UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SISTEMA BACKOFFICE PARA EVENTOS DEL DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

STEVENS PATRICHS EGLI ADRIAZOLA

PROFESORA GUÍA:
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:
CLAUDIO GUTIÉRREZ GALLARDO
DANIEL PEROVICH GEROSA

SANTIAGO DE CHILE
2024

Resumen

El Departamento de Ciencias de la Computación (DCC) cuenta con numerosos sistemas desarrollados por alumnos o en el marco de cursos. Muchos de estos sistemas están en proceso de modernización para mejorar su mantenibilidad y gobernabilidad, siguiendo los estándares actuales del Área de Desarrollo del Departamento. Entre ellos se encuentra el Sistema de Administración de Noticias y Eventos (SANE), utilizado por el equipo de Comunicaciones del DCC.

Desde su implementación en 2017, este sistema ha carecido de mejoras y actualizaciones, lo que ha provocado un desalineamiento con los estándares actuales del Área de Desarrollo, generando importantes desafíos de mantenibilidad. Su complejidad se acentúa debido a su naturaleza multifuncional, que abarca la gestión de noticias y eventos, además de incluir un sitio web público para consultas. Esta concentración de responsabilidades, sumada a un módulo de creación de contenido excesivamente personalizable cuyas opciones, en su mayoría, no son utilizadas por los usuarios, no solo complica la experiencia del usuario, sino que también añade capas adicionales de dificultad al mantenimiento del sistema.

Para abordar esta problemática, se implementó un sistema backoffice exclusivamente para la gestión de eventos creados por el DCC y sus elementos, manejado por el Área de Comunicaciones. El objetivo de este trabajo fue desarrollar un nuevo sistema tomando elementos del sistema actual y adaptándolos para crear un sistema más atómico que mejore la usabilidad y mantenibilidad en comparación con el SANE. Para ello, se llevaron a cabo varias reuniones para refinar las interfaces y funcionalidades propuestas, se simplificó el modelo del SANE con algunas modificaciones y se diseñó una API que permitirá entregar los eventos creados a otros sistemas del DCC que tengan el objetivo de mostrarlos.

Durante la implementación de la solución, se validaron todos los elementos diseñados con los usuarios, obteniendo muy buenos resultados en cuanto a usabilidad y satisfacción. En términos de mantenibilidad, se lograron métricas que indican que el sistema implementado es tan mantenible como el SANE, siendo ligeramente más simple de modificar. Además, se realizó un trabajo de documentación adicional revisado por el Área de Desarrollo, garantizando que el sistema tiene un alto nivel de madurez para ser desplegado en los sistemas del DCC. Con estos hechos, se concluyó que se cumplió el objetivo principal, demostrando que este sistema es más usable y mantenible tanto para el Área de Comunicaciones como para el Área de Desarrollo.

A mi familia, mis gatos y todos quienes me acompañaron en este proceso.

Agradecimientos

En primer lugar, quiero agradecer a mi familia por brindarme todas las facilidades para estudiar esta carrera. A mi hermano Alexander, por su paciencia y ayuda con conceptos y problemas relacionados con las ciencias físicas y matemáticas. También a mis gatos, Pepe y Kann, cuya compañía y cariño han sido un pilar importante en este proceso.

Agradezco a la profesora Jocelyn guiar esta memoria y su ayuda en la corrección de errores relacionados con el formato del documento. También extiendo mi agradecimiento al área de comunicaciones, especialmente a Ana y Paulette, por su colaboración en el desarrollo de esta memoria. Asimismo, agradezco al área de desarrollo, en particular a Juan Pablo Arraigada y Luis Herrera, por su valiosa contribución en el apartado técnico.

Por otro lado, quiero expresar mis agradecimientos al Web Intelligence Centre del Departamento de Ingeniería Civil Industrial, que me brindó la oportunidad de realizar mi segunda práctica y trabajar en paralelo al desarrollo de esta memoria. En particular, agradezco a Carlos Vega por su ayuda con dudas técnicas y consejos, los cuales me permitieron despejar varias dudas y barreras mientras avanzaba.

Por último, quiero agradecer a mis grandes amigos del colegio: Fabián Miranda, Héctor Navarrete, Danko Gómez, Daniel Crovetto, Vanessa Malhue y Christopher Barrera. A pesar de haber tomado caminos tan distintos, nunca perdimos el contacto y seguimos reuniéndonos regularmente. También quiero agradecer a las amistades que hice durante la universidad: al "Team No Sé" - Beltrán Cubillos, Gonzalo Alfaro, Alex Sepúlveda y Juan (Pancho) - con quienes compartí largas tardes de estudios del ya lejano plan común. Finalmente, a mis amigos del DCC - Jorge Araya, Agustín Cortés, Jesús Cáceres y José Yuseff - con quienes compartí numerosas tareas y trabajos durante la especialidad.

Tabla de Contenido

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

El Departamento de Ciencias de la Computación (DCC) de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile ha emprendido en los últimos años el plan de mejorar la infraestructura de sus sistemas que han sido desarrollados dentro del departamento. Para ello, se han llevado a cabo tareas de estandarización de prácticas y uso de tecnologías en los proyectos actuales, además de refactorizar algunos ya existentes que poseen años de antigüedad.

El desafío con estos sistemas más antiguos radica en que, en muchos casos, no fueron desarrollados por el mismo área de desarrollo. En cambio, fueron producto del trabajo de alumnos memoristas del departamento, iniciativas de profesores o como parte de cursos del DCC, como es el caso de Ingeniería de Software II (CC5401). En estos casos, no se aplicaban los estándares y lineamientos actuales del área de desarrollo del DCC [16]. Hoy en día, se está trabajando para adaptar estos sistemas a los estándares actuales, considerando que deben ser mantenidos por el área de desarrollo debido a que siguen en producción.

Un ejemplo de estos sistemas es el Sistema de Administración de Noticias y Eventos (SANE) ocupado por el área de comunicaciones del DCC. Anterior a este, el DCC no contaba con algún sistema que le permitiera administrar eventos y noticias. Los sistemas que existieron consistían en páginas básicas que incluso no tenían considerados a los eventos. Sin embargo, en el año 2017, en el marco del trabajo de título de Francisco Madrid [10], se creó un sistema que buscaba centralizar la creación y difusión de las noticias y eventos del DCC.

Hasta ahora, este sistema ha demostrado ser lo suficientemente robusto como para seguir siendo utilizado por el área de comunicaciones del DCC durante varios años. Gracias a él, se han publicado un total de más de 700 noticias y cerca de 300 eventos hasta la fecha. Estas noticias y eventos son publicados a través de la web del SANE[1], la cual no estuvo expuesta de forma pública hasta la creación del nuevo sitio web del DCC[2] en el 2021, ya que las noticias

[1]Sitio oficial del SANE: `https://comunicaciones.dcc.uchile.cl/`
[2]Sitio web del DCC: `https://www.dcc.uchile.cl/`

y eventos se publicaban manualmente en la página del DCC. Estas noticias y eventos, por otro lado, son difundidos por las televisiones públicas del departamento y redes sociales.

1.2. Problema

A pesar de que el SANE es un sistema robusto, es un sistema centralizado que concentra en su sitio tanto las noticias y eventos como la principal fuente de información. Esto puede llegar a ser un problema, ya que el hecho de que el sistema tenga tantas responsabilidades genera inconvenientes a la hora de su mantenibilidad. Por ejemplo, en caso de que el sistema falle, el departamento se quedaría sin poder publicar tanto noticias como eventos, junto con su lectura, hasta poder solucionar el problema.

En el trabajo de esta memoria, se abordo el desafío de crear un nuevo sistema de eventos. El objetivo es desarrollar un sistema *backoffice* [3], es decir, un software que permita realizar tareas administrativas, en este caso, administrar eventos, sin la necesidad de que interactúen usuarios externos con la aplicación. De aquí, se rescataron aquellas funcionalidades que son más críticas y se ajustaron al uso que les dan los usuarios bajo los estándares y lineamientos que posee el área de desarrollo del DCC. Esto permitió crear un sistema más atómico, con menos responsabilidades y más mantenible por el departamento.

Para lograr esto, fue necesario realizar una labor de reingeniería [15] al sistema actual. Esto implico en primera instancia hacer un estudio del sistema actual para luego poder reimplementar las funcionalidades más críticas del sistema y los usuarios que administran los eventos.

Además, se creó una Interfaz de Programación de Aplicaciones, también conocida como API, que es un conjunto de definiciones y protocolos que facilitan la integración del software de las aplicaciones. Esta API será lo suficientemente completa como para alimentar otros sistemas del departamento, como por ejemplo, el sitio oficial del DCC. Actualmente, este sitio solo puede acceder a los titulares e imagen principal de los eventos y noticias.

1.3. Objetivos

Objetivo General

Diseñar e implementar un nuevo sistema para la publicación y administración de eventos del DCC. Este sistema, que se utilizará como backoffice, se centrará en mejorar la mantenibilidad y la usabilidad con respecto al sistema actual de noticias y eventos.

Objetivos Específicos

1. Clasificar e identificar las funcionalidades y características esenciales del sistema de eventos actual para su reimplementación, descartando las que no aportan valor.
2. Diseñar e implementar las interfaces gráficas que necesitará el sistema.
3. Diseñar e implementar una API que permita al nuevo servicio de eventos comunicarse con otros servicios del DCC, junto con evaluar la simplificación y actualización del modelo de datos en comparación al actual.
4. Comprobar que el nuevo sistema de eventos es más usable que el sistema existente.
5. Demostrar que el nuevo sistema de eventos es más mantenible que el sistema SANE actual, logrando una cobertura de pruebas del 70-80 % y mejorando los indicadores de calidad del código como la complejidad ciclomática y el índice de mantenibilidad [5, 11] del SANE.

1.4. Solución Propuesta

Actualmente, el SANE es un sistema que concentra diversas responsabilidades en su función. Por un lado, actúa como sitio web que alberga las noticias y eventos del DCC, permitiendo que usuarios externos los consulten directamente. Por otro lado, otros sistemas acceden a sus datos mediante la API del SANE, como se ilustra en la Figura 1.1. El nuevo sistema, diseñado para reemplazar al SANE en el aspecto de los eventos, reducirá las responsabilidades del sistema actual. Las consultas de eventos se realizarán exclusivamente a través de la nueva API destinada para este fin, ya que el sistema será de uso interno y los demás sistemas se comunicarán con él mediante esta API, como se muestra en la Figura 1.2.

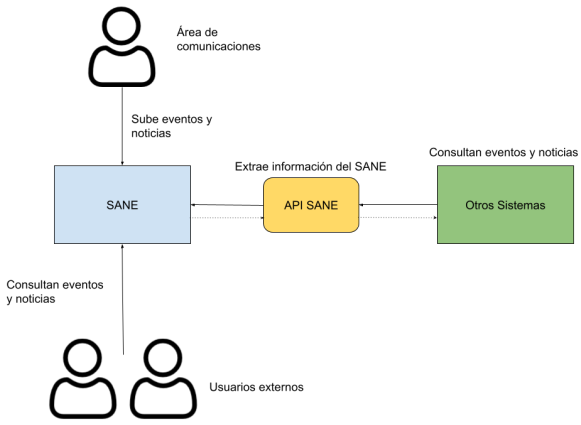


Figura 1.1: Diagrama actual del SANE.

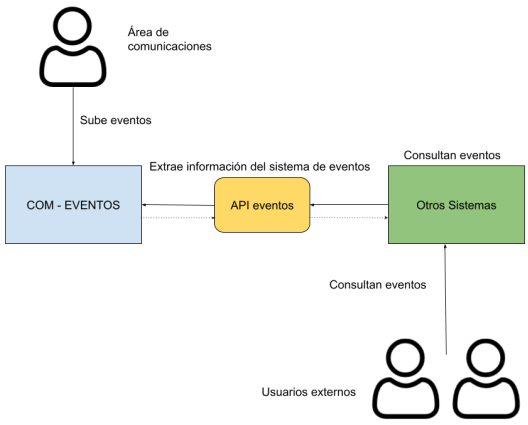


Figura 1.2: Diagrama del nuevo sistema de eventos.

Considerando que la interfaz de creación de eventos del sistema actual contenía numerosos elementos, muchos de los cuales resultan innecesarios en el nuevo sistema, se han realizado

ajustes pertinentes. Por ejemplo, se ha eliminado el sistema de *tags* y se ha dejado de solicitar información innecesaria. Además, se ha prescindido de la integración con el sistema SAR, que no funciona correctamente. Según el área de desarrollo, en el futuro se planea realizar una reimplementación de este sistema, por lo que intentar arreglar su implementación actual podría generar problemas a futuro.

Para su implementación, se realizó un análisis exhaustivo de la implementación y documentación actual del SANE para evaluar las funcionalidades esenciales para el nuevo sistema. Dado que el área de desarrollo proporciona una plantilla para el desarrollo de software dentro del departamento, construida con Django en sus versiones 4 o superior, fue necesario adaptar estas funcionalidades a dicho marco. Para el desarrollo del nuevo sistema, se utilizó finalmente la versión 5.2.2 de Django, junto con la plantilla proporcionada por el área de desarrollo, y se empleó GIT para el control de versiones.

Dado que la plantilla de Django incluye un sistema de autenticación a través del portal DCC, se llevó a cabo una revisión y replanteamiento del modelo de datos del SANE. El modelo actual considera la forma de autenticación de los usuarios y elementos que podrían no ser necesarios, como los *tags*. En este proceso de revisión, se evaluaron los elementos que se mantendrían del modelo actual del SANE y qué otros elementos se agregarían al nuevo modelo de datos. En general, se buscó simplificar el modelo a uno más sencillo. Para el caso del modelo de datos, se continuó usando el motor de bases de datos PostgreSQL, pero con la versión más reciente.

En cuanto al front-end del nuevo sistema, en primer lugar, se diseñó una interfaz que sería utilizada por el área de comunicaciones y que fue adaptada según la retroalimentación proporcionada por las usuarias principales. Para su implementación, se descartó el uso de un framework de front-end independiente; en su lugar, se utilizará el sistema de Django Templates[3] para construir la interfaz. Además, se planteó usar Bootstrap[4], framework integrado en la plantilla de desarrollo, para aplicar los estilos y utilizar componentes correspondientes a las vistas. Para algunas funcionalidades interactivas, no se descartó el uso de librerías como jQuery UI[5] o SortableJS[6].

Adicionalmente, se desarrolló una API del tipo *Representational State Transfer* (REST). Una API REST adhiere a los principios de la arquitectura REST [1], un conjunto de restricciones basado en el sistema cliente-servidor sin estado. En este sistema, las solicitudes web se tratan como recursos independientes identificados por URLs o *endpoints*. Estos recursos permiten a los usuarios interactuar utilizando métodos HTTP como GET, POST, PUT y DELETE. Estas APIs proporcionan una forma flexible y ligera de integrar aplicaciones y conectar componentes en arquitecturas de microservicios [14].

Este nuevo sistema de eventos puede considerarse como un microservicio, dado que se desplegará de manera independiente y contará con su propio modelo de gestión de datos, enfocado únicamente en los eventos del DCC. Por lo tanto, el uso de una API REST para facilitar la comunicación entre este nuevo sistema y otros externos es el enfoque más adecuado.

[3]Django Templates : `https://docs.djangoproject.com/en/4.2/topics/templates/`
[4]Bootstrap: `https://getbootstrap.com/`
[5]jQuery UI: `https://jqueryui.com/`
[6]SortableJS: `https://sortablejs.github.io/Sortable/`

Para su desarrollo, inicialmente se consideró el uso de Django REST Framework[7], una librería que simplifica la creación de APIs REST en un entorno Django. Sin embargo, más adelante en la implementación se descartó su uso debido a su complejidad considerando el simple modelo de datos. En su lugar, se optó por una librería más sencilla de usar, más acorde a la complejidad del modelo de datos y las necesidades del proyecto. Esta API transmite varios elementos de un evento para describirlo de manera completa, incluyendo detalles como la ubicación, la fecha, la modalidad, el contenido del evento, enlaces o URL relevantes, imágenes referenciales, entre otros.

También se definieron formas para filtrar la información consultada por la API, como por ejemplo entregar los próximos eventos a la fecha actual, eventos pasados, eventos por su modalidad, etc., y enviar texto enriquecido. Esto permitirá entregar el texto del cuerpo de un evento con el formato exacto para que pueda ser visualizado y aplicado en otros contextos, como en la web oficial del DCC.

1.5. Metodología

El desarrollo de este sistema se basó en una metodología ágil, caracterizada por reuniones frecuentes tanto con los usuarios finales como con el área de desarrollo. El proceso se estructuró principalmente en tres fases: diseño y análisis, implementación, y validación de la solución.

Diseño y análisis

En la fase de diseño, se llevaron a cabo una serie de reuniones para comprender a fondo el sistema SANE y su estado actual, lo que permitió analizar sus problemas y limitaciones. Tras este análisis, se determinaron los elementos que se conservarían del sistema existente y se definió el nuevo modelo de datos. Posteriormente, se realizó una reunión para presentar los diseños de las interfaces propuestas, recopilando la retroalimentación de los usuarios sobre aspectos visuales y funcionalidades. Esta información fue crucial para refinar los diseños antes de pasar a la fase de implementación.

Implementación

La implementación de la solución se dividió en tres etapas principales:

1. Desarrollo de interfaces: Se implementaron las interfaces diseñadas en la fase anterior. Se realizó una reunión con el área de comunicaciones para obtener retroalimentación temprana, lo que permitió realizar ajustes y añadir funcionalidades no contempladas inicialmente en la fase de diseño.

[7]Django REST framework: `https://www.django-rest-framework.org/`

2. Definición de modelos: Se definieron los modelos de Django correspondientes al modelo de datos establecido en la fase de diseño. En esta etapa, se comenzaron a conectar las funcionalidades y vistas correspondientes.

3. Implementación de API: Finalmente, se implementaron los endpoints de la API diseñada. Esta API tiene como función principal entregar los eventos almacenados en el sistema, con la posibilidad de aplicar ciertos filtros a las solicitudes.

Este enfoque metodológico permitió un desarrollo iterativo y centrado en el usuario, asegurando que el sistema esté en todo momento alineado las necesidades de los usuarios finales.

Validación

Una vez implementado el sistema, se procedió a evaluarlo desde dos perspectivas complementarias: la usabilidad desde el punto de vista de los usuarios finales y la mantenibilidad del sistema, centrándose en el código fuente y la documentación. Esta evaluación dual proporcionó una visión integral de la eficacia y eficiencia del nuevo sistema, abarcando tanto la experiencia del usuario como la sostenibilidad técnica a largo plazo.

Para evaluar la usabilidad, se identificó como usuarios objetivos a las periodistas y miembros del área de comunicaciones del DCC, quienes son las administradoras y usuarias regulares del sistema SANE actual y serán las principales usuarias del nuevo sistema de eventos.

Las pruebas de usabilidad consistieron en sesiones prácticas donde se solicitó a las administradoras realizar tareas similares a las que realizan en el SANE, como la creación y publicación de eventos. Tras completar estas pruebas, se aplicó la escala SUS (System Usability Scale) a ambos sistemas para obtener una evaluación rápida y confiable de la usabilidad. El objetivo establecido fue que el nuevo sistema de eventos obtuviera un puntaje superior a 68 en esta escala. Adicionalmente, se recopiló retroalimentación cualitativa de las administradoras sobre su experiencia con ambos sistemas, lo cual ayudó a identificar áreas específicas de mejora y complementó los resultados cuantitativos del SUS.

En cuanto a la evaluación de la mantenibilidad del sistema, se implementaron pruebas unitarias utilizando la librería especializada *pytest*[8]. Estas pruebas son fundamentales para garantizar la mantenibilidad, ya que permiten verificar que las modificaciones o actualizaciones futuras no comprometan la funcionalidad existente. Las pruebas unitarias abarcaron diversos casos para asegurar la correcta funcionalidad del sistema en relación con los componentes de Django y la API.

Además, se realizó una comparación de la calidad del código entre el sistema SANE actual y el nuevo sistema desarrollado. El objetivo principal fue mejorar la cohesión [6] en el nuevo sistema, considerando que este se enfoca exclusivamente en la administración y publicación de eventos. La cohesión, que representa la estrecha relación entre los componentes de un sistema, es un indicador clave de la mantenibilidad.

[8]pytest: `https://docs.pytest.org/en/7.4.x/`

Para cuantificar la mantenibilidad del sistema, se utilizaron indicadores de calidad del código como la complejidad ciclomática y el índice de mantenibilidad. Estos indicadores se calcularon mediante las librerías especializadas *radon*[9] y *code-quality*[10]. Se establecieron como objetivos un índice de mantenibilidad de 85 o superior y una complejidad ciclomática entre 1 y 5, valores que son considerados óptimos en la industria para garantizar un código altamente mantenible y de calidad.

[9]radon: `https://radon.readthedocs.io/en/latest/`
[10]code-quality: `https://pypi.org/project/code-quality/`

Capítulo 2

Estado del arte

En el ámbito universitario, la gestión y difusión de eventos y noticias juega un papel crucial en la comunicación interna y externa. Actualmente, existen tres sistemas principales que abordan esta necesidad en diferentes niveles dentro de la Universidad de Chile: el SANE, sistema propio del DCC; la plataforma Siembra, diseñada para toda la universidad; y Magnolia, utilizada a nivel de facultad. Cada uno de estos sistemas tiene características únicas y cumple funciones específicas en su respectivo ámbito.

2.1. SANE

El sistema SANE (Sistema de Administración de Noticias y Eventos) ofrece un sitio web público, como se muestra en la Figura 2.1, que permite a cualquier persona consultar noticias y eventos publicados por el área de comunicaciones del DCC. Por ejemplo, al acceder a la sección de Eventos, se presenta un listado ordenado cronológicamente, como se aprecia en la Figura 2.2.

Figura 2.1: Sitio web del SANE

Figura 2.2: Listado de eventos dentro de la web pública del SANE

El SANE cuenta con un sistema de usuarios con diferentes roles. Específicamente, los usuarios con roles de administración, pertenecientes al área de comunicaciones, tienen la capacidad de gestionar todas las noticias y eventos subidos al sitio. Esto se realiza a través de una interfaz de administración que facilita la gestión del contenido existente y la creación de nuevo material. Esta interfaz permite a los administradores manejar diversos elementos, incluyendo eventos, noticias, imágenes, documentos, lugares, organizadores, etiquetas y usuarios.

Para la creación o edición de noticias y eventos, se dispone de una interfaz interactiva de construcción. Esta interfaz es accesible a través de la sección de administración de noticias y eventos, utilizando el botón para crear un nuevo elemento (ver Figura A.1 del Anexo). La Figura 2.3 muestra la sección de construcción de eventos, que se divide en cuatro áreas principales:

- Sección A: Permite a los usuarios agregar imágenes desde el almacenamiento local o buscar imágenes cargadas por nombre, con opción de previsualización.

- Sección B: Contiene contenedores que determinan la organización de los elementos de la sección C.
- Sección C: Alberga los elementos individuales que pueden ser utilizados en la construcción del evento.
- Sección D: Es la zona donde se ensambla el evento, permitiendo arrastrar y soltar elementos de las secciones B y C.
- Sección E: Es un menú de selección con plantillas en como se ordenan los elementos de un evento.
- Sección F: Buscador de *Tags* registrados en el sistema para asociarlo a un evento.
- Sección G: Botones de configuración del evento

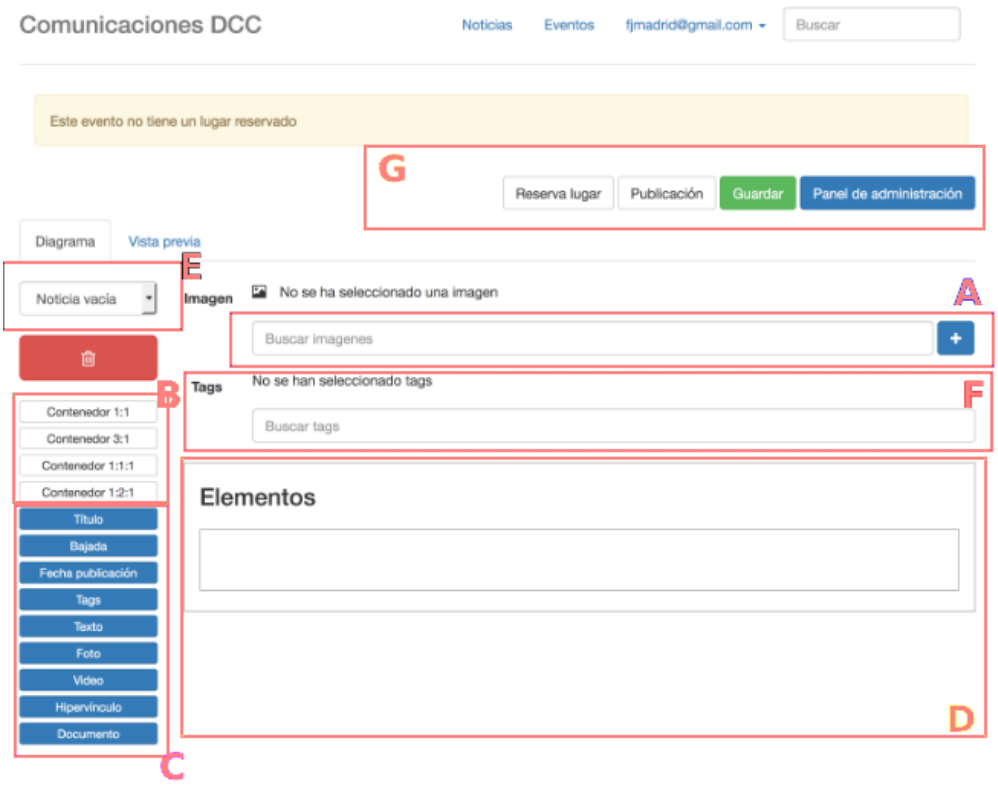


Figura 2.3: Construcción de evento desde el SANE[1], con etiquetas en sus diferentes secciones

El SANE demuestra ser un sistema complejo que maneja diversos elementos. Cuenta con un sitio web público y un apartado de administración tanto para noticias como para eventos. Sin embargo, en el apartado del constructor de eventos, la cantidad de elementos sugiere que el sistema podría estar sobrecargado en relación con la forma en que se presentan los eventos finalmente. En el próximo Capítulo 3 se ofrece un análisis más profundo sobre las problemáticas de este sistema y se detalla cómo se diseñó la solución implementada que se describe en el Capítulo 4.

[1]Fuente: Memoria de F. Madrid [10].

2.2. Plataforma SIEMBRA

La plataforma SIEMBRA es un proyecto de mejoramiento de la gestión institucional encabezado por la Vicerrectoría de Extensión y Comunicaciones de la Universidad de Chile (VEXCOM). Nace con la intención de fortalecer la política y los mecanismos que orientan la labor universitaria en los ámbitos de la extensión, la vinculación con el medio y las comunicaciones. Fue implementada por primera vez en el marco de la memoria de Darío Cáceres en el año 2021 [9]. Posteriormente, se continuó desarrollando durante los siguientes semestres en el curso de Ingeniería de Software II (CC5401), hasta su estreno a principios de enero de 2024 para la comunidad de la Universidad de Chile. SIEMBRA permite gestionar y dar a conocer las múltiples iniciativas de extensión, vinculación con el medio y comunicaciones desarrolladas por docentes, estudiantes y funcionarios/as de la Universidad.

Al acceder al sitio web[2], un usuario con credenciales válidas de la universidad puede ver un listado de las iniciativas que ha creado, junto con su estado, como se muestra en la Figura 2.1. Además, existe una pestaña para las iniciativas de extensión incompletas, permitiendo a los usuarios continuar con su elaboración.

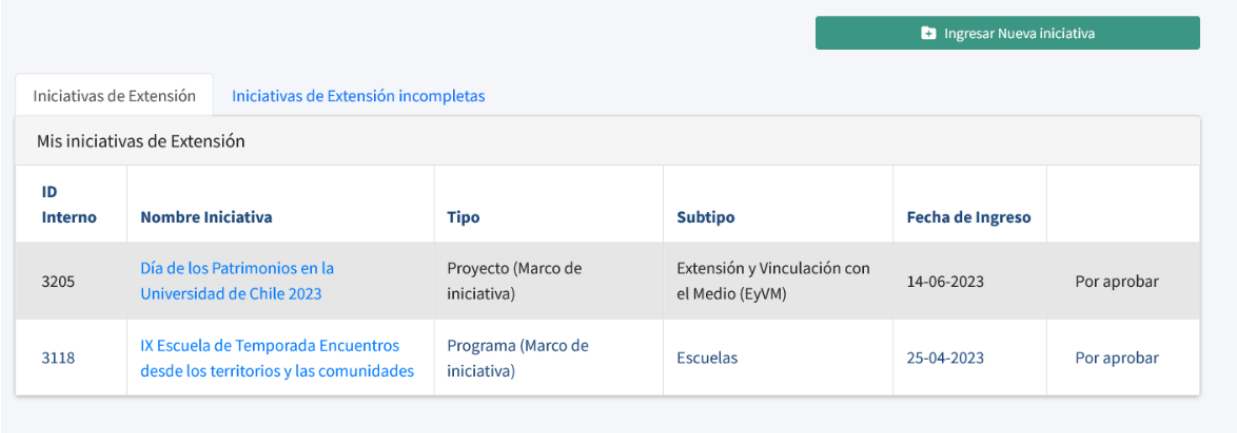


Figura 2.1: Tabla con el listado de iniciativas del usuario

Para crear una nueva iniciativa, el usuario puede hacer clic en el botón "Ingresar nueva iniciativa" ubicado encima de la tabla en la Figura 2.1. Al hacerlo, se muestra un breve instructivo sobre el uso del sistema, como se aprecia en la Figura 2.2.

[2]Plataforma Siembra: `https://plataformasiembra.uchile.cl/`

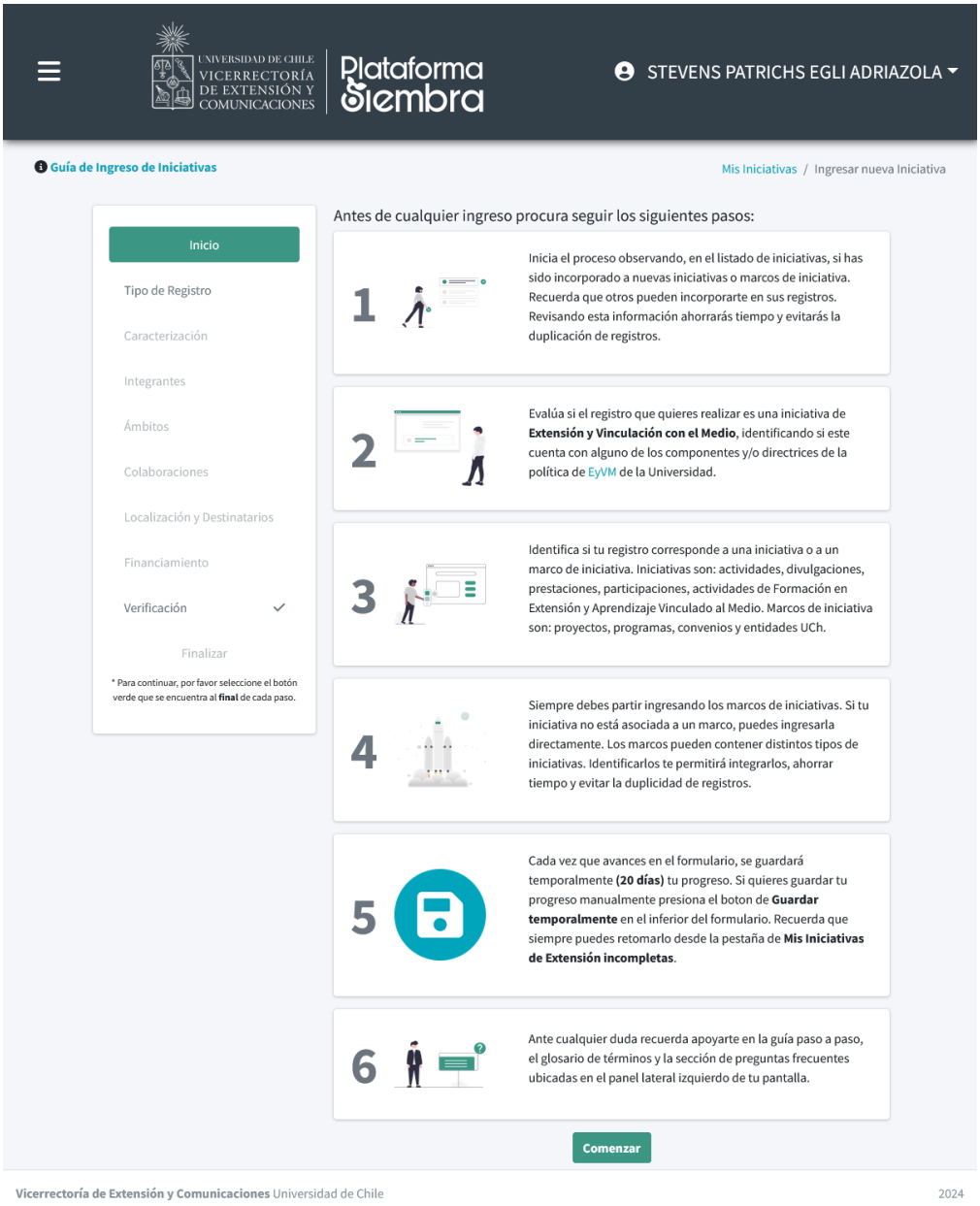


Figura 2.2: Instructivo al ingresar una nueva iniciativa en Siembra

Una vez que el usuario comienza el ingreso de datos, debe completar numerosos campos y formularios sobre la iniciativa, como se ilustra en la Figura 2.3.

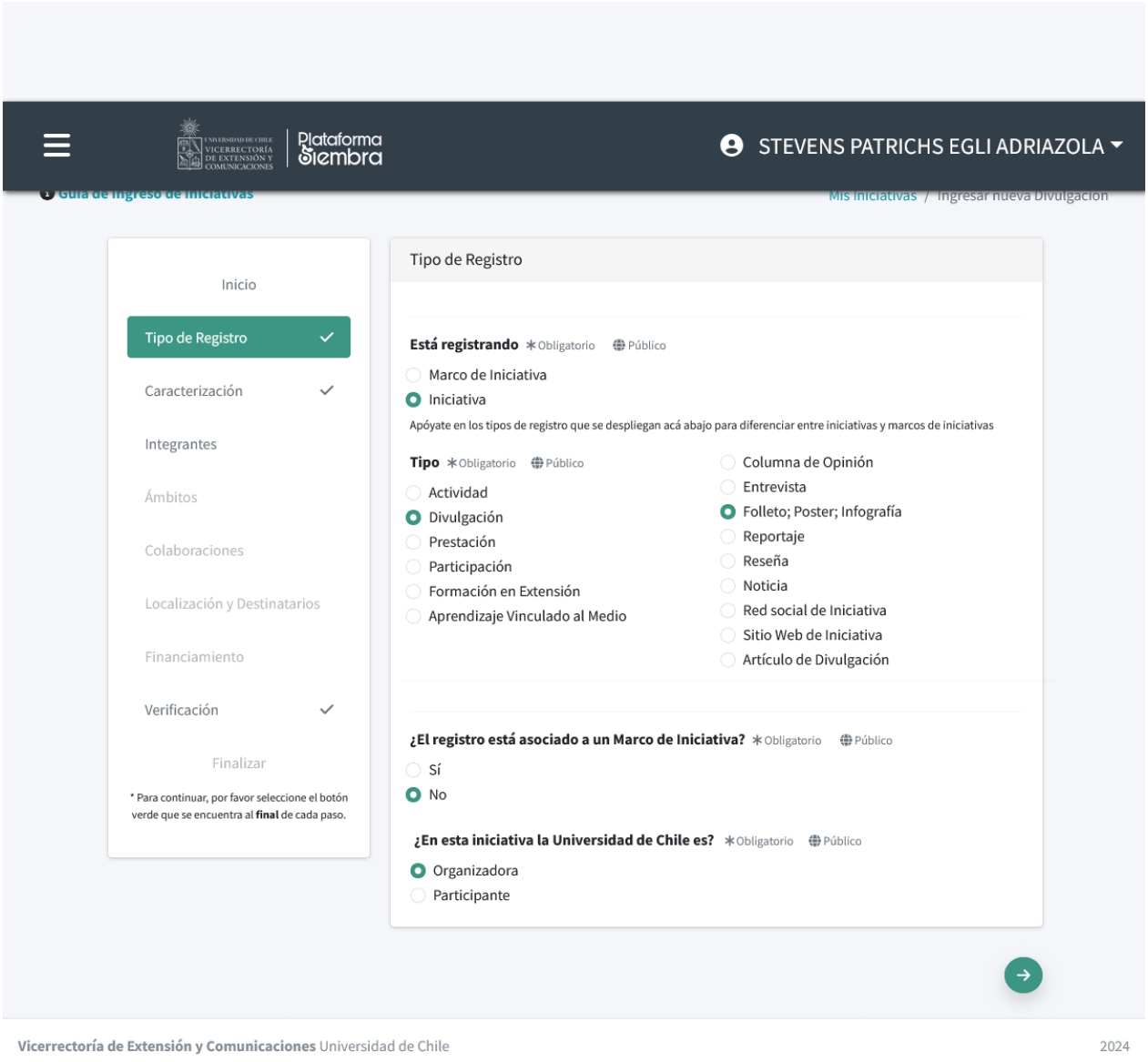


Figura 2.3: Iniciativa siendo completada en los formularios de Siembra

Aunque el sistema tiene como objetivo principal permitir la carga de iniciativas, incluyendo eventos, actualmente la plataforma no cuenta con una forma de navegar entre todo el contenido subido ni posee una API pública para extraer esta información de ninguna manera.

2.3. Magnolia

Magnolia, también conocida como *agenda*, es un sistema que permite subir eventos y noticias en los sitios asociados al sitio web de la Universidad de Chile, específicamente en la página web de la Facultad de Ciencias Físicas y Matemáticas (FCFM). Este sistema facilita la publicación de diversos tipos de eventos como conferencias, presentaciones, encuentros y exposiciones. Una característica importante es que al publicar en la sección de eventos del

sitio web de la FCFM, automáticamente se publica también en los eventos del sitio web de la Universidad de Chile.

Magnolia concentra eventos a nivel de facultad, incluyendo a todos los departamentos. La web cumple la función de agenda y permite a los usuarios agendar notificaciones dependiendo del evento de interés mediante su web[3]. La interfaz que se muestra a cualquier usuario que accede se puede ver en la Figura 2.1, y el detalle de un evento específico se presenta como en la Figura 2.2.

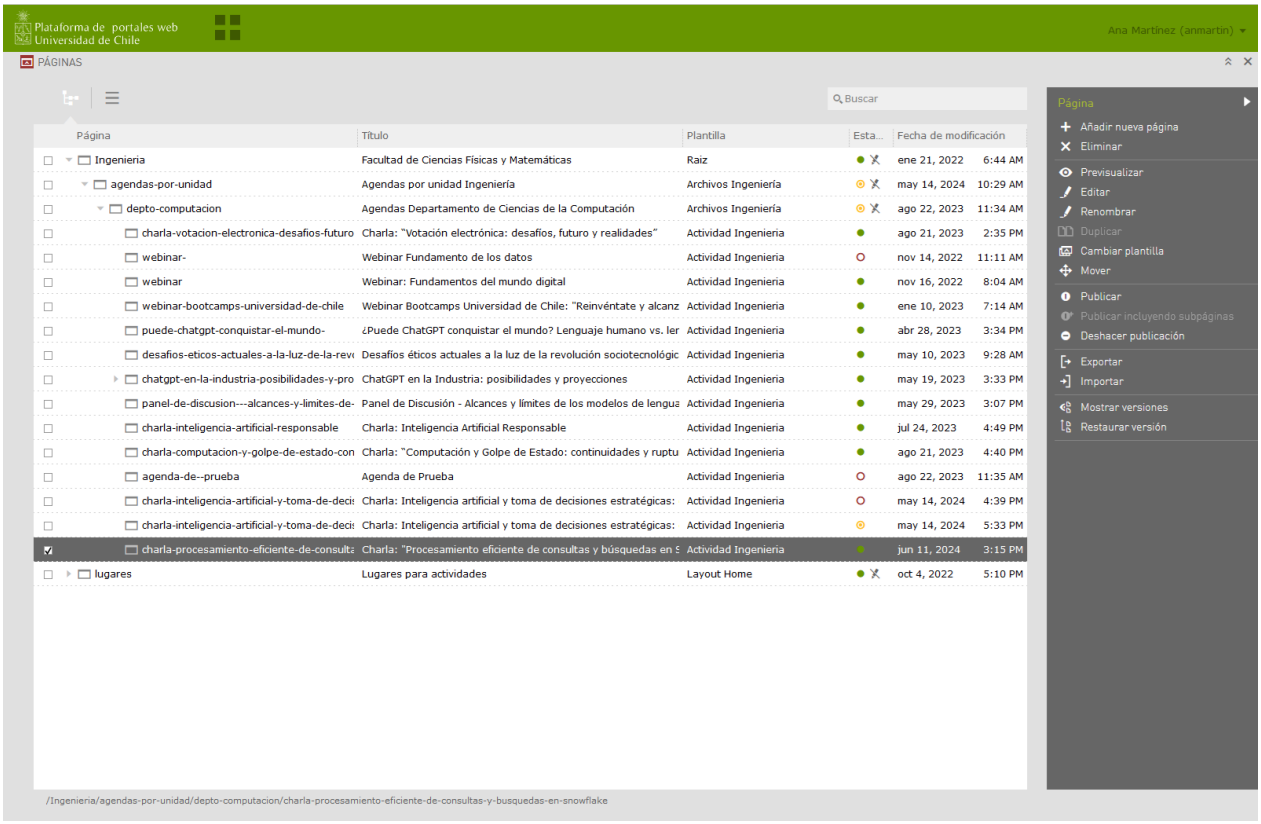


Figura 2.1: Listado de eventos en la web pública de Magnolia

[3]Agenda - Facultad de Ciencias Físicas y Matemáticas: https://ingenieria.uchile.cl/agenda

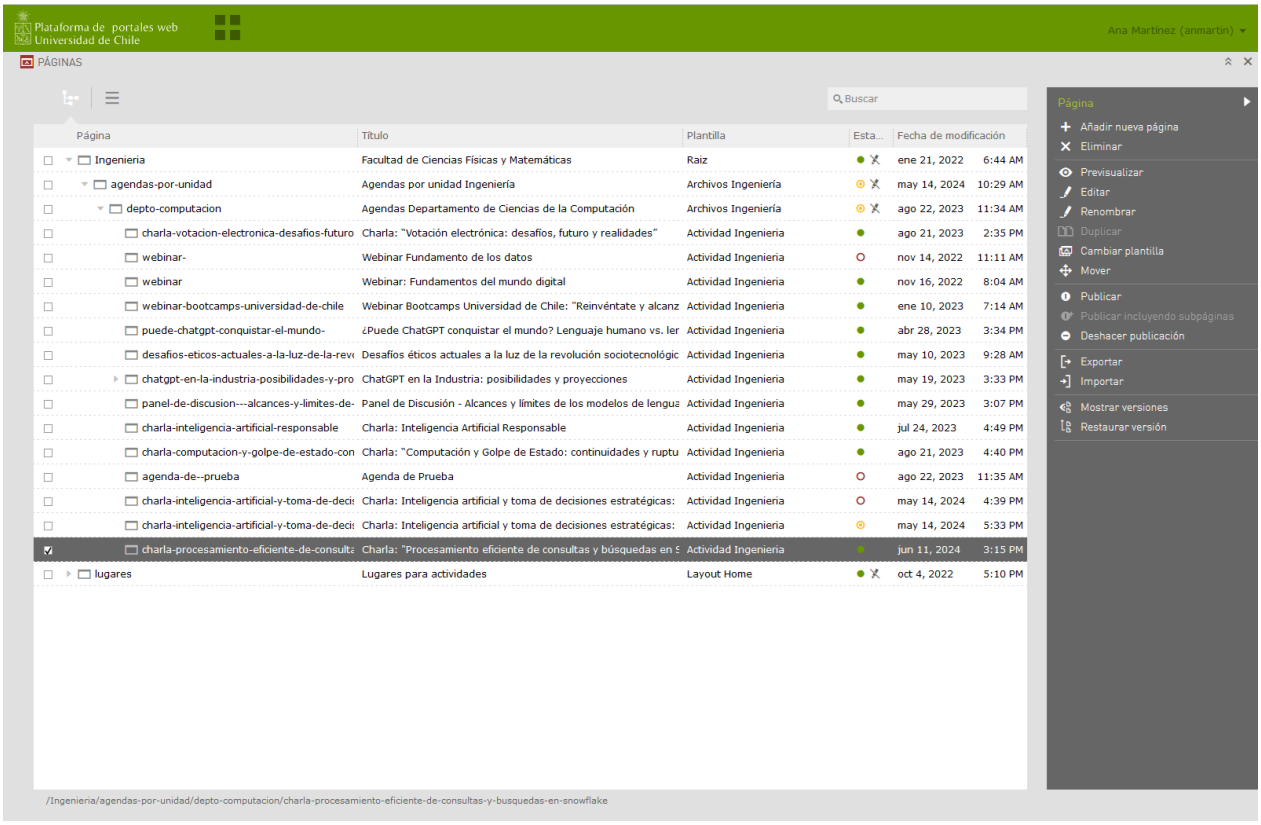


Figura 2.2: Detalle de un evento en la web pública de Magnolia

A pesar de que el área de comunicaciones solo tiene acceso al apartado de eventos, Magnolia presenta funcionalidades similares a las del SANE. El sistema de administración ofrece una vista que muestra un listado de los eventos, como se puede observar en la Figura 2.3.

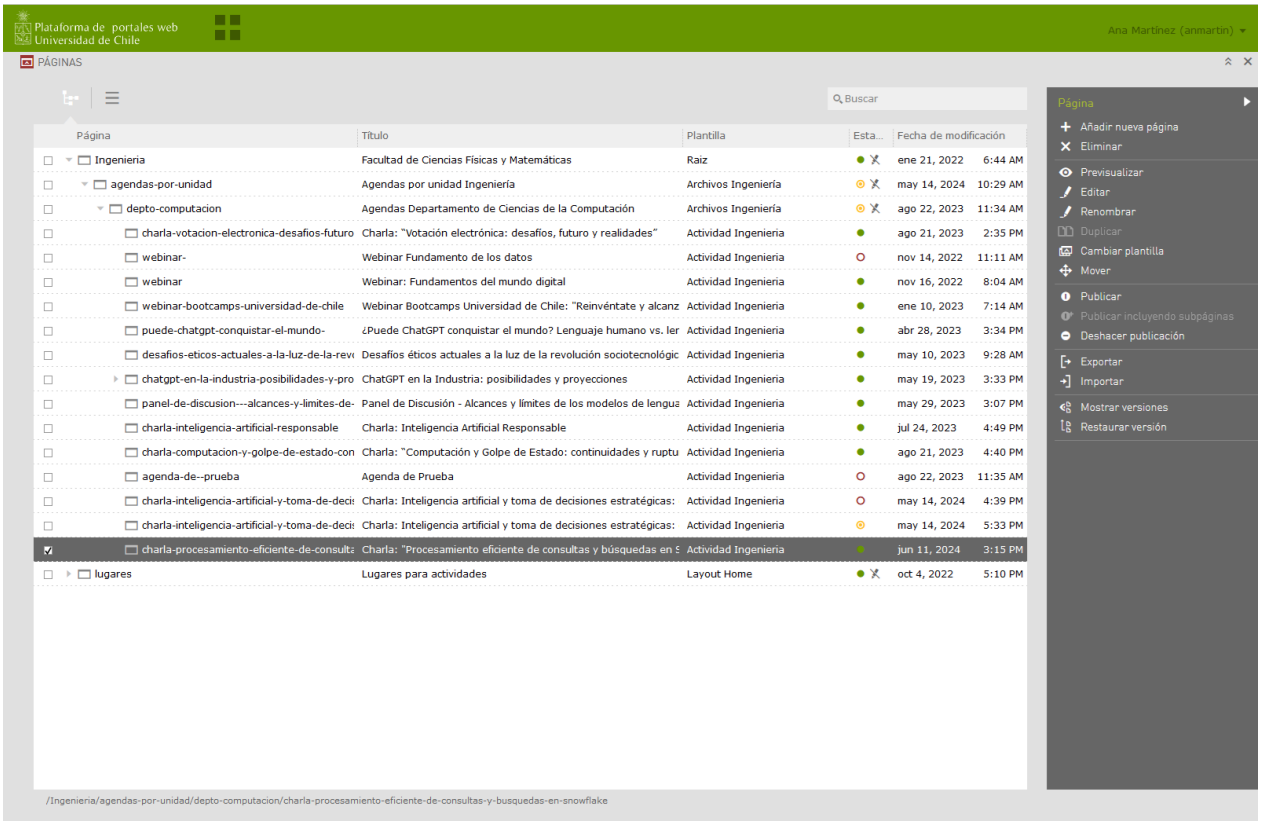


Figura 2.3: Listado de eventos en el panel de administración de Magnolia

En esta vista de administración, se presenta un panel al lado izquierdo que permite editar, entre otras opciones, un evento existente o crear uno nuevo. La creación de un nuevo evento se realiza a través de una interfaz específica, como se muestra en la Figura 2.4.

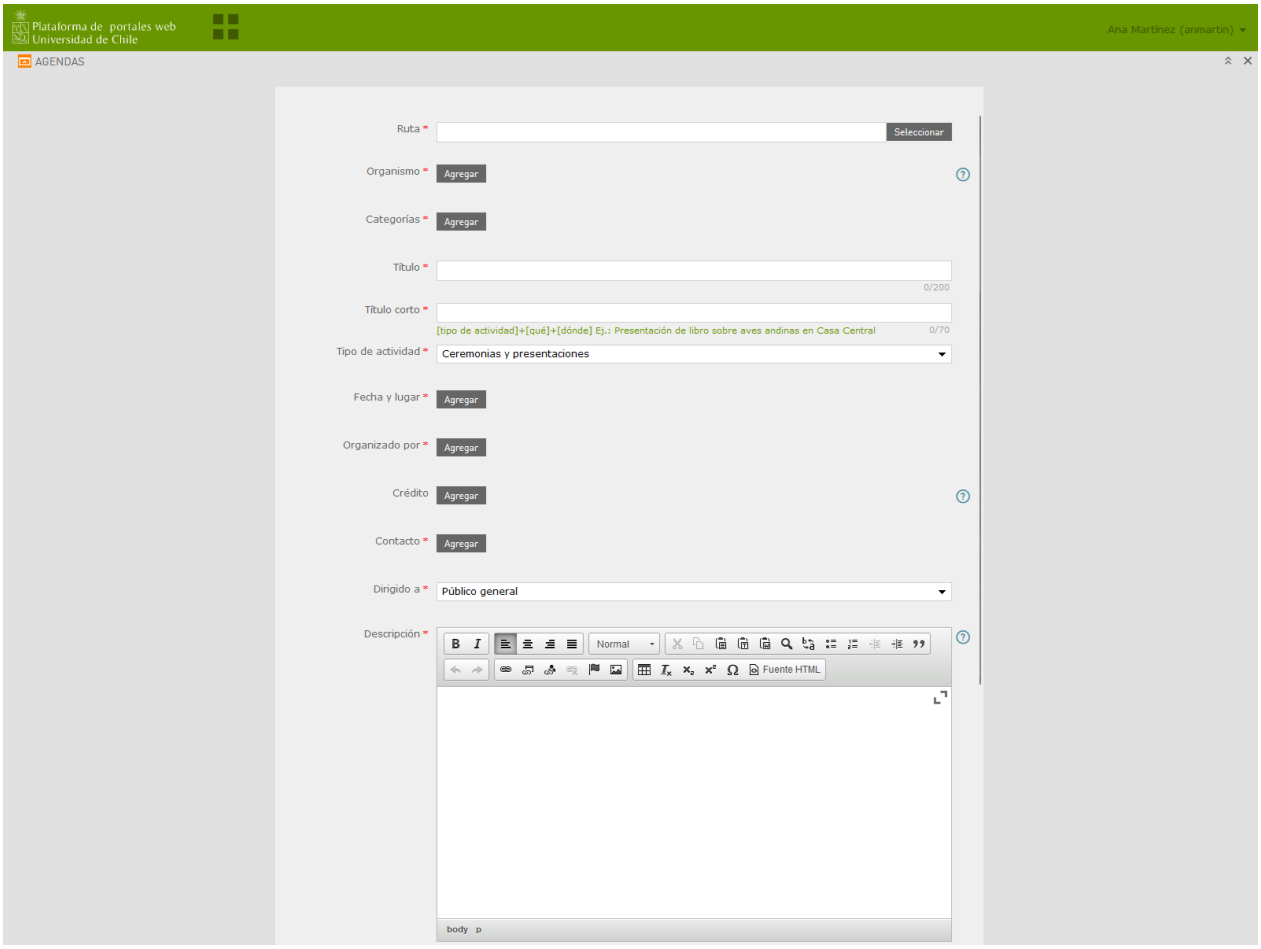


Figura 2.4: Creación de un evento desde el panel de administración de Magnolia

Para la edición de eventos existentes, Magnolia proporciona una vista dedicada, ilustrada en la Figura 2.5.

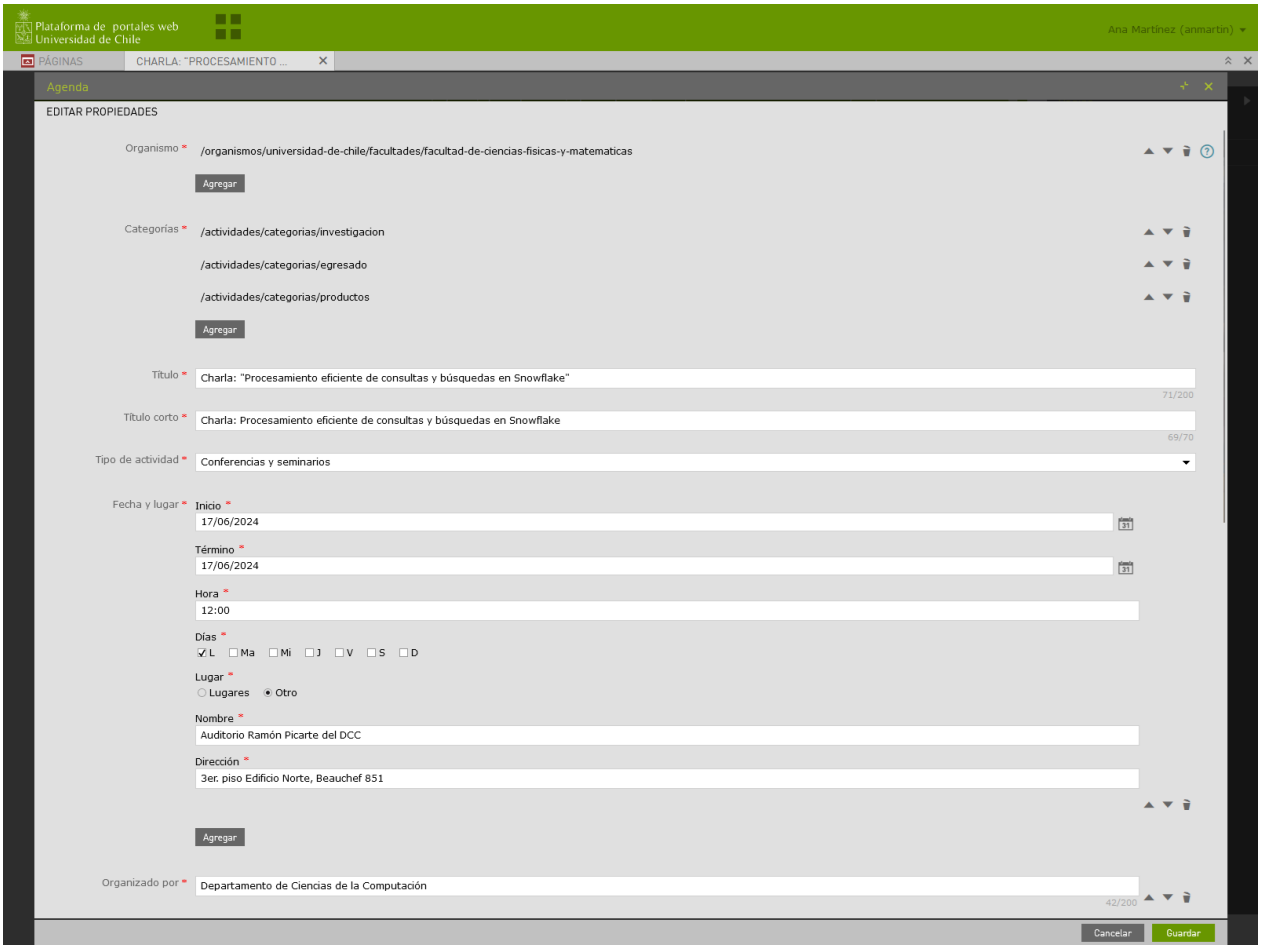


Figura 2.5: Edición de un evento desde el panel de administración de Magnolia

Además de la gestión de eventos, el sistema ofrece la opción de manejar los archivos subidos. Esta funcionalidad permite crear directorios y subir archivos al sistema, como se puede apreciar en la Figura 2.6.

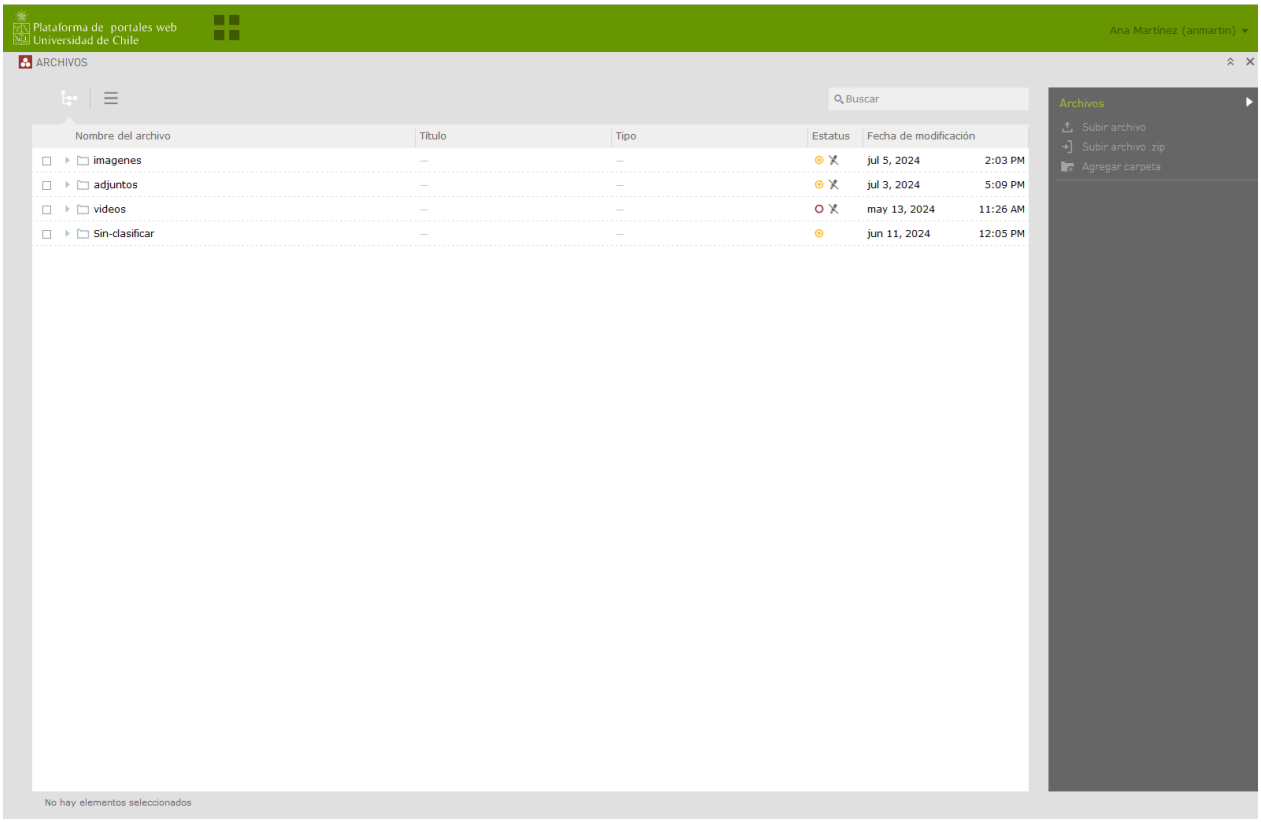


Figura 2.6: Manejo de archivos desde el panel de administración de Magnolia

Este sistema, a pesar de ser muy similar al SANE, no es parte del DCC. Además, solamente alimenta ciertos sitios de la Universidad de Chile y no dispone de una API que permita extraer estos eventos para conectarse a otros sistemas externos, como por ejemplo la web del DCC o sistemas más específicos como las televisiones del departamento. Esta limitación no permite una difusión más localizada a nivel del DCC.

Capítulo 3

Análisis y Diseño

En el Capítulo 3 se discutieron varios sistemas que realizan tareas de administración relacionadas con eventos. Esta memoria se enfoca específicamente en el desarrollo de un sistema que busca solucionar varios de los problemas que posee el SANE. Este capítulo detalla el trabajo realizado: la Sección 3.1 incluye una inspección exhaustiva del SANE para comprender tanto la estructura del modelo de datos como la API actual, así como los problemas que presentaba el sistema en cuanto a sus funcionalidades y usabilidad en general. Luego, en la Sección 3.2 se presenta el trabajo de diseño realizado para desarrollar la solución que se implementó, dividido en varias componentes.

3.1. Análisis

A continuación, se abordarán los problemas del SANE, enfocándose particularmente en la sección del sistema que permite la administración de los eventos. Luego, se enlistarán los elementos que este nuevo sistema de eventos buscará solucionar, cuya implementación se describe más adelante en el Capítulo 4.

3.1.1. Problemas de SANE

El sistema SANE, a pesar de su alto nivel de personalización, tiene características que rara vez se utilizan. Esto complica la interfaz sin proporcionar un beneficio claro. Por ejemplo, los administradores, quienes pertenecen al área de Comunicaciones del DCC, generalmente utilizan las opciones preconfiguradas de la sección E, en lugar de los elementos que aparecen en las secciones B y C de la interfaz de la Figura 2.3 del Capítulo 2.

Por otro lado, el sistema cuenta con una función para administrar etiquetas o tags. Estas etiquetas se asocian en la creación de una noticia o evento a través de la sección F de la Figura 2.3 y permiten a los usuarios que se registran en la web del SANE seleccionar sus intereses en sus perfiles. La idea de esta funcionalidad era que, cuando se publicaran noticias o eventos relacionados con estas etiquetas, se notificaría a los usuarios a través de un sistema

de notificaciones. Sin embargo, este sistema de notificaciones nunca llegó a producción, por lo que esta característica carece de utilidad y los eventos y noticias se suben sin los *tags* asociados.

Para publicar un evento, los administradores deben completar varias opciones (ver la Figura B.1 en el Anexo). No queda claro el propósito de la opción de categoría, ya que su uso parece ser más bien interno del sistema. La función de reservar lugares, ubicada en el botón "Reservar lugar", opera mediante una integración con el Sistema de Administración de Recursos (SAR) del DCC, pero no funciona correctamente y está limitada a lugares dentro del DCC, lo que la hace engorrosa para los administradores. Además, los elementos de las listas para otros campos están en el orden en que se agregaron, como por ejemplo las imágenes o los organizadores, lo que dificulta encontrar elementos específicos cuando la lista es larga y no se posee algún tipo de buscador dentro de estas.

El sistema tiene limitaciones significativas, como no permitir eventos ni imágenes con el mismo nombre, incluso si son para fechas diferentes. Además, la carga de imágenes de alta resolución a menudo falla sin proporcionar alertas, lo que es un problema de usabilidad. La administración de lugares no es adecuada para eventos remotos, ya que los usuarios deben ingresar estos eventos ubicados como lugares físicos, lo que no es apropiado para este tipo de evento. Además, la recopilación de datos de organizadores, como números de contacto, puede plantear preocupaciones de privacidad, ya que es un campo obligatorio para poder publicar un evento. Por otro lado, existen algunos problemas técnicos y diferencias que ocurren en ciertos navegadores, como el bloqueo del cursor al editar componentes de texto de un evento o fallos en la paginación de elementos.

En cuanto a las tecnologías utilizadas en el SANE, estas se encuentran bastante obsoletas, incluyendo Django 1.11.2 para el backend, AngularJS 1.6.4 para el frontend y PostgreSQL 9.5.7 para la base de datos. Estas versiones están significativamente desactualizadas y datan del año 2017, lo cual plantea desafíos en términos de seguridad y soporte [17].

En resumen, el sistema SANE actual, aunque funcional, presenta varios desafíos y limitaciones que necesitan ser abordados. Desde la obsolescencia de las tecnologías utilizadas hasta la complejidad de la interfaz y las funcionalidades innecesarias, estos problemas subrayan la necesidad de un nuevo sistema que posea una usabilidad mejor que la del actual. El nuevo sistema buscará simplificar la interfaz, eliminar las funcionalidades innecesarias y mejorar las existentes, al tiempo que se actualizará a tecnologías más recientes y seguras. Con estos cambios, se espera que el nuevo sistema sea más útil para los usuarios y más fácil de mantener para el área de desarrollo.

3.1.2. Requisitos Abordados en la memoria

Con todos estos problemas descritos dentro del SANE se Abordaron de forma enlistaron los siguientes problemas que se abordaron en esta memoria:

- Diseño de un sistema que contemple únicamente eventos
- Simplificación de la interfaz destinada a la administración de eventos, especialmente en

la sección de construcción de eventos.

- Simplificación del modelo de datos actual a uno que considere solamente eventos.
- Simplificación de las funcionalidades y ajustes según los comentarios de los usuarios finales.
- Creación de un sistema a partir de tecnologías más recientes y actualizadas, bajo los lineamientos del área de desarrollo del DCC
- Disponibilidad una API REST que permita al nuevo sistema poder comunicarse con otros sistemas

3.2. Diseño

Para abordar los puntos mencionados anteriormente, se diseñó una solución integral que abarca varios aspectos del sistema. El proceso de desarrollo siguió una secuencia lógica, comenzando con la elaboración de los diseños de la interfaz de usuario, detallados en la Sección 3.2.1. Posteriormente, se realizaron modificaciones al modelo de datos, tomando como referencia parte del modelo actual del SANE, cambios que se describen en la Sección 3.2.2. Además, se diseñó la funcionalidad de la API REST del sistema, crucial para establecer comunicaciones con otros servicios, cuya descripción se encuentra en la Sección 3.2.3.

3.2.1. Diseño de las interfaces

Para el diseño de las interfaces del sistema, se recurrió a Figma[1], una herramienta que facilita la creación y traducción de diseños de interfaces de usuario a código. Estos diseños se perfeccionaron en varias reuniones con los miembros del área de comunicaciones del DCC, específicamente con las usuarias finales, Ana Martínez, periodista del DCC, y Paulette Filla, diseñadora gráfica.

Se presentaron todos los diseños a ellas, explicando el flujo propuesto para la aplicación y discutiendo los elementos que se mantendrían del SANE en este nuevo sistema. La retroalimentación obtenida permitió corregir varios elementos en términos de contrastes y funcionalidades, e incorporar estilos y diseños similares a los que se utilizan en la mayoría de los sistemas del DCC. Como resultado de este proceso colaborativo, se obtuvieron las siguientes interfaces:

Administrar eventos

Al iniciar sesión en la aplicación, un usuario del área de comunicación que quiere obtener una visión general de las actividades para poder tener una mejor gestión de los eventos puede revisar la tabla de eventos que se muestra en la Figura 3.1. Esta tabla, similar a la que se

[1]Sitio web de Figma: `https://www.figma.com/`

utiliza en el SANE para los eventos (Figura A.2 del Anexo A), permite visualizar los eventos publicados o que están por publicarse, indicados con el símbolo más a la derecha de la tabla.

Además, tiene acceso a la opción de editar un evento presionando el botón azul con un icono de lápiz, o eliminarlo en caso de no ser necesario, usando el botón rojo situado al lado del botón de edición. En este caso, la tabla presenta una opción de filtrar los elementos de las columnas, además de poder filtrar por las palabras tanto del título como del contenido dentro de un evento, utilizando el recuadro situado encima de la tabla. Todos estos elementos mostrados en la tabla tendrán una paginación para la facilidad de navegación entre todos los eventos. las columnas de esta tabla se mantienen de la vista de administración de eventos del SANE, donde la novedad es la mejora al buscador, el cual no funcionaba correctamente y el añadido del indicador publicado o no publicado.

En el caso de querer crearon nuevo evento, el usuario tiene la opción apretada el boton "Nuevo evento" mostrará la vista de crear un evento de la Sección 3.2.1.

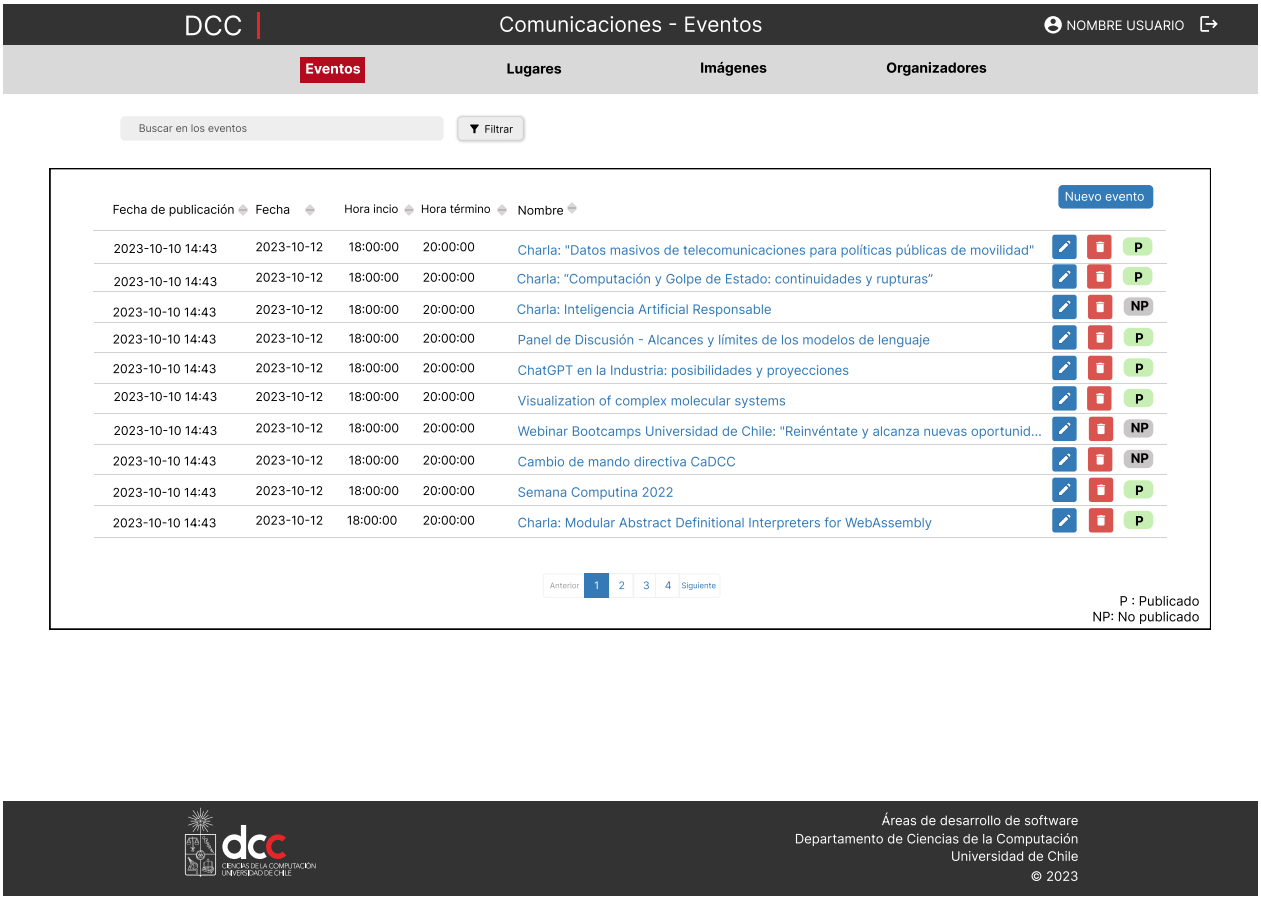


Figura 3.1: Vista de administración de eventos.

Crear eventos

En el caso de que un usuario del área de comunicaciones desee tener la capacidad de crear nuevos eventos de manera eficiente, puede optar por rellenar un formulario sencillo a través de un modal, un componente que aparece sobre el contenido de la página o pantalla que se

está visualizando, como se muestra en la Figura 3.2. Esto contrasta con el SANE, donde uno debe acceder directamente a la vista que arma los eventos de la Figura 2.3, y además, es necesario guardarlos, de lo contrario, se pierde la información.

Una vez ingresado un nombre o titular de evento, se muestran las siguientes opciones de plantilla que preconfiguran los elementos de la construcción de un evento:

- Un evento normal contiene un título, que corresponde al nombre del evento, los expositores, el contenido del texto o cuerpo del evento y una imagen pequeña a la derecha de los elementos.
- Un evento afiche contiene los mismos elementos, solo que debajo de estos se encuentra la imagen o foto con un tamaño grande.
- Sin plantilla no contiene elementos de base

Si luego se presiona el botón "Crear evento", se cambia a la vista descrita en la Sección 3.2.1. En el caso de que se presione el botón de cancelar, se cierra el modal y se mantiene la vista de administración de eventos descrita en la Sección 3.2.1.

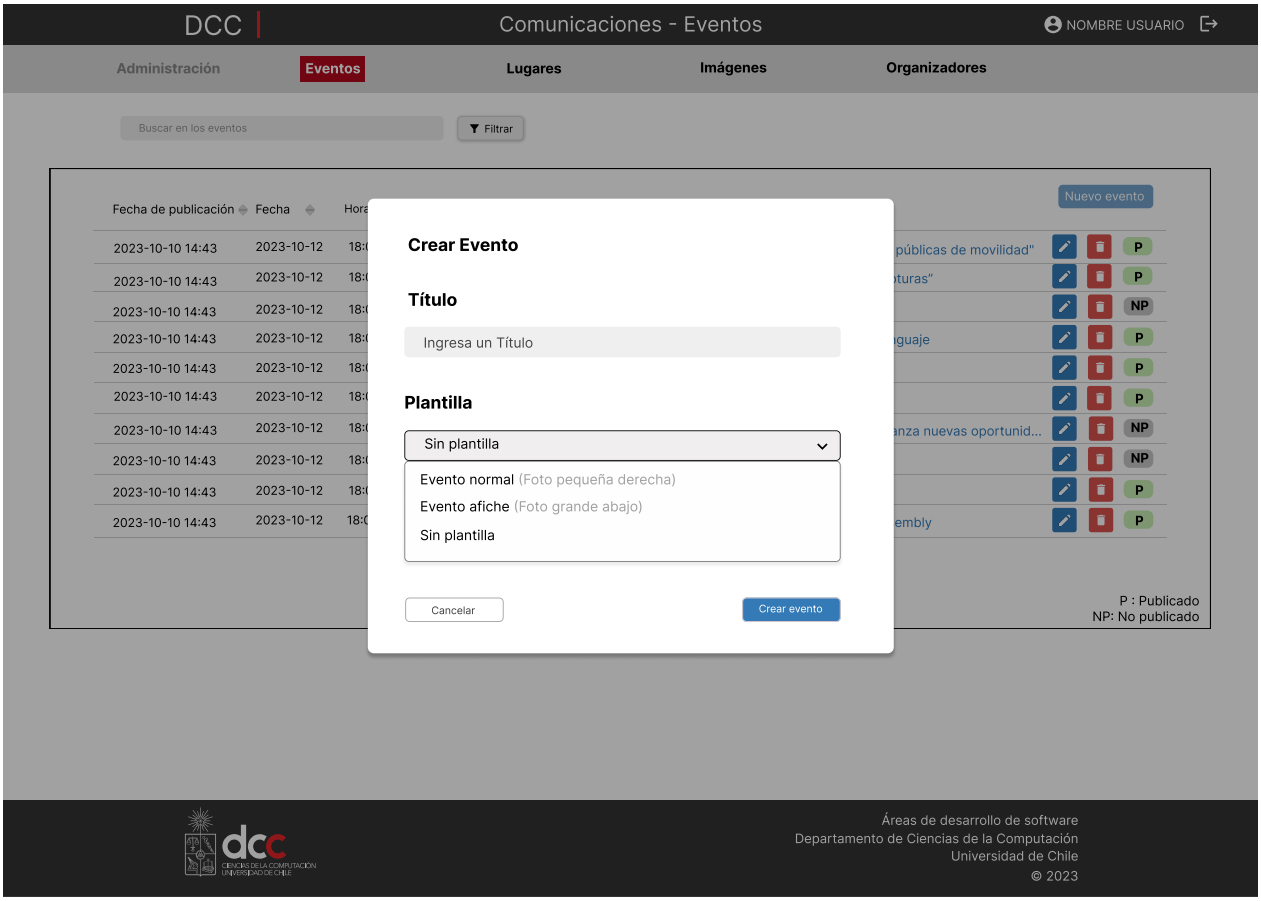


Figura 3.2: Vista de administración de eventos, luego de presionar el botón "Nuevo evento".

Administrar lugares

Si algún miembro del área de comunicaciones necesita administrar los lugares registrados en el sistema para poder agregar, eliminar o editar estos, puede hacerlo presionando "Lugares" en el encabezado de la vista. Aquí encontrará una tabla similar a la de la vista de administración de eventos de la sección 3.2.1, donde también está la opción de editar y eliminar, como se muestra en la Figura 3.3.

En comparación con el SANE, se mantienen el nombre del lugar y la dirección. Sin embargo, para estandarizar los lugares y evitar que los detalles se mezclen con el nombre del lugar o la dirección, como ocurre en el SANE con los datos ingresados, se agregó una columna de "detalles". Esta columna contiene indicaciones adicionales que permiten ubicar el lugar dentro de la dirección.

En el caso de querer agregar un nuevo lugar, se puede hacer clic en el botón de "Nuevo lugar". Esto mostrará un modal similar al de la vista de creación de un evento de la Sección 3.2.1, donde los campos a rellenar son el nombre del lugar, su detalle y la dirección. En el caso de editar un lugar existente de la tabla, se muestreará un modal con los mismos campos pero con los datos rellenados y con un botón para guardar los cambios.

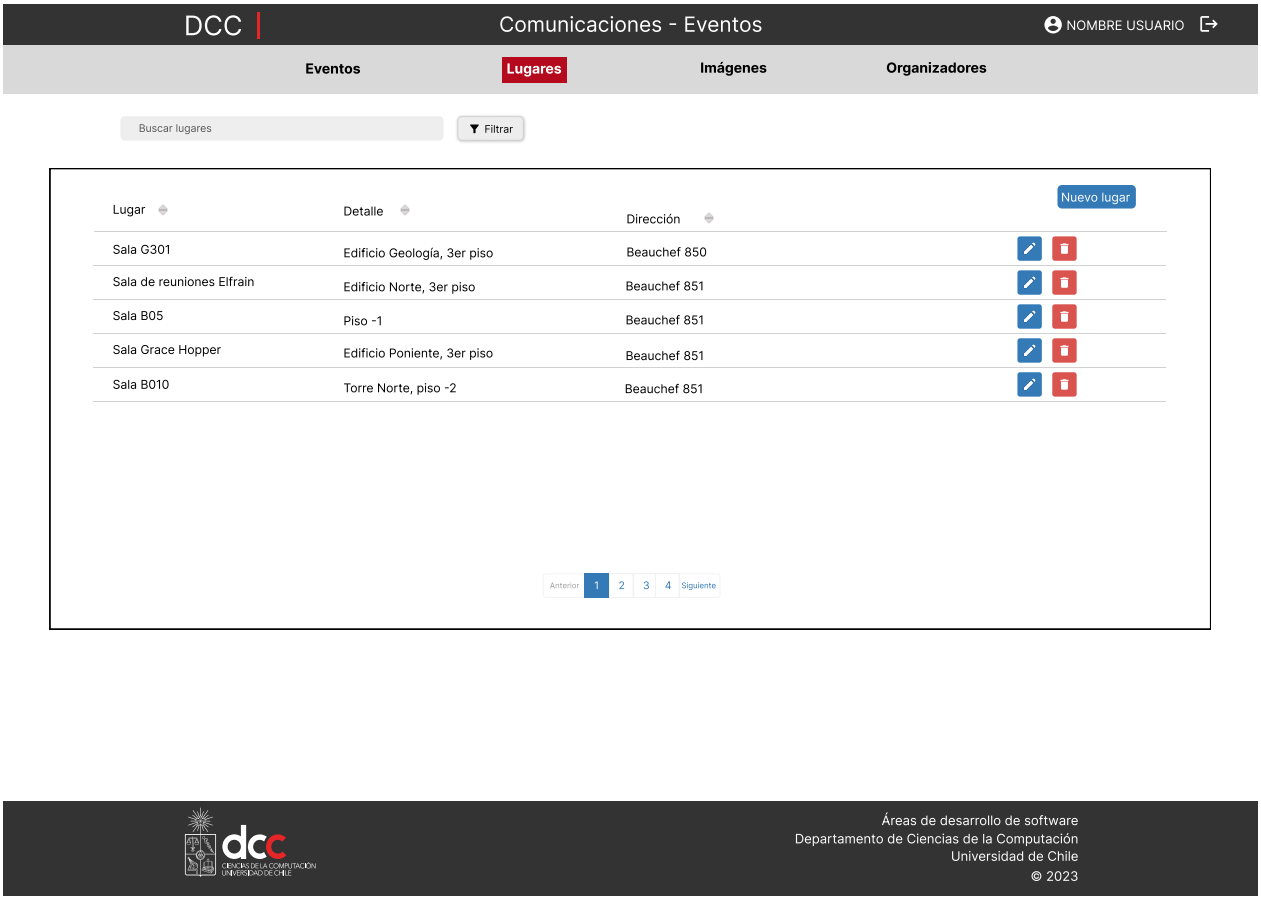


Figura 3.3: Vista de administración de lugares.

Administrar imágenes

En el caso de que un usuario necesite administrar imágenes sobre eventos para organizar un repositorio imágenes, puede ingresar a la vista de administración de imágenes presionando sobre "Imágenes" en el encabezado. Aquí, al igual que en las tablas de la Sección 3.2.1 y Sección 3.2.1, está la opción de editar y eliminar un elemento como se ve en la Figura 3.4.

Al igual que el SANE, se mantiene una tabla para administrar los elementos con los botones de editar y eliminar, y su nombre. La diferencia es que aquí se agregó una columna para poder filtrar las imágenes según su fecha de subida, además del nombre usando el buscador de la parte superior, ya que en este nuevo sistema se permitirá la repetición de nombres al ser ingresado, algo que en el SANE no es posible. También se le agrega un pequeño modal que se muestra al colocar el cursor encima del icono de imagen a la izquierda de la fila de cada elemento, que muestra una pequeña visualización de esta.

Luego, si el usuario lo requiere, al igual que en las secciones anteriores, dispone en esta vista un botón "Nueva imagen", que despliega un modal similar a la Sección 3.2.1 que permite subir una imagen y ponerle un nombre para identificarla.

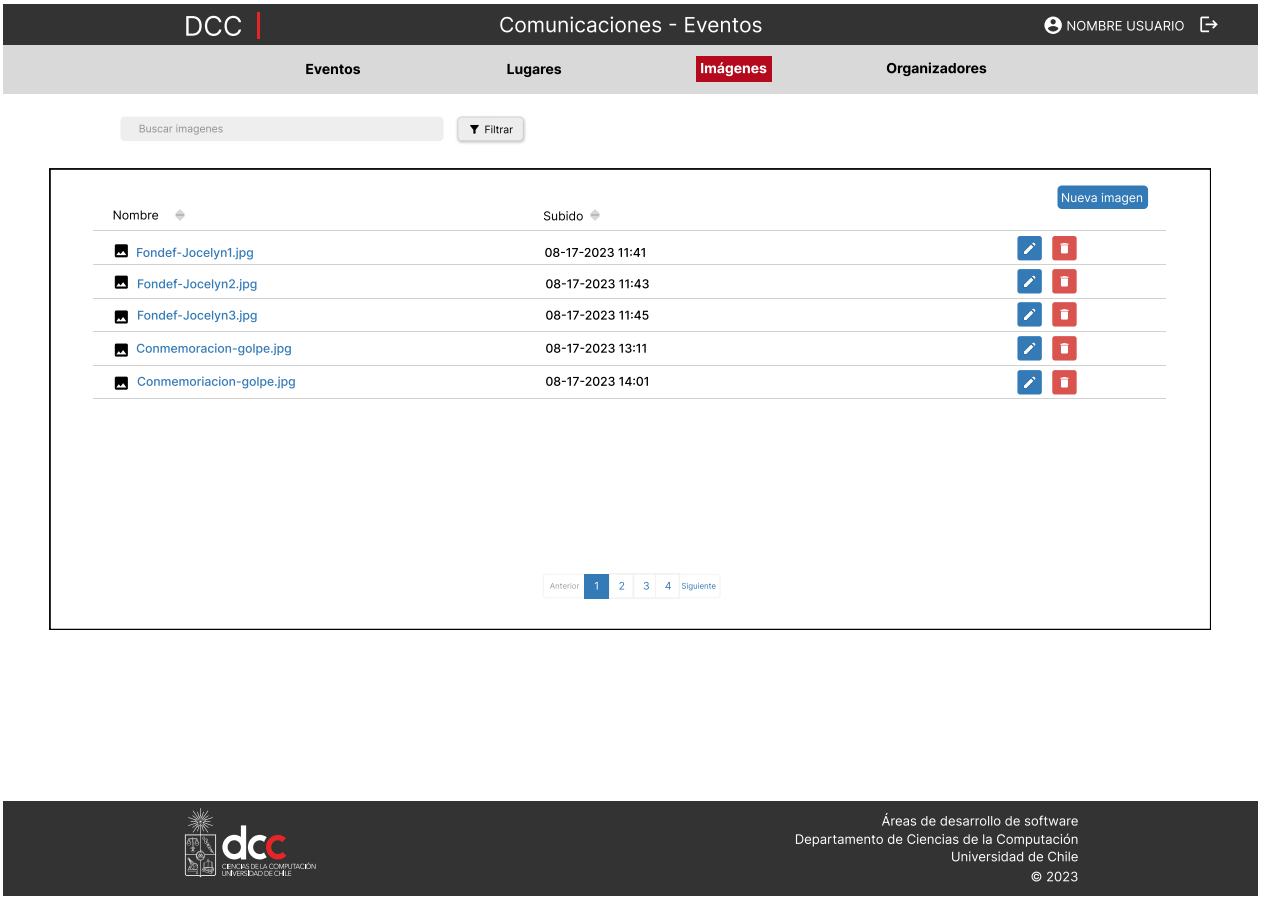


Figura 3.4: Vista de administración de imágenes.

Administrar organizadores

En el caso de que un usuario del área de comunicaciones necesite administrar los organizadores registrados ,es decir, añadir, eliminar o editar organizadores, puede acceder a la vista de administración de organizadores de eventos seleccionando “Organizadores” en el encabezado. Al igual que en las secciones de administración anteriores, se dispone de una tabla para ver los organizadores ingresados en el sistema, como se muestra en la Figura 3.5.

Los elementos que se han incorporado del SANE son las columnas de nombre y correo en las tablas. Aunque el SANE incluye una columna adicional para el número de teléfono, esta no se ha agregado en este caso. La razón es que este dato no siempre está disponible para el área de comunicaciones y, por cuestiones de privacidad, se decidió junto con las usuarias finales y lo mencionado en la Sección 2, que era preferible no incluir este dato. En muchos casos, se utilizaba un teléfono del DCC en lugar de un número privado, ya que este era un campo obligatorio. En este nuevo sistema, se elimina esa obligatoriedad de proporcionar todos los datos. En su lugar, solo será obligatorio proporcionar un nombre, y el correo será un campo opcional.

Finalmente, si se quiere añadir un nuevo organizador, al igual que en las secciones anteriores, existe un botón “Nuevo organizador” el cual despliega un modal similar al de la vista de la Sección 3.2.1 que permitirá ingresar el nombre del organizador y si es necesario su correo.

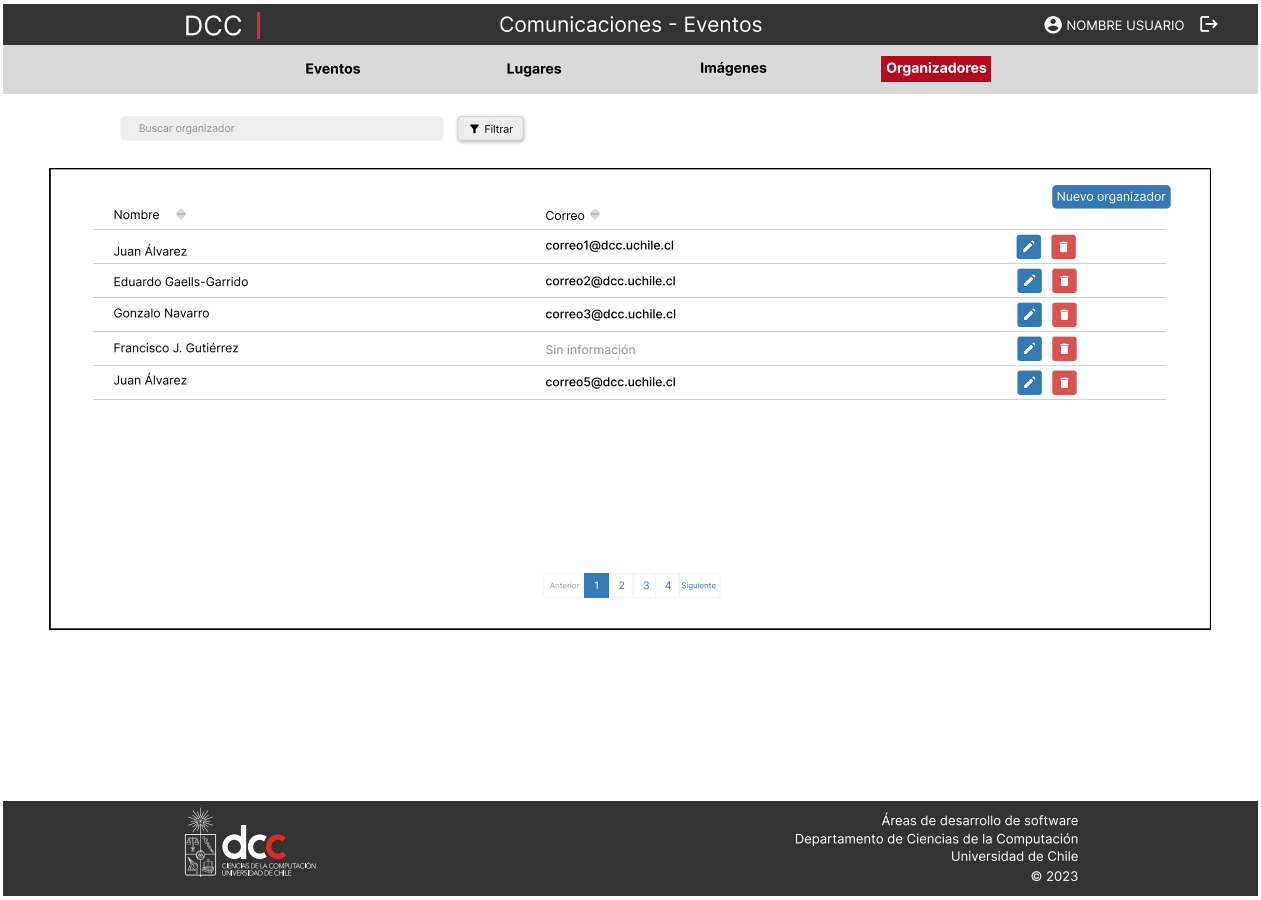


Figura 3.5: Vista de administración de organizadores.

Constructor de eventos

Si un usuario del área de comunicaciones quiere crear un nuevo evento con información, este puede ingresar a la vista de la Figura 3.2 y presionar el botón "Crear evento". Se verá la vista de la Figura 3.6, donde el nombre del evento será el título ingresado en el formulario. Otra forma de llevar a esta vista, es en el caso de que un usuario quiera editar un evento para complementar la información, este puede hacerlo apretando el botón de edición en la Sección 3.2.1. Por defecto, en ambos casos se muestra seleccionada por defecto la pestaña "Editor" en la parte superior.

En relación con el SANE, se han conservado diversos elementos de configuración. En este caso, y tomando como referencia una parte del diseño de la Figura 2.3, se mantendrán los elementos de configuración de la fecha de publicación, realización y el organizador. A diferencia del SANE, que requiere presionar un botón adicional para abrir un modal y poder configurar estos elementos, como se explica en la Sección B del Anexo, ahora se configuran directamente desde la misma vista. Se han añadido campos nuevos, como la modalidad, que es un menú de selección con las opciones Remoto, Presencial e Híbrido, y un enlace en caso de que el evento sea de tipo híbrido o remoto. También se ha incorporado un campo para el ingreso del lugar, el cual en el SANE debía ser ingresado mediante la integración del SAR, algo que no está contemplado en este nuevo sistema. Finalmente, se han incorporado interruptores que indican si un evento publicado es visible para los televisores o para la web del DCC. Además, si se está editando un evento ya publicado, se encontrará un botón titulado "Ocultar evento" en el extremo derecho. Este botón permite marcar un evento publicado como "No publicado".

En el centro de la Figura 3.6 se encuentran los siguientes componentes:

- Título: Corresponde al titular o nombre del evento. Este campo se autorrellena con el ingresado en la vista previa a crear el evento de la Figura 3.7.
- Expositores: Representan a las personas que dan una charla o exponen en un evento, se muestra como un subtítulo de un titular.
- Texto: Contiene el texto o cuerpo que posee un evento, destinado a colocar información más detallada de este. Al pulsar en el campo para editarlo, se abre un modal que permitirá escribir texto enriquecido como se ve en la Figura 3.7.
- Imagen: Permite agregar una imagen buscando entre las subidas al sistema por el nombre, luego con la capacidad de previsualizarla. Al presionar el botón azul dentro de este elemento se puede agregar una imagen al evento y al sistema.

En este caso, se cambió el nombre "Bajada" al de "Expositores" y para el caso del campo Texto, se desplegará el modal ilustrado en la Figura 3.7 para poder escribir detalles de forma más cómoda. En el caso del SANE, este texto se edita en un rectángulo, que dependiendo de los elementos que se incluyen en el constructor, el espacio visible del texto se hace significativamente pequeño, lo que dificulta su legibilidad. También se eliminaron varios componentes de la sección de elementos, los cuales no eran utilizados, y solo complicaban el uso de la interfaz. Aquí solo se mantuvieron los elementos principales, y se ajustaron tanto sus posiciones

como la cantidad de elementos a uno solo por evento. Esto permite tener un formato más estandarizado para los eventos, ya que los aquellos que están publicados en SANE generalmente no difieren de las dos plantillas preestablecidas descritas en la Sección 3.2.1 en cuanto a su estructura.

Finalmente, si se rellena la información de un evento o se trata de editar uno existente mediante el boton de editar de la vista de la Sección 3.2.1, esta se verá como en la Figura 3.8.

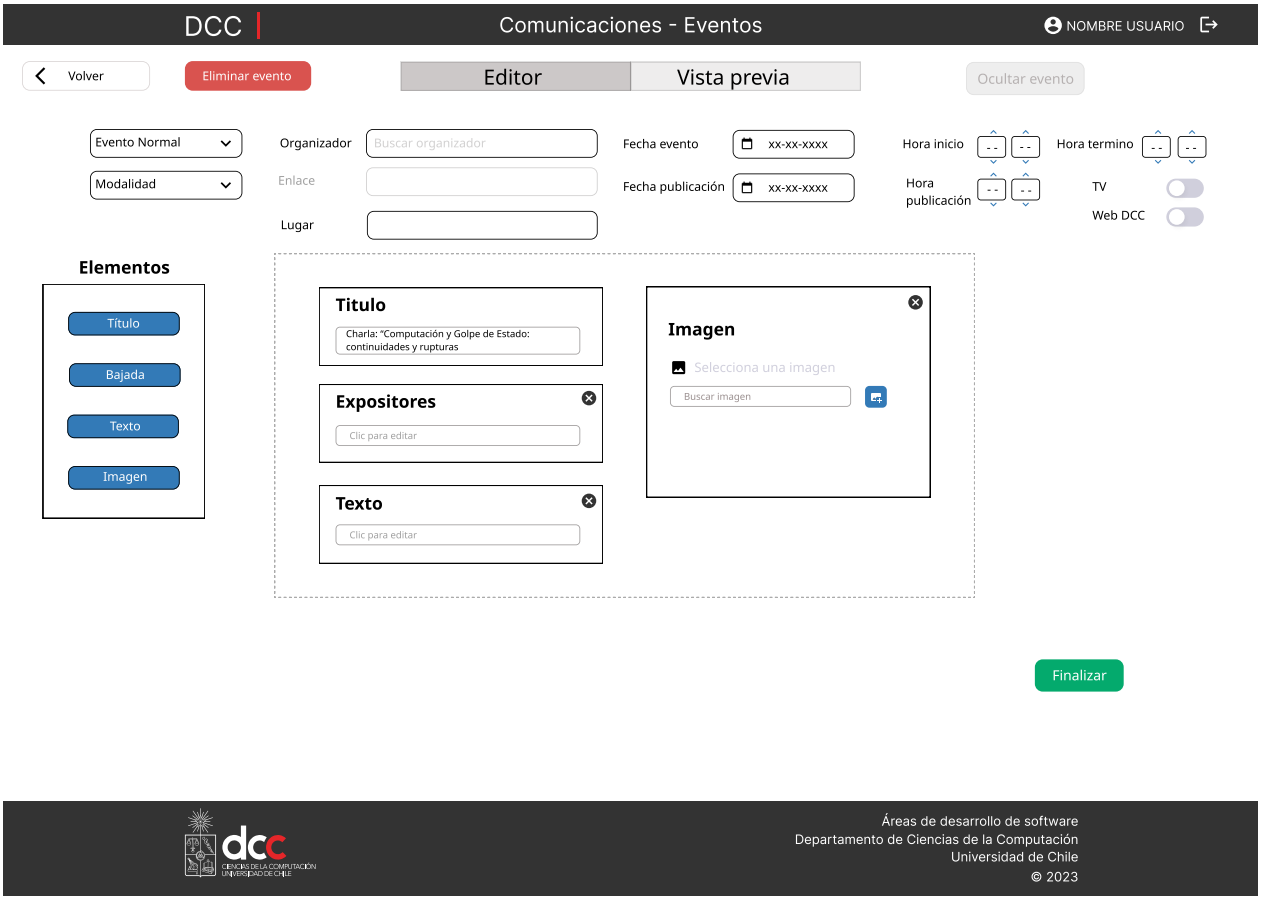


Figura 3.6: Vista de creación de eventos, nuevo evento sin datos.

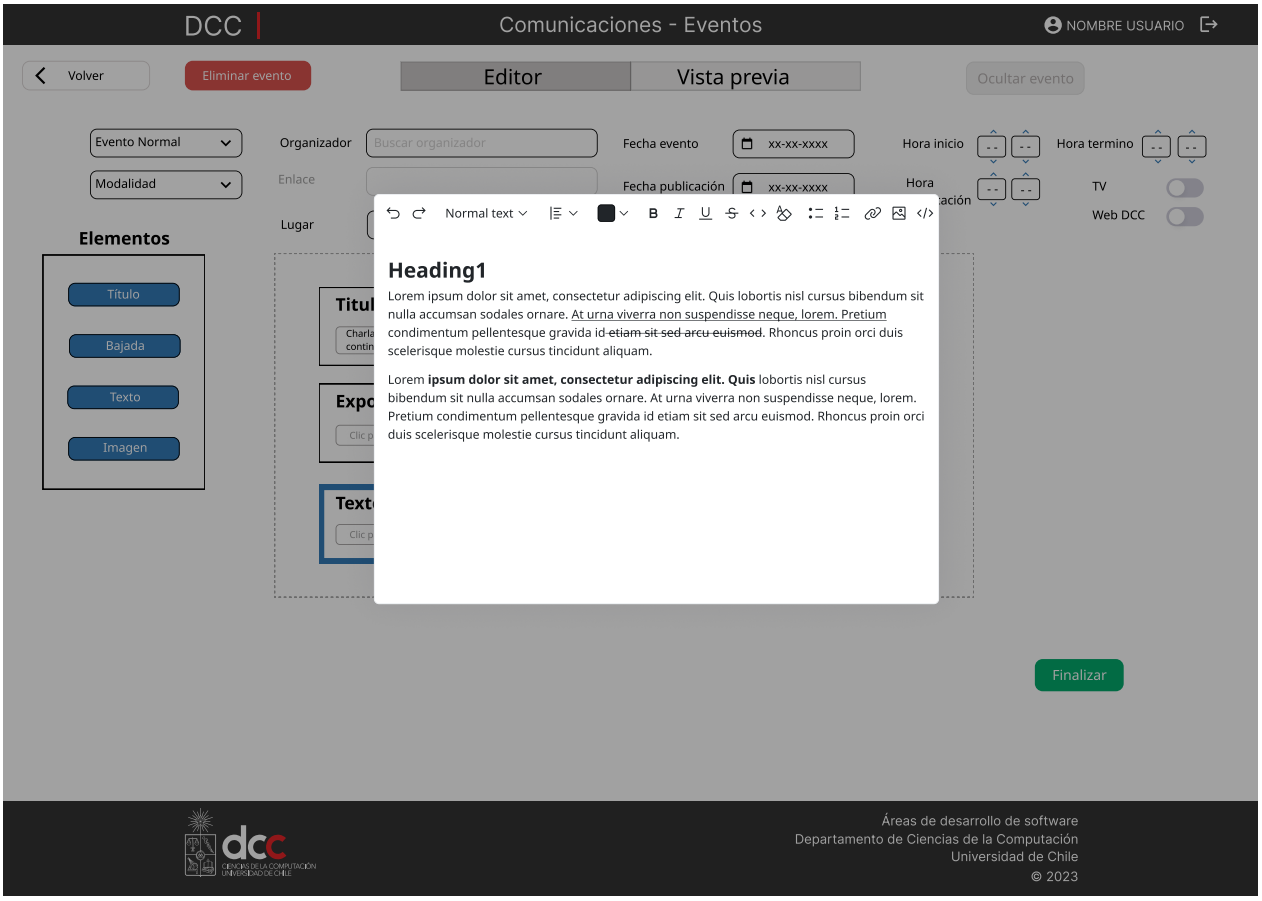


Figura 3.7: Vista de creación de eventos, edición del cuerpo de un evento con texto enriquecido.

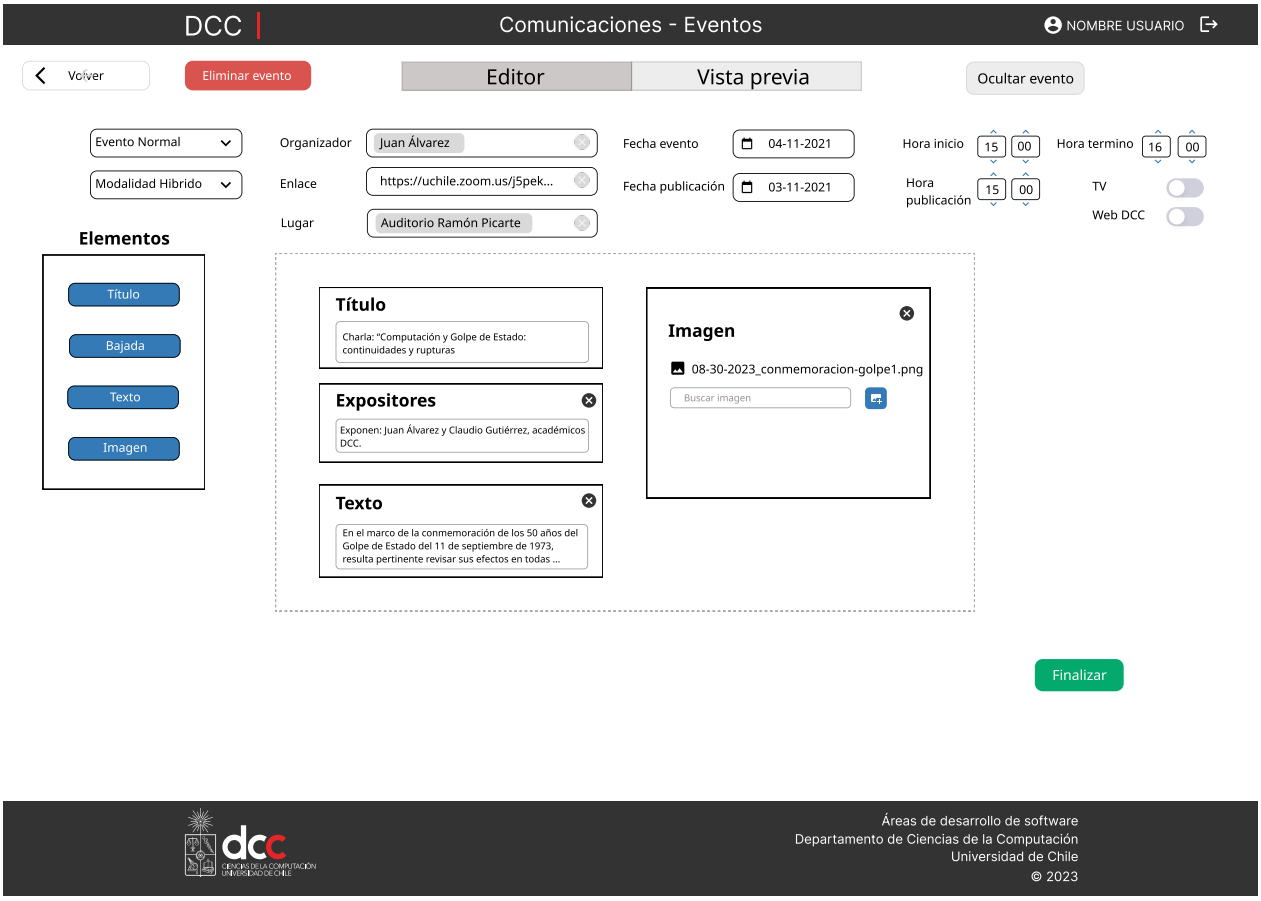


Figura 3.8: Vista de creación de eventos, con información rellenada de un evento

Vista previa

Si un usuario del área de comunicaciones desea verificar que la información ingresada en un evento se presenta correctamente, puede hacerlo accediendo a la pestaña "Vista previa" ubicada en la parte superior de la vista de construcción de eventos de la Sección 3.2.1. Esto mostrará la Figura 3.9, que se basa en un evento real registrado en el sitio web del SANE[2].

Este diseño adopta un formato muy similar al sitio del web del SANE, pero se han agregado nuevos elementos como la modalidad y el enlace, y se han reordenado los elementos de Fecha del evento, hora inicio y fin, y lugar para facilitar la comprensión de la información.

En otros sistemas, es probable que la información se presente de forma distinta, ya que la idea es entregar la información con parte del estilo que tendrá su contenido. Sin embargo, será responsabilidad exclusiva del sistema que llame a la API del nuevo sistema de eventos decidir cómo usar, estilizar y disponer la información. Esta vista previa tiene como objetivo dar una referencia de cómo se va a enviar la información.

[2]Evento registrado en el SANE: `https://comunicaciones.dcc.uchile.cl/events/306-charla-computacion-y-golpe-de-estado-continuidades-y-rupturas/`

Figura 3.9: Vista previa de un evento, basado en un evento del SANE.

3.2.2. Modelo de datos

Para el modelo de datos de este nuevo sistema, se seleccionaron elementos específicos del modelo de datos de eventos del SANE, el cual se ilustra en la Figura 3.10. Esta figura es una versión simplificada de la Figura D.1 que se encuentra en el Anexo D. Se optó por descartar elementos relacionados con las noticias, el sistema de *tags* y las entidades asociadas al sistema de usuarios. Como se mencionó en la Sección 1.4, la plantilla que se utilizará para el desarrollo incorpora un sistema de autenticación de usuarios a través del portal DCC, eliminando así la necesidad de almacenar usuarios en el modelo de datos de la aplicación. Además, se prescindirá de las entidades relacionadas con las reservas de lugares y la integración con el SAR, ya que esta implementación no funciona correctamente y no es el enfoque principal de este trabajo.

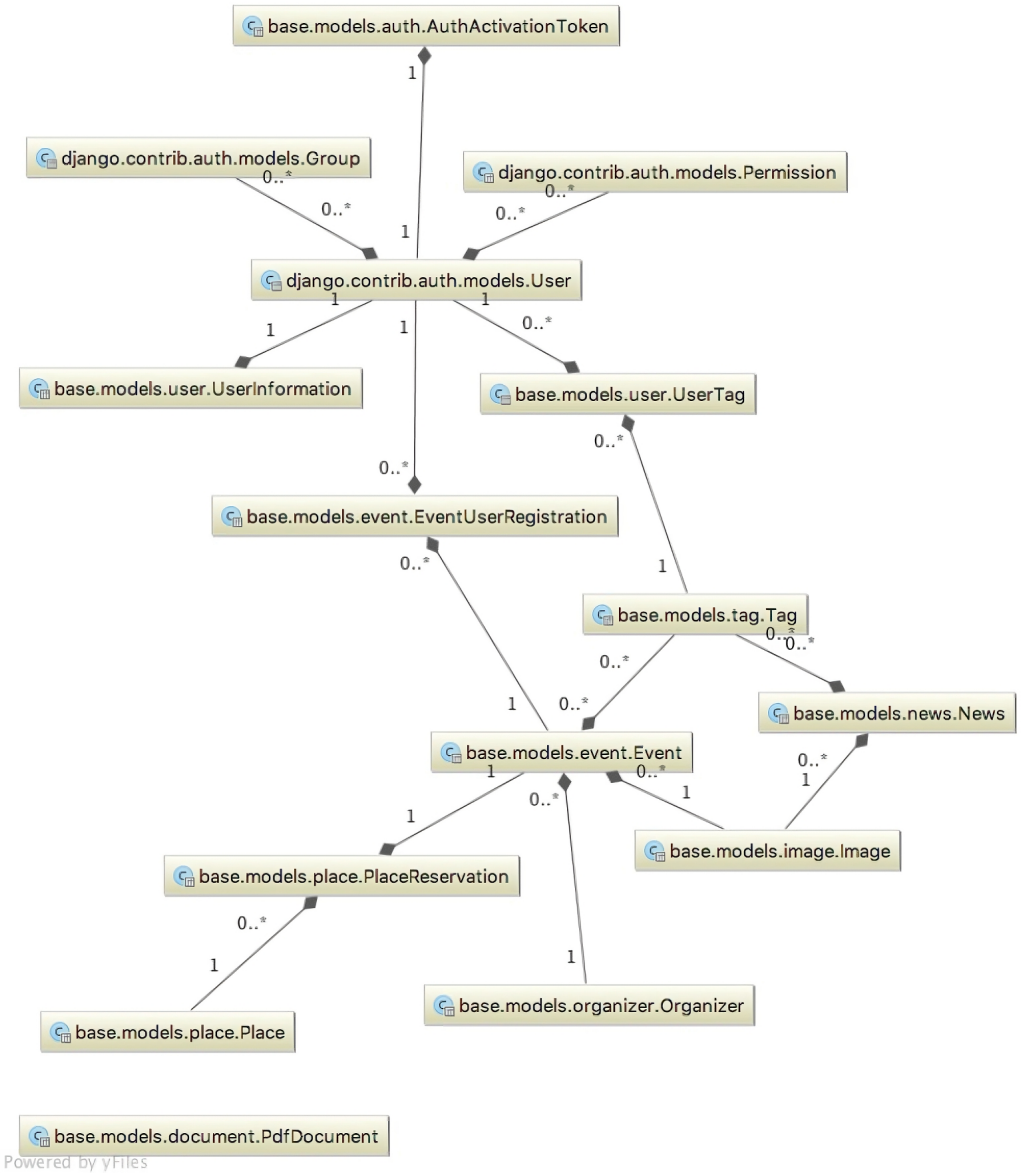


Figura 3.10: Modelo de datos del SANE simplificado.

El resultado de esto es un modelo de datos más sencillo que solamente está centrado en los eventos y sus componentes, como se puede ver en la Figura 3.11.

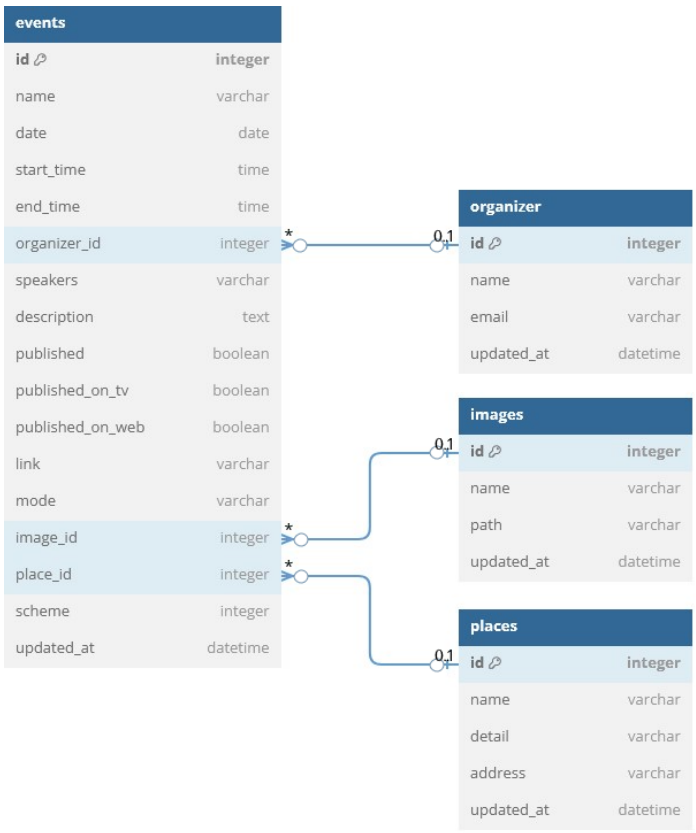


Figura 3.11: Diagrama de entidad relación del modelo de datos.

A continuación, se describen las entidades del modelo de datos:

Entidad ***events:***

Esta es la entidad principal que almacena la información de cada evento. Los atributos de esta entidad son los siguientes:

- *id*: Es el identificador único de la entidad.
- *name*: El nombre o título del evento.
- *date*: La fecha del evento.
- *start_time* y *end_time*: La hora de inicio y fin del evento.
- *organizer_id*: Un identificador que hace referencia a la clave primaria en la entidad **organizer**. Representa al organizador del evento.
- *speakers*: Los expositores del evento.
- *description*: La descripción del evento.
- *published*: Un valor booleano que indica si el evento está publicado.
- *published_on_tv*: Un valor booleano que indica si el evento está publicado en las pantallas del DCC.

- *published_on_web*: Un valor booleano que indica si el evento está publicado en la web del DCC.
- *link*: Un enlace al evento.
- *mode*: La modalidad del evento (por ejemplo, "presencial", "remoto" o "híbrido").
- *image_id*: Un atributo que identifica al atributo de entidad **images**.
- *place_id*: Identifica al atributo de entidad **places**.
- *scheme*: Representa la opción de cómo están distribuidos los elementos, en concreto si las imágenes están al lado o abajo.
- *updated_at*: La fecha y hora de la última actualización del registro del evento.

Entidad ***images:***

Esta entidad almacena la información de las imágenes que pueden ser utilizadas en los eventos. Los atributos de esta entidad son los siguientes:

- *id*: Es el identificador único de la entidad.
- *name*: El nombre de la imagen.
- *path*: La ruta al archivo de la imagen.
- *updated_at*: La fecha y hora de la última actualización del registro de la imagen.

Entidad ***organizer:***

Esta entidad almacena la información de los organizadores que pueden ser utilizados en los eventos. Los atributos de esta entidad son los siguientes:

- *id*: Es el identificador único de la entidad.
- *name*: El nombre del organizador.
- *email*: El correo electrónico del organizador.
- *updated_at*: La fecha y hora de la última actualización del registro del organizador.

Entidad ***places:***

Esta entidad almacena la información de los lugares que pueden ser asociados a un evento. Los atributos de esta entidad son los siguientes:

- *id*: Es el identificador único de la entidad.
- *name*: El nombre del lugar.

- *detail*: Detalles adicionales sobre el lugar.
- *address*: La dirección del lugar.
- *updated_ at*: La fecha y hora de la última actualización del registro del lugar.

En lo que respecta a las relaciones entre las entidades, cada evento puede estar asociado a una imagen, un lugar y un organizador, o no estar asociado a ninguno de estos. Esto se representa como una relación de uno a muchos o de cero a muchos desde las entidades ***images***, ***places*** y ***organizer*** hacia la entidad ***events***. En un modelo relacional, estas relaciones se implementarían mediante claves foráneas en la tabla events que harían referencia a las claves primarias de las otras tablas.

3.2.3. Diseño de la API

En esta sección, se presenta el diseño propuesto para la API REST del nuevo sistema de eventos, tomando como referencia parte de los endpoints actuales del SANE enumerados en el Anexo C. Estos endpoints listados del SANE tienen el propósito de gestionar la creación, actualización y eliminación de eventos. Además, proporcionan funcionalidades como manejar las reservas de lugares y obtener detalles de registros de usuarios.

Los endpoints utilizados públicamente para obtener los eventos presentes en el SANE se mantendrán en el nuevo sistema. Ahora, obtener el listado de eventos no solo proporcionarán los eventos publicados, sino que también se podrán entregar parámetros para filtrar la respuesta según sea necesario.

En este contexto, a diferencia del sistema SANE, el propósito de entregar la información de los eventos desde el nuevo sistema es proporcionarla de manera más clara y precisa. Esto se logrará mediante la eliminación de los JSONs anidados que SANE utiliza para entregar instrucciones de configuración, como la cantidad de columnas en un evento o los ítems, entre otros. Esta característica, útil para ordenar la información en la web de SANE, ya no será necesaria, ya que los eventos se presentarán de forma más estandarizada. Solo se entregará en un objeto JSON los campos de organizadores, imágenes y lugares. Como se discutió en la Sección 3.2.1, la responsabilidad de estilizar y ordenar la información recaerá en quien consuma esta información. Al simplificar los elementos del JSON de respuesta y eliminar estos JSONs anidados que estructuraban el evento, se logra una presentación de la información más clara y precisa.

Los demás endpoints del SANE no serán considerados para el nuevo sistema, como se explica en la Sección 3.2.2. No serán necesarios los elementos relacionados con reservas de lugares, registro de usuarios y el carrusel en la interfaz.

Además de conservar los endpoints que posibilitan la creación, actualización y eliminación de eventos, se añadirá un nuevo endpoint de método GET. Este permitirá la descarga de un archivo CSV con los datos de los eventos filtrados, resultando útil para informar a otras áreas de la facultad sobre los eventos que tienen lugar en el DCC, utilizando un formato más universal. En relación con estos nuevos endpoints, se crearán otros que permitan la

creación, consulta, edición o eliminación de elementos vinculados a lugares, organizadores e imágenes. Esto no solo facilitará la administración interna de los elementos del sistema, sino que también será una buena práctica para escalar el sistema en el futuro.

En cuanto a las direcciones de los endpoints, todas comenzarán con `/api/v1/`, indicando que la dirección responde como un llamado a la API del sistema y representa una versión específica de la misma, una práctica que permite mantener varias versiones de una API en caso de desarrollos futuros [2]. Además, se requerirá un método de autorización mediante un token en el encabezado de la solicitud HTTP. Para obtener dicho token, se accederá mediante otro endpoint que validará las credenciales del usuario de la API y verificará sus permisos pertinentes.

A continuación, se describen los endpoints propuestos para su desarrollo en el próximo semestre, abarcando aquellos referentes a Eventos en la Sección 3.2.3, Imágenes en la Sección 3.2.3, Organizadores en la Sección 3.2.3 y Lugares en la Sección 3.2.3. Finalmente, se presentan algunos ejemplos de uso con las respuestas esperadas en la Sección 3.2.3.

Eventos

A continuación se detallan los endpoints referentes a los eventos:

1. GET /events: Obtiene una lista de todos los eventos. En el caso de añadir parámetros a la consulta se filtran según los parámetros agregados.
   - **Parámetros de consulta:** date, place, mode, organizer, start_date, end_date.
   - **Respuesta:** Un arreglo de objetos JSON de los eventos filtrados según los parámetros de la consulta.
2. GET /events/{id}: Obtiene los detalles de un evento específico.
   - **Parámetros de ruta:** id (requerido)
   - **Respuesta:** Un objeto JSON de un evento.
3. GET /events/download: Genera un archivo CSV con los eventos. En el caso de añadir parámetros a la consulta se filtran según los parámetros agregados.
   - **Parámetros de ruta:** date, place, mode, organizer, start_date, end_date.
   - **Respuesta:** Archivo CSV con los eventos filtrados según los parámetros de la consulta.
4. POST /events: Crea un nuevo evento.
   - **Cuerpo de la solicitud (requerido):** Un objeto JSON del lugar con los siguientes campos:

Campo	Descripción	Tipo de dato	Requerido
name	Nombre visible del evento	string	Sí
date	Fecha de realización del evento	string	No
start_time	Hora de inicio del evento	string	No
end_time	Hora de fin del evento	string	No
speakers	Texto que indica quienes dan la charla en el evento	string	No
text	Detalles de un evento. Se permite el texto enriquecido	string	No
published	Parámetro que indica si un evento está publicado. Por defecto, tiene el valor false	bool	No
published_on_tv	Parámetro que indica si un evento publicado está disponible en las TVs del DCC. Por defecto, tiene el valor false	bool	No
published_on_web	Parámetro que indica si un evento publicado está disponible en la web del DCC. Por defecto, tiene el valor false	bool	No
link	Url que permite asistir al evento si es remoto o hibrido	string	No
mode	Modalidad de un evento. Las opciones posibles son "hybrid", "remote" e "in-person". Por defecto, la opción "in-person" está establecida	string	No
organizer_id	Id del organizador asociado al evento	int	No
image_id	Id de la imagen asociada al evento	int	No
place_id	Id del lugar en el que se organiza el evento	int	No
scheme	Opción del tipo de imagen. Las posibles opciones son 0 (sin imagen), 1 (imagen pequeña), 2 (imagen grande). Por defecto, el valor es 0	int	No

- **Respuesta:** El objeto JSON del evento creado.

5. PUT /events/{id}: Actualiza un evento existente.
   - **Parámetros de ruta:** id (requerido).
   - **Cuerpo de la solicitud (requerido):** Un objeto JSON de un evento con los campos a actualizar. Estos campos pueden ser los mismos del cuerpo de solicitud del método POST.
   - **Respuesta:** El objeto JSON del evento actualizado.
6. DELETE /events/{id}: Elimina un evento.
   - **Parámetros de ruta:** id (requerido).
   - **Respuesta:** Un mensaje de confirmación de la eliminación.

Imágenes

A continuación se detallan los endpoints referentes a los Imágenes:

1. GET /images: Obtiene una lista de todas las imágenes.
   - **Respuesta:** Un array de objetos JSON de las imagenes.
2. GET /images/{id}: Entrega una imagen específica según su id.
   - **Parámetros de ruta:** id (requerido).
   - **Respuesta:** Un objeto JSON de una imagen.
3. POST /images: Carga una nueva imagen.

- **Cuerpo de la solicitud (requerido):** Un objeto JSON de la imagen con los siguientes campos:

Campo	Descripción	Tipo de dato	Requerido
name	Nombre de la imagen	string	Sí
file	Archivo de la imagen	file	Sí

- **Respuesta:** El objeto JSON de la imagen creado.

4. PUT /images/{id}:Actualiza una imagen existente.
    - **Parámetros de ruta:** id (requerido).
    - **Cuerpo de la solicitud (requerido):** Un objeto JSON de una imagen con los campos a actualizar. Estos campos pueden ser los mismos del cuerpo de solicitud del método POST.
    - **Respuesta:** El objeto JSON de la imagen actualizado.
5. DELETE /images/{id}: Elimina una imagen.
    - **Parámetros de ruta:** id (requerido).
    - **Respuesta:** Un mensaje de confirmación de la eliminación.

Organizadores

A continuación se detallan los endpoints referentes a los organizadores:

1. GET /organizers Obtiene una lista de todos los organizadores.
    - **Respuesta:** Un array de objetos JSON de los organizadores.
2. GET /organizers/{id}: Obtiene los detalles de un organizador específico.
    - **Parámetros de ruta:** id (requerido).
    - **Respuesta:** Un objeto JSON de un organizador.
3. POST /organizers: Crea un nuevo organizador.
    - **Cuerpo de la solicitud (requerido):** Un objeto JSON del organizador con los siguientes campos:

Campo	Descripción	Tipo de dato	Requerido
name	Nombre del organizador	string	Sí
email	Correo electrónico del organizador	string	No

    - **Respuesta:** El objeto JSON del organizador creado.
4. PUT /organizers/{id}: Actualiza un organizador existente.
    - **Parámetros de ruta:** id (requerido).

- **Cuerpo de la solicitud (requerido):** Un objeto JSON de un organizador con los campos a actualizar. Estos campos pueden ser los mismos del cuerpo de solicitud del método POST.
- **Respuesta:** El objeto JSON del organizador actualizado.

5. DELETE /organizers/{id}: Elimina un organizador.
   - **Parámetros de ruta:** id (requerido).
   - **Respuesta:** Un mensaje de confirmación de la eliminación.

Lugares

A continuación se detallan los endpoints referentes a los lugares:

1. GET /places: Obtiene una lista de todos los lugares.
   - **Respuesta:** Un array de objetos JSON de lugares.
2. GET /places/{id}: Obtiene los detalles de un lugar específico.
   - **Parámetros de ruta:** id (requerido).
   - **Respuesta:** Un objeto lugar.
3. POST /places: Crea un nuevo lugar.
   - **Cuerpo de la solicitud (requerido):** Un objeto JSON del lugar con los siguientes campos:

Campo	Descripción	Tipo de dato	Requerido
name	Nombre del lugar	string	Sí
detail	Detalle del lugar	string	Sí
address	Dirección del lugar	string	Sí

   - **Respuesta:** El objeto JSON del lugar creado.
4. PUT /places/{id}: Actualiza un lugar existente.
   - **Parámetros de ruta:** id (requerido).
   - **Cuerpo de la solicitud (requerido):** Un objeto JSON del lugar con los campos que se desean actualizar. Estos campos pueden ser los mismos que los del cuerpo de la solicitud del método POST.
   - **Respuesta:** El objeto JSON del lugar actualizado.
5. DELETE /places/{id}: Elimina un lugar.
   - **Parámetros de ruta:** id (requerido).
   - **Respuesta:** Un mensaje de confirmación de la eliminación.

Ejemplos de uso

A continuación, se presentan algunos ejemplos de uso de los endpoints, junto con las respuestas esperadas.

1. **Obtener eventos según aspectos específicos:**

   El siguiente endpoint devuelve una lista de eventos que tienen fecha posterior al 1 de mayo de 2023, se realizan en el lugar con id igual a 74, son organizados por el organizador de id igual a 2 y tienen modalidad híbrida.

```
GET /events?start_date=2023-05-01&place=1&mode=remote&organizer=2
```

   Ejemplo de respuesta:

```
[
    {
    "id": 23,
    "name": "¿Puede ChatGPT conquistar el mundo? Lenguaje humano vs lenguaje de máquina",
    "date": "2023-05-03",
    "start_time": "16:00",
    "end_time": "17:00",
    "speakers": "Presentador: Jorge Ortiz DCC Uchile, IMFD, Awto.",
    "text": "<p><strong>Jorge Ortiz. Licenciado en Letras Hispánicas de la Pontificia Universidad Católica, Magíster (c) en Ciencias de la Computación de la Universidad de Chile y Estudiante IMFD</strong>. Sus áreas de investigación y desarrollo abarcan el procesamiento de Lenguaje Natural, la Ciencia de Datos y la Lingüística.</p>\n",
    "published": true,
    "published_on_tv": false,
    "published_on_web": true,
    "link": "https://uchile.zoom.us/j/90921?pwd=blZPdVI5a0h",
    "mode": "hybrid",
    "organizer_id": {
                      "id": 2,
                      "name": "Juan Rodriguez",
                      "email": "null"
                  },
    "place":{
              "id": 74,
              "name": "Sala Flajolet",
              "detail": "Edificio Poniente , 3er piso",
              "address": "Beauchef 851"
              },
    "image": {
              "id": 479,
              "file": "/media/images/AJ344WEfdk.png",
              "name": "CharlaGTP.png"
              },
    "scheme": 1
    },

```

```
    {
    "id": 23,
    "name": "Encuentro de Bienvenida Comunidad DCC",
    "date": "2023-05-08",
    "start_time": "16:00",
    "end_time": "17:00",
    "speakers": null,
    "text": "<p>El objetivo de esta actividad es reencontrarnos como
        comunidad en este inicio de clases.</p>\n<p>Para participar
       desde el Auditorio Picarte del DCC, les informamos que
       contamos con aforo limitado. De lo contrario, pueden
       conectarse vía Zoom en https://bit.ly/BienvenidaDCC2022.</p>\
       n",
    "published": true,
    "published_on_tv": false,
    "published_on_web": false,
    "link": "https://bit.ly/BienvenidaDCC2022",
    "mode": "hybrid",
    "organizer":{
                  "id": 2,
                  "name": "Juan Rodriguez",
                  "email": "null"
              },
    "place": {
                "id": 74,
                "name": "Sala Flajolet",
                "detail": "Edificio Poniente , 3er piso",
                "address": "Beauchef 851"
              },
    "image": null,
    "scheme": 0
    }
    ...
]
```

2. **Crear un nuevo organizador:**

   El siguiente endpoint crea un organizador con nombre “Juan Rodriguez” sin un correo asociado.

```
POST /organizers
```

   Cuerpo de la solicitud:

```
{
        "name": "Juan Rodriguez"
}
```

Ejemplo de respuesta:

```
{
        "id": 74,
        "name": "Juan Rodriguez",
        "email": "null"
}
```

3. **Actualizar la información de un lugar:**

El siguiente endpoint actualiza el campo del detalle de un lugar con un id igual a 23.

```
PUT /places/23
```

Cuerpo de la solicitud:

```
{
        "detail": "Edificio Escuela, Hall Sur"
}
```

Ejemplo de respuesta:

```
{
        "id": 74,
        "name": "Sala S05",
        "detail": "Edificio Escuela, Hall Sur, 1er piso",
        "address": "Beauchef 850"
}
```

4. **Eliminar una imagen:**

El siguiente endpoint elimina una imagen del sistema con un id igual a 3.

```
  DELETE /images/3
```

Ejemplo de respuesta:

```
{
        message: "Image successfully deleted"
}
```

Capítulo 4

Implementación

La implementación de la solución se llevó a cabo siguiendo el diseño detallado en la Sección 3.2 del Capítulo 3. Este proceso se estructuró en dos componentes principales: el desarrollo del front-end y el back-end de la aplicación.

El front-end se centró en la materialización de las interfaces propuestas en el diseño inicial, asegurando una experiencia de usuario intuitiva y eficiente. Paralelamente, el desarrollo del back-end abarcó la implementación del modelo de datos y la creación de la API REST, ambos elementos definidos previamente en la Sección 3.2.

4.1. Arquitectura del Sistema

La implementación se basó en la estructura del template proporcionado por el área de desarrollo del DCC, también conocido como *Django Boilerplate*. Este template es un proyecto de Django con una estructura de archivos y configuración predefinida, entregado a los memoristas o estudiantes que trabajan en desarrollos de sistemas dentro del DCC. Además de estar desarrollado con Django, el template incluye algunas librerías de JavaScript como jQuery y Bootstrap 5. La arquitectura del sistema se ilustra en la Figura 4.1.

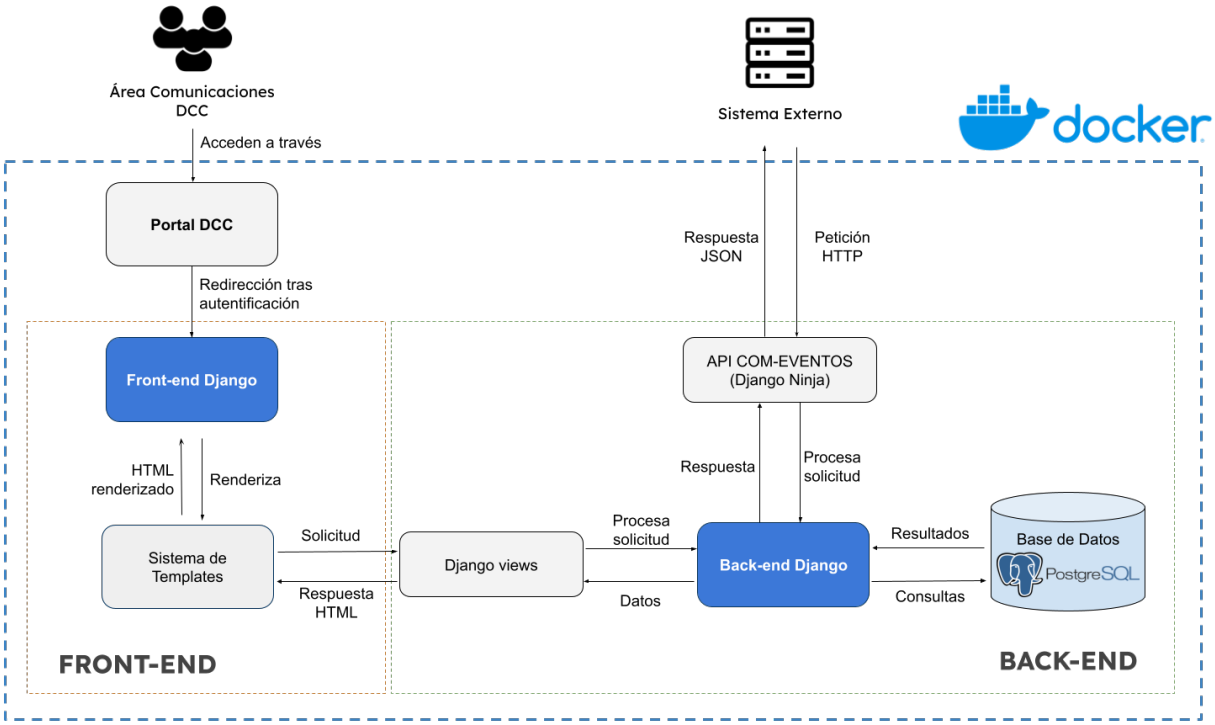


Figura 4.1: Diagrama de arquitectura del sistema.

4.2. Front-end

Para el desarrollo del front-end de la aplicación, se utilizó la estructura de un proyecto del *Django Boilerplate* proporcionada por el área de desarrollo del DCC. Esta estructura se actualizó a la versión más reciente de Django (5.0.6).

La implementación de las vistas se realizó mediante el sistema de templates de Django, que consiste en archivos HTML con funcionalidades integradas de Python, como bucles *for* o condicionales *if* dentro del código HTML. Además, se incorporaron pequeñas funcionalidades utilizando jQuery 3.7.1, una librería de JavaScript que simplifica la escritura de código en este lenguaje. Para estilizar parte de la aplicación y utilizar ciertos componentes, se empleó el framework CSS Bootstrap en su versión 5.3.3.

En cuanto a la estructura de la aplicación, se implementaron cinco vistas principales: cuatro destinadas a mostrar información sobre eventos, organizadores, lugares y documentos, y una última vista dedicada al constructor de eventos.

4.2.1. Listados

Para facilitar la visualización de los elementos ingresados al sistema, se crearon vistas específicas para cada tipo de listado. Estas vistas respetaron los diseños propuestos en la Sección 3.2.1 del Capítulo 3 y se dividieron en los siguientes listados, exceptuando el listado de Imágenes, el cual se renombró durante el desarrollo a "Documentos" por las razones explicadas más adelante:

- Listado de Eventos
- Listado de Lugares
- Listado de Organizadores
- Listado de Documentos

En la Figura 4.1 se puede observar el listado de eventos. Esta interfaz permite a los usuarios administrar los elementos del listado, ofreciendo opciones para editar, eliminar o crear nuevos elementos. El funcionamiento e interfaz es similar para los listados de organizadores y lugares.

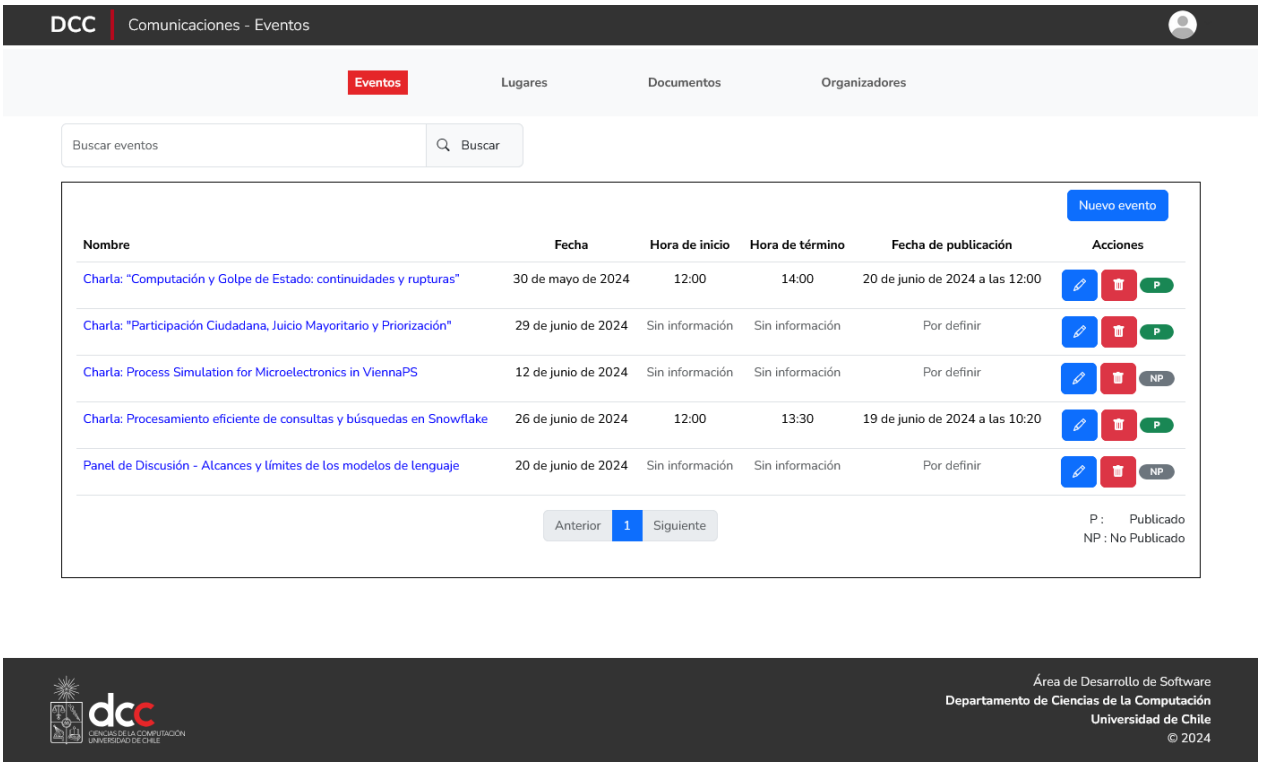


Figura 4.1: Vista del listado de eventos.

Si un usuario hace clic en el botón "Nuevo Evento", se abrirá un modal que permite completar los datos mínimos para crear un evento, como se muestra en la Figura 4.2.

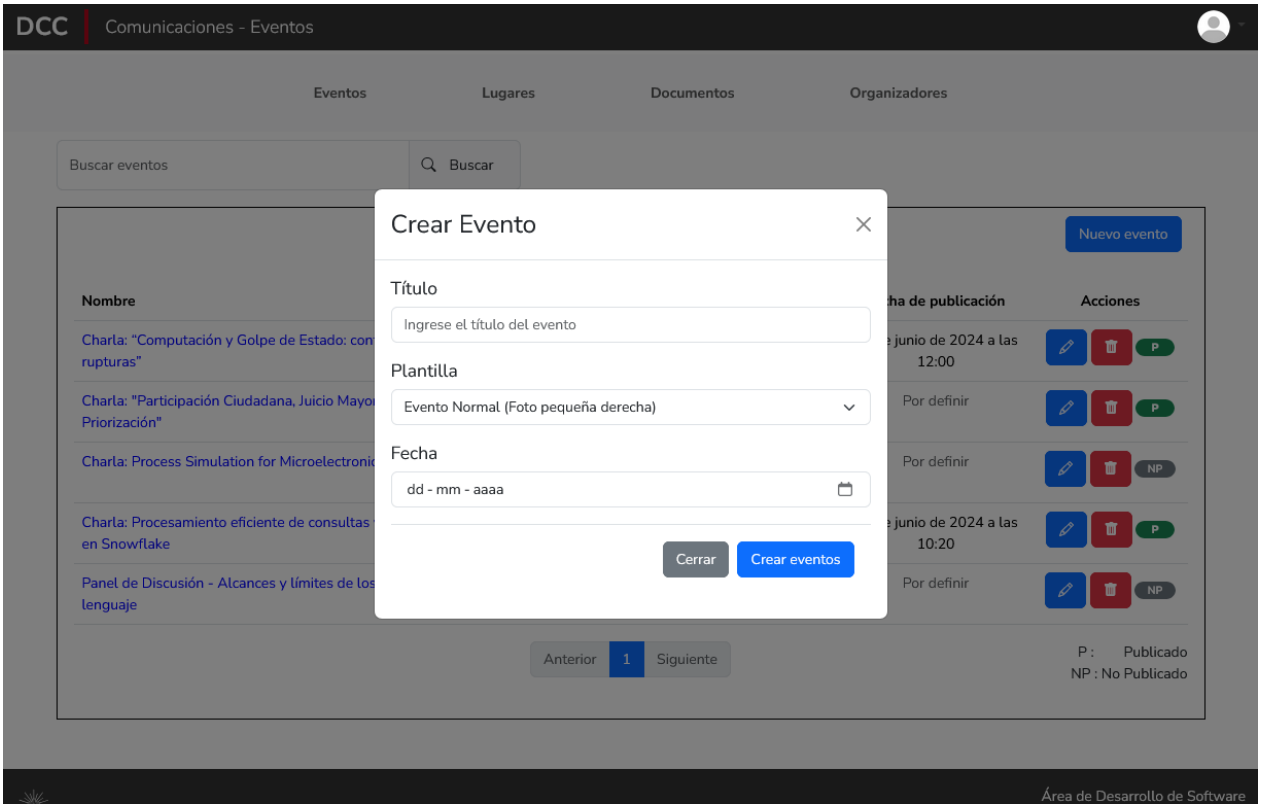


Figura 4.2: Modal de creación de un nuevo evento.

El listado de documentos presenta una diferencia significativa, debido a que originalmente mostraría solo un listado de las imágenes subidas al sistema. Tras una reunión con las usuarias finales para mostrar los avances en la implementación de las interfaces, se añadió el requerimiento de poder subir tanto imágenes como archivos PDF. Esta funcionalidad se incorporó para cubrir los casos en que un evento incluya un programa o archivo de interés en formato PDF, pero por limitaciones de tiempo y conveniencia, se integró dentro de la misma sección tanto imágenes como documentos.

Como se puede apreciar en la Figura 4.3, el listado de documentos cuenta con dos botones: uno para añadir documentos y otro para añadir imágenes. La Figura 4.4 muestra el modal que se abre al presionar el botón para añadir documentos. El proceso para añadir imágenes funciona de manera similar.

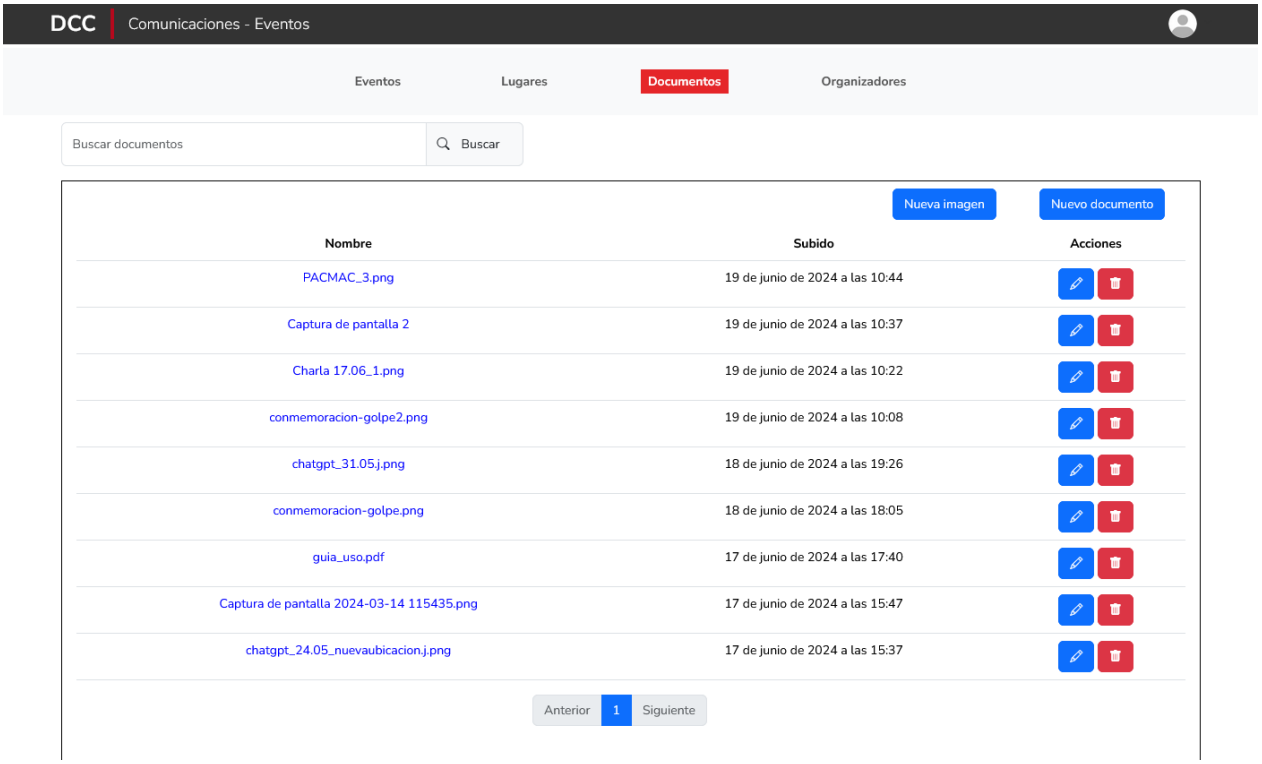


Figura 4.3: Vista del listado de documentos.

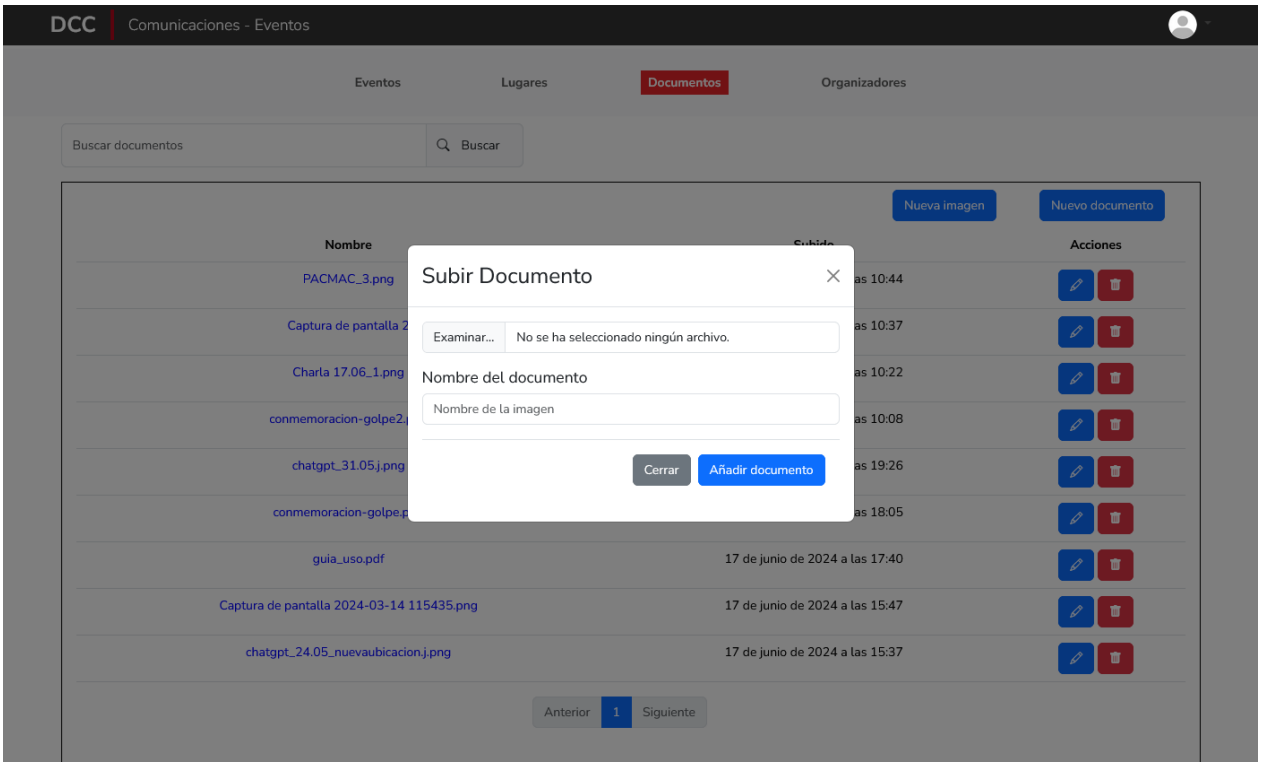


Figura 4.4: Modal desplegado para agregar un nuevo documento.

4.2.2. Constructor de eventos

Desde el listado de eventos, el usuario puede acceder a esta vista al crear un nuevo evento completando los datos del modal, como se muestra en la Figura 4.2, o simplemente utilizando el botón de edición junto a un evento existente del listado. Esta vista permite revisar y editar diferentes aspectos de la configuración del evento.

La interfaz del constructor de eventos se divide en tres secciones principales: configuración, acciones y edición de contenido, como se puede observar en la Figura 4.5.

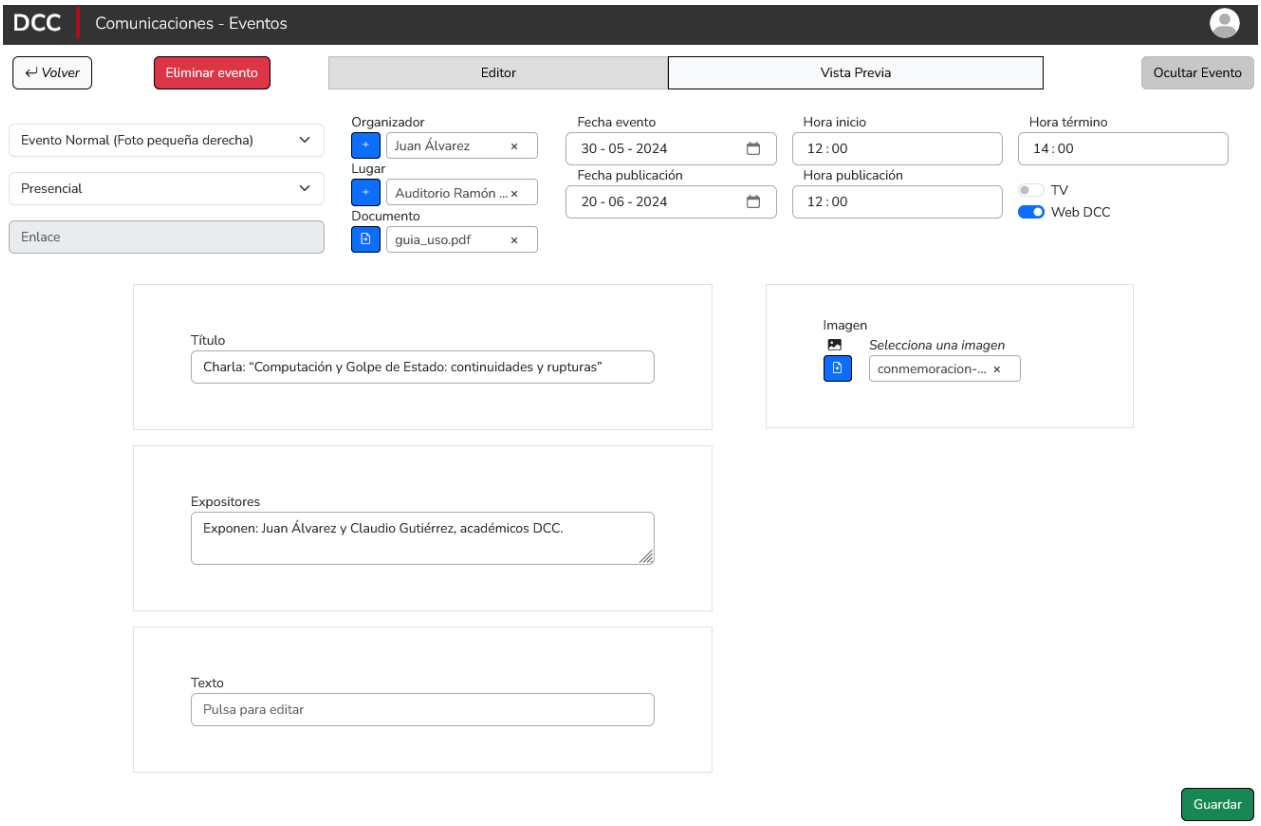


Figura 4.5: Vista del constructor de eventos.

Botones de acciones

En la parte superior de la vista se encuentran la mayoría de los botones que permiten realizar distintas acciones:

- *Botón "Volver"*: Regresa a la vista de la lista de eventos (Figura 4.1).
- *Botón "Eliminar evento"*: Muestra un modal de confirmación de eliminación. Si se confirma, redirige al usuario a la lista de eventos.
- *Botones de "Vista previa" y "Editor"*: Permiten alternar entre la edición del evento y su visualización previa. Los cambios en el editor requieren confirmación para guardar.

- *Botón "Publicar evento"/ "Ocultar evento"*: Cambia el estado de publicación del evento. Los eventos se crean inicialmente como no publicados.
- *Botón "Guardar"*: Ubicado en la parte inferior de la vista, de color verde. Al presionarlo, se guarda el evento y se redirige a la vista previa con la información actualizada.

Configuración de evento

En la parte superior de la vista se encuentra la sección que permite modificar los campos que permiten ajustar las características del evento:

1. *Modalidad*: Indica si un evento es remoto, híbrido o presencial.
2. *Plantilla utilizada*: Configura la disposición visual del evento (imagen asociada a un lado, abajo, o solo una imagen), se define al crear un evento, pero se puede modificar dentro.
3. *Enlace*: URL referente al evento (solo disponible para eventos híbridos o remotos).
4. *Organizador*: Un menú desplegable que te permite seleccionar un organizador previamente cargado en el sistema o agregar uno nuevo utilizando el botón azul adjunto.
5. *Lugar*: Un menú desplegable que te permite seleccionar un lugar previamente cargado en el sistema o agregar uno nuevo utilizando el botón azul adjunto.
6. *Documento*: Un menú desplegable que te permite seleccionar un documento previamente cargado en el sistema o agregar uno nuevo utilizando el botón azul adjunto.
7. *Fecha del evento*: Fecha de realización del evento.
8. *Fecha de publicación*: Fecha en que se publicará el evento.
9. *Publicado en TV*: Cambia el estado de publicación del evento en las TVs del DCC.
10. *Publicado en Web*: Cambia el estado de publicación del evento en la web del DCC.

En el caso de las plantillas, esta permite seleccionar entre los siguientes tres esquemas disponibles:

- ***Evento Normal***: La imagen se coloca a la derecha del contenido (Figura 4.6).
- ***Evento Afiche***: La imagen se coloca en la parte inferior del contenido y es de mayor tamaño (Figura 4.7).
- ***Sin plantilla***: Solo se muestra el título y la imagen debajo (Figura 4.8).

Figura 4.6: Esquema "Evento Normal".

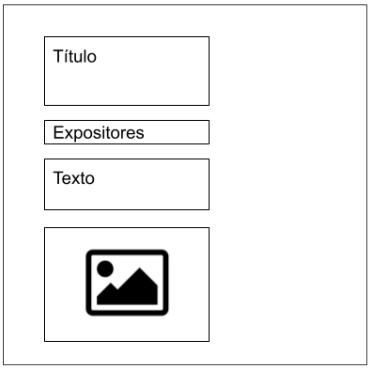


Figura 4.7: Esquema "Evento Afiche".

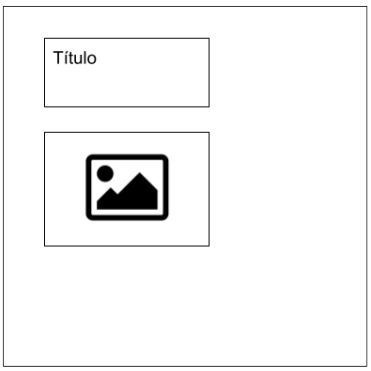


Figura 4.8: Esquema "Sin plantilla".

Información de un evento

Por otro lado, se tienen los elementos que son editables dentro del contenido de un evento en la parte central del constructor, aquí se componen de los siguientes elementos:

- *Título*: El nombre o titular con el que se identifica UN evento. Este campo es obligatorio
- *Expositores*: Corresponde al texto justo debajo del título. La idea es presentar los expositores de un evento
- *Texto*: Es el contenido principal del texto. Este presenta un modal con un textarea que permite editar con texto enriquecido el contenido del evento escrito.
- *Imagen*: Una imagen asociada al evento.

Finalmente, luego de tener un evento con todos sus datos rellenados al guardar los datos al editar, se lograría apreciar como se quiere visualizar en la vista previa como se ve en la Figura 4.9. Cabe destacar que respetar la plantilla y la disposición de la información depende luego del servicio o sistema que extraiga esta información desde la API.

Figura 4.9: Vista Previa al terminar de editar un evento.

4.3. Back-end

El desarrollo del backend de la aplicación se estructuró en dos componentes principales. En primer lugar, se implementó el modelo de datos, detallado en la Sección 4.3.1, donde se describen las modificaciones realizadas con respecto al diseño original. En segundo lugar, se desarrolló la API REST, cuya implementación y endpoints se explican en profundidad en la Sección 4.3.2.

4.3.1. Modelo de datos

El modelo de datos del sistema, como se mencionó en la Sección 3.2, mantuvo en gran medida el diseño original, pero se incorporaron algunos elementos adicionales para satisfacer requerimientos que surgieron durante la implementación. El cambio más significativo fue la adición de una nueva tabla al modelo destinada a documentos, necesaria para relacionar archivos PDF en ciertos casos.

Esta nueva tabla comparte atributos similares a los de la tabla de imágenes, pero está específicamente diseñada para documentos. Además, en ambas tablas se agregó un atributo extra *doc_type*, que se completa por defecto para identificar el tipo de documento. Este cambio fue necesario para implementar el listado mixto que se muestra en la Figura 4.3, que incluye tanto imágenes como PDFs, facilitando la diferenciación de cada elemento al editar sus nombres o eliminarlos.

La Figura 4.1 muestra un diagrama entidad-relación generado mediante la librería *Django-extensions*. Los elementos en color grisáceo son aquellos marcados como no obligatorios, re-

flejando un requerimiento que permite crear eventos con datos mínimos cuando no se dispone de información completa. Los elementos de color más oscuro son aquellos considerados necesarios para crear el elemento o que se rellenan automáticamente (como los campos booleanos o el mencionado doc_type).

En cuanto a la cardinalidad de las relaciones, como se mencionó también en la Sección 3.2.2, los organizadores, lugares, imágenes (Ahora Documentos) pueden estar asociados a ninguno o varios eventos, mientras que cada evento puede asociarse a ninguno o un elemento de cada tabla. Es importante señalar que la cardinalidad mostrada en la Figura 4.1 fue agregada manualmente, ya que la librería *Django-extensions* no considera las relaciones entre tablas en términos de cardinalidad.

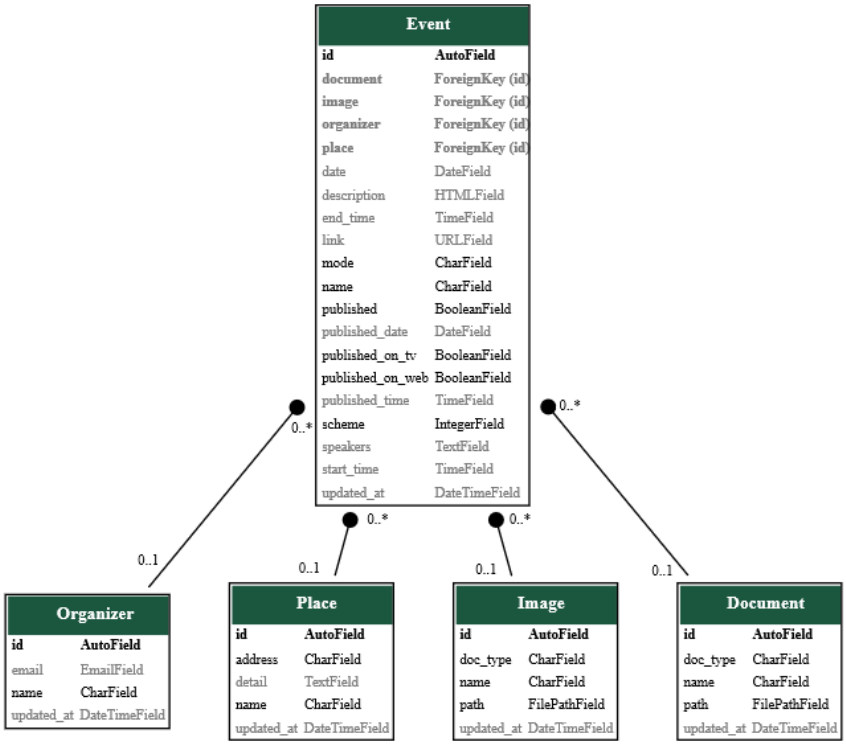


Figura 4.1: Diagrama entidad-relación generado por *Django-extensions*

4.3.2. API REST

Al igual que en el diseño original de la API, se crearon endpoints para extraer, modificar, eliminar y crear datos para cada elemento del modelo de datos. De manera similar al caso de las imágenes, se implementaron endpoints para los documentos que fueron considerados durante la implementación.

Un cambio significativo en esta implementación fue la exploración de una nueva librería llamada *Django Ninja*[1] para generar estos endpoints, sugerida por el área de desarrollo. Dado que el modelo de datos no es muy complejo, se evaluó que usar Django Rest Framework podría

[1]Web de Django Ninja: `https://django-ninja.dev/`

no ser la mejor opción, ya que la falta de experiencia con esta librería podría demandar más tiempo de aprendizaje y requiere una considerable inversión de tiempo en código. En cambio, Django Ninja resultó ser una librería más amigable y directa para implementar los endpoints, siguiendo la filosofía de otra librería popular de Python para crear APIs llamada *FastAPI*. Django Ninja genera automáticamente una documentación interactiva utilizando menos código, entregando el mismo resultado esperado.

A continuación, se presenta un resumen de los endpoints del sistema. Para más detalles sobre los parámetros y sus respuestas, se puede consultar el Anexo E.

Eventos

- GET /api/v1/events/: Entrega un listado de todos los eventos disponibles. Tiene la posibilidad de filtrar la respuesta con varios campos.
- POST /api/v1/events/: Permite la creación de un nuevo evento.
- GET /api/v1/events/{id}: Recupera la información detallada de un evento. específico
- PUT /api/v1/events/{id}: Actualiza la información de un evento existente.
- DELETE /api/v1/events/{id}: Elimina un evento específico del sistema.
- GET /api/v1/download/: Entrega un archivo CSV con todos los eventos. Tiene la posibilidad de filtrar la respuesta con varios campos.

Organizadores

- GET /api/v1/organizers/: Proporciona una lista de todos los organizadores registrados
- POST /api/v1/organizers/: Permite registrar un nuevo organizador en el sistema
- GET /api/v1/organizers/{organizer_id}: Obtiene los detalles de un organizador específico
- PUT /api/v1/organizers/{organizer_id}: Actualiza la información de un organizador existente
- DELETE /api/v1/organizers/{organizer_id}: Elimina un organizador del sistema

Lugares

- GET /api/v1/places/: Entrega un listado de todos los lugares disponibles
- POST /api/v1/places/: Permite agregar un nuevo lugar al sistema
- GET /api/v1/places/{place_id}: Recupera la información detallada de un lugar específico
- PUT /api/v1/places/{place_id}: Actualiza la información de un lugar existente
- DELETE /api/v1/places/{place_id}: Elimina un lugar específico del sistema

Documentos

- GET /api/v1/documents/: Proporciona una lista de todos los documentos almacenados
- POST /api/v1/documents/: Permite subir un nuevo documento al sistema
- GET /api/v1/documents/{document_id}: Recupera el archivo de un documento específico
- PUT /api/v1/documents/{document_id}: Actualiza la información o el archivo de un documento existente
- DELETE /api/v1/documents/{document_id}: Elimina un documento específico del sistema

Imágenes

- GET /api/v1/documents/images/: Entrega un listado de todas las imágenes almacenadas
- POST /api/v1/documents/images/: Permite subir una nueva imagen al sistema
- GET /api/v1/documents/images/{image_id}: Recupera el archivo de una imagen específica
- PUT /api/v1/documents/images/{image_id}: Actualiza la información o el archivo de una imagen existente
- DELETE /api/v1/documents/images/{image_id}: Elimina una imagen específica del sistema

Capítulo 5

Validación

La validación de la solución implementada en el Capítulo 4 se dividió en dos aspectos principales. Primero, se evaluó la opinión de los usuarios finales mediante una encuesta que midió la usabilidad del sistema. Segundo, se midió la mantenibilidad del sistema utilizando indicadores de calidad del código y se realizaron diversas pruebas unitarias para verificar su funcionamiento. Además, se añadió una validación extra para comprobar que el sistema implementado es lo suficientemente robusto para cumplir con los estándares del Área de Desarrollo.

5.1. Usabilidad del sistema

Para evaluar la usabilidad del sistema, se utilizó la Escala de Usabilidad del Sistema o System Usability Scale (SUS). Esta escala tiene un valor entre 0 y 100 y se deriva de las respuestas a diez preguntas realizadas a los usuarios de una aplicación o sistema:

1. Creo que me gustaría utilizar este sistema con frecuencia.
2. Encontré el sistema innecesariamente complejo.
3. Pensé que sistema era fácil de usar.
4. Creo que necesitaría el apoyo de un técnico para poder utilizar este sistema.
5. Descubrí que las diversas funciones de este sistema estaban bien integradas.
6. Pensé que había demasiada inconsistencia en este sistema.
7. Me imagino que la mayoría de la gente aprendería a utilizar este sistema muy rápidamente.
8. Encontré el sistema muy complicado de usar.
9. Me sentí muy seguro al utilizar sistema.
10. Necesitaba aprender muchas cosas antes de poder empezar con este sistema.

Cada pregunta se responde utilizando una escala de cinco puntos:

1. Totalmente en desacuerdo
2. En desacuerdo
3. Neutro
4. De acuerdo
5. Totalmente de acuerdo

El cálculo del índice de usabilidad se realiza mediante la siguiente fórmula, donde R_i representa el valor de la respuesta i:

$$2{,}5 \cdot [(R_1 + R_3 + R_5 + R_7 + R_9 - 5) + (25 - R_2 - R_4 - R_6 - R_8 - R_{10})]$$

Durante una reunión el día 19 de junio de 2024, se aplicó el SUS a las usuarias finales de la aplicación, Ana Martínez, periodista del departamento, y Paulette Filla, diseñadora gráfica. Realizaron una serie de tareas en orden, con el objetivo de explorar la mayoría de las funcionalidades vitales del sistema y familiarizarse con él. Las tareas incluyeron:

1. Eliminar la ubicación duplicada del sistema
2. Editar el nombre duplicado para diferenciar entre dos imágenes
3. Verificar si Ivana Bachmann figura como organizadora, si no existe agregarla
4. Verificar si Juan Álvarez figura como organizador, si no existe agregarlo
5. Agregar un nuevo organizador y asociarlo a un evento ya creado
6. Cambiar el estado de un evento publicado a no publicado
7. Cambiar el estado de un evento no publicado a publicado
8. Editar el cuerpo (texto) de un evento y aplicar estilo
9. Crear un evento completo agregando nuevos elementos cuando sea posible (documento, imagen, organizador y lugar)
10. Buscar cada elemento añadido en el nuevo evento entre cada listado y visualizarlo

A continuación se registraron los siguientes resultados:

Pregunta	Periodista	Diseñadora
1) Creo que me gustaría utilizar este sistema con frecuencia.	4	4
2) Encontré el sistema innecesariamente complejo.	1	3
3) Pensé que sistema era fácil de usar.	4	4
4) Creo que necesitaría el apoyo de un técnico para poder utilizar este sistema	1	2
5) Descubrí que las diversas funciones de este sistema estaban bien integradas.	4	5
6) Pensé que había demasiada inconsistencia en este sistema.	1	2
7) Me imagino que la mayoría de la gente aprendería a utilizar este sistema muy rápidamente.	3	5
8) Encontré el sistema muy complicado de usar.	1	2
9) Me sentí muy seguro al utilizar sistema.	4	4
10) Necesitaba aprender muchas cosas antes de poder empezar con este sistema.	2	2
Usabilidad:	82,5	77,5

Se obtuvo un puntaje promedio de 80,8, que al ser mayor a 68 (el promedio), indica que este sistema tiene una buena recepción, ya que es sencillo de usar y satisface las necesidades de los usuarios de manera efectiva. Finalmente, se realizaron las siguientes preguntas adicionales, haciendo referencia al sistema actual como el SANE:

- "En comparación al sistema actual , ¿Qué es lo que más te gusta?"
- "¿Sientes que falta algo en este nuevo sistema que esté en el actual sistema?"
- "¿Tienes algún comentario final o feedback?"

De las respuestas de periodista, se obtuvieron las siguientes conclusiones:

1. El sistema tiene menos pasos en comparación al SANE.
2. El sistema es más simple en su diseño y está pensado en las necesidades del usuario.
3. La interfaz del nuevo sistema es mucho más amigable, con texto más grande y mayor claridad. Además, se requiere abrir menos ventanas para realizar tareas y está todo lo necesario para editar un evento dentro del constructor.
4. No se extrañó nada trascendental o imprescindible del SANE en materia de eventos.

De las respuestas de diseñadora, se obtuvieron las siguientes conclusiones:

1. El sistema no parece estar sobrediseñado, en comparación al SANE.
2. El sistema visualmente se ve mejor y se apega al estilo actual del DCC.
3. Se podría mejorar el funcionamiento del botón de publicar/ocultar un evento, debido a que no entrega un feedback claro del estado del evento. También se debería revisar la forma en que se redirige al guardar los cambios en un evento.
4. Ciertos elementos podrían ordenarse un poco mejor dentro del espacio.

En resumen, considerando el índice de usabilidad obtenido (80 en promedio), se puede afirmar que el sistema cumple con los estándares de usabilidad para los usuarios. Los usuarios también destacaron la simplicidad y eficiencia del nuevo sistema, que se dedica exclusivamente a la administración de eventos. Valoraron que este sistema requiere menos pasos, está diseñado pensando en sus necesidades y presenta una interfaz más amigable. Además, se proporcionaron valiosos comentarios sobre la usabilidad, sugiriendo mejoras en el botón de publicar/ocultar eventos y en la redirección en la web.

5.2. Mantenibilidad del sistema

En esta sección se comentan los métodos utilizados para validar la mantenibilidad de la solución implementada. Se realizaron principalmente dos comprobaciones: por un lado, se calcularon dos métricas de software para compararlas con las actuales del SANE; y por otro lado, se llevaron a cabo un conjunto de pruebas unitarias para evaluar la cobertura del código. Finalmente, como medida adicional, se añadió una comprobación extra relacionada con una checklist de madurez del sistema para el Área de Desarrollo.

5.2.1. Métricas de software

En el ámbito del desarrollo de software, es crucial contar con indicadores cuantitativos que permitan evaluar la calidad del código. Estos indicadores proporcionan información valiosa sobre la facilidad de mantenimiento, la probabilidad de errores y la complejidad general del sistema. Para este análisis, se utilizó la librería *Radon* de Python, una herramienta especializada que permite calcular diversas métricas de calidad de software directamente a partir del código fuente. Entre las métricas más utilizadas y aceptadas que *Radon* proporciona se encuentran la complejidad ciclomática y el índice de mantenibilidad. Estas métricas ofrecen una visión objetiva de la calidad del código, permitiendo a los desarrolladores y gestores de proyectos tomar decisiones informadas sobre el mantenimiento y la evolución del software.

Complejidad Ciclomática

La complejidad ciclomática, introducida por Thomas J. McCabe en 1976 [12], es una de las métricas de software de más reconocidos debido a su independencia del lenguaje de programación. Esta métrica define el número de caminos independientes dentro de un fragmento de código y determina la cota superior del número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.

La librería *Radon* calcula la complejidad ciclomática contando el número de decisiones lógicas en el código fuente. En términos generales, *Radon* analiza el árbol de sintaxis abstracto del código fuente y aumenta la complejidad en 1 por cada *if*, *elif*, *for*, *while* , *except*, *assert*, *compresiones* (listas/conjuntos/diccionarios con operadores booleanos) y operadores booleanos (*and* y *or*) [13].

Los valores de la complejidad ciclomática se interpretan según los siguientes rangos:

Complejidad Ciclomática	Evaluación del Riesgo
1-10	Programa Simple, sin mucho riesgo
11-20	Más complejo, riesgo moderado
21-50	Complejo, Programa de alto riesgo
50	Programa no testeable, Muy alto riesgo

Tabla 5.1: Rangos de Complejidad Ciclomática

Índice de Mantenibilidad

El índice de mantenibilidad es una métrica que evalúa qué tan fácil es mantener, actualizar y comprender un código fuente. Esta métrica combina otras métricas como la complejidad ciclomática, la cantidad de líneas de código y el Volumen de Halstead para proporcionar una evaluación integral de la mantenibilidad del código.

La fórmula utilizada por *Radon* para calcular el índice de mantenibilidad es una derivada de la fórmula original propuesta por Coleman, Ash, Lowther y Oman en el año 1994 [13, 7]:

$$MI = \text{máx}\left[0, 100\frac{171 - 5{,}2\ln V - 0{,}23G - 16{,}2\ln L + 50\sin(\sqrt{2{,}4C})}{171}\right]$$

Donde:

- V es el volumen de Halstead
- G es la Complejidad ciclomática
- L es el número de líneas del código fuente (SLOC)
- C es el porcentaje de líneas comentadas

Los valores del índice de mantenibilidad se interpretan según la siguiente escala:

Valor	Mantenibilidad
100 - 20	Muy alta
19 - 10	Media
9 - 0	Muy baja

Tabla 5.2: Rangos del Índice de Mantenibilidad

Análisis de Métricas del Proyecto

Para el cálculo de estas métricas se utilizó la librería de Python *Radon*, que permite analizar el código fuente de un sistema y calcular diversas métricas de calidad. El análisis

se realizó sobre la carpeta raíz del proyecto, obteniendo valores para todos los archivos y funciones.

Complejidad Ciclomática

Los resultados del análisis de complejidad ciclomática son los siguientes:

- Número total de funciones/métodos analizados: 153
- Suma total de complejidades ciclomáticas: 324
- Promedio de complejidad ciclomática: 2.12

Con un promedio de complejidad ciclomática de 2.12, el sistema se sitúa en el rango más bajo de la escala (1-5), lo que indica un riesgo muy bajo. Esto sugiere que el código es simple, fácil de entender y de mantener.

Índice de Mantenibilidad

Para el índice de mantenibilidad, los resultados obtenidos son:

- Número total de archivos analizados: 68
- Suma total de índices de mantenibilidad: 5824.47
- Promedio del índice de mantenibilidad: 85.65

El promedio del índice de mantenibilidad de 85.65 se sitúa en el rango más alto de la escala (100-20), correspondiente a una mantenibilidad muy alta. Este resultado indica que el sistema es altamente mantenible, lo que facilita su desarrollo continuo y evolución a largo plazo.

Discusión

Al comparar las métricas obtenidas con las presentadas en la Memoria de F.Madrid [10], considerando tanto el back-end como el front-end, se obtiene la siguiente tabla:

Sistema	**Complejidad Ciclomática**	**Índice de Mantenibilidad**
Sistema implementado	2.12	85.65
SANE (Back-end)	2.68	89.47
SANE (Front-end)	1.19	76.81

Tabla 5.3: Comparación de las métricas promedio de complejidad ciclomática e índice de mantenibilidad

El análisis de las métricas de complejidad ciclomática e índice de mantenibilidad destaca la alta mantenibilidad del sistema implementado, con un índice de 85.65. Esta alta mantenibilidad se atribuye a la utilización del sistema de templates y funcionalidades propias de Django, que simplifica la arquitectura al integrar el manejo del front-end y back-end, sin utilizar ningún framework o librería de JavaScript (exceptuando algunas funcionalidades puntuales). En el caso del front-end del SANE, este fue implementado mediante la librería AngularJs.

Comparativamente, el back-end del SANE muestra una mantenibilidad ligeramente superior con un índice de 89.47, indicando una estructura de código bien gestionada. Sin embargo, el front-end presenta un índice de mantenibilidad más bajo (76.81), sugiriendo desafíos en la facilidad de mantenimiento del código del front-end.

Estos resultados indican que, aunque el sistema implementado tiene una mantenibilidad similar al del back-end, el front-end del SANE tiene una mantenibilidad menor. Considerando que el sistema implementado combina tanto front-end como back-end, se puede concluir que ha habido una mejora en este aspecto al integrar ambos ambientes, además de que sí sé considerar un promedio de los índices de mantenibilidad del front-end y el back-end, el sistema implementado tendría mejor mantenibilidad.

5.2.2. Pruebas unitarias

Las pruebas unitarias son una parte fundamental del desarrollo de software moderno, diseñadas para verificar el correcto funcionamiento de componentes individuales o unidades de código. Estas pruebas ayudan a facilitar la refactorización y mejora del código, y proporcionan una documentación viva de cómo se espera que funcione cada parte del sistema. Además, las pruebas unitarias aumentan la confiabilidad del software y permiten a los desarrolladores realizar cambios con mayor confianza.

En el contexto de esta implementación, las pruebas unitarias se enfocaron exclusivamente en el back-end del sistema. Esta decisión se tomó debido a que, al ser un backoffice, el sistema se centra principalmente en funcionalidades del back-end. La mayor parte de la aplicación fue desarrollada utilizando Python y características de Django, con solo algunas funcionalidades específicas del front-end implementadas en JavaScript. Por esta razón, se decidió concentrar las pruebas unitarias en el back-end.

Para la realización de estas pruebas se utilizó pytest, un framework de pruebas general para Python. Este framework se emplea para escribir y ejecutar pruebas unitarias, de integración y funcionales en cualquier proyecto Python. Además, se usó el plugin django-pytest, el cual facilita la integración de estas pruebas en Django, proporcionando utilidades para probar vistas, modelos, formularios y otros componentes específicos del framework.

Para evaluar la cobertura de estos tests, se empleó el plugin pytest-cov. Esta herramienta se utilizó para generar tanto un archivo .txt como un archivo HTML con los resultados detallados de la cobertura. A continuación se muestra una tabla que resume el resultado de las pruebas unitarias:

Statements	% de statements	Branches	% de branches	Cantidad de test
1607/1730	93 %	377/430	91 %	74

Tabla 5.4: Test Coverage del sistema implementado y la cantidad de tests

En cuanto a los resultados obtenidos, se observó una cobertura total del proyecto del 93 % en términos de declaraciones y un 91 % de cobertura de ramas en un total de 74 tests. Estos resultados son considerablemente buenos, ya que una cobertura por encima del 80 % es generalmente aceptada dentro de los estándares de la industria[8].

5.2.3. Checklist de madurez

Para garantizar la mantenibilidad del sistema a largo plazo y comprender su nivel de madurez desde una perspectiva de gobernabilidad, se utilizó como base el documento de Requisitos de Gobernabilidad e Integración de Sistemas del DCC [16]. En respuesta a un requerimiento del área de desarrollo, durante el año 2024 se definió una *Checklist* [4] que permite evaluar el nivel de madurez de los sistemas desarrollados por memoristas y tesistas dentro del DCC.

Esta checklist establece tres niveles de madurez: básico, intermedio y avanzado. Un sistema puede cumplir con puntos de diferentes niveles, pero para alcanzar un nivel superior, debe satisfacer todos los requisitos obligatorios del nivel específico. Los detalles completos de los niveles de la checklist se pueden consultar en el Anexo F.

Para alcanzar un nivel básico, el sistema debe contar con una descripción clara del problema o proceso que aborda, acompañada de sus objetivos y alcance bien definidos. Además de documentar la arquitectura del software y su stack tecnológico utilizado. Por otro lado, también indicar su modelo de datos coherentes y directrices detalladas para el despliegue en producción. Finalmente, debe tener una validación de un tercero, como un profesor un guía.

En el nivel intermedio, además de cumplir con los requisitos básicos, el sistema debe demostrar una mayor madurez en su desarrollo. Esto implica adherirse a estándares de codificación y utilizar herramientas de linting para mantener la calidad del código. La documentación requiere de un README.md que incluye una descripción del proyecto y guías detalladas para su despliegue local y configuración. En este nivel, es se deben realizar pruebas exhaustivas en un ambiente similar al de producción.

El nivel avanzado , además de cumplir con todos los requisitos de los niveles anteriores, incluyendo la arquitectura del ecosistema en el que se integrará. La documentación se expande para incluir un resumen conciso del propósito del programa y su audiencia objetivo, así como una sección que aborde mejoras futuras y posibles evoluciones del sistema. Se pone especial énfasis en la facilidad de configuración y despliegue, requiriendo que un ambiente de desarrollo pueda ser levantado en menos de 30 minutos.

Resultado

A continuación se muestra la Tabla 5.5 con los resultados de la evaluación del sistema implementado según la checklist de madurez, realizada por Luis Herrera, miembro del área de Desarrollo de Software del DCC. Esta tabla detalla los ítems que el sistema cumple, categorizados en tres niveles: ítems BX para el nivel básico, IX para el nivel intermedio, y AX para el nivel avanzado:

Ítem	**Evaluación**	**Cumplimiento**
(B1) Incluye descripción del problema	Dentro del README	Cumple
(B2) Incluye Objetivo y alcance	Dentro del README	Cumple
(B3) Incluye Arquitectura del software	Dentro del README	Cumple
(B4) Incluye Stack tecnológico	Extiende del *django-boilerplate*	Cumple
(B5) Incluye modelo de datos	Dentro del README	Cumple
(B6) Indicaciones para el deployment	Extiende del *django-boilerplate*	Cumple
(B7) Validación de la funcionalidad por parte de un tercero	Validado con usuarios finales	Cumple
(B8) Indica APIs que disponibiliza (opcional)	Disponible en *api/v1/docs*	Cumple
(I1) Sigue y se ajusta a algún estándar de codificación	Extiende del *django-boilerplate*	Cumple
(I2) Utiliza y cumple con las recomendaciones de un linter	Extiende del *django-boilerplate*	Cumple
(I3) Incluye título y una descripción del proyecto.	Dentro del README	Cumple
(I4) Contiene información detallada sobre cómo desplegar de forma local	Dentro del README	Cumple
(I5) Incluye instrucciones sobre la configuración	Dentro del README	Cumple
(I6) Probado en un ambiente de testing similar a producción	Dentro del README	Cumple
(I7) Evidencia de que se ha realizado pruebas exhaustivas	Validado con usuarios finales	Cumple
(I8) El código se encuentra disponible en un VCS (opcional)	Extiende del *django-boilerplate*	Cumple
(I9) Probado en un ambiente de testing (opcional)	Este ítem es opcional	Cumple
(A1) Incluye Arquitectura del ecosistema	Dentro del README	Cumple
(A2) Incluye un resumen conciso del propósito y usuarios objetivo	Dentro del README	Cumple
(A3) Incluye una sección sobre mejoras futuras	Dentro del README	Cumple
(A4) Contiene indicaciones sobre archivos de configuración	Dentro del README	Cumple
(A5) Levanta un ambiente de desarrollo en 30 minutos	Extiende del *django-boilerplate*	Cumple
(A6) Disponibiliza APIs para integrar sistemas (opcional)	Disponible en *api/v1/docs*	Cumple
(A7) Probado en un ambiente de testing del ADS (opcional)	Este ítem es opcional	No Aplica

Tabla 5.5: Ítems de la checklist evaluados ordenados por nivel.

Gran parte de los elementos requeridos por el área de desarrollo consisten en diagramas y documentación, los cuales se han incluido en el archivo README.md del proyecto (ver Sección G del Anexo). Además, al utilizar el template denominado django-boilerplate, proporcionado por el área de desarrollo, el sistema cumplía de antemano con varios puntos de la evaluación. Tras la revisión mediante la checklist y la validación por parte del área de desarrollo, se ha determinado que este sistema alcanza un nivel de madurez avanzado, cumpliendo con todos los puntos requeridos.

Este nivel de madurez indica que el sistema está suficientemente desarrollado para su despliegue en un entorno de producción y cumple con los estándares actuales del DCC en términos de mantenibilidad. Esta clasificación asegura que el sistema no solo es funcional, sino que también es sostenible a largo plazo dentro de la infraestructura del departamento. Además, en cuanto al impacto en el DCC, el sistema promueve buenas prácticas, mantiene altos estándares de calidad, facilita la supervisión y evaluación, y permite una integración eficiente con otros proyectos, potenciando así su escalabilidad.

Capítulo 6

Conclusiones

El sistema actual, aunque funcional, presentaba varios problemas en la administración de los eventos del DCC que afectaban negativamente su usabilidad. El ejemplo más claro era el constructor, con una interfaz sobrediseñada que complicaba su uso cuando solo algunas opciones se utilizaban cotidianamente. Además, el SANE, al ser un sistema relativamente antiguo, tenía muchos componentes desactualizados y albergaba demasiadas responsabilidades, lo que aumentaba el riesgo y las complicaciones en su mantenibilidad.

El trabajo realizado en esta memoria se enfocó en el diseño de un nuevo sistema, centrado exclusivamente en eventos. Se evaluaron iterativamente los diseños y mockups de la interfaz con los usuarios finales, logrando implementar cambios tanto en aspectos visuales como en funcionalidades, ayudando a completar el objetivo específico (1) y (2).

El proyecto también requirió un rediseño del modelo de datos, tomando como referencia el modelo del SANE, pero considerando solo los eventos. Se eliminaron componentes que en la práctica no se utilizaban y otros que no serían necesarios, como el modelo de usuarios, ya que este campo estaba cubierto por el template proporcionado por el área de desarrollo del DCC, lo que ayudo a completar en gran parte el objetivo específico (1).

Se rescataron las ideas de los endpoints del SANE, especialmente los expuestos para entregar el listado de eventos, y se implementó el CRUD de elementos del sistema para conectar este backoffice a otros , aportando a completar el objetivo específico (1) y (3). De forma provisional, se dejaron expuestos solo los endpoints del tipo GET, mientras que los demás métodos (POST, DELETE y PUT) se configuraron para requerir autenticación previa, pudiendo ser usados solo dentro del sistema al iniciar sesión mediante el portal DCC.

Los resultados obtenidos mediante la evaluación de los usuarios con la escala SUS fueron bastante positivos. El sistema demostró ser amigable, y los usuarios resaltaron su simpleza y diseño apegado al estilo visual del DCC. Esta validación demostró que este sistema, en comparación con el SANE, mejoraba significativamente su usabilidad, contribuyendo a cumplir el objetivo específico (4).

Respecto a las métricas del código, a pesar de no tener valores muy distintos entre sí, se pudo deducir que hay un grado de mejora. Considerando que el sistema es principalmente

un back-end, se podría decir que el objetivo (5) se cumplió en gran parte al igualar estas métricas, junto con lograr obtener una cobertura de pruebas unitarias superior al 90 % en las declaraciones y las ramas. Adicionalmente, se incluyó una validación extra consistente en una checklist respaldada y validada por el área de desarrollo, remarcando elementos que tiene el sistema y esta memoria que permiten tener una documentación clara del sistema, algo fundamental para su mantenimiento, ayudando así a cumplir el objetivo principal de esta memoria: crear un sistema enfocado en eventos más usable y mantenible para el DCC.

Como trabajo futuro, se propone integrar la API de este sistema a otros del DCC, como las TVs del departamento y la web del DCC. Aunque se incluyeron campos booleanos dentro de los eventos para este propósito, estos no se llegaron a probar finalmente. Actualmente, los endpoints necesarios para esto están disponibles. Una de las razones por las que no se implementó esta integración es la necesidad de exportar y adaptar los aproximadamente 300 eventos creados por el SANE al nuevo modelo de datos simplificado, utilizando los endpoints de eventos de la API del SANE. Esto es importante para evitar la pérdida de información sobre eventos previos, como ocurrió en el cambio del sistema anterior al SANE.

También se podrían mejorar algunos aspectos estéticos, específicamente del constructor, aprovechando mejor los espacios según los comentarios de un usuario final. La navegación al crear un evento podría mejorarse en cuanto a qué vista mostrar, ya que actualmente la redirección a la vista previa podría ser confusa para algunos usuarios. En relación con la vista previa, podría mejorarse la experiencia de usuario, implementando una actualización más dinámica de los campos, sin necesidad de guardar los cambios, evitando así la pérdida de información al recargar o volver a la vista anterior.

El desarrollo de esta memoria requirió profundizar en el conocimiento de las tecnologías utilizadas, especialmente en el framework Django para crear el back-end de la aplicación. Aunque se utiliza en proyectos de algunas asignaturas, fue necesario ahondar en aspectos como la investigación de librerías para editar texto enriquecido y complementar las vistas de Django para autocompletar búsquedas. También se profundizó en el entendimiento de las métricas usadas para validar el software, el testing y la escala SUS para validar sistemas, conocimientos que servirán para futuros proyectos.

Finalmente, se valora la experiencia obtenida al trabajar con los usuarios finales, factor clave para el buen desarrollo de esta memoria. También fue muy positivo el trabajo conjunto con el área de desarrollo para estructurar el proyecto basado en el template y recibir recomendaciones que ayudaron al desarrollo, como el uso de otra librería para crear la API y la utilizacion de un metodo de validacion extra que ayudo a reforzar la mantenibilidad del proyecto a futuro.

Bibliografía

[1] *What is a REST API?* Sitio web. Visitado el 17 de septiembre de 2023, de `https://www.ibm.com/topics/rest-apis`.

[2] *API versioning.* Sitio web. Visitado el 29 de noviembre de 2023, de `https://blog.treblle.com/api-versioning-all-you-need-to-know/`.

[3] *Backoffice definition.* Sitio web. Visitado el 18 de septiembre de 2023, de `https://www.netsuite.com/portal/resource/articles/human-resources/back-office.shtml`.

[4] *Checklist de Entrega para Sistemas Desarrollados por Estudiantes.* Repositorio de GitHub del Área de desarrollo del DCC. Visitado el 17 de junio de 2024, de `https://github.com/DCC-FCFM-UCHILE/dev.dcc.uchile.cl/blob/main/CHECKLIST_ENTREGA.md`.

[5] *Is your code too complicated?* Sitio web. Visitado el 18 de octubre de 2023, de `https://sourcery.ai/blog/code_complexity/`.

[6] *Cohesión y Acoplamiento.* Sitio web. Visitado el 18 de octubre de 2023, de `https://blog.koalite.com/2015/02/cohesion-y-acoplamiento/`.

[7] Coleman, D., D. Ash, B. Lowther y P. Oman: *Using metrics to evaluate software system maintainability.* Computer, 27(8):44–49, 1994.

[8] *Code Coverage Best Practices.* Sitio web. Visitado el 18 de julio de 2024, de `https://testing.googleblog.com/2020/08/code-coverage-best-practices.html`.

[9] Cáceres Muñoz, D: *Desarrollo de un sistema de gestión de actividades de extensión.* Disponible en `https://repositorio.uchile.cl/handle/2250/182763`, 2021.

[10] Madrid Cabezas, F: *Integración y evolución de sistemas de información del DCC.* Disponible en `https://repositorio.uchile.cl/handle/2250/148981`, 2017.

[11] *Maintainability Index - What is it and where does it fall short?* Sitio web. Visitado el 18 de octubre de 2023, de `https://sourcery.ai/blog/maintainability-index/`.

[12] McCabe, T.J.: *A Complexity Measure.* IEEE Transactions on Software Engineering, SE-2(4):308–320, 1976.

[13] *Introduction to Code Metrics.* Sitio web. Visitado el 06 de julio de 2024, de `https://radon.readthedocs.io/en/latest/intro.html#maintainability-index`.

[14] *¿Qué son y para qué sirven los microservicios?* Sitio web. Visitado el 05 de diciembre de 2023, de https://www.redhat.com/es/topics/microservices.

[15] *What is Software Reengineering.* Sitio web. Visitado el 3 de septiembre de 2023, de https://fullscale.io/blog/software-reengineering/.

[16] *Requisitos de gobernabilidad e integración de sistemas del DCC.* Repositorio de GitHub del Área de desarrollo del DCC. Visitado el 6 de agosto de 2023, de https://github.com/DCC-FCFM-UCHILE/dev.dcc.uchile.cl/blob/main/REQUISITOS_GOBERNABILIDAD.md.

[17] *Why is it so important to use up-to-date software versions?* https://medium.com/@tferreiraw/why-is-it-so-important-to-use-up-to-date-software-versions-d69c7628f1ec, Visitado el 16-09-2023.

Anexo A

Administración de noticias y eventos del SANE

Cuando un administrador inicia sesión, tiene a su disposición las opciones 'Mi perfil', 'Administración' y 'Salir'. La sección de administración permite al usuario acceder a diferentes pestañas: eventos, noticias, imágenes, documentos, lugares, organizadores, etiquetas y usuarios. Dependiendo de la pestaña seleccionada, los usuarios pueden acceder a un panel de administración donde tienen la posibilidad de ver, editar o eliminar elementos. Esto se ilustra en la Figura A.1 para el caso de las noticias, y en la Figura A.2 para los eventos, ambos extraídos de la memoria de F. Madrid [10].

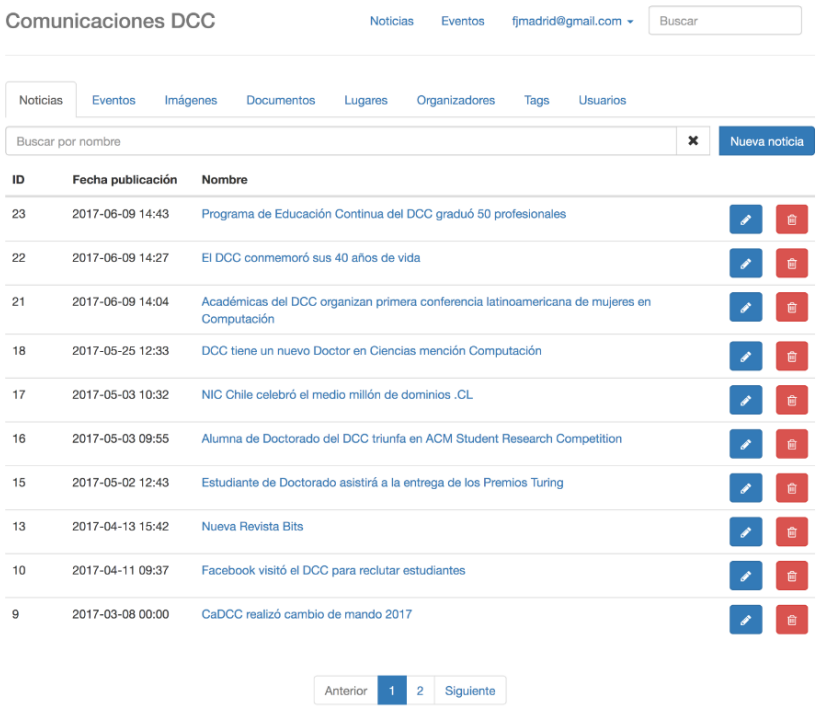


Figura A.1: Pestaña de administración de noticias desde el SANE.

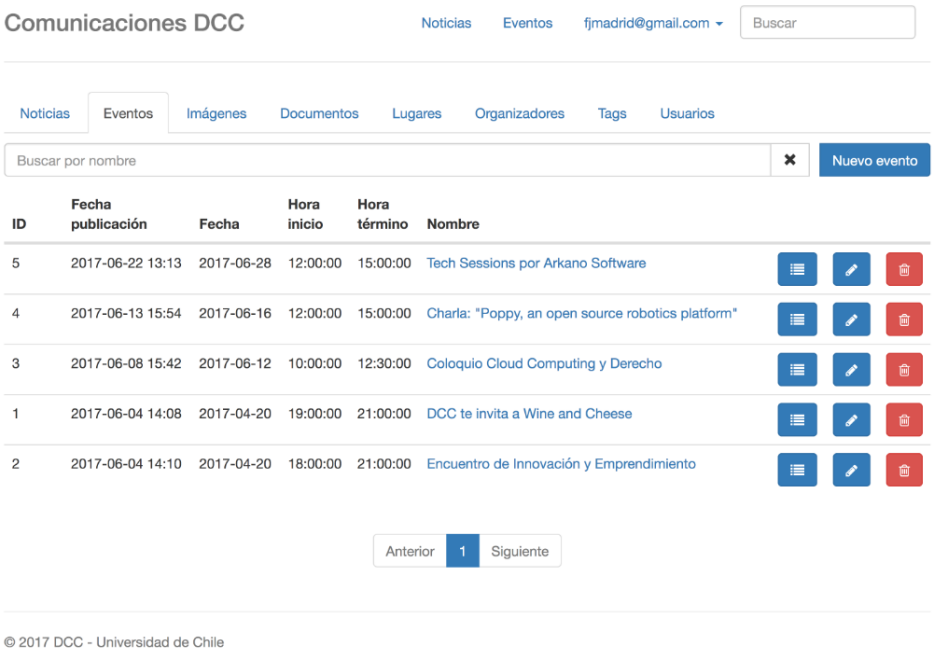


Figura A.2: Pestaña de administración de eventos desde el SANE.

Anexo B

Opciones para la publicación de un evento

Al presionar el botón "Publicación" de la sección G en la Figura 2.3, se despliega una ventana que tiene varios campos obligatorios para publicar un evento, como la categoría, el organizador, la fecha de publicación, la fecha del evento, la hora de publicación y la hora de finalización, como se ve en la Figura B.1.

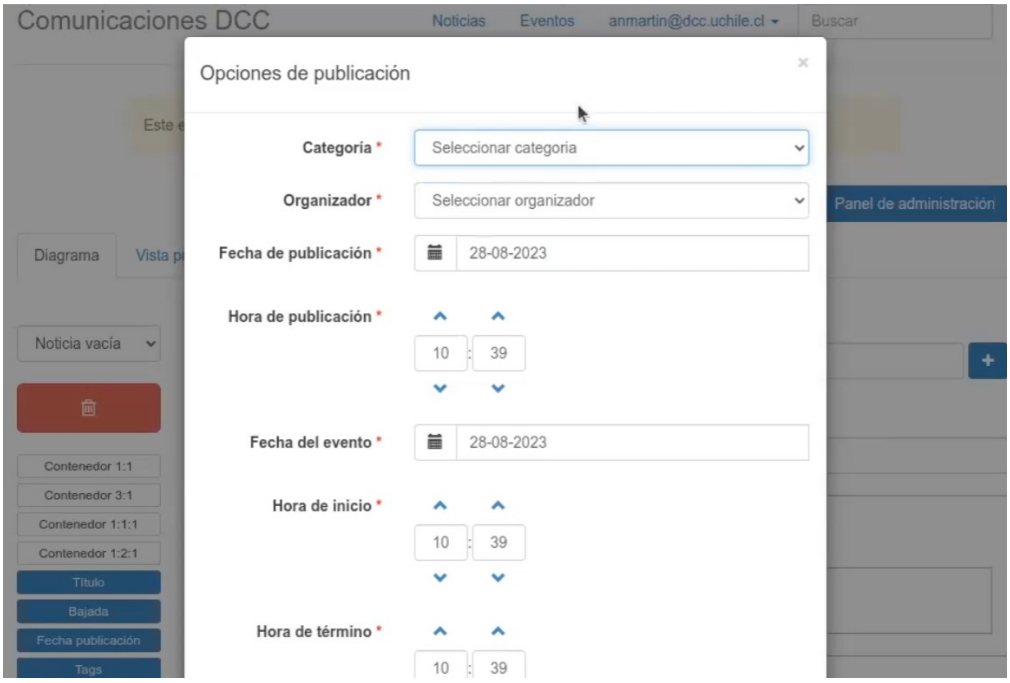


Figura B.1: Ventana de opciones para la publicación de un evento.

Anexo C

Endpoints declarados en el SANE

Al hacer un análisis de la API actual del SANE, se enlistan los siguientes endpoints relacionados a los eventos dentro del sistema:

1. **GET** */api/v1/events/*: Obtiene una lista de eventos, filtrando los eliminados e inactivos.
2. **GET** */api/v1/events/{event_id}/*: Obtiene detalles de un evento específico.
3. **POST** */api/v1/events/*: Crea un nuevo evento.
4. **PUT** */api/v1/events/{event_id}/*: Actualiza detalles de un evento específico.
5. **DELETE** */api/v1/events/{event_id}/*: Elimina un evento específico.
6. **GET** */api/v1/events/{event_id}/place-reservation/*: Obtiene detalles de la reserva de lugar para un evento.
7. **POST** */api/v1/events/{event_id}/place-reservation/*: Crea una nueva reserva de lugar para un evento.
8. **DELETE** */api/v1/events/{event_id}/place-reservation/*: Elimina una reserva de lugar para un evento.
9. **GET** */api/v1/events/{event_id}/registrations/download/*: Descarga un archivo CSV con detalles de usuarios registrados en un evento.
10. **DELETE** */api/v1/events/{event_id}/carousel/*: Elimina un evento específico.
11. **GET** */api/v1/events/dump*: Obtiene una lista de eventos con todos los datos.

De estos endpoints, los que son usados de forma pública, es decir, no requieren autorización de ningún tipo para su uso, son los números (1) y (2).

Anexo D

Diagrama entidad relación del SANE

Actualmente, el SANE cuenta con el siguiente diagrama entidad relación de la Figura D.1 para manejar eventos y noticias de manera detallada, extraída del Anexo B.2 de la memoria de F. Madrid [10].

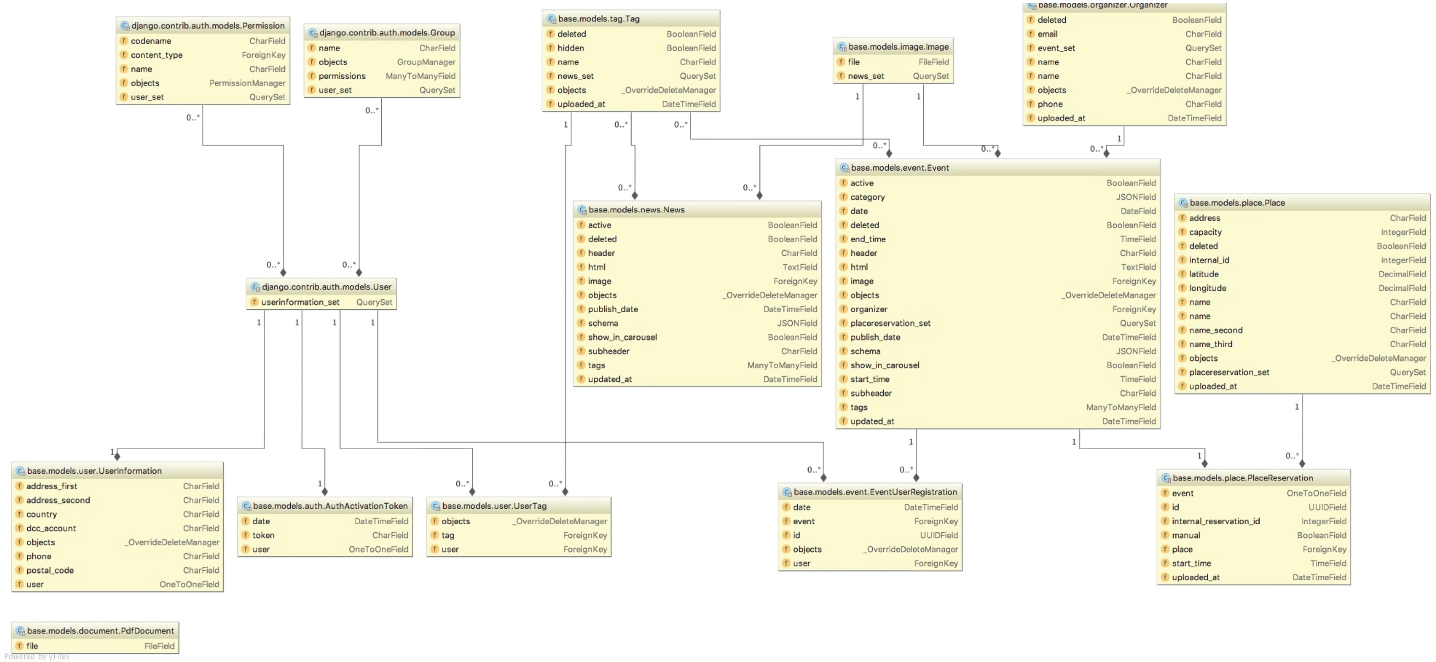


Figura D.1: Diagrama entidad relación detallado del SANE generado por el framework Django.

Anexo E

Documentación Endpoints de la API

En esta sección se presentan los endpoints creados para este sistema, junto con los datos esperados y las respuestas en formato JSON cuando corresponde. Estos endpoints también son accesibles de manera interactiva en la ruta */api/v1/docs*, generada automáticamente por Django Ninja a través de *SwaggerUI*[1].

La documentación interactiva proporcionada por SwaggerUI ofrece una interfaz intuitiva que permite a los desarrolladores y usuarios de la API explorar y probar los endpoints directamente desde el navegador. Esto facilita la comprensión y el uso de la API, ya que proporciona:

- Una lista completa de todos los endpoints disponibles.
- Detalles sobre los métodos HTTP soportados (GET, POST, PUT, DELETE, etc.).
- Esquemas de los datos de entrada requeridos para cada endpoint.
- Ejemplos de respuestas para cada endpoint.
- La capacidad de probar los endpoints directamente desde la interfaz.

A continuación, se detallan los principales endpoints de la API, incluyendo su propósito, los parámetros que aceptan y el tipo de respuesta que proporcionan. Esta documentación sirve como referencia rápida para los desarrolladores que trabajen con la API, complementando la interfaz interactiva proporcionada por SwaggerUI.

E.1. Eventos

E.1.1. Listar Eventos

- **Endpoint**: GET /api/v1/events/

[1]Sitio web de Swagger: `https://swagger.io/tools/swagger-ui/`

- **Descripción**: Obtiene una lista paginada de eventos.
- **Campos de filtro**:
  - `date`: Fecha (opcional)
  - `place_id`: ID del lugar (entero, opcional)
  - `mode`: Modo del evento (string, opcional)
  - `organizer_id`: ID del organizador (entero, opcional)
  - `start_date`: Fecha de inicio (string, opcional)
  - `end_date`: Fecha de finalización (string, opcional)
  - `limit`: Límite de resultados por página (defecto: 100)
  - `offset`: Desplazamiento para paginación (defecto: 0)
- **Respuesta esperada**: Un objeto `PagedEventOut` que contiene una lista de eventos y el conteo total.


```
{
  "items": [
    {
      "name": "string",
      "date": "2024-07-18",
      "published_date": "2024-07-18",
      "published_time": "00:44:08.349Z",
      "start_time": "00:44:08.349Z",
      "end_time": "00:44:08.349Z",
      "organizer": {
        "name": "string",
        "email": "string",
        "id": 0
      },
      "speakers": "string",
      "description": "string",
      "published": false,
      "published_on_tv": false,
      "published_on_web": false,
      "link": "string",
      "mode": "string",
      "image": {
        "name": "string",
        "path": "string",
        "id": 0
      },
      "document": {
        "name": "string",
        "path": "string",
        "id": 0
      },
      "place": {
```

```
        "name": "string",
        "address": "string",
        "detail": "string",
        "id": 0
      },
      "scheme": 0,
      "id": 0
    }
  ],
  "count": 0
}
```

E.1.2. Crear Evento

- **Endpoint**: POST /api/v1/events/
- **Descripción**: Crea un nuevo evento.
- **Datos esperados**: Un objeto `EventPost` con los detalles del evento.

```
{
  "name": "string",
  "date": "2024-07-18",
  "published_date": "2024-07-18",
  "published_time": "00:56:30.388Z",
  "start_time": "00:56:30.388Z",
  "end_time": "00:56:30.388Z",
  "organizer_id": 0,
  "speakers": "string",
  "description": "string",
  "published": false,
  "published_on_tv": false,
  "published_on_web": false,
  "link": "string",
  "mode": "0",
  "image_id": 0,
  "place_id": 0,
  "scheme": 0
}
```

- **Respuesta esperada**: Un objeto `EventResponse` con los detalles del evento creado.

```
{
  "name": "string",
  "date": "2024-07-18",
  "published_date": "2024-07-18",
  "published_time": "00:56:30.389Z",
  "start_time": "00:56:30.389Z",
  "end_time": "00:56:30.389Z",
  "organizer_id": 0,
```

```
  "speakers": "string",
  "description": "string",
  "published": false,
  "published_on_tv": false,
  "published_on_web": false,
  "link": "string",
  "mode": "0",
  "image_id": 0,
  "place_id": 0,
  "scheme": 0,
  "id": 0
}
```

E.1.3. Obtener Evento

- **Endpoint**: GET /api/v1/events/{id}
- **Descripción**: Obtiene los detalles de un evento específico.
- **Respuesta esperada**: Un objeto `EventOut` con los detalles del evento.

```
{
  "name": "string",
  "date": "2024-07-18",
  "published_date": "2024-07-18",
  "published_time": "00:56:07.255Z",
  "start_time": "00:56:07.255Z",
  "end_time": "00:56:07.255Z",
  "organizer": {
    "name": "string",
    "email": "string",
    "id": 0
  },
  "speakers": "string",
  "description": "string",
  "published": false,
  "published_on_tv": false,
  "published_on_web": false,
  "link": "string",
  "mode": "string",
  "image": {
    "name": "string",
    "path": "string",
    "id": 0
  },
  "document": {
    "name": "string",
    "path": "string",
    "id": 0
```

```
  },
  "place": {
    "name": "string",
    "address": "string",
    "detail": "string",
    "id": 0
  },
  "scheme": 0,
  "id": 0
}
```

E.1.4. Actualizar Evento

- **Endpoint**: PUT /api/v1/events/{id}
- **Descripción**: Actualiza los detalles de un evento existente.
- **Datos esperados**: Un objeto `EventUpdate` con los campos a actualizar.

```
{
  "name": "string",
  "date": null,
  "published_date": null,
  "published_time": "00:56:30.399Z",
  "start_time": "00:56:30.399Z",
  "end_time": "00:56:30.399Z",
  "organizer_id": 0,
  "speakers": "string",
  "description": "string",
  "published": true,
  "published_on_tv": true,
  "published_on_web": true,
  "link": "string",
  "mode": "string",
  "image_id": 0,
  "place_id": 0,
  "scheme": 0
}
```

- **Respuesta esperada**: Un objeto `EventOut` con los detalles actualizados del evento.

```
{
  "name": "string",
  "date": "2024-07-18",
  "published_date": "2024-07-18",
  "published_time": "00:56:30.400Z",
  "start_time": "00:56:30.400Z",
  "end_time": "00:56:30.400Z",
  "organizer": {
    "name": "string",
```

```
    "email": "string",
    "id": 0
  },
  "speakers": "string",
  "description": "string",
  "published": false,
  "published_on_tv": false,
  "published_on_web": false,
  "link": "string",
  "mode": "string",
  "image": {
    "name": "string",
    "path": "string",
    "id": 0
  },
  "document": {
    "name": "string",
    "path": "string",
    "id": 0
  },
  "place": {
    "name": "string",
    "address": "string",
    "detail": "string",
    "id": 0
  },
  "scheme": 0,
  "id": 0
}
```


E.1.5. Eliminar Evento

- **Endpoint**: DELETE /api/v1/events/{id}
- **Descripción**: Elimina un evento específico.
- **Respuesta esperada**: Código de estado 200 si la operación fue exitosa.

E.1.6. Descargar Eventos

- **Endpoint**: GET /api/v1/events/a/
- **Descripción**: Descarga eventos con filtros opcionales.
- **Campos de filtro**: Similares a los de Listar Eventos.
- **Respuesta esperada**: Un archivo descargable con los eventos filtrados.

E.2. Organizadores

E.2.1. Listar Organizadores

- **Endpoint**: GET /api/v1/organizers/
- **Descripción**: Obtiene una lista paginada de organizadores.
- **Campos de filtro**:
  - `limit`: Límite de resultados por página (defecto: 100)
  - `offset`: Desplazamiento para paginación (defecto: 0)
- **Respuesta esperada**: Un objeto `PagedOrganizerOut` que contiene una lista de organizadores y el conteo total.

```
{
  "items": [
    {
      "name": "string",
      "email": "string",
      "id": 0
    }
  ],
  "count": 0
}
```

E.2.2. Crear Organizador

- **Endpoint**: POST /api/v1/organizers/
- **Descripción**: Crea un nuevo organizador.
- **Datos esperados**: Un objeto `OrganizerIn` con los detalles del organizador.

```
{
  "name": "string",
  "email": "string"
}
```

- **Respuesta esperada**: Un objeto `OrganizerOut` con los detalles del organizador creado.

```
{
  "name": "string",
  "email": "string",
  "id": 0
}
```

E.2.3. Obtener Organizador

- **Endpoint**: GET /api/v1/organizers/{organizer_id}
- **Descripción**: Obtiene los detalles de un organizador específico.
- **Respuesta esperada**: Un objeto `OrganizerOut` con los detalles del organizador.

```
{
  "name": "string",
  "email": "string",
  "id": 0
}
```

E.2.4. Actualizar Organizador

- **Endpoint**: PUT /api/v1/organizers/{organizer_id}
- **Descripción**: Actualiza los detalles de un organizador existente.
- **Datos esperados**: Un objeto `OrganizerUpdate` con los campos a actualizar.

```
{
  "name": "string",
  "email": "string"
}
```

- **Respuesta esperada**: Un objeto `OrganizerOut` con los detalles actualizados del organizador.

```
{
  "name": "string",
  "email": "string",
  "id": 0
}
```

E.2.5. Eliminar Organizador

- **Endpoint**: DELETE /api/v1/organizers/{organizer_id}
- **Descripción**: Elimina un organizador específico.
- **Respuesta esperada**: Código de estado 200 si la operación fue exitosa.

E.3. Lugares

E.3.1. Listar Lugares

- **Endpoint**: GET /api/v1/places/

- **Descripción**: Obtiene una lista paginada de lugares.
- **Campos de filtro**:
  - `limit`: Límite de resultados por página (defecto: 100)
  - `offset`: Desplazamiento para paginación (defecto: 0)
- **Respuesta esperada**: Un objeto `PagedPlaceOut` que contiene una lista de lugares y el conteo total.

```
{
  "items": [
    {
      "name": "string",
      "address": "string",
      "detail": "string",
      "id": 0
    }
  ],
  "count": 0
}
```

E.3.2. Crear Lugar

- **Endpoint**: POST /api/v1/places/
- **Descripción**: Crea un nuevo lugar.
- **Datos esperados**: Un objeto `PlaceIn` con los detalles del lugar.

```
{
  "name": "string",
  "address": "string",
  "detail": "string"
}
```

- **Respuesta esperada**: Un objeto `PlaceOut` con los detalles del lugar creado.

```
{
  "name": "string",
  "address": "string",
  "detail": "string",
  "id": 0
}
```

E.3.3. Obtener Lugar

- **Endpoint**: GET /api/v1/places/{place_id}
- **Descripción**: Obtiene los detalles de un lugar específico.

- **Respuesta esperada**: Un objeto `PlaceOut` con los detalles del lugar.

```
{
  "name": "string",
  "address": "string",
  "detail": "string",
  "id": 0
}
```

E.3.4. Actualizar Lugar

- **Endpoint**: PUT /api/v1/places/{place_id}
- **Descripción**: Actualiza los detalles de un lugar existente.
- **Datos esperados**: Un objeto `PlaceUpdate` con los campos a actualizar.

```
{
  "name": "string",
  "address": "string",
  "detail": "string"
}
```

- **Respuesta esperada**: Un objeto `PlaceOut` con los detalles actualizados del lugar.

```
{
  "name": "string",
  "address": "string",
  "detail": "string",
  "id": 0
}
```

E.3.5. Eliminar Lugar

- **Endpoint**: DELETE /api/v1/places/{place_id}
- **Descripción**: Elimina un lugar específico.
- **Respuesta esperada**: Código de estado 200 si la operación fue exitosa.

E.4. Documentos

E.4.1. Listar Documentos

- **Endpoint**: GET /api/v1/documents/
- **Descripción**: Obtiene una lista paginada de documentos.

- **Campos de filtro**:
  - `limit`: Límite de resultados por página (defecto: 100)
  - `offset`: Desplazamiento para paginación (defecto: 0)
- **Respuesta esperada**: Un objeto `PagedDocumentOut` que contiene una lista de documentos y el conteo total.

```
{
 "items": [
    {
      "id": 0,
      "name": "string",
      "path": "string"
    }
  ],
  "count": 0
}
```

E.4.2. Crear Documento

- **Endpoint**: POST /api/v1/documents/
- **Descripción**: Crea un nuevo documento.
- **Datos esperados**: Un formulario con campos `name` (string) y `file` (archivo binario).
- **Respuesta esperada**: Un objeto `DocumentOut` con los detalles del documento creado.

```
{
  "id": 0,
  "name": "string",
  "path": "string"
}
```

E.4.3. Obtener Archivo de Documento

- **Endpoint**: GET /api/v1/documents/{document_id}
- **Descripción**: Obtiene el archivo de un documento específico.
- **Respuesta esperada**: El archivo del documento.

E.4.4. Actualizar Documento

- **Endpoint**: PUT /api/v1/documents/{document_id}
- **Descripción**: Actualiza los detalles de un documento existente.

- **Datos esperados**: Un objeto `DocumentUpdate` con el nombre actualizar.

```
{
  "name": "string"
}
```

- **Respuesta esperada**: Un objeto `DocumentOut` con los detalles actualizados del documento.

```
{
  "id": 0,
  "name": "string",
  "path": "string"
}
```

E.4.5. Eliminar Documento

- **Endpoint**: DELETE /api/v1/documents/{document_id}
- **Descripción**: Elimina un documento específico.
- **Respuesta esperada**: Código de estado 200 si la operación fue exitosa.

E.5. Imágenes

E.5.1. Listar Imágenes

- **Endpoint**: GET /api/v1/documents/images/
- **Descripción**: Obtiene una lista paginada de imágenes.
- **Campos de filtro**:
  - `limit`: Límite de resultados por página (defecto: 100)
  - `offset`: Desplazamiento para paginación (defecto: 0)
- **Respuesta esperada**: Un objeto `PagedImageOut` que contiene una lista de imágenes y el conteo total.

```
{
  "items": [
    {
      "id": 0,
      "name": "string",
      "path": "string"
    }
  ],
  "count": 0
}
```

E.5.2. Crear Imagen

- **Endpoint**: POST /api/v1/documents/images/
- **Descripción**: Crea una nueva imagen.
- **Datos esperados**: Un formulario con campos `name` (string) y `file` (archivo binario de imagen).
- **Respuesta esperada**: Un objeto `ImageOut` con los detalles de la imagen creada.

```
{
  "id": 0,
  "name": "string",
  "path": "string"
}
```

E.5.3. Obtener Archivo de Imagen

- **Endpoint**: GET /api/v1/documents/images/{image_id}
- **Descripción**: Obtiene el archivo de una imagen específica.
- **Respuesta esperada**: El archivo de la imagen.

E.5.4. Actualizar Imagen

- **Endpoint**: PUT /api/v1/documents/images/{image_id}
- **Descripción**: Actualiza los detalles de una imagen existente.
- **Datos esperados**: Un objeto `ImageUpdate` con el nombre a actualizar

```
{
  "name": "string"
}
```

- **Respuesta esperada**: Un objeto `ImageOut` con los detalles actualizados de la imagen.

```
{
  "id": 0,
  "name": "string",
  "path": "string"
}
```

E.5.5. Eliminar Imagen

- **Endpoint**: DELETE /api/v1/documents/images/{image_id}
- **Descripción**: Elimina una imagen específica.
- **Respuesta esperada**: Código de estado 200 si la operación fue exitosa.

Anexo F

Detalle de la Checklist de madurez

El área de desarrollo cuenta con una checklist que permite comprender el nivel de madurez de estos sistemas desarrollados por memorias desde una perspectiva de Gobernabilidad. A continuación se enlistan el desglose de los tres niveles:

F.1. Nivel basico

En un nivel básico, el sistema debe poseer de manera obligatoria:

1. Descripción del problema o proceso que busca resolver/abordar. Se recomienda utilizar la herramienta de Tablero Digital (IS2), BPMN o al menos una descripción en texto plano.
2. Objetivo y alcance del sistema. Se recomienda utilizar la herramienta de Tablero Digital (IS2) o al menos una descripción en texto plano.
3. Arquitectura del software. Se recomienda utilizar un framework de especificación de arquitectura como C4model o al menos una descripción clara en texto plano acompañada por diagramas que faciliten su comprensión).
4. Stack tecnológico. Debe incluir lenguajes de programación, plataformas, frameworks, sus versiones y configuraciones necesarias.
5. Modelo de datos. Se recomienda utilizar una herramienta de modelado como MySQL Workbench que permita revisar la coherencia de las relaciones.
6. Indicaciones para el deployment. Debe incluir los pasos y requisitos necesarios para que un tercero pueda levantar un ambiente de desarrollo y recomendaciones para su puesta en producción.
7. Existe evidencia de validación de la funcionalidad por parte de un tercero. Usualmente, la validación del profesor guia es suficiente.
8. (opcional) APIs que disponibiliza, en caso de que posea.

F.2. Nivel Medio

En un nivel intermedio, el sistema debe poseer de manera obligatoria:

1. El código sigue y se ajusta a algún estándar de codificación (por ejemplo, PEP8).
2. El código utiliza y cumple con las recomendaciones de un linter (por ejemplo, flake8).
3. El README.md incluye: título y una descripción del proyecto.
4. El README.md u otro archivo/documento en el repositorio contiene información detallada sobre cómo desplegar el sistema de forma local.
5. El README.md u otro archivo/documento en el repositorio incluye instrucciones sobre la configuración necesaria para el sistema, incluyendo las variables de entorno necesarias, dependencias y servicios externos necesarios y cómo instalar cualquier dependencia externa.
6. El sistema fue probado en un ambiente de testing con características similares a las de producción.
7. Existe evidencia de que se ha realizado pruebas exhaustivas de todas las funcionalidades del sistema.
8. (opcional) El código se encuentra disponible en un VCS de propiedad del ADS.
9. (opcional) El sistema fue probado en un ambiente de testing provisto por el ADS.

F.3. Nivel avanzado

En nivel avanzado, el sistema debe poseer de manera obligatoria lo siguiente:

1. Arquitectura del ecosistema. Se recomienda utilizar un framework de especificación de arquitectura como C4model o al menos una descripción clara en texto plano acompañada por diagramas que faciliten su comprensión).
2. El README.md u otro archivo/documento en el repositorio incluye un resumen conciso que describa qué hace el programa y para quién está destinado.
3. El README.md u otro archivo/documento en el repositorio incluye una sección sobre mejoras futuras tales como nuevas integraciones de API, optimizaciones de rendimiento y/o mejoras de interfaz, destacando también ideas no implementadas en la versión actual pero valiosas para desarrollos futuros.
4. El README.md u otro archivo/documento en el repositorio contiene indicaciones sobre archivos de configuración, variables de entorno y/o servicios externos que deben ser configurados previo a su puesta en test/producción.
5. Es posible levantar un ambiente de desarrollo desde cero en menos de 30 minutos.

6. (opcional) El sistema disponibiliza APIs que permiten a otros sistemas integrarse con él.
7. (opcional) El sistema fue probado en un ambiente de testing provisto por el ADS.

Anexo G

README del proyecto

En esta sección se registra el contendio que presenta el README del proyecto alojado en el repositorio de github que es propiedad del area de dearollo del DCC:

G.1. Descripción del Proyecto

COM-EVENTOS es un sistema de backoffice diseñado para el Área de Comunicaciones del Departamento de Ciencias de la Computación (DCC) de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. Su principal función es la gestión de eventos organizados por el departamento. Este proyecto para Django está construido utilizando el framework Django 5.x siguiendo algunas de las prácticas recomendadas en Two Scoops of Django 3.x. La documentación oficial de Django en conjunto con las buenas prácticas documentadas en el libro son recomendadas antes de modificar este proyecto.

G.1.1. Objetivos

- Ser la principal fuente de verdad para los eventos del departamento.
- Proporcionar una API para que otros sistemas del departamento puedan consumir la información de eventos.
- Reemplazar el actual Sistema de Administración de Noticias y Eventos (SANE) en lo que respecta a la gestión de eventos, con el objetivo de mejorar tanto la usabilidad como la facilidad de mantenimiento.

G.2. Descripción del Proceso que Apoya

G.2.1. Problema del sistema actual

El DCC organiza eventos regularmente. Actualmente, la gestión de estos eventos se realiza a través de un sistema que presenta problemas de usabilidad y mantenibilidad.

G.2.2. Actores

En este proyecto se proyectan los siguientes actores:

- Usuario del Área de Comunicaciones
- Sistema de Eventos
- API Eventos
- Sistemas externos

G.2.3. Macrotareas y Workflows

En cuanto a las macrotareas y workflows, se proyectan las siguientes:

1. Crear Evento: Un usuario del área de comunicaciones accede al Sistema de Eventos para crear un evento.
2. Editar Evento: Al crear un evento, se accede a un constructor de eventos donde el usuario puede editar la configuración y propiedades de un evento.
3. Agregar elementos a un evento: Un usuario del área de comunicaciones accede al Sistema de Eventos para agregar lugares, organizadores y documentos asociados a un evento dentro del constructor.
4. Visualizar listado de eventos: Dentro del sistema, un usuario del Área de Comunicaciones puede acceder a un listado de los eventos registrados con su estado e información relevante.
5. Disponibilizar evento mediante API: El evento creado y configurado se disponibiliza mediante una API para ser consumido por otros sistemas.
6. Consumir eventos y elementos mediante API: Otros sistemas externos consumen los eventos disponibles en la API para mostrarlos en sus interfaces.

G.2.4. Dolores del proceso actual

Este Sistema busca ser un remplazo del Sistema de Administración Noticias y Eventos (SANE) que actualmente se encuentra en uso en el Departamento de Ciencias de la Computación de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

- Problema: Actualmente el SANE maneja muchas responsabilidades
  - Causa: Manejo de Noticias y Eventos en un mismo sistema, junto con una web para acceder a los eventos
  - Oportunidad: Separar la gestión de eventos en un sistema aparte
- Problema: El SANE es un sistema con problemas de usabilidad
  - Causa: Interfaz de usuario poco amigable
  - Oportunidad: Crear un sistema con una interfaz de usuario amigable
- Problema: El SANE es un sistema que puede presentar desafios de mantenibilidad en el futuro
  - Causa: Código antiguo y sin actualizaciones desde el año 2017
  - Oportunidad: Crear un sistema con código actualizado y mantenible por el DCC en el futuro

G.3. Tecnologías Utilizadas

En este nuevo sistema se utilizan las siguientes tecnologías:

- Django 5.0.6
- Python 3.10.12
- PostgreSQL 13.3
- Django Ninja 1.1.0 (para la API)
- pytest-django 4.8.0

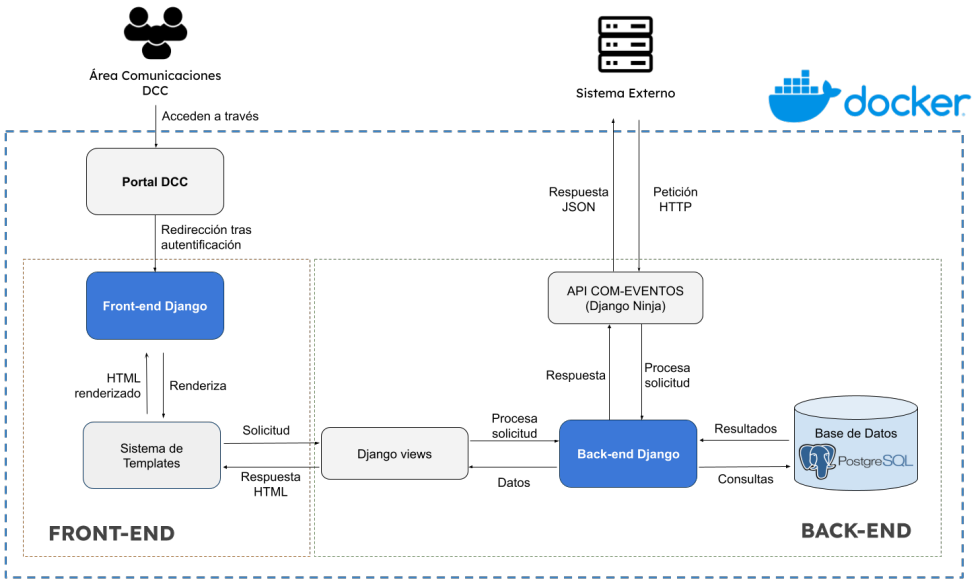
Para más detalles, consulte el directorio `_requirements` y `_npm` para el caso de librerías de JavaScript.

G.4. Arquitectura del sistema

Este sistema utiliza una arquitectura basada en Django, con una API REST para la disponibilidad de eventos. Los usuarios acceden a través de una interfaz web y se autentican usando sus credenciales de MiUchile a través del Portal DCC.

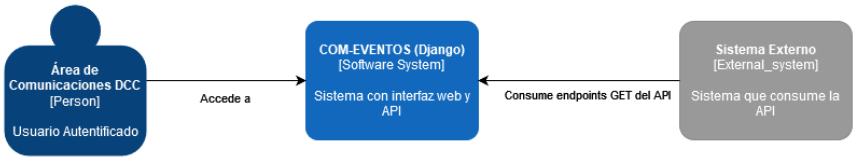
G.4.1. Diagramas de Arquitectura

A continuación se muestra un Diagrama de la arquitectura del sistema de forma más general:

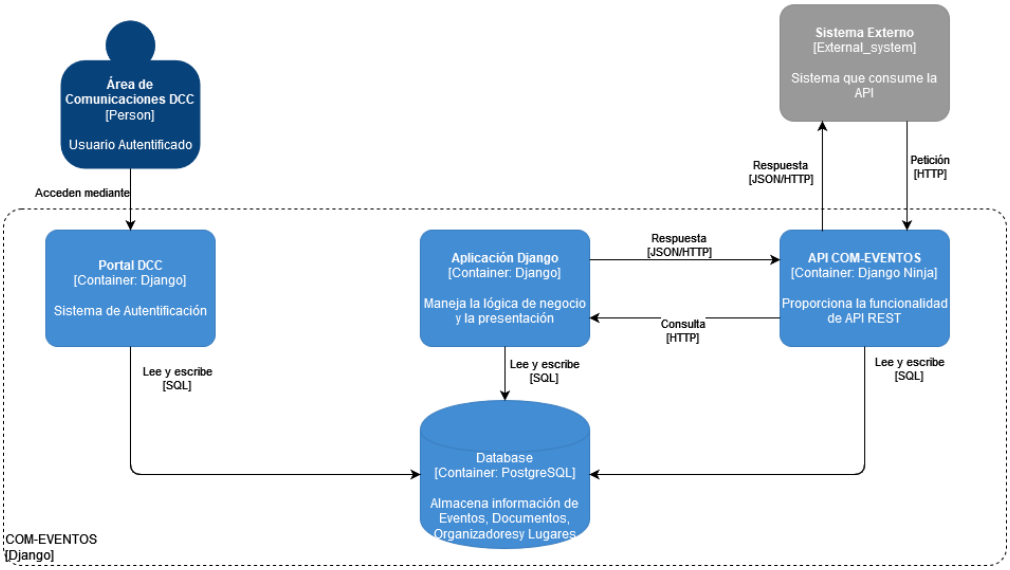


Por otro lado tambien muestran los siguientes diagramas de arquitectura basados en el modelo C4:

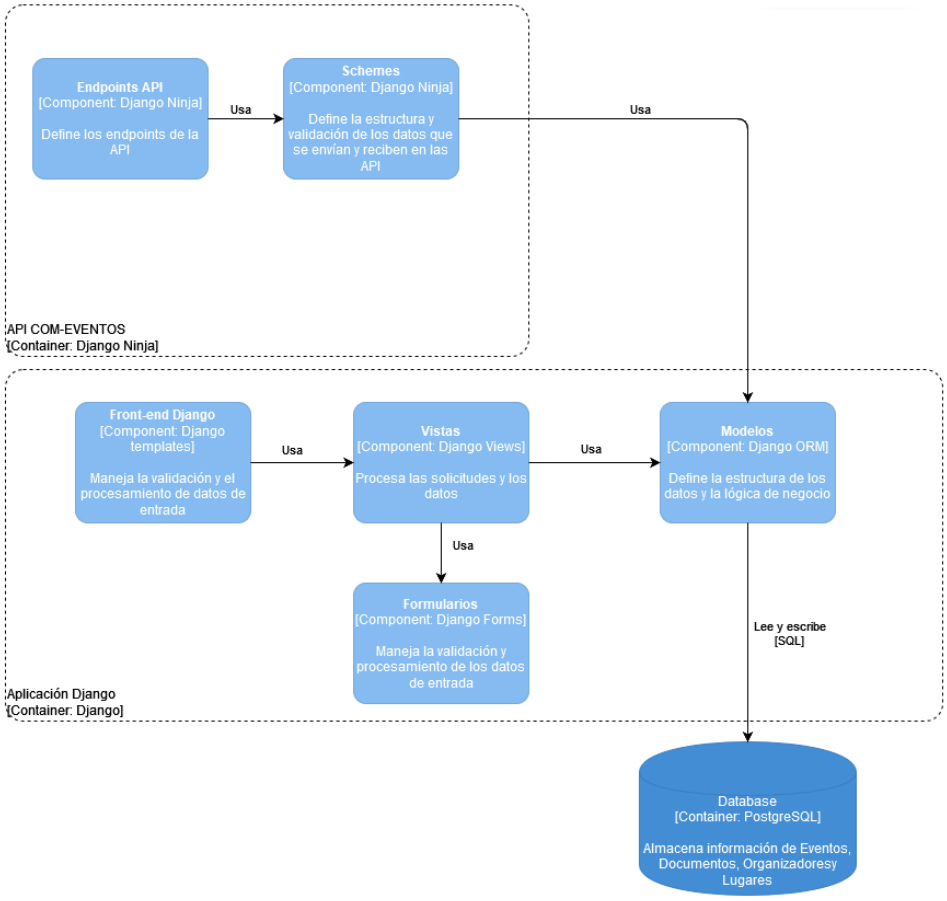
- Diagrama de Contexto



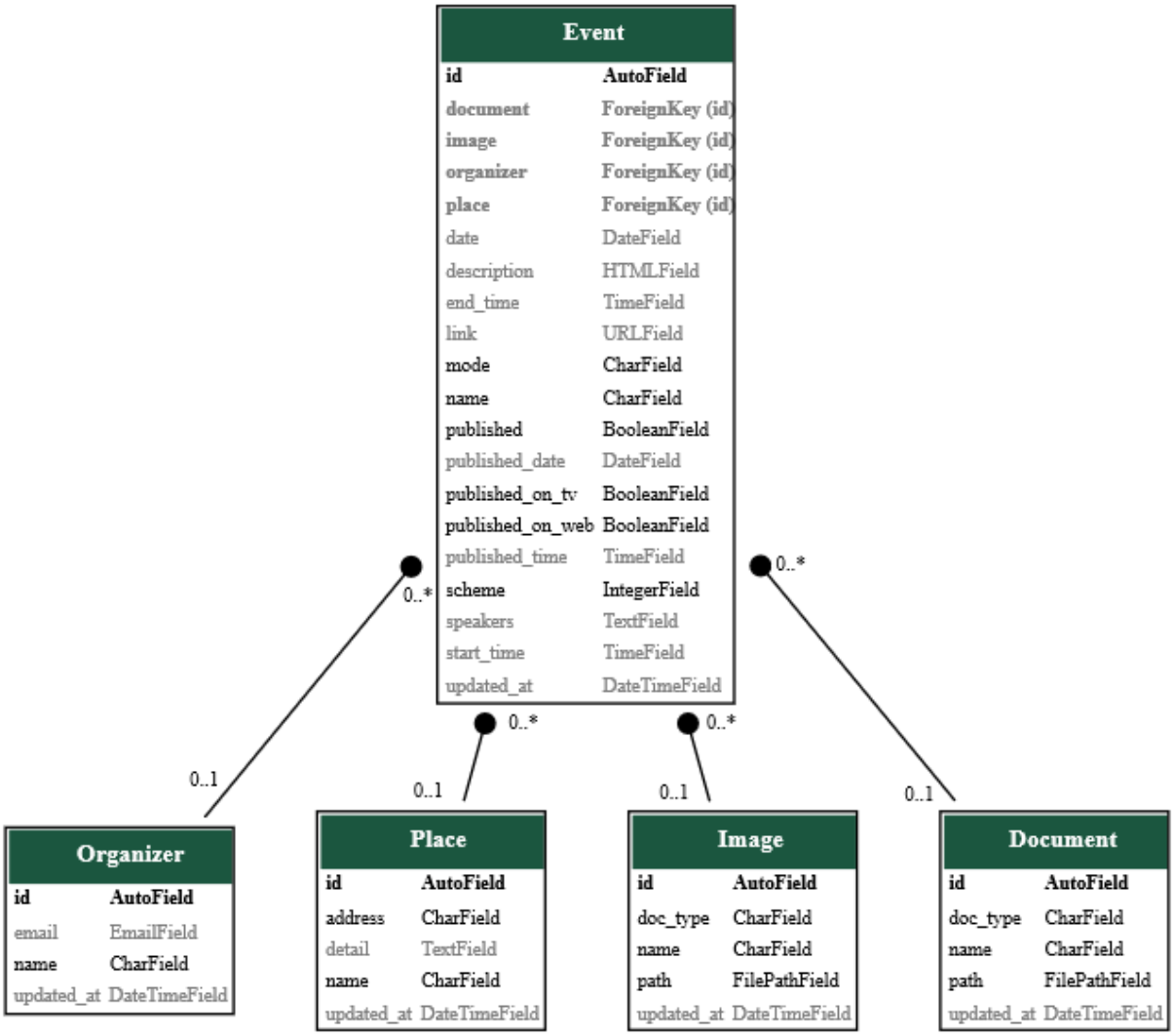
- Diagrama de Contenedores



- Diagrama de Componentes



G.4.2. Modelo de Datos



Las tablas Organizer, Place, Image y Document pueden asociarse a 0 o varios Eventos, y un evento puede asociarse a 0 o varios organizadores, lugares, imágenes y documentos.

G.5. Integración MiUchile

Este template incluye el proyecto `django_utils` que corresponde a una integración que permite utilizar las credenciales de MiUchile con este sistema. Para poder utilizar esta funcionalidad en el ambiente de desarrollo (local o gitpod) debe enviar un correo a:

`desarrollo@dcc.uchile.cl`

indicando los RUTs de los integrantes del equipo.

G.6. Documentación API

La documentación de la API está disponible en `'host'/api/v1/docs`, generada automáticamente con Swagger.

G.7. Permisos de Usuario

En ambientes de testing o producción, se recomienda crear un usuario superadmin para acceder a la interfaz de administración de Django. Es necesario ajustar los permisos para los usuarios del Área de Comunicaciones y Administradores del Área de desarrollo del DCC.

G.8. Posibles cambios o implementaciones futuras

A continuación se listan algunas de las posibles mejoras o implementaciones futuras que se pueden realizar en el sistema:

- Modificar el modelo de datos para fusionar las tablas de Imagen y Documento en una sola tabla.
- Mejorar reposponsividad de la interfaz del constructor de eventos.
- Evaluar adiciones o mejoras a la interfaz del constructor de eventos.
- Hacer una tarea de adaptar los datos de eventos existentes creados por el SANE al modelo de este nuevo sistema.
- Evaluar la forma de permitir accesos a endpoints de la API con un token de acceso.
- Implementar el uso de la API de eventos en sistemas externos del DCC.

G.9. Pruebas unitarias

Entre los elemetos de automatización se incluye la ejecución de pruebas unitarias. Estas pruebas se encuentran en los directorio `app/modulo/tests` donde `modulo` corresponde al modulo de la aplicación que se esta probando. Para correr las pruebas se puede ejecutar el siguiente comando dentro del directorio `app` (dentro del contenedor de Docker con Django):

```
make test
```

Este comando genera un archivo .txt y uno .html con el resultado de las pruebas en el directorio `app`.

G.10. Instalación y Configuración

G.10.1. Variables de entorno

El sistema necesita utilizar unas variables en entorno para funcionar correctamente en un ambiente de produccion. Estas variables deben ser configuradas en un archivo `.env` en el directorio `.docker/app/`. Por ejemplo en un ambiente de testing estas variables deben ser configuradas de la siguiente forma:

```
API_BASE="https://test.dcc.uchile.cl/comunicaciones/eventos/api/v1/"
API_BASE_URL="https://test.dcc.uchile.cl/comunicaciones/eventos/"
```

Estas variables ayudan a simplificar los usos de la API y la URL base del sistema. En un ambiente de desarrollo local estas variables no son necesarias.

G.10.2. Al clonar el repositorio

Al clonar un proyecto basado en este template recuerde hacerlo usando la opción `-recursive` para que se descargue junto con el submodulo django_utils.

```
git clone --recursive git@github.com:DCC-FCFM-UCHILE/com-eventos.git
```

G.10.3. Automatización

Este proyecto considera la automatización de algunas tareas, por ejemplo:

- Actualización automática de los requirements.
- Actualización automática de los submodules.
- Formateo automático usando PEP8 usando `black` y linter del código usando `flake8`.

Estas automatizaciones se basan en rutas y nombres que deben ser actualizadas. Revisa las disponibles en:

- app/Makefile

G.10.4. Al correrlo de forma local con Docker

Este proyecto se encuentra configurado con Docker para levantar un ambiente local de desarrollo. El Portal es compatible para trabajar con Apps que funcionen en localhost, para más información contacte al Área de Desarrollo del DCC.

```
dev@DCC ~/.docker/ $ docker-compose up -d --build
dev@DCC ~/.docker/ $ docker exec com-eventos python manage.py migrate
dev@DCC ~/.docker/ $ docker exec -it com-eventos python manage.py createsuperuser
```

Este proyecto incluye un archivo `.docker/Makefile` con comandos útiles para el desarrollo y correr el un ambiente de local/produccion.

```
Para correr el proyecto en local
cd .docker
make _build
Para correr el proyecto en produccion
cd .docker
make build
```