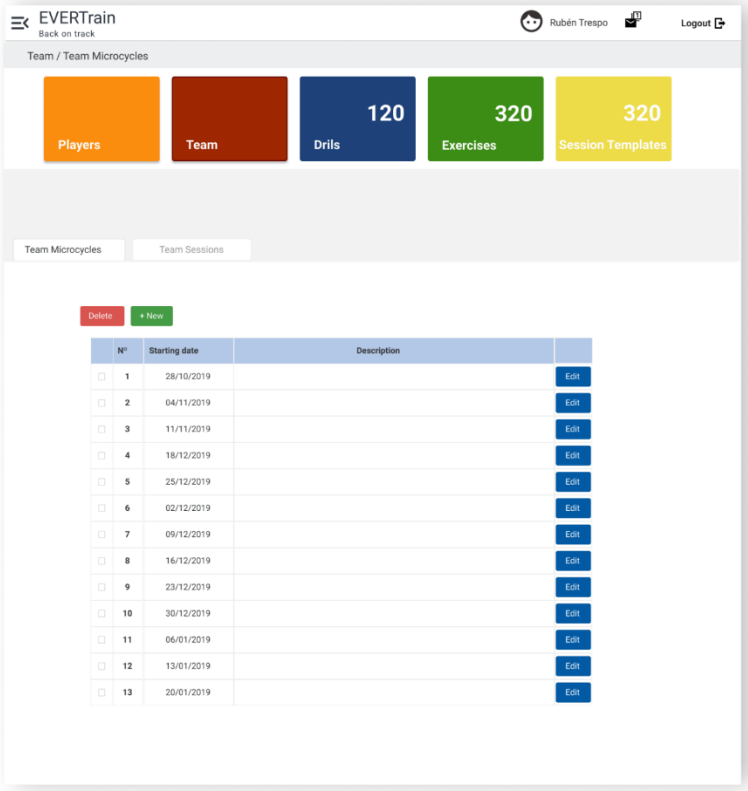

EVERTrain
Back on track
Rubén Trespo
Logout
Players / Taora Tibuta / Daily medical report
Taora Tibuta
Player Microcycles
Team Microcycles
Daily medical report
Injuries
Profile
Delete
+ New
N°
Starting date
Description
1 28/10/2019
2 29/11/2019
3 30/11/2019
4 01/12/2019
5 02/12/2019
6 03/12/2019
7 04/12/2019
8 05/12/2019
9 06/12/2019
10 07/12/2019
11 08/12/2019
12 09/12/2019
13 10/12/2019

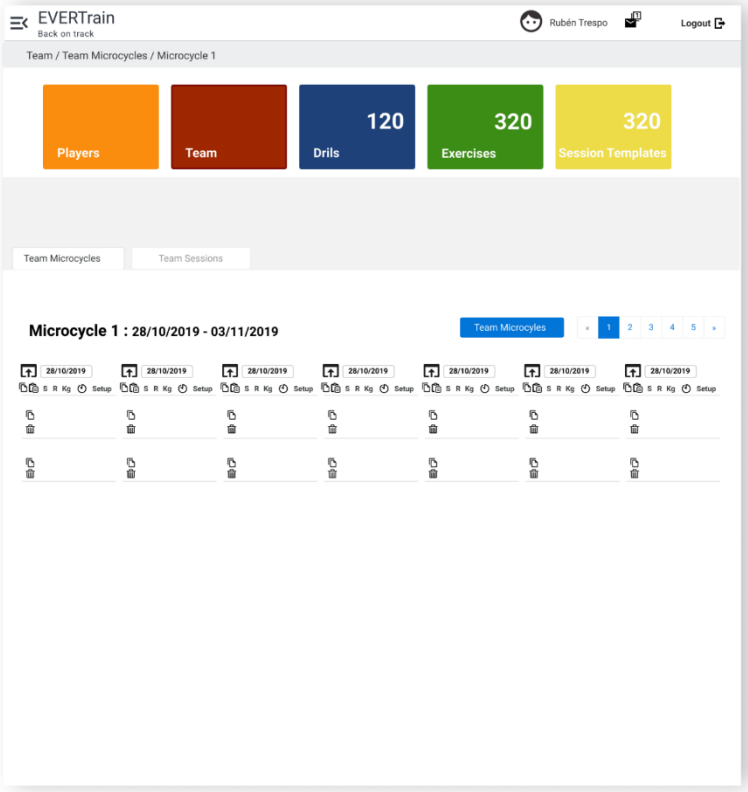

EVERTrain
Back on track
Rubén Trespo
Logout
Players / Taora Tibuta / Injuries
Taora Tibuta
Player Microcycles
Team Microcycles
Daily medical report
Injuries
Profile
Delete
+ New
N°
Date
Days
Injury
Tipology
Localization
Tissue
Mechanism
Activity
Severity
Active
1 22/07/2019 56 Talar dome osteochondral injury Overuse complaints Ankle Cartilage Severe Finish Edit
2 28/10/2019 10 Hip Sprain Mild Ongoing Edit

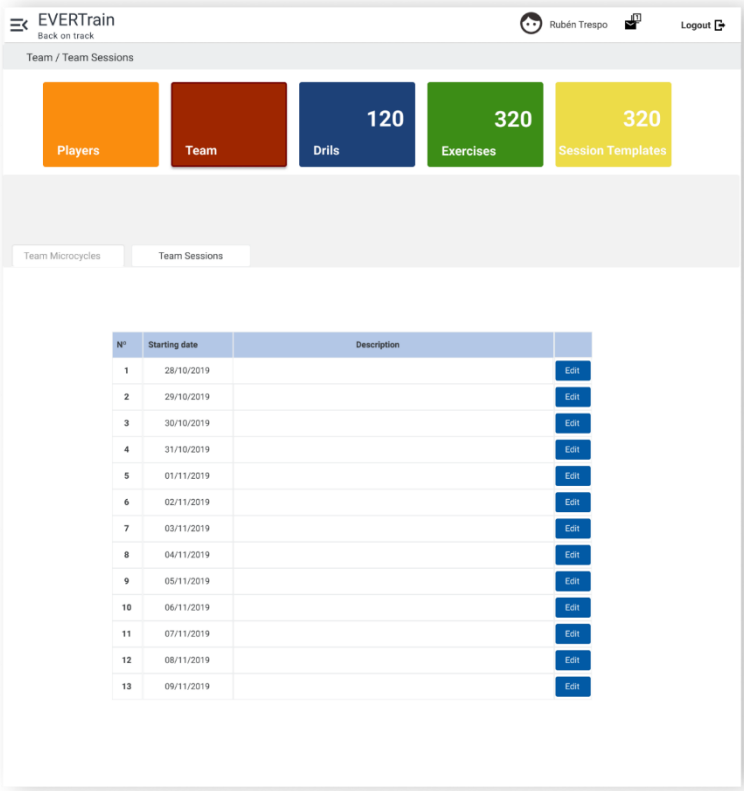

EVERTrain
Back on track
Rubén Trespo
Logout
Players / Taora Tibuta / Injuries
Taora Tibuta
Player Microcycles
Team Microcycles
Daily medical report
Injuries
Profile
Injuries
Injury Date
01/12/2019
Estimation
01w 01d
Return to train
01/12/2019
Return to play
01/12/2019
Injury
Hip Sprain
Localization
Ankle
Mechanism
Kick
Activity
Training
Severity
Mild
Notes
Delete File
Chose File
Submit

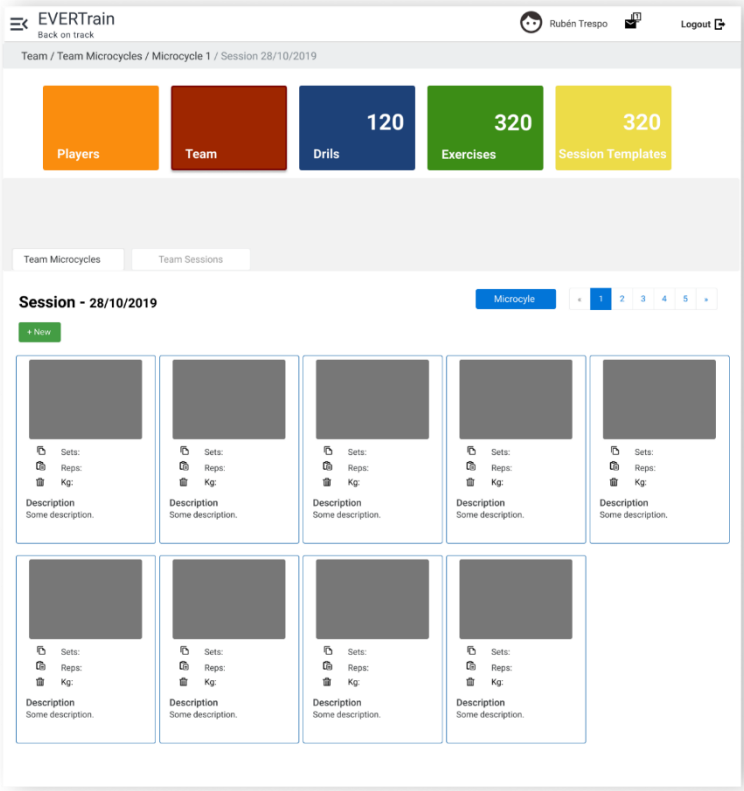

EVERTrain
Back on track
Rubén Trespo
Logout
Players / Taora Tibuta / Profile
Taora Tibuta
Player Microcycles
Team Microcycles
Daily medical report
Injuries
Profile
Name
Enter name
Enter Surname
Nick
Enter nickname
Number
Number
Posaition
Center back
Foot
Left
Team
Under-23
First team
On loan
Notes
Chose File
Submit

Equipo

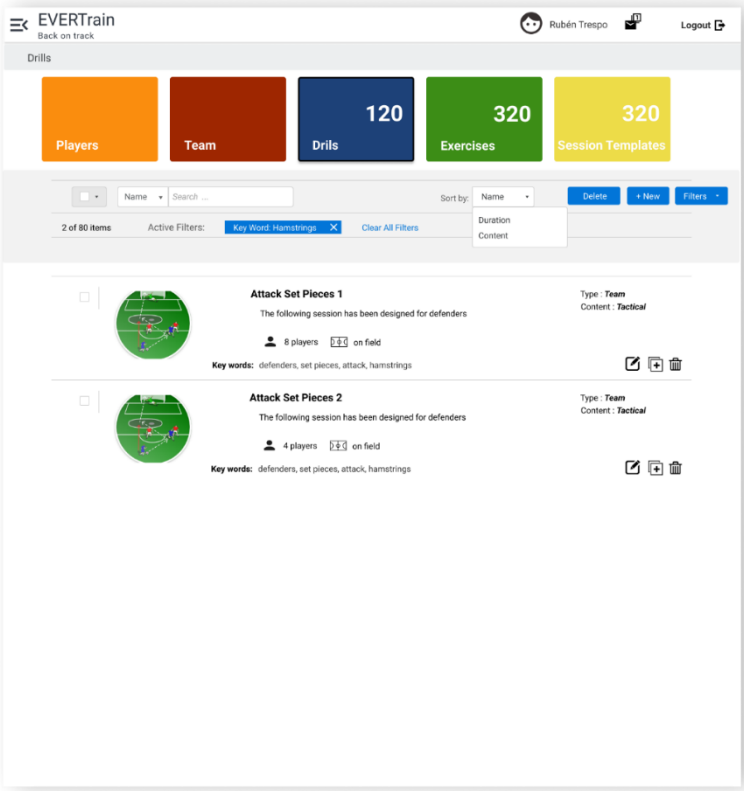

EVERTrain
Back on track
Rubén Trespo
Logout
Team / Team Microcycles
Players
Team
120
Drils
320
Exercises
320
Session Templates
Team Microcycles
Team Sessions
Delete
+ New
N°
Starting date
Description
1 28/10/2019 Edit
2 04/11/2019 Edit
3 11/11/2019 Edit
4 18/12/2019 Edit
5 25/12/2019 Edit
6 02/12/2019 Edit
7 09/12/2019 Edit
8 16/12/2019 Edit
9 23/12/2019 Edit
10 30/12/2019 Edit
11 06/01/2019 Edit
12 13/01/2019 Edit
13 20/01/2019 Edit


EVERTrain
Back on track
Rubén Trespo
Logout
Team / Team Microcycles / Microcycle 1
Players
Team
120
Drils
320
Exercises
320
Session Templates
Team Microcycles
Team Sessions
Microcycle 1 : 28/10/2019 - 03/11/2019
Team Microcycles

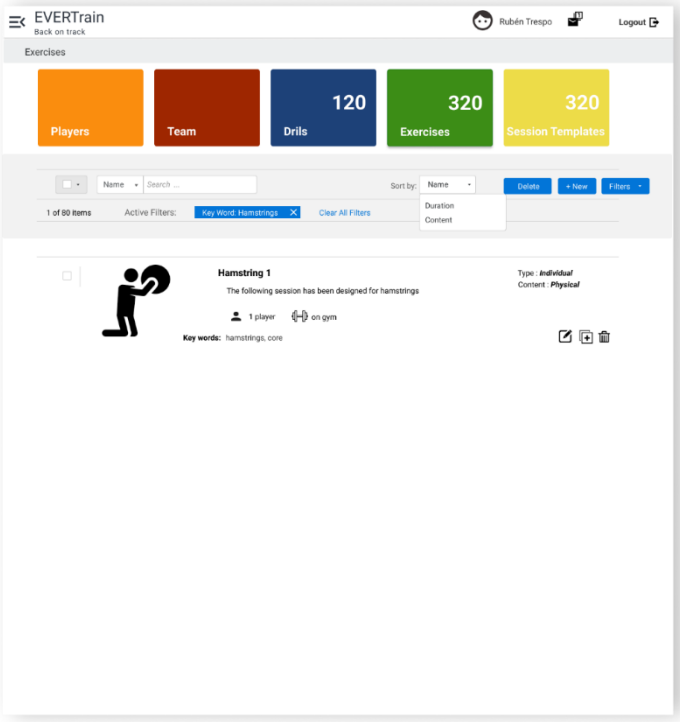

EVERTrain
Back on track
Rubén Trespo
Logout
Team / Team Sessions
Players
Team
120
Drils
320
Exercises
320
Session Templates
Team Microcycles
Team Sessions
Nº
Starting date
Description
1 28/10/2019 Edit
2 29/10/2019 Edit
3 30/10/2019 Edit
4 31/10/2019 Edit
5 01/11/2019 Edit
6 02/11/2019 Edit
7 03/11/2019 Edit
8 04/11/2019 Edit
9 05/11/2019 Edit
10 06/11/2019 Edit
11 07/11/2019 Edit
12 08/11/2019 Edit
13 09/11/2019 Edit


EVERTrain
Back on track
Rubén Trespo
Logout
Team / Team Microcycles / Microcycle 1 / Session 28/10/2019
Players
Team
120
Drils
320
Exercises
320
Session Templates
Team Microcycles
Team Sessions
Session - 28/10/2019
Microcyle
+ New
Sets:
Reps:
Kg:
Description
Some description.

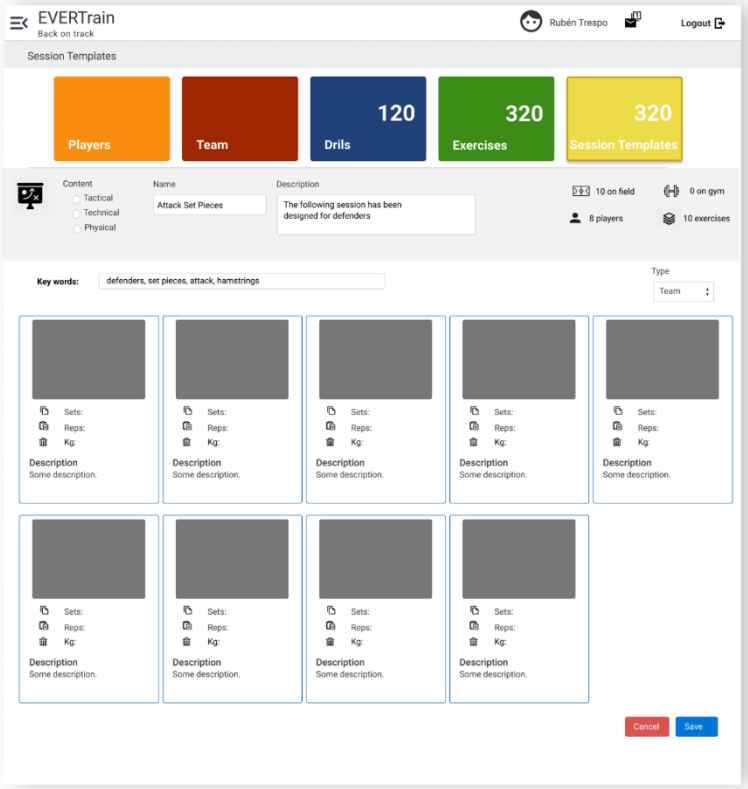
Tareas

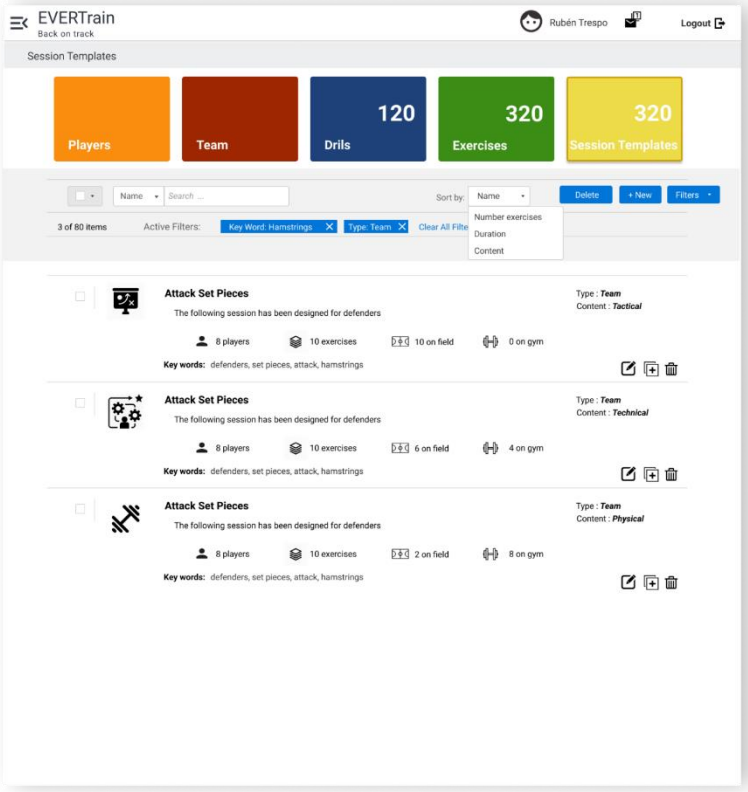


Ejercicios



Plantillas de sesión





8.2. Diagramas de casos de uso individuales

IDENTIFICADOR	NOMBRE
Actor	Actores participantes en el caso de uso
Descripción	Breve descripción del caso de uso
Precondiciones	Condiciones que deben cumplirse para poder ejecutar el caso de uso
Postcondiciones	Condiciones que deben cumplirse al finalizar la ejecución del caso de uso
Escenario básico	Flujo de ejecución del caso de uso

Tabla 1. Plantilla de descripción textual de casos de uso

IDENTIFICADOR	**CU - 01**	NOMBRE	**Iniciar sesión**
Actor	Usuario		
Descripción	El usuario introducirá sus credenciales y la aplicación se encargará de la autenticación. Si los datos del usuario son correctos el servidor responderá afirmativamente y, negativamente en caso contrario.		
Precondiciones	El usuario debe haber sido registrado con anterioridad para poder disponer de las credenciales dentro de la base de datos (nombre de usuario, correo y contraseña) y tener acceso a la aplicación.		
Postcondiciones	El sistema mostrará el nombre del usuario registrado en la interfaz y el resto de los avisos o informes generados irán asociados a dicho usuario.		
Escenario básico	1- Se introducen los datos de usuario: Email y contraseña. 2- Se envían los datos al servidor para que sean procesados. 3- En caso afirmativo se accede al menú principal, y en caso negativo se visualiza un mensaje de error para notificar que el acceso no ha sido posible.		

IDENTIFICADOR	**CU - 02**	NOMBRE	**Cerrar sesión**
Actor	Usuario		
Descripción	El usuario abandonará la aplicación cerrando la sesión que estaba abierta.		
Precondiciones	El usuario debe haber iniciado sesión con anterioridad.		
Postcondiciones	Se mostrará la pantalla de Inicio de sesión y no se tendrá acceso al sistema.		
Escenario básico	1- Desde cualquier vista el usuario selecciona la opción de Cerrar sesión (Logout). 2- El sistema abandona la sesión que estaba abierta y se visualiza la vista de Inicio de sesión.		

IDENTIFICADOR	**CU - 03**	NOMBRE	**Definir temporada**
Actor	Usuario		
Descripción	El usuario añade una nueva temporada al sistema. De esta forma se generarán los microciclos individuales y grupales para toda la plantilla.		
Precondiciones	El usuario debe haber iniciado sesión con anterioridad.		
Postcondiciones	Una vez definido, el usuario tendrá acceso a la plantilla del equipo en la temporada seleccionada con toda la configuración de la temporada.		
Escenario básico	1- Se accede a la aplicación. 2- Se accede a la pantalla de Generación de temporada (New Season). 3- Se introducen el nombre del equipo y el escudo, así como las fechas de inicio y final, los partidos a lo largo de esa temporada y los jugadores de los cuales dispondrá el equipo (en caso de no estar predefinidos, es suficiente con el nombre). 4- El software genera un equipo nuevo y le asocia los jugadores con los respectivos microciclos individuales y grupales.		

IDENTIFICADOR	**CU - 04**	NOMBRE	**Gestión de ejercicios**
Actor	Usuario		
Descripción	El usuario añade un nuevo ejercicio a la base de datos.		
Precondiciones	El usuario debe haber iniciado sesión con anterioridad.		
Postcondiciones	El usuario tendrá acceso al ejercicio desde la pantalla de generación de plantillas o de generación de sesiones específicas dentro de un microciclo.		
Escenario básico	1- Se accede a la aplicación. 2- Desde el menú principal se accede al apartado Ejercicios (Exercises). 3- Se selecciona la opción de añadir ejercicio. 4- Se define un nuevo ejercicio: nombre, imagen, descripción, tipo de ejercicio (ejercicio o tarea), número de jugadores, localización, ... 5- Se guarda el ejercicio. 6- La aplicación vuelve a la pantalla de Ejercicios y muestra el ejercicio en la lista.		

IDENTIFICADOR	**CU - 05**	NOMBRE	**Gestión de plantillas de sesión**
Actor	Usuario		
Descripción	El usuario añade una nueva plantilla de sesión a la base de datos con el afán de agilizar la planificación de las progresiones de recuperación o simplemente de tener sesiones genéricas de entrenamiento.		

Precondiciones	El usuario debe haber iniciado sesión con anterioridad.
Postcondiciones	El usuario tendrá acceso a la plantilla de sesión desde la vista de edición de microciclo.
Escenario básico	1- Se accede a la aplicación. 2- Desde el menú principal se accede al apartado de generación de plantillas (Session Templates). 3- Se selecciona la opción de Añadir sesión. 4- Se define una nueva sesión: nombre, descripción, contenido, tipo, palabras clave ... 5- Se definen los ejercicios de la sesión con el número de repeticiones, series y carga. 6- Se guarda la sesión. 7- La aplicación vuelve a la pantalla de Plantillas de sesión y muestra la sesión en la lista.

IDENTIFICADOR	**CU - 06**	NOMBRE	**Gestión de jugador**
Actor	Usuario		
Descripción	El usuario añade (o modifica) un jugador al equipo.		
Precondiciones	El usuario debe haber iniciado sesión con anterioridad. A su vez, debe haber seleccionado la temporada y el equipo.		
Postcondiciones	Los datos del jugador introducido se verán actualizados en la lista de jugadores.		
Escenario básico	1- Se accede a la aplicación. 2- Una vez seleccionado el equipo y la temporada, desde el menú principal se accede al apartado de Jugadores (Players). 3- Se selecciona la opción de añadir jugador o se selecciona el jugador a editar. 4- Se accede al perfil del jugador. 5- Se editan las propiedades del jugador 6- Se guardan los cambios.		

IDENTIFICADOR	**CU - 07**	NOMBRE	**Gestión datos del jugador**
Actor	Usuario		
Descripción	El usuario modifica el perfil de un jugador del equipo.		
Precondiciones	El usuario debe haber iniciado sesión con anterioridad. A su vez, debe haber seleccionado la temporada y el equipo. El usuario debe haber accedido al jugador tal y como se indica en el CU-6 (pasos 1 - 4).		

Postcondiciones	Los datos del jugador introducido se verán actualizados en la lista de jugadores.
Escenario básico	1- Se accede a la aplicación. 2- Una vez seleccionado el equipo y la temporada, desde el menú principal se accede al apartado de Jugadores (Players). 3- Se selecciona el jugado a editar. 4- Se accede al perfil del jugador. 5- Se editan los datos personales del jugador: nombre, alias, número, posición, imagen, … 6- Se guardan los cambios.

IDENTIFICADOR	**CU - 08**	NOMBRE	**Gestión informe lesional del jugador**
Actor	Usuario		
Descripción	El usuario introduce datos lesionales del jugador.		
Precondiciones	El usuario debe haber iniciado sesión con anterioridad. A su vez, debe haber seleccionado la temporada y el equipo. El usuario debe haber accedido al jugador tal y como se indica en el CU-6 (pasos 1 - 4).		
Postcondiciones	Los datos lesionales del jugador deben estar disponibles para exportar. A mayores, el estado lesional debe ser visible dentro de la vista del equipo para identificar los jugadores que requieren sesiones de recuperación.		
Escenario básico	1- Se accede a la aplicación. 2- Una vez seleccionado el equipo y la temporada, desde el menú principal se accede al apartado de Jugadores (Players). 3- Se selecciona el jugado a editar. 4- Se accede al apartado lesional del jugador. 5- Se edita o se añade una nueva lesión: fecha de lesión, vuelta estimada al entrenamiento, tipología, localización, severidad, … 6- Se guardan los cambios.		

IDENTIFICADOR	**CU - 09**	NOMBRE	**Gestión informe médico del jugador**
Actor	Usuario		
Descripción	El usuario introduce datos lesionales del jugador. La finalidad de este dato es el de tener información sobre cómo se siente ese día el jugador.		
Precondiciones	El usuario debe haber iniciado sesión con anterioridad. A su vez, debe haber seleccionado la temporada y el equipo. El usuario debe haber accedido al jugador tal y como se indica en el CU-6 (pasos 1 - 4).		
Postcondiciones	Los datos médicos del jugador deben estar disponibles para exportar.		

Escenario básico	1- Se accede a la aplicación. 2- Una vez seleccionado el equipo y la temporada, desde el menú principal se accede al apartado de Jugadores (Players). 3- Se selecciona el jugado a editar. 4- Se accede al perfil médico del jugador. 5- Se edita o se añade el estado para un día en concreto. 6- Se guardan los cambios.

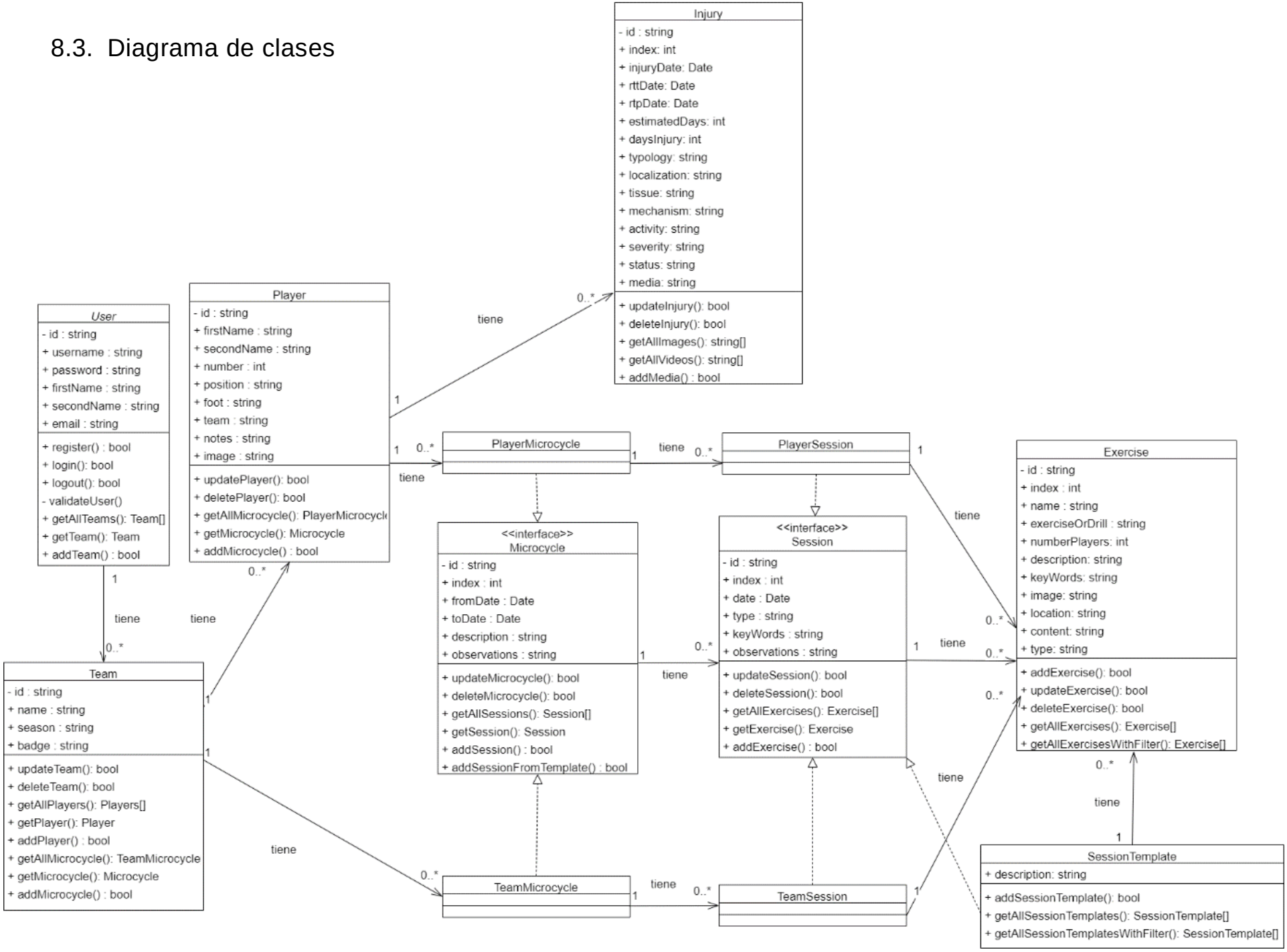
IDENTIFICADOR	**CU - 10**	NOMBRE	**Gestión microciclos individuales**
Actor	Usuario		
Descripción	El usuario diseña el microciclo individual de un jugador.		
Precondiciones	El usuario debe haber iniciado sesión con anterioridad. A su vez, debe haber seleccionado la temporada y el equipo. El usuario debe haber accedido al jugador tal y como se indica en el CU-6 (pasos 1 - 4).		
Postcondiciones	El microciclo está disponible para exportar.		
Escenario básico	1- Se accede a la aplicación. 2- Una vez seleccionado el equipo y la temporada, desde el menú principal se accede al apartado de Jugadores (Players). 3- Se selecciona el jugado a editar. 4- Se accede al apartado de microciclos individuales del jugador. 5- Se selecciona el microciclo. 6- Se diseñan las sesiones diarias del microciclo.		

IDENTIFICADOR	**CU - 11**	NOMBRE	**Gestión microciclos grupales**
Actor	Usuario		
Descripción	El usuario diseña el microciclo grupal del equipo.		
Precondiciones	El usuario debe haber accedido al jugador tal y como se indica en el CU-6 (pasos 1 - 4). De forma alternativa también puede tener acceso a través de la vista del equipo.		
Postcondiciones	El microciclo está disponible para exportar.		

Escenario básico	Opción 1: Acceso a través del jugador. 1- Se accede a la aplicación. 2- Una vez seleccionado el equipo y la temporada, desde el menú principal se accede al apartado de Jugadores (Players). 3- Se selecciona el jugado a editar. 4- Se accede al apartado de microciclos grupales del jugador. 5- Se selecciona el microciclo. 6- Se diseñan las sesiones diarias del microciclo. Opción 2: Acceso a través de la vista del equipo. 1- Se accede a la aplicación. 2- Se selecciona el equipo y la temporada y se accede a la vista del equipo (Team). 3- Se accede al apartado de microciclos. 4- Se diseñan las sesiones diarias grupales.

IDENTIFICADOR	**CU - 12**	NOMBRE	**Gestión equipo**
Actor	Usuario		
Descripción	El usuario edita los datos del equipo		
Precondiciones	El usuario debe haber iniciado sesión con anterioridad. A su vez, debe haber seleccionado la temporada y el equipo.		
Postcondiciones	Los datos introducidos acerca de los jugadores que forman parte del equipo deben estar disponibles a través de las diferentes vistas.		
Escenario básico	1- La gestión del equipo implica directamente la gestión individual de todos sus jugadores. Por lo que implica la realización de la tarea descrita en el caso de uso **CU – 06.**		

8.3. Diagrama de clases



8.4. Estructura de la clase principal de la aplicación

```
import "reflect-metadata";
import express, { Application } from 'express'
import {createConnection} from "typeorm";
import exphbs from 'express-handlebars';
import path from 'path';
import morgan from 'morgan'
import PlayerRoutes from "./routes/player.routes";

export class App {
    app: Application;

    constructor(private port?: number | string) {
        this.app = express();
        this.settings();
        this.middlewares();
        this.initializeControllers();
    }

    private settings() {
        this.app.set('port', this.port || process.env.PORT || 3000);
        this.app.set('views', path.join(__dirname , 'views'));
        this.app.engine('.hbs', exphbs({
            // … view engine … // }));
        this.app.set('view engine', '.hbs');
    }

    private middlewares() {
        this.app.use(morgan('dev'));
        this.app.use(express.json());
        this.app.use(express.urlencoded({extended: false}))
    }

    private initializeControllers(){
        this.app.use(PlayerRoutes);
    }

    async listen(): Promise<void> {
        await this.app.listen(this.app.get('port'));
        console.log('Server on port', this.app.get('port'));
    }

    public databaseConnection() {
        createConnection().then(async connection => {
            console.log("Now, lets go.");
        }).catch(error => console.log(error));
    }
}
```

Ejemplo del archivo app.ts

8.5. Configuración Inicial del proyecto

Para utilizar TypeORM en nuestra aplicación es necesario realizar algunos pasos e instalar algunas dependencias:

1- Instalar TypeORM de forma global.

```
npm install typeorm -g
```

2- Inicializar el proyecto de TypeORM. Esto creará un archivo llamado package.json en la carpeta de la aplicación EverTrain junto con un esqueleto de la aplicación. Además, con este comando ya se indica a la aplicación la base de datos a utilizar (mysql).

```
typeorm init --name EverTrain --database mysql
```

3- Añadir Typescript y TS-Node como dependencias de desarrollo.

```
npm install typescript ts-node --save-dev
```

4- Añadir los paquetes: express y body-parser.

```
npm install express body-parser --save
```

5- Instalar definiciones de tipado para aquellas librerías en las que existan dependencias

```
npm install @types/node @types/express @types/body-parser –save
```

6- Instalar el driver de la base de datos MySQL:

```
npm install mysql --save
```

7- Instalar reflect-metadata. Permite la reflexión de código: capacidad de un programa de examinar y modificar su propia estructura en tiempo de ejecución. Dado que TypeORM trabaja con decoradores (como @Entity o @Column) para definir la estructura y los tipos de los datos que almacenan las tablas, este paquete es usado para parsear esos decoradores y usarlos para construir peticiones sql.

```
npm install reflect-metadata --save
```

8.6. Archivo tsconfig.json

```
{
   "compilerOptions": {
     /* Basic Options */
     "target": "es6”,           /* Specify ECMAScript target version: 'ES3' (defau
lt), 'ES5', 'ES2015', 'ES2016', 'ES2017', 'ES2018', 'ES2019' or 'ESNEXT'.*/
     "module": "commonjs”,      /* Specify module code generation: 'none', 'common
js', 'amd', 'system', 'umd', 'es2015', or 'ESNext'.*/
      "sourceMap": true,        /* Generates corresponding '.map' file.*/
      "outDir": "./build",      /* Redirect output structure to the directory.*/
     /* Strict Type-Checking Options */
      "strict": true,           /* Enable all strict type-checking options.*/
      "strictPropertyInitialization": false,  /* Enable strict checking of propert
y initialization in classes.*/
     /* Module Resolution Options */
      "moduleResolution": "node",           /* Specify module resolution strategy
: 'node' (Node.js) or 'classic' (TypeScript pre-1.6).*/
      "esModuleInterop": true,              /* Enables emit interoperability betw
een CommonJS and ES Modules via creation of namespace objects for all imports. Imp
lies 'allowSyntheticDefaultImports'.*/
     /* Experimental Options */
       "experimentalDecorators": true,        /* Enables experimental support for
ES7 decorators. */
       "emitDecoratorMetadata": true          /* Enables experimental support for
emitting type metadata for decorators.*/
   },
   "include": ["src/**/*"],
   "exclude": ["node_modules"]
 }
```

8.7. Ejemplo de la entidad Player

```
import { Entity, Column, PrimaryGeneratedColumn } from "typeorm";
/* …. */
@Entity("Player")
export class PlayerEntity {

    @PrimaryGeneratedColumn({ name: "id", type: "smallint" })
    id: number;

    @Column({ name: "nick", length: 100 })
    nick: string;

    @Column({ name: "firstName", length: 100 })
    firstName: string;

    @Column({ name: "secondName", length: 100 })
    secondName: string;

    @Column({ name: "number", type: "smallint" })
    number: number;

    @Column({ name: "role", type: "enum", enum: PlayerRole,
default: PlayerRole.JOKER })
    role: PlayerRole

    @Column({ name: "foot", type: "enum", enum: Foot, default: Foot.RIGHT })
    foot: Foot

    @Column({ name: "team", type: "enum", enum: Team, default: Team.FIRST })
    team: Team

    @Column({ name: "notes", length: 1000 })
    notes: string;

    @Column({ name: "imageURL", length: 500 })
    imageURL: string;
/* … */
}
```

8.8. Ejemplo de relación 1:N con TypeORM

```
import { /* ... Others ...*/ OneToMany } from "typeorm";
import { PlayerMicrocycleEntity } from "./PlayerMicrocycle";

@Entity("Player")
export class PlayerEntity {

/* … Columns …*/

    @OneToMany(type => PlayerMicrocycleEntity, PlayerMicrocycle => PlayerMicro
cycle.player)
    playerMicrocycle: PlayerMicrocycleEntity;
}
```

```
import { /* ... Others ...*/ ManyToOne, JoinColumn } from "typeorm";
import { PlayerEntity } from "./Player";

@Entity("PlayerMicrocycle")
export class PlayerMicrocycleEntity {

/* ... Columns ...*/

    @ManyToOne(type => PlayerEntity, Player => Player.playerMicrocycle)
    @JoinColumn()
    player: PlayerEntity;
}
```

8.9. Ejemplo del controlador PlayerController

```
import { Request, Response } from 'express';
import {getConnection} from "typeorm";
import {PlayerEntity as Player} from "../entity/Player";

class PlayerController {

    public async createPlayer(req: Request, res: Response) {
        /* … */
        let repository = getConnection().getRepository(Player);
        await repository.save(newPlayer);
        res.redirect("/");

    }

    public async findAllPlayers(req: Request, res: Response) {

        /* … */
        let repository  = getConnection().getRepository(Player);
        const playerData = await repository.find();
        res.render("index", {players: playerData});

    }

    public async findPlayer(req: Request, res: Response) {
        /* … */
        let repository = getConnection().getRepository(Player);
        let playerData = await repository.findOne({where: {"id" : idPlayer}});
        res.render("edit", {player: playerData});
    }

    public async updatePlayer(req: Request, res: Response) {
        /* … */
        let repository = getConnection().getRepository(Player);
        await repository.update(idPlayer, newPlayer);
        res.redirect("/");
    }

    public async deletePlayer(req: Request, res: Response) {
        /* … */
        let repository = getConnection().getRepository(Player);
        await repository.delete(idPlayer);
        res.redirect("/");
    }
    /* … other methods … */
}
export default new PlayerController();
```

8.10. Instalación de Handlebars

Para utilizar Handlebars es muy sencillo. Tan solo basta con instalarlo desde NPM.

```
npm install express-handlebars --save
```

Y a mayores, hemos de configurar express para indicar en que directorios se localizarán los diferentes componentes. Por defecto las páginas (pages) se encuentran en el directorio de views, por lo que no hace falta indicarlo:

```
        this.app.engine('.hbs', exphbs({
            extname: '.hbs',
            defaultLayout: 'main',
            layoutsDir: path.join(this.app.get('views'), 'layouts'),
            partialsDir: path.join(this.app.get('views'), 'partials'),
            helpers: require('./lib/helpers')
        }));
```

Configuración de Handlebars

8.11. Paquetes útiles para desarrollar una PWA

Algunos de los paquetes npm aquí mencionados han sido ya incluidos (aunque no mencionados) en las etapas previas de implementación. Sin embargo, y dado que pueden ser de gran utilidad a la hora de depurar el código que se ha de implementar y que nos ayudarán a satisfacer las necesidades y objetivos, se ha decidido realizar mención especial al inicio de este apartado.

1- Instalar nodemon: Este paquete de node permanece se encarga de actualizar la aplicación sin necesidad de pararla y arrancarla en caso de que se realicen cambios en los ficheros del proyecto. Si no se tiene instalado previamente, en este apartado será de gran ayuda.

```
npm install nodemon --save
```

Debemos asegurarnos de que en el archivo: package.json estamos iniciando nuestra instancia con nodemon:

```
"scripts": {
    "build": "tsc",
    "dev": "nodemon src/index.ts --exec ts-node",
    "start": "ts-node src/index.ts"
},
```

2- Instalar Morgan: Paquete utilizado para registrar detalles de solicitudes realizadas por los controladores. En la consola en la que se ejecuta la aplicación se puede ver el registro de las peticiones y la obtención de datos de la base de datos.

```
npm install morgan --save

npm install @types/morgan --save
```

NOTA: En esta sección se han mencionado algunos de los módulos npm de más relevancia que han sido utilizados para poder explicar la estructura básica de la aplicación. Sin embargo, a la hora de desarrollar el producto completo será necesario instalar otros paquetes aquí no mencionados.

---

9. BIBLIOGRAFÍA

Actimet. (2020). Retrieved from https://www.actimet.com/what-it-offers/

Ade, J., Fitzpatrick, J., & Bradley, P. S. (2016). High-intensity efforts in elite soccer matches and associated movement patterns, technical skills and tactical actions. Information for position-specific training drills. *Journal of Sports Sciences*, *34*(24), 2205–2214. https://doi.org/10.1080/02640414.2016.1217343

Angular. (2019). Angular - What is Angular?

Ater, T. (2017). *Building Progressive Web Apps: bringing the power of native to the browser*. O'Reilly Media, Inc.

Athlete monitoring. (2020). Retrieved April 18, 2020, from https://www.athletemonitoring.com/

Bashir, S., & Qureshi, R. J. (2012). Hybrid Software Development Approach for Small to Medium Scale Projects: RUP XP & Scrum. *Sci.Int. (Lahore)*, *24*(4), 381–384. Retrieved from http://www.sci-int.com/pdf/197094943510-Integration of XP-RUP-Scrum Rizwan Jamil -S381-384.pdf

Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding TypeScript. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-662-44202-9_11

bin Uzayr, S., Cloud, N., Ambler, T., bin Uzayr, S., Cloud, N., & Ambler, T. (2019). Vue.js. In *JavaScript Frameworks for Modern Web Development*. https://doi.org/10.1007/978-1-4842-4995-6_14

Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T.-M. (2017). Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. *Proceedings of the 13th International Conference on Web Information Systems and Technologies*, (Webist), 344–351. https://doi.org/10.5220/0006353703440351

Byun, H., Chiu, W., & Bae, J. S. (2018). Exploring the adoption of sports brand apps: An application of the modified technology acceptance model. *International Journal of Asian Business and Information Management*, *9*(1), 52–65. https://doi.org/10.4018/IJABIM.2018010105

CoachMePlus. (2020). Retrieved April 18, 2020, from https://coachmeplus.com/elite-sports-applied-sports-science/

Cohn, M. (2010). *Succeeding with Agile. Software development using Scrum*. Retrieved from www.XProgramming.com

Crispin, L., Gregory, J., Lead, A. P., Mentor, O., & Services, I. K. (2009). *Agile Testing: A Practical Guide for Testers and Agile Teams*. *Vasa*. https://doi.org/10.1007/s13398-014-0173-7.2

Doglio, F. (2015). *Pro REST API Development with Node.js*. *Pro REST API Development with Node.js*. https://doi.org/10.1007/978-1-4842-0917-2

Electrons, B. V. (2018). Progressive web apps [bits versus electrons]. *IEEE Consumer Electronics Magazine*, *7*(2), 106--117. https://doi.org/10.1109/MCE.2017.2776463

Elliott, E. (2014). *JavaScript Applications: Robust Web Architecture With Node, HTML5, and Modern JS Libraries*. Retrieved from http://oreilly.com/catalog/errata.csp?isbn=9781491950296 for

Fusion Sport. (2020). Retrieved April 18, 2020, from https://www.fusionsport.com/smartabase-integrations/

Gackenheimer, C. (2015). *Introduction to React*. *Introduction to React*. https://doi.org/10.1007/978-1-4842-1245-5

Gregory, J., & Crispin, L. (2014). *More agile testing: learning journeys for the whole team*. Addison-Wesley Professional.

Hahn, E. H. (2016). *Express in Action: Writing, building, and testing Node.js applications*.

Hesterberg, T. (2011). Bootstrap. *Wiley Interdisciplinary Reviews: Computational Statistics*. https://doi.org/10.1002/wics.182

Hume, D. A. (2017). *Progressive web apps*. Manning Publications Co.

Introducción a Express/Node. (2020). Retrieved June 15, 2020, from https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

Khriyenko, O. (2015). Semantic UI: Automated Creation of Semantically Personalized User Interface. *GSTF Journal on Computing (JoC)*. https://doi.org/10.7603/s40601-014-0016-6

Krol, J., Mithun, S., & D'mello, B. (2014). *Web Development with MongoDB and Node.js*. Retrieved from https://books.google.com/books?id=hZifBAAAQBAJ&pgis=1

Kruchten, P. (2000). The Rational Unified Process--An Introduction (p. 255).

Lago-Peñas, C. (2013). *¿Por qué? Verdades y mitos sobre el rendimiento en el fútbol. Verdades y mitos sobre el rendimiento en el fútbol.* (INDE, Ed.).

Lago, C., Casais, L., Dominguez, E., & Sampaio, J. (2010). The effects of situational variables on distance covered at various speeds in elite soccer. *European Journal of Sport Science*,

10(2), 103–109. https://doi.org/10.1080/17461390903273994

M. Lerner, R. (2011). At the forge: Mustache.js. *Linux Journal*, (210).

Majchrzak, T. A., Biørn-Hansen, A., & Grønli, T.-M. (2018). Progressive Web Apps: the Definite Approach to Cross-Platform Development? (pp. 5735–5744). https://doi.org/10.24251/HICSS.2018.718

Manricks, G. (2013). *Instant Handlebars.js*.

Mardan, A. (2014). *Express.js Guide: The Comprehensive Book on Express.js*. (CreateSpace Independent Publishing Platform, Ed.).

Mardan, Azat. (2014). Template Engines and Consolidate.js. In *Pro Express.js* (pp. 75–85). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-0037-7_5

Mardan, Azat. (2018). *Practical Node.js*. Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-3039-8

Miclea, L., Stoian, I., Enyedi, S., IEEE Computer Society. Technical Council on Test Technology, & Institute of Electrical and Electronics Engineers. (2018). 2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR) : THETA 21st edition : 24th-26th May, Cluj-Napoca, Romania : proceedings. *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, 1–5.

Miñano-Espin, J., Casáis, L., Lago-Peñas, C., & Gómez-Ruano, M. Á. (2017). High Speed Running and Sprinting Profiles of Elite Soccer Players. *Journal of Human Kinetics*, *58*(1), 169–176. https://doi.org/10.1515/hukin-2017-0086

Node js. (2020). Retrieved March 21, 2020, from http://nodejs.org/

Rozentals, N. (2017). *Mastering TypeScript: build enterprise-ready, industrial strength web applications using TypeScript and leading JavaScript frameworks*.

Santos, P., Lago-peñas, C., García-garcía, O., Santos, P., & Lago-peñas, C. (2017). The influence of situational variables on defensive positioning in professional soccer positioning in professional soccer. *International Journal of Performance Analysis in Sport*, *8668*(July), 1–8. https://doi.org/10.1080/24748668.2017.1331571

Sarmento, H., Figueiredo, A., Lago-Peñas, C., Milanovic, Z., Barbosa, A., Tadeu, P., & Bradley, P. S. (2017). The Influence of Tactical and Situational Variables on Offensive Sequences during Elite Football Matches. *Journal of Strength and Conditioning Research*, 1. https://doi.org/10.1519/JSC.0000000000002147

Soccer SystemPro. (2020). Retrieved April 18, 2020, from https://soccersystempro.com/

Somoza, J. K. (2019). *Diseño y desarrollo de un juego de puzles como Aplicación Web Progresiva (PWA)*. Universitat Oberta de Catalunya.

TypeORM. (2020). Retrieved May 23, 2020, from https://typeorm.io/#/

Wohlgethan, E. (2018). *Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js*. HAmburg University of Applied Sciences.