UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación web SinglePage usando el Framework MeanStack para el control de presencia en un polideportivo

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Víctor Martínez Palomares

Tutor: José Vicente Busquets Mataix

Curso 2018-2019

Resum

Desenvolupament d'una aplicació web per a un poliesportiu que aglutina multitud de funcions. Aquestes funcions sent: el control d'accés al centre esportiu, l'accés a la informació del poliesportiu a qualsevol usuari, usuaris els quals poden donar-se d'alta, accedir i modificar les seves dades personals i finalment, el personal del poliesportiu pot administrar els diferents elements del centre. Desenvolupada completament fent ús de les tecnologies encunyades sota el nom de Stack MEAN. El desenvolupament es fa seguint el model del desenvolupament per etapes i pos aèmfasi en totes i cadascuna de les diferents fases.

Paraules clau: Angular, Mongo, Express, Node, Mean Stack, JavaScript, aplicació web, polideportiu, gimnàs, REST

Resumen

Desarrollo de una aplicación web para un polideportivo que aglutina multitud de funciones. Tales funciones siendo: el control de acceso al centro deportivo, el acceso a la información del polideportivo a cualquier usuario, usuarios los cuales pueden darse de alta, acceder y modificar sus datos personales y finalmente, el personal del polideportivo puede administrar los diferentes elementos del centro. Desarrollada por completo haciendo uso de las tecnologías acuñadas bajo el nombre de Stack MEAN. El desarrollo se hace siguiendo el modelo del desarrollo por etapas y pone hincapié en todas y cada una de las diferentes fases.

Palabras clave: Angular, Mongo, Express, Node, Mean Stack, JavaScript, aplicación web, polideportivo, gimnasio, REST

Abstract

Development of a web application for a sports center that brings together a multitude of functions. Such functions being: the access control to the sports center, the access to the information of the sports center to any user, users who can register, access and modify their personal data and finally, the staff of the sports center can manage the different elements of the center. Fully developed using the technologies coined under the name of Stack MEAN. The development is done following the software development life cycle model and emphasizes each and every one of the different phases.

Key words: Angular, Mongo, Express, Node, Mean Stack, JavaScript, web app, sports centre, gym, REST

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

CAPÍTULO 1
Introducción

En la actualidad, gracias al avance de las nuevas tecnologías y el, cada vez más común, uso de dispositivos con conexión a internet, comúnmente llamado *Internet of Things* (IoT) [1] se abre la posibilidad de diseñar aplicaciones con completa funcionalidad que ejecuten parte de su lógica exclusivamente en los navegadores web de los ordenadores cliente.

Más allá de esto, estas aplicaciones pueden aprovechar el antes mencionado *Internet of Things* para interactuar con sistemas externos embebidos que pueden aprovechar para aumentar su potencial y sumar más funcionalidad a estos.

Estas webs se han popularizado mucho en los últimos años [2] dado su gran versatilidad, al ser posible ejecutarlas sin ninguna instalación previa y en multitud de dispositivos diferentes, y dada también a la comodidad que proporciona al usuario ya que en ningún momento se tiene que preocupar de configuraciones ni actualizaciones de la misma. En cambio, esta responsabilidad es tomada por la parte encargada del despliegue de la aplicación.

Este proyecto se centra en el diseño de una aplicación web de una sola página en la que se podrá obtener toda la información relativa a una cadena de centros polideportivos. Más allá, debido a la arquitectura sobre la que funcionará la web, podrá también facilitar información de seguridad y de registro a los sistemas externos encargados de la seguridad de estos centros polideportivos.

Por otro lado, cada vez es más popular el uso de lenguajes de programación como JavaScript dado el soporte que los navegadores actuales proporcionan [3].

El proyecto se apoya en su totalidad en este lenguaje, más particularmente en una *Stack* en la que todos sus componentes son frameworks que funcionan sobre este lenguaje. Esta *Stack* es conocida con el nombre de MEAN siguiendo las siglas de las tecnologías que utiliza.

- M por MongoDB siendo el sistema de base de datos no relacional el cual proporciona la persistencia de los datos.

- E por Express.js. El Framework de Node.js donde se aglutinan todas las herramientas para facilitar la programación de la lógica de negocio en la parte del servidor.

- A por Angular. Framework de JavaScript (TypeScript) usado para agilizar el desarrollo de la interfaz y procesamiento de información en el navegador del cliente.

- N por Node.js. Framework que sirve como plataforma para el funcionamiento del servidor donde corre la aplicación web y que funciona en su totalidad con código JavaScript.

Se ahondará más en estas tecnologías en el capítulo de Implementación.

1.1 Motivación

La razón del desarrollo de este proyecto radica en la voluntad de mejorar tanto el servicio al cliente como la administración del sistema de seguridad de un polideportivo.

Es muy común encontrar escenarios en los que las webs de centros polideportivos están basadas en Hiperenlaces, en las que no se puede desarrollar una funcionalidad suficiente como para que el usuario tenga una adecuada experiencia siendo capaz de realizar el seguimiento de su cuenta y de obtener todo tipo de información sobre el polideportivo.

Sumado a esto, para la administración y control de seguridad son necesarios programas muy pesados que en ocasiones son tediosos de configurar y para los cuales se necesitan equipos de escritorio con suficientes requisitos como para poder ejecutarlas.

El trabajo de este proyecto propone solucionar estos dos problemas con una solución única que serviría tanto para los usuarios como para administradores de un polideportivo y tan solo necesitaría de un ordenador con conexión a internet y un navegador web instalado.

1.2 Objetivos

El objetivo principal del proyecto es el de proponer una aplicación web de una sola página que sea capaz de llevar a cabo las siguientes acciones:

- Permitir que los usuarios de un polideportivo sean capaces, de forma fácil y sin configuraciones, de acceder a una plataforma en la que poder revisar la información relativa al centro y a su suscripción.
- Facilitar el trabajo de administración al personal del polideportivo ofreciendo una plataforma de acceso inmediato donde poder llevar la administración de la información de cada centro y de sus clientes.
- Permitir, de forma adicional, la comunicación con sistemas embebidos externos para proporcionar control de acceso al centro polideportivo.

1.3 Metodología

En esta sección se explica la metodología usada durante el proceso de análisis, planificación y desarrollo de la aplicación web.

La metodología utilizada durante todo el proceso ha sido una metodología ágil, particularmente la llamada *scrum*. La metodología ágil consiste en un proceso iterativo que divide las fases en cortos periodos de tiempo en los cuales se emplea el *feedback* del cliente para mantenerse adaptado a los posibles cambios de requisitos que surgen durante el proceso de desarrollo [4].

Cada iteración tiene como objetivo el desarrollo de nuevas funcionalidades y la entrega continuada versiones del producto funcionales aunque incompletas. Estas entregas periódicas permiten, mediante su análisis y testeo, la incorporación de nuevos requisitos o la redirección de otros en los siguientes periodos de desarrollo.

En la metodología *scrum* [5] se realizan periodos llamados *sprints* con duración entre 1 y 4 semanas durante los cuales se realizan las tareas planificadas al comienzo de dicho periodo, esta planificación se denomina *planning* y consiste en un reunión del equipo con el objetivo de seleccionar las tareas a realizar de entre todas las existentes (*backlog*) según su urgencia y prioridad.

Más allá de esto, todos los días se da una reunión breve del equipo (*daily*) donde cada miembro del equipo explica en lo que ha trabajado el día anterior y comunica sus decisiones y dudas al resto del equipo.

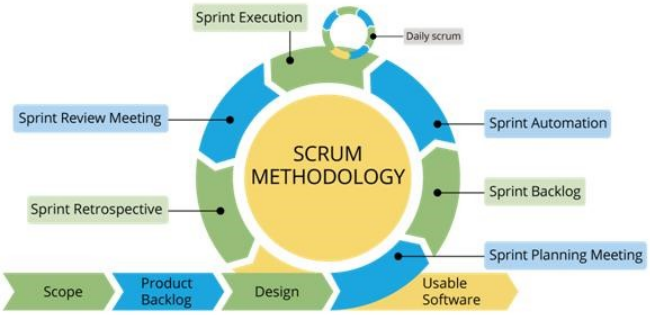

Figura 1.1: Ciclo de desarrollo con metodología *scrum*
Fuente: https://peskytruth.wordpress.com/2018/04/14/scrum-methodology/

1.4 Estructura de la memoria

La estructura que se va a seguir a lo largo del trabajo es la siguiente:

- En primer lugar se hará un análisis de las soluciones en el mercado de productos similares para poder encontrar puntos de diferenciación y potenciales ventajas competitivas. Se llevará a cabo un análisis cuantitativo de las funcionalidades.
- En segundo lugar se hará un desarrollo del alcance del producto, de su funcionalidad y también se listarán tanto los requisitos funcionales como los no funcionales.
- A continuación se describirán las etapas de la creación de un software, que son:

  - Análisis. En la cual se analizaran los casos de uso y flujos de actividad haciendo uso de diagramas.
  - Diseño. Donde se mostrará las decisiones de arquitectura y de interfaz tomadas para el desarrollo del producto.
  - Implementación. Aquí se desarrollarán las tecnologías utilizadas para el desarrollo del producto y el entorno elegido para esto. Además se ilustrará la estructura del mismo y se explicarán algoritmos importantes utilizados en su programación.
  - Pruebas. Contemplando la última fase de desarrollo, se explicarán las diferentes tecnologías utilizadas para las pruebas y verificación del correcto funcionamiento del producto.

- Finalmente se expondrán los resultados y las conclusiones obtenidas sobre la solución software desarrollada, así como una pequeña reflexión sobre las posibles mejoras a futuro.

  Cabe destacar que al final de la memoria, antes de los anexos, se encuentra un glosario con una breve explicación de cada uno de los tecnicismos utilizados durante la redacción de la misma.

CAPÍTULO 2
Estado del arte

2.1 Introducción

En los últimos años el uso de centros polideportivos para la realización de actividades deportivas ha crecido de forma sustancial. [6] Junto a esto, la oferta de gimnasios ha seguido un aumento continuo, dada esta gran competitividad, muchos de ellos utilizan páginas web para darse y conocer y como toma de contacto con los potenciales usuarios.

Entre estas webs, podemos encontrar de todos los tipos. Desde webs tradicionales con Hipertexto hasta las más sofisticadas aplicaciones webs que no solo permiten obtener información sobre el centro deportivo sino que además permiten al usuario llevar control sobre su membresía e incluso hacer reservas sobre diferentes actividades.

A continuación se va a explicar brevemente las principales diferencias entre webs tradicionales basadas en Hiperenlaces ó estáticas y las webs dínamicas.

2.1.1. Web estática y web dinámica

Una página web se trata nada más y nada menos que de un archivo en formato HTML (*Hypertext Markup Language*) que es enviado por un servidor y más tarde es renderizado por el navegador web. Este HTML puede contener texto y Hiperenlaces (enlaces a otras páginas web HTML)[8].

Las páginas web estáticas deben su nombre a la ausencia de movimiento en ellas, en su mayoría se tratan tan solo de HTML simple con texto e imágenes, podrían considerarse equivalentes a un documento. Los servidores que sirven esté tipo de webs suelen tener multitud de archivos HTML los cuales van siendo enviados al navegador del usuario durante el proceso de navegación. El usuario puede percatarse, porque cada navegación a una vista nueva conlleva un refresco del navegador ya que se carga un documento (HTML) totalmente distinto.

Por otro lado, las páginas web dinámicas, también conocidas como aplicaciones web o páginas *single page* deben estos nombres a su funcionamiento, ya que se comportan como una aplicación estándar de escritorio corriendo en el navegador. Esto sucede de está manera porque el HTML que el servidor envía al navegador

del usuario, además de texto e imágenes, contiene *scripts* que ejecutan acciones en lenguaje JavaScript.

Para el desarrollo de estas páginas web es muy común utilizar diferentes *frameworks* de JavaScript que crean lotes con archivos de código y documentos HTML, de está forma, toda las diferentes vistas de la web son servidas dinámicamente por el programa JavaScript sin necesidad del navegador de refrescar y cargar una nueva página, de ahí la notación *single page*.

2.2 Critica al estado del arte

Este proyecto tiene como objetivo proporcionar un valor diferencial sobre las webs de centros polideportivos existentes. Se pretende, no tan solo ofrecer funcionalidad al usuario, sino que también tiene como objetivo ser útil para el personal administrador del centro.

Para esto se va a hacer un breve análisis de varías web actualmente en el mercado con el objetivo de identificar los puntos positivos y negativos de cada una de ellas.

2.2.1. GoFit

GoFit[1] es una cadena comercial de centros deportivos, siendo su plataforma web una de las más visitadas de entre los centros deportivos españoles.[7]

Su web podría considerarse una web tradicional, no es una aplicación web, ya que cada acción en la web redirecciona a una página HTML por lo que no es *single page*.

La web tiene gran funcionalidad, ya que permite ver la información de actividades, horarios, cursos y precios de cada uno de sus centros. A pesar de esto, como punto negativo, no permite darse de alta ni permite al usuario comprobar información sobre su membresía.

2.2.2. VivaGym

VivaGym[2], también otra cadena con numerosos gimnasios en todas las ciudades importantes de España, tiene una plataforma web para sus usuarios y visitantes.

Al contrario que GoFit, este centro sí que cuenta con una web dinámica *single page* programada con el Framework de JavaScript llamado ***Aurelia***.

Es una web con una interfaz simple pero funcional, además de la posibilidad de obtener información sobre sus cuotas, actividades y gimnasios, cuenta con una parte tan solo visible para sus miembros en la que es posible actualizar los datos

[1] http://www.go-fit.es/
[2] https://www.vivagym.es/

Figura 2.1: Captura de pantalla de GoFit en la vista de actividades
Fuente: http://www.go-fit.es/Horarios/Index

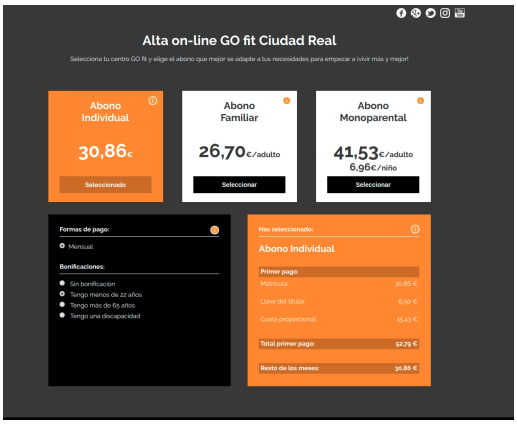

Figura 2.2: Captura de pantalla de GoFit en la vista de cuotas
Fuente: http://www.go-fit.es/Horarios/Index

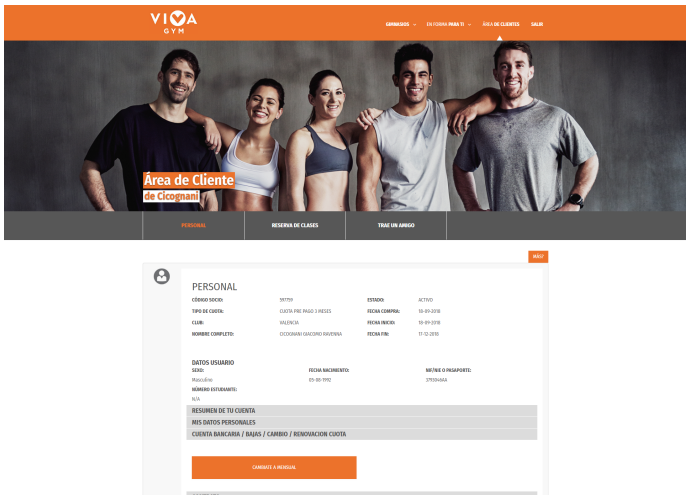

Figura 2.3: Captura de pantalla de VivaGym en la parte reservada a los clientes
Fuente: https://www.vivagym.es/login

y comprobar el estado de la membresía, como se puede observar en la figura 2.3. También permite inscribirse de forma totalmente *online*.

2.2.3. McFit

Como último análisis, se comprueba la página web de McFit[3] una cadena de centros deportivos internacional con gran presencia en España.

Se observa que se trata de una web estática, de la misma forma que GoFit. Está diseñada con *TYPO3* que se trata de una herramienta para el diseño de páginas web que utiliza tecnologías como PHP y TypeScript (subconjunto de JavaScript).

Con respecto a su funcionalidad, encontramos las opciones básicas de información sobre centros y actividades. Como punto adicional, en la figura 2.4 se

[3]https://www.mcfit.com/es

puede observar que tiene una zona exclusiva para los clientes, aunque no permite darse de alta de forma *online*.

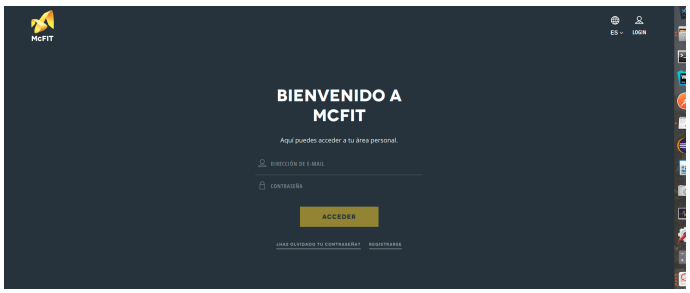

Figura 2.4: Captura de pantalla de McFit de su portal para clientes
Fuente: https://secure.mcfit.com/login

2.2.4. Análisis cuantitativo

En esta sección se realiza un análisis cuantitativo de la tipología y de las funciones consideradas críticas sobre las soluciones similares expuestas en los puntos anteriores.

	GoFit	VivaGym	McFit
Tipo de web	Estática	Dinámica	Estática
Framework JavaScript	No aplicable	Aurelia	No aplicable
Info. centros deportivos	Sí	Sí	Sí
Info. actividades dirigidas	Sí	Sí	Sí
Alta online	No	Sí	No
Acceso usuario	No	Sí	Sí
Administración	No	No	No

Tabla 2.1: Tabla de análisis cuantitativo de soluciones similares

2.3 Propuesta

En los apartados anteriores se ha realizado un análisis de soluciones similares al problema planteado en el proyecto. Se han seleccionado estás plataformas porque son similares entre sí a pesar de que cada una de ellas presenta ciertos elementos característicos.

Como se puede observar en la tabla 2.1, la solución que se plantea en este trabajo sería más similar a VivaGym ya que es la única web de las analizadas que se trata de una web dinámica. A diferencia de esta, se utilizará Angular en lugar de Aurelia como *Framework* de JavaScript ya que a pesar de este primero ser más verboso[9], presenta mucha más funcionalidad y soporte al tener una gran empresa de software como es *Google*.

Con respecto a la funcionalidad, de nuevo VivaGym sería la más similar dado que es la web con más funcionalidad. Permite tanto acceder a datos del usuario como darse de alta de forma totalmente *online*.

Más allá de esto, en la solución a desarrollar se añadirá como punto diferencia la posibilidad de tener un espacio para administrar los diferentes aspectos del centro deportivo y se implementará una interfaz de seguridad que podrá ser accesible por programas embebidos externos.

CAPÍTULO 3
Especificación de requisitos

3.1 Introducción

En esta sección se va a concretar el producto a desarrollar en el proyecto, así como su alcance y su funcionalidad.

A parte de eso, se listarán de forma concisa tanto los requisitos funcionales como los no funcionales.

3.1.1. Alcance del sistema

Se va a desarrollar una aplicación web con el nombre clave de *Iron Squat* (emulando el nombre de una potencial cadena de centros deportivos). Para ello se utilizará las herramientas que proporciona el *Stack* MEAN, basado en su totalidad en el uso de JavaScript como lenguaje de programación tanto en la parte de la interfaz como en la parte del servidor. Más información sobre las tecnologías utilizadas en el capítulo Implementación.

La web permitirá gran variedad de usos. Por un lado, los usuarios podrán acceder a la web y comprobar toda la información correspondiente a los diferentes centros deportivos, las actividades y cursos impartidos en los centros, los horarios de las actividades y multitud más de información.

Por otro lado, un usuario administrador podrá autenticarse en la ventana de acceso y acceder al panel de administración en el que tendrá acceso directo a los datos de toda la base de datos del sistema, teniendo disponible un *CRUD* de los usuarios, centros, actividades y demás.

Por último, el servidor de la propia aplicación, mediante protocolo *REST* pondrá a disposición varios *Endpoints* contra los cuales sistemas externos, como por ejemplo los tornos del centro deportivo, podrán mandar diferentes peticiones de información tales como el permiso de acceso de cierto usuario o la inscripción de un usuario a cierta actividad.

El objetivo del producto es ofrecer una plataforma con la mayor cantidad de potenciales actores que interactúen con ella, por esa razón se aúnan multitud de funcionalidades dentro de un mismo producto. Tanto para los usuario como

los administradores y también sistemas externos. Se ilustrarán todos los posibles actores en el capítulo Análisis.

3.2 Requisitos específicos

3.2.1. Requisitos funcionales

En este apartado se van a listar las funcionalidades más importantes del producto. Las funcionalidades se dividirán según los actores. Cada una de ellas tendrá una referencia que se utilizará en el capítulo de Análisis para la explicación del caso de uso relacionado con cada requisito funcional.

- Usuario no registrado (UNR):
  - Ver todos los centros sobre mapa del mundo (UNR-01).
    El sistema muestra la localización exacta de los centros deportivos sobre un mapa usando la *API* de *Google Maps*.
  - Ver la información detallada sobre un centro (UNR-02).
    El sistema muestra toda la información sobre el centro deportivo que esté seleccionado.
  - Ver la información detallada sobre las cuotas (UNR-03).
    El sistema lista todas las cuotas disponibles y toda su información relevante.
  - Ver la información detallada sobre las actividades disponibles y sus horarios (UNR-04).
    El sistema muestra un listado con la descripción de las actividades y una imagen representativa, así como un calendario con las clases disponibles de cada actividad en la próxima semana.
  - Ver la información detallada sobre los cursos impartidos (UNR-05).
    El sistema muestra un listado con la descripción de los cursos impartidos en el centro deportivo junto a una imagen representativa de este.
  - Solicitar un día de prueba gratis (UNR-06).
    Se permite rellenar un formulario para inscribirse al centro deportivo de forma gratuito para probar.
  - Darse de alta en el centro deportivo (UNR-07).
    Se permite rellenar datos del usuario junto a método de pago para inscribirse al centro deportivo de forma totalmente *online*.
- Usuario registrado (UR):
  - Ver todos los centros sobre mapa del mundo (UR-01).
    El sistema muestra la localización exacta de los centros deportivos sobre un mapa usando la *API* de *Google Maps*.

- Ver la información detallada sobre un centro (UR-02).

  El sistema muestra toda la información sobre el centro deportivo que esté seleccionado.

- Ver la información detallada sobre las cuotas (UR-03).

  El sistema lista todas las cuotas disponibles y toda su información relevante.

- Ver la información detallada sobre las actividades disponibles y sus horarios (UR-04).

  El sistema muestra un listado con la descripción de las actividades y una imagen representativa, así como un calendario con las clases disponibles de cada actividad en la próxima semana.

- Verla información detallada sobre los cursos impartidos (UR-05).

  El sistema muestra un listado con la descripción de los cursos impartidos en el centro deportivo junto a una imagen representativa de este.

- Autenticarse en el sistema (UR-06).

  El usuario es capaz de introducir sus datos y obtener un token de autenticación.

- Ver su información personal (UR-07).

  El sistema muestra los detalles personales del usuario autenticado.

- Modificar sus datos personales (UR-08).

  El sistema permite al usuario cambiar sus datos, excepto el DNI (único) siempre y cuando estos sean correctos.

- Modificar sus datos bancarios (UR-09).

  El sistema permite al usuario cambiar su método de pago, siempre y cuando este sea correcto.

- Renovar su cuota (UR-10).

  El usuario es capaz de seleccionar cual será la próxima cuota que se le recarge.

- Darse de baja del sistema (UR-11).

  El sistema permite, en todo momento, al usuario darse de baja de la aplicación borrando definitivamente todos sus datos.

- Reservar asistencia a actividades dirigidas (UR-12).

  El sistema permite al usuario seleccionar una clase de una actividad durante la semana posterior y reservar su asistencia.

- Cancelar asistencia a actividades dirigidas (UR-13).

  El sistema permite al usuario cancelar cualquier reserva de clases que desee.

- Salir de la aplicación (UR-14).

  El sistema permite al usuario registrado abandonar su perfil y volver a la aplicación web como un usuario no registrado.

- Administrador (A):

- Ver todos los centros sobre mapa del mundo (A-01).
  El sistema muestra la localización exacta de los centros deportivos sobre un mapa usando la *API* de *Google Maps*.
- Ver la información detallada sobre un centro (A-02).
  El sistema muestra toda la información sobre el centro deportivo que esté seleccionado.
- Ver la información detallada sobre las cuotas (A-03).
  El sistema lista todas las cuotas disponibles y toda su información relevante.
- Ver la información detallada sobre las actividades disponibles y sus horarios (A-04).
  El sistema muestra un listado con la descripción de las actividades y una imagen representativa, así como un calendario con las clases disponibles de cada actividad en la próxima semana.
- Verla información detallada sobre los cursos impartidos (A-05).
  El sistema muestra un listado con la descripción de los cursos impartidos en el centro deportivo junto a una imagen representativa de este.
- Autenticarse en el sistema (A-06).
  El administrador es capaz de introducir sus datos y obtener un token de autenticación que le proporciona permisos según su rol.
- Ver los registros en base de datos de los centros, actividades, clases, cursos, cuotas y usuarios del sistema (A-07).
  El administrador podrá ver todos los registros de base de datos, podrá navegar entre las diferentes secciones. Cada sección corresponderá a una colección diferente.
- Añadir registros en base de datos de los centros, actividades, clases, cursos, cuotas y usuarios del sistema (A-08).
  El administrador podrá añadir nuevos registros a la base de datos mediante un formulario que dependerá de la colección seleccionada.
- Editar registros en base de datos de los centros, actividades, clases, cursos, cuotas y usuarios del sistema (A-09).
  El administrador podrá modificar registros existentes en las diferentes colecciones mediante un formulario.
- Eliminar registros en base de datos de los centros, actividades, clases, cursos, cuotas y usuarios del sistema (A-10).
  El administrado podrá borrar registros de las diferentes colecciones de la base de datos. Se permitirá también borrado múltiple.
- Salir de la aplicación (A-11).
  El sistema permite al administrador abandonar su perfil y volver a la aplicación web como un usuario no registrado.

- Sistema externo (SE):

  - Autenticarse en el sistema (SE-01).
    El sistema permitirá enviar una petición REST para autenticarse y devolverá un token con los permisos propios del usuario.

- Obtener lista de usuarios con sus detalles (SE-02).

  El sistema permitirá obtener mediante petición REST los registros de usuarios con sus detalles.

- Obtener detalles de un usuario (SE-03).

  El sistema permitirá obtener mediante petición REST la información detallada de un usuario cuya identificación se recibirá por parámetro de la ruta.

- Obtener lista de actividades dirigidas (SE-04).

  El sistema permitirá obtener mediante petición REST un listado de todas las actividades dirigidas disponibles en el centro deportivo.

- Obtener lista de clases de una actividad dirigida (SE-05).

  El sistema permitirá obtener mediante petición REST los detalles de las clases referentes a una actividad dirigida que se recibirá por parámetro de la ruta.

- Obtener reservas de clases de un usuario (SE-06).

  El sistema permitirá obtener mediante petición REST los detalles de las reservas de un usuario cuya identificación se recibirá por parámetro de la ruta.

- Obtener todas las reservas para un clase de una actividad dirigida (SE-07).

  El sistema permitirá obtener mediante petición REST un listado de todas las reservas de usuarios sobre una clase de una actividad dirigida. La identificación de la clase se recibirá como parámetro de la ruta.

- Comprobar acceso de un usuario al centro deportivo (SE-08).

  El sistema permitirá obtener mediante petición REST comprobar el permiso de acceso de un usuario al centro deportivo. El identificador del usuario se recibirá en la petición como parámetro de la ruta. Se comprobará que el usuario posee en período actual una cuota válida.

- Comprobar acceso de un usuario a una clase de una actividad dirigida (SE-09).

  El sistema permitirá obtener mediante petición REST comprobar el permiso de acceso de un usuario a una clase de una actividad dirigida. Ambos identificadores, del usuario y de la clase, se recibirán como parámetros de la petición respectivamente. Se comprobará si el usuario tiene una reserva válida para esa clase.

3.2.2. Requisitos no funcionales

- Disponibilidad: El sistema debe estar disponible cuando el usuario o los sistemas externos accedan a él. Tanto el servidor REST como el servidor que devuelve la interfaz deben estar desplegados en todo momento. Un posible servidor secundario estará funcionando en caso de que el principal fallará.

- Seguridad: Un usuario anónimo tendrá tan solo acceso a la información pública, mientras que un usuario autenticado tan solo tendrá, de forma adicio-

nal, información sobre su membresía. Por otro lado, el usuario administrador del sistema, bajo previa autenticación, será el único que tenga acceso a toda la información de la base de datos. Más información en la sección Seguridad del capítulo Implementación. Los tokens que reciben los usuarios al autenticarse caducan tras un tiempo previsto. En caso de caducidad, el usuario tendrá que autenticarse de nuevo.

- Fiabilidad: El sistema, tanto en su parte de interfaz como de servidor, será testeado de forma unitaria (caja blanca) para garantizar el correcto funcionamiento de cada uno de sus componentes. De forma adicional, tendrá que cumplir con ciertos test de integración que garantizarán el correcto flujo de información del sistema. Más información en el capítulo Pruebas.

- Mantenibilidad: El software se desarrollara utilizando un control de versiones, en este caso *Git* y siguiendo el patrón de desarrollo *GitFlow* para facilitar el trabajo en equipo y asegurar un software mantenible en el tiempo y una versión estable en todo momento. Más información en la sección Control de versiones del capítulo Implementación.

- Portabilidad: El sistema debe ser accesible en cualquier lugar con el menor número de herramientas posibles. Tanto solo será necesario un dispositivo portátil con acceso a internet y un navegador web ejecutándose.

CAPÍTULO 4
Análisis

4.1 Introducción

En está sección de la memoria, se utilizan diagramas UML para desarrollar los modelos siguiente las especificaciones dadas en el capítulo Especificación de requisitos.

Los diagramas UML son un compuesto de elementos gráficos utilizados para formar diagramas siguiendo un lenguaje con reglas propias. La finalidad de estos diagramas es representar una forma simple de la realidad de la aplicación, describiendo así lo que hará el sistema sin necesidad de poner atención a la implementación del mismo [11].

4.2 Diagrama de clases

Antes de proceder con la explicación de las clases, se aclara el uso de P-KEY y F-KEY en la figura 4.1. Pese a que estas notaciones son propias de una base de datos relacional [12] lo cual no es objetivo de este capítulo y sí lo es de los correspondientes a la fase de Diseño y Implementación, se van a utilizar para ilustrar la diferente naturaleza de los atributos de cada clase.

Por ello, P-KEY, del inglés *primary key*, se referirá al atributo que diferencia de forma única a cada instancia de esa clase. Por otro lado, F-KEY, del ingés *foreign key*, se referirá al atributo de otra clase que a su vez es P-KEY de la misma, por el cual se podrá relacionar una clase con otra.

Como se puede observar en la figura 4.1 se detalla el diseño de las clases que forman el modelo de la aplicación a desarrollar. De forma detallada las clases son las siguientes:

- **Usuario**: cualquier usuario que interactué con el sistema, incluye también a los usuarios administradores y los sistemas externos que anteriormente tendrán que estar dados de alta con su propio usuario. El atributo de 'Rol del usuario' hará la diferenciación entre los múltiples usuarios. Como se puede observar un usuario deberá tener contratada una cuota y también podrá tener indefinidas reservas de clases.

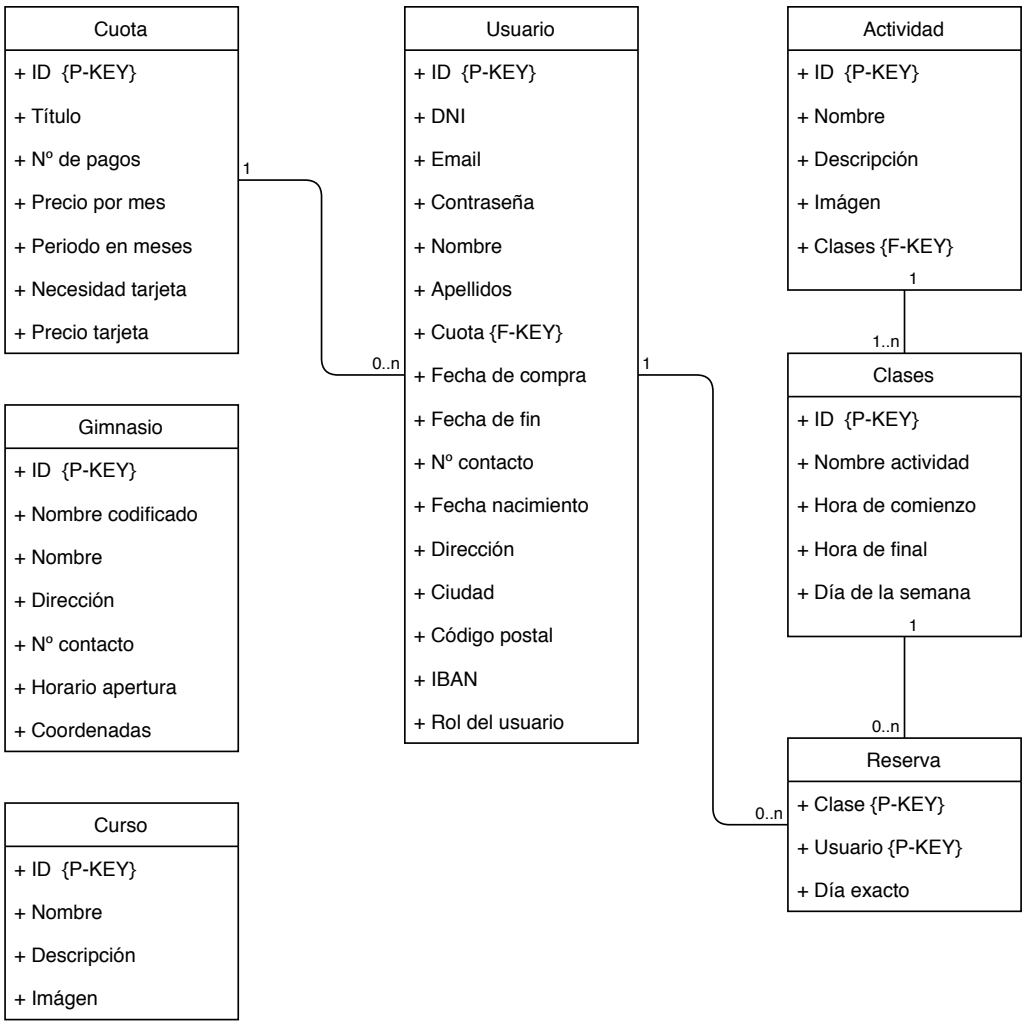

Figura 4.1: Diagrama de clases de la aplicación web

- **Cuota**: diferentes contratos o membresías que los usuarios de los centros deportivos pueden contratar.

- **Gimnasio**: centros deportivos existentes a los que tendrán acceso los clientes.

- **Curso**: cursos disponibles en los diferentes centros deportivos. No existe la posibilidad de reserva ya que funcionarían bajo demanda y disponibilidad.

- **Actividad**: actividades dirigidas disponibles en los centros deportivos. Cada actividad debe tener al menos una clase impartida hasta un número indefinido de ellas.

- **Clase**: clases que están siempre referidas a alguna actividad y de la que pueden haber indefinidas reservas por parte de un usuario del centro deportivo.

- **Reserva**: reservas que sirven para relacionar un usuario con la clase a la que desea asistir y especifica de forma exacta la clase según el día elegido.

4.3 Actores

En los diagramas UML un actor representa los roles que un potencial usuario puede ocupar con respecto a su interacción con el sistema. Estos actores pueden representar roles jugados por seres humanos, hardware externo u otros tipos de entidades [10].

En esta sección se van a definir los diferentes actores que pueden interactuar con el sistema. Ya se han nombrado anteriormente en la sección Requisitos funcionales del capítulo Especificación de requisitos.

Además, en la sección Diagramas de casos de uso se van a mostrar los casos de uso asociados a cada actor.

4.3.1. Listado de actores

- Usuario no registrado. Este actor representara el rol de un usuario anónimo que accede a la aplicación web sin autenticación. Cualquier usuario de la página web será un usuario no registrado hasta el momento en el que se autentique o se registre.

- Usuario registrado. Este actor representa a los usuarios una vez autenticados o registrados en el sistema.

- Administrador. Este actor representa al personal trabajador del centro deportivo en su rol de administrador de la aplicación web.

- Sistema externo. Este actor representa a los diferentes sistemas embebidos o no, independientes de está aplicación web, que interactúen con esta mediante la interfaz REST que esta implementa.

4.4 Diagramas de casos de uso

En esta sección se van a listar y explicar con mayor nivel de detalle todos los casos de uso a tener en cuenta en la etapa de Análisis.

También se mostrarán los diagramas UML de casos de uso correspondientes a cada actor para ilustrar de forma más concisa las posibilidades que brinda el sistema.

Cada uno de los casos de uso consistirá de una referencia a su funcionalidad (listadas en el apartado Requisitos funcionales de la etapa de Especificación de requisitos), una breve descripción de lo que conlleva dicha acción, un conjunto de entradas que el actor tendrá que introducir al sistema, a continuación el proceso que realiza el sistema, un breve flujo de interacción y ,finalmente, el conjunto de salidas que se obtienen por parte del sistema.

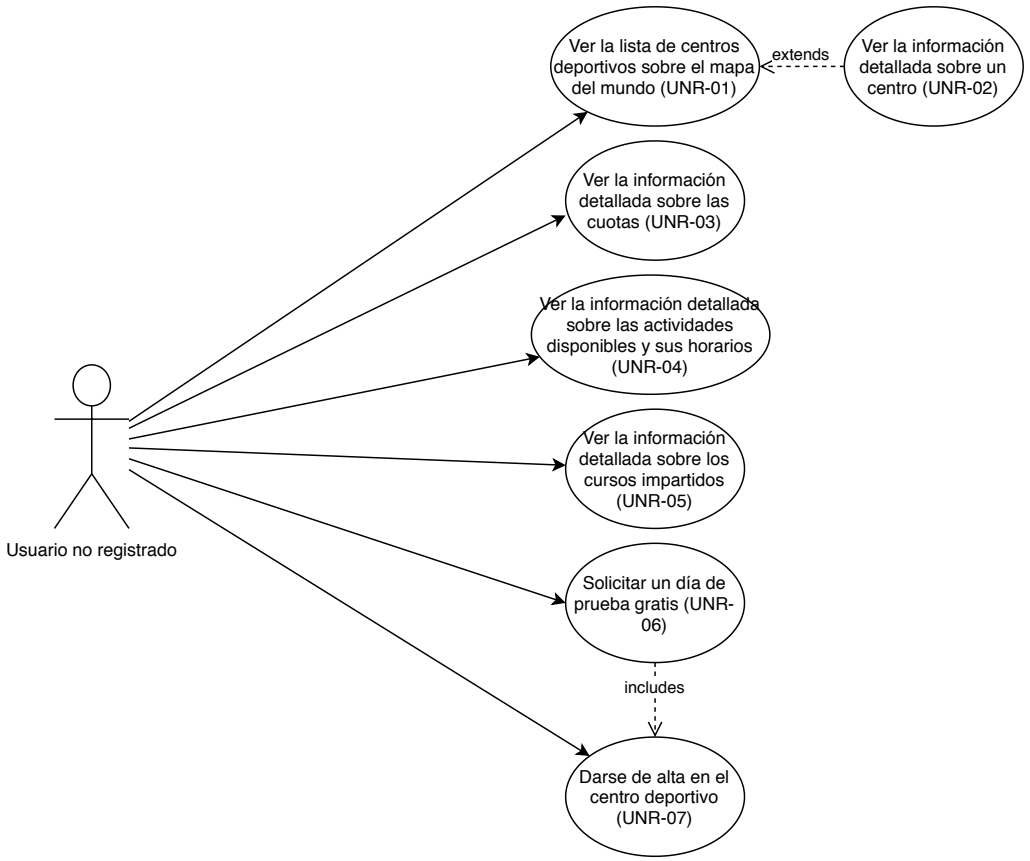

Figura 4.2: Diagrama de casos de uso de usuario no registrado

UNR-01, UR-01, A-01	
Descripción	Ver la lista de centros deportivos sobre el mapa del mundo.
Entradas	2 opciones: ▪ Clic en la pestaña 'CENTROS' en la ventana principal. ▪ Clic en imagen 'Encuentra tu gimnasio' en la ventana principal.
Procesos	Navega al componente de centros deportivos.
Salidas	Muestra la lista de centros deportivos y un mapa de España con los diferentes centros sobre él.

Tabla 4.1: Caso de uso (UNR-01, UR-01, A-01). Ver la lista de centros deportivos sobre el mapa del mundo

Figura 4.3: Diagrama de casos de uso de usuario registrado

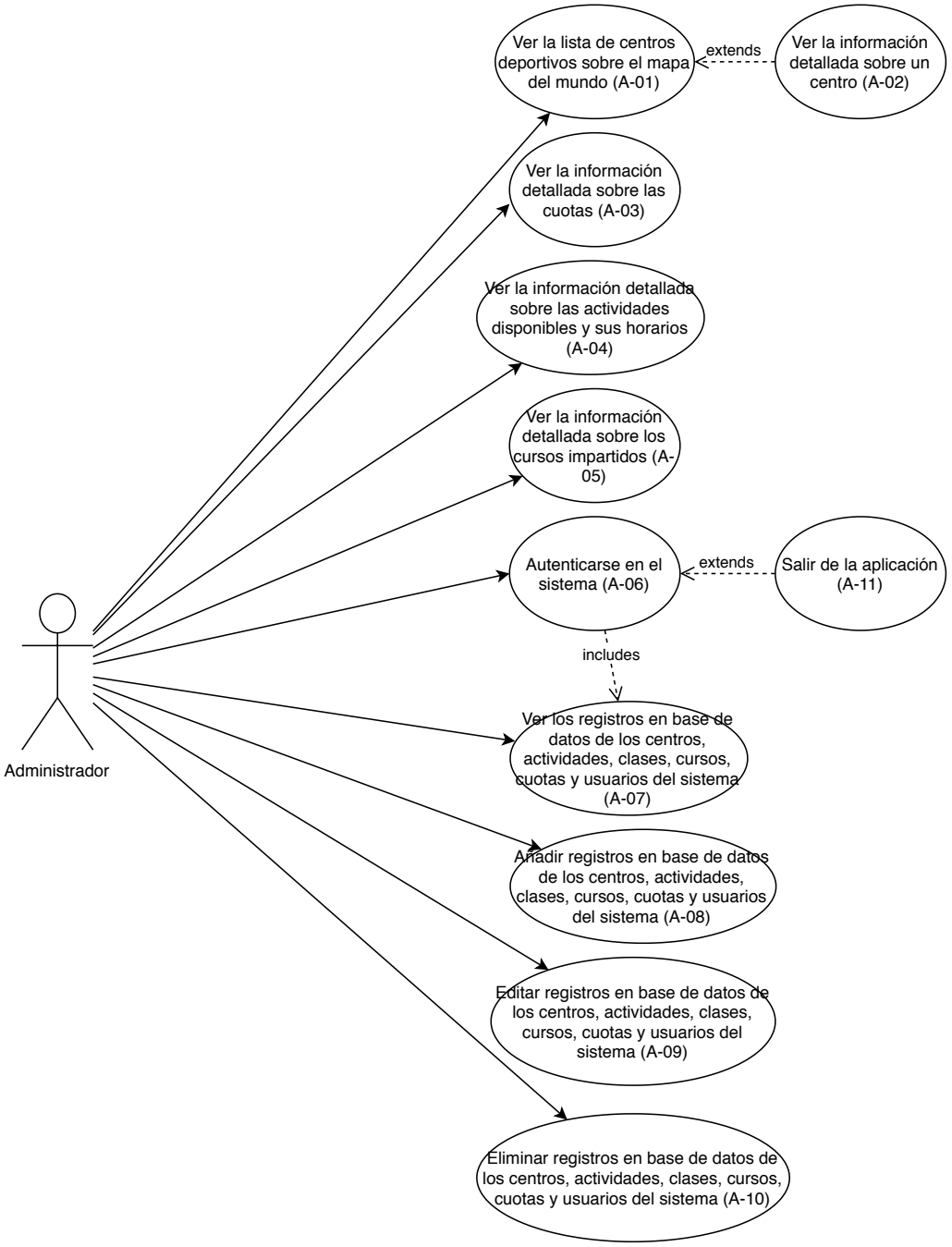

Figura 4.4: Diagrama de casos de uso de administrador

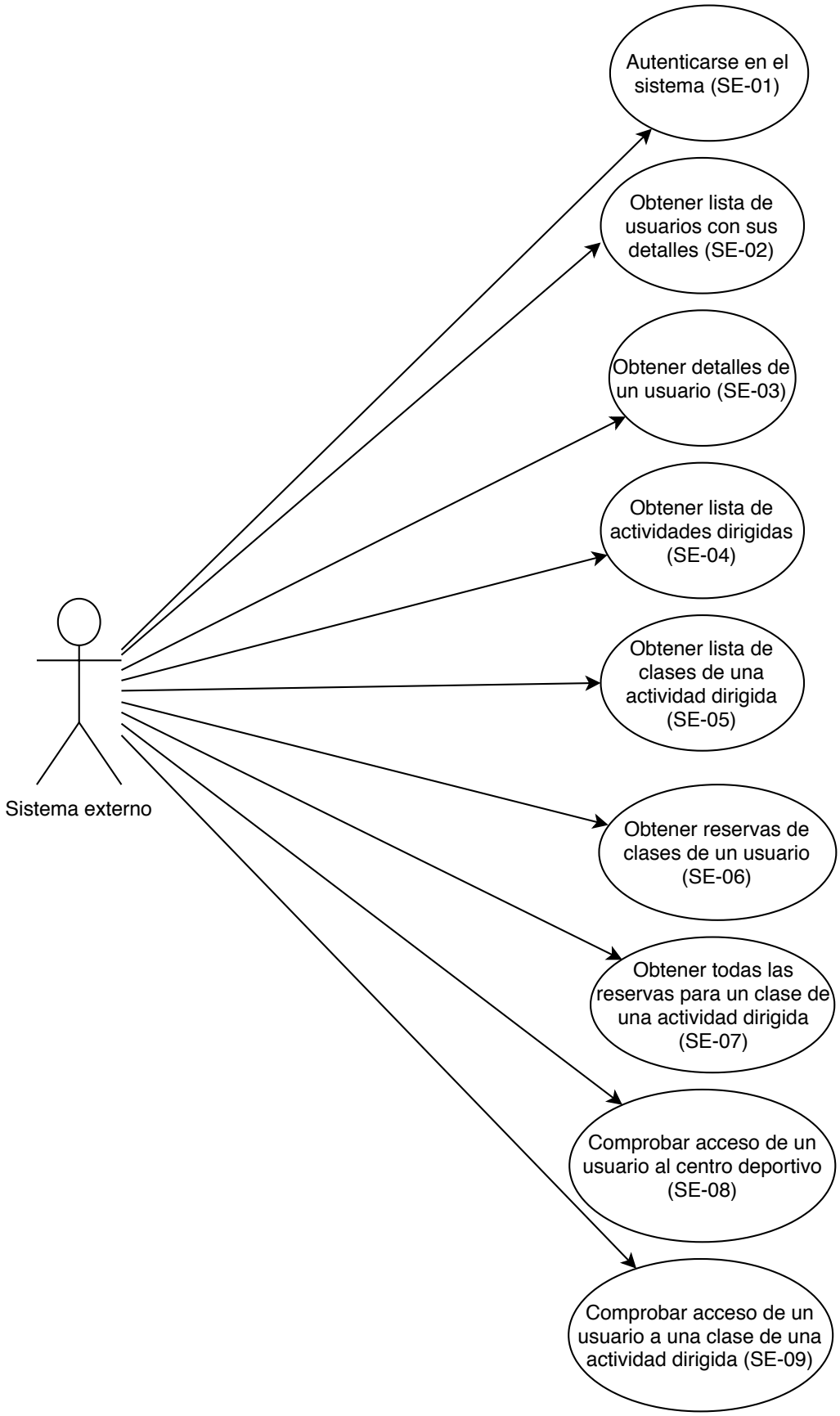

Figura 4.5: Diagrama de casos de uso de sistema externo

UNR-02, UR-02, A-02	
Descripción	Ver la información detalla sobre un centro.
Entradas	2 opciones: ▪ Clic en botón 'IR' de la lista de centros. ▪ Clic en botón 'VER ESTE CENTRO' desde el mapa.
Procesos	Navega a la vista de detalle del centro.
Salidas	▪ Los detalles del centro. ▪ La situación exacta del centro en el mapa. ▪ El horario de clases de las actividades dirigidas del centro. ▪ Un breve detalle y descripción de las instalaciones del centro.

Tabla 4.2: Caso de uso (UNR-02, UR-02, A-02). Ver la información detallada sobre un centro

UNR-03, UR-03, A-03	
Descripción	Ver un listado de las cuotas con sus detalles.
Entradas	Clic en la pestaña 'CUOTAS' en la ventana principal.
Procesos	Navega a la vista de listado de cuotas.
Salidas	Listado de cuotas con su información de precio, duración mínima y facilidades incluidas.

Tabla 4.3: Caso de uso (UNR-03, UR-03, A-03). Ver la información detallada sobre las cuotas

UNR-04, UR-04, A-04	
Descripción	Ver información detallada sobre las actividades disponibles.
Entradas	Clic en la pestaña 'ACTIVIDADES' en la ventana principal.
Procesos	Navega a la vista de listado de actividades.
Salidas	▪ Listado completo de actividades con breve descripción. ▪ El horario de clases de las actividades dirigidas del centro. ▪ Un breve detalle y descripción de las instalaciones del centro.

Tabla 4.4: Caso de uso (UNR-04, UR-04, A-04). Ver la información detallada sobre las actividades disponibles y sus horarios

UNR-05, UR-05, A-05	
Descripción	Ver información detallada sobre los cursos impartidos.
Entradas	Clic en la pestaña 'CURSOS' en la ventana principal.
Procesos	Navega a la vista de listado de cursos.
Salidas	▪ Listado completo de cursos con breve descripción. ▪ Un breve detalle y descripción de las instalaciones del centro.

Tabla 4.5: Caso de uso (UNR-05, UR-05, A-05). Ver la información detallada sobre los cursos impartidos

UNR-06	
Descripción	Solicitar un día de prueba gratis en un centro deportivo.
Entradas	▪ Clic en la pestaña 'PRUEBA GRATIS' en la ventana principal. ▪ A continuación, rellena los datos de nombre, apellidos, email, DNI y contraseña. ▪ Finalmente clic en el botón 'Enviar'.
Procesos	▪ Navega a la vista de solicitud de un día de prueba gratis. ▪ Una vez enviados los datos, comprueba que todos los campos del formulario son correctos y que el usuario todavía no está registrado. ▪ Registra el usuario, navega al área de cliente y envía un correo electrónico con la información necesario.
Salidas	▪ Ventana con un formulario a rellenar. ▪ Una vez rellenado el formulario, navega al área de cliente con los datos de la nueva cuenta y notifica del envío de un correo electrónico.

Tabla 4.6: Caso de uso (UNR-06). Solicitar un día de prueba gratis

UNR-07	
Descripción	Darse de alta en el sistema.
Entradas	▪ Clic en el botón 'INSCRIBIRSE' en la cabecera de la aplicación. ▪ A continuación, rellenar el formulario de email, contraseña, nombre, apellidos, DNI, fecha de nacimiento, dirección. ▪ Elegir la cuota preferida. ▪ Introducir los datos bancarios. ▪ Finalmente clic en el botón 'Inscribirse'.
Procesos	▪ Navega a la vista de inscripción. ▪ Una vez enviados los datos, comprueba que todos los campos del formulario son correctos y que el usuario todavía no está registrado. ▪ Registra el usuario y navega al área de cliente.
Salidas	▪ Ventana con un formulario a rellenar. ▪ Una vez rellenado el formulario, navega al área de cliente con los datos de la nueva cuenta.

Tabla 4.7: Caso de uso (UNR-07). Darse de alta en el centro deportivo

UR-06, A-06	
Descripción	Autenticarse en el sistema desde la aplicación.
Entradas	▪ Clic en el botón 'AREA DE CLIENTE' en la cabecera de la aplicación. ▪ A continuación, rellenar las credenciales email, contraseña. ▪ Finalmente clic en el botón 'INICIAR SESIÓN'.
Procesos	▪ Navega a la vista de autenticación. ▪ Una vez enviados los datos, comprueba que las credenciales son correctas. ▪ Navega al área de cliente si es un usuario o, en su defecto, al área de administración si se trata de un administrador.
Salidas	▪ Ventana con un formulario a rellenar. ▪ Una vez rellenado el formulario, navega al área de cliente ó al área de administrador según el rol del usuario autenticado.

Tabla 4.8: Caso de uso (UR-06, A-06). Autenticarse en el sistema

UR-07	
Descripción	Ver la información personal.
Entradas	Clic en el botón 'AREA DE CLIENTE' en la cabecera de la aplicación cuando el usuario está autenticado.
Procesos	Navega al área de cliente.
Salidas	Muestra todos los datos del usuario además de su código de socio, su cuota, fecha de inicio y de finalización de esta.

Tabla 4.9: Caso de uso (UR-07). Ver la información personal

UR-08	
Descripción	Modificar datos personales del usuario.
Entradas	▪ Clic en el acordeón 'Datos personales'. ▪ Modificar los datos del formulario que se desee cambiar. ▪ Clic en el botón 'MODIFICAR'
Procesos	▪ Abre el acordeón de 'Datos personales' en la pestaña 'PERSONAL' mostrando los datos actuales del usuario en un formulario. ▪ Una vez se hace clic en el botón 'Modificar' comprueba que los nuevos datos son válidos y actualiza en caliente los datos del usuario.
Salidas	Muestra todos los datos del usuario actualizados.

Tabla 4.10: Caso de uso (UR-08). Modificar los datos personales

UR-09	
Descripción	Modificar datos bancarios del usuario.
Entradas	▪ Clic en el acordeón 'Cuenta bancaria'. ▪ Modificar el campo 'IBAN' en el formulario con un valor válido. ▪ Clic en el botón 'MODIFICAR'
Procesos	▪ Abre el acordeón de 'Cuenta bancaria' en la pestaña 'PERSONAL' mostrando el IBAN actual registrado por el usuario. ▪ Una vez se hace clic en el botón 'Modificar' comprueba que el nuevo valor es válido y actualiza en caliente el 'IBAN' del usuario.
Salidas	Muestra el valor del 'IBAN' actualizado.

Tabla 4.11: Caso de uso (UR-09). Modificar los datos bancarios

UR-10	
Descripción	Renovar cuota del usuario.
Entradas	▪ Clic en el acordeón 'Renovación cuota'. ▪ Selecciona la cuota deseada. ▪ Clic en el botón 'CONTRATAR CUOTA'.
Procesos	▪ Abre el acordeón de 'Renovación cuota' en la pestaña 'PERSONAL' mostrando el listado de las cuotas disponibles para el usuario. ▪ Una vez se selecciona la cuota deseada y se hace clic en 'Contratar cuota' se abre un modal notificando al usuario de la fecha de comienzo de dicha cuota para que se confirme.
Salidas	Muestra un modal con la fecha de comienzo de la nueva cuota y pidiendo confirmación. Una vez se confirma actualiza el dato de finalización de membresía del usuario.

Tabla 4.12: Caso de uso (UR-10). Renovar cuota

UR-11	
Descripción	Darse de baja del sistema.
Entradas	▪ Clic en el acordeón 'Bajas'. ▪ Clic en el botón 'Darse de baja'.
Procesos	▪ Abre el acordeón de 'Bajas' en la pestaña 'PERSONAL' mostrando un botón para darse de baja. ▪ Una vez se hace clic el botón, abre un modal pidiendo confirmación al usuario para continuar con el proceso de baja. ▪ Si se confirma, navega a la ventana principal, cancela la autenticación del usuario y borra todos los datos relacionados con el usuario de la base de datos.
Salidas	Muestra un modal pidiendo confirmación. Una vez se confirma, navega a la ventana principal como un usuario sin autenticación.

Tabla 4.13: Caso de uso (UR-11). Darse de baja del sistema

UR-12	
Descripción	Reservar asistencia a clases de actividades dirigidas.
Entradas	▪ Clic en la pestaña 'RESERVA DE ACTIVIDADES'. ▪ Clic en una clase del calendario y clic en confirmación de la reserva.
Procesos	▪ Abre el calendario con las clases de las actividades para los siguientes siete días. ▪ Cuando se hace clic sobre una clase, abre un modal pidiendo confirmación. ▪ Si se confirma, guarda la reserva en base de datos y muestra al usuario la clase de otro color para que sea reconocible la reserva.
Salidas	Muestra las clases de actividades en la próxima semana. Aquellas que tienen reserva aparecen con un color característico.

Tabla 4.14: Caso de uso (UR-12). Reservar asistencia a actividades dirigidas

UR-13	
Descripción	Cancelar asistencia a clases de actividades dirigidas.
Entradas	▪ Clic en la pestaña 'RESERVA DE ACTIVIDADES'. ▪ Clic en una clase del calendario y clic en cancelación de la reserva.
Procesos	▪ Abre el calendario con las clases de las actividades para los siguientes siete días. ▪ Cuando se hace clic sobre una clase ya reservada, abre un modal pidiendo confirmación de cancelación. ▪ Si se confirma, borra la reserva en base de datos y muestra al usuario la clase del color estandar.
Salidas	Muestra las clases de actividades en la próxima semana. Aquellas que tienen reserva aparecen con un color característico. Se cambia el color de la actividad de la que se ha cancelado la reserva.

Tabla 4.15: Caso de uso (UR-13). Cancelar asistencia a actividades dirigidas

UR-14, A-11	
Descripción	Abandonar la aplicación como usuario autenticado.
Entradas	Clic en el botón 'SALIR' de la cabecera.
Procesos	▪ Elimina el token del usuario autenticado. ▪ Navega a la ventana principal como usuario no autenticado.
Salidas	Muestra la ventana principal como usuario no autenticado.

Tabla 4.16: Caso de uso (UR-14, A-11). Salir de la aplicación

A-07	
Descripción	Ver los registros de la base de datos de los centros, actividades, clases, cursos, cuotas y usuarios del sistema.
Entradas	Clic en el botón 'AREA DE ADMINISTRADOR' de la cabecera.
Procesos	Navega al área de administrador.
Salidas	Muestra el área de administrador con una pestaña para los registros de la base de datos de los centros, actividades, clases, cursos, cuotas y usuarios respectivamente.

Tabla 4.17: Caso de uso (A-07) Ver los registros en base de datos

A-08	
Descripción	Añadir registros en base de datos de los centros, actividades, clases, cursos, cuotas y usuarios del sistema.
Entradas	▪ Clic en la pestaña correspondiente del área de administrador. ▪ Clic en el botón 'Añadir'. ▪ Rellenar los datos del formulario que aparece en el modal. ▪ Clic en 'Crear'.
Procesos	▪ Muestra la tabla de los registros de la base de datos del a pestaña correspondiente. ▪ Al hacer clic en 'Añadir' muestra un modal con los datos necesarios para crear un nuevo registro. ▪ Una vez se rellenar el formulario, se añade el registro en base de datos y se actualiza la tabla.
Salidas	Muestra un formulario a rellenar para añadir el nuevo registro. Una vez rellenado muestra la tabla actualizada.

Tabla 4.18: Caso de uso (A-08). Añadir registros en base de datos

A-09	
Descripción	Modificar registros en base de datos de los centros, actividades, clases, cursos, cuotas y usuarios del sistema.
Entradas	▪ Clic en la pestaña correspondiente del área de administrador. ▪ Seleccionar el elemento que se desea modificar. ▪ Clic en el botón 'Modificar'. ▪ Cambiar los datos del formulario que aparece en el modal. ▪ Clic en 'Modificar'.
Procesos	▪ Muestra la tabla de los registros de la base de datos del a pestaña correspondiente. ▪ Al hacer clic en 'Modificar' muestra un modal con los datos del actual registro. ▪ Una vez se cambia algún dato del formulario, se actualiza el registro en base de datos y se actualiza la tabla.
Salidas	Muestra un formulario a rellenar para modificar el registro. Una vez rellenado muestra la tabla actualizada.

Tabla 4.19: Caso de uso (A-09). Editar registros en base de datos

A-10	
Descripción	Eliminar registros en base de datos de los centros, actividades, clases, cursos, cuotas y usuarios del sistema.
Entradas	▪ Clic en la pestaña correspondiente del área de administrador. ▪ Seleccionar el/los elemento/s que se desea borrar. ▪ Clic en el botón 'Borrar'. ▪ Clic en 'Confirmar' en el modal.
Procesos	▪ Muestra la tabla de los registros de la base de datos del a pestaña correspondiente. ▪ Al hacer clic en 'Borrar' muestra un modal pidiendo confirmación. ▪ Una vez se confirma, se borran los elementos de la base de datos y se actualiza la tabla.
Salidas	Muestra un modal pidiendo confirmación. Una vez confirmado muestra la tabla actualizada.

Tabla 4.20: Caso de uso (A-10). Eliminar registros en base de datos

SE-01	
Descripción	Autenticarse en el sistema.
Entradas	Enviar una petición a la interfaz REST del servidor con el usuario y la contraseña.
Procesos	▪ Comprobar que el usuario es válido y es administrador. ▪ Enviar token.
Salidas	Token válido para realizar peticiones REST al servidor.

Tabla 4.21: Caso de uso (SE-01). Autenticarse en el sistema

SE-02	
Descripción	Obtener lista de usuarios con sus detalles mediante interfaz REST.
Entradas	Enviar una petición GET a la interfaz REST del servidor al *end-point* de usuarios.
Procesos	▪ Comprobar que el token de la cabecera es válido. ▪ Enviar lista de usuarios en la base de datos.
Salidas	Lista de usuarios de la base de datos con sus detalles.

Tabla 4.22: Caso de uso (SE-02). Obtener lista de usuarios con sus detalles

SE-03	
Descripción	Obtener detalles de un usuario mediante interfaz REST.
Entradas	Enviar una petición GET a la interfaz REST del servidor al *end-point* de usuario con el id del usuario como parámetro de la ruta.
Procesos	▪ Comprobar que el token de la cabecera es válido. ▪ Enviar detalles del usuario.
Salidas	Detalles del usuario solicitado.

Tabla 4.23: Caso de uso (SE-03). Obtener detalles de un usuario

SE-04	
Descripción	Obtener lista de actividades mediante interfaz REST.
Entradas	Enviar una petición GET a la interfaz REST del servidor al *end-point* de actividades.
Procesos	▪ Comprobar que el token de la cabecera es válido. ▪ Enviar lista de actividades de la base de datos.
Salidas	Lista de actividades con sus clases.

Tabla 4.24: Caso de uso (SE-04). Obtener lista de actividades dirigidas

SE-05	
Descripción	Obtener lista de clases de una actividad dirigida mediante interfaz REST.
Entradas	Enviar una petición GET a la interfaz REST del servidor al *endpoint* de actividades con el id de la actividad como parámetro de la ruta.
Procesos	▪ Comprobar que el token de la cabecera es válido. ▪ Enviar lista de clases de la actividad.
Salidas	Lista de clases de la actividad solicitada.

Tabla 4.25: Caso de uso (SE-05). Obtener lista de clases de actividad dirigida

SE-06	
Descripción	Obtener reservas de clases de un usuario.
Entradas	Enviar una petición GET a la interfaz REST del servidor al *endpoint* de actividades con el id del usuario como parámetro de la ruta.
Procesos	▪ Comprobar que el token de la cabecera es válido. ▪ Enviar lista de reservas del usuario.
Salidas	Lista de reservas del usuario solicitado.

Tabla 4.26: Caso de uso (SE-06). Obtener reservas de clases de un usuario

SE-07	
Descripción	Obtener todas las reservas para una clase de una actividad dirigida.
Entradas	Enviar una petición GET a la interfaz REST del servidor al *endpoint* de actividades con el id de la clase como parámetro de la ruta.
Procesos	▪ Comprobar que el token de la cabecera es válido. ▪ Enviar lista de reservas de esa clase.
Salidas	Lista de reservas de la clases solicitada.

Tabla 4.27: Caso de uso (SE-07). Obtener todas las reservas para una clase de actividad dirigida

SE-08	
Descripción	Comprobar acceso de un usuario al centro deportivo.
Entradas	Enviar una petición GET a la interfaz REST del servidor al *endpoint* de usuarios con el id del usuario como parámetro de la ruta.
Procesos	▪ Comprobar que el token de la cabecera es válido. ▪ Enviar el resultado de comprobar si el usuario tiene cuota válida para acceder al centro deportivo.
Salidas	Resultado de permiso de acceso del usuario.

Tabla 4.28: Caso de uso (SE-08) Comprobar acceso de un usuario al centro deportivo

SE-09	
Descripción	Comprobar acceso de un usuario a una clase de una actividad dirigida.
Entradas	Enviar una petición GET a la interfaz REST del servidor al *end-point* de usuarios con el id del usuario y de la clase como parámetros de la petición.
Procesos	▪ Comprobar que el token de la cabecera es válido. ▪ Enviar el resultado de comprobar si el usuario tiene una reserva sobre la clase.
Salidas	Resultado de permiso de acceso del usuario a la clase.

Tabla 4.29: Caso de uso (SE-09). Comprobar acceso de un usuario a una clase de una actividad dirigida

4.5 Diagramas de actividad

En esta sección se encontrará una secuencia de diagramas de actividad que ilustran algunos de los casos de los flujos de los casos de uso más importantes de la aplicación.

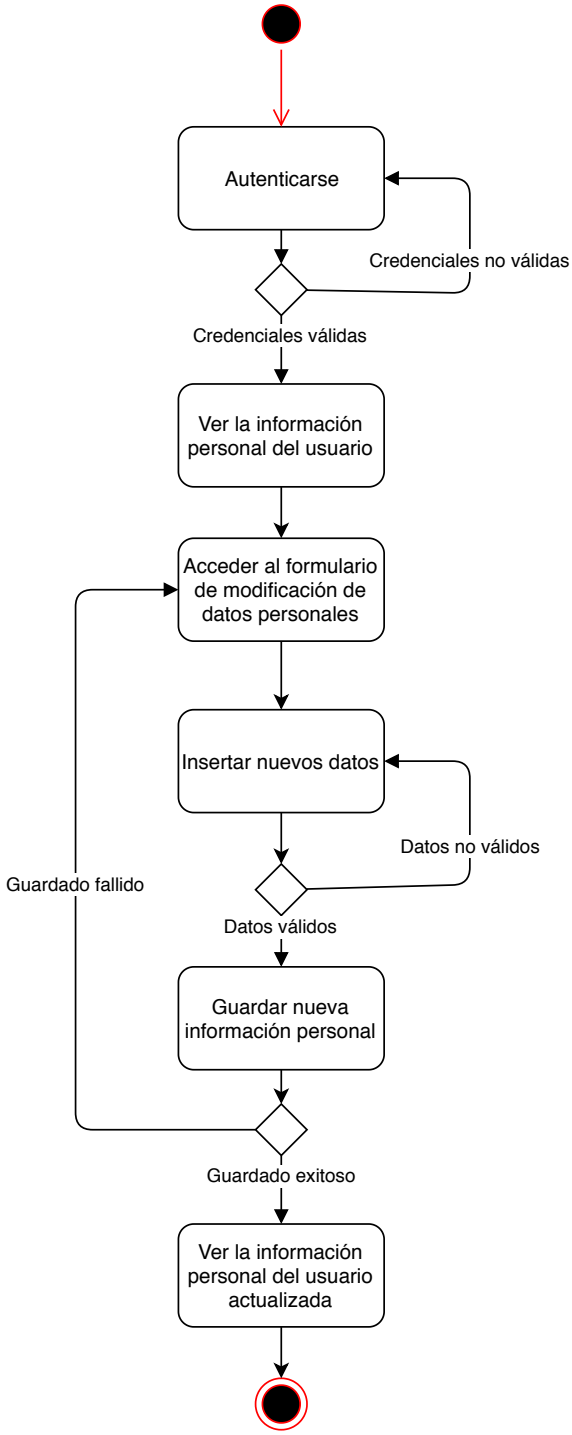

Figura 4.6: Diagrama de actividad. Modificar datos de usuario

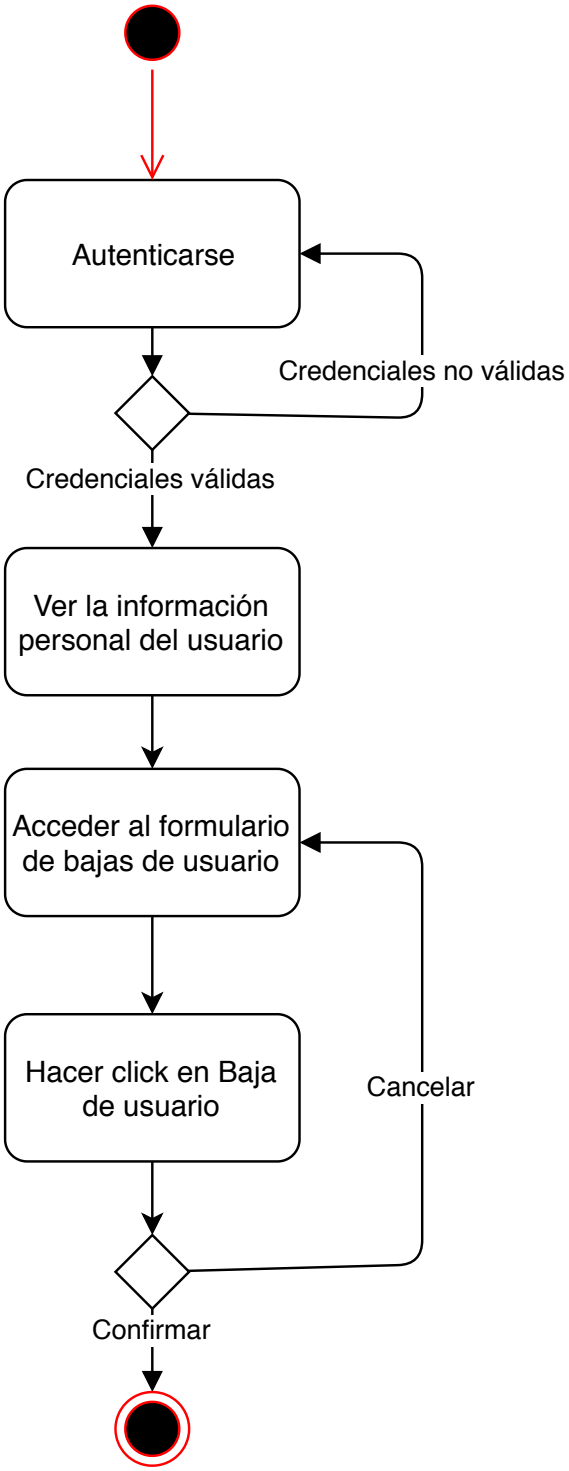

Figura 4.7: Diagrama de actividad. Darse de baja

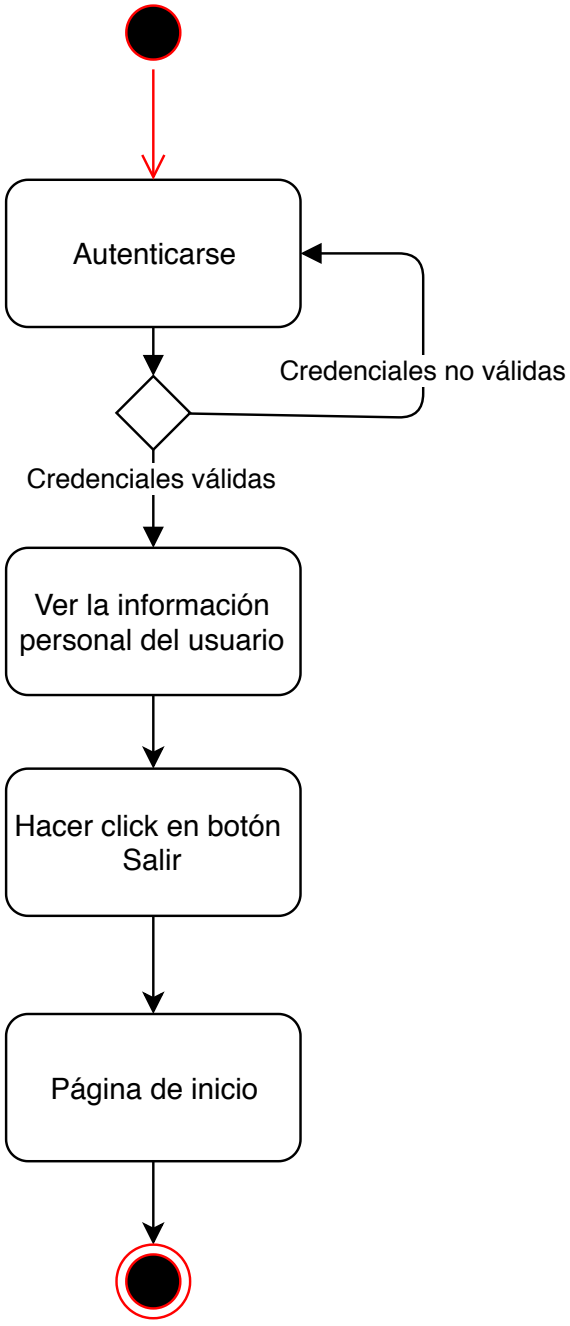

Figura 4.8: Diagrama de actividad. Salir de la aplicación

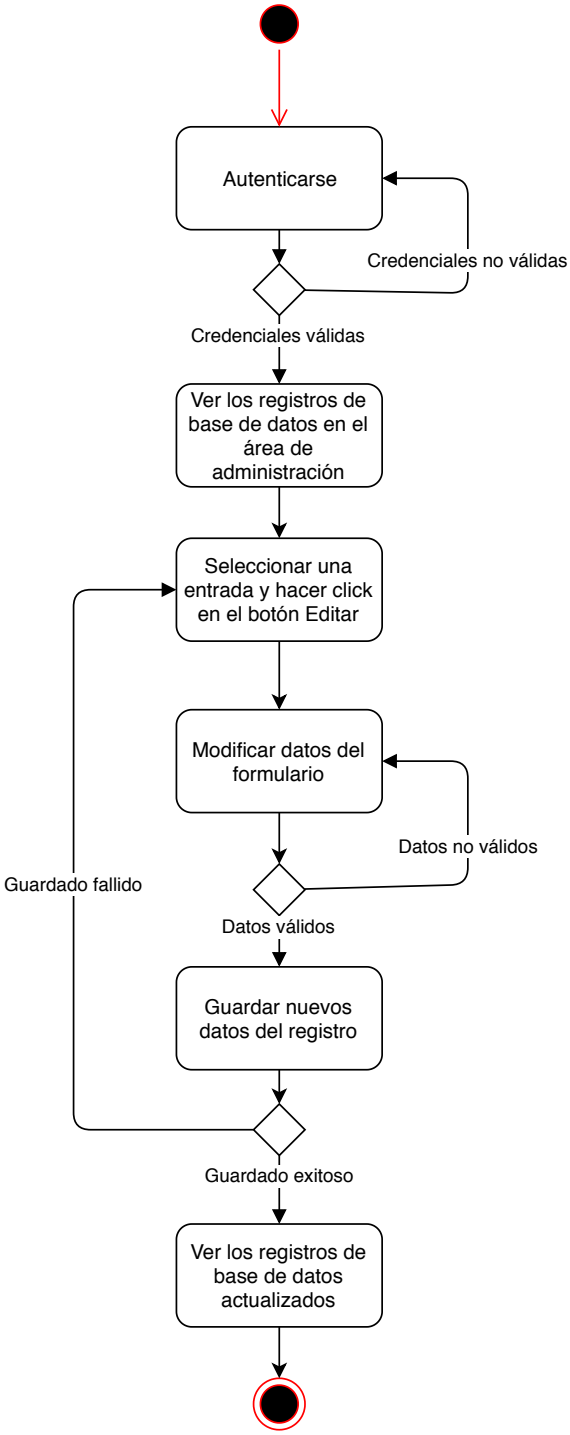

Figura 4.9: Diagrama de actividad. Modificar registros en la base de datos

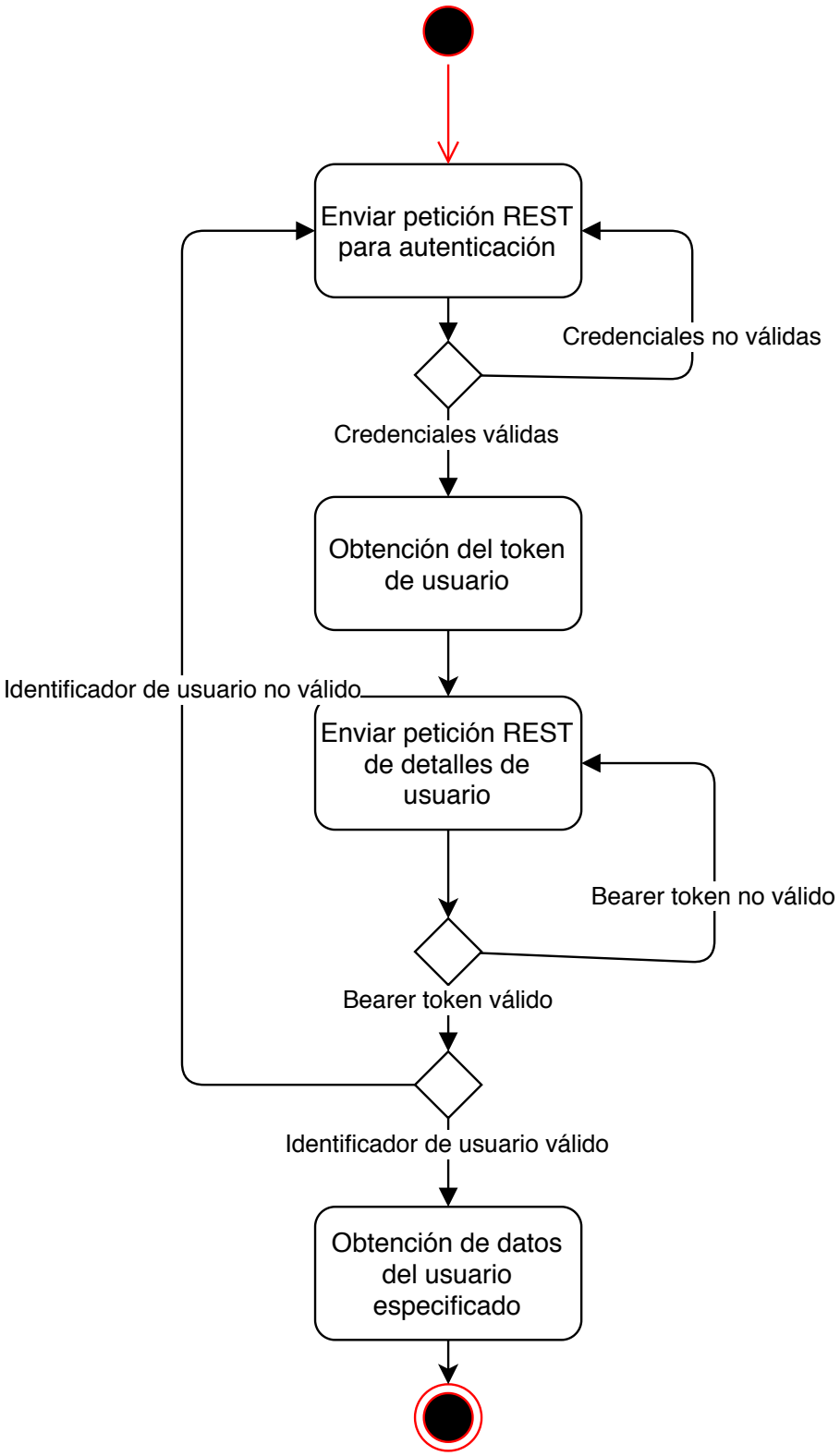

Figura 4.10: Diagrama de actividad. Obtención de detalles de usuario mediante petición REST

CAPÍTULO 5
Diseño

5.1 Introducción

En esta de la memoria se expondrá en primer lugar la arquitectura elegida para la aplicación y más tarde se mostrará algunos esbozos o *mockups* de las ventanas más importantes de la misma.

5.2 Arquitectura del sistema

La arquitectura cliente/servidor es un modelo de diseño de software en el que las tareas que componen su funcionamiento están divididas en diferentes máquinas, particularmente en unas llamadas cliente y otras llamadas servidor.[15]

- Cliente. En el caso de las aplicaciones web, está es la parte correspondiente al *Frontend*, es la parte del software que el usuario desde el navegador se descarga y ejecuta. De está forma se relaja la carga de trabajo que corresponde al servidor. Sin embargo, los procesos que ejecuta está parte del software son usualmente procesos no complejos y de mera comunicación con el servidor.

- Servidor. Está es la parte correspondiente al *Backend* y a la base de datos. En este lado, el software puede estar ejecutándose en una o varias máquinas, facilitando de esta manera la escalabilidad mediante el escalado horizontal. Más allá de esto, se centralizan todos los procesos correspondientes con la manipulación de la información y los accesos de seguridad por lo que un cliente defectuoso o malicioso no puede dañar el sistema.

Por otro lado, la programación por capas es un patrón de arquitectura de software muy utilizado en el diseño de aplicaciones web cuyo objetivo principal es el de la separación del software en diferentes capas (modelo-vista-controlador) [13] [14]. Al separar la lógica entre estas diferentes capas la implementación de cada una de ellas no repercute en el funcionamiento de las demás, por lo que la mantenibilidad se asegura en todo momento y se facilita el proceso de cambio de lógica en la aplicación.

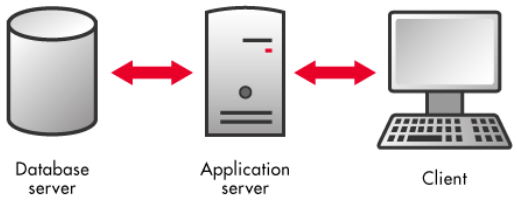

Figura 5.1: Patrón de arquitectura Cliente/Servidor
Fuente: http://www.schilling.dk/web/guest/technology

De forma breve se podría definir el funcionamiento de las diferentes capas de la siguiente manera:

- **Modelo**. Es la capa de persistencia, donde se persisten los datos, la encargada de la representación de la información. Esta capa recibe peticiones de manipulación de los datos por parte del controlador. Aquí se encontraría la implementación de la base de datos.
- **Vista**. Es la capa de presentación, la que el usuario ve e interactúa con ella. Aquí se encontraría la parte de *Frontend* de la aplicación.
- **Controlador**. Es la capa de lógica de negocio, donde se encuentran los algoritmos que procesan las peticiones del *Frontend*. Aquí se encontraría la parte de *Backend* de la aplicación.

Como se puede observar en la figura 5.2, el flujo podría dividirse en los siguientes pasos:

1. El usuario realiza una petición al controlador (en ocasiones mediante la vista).
2. El controlador procesa la petición y accede al modelo para recuperar los datos correspondientes.
3. El controlador actualiza la vista con los nuevos datos y el usuario es notificado de este cambio en la vista, obteniendo de esta forma el resultado de su petición inicial.

5.2.1. Capa de persistencia (modelo)

El modelo es la información, la capa de persistencia, la cual representa las entidades incluidas en el Diagrama de clases de la aplicación web del capítulo de Análisis.

Esta capa tan solo se comunica con la capa de lógica de negocio, está es la encargada de mantener la información persistente de la aplicación y de proporcionarle al controlador la representación de los datos que este le pueda solicitar.

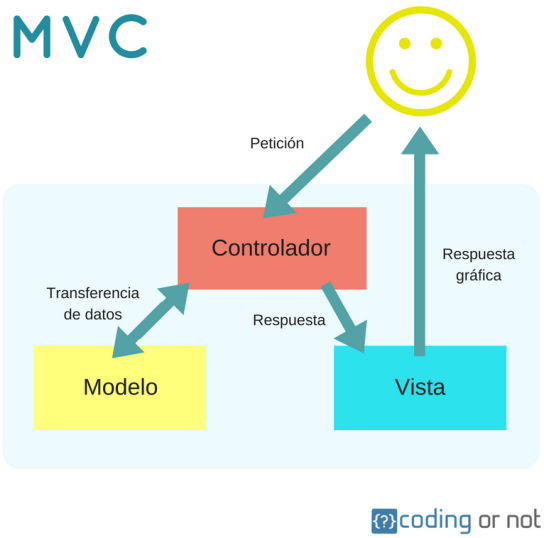

Figura 5.2: Patrón de arquitectura MVC (Modelo-Vista-Controlador)
Fuente:
https://codingornot.com/mvc-modelo-vista-controlador-que-es-y-para-que-sirve

5.2.2. Capa de lógica de negocio (controlador)

El controlador o capa de lógica de negocio se encarga del procesamiento de peticiones por parte de la vista, pudiendo para ello, realizar peticiones a la capa de persistencia para recuperar la información necesario.

La mayor carga de procesamiento se da en esta capa debido a que es una capa fácilmente escalable y que proporciona seguridad debido a que se ejecuta en máquinas bajo el control del proveedor y no en las máquinas de los usuarios.

La total independencia con respecto a las otras capas permite poder añadir funcionalidades a esta capa sin depender de la vista, sino que se puede implementar de forma adicional una API RESTful mediante la cual es posible ofrecer funcionamiento a sistemas externos que se comuniquen con la aplicación mediante este protocolo. [16].

Se detalla el funcionamiento en la sección de Interfaz RESTful del capítulo Implementación.

5.2.3. Capa de presentación (vista)

La vista corresponde a la capa de presentación. Esta capa es la encargada de representar la información de forma intuitiva para el usuario y de procesar las interacciones de este. También se encarga de comunicar a la capa de lógica de negocio las peticiones que el usuario realiza.

Debido a la arquitectura cliente/servidor, está capa además de representar la información, realiza parte del procesamiento de la aplicación pero que en todo momento se tratan de procesos correspondientes a la interfaz y el tratamiento de la información que llega desde el controlador.

5.3 Interfaz

En esta sección se mostrará el diseño razonado de las ventanas más importantes en el flujo de la aplicación. Se mostrarán bocetos de cada ventana, estos bocetos han sido diseñados con la herramienta online gratuita llamada *NinjaMock* (https://ninjamock.com/).

Las ventanas bocetadas serán las siguientes y cada una constará de un esquema que explicará el funcionamiento de los elementos de cada una de ellas.

- Ventana principal. Ventana a la que tiene acceso cualquier usuario, tanto registrado como no registrado. Está ventana será totalmente visual y servirá en su totalidad de menú para acceder a todas los servicios y funcionalidades de la aplicación.
- Ventana de centros deportivos. En esta ventana se mostrará un listado de los centros deportivos existentes con acceso a cada uno de ellos junto a un mapa del mundo donde se encontrarán situados estos.
- Ventana de detalle de un centro deportivo. La ventana mostrará lista la información de dirección, contacto y horarios del centro. Además del centro situado en un mapa del mundo, el listado de actividades que se imparten en ese centro y una zona de información acerca de las instalaciones del centro.
- Ventana de registro. En esta ventana se encontrará el formulario a rellenar para darse de alta en el sistema.
- Ventana de autenticación. Ventana con el formulario para acceder al perfil de usuario.
- Ventana de área de cliente. Ventana con toda la información del usuario así como formularios para actualizar esta información, opción para darse de baja y para reservar actividades.
- Ventana de área de administrador. Esta ventana tan solo estará disponible para los usuarios administradores y constará de diferentes tablas en las que manipular los registros de base de datos de las diferentes entidades de la aplicación.

5.3.1. Ventana principal

En el Boceto de la ventana principal podemos apreciar los diferentes elementos:

- Cabecera. Este elemento estará presente en todas las vistas de la aplicación. Podemos ver el logotipo de la aplicación que a su vez sirve como botón para ser direccionados de vuelta a la ventana principal desde cualquier otra ventana. Más allá de esto vemos los botones de 'ÁREA DE CLIENTE' y 'INSCRÍBETE' que redirigen a la ventana de autenticación y a la de registro respectivamente.

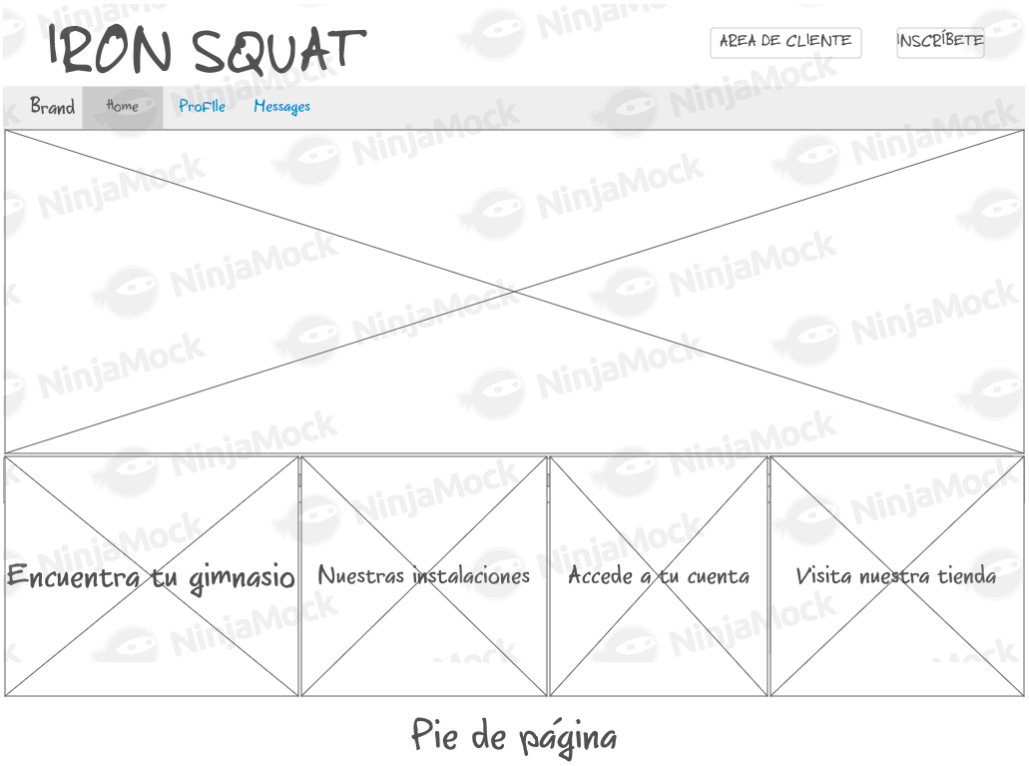

Figura 5.3: Boceto de la ventana principal

- Barra de menús. En esta barra encontramos acceso a todas las diferentes funcionalidades de la aplicación como los centros deportivos, la opción de obtener un día de prueba gratis, información sobre las cuotas, actividades, cursos y tienda.
- Encuentra tu gimnasio. Imagen que sirve como botón para redirigir al usuario a la ventana de centros deportivos.
- Nuestras instalaciones. Imagen que tiene el funcionamiento de un botón que redirige a una ventana que muestra información sobre las diferentes instalaciones con las que cuenta el centro deportivo.
- Accede a tu cuenta. Imagen que sirve como botón para redirigir a la ventana de autenticación.
- Visita nuestra tienda. Imagen que sirve como botón para redirigir a la ventana de la tienda (ventana no implementada en el proyecto por salirse del ámbito del mismo).
- Pie de página. Por último vemos el pie de página con botones que enlazan a las redes sociales de la aplicación y a la información legal de la misma. Este elemento, como el de cabecera, también se encontrará presente en todas las diferentes ventanas de la aplicación.

5.3.2. Ventana de centros deportivos

En el Boceto de la ventana de centros deportivos podemos destacar los siguientes elementos:

Figura 5.4: Boceto de la ventana de centros deportivos

- Encuentra tu gimnasio. Listado de los centros deportivos dados de alta en la aplicación con un botón para redirigir a los detalles de cada uno.
- Mapa del mundo. Mapa de Google Maps en el que se puede navegar y donde aparecen marcadas las localizaciones de todos los centros que aparecen en la lista de al lado.

5.3.3. Ventana de detalle de un centro deportivo

En el Boceto de la ventana de detalle de un centro deportivo podemos diferenciar entre los siguientes elementos:

- Selecciona tu gimnasio más cercano. Un desplegable en el que poder cambiar dinámicamente el gimnasio seleccionado del que se muestran los detalles.
- Detalles del centro deportivo. Detalles de dirección, contacto, horarios y precios del centro deportivo seleccionado.
- Mapa del mundo. Mapa de Google Maps en el que aparece la dirección exacta donde localizar el centro deportivo seleccionado.
- Calendario. En este calendario aparecen las clases programadas de actividades en el centro deportivo para los siguientes 7 días.
- Nuestras instalaciones. Breve disposición de las diferentes instalaciones con las que cuenta el centro deportivo seleccionado.

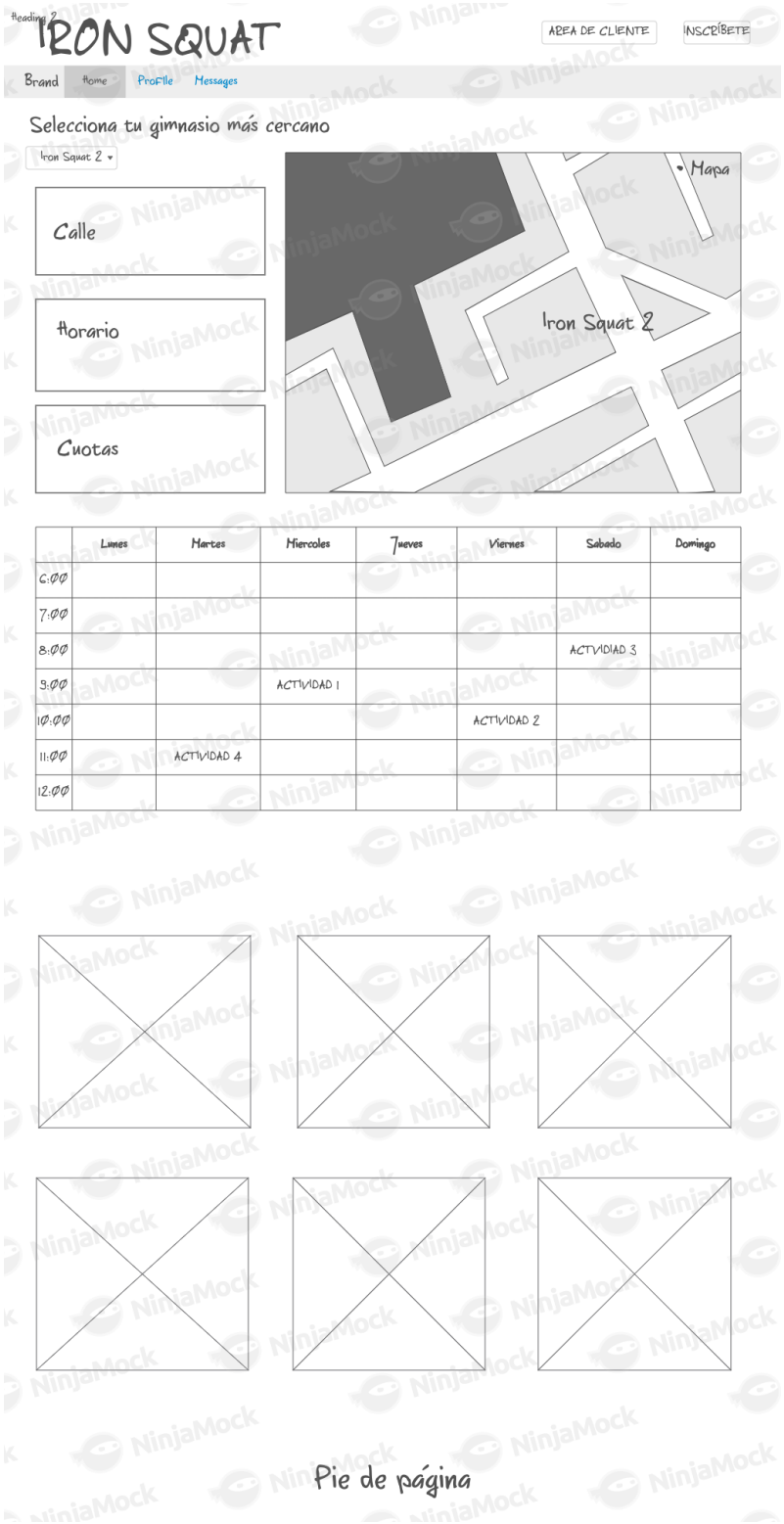

Figura 5.5: Boceto de la ventana de detalle de un centro deportivo

5.3.4. Ventana de registro

En el Boceto de la ventana de registro se encuentra un formulario a rellenar para darse de alta en la aplicación. Esto formulario estaría dividido en los diferentes pasos:

- Inserta email y contraseña.
- Rellena tus datos personales. Donde habrá que rellenar todos los campos que definen la entidad de usuario tales como Nombre, Apellidos, DNI, nº de contacto, fecha de nacimiento, dirección, CP y ciudad.
- Elige una de nuestras cuotas. El usuario deberá seleccionar la cuota deseada entre las diferentes opciones que permite la aplicación.
- Introduce datos de pago. Se encuentra un campo en el que rellenar la información de pago del usuario que desea darse de alta.

Figura 5.6: Boceto de la ventana de registro

5.3.5. Ventana de autenticación

En el Boceto de ventana de autenticación también dispone de un formulario de acceso en el que introducir las credenciales del usuario que corresponden con su email y su contraseña de accesos.

Figura 5.7: Boceto de ventana de autenticación

5.3.6. Ventana de área cliente

El Boceto de la ventana de área cliente es una de las ventanas más importantes y con más funcionalidad de la aplicación junto con la **??**. Esta ventana está disponible para todos los usuarios que estén registrados en el sistema y consta de los siguientes elementos:

- INFORMACIÓN PERSONAL. Carta con toda la información disponible del usuario aglutinada.
- Datos personales. Formulario en el que el usuario puede modificar sus datos personales de forma rápida excepto el nombre, apellido y el DNI.
- Cuenta bancaria. Campo donde introducir el número del nuevo IBAN el cual el usuario desee utilizar para realizar los pagos de la membresía del centro deportivo.
- Renovación cuota. Desplegable donde seleccionar la nueva cuota que el usuario desee contratar cuando se finalice la validez de la actual.
- Bajas. Botón para darse de baja del sistema y borrar los datos del usuario del registro de la base de datos.
- RESERVA DE ACTIVIDADES. Pestaña en la cual se tiene acceso al calendario de clases programadas de actividades dirigidas en la próxima semana y en la el usuario es capaz de administrar sus propias reservas añadiendo o eliminando reservas a esas clases.

Figura 5.8: Boceto de la ventana de área cliente

5.3.7. Ventana de área de administrador

El Boceto la de ventana de área de administrador es una ventana disponible tan solo para usuarios administradores del sistema y sirve para manipular los registros en base de datos de todas las entidades de la aplicación. Se divide en pestañas, cada pestaña corresponde a una entidad diferente:

- GIMNASIOS. Registros de los centros deportivos dados de alta en el sistema.
- ACTIVIDADES. Registros de las actividades dirigidas que se importan en los centros deportivos.
- EVENTOS. Registros de las clases impartidas de cada actividad dirigida disponible.
- CURSOS. Registros de los cursos disponibles bajo demanda en los centros deportivos.
- CUOTAS. Registros de las diferentes cuotas disponibles para ser miembro de los centros deportivos.
- USUARIOS. Registros de los usuarios registrados en el sistema, tanto los usuarios normales como los administradores.

En todas las pestañas se podrá añadir, modificar y eliminar registros.

Figura 5.9: Boceto la de ventana de área de administrador

CAPÍTULO 6
Implementación

En esta capítulo de la memoria se va a abordar todo lo relacionado con la fase de desarrollo de la aplicación. Por un lado se explicarán las tecnologías y librerias utilizadas para la programación del software. Además se hará una breve explicación del control de versiones utilizado así como del entorno y editor de código fuente utilizados.

A continuación se detallará la estructura de ficheros decidida y por último se explicará con detalle diversas partes de la programación que son consideradas más críticas y de más interés.

6.1 Tecnologías utilizadas

6.1.1. MEAN Stack

MEAN Stack es un paquete de tecnologías con el que es posible desarrollar una página web utilizando JavaScript como lenguaje de programación en todas las capas de desarrollo. Debido al auge del uso de JavaScript en el desarrollo de aplicaciones web, de móvil e incluso de escritorio (*Progresive Web Applications*), este paquete ha cogido cada vez más peso como elección para desarrollar aplicaciones web[17].

Este paquete debe sus siglas a las tecnologías que lo conforman:

- **M por MongoDB** siendo el sistema de base de datos no relacional el cual proporciona la persistencia de los datos.
- **E por Express.js**. El Framework de Node.js donde se aglutinan todas las herramientas para facilitar la programación de la lógica de negocio en la parte del servidor.
- **A por Angular**. Framework de JavaScript (TypeScript) usado para agilizar el desarrollo de la interfaz y procesamiento de información en el navegador del cliente.
- **E por Express.js**. El Framework de Node.js donde se aglutinan todas las herramientas para facilitar la programación de la lógica de negocio en la parte del servidor.

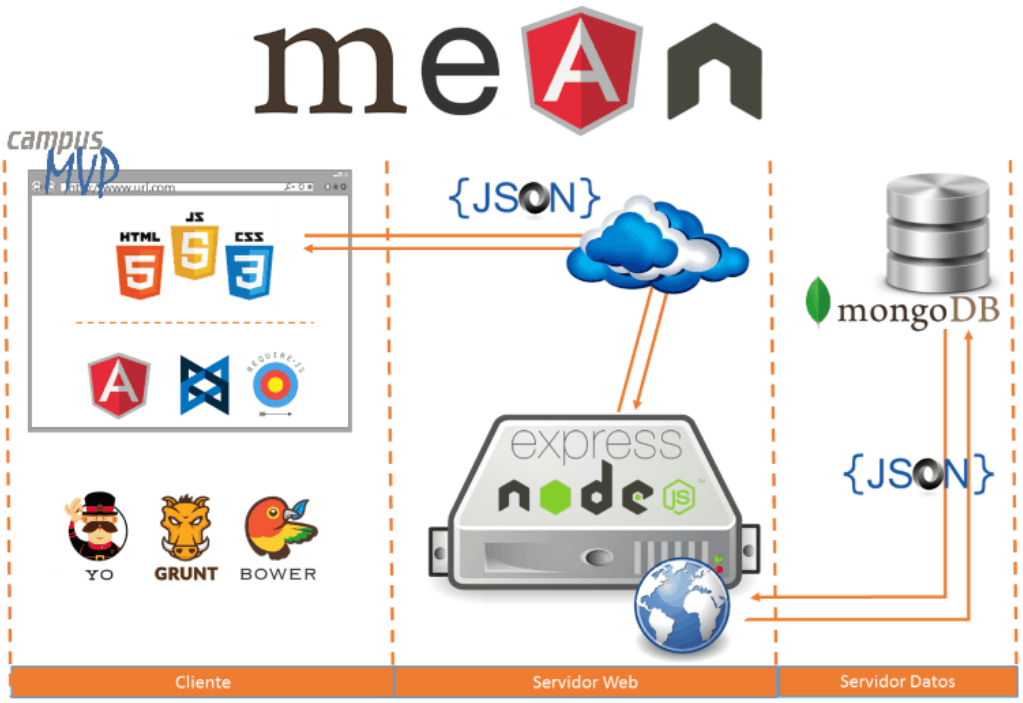

Figura 6.1: Tecnologías que forman el Stack MEAN
Fuente:
https://www.campusmvp.es/recursos/post/Que-es-el-stack-MEAN-y-como-escoger-el-mejor-p

A continuación se van a explicar las tecnologías que forman este paquete y el porque de la elección de esta para el desarrollo del proyecto.

MongoDB

MongoDB se corresponde con la capa de persistencia ya que es una tecnología de base de datos no relacional. Las bases de datos no relacionales, también llamadas NoSQL, son una elección a tener en cuenta cuando se trata de desarrollara aplicaciones web debido a que tiene ciertas ventajas con respecto a las bases de datos relacionales (SQL) como las siguientes [18] [19] [20] [21]:

- Mayor flexibilidad. Las colecciones de documentos pueden guardar documentos con diferentes campos entre ellos, es una capacidad muy importante cuando hay una evolución constante de las estructuras de datos que la aplicación persiste.

- Mayor velocidad y mayor capacidad de escalado. Debido a que no existen relaciones entre las entidades la capacidad de escalado horizontal es mucho mayor con respecto a una base de datos relacional. De está forma el coste es mucho más reducido con respecto a la escalabilidad vertical que se da en estas últimas.

Express.js

Express.js es un Framework para el desarrollo de aplicaciones web que trabajo sobre Node.js, es considerado el Framework estándar de facto para programar en Node.js [22].

Se corresponde con la capa de lógica de negocio de la aplicación, es la capa donde se aglutina la mayor parte del procesamiento y todos los procesos relacionados con las seguridad. A su vez es la parte del servidor, *Backend* y se ejecutará en maquinas distintas a las del usuario de la aplicación web.

Las principales ventajas que esta tecnología presenta para el desarrollo de la aplicación con respecto a Framework de *Backend* de otros lenguajes de programación populares como sería el Framework Spring de Java [23][24] son:

- Más sencillo y rápido de configurar.
- Menos verboso. Debido a su naturaleza, al ser JavaScript, un lenguaje no tipado y pensado inicialmente para *Frontend* resulta mucho más rápido crear una API con este Framework.
- Mejor rendimiento y menos consumo de memoria y espacio en disco. Debido en su mayor medida a que Node.js corre directamente sobre el sistema en contraposición a Java que se ejecuta sobre JVM (Java Virtual Machine).
- Mejor integración con base de datos no relacionales. Debido a la frecuencia con que bases de datos no relacionales como MongoDB son usados junto a frameworks de Node.js como Express.js existen librerías que facilitan en gran medida el proceso de integración.

Angular

Angular es un Framework para *Frontend* basado en TypeScript. TypeScript es un lenguaje escrito sobre JavaScript (un superset) que añade herramientas de programación orientada a objetos a este último [25].

Se corresponde con la capa de presentación, ya que es un Framework diseñado para servir los documentos HTML, interactuar con ella y navegar por los distintas vistas HTML.

Angular es un Framework propiedad de Google y es uno de los frameworks preferidos para el desarrollo de la interfaz de aplicaciones web. Algunas de sus ventajas sobre Aurelia.js, Framework de JavaScript que se ha visto anteriormente en el capítulo Estado del arte son las siguientes:

- Expansión mucho mayor. Debido a ser un Framework de Google y uno de los más populares entre los desarrolladores de aplicaciones web cuenta con una comunidad mucho más grande y un equipo de desarrolladores trabajando continuamente en su mantenibilidad y el aumento de sus funcionalidades.
- Configuración sobre convención. Un punto conflictivo, ya que resulta en más trabajo de configuración con respecto a frameworks que se basan en

convenciones como Aurelia. Sin embargo, esto permite un total control y personalización del mismo.

- Tiene soporte en la mayoría de navegadores existentes, tanto modernos como no tan recientes, a diferencia de Aurelia que no es soportado por los navegadores más antiguos, impidiendo así su funcionamiento.

Algo muy importante a indicar es que el Framework de Angular sufrió una reestructuración completa en la totalidad de su funcionamiento por parte del equipo de Google que supuso un cambio sin posibilidad de actualizar y de está forma el paso de AngularJS a Angular 2. A partir de esta última todos los cambios han sido menores y se ha podido actualizar de forma automática. Para el desarrollo del proyecto se hará uso de Angular en su versión 7. Algunos de los cambios que se introdujeron fueron [26]:

- Se cambio de una programación orientada a módulos a una programación orientada a componentes, siendo estos últimos mucho más sencillos y efectivos a la hora de reutilizar código dentro de la aplicación.
- Paso de utilizar JavaScript a utilizar TypeScript (superset de JavaScript) por lo que se le añaden herramientas para la programación orientada a objetos que facilitan en gran medida la programación orientada a componentes.
- Se introduce una consola interactiva con la que la creación de los diferentes elementos del software, así como la creación del proyecto, el despliegue y la depuración se vuelven mucho más sencillos.

NodeJS

NodeJS es un Framework de JavaScript que sirve como plataforma para la parte del servidor de la aplicación, el *Backend*.

Debe su popularidad a ser un Framework de JavaScript sacado de su contexto de ejecución habitual, ya que este es un lenguaje diseñado para ser ejecuta por los navegadores web. Sin embargo, NodeJS permite la ejecución de aplicaciones desde el punto de vista del servidor [27].

Debido a que es un Framework de JavaScript, se comunica con la capa de presentación mediante datos en forma de JSON (*JavaScript Object Notation*), por lo que la implementación más común es en forma de API REST.

La arquitectura REST (*Representational State Transfer*) se basa en un protocolo cliente/servidor sin estado, es decir, cada comunicación entre cliente y servidor contiene toda la información necesario, por lo que es necesario mantener ningún tipo de relación entre cliente y servidor. Esto permite la posibilidad de un servidor responda de manera rápida y eficaz a peticiones recibidas de diferentes clientes. Más información sobre esta implementación en la sección de Programación.

6.1.2. Librerías utilizadas

Una librería de software es un conjunto de implementaciones funcionales desarrolladas en cierto lenguaje de programación que ofrece una interfaz para inter-

actuar con ella. Su objetivo principal es ser usada por otros programas (nunca de forma autónoma) y facilitan la implementación de ciertas funcionalidades.

Los diferentes frameworks que componen el Stack MEAN importan consigo multitud de librerías para implementar sus funcionalidades. Maś allá de estas, en este proyecto se hará uso de las siguientes librerías externas:

Mongoose Librería usada en la parte del servidor que facilita las operaciones entre la base de datos MongoDB y la capa de lógica de negocio que ejecuta NodeJS. Las principales ventajas son que trabaja con JSON y además la creación de colecciones automáticamente creando esquemas de cada entidad [28].

Mongoose Unique Validator Es una librería (podría considerarse plugin) que añade funcionalidad a la librería *mongoose* añadiendo validación explícita cuando se guarda un objeto utilizando la librería *mongoose* [**?**].

Bcrypt Es una librería utilizada en la parte del servidor cuya utilidad es encriptar las contraseñas que se persisten en la base de datos antes de guardarlas [30].

La encriptación se realiza mediante una función de *hash* basado en el cifrado *Blowfish*[31].

JsonWebToken Librería utilizada por Express.js cuya función es la generación de tokens de acceso. Este token es firmado por la clave del servidor por lo que tanto cliente como servidor guardan constancia de la validez del token. Los token pueden dar diferentes permisos a los usuarios y pueden tener una fecha de caducidad [32].

Body Parser Librería utilizada por Express.js como *Middleware* para tratar tanto los parámetros como el cuerpo de la petición HTTP convirtiendo estos en objetos JSON, de esa forma facilitando la comunicación entre capas del software [33].

Angular Material Components Librería utilizada por Angular que añade componentes de interfaz siguiendo los estilos y estándares de *material-design*. Añade además componentes que facilitan la estructura de los menús y la creación de los formularios [34].

Angular Calendar Librería utilizada en la capa de presentación por el Framework Angular para añadir la lógica de un calendario que maneje eventos en fechas determinadas. Es posible enganchar eventos de Angular a la interacción con él, siendo posible de esta manera añadir y borrar elementos del calendario [35].

Angular Maps Librería utilizada por Angular que proporcionándole una clave válida de Google Cloud Platform para poder acceder a su API, proporciona la posibilidad de añadir un mapa del mundo de Google Maps embebido en la aplicación web [36].

apiDoc Librería utilizada en la parte del servidor cuya funcionalidad es crear documentación automáticamente a partir de anotaciones estándar añadidas en los *Endpoints* en Express.js. Esta librería genera su propia vista HTML con la documentación completa de la API [37].

6.1.3. Control de versiones

Un sistema de control de versiones tiene el objetivo de gestionar los diversos cambios que se realizan sobre los elementos de un software. Una versión es llamado al estado en el que se encuentra el producto en un momento de desarrollo específico [38]. De esta forma los sistemas de control de versiones y prácticamente imprescindibles para el desarrollo en equipo de un mismo proyecto.

Estos sistemas proporcionan lo siguiente:

- Mecanismos de almacenamiento de los elementos a gestionar, por lo que persiste todas y cada una de las versiones por las que el desarrollo de software ha pasado.
- Posibilidad de hacer cambios sobre los elementos. Debido a que almacena todas las versiones del software, es posible comparar versiones y realizar cambios en todo momento.
- Generación de informes de los cambios introducidas en cada versión.

Hay multitud de sistemas de control de versiones, cada uno con sus ventajas y desventajas, para el desarrollo de este proyecto se va a hacer uso de Git.

Git es un sistema de control de versiones distribuido diseñado por Linus Torvals. Es a día de hoy el control de versiones más expandido en todo el mundo, algunas de sus ventajas competitivas con respecto a otros controles de versiones populares como Subversion son los siguientes:

- Las diferentes ramas de desarrollo tan solo son una referencia a cierta versión del software, por lo tanto el proceso de creación, modificación y borrado de ramas es sumamente sencillo y ágil [39].
- Cada desarrollador tiene un repositorio local a parte del repositorio remoto donde se encuentra el código fuente compartido. De esta forma es muy sencillo y poco conflictivo el desarrollo en equipo [40].

Gran parte de la ventaja de Git es su carácter distribuido, esto se consigue gracias a la eficacia del desarrollo basado en ramas, en este proyecto se va a utilizar esa estrategia.

El desarrollo en ramas, como podemos ver en la figura 6.2, consiste en crear una copia en local de la rama remota principal, llamada *master*. En esta nueva rama se realizan los cambios que precise la nueva versión del software y , tras esto, se realiza una operación de fusión de nuevo con la rama principal. En esta operación de fusión se pueden revisar los cambios realizados durante el desarrollo de la rama y resolver los conflictos (en caso de que los hubiere) que hayan podido

suceder tras la mezcla de otras ramas con la rama remota principal (*master*). Esto permite el desarrollo de nuevas funcionalidades de forma distribuida y paralela con la seguridad de la consistencia entre estas debido a la resolución de conflictos antes de la fusión de las ramas [41].

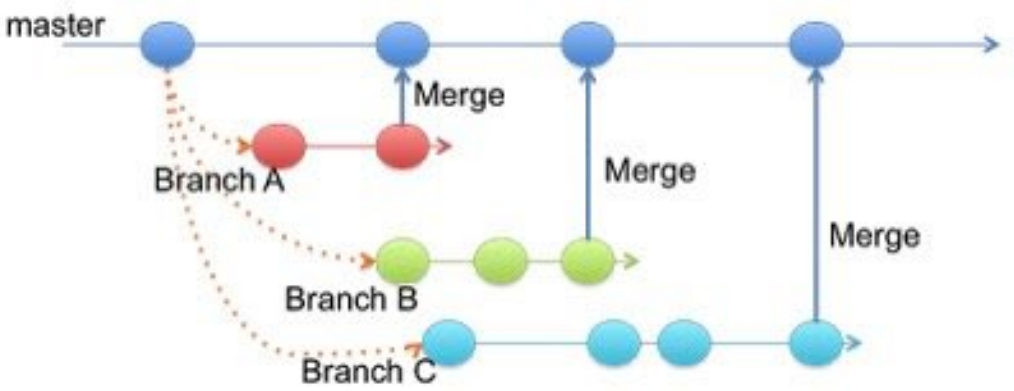

Figura 6.2: Git. Estrategia de desarrollo en ramas
Fuente:
https://www.javacodegeeks.com/2015/11/git-branching-strategies.html

6.2 Estructura de ficheros

Una parte muy importante en el desarrollo de aplicaciones es la estructura de ficheros del proyecto. Es recomendable seguir estándares y buenas prácticas con el fin de facilitar el desarrollo y la navegación entre ficheros durante la fase de desarrollo [42].

Debido a que el desarrollo de aplicaciones web contempla una gran cantidad de archivos de diferentes formatos y con diferentes objetivos es muy importante prestar atención a este factor.

Entre las diferentes modalidades de estructuración de ficheros se encuentra dos muy importantes, estructura sintáctica y estructura semántica.

Por un lado la estructura sintáctica consiste en estructurar los ficheros con respecto al formato de sus archivos, como se puede observar en la figura 6.3. Esta estrategia resulta sencilla de seguir y útil para proyectos con tamaño reducido.

Por otro lado la estructura semántica, figura 6.4, consiste en estructurar los ficheros según la funcionalidad que proporcional los diferentes elementos del software. Esta estrategia es más útil para proyectos con gran cantidad de funcionalidades.

En este proyecto, debido a que se sigue una arquitectura cliente-servidor, la parte del sistema del cliente y la del servidor serán totalmente independientes por lo que cada una tendrá su propia estructura.

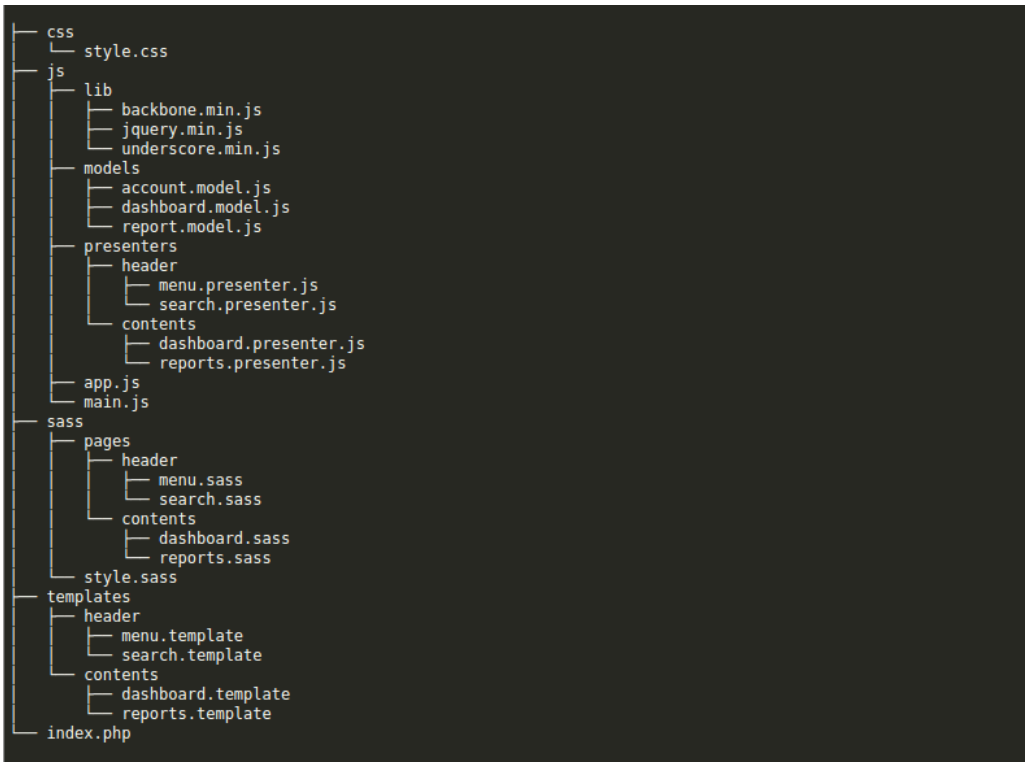

Figura 6.3: Estructura de ficheros sintáctica
Fuente: http://www.zsoltnagy.eu/web-application-file-structures/

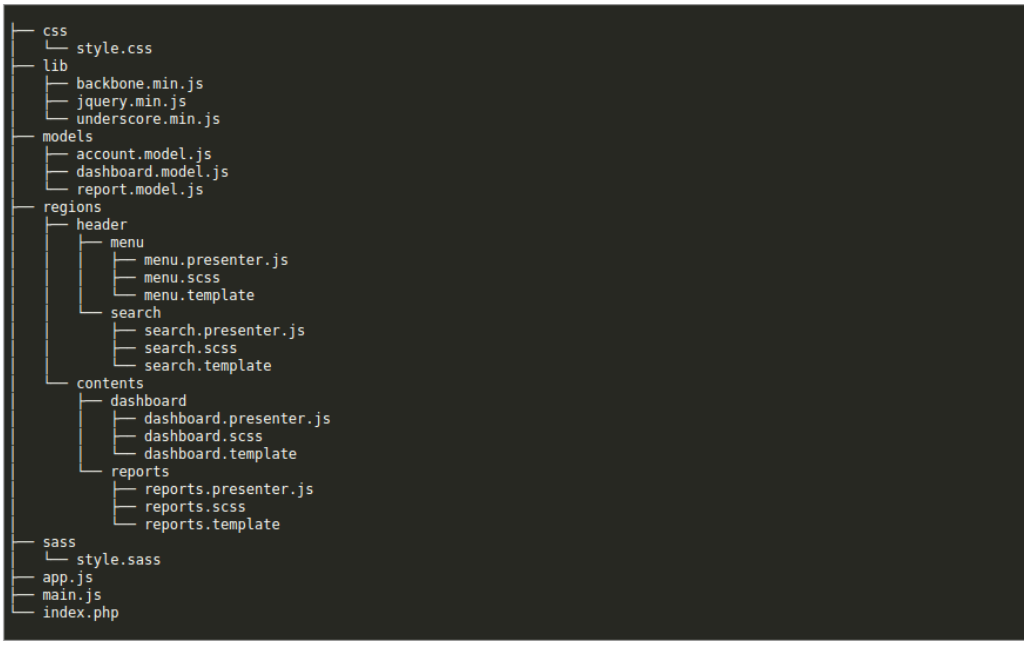

Figura 6.4: Estructura de ficheros semántica
Fuente: http://www.zsoltnagy.eu/web-application-file-structures/

6.2.1. Frontend

La parte del cliente se desarrolla utilizando el Framework Angular, el cual sigue un esquema por componentes, por lo que la estructura innata es basada en la semántica.

Como se puede observar en la figura 6.5, cada carpeta representa una agrupación de elementos funcionales de la aplicación. Cada elemento funcional es un fichero de componente que contiene todos los archivos necesarios para su funcionamiento.

- CSS. Estos archivos contienen las clases de estilos que aplican a los documentos HTML.
- HTML. Son los documentos que el navegador muestra con los cuales el usuario puede interactuar.
- TS. Archivos TypeScript donde se encuentra la lógica para la interacción y navegación entre documentos HTML.
- SPEC.TS. Archivos de testeo unitario para la lógica del componente.

Además de los ficheros de componente, existen los archivos de módulo y de servicio.

- Módulo. Archivos núcleo de la aplicación donde se aglutinan todas las importaciones y exportaciones de otros módulos, servicios y componentes para construir la estructura lógica de la aplicación.
- Servicio. Proporcionan funcionalidad específica a los componentes, están formados tan solo por un archivo de TypeScript ya que no contienen interacción con el usuario. La lógica de estos archivos es la que envía las peticiones de datos al servidor.

También se puede observar que para los elementos reutilizados por otros componentes de la aplicación exsite una carpeta compartida (*shared*) que contiene componentes, modelos y servicios que son utilizados por otros componentes diferentes en toda la aplicación.

6.2.2. Backend

La parte del servidor presenta una estructura semántica muy sencilla, debido en gran parte a la simplicidad del Framework Express.js. Se pueden observar los diferentes elementos:

- Rutas (*routes*). Fichero con archivos que enumeran las diferentes rutas de los *Endpoints*.
- *Middleware*. Fichero que aglutina la lógica de los diferentes middlewares que se aplican antes del comienzo de la lógica de las mismas.

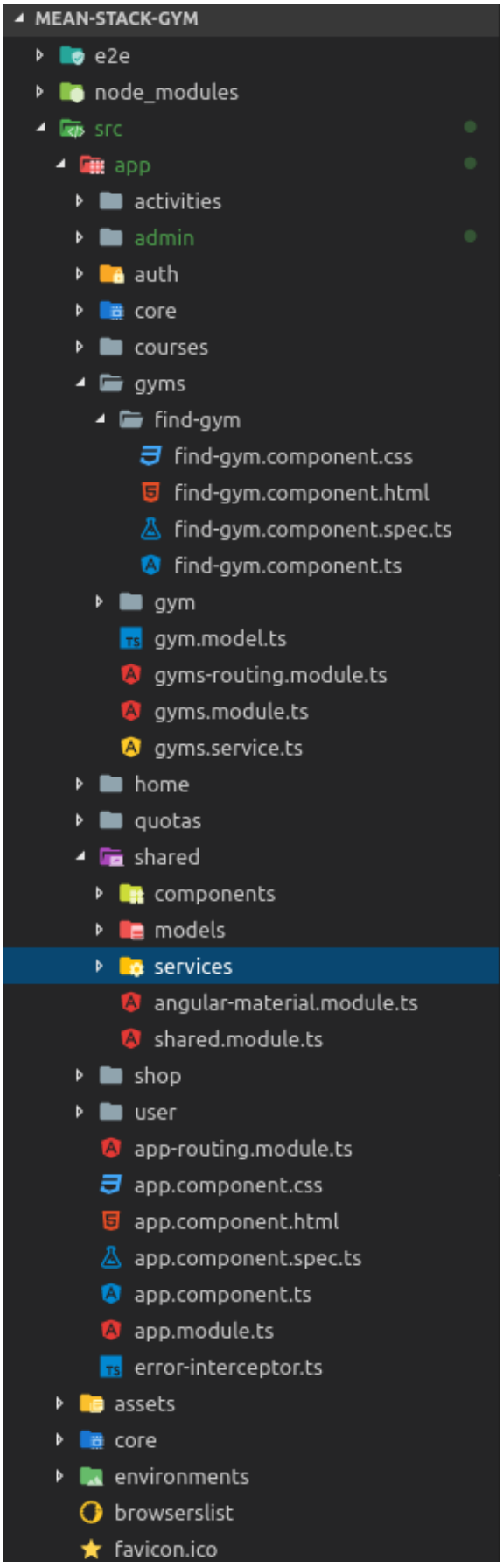

Figura 6.5: Estructura de ficheros del frontend

- Controladores (*controllers*). Fichero donde se encuentra los archivos que contienen la lógica que ejecutan las diferentes rutas. Estos archivos son los encargados de realizar cambios y recuperar información en la base de datos comunicándose con la capa de persistencia.

Cabe destacar que existe otra carpeta, llamada *REST* donde se implementa la parte RESTful de la aplicación. Esta interfaz RESTful de la aplicación tiene su propia estructura, equivalente a la del resto del *Backend* por que funciona totalmente independiente y su objetivo no es resolver las peticiones de la capa de presentación, sino que permite la interacción de sistemas externos directamente con el servidor mediante peticiones HTTP y enviando datos median JSON. Más información en la sección de Interfaz RESTful.

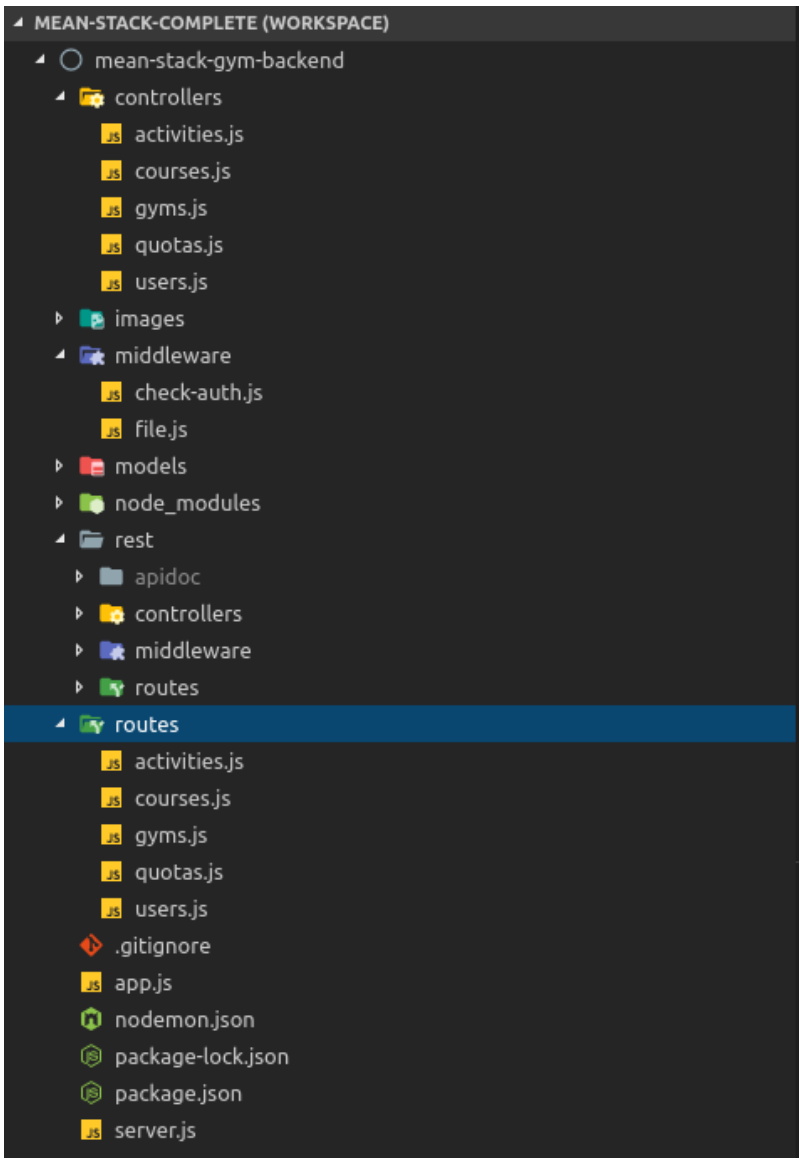

Figura 6.6: Estructura de ficheros del backend

6.3 Programación

En está sección se realizará una breve explicación de los algoritmos más importantes desarrolladas para el funcionamiento de la aplicación.

6.3.1. Seguridad

Para garantizar la seguridad de la aplicación el proceso de autenticación es una parte primordial. Tanto la creación del usuario, como la autenticación y la gestión del token son parte de este proceso.

Dado que el desarrollo de la aplicación se divide entre la parte del cliente y la del servidor, se ilustrará el flujo seguido en las dos diferentes partes, tanto el *Frontend* como el *Backend*.

Frontend

Obtención del token En primer lugar se obtendrá el token para enviar peticiones al servidor como usuario registrado. Esto se realiza mediante una petición de autenticación en la que se envían las credenciales del usuario al servidor, como se puede observar en el siguiente algoritmo.

```
/*
* Metodo empleado para autenticar al usuario y guardar el token
*/
login(email: string, password: string) {
    this.http
      .post<{ token: string; expiresIn: number; userId: string;
          userAdmin: boolean }>(
        BACKEND_URL + '/login',
        {email, password}
      )
      // Subscripcion a la respuesta que llega del servidor
      .subscribe(
        response => {
        // Token para validacion de usuario
          const token = response.token;
          this.token = token;
          // Si el token existe, se guardan los datos de validacion en
              el localStorage
          if (token) {
            const expiresInDuration = response.expiresIn;
            // Se lanzara un evento para desloguearse cuando el token
                caduque
            this.setAuthTimer(expiresInDuration);
            this.isAuthenticated = true;
            this.userId = response.userId;
            this.userAdmin = response.userAdmin;
            this.authStatusListener.next(true);
            const now = new Date();
            const expirationDate = new Date(
              now.getTime() + expiresInDuration * 1000
            );
            // Se guardan los datos de validacion
```

```
30            this.saveAuthData(token, expirationDate, this.userId, this.
                  userAdmin);
31            // Se navega a la vista dependiendo del rol del usuario
32            if (this.userAdmin) {
33              this.router.navigate(['admin']);
34            } else {
35              this.router.navigate([`/user/${this.userId}`]);
36            }
37          }
38        },
39        // Si el servidor envia un error, se comunica al usuario que
              los credenciales no son v lidos
40        error => {
41          this.authStatusListener.next(false);
42          this.router.navigate(['auth/login'], { queryParams: { auth: '
                invalid'}});
43        }
44      );
```

Algoritmo 6.1: Petición de autenticación de usuario

Tras recibir la respuesta del servidor, en caso de que esta sea satisfactoria, se guarda el token, el identificador del usuario y sus roles en el *localStorage* (datos que el navegador puede guardar) para poder se accedidos la próxima vez que el usuario utilice la aplicación.

Además de esto, se programa un evento que se lanzará cuando el token caduque para que abandone el registro de la aplicación automáticamente. Si el usuario quiere seguir teniendo acceso a su perfil deberá volver a enviar una petición de autenticación con sus credenciales. Esto se realiza de tal forma para asegurar que un posible token robado tenga limitada su acción en el tiempo.

Guarda en los componentes que precisan usuario registrado Pese a que la opción de AREA DE CLIENTE tan solo se muestra cuando un usuario está registrado, alguien podría intentar manipular la URL de la aplicación e intentar navegar a la ruta que carga el componente del perfil del usuario. Para evitar que esto suceda, se implementa una guarda que protege la carga de este componente, de tal forma que tan solo permite cargar el componente si el usuario está autenticado.

```
1  /*
2  * Metodo que intercepta la carga del componente de perfil de usuario y
       lo permite tan solo si el usuario est  autenticado en el sistema
3  */
4  canActivate(
5      route: ActivatedRouteSnapshot,
6      state: RouterStateSnapshot
7    ): boolean | Observable<boolean> | Promise<boolean> {
8    // Comprueba si el usuario esta autenticado
9      const isAuth = this.authService.getIsAuth();
10     // Si no esta autenticado, navega a la pagina de login
11     if (!isAuth) {
12       this.router.navigate(['/auth/login']);
13     }
14     return isAuth;
15   }
```

Algoritmo 6.2: Guarda del componente de perfil de usuario

Interceptor de peticiones HTTP Una vez el usuario está autenticado y ha obtenido un token válido, debido a que no se guarda estado entre cliente y servidor, es necesario que en la petición HTTP demuestre al servidor que de hecho las peticiones se realizan por un usuario registrado.

Para facilitar esta comunicación, se programa un interceptor que se activará con cada petición HTTP saliente y que, en caso de tener un token guardado en el *localStorage*, lo añadirá automáticamente a la cabecera de *Authoritation* de la petición HTTP para que el servidor verifique que el usuario está registrado.

```
/*
* Metodo que intercepta las peticiones \gls{HTTP} salientes y anade la
    cabecera de autenticacion con el token que el navegador tiene
    almacenado en el localStorage
*/
intercept(req: HttpRequest<any>, next: HttpHandler) {
    // Se obtiene el token del localStorage
    const authToken = this.authService.getToken();
    // Se anade la cabecera a la peticion HTTP y se reanuda su envio
    const authRequest = req.clone({
      headers: req.headers.set('Authorization', 'Bearer ' + authToken)
    });
    return next.handle(authRequest);
  }
```

Algoritmo 6.3: Interceptor de peticiones HTTP

Backend

Autenticación de usuario Cuando el *Frontend* envía una petición HTTP para autenticar a un usuario, el *Backend* realiza una serie de procesos para comprobar que las credenciales son válidas y en ese caso generar un token nuevo firmado y enviarlo de vuelta al cliente.

Como se puede observar en el algoritmo, en primer lugar comprueba si el usuario existe el la base de datos del sistema, para esto se comunica con la capa de persistencia de la aplicación. En caso afirmativo, compara la contraseña introducida con la contraseña encriptada que se encuentra guardada en la base de datos. Si esta última comprobación es exitosa, genera un nuevo token firmándolo con su clave y con los datos del usuario, le establece una fecha de caducidad y lo envía como respuesta de vuelta al cliente.

```
exports.userLogin = (req, res, next) => {
  let fetchedUser;
  // Comprueba si el usuario existe en el sistema
  User.findOne({ email: req.body.email })
    .then(user => {
      if (!user) {
        return res.status(401).json({
          message: "Auth failed"
        });
      }
      fetchedUser = user;
      // Comprueba que la contrasena sea valida
      return bcrypt.compare(req.body.password, user.password);
    })
```

```
    .then(result => {
      if (!result) {
        return res.status(401).json({
          message: "Auth failed"
        });
      }
      // Crea un token firmado con la clave del servidor y los datos
         del usuario, con fecha de caducidad
      const token = jwt.sign(
        { email: fetchedUser.email, userId: fetchedUser._id },
        process.env.JWT_KEY,
        { expiresIn: "1h" }
      );
      // Expide el token
      res.status(200).json({
        token: token,
        expiresIn: 3600,
        userId: fetchedUser._id,
        userAdmin: fetchedUser.isAdmin,
      });
    })
    .catch(err => {
      return res.status(401).json({
        message: "Credenciales no v lidos"
      });
    });
};
```

Algoritmo 6.4: Autenticación de usuario en el servidor

Middleware de comprobación de autenticación Para asegurar el acceso a ciertas rutas (*Endpoints*) limitadas a usuarios registrados, el servidor aplica un *Middleware* que, antes de ejecutar la lógica de la ruta accedida, comprueba que en la cabecera de la petición llega un token y este es válido. En caso de que este proceso falle significará que no se encuentra token o bien no es válido o ha caducado, por lo tanto, en ninguno de los casos se ejecutará la petición y se enviará un error de autenticación al cliente.

```
// Para editar un usuario se anade un \textit{\gls{Middleware}} que
   comprueba la autenticacion
router.put("/edit", checkAuth, UserController.updateUser);
```

Algoritmo 6.5: Aplicación de middleware de autenticación

```
 // Se extrae el token de la cabecera y se verifica su validez
 try {
    const token = req.headers.authorization.split(" ")[1];
    const decodedToken = jwt.verify(token, process.env.JWT_KEY);
    req.userData = { email: decodedToken.email, userId: decodedToken.
       userId };
    // En caso de que el token sea valido, deja procesar la peticion
    next();
    // En caso de que el token no sea valido, se envia un error al
       cliente
  } catch (error) {
    res.status(401).json({ message: "You are not authenticated!" });
  }
```

Algoritmo 6.6: Middleware de comprobación de autenticación

6.3.2. Interfaz RESTful

Esta parte de la lógica de negocio es totalmente independiente de la capa de presentación y su función es ser alcanzada por peticiones de sistemas externos, por esa razón se implementa siguienda una interfaz RESTful.

Una API RESTful se basa en un protocolo cliente/servidor sin estado, es decir, cada comunicación entre cliente y servidor contiene toda la información necesario, por lo que es necesario mantener ningún tipo de relación entre cliente y servidor.Esta API recibe peticiones HTTP y envía datos en forma de JSON (JavaScript Object Notation).

Obtención del token Para la obtención del token se sigue la misma lógica que en la petición de autenticación de la parte web del *Backend* Autenticación de usuario. En este caso los credenciales de usuario se envían en el cuerpo de la petición en un JSON.

Seguridad de acceso

Esta interfaz REST permite obtener información potencialmente útil para sistemas de control externos como pueden ser los detalles de los usuarios, de las actividades dirigidas y de las clases impartidas de cada actividad pero además de esto, se hace hincapié en dos peticiones de comprobación de seguridad de acceso que permiten a sistemas externos dar o denegar acceso a los usuarios del sistema.

- **Acceso de un usuario al centro deportivo**. Se realiza mediante una petición enviando el identificador del usuario en la petición HTTP, el sistema responderá de forma afirmativa o negativa.

  El sistema comprueba en primer lugar si ese usuario está dado de alta en el sistema, es decir, existe en los registros de base de datos y en ese caso comprueba la fecha de validez de la cuota, si la fecha de validez es posterior a la fecha actual entonces envía una respuesta satisfactoria.

  Cabe destacar que si la cuota del usuario corresponde a la de 1 día de prueba gratis gratis, la fecha de validez se modifica al día actual, ya que esta cuota expira en el momento en que el usuario accede al centro deportivo por primera vez.

- **Acceso de un usuario a una clase**. En este caso se realiza mediante una petición que envía como parámetros tanto el identificador del usuario como el de la clase específica a la que se quiere acceder.

  El sistema realiza una comprobación de que existe el usuario, en ese caso comprueba en los registros de la colección de reservas en la base de datos

si el usuario tiene una reserva válida para esa clase en ese día exacto de la semana. En caso afirmativo, envía una respuesta satisfactoria.

```
exports.getUserAccess = (req, res, next) => {
  let access = false;
  const userId = req.params.userId;
  // Se comprueba que el usuario existe
  const getQuery = User.findOne({_id: userId}).populate('quota');
  getQuery.then((user) => {
  // Se comprueba si la fecha de validez de su couta es posterior a la actual
    if (user.endDate > new Date()) {
      access = true;
      // si es cuota de un dia de prueba se modifica el fin de validez al momento actual
      if (user.quota.periodInMonths === 0) {
        User.findOneAndUpdate({_id: userId}, { $set: {endDate: new Date()}}).exec();
      }
    }
    res.status(200).json({
      access
    });
  }).catch(error => {
    res.status(404).json({
      message: "Fetching users failed!"
    });
  });
}
```

Algoritmo 6.7: Comprobacíon de acceso de un usuario al centro deportivo

```
exports.getUserAccessToEvent = (req, res, next) => {
  let access = false;
  const userId = req.query.userId;
  const eventId = req.query.eventId;
  // Se comprueba si el usuario existe
  const getQuery = User.findOne({_id: userId}).populate('quota');
  getQuery.then((user) => {
    let start = new Date();
    start.setHours(0,0,0,0);
    let end = new Date();
    end.setHours(23,59,59,999);
    // Se comprueba si existe una reserva del usuario en la clase
    Reservation.findOne({user: user._id, event: eventId, exactDate: {$gte: start, $lt: end}}).then((reservation) => {
      if (reservation) {
        access = true;
        res.status(200).json({
          access
        });
      } else {
        res.status(401).json({
          message: "No reservation found",
        })
      }
```

Algoritmo 6.8: Comprobación de acceso de un usuario a una clase

6.3.3. Documentación

Debido a la naturaleza independiente de esta API y con el objetivo del uso de esta interfaz RESTful por sistemas externos a nuestra aplicación, esta parte del software está documentada usando la librería apiDoc. De esta forma, cualquier sistema al que se le permita acceso mediante credenciales tendrá la información necesaria para poder interactuar con el sistema y realizar el control de presencia del centro deportivo. En el anexo Documentación API REST se encuentran las instrucciones para generar esta documentación.

```
6
7   /**
8    * @api {post} api/v1/auth Request Token
9    * @apiName GetToken
10   * @apiGroup Auth
11   *
12   * @apiParam (body) {String} email Users email.
13   * @apiParam (body) {String} password Users password.
14   *
15   * @apiParamExample {json} Request-Example:
16   * { "email": "email@email.com",
17   *    "password": "pass4me" }
18   * @apiSuccess {String} token Valid access token.
19   * @apiSuccess {Number} expiresIn Duration of the token in seconds.
20   * @apiSuccess {String} userId Id of the user authenticated.
21   * @apiSuccess {Boolean} userAdmin "true" if the user authenticated is admin of the system.
22   *
23   * @apiSuccessExample Success-Response:
24   *     HTTP/1.1 200 OK
25   *     {
26   *       token: "fsadfdSFASAfASDfsaFdsaFSA",
27   *       expiresIn: "3600",
28   *       userId: "32fSD453FDAERA"      You, a month ago • api-doc
29   *       userAdmin: "true"
30   *     }
31   *
32    * @apiError InvalidAuth The credentials were not valid.
33   *
34   * @apiErrorExample Error-Response:
35   *     HTTP/1.1 401 Invalid credentials.
36   *     {
37   *       "message": "Auth failed"
38   *     }
39   */
40  router.post("", AuthController.getToken);
41
```

Figura 6.7: *Endpoint* de obtención de token documentado con apiDoc

6.4 Interfaz

La interfaz se ha implementado sobre los bocetos mostrados en la sección Interfaz del capítulo Diseño. En está sección se mostrarán algunas de las ventanas más características de la aplicación como son la Ventana de centros deportivos y la Ventana de detalle de un centro deportivo.

Para la implementación de la interfaz se han utilizado los componentes de Material de Angular (*Angular Material Components*) `https://material.angular.io/components/categories`.

Son componentes de uso público diseñados y mantenidos por el equipo de Angular que son implementados siguiendo las directrices del Framework Angu-

lar y mantienen una composición conjunta siguiendo el estilo de diseño Material (*Material Design*).

Material Design es un lenguaje de diseño desarrollado por Google que apareció por primera vez en el 2014. Desde esta fecha hasta el momento actual se ha ido haciendo más popular su uso tanto en dispositivos móviles como en escritorio debido a su filosofía de sistema único [43] [44].

Las características principales que definen este lenguaje son la tipografía clara, colores sólidos, llamativos y el uso del eje Z para la organización de los elementos y la sensación de profundidad.

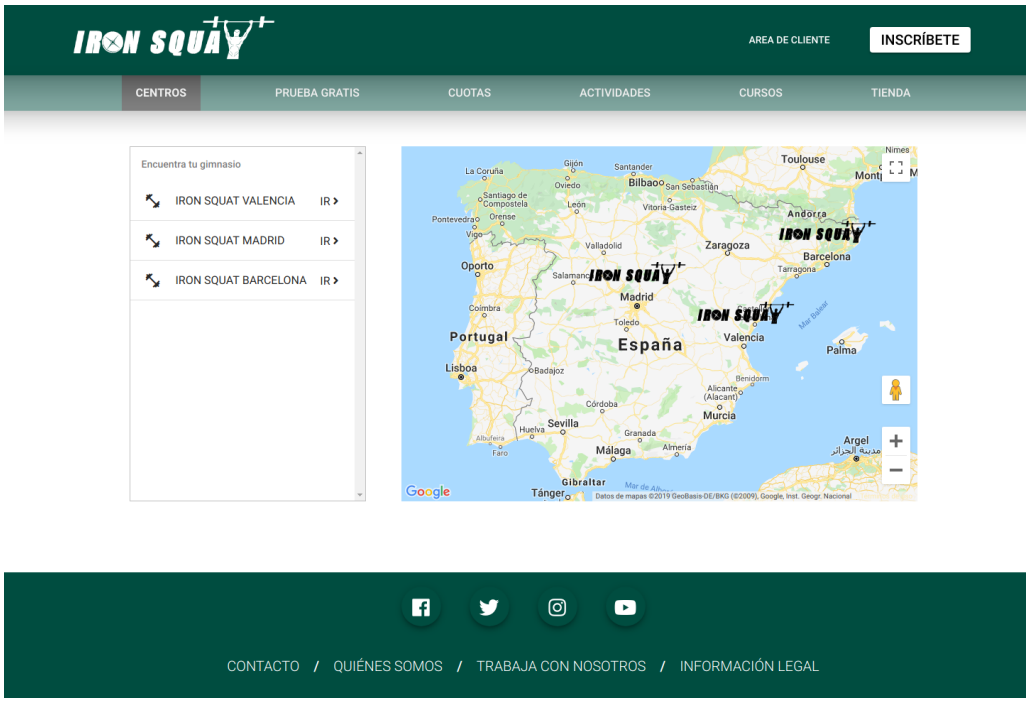

Figura 6.8: Ventana de centros deportivos

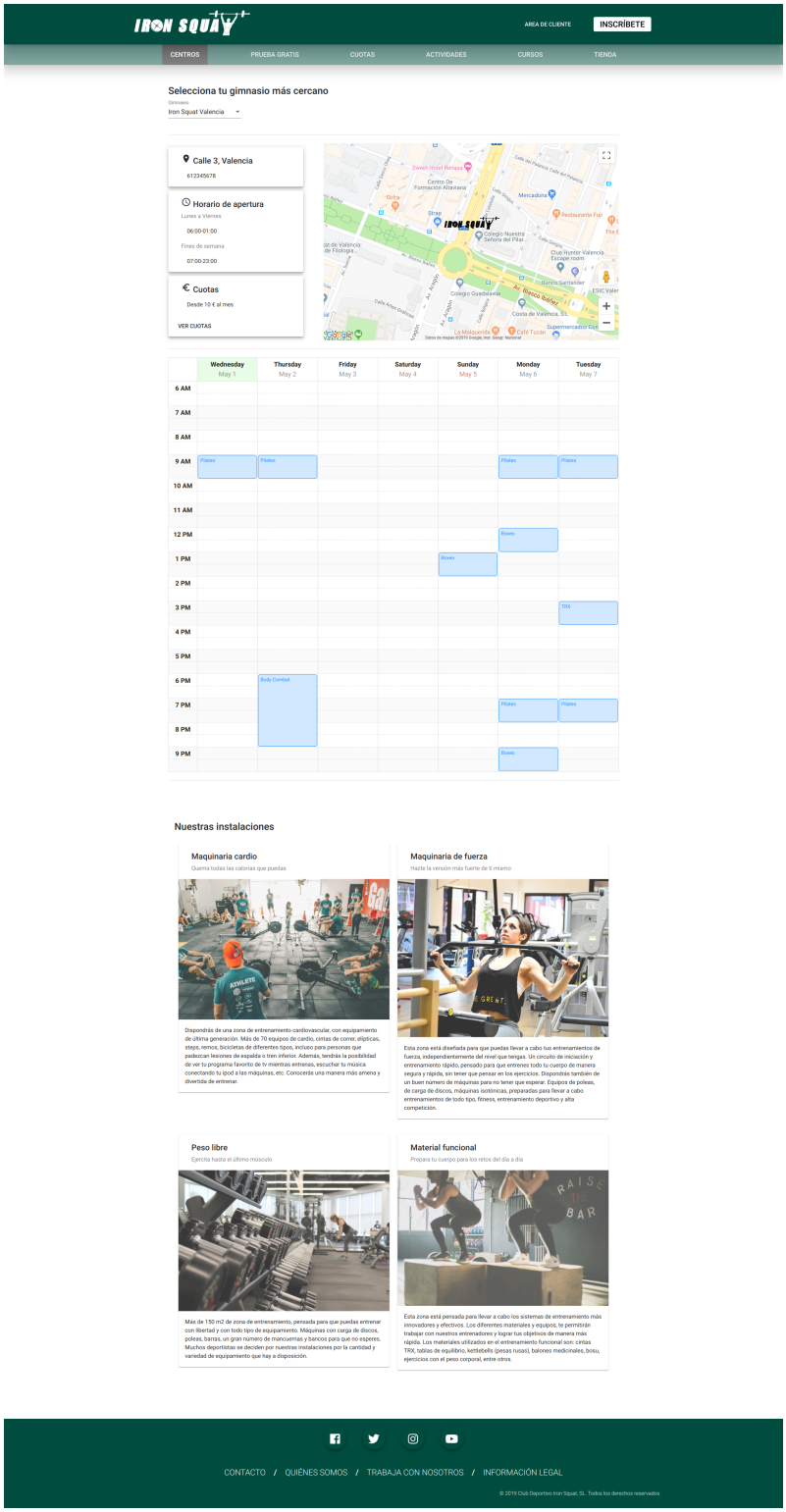

Figura 6.9: Ventana de detalle de un centro deportivo

CAPÍTULO 7
Pruebas

Para asegurar el correcto funcionamiento de los diferentes componentes de la aplicación desarrollada en todo momento y con el objetivo de facilitar la comprensión del código en un posible escenario de trabajo en equipo se añaden diferentes tipos de test que evalúan el correcto funcionamiento de la aplicación. De esta se aumenta en gran medida la capacidad de mantenibilidad del software.

Se van a implementar dos tipos diferentes de test: unitarios y de integración o punto a punto.

7.1 Pruebas unitarias

Las pruebas unitarias sirven para comprobar el correcto funcionamiento de una unidad de código [45]. Estos test aseguran que cada unidad funciona correctamente por sí sola. Se escriben diferentes casos de prueba que validan el funcionamiento en la mayor cantidad de escenarios posibles para esa unida de código. También son conocidos como pruebas de caja blanca porque se comprueba la funcionalidad interna que conforma el código [46].

Se va a mostrar un ejemplo de prueba unitaria en la programación del cliente utilizando el Framework ***Jasmine***.

Jasmine es un Framework de JavaScript diseñado para facilitar la implementación de pruebas unitarias [47]. Angular está integrado de forma nativa con este Framework y algunas de sus ventajas son las siguientes:

- Tiene soporte para pruebas asíncronas, por lo que se puede comprobar el funcionamiento de tareas que tardan tiempo en responder.
- Es muy sencillo de leer y de interpretar debido a su sintaxis explícita.
- Permite probar unidades de código independientes creando *espías* de las demás unidades proporcionándole de antemano el resultado de las interacciones con estas, pudiendo así aislar por completo el funcionamiento de la propia unidad de código a probar.

Se va a mostrar un ejemplo de prueba implementada para ilustrar las posibilidades que ofrece. Se puede observar en en siguiente algoritmo.

```
describe('HeaderComponent', () => {
  let component: HeaderComponent;
  let router: Router;
  let authService: AuthService;
  let fixture: ComponentFixture<HeaderComponent>;

    ...

  beforeEach(() => {
    fixture = TestBed.createComponent(HeaderComponent);
    component = fixture.componentInstance;
    router = TestBed.get(Router);
    authService = TestBed.get(AuthService);
    fixture.detectChanges();
  });

    ...

  it('should show user profile button when loged in', () => {
    spyOn(authService, 'getIsAuth').and.returnValue(true);
    spyOn(authService, 'getAuthStatusListener').and.returnValue(new
        Observable((observer: any) => {
      observer.next(true);
      observer.complete();
    }));
    fixture.detectChanges();
    expect(fixture.debugElement.query(By.css('.user-profile-button'))).
        not.toBeNull();
  });
    ...
});
```

Algoritmo 7.1: Prueba unitaria del componente cabecera (*header*)

En primer lugar, notese que con el fin de demostrar el funcionamiento de la herramienta se han recortado las partes del código donde se hacen las importaciones de los módulos necesarios y el resto de pruebas adicionales del componente.

Como se puede obsevar, cada clase de test comienza con un apartado *describe()* en el que se declaran los componentes y los servicios que se van a utilizar.

Más tarde en el apartado *beforeEach()*, que se ejecutará siempre antes de la ejecución de cada prueba (cada *it()*), se crean los *mocks* (simulaciones) de los componentes y servicios para manipular sus respuestas ya que en ningún momento se quiere acceder a ellos para de esta forma aislar el funcionamiento del componente al que se le aplican las pruebas.

Por último, una clase de test tendrá tantos apartados *it()* como pruebas realice sobre la unidad de código. Cada *it()* va seguido de una descripción en lenguaje natural de su objetivo de test. Como se puede observar en el ejemplo, se crean 'espías' de un servicio exterior para manipular de esta forma su respuesta y después se comprueba que existe un elemento de la forma esperada, en este caso el botón de acceso al perfil de usuario.

Una vez programadas las pruebas y para comprobar el resultado de las mismas, se ejecuta el comando 'ng test' utilizando la consola de Angular, genera el

informe 7.1 que muestra el resultado de todos las pruebas unitarias implementadas para las diferentes unidades de código.

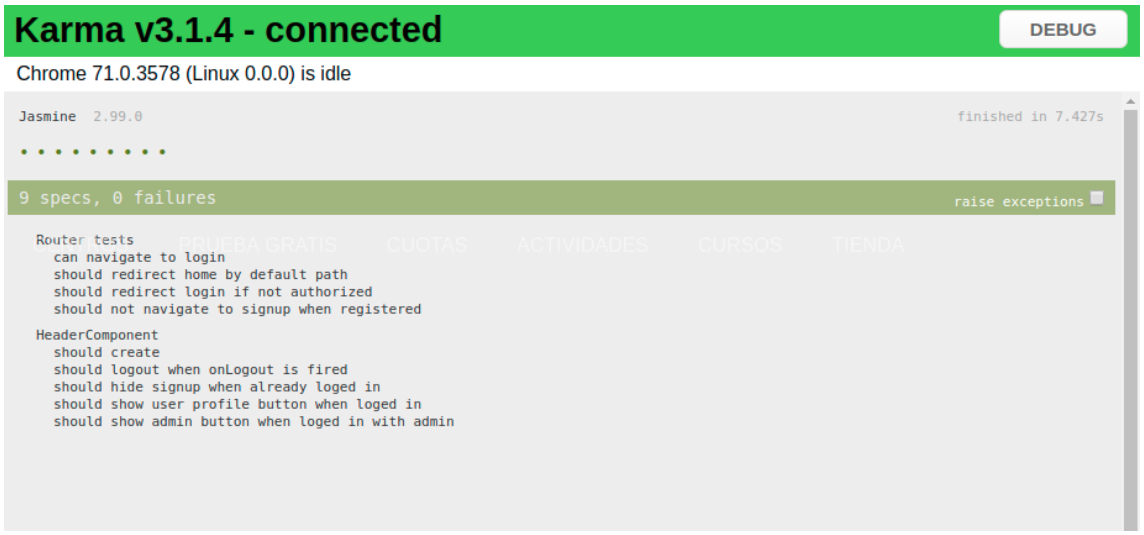

Figura 7.1: Informe de pruebas unitarias

7.2 Pruebas de integración

Las pruebas de integración, por otro lado, comprueban el correcto funcionamiento del flujo de la aplicación, que todas las unidades de código interactúan y están bien integradas entre si [48]. Comprueban que el resultado de la ejecución de la aplicación con diferentes entradas es equivalente a las salidas esperadas. Estas pruebas también son llamadas pruebas de caja negra [49] debido a que no se tiene en cuenta en ningún momento el funcionamiento interno de la aplicación, tan solo su resultado a partir de las entradas proporcionadas.

Para demostrar la funcionalidad se va a mostrar un ejemplo de prueba punto a punto *e2e (end to end)* en el proyecto. Para esto se va a hacer uso de la herramienta ***Protactor*** que está totalmente integrada con Angular de forma nativa y permite lanzar un servidor en local en el cual se ejecuta de forma automática la aplicación en un navegador [50]. En el navegador se ejecutan los comandos que se le programan de forma lineal a la prueba y tras eso comprueba que los resultados sean los esperados.

```
1    ...
2    describe('Auth App Desktop', () => {
3      let page: AuthPage;
4
5      ...
6
7  it('Should login when you click on login and navigate to the user
       area', async () => {
8      // LOGIN
9      page.navigateToLogin();
10     await browser.driver.navigate().refresh();
11     page.getInputUserNameLogin().sendKeys('v41196@gmail.com');
12     page.getInputPasswordLogin().sendKeys('palabra');
```

```
13      await page.getSendLoginButton().click();
14      // Redirige al login
15     browser.waitForAngularEnabled(false);
16     browser.sleep(2000);
17      browser.getCurrentUrl().then((urlUser => {
18        expect(urlUser).toContain('/user/5c8165f432e1ad76c969a7ac');
19      }));
20
21   });
22
23   it('Should navigate to activities', async () => {
24     await page.navigateToActivities();
25     browser.sleep(2000);
26     browser.getCurrentUrl().then((urlActivities => {
27       expect(urlActivities).toContain('/activities/at/any');
28     }));
29   });
30
31   it('Should logout when you click on logout', async () => {
32     await page.getLogoutButton().click();
33     browser.getCurrentUrl().then((urlHome) => {
34       expect(urlHome).toContain('/');
35     });
36     // No puede redirigir a explorer
37     await page.navigateToUser();
38     browser.getCurrentUrl().then((urlHome) => {
39       expect(urlHome).toContain('/');
40     });
41   });
42 });
```

Algoritmo 7.2: Prueba de integración punto a punto (*e2e*)

Finalmente la herramienta ***Protactor*** devuelve un informe de resultados obtenidos tras la ejecución de las pruebas punto a punto. Se puede ver en la figura 7.2

```
Jasmine started

  Auth App Desktop
    ✓ Should login when you click on login and navigate to the user area
    ✓ Should navigate to activities
    ✓ Should logout when you click on logout

Executed 3 of 3 specs SUCCESS in 17 secs.
[19:29:09] I/launcher - 0 instance(s) of WebDriver still running
[19:29:09] I/launcher - chrome #01 passed
```

Figura 7.2: Informe de pruebas de integración (punto a punto)

CAPÍTULO 8
Conclusiones

En el trabajo realizado se ha abordado un desarrollo de software en todas sus fases:

- Especificación de requisitos. Esta fase proporciona una visión global del proyecto y se centra en la descripción del problema desde el punto de vista de sus requisitos.
- Análisis. Se analizan desde diferentes puntos de vista los requisitos determinados en la fase anterior.
- Diseño. Se describe como el sistema va a satisfacer esos requisitos. Se diseña de esta forma la arquitectura del sistema.
- Implementación. En esta fase se codifica el software que hará operativo todo el sistema planteado en las fases anteriores. Está a su vez muy ligado con la fase de pruebas.
- Pruebas. Se desarrollan pruebas para verificar en todo momento el funcionamiento satisfactorio del software implementado en la fase anterior. Es muy común que esta fase se lleva a cabo simultáneamente con la de implementación y se retro-alimenten de forma cíclica.

Todo este desarrollo ha sido una mezcla de la aplicación de gran parte de los conocimientos obtenidos durante el estudio del Grado junto a información adicional obtenida tras la investigación sobre el ámbito del proyecto.

El uso del Stack MEAN para abordar el proyecto haciendo uso de tecnológias punteras de última generación ha sido un reto que ha supuesto un gran aprendizaje y una sensación muy satisfactoria tras la finalización de la fase de implementación del proyecto.

La posibilidad de utilizar un lenguaje como JavaScript (NodeJS) para realizar la parte servidor del sistema y el uso de bases de datos no relaciones como es MongoDB ha supuesto un aprendizaje adicional al obtenido durante la carrera.

Junto a esto, la realización de una aplicación web *Singlepage* desde su punto inicial y el formar parte del desarrollo de todos sus componentes (*Frontend*, *Backend* y persistencia de datos) y con ello la integración entre estos ha supuesto un gran ejemplo que refleja el flujo de trabajo en el mundo profesional.

8.1 Trabajo futuro

A lo largo del desarrollo del proyecto surgen diferentes ideas de mejora para el software desarrollado que no llegan a ser implementadas por estar fuera del ámbito del mismo.

Sin embargo, podrían ser futuras líneas de trabajo interesantes para proporcionar mejoras continuadas al funcionamiento y a la capacidad de desarrollo del sistema.

- Despliegue del sistema en un servidor remoto. Con el objetivo de hacer la aplicación accesible al público de forma global sería necesario realizar el proceso de despliegue. Debido a que el proyecto consiste en una *Stack* completa incluida la base de datos no hay posibilidad de realizar el despliegue de forma gratuita.

  Sin embargo, sin tener esto en cuenta, hay multitud de servicios *Cloud* que permiten desplegar la aplicación junto a la base de datos en servidores remotos como: IBM Cloud (https://www.ibm.com/cloud/blog/build-deploy-mean-stack-a) o Heroku (https://www.heroku.com/).

- Interfaz *responsive* para móvil. Dado el alcance de la aplicación desarrollada sería muy interesante la posibilidad de ser utilizada en un dispositivo móvil.

  Para esto Angular proporciona herramientas de programación de interfaz *responsive* que podrían reajustar los elementos de la interfaz según el tamaño de la pantalla del dispositivo donde se están mostrando.

  Además de esto, herramientas como ***Apache Cordova*** [51] permiten la generación de aplicaciones nativas de Android y iPhone a partir de un aplicación web en JavaScript como la desarrolla en este proyecto.

- Internacionalización y localización de la aplicación. La posibilidad de mostrar la aplicación en diferentes idiomas bajo elección del usuario podría aumentar el público objetivo de la aplicación desarrollada.

  Angular proporciona sus propias herramientas de i18n (internacionalización) que una vez utilizas permite la creación de múltiples *builds* de la aplicación con tantos idiomas como se haya configurado.

- Uso de patrón ***Redux*** para los estados globales de la aplicación. *Redux* es una librería JavaScript que inicialmente fue diseñada para implementarse junto a la tecnología React.js pero que más tarde fue introducida para ser usada con otros frameworks de JavaScript como es Angular.

  Esta librería consiste en una API que implementa el patrón de diseño Flux y sirve para mantener los estados globales de la aplicación, desacoplándolos de la parte visual y de esta forma poder acceder a ellos en cualquier parte de la programación [52]. Es una alternativa al uso de servicios para el mantenimiento de estados globales en aplicaciones grandes.

- ***Jest***, alternativa a Jasmine para pruebas unitarias. Jest es, como Jasmine, un Framework de JavaScript diseñado para la creación de pruebas, en este caso

nos centraremos en las unitarias. Esta alternativa es interesante porque ha demostrado ser más rápida en la ejecución de las diferentes prueba [53] con respecto a Jasmine.

Jest es una herramienta que también ha sido creada por el equipo de React.js pero que se puede utilizar en otros frameworks como Angular. Jest es una capa superior que hace uso de Jasmine por lo que la API es muy similar, lo que significa que la migración de test Jasmine a Jest es significativamente sencilla.

- Documentación de toda la aplicación. Esto facilitaría el desarrollo en equipo del proyecta ya que los nuevos recursos incorporados al equipo tendrían disponible la explicación en lenguaje natural del funcionamiento de todas las partes del sistema.

Bibliografía

[1] *Internet of Things*. Todo lo que necesitas saber sobre el *Internet of Things* (IoT). Consultado el 14/04/2019 en https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right

[2] *WebApps*. Ventajas de las aplicaciones basadas en web. Consultado el 14/04/2019 en https://www.magicwebsolutions.co.uk/blog/the-benefits-of-web-based-applications.htm.

[3] *JavaScript*. Porcentaje de uso del lenguaje *JavaScript* en comparación con otros lenguajes de programación. Consultado el 14/04/2019 en https://www.educba.com/uses-of-javascript/.

[4] Metodologías ágiles. Todo lo que necesitas saber sobre metodologías ágiles. Consultado el 14/04/2019 en https://luis-goncalves.com/what-is-agile-methodology/.

[5] Metodologías ágiles. Todo lo que necesitas saber sobre metodologías ágiles. Consultado el 14/04/2019 en http://scrummethodology.com/.

[6] Popularidad del uso de gimnasios. 8 razones por las que la popularidad de los gimnasios está creciendo exponencialmente. Consultado el 14/04/2019 en https://www.entrepreneur.com/article/296797.

[7] Top 10 webs de gimnasios españoles en Enero 2018. Consultado el 15/04/2019 en https://www.cmdsport.com/fitness/actualidad-fitness/top-10-webs-de-gimnasios-enero-2018/.

[8] Diferencias entre webs estáticas y dinámicas Consultado el 16/04/2019 en https://brandmedia.es/diferencias-pagina-web-estatica-dinamica-mejor/.

[9] Aurelia vs Anguar(2+) Consultado el 19/04/2019 en https://medium.com/@krishankataria2015/aurelia-vs-angular-2-why-i-prefer-aurelia-over-angular-a9c96a82c13d.

[10] Diagramas de caso de uso. Consultado el 27/04/2019 en https://www.infor.uva.es/~chernan/Ingenieria/Teoria/Tema3D.pdf.

[11] Diagramas UML. Consultado el 27/04/2019 en https://www.teatroabadia.com/en/uploads/documentos/iagramas_del_uml.pdf.

[12] Base de datos relacional. Consultado el 27/04/2019 en https://es.wikipedia.org/wiki/Clave_primaria.

[13] Programación en 3 capas Consultado el 30/04/2019 en https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas.

[14] Patrón de arquitectura MVC (Modelo-Vista-Controlador Consultado el 29/04/2019 en https://codigofacilito.com/articulos/mvc-model-view-controller-explicado.

[15] Arquitectura cliente/servidor Consultado el 30/04/2019 en https://es.wikipedia.org/wiki/Cliente-servidor.

[16] API REST: Qué es? Consultado el 30/04/2019 en https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos.

[17] Qué es el Stack MEAN Consultado el 02/05/2019 en https://www.campusmvp.es/recursos/post/Que-es-el-stack-MEAN-y-como-escoger-el-mejor-para-ti.aspx.

[18] Bases de datos NoSQL: Guía definitiva Consultado el 02/05/2019 en https://blog.pandorafms.org/es/bases-de-datos-nosql/.

[19] Fundamentos de bases de datos NoSQL: MongoDB Consultado el 03/05/2019 en https://www.campusmvp.es/recursos/post/Fundamentos-de-bases-de-datos-NoSQL-MongoDB.aspx.

[20] Bases de datos NoSQL : Guía definitiva Consultado el 03/05/2019 en https://blog.pandorafms.org/es/bases-de-datos-nosql/.

[21] NoSQL vs SQL; principales diferencias y cuándo elegir cada una de ellas Consultado el 03/05/2019 en https://blog.pandorafms.org/es/nosql-vs-sql-diferencias-y-cuando-elegir-cada-una/.

[22] Express.js Consultado el 03/05/2019 en https://en.wikipedia.org/wiki/Express.js.

[23] TOP 5 BACKEND FRAMEWORKS Consultado el 03/05/2019 en https://medium.com/@Kaperskyguru/top-5-backend-frameworks-a14a390151d2.

[24] Web Development Comparison: Spring Boot vs. Express.js Consultado el 03/05/2019 en https://dzone.com/articles/web-development-comparison-springboot-vs-expressjs.

[25] ¿Qué es TypeScript? Consultado el 03/05/2019 en https://devcode.la/blog/que-es-typescript/.

[26] AngularJS vs Angular 2 Consultado el 03/05/2019 en https://communities.bmc.com/docs/DOC-99523.

[27] MEAN, un stack 100 % Javascript Consultado el 03/05/2019 en https://www.arsys.es/blog/programacion/mean-javascript-cloud/.

[28] Object Modeling in Node.js with Mongoose Consultado el 03/05/2019 en https://devcenter.heroku.com/articles/nodejs-mongoose.

[29] Node Package Manager. mongoose-unique-validator Consultado el 03/05/2019 en https://www.npmjs.com/package/mongoose-unique-validator.

[30] Node Package Manager. bcrypt Consultado el 03/05/2019 en https://www.npmjs.com/package/bcrypt.

[31] Blowfish_(cipher) Consultado el 03/05/2019 en https://en.wikipedia.org/wiki/Blowfish_(cipher).

[32] JSON_Web_Token Consultado el 03/05/2019 en https://es.wikipedia.org/wiki/JSON_Web_Token.

[33] Node Package Manager. body-parser Consultado el 03/05/2019 en https://www.npmjs.com/package/body-parser.

[34] Angular Material Components Consultado el 03/05/2019 en https://material.angular.io/.

[35] Angular Calendar Consultado el 03/05/2019 en https://mattlewis92.github.io/angular-calendar/#/kitchen-sink.

[36] Angular Maps Consultado el 03/05/2019 en https://angular-maps.com/api-docs/agm-core/components/agmmap.

[37] APIDOC. Inline Documentation for RESTful web APIs Consultado el 03/05/2019 en http://apidocjs.com/.

[38] Control de versiones Consultado el 04/05/2019 en https://es.wikipedia.org/wiki/Control_de_versiones.

[39] Git vs. SVN – What Is The Difference? Consultado el 04/05/2019 en https://www.perforce.com/blog/vcs/git-vs-svn-what-difference.

[40] StackOverflow. Why is Git better than Subversion? Consultado el 04/05/2019 en https://stackoverflow.com/questions/871/why-is-git-better-than-subversion.

[41] Adopt a Git branching strategy Consultado el 04/05/2019 en https://docs.microsoft.com/en-us/azure/devops/repos/git/git-branching-guidance?view=azure-devops.

[42] Web Application File Structures Consultado el 04/05/2019 en http://www.zsoltnagy.eu/web-application-file-structures/.

[43] Diseño de Interfaces. Material Design Consultado el 18/05/2019 en http://multimedia.uoc.edu/blogs/dii/es/tendencies/material-design/.

[44] Material Design Consultado el 18/05/2019 en https://en.m.wikipedia.org/wiki/Material_Design.

[45] Prueba unitaria Consultado el 07/05/2019 en https://es.wikipedia.org/wiki/Prueba_unitaria.

[46] Prueba de caja blanca Consultado el 07/05/2019 en https://es.wikipedia.org/wiki/Caja_blanca_(sistemas).

[47] Jasmine (Framework JavaScript para test) Consultado el 07/05/2019 en https://en.wikipedia.org/wiki/Jasmine_(JavaScript_testing_framework).

[48] Prueba de integración Consultado el 07/05/2019 en https://es.wikipedia.org/wiki/Pruebas_de_integraci%C3%B3n.

[49] Prueba de caja negra Consultado el 07/05/2019 en https://es.wikipedia.org/wiki/Caja_negra_(sistemas).

[50] Protactor (end to end testing for Angular) Consultado el 07/05/2019 en https://www.protractortest.org/#.

[51] Apache Cordova Consultado el 12/05/2019 en https://cordova.apache.org/.

[52] Cómo funciona Redux Consultado el 12/05/2019 en https://carlosazaustre.es/como-funciona-redux-conceptos-basicos/.

[53] Testing Angular faster with Jest Consultado el 12/05/2019 en https://www.xfive.co/blog/testing-angular-faster-jest/.

Glosario

A | B | C | E | F | H | J | L | M | R | S | T

A

API

API (Application Programming Interface) es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.. 12, 14, 43, 55–58, 68, 70, 78, 79

B

Backend

Backend es la capa de acceso a datos de un software o cualquier dispositivo, que no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos. El *Backend* también accede al servidor, que es una aplicación especializada que entiende la forma como el navegador solicita cosas.. 41, 42, 55, 56, 63, 64, 66, 68, 77, 89

C

CRUD

CRUD es el acrónimo de Crear, Obtener, Actualizar y Borrar (del original en inglés: Create, Read, Update and Delete). Se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.. 11

CSS

CSS (Cascading Style Sheets) es lo que se denomina lenguaje de hojas de estilo en cascada y se usa para estilizar elementos escritos en un lenguaje de marcado como HTML.. 61

E

Endpoints

Endpoints son las URLs de una API que responden a una petición.. 11, 58, 61, 67

F

Framework

Un *framework* (marco de trabajo) es el esquema que se establece y que se aprovecha para desarrollar y organizar un software determinado. Sirve para poder escribir código o desarrollar una aplicación de manera más sencilla.. 1, 2, 6, 8, 53, 55–57, 61, 70, 73, 78, 89

Frontend

Frontend es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios. HTML, CSS y JavaScript son los lenguajes principales del *Frontend*.. 41, 42, 55, 64, 66, 77, 89

H

Hiperenlaces

Un Hiperenlace es un elemento de la página que hace que el navegador acceda a otro recurso, otra página Web, un archivo, etc.... 2, 5

Hipertexto

El hipertexto es texto que contiene enlaces a otros textos.. 5

HTML

HTML, del inglés *HyperText Markup Language* (lenguaje de marcas de hipertexto). HTML es un lenguaje de marcado que se utiliza para el desarrollo de páginas web de Internet.. 5, 6, 55, 58, 61

HTTP

HTTP (Hypertext Transfer Protocol) es el protocolo de comunicación que permite las transferencias de información en la World Wide Web. Es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores.. 63, 66, 68

J

JavaScript

JavaScript es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas aunque existe una forma de JavaScript del lado del servidor.. 1, 2, 6–8, 11, 53, 55, 56, 73, 77, 78

JSON

JSON, acrónimo de JavaScript Object Notation, (notación de objeto de JavaScript) es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato independiente del lenguaje.. 56, 57, 63, 68, 87

L

localStorage

localStorage es una propiedad de HTML5 (web storage), que permiten almacenar datos en nuestro navegador web. Guarda información que permanecerá almacenada por tiempo indefinido; sin importar que el navegador se cierre.. 65, 66

M

Middleware

En Node.js un *middleware* es un bloque de código que se ejecuta entre la petición que hace el usuario (request) hasta que la petición llega al servidor.. 57, 61, 67

R

REST

REST (*Representational State Transfer*) es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. En la actualidad se usa para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) . 11, 14, 15, 56, 63, 68

S

Stack

Un conjunto de soluciones (también denominado *Stack* de soluciones), es un set de sistemas o componentes necesarios para realizar una solución funcional y robusta.. 1, 11, 78

T

TypeScript

TypeScript es un superconjunto de JavaScript, que esencialmente añade tipado estático y objetos basados en clases. TypeScript puede ser usado para desarrollar aplicaciones JavaScript que se ejecutarán en el lado del cliente o del servidor (Node.js). TypeScript extiende la sintaxis de JavaScript, por tanto cualquier código JavaScript existente debería funcionar sin problemas.. 1, 7, 53, 55, 56, 61

APÉNDICE A
Configuración del sistema

A.1 Instalando el proyecto

Para poder ejecutar el código desarrollado para el proyecto que se puede encontrar en la sección Código del proyecto será necesario instalar ciertos programas de antemano.

- **NodeJS**. Plataforma necesaria para poder trabajar tanto en el backend con express.js como en el *Frontend* con Angular, ya que ambos se despliegan en local sobre un servidor NodeJS https://nodejs.org/en/.
- **npm (Node Package Manager)**. Herramienta imprescindible para la instalaciones de los módulos de librerias que tanto el *Backend* como el *Frontend* necesitan para funcionar. Una vez instalado y en la carpeta raíz del proyecto, habrá que lanzar el comando "*npm install*"para instalar todas las librerías https://www.npmjs.com/.
- **Angular/CLI (Angular Command Line Interface)**. Herramienta que contiene Angular y además una serie de herramientas para facilitar la creación, depuración y despliegue de una aplicación creada en este Framework https://cli.angular.io/. Para instalarlo se hará uso de la herramienta anteriormente instalada **npm**.
- **Git**. Herramienta de control de versiones utiliza en el proyecto para llevar registro de los cambios y mantener en todo momento una versión estable del proyecto https://git-scm.com/downloads.
- **Visual Studio Code**. Editor de código fuente con control de versiones git integrado y herramientas de depuración https://code.visualstudio.com/.

A.2 Código del proyecto

Todo el código del proyecto desarrollado está disponible en la página web de almacenamientos de repositorios remotos *github*. La implementación del *frontend* se encuentra en: https://github.com/vicmarp2/mean-stack-gym y la del *Backend* en https://github.com/vicmarp2/mean-stack-gym-backend.

Ambos repositorios se encuentran ocultos debido a que contienen información de carácter privado como es la clave personal para la API de *Google Cloud Platform* utilizada para la implementación de un mapa con *Google Maps* en la aplicación. Será posible acceder a los repositorios bajo previa petición.

APÉNDICE B
Documentación API REST

Debido a la naturaleza independiente de esta API y con el objetivo del uso de esta interfaz RESTful por sistemas externos a nuestra aplicación, esta parte del software está documentada usando la librería apiDoc. De esta forma, cualquier sistema al que se le permita acceso mediante credenciales tendrá la información necesaria para poder interactuar con el sistema y realizar el control de presencia del centro deportivo.

Para generar esta documentación habrá que seguir los siguientes pasos:

- Instalar la librería apidoc de forma global haciendo uso de npm como se puede obsevar en la figura B.1.

```
npm install apidoc -g
```

Figura B.1: Instalación de apidoc usando npm

- Lanzar el comando de la figura B.2 para generar la web de documentación. El parámetro i.[es] la ruta de origen, o.[es] la ruta de destino donde generar los archivos de documentación y el parámetro t.[es] la ruta que apunta a un *template* propio en caso de que no se desee utilizar el predeterminado.

```
apidoc -i myapp/ -o apidoc/ -t mytemplate/
```

Figura B.2: Generación de documentación con apidoc

Es posible ver un ejemplo de la documentación resultante en la página web de la librería http://apidocjs.com/#examples.

Auth

Auth - Request Token

0.0.0

POST

```
api/v1/auth
```

body

Campo	Tipo	Descripción
email	String	Users email.
password	String	Users password.

Request-Example:

```
{
    "email": "email@email.com",
    "password": "pass4me"
}
```

Success 200

Campo	Tipo	Descripción
token	String	Valid access token.
expiresIn	Number	Duration of the token in seconds.
userId	String	Id of the user authenticated.
userAdmin	Boolean	"true" if the user authenticated is admin of the system.

Success-Response:

```
HTTP/1.1 200 OK
{
  token: "fsadfdSFASAfASDfsaFdsaFSA",
  expiresIn: "3600",
  userId: "32fSD453FDAERA"
  userAdmin: "true"
}
```

Error 4xx

Nombre	Descripción
InvalidAuth	The credentials were not valid.

Error-Response:

```
HTTP/1.1 401 Invalid credentials.
{
  "message": "Auth failed"
}
```

Users

Users - Request User access to event

0.0.0

GET

```
api/v1/users/access/event
```

Header

Campo	Tipo	Descripción
token	String	Authorization token.

Header-Example:

```
{
    "Authorization": "Bearer {{token}}"
}
```

Parámetro

Campo	Tipo	Descripción
userId	String	Users unique ID.
eventId	String	Event unique ID.

Success 200

Campo	Tipo	Descripción
access	Boolean	"True" if user has reservation for the event.

Figura B.3: Documentacíon de la API REST