UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

WParty App

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Giner García, Antonio José

Tutor/a: Terrasa Barrena, Andrés Martín

Cotutor/a: Marzal Calatayud, Eliseo Jorge

CURSO ACADÉMICO: 2021/2022

Resumen

WParty App será una aplicación web y móvil (iOS y Android) que ayudará, tanto a empresas como a usuarios particulares, a la organización de forma online de eventos. Esta aplicación contará con funcionalidades como la creación de eventos públicos y privados, venta de entradas, controles de acceso y creación de fondos de gasto.

Está aplicación tendrá una arquitectura estructurada por capas. En este TFG desarrollaremos la lógica de negocio y la lógica de datos. La lógica de negocio estará implementada en un middleware en forma de API REST que, mediante las operaciones típicas de tipo "CRUD" vía HTTP, realizará peticiones a la capa de persistencia para consultar o modificar la información requerida. Por otra parte, la capa de persistencia recogerá esas peticiones en un back-end. En particular, contaremos con dos bases de datos: una base de datos de almacenamiento de datos CORE y otra base de datos de acceso rápido, cuyo objetivo será permitir un acceso directo desde la lógica de negocio para habilitar consultas de manera más eficiente.

Por otro lado, el desarrollo de este proyecto será de tipo *Test Driven Development* (TDD), así que también el TFG también incluirá cómo funciona esta metodología y se aportarán todos los casos de prueba.

Finalmente, en este trabajo abordaremos también aspectos tales como el presupuestado de costes de explotación de los procesos y el almacenamiento, la explotación económica de nuestra aplicación y la seguridad dentro de toda nuestra infraestructura.

Palabras clave: aplicación web, aplicación móvil, empresas, particulares, online, eventos, capas, lógica de negocio, API, REST, HTTP, peticiones, CRUD, persistencia, CORE, acceso rápido, TDD, costes, seguridad.

Abstract

WParty App will be a web and smartphone (iOS and Android) application that will help both, companies and private users, to organize events online. This application will have functionalities such as the creation of public and private events, ticket sales, access control and the creation of expense pulls.

This application will have a layered architecture. In this TFG we will develop the business and data logic.

This application will have a layered architecture. In this TFG we will develop the business logic and the data logic. The business logic will be implemented in a middleware as a REST API that, by typical "CRUD" HTTP operations, will make requests to the persistence layer to query or modify the required information. On the other hand, the persistence layer will collect these requests in a back-end. We will have two databases: a CORE data storage database and a quick access database, whose objective will be to allow direct access from the business logic to enable queries in a more efficient way.

The development of this project will be a Test-Driven Development (TDD), so we will also see in this document how it works, and we will provide all the test cases.

In this work we will also address issues such as the budgeting of operating costs of processes and storage, the economic exploitation of our application and the security within our entire infrastructure.

Keywords: web application, smartphone application, company, private, online, events, layers, business logic, API, REST, HTTP, requests, CRUD, persistence, CORE, fast access, TDD, costs, security.

Tabla de contenidos

Índice de tablas

Índice de ilustraciones

1 Introducción

1.1 Motivación

Este proyecto se plantea originalmente a través de una necesidad personal frente a la complejidad que presenta la gestión de un evento para un usuario particular. En la actualidad existe una carencia clara de aplicaciones de gestión de eventos en el mercado, orientadas a usuarios particulares que quieran delegar la gestión económica y de invitados a un sistema que maneje toda esa casuística de forma automática, además de tener reunida en un mismo punto toda la información de asistencia y de recaudación de los propios eventos.

Hoy en día, la organización de eventos privados por parte de particulares se suele gestionar creando grupos en aplicaciones de mensajería incluyendo a los invitados y pidiendo confirmación de asistencia, o de forma más rudimentaria aún, usando cartas postales acreditando al receptor como invitado a un evento (esta alternativa se sigue utilizando hoy en día para eventos formales como bodas, por ejemplo). Esta forma de organización presenta además claros problemas, entre los que destacaríamos:

- La gestión suele recaer sobre una persona o un grupo reducido de personas.
- La gestión de pagos por parte de los organizadores añade más problemática a la gestión.
- Existen problemas para calcular el número de asistentes reales al evento.
- Puede haber desconfianza por parte de los asistentes de la gestión del precio de las invitaciones.

Por otro lado, desde el punto de vista de los posibles asistentes a un evento, también encontramos problemas para consultar la oferta de eventos disponibles. No existen productos en el mercado que permitan a un usuario consultar los eventos que existen en una zona o lugar concreto, y gestionar su asistencia a través de ellos. Incluso, muchas empresas utilizan a personas influyentes en redes sociales para promocionar sus eventos y a través de ellos gestionar listas de invitados. Esto produce un esfuerzo extra, porque los gestores de eventos tienen que gestionar también la lista de estas personas externas a la empresa.

En definitiva, además de basarnos en la experiencia personal, hemos identificado en esta área varias necesidades de usuarios de distintos perfiles:

- Existen empresas que necesitan una plataforma con la que promocionar y gestionar sus eventos, incluyendo la venta de entradas y el control de la asistencia a ellos. Idealmente, la plataforma puede ofrecerles estadísticas de los asistentes e incluso información complementaria como el número total de visualizaciones que tiene su publicación u otras estadísticas que ayuden al desarrollo empresarial.

- Existen usuarios interesados en asistir a eventos en un lugar específico. En este caso, la necesidad incluye al menos poder consultar la oferta local de eventos y si es posible, gestionar la preventa de asistencia a un evento en concreto y facilitar el acceso a este.
- Finalmente, existen usuarios particulares que quieren organizar un evento (ya sea público o privado) y que necesitan herramientas que le faciliten dicha organización.

En definitiva, la problemática que aborda este trabajo es mejorar la gestión de una tarea tan costosa como es la organización de eventos, facilitando herramientas que puedan utilizar tanto empresas como particulares, de forma que puedan reducir el tiempo y esfuerzo que habitualmente dedican a este tipo de actividades.

1.2 Objetivos

El objetivo último de este proyecto es desarrollar una aplicación que permita una gestión integral de organización de eventos, tanto de particulares como de empresas, realizando un diseño e implementación modernos tecnológicamente y eficientes con los recursos. Dada la envergadura de un proyecto así, este Trabajo Fin de Grado se va a centrar en diseñar e implementar el sistema de soporte (*back end*) que permita en un futuro implementar la aplicación, o aplicaciones, que interaccionarían con los distintos usuarios.

A partir de ese objetivo general, se plantea un conjunto de objetivos secundarios enfocados a conseguir este sistema que dote a dicha aplicación de usuario de una infraestructura estable, escalable y eficiente. Por ello, los objetivos secundarios son los siguientes:

- Los flujos de las operaciones deben de resolverse en unos tiempos relativamente bajos, reduciendo al mínimo los tiempos de espera del usuario, pero sin eliminar la fiabilidad y consistencia que debe de tener un sistema estable.
- Todo el código debe de ser implementado siguiendo estándares y buenas prácticas existentes actualmente, facilitando así la mantenibilidad, escalabilidad e incorporación de nuevos desarrolladores al proyecto, si llega el caso.
- Las pruebas de integración deben de ser capaces de cubrir al menos el 98% de las líneas de código implementadas en nuestra solución. Esto no solo nos dará una gran fiabilidad, sino que además nos ayudará en temas de posicionamiento SEO en diferentes motores de búsqueda usados en internet.
- Las distintas partes del sistema deben realizar un uso eficiente de la memoria usada para sus procesos, para que la aplicación sea más sostenible con el medio ambiente y ahorrar costes en infraestructura hardware o recursos en la nube.
- La seguridad del sistema se considera un pilar fundamental de nuestro proyecto. En concreto, los datos sensibles del usuario deben de estar totalmente enmascarados en nuestra base de datos, y los flujos que recorran estos datos deben estar convenientemente protegidos.

1.3 Estructura del documento

La memoria se estructura en 9 capítulos más uno extra que incluye la bibliografía y un anexo. A continuación, comentaremos uno por uno los capítulos que forman parte de esta memoria.

El primer capítulo es introductorio al proyecto. Este es fundamental para dar contexto y explicar el porqué de este proyecto. En este capítulo se introducen los objetivos que queremos alcanzar para poder llegar a la solución esperada y esta estructura de la memoria.

En el capítulo siguiente, denominado “Estado del arte”, analizamos sistemas en los que se cubren algunas de las funcionalidades que este proyecto realizará y otras funciones que no son cubiertas por su solución. En este capítulo también se revisa el contexto tecnológico de los sistemas a implementar, dando una introducción a las tecnologías principales que usaremos en nuestro proyecto.

En el Capítulo 3 se realiza un análisis exhaustivo de los requisitos del proyecto, definiendo todos los requisitos que nuestro proyecto debe de cumplir para que la solución sea una válida. Además, este capítulo incorpora un apartado en el que analizamos y definimos los casos de uso en los que un usuario de nuestra aplicación se vería inverso.

A partir de este punto, la memoria trata todos los temas que tienen que ver con el desarrollo del sistema como tal.

Para comenzar, en el Capítulo 4 se muestra la arquitectura global del proyecto, incluyendo todos los sistemas que conformarán nuestra solución, ofreciendo inicialmente un diseño general. Posteriormente, se explica en detalle el diseño sistema por sistema, para que cada uno se adecúe a los requisitos definidos anteriormente.

Una vez definidos todos los sistemas que conforman la solución, en el Capítulo 5, “Desarrollo de la solución”, explicaremos como hemos llevado a cabo esta solución, dando evidencias y ejemplos de ello. En este capítulo se incluirá también la descripción de los despliegues en los sistemas de *hosting* que usaremos para albergar nuestros sistemas y bases de datos.

A continuación, el Capítulo 6 se destina exclusivamente a definir las pruebas unitarias y de integración de la solución. Las pruebas son igual o más importantes que el propio desarrollo, ya que nos garantizan que nuestra solución es fiable y, más importante aún, se cumplen los todos los requisitos definidos en apartados previos.

Finalmente, el Capítulo 7 se comentarán las conclusiones que hemos alcanzado durante el desarrollo del proyecto, y revisaremos el grado de cumplimiento de los objetivos planteados inicialmente. Además, en este capítulo, se van a relacionar los estudios cursados con el trabajo definido y desarrollado. Para ello, se indicarán los conocimientos necesarios para poder llevar a cabo el proyecto, explicando en cada caso su grado de aportación. Indicaremos también otros

conocimientos más relacionados con las habilidades blandas, adquiridas en el transcurso del grado, y que también son primordiales para llevar un proyecto de estas características. Para finalizar este capítulo de conclusiones, destinamos un apartado a los trabajos futuros, que engloban algunas de las posibles mejoras que se podrían aportar al proyecto.

Respecto a los anexos, se incluye uno específico sobre la relación del proyecto con los Objetivos de Desarrollo Sostenible (ODS) En este anexo explicamos cómo hemos llevado a cabo la solución siguiendo una estrategia muy vinculada a la conciencia cívica y responsable con el desarrollo de la sociedad y el cuidado del mundo que nos rodea.

2 Estado del arte

En este capítulo de la memoria, describimos algunos de los antecedentes existentes en el mercado que realicen funciones similares a las que nosotros implementaremos. Para ello, hemos escogido dos empresas, plenamente dedicadas a la gestión de eventos, que disponen de sendas plataformas que automatizan procesos similares a los que nuestro proyecto realizará. Analizando las funciones que realizan estas aplicaciones, veremos si realmente nuestro producto es válido y aprovechable por un usuario que esté usando otra aplicación similar.

Para que una aplicación sea resulte útil y tenga demanda, no es necesario que sea totalmente revolucionaria, sino que, añadiendo valor a las soluciones ya existentes, podemos captar muchos usuarios que estén usando esas aplicaciones pero que necesiten otras funcionalidades no disponibles. Además, en este capítulo analizaremos también funciones que nosotros no implementamos. Este análisis, nos será de gran ayuda para poder recopilar ideas de expansión de nuestro proyecto a partir de nuestra propuesta inicial.

Finalmente, el capítulo también incorpora secciones que analizan y una breve introducción a las tecnologías principales que componen este proyecto. Gracias a estas secciones, el lector será capaz de entender las decisiones que hemos tomado para escoger las tecnologías que creemos más adecuadas para cada componente de la solución.

2.1 Análisis de soluciones existentes

Una plataforma de gestión de eventos es una aplicación ya sea web, dispositivo móvil o de escritorio, de la que propietaria una empresa u organización, la cual, ofrece servicios o sistemas en los que un usuario puede delegar funciones a la hora de gestionar un evento. La mayoría de las empresas dedicadas a la gestión de eventos ofrecen servicios de contratación de personal, dispositivos de control de acceso, venta de entradas y marketing, entre otras.

No son muy conocidas las plataformas que ofrezcan una aplicación en la que un usuario pueda consultar a qué eventos puede asistir en una localidad en concreto. La mayoría de los usuarios deben recurrir a otras aplicaciones que no tienen un espacio dedicado para ello y normalmente deben acceder a otro portal para poder gestionar su asistencia.

Generalmente, para que un usuario pueda conocer que eventos se encuentran cerca de él, tiene que hacer uso de varias herramientas, como ya hemos comentado y, además, realizar un trabajo de investigación como preguntar a los residentes para poder conocer la oferta de una zona. Con una herramienta similar a la que proponemos, todas esas necesidades estarían recogidas en una única aplicación.

Existen múltiples plataformas online que permiten la organización de eventos, pero están muy orientadas a la aplicación empresarial. Para corroborarlo, analizamos varios ejemplos de este tipo de aplicaciones a continuación.

2.1.1 Beamian

Beamian [1] es una plataforma híbrida de gestión de eventos, es decir, es una plataforma en la que los usuarios son capaces de gestionar eventos virtuales y presenciales.

Se trata de un portal web en el que otras empresas solicitan un servicio de gestión de eventos. La empresa organizadora se encarga de organizar de forma integral todo lo relativo a la gestión del evento: venta de entradas, información sobre asistentes, contratación de personal, reserva de instalaciones, creación de una plataforma online para un evento online, etc.

Esta empresa, no dispone de una aplicación en la que un usuario, o posible consumidor, pueda consultar los eventos ofrecidos por la empresa y, por lo tanto, tampoco puede consultar la oferta local. Además, un usuario particular que quiera crear su propio evento tendría que pagar un elevado coste para poder organizar un evento a través de esta empresa. Es por esto, por lo que, catalogamos a esta empresa como una empresa de subcontratación de gestión de eventos.

Pero hay un servicio ofrecido por esta empresa que merece una mención especial, y es que dispone de un equipo personalizado para cada evento bajo demanda. La empresa pone a la disposición del cliente un grupo de personas que estarán totalmente centradas en la organización de ese evento en particular. De esta forma, se obtiene un nivel de detalle y de compromiso muy alto.

2.1.2 Meetmaps

Meetmaps [2] es una empresa muy parecida a la anterior. Cuenta con una aplicación que es capaz de gestionar eventos presenciales, híbridos y virtuales en donde sus servicios están muy orientados al ámbito empresarial, con una interfaz y unas herramientas más sofisticadas que Beamian.

Se trata de una empresa muy conocida que ha sido capaz de gestionar eventos empresas tan grandes como *Huawei*, *HP*, *American Express*, y el equipo de fútbol "*FC Barcelona*", entre otras muchas.

Analizando las funciones que ofrece, vemos que dispone de servicios y funcionalidades para la propia organización de eventos que encontramos muy interesantes, como son:

- Alquiler de hardware específico, como terminales de control de acceso o presentaciones interactivas.
- Soporte personalizado a cada evento. La empresa te da la opción de contratar un equipo para la gestión del evento que acuda expresamente para un evento en concreto.
- Herramientas de activación de público. Estas herramientas son utilizadas por los organizadores de eventos para generar *engagement* [3] vía encuestas en directo, creando

formularios de solicitud a un puesto de trabajo, concursos y más ejemplos que se podrían usar para generar ese sentimiento en los asistentes.

En nuestra opinión, esta empresa está mucho más enfocada a la gestión de congresos ya sean tecnológicos o de cualquier otro ámbito, ya que, ofrece un servicio de creación de eventos únicamente para empresas. Es decir, lo que ofrece esta empresa puede considerarse como una forma de subcontratar por parte de otras empresas la creación y gestión de congresos y mítines de todo tipo.

En contraposición y analizando las funciones que implementa esta aplicación, no se trata de una aplicación que utilizarían otras empresas dedicadas a la organización de eventos. Muchas empresas ya tienen cubierta la parte de la gestión, y simplemente quieren dar visibilidad a sus eventos con el objetivo de captar nuevos asistentes. *Meetmaps,* al no disponer de un portal o una aplicación en la que otras empresas publiquen sus eventos, no es capaz de hacer esa difusión que muchas empresas buscan a través de la contratación de personas o servicios externos.

2.2 Contexto tecnológico

Esta sección está destinada a presentar las tecnologías más representativas que hemos escogido para el desarrollo de este proyecto.

La mayoría de las tecnologías usadas masivamente presentan ventajas e inconvenientes, y existen argumentos a favor y argumentos en contra de su uso. Al utilizar una tecnología, debemos sopesar cuáles son los argumentos que introducen un impacto positivo en nuestro caso de uso, y cuáles plantean un impacto negativo.

Existen numerosos ejemplos de enfrentamientos entre desarrolladores, defendiendo que cierta tecnología es superior a otra, aunque en nuestra opinión no suele existir una tecnología definitiva, sino que todo depende del uso que queramos darle.

Gracias a la experiencia personal previa y durante el transcurso de este proyecto, veremos que la gran mayoría de tecnologías son buenas si sabemos cómo aplicarlas y en qué casos utilizarlas.

Las aplicaciones desarrolladas en Java están muy estigmatizadas. Los argumentos usados son que el desarrollador dispone de menor flexibilidad a la hora de desarrollar el código y que se obtienen peores resultados en métricas como el tiempo de ejecución, el tiempo de despliegue de la aplicación o un alto uso de memoria [4]. Por contra, muchos desarrolladores defienden Java como un buen lenguaje en el que implementar una aplicación Back-end [5] ya que ofrece ventajas en el tipado de variables y en el control del flujo de ejecución, además de ser un lenguaje orientado a objetos, por lo que es sencillo representar elementos del mundo real dentro de la propia aplicación.

Con esta simple argumentación vemos que todo depende del contexto con el que queramos usar una tecnología contrastada.

En los siguientes subapartados de esta sección revisaremos los sistemas principales que van a componer nuestra solución. En cada uno de ellos haremos una pequeña introducción a las diferentes tecnologías elegidas para su desarrollo y ofreceremos argumentos del porqué de su elección. En concreto, estos sistemas son la base de datos relacional, la base de datos no relacional el sistema de *back-end* y el *middleware*.

2.3 Base de datos relacional

Una base de datos relacional [6] es un tipo de implementación de almacenamiento estático de información que sirve para guardar los datos de nuestra aplicación de forma enlazada e indexada. La principal peculiaridad que presentan estas bases de datos es que los datos que se almacenan en ellas se estructuran de una forma determinada, concretamente en tablas.

Este modelo de almacenamiento de información es muy útil para los desarrolladores, ya que estas tablas son una forma de representación del mundo real. Es por ello, que la implementación de un modelo relacional tiene una gran concordancia con los objetos reales que se quieren representar en una aplicación. Eso sí, para ello el modelado de una base de datos relacional debe de ser el adecuado, ya que en caso contrario podríamos llegar a incoherencias e inconsistencias en los datos almacenados.

Las ventajas que tendremos al usar una implementación de una base de datos relacional para almacenamiento persistente son:

1. Seguridad: Al tener un sistema que permite la creación de roles que gestionen los permisos sobre las diferentes tablas, hay un mayor control sobre quién puede realizar operaciones sobre la base de datos e incluso es posible restringir ciertas operaciones a una serie de usuarios.
2. Longevidad de la tecnología: Las bases de datos relacionales se llevan usando por décadas, por lo que se han creado múltiples recursos para los desarrolladores. Además, debido al gran uso de los desarrolladores, es una tecnología muy depurada, por lo que los errores no controlados serán mínimos.
3. Estandarización: La manera habitual de manipular y consultar BBDD relacionales es mediante el lenguaje SQL [7]. Este lenguaje es muy rígido y estandarizado, lo cual ayuda a los usuarios a no cometer errores a la hora de realizar operaciones relativas a la modificación de los datos o de las propias tablas.
4. Lenguaje natural: Una de las grandes virtudes que tiene SQL, es una curva de aprendizaje muy alta porque, básicamente, usa un lenguaje natural para definir las operaciones con la que se opera dentro de las tablas y los datos.

Las bases de datos relacionales tienen múltiples implementaciones (Oracle, MariaDB, MySQL…) y, entre ellas, para nuestro proyecto utilizaremos la implementación PostgreSQL.

PostgreSQL es un motor de base de datos que, según su página oficial, es "el motor de base de datos más potente del mundo" [8]. Esta implementación cuenta con las siguientes ventajas:

- Es de código abierto por lo que, al estar expuesto a la comunidad, ayuda a la resolución de errores, aporta nuevas funcionalidades y mejoras al software, y cuenta con una comunidad extensa que aporta soluciones a múltiples problemáticas que otros desarrolladores pueden encontrarse en el proceso de creación de bases de datos.
- PostGIS: Es un servicio de geolocalización incluido en PostgreSQL, que nos facilitará el proceso de elaboración de analíticas en un futuro, e incluso utilizar este servicio para la localización de nuestros eventos.
- PL/PgSQL: Es un lenguaje de programación propio de PostgreSQL que nos Ayurá a aligerar la carga de nuestros middlewares automatizando procesos que no requieren de respuesta al usuario final.
- Cumple con el estándar ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad) que es por supuesto fundamental para el mantenimiento correcto de la base de datos y el correcto funcionamiento de las operaciones realizadas en ella.

Por todas estas ventajas de PostgreSQL frente a otras implementaciones, nos hemos decantado por esta tecnología. PL/PgSQL nos ayudará a eliminar código relativo al manejo de los datos en otras aplicaciones del sistema, ayudando así a la encapsulación de la base de datos y a la escalabilidad del sistema. Por otra parte, PostGIS ofrece un abanico de opciones en el crecimiento de funcionalidades a poder implementar en la aplicación, que supone una ventaja de PostgreSQL muy importante que no ofrecen otros sistemas como MySQL.

2.4 Base de datos no relacional

Una base de datos no relacional [9] es un modelo de almacenamiento persistente al igual que una base de datos relacional, pero que, a diferencia de este, sus entradas no se representan en filas dentro de tablas, sino que hay múltiples opciones para su representación. Esta representación puede ser a nivel de contenedor, documentos, objetos indexados u otras opciones.

Igual que en el apartado anterior, también existen múltiples alternativas tecnológicas en bases de datos no relacionales. En este caso, el producto que hemos elegido se denomina ElasticSearch [38]. Respecto a otras implementaciones o productos similares como DynamoDB [10], MemoryDB [11] o Apache Cassandra [12], entre otras, hemos elegido ElasticSearch por ofrecer varias ventajas que comentamos a continuación.

ElasticSearch es un motor de búsqueda y recuperación de documentos desarrollado en Java y basado en una API de Apache (Lucene [13]) que ofrece respuestas muy rápidas ante consultas. Una de las consultas que se pueden realizar contra esta base de datos y una de las principales ventajas

que ofrece, son las búsquedas de texto completo. Este tipo de búsquedas resultan ideales para uno de nuestros casos de uso principales, ya que el enfoque que tendrá esta base de datos dentro de nuestro proyecto será la de actuar como memoria caché de acceso rápido dentro de un buscador en la aplicación de usuario.

Otra de las principales ventajas que ofrece Elastic frente a otras tecnologías de bases de datos no relacionales, es que el motor de búsqueda está expuesto a través de una interfaz de tipo API REST. Esto nos va a ahorrar mucho tiempo en desarrollo porque básicamente, no vamos a tener que encapsular la base de datos con una aplicación para hacer consultas legibles y seguras. Además, al ser RESTful, esta API retorna respuestas mediante objetos JSON, que son objectos propios nativos de JavaScript, por lo que ahorrará mucho tiempo de ejecución y desarrollo a la hora de mapear esas respuestas. Además, las respuestas ofrecen métricas que ponderan su valor, es decir, que las propias respuestas ofrecidas por el motor de búsqueda ya vienen ordenadas según la relevancia que tienen sobre la entrada del usuario y de una serie de parámetros que se incluyen en las llamadas a la consulta que realicemos.

Pero no solamente ofrece estas ventajas a nivel de lógica de negocio, sino que el entorno Elastic ofrece servicios adicionales que se usan en análisis de sistemas distribuidos, tales como analíticas de logs, análisis de seguridad, reportes gráficos de uso en documentos y funcionalidades, y un largo etcétera. Gracias a este entorno, estos servicios están automatizados una vez se instalan las dependencias correspondientes.

Finalmente, a nivel de funcionalidad propia de una base de datos, ElasticSearch ofrece todos los estándares ACID [9], de manera que garantiza que las transacciones sobre una base de datos se procesen de una manera fiable.

2.5 Back-end

Un sistema Back-end se caracteriza por ser una aplicación, o un conjunto de ellas, encargada del acceso a la capa de persistencia y de la aplicación de la lógica de negocio de un sistema completo. Como en nuestro caso de uso ya tenemos una aplicación encargada de la lógica de negocio, esta aplicación Back-end será la encargada del acceso a la base de datos principal.

Java es uno de los lenguajes más aceptado por los programadores del mundo. Esto se debe a que es un lenguaje orientado a objetos (POO), por lo que es muy sencillo representar objetos del mundo real en el propio código de la aplicación utilizando clases que establezcan parámetros que describan de alguna forma el objeto en concreto a representar.

Para este componente buscamos una tecnología que sea capaz de enmascarar las operaciones que realizamos sobre la base de datos principal de forma escalable, multiplataforma y que haga un buen

uso de la memoria donde se hospede la aplicación. Java cumple todos esos requisitos que necesitamos.

Java es uno de los lenguajes más aceptado por los programadores del mundo. Esto se debe a que es un lenguaje orientado a objetos (POO), por lo que es muy sencillo representar objetos del mundo real en el propio código de la aplicación utilizando clases que establezcan parámetros que describan de alguna forma el objeto en concreto a representar. Además, añade una característica que aumenta el rendimiento frente a otras tecnologías: es un lenguaje multihilo. Esto le permite obtener una mejora en peticiones que lleguen de manera simultánea a nuestra aplicación, atendiendo estas peticiones al mismo tiempo y sin riesgo a errores por condiciones de carrera.

Para hacer peticiones a nuestro sistema Back-end, utilizaremos peticiones HTTP por lo que necesitamos un componente que sea capaz de transformar esas peticiones HTTP a peticiones a la aplicación para que devuelva un resultado de la operación requerida y a su vez transforme esta respuesta a una respuesta HTTP. Ahí es donde entra en juego Spring Boot [15].

Spring Boot, parte del framework de Spring, es ideal para estos casos. Esta librería provee a una aplicación Java un servidor Tomcat [16] preinstalado. De esta forma, tras una simple configuración previa, disponemos de un servidor web preparado para escuchar peticiones HTTP y elaborar las propias respuestas HTTP. Esto en el caso en el que queramos una instalación de un servidor personalizado. En el caso en el que no queramos un servidor personalizado, se trataría de una solución "*Plug and play*", es decir que, integrando la librería de Spring Boot, por defecto, tendríamos un servidor web incorporado sin necesidad de realizar ninguna tarea extra.

Gracias a esto, el despliegue del Back-end será muy sencillo. Para ello, será suficiente con instalar un entorno Java en el servidor que vaya a albergar la aplicación y ejecutar el archivo ejecutable (de tipo `jar`) que tendremos como resultado de compilar el programa Java.

Spring Framework ofrece otras muchas ventajas, como, por ejemplo:

- Anotaciones que nos facilitarán el desarrollo y la implementación de peticiones SQL a la base de datos relacional.
- Seguridad en el acceso a las operaciones declaradas en la aplicación.
- Ayuda al desarrollo orientado al *Interface Based Programming* (Programación Orientada en Interfaces) que otorga un mayor nivel de abstracción a los desarrolladores con respecto a la implementación de operaciones de las que sólo debe de conocer como invocar y que va a obtener de ellas.
- Provee un sistema automático de inyección de dependencias.

Con esta serie de ventajas podremos reducir los tiempos de desarrollo, añadir seguridad al proyecto de una manera automatizada y sencilla, encapsulación entre capas de la aplicación y un sistema de inyección de dependencias gestionado de forma automática.

2.6 Middleware

Una aplicación *middleware*, es un programa intercalado entre la capa de usuario y la capa de consistencia, que se suele añadir a un sistema para obtener beneficios tales como:

- Mayor control sobre las inserciones y consultas a la base de datos *core.*
- Coordinación entre todas las aplicaciones involucradas en el proyecto.
- Un añadido extra en la seguridad de acceso a la aplicación y a determinadas operaciones que manejen información sensible de los usuarios.
- Y una infraestructura más escalable al no tener toda la lógica de negocio incluida en una misma aplicación.

Para poder implementar esta capa hemos escogido una tecnología basada en JavaScript [18]. Este lenguaje es muy aceptado por los desarrolladores y, por tanto, existe mucha información y documentación sobre su uso en internet. Además, al ser tan usado, existen miles de soluciones a problemáticas comunes, incluso con librerías de otros desarrolladores que ayudarán a la mantenibilidad del código.

Uno de los principales problemas que presenta JavaScript frente a Java, es el tipado dinámico [19]. Esta característica, puede provocar errores no controlados que pueden cortar la ejecución de una aplicación. Una terminación abrupta de la aplicación es uno de los mayores inconvenientes de un middleware basado en el lenguaje JavaScript, aunque estos errores en la gran mayoría de ocasiones provienen de fallos en la programación.

Existen muchos controladores en el mercado que son capaces de detectar la caída de una aplicación de forma no deseada y frente a ello, volver a arrancarla de forma automática, pero para obtener una implementación más escalable y segura frente a errores, se va a utilizar una superclase de JavaScript llamada TypeScript [20].

TypeScript es un lenguaje construido en un nivel superior de JavaScript, es decir, es un lenguaje de programación que utiliza el propio motor y compilador que usa JavaScript, pero introduce algunos cambios con respecto al lenguaje original.

Los cambios más reseñables y por los que en este proyecto se usará TypeScript en lugar de JavaScript son:

- Introducción de elementos del lenguaje orientado a objetos. Gracias a ello podremos crear clases que recreen, dentro de nuestro código, objetos del mundo real. Además, se añade la posibilidad de crear interfaces, permitiendo así la herencia y la reutilización de código.
- Tipado estático [21]. Este tipo de tipado es ideal para un entorno crítico basado en JavaScript. Esto ofrece un mayor control sobre el código desarrollado y una mejor prevención de errores.

JavaScript fue creado para su ejecución dentro del motor de un navegador web, pero su rápida aceptación por los desarrolladores, su versatilidad y su rendimiento llevaron a los propios desarrolladores del lenguaje a escalar el lenguaje fuera del entorno web. Para ello se creó Node.js [22].

Utilizaremos Node.js porque es un entorno de tiempo de ejecución multiplataforma orientado a la capa de servidor basado en JavaScript siendo muy ligero y eficiente. El entorno ofrece un modelo de entrada y salida sin bloqueo controlado por eventos.

Para poder implementar esto, se crea un bucle instaurado en un socket de la máquina que alberga la aplicación. El bucle, se encuentra escuchando eventos de forma ininterrumpida para poder efectuar las operaciones requeridas por las solicitudes.

Gracias a ello, nuestra aplicación nunca se verá envuelta en un bloqueo de procesos ya que no existen los bloqueos dentro de una tecnología basada en este paradigma. En nuestro caso, los eventos serán solicitudes HTTP, que usaremos para poder intercambiar información entre la aplicación final de usuario y nuestra base de datos.

El *framework* Node.js ofrece un sistema asíncrono por lo que, por ejemplo, podremos ejecutar código mientras se ejecuta una petición HTTP no bloqueante. Por este motivo en concreto, nos hemos decantado por esta tecnología para montar nuestro middleware. Será muy común la realización de peticiones HTTP a las bases de datos de las que no necesitamos su respuesta para poder seguir con la ejecución del proceso. Esto nos ahorrará mucho tiempo de ejecución y por lo tanto una velocidad de respuesta al usuario mucho menor que si estuviéramos usando una tecnología síncrona. Sí es cierto que Java nos ofrece algunas funcionalidades asíncronas, pero la simplicidad en el desarrollo que ofrece Node.js es muy superior a la que nos ofrece Java.

3 Análisis del problema

Una vez mostrado todo el contexto tecnológico y realizado el análisis de las soluciones existes, se va a analizar y especificar los problemas a los que nos enfrentaremos en este proyecto.

En este capítulo de la memoria se definirán todos los requisitos que deberá cumplir el sistema y se describirán los casos de uso que se quieren cubrir. Estos casos de uso, a su vez, definirán los flujos que recorrerá un usuario que use la aplicación. Además, en este capítulo de la memoria se ha incluido un apartado específico para la definición del diagrama de clases del proyecto.

3.1 Especificación del problema

El análisis de requisitos en un proyecto es uno de los apartados más importantes porque se centra en la definición de las operaciones y funcionalidades que debe de reunir un sistema.

Reunir en un apartado todos los flujos y requisitos que debe de cumplir nuestro proyecto, ayuda a no perder el enfoque y la dirección que debemos de seguir en el desarrollo de la aplicación para que nuestra idea inicial y el resultado final sea lo más parecido posible.

3.1.1 Requisitos funcionales

Los requisitos funcionales recogen todos los objetivos relativos al usuario. Por lo que, en este apartado, recopilaremos todas las funciones que nuestro proyecto debe de cubrir para que la aplicación tenga el comportamiento que deseamos.

Estos requisitos, definen funciones del sistema de forma específica para que la solución final cubra todos los flujos que queremos representar en él.

A continuación, procederemos a definirlos:

Identificador	***Descripción***
F1	Permitir la creación de usuarios particulares
F2	Permitir la creación de usuarios empresariales
F3	Permitir la eliminación de usuarios
F4	Permitir la creación de eventos privados

F5	Permitir la creación de eventos públicos
F6	Permitir la asistencia a un evento
F7	Permitir la compra de entradas
F8	Permitir el envío de invitaciones a eventos privados
F9	Generar una entrada.
F10	Permitir la cancelación de asistencia a un evento
F11	Generar correos electrónicos personalizados
F12	Permitir la inserción de comentarios en los eventos
F13	Anexar documentos acreditativos en eventos privados
F14	Permitir la búsqueda de eventos

Tabla 1: Requisitos funcionales

3.1.2 Requisitos no funcionales

En otra parte, se encuentran los requisitos no funcionales. Estos requisitos están enfocados a recopilar todos los objetivos funcionales que deben de cumplir nuestras aplicaciones, siendo estos fundamentales, para ofrecer al usuario final un producto de calidad, seguro y eficiente con los recursos disponibles.

A estos requisitos, también se le llaman estándares de calidad teniendo mucho sentido porque en ellos siempre se habla sobre temas como tiempos de respuesta, seguridad o compatibilidades por enumerar algunos de los más comunes.

Para nuestro proyecto hemos establecido los siguientes:

Identificador	***Descripción***
AF1	La aplicación debe obtener tiempos de respuesta bajos en las peticiones
AF2	El código desarrollado en toda la aplicación debe de cumplir todos los estándares de legibilidad y eficiencia necesarios

AF3	Las pruebas realizadas sobre la aplicación deben de cubrir al menos el 98% de las líneas de código implementadas
AF4	Las rutas expuestas en los controladores HTTP que determinan las operaciones deben de ser totalmente descriptivas
AF5	La aplicación Back-end debe de hacer un uso eficiente de la memoria utilizada
AF6	Las peticiones SQL sobre la base de datos relacional deben de ser precisas y recuperar únicamente los datos necesarios para el correcto funcionamiento de las operaciones
AF7	Los datos sensibles del usuario deben de viajar siempre enmascarados
AF8	Los datos más sensibles del usuario se guardarán de forma cifrada en la base de datos
AF9	Las operaciones que manejen datos sensibles del usuario deben de estar especialmente protegidas

Tabla 2: Requisitos no funcionales

3.2 Casos de uso

En este aparto mostraremos de forma detallada cada caso de uso y los actores involucrados en la aplicación.

Para ello comenzaremos con una definición generalizada de los casos de uso que queremos cubrir para que, posteriormente podamos detallar los fundamentos de nuestro proyecto:

Un usuario de tipo empresa puede crear un evento con una cantidad limitada o ilimitada de entradas que los usuarios pueden comprar. El organizador puede elegir si las entradas son nominales, la edad mínima de acceso y el precio. Un usuario puede usar una acción llamada “Asistiré” por la que no comprará la entrada en nuestra aplicación, pero sí indica que asistirá al evento. Esta opción “Asistiré” puede ser desmarcable por el organizador del evento.

Un usuario de tipo particular puede crear un evento público o privado (en caso de ser privado, se podrá acceder solamente por invitación) con una cantidad limitada o ilimitada de entradas. El organizador puede elegir si las entradas son nominales, la edad mínima de acceso y el precio. Un usuario puede usar una acción llamada “Asistiré” por la que no comprará la entrada por nuestra aplicación, pero sí indica que asistirá al evento. Esta opción “Asistiré” puede ser desmarcable por el organizador del evento.

Un usuario podrá buscar eventos públicos cercanos a él, a su lugar de residencia en caso de no disponer de su ubicación en tiempo real o una ubicación a su elección.

Los usuarios podrán buscar eventos filtrando por tipo de evento. Dispondremos de eventos de tipo:

1. Festivos
2. Deportivos
3. Espectáculos
4. Convenciones

Un usuario organizador particular para un evento privado puede habilitar una opción que permita la visualización, por parte de los asistentes, la recaudación del evento por la venta de entradas acreditando así el precio de la entrada. Estas acreditaciones pueden ser texto simple, archivos o imágenes.

Los eventos cuentan con un apartado llamado “Comentarios” en los que los usuarios asistentes al evento pueden comentar y enviar opiniones sobre un evento. Estos comentarios y valoraciones se recogerán para elaborar una valoración global de un organizador.

El sistema debe de ser capaz de enviar correos electrónicos a los usuarios asistentes para enviar información sobre el evento al cual el usuario asistirá y deberá de pedir sus impresiones al usuario asistente.

3.2.1 Descripción de los actores

En el contexto de nuestro proyecto, definiremos como actores a los usuarios capaces de interactuar con la aplicación.

La aplicación está pensada para los siguientes tipos de actores:

1. **Organizador empresa:** Este rol está definido para usuarios con representación empresarial, que tendrá el acceso a la aplicación a un subconjunto reducido de los aplicativos desarrollados.
2. **Organizador particular:** Rol definido para usuarios físicos destinado a usuarios personales que quieren hacer uso de los flujos de creación de eventos tanto públicos como privados.
3. **Asistente:** Extensión del rol particular, que añade funcionalidades y da acceso a otros flujos de la aplicación como la adquisición de entradas.
4. **Sistema:** Actor que identifica a la aplicación que es capaz de interactuar con el usuario para enviar correos electrónicos.

3.2.2 Diagrama de los casos de uso

Los diagramas de uso son una forma gráfica de representar los flujos que debe de cubrir una aplicación o sistema. En este tipo de diagramas debemos representar los casos de uso de la aplicación y los actores que participan en ellos. En la Ilustración 1 podemos ver la representación de este diagrama. Este diagrama nos servirá de referencia para poder implementar todos los casos de uso dentro de nuestro proyecto.

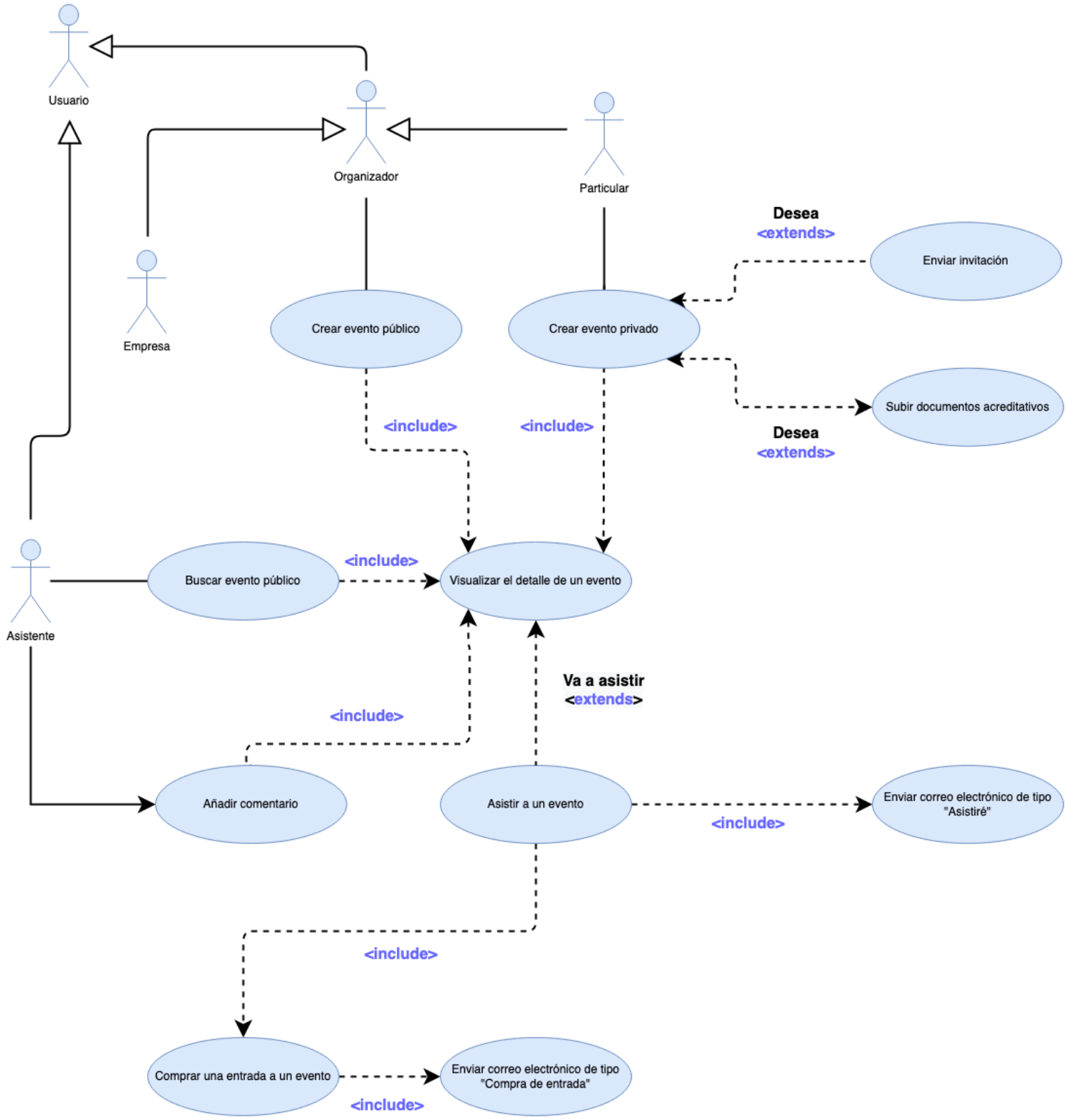


Ilustración 1. Diagrama de casos de uso

3.2.3 Descripción de los casos de uso

En este apartado vamos a describir cada caso. Mostrando en profundidad cada uno de ellos:

<table>
<tr><td>Nombre</td><td colspan="2">Crear evento público</td></tr>
<tr><td>Descripción</td><td colspan="2">Un usuario organizador crea un evento público. En este punto el usuario organizador puede habilitar diferentes o desactivar opciones:
Entradas nominales: Esta opción marca que la compra de entradas al evento, deben de incluir información personal acreditativa de la persona que va a asistir al evento con ella.
Opción “asistiré”: Esta opción habilita o deshabilita la opción “Asistiré” por la que un usuario con rol asistente, indica que asistirá al evento, pero no hará efectiva la compra de una entrada dentro de nuestra plataforma.
Edad mínima de acceso: El organizador puede determinar la edad mínima que deben tener los asistentes. Está opción no es requerida, por lo que el organizador puede no incluirla.
El organizador también deberá indicar el tipo de evento diferenciando entre:
Eventos festivos
Eventos deportivos
Espectáculos
Convenciones</td></tr>
<tr><td rowspan="8">Secuencia de uso</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Estar autenticado en el sistema</td></tr>
<tr><td>2</td><td>Incluir título del evento</td></tr>
<tr><td>3</td><td>Incluir descripción del evento</td></tr>
<tr><td>4</td><td>Incluir localización del evento</td></tr>
<tr><td>5</td><td>Marcar el tipo de evento</td></tr>
<tr><td>6</td><td>Marcar o desmarcar la opción “Entradas nominales”</td></tr>
<tr><td>7</td><td>Marcar o desmarcar la opción “Asistiré”</td></tr>
</table>

	8	Incluir la edad mínima de acceso al evento
	9	Enviar el formulario
Postcondición	Se creará el evento en nuestra base de datos y por tanto visible para usuarios de tipo asistente.	
Excepciones	Formato erróneo (En cualquiera de los campos). Campo requerido no cumplimentado (En cualquiera de los campos requeridos).	

Tabla 3: Caso de uso de creación de un evento público

Nombre	Crear evento privado	
Descripción	Un usuario organizador crea un evento público. En este punto el usuario organizador puede habilitar diferentes o desactivar opciones: • Entradas nominales: Esta opción marca que la compra de entradas al evento, deben de incluir información personal acreditativa de la persona que va a asistir al evento con ella. • Opción "asistiré": Esta opción habilita o deshabilita la opción "Asistiré" por la que un usuario con rol asistente, indica que asistirá al evento, pero no hará efectiva la compra de una entrada dentro de nuestra plataforma. • Edad mínima de acceso: El organizador puede determinar la edad mínima que deben tener los asistentes. Está opción no es requerida, por lo que el organizador puede no incluirla. El organizador también deberá indicar el tipo de evento diferenciando entre: • Eventos festivos • Eventos deportivos • Espectáculos • Convenciones	
Secuencia de uso	**Paso**	**Acción**

	1	Estar autenticado en el sistema
	2	Incluir título del evento
	3	Incluir descripción del evento
	4	Incluir localización del evento
	5	Marcar el tipo de evento
	6	Marcar o desmarcar la opción “Entradas nominales”
	7	Marcar o desmarcar la opción “Asistiré”
	8	Incluir la edad mínima de acceso al evento
	9	Enviar el formulario
Postcondición	Se creará el evento en nuestra base de datos. Una vez dado de alta el evento privado, el usuario organizador podrá mandar las invitaciones al evento por las que los usuarios asistentes podrán acceder al acontecimiento.	
Excepciones	Formato erróneo (En cualquiera de los campos). Campo requerido no cumplimentado (En cualquiera de los campos requeridos).	

Tabla 4: Caso de uso de creación de un evento privado

Nombre	Enviar invitación.	
Descripción	Un usuario organizador de tipo particular una vez creado un evento privado, procederá a enviar invitaciones al propio evento.	
Secuencia de uso	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Creación de evento privado
	3	Enviar invitación

Postcondición	El usuario asistente aceptará o declinará la invitación. En caso de aceptarla, se dará de alta al usuario como asistente al evento. Dependerá de la configuración del evento el flujo de aceptación.
Excepciones	No aplica

Tabla 5: Caso de uso del envío de una invitación

Nombre	Subir documentos acreditativos	
Descripción	Un usuario organizador de tipo particular una vez creado un evento privado, puede subir archivos para acreditar el coste del evento.	
Secuencia de uso	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Creación de evento privado
	3	Subir archivos dentro del evento.
Postcondición	Un usuario asistente al evento, dentro del detalle del evento podrá visualizar o descargar los archivos que el usuario organizador ha subido.	
Excepciones	Formato de archivo no aceptado. Tamaño de archivo no aceptado.	

Tabla 6: Caso de uso de la subida de documentos acreditativos

Nombre	Buscar evento público	
Descripción	Un usuario asistente puede buscar un evento filtrando por alguno de los tipos disponibles dentro del sistema.	
Secuencia de uso	**Paso**	**Acción**

	1	Estar autenticado en el sistema
	2	Buscar un evento público
Postcondición	Acceso a la visualización del evento seleccionado por el usuario asistente.	
Excepciones	Evento no encontrado.	

Tabla 7: Caso de uso de la búsqueda de un evento público

Nombre	Visualizar el detalle de un evento	
Descripción	Un usuario asistente puede buscar un evento filtrando por alguno de los tipos disponibles dentro del sistema.	
Secuencia de uso *Desde la creación de un evento*	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Crear un evento
	3	Acceder al detalle de un evento
Secuencia de uso *Desde la búsqueda de un evento público*	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Buscar un evento público
	3	Acceder al detalle de un evento
Postcondición	El usuario asistente podrá ver el detalle completo del evento, incluir comentarios sobre el evento, marcar la opción "asistiré" y comprar la entrada del evento. El usuario organizador podrá ver el resultado del evento que ha dado de alta dentro del sistema. También podrá hacer modificaciones sobre él en este flujo.	
Excepciones	Evento no encontrado.	

Tabla 8: Caso de uso de la visualización de un evento

Nombre	Asistir a un evento	
Descripción	Un usuario asistente es miembro de la lista de asistentes a un evento.	
Secuencia de uso *Acceder vía invitación a un evento privado.*	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Aceptar invitación
Secuencia de uso *Acceder a un evento público.*	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Buscar un evento público
	3	Marcar la opción asistiré (en caso de estar activada esta marca)
Postcondición	El usuario desde ese momento indica al organizador del evento que asistirá al evento. Acceso al caso de uso "Enviar correo electrónico de tipo 'Asistiré'".	
Excepciones	Capacidad máxima alcanzada. No cumplimiento de la edad mínima requerida.	

Tabla 9: Caso de uso de asistencia a un evento

Nombre	Comprar una entrada a un evento	
Descripción	Un usuario asistente a un evento procede a comprar la entrada de un evento. Si se trata de una entrada nominal, el usuario deberá cumplimentar todos los datos relativos a la identificación del usuario.	
Secuencia de uso *Acceder vía invitación a un evento privado.*	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Aceptar invitación
	3	Comprar una entrada a un evento
Secuencia de uso	**Paso**	**Acción**
	1	Estar autenticado en el sistema

Acceder a un evento público.	2	Buscar un evento público
	3	Comprar una entrada a un evento
Postcondición	El usuario desde ese momento indica al organizador del evento que asistirá al evento. Acceso al caso de uso "Enviar correo electrónico de tipo 'Compra de entrada'".	
Excepciones	Capacidad máxima alcanzada. No cumplimiento de la edad mínima requerida.	

Tabla 10: Caso de uso de la compra de una entrada a un evento

Nombre	Añadir comentario	
Descripción	Un usuario asistente a un evento puede incluir un comentario dentro del detalle de un evento. Este comentario será recibido por texto y una calificación. Esa calificación solo se podrá introducir una vez el evento ya se haya producido.	
Secuencia de uso *Evento privado.*	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Aceptar invitación
	3	Visualizar el detalle de un evento
	4	Añadir comentario
Secuencia de uso *Evento público.*	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Buscar un evento público
	3	Visualizar el detalle de un evento
	4	Añadir comentario
Postcondición	Se incluirá un nuevo comentario dentro vinculado a un evento en concreto.	

Excepciones	Formato no aceptado. Comentario ofensivo.

Tabla 11: Caso de uso para añadir un comentario a un evento

Nombre	Enviar correo electrónico de tipo "Asistiré"	
Descripción	El sistema envía un correo electrónico con la información del evento y un archivo ICS para que el usuario pueda añadir el evento a su calendario.	
Secuencia de uso	**Paso**	**Acción**
	1	Estar autenticado en el sistema
	2	Buscar un evento
	3	Visualizar el detalle de un evento
	4	Marcar la opción “Asistiré”
Postcondición	Un usuario asistente recibe un correo electrónico relativo al evento en el que ha marcado la opción “Asistiré”.	
Excepciones	Evento no encontrado. Correo electrónico no encontrado.	

Tabla 12: Caso de uso del envío de un correo electrónico de tipo “Asistiré”

Nombre	Enviar correo electrónico de tipo "Compra de entrada"	
Descripción	El sistema envía un correo electrónico con la información del evento, un archivo ICS para que el usuario pueda añadir el evento a su calendario y la(s) entrada(s) adquirida(s) en la aplicación. En el caso de que el usuario ya haya recibido un correo electrónico de tipo “Asistiré”, solo se enviará un correo electrónico incluyendo la(s) entrada(s) adquirida(s) en la aplicación.	
Secuencia de uso	**Paso**	**Acción**

	1	Estar autenticado en el sistema
	2	Buscar un evento
	3	Visualizar el detalle de un evento
	4	Marcar la opción “Asistiré”
	5	Adquirir una(s) entrada(s)
Postcondición	Un usuario asistente recibe un correo electrónico al comprar una(s) entrada(s).	
Excepciones	Evento no encontrado. Correo electrónico no encontrado.	

Tabla 13: Caso de uso del envío de un correo electrónico de tipo “Compra de entrada”

3.3 Diagrama de clases

En este apartado, se va a definir el modelo de clases para la capa de negocio y en el que se va a basar la implementación de la base de datos.

La Ilustración 2 muestra este diagrama UML donde se representan las clases con los atributos que necesita, incluyendo las relacionales entre las distintas clases.

Se han añadido decoradores propios del modelo relacional para conseguir un modelo más cercano al diseño lógico-relacional en el que se desarrolla la base de datos del proyecto.

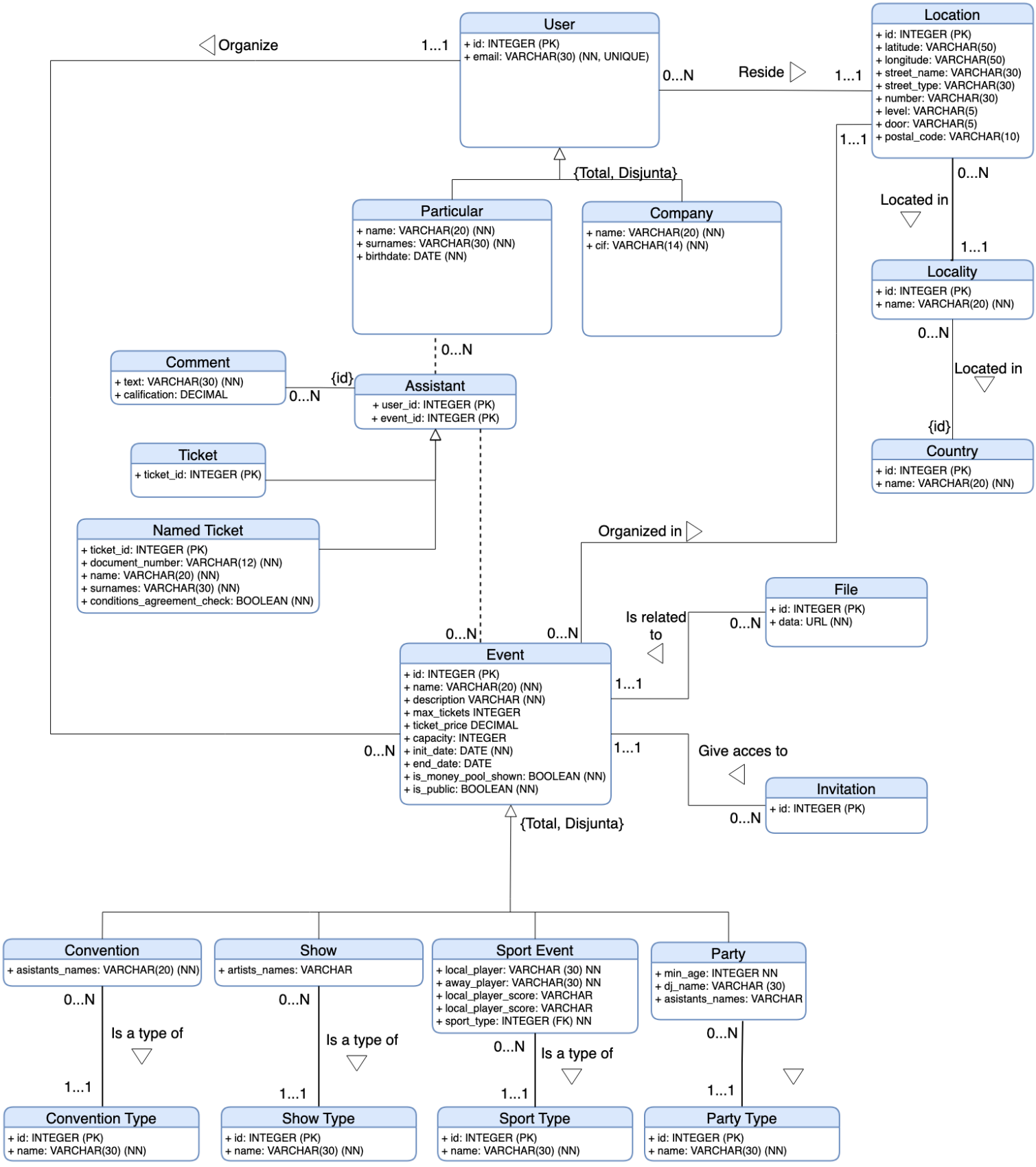


Ilustración 2. Relación UML de la Base de Datos Principal

4 Diseño de la solución

En este capítulo de la memoria se definirá el diseño completo de la aplicación. Este capítulo es el resultado de todo el análisis realizado en el capítulo anterior, por lo que, trataremos de diseñar un sistema capaz de cubrir todos los requisitos previamente definidos. Gracias a este capítulo seremos capaces de desarrollar la solución que buscamos de manera fiable y con una base asentada de conocimiento de las tecnologías involucradas en el proyecto.

4.1 Arquitectura del sistema

La arquitectura del software engloba a un conjunto de elementos por los que se compone un sistema informático para poder cumplir con los requisitos formulados.

En nuestro caso, utilizaremos una arquitectura dividida en capas. Esta arquitectura multicapa otorga al sistema un diseño con diferentes niveles de abstracción para así disponer de una serie de subsistemas que, correctamente conectados entre sí, formen la solución final.

Obtendremos muchas ventajas implementado una arquitectura por capas:

- Cada capa que compone el sistema se define de forma independiente del resto, por lo que, la inclusión de nuevas capas en el futuro será más sencilla que si las capas son totalmente dependientes entre ellas.
- Permite la utilización de tecnologías diferentes para cada uno de los niveles que componen un proyecto, pudiendo así optimizar los procesos involucrados en él.
- La inclusión de mejoras en las tecnologías utilizadas en una capa en concreto no afecta al resto y, por lo tanto, ahorrará tiempo de desarrollo en una supuesta mejora que afecte a todo el sistema.

Pero también es cierto que obtendremos ciertas desventajas como, por ejemplo:

- Posibles problemas en la velocidad de respuesta de la aplicación. Esto se debe a que una petición realizada por el usuario, en la mayoría de los casos, debe de recorrer varios niveles del sistema para obtener una respuesta. La solución a este problema recae directamente sobre el diseño e implementación del sistema para que los tiempos de respuesta sean los más bajos posibles.
- Aumento en el coste en infraestructura ya que, al disponer de más aplicaciones a implementar, se debe de emplear más tiempo en el desarrollo y emplear más infraestructura para albergar nuestros sistemas remotos.
- Un cambio en el nivel más bajo de la arquitectura puede ocasionar cambios en todos los niveles de la arquitectura. Esto obliga a elaborar pruebas específicas para todos los niveles de abstracción en los que se divide el sistema completo.

Como podemos ver existen ventajas y desventajas, pero para nuestro caso de uso y la importancia que le damos a la escalabilidad del proyecto, pudiendo introducir soluciones desacopladas a los sistemas ya existentes, se ha decidido introducir este paradigma para el diseño de la solución.

Por ello se va a proceder al diseño de la infraestructura que compone el sistema completo de nuestro proyecto *software*. Este diseño lo podemos representar a través de la Ilustración 3.

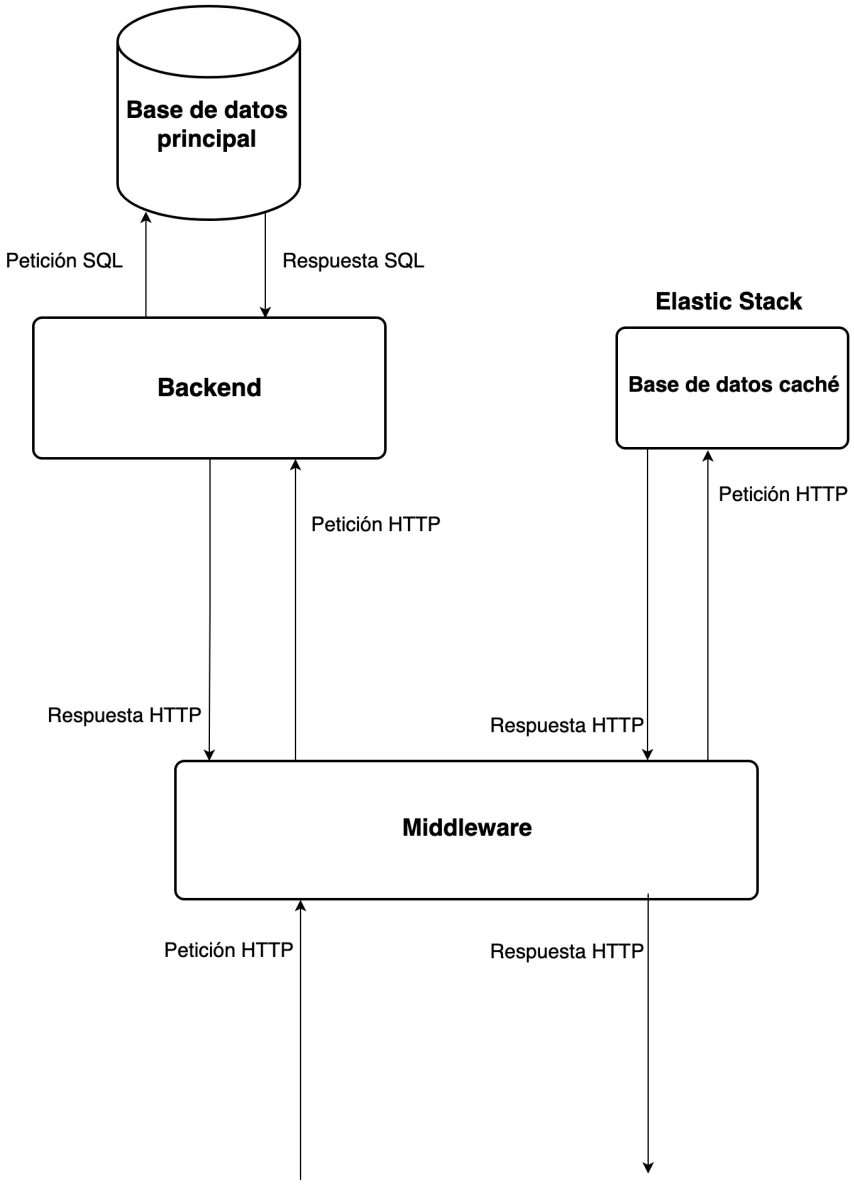


Ilustración 3. Diseño del sistema

4.2 Diseño detallado

En este apartado se entrada en detalle de la construcción de la arquitectura detallada para cada sistema que compone el sistema completo. Se trata de uno de los apartados más importantes de este proyecto, ya que, este diseño será el modelo por el que se debe de desarrollar cada sistema en específico.

A continuación, se definirán cada uno de los componentes que forman el sistema completo.

4.2.1 Estructura de la base de datos principal

La base de datos principal será la encargada de almacenar de forma segura toda la información de nuestra aplicación volcando en ella todos los datos de los usuarios y de los eventos.

Una vez ya definidos todos los casos de uso y todos los requisitos del sistema, vamos a definir el esquema lógico relacional. Para ello, podemos observar el modelo resultante en la Ilustración 4. Como podemos observar, se trata de una esquematización completa y detallada. Esto proveerá de una alta escalabilidad y de una gran consistencia entre la estructura de la base de datos y las especificaciones del proyecto. La escalabilidad la conseguimos implementado tablas como “Event” o “User”. Estas tablas permiten la creación de nuevos tipos de eventos o nuevos tipos de usuarios sin afectar al resto del sistema, ya que, mediante el diseño de la base de datos, estamos exigiendo unos datos mínimos que deben de contener un tipo de evento nuevo o un nuevo tipo de usuario.

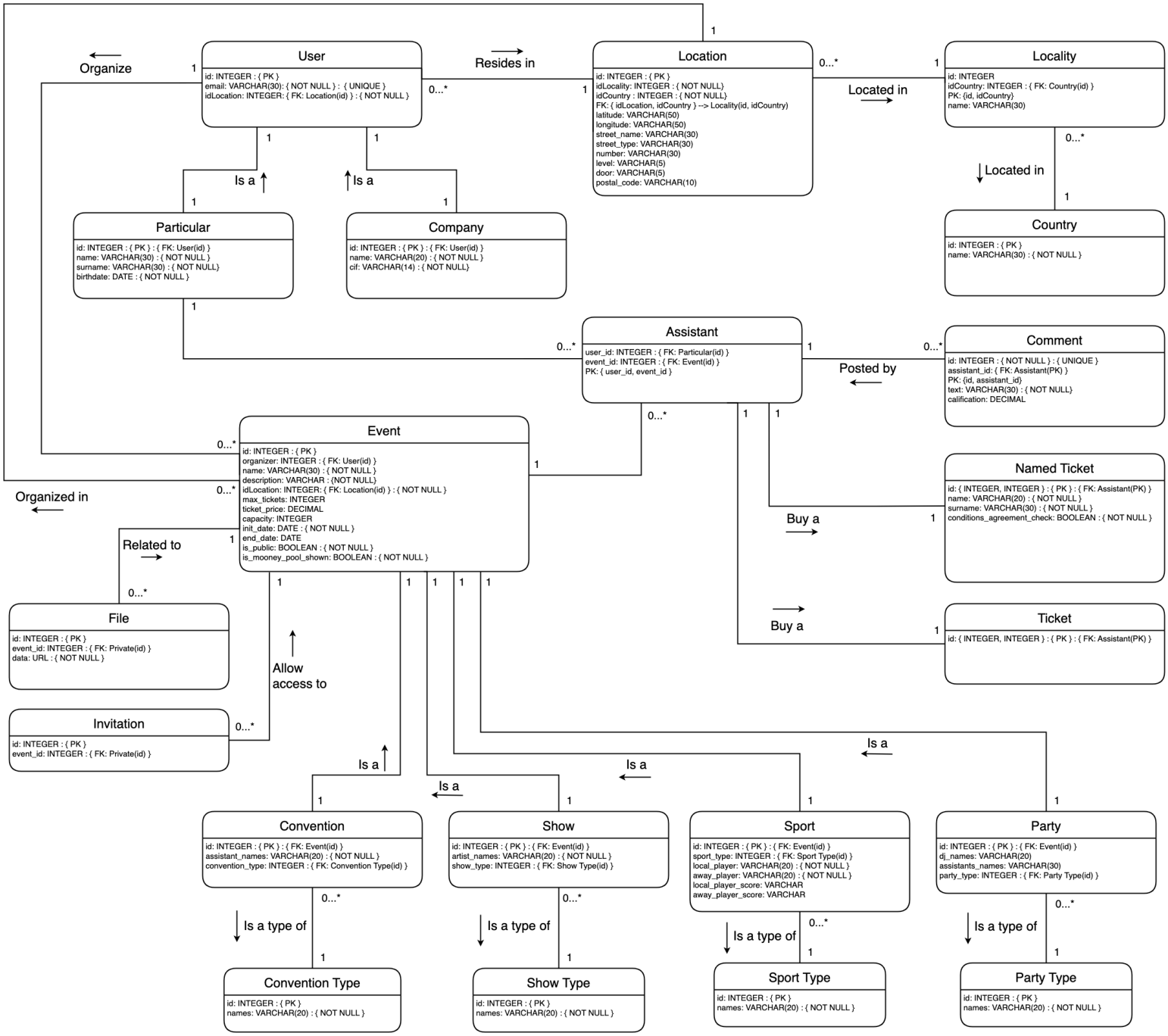


Ilustración 4. Diseño de la base de datos principal

Para su implementación, como ya hemos definido en el apartado “2.3 Base de datos relacional”, utilizaremos una implementación del esquema SQL para bases de datos relacionales, llamada PostgreSQL.

4.2.2 Estructura de la base de datos caché

Este tipo de base de datos se suelen utilizar para mostrar datos de la manera más rápida posible, por lo que, para intentar maximizar eficiencia y velocidad de respuesta se debe almacenar la menor cantidad de información posible.

Debido a esto, se ha decidido almacenar la información solamente de los eventos, utilizando esta información almacenada para obtener una respuesta muy rápida y mostrar al usuario un portal inicial en el que poder ver todos los eventos a los que puede asistir, e implementar un sistema de buscador de eventos por texto.

Como se ha comentado anteriormente, se propone utilizar ElasticSearch. Este motor de búsqueda organiza sus estructuras de datos en índices, por lo que para insertar y consultar los eventos de la aplicación se ha~~n~~ de crear un índice que almacene todos ellos.

Para establecer una similitud y, por lo tanto, una sincronización entre la base de datos principal de la aplicación y esta base de datos caché, en este índice de eventos creado en la base de datos caché se almacenarán todos los eventos que a su vez estén contenidos en la tabla “Evento” de la base de datos principal de forma que se consiga una relación directa entre ellos. Para ello, el valor del parámetro “id” de la tabla “Event” de la base de datos relacional, debe de estar presente de alguna forma representado en la base de datos caché para así poder relacionar los eventos guardados en ambas bases de datos. En la Ilustración 5 podemos observar un ejemplo de las entidades que serán almacenadas en la base de datos caché.

```typescript
export interface BaseEvent {
    id?: number;
    name: string;
    description: string;
    maxTickets?: number;
    ticketPrice?: number;
    capacity?: number;
    initDate: string;
    endDate?: string;
    isPublic: boolean;
    isMooneyPoolShown: boolean;
    eventType: number;
    organizerType: number;
    organizer: Organizer;
    location: Location;
    assistance: number;
}

export interface Organizer {
    id: number;
    email: string;
    location: Location;
}

export interface Location {
    id: number;
    latitude?: string;
    longitude?: string;
    streetName?: string;
    streetType?: string;
    number?: number;
    level?: number;
    door?: number;
    postalCode?: number;
    localityId: number;
    localityName: string;
    countryId: number;
    countryName: string;
}
```

Ilustración 5. Ejemplo de declaración de entidades en TypeScript

ElasticSearch utiliza un formato JSON [23] para almacenar las entradas al índice. Por este motivo, mostramos el modelado en una notación propia de TypeScript llamada “interface” [24] que representa de una forma estandarizada un objeto JavaScript (JSON).

Como se puede observar en la Ilustración 5, esa relación directa entre la base de datos relacional y la base de datos caché se mantiene con ese parámetro “id” que representa la clave primaria de la base de datos principal. Además, podemos observar que la relación de correspondencia entre las tablas de las bases de datos es muy similar, salvo una entrada concreta el modelo “BaseEvent” denominada “assistance”. Este parámetro es calculado a partir de la tabla “Assistant” de la base de datos principal, para poder devolver directamente el número total de asistentes a un evento en concreto.

4.2.3 Estructura de la aplicación Back-end

Para el diseño de la aplicación back-end vamos a utilizar una arquitectura CQRS [25] (Command Query Responsabilty Segregation o en separación de comandos y consultas). CQRS es un diseño que divide el software en dos secciones:

1. *Command* (Comandos): Se entiende como un comando a todo proceso que altere el estado actual del sistema, es decir, en nuestro caso se atañe a toda aquella operación de inserción o modificación sobre la base de datos.
2. *Query* (Consultas): Como su propio nombre indica, toda aquella operación de consulta sobre el estado actual del sistema y que no modifique el estado de este.

Esta es una arquitectura ideal para nuestro caso de uso, ya que, esta aplicación será la encargada de realizar las operaciones CRUD [26] sobre la base de datos principal del proyecto ayudando así a la estabilidad de la aplicación. Con esta división una parte de la aplicación será la encargada de todas las lecturas a la base de datos y otra parte se encargará de mantener el sistema de forma consistente con las operaciones de creación, actualización y eliminación de los datos del sistema.

A su vez, en nuestro proyecto, estas dos secciones se van a dividir en tres capas:

1. Capa de acceso a datos: Es la encargada de realizar las peticiones a cada una de las tablas de la base de datos.
2. Capa de lógica: Será la encargada de coordinar las peticiones a la base de datos para que el estado de nuestro sistema sea totalmente consistente, es decir, que todas las reglas relacionales entre las tablas de la base de datos relacional se cumplan. Por ejemplo, añadir transaccionalidad [27] a los comandos ejecutados o hacer las consultas necesarias para que el objeto entidad contenga los atributos necesarios.
3. Controladores: Esta capa es la encargada de recibir la petición realizada a la aplicación, enviar la petición a la lógica de negocio que satisfaga la petición y si es necesario enviar una respuesta al cliente (en este caso se entiende como cliente a cualquier sistema autorizado que realice peticiones a esta aplicación). Además, para añadir más encapsulación entre capas, estas mismas capas solo se podrán comunicar entre ellas a través de interfaces, obteniendo así una abstracción completa de la implementación de cada proceso, ya sea de la capa de negocio o del acceso de datos.

La Ilustración 6 representa de forma gráfica el diseño de esta aplicación.

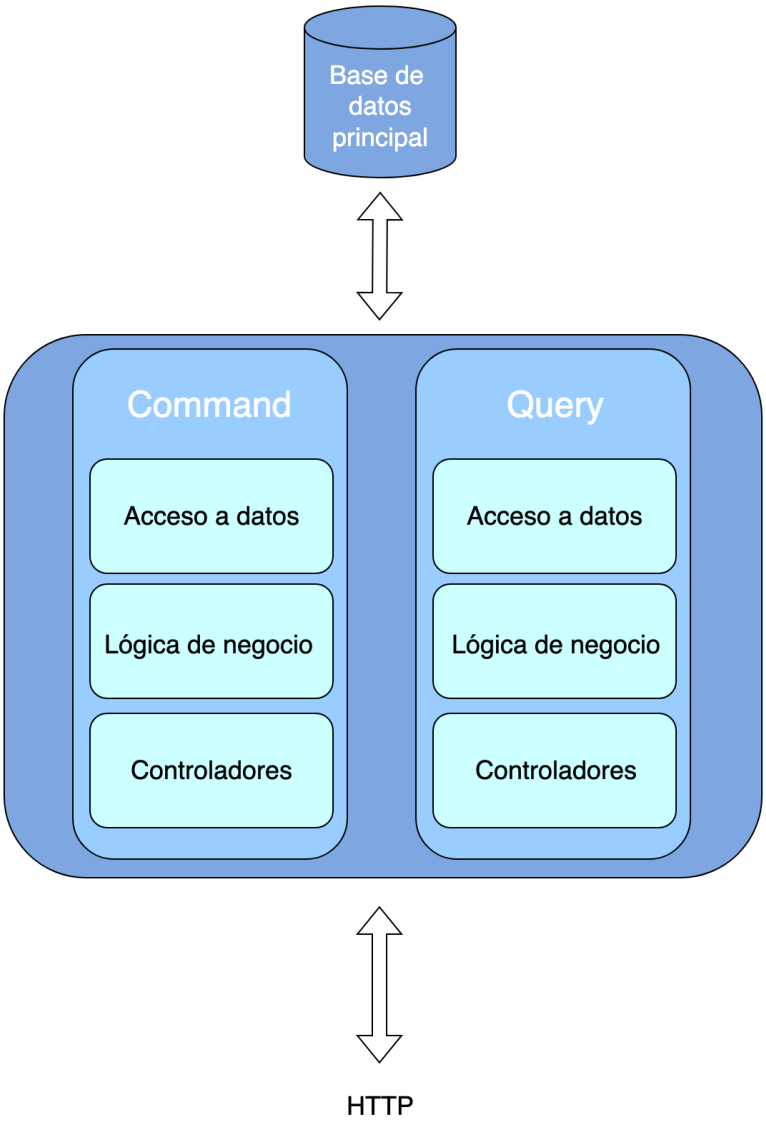


Ilustración 6. Diseño detallado de la aplicación Back-end

4.2.4 Estructura de la aplicación Middleware

La estructura de la aplicación Middleware debe tener una orientación clara de coordinación entre ambas bases de datos, la base de datos relacional y la caché. Además, la aplicación principal debe de ser la encargada de aplicar las lógicas de negocio y la responsable de la integración de posibles nuevos sistemas para mejorar el sistema en el futuro.

Atendiendo a estas necesidades, el diseño que se ha escogido para llevar a cabo la solución es una arquitectura hexagonal (o arquitectura cebolla) [28]. Una arquitectura hexagonal se basa en el esquema representado en la Ilustración 7 que ponemos a modo de ejemplo.

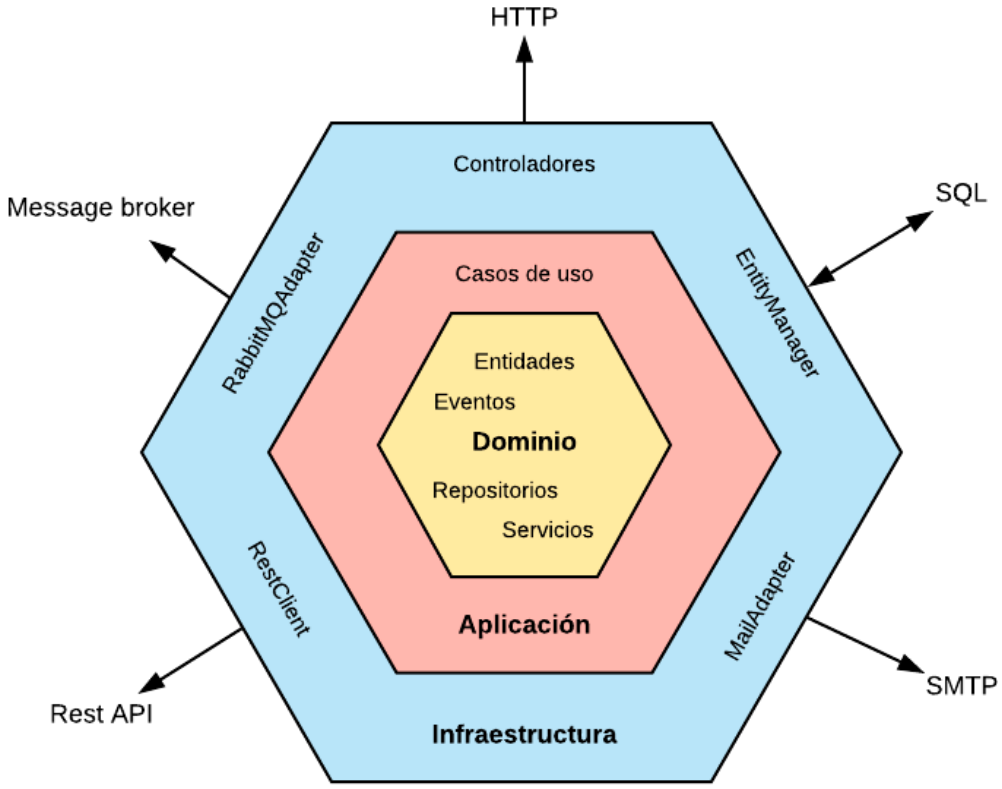


Ilustración 7. Ejemplo de Arquitectura Hexagonal

La arquitectura hexagonal propone una estructura como la siguiente:

- Infraestructura: Hace referencia a las conexiones externas a otros sistemas. Ejemplos de ello pueden ser una conexión a base de datos o una conexión HTTP con otro sistema.
- Aplicación: Engloba a todos los casos de uso y a la lógica de negocio del sistema.
- Dominio: Se incluye en ella todos los objetos que representen entidades del proyecto, código que transformen estas entidades o servicios que se usen en el sistema completo.

Se trata de una arquitectura altamente escalable, ya que, el dominio del sistema es algo que raramente se vaya a cambiar y es por esto por lo que se coloca en el medio del esquema. Por ello es fundamental identificar qué forma parte del dominio y qué no, porque un cambio en el dominio afecta directamente al sistema completo. Por este motivo, se coloca la capa de infraestructura en la capa más exterior, ya que suele ser la capa más volátil del sistema. Siempre surgen nuevas versiones o tecnologías que cumplen mejor nuestras necesidades, por lo que, colocando esta capa en la parte más superficial del sistema, somos capaces de desacoplar al resto de capas de cambios en componentes externos al proyecto.

Aplicando este modelo al diseño de nuestra solución, el esquema resultante puede verse a continuación en la Ilustración 8.

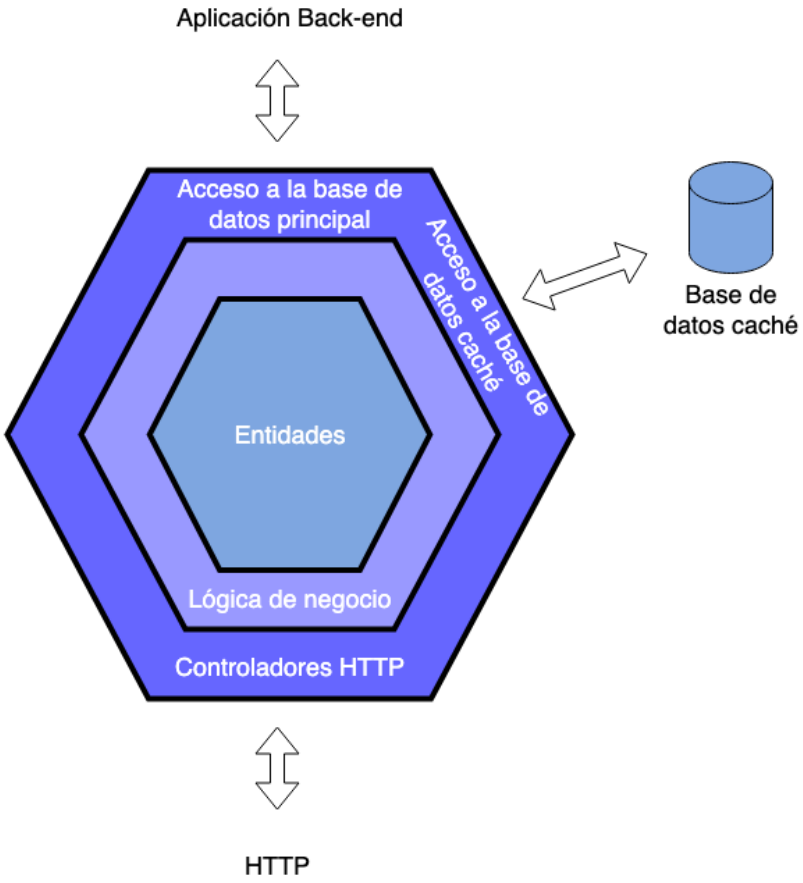


Ilustración 8. Arquitectura detalla del proyecto

En nuestro sistema, como se puede observar en la Ilustración 8, hemos dividido la aplicación en los siguientes módulos:

- Infraestructura:
  - Controladores HTTP: Este módulo compone todas las entradas de las peticiones a la aplicación.
  - Acceso a la base de datos principal: En este módulo se encontrarán las peticiones a la aplicación Back-end.
  - Acceso a la base de datos caché: El módulo encargado de manejar el estado de la base de datos de acceso rápido.
- Aplicación: Solo se ha incluido un módulo y se trata del módulo correspondiente a la lógica de negocio. En el coordinaremos ambas bases de datos y se implementarán los casos de uso de nuestra aplicación.
- Entidades: Para este módulo se ha reservado espacio para las interfaces que representan nuestras entidades y *helpers* (código reutilizable en todo el sistema).

Para demostrar la escalabilidad de este sistema, supongamos por ejemplo que en nuestro sistema se decide cambiar la base de datos caché de una implementación ElasticSearch a una implementación MongoDB (base de datos no relacional). Adaptando únicamente la implementación de "Acceso a la base de datos caché", tendríamos el sistema totalmente listo sin la necesidad de adaptar otros componentes del sistema como son: la lógica de negocio o el componente que contiene los controladores HTTP.

5 Desarrollo de la solución propuesta

Después de explicar el diseño del sistema, en este capítulo de la memoria se expondrá todo lo relativo al desarrollo tanto de las implementaciones de las bases de datos como de la solución creada para las dos aplicaciones que componen nuestro sistema.

5.1 Implementación de la base de datos principal

Como ya hemos comentado con anterioridad, para implementar la base de datos principal se ha decidido utilizar una base de datos relacional de tipo SQL, en concreto PostgreSQL. Para este proyecto se ha escogido la versión de PostgreSQL número 14, siendo esta la versión LTS (estable) más reciente publicada al momento de este desarrollo.

PostgreSQL nos ofrece tanto una aplicación de consola (cli) como una aplicación gráfica con la que gestionar bases de datos la base de datos. Con cualquiera de ambas aplicaciones es posible crear una base de datos, declarar tablas, introducir datos y todo lo necesario para poder trabajar con la base de datos. En nuestro caso se ha utilizado la aplicación gráfica denominada “PgAdmin” en su versión 4. La aplicación gráfica tiene muchas funciones, además de ofrecer una representación visual de nuestras tablas y sus propiedades, ofrece una herramienta “PSQL tool”. Esta herramienta, que es similar a la aplicación de consola que ofrece PostresSQL, es muy útil para configurar todos los roles y permisos de la base de datos, y otra herramienta llamada “Query tool” que se ha utilizado para introducir las consultas SQL con la que se ha elaborado la base de datos.

Para introducir los datos a las tablas, se ha utilizado “mockaroo” [29], una herramienta online que crea datos de prueba de forma automatizada. De esta forma, se ahorra mucho tiempo introduciendo datos de pruebas para las distintas integraciones, o para probar las restricciones y el funcionamiento de la propia base de datos.

La implementación de esta base de datos se ha basado en el esquema mostrado en la Ilustración 4. Vamos a mostrar cómo hemos realizado esta implementación mediante un ejemplo del desarrollo de las tablas de la base de datos relacional. La tabla “Event” es la encargada de almacenar la información común a todos los eventos de cualquier tipo. Esto se ve reflejado por la relación 1 a 1 señalada en la Ilustración 9.

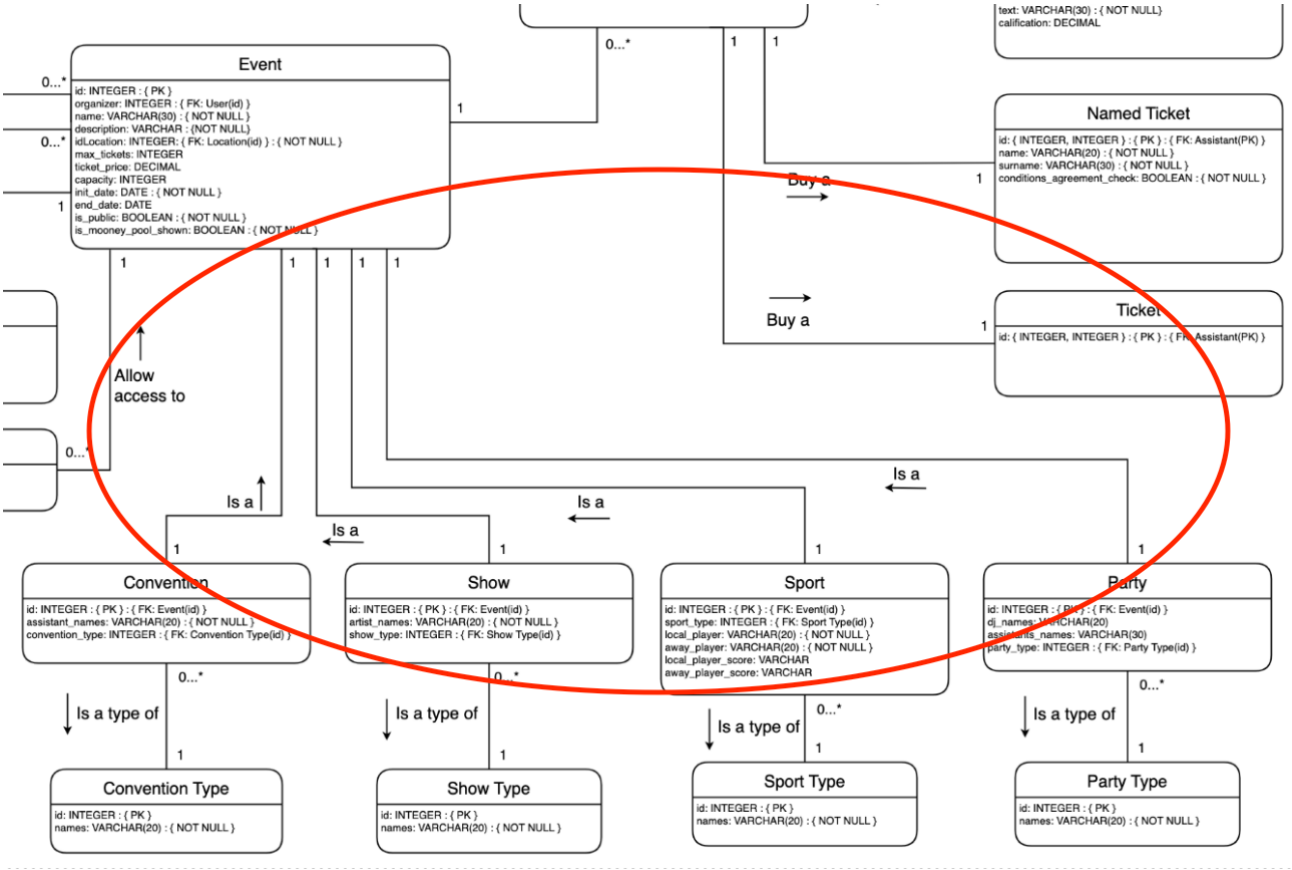


Ilustración 9. Relación 1 a 1 de la base de datos principal

Teniendo en cuenta lo señalado en la Ilustración 9, la tabla "*Event"* tendrá una representación SQL como se puede observar en la Ilustración 10.

```
5  CREATE TABLE IF NOT EXISTS public.event
6  (
7      id integer NOT NULL DEFAULT nextval('event_id_seq'::regclass),
8      description character varying COLLATE pg_catalog."default" NOT NULL,
9      location_id integer NOT NULL,
10     name character varying(100) COLLATE pg_catalog."default" NOT NULL,
11     max_ticket integer,
12     ticket_price numeric,
13     capacity integer,
14     init_date character varying(60) COLLATE pg_catalog."default" NOT NULL,
15     end_date character varying(60) COLLATE pg_catalog."default",
16     is_public boolean NOT NULL,
17     is_mooney_pool_shown boolean NOT NULL,
18     event_type integer NOT NULL,
19     organizer integer NOT NULL,
20     organizer_type integer NOT NULL DEFAULT 1,
21     CONSTRAINT pk_company PRIMARY KEY (id),
22     CONSTRAINT idlocation_fk_location_id FOREIGN KEY (location_id)
23         REFERENCES public.location (id) MATCH SIMPLE
24         ON UPDATE CASCADE
25         ON DELETE CASCADE,
26     CONSTRAINT organizer_fk_user_id FOREIGN KEY (organizer)
27         REFERENCES public.app_user (id) MATCH SIMPLE
28         ON UPDATE CASCADE
29         ON DELETE CASCADE
30 )
```

Ilustración 10. Declaración SQL de la tabla "Event"

La clave primaria (declarada como tal en la restricción "pk_event" de la Ilustración 10) es autogenerada por la declaración: `DEFAULT nextval('event_id_seq'::regclass)`. Esta declaración viene del lenguaje propio de PostgresSQL, que interpreta el valor de la secuencia event_id_seq y le aplica al valor 'id' el siguiente número a esta secuencia. La secuencia no es más que el acumulado total de valores introducidos en la tabla.

En SQL para declarar las restricciones por clave ajena, se deben declarar vía restricciones. En esta tabla en concreto se han declarado dos, la constraint idlocation_fk_location_id y organizer_fk_user_id que representan las siguientes relaciones señaladas en la Ilustración 11.

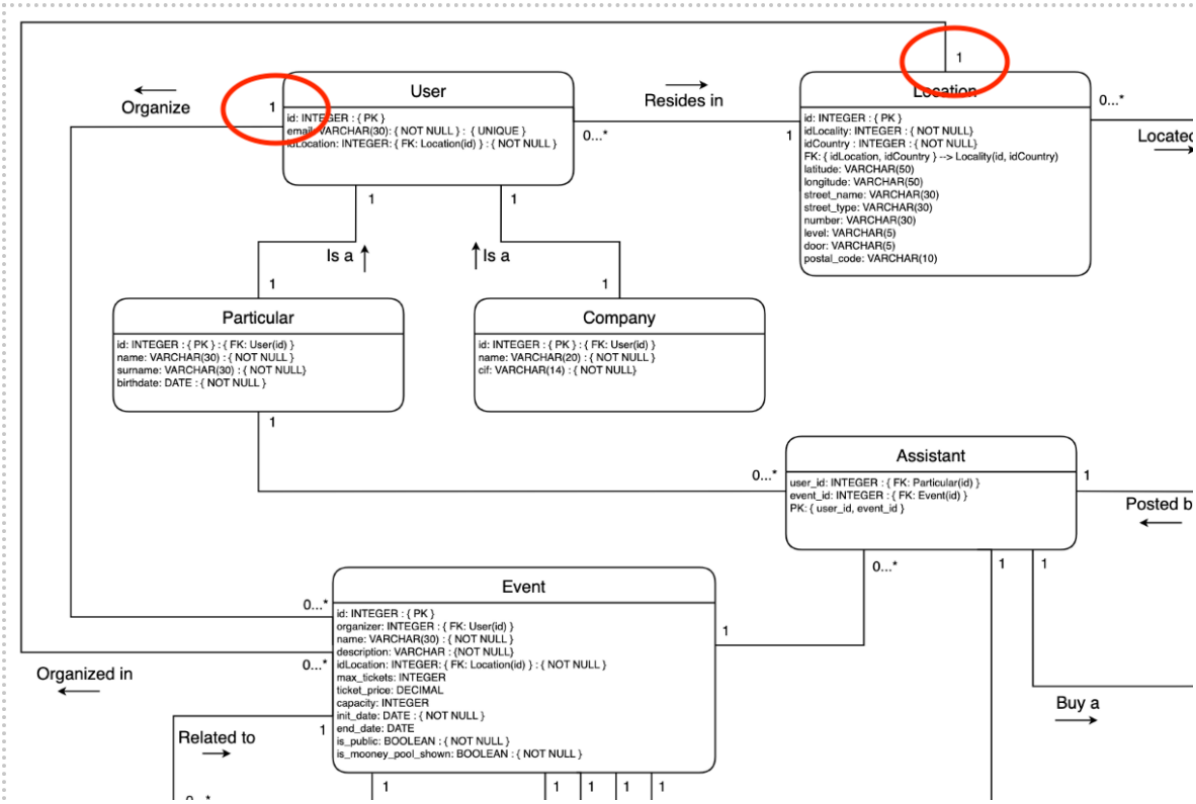


Ilustración 11. Relación 1 a muchos de la tabla "Event"

5.2 Desarrollo de la aplicación Back-end

La aplicación Back-end se ha definido como una aplicación basada en Spring Boot, utilizando Java como lenguaje de desarrollo. Utilizaremos la versión de Java 17 (JDK 17.0.3) siendo la versión LTS más reciente en la actualidad.

Para gestionar todas las dependencias a librerías externas se ha utilizado Maven [30]. Este gestor de dependencias es capaz de descargar e instalar todas las dependencias del proyecto mediante un archivo con extensión XML, agilizando así el proceso de desarrollo de la aplicación. En la Ilustración 12 podemos observar todas las librerías externas usadas en la aplicación.

Ilustración 12. Dependencias externas de la aplicación Back-end

Las librerías externas, mostradas en la Ilustración 12, usadas en esta aplicación es el siguiente:

- Swagger [31] y Open API [32]: Estas librerías se usan para poder crear una documentación interactiva de forma automática.
- JPA o *Java Persistence API* [33]: JPA es una librería que ofrece, mediante una interfaz autogenerada, métodos que realizan consultas SQL a través de código Java.
- JDBC o *Java Database Connectivity* [34]: Nos permite realizar conexiones a bases de datos.
- Hibernate [35]: La librería Hyperlink es capaz de crear un ORM [36] (*Object Relational Mapping* o *Mapeo Objeto-Relacional*) a partir de unas clases Java con una serie de anotaciones específicas.
- Spring Security [37]: Es una librería propia del entorno Spring que es capaz de ofrecer un control de acceso a la aplicación completa.
- MapStruct [38]: Una librería que es capaz de autogenerar *mappers*, es decir, métodos que trasladan datos de un objeto a otro, con la creación de interfaces o clases abstractas.
- Lombok [39]: Librería que a través de anotaciones permiten crear código Java muy repetitivo como, por ejemplo, los constructores vacíos de las clases.

Para este apartado, hemos creado una sección inicial para mostrar la estructura de paquetes y por la que seguimos el patrón CQRS, tres secciones en las que se hablarán de las capas en las que se han dividido la aplicación, una sección destinada a la seguridad de la aplicación y una sección final en la que se hablará sobre la documentación de la API.

5.2.1 Estructura de paquetes

En esta sección nos vamos a centrar en la estructura de paquetes de la aplicación para cumplir con la arquitectura declarada en la sección “4.2.3 Estructura de la aplicación Back-end”, la arquitectura CQRS.

En la Ilustración 13 se representa la división de la aplicación en 3 paquetes principales:

1. Paquete “command”: En este paquete se han incluido todos los flujos que alteren el estado de la base de datos.
2. Paquete “commons”: En este paquete se han incluido todas las entidades y modelos de la aplicación, las clases que mapean los datos de un modelo a otro y el sistema de login para los usuarios.
3. Paquete “query”: En este paquete se encuentran todas las consultas al estado del sistema.

De esta forma todos los controladores y todos los flujos de la aplicación quedarán divididos dependiendo del rol que tengan en el sistema.

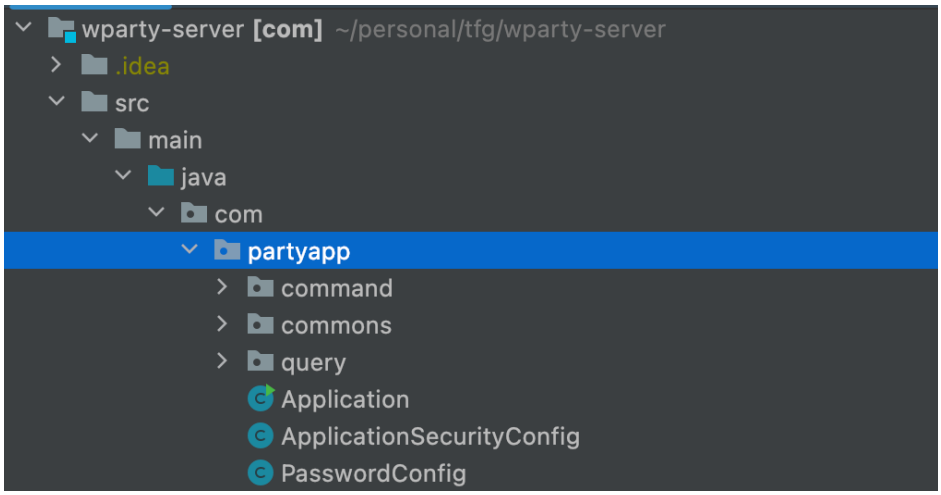


Ilustración 13. Estructura de paquetes de la aplicación Back-end

Dentro de los paquetes de “*command*” y “*query*”, como se parecía en la Ilustración 14, tenemos una estructura similar dentro de los paquetes. Los paquetes tienen en el primer nivel el descriptor de las tablas y los casos de uso de la aplicación.

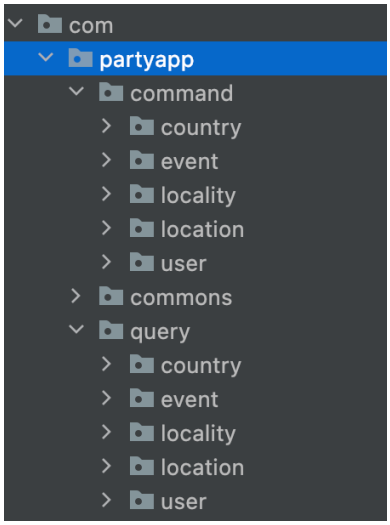


Ilustración 14. Separación en paquetes de la aplicación Back-end divididos por tabla de la base de datos principal

A partir de ahí existen dos tipos de paquetes: Los paquetes que engloban tablas con relaciones 1 a 1, es decir, tablas "padre", y los paquetes que contienen tablas autodescriptivas o tablas que no tienen una clase "padre".

El subnivel de los paquetes que contienen flujos relativos a tablas que no son autodescriptivas, contienen paquetes similares a los paquetes de primer nivel. En la Ilustración 15 podemos observar esta jerarquía, además, los paquetes señalados en la Ilustración 15 sí se refieren a las tablas de la base de datos principal de manera específica. Para ello, se ha tenido en cuenta que siga una estructura similar a las tablas autodescriptivas.

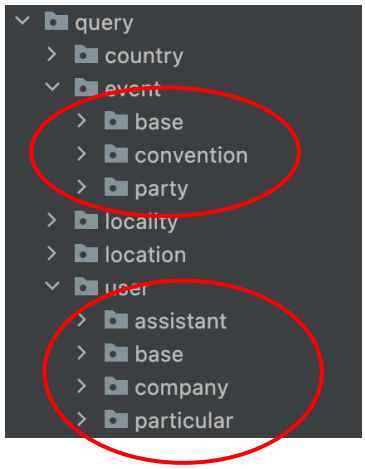


Ilustración 15. Paquetes que representan tablas no autodescriptivas

El siguiente subnivel a estos paquetes autodescriptivos, sí hace referencia a la estructura que se ha descrito en el apartado "4.2.3 Estructura de la aplicación Back-end". La Ilustración 16 es un ejemplo de ello.

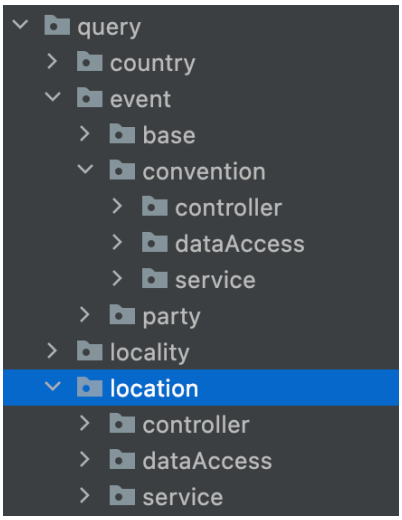


Ilustración 16. Estructura de paquetes que representan la separación entre capas de la aplicación Back-end

5.2.2 Controladores

Los controladores de la aplicación son los encargados de enrutar las peticiones HTTP que llegan a nuestro servidor hacia los flujos que resuelvan esta petición. El encargado de ello es Spring.

El *framework* provee de una serie de anotaciones por la que encaminar el tráfico hacia nuestro código Java como tal. La Ilustración 17 es un ejemplo de un controlador de nuestra aplicación:

```java
@RequestMapping("event/party")
@RestController
public class PartyEventQueryController implements IPartyEventQueryController {

    3 usages
    @Autowired
    private IPartyEventQueryService partyEventService;

    Antonio Giner
    @Override
    @GetMapping("/{id}")
    public PartyEventDTO getEventDetail(@PathVariable("id") Long id) {
        PartyEventDTO event = partyEventService.getEventDetail(id);
        return event;
    }

    Antonio Giner
    @Override
    @GetMapping("")
    public List<PartyEventDTO> getAllEvents() { return partyEventService.getAllEvents(); }

    Antonio Giner
    @Override
    @GetMapping("/assistance/{id}")
    public Integer getEventAssistance(@PathVariable(value = "id") Long id) {
        return partyEventService.getEventAssistance(id);
    }

}
```

Ilustración 17. Clase "PartyEventQueryController" de la aplicación Back-end

En esta imagen se pueden observar las siguientes anotaciones que identifican a la clase como un controlador de una aplicación:

- `RequestMapping`: Esta anotación indica al controlador de peticiones HTTP de Spring de que tiene que encaminar todo el tráfico de peticiones que accedan a la ruta relativa ("evento/party" en el ejemplo) a esta clase.
- `RestController`: Además de informar al inversor de dependencias de Spring de la existencia de esta clase como disponible para inyectar su dependencia, aporta funcionalidades típicas de una API REST (como por ejemplo transformar un objeto enviado en el *body* de una petición al objeto correspondiente).

En la imagen también aparecen otras anotaciones como `@GetMapping` que, a partir de la ruta relativa declarada en la clase, indica que a este método se accede vía una petición con HTTP con el verbo *GET* y la continuación de la ruta relativa debe de ser la declarada en esa anotación. Se pueden declarar también valores variables en la ruta usando la anotación `@PathVariable` que traslada como entrada del método el valor introducido en la petición.

El ejemplo, también muestra cómo se accede a la lógica de negocio de los distintos casos de uso. En este caso de uso en concreto, debe de encaminar hacia el servicio `partyEventService`, pero no accedemos directamente a su implementación, sino que, accedemos vía su interfaz. De esta forma garantizamos la abstracción entre los controladores y los servicios en este caso, pero en general en el proyecto, se ha seguido esta estructura para cumplir la abstracción entre capas del diseño.

Usando la anotación `@Autowired` de Spring, indicamos al inversor de dependencia de Spring de que la clase `IPartyEventQueryService` es instancia en esta clase y por lo tanto debe de satisfacer la dependencia en ella.

5.2.3 Servicios

Los servicios son los encargados de aplicar la lógica de negocio en esta aplicación. En esta aplicación en concreto la lógica de negocio es la encargada de mantener el estado del sistema de manera consistente. Para ello, vamos a describir una clase "*command*" mostrada en la Ilustración 18.

```java
@Service
public class PartyEventCommandService implements IPartyEventCommandService {

    3 usages
    @Autowired
    private IPartyEventCommandDA partyEventDA;

    1 usage
    @Autowired
    private IBaseEventCommandDA baseEventCommandDA;

    1 usage
    @Autowired
    private ILocationCommandService locationCommandService;

    5 usages
    @Autowired
    private CommandMapper mapper;

    1 usage  Antonio Giner
    @Override
    @Transactional
    public PartyEventDTO createEvent(PartyEventDTO event) {
        if (event.getLocation().getId() == null) {
            LocationDTO locationDTO = locationCommandService.saveLocation(event.getLocation());
            event.setLocation(locationDTO);
        }
        BaseEventCommandDAO baseEvent = baseEventCommandDA.createBaseEvent(
            mapper.toBaseEventCommandDAO(event)
        );
        event.setId(baseEvent.getId());
        partyEventDA.createPartyEvent(mapper.toPartyEventCommandDAO(event));
        return event;
    }
```

Ilustración 18. Clase "PartyEventCommandService" de la aplicación Back-end

Esta clase ya cuenta con más dependencias que los controladores, ya que deben de ser capaces de satisfacer todas las dependencias relacionales de las tablas afectadas. La única dependencia

instanciada que no está relacionada con las tablas de la base de datos es la dependencia de la clase `CommandMapper`.

Para no trasladar la información directamente de las tablas a los demás sistemas, se han creado objetos de tipo DTO. Un DTO (*Data Transfer Object* u Objeto de Transmisión de Datos) es un objeto, que como su propio nombre indica, es responsable de trasladar la información entre capas de un sistema. Se ha optado por esta solución porque, como se ha comentado en el diseño del proyecto, en realidad esta aplicación es parte de la capa de acceso a datos, por lo que, estos objetos que se retornan a la aplicación middleware deben de ser de este tipo.

Aquí es donde entra en acción la clase `CommandMapper`. Esta clase se ha utilizado para mapear datos desde la petición HTTP a la aplicación hacia un objeto válido para la base de datos. Ocurre del mismo modo, pero aplicando el flujo inverso. Se ha creado otra clase denominada `QueryMapper`, que atañe a los objetos devueltos por la base de datos principal que son devueltos por el controlador como respuesta a la petición a la aplicación. En ejemplo de ello es la Ilustración 19.

```
13  @Service
14  public class PartyEventQueryService implements IPartyEventQueryService {
        3 usages
15      @Autowired
16      private IPartyEventQueryDA partyEventDA;
17
        2 usages
18      @Autowired
19      private QueryMapper mapper;
20
        Antonio Giner
21      @Override
22      public PartyEventDTO getEventDetail(Long id) {
23          PartyEventDAO partyEventDAO = partyEventDA.getPartyEventDetail(id);
24          PartyEventDTO res = mapper.toPartyEventDetailDto(partyEventDAO);
25          return res;
26      }
27
```

Ilustración 19. Clase "PartyEventQueryService" de la aplicación Back-end

Para poder generar estos mapeos, se ha optado por la librería *MapStruct*. Esta librería proporciona mapeos entre objetos de forma automática, trasladando los valores de los parámetros comunes de un objeto a otro. La automatización de esta librería es precisa y conveniente en la mayoría de los casos, ayudando a la reducción del tiempo de desarrollo y a la mantenibilidad del código, ya que, no recae sobre nosotros la responsabilidad de la mantenibilidad de ese código. En el caso en que necesitemos algo de personalización, la librería ofrece como solución la utilización de la anotación `@Mapping`. Esta anotación, mostrada en la Ilustración 20, permite asociar parámetros de la clase destino a parámetros que no son nombrados de la misma forma en la clase fuente. Aunque si queremos tener aún más personalización, como se puede observar en la Ilustración 21, podemos aprovechar estos métodos que manejan los mapeos que son más simples y montar nuestros métodos personalizados de mapeo de datos.

```java
@Mapper(componentModel = "spring")
public abstract class QueryMapper {

    3 usages
    @Autowired
    protected DatabaseStringToListHelper listHelper;


    // Location Mappers
    6 usages  1 implementation  Antonio Giner
    @Mapping(
            target = "localityId",
            source = "source.locality.localityId"
    )
    @Mapping(
            target = "localityName",
            source = "source.locality.name"
    )
    @Mapping(
            target = "countryId",
            source = "source.locality.country.id"
    )
    @Mapping(
            target = "countryName",
            source = "source.locality.country.name"
    )
    public abstract LocationDTO toLocationDto(LocationDAO source);

```

Ilustración 20. Clase "QueryMapper" de la aplicación Back-end

```java
    // Party Event Mappers
    1 usage  1 implementation  Antonio Giner
    @Mapping(
            target="initDate",
            source = "source.initDate",
            dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSXXX"
    )
    @Mapping(
            target="endDate",
            source = "source.endDate",
            dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSXXX"
    )
    public abstract PartyEventDTO toPartyEventDto(BaseEventDAO source);

    2 usages  Antonio Giner
    public PartyEventDTO toPartyEventDetailDto(PartyEventDAO source) {
        PartyEventDTO res = toPartyEventDto(source.getEvent());
        List<String> diskJockeys = listHelper.getListFromString(source.getDiskJockeys());
        List<String> assistants = listHelper.getListFromString(source.getAssistants());
        res.setDiskJockeys(diskJockeys);
        res.setAssistants(assistants);
        res.setPartyTypeId(source.getPartyType().getId());
        res.setPartyTypeName(source.getPartyType().getName());
        return res;
    }

```

Ilustración 21. Método personalizado de mapeo de datos de la aplicación Back-end

Otra parte importante que comentar sobre los servicios es la anotación `@Transactional`. Los métodos implementados en los servicios de la aplicación, que se declaran con esta anotación, permiten a estos métodos que modifiquen el estado del sistema de forma más segura. Aquí es donde se ha utilizado la librería *Hibernate*.

Esta librería permite crear entidades que se asemejen a las tablas declaradas en la base de datos en un formato amigable para un desarrollo basado en Java, o lo que es lo mismo, un ORM [36] (*Object Relational Mapping* o *Mapeo Objeto-Relacional*). Gracias a esto, la aplicación es capaz de crear una virtualización de la base de datos a la que queramos conectarnos en tiempo de ejecución.

Las entidades son clases Java que representan las tablas y relaciones de nuestra base de datos. Esto se consigue gracias a anotaciones como las señaladas en la Ilustración 22. Estas anotaciones describen propiedades de nuestra base de datos como el nombre de la tabla ("`@Table`"), la cable primaria ("`@Id`") o la cardinalidad de una relación ("`@ManyToOne`").

```
9   @Entity
10  @Table(name = "app_user")
11  public class BaseUserDAO {
12      @Id
13      @GeneratedValue(strategy = GenerationType.AUTO)
14      @Column(name = "id", updatable = false, nullable = false)
15      private Long id;
16
17      @Column(name = "email", nullable = false)
18      private String email;
19
20      @ManyToOne
21      @JoinColumn(name = "location_id")
22      private LocationDAO location;
23  }
```

Ilustración 22. Clase "BaseUserDAO" de la aplicación Back-end

Gracias a la funcionalidad que ofrece *Hibernate*, los métodos que usen la anotación `@Transactional`, pueden usar el ORB de *Hibernate*, permitiendo que no se ejecute ninguna operación hasta que todas las restricciones sean comprobadas previamente. De esta forma, se ha asegurado que estos flujos dejen el estado del sistema en un estado consistente.

5.2.4 Acceso a datos

La capa de acceso a datos es la encargada de trasladar las peticiones de la aplicación middleware hacia la base de datos principal del proyecto. En las anteriores capas se ha trabajo con objetos DTO, en este caso necesitamos objetos que representen las tablas de la base de datos. Es por ello, que los

objetos que aparecen representados en esta capa son objetos DAO [40] (Data Access Object u Objeto de Acceso a Datos). Esto ayuda a la encapsulación entre capas, no manejando los mismos objetos entre capas.

Para conectar la base de datos principal a la aplicación Back-end se ha utilizado *JDBC*, una librería muy popular propiedad de Oracle, al igual que Java, por lo que se integra de manera perfecta en nuestro proyecto. Esta librería nos permite realizar conexiones a bases de datos con la simple integración de la librería en nuestro proyecto y la declaración de las variables de entorno oportunas. Como se puede observar en la Ilustración 23, con la simple declaración de la ruta en donde se encuentra desplegada nuestra base de datos, usando su protocolo propio "*jdbc*", en donde le indicamos la implementación de la base de datos SQL escogida, y dando una combinación válida de usuario y contraseña para acceder a la base de datos, ya estaríamos conectados nuestra base de datos.

```
spring.datasource.url=jdbc:postgresql://******/party_app_database
spring.datasource.username=*******
spring.datasource.password=*******
```

Ilustración 23. Datos de acceso a la base de datos principal desde la aplicación Back-end

Una ventaja que ofrece esta librería es que la conexión se realiza cuando se inicia la aplicación, es decir, en tiempo de arranque. Por lo que, si esa conexión no se realiza con éxito en el despliegue de la aplicación, la librería interrumpiría dicho despliegue, ahorrando errores posteriores en ejecución en el entorno productivo.

Una vez ya se ha conectado la aplicación a la base de datos principal, veremos cómo se realizan las consultas a esta base de datos. Para esta tarea utilizaremos la librería *JPA* que ofrece, mediante una interfaz, métodos que realizan consultas SQL de forma automática. Esta interfaz cubre los métodos más comunes como puede ser el de encontrar por clave primaria, retornar todos los datos de una tabla o guardar una nueva entrada en una tabla. Para ello, debemos declarar una nueva interfaz que extienda a la clase `JpaRepository` y posteriormente, introducir como clases genéricas la clase que define la tabla a la que hace referencia este repositorio y el tipado de la propiedad que hemos definido como clave primaria. Como ejemplo se puede observar la Ilustración 24.

```
2 usages  Antonio Giner
public interface LoginRepository extends JpaRepository<LoginDAO, Long> {
    1 usage  Antonio Giner
    @Query(value = "select * from login where email=:email", nativeQuery = true)
    public LoginDAO getUserByEmail(@Param("email") String email);
}
```

Ilustración 24. Ejemplo de clase que extienda la clase "JpaRepository" de la aplicación Back-end

Con esta declaración, *JPA* genera métodos predeterminados, como el método "`save`" que aparece en la Ilustración X. Pero también podemos crear métodos personalizados, como se observa en la Ilustración X. Usando la anotación `@Query`, dota al método de funcionalidad SQL, de esta forma, cada vez que se use este método, *JPA* generará la consulta SQL declarada en el `value` de la anotación.

Una buena práctica que hemos seguido como se puede ver en la Ilustración X es que para devolver errores en las consultas o modificaciones en la base de datos, no se devuelven códigos de error u objetos vacíos, sino que, se lanzan errores que cortan la ejecución del código facilitando así la legibilidad del código y la mantenibilidad de este. El lanzamiento de errores en estás casuísticas son altamente importantes para los métodos en la capa de servicios declarados como transaccionales (Ilustración 25). Estos métodos son capaces de detectar los errores y así poder deshacer las operaciones previamente ejecutadas, manteniendo el sistema en un estado consistente.

```
42      @Override
43      public AssistantEventCommandDAO addAssistantToEvent(AssistantEventCommandDAO request) {
44          try {
45              AssistantEventCommandDAO assistant = this.assistanceEventCommandRepository.save(request);
46              if (assistant != null) {
47                  return assistant;
48              }
49          } catch (Exception e) {
50              throw new ResponseStatusException(
51                      HttpStatus.INTERNAL_SERVER_ERROR,
52                      "Error saving an assistant to party event"
53              );
54          }
55          throw new ResponseStatusException(
56                  HttpStatus.INTERNAL_SERVER_ERROR,
57                  "Error saving an assistant to party event"
58          );
59      }
60
```

Ilustración 25. Método "addAssistantToEvent" de acceso a datos de la aplicación Back-end

5.2.5 Seguridad

La seguridad es un apartado muy importante en todo desarrollo de software y para ello hemos confiado en la gran fiabilidad de Spring Framework. Estos nos ofrecen una solución llamada Spring Security. Esta ofrece una solución sencilla para una problemática tan complicada como es la seguridad.

Spring Security es una librería propia del entorno Spring que es capaz de ofrecer un control de acceso a la aplicación completa. Nuestra aplicación, al ser una aplicación a la que se accede

mediante HTTP, en la que se va a integrar ~~con ella~~ una única aplicación (nuestro middleware), se ha optado por elegir una seguridad por usuario de aplicación.

Creando un usuario de aplicación, cuya contraseña no sea conocida, tendremos restringido el acceso a nuestros *endpoints* (rutas por la que acceder a nuestros servicios) solamente a las peticiones HTTP que incorporen en una cabecera "*Authorization*" el usuario de aplicación autorizado. El patrón que debe de seguir una cabecera de este tipo debe de ser el siguiente: En primer lugar, se debe de incluir el prefijo "*Basic*" y a continuación, se encriptan las credenciales (separadas por el carácter ':') usando un algoritmo Base64 [41].

```
Authorization: Basic base64{ usuario:contraseña }
```

Esta práctica añade muy poca seguridad, ya que, el algoritmo es fácilmente ejecutable, es decir, tiene muy poco coste operacional. Existen multitud de herramientas online que permiten descifrar una cadena de texto encriptada con este algoritmo. Esto es una convención de la comunidad para que, por lo menos, nuestros datos no viajen en texto plano por la red.

Para poder aplicar esta librería en nuestro proyecto, debemos de incluir la configuración que se puede observar en la Ilustración 26.

```java
@Configuration
@EnableWebSecurity
public class ApplicationSecurityConfig {

    2 usages
    private final PasswordEncoder passwordEncoder;

    Antonio Giner
    public ApplicationSecurityConfig(PasswordEncoder passwordEncoder) { this.passwordEncoder = passwordEncoder; }

    @Autowired
    private PasswordConfig passwordConfig;
    Antonio Giner
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .authorizeHttpRequests(
                        (auth) -> auth.anyRequest().authenticated()
                )
                .httpBasic(withDefaults());
        return http.build();
    }

    Antonio Giner
    @Bean
    public InMemoryUserDetailsManager userDetailsService() {
        UserDetails middlewareUser = User.builder()
                .username("partyAppMiddleware")
                .password(passwordEncoder.encode( rawPassword: "[redacted]"))
                .roles("APP")
                .build();
        return new InMemoryUserDetailsManager(middlewareUser);
    }
}
```

Ilustración 26. Clase encargada de la configuración de seguridad de la aplicación Back-end

La anotación `@Configuration` indica al inversor de dependencias de Spring que esta clase contiene métodos con la anotación `@Bean`, es decir, que son métodos usados globalmente en el proyecto.

Pero si se revisa nuestro proyecto, no hay ninguna referencia explícita a ningún método de esta clase. Aquí es donde entra la anotación `@EnableWebSecurity`. Esta anotación provee a todos los servicios de la aplicación de los filtros, es decir, los métodos anotados como `@Bean`, que han sido creados en esta clase. De esta forma, todos los servicios que ofrece la aplicación requieren esta autenticación.

Esta configuración es muy personalizable, ya que se pueden añadir filtros específicos para una ruta en concreto, pero en nuestro caso de uso no resulta necesario, ya que el objetivo es que la API sea consumida exclusivamente por el middleware de nuestra solución, y este middleware debe de tener acceso a todos los servicios porque este debe de ser capaz de modificar el estado del sistema por completo, siendo el encargado de la coordinación y de la lógica de negocio.

5.2.6 Documentación de la API

La documentación es una parte fundamental del desarrollo de software. Nos ayuda a tener la necesidad de explicar a otros equipos, que tengan la intención de integrar nuestros servicios en sus aplicaciones, de forma explícita. Hoy en día, la forma más sencilla de redactar la documentación de una API REST es utilizando un "*Swagger*" [31].

Un *Swagger* es una representación gráfica de los servicios que ofrece una API REST. Esta solución se ha vuelto tan popular porque, además de ofrecer una interfaz amigable, ofrece interactividad directa con la aplicación en cuestión. Un *Swagger* es capaz de realizar ~~de~~ peticiones HTTP contra la API que representa, pudiendo así tener una primera toma de contacto con la API antes de consumirla directamente en la integración en cuestión.

Aquí es donde entra en acción "*Open API*". *Open API* es una librería que permite con la simple declaración una anotación, señalada en la Ilustración 27, generar automáticamente una documentación completa de la aplicación en formato JSON. Eso sí, esta librería es específica para una aplicación desarrollada bajo Spring Framework.

```
11  @SpringBootApplication()
12  @OpenAPIDefinition(info = @Info(title = "WParty Server APP"))
13  public class Application {
14
        Antonio Giner
15      public static void main(String[] args) { SpringApplication.run(Application.class, args); }
18
19
20  }
```

Ilustración 27. Clase "Application" de la aplicación Back-end

```
{
    "openapi": "3.0.1",
    "info": {
        "title": "WParty Server APP"
    },
    "servers": [
        {
            "url": "http://localhost:8080",
            "description": "Generated server url"
        }
    ],
    "paths": {
        "/user/particular": {
            "get": {
                "tags": [
                    "particular-user-query-controller"
                ],
                "operationId": "getAllUsers_1",
                "responses": {
                    "200": {
                        "description": "OK",
                        "content": {
                            "*/*": {
                                "schema": {
                                    "type": "array",
                                    "items": {
                                        "$ref": "#/components/schemas/ParticularUserDTO"
                                    }
```


Ilustración 28. Objeto documentación en formato JSON de la aplicación Back-end

A su vez, Spring Boot es capaz de interpretar un objeto JSON creado por *Open API* para construir una página *Swagger* en formato HTML. En la Ilustración 29 se puede observar la interpretación por parte de un navegador web del archivo HTML generado por Spring.

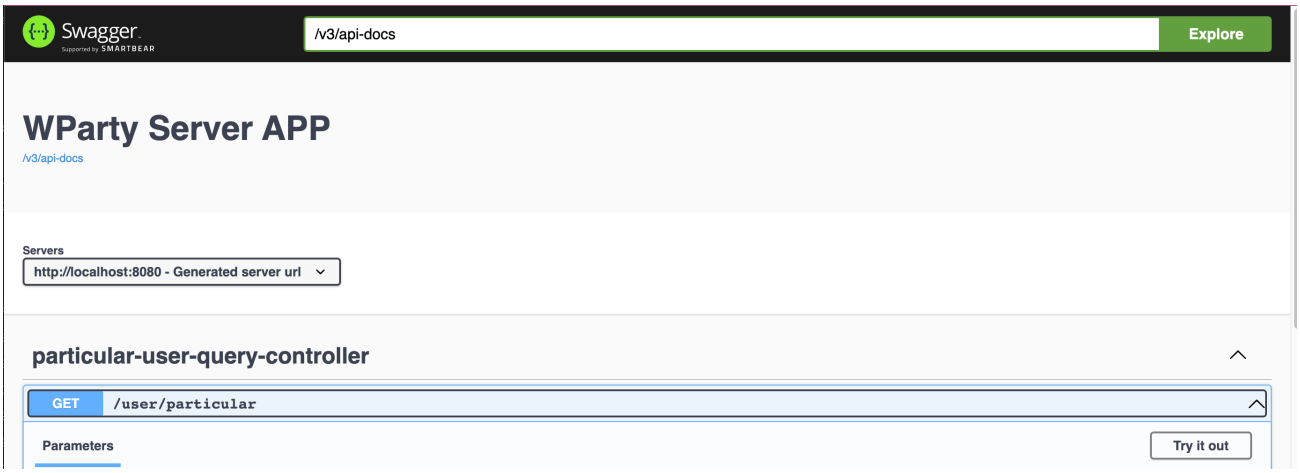


Ilustración 29. Documentación de la aplicación Back-end

Esta librería ofrece una solución *no code* (sin la necesidad de desarrollar código por el consumidor) para una tarea tan tediosa como lo es la documentación de nuestras APIs.

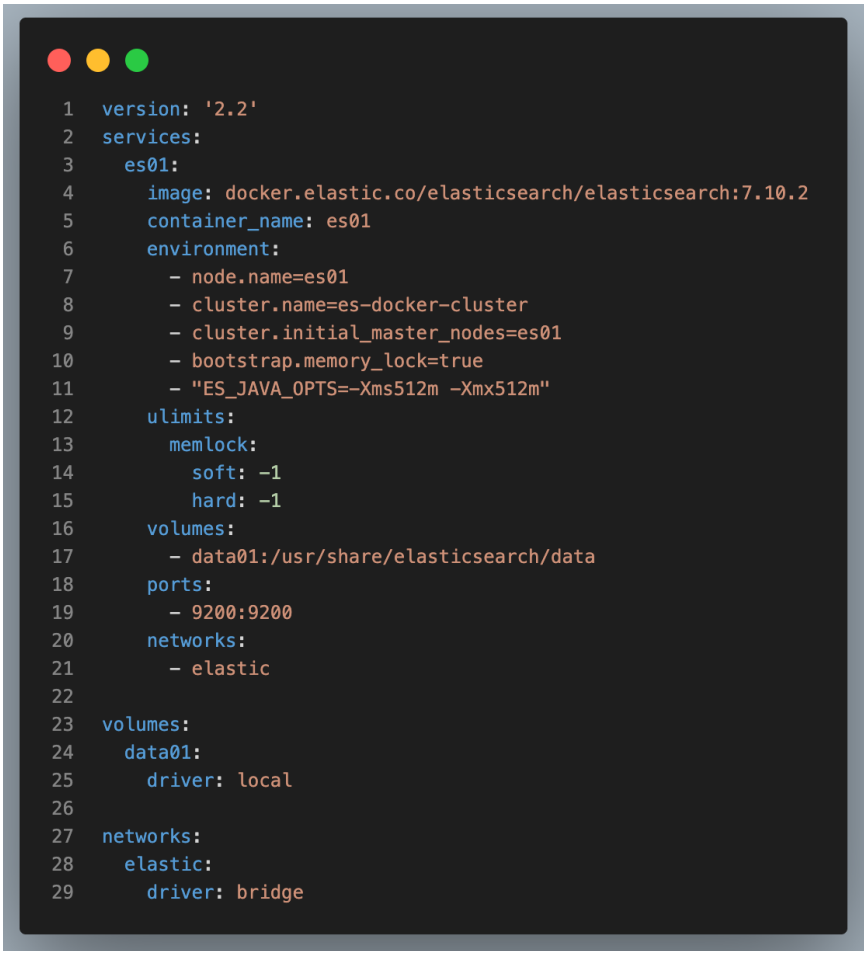
5.3 Implementación de la base de datos caché

Para la implementación de la base de datos no relacional del proyecto no se necesita instalar la tecnología ElasticSearch en local. Utilizando Docker [42], una tecnología que permite crear contenedores para poder albergar distintos servicios, podemos crear una instancia de una base de datos ElasticSearch en nuestra máquina local. Para ello debemos tener Docker instalado en nuestra máquina y utilizar el siguiente comando en la terminal:

```
$ ~ docker pull docker.elastic.co/elasticsearch/elasticsearch:8.3.3
```

El comando descarga por nosotros la tecnología necesaria para que nosotros de manera local podamos trabajar con ella. Ahora toca configurar el contenedor para que instancie una base de datos ElasticSearch.

Docker es capaz de interpretar un archivo llamado "*docker-compose.yml*" para configurar el contenedor y así poder correr diferentes programas, aplicaciones o en este caso una base de datos ElasticSearch. El archivo que debemos de utilizar para nuestro caso de uso es el que se muestra en la Ilustración 30.



```yaml
version: '2.2'
services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.10.2
    container_name: es01
    environment:
      - node.name=es01
      - cluster.name=es-docker-cluster
      - cluster.initial_master_nodes=es01
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - data01:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    networks:
      - elastic

volumes:
  data01:
    driver: local

networks:
  elastic:
    driver: bridge
```

Ilustración 30. Archivo "docker-compose.yml" generado para la configuración de la base de datos caché

Posteriormente a la construcción de este archivo, lanzando la siguiente secuencia de comandos en la misma ruta donde tienes almacenado el archivo de configuración, tenemos listo el entorno de trabajo:

- `$ ~ docker start up`: Se crea el contendor en donde se ubicará la base de datos.
- `$ ~ docker-compose up`: Se levanta una imagen de este contenedor y lo deja expuesto en el puerto declarado en el parámetro `ports` del archivo de configuración (Ilustración 30).

Una de las ventajas que tiene una base de datos Elastic es que las consultas, las modificaciones y la propia configuración se puede realizar mediante peticiones HTTP. Pongámoslo en práctica.

Las bases de datos ElasticSearch estructuran sus datos por índices. Estos índices pueden contener datos de todo tipo, depende del desarrollador que forma darles a los datos. En nuestro caso de uso concreto, solamente queremos almacenar los datos relativos a los eventos, por lo que se ha creado un único índice para almacenar todos los eventos. Para crear este índice debemos de ejecutar una petición HTTP como la siguiente:

```
curl –-location –request PUT ‘localhost:9200/event’
```

De esta forma tan sencilla, tenemos listo nuestro índice. Para poder insertar datos dentro de este índice debemos de usar peticiones con la siguiente estructura:

```
curl --location -g --request PUT 'localhost:9200/event/_doc/{eventId}'\

--header 'Content-Type: application/json'\

--data-raw ‘Objeto en formato JSON de un evento’
```

Como podemos apreciar, debemos de lanzar una petición HTTP usando el verbo PUT accediendo a una ruta que siga el patrón:

```
{ Índice donde almacenar los datos}/_doc/{ (Opcional) clave primaria }
```

La clave primaria es totalmente opcional, pero como hemos definido en la “4.2.2 Estructura de la base de datos caché”, la clave primaria de un evento en la base de datos caché debe de ser la misma que el evento al que hace referencia en la base de datos principal.

Para poder consultar los valores de todo un índice de la base de datos, debemos de ejecutar la siguiente petición:

```
curl --location --request GET 'localhost:9200/event/_search' \

--header 'Content-Type: application/json' \

--data-raw '{
```

```
    "query" : {
        "match_all" : {}
    }
}'
```

Y para poder consultar un objeto en específico teniendo su identificador se realiza de esta forma:

```
curl --location --request GET 'localhost:9200/event/_doc/20'
```

Como se puede apreciar todas las modificaciones y consultas se realizan de manera sencilla, simplemente hay que conocer la documentación de esta tecnología [43]. Por lo tanto, solo nos quería integrar todas estas peticiones HTTP a nuestra aplicación middleware que es la encarga de enrutar las peticiones a esta base de datos.

5.4 Desarrollo de la aplicación middleware

En esta parte perteneciente al desarrollo de la solución, vamos a describir el resultado final del diseño de la aplicación middleware usando el *framework* Node.js, utilizando el lenguaje JavaScript. Aunque como hemos comentado en las secciones introductorias, se ha utilizado la superclase TypeScript para el desarrollo.

¿Por qué se ha dicho que se ha usado TypeScript solo para desarrollo? Responderemos a esta pregunta en este apartado.

En la Ilustración 31 de puede observar lo siguiente:

- En la parte más interna de la aplicación podemos ver la combinación de JavaScript y TypeScript por la que se ha desarrollado la aplicación.
- Como parte del núcleo de la aplicación tenemos el *framework* Node.js, por el que se ha construido la aplicación, y el gestor de dependencias NPM.
- En la parte más externa, se puede observar las librerías externas usadas en esta aplicación.

Las dos librerías integradas en la aplicación son las encargadas lo siguiente:

- Recibir una petición a la aplicación y devolver una respuesta al usuario que la realice.
- Realizar peticiones tanto a la base de datos caché, como realizar las peticiones a la base de datos principal y, por lo tanto, a la aplicación Back-end.

La librería Express [44] se ha utilizado en la aplicación para dotar a la aplicación de funciones de servidor HTTP. Como se ha comentado en el apartado "4.2.4 Estructura de la aplicación Middleware" las peticiones a la aplicación se realizarán mediante este protocolo de red, por lo que, esta librería es fundamental para el proyecto. En nuestro caso de uso, la librería actuará como puerta de entrada y salida a las peticiones realizadas a la aplicación.

Por el contrario, se ha integrado una librería que sea capaz de realizar todo tipo de peticiones HTTP y además tenga una naturaleza asíncrona. Es por ello por lo que se ha decidido usar la librería "*axios*" [45] ya que cumple con todos los requisitos.

A continuación, tendremos dos apartados dedicados a la propia tecnología que compone la aplicación, mostradas en la Ilustración 31, y un último apartado en el que veremos cómo se ha desarrollado un flujo completo de una petición a la aplicación.

Ilustración 31. Tecnologías utilizadas en la aplicación middleware

5.4.1 Node.js y NPM

Esta librería es capaz de dotar al código propio del lenguaje JavaScript ejecutarse fuera del navegador. JavaScript está diseñado para ejecutarse en un entorno web, pero gracias a esta librería podemos crear aplicaciones que lo hagan, por ejemplo, en una máquina Unix [46], como es nuestro caso en un entorno de desarrollo y posteriormente en un entorno productivo.

Para poder gestionar las dependencias externas de la aplicación, del mismo modo que se ha usado Maven en la aplicación Back-end, Node.js tiene un gestor de dependencias propio, llamado NPM.

NPM [47] o Node Package Manager, es uno de los gestores de dependencias más utilizados en todo el mundo, debido también a la gran extensión que tiene el lenguaje JavaScript y la librería Node.js. Se trata, de un gestor de dependencias tremendamente potente, capaz de instalar y gestionar todas

las dependencias de una librería externa con un simple comando en la terminal. Por ejemplo, para iniciar el desarrollo de nuestra aplicación, escribiendo la siguiente línea en una terminal (eso sí, en el directorio elegido para este fin) NPM es capaz de crear un esqueleto básico de una aplicación Node.js:

```
npm init -y
```

Para ello crea un archivo llamado "`package.json`", creado para poder incluir todas las dependencias del proyecto. Este tiene un funcionamiento similar al que tiene un archivo pom.xml en una aplicación Maven (usada para crear la aplicación Back-end).

Aunque para poder ejecutar una aplicación Node.js, no nos basta solamente con este archivo, necesitamos generar un archivo "`package-lock.json`". Ejecutando el siguiente comando, NPM lo creará automáticamente:

```
npm install
```

Este archivo contiene todas las dependencias del proyecto, además de las dependencias internas de cada librería. Pero este comando, no solo crea este archivo, sino que, crea un directorio llamado "`node_modules`" que contiene todas las librerías listas para poder ser usadas en nuestro proyecto. Para nuestra aplicación hemos generado un archivo "`package.json`", representado en la Ilustración 32 en él se pueden resaltar las siguientes partes:

- Scripts: En este descriptor del JSON del archivo incluimos algunos alias [48] para poder ejecutar algunas operaciones, por ejemplo, si lanzamos en consola el comando "`npm run dev`", NPM es capaz de interpretar que en realidad queremos ejecutar "`nodemon src/app.ts`". Este enmascaramiento de comandos es muy útil para que si, por ejemplo, cambiamos la ejecución de "`nodemon src/app.ts`", para el resto de los desarrolladores del proyecto, este cambio será totalmente transparente. Pero también sirve también para poder encadenar comandos y englobarlos en un único comando, facilitando la fase de desarrollo e incluso la de construcción.
- Dependencias: Como se puede apreciar en la imagen, existen dos descriptores para depencias: "`devDependencies`" y "`dependencies`". Esto es útil para incluir dependencias que son solamente del entorno de desarrollo. En nuestro caso de uso, hemos incluido una serie de dependencias para el entorno de desarrollo, de las que cabe destacar las siguientes:
    1. Dependencias de buenas prácticas: Nos ayudan a escribir un código de mejor calidad, es decir, más legible. Para esta aplicación hemos escogido "*Prettier*" [49] y "*ESLint*" [50]. La inclusión de esta dependencia en el entorno de desarrollo obliga a la aplicación que en cada construcción de desarrollo compruebe si estas buenas prácticas se están cumpliendo.
    2. Nodemon [51]: Es una librería que permite el *live reload* [52] en un entorno de desarrollo.

```
{
  "name": "wparty-middleware",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "build": "tsc --project ./",
    "dev": "nodemon src/app.ts",
    "start": "node src/app.ts",
    "lint": "eslint"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@types/express": "^4.17.13",
    "@types/node": "^18.6.5",
    "eslint": "^8.21.0",
    "eslint-config-prettier": "^8.5.0",
    "eslint-plugin-prettier": "^4.2.1",
    "nodemon": "^2.0.19",
    "prettier": "^2.7.1",
    "ts-node": "^10.9.1",
    "typescript": "^4.7.4"
  },
  "dependencies": {
    "axios": "^0.27.2",
    "body-parser": "^1.20.0",
    "express": "^4.18.1",
    "rxjs": "^7.5.6"
  }
}

```

Ilustración 32. Archivo "package.json" de la aplicación middleware

5.4.2 Lenguaje de desarrollo

En nuestra aplicación middleware, al utilizar Node.js para construir nuestra aplicación, estamos obligados a usar JavaScript. Este es un lenguaje con un gran rendimiento y altamente aceptado por la comunidad, pero existe una gran problemática con el control del flujo del código.

Para mejorar el control del código, se ha añadido tipado estático a las variables declaradas y a los propios métodos implementados en nuestra aplicación usando la superclase TypeScript en el desarrollo.

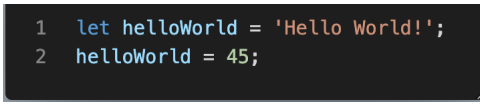
Hablamos de utilizar TypeScript en el desarrollo del código y no en un entorno productivo porque el código desarrollado, antes de crear la construcción de producción, es decir, antes de preparar la aplicación para ser utilizada por los usuarios finales, se realiza una construcción que hace una

traducción de código de TypeScript a JavaScript. Esto aporta una gran ventaja a la hora de predecir posibles errores en un entorno productivo. Para ello veamos un ejemplo.

```
6
7
8   let helloWorld: string;
9   helloWorld = 'Hello World!';
10
11  helloWorld = 45;
12
13
```

Ilustración 33. Ejemplo TypeScript de error de compilación

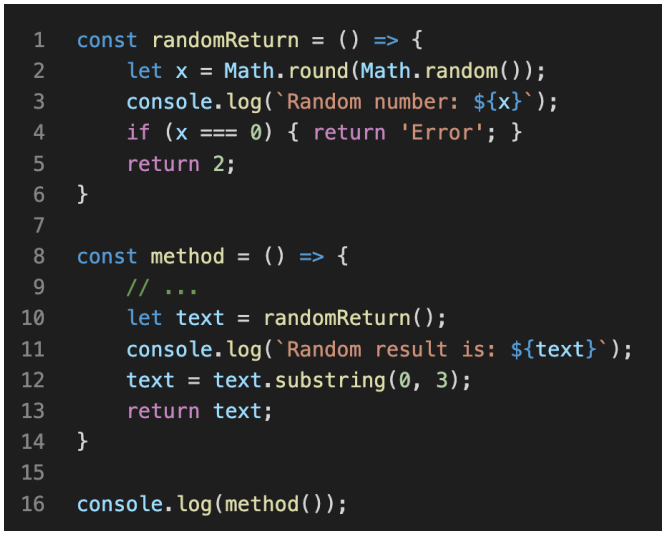
En la Ilustración 33 vemos como se ha realizado un tipado a la variable "`helloWorld`" como un *string* (cadena de texto). Posteriormente asignamos una cadena de texto a la variable y no existe ningún problema, pero finalmente decidimos que la variable tenga un valor numérico. Como se puede observar en la Ilustración 33, el propio editor ya nos avisa de que existe un error. Mientras que en la Ilustración 34 no existe ningún error porque en JavaScript existe el tipado dinámico [53] que lo permite.



```
1  let helloWorld = 'Hello World!';
2  helloWorld = 45;
```

Ilustración 34. Ejemplo JavaScript de tipado dinámico

En el código que se puede ver en la Ilustración 34, en concreto, no habría ninguna necesidad de utilizar la superclase TypeScript. Pero imaginemos un método como declarado en la Ilustración 35.



```
1  const randomReturn = () => {
2      let x = Math.round(Math.random());
3      console.log(`Random number: ${x}`);
4      if (x === 0) { return 'Error'; }
5      return 2;
6  }
7
8  const method = () => {
9      // ...
10     let text = randomReturn();
11     console.log(`Random result is: ${text}`);
12     text = text.substring(0, 3);
13     return text;
14 }
15
16 console.log(method());
```

Ilustración 35. Ejemplo de posible error en tiempo de ejecución JavaScript

Si no se ha testeado bien, en un entorno productivo podría ocurrir un error como el representado en la Ilustración 36.

```
→  helloWorld node index.js
Random number: 0
Random result is: Error
Err
→  helloWorld node index.js
Random number: 1
Random result is: 2
/Users/aginer/trash/helloWorld/index.js:14
    text = text.substring(0, 3);
                ^

TypeError: text.substring is not a function
    at method (/Users/aginer/trash/helloWorld/index.js:14:17)
    at Object.<anonymous> (/Users/aginer/trash/helloWorld/index.js:18:13)
    at Module._compile (node:internal/modules/cjs/loader:1105:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1159:10)
    at Module.load (node:internal/modules/cjs/loader:981:32)
    at Function.Module._load (node:internal/modules/cjs/loader:822:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:77:12)
    at node:internal/main/run_main_module:17:47
```

Ilustración 36. Error de ejecución JavaScript

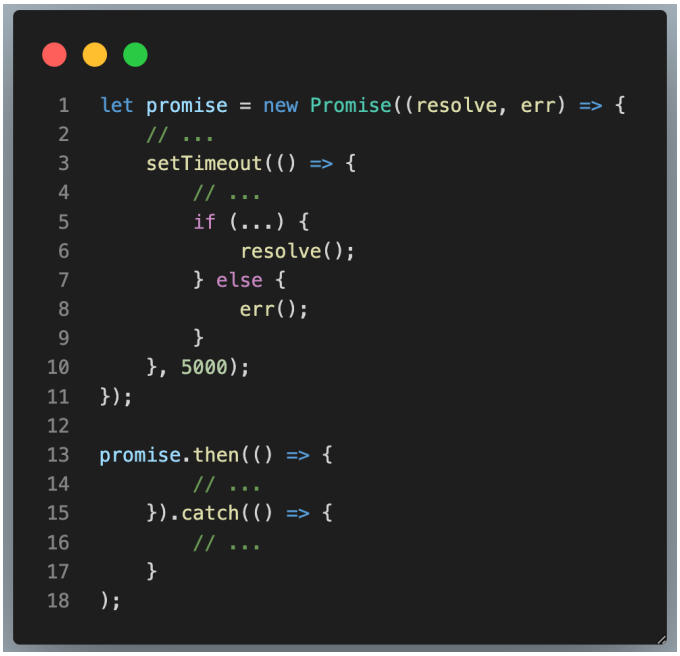
En la primera ejecución de la Ilustración 36, la ejecución ha sido satisfactoria. Pero en la segunda ejecución, nuestra aplicación ha lanzado una excepción por acceder a un método propio de un objeto de tipo *string* a una variable asignada como numérica. En este ejemplo tan simple, estos errores se detectan muy rápido, pero en un código más complejo o en un flujo raramente accesible y no probado, estos errores pueden ocurrir frecuentemente. Un gran problema que tiene una implementación Node.js es que cuando se detecta un error no controlado, la aplicación cierra su ejecución por completo y por tanto deja de dar servicio a los usuarios.

Existen múltiples soluciones para que, ante un error no controlado, la aplicación vuelva a arrancar de forma automática. Usando la superclase TypeScript, los errores no controlados se reducen y por lo tanto obtenemos un código, no solo más fiable, sino que, obtenemos un código más seguro y en algunos casos mucho más legible.

Dejando de lado los defectos que puede tener JavaScript y que intentamos mejorar usando TypeScript, hablemos sobre una de las grandes ventajas que ofrece este lenguaje y es la posibilidad implementar código asíncrono.

El código asíncrono suele ser un arma de doble filo, ya que, añade complejidad al flujo de ejecución. Para ayudar a los desarrolladores JavaScript ha creado las promesas y la cláusula "*async*/*await*".

Las promesas son objetos JavaScript que describen tanto la terminación correcta de la ejecución de un código asíncrono, como la terminación abrupta de él. Un ejemplo de ello pueden ser las peticiones que se suelen implementar a través de librerías. Estas librerías suelen usar objetos de este tipo para consumir el resultado de una petición desde el lado del cliente (en este caso nuestra aplicación). La sintaxis que sigue la resolución de una promesa se puede observar en la Ilustración 37. De esta forma, cuando un código asíncrono finaliza, a través de estos métodos podemos actuar ante el resultado obtenido.

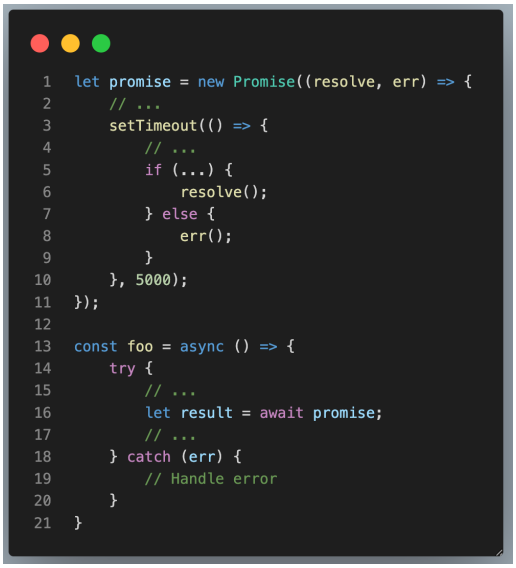


```
let promise = new Promise((resolve, err) => {
    // ...
    setTimeout(() => {
        // ...
        if (...) {
            resolve();
        } else {
            err();
        }
    }, 5000);
});

promise.then(() => {
        // ...
    }).catch(() => {
        // ...
    }
);
```

Ilustración 37. Sintaxis de control del flujo mediante promesas JavaScript

Pero esto crea un código poco limpio por lo que los desarrolladores del lenguaje JavaScript implementaron las cláusulas *async*/*await*. Observando la Ilustración 38, se puede apreciar un código más limpio que el código mostrado en la Ilustración 37, e incluso incluye una ventaja frente a la anterior implementación y es que, informamos a los posibles consumidores de que este método contiene código asíncrono. Para nuestra aplicación se ha decido usar las cláusulas *async*/*await* con el objetivo de obtener un código limpio y de fácil lectura.



```
let promise = new Promise((resolve, err) => {
    // ...
    setTimeout(() => {
        // ...
        if (...) {
            resolve();
        } else {
            err();
        }
    }, 5000);
});

const foo = async () => {
    try {
        // ...
        let result = await promise;
        // ...
    } catch (err) {
        // Handle error
    }
}
```

Ilustración 38. Sintaxis de las cláusulas async/await en JavaScript

5.4.3 Flujo de las peticiones

En este subapartado comentaremos el flujo completo que recorre una petición realizada a la aplicación middleware. Para ello, tomaremos como ejemplo una petición de creación de un evento de tipo “Fiesta”:

```
curl --location --request POST \

'localhost:3000/wparty-middleware/1.0/event/party' \

--header 'Content-Type: application/json' \

--data-raw ' { Party Object } '
```

Como puerta de entrada de peticiones HTTP, se encuentra la librería Express, enrutando la petición al fichero “app.ts”, siendo este el lanzador de nuestra aplicación. En la Ilustración 39 se puede apreciar como instanciamos a Express a través de su constructor. De esta manera, se ha construido un servidor web que se comporta como una API REST. También podemos observar el puerto escogido para la recepción de las peticiones y un método “`routerApi`”.

```
1  import express from 'express';
2  import { routerApi } from './rounting';
3  import bp from 'body-parser';
4
5  const app = express();
6  const port = 3000;
7
8  app.use(bp.json())
9  app.use(bp.urlencoded({ extended: true }))
10
11 routerApi(app);
12
13 app.listen(port, () => {
14     console.log(`The app is runnig in port ${port}!`);
15 });
```

Ilustración 39. Archivo “app.ts” de la aplicación middleware

Express nos ofrece la posibilidad de utilizar *routers* dentro de la aplicación, por lo que se ha construido un *router* global que es capaz de redireccionar las peticiones a un *subrouter* pudiendo separar así cada caso de uso. En la Ilustración 40 podemos ver el *router* que hemos creado.

```
const baseAppPath = '/wparty-middleware';
const apiVersion = '1.0';

export function routerApi(app) {
    app.use(`${baseAppPath}/${apiVersion}/event`, eventRouter);
    app.use(`${baseAppPath}/${apiVersion}/event/convention`, conventionEventRouter);
    app.use(`${baseAppPath}/${apiVersion}/event/party`, partyEventRouter);
    app.use(`${baseAppPath}/${apiVersion}/login`, loginRouter);
    app.use(`${baseAppPath}/${apiVersion}/user/particular`, particularRouter);
    app.use(`${baseAppPath}/${apiVersion}/user/company`, companyRouter);
}
```

Ilustración 40. Router global de la aplicación middleware

En nuestro ejemplo, el *router* de segundo nivel que actúa es el *router* "`partyEventRouter`". En la Ilustración 41, ya podemos ver el controlador específico de los flujos que alteran el estado de los eventos de tipo "Fiesta".

```
export const partyEventRouter = express.Router();
const partyEventService = new PartyEventService();

partyEventRouter.get('/:id', async (req, res) => {
    const { id } = req.params;
    let partyEventDetail = await partyEventService.findById(Number(id));
    res.json(partyEventDetail);
});

partyEventRouter.post('/', async (req, res) => {
    let result = await partyEventService.save(req.body);
    if (!result) {
        res.status(500).json({ message: 'Error saving Party Event' })
    } else {
        res.status(201).json(result);
    }
});

partyEventRouter.post('/assistant', async (req, res) => {
    let result = await partyEventService.addAssistantToEvent(req.body);
    res.status(201).json(result);
});

partyEventRouter.delete('/assistant', async (req, res) => {
    await partyEventService.deleteAssistantToEvent(req.body);
    res.status(200).send();
});
```

Ilustración 41. Controlador de la ruta 'event/party'

El controlador invoca al método correspondiente de la capa de la lógica de negocio, para que procese la información enviada. En este caso, el método "`save`" de la clase

"`PartyEventService`" mostrado en la Ilustración 42. El método "`save`" es el encargado de coordinar ambas bases de datos, por lo que, primero se encarga de insertar la información requerida en la base de datos principal.

```
save = async (request: PartyEvent) => {
        try {
            let { data: res } = await this.performSavePartyEvent(request);
            if (res) {
                eventService.save({ ...res as BaseEvent }, res.id);
                return res;
            }
        } catch (err) {
            console.log(err);
        }
    }
```

Ilustración 42. Lógica de negocio del flujo: Guardar un evento de tipo 'Party Event'

El método "`save`" es el encargado de coordinar ambas bases de datos, por lo que, primero se encarga de insertar la información requerida en la base de datos principal. Para poder introducir una nueva en entrada en la base de datos y por lo tanto invocar la aplicación Back-end se ha hecho uso de la librería "*axios*" para la construcción de peticiones HTTP.

```
export const savePartyEvent = (request: PartyEvent) => {
    return axios.put<PartyEvent>(`${baseUrl}`, { ...request });
}
```

Ilustración 43. Petición HTTP de guardado de un evento de tipo 'Party Event'

A través de la cláusula "`await`", que se puede apreciar en la Ilustración 42, se indica que se debe de esperar la respuesta de la aplicación Back-end para continuar con la ejecución del código. Una vez se ha obtenido respuesta por parte de la aplicación Back-end, existen 3 flujos a seguir:

- La ejecución ha transcurrido correctamente y obtenemos el objeto respuesta de la aplicación Back-end: Ejecutamos el bloque "`if`" e insertamos en la base de datos caché la respuesta a la petición mapeada como un evento básico, ya que, la base de datos caché almacena datos de evento de todo tipo y no se ha definido el uso de propiedades específicas de cada tipo de evento. Para ello, accedemos a la capa de lógica de negocio genérica para eventos. Este servicio se observa en la Ilustración 44.

Para poder insertar la nueva entrada requerida volvemos a hacer uso de la librería "*axios*". De esta forma, se construye la petición HTTP necesaria, observable en la Ilustración 45 para introducir valores en el índice "`event`" de la base de datos caché con una clave primaria predefinida. Como podemos observar en la Ilustración 42, para la petición de inserción en la base de datos caché no se ha introducido la cláusula "await", por lo que la petición se realizará de forma asíncrona y podremos dar una respuesta al usuario sin la necesidad de esperar a que la petición sea resuelta.

- La ejecución ha transcurrido de forma correcta pero no se ha obtenido un objeto de respuesta: En este caso, aunque podamos dar la solución como válida, entendemos que de alguna forma ha ocurrido un error en el proceso de guardado en la base de datos principal. Por esto, no ejecutamos la petición de guardado en la base de datos caché para no provocar errores en la aplicación middleware.
- Ha ocurrido un error durante el transcurso del código: Mediante este bloque try/catch prevenimos a la aplicación de errores no controlados y por lo tanto que corten la ejecución por completo de la aplicación.

Ilustración 44. Lógica de negocio de guardado de un evento básico

Ilustración 45. Petición de guardado de un evento a la base de datos caché

Una vez ya se han ejecutado todas la lógica de negocio y las peticiones pertinentes, se debe de construir la respuesta al cliente. Para ello retomamos la Ilustración 41. Al marcar el método invocado como `async`, se debe de aplicar la cláusula `await` para esperar la respuesta de la capa de negocio. Cuando ya se ha obtenido respuesta, si la respuesta contiene un objeto devuelto por la base de datos principal, respondemos al cliente con este mismo objeto y un código HTTP 201.

Así indicamos al cliente, de que no ha ocurrido ningún error durante el proceso y su evento de tipo “Fiesta” ha sido creado correctamente.

Pero, por otra parte, si se ha obtenido un objeto vacío, se interpreta como que no se han cumplido las pertinentes inserciones en la base de datos y retornamos un error de servidor con el código HHTP 500.

6 Pruebas

Las pruebas son una parte fundamental del desarrollo de software. Todo desarrollo de código que no vaya acompañado con un conjunto de pruebas que certifiquen el sistema como fiable, será un desarrollo incompleto y de poca calidad.

En nuestro proyecto hemos escogido realizar test de integración [49]. Estas pruebas se identifican porque no cubren métodos aislados, sino que, cubren flujos completos de un sistema. Es ideal para nuestro proyecto, ya que, nuestras aplicaciones están divididas por capas y es por esto por lo que tenemos que probar si esta integración entre capas es segura y se ha realizado correctamente. Pero no solo hay que probar que nuestro código funciona correctamente, también debemos cerciorarnos en que nuestro código se comporta correctamente frente a errores. Con esto nos referimos a que el código debe de ser capaz de responder con una respuesta adecuada frente a acciones no deseadas, por ejemplo, a acciones que dejarían el sistema en un estado inconsistente.

Como nuestras aplicaciones se comportan como una API, podemos probar estos flujos directamente realizando peticiones HTTP a las aplicaciones. De esta forma, estaremos probando el comportamiento de los flujos y casos de uso del proyecto. Para poder realizar estas peticiones, se ha utilizado una aplicación llamada *Postman* [55]. Este programa ofrece una interfaz gráfica para la configuración de una petición HTTP. Además, nos permite la creación de proyectos para poder guardar las peticiones y crear grupos de pruebas. De esta forma, podremos incluirlas en la aplicación en concreto y de esta forma cualquier desarrollador que entre al proyecto, tiene disponible un conjunto de pruebas para poder testear que sus cambios en el proyecto no afectan al flujo ya desarrollado.

Para comenzar estas pruebas, debemos de empezar a realizar las pruebas de fuera hacia dentro, es decir, debemos de empezar con el sistema más desacoplado y terminar con el sistema que tenga más dependencias de otras aplicaciones. Esto se debe a que las pruebas de integración prueban que los distintos componentes estén correctamente integrados entre sí. Este flujo de trabajo ayuda a detectar errores más fácilmente. Un ejemplo de ello puede ser el siguiente: Imaginemos que detectamos un error en el sistema más acoplado, pero el sistema más desacoplado aún no se ha testeado. Para poder encontrar el error tendríamos que ir tirando del hilo hasta encontrar el error y puede ocurrir que el error se encuentre en el sistema más desacoplado. De la forma correcta, si tuviéramos certificado el sistema más desacoplado podríamos centrarnos en descubrir el error en el sistema que estemos probando en ese instante. Por este motivo, comenzaremos las pruebas con la aplicación Back-end.

Como ya se ha comentado, utilizaremos *Postman* para realizar las pruebas. Para poner un ejemplo de cómo se han llevado a cabo las pruebas, se ha utilizado un flujo de creación de un evento.

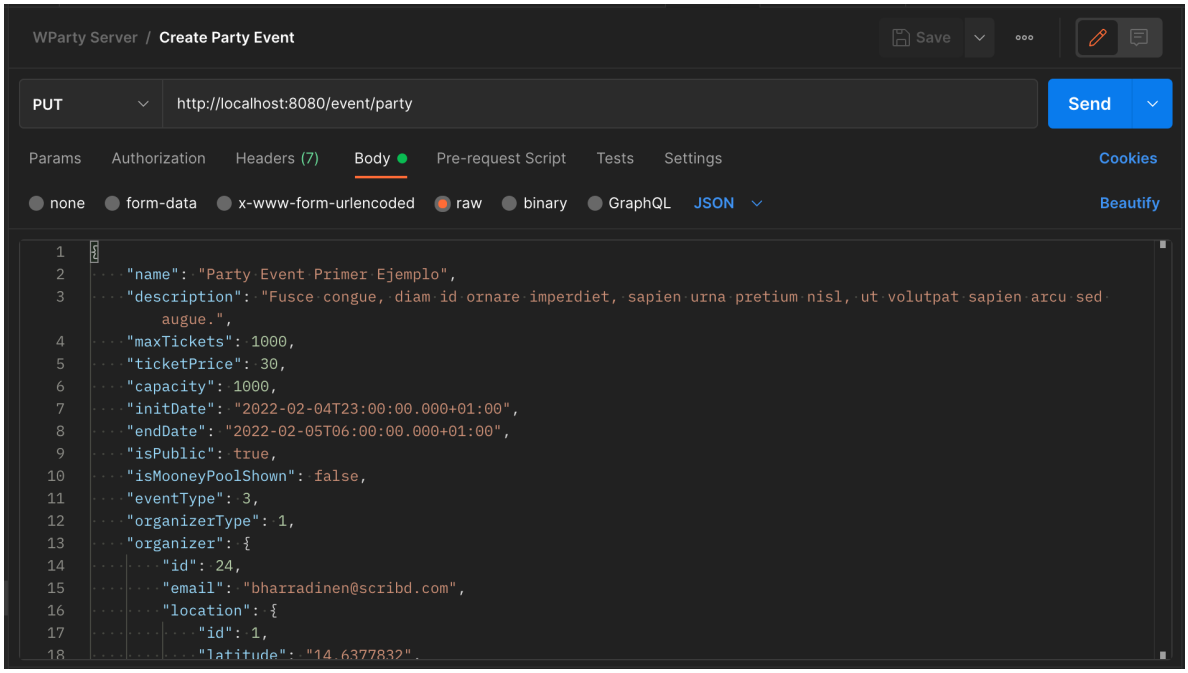


Ilustración 46. Petición, a través de Postman, de creación de evento de tipo 'Party Event'

Comencemos probando el flujo de inserción. En la Ilustración 46 podemos ver la petición HTTP que hemos generado usando Postman. Para este flujo en concreto en una inserción deberíamos tener en cuenta las siguientes dependencias de la base de datos principal:

- Dependencia directa a la tabla "Location".
- Dependencia de herencia entre la tabla "Event" y la tabla "Party".
- Dependencia directa con la tabla "User".

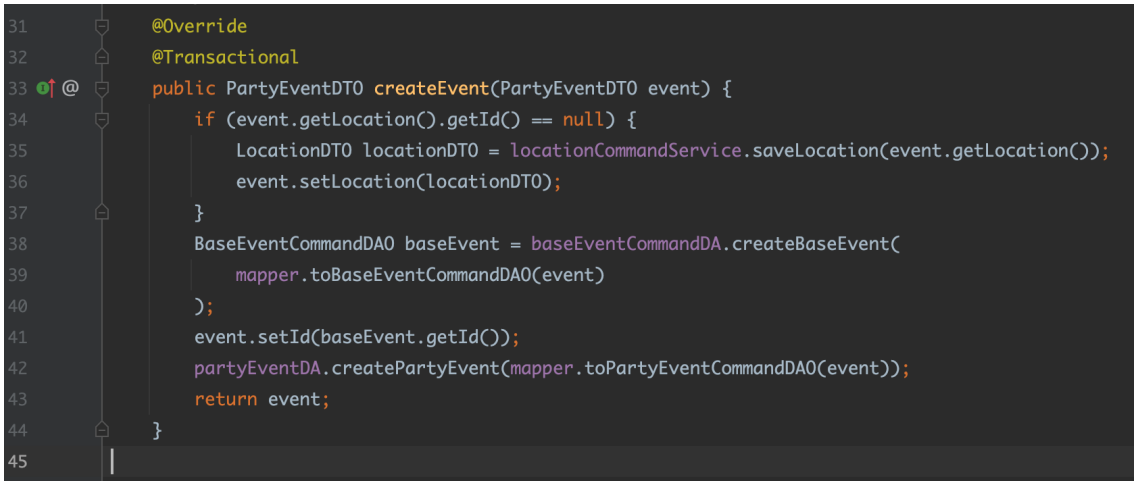


Ilustración 47. Lógica de persistencia del flujo "Creación de un Evento de tipo 'Party Event'" de la aplicación Back-end

Como se muestra en la Ilustración 47, en el desarrollo de la lógica de negocio, para esta aplicación la mantenibilidad de la consistencia se han tomado las siguientes decisiones:

- La dependencia con la tabla de "Location", se comprueba si se trata de una nueva dirección o de una ya existente. Si no existe, se crea un nueva entrada para la tabla "Location" o por el contrario, si existe una ubicación con las mismas características, se aprovecha una ya existente. Para ello se accede a la lógica de negocio propia del caso de uso del servicio "`LocationCommandService`".
- Como se trata de un nuevo evento, se debe crear un evento base en primer lugar, para que la base de datos autogenere una nueva clave primaria. Por este motivo, debemos de recibir la respuesta de la base de datos ante la petición de creación de una nueva entrada para la tabla "Event, para que posteriormente, podamos introducir la clave primaria autogenerada. Una vez recibida la calve primaria, estamos listos para la inserción de una nueva entrada en la tabla "Party Event".
- Para la dependencia con la tabla User, no se ha introducido ninguna lógica de consistencia, ya que el usuario debe de estar previamente autenticado y con un usuario válido, por lo que, si este dato es inconsistente deberíamos de rechazar la petición.

Con estas restricciones definidas tenemos los siguientes flujos también definidos que debemos de testear:

- Inconsistencia en la tabla "Party Event": Debemos de testear que los parámetros definidos como NOT NULL (es decir, no vacíos) se envían, además de que deben de insertarse con el formato correcto.
- Inconsistencia en la tabla "Base Event": Ídem al punto anterior, pero afectando a la tabla "Event"
- Entrada de localización ya existente: En este punto existen dos flujos a seguir:
    - En la entrada a la petición ya se incluye una clave primaria para esta ubicación.
    - En la entrada a la petición no se incluye una clave primaria, pero en la base de datos ya se incluye una entrada similar.
- Entrada de localización no existente: Debemos de comprobar que ante una nueva localización, el sistema guarda la información correctamente a menos de que se incluya una entrada no válida, es decir, el objeto incluido en la petición no sea correcto (campos NOT NULL, formato, etc.)
- Dependencia con la tabla "User": Debemos de comprobar que ante una entrada con un usuario válido no se produce ningún problema, pero ante una inconsistencia con el usuario, el flujo de inserción se detiene y se devuelve un error de inconsistencia.

Como ejemplo visual, en la Ilustración 48 podemos ver como se la aplicación responde con un código HTTP 201 *Created* indicando que la inserción se realizó correctamente insertando la nueva entrada en la base de datos principal.

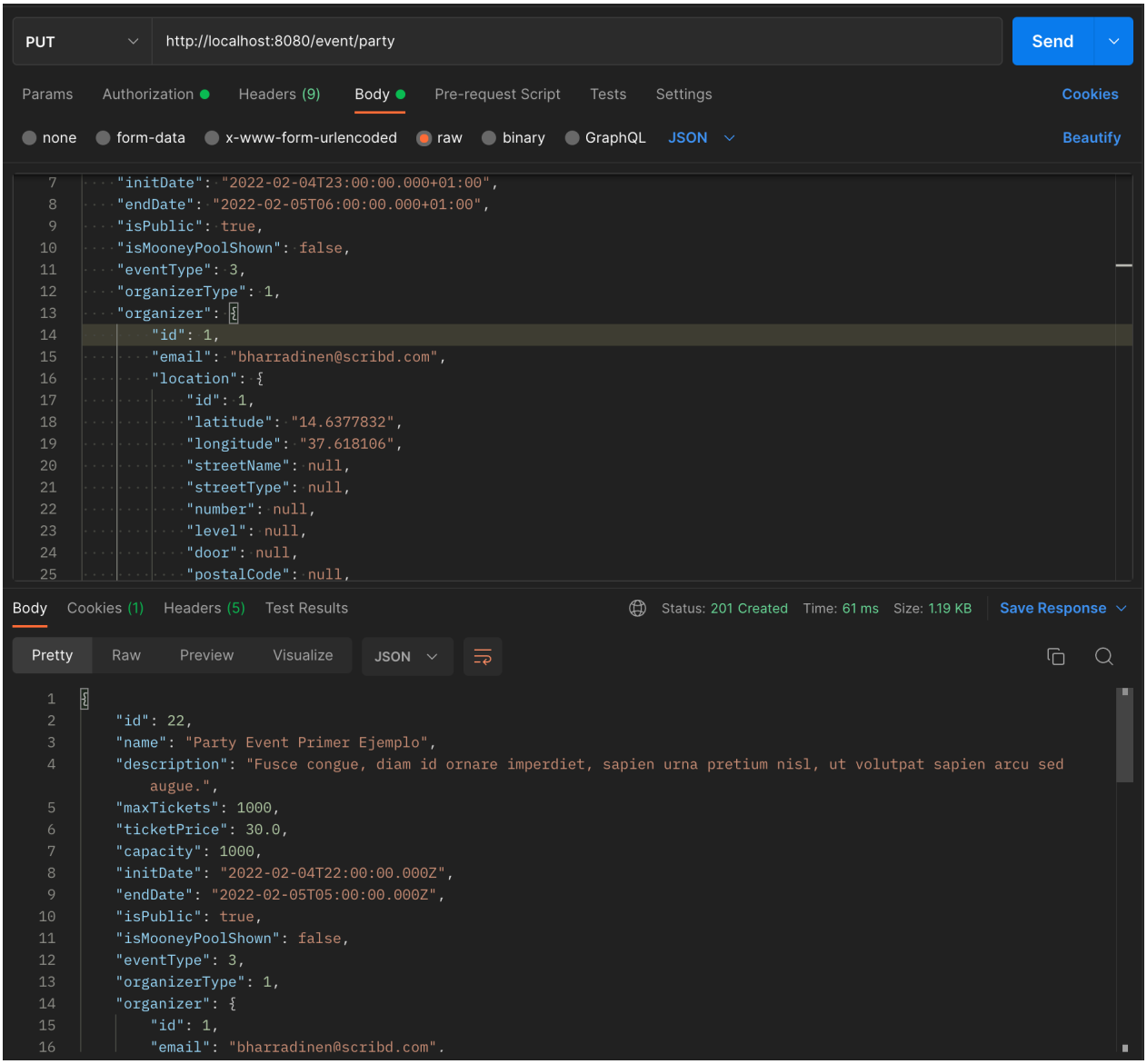


Ilustración 48. Ejemplo de creación de evento de tipo 'Party Event' de manera satisfactoria

Mientras que en la Ilustración 49 podemos observar cómo ante el error en la entrada introduciendo un usuario no existente la aplicación devuelve un error HTTP 500 *Internal Server Error* manteniendo el sistema en un estado consistente.

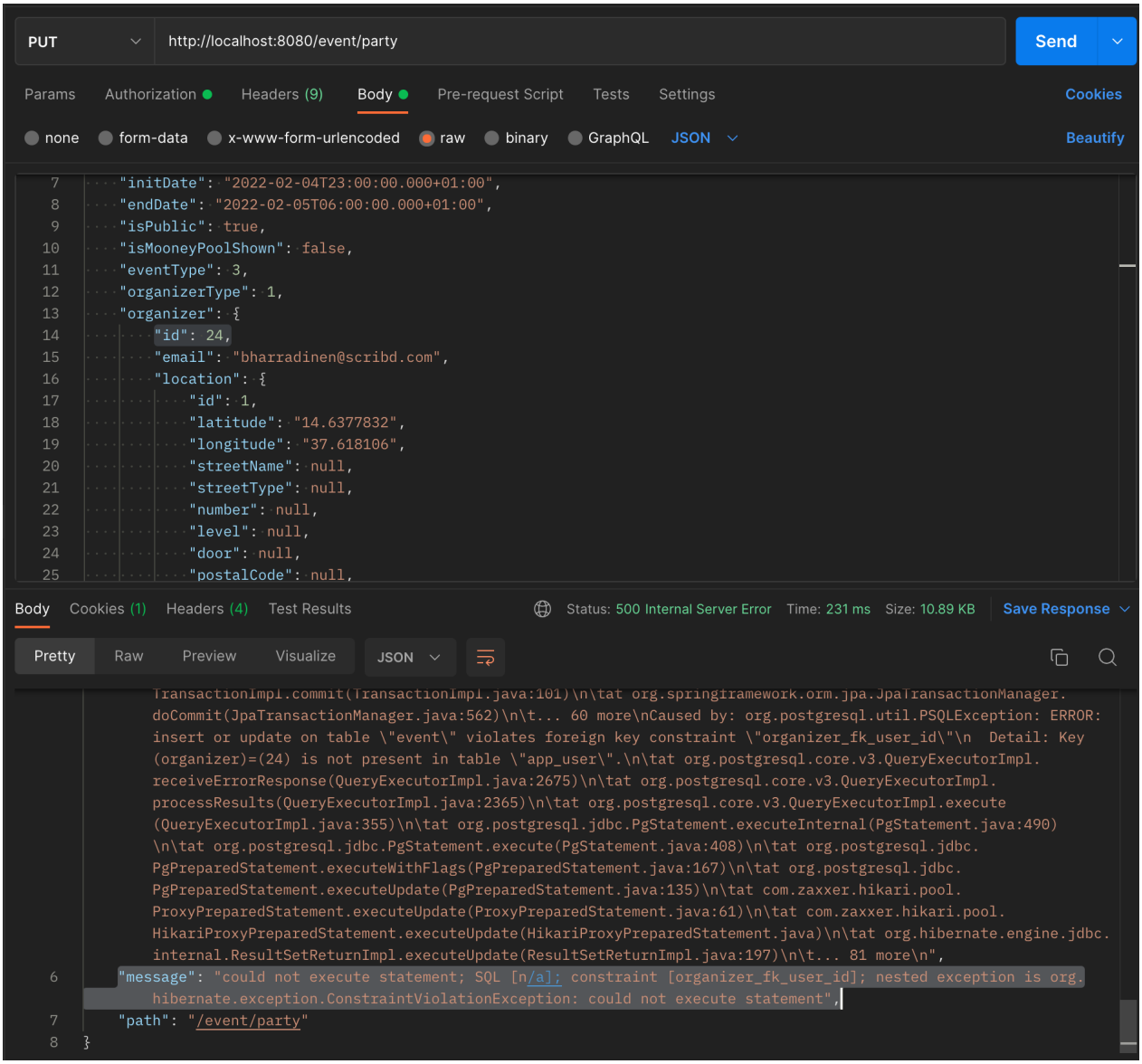


Ilustración 49. Ejemplo de creación de evento de tipo 'Party Event' de manera errónea

Para continuar con la misma filosofía de testeo, para mostrar cómo se han desarrollado las pruebas de integración de la aplicación middleware pondremos un ejemplo de creación de un evento de tipo "Party Event".

```
save = async (request: PartyEvent) => {
        try {
            let { data: res } = await this.performSavePartyEvent(request);
            if (res) {
                eventService.save({ ...res as BaseEvent }, res.id);
                return res;
            }
        } catch (err) {
            console.log(err);
        }
    }
```

Ilustración 50. Lógica de negocio en el flujo "Guardar evento de tipo 'Party Event'" en la aplicación middleware

La implementación realizada de este flujo se puede observar en la Ilustración 50. Como se puede observar este flujo se encarga de guardar el evento de tipo "Party Event" en la base de datos principal, es decir, a través de la aplicación Back-end (ya testeada previamente, por lo que no debemos de centrarnos en el buen funcionamiento de esta), y encargada de guardar el evento en la base de datos caché.

Para ello debemos de comprobar los siguientes casos:

- Un evento válido se guarda en ambas bases de datos, devolviendo al cliente un código HTTP 201 *Created*
- Un evento no válido para la base de datos principal no debe de guardarse en la base de datos caché.

En la Ilustración 51 se puede apreciar como la aplicación middleware nos devuelve el código HTTP correcto para el caso en que el objeto sea un objeto válido, mientras tanto, en la Ilustración 52 vemos como la aplicación nos ha devuelto un error de servidor y por lo tanto, no se ha guardado el evento en la base de datos caché.

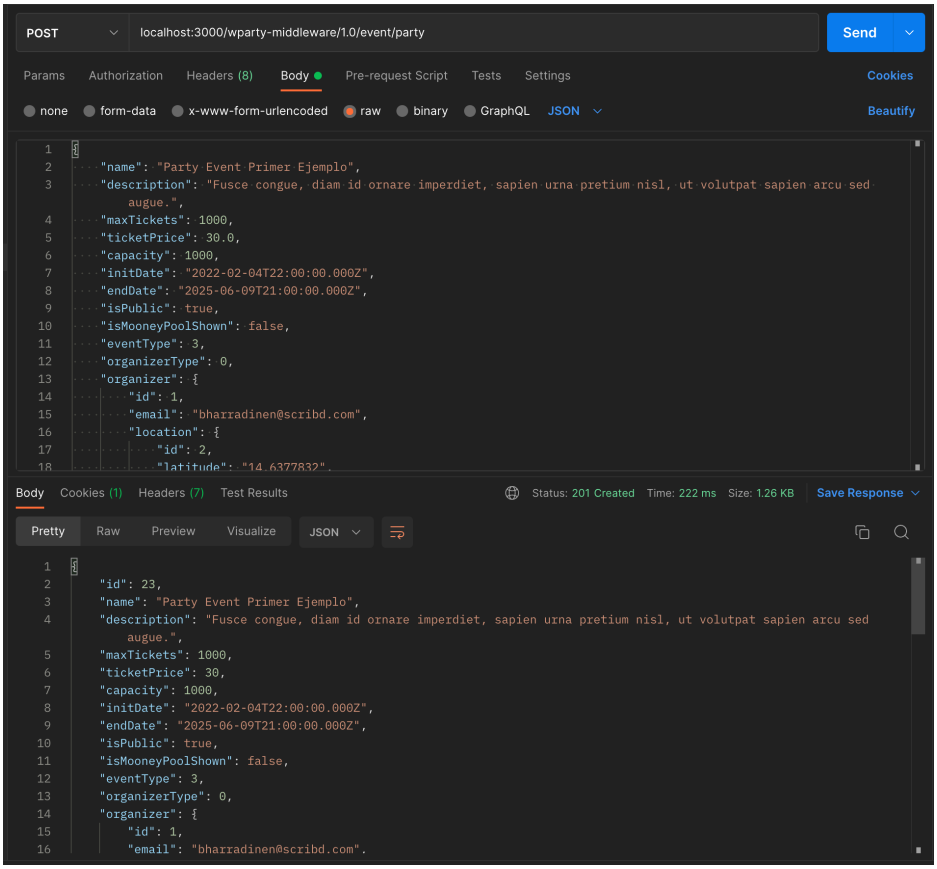


Ilustración 51. Ejemplo de petición de guardado de un evento a través de la aplicación middleware

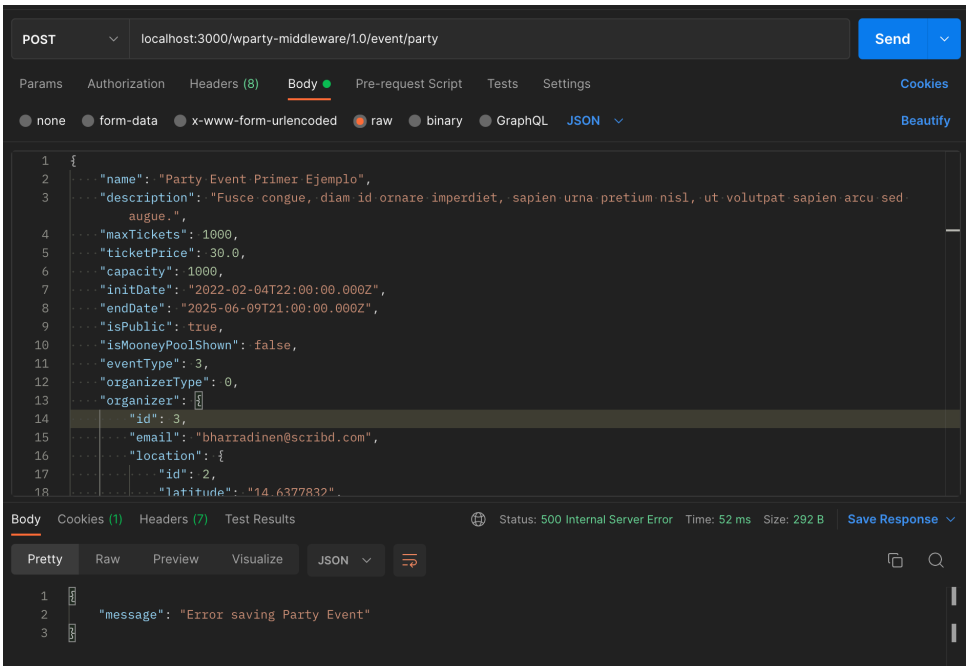


Ilustración 52. Ejemplo de error de guardado de un evento en la aplicación middleware

7 Conclusiones

En este capítulo final de la memoria se van a repasar los objetivos iniciales, indicando en cada caso su grado de consecución. Asimismo, se hará alusión a las habilidades tanto técnicas como personales y los conocimientos que se han necesitado para poder abordar el desarrollo de este proyecto, y en concreto, las asignaturas cursadas durante el grado.

Finalmente, se ha reservado un apartado para comentar posibles mejoras que se podrían abordar en el futuro. Algunas mejoras no se han podido abordar por falta de tiempo, y otras por falta de experiencia en algunos casos, y quedarían como posibles ampliaciones a este trabajo.

7.1 Grado de cumplimiento de los objetivos

En este proyecto se ha desarrollado un sistema de *back end* que ofrece un soporte adecuado a una aplicación integral de gestión de eventos, tanto para particulares como para empresas. El proyecto ha abarcado todas las fases del desarrollo, incluyendo el análisis de los requisitos, el diseño de los distintos componentes del sistema, su implementación, y las pruebas que garantizan un correcto funcionamiento. En general, el sistema desarrollado es un sistema robusto, ya que, se compone de aplicaciones y bases de datos dedicadas a casos de uso concretos, y además, es capaz de ser ampliado muy fácilmente, es decir, con una alta escalabilidad, lo que en un futuro nos dará una ventaja competitiva frente a otras aplicaciones, siendo muy sencillo la ampliación de funcionalidades del sistema.

Si nos centramos en los objetivos más particulares que se planteaban inicialmente, relacionados con los requisitos fundamentales que se exigían a la solución, podemos destacar los siguientes:

- Se cumplen todos los requisitos funcionales definidos en el Capítulo 3, garantizando un correcto funcionamiento desde el punto de vista de la operativa esperada del sistema.
- El sistema desarrollado ofrece unos tiempos de respuesta adecuados tanto respecto a inserciones de datos en el sistema como a consultas contra la base de datos principal. Pero además, el uso de la base de datos caché optimiza los tiempos de respuesta en flujos que accedan a ella. Como ejemplo, podríamos observar el tiempo de respuesta de la Ilustración 51, que incluye una inserción a la base de datos principal y una inserción a la base de datos caché.
- Todos los componentes desarrollados en este proyecto han seguido los principios del patrón SOLID [56] y las buenas prácticas descritas por Robert C. Martín en la colección de libros denominada "*The Clean Code*" [57]. Por lo tanto, nuestro código es altamente legible, efiente y escalable en el tiempo.

- Respecto a las pruebas de integración realizadas, gracias a disponer del código de las aplicaciones (pruebas de caja blanca), se ha conseguido generar las pruebas necesarias para poder recorrer el código en su totalidad, garantizando así su correcto funcionamiento.
- Los *endpoints*, o rutas de acceso a los servicios de una aplicación, describen recursos del sistema, es decir, definen a los objetos representados en el sistema que afectan a la petición como puede ser un evento o un usuario. De esta forma, se sigue un patrón por el cual, los propios *endpoints* describen a que recurso del sistema se está accediendo. Gracias a los métodos de petición HTTP (como lo son GET, POST o PUT) damos un contexto completo a una petición, ya que, estos métodos describen acciones a realizar relativo al recurso incluido en el *endpoint* de la petición.
- Se ha optimizado el uso de la memoria, de forma que la aplicación Back-end hace uso de las variables que son estrictamente necesarias para el caso de uso y la legibilidad del código. De hecho, no se han incluido *streams* o *buffers* para el almacenamiento de largas secuencias de datos para no incluir más uso de memoria.
- En cuanto a las consultas SQL, en la aplicación Back-end no se hacen consultas genéricas de recuperación del total de los datos almacenados, algo que suele ser una práctica habitual. En muchas ocasiones, las aplicaciones realizan peticiones en las que se incluyen una gran cantidad de datos, para que posteriormente filtrar esos resultados mediante código y dar una respuesta al consumidor de ese servicio. A diferencia de ello, nuestra aplicación Back-end solamente realiza las peticiones SQL específicas para poder recuperar los datos necesarios, mejorando así la eficiencia global del sistema.
- Respecto a la seguridad, se han tomado tres decisiones fundamentales para conseguir un grado suficiente y adecuado para nuestro sistema. En particular:
    - Se ha utilizado un cifrado de tipo sha256 para almacenar la contraseña de los usuarios del proyecto.
    - Para el envío de la petición de creación de usuario y de login, la contraseña se mantiene cifrada con un algoritmo base64 entre la aplicación middleware y la aplicación Back-end.
    - La inclusión del componente Spring Security en la aplicación Back-end con usuario y contraseña que únicamente conoce la aplicación middleware asegura que todos los flujos están protegidos con esas credenciales, incluyendo por supuesto aquellos que manejan datos sensibles.

7.2 Relación del trabajo desarrollado con los estudios cursados

Este proyecto es el resultado la puesta en práctica de los conocimientos que se han adquirido durante el transcurso del grado, la experiencia laboral que se ha conseguido en la compañía en la que me encuentro trabajando, y el propio desarrollo personal que se ha obtenido aprendiendo de manera autodidacta.

En concreto, si nos centramos en los conocimientos adquiridos durante el grado. podemos destacar una serie de asignaturas que tienen relación directa en el desarrollo del proyecto:

- Asignaturas como “Introducción a la informática y a la programación” o “Programación” me han ayudado a interpretar el código de los lenguajes de programación y cómo desarrollarlo de una forma adecuada.
- Las asignaturas de “Desarrollo Web” y “Sistemas y servicios en red” han sentado las bases a partir de las cuales se ha desarrollado este sistema para su posterior consumo en una web o aplicación móvil.
- Para toda la comunicación entre aplicaciones y bases de datos, y la seguridad en la red, he utilizado los conocimientos adquiridos en las asignaturas de “Redes de Computadores”, “Seguridad en redes y sistemas informáticos” y “Redes corporativas”.
- Para el diseño, la creación y el manejo de las bases de datos, he aprovechado los conceptos aprendidos en las asignaturas “Bases de datos y sistemas de información” y “Tecnología de bases de datos”.
- La asignatura de “Ingeniería del software” ha sido muy relevante respecto a la organización y diseño del sistema completo que compone nuestro proyecto.
- Finalmente, las prácticas de empresa han sido totalmente fundamentales para entender cómo funciona un proceso de diseño y desarrollo de un componente tecnológico en un entorno empresarial. En ellas hemos adquirido tanto capacidades técnicas que han completado nuestra formación, como buenas prácticas que ayudan al proceso de desarrollo de software, en aspectos como el desarrollo del propio código o el uso de un gestor de versiones para controlarlo.

7.3 Trabajos futuros

En primer lugar, debemos destacar como es lógico la ampliación natural y evidente de este trabajo, que es la implementación de la una aplicación (*front-end*) que interaccione con el usuario y que reciba el soporte del sistema desarrollado en este trabajo, completando así la solución de principio a fin.

Pero además, durante el transcurso del diseño y la implementación del proyecto, hemos observado algunas posibles mejoras que podrían optimizar y mejorar ciertos aspectos de este. A continuación, se ha elaborado una lista de algunas de las posibles mejoras del proyecto:

1. Incluir un sistema de logs centralizado a través de *Kibana* [58]: *Elastic Stack* tiene un producto denominado *Kibana* que es capaz de almacenar un sistema de logs de forma centralizada. Además, este sistema permite la creación de un portal con estadísticas como los eventos más buscados o el evento con más asistentes, que podríamos ofrecer a los usuarios de tipo “Empresa” para que pudieran analizar esos datos para usarlos en su modelo de negocio o crecer a través de ese análisis.
2. Pasar de un modelo de peticiones HTTP a un modelo gestionado a través de eventos: Este modelo es adecuado para proyectos que necesiten alta escalabilidad y se necesiten más de

un almacén de datos. Esto incluiría un sobrecoste para el proyecto, pero si queremos obtener la mayor escalabilidad posible para un proyecto, esta es la mejor opción. Además de un mayor coste de programación del código, incluye la integración de gestores de colas como RabbitMQ [59] lo que supone otro sobrecoste. Para nuestro proyecto, que está orientado al mercado nacional, no se adecuaba del todo a nuestras necesidades, pero si quisiéramos expandirnos internacionalmente deberíamos escoger esta opción. Habiendo elegido una arquitectura hexagonal para nuestro proyecto, los cambios en el proyecto serían mínimos, solamente deberíamos cambiar los controladores HTTP por unos controladores *pub/sub* [60] que cumplan con nuestras necesidades.

3. Añadir una seguridad común para todo el sistema: Se podría utilizar un sistema de JWT [61] (*JSON Web Token*) para poder almacenar la información de acceso relevante del usuario y de esta forma, no solo restringir los accesos a la aplicación middleware, sino que, podríamos bloquear el acceso a algunos aplicativos para los que no tuviera permisos.

8 Bibliografía

[1] Portal de Beamian: https://beamian.es/

[2] Portal de Meetmaps: https://welcome.meetmaps.com/

[3] Definición de *engagement*: https://rockcontent.com/es/blog/que-es-engagement/#:~:text=El%20Engagement%20puede%20definirse%20como,y%20mensajes%20de%20la%20misma.

[4] Definición de una aplicación Back-end: https://techterms.com/definition/backend

[5] Debate entre desarrolladores debatiendo si se debiese seguir usando Java: https://dev.to/brunoborges/java-15-in-2020-reasons-to-not-use-java-3ekg

[6] Definición de una base de datos relacional por Oracle: https://www.oracle.com/es/database/what-is-a-relational-database/

[7] Definición de SQL por Microsoft Support: https://support.microsoft.com/es-es/office/access-sql-conceptos-b%C3%A1sicos-vocabulario-y-sintaxis-444d0303-cde1-424e-9a74-e8dc3e460671

[8] Página oficial de PostgreSQL: https://www.postgresql.org/

[9] Definición de una base de datos no relacional por Amazon Web Services: https://aws.amazon.com/es/nosql/

[10] Portal de Amazon DynamoDB: https://aws.amazon.com/es/dynamodb/

[11] Portal de Amazon MemoryDB: https://aws.amazon.com/es/memorydb/

[12] Portal de Apache Cassandra: https://cassandra.apache.org/_/index.html

[13] Descripción de Apache Lucene por Wikipedia: https://es.wikipedia.org/wiki/Lucene

[14] Definición de estándares ACID por *Piperlab*: https://piperlab.es/glosario-de-big-data/acid/

[15] Documentación de Spring Boot: https://spring.io/projects/spring-boot

[16] Página web oficial de Apache Tomcat: https://tomcat.apache.org/

[17] Portal web de Spring Framework: https://spring.io/

[18]: JavaScript por MdnDocs: https://developer.mozilla.org/es/docs/Web/JavaScript

[19] Definición de tipado dinámico en JavaScript: https://developer.mozilla.org/es/docs/Glossary/Dynamic_typing

[20] Portal de la super clase de JavaScript llamado TypeScript: https://www.typescriptlang.org/

[21] Definición de tipado estático: https://ciberninjas.com/lenguaje-tipado/

[22] Portal de Node.js: https://nodejs.org/es/

[23] Definición del formato JSON por "mdn web docs": https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON

[24] Definición de una 'interface' TypeScript: https://www.typescriptlang.org/docs/handbook/interfaces.html

[25] Arquitectura CQRS por Microsoft Docs: https://docs.microsoft.com/es-es/azure/architecture/patterns/cqrs

[26] Definición de operaciones CRUD sobre una base de datos por IONOS: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/crud-las-principales-operaciones-de-bases-de-datos/

[27] Transacciones SQL por Diego Lázaro: https://diego.com.es/transacciones-en-sql

[28] Arquitectura hexagonal por Eduardo Salguero: https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f

[29] Herramienta 'mockaroo': https://www.mockaroo.com/

[30] Portal de Apache Maven Project: https://maven.apache.org/

[31] Portal de Swagger: https://swagger.io/

[32] Especificación de Open API 3: https://swagger.io/specification/

[33] Documentación de Spring JPA: https://spring.io/projects/spring-data-jpa

[34] Documentación de JDBC: https://www.oracle.com/es/database/technologies/jdbc-migration.html

[35] Portal de Hibernate: https://hibernate.org/

[36] Definición de un ORM por Deloitte: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html

[37] Documentación de Spring Security: https://spring.io/projects/spring-security

[38] Portal de MapStruct: https://mapstruct.org/

[39] Portal de Project Lombok: https://projectlombok.org/

[40] Patrón de diseño DAO: https://www.baeldung.com/java-dao-pattern#:~:text=1.,mechanism)%20using%20an%20abstract%20API.

[41] Definición de la codificación base64: https://es.wikipedia.org/wiki/Base64

[42] Portal de Docker: https://www.docker.com/

[43] Documentación de ElasticSearch: https://www.elastic.co/guide/index.html

[44] Portal de Express: https://expressjs.com/es/

[45] Documentación de la librería "Axios": https://axios-http.com/es/docs/intro

[46] Definición de sistema operativo UNIX por "Profesional Review": https://www.profesionalreview.com/2022/07/11/sistema-operativo-unix/

[47] Portal de NPM: https://www.npmjs.com/

[48] Definición de alias por "Undefined Shell": https://undefined.sh/comandos-alias-para-la-terminal/#:~:text=Los%20comandos%20alias%20de%20la,e%20incluso%20otros%20comandos%20alias.

[49] Portal de Prettier: https://prettier.io/docs/en/index.html

[50] Portal de ESLint: https://eslint.org/

[51] Librería de nodemon en NPM: https://www.npmjs.com/package/nodemon

[52] Definición de live reload por "GeeksForGeeks": https://www.geeksforgeeks.org/difference-between-hot-reloading-and-live-reloading-in-react-native/

[53] Definición de tipado dinámico por "MDN Web Docs": https://developer.mozilla.org/es/docs/Glossary/Dynamic_typing

[54] Definición de pruebas de integración por Manuel Cillero: https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/integracion/

[55] Portal de Postman: https://www.postman.com/

[56] Principios SOLID por Carlos Macías: https://www.enmilocalfunciona.io/principios-solid/

[57] "Clean Code" de Robert C. Martin: https://www.oreilly.com/library/view/clean-code-a/9780136083238/

[58] Portal de Kibana: https://www.elastic.co/es/kibana/

[59] Portal de RabbitMQ: https://www.rabbitmq.com/

[60] Definición del model Pub/Sub por "Amazon Web Service": https://aws.amazon.com/es/pub-sub-messaging/

[61] Portal de JWT: https://jwt.io/introduction

Anexo A. Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Proce de**
ODS 1. **Fin de la pobreza.**			X	
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**			X	
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**			X	
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**	X			
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades.**			X	
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Este proyecto está centrado en el desarrollo económico en el sector ocio de cualquier zona. Por lo que todas los objetivos de crecimiento económico de este glosario están de mayor o menor medida con la motivación del proyecto.

Comencemos a desglosar los elementos marcados en tabla que encuentran relación con este proyecto:

- **ODS 1 – Fin de la pobreza**: La salida del umbral de la pobreza de una sociedad está altamente ligada al desarrollo empresarial e industrial de una zona empobrecida. Como explica la fundación *CODESPA* en su portal web: "*La empresa, por su propia naturaleza,*

***repercute en aspectos fundamentales del bienestar social**. Sus inversiones, operaciones y cadenas de valor impactan en:*

- *Generación de **empleo**.*
- ***Desarrollo** de capital humano.*
- *Transferencia de **tecnologías**.*
- *Construcción de **infraestructuras**.*
- ***Creación y distribución de productos y servicios** para los consumidores."*

Nuestro proyecto puede ayudar a una sociedad en vías de desarrollo a la atracción de turismo a esa zona. De forma, las empresas de esa sociedad pueden clonar modelos de éxito en otras zonas geográficas a través del análisis de comportamientos en nuestro sistema y además, posibilitar a los posibles emprendedores de estas zonas una herramienta online ya desarrollada sin coste para ellos.

- **ODS 3 – Salud y bienestar**: Hoy en día la salud mental es un pilar fundamental en nuestra sociedad. Un proyecto como el nuestro ayuda a cualquier persona que necesite organizar un evento. Un usuario que utilice nuestro proyecto estaría liberando el alto coste mental y emocional que contiene la organización de un evento para que nuestra aplicación lo haga por él. De esta forma el usuario prevendría estrés que podría derivar en una ansiedad. También debemos comentar que el ocio es fundamental para poder conciliar la vida laboral con la vida personal. Nuestro proyecto ayuda a los posibles consumidores de ocio a hacerlo de forma más rápida, planificada y de forma automática.
- **ODS 5 – Igualdad de género:** La igualdad de género es un objetivo que toda la sociedad debería centrarse en resolver. Nuestro proyecto no hace ningún tipo de distinción entre perfiles masculinos o femeninos. Uno de los grandes problemas que son relativos a esta igualdad de género, es la brecha salarial. El emprendimiento es una parte fundamental para poder reducirlo. Usando nuestro proyecto, podríamos ayudar a traer más ingresos a una empresa en la que participen mujeres pudiendo así obtener más ingresos y reduciendo esta brecha salarial.
- **ODS 8 – Trabajo decente y crecimiento económico:** Este proyecto esta focalizado en este apartado del desarrollo sostenible. El proyecto se centra en ofrecer a las empresas otra vía de gestión de eventos, además de dar visibilidad y publicidad a un evento. El crecimiento económico viene acompañado de crecimiento en calidad laboral, por lo que, las condiciones laborales también se verían incrementadas.
- **ODS 9 – Industria, innovación e infraestructuras.:** Relacionado con el punto anterior, encontramos que a mayor nivel de ingresos y de movimiento de divisas, el estado es capaz de recaudar más ingresos vía impuestos. Esta subida de beneficios por parte del estado podría utilizarse para mejorar, incrementar y crear nuevas infraestructuras allá donde se necesite.
- **ODS 10 – Reducción de las desigualdades:** El aumento de ingresos por parte de los trabajadores y empresarios acarrea una clara reducción de las desigualdades, permitiendo a esas personas adquirir más bienes de primera necesidad o poder destinar esos ingresos a otras necesidades, las cuales anteriormente, no tenían acceso a ellas.

De esta forma, nuestro proyecto ayuda a crear una sociedad más justa, apoyando el crecimiento económico y social.