Final de proyecto

TFG DAW

Web dogginer

Autores

Pablo Fernández Sato
David Fernández Peralta
Jana Montero Perales

Introducción

Este Trabajo de Fin de Grado surge con la idea de desarrollar una aplicación web que actúe como punto de encuentro entre Adiestradores caninos y Clientes, basándose en modelos existentes de plataformas de conceptos similares (por ejemplo, https://www.bodas.net/).

El tema elegido, adiestramiento canino, se basa en las experiencias personales de los integrantes del grupo. En su convivencia con mascotas, han necesitado ayuda a la hora de resolver problemas específicos, adiestramiento general o incluso marcos de socialización con otros animales. En el pasado se ha recurrido a círculos locales, ante la falta una plataforma unificada más amplia.

El objetivo de esta aplicación es establecer una plataforma donde familias con mascotas puedan encontrar a adiestradores, ponerse en contacto con ellos protegiendo su privacidad y acceder a eventos relacionados con mascotas.

Índice

1. RESUMEN

Dogginer es una Aplicación Web SPA (Single Page Application) que actúa como plataforma para poner en contacto a dueños de mascotas (clientes) con adiestradores. Facilita la búsqueda de adiestradores, la publicación de eventos y el contacto entre adiestradores y clientes.

La privacidad y la protección de datos personales han sido una prioridad a la hora de diseñar la aplicación, por lo que las interacciones entre clientes y adiestradores se realizan de manera segura.

La funcionalidad de la página variará dependiendo de si el usuario está logueado o no y, en caso de que lo esté, de cuál es su rol.

Usuario no logueado

El usuario no logueado podrá acceder a los perfiles de adiestrador, así como la lista de eventos públicos.

Dentro de la vista de adiestrador, podrá realizar búsquedas por nombre y aplicar filtros de búsqueda basados en etiquetas (por ejemplo “cachorros”). Cuando se aplican varios filtros, los resultados incluirán adiestradores que cumplan al menos uno de ellos.

Un usuario no logueado no podrá contactar con un adiestrador ni registrarse a ningún evento.

Un usuario no logueado puede además registrarse (crear una cuenta) como adiestrador o como cliente, o loguearse si ya tiene una cuenta.

Nota: un usuario tan sólo se podrá registrar con una cuenta de email que no exista ya en la Base de Datos (BD). Esto significa que una misma cuenta de email no puede estar asociada a más de un usuario.

Usuario logueado como adiestrador

Un usuario logueado como adiestrador podrá, además de acceder a la lista de eventos públicos y de perfiles de adiestrador, realizar las siguientes acciones:

- Acceder a una lista de eventos organizados por sí mismo, ya sean públicos o privados.
- Crear eventos - ya sean públicos (visibles para todos los usuarios y accesibles para registro de cualquier cliente) o privados (especificando qué clientes podrán registrarse, pudiendo acceder a los usernames de clientes registrados).
- Cancelar eventos organizados por sí mismo.

- Contactar con un cliente, siempre que el cliente esté registrado en al menos un evento del adiestrador (esta medida se toma para evitar que adiestradores puedan contactar con todos los usuarios, como medida antispam).
- Enviar un anuncio (broadcast) a todos los clientes registrados en alguno de sus eventos.

Usuario logueado como cliente

Un usuario logueado como adiestrador podrá, además de acceder a la lista de eventos públicos y de perfiles de adiestrador, realizar las siguientes acciones:

- Acceder a una lista de eventos en los que está registrado, ya sean públicos o privados.
- Registrarse a eventos futuros, siempre que sean públicos (y no se haya alcanzado un número de registros igual al aforo máximo) o que el usuario haya sido invitado.
- Cancelar su asistencia a eventos futuros.
- Evaluar a un adiestrador, siempre que esté registrado en al menos un evento organizado por dicho adiestrador - esta iteración de la aplicación no incluye funcionalidad de confirmación de asistencia, por lo que el registro es suficiente.
- Contactar con cualquier adiestrador (por ejemplo, para solicitar una sesión de adiestramiento privada).

Nota: los contactos entre cliente y adiestrador se gestionan por emails enviados desde la aplicación a la dirección de email asociada a la cuenta de usuario. Para proteger la privacidad de los usuarios, en ningún momento se comparten sus direcciones de email.

2. MÓDULOS FORMATIVOS APLICADOS EN EL TRABAJO

Este proyecto utiliza las bases obtenidas durante el curso en las siguientes asignaturas:

- **Programación:** La web está realizada siguiendo las pautas de programación aprendidas durante estos dos años tanto en la asignatura de Programación, como en la de Desarrollo de Aplicaciones Web en Entorno Servidor.
- **Entornos de Desarrollo:** Durante el desarrollo de este proyecto se ha Git como herramienta de control de versiones, usando GitHub para los repositorios remotos, lo que ha facilitado el trabajo en paralelo sobre el mismo código. También se han usado Diagramas de Casos de Uso y documentado el código en el back-end siguiendo las pautas aprendidas.
- **Lenguajes de Marca:** Como lenguaje base de presentación de las vistas se ha utilizado HTML5 y CSS3.
- **Desarrollo de Aplicaciones Web Entorno Cliente:** Ha sido una pieza fundamental al estar basadas casi todas las tecnologías utilizadas en javascript, aparte del uso de funciones javascript tanto en el back como en el front.
- **Diseño de Interfaces:** Se ha usado Flexbox como modelo de diseño CSS.
- **Despliegue de Aplicaciones:** Al desplegar el back-end se ha usado parte de los conocimientos aprendidos en la asignatura, así como investigación propia.

3. HERRAMIENTAS/LENGUAJES UTILIZADOS

Git/GitHub

Se ha usado la herramienta Git para el desarrollo del proyecto, y GitHub como repositorio remoto, lo que ha facilitado el trabajo en paralelo y a distancia. También han servido los commit para poder ir controlando los avances en el proyecto.

Vue.js

Vue.js es un *framework* de javascript de código abierto que proporciona un modelo de programación declarativo y basado en componentes que sirve para la creación de interfaces de usuario y *Single Page Applications* (SPA). Amplía el lenguaje HTML estándar con el uso de templates que permiten modificar y actualizar el árbol DOM dependiendo del estado del javascript. A su vez, es completamente flexible y se adapta fácilmente a diversos casos de uso. La estructura de componentes de Vue.js permite dividir y encapsular el código reutilizable, facilitando su implementación en distintas partes del proyecto. Los Componentes pueden estar formados a su vez de otros Componentes más pequeños, de esta forma se puede ir simplificando su lógica, lo que facilita tener ciertos Componentes que se especializan en tareas muy concretas, posibilitando su implementación allá donde sean necesarios. En este proyecto se han utilizado dos tipos de **Componentes**. Por un lado, componentes propios dónde se ha definido la lógica necesaria que era necesario que cumpliese (como los componentes de formulario), mientras que por otro, se han utilizado aquellos componentes que, mantenidos por la comunidad, facilitan ciertas funcionalidades (cómo el Multiselect).

MongoDB

Gestor de BBDD no relacional, basada en documentos y organizada en colecciones. La flexibilidad de su estructura es ideal para aplicaciones con requisitos cambiantes. Permite definir el esquema de cada tipo de documento, con distinto grado de flexibilidad. Las relaciones se pueden representar de dos maneras distintas: documentos embebidos o referencias de ids. Una característica muy potente de Mongo es la versatilidad de las funciones *aggregate*, que permiten procesar y agrupar documentos y realizar computaciones de manera rápida.

Mongoose

Librería de modelo de datos que facilita la integración de Mongo en aplicaciones de Node.js. Simplifica el código para implementar la conexión, definir los esquemas y realizar operaciones.

Node.js

Entorno de ejecución de javascript como back-end. Es open source y dispone de multitud de librerías open source bien documentadas y fáciles de incorporar al proyecto. Al instalar Node.js, instalamos también NPM, un gestor de paquetes que te permite instalar, actualizar y desinstalar módulos de manera sencilla por medio de la shell. Esto resulta en una aplicación fácilmente portable y sencilla de mantener.

Express.js

Librería de Node.js que permite gestionar rutas de forma sencilla. La aplicación se crea como un servidor *express* que utiliza distintos routers. Las rutas y el middleware se implementan en forma de routers de *express*.

Swagger

Herramienta open source para definir y documentar APIs siguiendo el protocolo OpenAPI. El editor de swagger genera una documentación fácil de comprender y visualizar, así como un marco para enviar directamente requests al servidor.

Sendgrid

Sendgrid es un servicio de envío de emails, que permite realizar esta operación programáticamente. Además de evitar la necesidad de crear un servidor SMTP, simplifica el desarrollo de esta operación y es reconocido por distintos proveedores de email (lo que ayuda a evitar que los emails enviados se clasifiquen como spam). Se ha escogido este servicio frente a otros (como MailChimp) por la simplicidad de integración con Node.js y por disponer de una versión gratuita permanente al mantenerse por debajo de 100 emails diarios.

Amazon Web Services (AWS)

Plataforma cloud elástica con servicios de hosting especializados en distintas configuraciones. AWS ofrece distintas modalidades de servicio (desde infraestructura customizable hasta software como servicio). Hay incontables servicios de hosting. La elección de AWS se basa en tres factores:

- Tier gratuito que cubre las necesidades del proyecto.
- Servicio específico para aplicaciones desarrolladas con Node.js.
- Simplicidad del despliegue, lo que facilita los cambios frecuentes característicos de un proyecto en desarrollo.

Figma

Es un software gratuito que se usa para prototipos de páginas web donde permite ver como va ser el aspecto final de la web pero sin tener que meter código en html y css.

4. COMPONENTES DEL EQUIPO

Jana Montero Perales

Diseño e implementación del BE (API y BD), búsqueda de adiestradores en FE y vista de Eventos en el FE.

Pablo Fernández Sato

Lógica de cliente y mayoría de vistas en el FE (diseño por componentes).

David Fernández Peralta

Experiencia de usuario, diseño de interfaz e implementación de estilos

5. FASES DEL PROYECTO

El proyecto se ha realizado de manera iterativa, reajustando el código e incluso el diseño a lo largo de toda la evolución.

Cabe resaltar que, en general, el back-end (BE), el front-end (FE) y el diseño de la experiencia de usuario y la interfaz (UX/UI) se han trabajado en paralelo, aunque siempre ha habido dependencia de que la funcionalidad estuviese implementada en el BE antes de poder completarla en el FE.

5.1. Casos de uso

El concepto inicial del proyecto consistió en la definición de objetivos. Para ello se realizaron casos de uso basados en la trayectoria esperada de un usuario por la aplicación:

Usuario anónimo

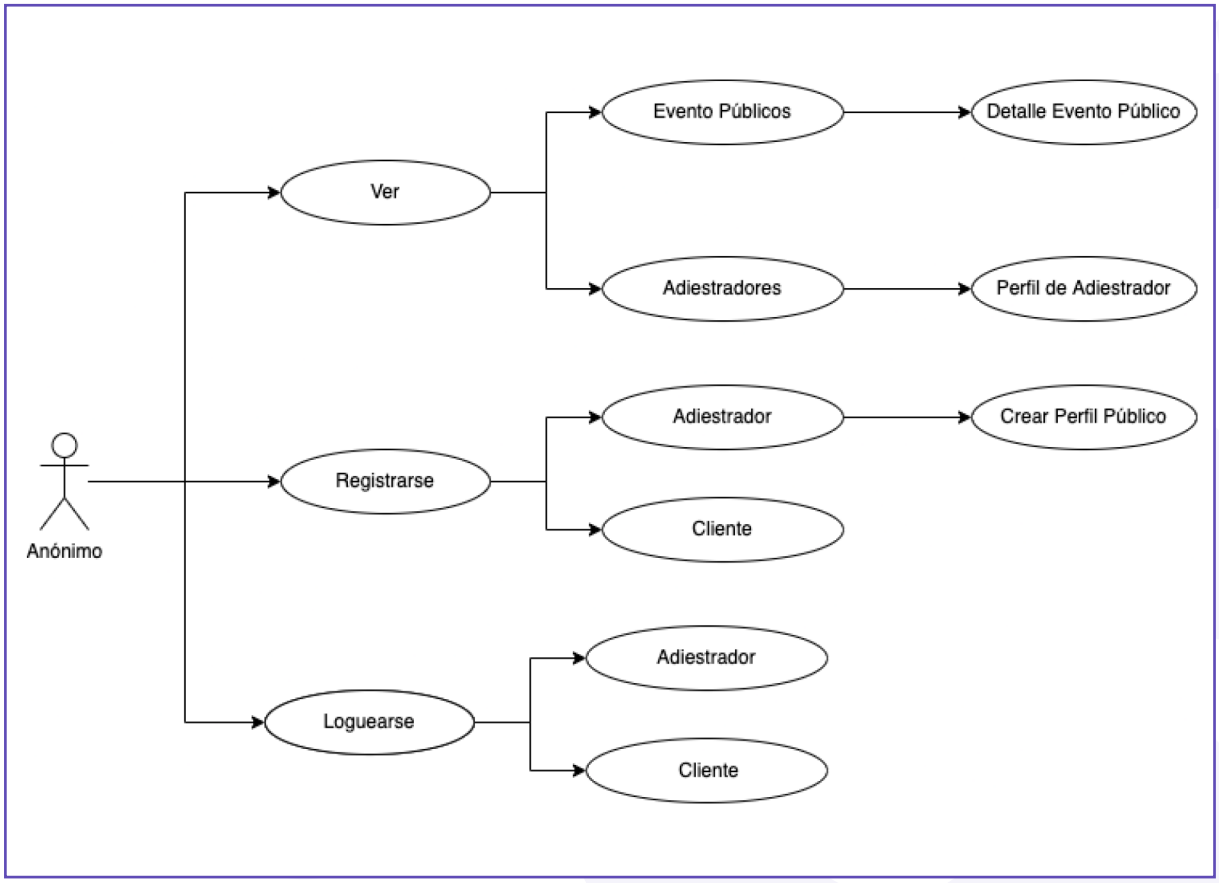

Fig. 1. Casos de uso, usuario no logeado.

En este Proyecto permite la navegación a usuarios anónimos, pero con una funcionalidad muy limitada. Un usuario anónimo podrá:

- **Navegar por la Web:** Con acceso a las siguientes vistas:
  - **Eventos:** Donde podrá navegar a través de la lista de eventos públicos y podrá acceder a sus detalles al hacer click sobre ellos.
  - **Adiestradores:** Podrá navegar a través de la lista de adiestradores, podrá realizar búsquedas por nombre y filtro. También podrá ver el perfil público del adiestrador y los eventos públicos del mismo.
- **Registrarse en la Web:** Un usuario anónimo podrá registrarse en nuestra web ya sea como:
  - Adiestrador.
  - Cliente.
- **Loguearse:** Un usuario anónimo obviamente podrá acceder al login para identificarse y dejar de ser anónimo.

Usuario cliente

Un cliente tendrá una funcionalidad ampliada y definida. Un cliente podrá acceder a:

- Dos vistas de eventos:
  - **Eventos públicos:** donde verá la lista completa de eventos públicos de los que podrá ver el detalle del evento, confirmar su asistencia o cancelarla.
  - **Mis Eventos:** verá una lista de todos los eventos, públicos y privados, a los que ha confirmado su asistencia.
- **Vista de adiestradores:**
  - Podrá Valorar al Adiestrador (siempre y cuando esté registrado a uno de sus eventos).
  - Podrá ponerse en contacto con el adiestrador.
  - Podrá ver los eventos de ese adiestrador.

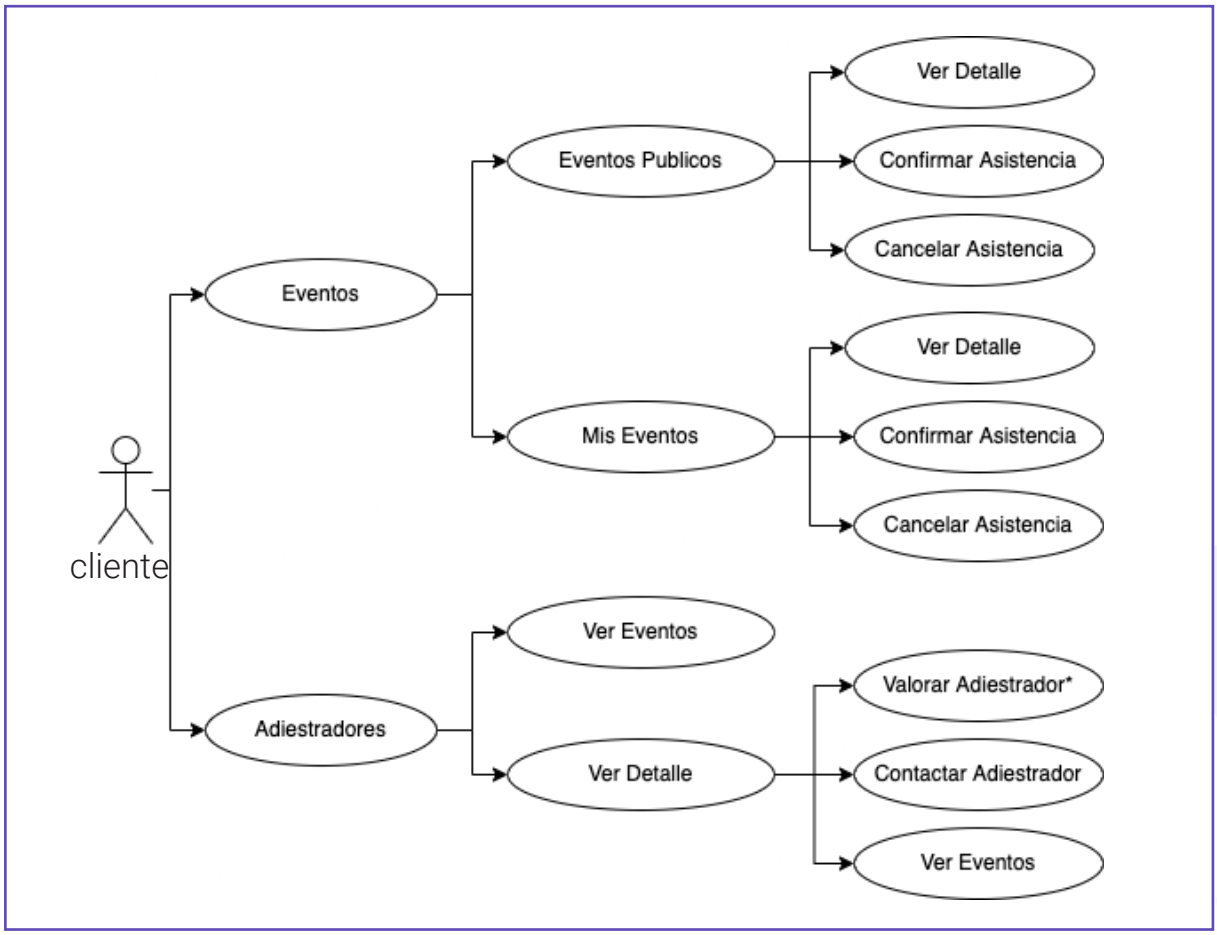

Fig. 2. Casos de uso, cliente logeado.

Usuario Adiestrador

Un usuario registrado como Adiestrador tendrá una funcionalidad ampliada y definida. Un Adiestrador podrá acceder a:

- Dos vistas de eventos:
  - **Eventos públicos:** donde verá la lista completa de eventos públicos de los que podrá ver los detalles del evento o, en caso de ser el organizador, cancelarlo.
  - **Mis Eventos:** verá una lista de todos los eventos, públicos y privados, de los que es organizador.
- **Lista de Adiestradores**, realizar búsquedas por nombre y filtro y ver su perfil público.
- **Mi Perfil**, desde donde podrá:
  - **Comunicarse** con sus clientes, ya sea mediante mensajes Broadcast (mensajes a todos sus clientes), o mensajes personalizados a cada cliente.
  - **Crear eventos** Públicos y Privados.

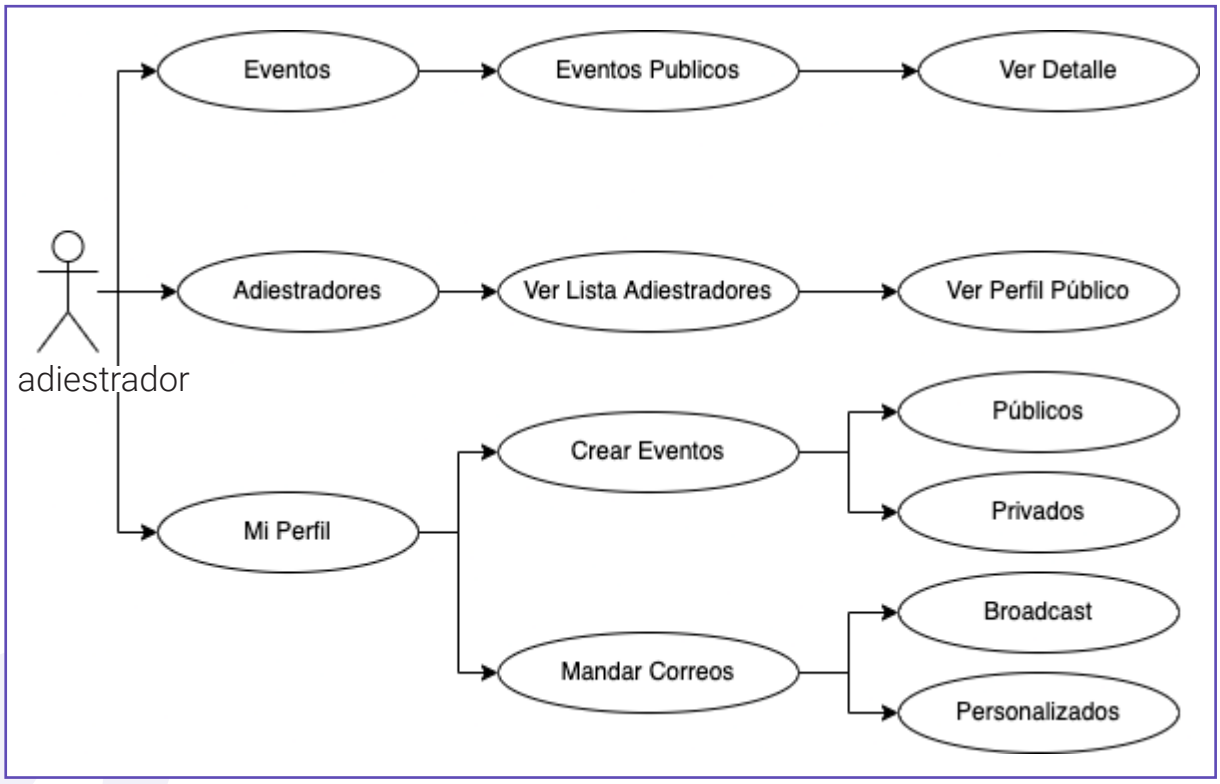

Fig. 3. Casos de uso, adiestrador logeado.

A partir de estos diagramas, se definió el modelo de datos a utilizar:

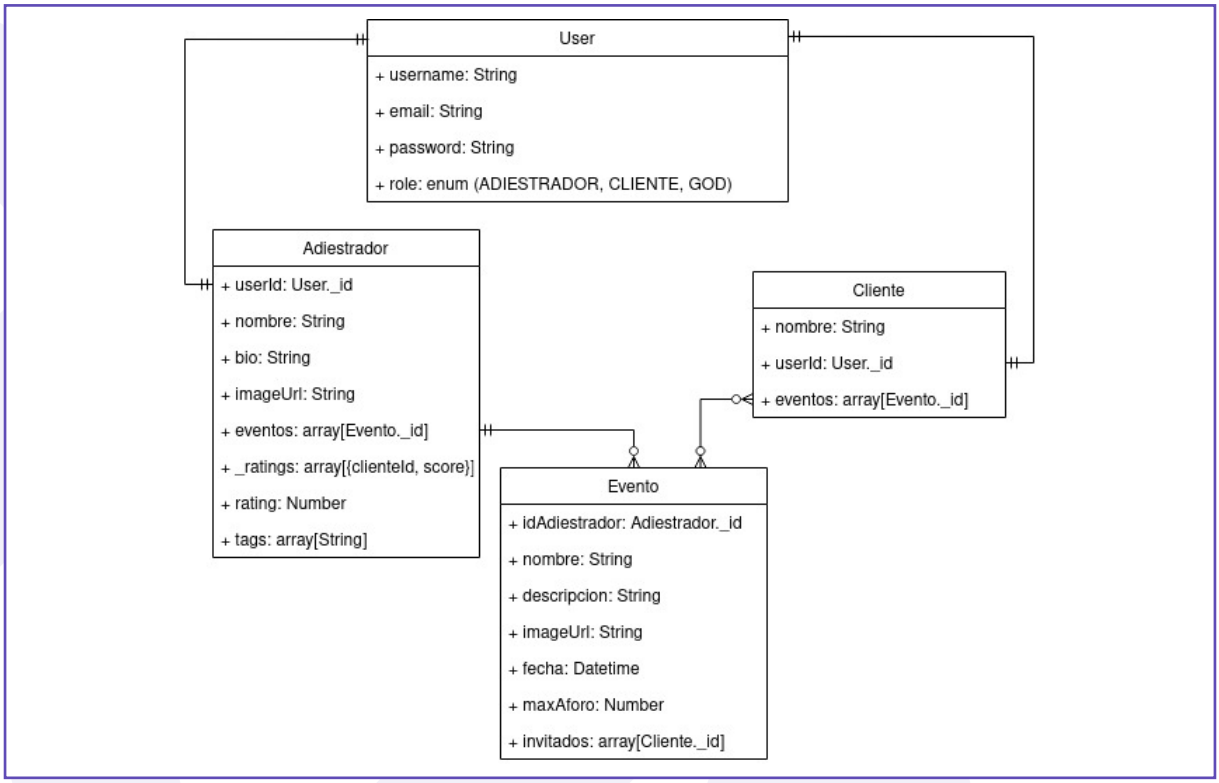

Fig. 4. Modelo de datos.

5.2. Modelo de datos

A. Esquemas

- **User:** Un usuario registrado en la plataforma. Puede ser un Adiestrador, un Cliente o un Administrador (GOD). Nota: este es el único recurso cuyo nombre está en inglés en lugar de en español, y se propaga en toda la implementación del back-end. La decisión de mantenerlo en inglés se tomó en un principio por si más adelante se incluían servicios externos que contasen con ciertas clases. Al final no ha sido necesario, pero se ha mantenido el nombre en inglés.
- **Adiestrador:** Un adiestrador registrado en la plataforma. Cuenta con un perfil público para que se le pueda contactar. Puede organizar eventos (públicos o privados).
- **Cliente:** Un cliente registrado en la plataforma. Su información es privada. Puede inscribirse en eventos públicos o privados (en caso de estar invitado).
- **Evento:** Cualquier sesión organizada por un adiestrador. Puede ser público (en cuyo caso se podrá inscribir cualquier cliente hasta cumplir el aforo máximo) o privado (en cuyo caso tan solo se podrán inscribir clientes que hayan sido invitados). La contratación de servicios de adiestramiento en la plataforma se modela en forma de eventos privados. Todo evento sin invitados se considera público, ya que no tiene sentido tener un evento privado sin ningún invitado.

B. Relaciones

Aunque este modelo no es de una BD relacional, sí cabe describir las principales relaciones que tienen lugar:

- **User-Cliente y User-Adiestrador:** Cada perfil de cliente y de adiestrador estará ligado a un usuario. A pesar de tratarse de relaciones 1:1, se separan así las acciones relacionadas con el usuario (login y registro) y aquellas relacionadas con el rol (cliente o adiestrador). Esta encapsulación ha resultado útil a la hora de implementar autenticación y autorización. También permite la implementación de nuevos roles en un futuro.
- **Adiestrador-Evento:** Cada adiestrador puede organizar cero, uno o varios eventos. Cada evento debe estar organizado por un adiestrador. No se contempla la posibilidad de eventos coorganizados por más de un adiestrador, u organizados por otro rol.

- **Cliente-Evento:** Cada cliente puede estar inscrito en cero, uno o varios eventos. A su vez, un evento puede tener cero, uno o varios clientes invitados. Cabe destacar que la lista de clientes invitados no tiene por qué coincidir con la lista de usuarios registrados. Son conceptos distintos.

5.3. Base de datos

Se ha utilizado MongoDB, con documentos para cada una de las entidades descritas. Mongo permite definir el esquema de cada tipo de documento, con distinto grado de flexibilidad. En este proyecto se ha escogido definir un esquema rígido para asegurar consistencia en el comportamiento de la aplicación.

Mongo permite ilustrar las relaciones de dos maneras distintas: documentos embebidos o referencias de ids. En este proyecto se ha escogido operar por referencias, para reducir la carga en queries. Se ha desplegado la BD en Mongo Atlas, para facilitar el desarrollo distribuido y el troubleshooting (al acceder todos los miembros del equipo a los mismos datos).

5.4. Back-end

Nota: el BE incluye funcionalidad más allá del alcance de este proyecto. En particular, se contempla un rol de administrador que no se ha implementado en el FE, ya que su objetivo principal ha sido agilizar el testeo durante un periodo específico de la implementación. Se ha decidido mantener esa funcionalidad para abrir la puerta a la continuación del proyecto más adelante.

5.4.1. Diseño

El BE expone una API REST, desacoplando así BE y FE. A la hora de diseñar la API, se aplican buenos principios de diseño:

- **Design-first:** se ha diseñado el comportamiento de la API (visualización mediante swagger), antes de comenzar su desarrollo. Aunque el diseño global de la API ha ido evolucionando, la sistemática de diseñar la nueva funcionalidad antes de desarrollarla se ha mantenido.
- **Estructura capas:** añade seguridad y robustez al BE.
- **Stateless:** el comportamiento del BE no depende del estado.
- **Códigos de respuesta representativos:** que cumplen el HTTP estándar según la W3.
- **Uso del verbo adecuado:** se escogen los verbos/métodos en cada ruta en función de la acción a realizar. Un ejemplo es el uso de PATCH en lugar de PUT para las actualizaciones,

ya que en ningún caso se sustituye el documento en BD por uno nuevo (como indica el método PUT), sino que se hacen updates parciales del documento existente.

- **Rutas expresivas que apuntan a recursos:** se intenta que los nombres de las rutas sean claros, y hagan referencia a recursos en vez de acciones (por ejemplo, "/clientes" en vez de "/mostrarClientes"). En ocasiones se ha comprometido este principio (por ejemplo, con "users/login"), si el resultado de seguirlo podría confundir al usuario.

El diseño de la API se ha documentado con swagger, y está disponible a través de la ruta "/swagger". En lugar de documentarlo en un archivo aparte, se ha integrado en el propio código, usando los módulos *swagger-jsdoc* y *swagger-ui-express*. El objetivo era simplificar la implementación y el mantenimiento de la documentación. Sin embargo, esta decisión no ha sido la más acertada: se pierde la ventaja de obtener un archivo independiente que se puede compartir con partners futuros, mientras que la comodidad de mantenimiento esperada no ha sido tal.

5.4.2. Estructura

El flujo de una request a través del BE es el siguiente:

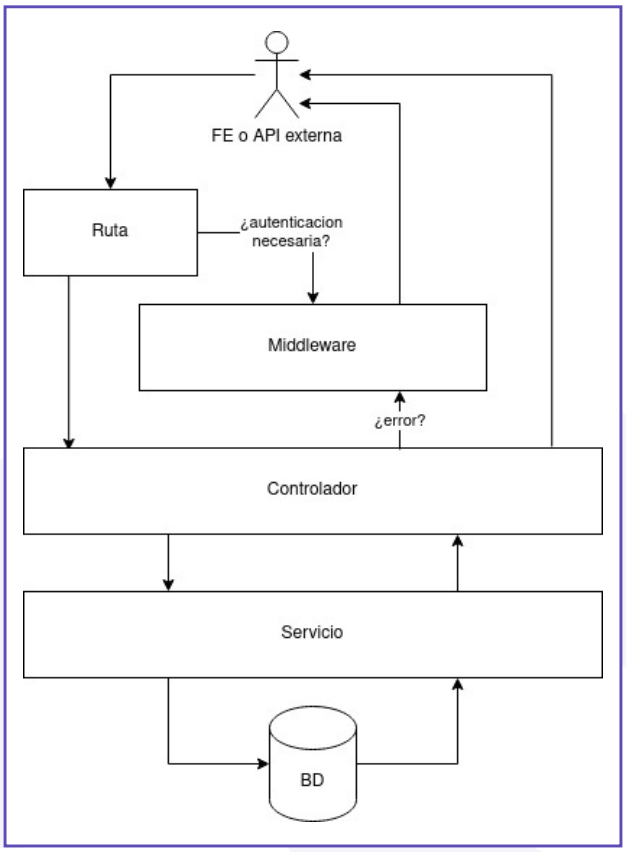

Fig. 5. Flujo de una request.

Para apoyar este flujo, la estructura del back cuenta con los siguientes módulos:

Routes

Contiene las rutas y la documentación de swagger correspondiente. Aparte de tener una función informativa, es en este módulo donde se redirigen (en caso de que sea necesario) las requests al middleware antes de enviarlas al controlador correspondiente.

La aplicación cuenta con 5 endpoints principales: uno para cada esquema de la BD y uno adicional para inicializar la BD con datos demo. Este último no estaría accesible en una versión de Producción, pero ha sido útil a la hora de testear funcionalidad, y se ha decidido mantenerlo, dado que el alcance de este proyecto es educacional.

Por claridad y comodidad, las subrutas con varias acciones se han separado de las rutas de endpoints principales en archivos independientes.

Controllers

La capa de controladores procesa las requests e interactúa con la capa de servicio para realizar la acción pertinente (recoger un recurso, actualizar información... etc.) antes de enviar la response al cliente. Hay un controlador por cada esquema de la BD.

Services

La capa de servicio se encarga de la lógica de negocio y de la interacción con la BD. Hay 5 servicios: uno por cada esquema en la BD y otro que gestiona el envío de emails por Sendgrid.

Models

Contiene los modelos de la BD. Aquí se definen los esquemas de Mongoose, y Mongoose se encarga de convertirlos a esquemas de MongoDB para que se implementen en la BD. El uso de Mongoose habilita además dos funcionalidades interesantes:

- **Funciones "pre":** funciones que se implementan antes de realizar una acción en BD. Por ejemplo, antes de guardar un adiestrador (ya sea un documento nuevo o una actualización a un documento existente), se calcula la media de sus ratings.

```
adiestradorSchema.pre('save', function () {
  if (this._ratings && this._ratings.length) {
    this.rating = calculateRating(this._ratings);
  }
});

// eslint-disable-next-line space-before-function-paren
function calculateRating(ratings) {
  if (!ratings || !ratings.length) return null;
  return (
    ratings.map((r) => r.score).reduce((sum, val) => sum + val) / ratings.length
  );
}
```

Fig. 6. Funciones pre Mongoose.

- Atributos virtuales: atributos que no se guardan en BD, sino que se añaden al convertir un documento de la BD a un documento JSON o un objeto. Por ejemplo, los atributos "terminado" y "privado" de un evento no se guardan en BD, tan solo se evalúan a la hora de devolver la información en base a la fecha y al atributo de invitados.

```
eventoSchema.virtual('terminado').get(function () {
  return this.fecha < Date.now();
});
eventoSchema.virtual('privado').get(function () {
  return this.invitados && this.invitados.length > 0;
});

eventoSchema.set('toObject', { virtuals: true });
eventoSchema.set('toJSON', { virtuals: true });
```

Fig. 7. Atributos virtuales de Mongoose.

Middleware

Contiene los componentes de middleware creados en este proyecto: autenticación, autorización y gestión de errores.

El gestor de errores es muy básico: comprueba si el error tiene el atributo httpStatus (en cuyo caso es un error controlado generado por nosotros) o si coincide con errores conocidos y esperados de Mongoose. En caso contrario, se enviará una response con status 500 y un error genérico.

El middleware de autenticación recibe una request y comprueba si contiene token de autenticación (ver sección de 5.4.3. Autenticación) y si el token es válido. Si es así, añadirá información del usuario a la request.

El middleware de autorización, además de comprobar la validez del token, comprueba que el usuario está relacionado con el id (de adiestrador o cliente) enviado como parámetro. También se ha implementado middleware que verifica si el usuario tiene el rol de GOD para las rutas reservadas para el administrador.

Util

Contiene utilidades: archivos de configuración, la clase propia de error (*HttpError*) y funciones auxiliares.

5.4.3. Autenticación

El sistema de autenticación es Bearer Authentication: para las rutas que requieren autenticación, la request debe incluir un token válido (que se obtiene mediante la ruta "/users/login", incluyendo en el body de la request un email y contraseña válidos).

Para la generación del token se ha utilizado el módulo *jsonwebtoken*. Al generar el token se incluye en él información del usuario y se le asigna una vida de 1 hora (después de la cual el token no se considerará válido).

Nota: por seguridad, las contraseñas de los usuarios se almacenan encriptadas utilizando el módulo *bcrypt*, y por lo tanto la validación a la hora de generar el token utiliza el mismo módulo para comparar el hasheado almacenado con el valor hasheado del body de la request.

5.4.4. Envío de emails

El contacto entre adiestradores y clientes se facilita por medio de emails. Para proteger la privacidad del usuario, los emails se envían desde al aplicación; la dirección de email que aparece en el remitente será el username del remitente seguido del dominio "*@dogginer.com*".

Los emails se envían por medio del servicio externo Sendgrid, configurado en la aplicación con el módulo *@sendgrid/mail*.

El servicio de emails también se utiliza para generar un email de reseteo de contraseña, que es el único método por el que se puede actualizar. Esta funcionalidad queda fuera del alcance del proyecto y por lo tanto no se ha implementado en el FE.

5.4.5. Despliegue

La API REST se ha desplegado en un servidor cloud utilizando AWS. El despliegue tan sólo requiere que se comprima el proyecto en un zip y se defina las variables de entorno necesarias.

5.5. Front-end

El front-end está basado en una estructura de componentes. Desde el archivo principal (*App.Vue*) se cargan los dos componentes comunes a todo el proyecto, el *Header* y el *Footer*, y entre ambos se renderizan a través del *router-view* el resto de vistas de las que dispone la web.

```
App.vue > {} "App.vue" > script
<template>
  <Header :userLogin="login" />
  <router-view />
  <Footer />
</template>

<script>
//Componentes
import Header from './components/partials/Header.vue';
```

Fig. 8. Componente principal front-end.

Estas vistas se enrutan desde el archivo *router*, donde se especifica el nombre de la vista, el path, a qué componente corresponde y si se transporta información por el path (props).

Estas vistas a su vez están agrupadas. Por un lado están *Vistas Principales* que son las que corresponden a las funcionalidades primarias de la web:

- Home.
- Eventos.
- Adiestradores.
- Registrarse.
- Login.

```
const routes = [
  // ------------- VIEWS1 -------------------
  {
    path: '/',
    name: 'home',
    component: Home,
  },
  {
    path: '/eventos',
    name: 'Eventos',
    component: Eventos,
  },
  {
    path: '/adiestradores',
    name: 'Adiestradores',
    component: Adiestradores,
  },
  {
    path: '/elige',
    name: 'elige',
    component: Registro,
  },
  {
    path: '/adiestradores/:id/miperfil',
    name: 'miPerfil',
    component: MiPerfil,
    props: true,
  },
  {
    path: '/login',
    name: 'login',
    component: Login,
  },
  // ------------- REGISTRO -------------------
  {
```

Fig. 9. Rutas front-end.

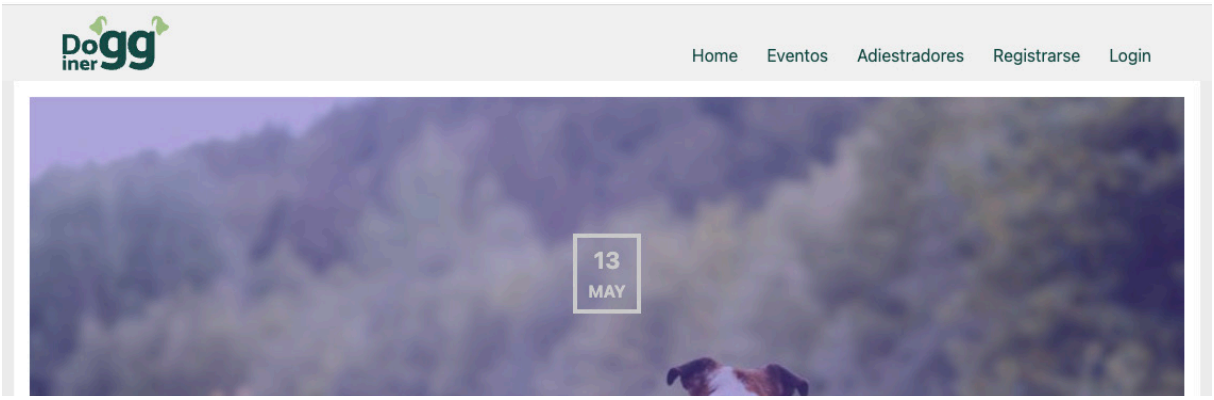

Fig. 10. Menu de navegación.

Después se dispone de una serie de *subvistas* que serían las que desarrollan alguna funcionalidad adherida a las vistas principales y por último están los componentes que forman dichas vistas y que aplican funcionalidades específicas.

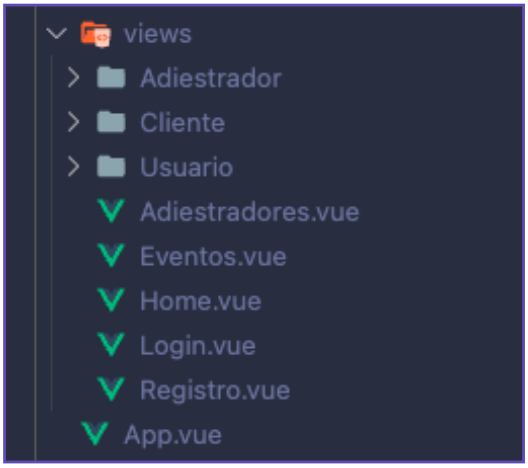

Fig. 11. Estructura de vistas y subvistas.

Por su parte, toda esta funcionalidad, como las llamadas al back-end, son desarrolladas en la carpeta *composables* por funciones javascript que agrupamos por carpetas según sea su destinatario.

Para explicar las funcionalidades del Front se empezará en la pantalla de Registro. Aquí se presentan dos enlaces a las dos opciones de que dispone el Usuario a la hora de registrarse, se puede registrar como Adiestrador o como Cliente.

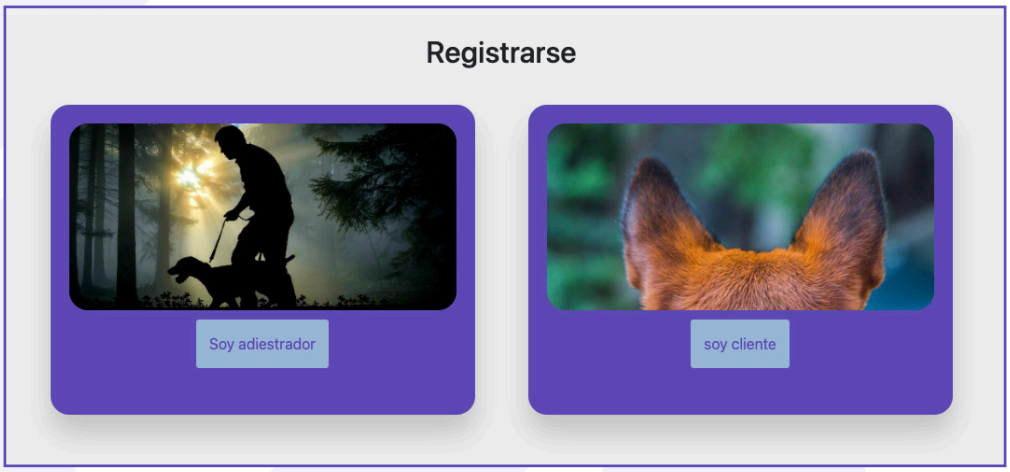

Fig. 12. Opciones de registro.

Fig. 13. Formularios de registro como cliente o adiestrador.

Estos enlaces conducen a dos subvistas distintas que desarrollaran distinta lógica y aunque ambas implementan el mismo componente, el Formulario de Registro, la vista del formulario será distinta para el usuario, presentando nuevos inputs para el adiestrador, para así, en el mismo paso en que se registra en la plataforma, nos facilita la información para su perfil público.

El Cliente tendrá que introducir su username, email y password. Mientras el Adiestrador para poder registrarse tendrá que añadir además una pequeña descripción/biografía de sí mismo, tendrá la opción de añadir una url a una imagen para su perfil y podrá añadir una serie de tags[1] para especificar sus especialidades. Como nota, indicar que solo se aceptan password seguras, de al menos 8 caracteres alfanuméricos (mínimo un número, una mayúscula y una minúscula). Todos los formularios son testeados con Expresiones Regulares.

El registro se realiza mediante una función javascript (createAdiestrador/createCliente) que con una llamada HTTP al endpoint facilitado por el back-end se pasa la información recogida en el formulario. En este caso, esta función contiene dos funciones ya que la bbdd está dividida en usuarios, adiestradores y clientes. En la primera llamada se inserta el email, el password del usuario y su rol (ya sea Cliente o Adiestrador). Esta llamada devuelve un id generado desde el

[1]Actualmente estos tags son una lista preestablecida, una futura mejora será la posibilidad de añadirlos dinámicamente.

back-end. Seguidamente se realiza una segunda llamada al back-end dependiendo de quién se esté registrando:

- Si se registra un cliente: En esta llamada se pasará un objeto cliente que se compone del id recibido en la llamada anterior, el nombre del cliente y se inicializa un array vacío de eventos.
- Si se registra un adiestrador: En esta llamada se pasará un objeto adiestrador que se compone del id recibido en la llamada anterior, el nombre del adiestrador, su biografía, la url de la imagen del perfil del adiestrador elegida y el array de tags seleccionados.

Se realizará esta segunda llamada al back-end donde se insertará el cliente/adiestrador en su colección.

Si el registro ha sido exitoso, la web redirigirá a la vista de Login, siempre sin recargar la página, solo el router-link, nuestro Header y nuestro Footer permanecen inalterados.

Si el registro no ha podido realizarse, se permanecerá en la misma vista y aparecerá un mensaje de error.

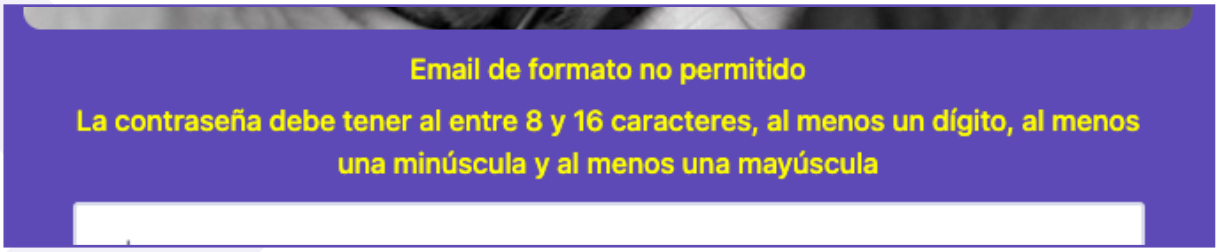

Fig. 14. Feedback intento de registro fallido.

Misma filosofía cumplirá la vista de Login (que implementa el componente Login), si el usuario se identifica correctamente le redirigirá a otra vista, mientras que si hay un error permanecerá en la misma vista y le mostrará un mensaje de error.

Para el login se ha usado el componente *store* de la librería *Vuex*, que es un *almacén* dónde guardar el estado de los componentes de la aplicación. Usando este componente se guarda en el *localStorage* del navegador los datos de validación (id, token, rol) para poder persistirlos y que no sea necesario tener que realizar llamadas al back-end en cada vista.

Fig. 15. Información almacenada en el navegador del cliente.

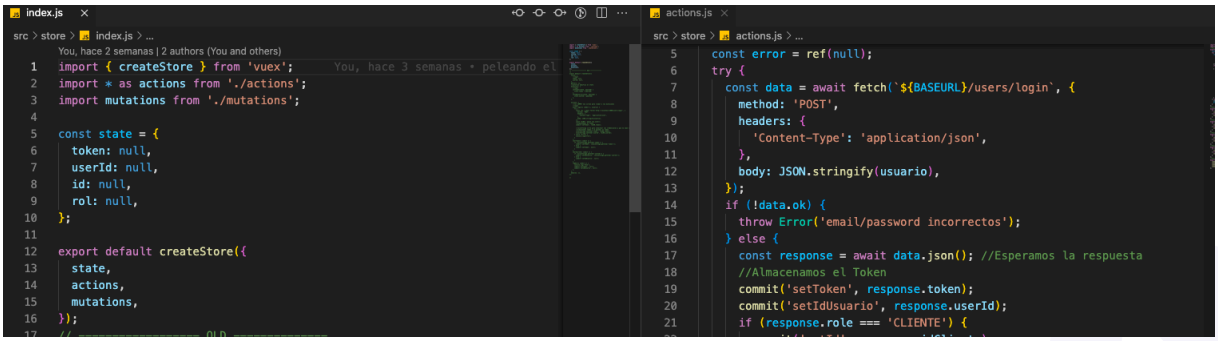

Fig. 16. Implementación parcial del componente store de Vuex.

En este punto hay que hacer mención a una de las características importantes de Vue.js, la programación reactiva, ya que al realizar un login correcto, el Header cambiará dejando de ofrecer las opciones de *Registro* y *Login,* y en su lugar mostrará la opción de *Logout.*

Fig. 17. Menú de navegación para usuario logeado.

Si un usuario se loguea exitosamente como Cliente le redirigirá a la vista principal de Home, mientras que si se loguea como Adiestrador le redirigirá a la vista privada del perfil correspondiente, una *subvista* en nuestro proyecto.

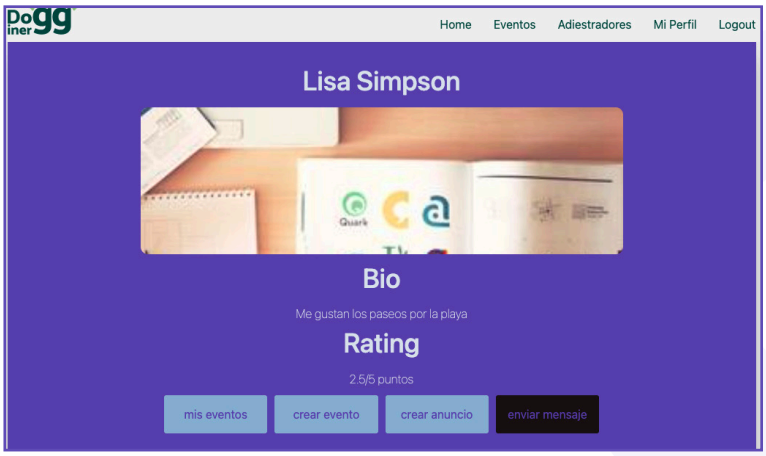

Fig. 18. Vista de perfil privado del adiestrador.

5.5.1. Vistas y Funcionalidades como Adiestrador

La vista privada del perfil de Adiestrador es similar a la vista del perfil público en muchos aspectos, sin embargo contiene 4 botones (*mis eventos, crear evento, crear anuncio, enviar mensaje*) que están ligados a 4 funciones javascript distintas. Estas funciones lo único que hacen es redirigir a una nueva *subvista* donde se aplica la funcionalidad de cada una de las opciones.

- Mis Eventos: Muestra una lista de los Eventos del Adiestrador con la opción de cancelar (solo es posible cancelar un evento si ya no hay clientes que tengan su asistencia confirmada a dicho evento).
- Crear Evento: Muestra un nuevo formulario donde se puede rellenar la información del evento que se quiere crear con la opción de añadir invitados[1]. Los invitados añadidos nunca podrán superar el aforo máximo establecido para el evento. Si no se añaden invitados, el evento pasa a ser público y a disposición de todos los clientes que quieran inscribirse hasta completar aforo.
- Crear Anuncio: Muestra otro formulario distinto donde se debe añadir un *Asunto* y el cuerpo del mensaje que se quiere mandar y este se mandará de forma broadcast a todos los clientes que tengan confirmada la asistencia a algún evento del adiestrador.
- Enviar Mensaje: Muestra el mismo formulario anterior (que es un componente) pero se le añade un input nuevo, el de añadir destinatario[2]. Este input será un select que desplegará una lista con todos los clientes a los que el adiestrador puede mandar un email (solo aquellos que tienen confirmada la asistencia a alguno de sus eventos) de los cuales tendrá que elegir uno al que se le quiere mandar un mensaje personalizado.

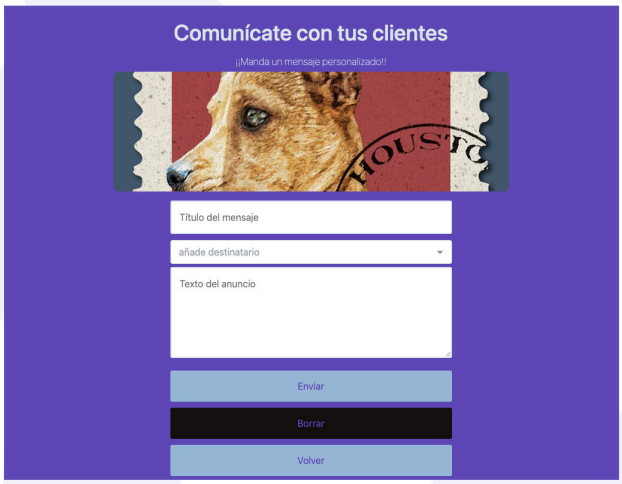

Fig. 19. Formulario de contacto adiestrador-cliente.

[1-2]Para esto se implementará un nuevo componente, un componente mantenido por la comunidad de desarrolladores Vue, en este caso es el *Multiselect*.

5.5.2. Vistas y Funcionalidades como Cliente

Al loguear como Cliente, redirigirá a la vista de Home. Desde aquí se puede navegar por las distintas pestañas de la web con toda la funcionalidad ya desbloqueada.

En Eventos habrá dos listas, una con todos los eventos públicos y aquellos eventos privados a los que el cliente ha sido invitado pero a los que aún no ha dado respuesta afirmativa, y otra con la lista de eventos a los que si que ha afirmado que asistirá. Si pasa el ratón por encima de los eventos tendrá la opción de confirmar/cancelar asistencia del evento. Si pulsamos sobre un evento se desplegará un Modal donde se podrán ver los detalles del Evento.

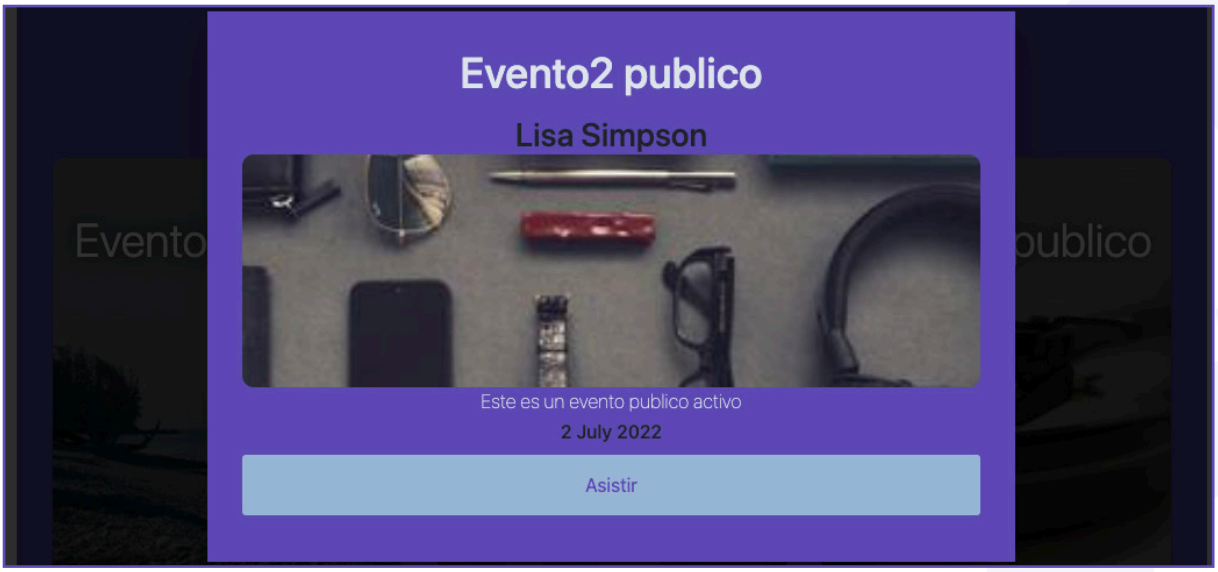

Fig. 20. Modal de detalles de evento.

Esta vista de Eventos está formada por dos componentes, la lista-eventos y el detalle-evento.

- Lista-Eventos: es un componente formado a su vez por dos componentes:
  - Filtro-Eventos: con dos botones que ejecutan las funciones que filtran la lista-eventos entre ver todos los eventos o solo los eventos del cliente.
  - Evento: Que es el componente donde se cargan los datos del evento y se despliega la lógica de la asistencia (en vista Cliente) y la gestión del mismo (en vista Adiestrador). Este Componente realizará un bucle para generar tantos componentes Evento como objetos haya en la bbdd.

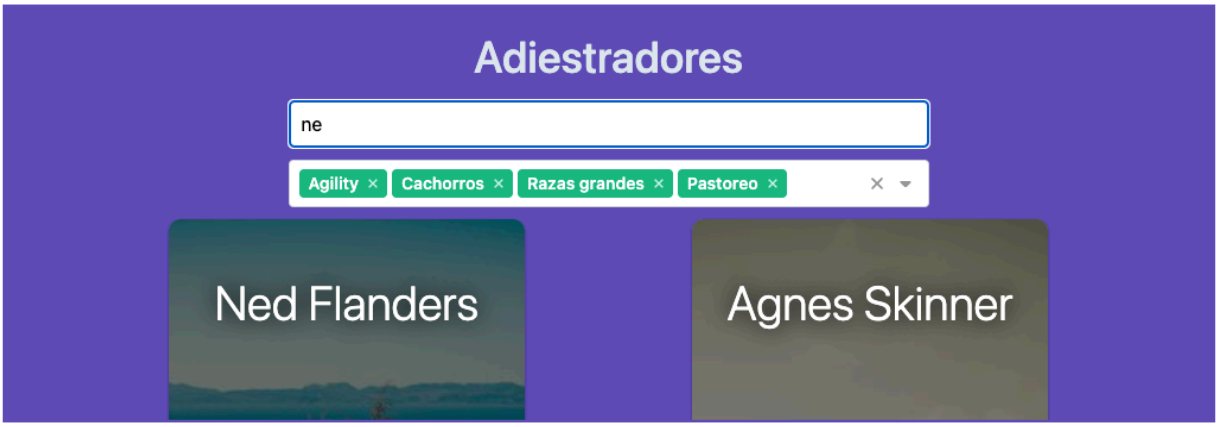

Fig. 21. Búsqueda y filtros de adiestrador.

En la vista de Adiestradores se verá la lista con todos los Adiestradores. Esta lista se podrá filtrar buscando al adiestrador por nombre o filtrando todos los adiestradores por tags.

Esta vista implementará un componente que es la propia Lista-Adiestradores, donde se aplicará la lógica del filtro por nombre. Para el filtro por tags por su parte se implementa el componente de Multiselect, ya que es una de las funcionalidades que facilita. También implementa el componente de Adiestrador realizando un bucle para generar tantos componentes Adiestrador como adiestradores haya en la bbdd.

En el componente Adiestrador se implementa la lógica de dos redireccionamientos. Por un lado hay un botón que redirecciona a la vista de los Eventos Públicos del Adiestrador, y el otro botón redirecciona a la vista del Perfil público del Adiestrador.

En esta vista (llamada DetalleAdiestrador) se implementa la lógica que devuelve, mediante una función javascript en composables, los datos del Adiestrador (nombre, imagen, bio, rating).

Las acciones que se pueden llevar a cabo en esta vista como Cliente son:

- **Contactar con el Adiestrador:** Este botón redirige a una vista que implementa el componente de formulario de email, desde dónde se puede mandar un mensaje al adiestrador.
- **Eventos:** Este botón redirige a la lista de Eventos del Adiestrador.
- **Valoración:** En caso de que el cliente esté registrador en algún evento del Adiestrador, tendrá acceso a un nuevo componente, el Formulario de Valoración. Este formulario consta de una puntuación mediante estrellas de un máximo de 5.

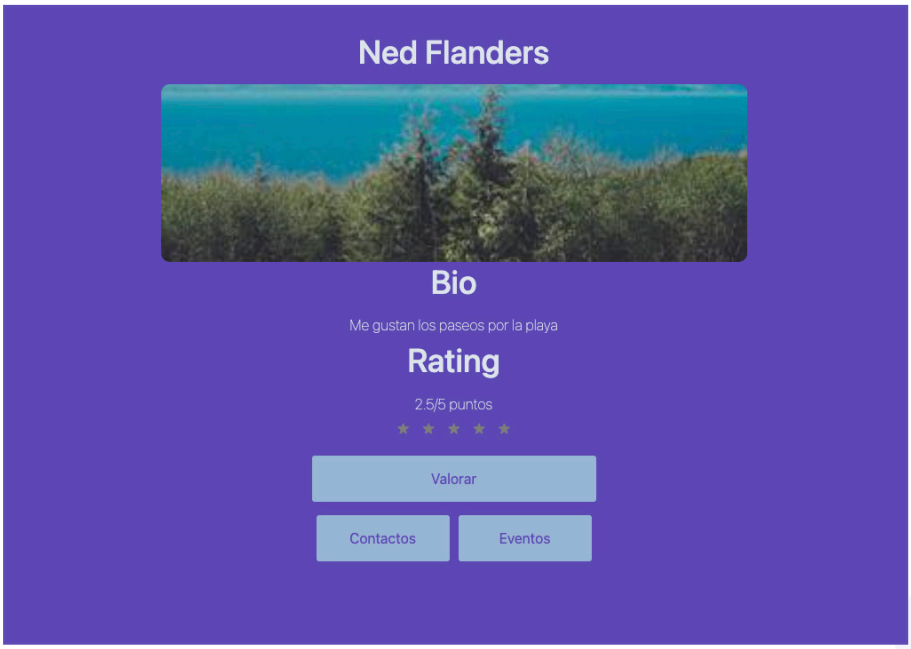

Fig. 22. Valoración de un adiestrador.

5.6. UI/UX

Para la parte de UI/UX se apostó por illustrator para la parte de creación de logotipado y creación de colores corporativos, también se usó figma para diseño de colocación y distribución de elementos de la página web, en los siguientes puntos se describen todas las partes de diseño:

- **Illustrator:** El primer paso es el diseño de tipografía que se va implementar para el logotipo, una tipografía moderna y con un cuerpo ancho para una buena visualización en cualquier proporción de tamaño que se use en un futuro. Después se relaciona el texto con la temática de la tipografía añadiendo unas orejas de perro en forma de comillas. Cuando se tiene un formato de logotipo en el que están de acuerdo los tres integrantes del grupo, se diseñan las diferentes variantes que puede tener el logo para diferentes entornos (header, footer… etc.). Ahora viene la parte de elegir los colores primarios y secundarios para web y logo. Se parte de un color verde, el cual tiene una relación de color con el entorno de un perro con la naturaleza y para el color secundario se busca un color que se distinga con el verde y llame más la atención al ojo humano con un tono morado moderno. Una vez que se tiene todo elegido, formato de logo y colores, viene la parte de sacar subcolores más claros y fondos para el footer. Se escogen dos versiones más claras de los colores principal y secundario y un negro que no es puro con un tono más claro.

Fig. 23. Diferentes modos de logo y colores.

- **Figma:** El proyecto se inicia con este software para hacer la cabecera, una posible página de introducción, todos los estilos de tipografía que se van a usar y los botones con sus hover. A la hora de seguir trabajando con el prototipado de figma, se calcula el tiempo que queda para terminar el proyecto y el equipo se da cuenta que no queda tiempo suficiente para hacer el prototipado y meter el código de css y html en Vue.js.

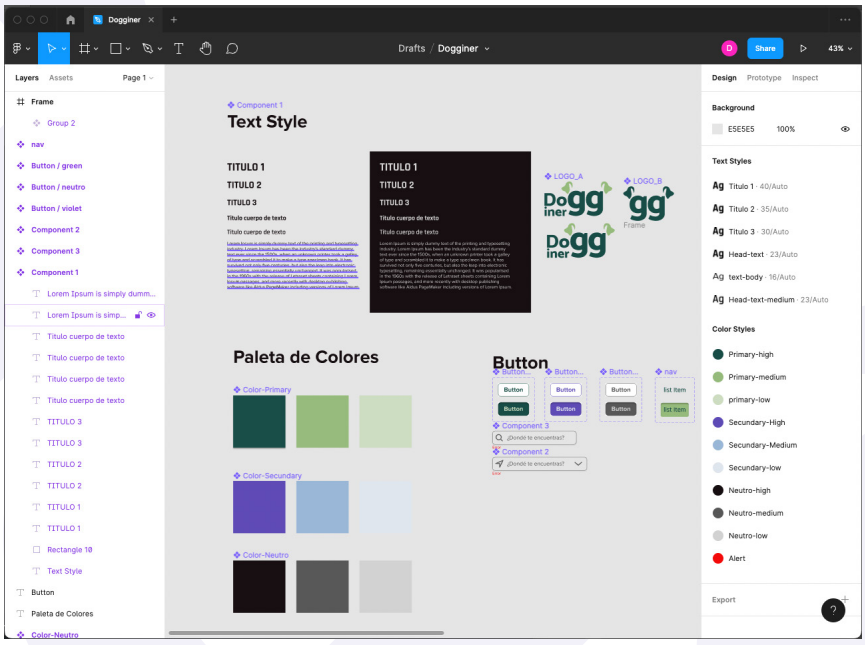

Fig. 24. Vista proyecto Figma.

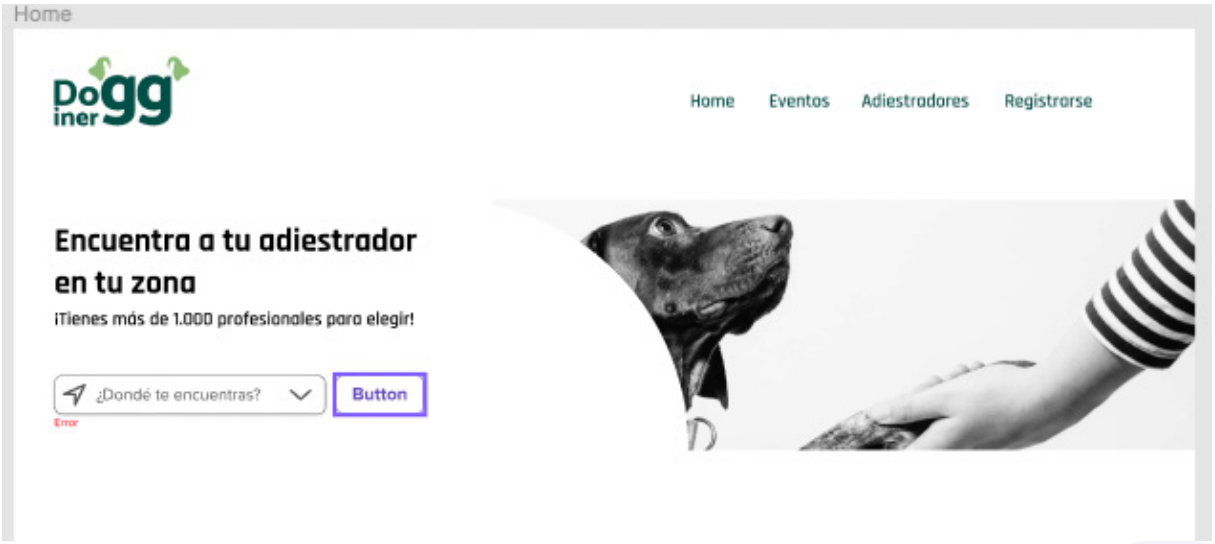

Fig. 24. Vista figma sin hover.

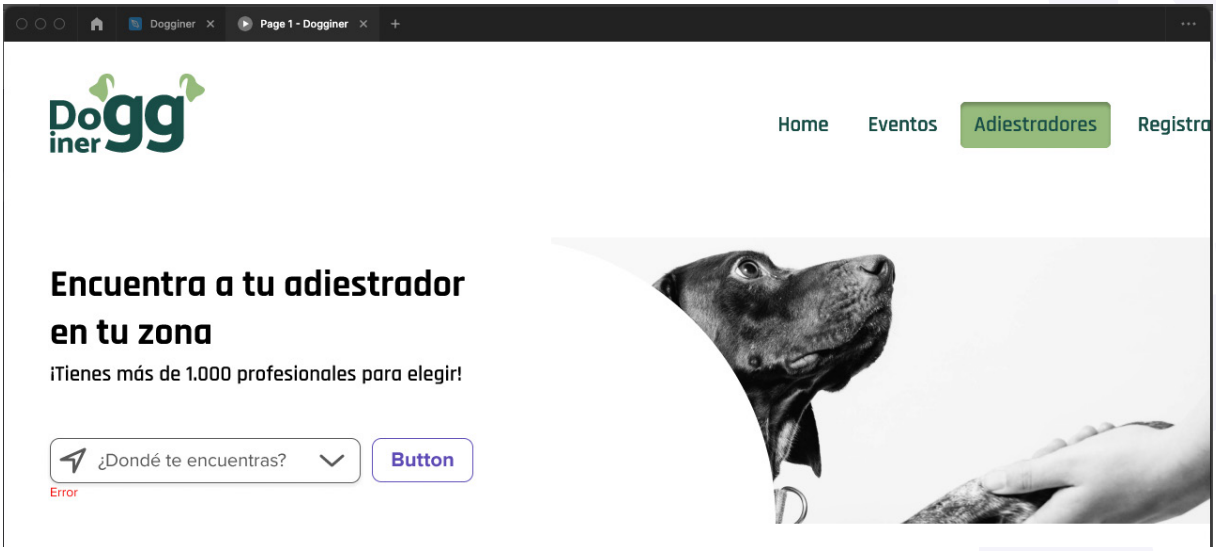

Fig. 25. Vista figma con hover.

- **Boostrap5:** Hay parte del proyecto que se implementa con boostrap5, inicialmente se intenta hacer el proyecto con boostrap un porcentaje medio y hacer cambios de los componentes de boostrap con SASS en la parte de colores y tipografías, pero conforme se va avanzando en el proyecto el equipo de front se da cuenta de las "zancadillas" que pone este framework con temas de estilos no visibles en el css. Se implementa contenedores y disposiciones de columnas para la estructura del proyecto con boostrap. Las cartas de eventos y adiestradores se implementa con boostrap al inicio del proyecto, pero a lo largo de este se genera varios errores de *padding* y *margin* originados por boostrap, para solventar este problema se decide crear las cartas sin boostrap.

A continuación se muestra un ejemplo de html predefinido con boostrap:

```
<template>
    <div class="container">
        <div class="row">
            <div class="col-12">
                <article class="blog-card">
                    <div class="blog-card__background">
```

Fig. 26. Clases de css con bootrap.

6. CONCLUSIONES

El proyecto ha cumplido los objetivos establecidos en el Anteproyecto. Si bien la funcionalidad implementada puede ser insuficiente para un entorno de Producción monetizable, sí es desplegable como primera iteración de un proyecto en desarrollo. Se cubren las necesidades identificadas (poner en contacto a adiestradores y clientes) en un entorno seguro que protege la privacidad de los usuarios. Se ofrece una página reactiva y fácil de navegar para el usuario. Además, la implementación encapsulada y desacoplada permite añadir futuras modificaciones fácilmente.

Dicho esto, hay muchos puntos de mejora de la funcionalidad ya implementada, así como posible expansión de los servicios ofrecidos. Estos son algunos ejemplos:

- Tags escogidas por los usuarios, con cierta validación para evitar duplicados.
- Posibilidad de envío de emails a múltiples destinatarios escogidos por el usuario.
- Migración de la documentación de swagger a un archivo separado.
- Mejoras en la gestión de errores, tanto en el front-end como en el back-end.
- Implementación de paginación para evitar lags cuando la BD tenga más contenido.
- Validación de asistencia a eventos.
- Refactorización del código.

Como proyecto de investigación para ahondar en los conocimientos adquiridos durante el curso, este trabajo ha sido muy fructífero. Unifica las distintas temáticas, vistas hasta ahora de manera independiente, y ha permitido tanto profundizar en los temas vistos como explorar áreas nuevas:

- Se ha profundizado significativamente en el manejo de javascript, HTML y CSS.
- Se ha profundizado la base conceptual y práctica del uso de APIs REST.
- Se ha explorado por primera vez Vue.js, Node.js, Express.js y MongoDB[1].
- Se ha experimentado con el uso de swagger, y servicios externos como SendGrid y Amazon Cloud.

[1]Entendemos que Node.js, Express.js y MongoDB se han visto parcialmente en el instituto en el tercer trimestre. Sin embargo, consideramos que se han explorado por primera vez durante el proyecto, ya que la materia se ha impartido después de la implementación del back-end.

7. BIBLIOGRAFÍA

Diseño de APIs REST

Fielding, et al (2016), "Status Codes Definitions", w3 https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Janet Wagner (2021), "API-First, API Design-First or Code-First: which should you choose", Stoplight, https://blog.stoplight.io/api-first-api-design-first-or-code-first-which-should-you-choose

Keshav Vasudevan (2017), "Design First or Code First: What's the Best Approach to API Development?", Swagger https://swagger.io/blog/api-design/design-first-or-code-first-api-development/

Lokesh Gupta (2021), "HTTP Methods", Restful Api https://restfulapi.net/http-methods/

Lokesh Gupta (2021), "REST Resource Naming Guide", Restful Api https://restfulapi.net/resource-naming/.

Lokesh Gupta (2022), "What is REST", Restful Api : https://restfulapi.net/

Turoriales

Programming with Mosh (2018) "Node.js Tutorial for Beginners" https://youtu.be/TlB_eWDSMt4

Traversy media (2019), "Nodejs Crash Course" https://youtu.be/fBNz5xF-Kx4

Traversy media (2021), "Vue JS Crash Course" https://youtu.be/qZXt1Aom3Cs

Web Dev Simplified (2019), "What is JWT and Why Should you Use JWT" https://youtu.be/7Q17ubqLfaM

Documentación

Boostrap (s.f) "Documentation"https://getbootstrap.com/docs/5.0/getting-started/introduction/

Express (s.f), "4.x API" https://expressjs.com/en/4x/api.html

Figma Design (s.f) https://help.figma.com/hc/en-us/categories/360002042553-Figma-design

MongoDB (s.f), "MongoDB Atlas", https://www.mongodb.com/docs/atlas/

MongoDB (s.f), "MongoDB Manual", https://www.mongodb.com/docs/manual/

Mongoose (s.f) "Api Docs" https://mongoosejs.com/docs/api.html

NodeJs (s.f), "Node.js v16.15.0 Documentation" https://nodejs.org/dist/latest-v16.x/docs/api/

Sass (s.f) "Documentation" https://sass-lang.com/documentation

Swagger (s.f), "OpenApi Guide" https://swagger.io/docs/specification/about/

Vue.js (s.f) "Guide", https://vuejs.org/guide/introduction.html

Vue.js (2020) "Vue 3 multiselect component with multiselect and tagging options" https://vuejsexamples.com/vue-3-multiselect-component-with-multiselect-and-tagging-options/

Vuex (s.f) "What is Vuex" https://vuex.vuejs.org/

ANEXOS

Anexo I - Código fuente

Código del front-end: https://github.com/PabloSato/FrontDoggin

Código del back-end: https://github.com/Hannah-bannanah/dogginer-be

Anexo II - Instrucciones de ejecución del FE

Instalación de Vue :`npm install -g @vue/cli`

Comando de ejecución en servidor local: npm run serve

Anexo III - Instrucciones de ejecución del BE

Nota: el BE está desplegado y accesible en:

http://dogginer-api.eu-west-3.elasticbeanstalk.com/ (añadir endpoint deseado)

El swagger del BE está accesible en

http://dogginer-api.eu-west-3.elasticbeanstalk.com/swagger/

Si se desea ejecutar el proyecto en local, se han de seguir las instrucciones del README.md:

1 Clonar este repositorio.

2 Copiar el archivo nodemon.plantilla.json:

2.1 Reemplazar '`<user>`' y '`<password>`' por los credenciales de acceso a la bbdd.

2.2 Reemplazar '`<passphrase>`' por la passphrase de seguridad de JWT.

2.3 Reemplazar '`<apikey>`' por la api key de sendgrid.

2.4 Reemplazar.

2.5 Renombrar archivo a 'nodemon.json'.

3 En la terminal, ejecutar npm install.

4 En la terminal, ejecutar npm run dev.

Los datos necesarios para el punto 2 son:

User: `hannahdog`

Password: `D0gg1n3r`

Passphrase JWT: `deviation_account_overheat_steadily_derived_overwrite_feminize_wispy_routing_humming_juggle_obsession`

Api key SendGrid:

`SG.hg0EtUfvQmOLBpotrux0Og.nlropB_piI9Cq0GTpfLLgDe_TBtYkdI-fW-VUe7SjuW8`

Url del servidor: `http://localhost:3000.`

Anexo IV - Instrucciones de inicialización de BD

El proyecto se entrega con una serie de datos demo cargados en la BD. Si se desea reinicializar la BD a estos datos tras hacer pruebas, se puede utilizar el endpoint "/inicializar". Es necesario estar autenticado y con privilegios de admin para poder acceder a esa ruta.

Por lo tanto, primero hay que enviar una request al endpoint "/users/login" con los siguientes datos en el body:

Email: `admin@dogginer.com`

Password: `God12345`

La response a esta request incluirá un token, que habrá que añadir como autenticación en el swagger:

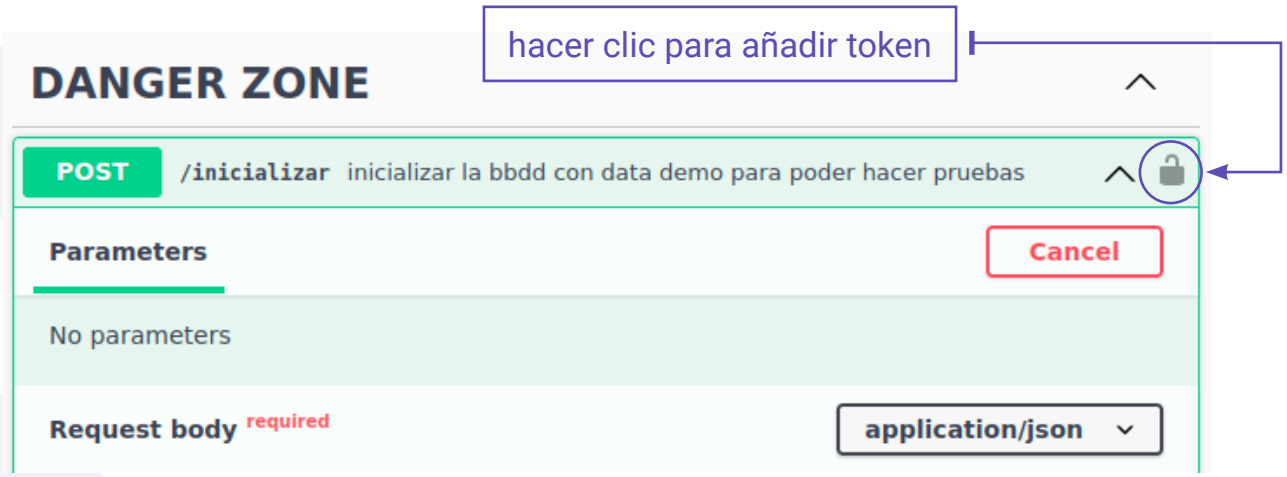

Fig. 27. Abrir menú de autorización en Swagger.

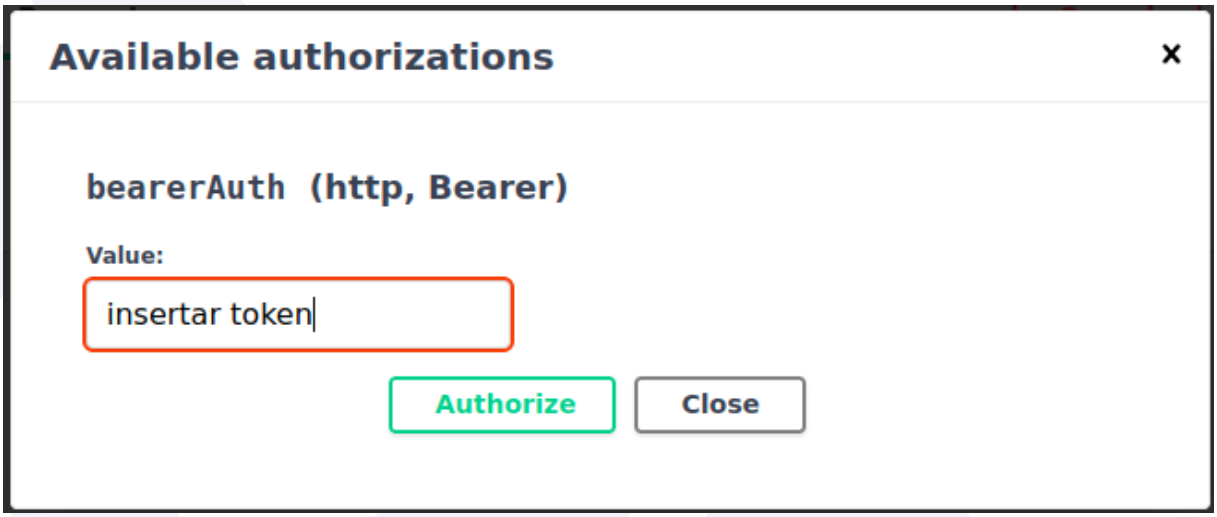

Fig. 28. Insertar token en Swagger.

Una vez añadido el token, el body de la request debe contener los siguientes campos:

inicializar: `true`

seguro: `true`

dbname: “`dogginer`”

La BD se reseteará con los datos demo. Se perderán los usuarios creados manualmente, así como registros y otras acciones sobre usuarios preexistentes. También cambiarán todos los ids.