DOCUMENTO

EVENTHUB-EVENTS / EVENTHUB-ASSISTANTS

URLs de los repositorios:

https://github.com/abernalmar/EventHub.git

https://github.com/abernalmar/events-service.git

https://github.com/jorsilman/EventHub-Frontend.git

https://github.com/jorsilman/EventHub-Assitants.git

Miembros del grupo: Ángela Bernal, Francisco Andrés Caro y Jorge Sillero

Nivel de acabado: Nivel Hasta ***7*** Puntos. Como microservicios avanzados, hemos llevado a cabo la integración con el microservicio de asistente, incorporando un servicio externo (api.random.org) que genera un código aleatorio para cada nuevo asistente creado. Además, hemos implementado un frontend unificado que se alinea con los demás microservicios de la aplicación. La documentación detallada de las APIs de ambos microservicios se encuentra disponible en Swagger. Asimismo, hemos establecido una API Gateway utilizando Docker para optimizar la gestión y el enrutamiento de las peticiones.

Contenido

Descripción de la aplicación

La aplicación consiste en una plataforma de gestión de eventos, en la cual los usuarios podrán crear eventos, añadir asistentes a dichos eventos y tener comunicaciones entre los distintos usuarios.

Descomposición en microservicios

La aplicación se ha dividido en los siguientes cuatro microservicios:

- **Users:** este microservicio se encarga de todas las operaciones relacionadas con los usuarios, como pueden ser, el registro, el login, el listado, la visualización del perfil, etc.

- **Communications:** este microservicio se encarga de las distintas comunicaciones posibles de la aplicación, estas pueden ser, comunicaciones entre los distintos usuarios, o envío de correos con las invitaciones a los asistentes.

- **Events:** este microservicio se encarga de todo lo relacionado con los eventos, es decir, la creación de estos, la edición, el listado o la eliminación. Como se explicará más adelante en el documento, este microservicio hace uso del microservicio de Users y el de Assistants.

- **Assistants:** este microservicio se encarga de todo lo relacionado con los asistentes, es decir, de la creación, edición, listado y eliminación. Como se explicará más adelante en el documento, este microservicio hace uso del microservicio de Events, Communications y de un servicio externo, api.random.org.

En nuestro caso, al ser 3 componentes los que formamos el grupo, nos hemos encargado del desarrollo de los microservicios de Assistants y Events.

Customer Agreement

CA_v7.docx

Análisis de la capacidad

Análisis de la capacidad.pdf

TCO

TCO.docx

Descripción API REST

EventHub-Assistants

- GET:
  - '/': Obtiene todos los asistentes de la base de datos.
    - Respuesta Exitosa (200 OK):
      - Cuerpo: Lista de objetos de asistentes.
    - Respuesta Fallida (500 Internal Server Error):
      - Cuerpo: Problema con la base de datos.
  - '/:id': Obtiene un asistente dado el id.
    - Parámetros de Ruta: id (string) - ID del asistente.
    - Respuesta Exitosa (200 OK):
      - Cuerpo: Objeto del asistente.
    - Respuesta Fallida (404 Not Found):
      - Cuerpo: Asistente no encontrado.
    - Respuesta Fallida (500 Internal Server Error):
      - Cuerpo: Problema con la base de datos.
  - '/event/:eventId': Obtiene los asistentes de un evento dado el eventId.
    - Parámetros de Ruta: eventId (string) - ID del evento.
    - Respuesta Exitosa (200 OK):
      - Cuerpo: Lista de objetos de asistentes.
    - Respuesta Fallida (500 Internal Server Error):
      - Cuerpo: Problema con la base de datos.
- POST:
  - '/': Crea un nuevo asistente.
    - Autenticación: Bearer Token
    - Parámetros de Solicitud:
      - name (string): Nombre del asistente.
      - surname (string): Apellido del asistente.
      - email (string): Correo electrónico del asistente.
      - eventId (string): ID del evento al que se registra el asistente.
      - username (string): Nombre de usuario del asistente.
    - Respuesta Exitosa (201 Created):
      - Cuerpo: Ninguno.
    - Respuesta Fallida (400 Bad Request):
      - Cuerpo: Error de validación.
    - Respuesta Fallida (500 Internal Server Error):
      - Cuerpo: Problema con la base de datos o al enviar el correo electrónico.
- PUT:
  - '/:id': Actualiza un asistente dado su id.
  - Autenticación: Bearer Token
  - Parámetros de Ruta: id (string) - ID del asistente.
  - Parámetros de Solicitud: Datos actualizados del asistente.
  - Respuesta Exitosa (204 No Content):
    - Cuerpo: Ninguno.
  - Respuesta Fallida (404 Not Found):
    - Cuerpo: Asistente no encontrado.

    - Respuesta Fallida (500 Internal Server Error):
        - Cuerpo: Problema con la base de datos.
- DELETE:
    - '/:id': Elimina un asistente dado su id.
    - Autenticación: Bearer Token
    - Parámetros de Ruta: id (string) - ID del asistente.
    - Respuesta Exitosa (204 No Content):
        - Cuerpo: Ninguno.
    - Respuesta Fallida (404 Not Found):
        - Cuerpo: Asistente no encontrado.
    - Respuesta Fallida (500 Internal Server Error):
        - Cuerpo: Problema con la base de datos.

API Documentada en Swagger:

https://app.swaggerhub.com/apis/JORGESILLEROUNI/Eventhub-Assistants/1.0.0

EventHub-Events

API de Eventos (EventHub-Events):

- **GET:**
    - **/**: Obtiene todos los eventos de la base de datos.
        - Respuesta Exitosa (200 OK):
            - Cuerpo: Lista de objetos de eventos.
        - Respuesta Fallida (500 Internal Server Error):
            - Cuerpo: Problema con la base de datos.
    - **/:name?**: Obtiene un evento dado el nombre si se especifica, de lo contrario, devuelve todos los eventos.
        - Parámetros de Ruta: name (string) - Nombre del evento (opcional).
        - Respuesta Exitosa (200 OK):
            - Cuerpo: Objeto del evento o lista de objetos de eventos.
    - **/id/:id**: Obtiene un evento dado su id.
        - Parámetros de Ruta: id (string) - ID del evento.
        - Respuesta Exitosa (200 OK):
            - Cuerpo: Objeto del evento.
        - Respuesta Fallida (404 Not Found):
            - Cuerpo: Evento no encontrado.

        - Respuesta Fallida (500 Internal Server Error):
            - Cuerpo: Problema con la base de datos.

- **POST:**
    - **/**: Crea un nuevo evento con los siguientes parámetros:
        - name
        - place
        - date
        - description
        - category
        - Autenticación: Bearer Token
        - Respuesta Exitosa (201 Created):
            - Cuerpo: Ninguno.
        - Respuesta Fallida (400 Bad Request):
            - Cuerpo: Error de validación.
        - Respuesta Fallida (500 Internal Server Error):
            - Cuerpo: Problema con la base de datos o al enviar el correo electrónico.
- **PUT:**
    - **/:name**: Actualiza un evento dado su nombre.
        - Autenticación: Bearer Token
        - Parámetros de Ruta: name (string) - Nombre del evento.
        - Respuesta Exitosa (200 OK):
            - Cuerpo: Objeto del evento actualizado.
        - Respuesta Fallida (404 Not Found):
            - Cuerpo: Evento no encontrado.
        - Respuesta Fallida (500 Internal Server Error):
            - Cuerpo: Problema con la base de datos.
- **DELETE:**
    - **/:name**: Elimina un evento dado su nombre.
        - Autenticación: Bearer Token
        - Parámetros de Ruta: name (string) - Nombre del evento.

- Respuesta Exitosa (204 No Content):
  - Cuerpo: Ninguno.
- Respuesta Fallida (404 Not Found):
  - Cuerpo: Evento no encontrado.
- Respuesta Fallida (500 Internal Server Error):
  - Cuerpo: Problema con la base de datos.

API Documentada en Swagger:

https://app.swaggerhub.com/apis/JORGESILLEROUNI/Eventhub-Events/1.0.0

Documento de requisitos

EventHub-Assistants

Listar asistentes de un evento

Como usuario del sistema, quiero poder listar los asistentes del evento para poder ver las personas que están invitadas.

1. En el backend se ha creado una función que permite, mediante llamadas a la API, obtener los asistentes dado el id de un evento:

```
/*GET assistants by eventId FUNCIONA*/
router.get('/event/:eventId', async function(req, res, next) {
  try {
    const eventId = req.params.eventId;

    // Buscar todos los asistentes con el eventId proporcionado
    const result = await Assistant.find({ eventId });

    if (result.length > 0) {
      // Si se encuentran asistentes, enviar la respuesta al cliente
      res.send(result);
    } else {
      // Si no se encuentran asistentes, enviar un código de estado 404
      res.sendStatus(404);
    }
  } catch (error) {
    // Manejar errores de manera adecuada
    console.error('Error al obtener asistentes por evento:', error);
    res.sendStatus(500);
  }
});
```

2. Por otra parte, en el frontend, se ha creado un componente, ListAssistants, que, mediante la llamada a la API, obtiene todos los asistentes de un evento y los muestra:

```
static async getAssistantsByEventId(eventId) {
    const headers = this.requestHeaders();
    const request = new Request(API_BASE_URL + `/assistants/event/${eventId}`, {
      method: 'GET',
      headers: headers
    });

    const response = await fetch(request);

    if (!response.ok) {
      throw new Error("Response not ok"+response.status);
    }

    return response.json();
}
```

```
const ListAssistants = () => {
    const [assistants, setAssistants] = useState([]);
    const [event, setEvent] = useState({});

    const eventId = useParams().eventId;

    console.log('eventId:', eventId);

    useEffect(() => {
        async function fetchAssistants() {
            try {
                const assistants = await AssistantAPI.getAssistantsByEventId(eventId);
                setAssistants(assistants);
            } catch (error) {
                console.error('Error fetching assistants:', error);
            }
        }
```

Añadir un asistente a un evento

Como usuario del sistema, quiero poder añadir un asistente a un evento para poder enviarle una invitación.

1. En el backend se ha creado una función que permite realizar una petición POST a la API. El funcionamiento es el siguiente:
    a. Se ha creado una función getRandomCode(), que hace uso de un Servicio Externo, en concreto api.random.org. Esta función realiza una petición a la API y obtiene un código aleatorio.
    b. Dentro de la función que realiza el POST, primero se obtiene el código haciendo uso de la función descrita anteriormente.
    c. Luego, se obtiene el evento haciendo uso del microservicio de Evenhub-Events.
    d. Después, se envía un correo electrónico al asistente con la invitación, haciendo uso del microservicio de Eventhub-Communications.
    e. Y por último, se guarda el asistente en la base de datos.

```
async function getRandomCode() {
  const url = 'https://api.random.org/json-rpc/4/invoke';

  const options = {
    method: 'POST',
    url,
    data: {
      jsonrpc: '2.0',
      method: 'generateStrings',
      params: {
        apiKey: '52ebd838-f349-49d6-98bf-5d28623752d3',
        n: 1,
        length: 5,
        characters: 'abcdefghijklmnopqrstuvwxyz0123456789',
      },
      id: 1,
    },
  };

  // Realiza la petición HTTP usando axios
  return axios(options);
}
```

```
router.post('/',
  passport.authenticate('bearer', { session: false }),
  async function(req, res, next) {
  const { name, surname, email, eventId, username } = req.body;

  // Utilizar la función getRandomCode para obtener un código aleatorio si
  let code;
  try {
    const response = await getRandomCode();
    code = response.data.result.random.data[0];
    console.log('Codigo aleatorio API:', code);
  } catch (error) {
    console.error('Error al obtener el código aleatorio:', error);
    code = Math.random().toString(36).substring(2, 7);
    console.log('Codigo aleatorio:', code);
  }
```

```
try {
  // Crear una instancia del modelo Assistant con los datos proporcionados
  const assistant = new Assistant({
    name,
    surname,
    email,
    eventId,
    username,
    code
  });

  let eventName, eventPlace, eventDate, eventTime;

  try {
      //Obtener el evento de la api en la url localhost:4000/api/v2/events/id/:id
      const event = await axios.get(`http://localhost:4000/api/v1/events/id/${eventId}`);
      //Obtener el nombre del evento
      eventName = event.data.name;
      //Obtener el lugar del evento
      eventPlace = event.data.place;
      //Obtener la fecha del evento
      eventDate = event.data.date;
      //Obtener la hora del evento
      eventTime = event.data.time;
  } catch (error) {
      console.error('Error al obtener el evento:', error);
  }
```

```
try {
    //Enviar una peticion POST a localhost:5000/api/v2/emails con los campos to, subject,
    await axios.post('http://localhost:5000/api/v2/emails', {
        to: email,
        subject: 'Registro en evento',
        text: `Hola ${name} ${surname}, te has registrado en el evento ${eventName} que se
        html: `<p>Hola ${name} ${surname}, te has registrado en el evento ${eventName} que
    });
} catch (error) {
    console.error('Error al enviar el email:', error);
}

// Guardar la instancia del modelo en la base de datos
await assistant.save();
// Enviar respuesta exitosa al cliente
res.sendStatus(201);
```

2. En el frontend, se ha creado un componente, CreateAssistant que muestra un formulario para añadir un asistente a un evento:

```
static async createAssistant(assistant) {
  try {
      const headers = this.requestHeaders();
      headers['Authorization'] = 'Bearer 37f7b2a4-2fdc-4e17-a72b-6570187d3cb6';

      const response = await axios.post(API_BASE_URL + '/assistants', assistant, { headers });
      return response.data;
  } catch (error) {
      console.error('Error creating assistant:', error);
      throw error;
  }
}
```

```
function CreateAssistant() {
    const [name, setName] = useState("");
    const [surname, setSurname] = useState("");
    const [username, setUsername] = useState("");
    const [email, setEmail] = useState("");

    const eventId = useParams().eventId;

    function onClick() {
        const assistant = {
        name: name,
        surname: surname,
        username: username,
        eventId: eventId,
        email: email,
        };

        try {
        const response = AssistantAPI.createAssistant(assistant);
```

Editar un asistente

Como usuario del sistema, quiero poder editar un asistente para poder modificar algunos de sus datos.

1. En el backend, se ha creado la función para permitir el método PUT de la API:

```
/* PUT assistant by Id FUNCIONA*/
router.put('/:id',
  passport.authenticate('bearer', { session: false }),
  async function(req, res, next) {
  const id = req.params.id;
  const updatedAssistant = req.body;

  try {
    // Buscar el asistente en la base de datos por su ID
    const result = await Assistant.findById(id);

    // Verificar si se encontró el asistente
    if (result) {
      // Actualizar los datos del asistente con los proporcionados en la solicitud
      result.name = updatedAssistant.name;
      result.surname = updatedAssistant.surname;
      result.email = updatedAssistant.email;
      result.eventId = updatedAssistant.eventId;
      result.username = updatedAssistant.username;

      // Guardar los cambios en la base de datos
      await result.save();

      // Enviar un código de estado 204 (Sin Contenido) como respuesta al cliente
      res.sendStatus(204);
```

2. En el frontend, se ha creado un componente, EditAssistant, para realizar la llamada a la API y mostrar un formulario para modificar los campos necesarios:

```
static async updateAssistant(assistant) {
  try {
      const headers = this.requestHeaders();
      headers['Authorization'] = 'Bearer 37f7b2a4-2fdc-4e17-a72b-6570187d3cb6';
      const response = await axios.put(API_BASE_URL + `/assistants/${assistant._id}`, assistant, { headers });
      return response.data;
  } catch (error) {
      console.error('Error updating assistant:', error);
      throw error;
  }
}
```

```
import { useEffect, useState } from "react";
import DatePicker from "react-datepicker";
import "react-datepicker/dist/react-datepicker.css";
import AssistantAPI from "./AssistantsAPI";
import Menu from "../Components/NavBar";
import { redirect, useParams } from 'react-router-dom';

function UpdateAssistant(){
    const id = useParams().id;
    const [name, setName] = useState("");
    const [surname, setSurname] = useState("");
    const [username, setUsername] = useState("");
    const [email, setEmail] = useState("");
    const [eventId, setEventId] = useState("");

    useEffect(() => {
        async function fetchAssistant() {
            try {
                const assistant = await AssistantAPI.getAssistant(id);
                setName(assistant.name);
                setSurname(assistant.surname);
                setUsername(assistant.username);
                setEmail(assistant.email);
                setEventId(assistant.eventId);
```

Eliminar un asistente

Como usuario del sistema, quiero poder eliminar un asistente, para que no esté en el evento.

1. En el backend, se ha creado la función para permitir el método DELETE:

```
router.delete('/:id',
  passport.authenticate('bearer', { session: false }),
  async function(req, res, next) {
  const id = req.params.id;

  try {
    const result = await Assistant.findOneAndDelete({ _id: id });
    console.log(result);
    if (result) {
      res.sendStatus(204);
    } else {
      res.sendStatus(404);
    }
  } catch (e) {
    // Manejar cualquier error que ocurra durante la eliminación
    debug("DB problem", e);
    res.sendStatus(500);
  }
});
```

2. En el frontend, dentro del componente AssistantItem, que se encarga de mostrar los datos del asistente dentro del listado, se ha desarrollado la funcionalidad de eliminar el asistente haciendo la llamada a la API.

```
static async deleteAssistant(id) {
    const headers = this.requestHeaders();
    headers['Authorization'] = 'Bearer 37f7b2a4-2fdc-4e17-a72b-6570187d3cb6';
    const request = new Request(API_BASE_URL + `/assistants/${id}`, {
      method: 'DELETE',
      headers: headers
    });

    const response = await fetch(request);

    if (!response.ok) {
      throw new Error("Response not ok"+response.status);
    }

    return response.json();
}
```

```
const handleDelete = async () => {
    try {
        const response = await AssistantAPI.deleteAssistant(assistant._id);
        console.log(response);
    } catch (error) {
        console.error('Error deleting assistant:', error);
    }
    //Recargar la página
    window.location.reload();
};
```

EventHub-Events

Listar eventos

Como usuario del sistema, quiero poder listar los eventos disponibles a los que poder acudir.

1. En el backend se ha creado una función que permite, mediante llamadas a la API, obtener los eventos disponibles:

```
/* GET events listing*/
router.get("/", async function (req, res, next) {
  try {
    const result = await Event.find();
    res.send(result.map((c) => c.cleanup()));
  } catch (e) {
    debug("DB problem", e);
    res.sendStatus(500);
  }
});
```

También obtener un evento por id y por nombre:

```
/* GET */
router.get("/:name?", async function (req, res, next) {
  var eventName = req.params.name;

  if (eventName) {
    // If eventId is provided, find and return the specific event
    var event = await Event.findOne({ name: eventName });
    if (event) {
      res.json(event);
    } else {
      res.status(404).json({ error: "Event not found" });
    }
  } else {
    // If no eventId provided, return the entire list of events
    const result = await Event.find();
    res.send(result.map((c) => c.cleanup()));
  }
});
```

```
/* GET by _id */
router.get("/id/:id", async function (req, res, next) {
  var eventId = req.params.id;

  if (eventId) {
    // If eventId is provided, find and return the specific event
    var event = await Event.findById(eventId);
    if (event) {
      res.json(event);
    } else {
      res.status(404).json({ error: "Event not found" });
    }
  } else {
    // If no eventId provided, return the entire list of events
    const result = await Event.find();
    res.send(result.map((c) => c.cleanup()));
  }
});
```

2. Por otra parte, en el frontend, se ha creado un componente, **Events** que mediante la llamada a la API, obtiene todos eventos y los muestra:

```
static async getAllEvents() {
  const headers = this.requestHeaders();
  const request = new Request(API_BASE_URL + `/events`, {
    method: "GET",
    headers: headers,
  });

  const response = await fetch(request);

  if (!response.ok) {
    throw Error("Response not valid" + response.status);
  }

  return response.json();
}
```

También métodos para obtener el evento por nombre e id, para que otros microservicios puedan usarlos, como por ejemplo el microservicio de Asistentes que añade un asistente a un evento determinado por el id.

```
static async getEvent(name) {
  const headers = this.requestHeaders();
  const request = new Request(API_BASE_URL + `/events/${name}`, {
    method: "GET",
    headers: headers,
  });

  const response = await fetch(request);

  if (!response.ok) {
    throw new Error("Response not ok" + response.status);
  }

  return response.json();
}
```

```
static async getEventById(id) {
  const headers = this.requestHeaders();
  const request = new Request(API_BASE_URL + `/events/id/${id}`, {
    method: "GET",
    headers: headers,
  });

  const response = await fetch(request);

  if (!response.ok) {
    throw new Error("Response not ok" + response.status);
  }

  return response.json();
}
```

```
function Events() {
  const [message, setMessage] = useState(null);
  const [events, setEvents] = useState([]);
  const [showNewEventFields, setShowNewEventFields] = useState(false);

  useEffect(() => {
    async function fetchEvents() {
      try {
        const e = await EventsApi.getAllEvents();
        setEvents(e);
      } catch (error) {
        setMessage("Could not contact with the server");
      }
    }

    fetchEvents();
  }, []);
```

Crear un evento

Como usuario del sistema, quiero poder crear un evento con sus campos.

1. En el backend se ha creado una función que permite realizar una petición POST a la API:

```
/* POST create a new event. */
router.post(
  "/",
  passport.authenticate("bearer", { session: false }),
  async function (req, res, next) {
    const { name, place, date, description, category } = req.body;

    const event = new Event({
      name,
      place,
      date,
      description,
      category,
    });
    try {
      await event.save();
      res.sendStatus(201); // Envía respuesta 201 solo cuando la operación es exitosa
    } catch (e) {
      if (e.errors) {
        console.error("Validation problem with saving", e);
        res.status(400).send({ error: e.message });
      } else {
        console.error("DB problem", e);
        res.status(500).send({
          error: "Internal Server Error: Failed to save event to the database.",
        });
      }
    }
  }
);
```

2. En el frontend, se ha creado un componente, *CreateEvent* que muestra un formulario para añadir un evento:

```
static async createEvent(event) {
  try {
    const headers = this.requestHeaders();
    const response = await axios.post(API_BASE_URL + `/events`, event, {
      headers,
    });
    return response.data;
  } catch (error) {
    console.error("Error creating event:", error);
    throw error;
  }
}
```

```
function NewEvent(props) {
  const [name, setName] = useState("");
  const [place, setPlace] = useState("");
  const [dateTime, setDateTime] = useState(new Date());
  const [category, setCategory] = useState("");
  const [description, setDescription] = useState("");
  const [formError, setFormError] = useState("");

  function onClick() {
    if (!name || !dateTime || !place) {
      setFormError("Name, place, and date are required fields.");
      return;
    }

    const newEvent = {
      name: name,
      place: place,
      date: dateTime.toISOString(),
      category: category,
      description: description,
    };

    const result = props.onAddEvent(newEvent);

    if (result) {
      setName("");
      setPlace("");
      setDateTime(new Date());
      setCategory("");
      setDescription("");
      setFormError("");
    }
  }
```

Editar un evento

Como usuario del sistema, quiero poder editar un evento para poder modificar algunos de sus datos.

1. En el backend, se ha creado la función para permitir el método PUT de la API:

```
/* PUT update an existing event. */
router.put(
  "/:name",
  passport.authenticate("bearer", { session: false }),
  async function (req, res, next) {
    var name = req.params.name;
    var updatedEvent = req.body;

    try {
      const existingEvent = await Event.findOne({ name: name });
      if (existingEvent) {
        Object.assign(existingEvent, updatedEvent);
        await existingEvent.save();
        res.json(existingEvent);
      } else {
        res.status(404).json({ error: "Event not found" });
      }
    } catch (error) {
      console.error("DB problem", error);
      res.status(500).send({
        error: "Internal Server Error: Failed to update event in the database.",
      });
    }
  }
);
```

2. En el frontend, se han creado los componentes, *EditEvent* y *EditableEvent*, para realizar la llamada a la API y mostrar un formulario para modificar los campos necesarios:

```
static async updateEvent(eventName, updatedEvent) {
  const headers = this.requestHeaders();
  const request = new Request(API_BASE_URL + `/events/${eventName}`, {
    method: "PUT",
    headers: {
      ...headers,
      "Content-Type": "application/json",
    },
    body: JSON.stringify(updatedEvent),
  });

  try {
    const response = await fetch(request);

    if (!response.ok) {
      throw Error("Response not valid" + response.status);
    }

    return response.json();
  } catch (error) {
    console.error("Error updating event:", error);
    throw error;
  }
}
```

```
function EditEvent(props) {
  const [name, setName] = useState(props.event.name);
  const [place, setPlace] = useState(props.event.place);
  const [dateTime, setDateTime] = useState(new Date(props.event.date));
  const [category, setCategory] = useState(props.event.category);
  const [description, setDescription] = useState(props.event.description);

  const onSave = () => {
    props.onSave({
      ...props.event,
      name: name,
      place: place,
      date: dateTime.toISOString(),
      category: category,
      description: description,
    });
  };
```

```
function EditableEvent({ event, onSave, onCancel }) {
  const [editableEvent, setEditableEvent] = useState({
    name: event.name,
    place: event.place,
    date: new Date(event.date),
    category: event.category,
    description: event.description,
  });

  const handleFieldChange = (fieldName, value) => {
    setEditableEvent((prevEditableEvent) => ({
      ...prevEditableEvent,
      [fieldName]: value,
    }));
  };

  const handleSave = () => {
    onSave(editableEvent);
  };
```

Eliminar un evento

Como usuario del sistema, quiero poder eliminar un evento.

1. En el backend, se ha creado la función para permitir el método DELETE:

```
/* DELETE delete an existing event. */
router.delete(
  "/:name",
  passport.authenticate("bearer", { session: false }),
  async function (req, res, next) {
    var name = req.params.name;
    try {
      const deletedEvent = await Event.findOneAndDelete({ name: name });
      if (deletedEvent) {
        res.json(deletedEvent);
      } else {
        res.status(404).json({ error: "Event not found" });
      }
    } catch (error) {
      console.error("DB problem", error);
      res.status(500).send({
        error:
          "Internal Server Error: Failed to delete event from the database.",
      });
    }
  }
);
```

2. En el frontend, dentro del componente EventItem, que se encarga de mostrar los datos del evento dentro del listado, se ha desarrollado la funcionalidad de eliminar el evento haciendo la llamada a la API.

```
static async deleteEvent(eventName) {
  const headers = this.requestHeaders();
  const request = new Request(API_BASE_URL + `/events/${eventName}`, {
    method: "DELETE",
    headers: headers,
  });

  try {
    const response = await fetch(request);

    if (!response.ok) {
      throw Error("Response not valid" + response.status);
    }

    return response.json();
  } catch (error) {
    console.error("Error deleting event:", error);
    throw error;
  }
}

static async deleteEventById(id) {
  const headers = this.requestHeaders();
  const request = new Request(API_BASE_URL + `/events/${id}`, {
    method: "DELETE",
    headers: headers,
  });

  try {
    const response = await fetch(request);

    if (!response.ok) {
      throw Error("Response not valid" + response.status);
    }

    return response.json();
  } catch (error) {
    console.error("Error deleting event:", error);
    throw error;
```

Análisis de esfuerzos

El análisis de esfuerzos de cada miembro está más detallado en los documentos adjuntos, esto sería un resumen:

- Jorge Sillero Manchón: 48:45h
- Ángela Bernal Martín: 73:03h
- Francisco Andrés Caro Albarrán: 47:45h

Clockify_Informe_Ángela_Bernal_Martín.pc

Clockify_Jorge_Sillero.pdf

Clockify_Time_Francisco_Andrés_Caro.pdf