PROYECTO DE GRADO

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito parcial para obtener el Título de INGENIERO DE SISTEMAS

CONFIGURACIÓN DINÁMICA DE INFORMACIÓN SOBRE LOS JUGADORES PARA LA ACADEMIA EMERITENSE FC EN UNA PLATAFORMA VIRTUAL DEPORTIVA

Por

Br. Eduardo Elías Gudiño Valecillos

Tutor: Dr. Gerard Páez

Noviembre 2022

Configuración dinámica de información sobre los jugadores para la Academia Emeritense FC en una plataforma virtual deportiva

Br. Eduardo Elías Gudiño Valecillos

Proyecto de Grado - Sistemas Computacionales.

Resumen: El siguiente proyecto tiene como objetivo generar los mecanismos necesarios mediante el uso de herramientas tecnológicas para el manejo de información sobre los jugadores para la Academia Emeritense FC. Este proyecto forma parte de un conjunto de funcionalidades para la plataforma de la Academia Emeritense FC, dónde se permite el manejo de la información del jugador, creación de equipos para las categorías correspondientes y búsqueda de información por atributos del jugador, permitiendo así visualizar la información de forma fácil y clara, mediante el uso de la web.

A continuación se indican las herramientas tecnológicas principales utilizadas en el desarrollo del proyecto, comenzando con la base de datos, se usó una no relacional como lo es MongoDB, esto con el fin de mantener la persistencia de los datos, en cuanto a la creación de la web se utilizó el lenguaje de programación Javascript, lenguaje de marcado de hipertexto HTML y el lenguaje de hojas de estilo CSS para crear la página web correspondiente, además del uso de la librería React de Javascript y el entorno de desarrollo que se utilizó Node.js.

Palabras clave: MongoDB, Javascript, HTML, Node.js, Configuración de información.

Índice General

Índice de Tablas

Índice de Figuras

Capítulo I

Planteamiento del problema

1.1 El problema

El manejo de la información se convirtió en una necesidad. La aparición de la tecnología permitió que la información sea utilizada en diferentes contextos de manera más amplia, facilitando además su uso y mejorando aspectos importantes como el rápido acceso, la organización, y el manejo estructurado de datos. La configuración dinámica de información repercute entre otras cosas en el acceso limpio a los datos, así como también la disminución en el tiempo de búsqueda y la facilidad de acceso a los mismos, siendo entonces una contribución positiva debido a la fácil disponibilidad, la toma de decisiones y el impacto en el desarrollo organizacional.

En el ámbito deportivo, el uso generalizado de la tecnología se fortalece, debido al aporte significativo que genera al ofrecer un fácil acceso a la información e incluso en algunos casos contribuyendo con soluciones inmediatas ante ocasiones particulares. El uso de la tecnología en este ámbito permite que el desarrollo de las actividades realizadas de manera cotidiana sea más ameno, práctico e intuitivo, aportando valor en el tiempo, así como también la organización/planificación de las actividades. En líneas generales el manejo de la información es una parte fundamental en cualquier organización. En virtud a vital importancia, las herramientas tecnológicas permiten una gestión eficiente de los datos en cuestión.

La Academia Emeritense F.C, es una academia de fútbol, ubicada en la ciudad de Mérida estado Mérida, fue fundada en el año 1993 y desde su inicio ha sido un lugar donde se forman numerosos futbolistas. En dicha academia, surge la necesidad de realizar el manejo de información sobre sus jugadores mediante el uso de la tecnología, tomando en consideración que los procesos que incluyen la información respectiva se

realiza de manera manual, almacenando además los datos mediante el uso de medios físicos, por lo que el acceso a la información se ve limitado y poco adaptable al cambio. Cabe destacar que la Academia no cuenta con los mecanismos tecnológicos necesarios para llevar a cabo las actividades de manera digital.

Lo mencionado anteriormente trae como consecuencia que el proceso de revisión, búsqueda y modificación de información, de algún jugador, tome mucho tiempo, y el procesamiento se ve amenazado a cometer diversos errores en la actualización de la ficha técnica de cada deportista, por otro lado el uso de medios físicos se deteriora con el paso del tiempo, ocasionando la pérdida de datos importantes arriesgando además la cronología de los datos, siendo la información de los involucrados un recurso valioso y necesario para el correcto funcionamiento de la organización.

De esta manera, a través de reuniones con los miembros de la Academia quienes son conocedores de los procesos que se llevan a cabo en dicho ente, quedó evidenciado que los procesos realizados, los cuales involucran la información del jugador se realizan de forma manual, causando una inversión de tiempo considerable al momento de requerir el acceso a dichos datos, por otro lado, los entrenadores no cuentan con un mecanismo para manejar equipos donde se puedan atacar las necesidades en particular de un conjunto de jugadores.

Sabiendo esto, y tomando en cuenta el conjunto de elementos mencionados anteriormente, se plantea la creación de los mecanismos necesarios para el manejo de la información sobre los jugadores de la Academia Emeritense F.C, lo cual busca facilitar la visualización y organización de la misma, permitiendo de esta manera, tener a la mano datos del jugador, como su información personal, deportiva y médica, por otra parte, se plantea el manejo de equipos con jugadores pertenecientes a una categoría en particular y la búsqueda de un jugador haciendo uso de sus atributos más

importantes, dando paso a la construcción del fácil acceso a la información, mediante el uso de la web.

1.2 Objetivos

1.2.1 Objetivo General

- Crear una aplicación web que permita el manejo de la información sobre los jugadores de la Academia Emeritense FC.

1.2.2 Objetivos específicos

- Identificar los datos relevantes sobre los jugadores de la Academia Emeritense F.C.

- Construir una interfaz web para la visualización de los datos sobre los jugadores de la Academia Emeritense FC.

- Crear una interfaz de programación de aplicaciones(API) para la interacción con los datos sobre los jugadores de la Academia Emeritense FC.

- Implementar la interacción entre la interfaz de programación de aplicaciones y la interfaz web para manejar la información sobre los jugadores de la Academia Emeritense FC.

1.3 Justificación

Este trabajo se enfoca en la configuración dinámica de información sobre los jugadores para la Academia Emeritense F.C, en una plataforma virtual deportiva, con el fin de tener un registro digital de los datos del jugador, la posibilidad de crear equipos dentro de una categoría y además, poder buscar algún jugador y tener acceso a la información correspondiente. En este sentido, la configuración dinámica de información, permite a los entrenadores y demás miembros de la academia, tener acceso rápido y detallado a los datos que muestran las características específicas del jugador. Este conjunto de datos, es útil para inferir falencias y virtudes, que puedan tener cada uno de ellos, así como también, la trayectoria deportiva e incluso un parte médico, donde se pueden observar, entre otras cosas, si presenta algún problema de salud o alguna lesión.

Para el entrenador, la creación de estos mecanismos es de suma importancia, ya que proveerá de primera mano la información sobre los jugadores, indicando sus características más relevantes, además, de permitir la configuración de equipos, dónde se pueden agregar y eliminar los distintos jugadores pertenecientes a una categoría, de la misma manera, se puede tener acceso a su información médica. En caso de requerir la información de un jugador, se puede realizar la búsqueda correspondiente. El acceso a las características, permite una visión global del jugador, esto tiene doble incidencia, ya que como se mencionó anteriormente, puede usarse para detectar las falencias (lo que permite al entrenador trabajar en mejorar ese aspecto) y por otra parte, para el provecho de las virtudes que presenten en el campo de juego, lo que es beneficioso para el equipo.

El uso de la tecnología en el ámbito deportivo, se ha hecho cada vez más fuerte con el pasar de los días, por lo que la creación de esta aplicación web que permite la configuración de información pretende ofrecer una solución al manejo de datos sobre

los jugadores, cabe destacar que la construcción de este desarrollo responde a la naturaleza de los procedimientos a seguir para la creación de una aplicación web.

Por esta razón, el alcance obtenido por esta configuración de información, es amplio, un caso particular de ello, se origina en el momento en que un jugador demuestre sus habilidades destacando del resto, lo cual pueda implicar un cambio de categoría, y el entrenador mediante la visualización de los datos, tiene los mecanismos necesarios para apreciar dichas características. En lo personal, lo positivo de esta configuración de información, es poder brindar al entrenador y demás miembros de la Academia, un perfil detallado del jugador, lo que busca aportar una mejora en el enfoque de los entrenamientos y así, aumentar la eficacia de los mismos, lo cual estaría contribuyendo al objetivo de la Academia, que es formar profesionales en el ámbito futbolístico.

1.4 Propósitos

Aportar en el desarrollo de los jugadores de la Academia Emeritense F.C, mediante la configuración de la información de los mismos, en una plataforma virtual deportiva, esto motivado a que los entrenadores tendrán a la mano datos, lo cuales le permitan dar una sesión de entrenamiento más detallada y personalizada en pro de lo individual y lo colectivo. De la misma manera, contribuye al manejo más práctico de dichos datos, ya que al encontrarse en la web, el ingreso y la visualización de los mismos, es mucho más rápido y sencillo. Todo esto creando los mecanismos tecnológicos para permitir el acceso a la información del jugador, los cuales estarían inmersos en una ficha con la información, así como también dar paso a la configuración de equipos y la búsqueda de jugadores.

Capítulo II

Marco Teórico

2.1 Antecedentes

Trabajo especial de grado en la Universidad de Los Andes(ULA) Br. Anny Vanessa Gil Calderon. (Febrero 2017), lleva por título “Diseñar, desarrollar e implementar un sistema de información web para la gestión de datos de ULA Startups”. El cual tiene como objetivo “Diseñar, desarrollar e implementar un sistema de información web para la gestión de datos de ULA Startups”. Calderon menciona que “Los sistemas de gestión son importantes para administrar sistemas web de formación debido a que dan soporte a un conjunto de utilidades para el seguimiento de los involucrados. Ofreciendo al usuario una interfaz agradable y de fácil uso, y la funcionalidad necesaria para que funcione de manera eficiente”. Dejando claro el valor de los sistemas de gestión y su importancia al momento de brindar al usuario una manera de interactuar con facilidad y eficiencia. Así mismo concluye que se desarrolló una nueva plataforma que facilita la gestión para un administrador, CEO, co fundador e inversionista bajo el esquema de aceleradora de startup.

Proyecto de Titulación en la Universidad Estatal Delsur de Manabí, Ecuador. Nora Tatiana Sornoza Ponce. (Julio 2017), lleva por título “ APLICACIÓN INFORMÁTICA PARA LLEVAR EL CONTROL DEL REGISTRO TÉCNICO Y METODOLÓGICO DE LOS DEPORTISTAS DE LA LIGA DEPORTIVA CANTONAL DEL CANTÓN JIPIJAPA.” El cual tiene como objetivo “Implementar una aplicación informática para llevar el control del registro técnico y metodológico de los deportistas de la LDC Jipijapa.” Concluyendo de esta manera que la aplicación informática fue de ayuda para que la información sea bien administrada y el tiempo de respuesta ante procesos sea optimizado.

2.2 Bases Teóricas

En este capítulo estarán los conceptos más importantes de las herramientas y procedimientos que han sido usados para llevar a cabo este trabajo.

2.2.1 Bases de datos

Camps Paré et al.(2005)“...una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única e integrada, a pesar de que debe permitir utilizaciones varias y simultáneas[1].”

Marqués. (2018) define base de datos:

Una base de datos es un conjunto de datos almacenados en memoria externa que están organizados mediante una estructura de datos. Cada base de datos ha sido diseñada para satisfacer los requisitos de información de una empresa u otro tipo de organización, como por ejemplo, una universidad o un hospital[2].

2.2.2 API

El concepto de APÏ proporcionado por Red Hat (2007):

Una API o interfaz de programación de aplicaciones es un conjunto de definiciones y protocolos que se usan para diseñar e integrar el software de las aplicaciones.

Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones; y ofrecen oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

A veces, las API se consideran como contratos, con documentación que representa un acuerdo entre las partes: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte[3].

En este caso particular de implementación la creación de las API se usará como mecanismo de comunicación para realizar consultas específicas sobre la base de datos

2.2.3 FrontEnd

Según Iván Bautista (2021)el Front-End se define como :

El frontend es la parte del desarrollo web que se dedica a la parte frontal de un sitio web, en pocas palabras del diseño de un sitio web, desde la estructura del sitio hasta los estilos como colores, fondos, tamaños hasta llegar a las animaciones y efectos.

Es esa parte de la página con la que interactúan los usuarios de la misma, es todo el código que se ejecuta en el navegador de un usuario, al que se le denomina una aplicación cliente, es decir, todo lo que el visitante ve y experimenta de forma directa[4].

2.2.4 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 en la que el navegador hace solicitudes[5].

2.2.5 MongoDB

MongoDB es una base de datos NoSQL orientada a documentos que apareció a mediados de la década de 2000. Se utiliza para almacenar volúmenes masivos de datos. A diferencia de una base de datos relacional SQL tradicional, MongoDB no se basa en tablas y columnas. Los datos se almacenan como colecciones y documentos[6].

2.2.6 Javascript

JavaScript es un lenguaje de programación o de secuencias de comandos que te permite implementar funciones complejas en páginas web, cada vez que una página web hace algo más que sentarse allí y mostrar información estática para que la veas, muestra oportunas actualizaciones de contenido, mapas interactivos, animación de Gráficos 2D/3D, desplazamiento de máquinas reproductoras de vídeo, etc., puedes apostar que probablemente JavaScript está involucrado[7].

2.2.7 Node.js

Se trata de un entorno de código abierto (open source) multiplataforma que ejecuta el código JavaScript fuera de un navegador. Este entorno de ejecución de JavaScript se orienta a eventos asíncronos (los eventos no dependen de que otros se hayan ejecutado previamente) y permite construir aplicaciones en red escalables, es decir, tiene la capacidad de realizar muchas conexiones de manera simultánea sin que tenga que leer el código línea a línea, ni abrir múltiples procesos[8].

2.2.8 Base de datos NoSQL

Según afirma Rackspace Technology (s.f):

NoSQL se refiere a una base de datos no relacional o no SQL. Una base de datos relacional es un formato de bases de datos muy estructurado basado en una tabla, como MySQL u Oracle. Las bases de datos NoSQL están orientadas a los

documentos y le permiten almacenar y recuperar datos en formatos que no sean tablas. Las aplicaciones modernas usan y generan tipos de datos complejos y que cambian constantemente, y las bases de datos relacionales no fueron diseñadas para gestionar este tipo de almacenamiento y recuperación de datos. Las bases de datos NoSQL son más flexibles y escalables.

Al trabajar con una base de datos NoSQL, usted puede agregar datos nuevos, sin tener que definirlos previamente en el esquema de la base de datos, lo que le permite procesar rápidamente grandes volúmenes de datos sin estructura, semi estructurados y estructurados.

El esquema dinámico de bases de datos NoSQL permite realizar desarrollos ágiles, que requieren iteraciones rápidas y significativas y durante los que no puede haber tiempo de inactividad[9].

2.2.9 React

React (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. React intenta ayudar a los desarrolladores a construir aplicaciones que usan datos que cambian todo el tiempo. Su objetivo es ser sencillo, declarativo y fácil de combinar. [10].

2.2.10 Métodos de petición HTTP [12]

Segun MDN contributors:

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs.

GET: El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

POST: El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

PUT: El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

DELETE: El método DELETE borra un recurso en específico.

2.2.11 Endpoint

A las URL's que reciben o retornan información de un Web API se les llama endpoints [13]

Capítulo III

Marco Metodológico

A continuación se describe la metodología que fue de utilidad para planificar y llevar un registro de las tareas a completar, lo cual permitió tener una mejor organización y manejo a la hora de desarrollar el producto.

Abellán, E. proporciona una descripción completa sobre la metodología Scrum[11]

3.1 Metodología Scrum

La metodología Scrum es un marco de trabajo o framework que se utiliza dentro de equipos que manejan proyectos complejos. Es decir, se trata de una metodología de trabajo ágil que tiene como finalidad la entrega de valor en períodos cortos de tiempo y para ello se basa en tres pilares: la transparencia, inspección y adaptación.

3.2 Bases de la Metodología Scrum

Al estar enmarcada dentro de las metodologías ágiles, Scrum se basa en aspectos como:

- La flexibilidad en la adopción de cambios y nuevos requisitos durante un proyecto complejo.
- El factor humano.
- La colaboración e interacción con el cliente.
- El desarrollo iterativo como forma de asegurar buenos resultados.

Los pilares o características de la metodología Scrum más importantes son 3:

3.2.1 Transparencia

Con el método Scrum todos los implicados tienen conocimiento de qué ocurre en el proyecto y cómo ocurre. Esto hace que haya un entendimiento “común” del proyecto, una visión global.

3.2.2 Inspección

Los miembros del equipo Scrum frecuentemente inspeccionan el progreso para detectar posibles problemas. La inspección no es un examen diario, sino una forma de saber que el trabajo fluye y que el equipo funciona de manera auto-organizada.

3.2.3 Adaptación

Cuando hay algo que cambiar, el equipo se ajusta para conseguir el objetivo del sprint. Esta es la clave para conseguir el éxito en proyectos complejos, donde los requisitos son cambiantes o poco definidos y en donde la adaptación, la innovación, la complejidad y flexibilidad son fundamentales.

3.3 Roles en el equipo Scrum

Con la metodología Scrum, el equipo tiene como foco entregar valor y ofrecer resultados de calidad que permitan cumplir los objetivos de negocio del cliente.

Para ello, los equipos de Scrum son auto-organizados y multifuncionales. Es decir, cada uno es responsable de unas tareas determinadas y de terminarlas en los tiempos acordados. Esto garantiza la entrega de valor del equipo completo, sin necesidad de ayuda o la supervisión minuciosa de otros miembros de la organización.

En Scrum existen 3 roles muy importantes : Product Owner, Scrum Master y Equipo de desarrollo.

3.3.1 Product Owner

Es el responsable de maximizar el valor del trabajo del equipo de desarrollo. La maximización del valor del trabajo viene de la mano de una buena gestión del Product Backlog, el cual explicaremos más adelante.

El Product Owner es el único perfil que habla constantemente con el cliente, lo que le obliga a tener muchos conocimientos sobre el negocio.

Para finalizar, un equipo Scrum debe tener solo un Product Owner y este puede ser parte del equipo de desarrollo.

3.3.2 Scrum Master

Es el responsable de que las técnicas Scrum sean comprendidas y aplicadas en la organización. Es el manager de Scrum, un líder que se encarga de eliminar impedimentos o inconvenientes que tenga el equipo dentro de un sprint, aplicando las mejores técnicas para fortalecer el equipo.

Dentro de la organización, el Scrum Master tiene la labor de ayudar en la adopción de esta metodología en todos los equipos.

3.3.3 Equipo de desarrollo

Son los encargados de realizar las tareas priorizadas por el Product Owner. Es un equipo multifuncional y auto-organizado. Son los únicos que estiman las tareas del product backlog, sin dejarse influenciar por nadie.

Los equipos de desarrollo no tienen sub-equipos o especialistas. La finalidad de esto es transmitir la responsabilidad compartida si no se llegan a realizar todas las tareas de un sprint.

3.4 Hitos de la Metodología de trabajo Scrum

El desarrollo iterativo se realiza en un sprint, que contiene los siguientes eventos:sprint planning, daily meeting, sprint review y sprint retrospective.

3.4.1 Sprint

El sprint es el corazón de Scrum, es el contenedor de los demás hitos del proceso. Todo lo que ocurre en una iteración para entregar valor está dentro de un sprint. La duración máxima es de un mes, el tiempo se determina en base al nivel de comunicación que el cliente quiere tener con el equipo. Los sprints largos pueden hacer que se pierda feedback valioso del cliente y poner en peligro el proyecto.

3.4.2 Sprint planning

En esta reunión todo el equipo Scrum define qué tareas se van a abordar y cuál será el objetivo del sprint. La primera reunión que se hace en el sprint puede llegar a tener una duración de 8 horas para sprints de un mes.

El equipo se hace las siguientes preguntas:

- ¿Qué se va a hacer en el sprint? En base a ello, se eligen tareas del product backlog.
- ¿Cómo lo vamos a hacer? El equipo de desarrollo define las tareas necesarias para completar cada ítem elegido del Product Backlog.

La definición de qué se va a hacer implica que el equipo tenga un objetivo y se encuentre comprometido con la entrega de valor que se hará al cliente al final del sprint. A esto se le llama sprint goal.

El resultado de esta reunión es el sprint goal y un sprint backlog.

3.4.3 Daily Meeting

Es una reunión diaria dentro del sprint que tiene como máximo 15 minutos de duración. En ella deben participar, sí o sí, el equipo de desarrollo y el Scrum Master. El Product Owner no tiene necesidad de estar presente.

En esta reunión diaria el equipo de desarrollo hace las siguientes tres preguntas:

- ¿Qué hice ayer?
- ¿Qué voy a hacer hoy?
- ¿Tengo algún impedimento que necesito que me solucionen?

Esta reunión es la más oportuna para poder inspeccionar el trabajo y poder adaptarse en caso de que haya cambio de tareas dentro de un sprint.

3.4.4 Sprint review

La revisión del valor que vamos a entregar al cliente se hace en esta reunión, al final de cada sprint. Su duración es de 4 horas para sprints de un mes, y es la única reunión de Scrum a la que puede asistir el cliente. En ella el Product Owner presenta lo desarrollado al cliente y el equipo de desarrollo muestra su funcionamiento. El cliente valida los cambios realizados y además brinda feedback sobre nuevas tareas que el Product Owner tendrá que agregar al Product backlog.

3.4.5 Sprint retrospective

La retrospectiva es el último evento de Scrum, tiene una duración de 3 horas para Sprints de un mes, y es la reunión del equipo en la que se hace una evaluación de cómo se ha implementado la metodología Scrum en el último sprint.

Es una gran oportunidad para el equipo Scrum de inspeccionarse a sí mismo, proponiendo mejoras para el siguiente sprint.

El resultado: una lista de mejoras que debe aplicar el siguiente día, ya que al finalizar la retrospectiva, inmediatamente comienza un nuevo sprint, que incluye el sprint planning, daily meeting, sprint review y la ya mencionada sprint retrospective.

3.5 Herramientas para la metodología Scrum

Las herramientas que se utilizan en Scrum están definidas para maximizar la transparencia dentro del equipo; es decir, que todos tengan una misma visión de lo que está ocurriendo en el proyecto.

Las herramientas principales de Scrum son: product backlog y sprint backlog.

3.5.1 Product backlog

Básicamente, el product backlog es el listado de tareas que engloba todo un proyecto. Cualquier cosa que debamos hacer debe estar en el product backlog y con un tiempo estimado por el equipo de desarrollo.

La responsabilidad exclusiva de ordenar el product backlog es del Product Owner, que se encuentra en constante comunicación con el cliente para asegurarse de que las prioridades están bien establecidas.

La ordenación también es 100% responsabilidad del Product Owner, por lo que las tareas que están más arriba deben de ser las de mayor prioridad.

El equipo de desarrollo elige tareas del product backlog en el sprint planning para generar tanto el sprint backlog como el sprint goal.

3.5.2 Sprint backlog

Es el grupo de tareas del product backlog que el equipo de desarrollo elige en el sprint planning junto con el plan para poder desarrollarlas. Debe ser conocido por todo el equipo, para asegurarse de que el foco debe estar en este grupo de tareas.

El sprint planning no cambia durante el sprint, solo se permite cambiar el plan para poder desarrollarlas.

3.6 Aplicación de la Metodología

Se realizó uso de la metodología Scrum con algunas variantes, esto tomando en cuenta la disponibilidad de las partes involucradas, siendo utilizado para constatar un cronograma de tareas a realizar donde la entrega de valor se mantuvo, el tiempo fue variable, se efectuaron reuniones periódicas con las partes interesadas para recopilación de información y también entrega de avances progresivamente.

Listado de tareas que se llevaron a cabo:

1. Creación/Validación de bocetos de las vistas.
2. Creación/Validación de los datos incluidos en los esquemas de base de datos.
3. Creación de ficha para visualizar la información del jugador.
4. Creación de vistas para manejo de categoría y equipos.
5. Creación de vista para búsqueda de jugador por atributo.
6. Creación de API para obtener información del jugador de la base de datos.
7. Creación de API para manejar información de categoría y equipos.
8. Uso de APIS junto con las vistas.

Sprint 1

Tareas realizadas:

- Creación/Validación de bocetos de las vistas

Como se llevaron a cabo:

- Se realizó un estudio de los datos disponibles, y en base a eso se clasificaron.
- Una vez definidos los datos, se crearon los bocetos de las vistas junto con el ingeniero de software del proyecto

Entrega de valor:

- Bocetos de las vistas a implementar, dando así un punto base para comenzar el desarrollo

Sprint 2

Tareas realizadas:

- Creación y validación de los datos incluidos en los esquemas de base de datos.

Cómo se llevaron a cabo:

- El ingeniero de software del proyecto, propuso un conjunto de datos en esquemas que conforman la base de datos.
- Se realizaron revisiones y acotaciones, sobre dichos datos.
- Se llegó a un acuerdo final de los datos con las acotaciones.

Entrega de valor:

- Esquemas de base de datos (mongoDB) definidos.

Sprint 3

Tareas realizadas:

- Creación de ficha para la información del jugador.

Cómo se llevaron a cabo:

- Tomando en cuenta el boceto creado, se procedió a realizar la ficha del jugador.
- Primero se tomó en cuenta la información general del jugador.
- Como segundo paso se tomó en cuenta la información deportiva.
- Como punto final se creó la vista con información médica.
- Se crearon estructuras de datos para simular la información que se toma de la base de datos.
- Se realizaron pruebas manuales con cada una de las funcionalidades descritas

Entrega de valor:

- Vista de jugador(ficha) funcional para la parte del frontEnd.

Sprint 4

Tareas realizadas:

- Creación de vistas para manejo de categoría y equipos.

Cómo se llevaron a cabo:

- En primera instancia se creó la vista principal, dónde se muestran todas las categorías.
- Se creó la vista con información de detalle para cada una de las categorías.
- Se creó la vista para crear un equipo.
- Se creó la vista para manejar el equipo.
- Se crearon estructuras de datos para simular la información que se toma de la base de datos.
- Se realizaron pruebas de uso manuales progresivas para cada una de las funciones de la vista.

Entrega de valor:

- Vistas que conforman el acceso a categorías y manejo de equipos funcional para la parte del frontEnd.

Sprint 5

Tareas realizadas:

- Creación de vista para búsqueda de jugador por atributo.

Cómo se llevaron a cabo:

- Tomando en cuenta los datos más relevantes y necesarios, para ser añadidos como parámetros a usar para realización de la búsqueda.
- Estudiando los bocetos creados, y en base a ellos se creó la vista correspondiente, tomando en cuenta los casos donde hay o no, resultados para la búsqueda de jugadores con los criterios dados.

- Creando estructuras de datos para simular la información que se toma de la base de datos.
- Realizando pruebas manuales para la búsqueda por atributo.

Entrega de valor:

- Desarrollo del frontEnd para la búsqueda del jugador.

Sprint 6

Tareas realizadas:

- Creación de API para obtener información del jugador de la base de datos.

Cómo se llevaron a cabo:

- Tomando en cuenta el esquema de mongo creado, y a partir de allí se crearon las consultas correspondientes.
- Se realizó testing a la API para constatar su correcto funcionamiento.

Entrega de valor:

- API para hacer consultas a la información del jugador.

Sprint 7

Tareas realizadas:

- Creación de API para manejar información de categoría y equipos.

Cómo se llevaron a cabo:

- Tomando en cuenta el esquema de mongo creado, y a partir de allí se creó el CRUD correspondiente para el manejo de equipos.
- Se realizó testing a la API para constatar su correcto funcionamiento.

Entrega de valor:

- API para el manejo de equipos.

Sprint 8

Tareas realizadas:

- Uso de APIS junto con las vistas.

Cómo se llevaron a cabo:

- Se implementaron las funcionalidades correspondientes, consumiendo las APIS creadas.
- Se realizaron pruebas manuales de funcionamiento.

Entrega de valor:

- BackEnd y FrontEnd funcionan en conjunto.

Capítulo IV

Diseño e Implementación

Este capítulo muestra los aspectos tomados en cuenta para llevar a cabo el proyecto, desde el estudio de los requerimientos, creación de la estructura gráfica (modelos, bocetos, wireframes) hasta la implementación, usando la lógica y herramientas tecnológicas necesarias para su desarrollo.

Tener claro los requerimientos es esencial para la realización del producto, eventualmente permite que el desarrollo sea más manejable, gracias al conocimiento sobre la naturaleza y uso, el cual se le dará en este caso a los datos, se puede dar paso a la creación de modelos los cuales serán usados posteriormente en la fase de implementación, una vez definido dichos datos, se crean los bocetos que definen la estructura, es decir, cada una de las interfaces del producto y modelos de datos, los cuales se toman como base para comenzar a desarrollar.

4.1 Requerimientos

4.1.1 Ficha del jugador

Es requerida una página donde se muestre la información del jugador, la cual se puede dividir en tres tipos:

4.1.1.1 Información de ámbito personal

En este apartado estará presente la información básica del jugador, como lo es su nombre, apellido, edad, dirección, nacionalidad, entre otros, de la misma manera, se considera la información referente a sus padres, en la cual se destacan elementos como su profesión, lugar de nacimiento e información de contacto. Ver tabla **4.1**.

Información general	Padres
Nombres	Nombre
Apellidos	Apellido
Edad	Cédula
Dirección	Profesión
Nacionalidad	Lugar de nacimiento
Sexo	Email
Fecha de nacimiento	Teléfono de habitación
Teléfono celular	Teléfono celular
Teléfono de habitación	
Lugar de nacimiento	
Email	
Cédula	
Institución educativa	
Año/grado	
Número de pasaporte	
Partida de nacimiento	
Fecha de inscripción	
Grupo sanguíneo	
Alergias/Operaciones	

***Tabla4.1*.** Datos básicos del jugador

4.1.1.2 Información de ámbito deportivo

La información de interés en el ámbito deportivo, contiene datos que indican el estado de forma del jugador, así como también características y referencias sobre su trayectoria deportiva, lo que permiten conocer un poco más sobre el mismo. En esta sección es importante los datos de estatura, peso, posición, pierna hábil, características generales y su antecedentes deportivos. Ver tabla **4.2**

Información personal	**Trayectoria deportiva**
Estatura	Título
Peso	Hito
Posición	
Pierna dominante	
Características	

Tabla4.2. Datos deportivos del jugador

4.1.1.3 Información de ámbito médico

Contiene la información indicativa sobre el estado de salud del jugador, así como también los antecedentes médicos de interés que ha suscitado, los cuales son de gran ayuda al momento de presentarse alguna molestia o problema de salud importante, también da la posibilidad de llevar un seguimiento, en caso de que se haya un diagnóstico, se deba seguir plan de tratamiento, y además, indique información general importante sobre el estado de salud del jugador. Ver tabla 4.3.

Antecedentes	Exámenes y diagnósticos
Patológico metabólico	Examen clínico
Patológico gastrointestinal	Paraclínico
Patológico neurológico	Diagnóstico
Patológico cardíaco	Plan de tratamiento
Patológico genitourinario	Observación final
Infeccioso vph	
Infecciosos vih	
Infeccioso hepatitis A	
Infeccioso hepatitis B	
Quirúrgico	
Tóxico alérgico	
Medicamentos	
Hematológicos	

***Tabla 4.3*.** Datos medicos del jugador

4.1.2 Formación de equipos

Los jugadores que hacen parte de la academia están ordenados en categorías, cada una aloja los inscritos en ella, cuando el número de personas en cada categoría se hace muy grande, es necesario un mecanismo que permita manejar equipos dentro de la categoría, darle un nombre y la posibilidad de añadir o eliminar un jugador para cada equipo, lo que hace que el entrenamiento se pueda realizar de manera focalizada con equipos formados con n jugadores.

4.1.3 Búsqueda por atributo

Es necesario un espacio que permita la búsqueda de un jugador inscrito en la Academia, para los cuales se usarán diferentes criterios, y se realizará un filtrado con los mismos, mostrando los jugadores que lo cumplan,además de esto, tener la posibilidad de ver su ficha (descrito en 4.1.1).

4.2 Wireframes

Previo al proceso de programación de la parte visual (Front-End), se crea un prototipo, el cual permite tener una idea previa de cómo se verá el sitio web. La importancia de la creación de estos bocetos o esquemas, es la facilidad con la cual se pueden realizar iteraciones hasta llegar a la versión deseada, así como también en caso de que surja la necesidad de añadir una nueva funcionalidad, pueda ser más sencillo incluirla en la estructura general de la página, sabiendo esto, uno de los pasos que se tomó en cuenta, fue la creación de dicha estructura, cuyo proceso se llevó a cabo en conjunto con el encargado de ingeniería de software del proyecto. En líneas generales esta estructura se crea para tener un vistazo y validación a priori de las partes involucradas.

Tomando en cuenta lo mencionado anteriormente, a continuación se muestran los wireframes creados para el sitio web.

Figura 4,1.Vista de categorías

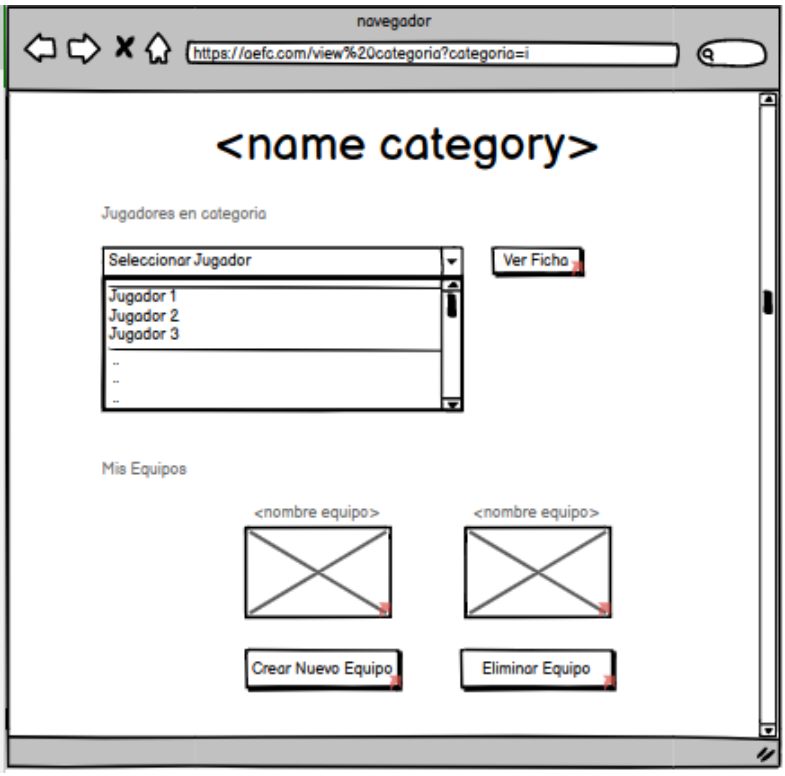


Figura 4.2. Vista de detalle para categoría

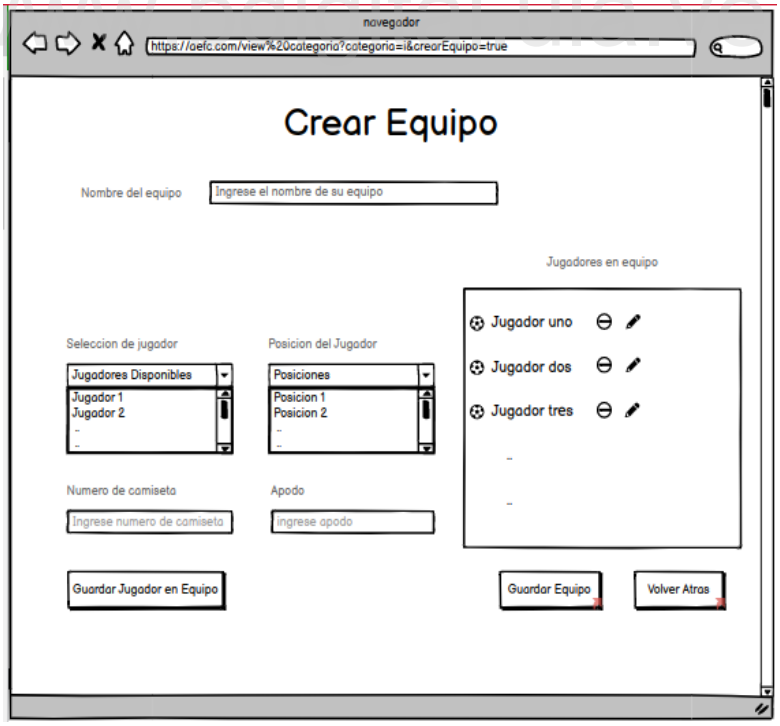


Figura 4.3. **Vista para creación de equipo**

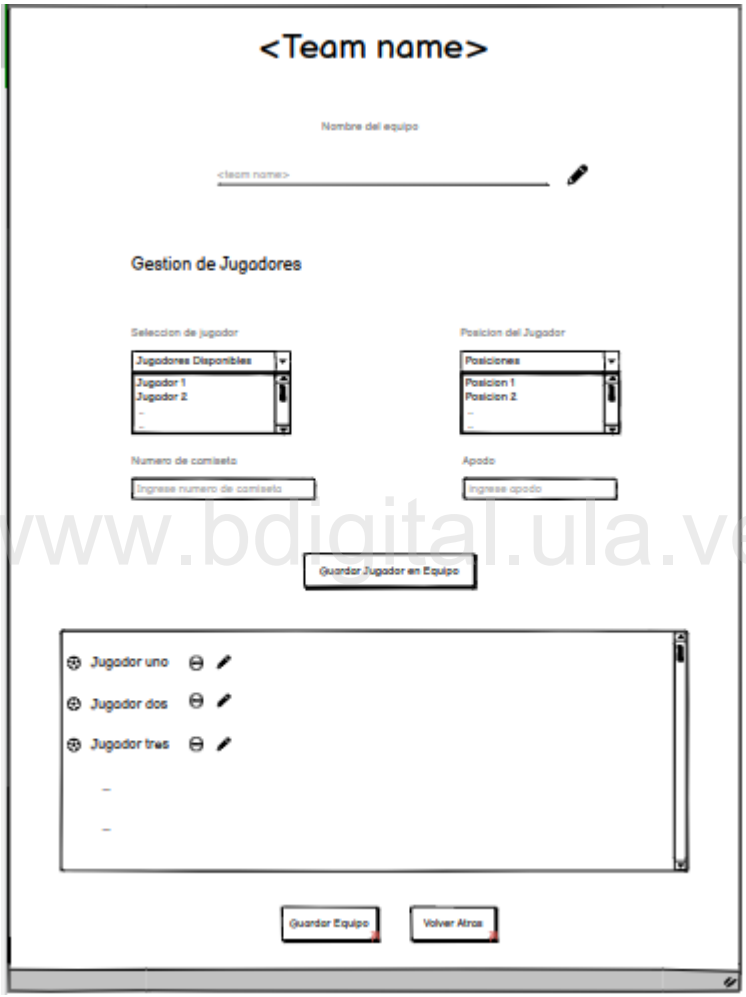


Figura 4.4. Vista para gestión de equipo

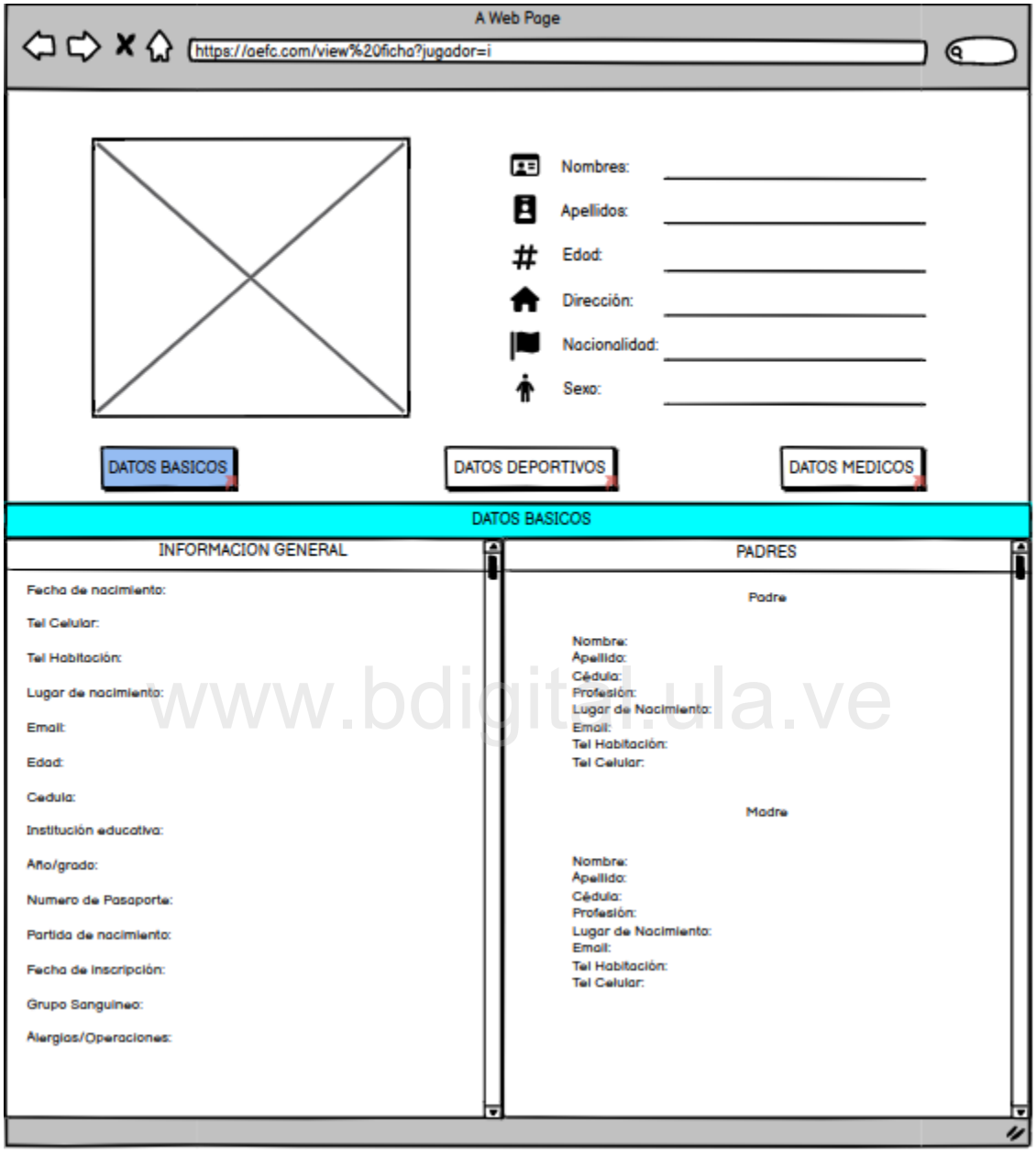


Figura 4.5. Vista de ficha del jugador para datos básicos

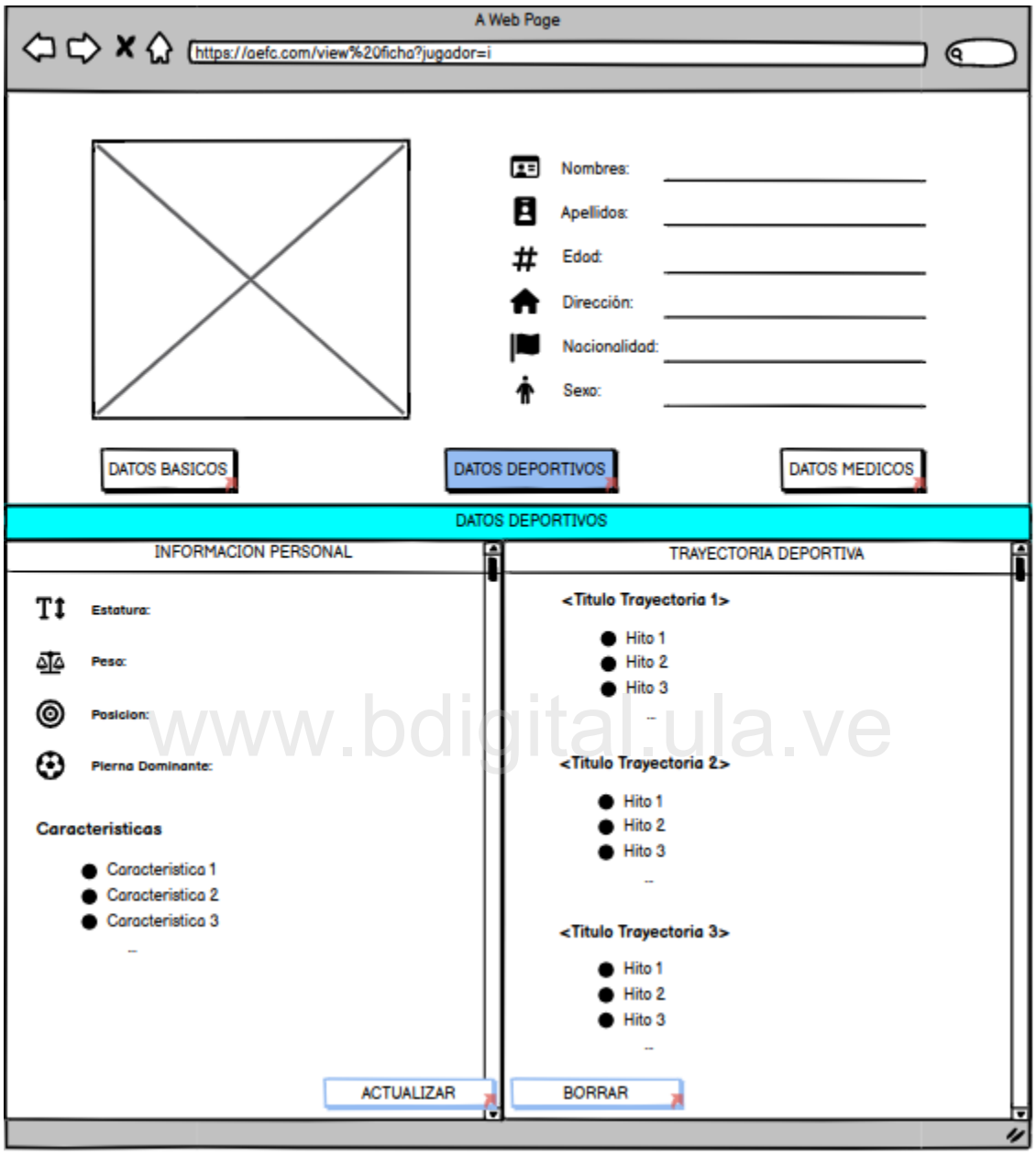


Figura 4.6. Vista de ficha del jugador para datos deportivos

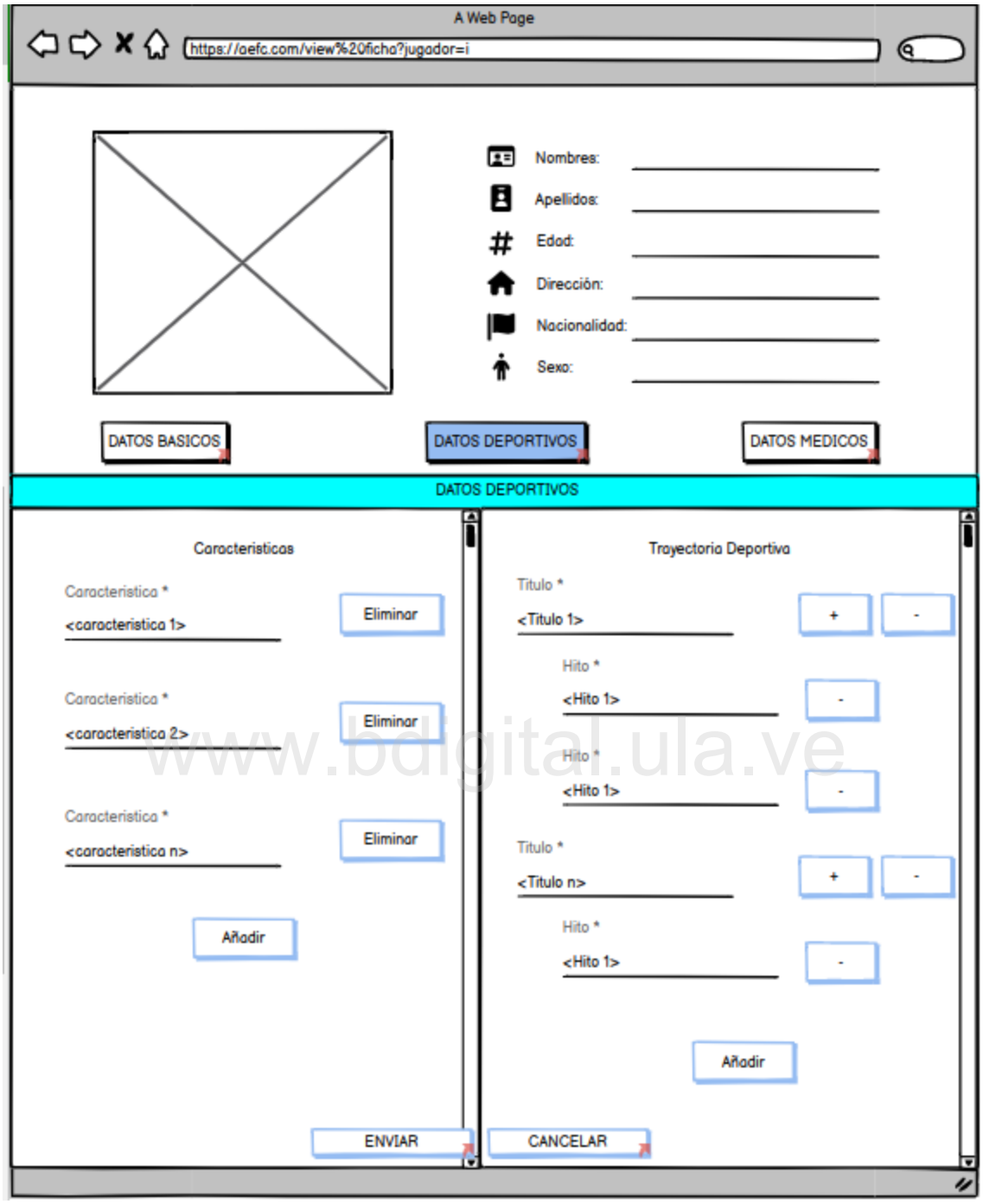


Figura 4.7. Vista de ficha del jugador para actualización de datos deportivos

Figura 4.8. Vista de ficha del jugador cuando no existen datos deportivos

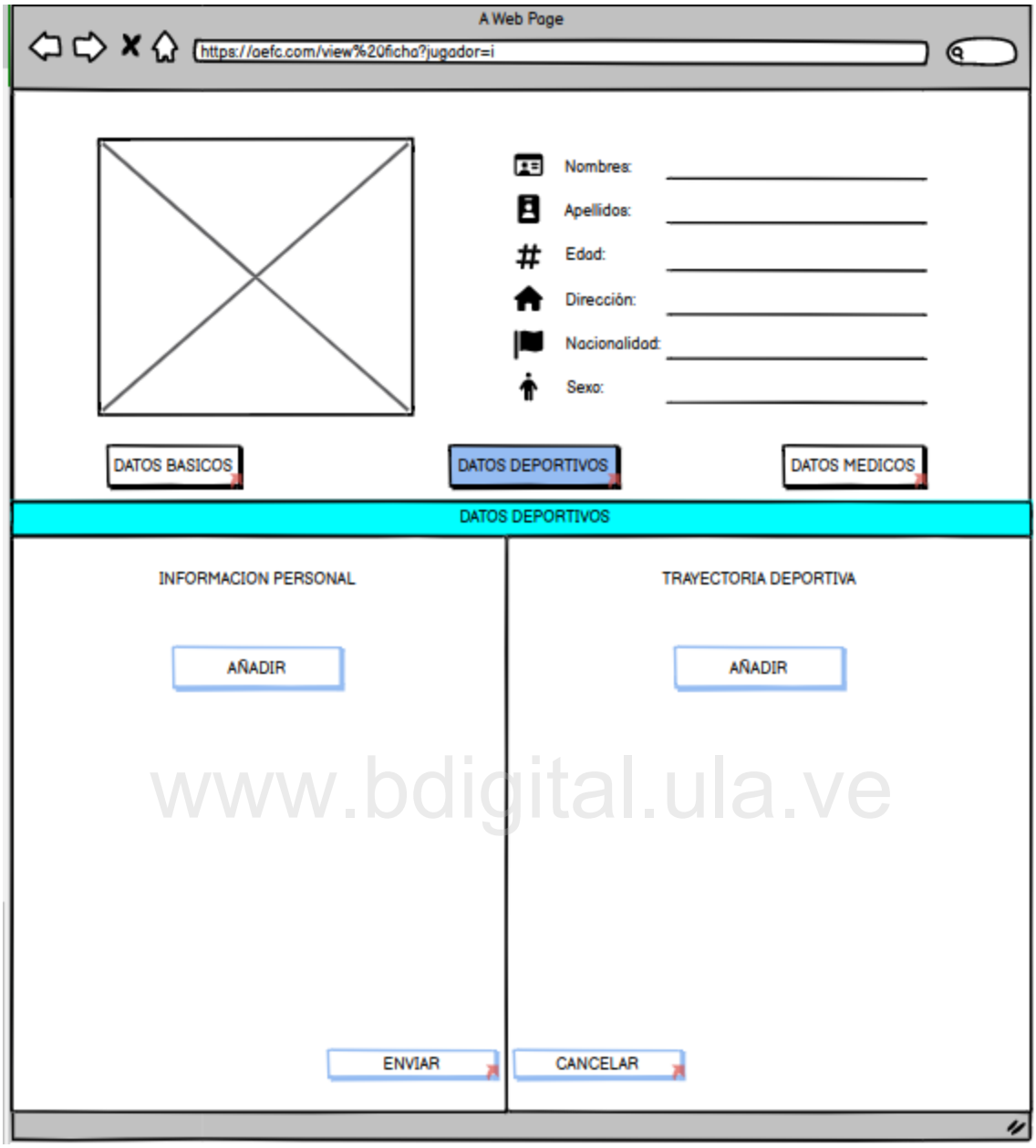


Figura 4.9. Vista de ficha del jugador para añadir datos deportivos

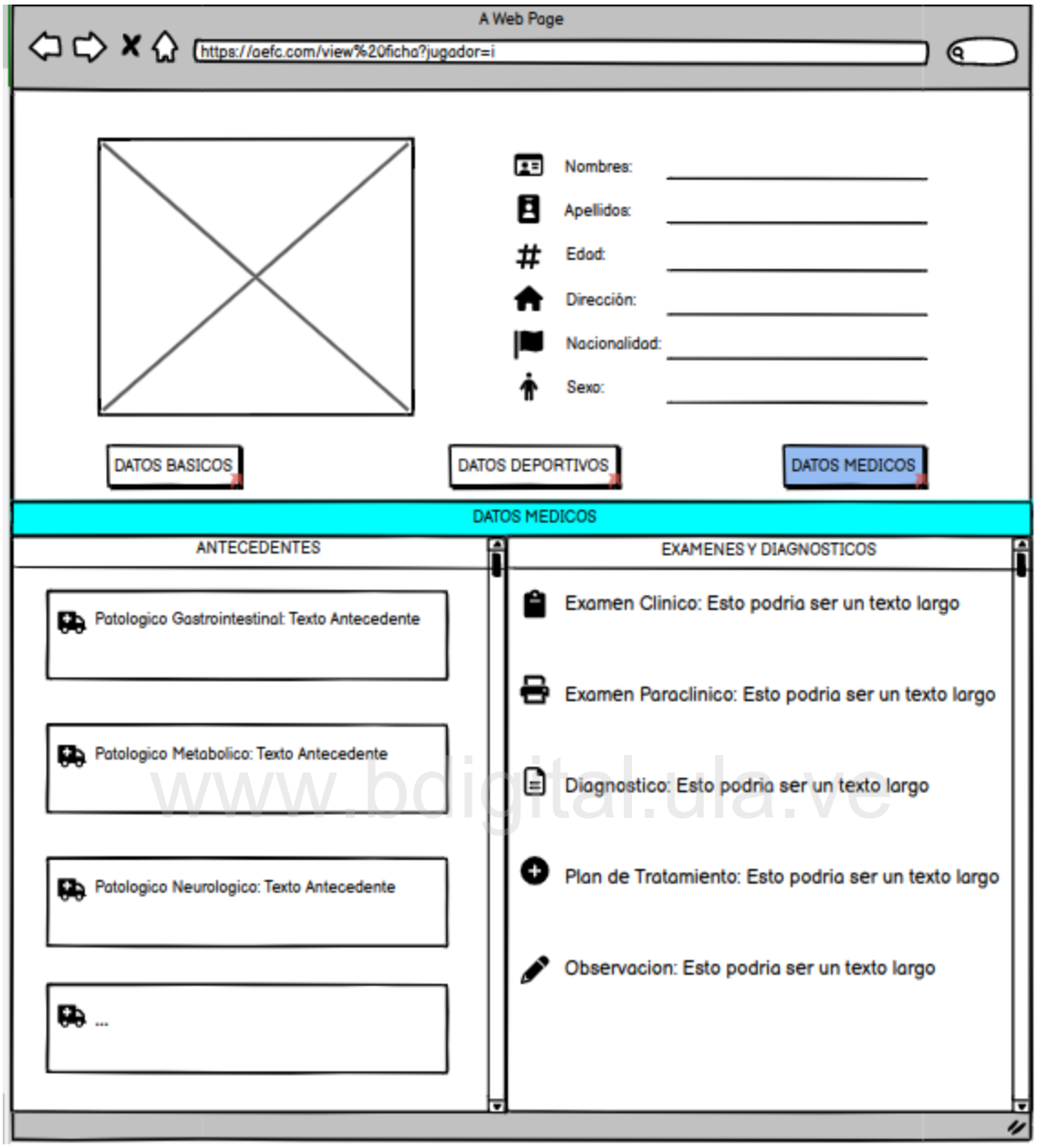


Figura 4.10.Vista de ficha del jugador con datos médicos.

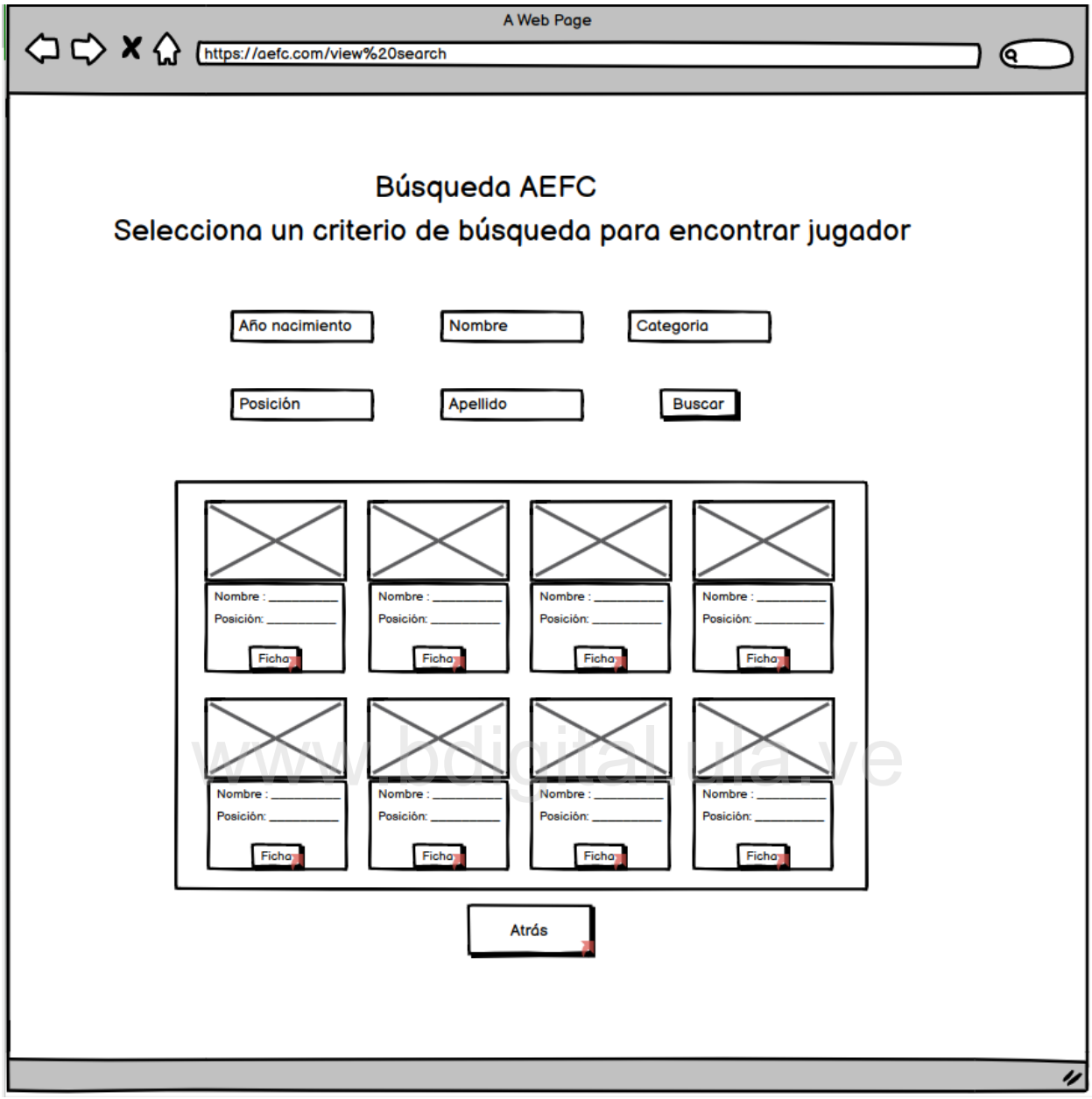


Figura 4.11. Vista de búsqueda de jugador por atributo

4.3 Esquemas

Una vez se hayan definido los datos de relevancia a usar en el desarrollo, y en vista de que se hace uso de una base de datos no relacional, como lo es mongoDB, uno de los pasos llevados a cabo para asegurar la persistencia de los datos, ha sido definir los esquemas. Estos esquemas ayudan a precisarla estructura de la información contenida en la base de datos, permitiendo así, realizar el uso adecuado de los mismos, en algunos casos estos esquemas contienen condiciones y reglas, que deben cumplir los datos para poder crearlos correctamente.

La definición de los esquemas, es el paso a realizar luego de conocer los datos a utilizar, los cuales se mantendrán en el tiempo.Esta definición es considerada parte importante del desarrollo, dichos esquemas fueron realizados por el encargado de ingeniería de software del proyecto, luego de intercambiar ideas y hacer las modificaciones correspondientes, se llegó a una versión final de los mismos.

A continuación, se muestran los esquemas utilizados.

Nombre	Tipo	Ejemplos
ID	ObjectID	62ec530146256aaa1d5605as1
primer_nombre	cadena(String)	Anthony
segundo_nombre	cadena(String)	Eduardo
primer_apellido	cadena(String)	Briceño
segundo_apellido	cadena(String)	Moreno
sexo	cadena(String)	masculino o femenino
edad	numero(Number)	4 , 23, 16
email	cadena(String)	the_chee_21@hotmail.com
phone	cadena(String)	+584147945243
phone_habitacion	numero(Number)	"02712216754"
pasaporte	cadena(String)	1234
id	numero(Number)	25934765

fecha_nacimiento	fecha(Date)	2010-05-30
lugar_nacimiento	cadena(String)	valera estado trujillo
partida_nacimiento	numero(Number)	501
nacionalidad	cadena(String)	venezolano
dir_hab	cadena(String)	Merida estado Merida, residencias mayeya apto 20 piso 2.
institucion	cadena(String)	Escuela las gonzales
año_grado	cadena(String)	7mo año
foto		
padres	Objeto(List ObjectID)	["62ec530046256aaa1d5605a7","62ec530146256aaa1d5605a9"]
historia_clinica	Objeto(ObjectID)	62ec530146256aaa1d5605ab9
categoria	Objeto(ObjectID)	62ec530146256aaa1d5605ac9
plantilla_jugador	Objeto(List Dictionary)	[{ plantilla: ObjectId numUniforme: 10 posiciones: [ObjectID,...], apodo: String }]
datos_deportivos	Objeto(ObjectID)	62ec530146256aaa1d5605ac9
grupo_sanguineo	cadena(String)	A+,O-
alergias_operaciones	cadena(String)	Alergico a la canela
fecha_inscripcion	Date	2022-03-12
fecha_actualizacion	Date	2022-03-12

Tabla 4.4. Esquema de datos para el jugador

Nombre	Tipo	Ejemplos
ID	ObjectID	62ec530146256aaa1d5605as1
primer_nombre	cadena(String)	Anthony
primer_apellido	cadena(String)	Briceño
primer_apellido	cadena(String)	Briceño
segundo_apellido	cadena(String)	Moreno
sexo	cadena(String)	masculino o femenino
email	cadena(String)	the_chee_21@hotmail.com
phone	cadena(String)	+584147945243
nacionalidad	cadena(String)	venezolano
sexo	cadena(String)	masculino o femenino
id	numero(Number)	25934765
profesion	cadena(String)	ingeniero civil
empresa	cadena(String)	ULA
fecha_inscripcion	Date	2022-03-12
fecha_actualizacion	Date	2022-03-12

Tabla 4.5. Esquema de datos de los padres del jugador

Nombre	Tipo	Ejemplos
ID	ObjectID	62ec530146256aaa1d5605as1

patologico_metabolico	Objeto(boleano,string)	{
		checked: true,
		description: 'retraso metabolico'
		}

patologico_gastrointestinal	Objeto(boleano,string)	{
		checked: true,
		description: 'retraso gastrointestinal'
		}

patologico_neurologico	Objeto(boleano,string)	{
		checked: true,
		description: 'retraso neurologico'
		}

patologico_cardiaco	Objeto(boleano,string)	{
		checked: true,
		description: 'retraso cardiaco'
		}

patologico_genitourinario	Objeto(boleano,string)	{
		checked: true,
		description: 'retraso genitourinario'
		}

infeccioso_vph	Objeto(boleano,string)	{
		checked: false,
		description: ''
		}

infeccioso_vih	Objeto(boleano,string)	{
		checked: false,
		description: ''
		}

infeccioso_hepatitisA	Objeto(boleano,string)	{
		checked: false,
		description: ''
		}

infeccioso_hepatitisB	Objeto(boleano,string)	{
		checked: false,
		description: ''
		}

antecedentes_totales	Numero(Number)	10,0
examen_clinico	cadena(String)	ejemplo examen clínico etc...
paraclinico	cadena(String)	ejemplo examen paraclínico etc...
diagnostico	cadena(String)	ejemplo diagnóstico etc...

plan_tratamiento	cadena(String)	ejemplo plan de tratamiento.
observacion_final	cadena(String)	ejemplo observación final etc...

quirurgico	Objeto(boleano,string)	{
		checked: false,
		description: "
		}

toxico_alergico	Objeto(boleano,string)	{
		checked: false,
		description: "
		}

medicamentos	Objeto(boleano,string)	{
		checked: false,
		description: "
		}

hematologicos	Objeto(boleano,string)	{
		checked: false,
		description: "
		}

fecha_inscripcion	Date	2022-03-12
fecha_actualizacion	Date	2022-03-12

***Tabla 4.6*.** Esquema de datos médicos del jugador

Nombre	Tipo de dato	Ejemplos
ID	ObjectID	62ec530146256aaa1d5605as1
peso_kg	Number(Double)	40-70,5
estatura_cm	Number(Double)	100.50 - 100.20
pierna dominante	cadena(String)	pierna derecha, pierna izquierda
caracteristicas	Array(List String)	['buen pie derecho','visión de campo']
antecedentes deportivos	Array(Object)	[{ titulo: 'campeonato pan', hitos: ['buenas apariciones','quedo de segundo con su equipo'] }]
fecha_registro	Date	2022-03-12
fecha_actualizacion	Date	2022-03-12

Tabla 4.7. Esquema de datos deportivos del jugador

Nombre	Tipo de dato	Ejemplos
ID	ObjectID	62ec530146256aaa1d5605az4
nombre	cadena(String)	sub 5, sub7…sub n
categoria_edad	numero(Number)	5, 10, 20
años	cadena(String)	2017 - 2018
minimo_jugadores	numero(Number)	6, 10
fecha_registro	Date	2022-03-12
fecha_actualizacion	Date	2022-03-12

***Tabla 4.8*.** Esquema de categorias

Nombre	Tipo de dato	Ejemplos
ID	ObjectID	62ec530146256aaa1d5605as1
nombre_plantilla	cadena(String)	Equipo Academia Equipo Guerrero
jugadores	Objeto(List ObjectID)	["62ec530046256aaa1d5605a7","62ec530146256aaa1d5605a9"]
fecha_registro	Date	2022-03-12
fecha_actualizacion	Date	2022-03-12

***Tabla 4.9*. Esquema de equipos**

4.4 Implementación FrontEnd

Una vez han sido definidos los bocetos, que son el modelo a seguir a lo largo del desarrollo, se procede a la creación de las vistas, usando las herramientas tecnológicas correspondientes. En este caso se usó HTML, CSS, Javascript, React y MUI, todo esto ejecutado sobre NodeJS.

HTML, usado para definir la estructura del sitio web y su contenido.

CSS, para crear estilos de los elementos de la página, en líneas generales todo lo necesario para añadir estilismo (colores, bordes, entre otros) y un mejor acabado a los elementos.

Javascript, es el lenguaje de programación usado para manejar la lógica, los datos y operaciones que se realizan sobre el sitio web.

React, la librería de Javascript que permite construir interfaces de usuarios con mayor facilidad.

Material UI(MUI), la librería que provee un conjunto de componentes reusables de React para crear interfaces.

NodeJS, es un entorno que permite ejecutar Javascript desde el computador.

4.4.1 Análisis

Tomando en consideración lo mencionado anteriormente, se tiene como base las herramientas tecnológicas que se han usado para llevar a cabo la creación del FrontEnd del proyecto, proceso el cual, comenzó creando la estructura básica de los elementos a mostrar en el sitio web, todo esto mediante el uso de React, el cual permite la construcción de componentes, los cuales, en este caso, son la unidad básica que al unirse dan forma al sitio web. El uso de los componentes, permite que cada unidad se convierta en un mecanismo para dar solución y encapsular un requerimiento en particular, por ejemplo, si se necesita mostrar la información general del jugador, es útil crear un componente que reciba estos datos y en base a ellos puedan renderizarse en el sitio web, esto es de gran ayuda, ya que si se requiere mostrar dichos datos con la misma estructura en otra interfaz del sitio web basta con usar nuevamente el componente ya creado.

Los componentes se comunican entre sí, haciendo usos de las propiedades, las cuales pueden ser funcionalidades(funciones) o datos necesarios para añadir un comportamiento específico usando la misma estructura básica de dicho componente, en otras palabras actúan como funciones recibiendo distintos parámetros para luego proporcionar la salida correspondiente(un componente de react, generalmente).

Una característica importante de los componentes es su estado, el cual actúa como memoria permitiendo guardar la información de la instancia del componente que se está visualizando en ese momento.

En líneas generales,en el proyecto se crearon los componentes necesarios para cada una de las vistas, recibiendo las propiedades correspondientes para su correcta

comunicación,lo que permitió conformar las partes dando forma a la funcionalidad/visualización de la página.

4.4.1.1 Ficha del jugador

Para la ficha del jugador se crearon distintos componentes los cuales se definen a continuación detalladamente.

Profile: En este componente se encuentra toda la lógica principal del perfil del jugador, en él, se elige cuál de las opciones disponibles se puede visualizar ya sea información deportiva, médica o básica, por lo que se comunica con otros 3 componentes que permiten justamente mostrar la información correspondiente para cada una de las opciones mencionadas.

SportInfo: Se manejan las diferentes funcionalidades para el ámbito deportivo, se encarga de elegir qué elemento renderizar, asi como tambien el manejo de cada una de las opciones que se puede llegar a presentar como; actualización de información, el ingreso de información,información disponible, no existe información, por lo que este se comunica con 4 componentes, que permiten manejar cada caso específicamente.

FormSport: Maneja datos y estados necesarios para el envío de un formulario para añadir información, se comunica con los componentes **FormSportLeft** y **FormSportRight** , en el cual se toman los datos ingresados por el usuario.

FormSportLeft: Se toman los datos correspondientes para agregar características del jugador.

FormSportRight: Se toman los datos correspondientes para agregar la historia deportiva del jugador.

SportDataInfo: Se realiza el manejo de la información deportiva del jugador en caso de que exista.

SportDataNoInfo: Alerta al usuario que no existe información, proporciona la posibilidad de ir a **FormSport** para agregar información.

SportUpdate: En caso de que exista información, se proporciona la posibilidad de actualizarla, por lo que este componente se comunica con los componentes **FormSportLeft** y **FormSportRight,** al ser una actualización estos componentes se cargan con los datos que están guardados y permite su carga con los nuevos datos. Este es un ejemplo de reutilización de componentes en este caso **FormSportLeft** y **FormSportRight.**

MedicalInfo: Muestra la información médica del jugador.

En la figura **4.12** se muestra el diagrama con la estructura de comunicación entre componentes.

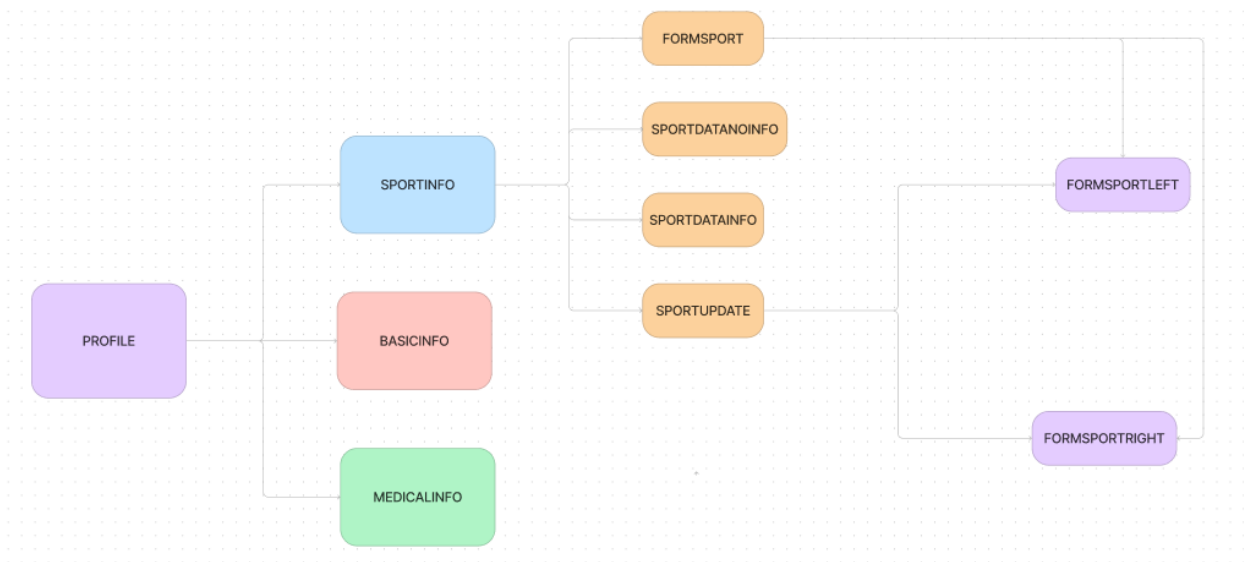


Figura 4.12. Comunicación entre componentes para la ficha del jugador.

4.4.1.2 Manejo de equipo

A continuación se describen cada uno de los componentes utilizados en la creación del manejo de equipos.

Categories: Componente principal dónde se cargan todas las categorías disponibles.

CategoriesDetails: Al seleccionar una categoría se utiliza este componente con la información correspondiente a la seleccionada. Se muestran los equipos creados en la categoría y se da la posibilidad de manejarlos, crearlos o eliminarlos. Se comunica con los componentes **CreateTeam, HandleTeam** y **Profile.**

CreateTeam: Aquí está la funcionalidad correspondiente para crear un nuevo equipo, los jugadores disponibles en la categoría seleccionada se pasan como parámetro a este componente. Utiliza el componente **ListPlayerElement**, dónde se muestran los jugadores que se van agregando al equipo.

HandleTeam: Se realiza el manejo del equipo que ha sido creado, en este apartado también se puede acceder a la ficha del jugador gracias a la comunicación con el componente **Profile**, también se comunica con **ListPlayerElementHandler**, el cual muestra y maneja las operaciones realizadas sobre los jugadores adscritos a la categoría.

Diagrama que muestra la comunicación entre componentes.

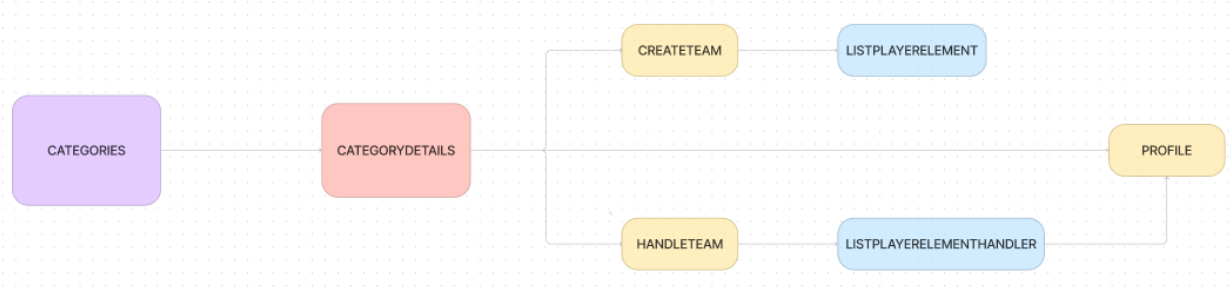


Figura 4.13.Comunicación entre componentes para el manejo de equipos.

4.4.1.3 Búsqueda de jugador

En el caso de la búsqueda del jugador se utilizan 3 componentes los cuales serán descritos a continuación. Para llevar a cabo la búsqueda se cargan los datos de los jugadores desde la base de datos a memoria (en una array), luego se realiza un filtro de elementos con cada uno de los atributos correspondientes.

AttributteSearch: Componente principal, dónde se muestran las opciones a elegir para el filtrado de los jugadores, aquí se manejan los estados para mostrar los jugadores encontrados así como también el caso en que no exista un jugador con esos atributos, este se comunica con **ShowPlayersFound.**

ShowPlayersFound: Se encarga de mostrar los jugadores que han sido encontrados en caso de que los haya, recibe por parámetro dichos jugadores y muestra una pequeña carta del jugador con la información más relevante, se comunica con **PlayerInfoSearch,** para mostrar dicha información de los jugadores

PlayerInfoSearch: En este se carga la información del jugador que es usada por **ShowPlayersFound**, lo que permite que al momento de realizar algún cambio en la manera de mostrar los resultados de jugadores sea fácil de modificar

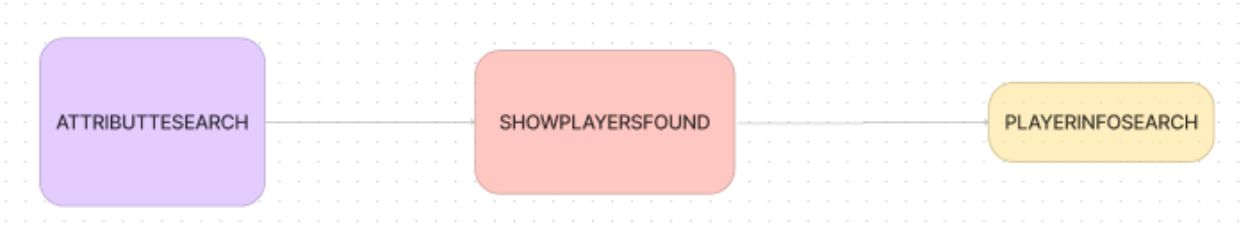


Figura 4.14. Comunicación entre componentes para la búsqueda de jugador.

4.4.2 Vistas

Una vez descrita la lógica usada en los componentes y la interacción que realizan los mismos, en este apartado se muestran las vistas que se han generado, haciendo uso de dichos componentes.

Figura 4.15. Vista de categorías

Figura 4.16. Vista detalle de categoría. Categoría sub 14

Figura 4.17. Vista crear equipo. Categoría sub 14

Figura 4.18.Vista de gestión de equipo.

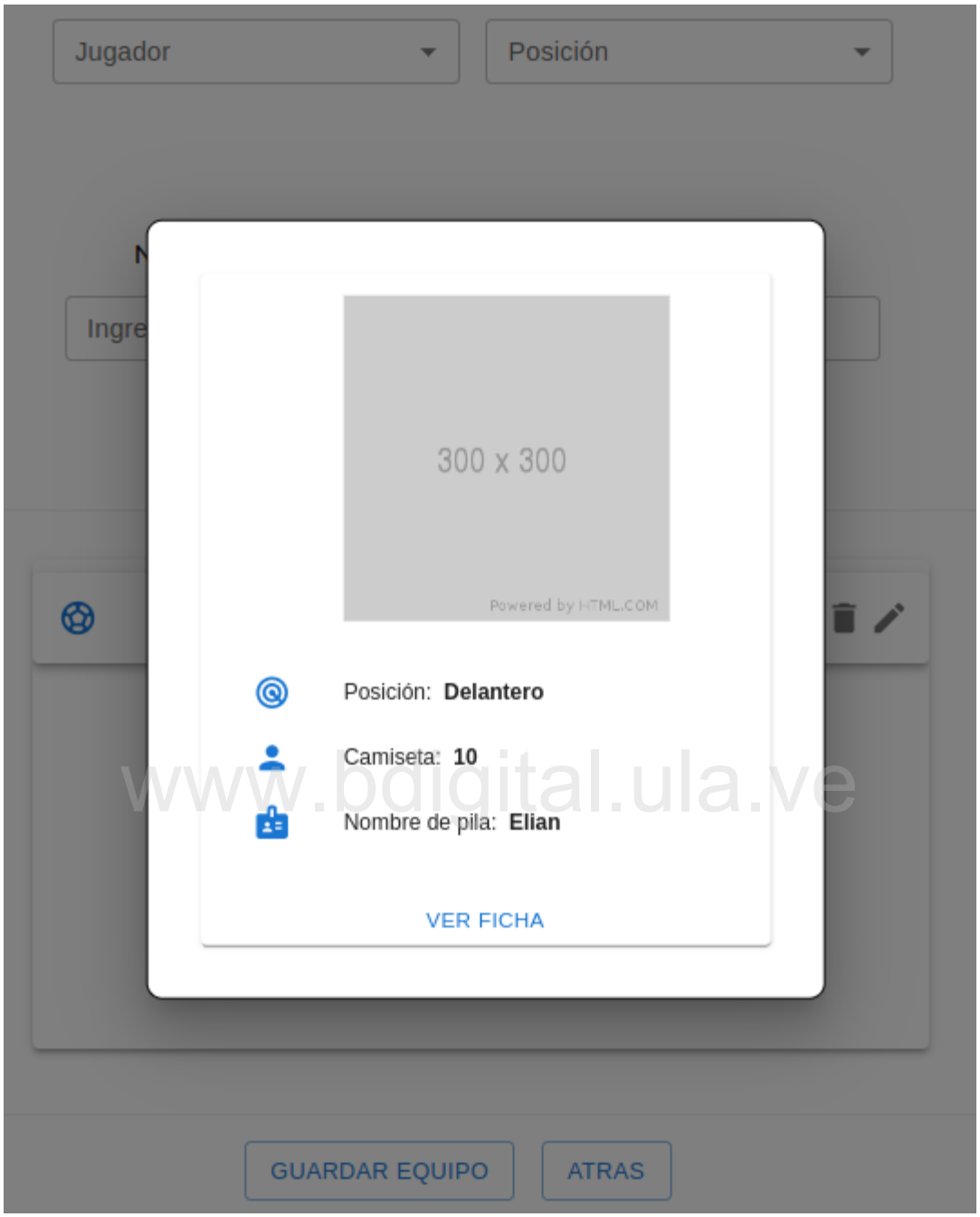


Figura 4.19. Vista ficha de equipo

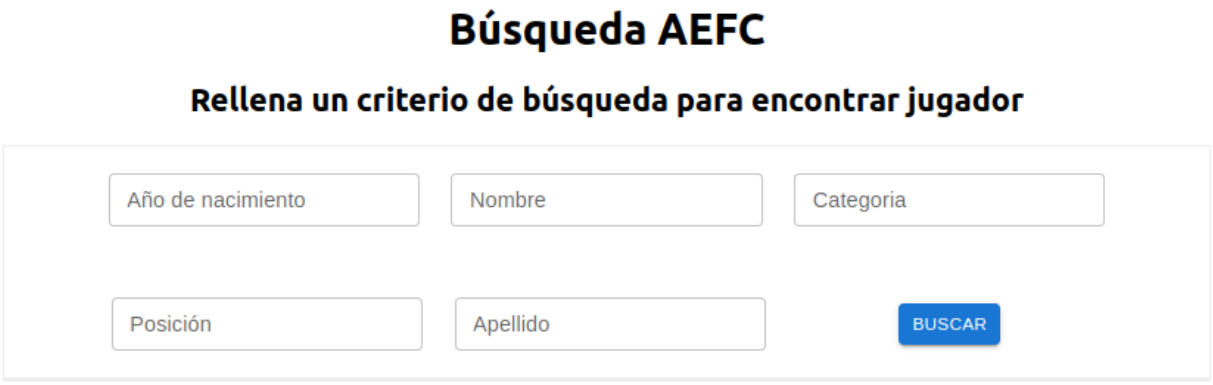


Figura 4.20. Vista búsqueda de jugador.

***Figura 4.21*. Vista búsqueda de jugador. No se encontraron resultados**

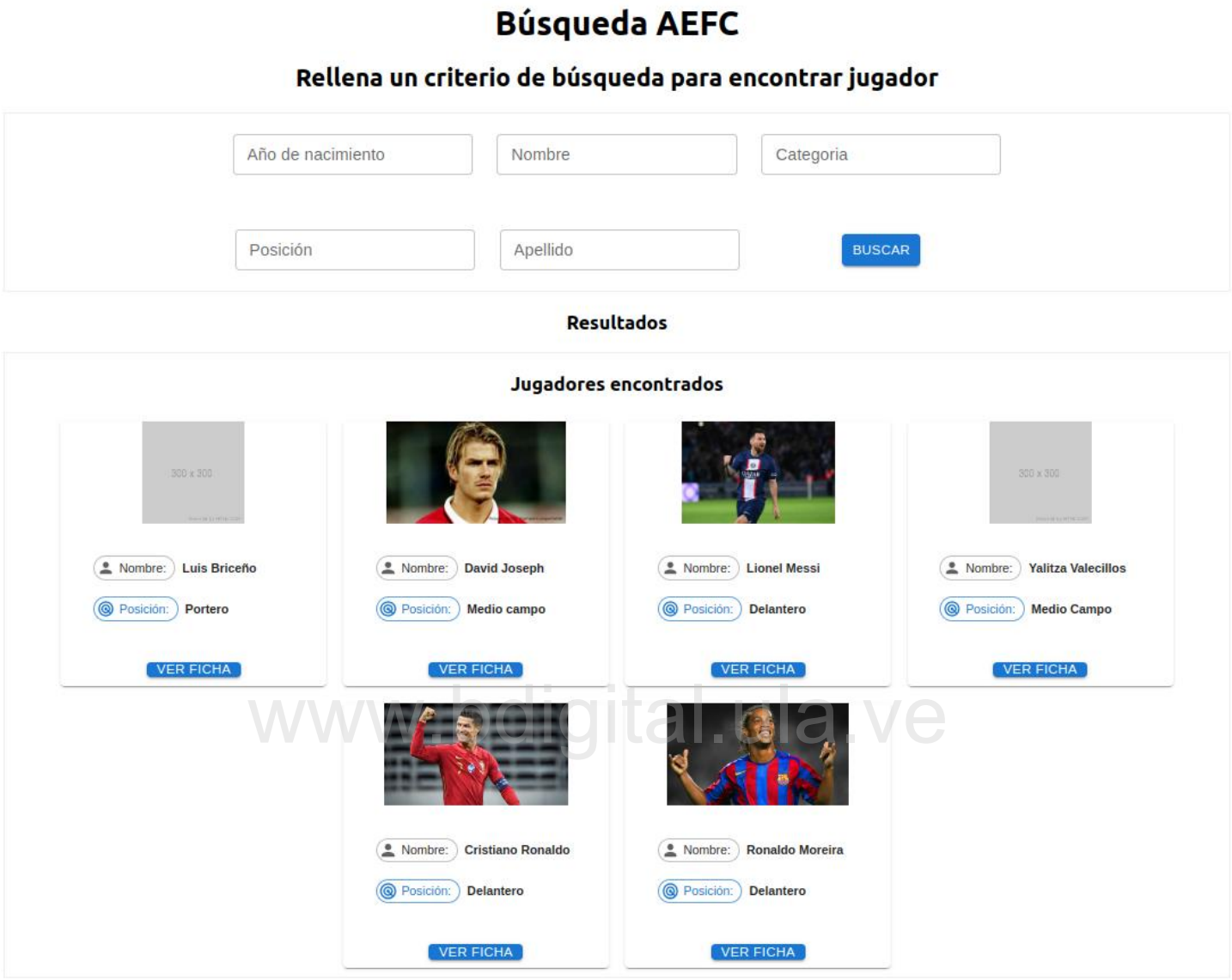


Figura 4.22. Vista búsqueda de Jugador. Resultados encontrados

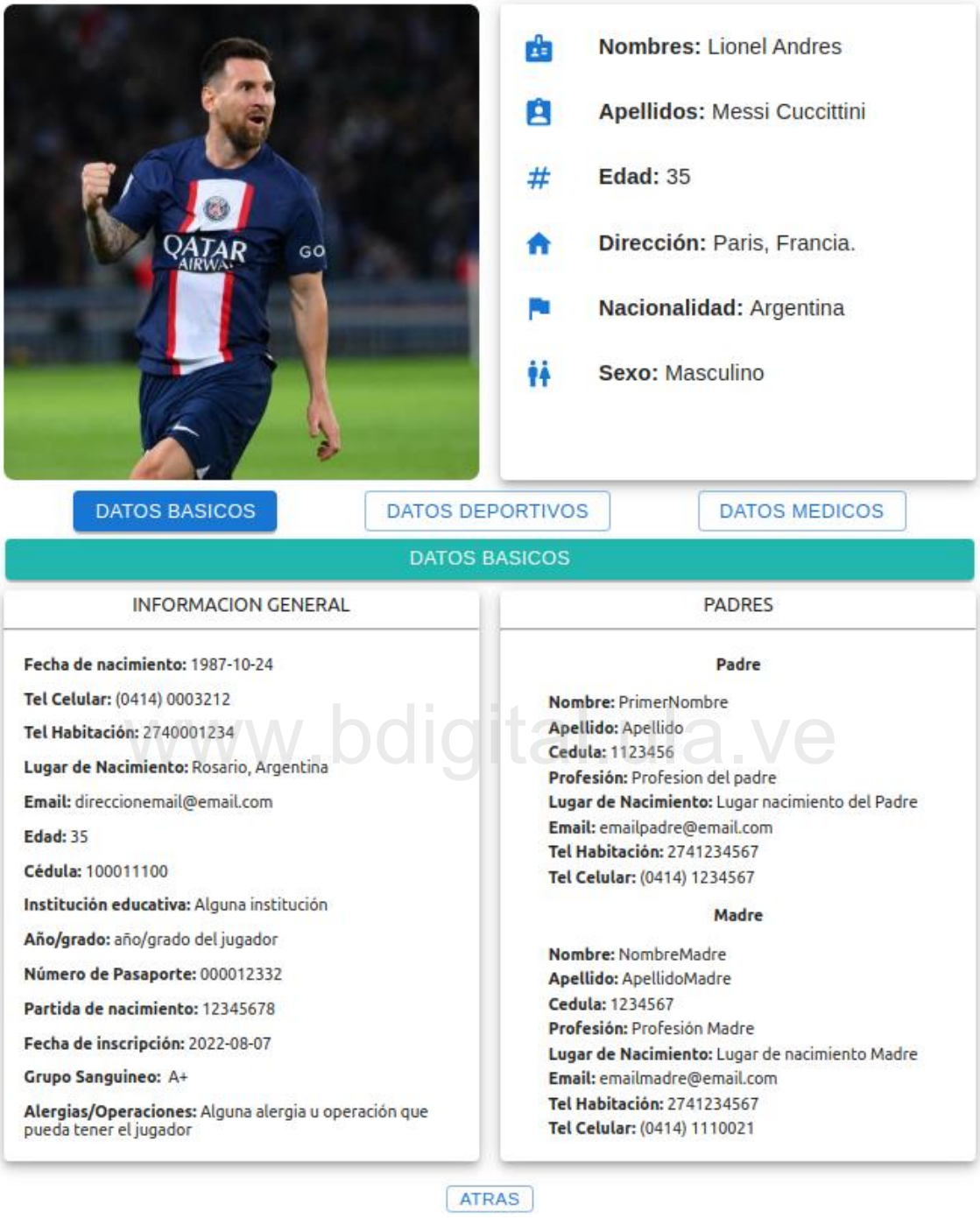


Figura 4.23.Vista jugador datos básicos

Figura 4.24.Vista jugador datos deportivos

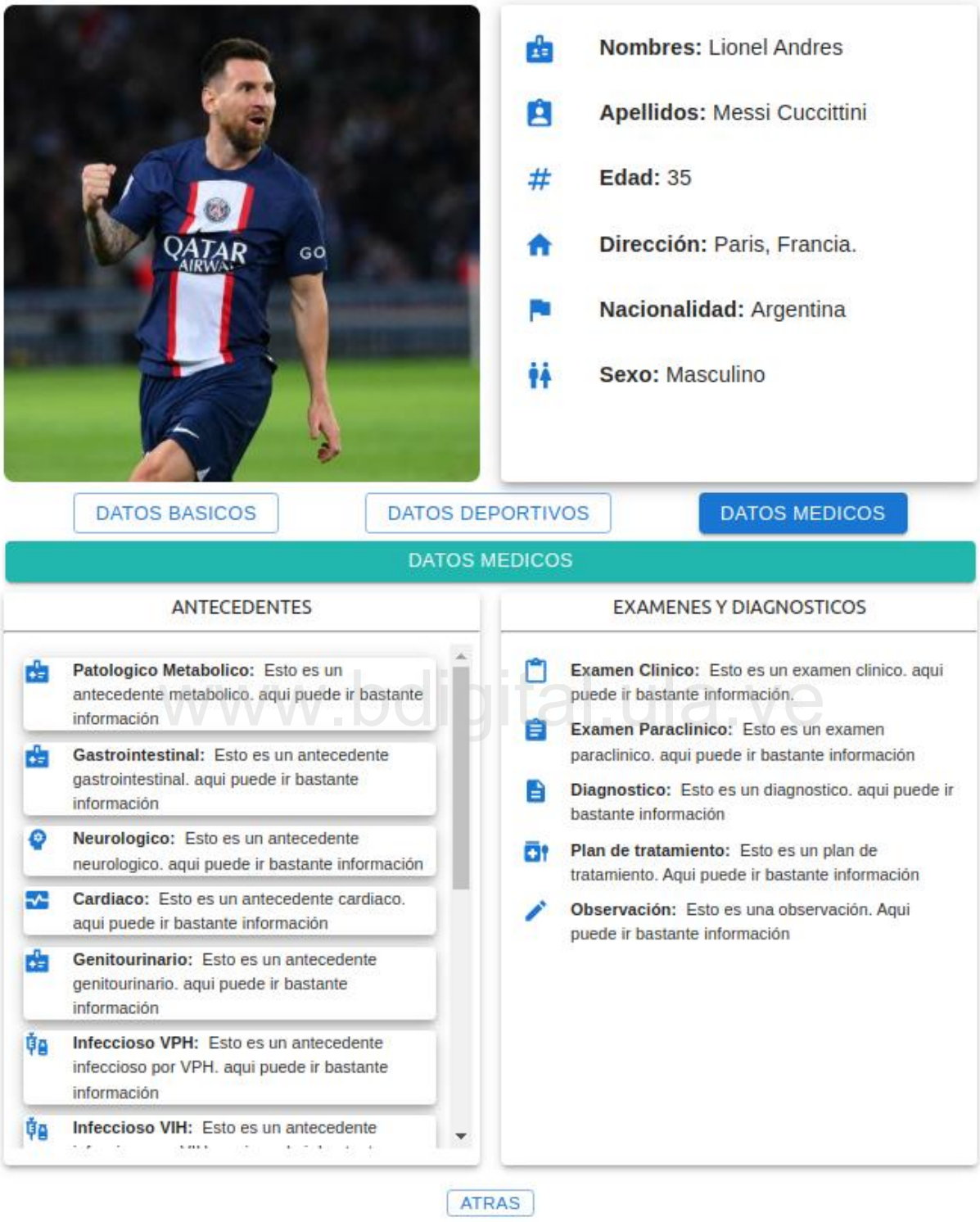


Figura 4.25. Vista jugador datos médicos.

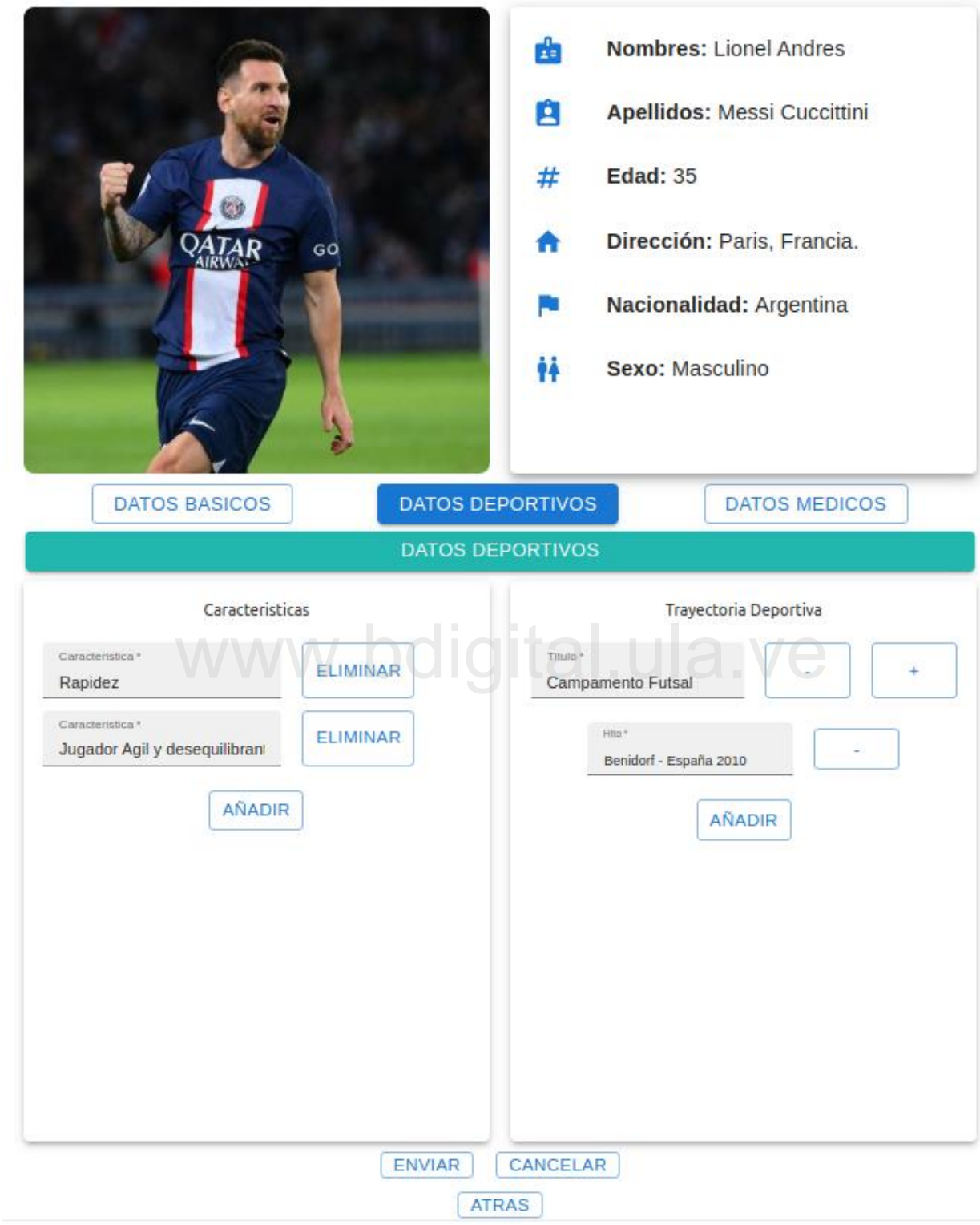


Figura 4.26.Vista datos deportivos. Actualización.

4.5 Implementación Backend

En la sección 4.3 se mencionaron los esquemas, estos contienen la estructura de los datos definida para el proyecto, en este apartado se muestra el uso dado a dichos esquemas. Vale la pena recalcar, que se está haciendo uso de una base de datos no relacional(NoSql), como lo es mongoDB, sabiendo esto, existe una librería que permite realizar la conexión entre la base de datos y el entorno de ejecución NodeJS, esta librería se llama **mongoose,** gracias al uso de la misma, se pueden definir los modelo de los datos y crear una comunicación haciendo uso de los mismos, lo que permite estructurarla y manejarla con mayor facilidad.

En la definición de lo que son estos objetos, los cuales representan el modelo de los datos, se agregan validaciones para cada campo, así como también el tipo para cada uno de ellos, lo cual permite un correcto manejo de los mismos, evitando el ingreso de datos erróneos. De la misma manera, los campos dentro de estos modelos pueden contener referencias a otros esquemas, esta es la manera en la cual se relacionan los datos entre distintos esquemas.

En definitiva, mongoose permite crear el modelo que define la estructura en la cual se van a guardar los datos en la base de datos MongoDB, una vez que estos son guardados se convierten en documentos los cuales están dentro de una colección que en este caso sería el esquema definido.

A continuación se muestran algunos de los esquemas que han sido creados para el manejo de los datos en la base de datos MongoDB, haciendo uso de mongoose, esto con el objetivo de ejemplificar un poco mejor lo mencionado anteriormente, dichos esquemas fueron proporcionados por el encargado de Ingeniería de software del proyecto.

```
const mongoose = require('mongoose');

const PlantillaSchema = new mongoose.Schema({
    nombre_plantilla: {
        type: String,
        trim: true,
        default: "Plantilla"
    },
    jugadores: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'jugador'
    }],
    categoria: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'categoria'
    }
},
    {
        timestamps: {
            createdAt: 'fecha_registro', // Use `created_at` to store the created date
            updatedAt: 'fecha_actualizacion' // and `updated_at` to store the last updated date
        }
    })

module.exports = mongoose.model('plantilla', PlantillaSchema)
```

Figura 4.27. Esquema en mongoose para el manejo de equipo.


```
const mongoose = require('mongoose');

const DatoDeportivoSchema = new mongoose.Schema({
    caracteristicas: [{type: String}],

    antecedentes_deportivos: [{
        titulo: String,
        hitos: [ {type:String} ]
    }]
},
{
    timestamps: {
        createdAt: 'fecha_registro', // Use `created_at` to store the created date
        updatedAt: 'fecha_actualizacion' // and `updated_at` to store the last updated date
    }
})
module.exports = mongoose.model('DatoDeportivo',DatoDeportivoSchema)
```

Figura 4.28. Esquema en mongoose para el manejo de datos deportivos del jugador.

4.5.1 API

Una vez definidos los esquemas, se tiene la base necesaria para comenzar a realizar la interacción con la base de datos, debido a que se realizan múltiples operaciones sobre la misma, surge la necesidad de crear una API, lo cual permite entre otras cosas estandarizar el uso de esta interacción, dando paso a la creación de un medio mediante el cual pueda integrarse en otras aplicaciones.

Para la creación de esta API, se realizó el uso del framework express, el cual permite crear aplicaciones web. En este punto con el uso de mongoose y express se tienen las herramientas necesarias para construir esta interfaz de programación de aplicación. El siguiente paso que se llevó a cabo fue crear las rutas correspondientes (endpoints), las cuales son el medio por el cual se realiza la interacción con la base de datos, en este caso. A continuación se muestra el proceso de comunicación que se lleva a cabo en este proceso

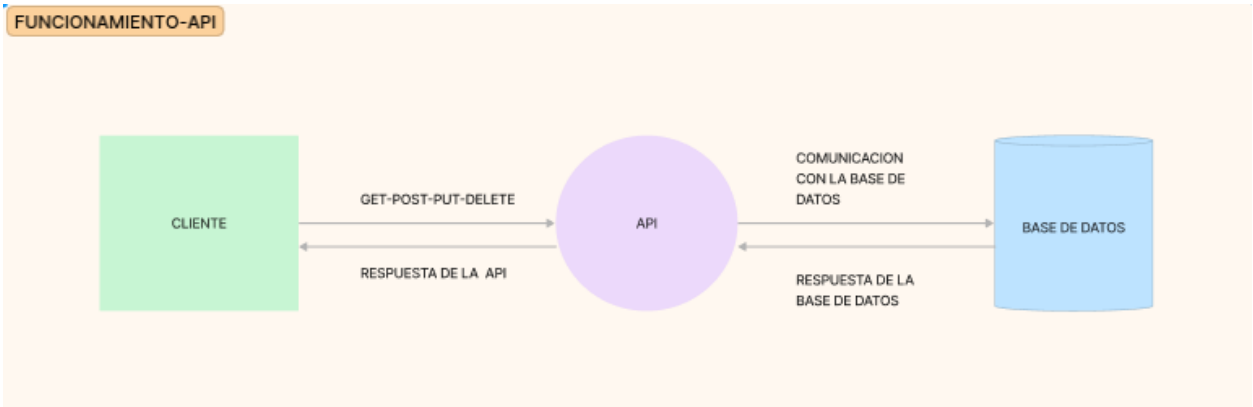


Figura 4.29. Funcionamiento de la API.

Las rutas que se utilizan para el funcionamiento de la interfaz de programación de aplicación son llamadas endpoints, en dichos endpoints es dónde se realiza el llamado al proceso que se llevará a cabo por la aplicación, funcionan como un activador, el cual al ser llamado realiza una operación correspondiente y que, al finalizar retorna un valor, es así como se puede realizar la creación de esta interfaz de manera que ofrece distintas funcionalidades lo cual permite que sean integradas en distintos contextos, en este caso se utilizó para realizar las operaciones necesarias sobre la base de datos lo cual fue de gran ayuda al momento de utilizar los datos en las vistas correspondientes.

El uso de los esquemas fue de suma importancia ya que estos fueron de gran ayuda a la hora de realizar las operaciones sobre la base de datos, por otro lado, gracias al uso de mongoose se realizó la conexión correspondiente a la base de datos y con express la conexión al servidor. Para llevar a cabo las operaciones, fue necesario realizar uso de los distintos métodos http, los cuales permiten, consultar (GET), añadir elementos (POST), borrar(DELETE) y también actualizar(PUT).

A continuación se muestran los endpoints realizados.

Para el manejo de equipos:

En la figura 4.30, se muestran las importaciones necesarias para la creación de los endpoints, express en este caso funciona de enrutador, esto permite que el programa principal pueda referenciar los endpoints que están siendo usados en este contexto. Por otro lado se usa el esquema de la plantilla para realizar la comunicación con la base de datos.

```
const express = require("express");
const PlantillaSchema = require("../models/plantilla");

const router = express.Router();
```

Figura 4.30. Importaciones necesarias para la API de manejo de equipo.

En la figura **4.31**, se muestra el endpoint usado para crear un equipo, este endpoint usa el método POST, toma como parámetros los elementos que se envían en el cuerpo de la petición y guarda la información correspondiente, si todo el proceso se llevó a cabo con normalidad retorna un estado de respuesta 200, caso contrario retorna un estado 500 y un mensaje con el error.

```
// create team
router.post('/squad', (req, res) => {
    const squad = PlantillaSchema(req.body);
    squad
        .save()
        .then((data) => res.json(data))
        .catch((error) =>
            res.status(500).json({ message: error })
            );
});
```

Figura 4.31. Endpoint para crear un equipo.

En la figura **4.32**, se muestra el endpoint usado para encontrar todos los equipos guardados en una categoría, se hace uso del método GET, toma como parámetro de url el id de la categoría a consultar, si el proceso se realiza correctamente retorna un estado de respuesta 200 y un array con la información encontrada (puede estar vacío si no hay equipos creados para esa categoría), por otro lado, si ha ocurrido un error retorna un estado de respuesta 500.

```
// find all teams into category
router.get('/squad/:id', (req, res) => {
    const { id } = req.params;
    PlantillaSchema
        .find({ categoria: id })
        .populate("jugadores")
        .then((data) => res.json(data))
        .catch((error) => res.status(500).json({ message: error }));
});
```

Figura 4.32. Endpoint para buscar equipos dentro de una categoría.

En la figura **4.33,** se muestra el endpoint utilizado para actualizar la información de un equipo, en este caso recibe un parámetro en la url, el cual es el id del equipo a actualizar, así como también en el cuerpo de la petición se envían los nuevos datos que se quieren reemplazar, el estado de respuesta es 200 si todo el proceso se realizó correctamente, realizando la actualización correspondiente, sino el estado de respuesta es 500, indicando así que algo salió mal.

```
//update squad
router.put("/squad/:id", (req, res) => {
    // res.send(req.params)
    const { id } = req.params;
    const { nombre_plantilla, jugadores, categoria } = req.body;
    PlantillaSchema
        .updateOne({ _id: id }, { $set: { nombre_plantilla, jugadores, categoria } })
        .then((data) => res.json(data))
        .catch((error) => res.status(500).json({ message: error }));
});
```

Figura 4.33. Endpoint para actualizar equipo dentro de una categoría.

En la figura **4.34,** se muestra el endpoint utilizado para eliminar un equipo, este recibe el parámetro id en su url, el cual es usado para detectar el elemento a eliminar, si el proceso se lleva a cabo correctamente se elimina el elemento y devuelve un estado de respuesta 200, sino retorna un estado de respuesta 500.


```
//delele squad
router.delete("/squad/:id", (req, res) => {
    // res.send(req.params)
    const { id } = req.params;
    PlantillaSchema
        .remove({ _id: id })
        .then((data) => res.json(data))
        .catch((error) => res.status(500).json({ message: error }));
});
```

Figura 4.34. Endpoint para actualizar equipo dentro de una categoría.

Datos deportivos

En este caso se realiza el uso del esquema de información deportiva para poder realizar operaciones sobre los datos deportivos guardados del jugador. Ver Figura 4.35. Cabe destacar que todos los endpoints contienen un estado de respuesta 500 en caso de que se presente algún error.

```
const express = require("express");
const sportSchema = require("../models/datosDeportivos");

const router = express.Router();
```

Figura 4.35. Importaciones para endpoints de datos deportivos

En la figura 4.36, se muestra en endpoint realizado para la creación de datos deportivos, se realiza el guardado de información en el esquema correspondiente.


```
// create sport data
router.post('/sport', (req, res) => {
    const user = sportSchema(req.body);
    user
        .save()
        .then((data) => res.json(data))
        .catch((error) => res.status(500).json({ message: error }));
});
```

Figura 4.36. Endpoint creación de datos deportivos.

En la figura 4.37, se muestra el endpoint para la consulta de los datos deportivos para un jugador en específico por lo que es necesario usar un id de datos deportivos del jugador

```
//get an specific user
router.get("/sport/:id", (req, res) => {
    const { id } = req.params;
    sportSchema
        .findById(id)
        .then((data) => res.json(data))
        .catch((error) => res.status(500).json({ message: error }));
})
```

Figura 4.37. Endpoint de consulta para información deportiva del usuario.

En la figura 4.38, se muestra el endpoint para realizar la actualización de los datos deportivos de un usuario, por lo que es necesario el id de datos deportivos para su actualización.

```
//update user
router.put("/sport/:id", (req, res) => {
    // res.send(req.params)
    const { id } = req.params;
    const {
        caracteristicas,
        antecedentes_deportivos
    } = req.body;
    sportSchema
        .updateOne({ _id: id }, {
            $set: {
                caracteristicas,
                antecedentes_deportivos
            }
        })
        .then((data) => res.json(data))
        .catch((error) => res.status(500).json({ message: error }));
});
```


Figura 4.38. Endpoint de consulta para información deportiva del usuario

Categoría

En la figura 4.39 se identifican los endpoints utilizados para crear una categoria y tambien para hacer consultas sobre la misma, en este caso el endpoint de crear categoría se realizó como medio de apoyo para llevar a cabo pruebas, puesto que las categorías ya están definidas en la base de datos y realmente en el proyecto se utiliza el método GET para realizar consultaz sobre ella.

```
const express = require("express");
const CategoriaSchema = require("../models/categoria");

const router = express.Router();

// create category
router.post('/category', (req, res) => {
    const categoria = CategoriaSchema(req.body);
    categoria
        .save()
        .then((data) => res.json(data))
        .catch((error) => res.status(500).json({ message: error }));
});

// get category
router.get("/categories", (req, res) => {
    CategoriaSchema
        .find()
        .then((data) => res.json(data))
        .catch((error) => res.status(500).json({ message: error }));
})
```


Figura 4.39. Endpoints utilizados para creación y consulta de categoría.

Jugador

En el caso del jugador se realizaron los endpoints con el objetivo de poder realizar consultas para obtener la información usada en la ficha y también modificar los campos de equipo del jugador, los endpoints creados se muestran a continuación.

```
// get all players
router.get("/jugadores", (req, res) => {
    jugadorSchema
        .find()
        .then((data) => res.json(data))
        .catch((error) => res.status(500).json({ message: error }));
})
```

Figura 4.40. Endpoint para obtener todos los jugadores.

```
//get an specific player
router.get("/jugador/:id", (req, res) => {
    const { id } = req.params;
    jugadorSchema
        .findById(id)
        .then((data) => res.json(data))
        .catch((error) => res.json({ message: error }));
})
```


Figura 4.41. Endpoint para obtener información de un jugador.

```
//update user
router.put("/jugador/:id", (req, res) => {
    // res.send(req.params)
    const { id } = req.params;
    const { datos_deportivos } = req.body;
    jugadorSchema
        .updateOne({ _id: id }, { $set: { datos_deportivos } })
        .then((data) => res.json(data))
        .catch((error) => res.json({ message: error }));
});
```

Figura 4.42. Endpoint para actualizar información deportiva de un jugador.

```
// players category
router.get("/jugadores/category/:id", (req, res) => {
    const { id } = req.params;
    jugadorSchema
        .find({ categoria: id })
        .then((data) => res.json(data))
        .catch((error) => res.json({ message: error }));
})
```

Figura 4.43. . Endpoint para consultar todos los jugadores en una categoría.

```
// update squad
router.put("/jugadores/update/squad/:id", (req, res) => {
    // res.send(req.params)
    const { id } = req.params;
    const { plantilla, numUniforme, posicion, apodo } = req.body;
    jugadorSchema
        .updateOne({ _id: id }, {
            $push: {
                plantilla_jugador:
                {
                    plantilla: plantilla,
                    numUniforme: numUniforme,
                    posicion: posicion,
                    apodo: apodo
                }
            }
        })
        .then((data) => res.json(data))
        .catch((error) => res.json({ message: error }));
});
```


Figura 4.44. Endpoint para actualizar la información de equipo del jugador.

Vale la pena destacar que, para el manejo de los endpoints existe una ruta principal que ha sido creada con el objetivo de que funcione como punto principal para el manejo de las APIS, aquí también se realizaron las conexiones correspondientes a la base de datos y el uso del servidor. Se puede observar que para todas las rutas se ha colocado el prefijo “/api”. Ver figura 4.45

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();


const jugadorRoutes = require('./routes/jugador');
const datosDeportivosRoutes = require('./routes/datosDeportivos');
const categoriesRoutes = require('./routes/categoria')
const squadRoutes = require('./routes/plantilla')
// const jugadorRoutesA = require('./routes/jugador_a');

const app = express();
app.use(cors());
port = process.env.PORT || 9000;

//middlewares
app.use(express.json());
app.use('/api', jugadorRoutes);
app.use('/api', datosDeportivosRoutes);
app.use('/api', categoriesRoutes);
app.use('/api', squadRoutes);
// app.use('/api', jugadorRoutesA);

//routes
app.get('/', (req, res) => {
    res.send('Welcome to my API');
});


// mongodb connection
mongoose
    .connect(process.env.MONGODB_URI)
    .then(() => console.log('Connected to MONGODB ATLAS'))
    .catch((error) => console.error(error));


app.listen(port, () => console.log('server running on port', port));
```


Figura 4.45. Archivo principal para el manejo de las APIS.

4.6 Pruebas

En el caso del Front-end las pruebas se realizaron progresivamente a medida que se crearon las vistas por lo que en este caso se realizaron de forma manual, haciendo uso de las funcionalidades pertenecientes a cada uno de los componentes, no se realizó uso de algún software para realizar pruebas.

Por otro lado, para las pruebas del lado del Back-end se utilizó el framework jest, el cual es uno de los más utilizados actualmente, junto con la librería supertest, con la cual se realiza el uso de los endpoints, en esta oportunidad las pruebas se llevaron a cabo sobre las APIS que han sido creadas. Se ejecutaron distintos casos de sobre la API y se comprobó si el estado de respuesta y los datos eran o no los esperados, en caso de fallo se revisaron los esquemas y los endpoints en busca de dar solución al mismo.

A continuación se muestran algunas pruebas realizadas sobre las APIS.

4.6.1 API de datos deportivos

Endpoint /api/squad para añadir un equipo (método POST), se realizó la solicitud a la ruta correspondiente y se crearon dos casos de prueba, uno donde todo funciona correctamente y otro donde el estado de respuesta debe ser 500, para esto se generó la prueba correspondiente. Ver figura **4.46**. La prueba fue realizada tomando el estado y la data de respuesta, debido a que en mongoDB se genera un Id al crear un nuevo documento, una de las condiciones es, que ese id exista, una vez que se comprueba su existencia, se lleva a cabo la siguiente condición la cual es que la data guardada corresponda con la ingresada y que los tipos de datos de respuesta sean los mismos, en general todas las pruebas realizadas sobre la API se llevaron a cabo de esta manera. De igual forma, se introdujo data que no corresponde con el formato del

esquema por lo que su estado de respuesta fue 500, funcionando de acuerdo a lo esperado.

Endpoint /api/squad para actualizar un equipo (método PUT), mediante la creación de un documento de prueba se pudo realizar el uso de la API y tomando la respuesta tanto de su estado como su cuerpo, se pudo constatar que la respuesta fue la esperada. Ver figura **4.47**

Endpoint /api/squad/:id para obtener los equipos disponibles en una categoría (método GET), mediante la creación de un documento de prueba usando el esquema de categoría se pudo realizar el uso de la API y tomando la respuesta tanto de su estado como su cuerpo, se pudo constatar que el resultado fue el esperado. En este caso se realizaron comparaciones de tipo donde se comprobó que devuelve un array en el cuerpo de respuesta, además del estado 200, cumpliendo con lo esperado. Ver figura **4.48**

Endpoint /api/squad/:id para borrar un equipo (método DELETE), en este caso se creó un documento de prueba, esto con la finalidad de obtener su id, una vez obtenido el id, se realizó el uso del endpoint para borrarlo, se compararon el estado y los datos de respuesta ,los cuales indican que el proceso se llevó a cabo correctamente, tomando en cuenta el estado 200 y el deletedCount = 1(indica que se borró un elemento), obteniendo así una operación satisfactoria. Ver figura **4.49**

Los resultados obtenidos de las pruebas se pueden visualizar en la figura **4.50**

```
//POST TEST
describe('POST /api/squad', () => {

    const NewSquad = {
        nombre_plantilla: "Test",
        jugadores: ["6352f56114567a398a655a3e", "6352f56114567a398a655a3e"],
        categoria: "6351cb38e343987c2722f216"
    };

    const BadSquad = { plantilla: "Test", jugadores: "Testing" };

    it('Good insertion', async () => {
        const response = await request(app).post('/api/squad').send(NewSquad);

        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
        expect(response.body._id).toBeDefined();
        expect(response.body.nombre_plantilla).toBe(NewSquad.nombre_plantilla);
        expect(response.body.jugadores).toBeInstanceOf(Array);
        expect(response.body.jugadores).toBeInstanceOf(Array);

    }, 100000);

    it('Bad insertion', async () => {
        const response = await request(app).post('/api/squad').send(BadSquad);

        expect(response.status).toBe(500);
    }, 10000);

});
```

Figura 4.46. Prueba con método POST al endpoint /api/squad de datos deportivos.


```
//PUT TEST
describe('PUT /api/squad', () => {

    let squad;
    update_object = {
        nombre_plantilla: 'Test2'
    }
    beforeEach(async () => {
        squad = await PlantillaSchema.create({
            nombre_plantilla: "Test",
            jugadores: ["6352f56114567a398a655a3e", "6352f56114567a398a655a3e"],
            categoria: "6351cb38e343987c2722f216"
        })
    });

    afterEach(async () => {
        await PlantillaSchema.findByIdAndDelete(squad._id);
    });

    it('Route works as expected', async () => {
        const response = await request(app).put(`/api/squad/${squad._id}`).send(update_object);

        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
    });

    it('Updated successfully', async () => {
        const response = await request(app).put(`/api/squad/${squad._id}`).send(update_object);

        expect(response.body.modifiedCount).toBe(1);
        expect(response.body.acknowledged).toBe(true)
        expect(response.body.matchedCount).toBe(1);
        // expect(response.body.name).toBe(update_object.nombre_plantilla);
    });

});
```

Figura 4.47. Prueba con método PUT al endpoint /api/squad de datos deportivos.

```
describe('GET /api/squad/{id}', () => {

    let response;
    let category;
    beforeEach(async () => {
        category = await CategoriaSchema.create({
            nombre: "Sub-Test",
            categoria_edad: 6,
            años: "2006-2007"
        })
        response = await request(app).get(`/api/squad/${category._id}`).send();
    })

    afterEach(async () => {
        await CategoriaSchema.findByIdAndDelete(category._id);
    });

    it('Route works as expected', async () => {
        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
    });

    it('Get request return an array', async () => {
        expect(response.body).toBeInstanceOf(Array);
    });

});
```

Figura 4.48. Prueba con método GET al endpoint /api/squad/:id de datos deportivos.


```
//DELETE TEST
describe('DELETE /api/squad', () => {

    const NewSquad = {
        nombre_plantilla: "Test",
        jugadores: ["6352f56114567a398a655a3e", "6352f56114567a398a655a3e"],
        categoria: "6351cb38e343987c2722f216"
    };
    let squad;
    let response;
    beforeEach(async () => {
        squad = await PlantillaSchema.create(NewSquad);
        response = await request(app).delete(`/api/squad/${squad._id}`).send();
    });

    it('Route works as expected', () => {
        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
    });

    it('Deleted succesfully', async () => {
        expect(response.body.deletedCount).toBe(1);
        expect(response.body.acknowledged).toBe(true);

        const foundSquad = await PlantillaSchema.findById(squad._id);
        expect(foundSquad).toBeNull();
    })
});
```

Figura 4.49. Prueba con método DELETE al endpoint /api/squad/ de datos deportivos.

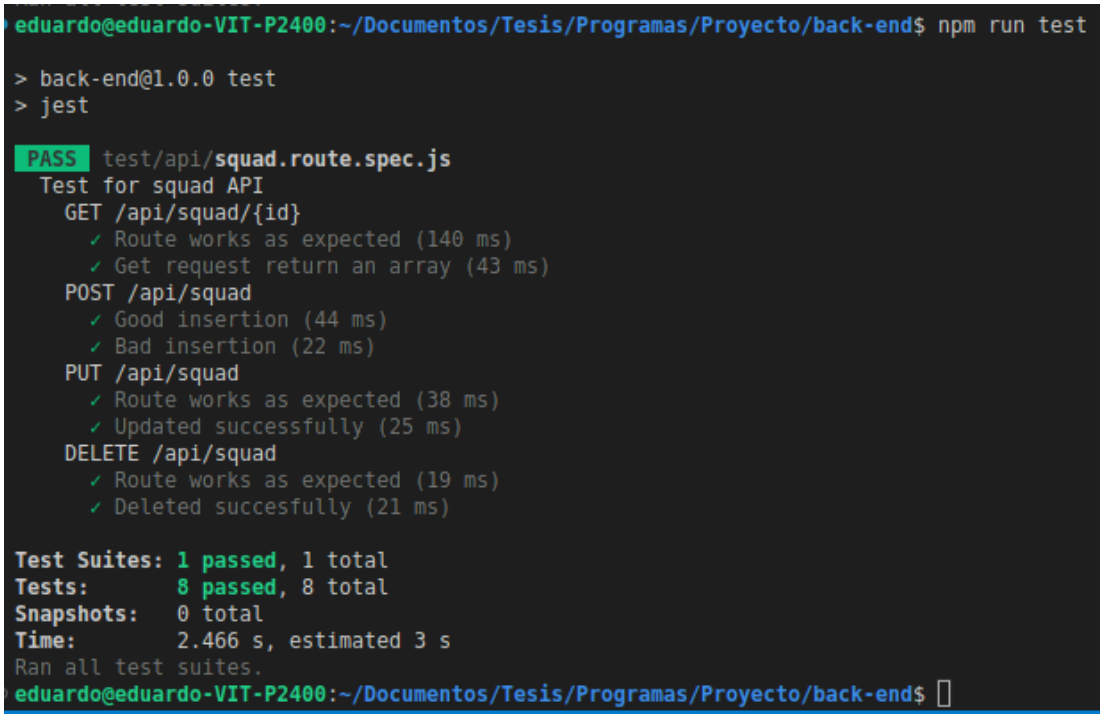


Figura 4.50. Resultado de pruebas realizadas a la API de datos deportivos.

4.6.2 API de categoría

En esta oportunidad se realizaron las pruebas respectivas sobre el endpoint /api/categories, el cual devuelve un array de objetos con las categorías disponibles, utilizando el método GET se realizó la consulta en la base de datos. La prueba realizada contempló la verificación de la ruta, es decir, se comprobó que funciona correctamente, así como también que el objeto devuelto es un array. El test realizado para este endpoint se muestra en la figura ***4.51***. El resultado en la figura ***4.52.***

```
const request = require('supertest');
const mongoose = require('mongoose');
const app = require('../../src/index_dev');

describe('Test for category endpoint', () => {
    beforeAll(async () => {
        client = mongoose.connect('mongodb://127.0.0.1/test');
    });

    afterAll(async () => {
        await mongoose.disconnect();
    });

    describe('GET /api/categories', () => {

        let response;
        beforeEach(async () => {
            response = await request(app).get('/api/categories').send();
        })

        it('Route works as expected', async () => {
            expect(response.status).toBe(200);
            expect(response.headers['content-type']).toContain('json');
        });

        it('Get request return an array', async () => {
            expect(response.body).toBeInstanceOf(Array);
        });

    });

})
```


Figura 4.51. Prueba realizada al endpoint /api/categories.

```
eduardo@eduardo-VIT-P2400:~/Documentos/Tesis/Programas/Proyecto/back-end$ npm run test

> back-end@1.0.0 test
> jest

 PASS  test/api/category.route.spec.js
  Test for category endpoint
    GET /api/categories
      ✓ Route works as expected (106 ms)
      ✓ Get request return an array (19 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.04 s, estimated 3 s
Ran all test suites.
eduardo@eduardo-VIT-P2400:~/Documentos/Tesis/Programas/Proyecto/back-end$
```

Figura 4.52. Resultado a prueba realizada al endpoint /api/categories.

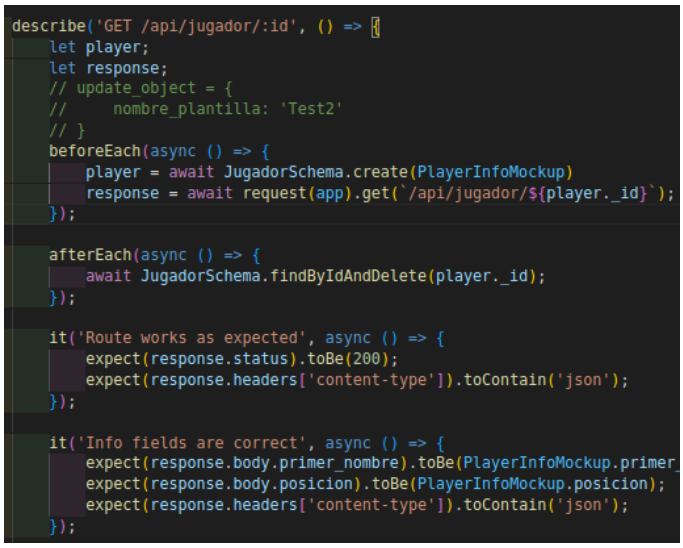
4.6.3 API de jugador

Endpoint /api/jugadores, haciendo uso del método GET, se comprueba que la ruta funciona correctamente y devuelve un array. Ver figura **4.53**


```
describe('GET /api/jugadores', () => {
    let response;
    beforeEach(async () => {
        response = await request(app).get('/api/jugadores').send();
    })

    it('Route works as expected', async () => {
        // const response = await request(app).get('/').send();

        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
    });

    it('Get request return an array', async () => {
        expect(response.body).toBeInstanceOf(Array);
    });

});
```

Figura 4.53. Prueba con el método GET realizada al endpoint /api/jugadores.

Endpoint /api/jugador/:id, haciendo uso del método GET, se comprueba que la ruta funciona correctamente y devuelve los datos correspondientes al jugador insertado. Ver figura **4.54**



```
describe('GET /api/jugador/:id', () => {
    let player;
    let response;
    // update_object = {
    //     nombre_plantilla: 'Test2'
    // }
    beforeEach(async () => {
        player = await JugadorSchema.create(PlayerInfoMockup)
        response = await request(app).get(`/api/jugador/${player._id}`);
    });

    afterEach(async () => {
        await JugadorSchema.findByIdAndDelete(player._id);
    });

    it('Route works as expected', async () => {
        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
    });

    it('Info fields are correct', async () => {
        expect(response.body.primer_nombre).toBe(PlayerInfoMockup.primer_
        expect(response.body.posicion).toBe(PlayerInfoMockup.posicion);
        expect(response.headers['content-type']).toContain('json');
    });
```

Figura 4.54. Prueba con el método GET realizada al endpoint /api/jugador/:id.

Endpoint /api/jugador/:id, haciendo uso del método PUT se comprueba que la ruta funciona correctamente y devuelve los datos correspondientes que indican que el jugador ha sido actualizado. Ver figura **4.55**

```
describe('PUT /api/jugador/:id', () => {
    let player;
    let response;
    beforeEach(async () => {
        player = await JugadorSchema.create(PlayerInfoMockup)
        response = await request(app).put(`/api/jugador/${player._id}`).send(update_object);
    });

    afterEach(async () => {
        await JugadorSchema.findByIdAndDelete(player._id);
    });

    it('Route works as expected', async () => {
        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
    });

    it('Updated successfully', async () => {
        response = await request(app).put(`/api/jugador/${player._id}`).send(update_object);
        // const response = await request(app).put(`/api/squad/${squad._id}`).send(update_obj

        expect(response.body.modifiedCount).toBe(1);
        expect(response.body.acknowledged).toBe(true)
        expect(response.body.matchedCount).toBe(1);
        // expect(response.body.name).toBe(update_object.nombre_plantilla);
    });
});
```

Figura 4.55. Prueba con el método PUT realizada al endpoint /api/jugador/:id.

Endpoint /api/jugador/:id, haciendo uso del método GET, se comprueba que la ruta funciona correctamente y devuelve los datos correspondientes que indican que el la respuesta obtenida es la esperada. Ver figura **4.56**

Endpoint /api/jugador/:id, haciendo uso del método PUT, se comprueba que la ruta funciona correctamente y devuelve los datos correspondientes que indican que el la información sobre el jugador ha sido actualizada. Ver figura **4.57**

Resultados de las pruebas realizadas. Ver figura **4.58**.

```
describe('GET /api/jugadores/category/:id', () => {

    let response;
    let category;
    beforeEach(async () => {
        category = await CategoriaSchema.create({
            nombre: "Sub-Test",
            categoria_edad: 6,
            años: "2006-2007"
        })
        response = await request(app).get(`/api/jugadores/category/${category._id}`).send();
    });

    afterEach(async () => {
        await CategoriaSchema.findByIdAndDelete(category._id);
    });

    it('Route works as expected', async () => {
        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
    });

    it('Get request return an array', async () => {
        expect(response.body).toBeInstanceOf(Array);
    });
});
```

Figura 4.56. Prueba con el método GET realizada al endpoint /api/jugadores/category/:id.

```
describe('PUT /api/jugadores/update/squad/:id', () => {
    let player;
    let response;
    let bad_response;
    const update_squad = {
        "plantilla": "63555650748bc589e8d89fc2",
        "numUniforme": "15",
        "posicion": "Delantero",
        "apodo": "Eduardo"
    }

    beforeEach(async () => {
        player = await JugadorSchema.create(PlayerInfoMockup)
        response = await request(app).put(`/api/jugador/${player._id}`).send(update_squad);
    });

    afterEach(async () => {
        await JugadorSchema.findByIdAndDelete(player._id);
    });

    it('Route works as expected', async () => {
        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
    });

    it('Updated successfully', async () => {
        // response = await request(app).put('/api/jugador/63635892504c83698c844a23').send(up
        // const response = await request(app).put(`/api/squad/${squad._id}`).send(update_obj

        expect(response.body.modifiedCount).toBe(1);
        expect(response.body.acknowledged).toBe(true)
        expect(response.body.matchedCount).toBe(1);
        // expect(response.body.name).toBe(update_object.nombre_plantilla);
    });
});
```

Figura 4.57. Prueba con el método PUT realizada al endpoint /api/jugadores/update/squad/:id.

```
describe('PUT /api/jugadores/update/squad/:id', () => {
    let player;
    let response;
    let bad_response;
    const update_squad = {
        "plantilla": "63555650748bc589e8d89fc2",
        "numUniforme": "15",
        "posicion": "Delantero",
        "apodo": "Eduardo"
    }

    beforeEach(async () => {
        player = await JugadorSchema.create(PlayerInfoMockup)
        response = await request(app).put(`/api/jugador/${player._id}`).send(update_squad);
    });

    afterEach(async () => {
        await JugadorSchema.findByIdAndDelete(player._id);
    });

    it('Route works as expected', async () => {
        expect(response.status).toBe(200);
        expect(response.headers['content-type']).toContain('json');
    });

    it('Updated successfully', async () => {
        // response = await request(app).put('/api/jugador/63635892504c83698c844a23').send(up
        // const response = await request(app).put(`/api/squad/${squad._id}`).send(update_obj

        expect(response.body.modifiedCount).toBe(1);
        expect(response.body.acknowledged).toBe(true)
        expect(response.body.matchedCount).toBe(1);
        // expect(response.body.name).toBe(update_object.nombre_plantilla);
    });
});
```

Figura 4.58. Resultado prueba de API de jugador

4.7 Integración

Una vez realizadas las funcionalidades correspondientes, tanto para el Backend como el Frontend, se utilizó la API creada en los componentes principales de cada una de las vistas, por lo que su manejo fue sencillo, esto permitió tener de manera aislada ambas partes, siendo un punto positivo ya que a la hora de realizar cambios se pueden realizar en el entorno que corresponda ya sea la parte visual o el manejo de datos. En líneas generales al realizar la integración se pudo notar que es una buena práctica construir el funcionamiento de las vistas y de los endpoints lo más general posible, así, el producto creado puede adaptarse a los cambios que correspondan e incluso usarse en diferentes contextos cuando se haya presentado algún cambio en los datos sin que esto afecte la estructura y funcionamiento general de las partes.

Capítulo V

Conclusión y trabajos futuros

5.1 Conclusiones

Este proyecto de grado se centró en la creación de una aplicación web que permite el manejo dinámico de la información sobre los jugadores de la Academia Emeritense F.C, abarca desde el estudio de la información a utilizar, hasta la creación de estructuras visuales y de datos que permiten el correcto manejo de la misma, todo esto haciendo uso de herramientas tecnológicas que dieron paso al desarrollo de este juego de funcionalidades, proporcionando facilidad al uso de la información pertinente al jugador, siendo así, un aporte positivo de cara al manejo de estos datos. Tomando en cuenta esto, se crearon los mecanismos necesarios para permitir que el usuario pueda manejar los datos de manera cómoda, ya sea visualizando la ficha del jugador, creando algún equipo o buscando información sobre algún jugador.

Este juego de funcionalidades mencionado anteriormente, ofrece una ficha deportiva con un conjunto de información completa sobre el jugador, ya sea en el ámbito personal, deportivo o médico, por otro lado, se pueden crear equipos dentro de una categoría, así como también realizar la búsqueda de algún jugador haciendo uso de atributos como nombre, posición y categoría.

Vale la pena destacar que para el desarrollo se siguieron un conjunto de pasos que fueron sumamente importantes para la consecución del mismo como lo fueron la creación de los wireframes y de los esquemas de bases de datos que dieron paso a un proceso de programación más ameno teniendo esta información como base.

La serie de herramientas usadas para llevar a cabo el desarrollo en este caso las más generales como lo son MongoDB, Express, React y Node funcionan muy bien en conjunto, además de ofrecer facilidades a la hora de desarrollar.

5.2 Trabajos futuros

- Integrar las funcionalidades en la plataforma de la Academia Emeritense FC.

- Realizar pruebas de uso en casos más generales con los miembros de la Academia en busca de mejoras.

- Nueva funcionalidad en la creación de equipo que permita descargar la plantilla creada.

Referencias

[1] Rafael Camps Paré,Luis Alberto Casillas Santillán,Dolors Costal Costa,Marc Gibert Ginestà,Carme Martín Escofet,Oscar Pérez Mora. (2005). *Bases de datos*. (primera edición ed.). Eureca Media, SL.

[2] Mercedes Marqués. (2011). *Base de datos.(*primera edición ed.*). Col·lecció Sapientia, 18*

[3] Red Hat .(31 de Octubre de 2017). ¿Qué es una API?https://www.redhat.com/es/topics/api/what- are-application-programming-interfaces

[4] Iván Jahel Bautista García.(30 de marzo de 2021).Backend y Frontend, ¿Qué es y cómo funcionan en la programación? https://www.servnet.mx/blog/backend-y-frontend-partes-fundamentales-de-la- programac%C3%B3n-de-una-aplicaci%C3%B3n-web

[5] Nicole Chapaval (2017). Qué es Frontend y Backend: diferencias y características – Platzi. https://platzi.com/blog/que-es-frontend-y-backend/

[6] DataScientest (7 de abril 2022). MongoDB : todo sobre la base de datos NoSQL orientada a documentos. https://datascientest.com/es/mongodb-todo-sobre-la-base-de-datos-nosql-orientada-a- documentos

[7] MDN contributors. (20 de abril de 2021). JavaScript : ¿Quieres convertirte en un desarrollador web front-end? https://developer.mozilla.org/es/docs/Web/HTML

[8] Henry(1 de nov. De 2021).¿Qué es Node.js y para qué se utiliza?. https://blog.soyhenry.com/que-es-node-js-y-para-que-se-utiliza/

[9] RACKSPACE TECHNOLOGY(s.f). ¿Qué son las bases de datos NoSQL?.https://www.rackspace.com/es/library/what-is-a-nosql-database

[10] Wikipedia(d.f). React. https://es.wikipedia.org/wiki/React

[11] Abellán, E. (5 de marzo del 2020).Scrum: qué es y cómo funciona esta metodología. We are marketing. https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html

[12] MDN contributors. (28 de septiembre 2022). Métodos de petición HTTP. https://developer.mozilla.org/es/docs/Web/HTTP/Methods

[13] Escobar, G. (17 de septiembre 2015). ¿Qué es un API?.https://blog.makeitreal.camp/que-es-un-api/