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

PLATAFORMA PARA LA ORGANIZACIÓN Y GESTIÓN DE EVENTOS DEPORTIVOS ESTUDIANTILES MASIVOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

SEBASTIÁN ALBERTO CISNEROS PINTO

PROFESORA GUÍA:
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:
JUAN ÁLVAREZ RUBIO
JÉRÉMY BARBAY

SANTIAGO DE CHILE
2023

Resumen

Desde el año 2016, se llevan a cabo anualmente los Juegos Deportivos de Ingeniería, conocidos como JING. Este es un evento deportivo inter-institucional en el que se reúnen ocho universidades de Chile y durante tres días se juega una serie de partidos, tanto de deportes convencionales como de e-sports y otras actividades recreativas. Las universidades compiten juntando puntos colectivamente de acuerdo a su posicionamiento en cada competencia, resultando como ganadora la institución con más puntos acumulados al final del evento.

La logística de este evento es considerablemente compleja, y es organizado simultáneamente por varias decenas de personas de distintos contextos. Se juegan centenares de partidos en un solo día, y llevar la cuenta los resultados y puntajes es una tarea engorrosa. Además, mantener la comunicación entre todos los equipos conlleva un caos comunicacional y logístico.

Como respuesta a esta problemática, en 2019 se desarrolló una plataforma en Django que apoyara esta gestión, llevando registro de las personas, equipos, instituciones, partidos y puntajes, y con un sistema de mensajería básica. Sin embargo, dicho desarrollo no logró finalizarse satisfactoriamente y no pudo ser usado. Luego, con el estallido social de 2019 y la pandemia de 2020, el proyecto quedó abandonado.

El objetivo de este trabajo de título fue retomar esta plataforma y conseguir un producto mínimo viable en base a ella. Debido a la falta de documentación y sustento metodológico en el proyecto, para poder retomarlo fue necesaria una exploración completa del sistema, junto con una reconstrucción de las necesidades de los usuarios mediante entrevistas.

Una vez realizado el análisis se decidió llevar a cabo una reestructuración y migración completa del sistema, configurando una API con Django REST Framework y una aplicación de página única con React.

Finalmente, se logró hacer la reestructuración basal propuesta, pero la implementación solo se completó parcialmente. En particular, el frontend no alcanzó un estado utilizable en un contexto realista, y la implementación parcial no logró ser validada externamente por usuarios. Sin embargo, la migración de tecnologías y la reinstauración de una base metodológica del proyecto, permiten que éste sea completado con mayor facilidad a futuro, habiendo lidiado ya con la parte más compleja de la reestructuración. Se considera que los objetivos se cumplieron parcialmente, pero que las partes cumplidas implican una mejora sustancial al sistema previo, no solo en términos técnicos, sino también de metodologías que permiten darle una segunda oportunidad al proyecto.

A mi familia, amigos y todos los que estuvieron ahí para afirmarme en cada una de las caídas todos estos años.

Agradecimientos

En primer lugar, agradezco con todo mi corazón a mis padres, que durante la larga duración de mi carrera y mi vida estuvieron siempre preocupados de darme todas la posibilidades para cumplir mis objetivos, con un sacrificio y dedicación que nunca voy a poder dejar de agradecer. A mis hermanos, que me aguantaron y apoyaron directa e indirectamente con su complicidad y buenos momentos.

A la larga lista de amigues que hice en la universidad. Desde mis primeros amigues de plan común, hasta mis amigues jóvenes de la pandemia. Los Frens, el Culto, Ofisalitos. Por las experiencias vividas, los carretes, el estudio y los chismes, que hicieron que mi carrera fueran los mejores años de mi vida.

Al Tomi, que sin su apoyo incondicional esta memoria no hubiera podido ser escrita. Preocupándose de que cumpliera con mis responsabilidades y dándome el apoyo emocional en los momentos más difíciles. Tu compañía y cariño me dieron el empujón que necesitaba para esta última etapa y sin ti no lo hubiera logrado.

A los CaDCC 2017, 2018, 2019 y cuántos otros más. La experiencia de haber trabajado con ustedes fue lo más valioso en términos profesionales y de relaciones interpersonales que pude haber tenido, mucho más que los ramos formales.

A mi profesora guía Jocelyn, que me apoyó semana a semana y creyó en mí para este trabajo y varias otras instancias en las que hemos podido trabajar juntos.

A toda la comunidad del DCC. Fueron varios años y varios recambios que me tocó ver, entre chats, la Salita, los paseos y todo lo demás, definitivamente han sido la mejor familia universitaria que pude haber deseado.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Índice de Fragmentos de Código

Capítulo 1

Introducción

1.1. Contexto

Los Juegos Deportivos de Ingeniería (JING) son un evento deportivo inter-institucional iniciado por primera vez en 2016 a manos de la Escuela de Ingeniería de la Pontificia Universidad Católica [1]. Cada año, una universidad distinta es sede de un torneo que reúne estudiantes de diversas facultades de ingeniería en Chile para competir en variadas disciplinas deportivas y recreativas.

Este evento deportivo, similar a otros eventos de la misma índole, se rige bajo cierta estructura organizativa con una complejidad logística bastante específica. Por ejemplo, en su edición de 2019, participaron 8 universidades chilenas, de Santiago, Viña del Mar y Valparaíso, en 30 disciplinas deportivas y juegos. Cada disciplina contó con varios partidos llevados a cabo en distintos sectores del Campus San Joaquín de la Universidad Católica, y se dividieron en tres categorías de actividad, cada una con un sistema de puntajes distinto. Finalmente, con la suma de puntos por equipo en cada partido, se determinó a la universidad ganadora [2].

La logística detrás de un evento de esta magnitud conlleva mantener un registro actualizado de los distintos equipos e instituciones participantes; las sedes y locaciones; los horarios, resultados y asistencia de un partido; y el flujo general del torneo y sus puntajes. También es importante mantener un flujo comunicativo entre cada una de las partes, teniendo que notificar a cada participante las informaciones, programas y reglamentos que le competan, además de posibles eventualidades que puedan presentarse.

Cabe mencionar que en el contexto de la pandemia por COVID-19, los JING se vieron obligados a adaptarse en su edición 2020 a un formato completamente en línea, cambiando las disciplinas deportivas físicas por juegos virtuales, yendo desde un estilo casual hasta juegos considerados e-sports profesionales. Este cambio de esquema trajo consigo un cambio en la logística organizativa, reafirmando la necesidad de herramientas especializadas y flexibles para asistir en la organización de forma colaborativa y en línea. Para el año 2021, a falta de suficientes centros deportivos estudiantiles constituidos, no se realizaron los JING, y este año 2022, se anunció una nueva edición en formato presencial para el mes de octubre.

Sumado a los conflictos organizacionales, se agrega que estos eventos son llevados a cabo por centros de estudiantes, que deben compatibilizar esta logística compleja con su vida académica, y distribuyen la responsabilidad entre varias personas administradoras, usualmente poco especializadas en gestiones de este estilo, que rotan constantemente cada año.

Actualmente toda la organización de los torneos se lleva a cabo en planillas de Excel o Google Sheets personalizadas, cuya metodología rota junto a la institución organizadora. El registro de participantes se realiza por Google Forms y la forma de contactarles no está estandarizada y, dependiendo de los datos que se tengan y de la persona que busque el contacto, suele ser a través de WhatsApp u otros sistemas de mensajería instantánea.

En base a todos estos problemas, durante el segundo semestre de 2019 los estudiantes del Departamento de Ciencias de la Computación Pablo Miranda y Javier Zambra realizaron un trabajo dirigido con la profesora Jocelyn Simmonds, guía de esta memoria, con el fin de plantear e implementar un primer acercamiento a una solución tecnológica para este asunto. Los estudiantes propusieron y desarrollaron una plataforma web que permite llevar un registro de las principales variables envueltas en la organización del torneo. A pesar de haber logrado implementar gran parte de la funcionalidad base, dado el tiempo limitado de un trabajo dirigido, el sistema no alcanzó un punto de producto viable mínimo como para poner en funcionamiento [3].

Se propuso, entonces, en esta memoria, retomar el desarrollo de esta plataforma, usando el trabajo previo como base para acercarse a un producto utilizable que permita asistir en la organización de eventos deportivos estudiantiles como los JING. Ello implica restaurar el proyecto anterior desde el estado en que se dejó, completar la implementación de funcionalidades inconclusas, re-evaluar los requisitos y eventuales mejoras y re-diseñar la interfaz de usuario a un nivel más especializado.

1.2. Situación Actual

En esta sección se describirá de modo introductorio el estado del software existente, considerando la capa de datos, la interfaz gráfica y la funcionalidad de la plataforma. Un análisis más detallado de los problemas en cada nivel se dará en el Capítulo 3.

El sistema de Miranda y Zambra está construido sobre el framework Django, tanto por el lado del servidor, como en la interfaz, utilizando el sistema de templates del mismo [4], con el framework de estilo Material Design for Bootstrap 4 (MDB) [5] para el diseño visual de las componentes.

El modelo de datos cuenta con una estructura base de Personas, Universidades, Locaciones, Eventos, Deportes, Partidos, Equipos, Noticias y Mensajes, que a grandes rasgos parecieran ser suficientes para la arquitectura requerida, pero que requerirán una reestructuración para solucionar algunos problemas mencionados más adelante.

La base de datos se encuentra configurada para utilizar el almacenamiento en la nube de Google Cloud Storage, pero según la profesora a cargo el acceso a dicha base de datos es

incierto, por lo que podría ser necesario re-configurarla. Para efectos de explorar la aplicación, se pretende establecer una base de datos local.

La interfaz utiliza los estilos del framework MDB 4. Los elementos principales de navegación son dos barras superiores jerárquicas. Una sirve para navegar entre las secciones mayores de la página y la otra para moverse entre subsecciones en aquellas que las tengan. Las vistas son mayoritariamente tablas que listan la data necesaria, con la posibilidad de buscar y ordenar (Figura 1.1). El ingreso de datos es principalmente mediante formularios genéricos y sin mayores validaciones, generando errores al ingresar mal la información. Los errores no tienen un manejo adecuado y se muestran al usuario de forma críptica, en modo de debugging de Django. El diseño a modo general sigue los estándares de Material Design que entrega MDB.

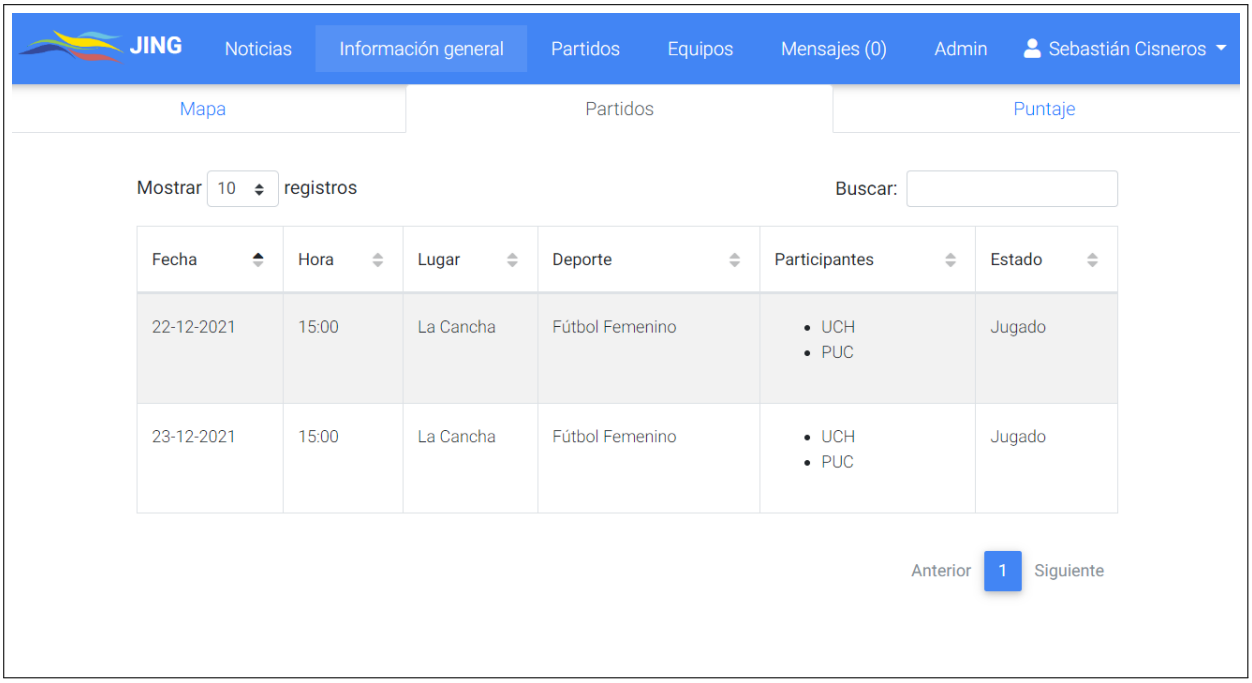

Figura 1.1: Vista de resumen de partidos.

Con estas características, la plataforma es capaz de almacenar, gestionar y listar la mayoría de los datos necesarios, solo habiendo algunos que fallan por configuración de la base de datos. Dado esto, se cree que el núcleo de la funcionalidad base se encuentra mayoritariamente implementado.

El proyecto falla al momento de tener múltiples eventos deportivos, pues la data automáticamente se relaciona al último evento creado, sin posibilidad de elegir libremente entre eventos. También, como ya se mencionó, existen varias validaciones y errores que no son correctamente capturados e informados por el sistema. Se pensaba que el sitio debería poder cargar data desde un archivo Excel o CSV, pero dicha funcionalidad no se encuentra en el proyecto. Por otro lado, aunque en el código hay referencias a un sistema de generación de códigos QR para controlar asistencia, no es posible llegar a esta función en el estado actual del proyecto, aunque sí existe un sistema básico de validación mediante códigos de 6 dígitos.

El código se encuentra en un repositorio público de Github [3], con aproximadamente 3.000 líneas de código en Python y 2.800 en HTML, repartidas en cerca de 40 views y 20 templates, y no cuenta con ningún tipo de documentación ni interna ni externa al código.

1.3. Estado del Arte

Existen diversas opciones de plataformas en línea que abordan el problema de organizar torneos deportivos. En su mayoría consisten en solo una herramienta sencilla para establecer la distribución de los partidos y equipos dentro de una configuración estándar de eliminación o puntajes, para un solo evento. Por sobre esas herramientas existen plataformas más complejas que permiten confeccionar eventos en el marco de un sistema multi-competencia, con registro de equipos, jugadores y progresos. Dentro de estos últimos se destacan cuatro aplicaciones: DoLeague, Tournify, Playinga y Competize. Éstas fueron descubiertas utilizando motores de búsqueda y rankings de plataformas de organización deportiva, y seleccionadas para esta revisión en base a su similitud con el producto objetivo.

Entre las aplicaciones seleccionadas se presentan objetivos y funcionamientos similares, con diferencias en la ejecución y capacidades. A continuación se describen brevemente sus beneficios y desventajas en el contexto abordado en esta memoria, con particular enfoque en la flexibilidad, usabilidad, funcionalidades ofrecidas y precios.

- DoLeague [6] es una plataforma de origen español que ofrece el servicio de gestión de torneos. En su descripción destacan la creación de un sitio web personalizado para el campeonato, con herramientas para su gestión en línea como, por ejemplo, gestión de partidos y resultados, visualización de disponibilidad de canchas, registro de inscripciones con sistema de pago y distintos tipos de configuraciones estándar de partidos, con múltiples deportes.

  El servicio se presta en un modelo freemium, con un nivel básico sin costo y dos niveles de suscripción de pago mensual para añadir más funcionalidades. Los niveles no afectan la cantidad de campeonatos, equipos ni jugadores, solamente agregan algunas comodidades de gestión, personalización y sistema de pagos que no se alinean necesariamente con las necesidades trabajadas en este proyecto.

  La interfaz, nativa en español, presenta una disposición poco estructurada y sobresaturada, con iconografías poco explicativas, exceso de lenguaje técnico sin explicaciones y enlaces de ayuda rotos.

  A pesar de que el sistema ofrece bastante flexibilidad en la estructuración de los campeonatos, partidos y registros de resultados, no es posible confeccionar un esquema como el utilizado tradicionalmente en los JING y otros eventos inter-institucionales, donde cada institución presenta sub-equipos que compiten en distintas disciplinas y divisiones obteniendo puntajes entre cada tipo de competencia que se suman a un pozo común institucional.

- Tournify [7] es un sistema de organización de torneos deportivos que se centra en la sencillez de la gestión. Ofrece una planificación de torneos estándar con fase de grupos

y eliminatorias; calendarización de partidos, fechas y locaciones; gestión de equipos y participantes; y registro de puntajes. No ofrece un sistema extenso de registro de jugadores ni notificaciones. Su versión gratuita permite hasta 8 equipos. Las versiones pagadas se avalúan por cada torneo, habiendo opciones de US$35 y US$100 por torneo, que aumentan el límite de equipos a 60 y sin límites, respectivamente.

La interfaz del sistema sigue un diseño moderno y más minimalista, basado en Material Design. Al tener menos opciones de personalización y funcionalidades más reducidas a lo nuclear, el look-and-feel general es más limpio y poco cargado. Se apoya en el uso de tarjetas y elementos arrastrables, haciéndola simple de usar desde los primeros usos. El sistema está originalmente en inglés, con traducción al español.

Haciendo uso de esta interfaz de tarjetas, el sistema permite construir formatos personalizados de competencia, permitiendo mezclar fases de grupos, eliminación y partidos singulares. Sin embargo, la configuración de los equipos y su participación tampoco permite confeccionar un esquema como el utilizado en los JING y eventos similares, y las opciones de formato de competencias es reducida y no cubre la diversidad de configuraciones necesarias.

- Playinga [8] es una plataforma todo-en-uno para una gestión completa de torneos. Ofrece un servicio complejo con funcionalidades como un sistema de registro, postulaciones y pagos; perfiles de usuarios y de equipos; notificaciones, anuncios y mensajes; gestión de locaciones, recintos y horarios; generación de calendarios y grupos; gestión de auspiciadores; entre otras. El servicio es completamente gratuito.

  La complejidad del servicio necesariamente implica una interfaz compleja, sin embargo, se encuentra bien estructurada y jerarquizada, utilizando distintos tipos de menú adecuados para cada sección. Utiliza iconografía personalizada y etiquetas/mensajes concisos e informativos. Durante la prueba del sitio se encontraron varios bugs críticos de interfaz, en particular en la gestión de equipos, que rompían algunos flujos y generaban data corrupta o incorrecta. El sitio está inglés con opción de español, pero la traducción es bastante precaria y con varios errores por traducciones directas mal contextualizadas.

  Con la flexibilidad ofrecida, sí podría ser posible configurar un campeonato por puntajes como el requerido, sin embargo, los deportes ofrecidos son limitados y consideran solo deportes físicos, no e-sports o videojuegos. Al ser altamente automatizada la progresión del torneo, las reglas de cada deporte limitan la configuración de los partidos, y el avance entre cada fase del torneo imposibilita la edición o corrección de fases anteriores. La alta dependencia en la automatización de la plataforma reduce considerablemente la flexibilidad de la misma.

- Competize [9] es una plataforma española que se describe como de gestión profesional para fútbol amateur. A pesar de estar inicialmente enfocada en fútbol, está generalizada para otros deportes. Ofrece servicios similares a los de DoLeague para la organización de eventos.

  Cada división deportiva, e.g. “Fútbol Varones”, se configura como una *competición*. En el sistema de precios de Competize, la versión gratuita admite solamente 1 competición activa. Los niveles de pago son de 9 € y 39 €, con 5 y 10 competiciones activas,

respectivamente, y la posibilidad de cotizar una cantidad mayor. Esta limitación conflictúa significativamente con las condiciones esperadas, que suelen ser varias decenas de competiciones.

En cuanto a usabilidad, el diseño visual en general parece bien estructurado, pero la navegación es bastante confusa, habiendo varias formas de llegar a las mismas vistas y mezclando distintos niveles jerárquicos de navegación en un mismo menú. No hay un flujo de uso claro. Algunas interacciones funcionan erráticamente, lo que hace más confuso el uso del sistema. El sitio está nativamente en español.

De forma similar a las otras plataformas, Competize no permite construir un esquema de campeonato de las características que requiere la organización de un evento como el estudiado en este proyecto, limitándose a estructuras predefinidas que no posibilitan gestionar un sistema de divisiones acumulativas y de condiciones flexibles.

Como conclusión de la revisión del estado del arte de plataformas similares al propósito de esta memoria, no se ha logrado encontrar software que satisfaga a cabalidad las implicancias del problema. El principal obstáculo recae en la confección de un esquema de campeonato que considere las condiciones específicas de los eventos inter-universitarios, como lo es la acumulación de puntajes por universidad o institución.

Otro problema común es la sobre-especialización de las plataformas existentes, con funcionalidades que no atañen al objetivo propuesto y que complejizan innecesariamente la interacción con la plataforma, como lo son los sistemas de postulaciones y pagos, gestión de recintos y personalización.

Por último, aunque las plataformas sí tienen flexibilidad en la esquematización de los torneos, ésta siempre es dentro de un conjunto preestablecido de reglamentos, tipos de torneo, deportes y competencias. Se dificulta o imposibilita la opción de replantear formatos, como por ejemplo, hacer enfrentamientos de más de 2 equipos simultáneos. Además, al enfocarse en automatizar lo más posible la progresión de los eventos, se deja poco espacio a la gestión manual de los partidos y progresiones, y son estrictas al impedir modificaciones o correcciones, con tal de resguardar la integridad de los flujos; por ejemplo, la posibilidad de cambiar los integrantes de un equipo una vez iniciado un evento. Ninguna plataforma explorada presenta la capacidad de edición y flexibilidad que exige un evento amateur organizado por estudiantes no-dedicados, que constantemente presenta inconvenientes, correcciones, cambios en los equipos e innovación en los formatos de competencia.

1.4. Objetivos

1.4.1. Objetivo General

Completar y complementar la plataforma existente de Miranda y Zambra, con el fin de generar un sistema funcional que permita organizar, gestionar y controlar eventos deportivos con múltiples organizadores, sedes y disciplinas, en un contexto estudiantil universitario.

1.4.2. Objetivos Específicos

1. Estabilizar y ejecutar nuevamente la plataforma actual, teniendo que explorar su funcionamiento sin una documentación clara, además de reparar y completar el sistema para alcanzar la versión mínima viable propuesta en el desarrollo inicial.

2. Determinar y replantear funcionalidades y requisitos esperados del sistema, contrastando las necesidades de los usuarios con el diagnóstico de los desarrollos existentes.

3. Re-diseñar y re-implementar la interfaz de usuario para entregar mejores estándares de usabilidad y robustez, que permitan utilizar la herramienta de manera autónoma y sin mayores capacitaciones.

4. Desplegar la plataforma en un sitio web para que pueda ser utilizada públicamente.

1.5. Solución propuesta

La solución que se plantea preliminarmente es continuar con la propuesta establecida en el primer desarrollo, complementando el trabajo para concluir un producto finalizado y utilizable. Las siguientes funcionalidades se presentan en abstracto como requisitos esperables de la plataforma, para luego ser definidas concretamente en el Capítulo 3, luego de la recopilación de información y análisis de la misma.

El producto esperado debe poder utilizarse para organizar distintos eventos deportivos independientes, de manera simultánea y colaborativa entre varios/as usuarios/as. Cada evento debe tener asociadas instituciones participantes, con sus respectivos equipos de personas. También se deben albergar las distintas sedes y locaciones en que se llevarán a cabo los partidos, asociadas a instituciones.

Una jerarquía de usuarios determinará los permisos y acciones disponibles dentro del sitio. Como mínimo, se espera tener usuarios básicos o jugadores, que podrán asignarse a los distintos equipos y partidos; coordinadores, que gestionarán partidos o disciplinas específicas; organizadores que manejen los usuarios y gestión de un evento deportivo completo; y administradores que gestionen el sitio y la creación de nuevas cuentas.

El sistema será capaz de llevar registro de los resultados de cada partido individual y calcular automáticamente los puntajes y tablas de posiciones con cada institución participante, considerando los resultados de sus miembros representantes en el marco de un sistema de puntajes diferenciados para distintas categorías de disciplinas. Los coordinadores podrán indicar cuándo un partido se llevó a cabo, e ingresar el resultado final.

La esquematización de los partidos y tipos de competición deberá ser manual y no estar sujeta a un conjunto de reglamentos y esquemas de torneo preestablecidos. El enfoque del sistema no estará en la automatización del progreso del campeonato, sino en el registro de enfrentamientos jugados y futuros. Se debe permitir a la organización poder construir de manera flexible y corregible la progresión del evento, considerando las variaciones implicadas en un evento estudiantil amateur.

Existirá un control de asistencia a los partidos, donde se podrá registrar rápidamente qué jugadores participan de la actividad. Esto funcionará con un sistema de códigos QR que cada jugador podrá utilizar para registrar su propia asistencia.

Un sistema de mensajería básico deberá permitir comunicarse diferenciadamente con personas, agrupaciones, equipos y/o instituciones completas, para poder informarles de horarios, novedades, cambios en el programa y otras informaciones dirigidas que se puedan requerir. Al mismo tiempo habrá una sección de noticias generales disponible para cualquier usuario.

Se diagnosticará el estado actual en cuanto a tecnologías y metodologías usadas, con el fin de realizar posibles mejoras y optimizaciones, no solo en términos de eficiencia sino en mantenibilidad y escalabilidad.

La interfaz del sistema debe ser responsiva y usable, entendiendo la responsividad como la adaptabilidad de la interfaz a distintos tamaños de pantallas y la usabilidad como la capacidad de un producto de ser usado de forma fácil, eficiente, eficaz. Esto considerando que podría ser usada en tiempo real y en terreno durante las competencias, por lo que el ingreso de información debiera ser lo más intuitivo posible. Dada la naturaleza competitiva de las actividades y la sensibilidad en el registro de resultados y puntajes, es prioritario que el sitio sea poco propenso a errores de ingreso, a la vez que sea fácilmente editable para corregirlos.

El sistema deberá quedar disponible para su uso en un entorno real. Se buscará la posibilidad de hospedar el sitio en alguna de las redes oficiales de la Universidad o el Departamento, y se establecerá un protocolo de creación de usuarios administradores que puedan crear nuevos eventos y gestionar el sitio, esto considerando que la naturaleza efímera de las instituciones organizadoras implica que se espera un alto recambio de administración y sería poco sostenible hospedar el sitio en algún servicio externo de pago.

1.6. Metodología a seguir

En esta sección se describe la planificación general utilizada en el desarrollo del proyecto y su estado de realización. Las principales tareas a completar que se plantearon al inicio del proyecto fueron las siguientes:

Exploración del sistema:

1. **Análisis técnico del sistema actual.** Se requiere revisar en profundidad las capacidades, falencias y condiciones del software original. En base a este análisis se decidirá el nivel de recuperabilidad de la plataforma, para determinar qué partes son reutilizables en esta iteración.

2. **Reevaluación de los requerimientos.** Dada la falta de documentación respecto al planteamiento de la plataforma actual, se requerirá volver a explorar las necesidades de usuarios finales para enfocar los requerimientos del proyecto. Esto se llevará a cabo mediante entrevistas. Con esta información y con el análisis técnico, se concretizarán

las decisiones en pos de la solución propuesta, definiendo funcionalidades y flujos de interacción.

3. **Rediseño de la plataforma.** Una vez establecidos los requerimientos del sistema se diseñará una nueva interfaz, que considere una mejor usabilidad y robustez que la actual. Se considerará la migración de tecnologías y frameworks utilizados en el frontend.

Implementación:

4. **Gestión del campeonato.** Completar la funcionalidad nuclear relativa a la confección de un sistema de campeonato con varios partidos y disciplinas, con registro de puntajes por institución.
5. **Sistema de mensajería.** Optimizar y mejorar el sistema de mensajes existente para permitir una mejor personalización de los destinatarios, para comunicar novedades por jugador/a, equipo e institución.
6. **Despliegue de la plataforma.** Disponibilizar la plataforma para su uso público.

Evaluación:

7. **Testing.** Simular la ejecución completa de un campeonato de la escala esperada. Se espera utilizar información real de alguna versión reciente de los JING.
8. **Validación con usuarios.** Aplicar metodologías de evaluación de usabilidad con usuarios reales.
9. **Correcciones.** Aplicar los ajustes necesarios luego de las primeras validaciones y, de ser necesario, volver a validar.

Inicialmente se planteó un proceso iterativo-incremental con metodologías ágiles, con al menos dos instancias de validación intermedia entre cada progreso relevante de la implementación. Sin embargo, tanto la exploración como la implementación tomaron más tiempo de lo estipulado, y finalmente se concretó una sola iteración larga de implementación.

Como efecto de esto, al término del proyecto no se logró completar el desarrollo del sistema de mensajería o una navegación adecuada, así como tampoco fue posible realizar una validación con usuarios reales. Una discusión más profunda de los efectos de este cambio en la planificación se dará en las secciones 5.4 *Discusión* y 6.4 *Reflexiones finales*.

1.7. Estructura del documento

En el Capítulo 2 de este informe, se describen las tecnologías y conceptos necesarios para comprender los desarrollos descritos en los capítulos siguientes.

Posteriormente, en el Capítulo 3 se hablará del análisis técnico de la situación original, se presentará la información recopilada mediante entrevistas con organizadoras de los eventos, y se finalizará concluyendo los cambios necesarios a realizar en la plataforma y los bosquejos diseñados, considerando los diagnósticos hechos.

El Capítulo 4 describe la implementación técnica de los cambios y diseños propuestos. Se detallarán los tres ejes principales de los cambios: El modelo de datos, la API del backend y la interfaz en el frontend. Finalmente, se hablará del despliegue del sistema en un servidor de acceso público.

El Capítulo 5 tratará las metodologías seguidas para realizar la validación interna del sistema, y se presentará la planificación de la validación externa inconclusa. Luego se discutirán las consecuencias e información obtenida gracias a estas técnicas.

Finalmente, en el Capítulo 6, se hará una revisión generalizada del trabajo realizado, contrastando con los objetivos planteados. Luego se darán ideas y lineamientos generales para trabajos futuros que puedan mejorar el sistema, para cerrar el informe con reflexiones y aprendizajes personales del memorista.

El informe cuenta con un Anexo A, que contiene una transcripción de las entrevistas tratadas en el análisis del problema.

Dados los estándares del área de conocimiento relativa a este trabajo de título, este informe tiene una alta recurrencia de anglicismos y otros conceptos técnicos de difícil traducción directa. Estos conceptos se escribirán generalmente en texto regular por simplicidad y legibilidad, y en cursiva solamente al presentarlos y describirlos por primera vez.

Capítulo 2

Marco teórico

En este capítulo se cubrirán los conocimientos básicos sobre las tecnologías utilizadas en este trabajo de título. Siendo éste un proyecto de desarrollo web, inicialmente se explicarán de forma superficial los principales conceptos estructurales de un sistema cliente-servidor, en la Sección 2.1.

Posteriormente, se describirán las tecnologías específicas utilizadas en conjunción con dicha estructura. En la Sección 2.2 se explicará el funcionamiento del framework Django como arquitectura de servidor, y la abstracción obtenida sobre él al utilizar Django REST Framework para establecer una API de consultas.

Por el lado del cliente, se explicará el funcionamiento básico del framework React, la estructura de componentes que se utilizará en este trabajo y la forma en que se relaciona con el servidor. En términos de la estilización visual del sitio, se hablará del framework Bootstrap, que entrega una capa predefinida y estandarizada de estilos estructurales y estéticos.

2.1. Arquitectura frontend-backend y requests

En ingeniería de software y desarrollo web, existe una arquitectura estándar de separación de responsabilidades que consiste en dividir el sistema en una capa presentacional, conocida como cliente o *frontend*, y una capa de acceso a datos conocida como servidor o *backend*. Para el resto de este informe, se preferirá el uso de los anglicismos frontend y backend, actualmente estandarizados en la industria.

En el frontend ocurren las interacciones directas con el usuario y se lleva a cabo todo el procesamiento *client-side*, es decir, que ocurre en la máquina cliente. Las capas más típicas del frontend corresponden a los lenguajes HTML, CSS y Javascript, que definen la estructura semántica, los estilos gráficos visuales y la capacidad funcional del sitio web, respectivamente.

Por el lado del backend se encuentra cualquier lenguaje de programación que permita procesar una consulta recibida desde el frontend, hacer algún tipo de procesamiento u obtención de información, y entregar una respuesta apropiada. Entre estos se encuentran típicamente

PHP, Python, Ruby, Perl, Java, C#, entre otros.

La comunicación entre el frontend y el backend se realiza a través de consultas o *requests* HTTP. El funcionamiento del protocolo HTTP queda fuera del alcance de esta explicación, pero básicamente una request HTTP permite entregar datos hacia un servidor y esperar una respuesta apropiada de vuelta. Estos datos contienen meta-datos, o *headers*, que pueden informar al servidor sobre las condiciones de la consulta, y un *payload* o *body*, que contiene la data enviada per se.

Algo interesante es que el cliente en un principio no tiene ninguna información más allá de la dirección a la que quiere consultar. Antes de la consulta inicial a un sitio, de cierta forma no existe un frontend como tal, y es el backend quien se encarga de construir los recursos del frontend y entregarle lo necesario para mostrarse al usuario.

En esa línea, se tienen *frameworks*, concepto para definir un conjunto de herramientas que sirven como base estructural para cumplir cierta función. que se encargan no solo de procesar información y consultar una base de datos, sino también de usar estos datos para generar una página web que el frontend pueda mostrar, enviándole el HTML, CSS y Javascript necesarios.

Un ejemplo de estos frameworks de backend es Django, que se construye sobre Python, y posee su propio sistema de plantillas o *templates* mediante el cual genera sitios web con data incrustada y los envía como respuesta.

Dado que el servidor envía la página ya construida, la generación de la página es dinámica, pero una vez que se envió, ésta ya es técnicamente estática. El cliente no puede acceder a más data hasta que solicite otra página completa. Las tecnologías actuales llevan más al extremo este dinamismo, y en lugar de generar dinámicamente páginas y enviarlas estáticamente al cliente, lo que hacen es enviar al cliente todo el código Javascript necesario para construirlas por sí mismo en el frontend. De esta forma, luego de enviar ese primer código, el cliente puede construir todo el sitio sin más consultas, y la comunicación con el servidor puede empezar a ser netamente sobre datos en lugar de páginas completas.

Dentro de los frameworks que hacen esta función de pre-generar el frontend completo, se encuentra React, uno de los más populares actualmente.

Estas dos últimas tecnologías mencionadas, Django y React son la base de la implementación de este proyecto, y serán explicadas con más detalle en las siguientes secciones.

2.2. Django y Django REST Framework

Django es un framework de backend escrito en Python, cuya principal función es facilitar la creación de sitios web con bases de datos, con énfasis en la reusabilidad, conectividad y extensibilidad de sus componentes siguiendo el modelo MVT o *Model-View-Template*. Para entender este patrón arquitectural se explican sus cuatro componentes principales:

- **Models**, o Modelos: Son una representación en Python de una base de datos. Haciendo la analogía, una clase es una tabla, las propiedades de la clase son columnas de la tabla,

y cada instancia de esa clase es una fila de la tabla. Al utilizar modelos de Django, en general no será necesario escribir consultas SQL manuales, ya que se implementan bajo un ORM, *Object-Relational Mapping*, o mapeo objeto-relacional [10]. Esto significa que actúan como una capa entre el programador y la base de datos, con el fin de poder hacer consultas típicas usando lenguaje Python y automatizando su optimización.

- **Templates**, o Plantillas: Son la parte visual de la aplicación. A través de éstas, Django puede construir respuestas y páginas en HTML, CSS y Javascript para enviar al cliente, pudiendo adoptar de cierta forma las funciones de un frontend. Estos templates implementan una sintaxis extendida de HTML, en la que pueden incrustarse los resultados obtenidos del procesamiento de datos usando decoradores especiales, de la forma `{% variable %}`. Sin embargo, las respuestas no necesariamente tienen que ser páginas HTML, y pueden ser cualquier tipo de data.

- **Views**, o Vistas: Contrario a lo que su nombre indica, las views de Django no son la parte visual, sino la funcional. Actúan como controlador intermediario encargado de recibir las requests, procesar su payload, consultar a la base de datos si es necesario, y juntar los resultados en una respuesta usando los templates como capa presentacional. Una view siempre recibirá una request HTTP y entregará una respuesta HTTP.

- **URLs**, o direcciones/rutas: La dirección URL es el identificador al que el cliente enviará una request. El enrutador de Django definirá qué view es la que debe responder a la solicitud en función de a qué URL se haya enviado. Esta dirección puede ser dinámica, por ejemplo, al solicitar la URL dinámica:

  `https://www.sitioejemplo.cl/cursos/2022/CC6909/integrantes/?nombre=A`

  el servidor sabría que tiene que invocar la view asociada a `/cursos/`, y podría recoger la información de la ruta como parámetros y pasárselos como argumento, de esta forma el cliente podría solicitar dinámicamente que la view obtenga de la base de datos todos los integrantes del curso CC6909 del año 2022, cuyo nombre empiece con A.

El flujo en el que interactúan estos conceptos en un ciclo común de consulta es el siguiente (ver la Figura 2.1):

1. Usuario ingresa a una **URL** en su navegador. Se envía una request HTTP.
2. Django recibe la request, y en base a la URL decide qué **view** deberá responderla.
3. La view procesa la request y realiza alguna operación funcional con los datos.
4. De ser necesario, la view puede consultar la base de datos a través de los **modelos**.
5. Una vez obtenido lo necesario para responder, la view utiliza un **template** para construir la respuesta con los datos.
6. Se envía la página construida como respuesta al usuario.

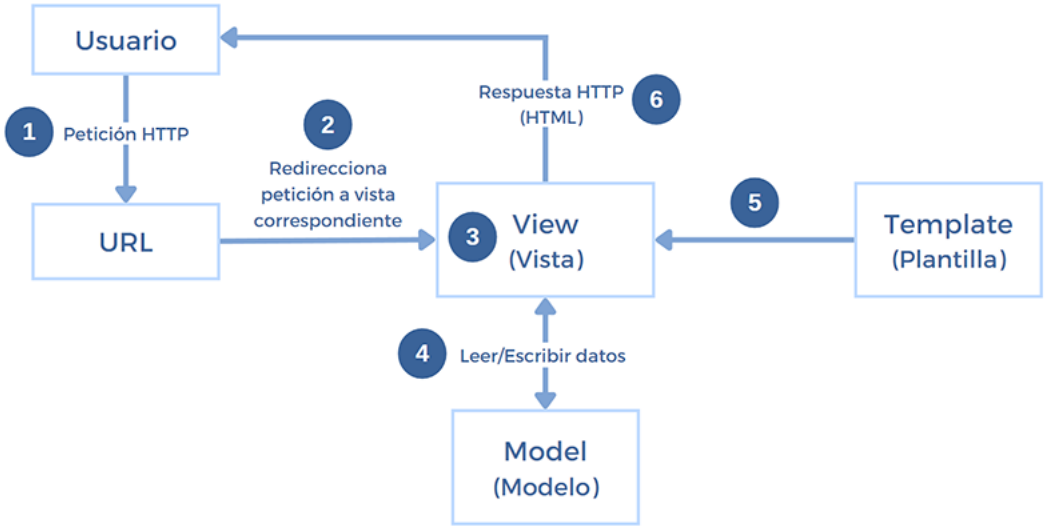

Figura 2.1: Arquitectura Model-View-Template utilizada por Django.

Habiendo descrito este flujo, se menciona que para este proyecto no se usarán los templates de Django, pues el frontend estará servido mediante React, descrito más adelante en este capítulo. En conjunción con ese frontend, se requiere que Django envíe respuestas solamente con datos en lugar de páginas ya construidas. Para esto se utilizará Django REST Framework. Las razones de este cambio se detallarán es el Capítulo 3, específicamente en la Subsección 3.3.1, al describir los cambios propuestos al sistema.

Una arquitectura de estilo REST, de ***RE****presentational* ***S****tate* ***T****ransfer* [11], establece estándares en la comunicación entre los sistemas computacionales y la web. Un sistema que sigue esta arquitectura se dice que es *RESTful*, y se caracteriza, principalmente, por las siguientes restricciones:

- Interfaz uniforme, basada en la identificación de recursos a través de representaciones. Por ejemplo, un cliente solicita al servidor una lista de `id` de usuario, y luego puede hacer solicitudes respecto de un usuario específico indicando su id al servidor.
- Sin estados. El servidor no guarda estados ni utiliza información de requests anteriores. Cada request debe incluir toda la información necesaria para procesar la solicitud.
- Separación cliente-servidor. El cliente no debe preocuparse por el almacenamiento o procesamiento de datos, mientras que el servidor no debe preocuparse por el estado de la interfaz en el frontend.
- Caché. Las respuestas del servidor deben informar si deben guardarse en una memoria caché, y por cuanto tiempo.

Una API, ***A****pplication* ***P****rogramming* ***I****nterface* o interfaz de programación de aplicaciones [12], se refiere a las funciones y procedimientos ofrecidos por un sistema para ser utilizados como capa de abstracción hacia otro sistema.

Juntando ambos conceptos, una API RESTful se refiere a la interfaz de consulta mediante la cual un sistema permite que se comuniquen desde el exterior, y que sigue los lineamientos descritos. Para este proyecto, se configurará una API RESTful que permita comunicar el frontend con el backend, enviando solamente paquetes atómicos de data. Las URLs dispuestas por la API para recibir requests se conocen como *endpoints*.

Django REST Framework [13] es una aplicación de Django que extiende su funcionalidad para implementar una API con las características mencionadas. Incluye bastante código reutilizable y se enfoca en facilitar la implementación de consultas típicas.

El conjunto de consultas más común, son las 4 operaciones básicas de un almacenamiento persistente, conocidas como CRUD. **C**reate-**R**ead-**U**pdate-**D**elete o crear-leer-editar-borrar [14]. Django REST pre-implementa estas operaciones y permite conectarlas directamente con los modelos de Django. Esto se realiza a través de *ViewSets*, o conjuntos de views, que son clases que implementan los métodos estándar para las 4 operaciones.

Django REST añade otra capa a la lógica de Django, los *serializers* o serializadores. Dado que se planteó que en esta API estaremos enviando solamente datos, y que estos deben seguir ciertos lineamientos de uniformidad y representación, se hace necesaria una forma de representar cada modelo como un diccionario de objetos, listas y valores primitivos, para poder ser enviados claramente al cliente. Este proceso se conoce como *serialización* de la data. Inversamente, es necesario saber cómo convertir de vuelta la información serializada que nos llegue desde el cliente al valor interno de un modelo de Django, proceso llamado *deserialización*. Es en estos procesos que se utilizan los serializadores de Django REST. Estos actúan como intermediarios entre la información guardada en Python y el lenguaje humano legible de comunicación con el cliente.

2.3. React

React [15] es un framework de JavaScript que permite construir sitios webs dinámicos mediante el uso de componentes simples. La ventaja de estos componentes es que cada una maneja su propio estado y lógica interna, permitiendo una arquitectura más reutilizable y desacoplada.

Una de las principales funciones de React es que no solo permite refrescar y actualizar componentes de forma individual, sin recargar el sitio completo, sino que también lo hace de forma optimizada, calculando qué componentes son afectadas cada vez que hay un cambio en el estado del sistema, y renderizando solo esas nuevamente. Esto permite crear páginas del tipo *Single-Page App*, o aplicaciones de página única, en la que el frontend carga la página una sola vez, pero tiene toda la información para poder generar el sitio completo de forma local, pudiendo alterar y actualizar cada componente del sitio de manera independiente, en tiempo real y sin consultar al servidor. Para el usuario, la experiencia de navegación mejora sustancialmente, al tener transiciones inmediatas entre diferentes secciones del sitio, y manteniendo estáticos los elementos que no necesitan ser recargados, aportando a la sensación de inmediatez.

Por supuesto, si bien el sitio generado tiene la información para construir las componentes a voluntad, no tiene acceso a la base de datos del backend. Para conjugar esta navegación inmediata mencionada con las consultas al backend, se utilizan requests asíncronas, que solicitan información al servidor paralelamente mientras el sitio sigue funcionando. Cuando el servidor responde, la data obtenida puede inyectarse en las componentes y cambiar sus estados, generando el re-renderizado o re-dibujado optimizado que se mencionó anteriormente. Así, el usuario prácticamente nunca deja ni recarga el sitio, solo navega a través de componentes que, de forma independiente, pueden consultar y mostrar información de los datos guardados en el servidor, sin frenar la experiencia del usuario.

2.4. Bootstrap y react-bootstrap

Volviendo a la capa simple de HTML y CSS, sin funcionalidades de Javascript, estos dos elementos constituyen la base presentacional de cada página. HTML define **qué** hay en el sitio, en un sentido semántico, y CSS define **cómo** se presenta esta semántica visualmente. Para ello, ambos interactúan a través de *clases*, que son básicamente etiquetas con las que se marcan los elementos HTML para que CSS pueda saber cómo representarlos.

La sigla CSS significa ***C****ascading* ***S****tyle* ***S****heet*, u hoja de estilos en cascada, y su nombre viene de que sus propiedades se van aplicando secuencial y jerárquicamente. Un mismo elemento puede tener varias reglas aplicadas, que se irán sobreescribiendo a medida que se vuelven más específicas. Esta jerarquía descendiente da pie a la analogía de cascada.

Una consecuencia de este comportamiento en cascada es que se puede aplicar una capa de estilo sin perjuicio de las que puedan seguir aplicándose por encima, permitiendo tener estilos estandarizados en el panorama general, a la vez que se admite una personalización flexible en el detalle. Esto significa que pueden construirse grandes conjuntos predefinidos de estilos que se ubiquen en la parte superior de la cascada para definir una base, sobre los que luego se puedan seguir aplicando estilos propios. Estos conjuntos masivos estandarizados se conocen como frameworks de CSS o de estilos, y uno de los más populares es Bootstrap [16].

Boostrap contiene una librería de varios cientos de clases CSS predefinidas y construidas coherentemente, con el fin de establecer un sistema de diseño basal y que siga ciertos estándares. Para poder usar estas clases, basta con aplicarlas en algún elemento HTML apropiado, igual que al escribir CSS propio. Con estas clases, Bootstrap permite un diseño adaptable a distintos tamaños de pantalla (i.e. *responsivo*), un sistema de grilla que posiciona fácilmente los elementos, una estética característica y algunas funcionalidades de interfaz usando Javascript. Se puede apreciar en la Figura 2.2 un ejemplo de un sitio web construido utilizando solamente los estilos ofrecidos por Bootstrap. Se destaca del ejemplo la barra de navegación sombreada, las tarjetas *Free*, *Pro* y *Enterprise* con una estructura coherente, los botones estilizados y la estandarización de las fuentes y tamaños de textos.

Como se mencionó anteriormente, React funciona a base de componentes reutilizables. Si bien es posible aplicar las clases de Bootstrap directamente en las componentes de React, esto rompe su paradigma declarativo. Para solucionar esto, existe una librería que implementa las clases y estilos de React como componentes funcionales declarativas, llamada *react-bootstrap*.

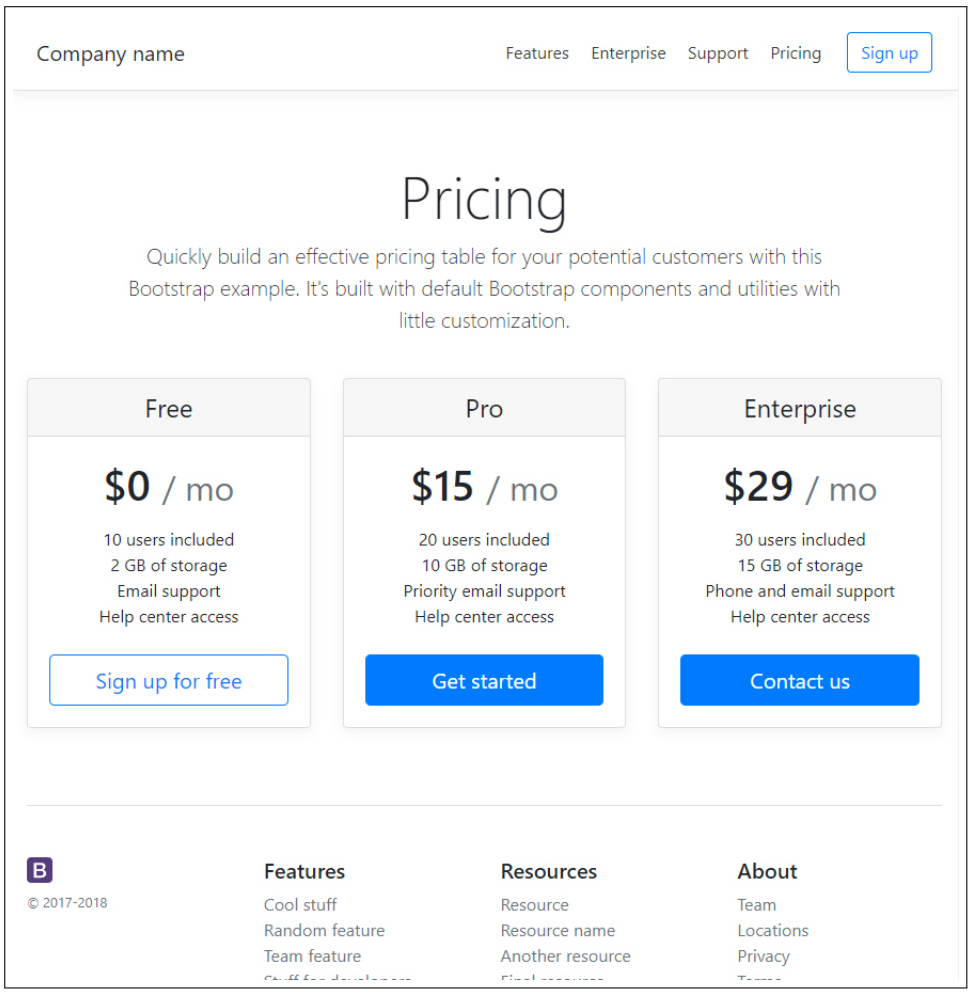

Figura 2.2: Ejemplo de sitio web de prueba estilizado usando Bootstrap.

Luego, estas componentes de react-bootstrap pueden usarse y componerse de la misma forma que cualquier otra componente, mediante props y anidamiento.

Capítulo 3

Análisis y diseño

En este capítulo se describirán los procesos de diagnóstico, investigación, proposición y diseño, realizados en base al software e información disponible al inicio del proyecto y en línea con los objetivos propuestos para esta memoria.

Inicialmente se detallará el proceso de revisión y diagnóstico de la plataforma actual de Miranda y Zambra, desglosando a nivel de código, capa de datos, funcionalidad disponible e interfaces gráficas. Se explicarán los problemas cruciales de cada ámbito para posteriormente determinar pisos mínimos en la etapa de recuperación del sistema.

En la siguiente subsección se expondrá el proceso de confección y ejecución de entrevistas realizadas a usuarias con conocimiento del campo de aplicación. Estas entrevistas tenían el objetivo de reconstruir y levantar los requisitos que llevaron a elaborar la primera iteración de la plataforma, para documentar y establecer una base metodológica que encauzara la reimplementación del proyecto.

Una vez constituida esta base metodológica, se plantearán los cambios concretos a realizar en la etapa de implementación. Se listará un análisis estructuralmente similar a la subsección de diagnóstico, proponiendo respuestas tangibles a los problemas detectados en ella.

Finalmente, se exhibirán bosquejos y propuestas para la interfaz gráfica del sistema, y se explicará el proceso y razonamiento detrás de su confección.

3.1. Diagnóstico del software actual

Se hizo un análisis exhaustivo de cada sección del código de Miranda y Zambra, junto con un testeo manual de calidad de la plataforma y su interfaz, con el fin de establecer cuáles son los arreglos mínimos necesarios para la consolidación del MVP en concordancia con el objetivo específico número 1. Además, esta exploración inicial sentará las bases para determinar complejidades y aterrizar los límites y dificultades del rediseño y reestructuración propuestos como segundo objetivo.

3.1.1. Código

Inicialmente, al intentar instalar el servidor utilizando las librerías declaradas en el archivo `requirements.txt`, se incurrieron en largos tiempos de espera, varios problemas de compatibilidad y otros errores. Inspeccionando manualmente los requerimientos y el código, se notó que se incluyen varias librerías complejas que no se utilizan en ninguna parte del software, en particular, `numpy` y `pandas`. Quitando éstas de los requerimientos, se logró instalar el resto satisfactoriamente.

El código está organizado siguiendo la estructura estándar Model-View-Template de Django. Las distintas secciones semánticas de información están divididas cada una en su propia *app* de Django, siendo estas las divisiones:

- `Administration`. Módulo en que se implementa el panel de administración, que centraliza la gestión (creación, modificación y borrado) de usuarios, deportes, eventos, lugares y organizaciones, haciendo el enlace con sus respectivos módulos.
- `Event`. Módulo de gestión de eventos, entendiendo *eventos* como campeonatos y torneos, como por ejemplo "JING 2021", "La Mona 2020", etc.
- `Location`. Módulo de gestión de lugares (e.g. "Cancha multiuso", "Piscina subterráneo").
- `Match`. Módulo de gestión de partidos. Se encarga de crear partidos y asociarlos a equipos, además de registrar el estado del partido (pendiente, en proceso, jugado), sus resultados y comentarios por equipo.
- `Message`. Módulo de mensajería. Registra los remitentes y destinatarios de un mensaje, su contenido y estado de lectura. Maneja la distribución de mensajes en distintas jerarquías, ya sea universidades, equipos, deportes o personas individuales.
- `News`. Módulo de noticias. Gestiona los anuncios generales asociados a un evento
- `Person`. Módulo de usuarios. Gestiona la información de personas individuales, logins, registros y validación de asistencias.
- `Sport`. Módulo de deportes. Se entiende como "deporte" lo que en lenguaje deportivo sería una división, es decir, una subcategoría de una disciplina deportiva. Por ejemplo: "Fútbol Damas", "Rocket League 4v4", "Tenis de mesa - Dobles". Gestiona el estado de un deporte, si éste está finalizado o en proceso.
- `Team`. Módulo de equipos. Maneja las vistas de gestión de equipos y sus miembros.
- `University`. Módulo de organizaciones. Gestiona las distintas universidades, institutos u organizaciones participantes, y su cualidad de organizadores.

Esta estructuración parece bastante sencilla e intuitiva, y aporta a separar el código en secciones manejables. Solamente la aplicación `Administration` pareciera estar sobrecargada, al intermediar en la gestión de varias de las otras aplicaciones.

Fuera de las estructuras funcionales, se encuentra la carpeta `templates`, que alberga todos los archivos HTML, que serían el frontend de la plataforma. Los templates también se separan en módulos de `Administration`, `Inicio`, `Match`, `Mensajes`, `Noticias`, `Team`. Notar que estos nombres están en español, a diferencia de las apps.

Un problema observado en los templates es la complejidad estructural de los mismos. Siendo un sistema con muchas tablas, formularios y modales, implementado con un framework de estilos con clases atómicas utilitarias, el código rápidamente se vuelve verboso, alcanzando hasta las 911 líneas en uno de los archivos.

Varios de los templates HTML tienen incrustado todo el código de JavaScript. Un inconveniente con esta disposición es que el template se envía completo en cada request al servidor, volviendo a enviar todo el código JavaScript en cada llamado. Si el código estuviera modularizado en archivos separados de los templates, podría optimizarse la carga mediante cachés locales. Se observa en la Figura 3.1 un ejemplo de esto. El código no es directo de entender al no tener funciones con nombre, y el bloque completo abarca cerca de 200 líneas.

En la Tabla 3.1 se puede apreciar una cuantificación de la magnitud del estructura de modelos y vistas de Django usadas en el proyecto, con aproximadamente 3.000 líneas de código, mientras que en la Tabla 3.2 se expone la utilización de lenguajes en el proyecto completo. Se destaca que los archivos HTML son pocos como para la cantidad de líneas totales (146 líneas por archivo, en promedio) y la complejidad y componentes del sitio.

Aplicación	*Modelos*	*Vistas*	*Líneas*
Administration	1	13	942
Event	1	0	34
Location	1	0	37
Match	2	7	478
Message	1	2	170
News	2	2	274
Person	3	8	395
Sport	2	2	216
Team	2	3	357
University	2	0	124
Total	**17**	**37**	**3.027**

Tabla 3.1: Estadísticas de código de aplicación original.

Lenguaje	*Archivos*	*Líneas*
Python	131	3.182
HTML	19	2.767
JavaScript	7	742
Total	**157**	**6.691**

Tabla 3.2: Estadísticas de código sin considerar librerías de estilos.

```
{% block scripts %}
<script src="{% static 'js/jquery-ui.min.js' %}"></script>
<script src="{% static 'js/datepicker-es.js' %}"></script>

<script>
    $(document).ready(function () {
        $.datepicker.setDefaults( $.datepicker.regional[ "es" ] );
        $("#datepicker-create").datepicker();

        $('#played-matches').DataTable({
            "language": {
                "url": "//cdn.datatables.net/plug-ins/1.10.19/i18n/Spanish.json"
            },
            "bInfo": false,
        });
        $('#pending-matches').DataTable({
            "language": {
                "url": "//cdn.datatables.net/plug-ins/1.10.19/i18n/Spanish.json"
            },
            "bInfo": false,
        });

        $('#sport').change(function () {
            let sportId = $(this).val()
            $('option[class^=team-sport-]').hide()
            if (sportId != '') {
                let teamClass = '[class=team-sport-' + sportId + ']'
                $('option' + teamClass).show()
                $('#teams-selector').prop('disabled', false)
            } else {
                $('#teams-selector').prop('disabled', true)
            }
        })

        $('#teams-selector').change(function () {
            let teamId = $(this).val()
            if (teamId != '') {
                let team = $('#teams-selector option:selected').text()
                $('#teams-selector option:selected').hide()
                $(this).val(undefined)

                let btn = $('#team-btn').clone().removeAttr('id').data('team', teamId).prop('hidden',
                    false)

                let input = $('#team-input').clone().attr('id', 'team-' + teamId)

                input.val(teamId)
                btn.append(team)

                $('#team-list').append(input, btn)
                $('[name=sport]').prop('disabled', true)
            }
        })

        $('#team-list').on('click', '.delete-btn .delete-team', function () {
            let btn = $(this).parent()
            let teamId = btn.data('team')
```

Figura 3.1: Código Javascript incrustado en templates.

3.1.2. Base de datos

El modelo de datos se divide análogamente al seccionamiento de apps del item anterior, teniendo cada aplicación una entidad base, y en algunos casos tablas intermedias o *through tables* para implementar relaciones entre entidades (ver la Figura 3.2). Por ejemplo, UniversityEvent, que implementa una relación muchos-a-muchos entre University y Event, con un campo `is_host` que indica si esa universidad es la organizadora del evento.

La base de datos en PostgreSQL se encontraba ligada a Google Cloud SQL. Dicha conexión actualmente no se encuentra disponible y esto impedía que el servidor pudiera correr correctamente. Para poder iniciar el servidor y seguir con el diagnóstico, fue necesario reconfigurar la base de datos para utilizar una local en SQLite3, por simplicidad. De forma similar, el almacenaje multimedia estaba conectado a un *bucket* o depósito de Google Cloud Storage que tampoco está disponible, por lo que se reconfiguró a una ruta en disco local.

Dentro de problemas particulares encontrados, se detecta que la entidad Person está directamente relacionada con Event, siendo esta última una llave foránea de la primera. Esto implica que cada vez que haya un evento nuevo y se agreguen sus participantes, una persona que participó en un evento anterior tendrá que volver a registrarse con un usuario distinto para asociarse a este nuevo evento, repitiendo su información personal y perdiendo la posibilidad de enlazar sus participaciones. Se asume que la razón de incluir Event como foráneo es para darle un rol a la persona, específico para ese evento. Sin embargo, una tabla intermedia podría resolver ambas situaciones.

Otro problema importante respecto al acoplamiento generado sobre un evento particular se da en el registro de los puntajes de cada equipo y su universidad correspondiente. El modelo Team tiene un campo `event_score` y el modelo University tiene un campo `overall_score`. Cada vez que se cierra un deporte se obtiene el valor asociado a la posición del equipo en ese deporte y se guarda directamente al `event_score` de ese equipo. Inmediatamente después, ese valor se suma al `overall_score` de la universidad del equipo y se actualiza en la base de datos.

Esto último implica que cada instancia de una universidad está ligada directamente a un evento específico por contener el puntaje asociado a él. Si una misma universidad participa en dos eventos distintos, deberá crearse una instancia nueva para cada uno. Además, al ir sumando los puntajes con cada cierre y tener dos sumatorias distintas para equipos y universidad en lugar de calcular dinámicamente el total en base a una información única, se podría llegar a inconsistencias entre los puntajes si en algún momento se cambia el puntaje de un equipo o el valor asociado al lugar obtenido, y no se actualiza el puntaje a nivel de la universidad.

La asignación de puntajes es poco flexible en el modelo FinalSportPoints. Tiene un campo `sport_type` que solo admite tres tipos (A, B y C), en referencia a las categorías usadas en los JING, y cada uno de estos tipos define el puntaje correspondiente a cada uno de las posiciones en la tabla. Por ejemplo, una instancia de FinalSportPoints podría establecer la regla de que quien obtenga el 3er lugar en un deporte de tipo A obtendrá 100 puntos. Esta entidad no está relacionada a un Event, por lo que las reglas se establecen para todo el sistema, no permitiendo hacer un sistema de puntos diferenciado para cada evento.

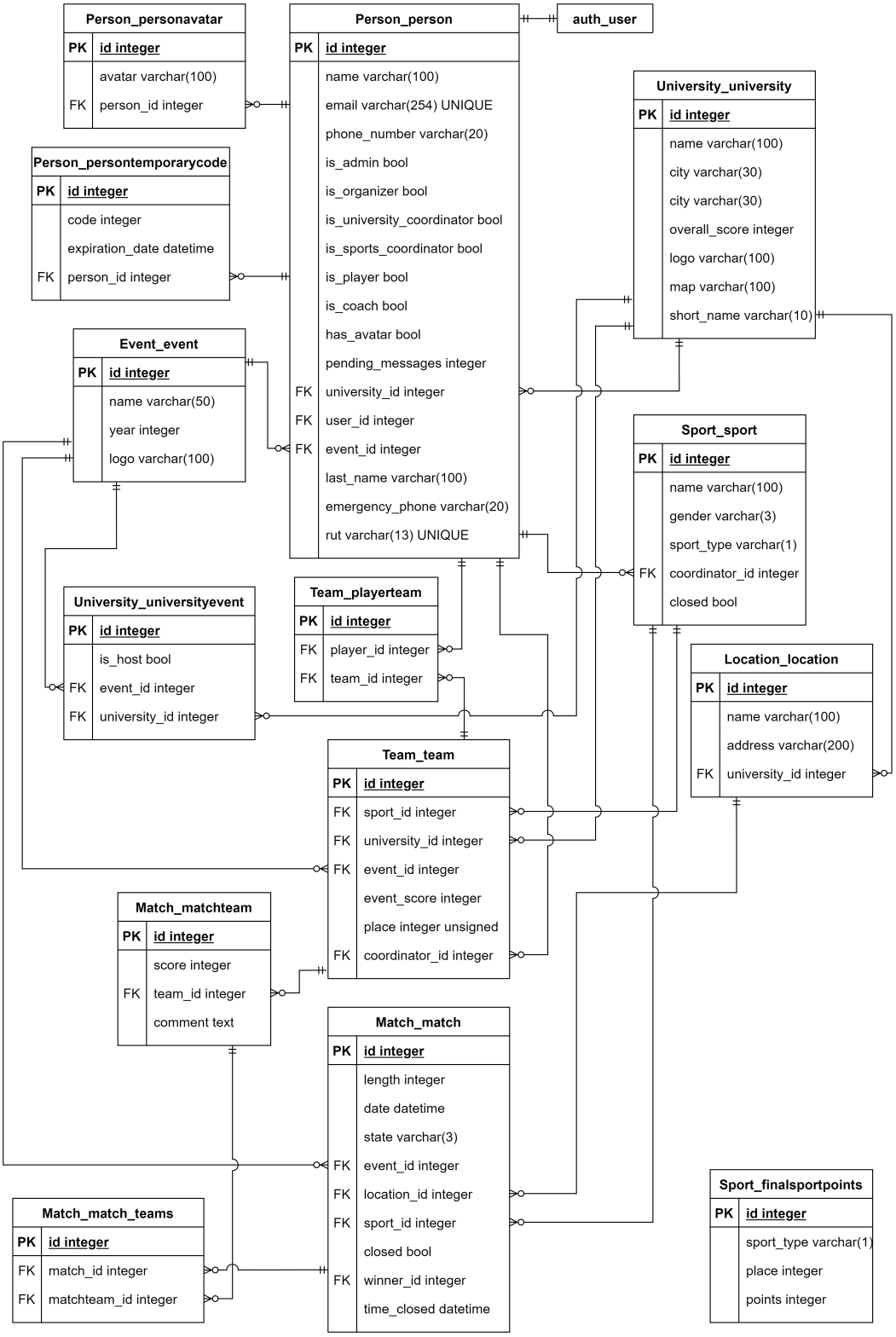

Figura 3.2: Modelo de datos relativo a la estructura del campeonato.

La entidad NewsCategory almacena mucha data en texto libre que después se inserta directamente a su estilización con CSS. Por ejemplo, su campo `btn_class` espera se ingrese directamente la clase de Bootstrap que define el tipo de botón que se renderizará. Esta dato

debería ser más semántico y que luego el frontend se encargue de entregar las clases CSS apropiadas.

3.1.3. Funcionalidad

Utilizando el administrador de Django para generar la data, se logró probar satisfactoriamente las acciones básicas en la gestión de un campeonato. El software actual contempla la creación de usuarios, equipos, organizaciones y las relaciones entre éstos; la gestión de distintos deportes, partidos y lugares; el registro de resultados y puntajes; y la conclusión de éstos últimos en una sumatoria institucional con su respectivo posicionamiento en la tabla.

Un inconveniente en el sistema de puntajes es que éste no permite reabrir una competencia o deporte una vez finalizado. No hay una vista de Django implementada que deshaga los efectos de declarar un partido o un deporte cerrados, haciendo imposible modificar resultados sin causar inconsistencias. Además, los puntajes se suman directamente a un atributo de cada equipo o institución al finalizar un partido. Esto se convierte en un problema de mantenibilidad ya que se debe procurar manualmente que cada vez que haya cierres o rectificaciones de partidos, se sumen o resten los puntos apropiados, y dificulta la posibilidad de declarar puntajes manualmente.

El sistema de mensajería no funciona. La selección de destinatarios funciona bien, pero al enviar la request se intenta enviar el mensaje y se arroja una excepción en el servidor por un elemento `Null`, sin dar feedback alguno para el usuario. El problema se logró ubicar en la consulta que se hace a la base de datos para identificar los usuarios destinatarios de los mensajes, obteniendo una respuesta vacía cada vez.

El sistema de registro de asistencia mediante QR se encuentra aparentemente incompleto, hallándose algunas implementaciones para generar un código en texto, convertirlo a imagen QR, y asociarlo a una persona, pero todo está en trozos atómicos de código, y al intentar forzar su ejecución se presentan diversos errores. Dado que tampoco hay una conexión directa con la interfaz, se asume que esta característica no logró finalizarse y quedó en un estado inconcluso.

3.1.4. Interfaz

A pesar de que la funcionalidad base de campeonato está casi completa, varios elementos de interfaz están faltantes o con fallas que dificultan o impiden llevar a cabo la gestión completa de un evento.

El problema más extendido es que en todo el sitio hay varios formularios y acciones sin ningún tipo de feedback ante errores. Por ejemplo, al crear un usuario con un correo que ya existe, se recarga la página y se consumen los datos, pero no ocurre nada, no se crea el usuario. Otro ejemplo con el mismo efecto ocurre al intentar crear un partido con menos de 2 equipos participantes. Se ejecuta la acción pero no se obtienen ni resultados ni mensajes de error, además de perder toda la información ingresada (ver Figura 3.3).

En línea con lo anterior, varios campos de formularios tienen textos de ayuda que aparecen en inglés, como el mensaje “right” en verde cuando un campo es correcto, como se observa en la Figura 3.4.

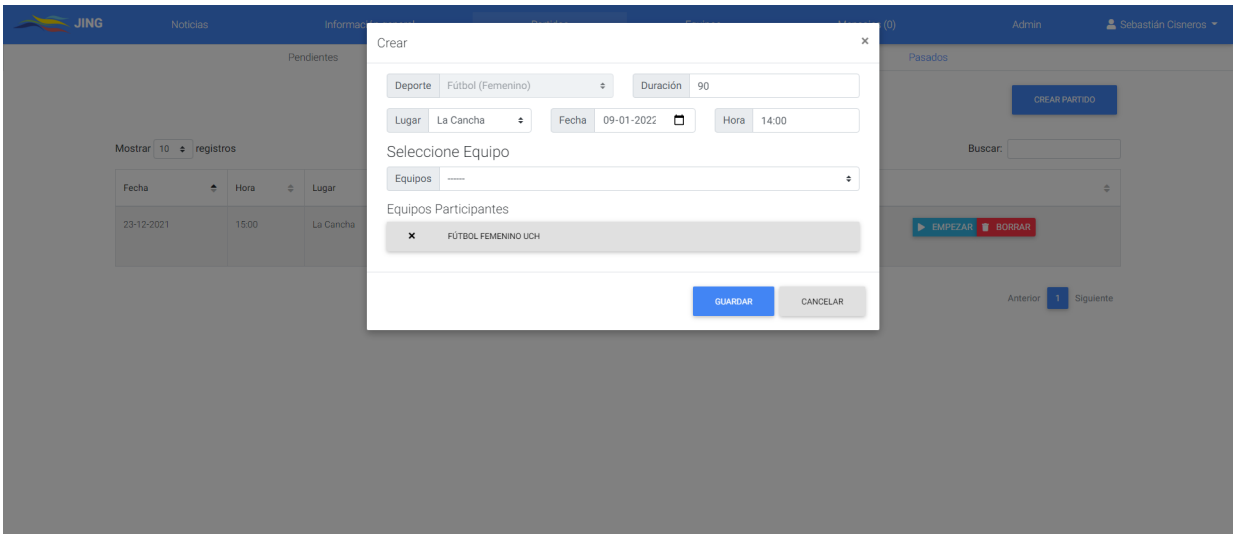

Figura 3.3: Creación de partido. Si se envía con solo un equipo, la página se recargará sin ningún efecto ni mensaje.

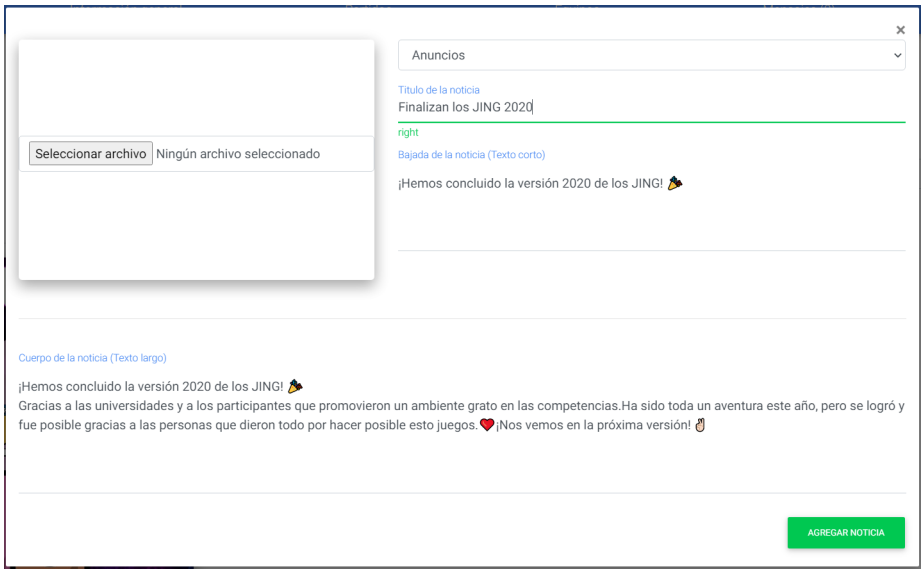

Figura 3.4: Creación de noticia. Notar el mensaje “right” en verde.

Algunas acciones, como el declarar como cerrada una competencia, hacen llamados asíncronos para traer información del servidor. Ninguna de estas consultas muestra un indicador de carga ni ninguna referencia de que se esté esperando una respuesta. Si la conexión se demora o falla, no ocurre nada y el usuario no tiene cómo saber en qué estado está el sitio.

Al crear usuarios se puede asignar un rol, pero no es posible asignar el rol “Deportista”, necesario para asignar al usuario en un equipo y que pueda participar de deportes. Esto solo

es posible desde el administrador de Django. En la Figura 3.5 se observa que la selección de roles no incluye este rol.

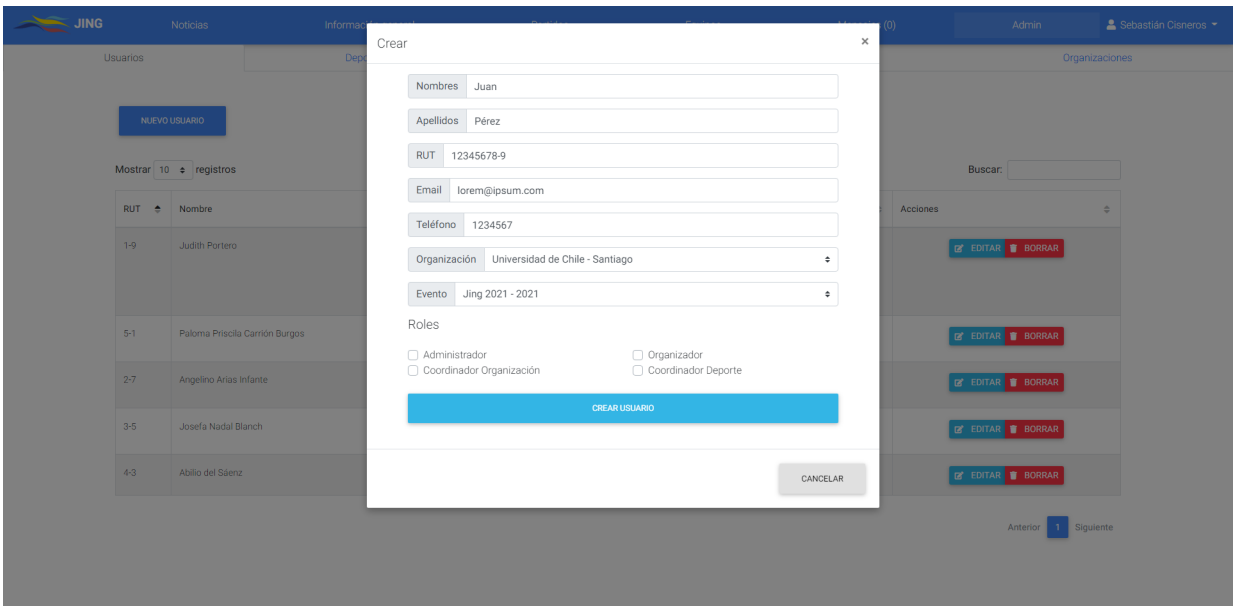

Figura 3.5: Creación de usuario. No es posible dar el rol *Deportista*.

Existe un menú de creación de eventos nuevos, pero el botón para finalizar el formulario no produce ningún efecto. Revisando el código se observa que este botón no está implementado funcionalmente, y no existen las vistas de Django para gestionar eventos.

3.2. Entrevistas a organizadoras

Además del análisis exploratorio del software, se realizaron entrevistas con dos personas que han participado como organizadoras de instancias pasadas de los JING y del torneo interdepartamental de la Facultad de Cs. Físicas y Matemáticas, *La Mona*.

3.2.1. Confección de la entrevista

Para la confección de la entrevista se siguieron los métodos investigativos recomendados por Pernice [17] y Nielsen [18]. En esa línea, se planteó un objetivo específico de la entrevista, que encauzará el resto de su confección.

Objetivo de la entrevista: Conocer cómo las personas organizadoras de los eventos gestionan los partidos y jugadores, qué procesos perciben como engorrosos o complejos, y cómo la gestión se ve afectada al estar en un contexto estudiantil y amateur.

La entrevista tenía dos enfoques. Primero, obtener información concreta sobre el proceso, la organización del campeonato, sus características y definiciones estructurales. En segundo

lugar, la entrevista tenía un enfoque perceptual y logístico, mas no conductual ni observacional, a diferencia de un estudio de usabilidad. No se esperaba analizar las preferencias de las usuarias en cuanto a qué productos o características necesitaban, o que explicaran en detalle su interacción pasada con otras plataformas, sino conocer cuáles son los procesos humanos involucrados, sus contextos emocionales, percepciones y dificultades.

En base a los lineamientos enunciados y con acercamiento a la Técnica del Incidente Crítico, se elaboró un conjunto de preguntas para guiar la conversación, listadas en la Tabla 3.3. Esta técnica utilizada, llamada en inglés Critical Incident Technique (CIT) [19], es un método de investigación que se enfoca en consultar ocurrencias específicas que hayan tenido especial impacto, positivo o negativo, en el resultado de una interacción. Se observa que varias de las preguntas buscan evocar recuerdos situacionales más que factuales o detallados.

Aún teniendo un conjunto de preguntas preestablecidas, al ser una situación conversacional se admitía posible que en el momento surgieran preguntas nuevas o que variaran en cantidad, orden o formulación.

¿Qué tipo de competencias o deportes organizan? ¿Cuál es la estructura del campeonato? ¿Cuántos equipos participan en una misma competencia?
¿Cómo es el sistema de puntajes que se utiliza?
¿Cuál es el proceso de postulación o registro de jugadores?
¿Se toma asistencia en los partidos? ¿Se verifica la identidad de los/as jugadores/as?
¿Cómo coordinan el registro del progreso del campeonato? ¿Qué herramientas han utilizado para apoyar la gestión?
Cuéntame sobre algún problema importante que hayan tenido por una descoordinación de los partidos o puntajes
¿Han tenido que modificar la estructura de una competencia luego de haberse iniciado?
Cuéntame de alguna ocasión en que haya sido muy importante poder comunicarte rápidamente con un equipo o jugador.
¿Conocían a los organizadores de la versión anterior? ¿Hubo apoyo de su parte para la organización?
¿Cuánto tiempo personal se le dedica a la organización? ¿Fue difícil coordinarlo con los estudios?
¿Cuántas personas gestionan un campeonato?

Tabla 3.3: Preguntas guía para entrevista a personas organizadoras

Se aplicó esta entrevista a dos personas: Fabiola Mariqueo, actual integrante del Centro Deportivo de Ingeniería (CDI), y Sofía Baeza, ex integrante del CDI y ex organizadora de los JING.

Fabiola ha organizado las últimas versiones del torneo La Mona, además de haber integrado parte de la organización de algunos JING, sin intervención directa. Dado esto, su entrevista fue mucho más enfocada a describir la situación de La Mona. Sofía participó como representante de la U. de Chile en la organización de los últimos JING presenciales en 2019, por lo que su entrevista se apegó más a este evento.

Las entrevistas completas fueron grabadas y transcritas (Anexo A), previo permiso de las entrevistadas, y la información recopilada se utilizó, en primer lugar, a modo de dar sentido y rectificar los objetivos de los desarrollos actuales que carecen de documentación y sustento metodológico, y posteriormente como insumo para un rediseño y reencauce de la plataforma.

3.2.2. Discusión de resultados

Lo primero que llama la atención es la marcada diferencia entre la organización de La Mona y los JING. Si bien son estructuralmente similares, en cuanto a ser un torneo multidisciplinario con puntajes acumulados por institución, la magnitud de cada uno difiere notoriamente.

Se resumen en primer lugar la información y conclusiones extraídas particularmente respecto al torneo La Mona:

1. Participan al rededor de 14 departamentos, con un cuerpo organizador de aproximadamente 20 personas en total.
2. Al ser un evento menos competitivo y dedicado, tiende a haber muchos más cambios imprevistos y ajustes in situ, en función de la disponibilidad de los equipos.
3. Aún considerando esta mayor incertidumbre logística, no hay mayores problemas tecnológicos. No ha sido particularmente problemático de llevar flexibilizando el sistema de planillas. Hay pequeños retrasos con los puntajes, pero nunca problemas serios por culpa de la plataforma.
4. La mayor complicación tecnológica es la comunicación. Actualmente a través de la aplicación de mensajería instantánea WhatsApp, el flujo comunicativo se contamina fácilmente para los supervisores que a veces deben estar en 12 grupos distintos simultáneamente, a la vez que utilizan la aplicación para sus vidas personales.
5. Se toma asistencia de los equipos, para tener registro y declarar walkovers[1] pero no por personas individuales.
6. Los puntajes no se diferencian según el tipo de deporte. Todas las competencias asignan el mismo puntaje según la posición.
7. Sí les interesaría un sistema para subir puntajes en tiempo real, para hacerlos públicos más rápida y transparentemente.

En esta entrevista se pudo detectar que en torneos más pequeños como La Mona el tamaño de la organización permite una comunicación más fluida, la envergadura de las competencias es más manejable, y el sentido más recreativo y menos competitivo admite una menor rigurosidad en la configuración de las competencias. Dado esto, se hace menos prioritaria una posible extensión del dominio de la plataforma.

[1]Victoria otorgada a una persona o equipo si no hay más competidores, porque se retiraron, fueron descalificados, o no asistieron.

Ya que este proyecto nace como propuesta específicamente para los JING, se evaluarán preferentemente los requerimientos asociados a ellos, manteniendo las experiencias de La Mona como insumo secundario y como base para considerar una extensión futura, fuera de esta memoria.

Las siguientes son las informaciones recopiladas en relación a los JING o a ambos eventos:

1. La comunicación también es a través de WhatsApp, pero en este caso se tienen cientos de personas. Entre coordinadores de cada deporte para cada universidad, personas de la organización, equipos de comunicaciones, encargados de sala, etc. Fácilmente puede haber 200 personas a nivel de coordinación.

2. Los puntajes se llevaban principalmente a mano, en planillas físicas. Luego cada noche, de madrugada, se consolidaban todos los puntajes en una planilla digital, no exento de discusiones.

3. El papel hacía todo menos transparente, y a veces se perdían, teniendo que llegar a acuerdos entre los equipos recordando los puntajes.

4. No se usaban formularios abiertos de inscripción. Los deportistas se acercaban a sus centros deportivos y ellos los inscribían en su planilla Excel, aunque también se inscribían estudiantes directamente.

5. Hay muchos partidos ocurriendo simultáneamente.

6. Por esto último, se usaban muchas salas y espacios poco conocidos de la sede, no solo canchas o lugares reconocibles. Se repartían mapas físicos.

7. En general, la asistencia de personas se tomaba en los buses, principalmente para asegurarse de que no se perdiera nadie.

8. Es un período bien corto (3 días), donde se mueve mucha gente. Muchos de otras regiones.

9. Dado que los JING son en 3 días de fin de semana, la organización suele dedicarse 100 % a eso durante ese período.

10. Hay mucha competitividad y honor en los JING. No se permiten reemplazos de última hora y en general no hay muchos ajustes, pues se debe respetar la competencia.

11. La aplicación original falló porque, en tiempo real durante la competencia, nunca pudieron autenticarse en la plataforma, por lo que no se pudo ni siquiera llegar a probar. Los desarrolladores ya habían terminado su trabajo dirigido y no estaban disponibles para solucionar fallas.

12. La pandemia promovió mucho los videojuegos y e-sports. Estos suelen tener formatos menos tradicionales, como partidos de 4 personas.

13. A veces los deportes cambian de categoría entre un año y otro. Al menos en experiencia de la entrevistada, los puntajes asignados a cada categoría nunca cambian.

14. La idea era marcar los resultados in situ. Apenas terminara un partido, que se marcara en la plataforma.

Con los puntos del 1 al 6 se confirma la necesidad de esta plataforma, y se ahonda en las razones humanas detrás de los requisitos. Esto se toma como validación metodológica de las funcionalidades planteadas en la plataforma original.

Por otro lado, del punto 7 punto y de la experiencia de La Mona, se desprende que el registro de asistencia no parece ser un requerimiento real del sistema.

De los puntos del 8 al 11 se desprende la importancia de una validación rigurosa y testing profundo antes de intentar usar el servicio en un contexto real. Los JING son una competencia seria, altamente competitiva, y en la que hay viajes y sacrificios al participar y organizar. No puede fallar el sistema de puntajes.

La puesta en producción no puede ser la primera prueba masiva. Dadas las caracterísicas de este trabajo de título, se pone en duda que este proyecto eventualmente pueda usarse para este año 2022.

Finalmente, con los puntos del 12 al 14 se reafirma la necesidad de una plataforma flexible, que no se enfoque en automatizar formatos preestablecidos, sino que sea solo una herramienta de apoyo. La coordinación debe tener el control manual de la competencia.

3.3. Cambios propuestos

Considerando los aspectos explorados en esta etapa diagnóstica, se planificaron, a modo general y sin orden particular, los siguientes cambios estructurales de la plataforma, en línea con el primer objetivo específico del proyecto.

El detalle de la ejecución de estos cambios se desarrollará en el Capítulo 4.

3.3.1. Frameworks y código

El cambio más importante a realizar será migrar el sistema para usar React [15] como framework de frontend, dejando Django como una API REST utilizando Django REST Framework [13].

Esta decisión se sustenta en que la complejidad de la plataforma esperada, que involucra distintos flujos jerárquicos que interactúan entre sí y requieren de elementos atómicos repetibles, se ve enormemente beneficiada por un sistema por componentes, como lo es React.

Otra razón es que un sistema de tipo Single-App Page, junto con las optimizaciones de re-rendering por Virtual DOM que entrega este framework, acelerarían la exploración de este sitio, que necesariamente implica mucha navegación entre secciones y una gran cantidad de micro-acciones al involucrar tantos partidos, jugadores y eventos.

La aplicación ya cuenta con una estructura de URLs y vistas similar a una API REST en la mayoría de sus acciones, facilitando la posibilidad de una migración.

Junto con esta migración, se hará un reinicio completo de las librerías y dependencias, para descartar aquellas que están en el código original, pero nunca se usan. Se actualizará Django desde su versión 2.2 a la 4.0.4, la más reciente al momento de iniciar el proyecto.

Se sustituirá el sistema de estilos de MDB (basado en Bootstrap), por Bootstrap original. Esto porque también se debe migrar el sistema de estilos a una estructura de componentes, y la librería `react-bootstrap` está diseñada para hacer esta migración sencilla. Se reconoce que también existe una versión de componentes oficial de MDB para React, pero tiene la desventaja de que cada componente incluye el prefijo `MDB`, añadiendo mucho ruido visual al código. Además, se consideró que Bootstrap es un sistema más ubicuo y conocido, en particular en contextos estudiantiles, por lo que adoptarlo directamente facilitaría la posibilidad de que sea mantenido en el futuro por estudiantes.

Todos los archivos de Javascript y CSS que no tengan un uso claro se eliminarán, considerando que al renovar las tecnologías es muy probable que todos queden en desuso. Se diseñarán nuevas clases CSS ad hoc, que eviten la necesidad de incluir elementos presentacionales a nivel de los datos, como se explicó en el diagnóstico previo.

Se estandarizará el idioma usado en el proyecto. Para los textos visibles por los usuarios finales se usará el español, por ser el idioma nativo del usuario objetivo. El código en su totalidad se escribirá en inglés, ya que al usar frameworks con estándares, funciones y nombres predefinidos (e.g. *get, set, order_by, Container, NavBar*), y conceptos cuya traducción los haría menos reconocibles (e.g. *queryset, viewset*), programar en español necesariamente implicaría una mezcla entre ambos idiomas poco estandarizable que propendería a reducir la mantenibilidad, por lo que se prefiere evitar.

3.3.2. Base de datos

Se descartará el almacenamiento en la nube de Google Cloud y se utilizará una base de datos local sencilla en SQLite.

Se buscará un desacoplamiento general del modelo Event. Por ejemplo, al desasociar Person de Event permitiría que una misma persona pueda participar de distintos eventos. En su lugar, se crearán tablas relacionales que permitan relaciones muchos-a-muchos entre los eventos y las entidades que lo requieran.

Se reimplementará la forma en que se designan los puntajes según la posición en la competencia. Se buscará que esta designación sea dependiente de cada Event para que cada uno tenga su propio sistema de puntos. En lugar de que cada vez que termine un partido se sumen directamente los puntos al equipo o institución, se mantendrá registro de las victorias, y los puntajes se calcularán consistentemente en función de ellas, permitiendo también sobreescritura manual. Para facilitar esto, los puntajes dejarán de ser atributos directos de cada Team y University, y pasarán a ser tablas relacionales que enlacen un equipo o institución con una posición en la tabla y un puntaje, desacoplando responsabilidades.

Se desacoplarán los roles de cada Person, también convirtiéndolos a tablas relacionales. De esta forma una misma persona podría tener distintos roles en distintos eventos, y se permitiría que más de una persona coordine un equipo, deporte, institución o evento.

3.3.3. Funcionalidades e interfaz

Dado que en la API se implementarán por defecto consultas CRUD para todos los modelos, se espera que esto indirectamente solucione problemas que, en el sistema antiguo, requerirían agregar Views nuevas o repararlas. Entre esos problemas se encuentra el sistema de mensajería y la flexibilidad de reabrir competencias cerradas.

El sistema de validación por QR se encuentra incompleto, con errores complejos, y en las entrevistas se evidenció la baja prioridad de esta funcionalidad, por lo que se decidió descartarla para esta iteración.

Se implementarán filtros ad hoc para cada tabla de datos, en reemplazo del campo de texto único que se encuentra en la plataforma antigua.

Al migrar a React, se espera tener un sistema de alertas por componentes reutilizables, que permita dar retroalimentación al usuario sobre errores en la plataforma y las consultas desde cualquier parte del sitio.

Similar a las alertas, se espera integrar un indicador de carga fácilmente incorporable a cualquier página del sitio, que permita al usuario saber cuándo se están realizando consultas asíncronas. Esto es particularmente importante en una Single-Page App, que implica una navegación constante entre secciones y constantes llamados a la API.

Se implementará un menú de selección de eventos que permita navegar entre distintas iteraciones de los JING. El evento activo se tratará como una especie de variable global para todo el sitio y será visible desde la barra de navegación, para que el usuario siempre sepa qué evento está explorando.

3.4. Diseños de interfaz preliminares

3.4.1. Navegación

Para la navegación del sitio, se mantuvo el concepto original de una barra superior, pero se suprimió la segunda barra superior de navegación jerárquica.

Se espera implementar una navegación más sencilla, que priorice las tareas inmediatas que los distintos tipos de usuario buscan hacer.

En el caso particular de un usuario deportista, éste en general solo tiene accesos de lectura en la plataforma, por lo que se reconocen las siguientes tareas inmediatas, ejemplificadas con casos de uso básicos:

- Ver los partidos. En particular, ver los partidos que le corresponde jugar, o los más próximos para saber a cuáles asistir. Por ejemplo, una usuaria deportista quiere saber a qué hora y en qué lugar se jugará la partida de Ajedrez que le corresponde jugar al día siguiente.

- Ver los resultados. Por ejemplo, un usuario quiere saber qué equipos han ganado las partidas de Mario Kart ya jugadas, para saber cuáles van ganando.

- Ver el mapa del recinto. Por ejemplo, un usuario debe jugar un partido y sabe el nombre del lugar donde se jugará, pero no sabe cómo llegar a él desde la entrada de la sede, por lo que requiere un mapa y/o indicaciones.

- Ver sus mensajes. Por ejemplo, un usuario necesita ver un mensaje enviado por su coordinadora universitaria donde le confirma su participación en la próxima competencia de Baile.

- Ver los últimos anuncios y noticias. Por ejemplo, una usuaria quiere enterarse de los últimos cambios que se han hecho en los horarios y locaciones de partidos, para asegurarse de que sus partidos de interés no han tenido novedades.

La capacidad para ver listados de personas, equipos, deportes e instituciones no se considera como una tarea inmediata necesaria para el usuario deportista común. Para esta etapa inicial del proyecto, se propone priorizar las acciones mencionadas, y luego evaluar el posicionamiento de otras acciones secundarias.

Para el caso de organizadores, en lugar de agregar un botón para cada grupo de entidades (personas, equipos, deportes, instituciones), se propone un solo botón de "Coordinación", que luego permita algún desglose entre las distintas tablas.

Considerando el listado dado, se reconoce que ver mensajes y ver noticias tienen ya una asociación previa relativamente estandarizada en redes sociales y otros sitios, donde la mensajería se encuentra en un ícono en la esquina superior derecha, y la página de inicio con noticias es accesible desde la esquina superior izquierda, desde el logo del sitio o un ícono representativo de una casa.

Con esto en mente, se propone disponer al lado izquierdo la navegación pertinente a la información (noticias, mapas), al lado derecho la navegación pertinente al usuario y mensajes, y al centro los accesos a las tablas de registros (partidos, resultados, coordinación).

Por último, dentro de los cambios propuestos se planteó la necesidad de poder seleccionar el evento activo, y que éste fuera visible constantemente para evitar que se estuviera revisando información antigua. Dado que esta selección se tratará como una variable global en la sesión del usuario, se propone ponerla junto al menú del mismo, a la derecha de la barra.

Teniendo en cuenta estas consideraciones, se bosquejan las barras de navegación mostradas en la Figura 3.6 y la Figura 3.7. La segunda incluye el botón "Coordinación" en la navegación central y será visible solo para usuarios con algún rol de coordinación en deportes, eventos o instituciones.

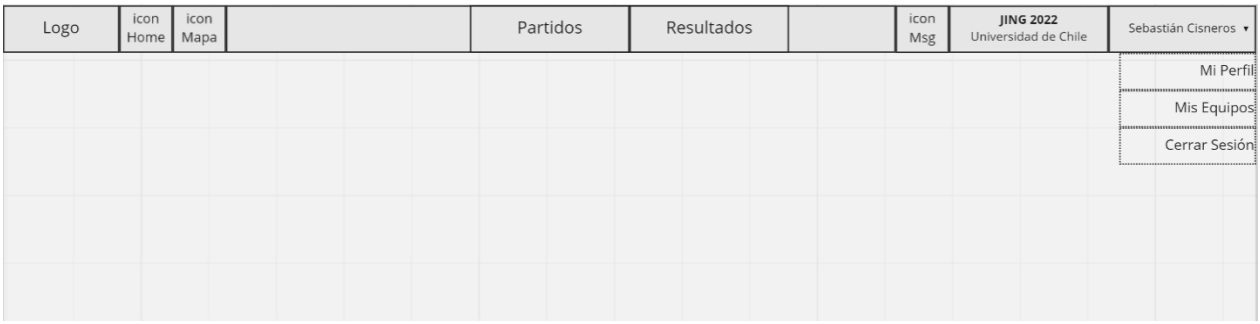

Figura 3.6: Bosquejo de barra de navegación para usuarios deportistas.

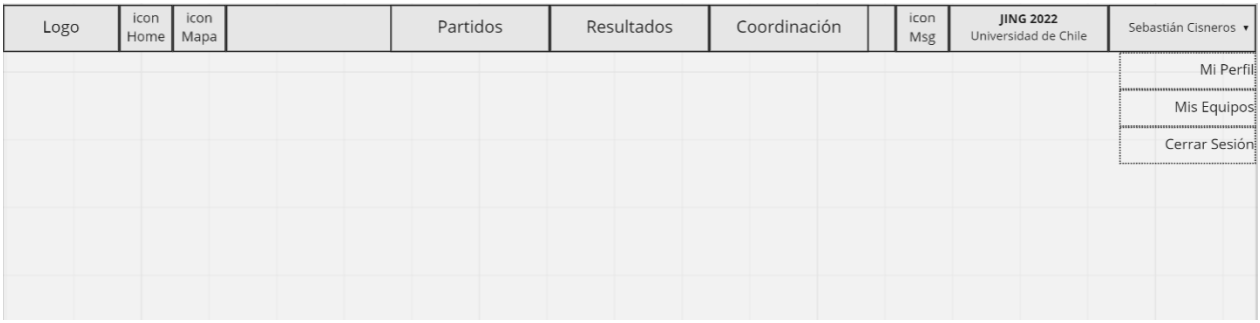

Figura 3.7: Bosquejo de barra de navegación para usuarios con algún rol de coordinación.

Cabe mencionar que en una primera iteración del bosquejo, se pensó implementar un sistema de barra lateral con los distintos deportes y equipos en que participara cada persona, según sus roles, como se observa en la Figura 3.8. Esto es muy similar a la barra lateral de la plataforma U-Cursos [20] (ver Figura 3.9). Este acercamiento a dicha plataforma se consideró inspirándose en su manejo satisfactorio de múltiples instancias de participación en distintos tipos de agrupaciones (e.g. cursos, comunidades, instituciones), con distintos roles en cada una (e.g. estudiantes, profesores, ayudantes) y con repeticiones u ocurrencias a lo largo de distintos períodos de tiempo. En principio, esto es bastante analógico a lo que se buscaba obtener.

Sin embargo, se descartó esta idea al notar que, en general, un usuario común no tendría tantas participaciones simultáneas como para requerir un menú dedicado, y que, a diferencia de U-Cursos, no hay tanto contenido interno en cada deporte como para justificar una separación de ese estilo. Además, una barra lateral de ese tipo quitaría la posibilidad de usar el espacio para otro tipo de contenido, como el sistema de filtros que se mostrará más adelante.

De todas formas, se consideró guardar este concepto para evaluar aplicarlo a usuarios administradores, que por defecto tendrían participación en todos los deportes de todos los eventos.

Figura 3.8: Bosquejo de navegación descartado.

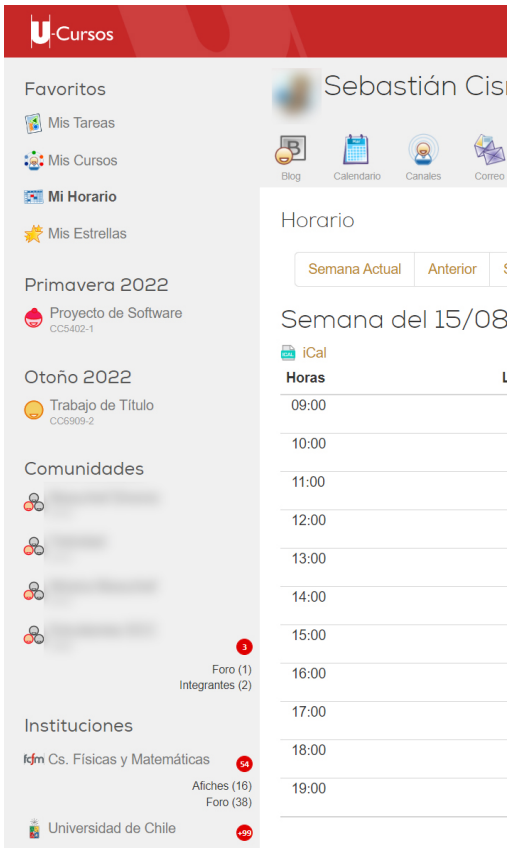

Figura 3.9: Barra lateral de U-Cursos.

3.4.2. Noticias y anuncios

Para la elaboración de esta sección, se revisaron las redes sociales que los JING han usado en los últimos años, Instagram[2] y Facebook[3], para determinar el tipo de contenidos que se solía compartir. Con esta exploración y con lo recopilado en las entrevistas, se inició una pequeña lluvia de ideas respecto a las distintas categorias y características generales de las noticias. en la Figura 3.10 se observa el resultado de este proceso.

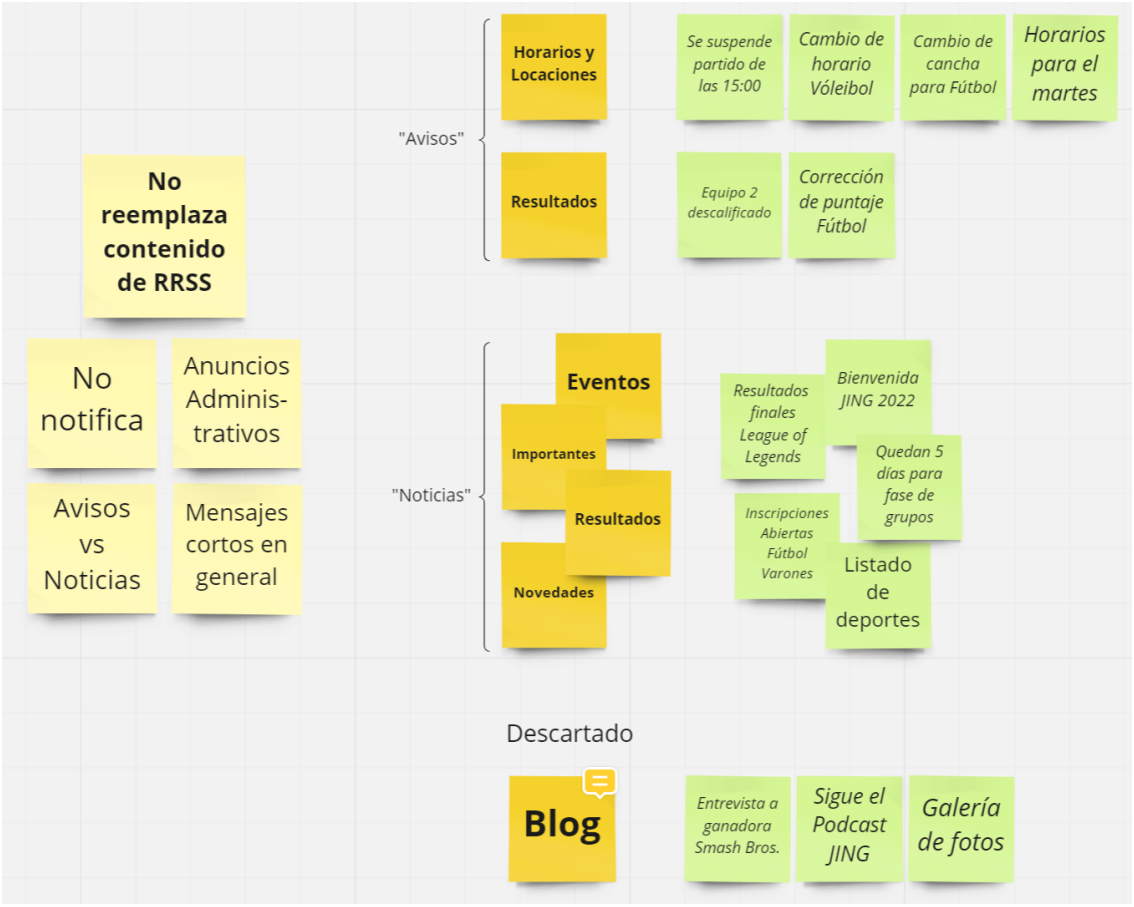

Figura 3.10: Lluvia de ideas sobre página de noticias.

En amarillo, al lado izquierdo, se disponen las características generales del contenido esperado en la plataforma. En resumen, se desprende que esta sección de noticias no debe buscar cumplir el rol de una red social, limitándose a anuncios administrativos cortos.

Al centro, en color mostaza, se encuentra las categorías deducidas del contenido encontrado en redes sociales, y a su derecha, en verde, ejemplos de titulares que calzan en esas categorías. Observando el contenido, se logra diferenciar entre aquel que podría ser determinante en el ámbito competitivo del evento, como lo son cambios de horarios y locaciones. A estos anuncios se les denominó *"Avisos"*. Por otro lado, se encuentran aquellos posts que también entregan información del evento, pero es información no-crítica o de menor urgencia,

[2]https://www.instagram.com/jingchile/
[3]https://www.facebook.com/JuegosDeportivosIngenieria/

que probablemente no tiene impacto directo en un sentido competitivo. A estos contenido se le llamó *"Noticias"*.

Finalmente, en línea con la decisión de no tomar un rol de red social, se considera el contenido que no incluye información del evento en sí, y que es principalmente orientado a generar audiencia o compartir momentos. Este contenido, denominado "Blog", se considera como poco pertinente a la plataforma. Por supuesto, el uso final que se le dé a esta sección será de difícil control, y en ningún caso se busca limitar o restringir contenido. El sentido de este análisis es solo establecer una priorización de contenidos.

Bajo estas consideraciones, se plantea que el sitio de noticias tenga una división clara entre los *Avisos* y las *Noticias*, de forma que ambos estén accesibles, pero que los avisos críticos puedan distinguirse fácilmente entre los no-críticos. De esta forma se llega al bosquejo planteado en la Figura 3.11.

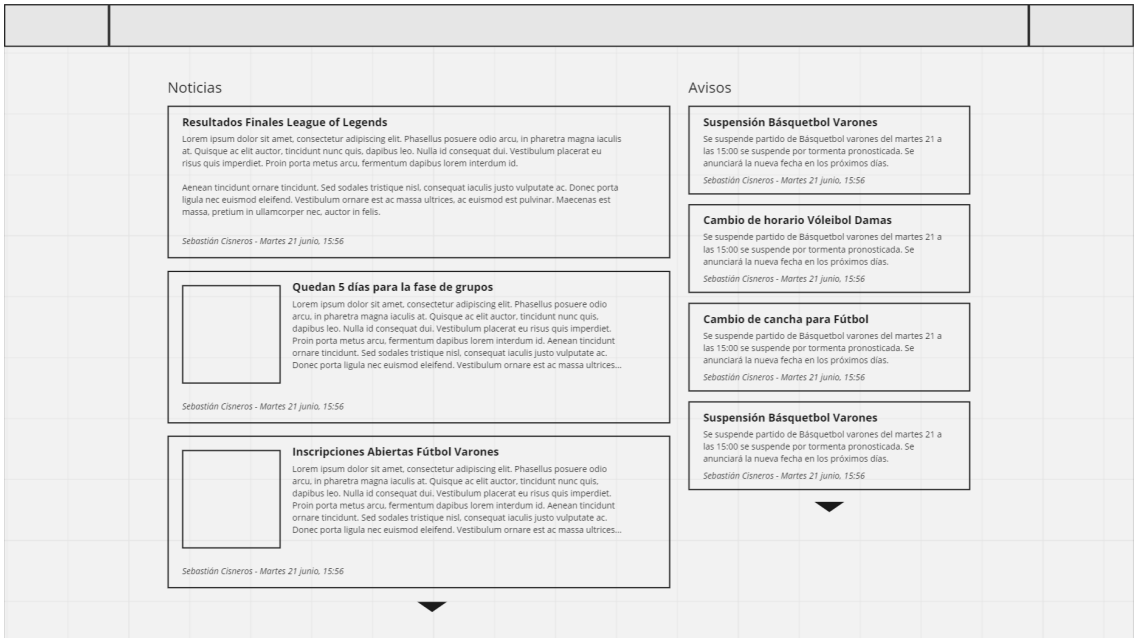

Figura 3.11: Bosquejo de página de noticias.

3.4.3. Partidos y otras tablas

La última vista importante que se planificó con detalle, fue la de Partidos, y análogamente, todas las vistas con tablas de datos.

Una falencia del diseño anterior era la poca flexibilidad en la búsqueda de datos, teniendo solo un input de texto que buscaba en todos los campos de todas las filas. Para resolver esto se propone un sistema de filtrado similar al de tiendas online y páginas de cotización de productos, con una barra lateral de filtros que puedan aplicarse independientemente. La Figura 3.12 muestra un bosquejo de la distribución del espacio. Se plantea además un sistema de paginación estándar, compatible con el hecho de que la API estará optimizada para enviar contenido parcialmente, en lugar de toda la información de una vez.

En particular para la tabla de Partidos, se realizó otra lluvia de ideas, visualizada en la Figura 3.13, en la que se calculó una proyección aproximada de registros para considerar cómo y cuánta información mostrar, llegando a proyectar aproximadamente 1.000 partidos por cada evento, tomando en cuenta el número de deportes y equipos participantes.

Se pensó en los filtros más posiblemente comunes para los partidos, considerándose como importante la capacidad de poder filtar por los partidos en los que el usuario tiene alguna participación. Se estableció que los filtros de texto deben tener una función de autocompletado, tomando en cuenta que podrían ser muchas opciones en cada variable.

Por último, se pensó en las acciones directas que se espera poder ejercer sobre cada partido. Si bien esto no se encuentra en el bosquejo, se propuso mantener los botones de acciones en la misma fila de cada registro, pero que se mostraran solo al desplazarse sobre ellos, en lugar de todos los botones a la vez.

Figura 3.12: Bosquejo de página de tablas filtradas.

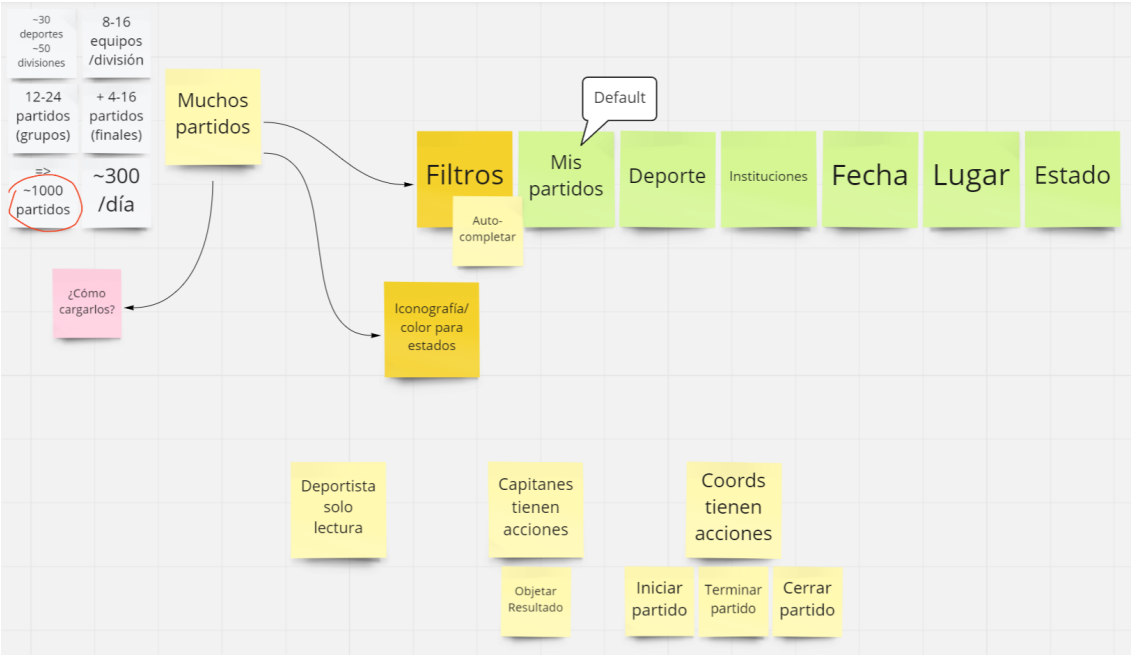

Figura 3.13: Lluvia de ideas sobre página de partidos.

3.4.4. Flujo del sitio

Finalmente, se exploraron las distintas acciones que cada tipo de usuario podría ejercer sobre cada una de las entidades. La Figura 3.14 muestra una vista extendida del tablero, a modo de mostrar el orden creciente de los roles, mientras que en la Figura 3.15 se puede apreciar con más detalle el tipo de anotaciones realizadas.

El objetivo de esta exploración era diseñar el flujo del sitio. Esto es, definir qué acciones y casos de uso tendría cada tipo de usuario, para definir la mejor armonización entre el cómo lograrían esa acción y dónde se ubicaría esa capacidad en la interfaz.

Sin embargo, el proyecto ya estaba en una etapa tardía y se debía empezar con la implementación, por lo que se dejó el proceso de diseño metodológico, y se inició implementando las vistas ya diseñadas.

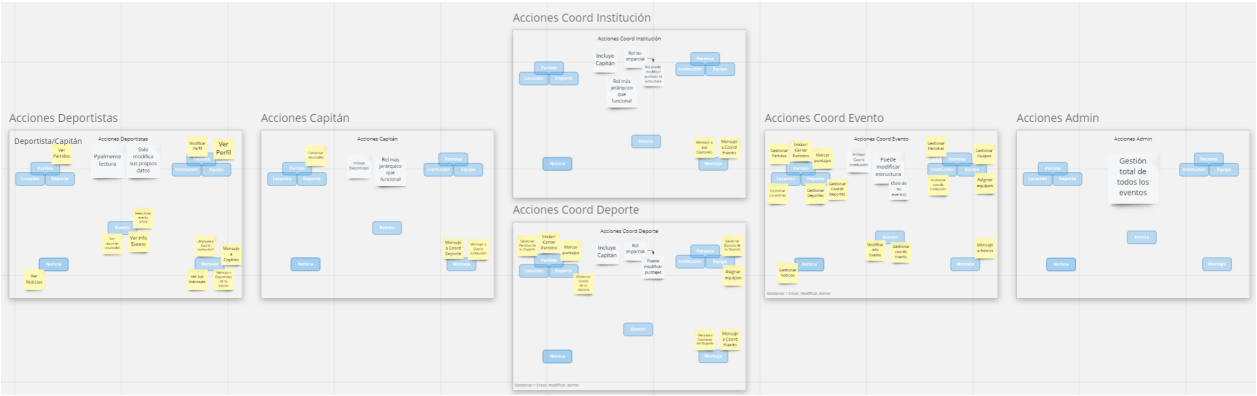

Figura 3.14: Vista general de la lluvia de ideas sobre acciones.

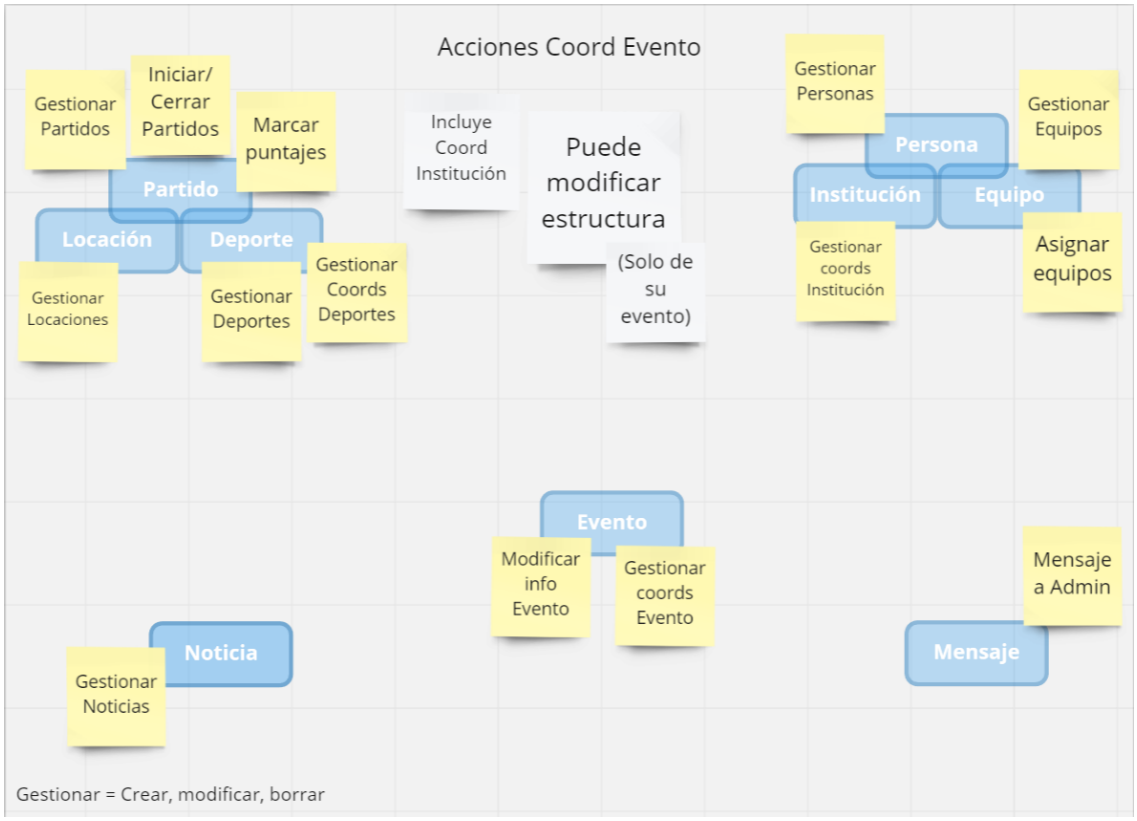

Figura 3.15: Vista particular de la lluvia de ideas sobre acciones.

Capítulo 4

Implementación de la solución

En este capítulo se iniciará haciendo una vista general del proceso de migración y las decisiones respecto a la plataforma anterior. Posteriormente se describirá más en detalle los cambios aplicados en cada una de las capas.

El modelo de datos de la plataforma original se mantuvo conceptualmente similar, conservando las entidades principales para representar las unidades de información atómicas (Match, Sport, Event, Location, etc). Sin embargo, se reestructuraron las relaciones entre ellas para diferenciar los registros de ocurrencias particulares (como la participación de una persona en un evento) y los modelos identitarios (como la información de una persona).

Por el lado del backend, las views de Django se usaron como referencia en términos funcionales, por ejemplo, para saber las acciones necesarias para cerrar un partido y calcular el puntaje, pero el cambio estructural a Django REST Framework modificó sustancialmente la sintaxis y flujos de código, por lo que la mayor parte del código fue reimplementada. Las consultas triviales de lectura, creación, modificación y borrado se manejan usando ViewSets de DRF, reduciendo significativamente el código necesario.

Las páginas en HTML de Django Templates no eran compatibles con la estructura de componentes dinámicas de React. En un principio se intentó usarlas como base y modificarlas iterativamente para los requerimientos del nuevo proyecto, pero el proceso de adaptación era innecesariamente complejo y finalmente se decidió desechar las plantillas antiguas y construir nuevas vistas desde cero.

4.1. Cambios en modelo de datos

En esta sección se describen los principales cambios hechos a la estructura de la base de datos. Se comentará respecto al cambio en tecnologías utilizadas y se abordará la principal problemática del modelo original: el alto acoplamiento de los datos en torno a un evento y su efecto en la funcionalidad. Finalmente, se presentará un listado específico de los cambios realizados a cada modelo y sus atributos.

En la Figura 4.1 se muestra un diagrama del modelo de datos modificado, que incluye solo las tablas relevantes a la estructura del sistema de campeonato, análogo al de la Figura 3.2.

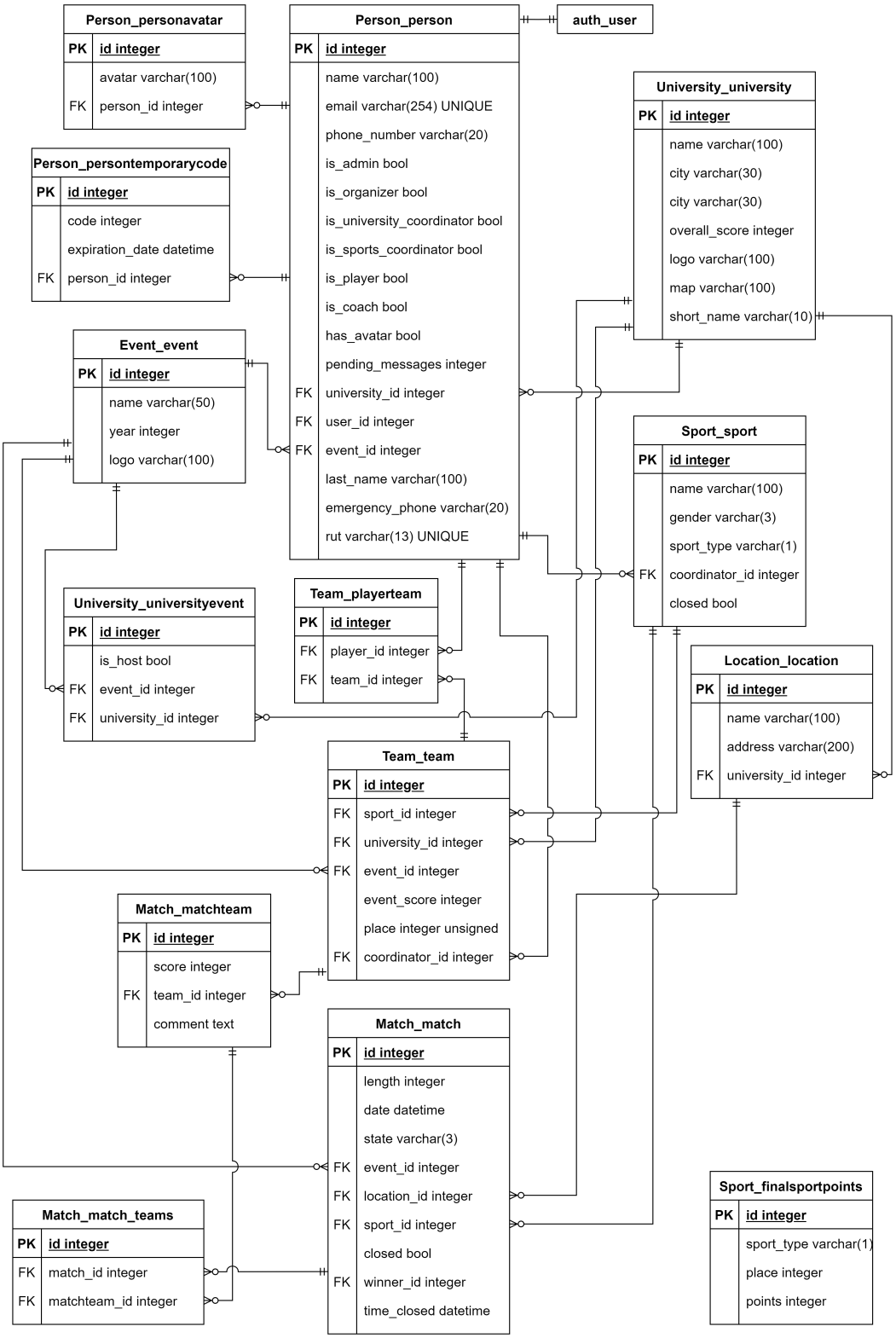

Figura 4.1: Modelo de datos modificado, relativo a la estructura del campeonato.

4.1.1. Tecnologías

Inicialmente la base de datos se alojaba en una nube remota de Google Cloud Storage, pero al no contar con las credenciales el acceso era incierto, y para efectos de la reconstrucción del proyecto no era necesario, por lo que se decidió dejar de utilizar este sistema y se configuró una base de datos local en SQLite.

Este cambio implica que la totalidad de los datos se encontrará de manera local en el servidor del proyecto. Dentro de los beneficios asociados al almacenamiento en la nube que se perderían, se destaca la garantía de estabilidad del servicio y datos al ser tercerizado en servidores dedicados, y la optimización en velocidad de las consultas. Sin embargo, no se estima que ninguno de esos beneficios se justifique en esta etapa del proyecto, y no ameritan la inversión de tiempo de configuración.

Se debe evaluar a futuro la necesidad de volver a utilizar un servicio de almacenamiento en la nube para la puesta en producción de la plataforma.

4.1.2. Distinción por evento

En términos funcionales, uno de los principales problemas del modelo original era la alta dependencia que tenían algunas entidades con un evento específico.

El modelo Person contenía una llave foránea al modelo Event, relacionándolos directamente. Esto significaba que si una misma persona participaba en dos eventos distintos, generaría dos instancias de Person, repitiendo mucha información y dificultando la posibilidad de relacionar a esta persona con sus distintas participaciones.

Los roles de cada persona (Admin, Organizador Evento, Coordinador Universidad, Coordinador Deporte) estaban definidos como un boolean dentro de Person. En lugar de eso, se crearon respectivas tablas relacionales para cada rol, listados en la Subsección 4.1.4. Esto permite indicar cuáles son las entidades específicas que coordinan y tener múltiples roles diferenciados en distintos eventos.

El modelo Sport, si bien no estaba directamente relacionado con Event mediante una llave foránea, sí tenía los campos `coordinator` y `closed`, que indican quién es la persona coordinadora de ese deporte y si el deporte se encuentra cerrado, respectivamente. Esto los relacionaba indirectamente, ya que ambos atributos son específicos para un evento particular (e.g. Fútbol Damas podría estar cerrado en los JING 2021, pero aún abierto en los JING 2022). Para independizar Sport, se creó la tabla relacional EventSport, que permite una relación muchos-a-muchos entre ambas, donde cada ocurrencia pueda tener sus atributos propios e independientes de la información del deporte en sí.

Un efecto de este último cambio es que el modelo Team, que semánticamente sí debe tener relación directa tanto con un evento como con un deporte, pasa a tener una llave foránea a EventSport, en lugar de Event y Sport por separado.

4.1.3. Puntajes y roles

Otra información crucial que se encontraba atada a un evento particular era el sistema de puntajes y posiciones. Este sistema es parte del núcleo de la plataforma y sus problemas destacan por sobre otras componentes del proyecto, ya que su consistencia y robustez son críticas para asegurar la fiabilidad y transparencia en la gestión de un evento deportivo competitivo.

En lugar de guardar los puntajes directamente en los modelos y actualizarlos en forma aditiva con cada cierre de deporte como se explica en la Subsección 3.1.2, se decidió guardar de forma independiente los posicionamientos de cada equipo en un deporte, creando la tabla SportStanding para ello. A modo de ejemplo, una instancia de SportStanding contendría la información necesaria para declarar que la Universidad de Concepción quedó en 3er lugar en Atletismo 100m. Esta nueva entidad no indica puntajes directamente, solo posiciones.

Análogamente, se crea la tabla EventStanding, que indica la posición final y definitiva de una universidad en un evento, con su respectivo puntaje acumulado. Por ejemplo, que la Universidad de Concepción quedó en 1er lugar en el evento JING 2022, con 120.310 puntos.

El proceso de cálculo y asignación de estos posicionamientos y puntajes se describe en la Subsección 4.2.3, al presentar el funcionamiento del servidor backend.

Los problemas que se resuelven con este acercamiento en contraste con la estructura anterior son los siguientes:

- Se quita la responsabilidad de guardar la información de puntajes directamente en las tablas de universidad, equipo o deporte. Eso reduce el acoplamiento de las mismas con un evento particular, y permite, por ejemplo, que una misma instancia de universidad participe en varios eventos distintos.
- Se reduce la posibilidad de inconsistencias. El sistema anterior mantenía una sumatoria fija de los puntajes de equipos y universidades. Esto obligaba a mantener sincronizados constantemente ambos puntajes, dificultando la mantenibilidad del código. Si en alguna parte del sistema se modificaba el puntaje de un equipo o el puntaje asociado a un deporte, el puntaje total de la universidad no vería reflejados estos cambios. En el modelo reestructurado, lo que se guardan son los resultados y posiciones, luego los puntajes tanto de equipo como de universidad son calculados dinámicamente en base a ellos.
- Se da mayor flexibilidad a correcciones manuales. Por la forma en que se calculan dinámicamente las posiciones y puntajes en el modelo reestructurado (explicada en la Subsección 4.2.3), es posible entregar a los coordinadores un cálculo preliminar en tiempo real de los valores de acuerdo a las reglas fijas, a la vez que les permite modificar estos cálculos antes de confirmarlos. Esto es útil para casos fuera del flujo regular, como por ejemplo, equipos amonestados o deportes cuyos posicionamientos son poco estándar (como los videojuegos).

Una posible desventaja de este cambio es que, al no almacenar los puntajes de cada equipo

en cada deporte, obliga a recalcularlos dinámicamente cada vez que se requieren. Sin embargo, como se menciona en la Subsección 3.4.3, la cantidad de partidos en cada evento es del orden de 1.000, repartidos en unos 50 deportes, por lo que no es un cálculo particularmente intenso considerando que no se espera que hayan múltiples eventos ocurriendo simultáneamente. Además, Django permite guardar resultados en caché de manera sencilla, por lo que podría usarse esa opción de ser necesario.

4.1.4. Resumen de cambios en el modelo

A continuación se detallarán todos los cambios realizados en el modelo de datos y sus atributos, junto a su justificación o utilidad. Se listarán los atributos separados por modelo, explicando si éstos fueron agregados, eliminados o modificados. Se mencionarán en una lista aparte los modelos nuevos agregados.

Event

- Atributos agregados:
  - `closed`: Indica si evento está cerrado para impedir cambios en puntajes.

Match

- Atributos agregados:
  - `played`: Indica si partido ya se jugó. Reemplaza a `state`.
  - `comment`: Comentario de cierre de partido, dado por organizador.
- Atributos eliminados:
  - `length`: Largo del partido en minutos. No se utilizaba.
  - `state`: Estado del partido. *"Por jugar"*, *"En curso"*, *"Finalizado"*. Se reemplaza por `played` y `closed`.
  - `teams`: Equipos participantes del partido. Se mueve a tabla relacional MatchTeam.
  - `winner`: Equipo ganador. Se mueve a tabla relacional MatchTeam.
- Atributos renombrados:
  - `time_finished`: Originalmente `time_closed`. Era confuso pues marca la hora en que finalizó el partido, no la hora de cierre.

MatchTeam

- Atributos agregados:
  - `match`: Partido relacionado. Relación muchos-a-muchos junto a campo `team`.
  - `is_winner`: Indica si este equipo ganó este partido.
  - `attended`: Indica si este equipo asistió a este partido.

NewsCategory

- Atributos eliminados:

  - `btn_class`: Nombre de clase CSS para estilizar botón según la categoría. No corresponde a capa de datos.

Person

- Atributos agregados:
  - `avatar`: Foto de perfil del usuario.
- Atributos eliminados:
  - `has_avatar`: Indica si persona tiene asociada un PersonAvatar. Innecesario al eliminar dicho modelo.
  - `is_university_coordinator`, `is_sports_coordinator`, `is_admin`, `is_coach`, `is_player`, `is_organizer`: Roles de usuario. Se trasladan a sus propias tablas relacionales.

PersonAvatar

- Se elimina el modelo completo. Establecía una relación muchos-a-muchos que no era necesaria ni se usaba en la plataforma.

Sport

- Atributos eliminados:
  - `coordinator`: Coordinador del deporte. Se reemplaza por SportCoordinator.
  - `closed`: Indica si el deporte está cerrado y con sus puntajes finales. Se traslada a EventSport.
  - `sport_type`: Indica tipo del deporte para la asignación de puntajes. Se traslada a EventSport.

Team

- Atributos eliminados:
  - `coordinator`: Coordinador/capitán del equipo. Se reemplaza por el nuevo modelo de rol TeamCoordinatorRole.
  - `event_score`: Puntaje en el evento. Se reemplaza por SportStanding.
  - `place`: Posición en la tabla de puntajes. Se reemplaza por SportStanding.

University

- Atributos eliminados:
  - `overall_score`: Puntaje en el evento. Se reemplaza por EventStanding.

Los siguientes modelos y sus atributos son nuevos en el modelo de datos. Su justificación de ser añadidos se describe en las Subsecciones 4.1.2 y 4.1.3.

EventSport

- `event`: Evento relacionado.
- `closed`: Deporte relacionado.
- `sport`: Indica si deporte está cerrado y con sus puntajes finales.
- `sport_type`: Tipo de deporte para este evento.

EventStanding

- `event`: Evento relacionado.
- `university`: Organización relacionada.
- `points`: Puntaje total de la organización en este evento.
- `place`: Lugar en la tabla de posiciones general de este evento.

SportStanding

- `event_sport`: Deporte relacionado (por EventSport).
- `university`: Organización relacionada.
- `place`: Lugar en la tabla de posiciones de este deporte.
- `participated`: Indica si esta organización participó en este deporte.

AdminRole, EventCoordinatorRole, UniversityCoordinatorRole, SportCoordinatorRole, TeamCoordinatorRole

- Estos modelos establecen los roles de usuarios. Cada uno relaciona un usuario con un evento, universidad, deporte o equipo, según corresponda.

4.2. Backend

En esta sección se presentará la estructura basal del funcionamiento del backend o servidor, que almacena y gestiona los datos de la aplicación.

Se describirá la transición a viewsets de Django REST Framework para desacoplar el procesamiento y comunicación de la información entre frontend y backend mediante una API, con su respectivo sistema de permisos; se explicará y listará la estructura de endpoints o puntos de acceso a las consultas y acciones de la API y se detallará el sistema de cálculo de puntajes, siendo ésta la acción más compleja y personalizada fuera de las consultas típicas.

4.2.1. Consultas CRUD base

Al migrar a Django REST Framework, se da la posibilidad de crear viewsets que implementan automáticamente las 4 funciones básicas CRUD, creación, lectura, edición y borrado.

Se crearon viewsets y serializadores base para cada modelo existente, generando dos endpoints para cada uno. Primero, el endpoint de forma `/api/model/` permite requests de tipo GET, que retornarán una lista con todas las filas del modelo, y requests de tipo POST, que crearán una nueva fila en el modelo con la información enviada en la request. Segundo, el

endpoint de forma `/api/model/[id]/`, que entrega la información de una instancia específica identificada por `[id]` y permite su edición y borrado.

Por supuesto, esto trae consigo sus propias complicaciones. Al configurar operaciones CRUD sobre los modelos, se abren decenas de posibles puntos de entrada ante usuarios maliciosos o requests equivocadas. Se debe procurar que cada punto de acceso tenga los permisos adecuados, para que ningún usuario no calificado pueda realizar cambios en la información del evento.

El sistema de permisos de Django REST facilita estructurar esto de forma sencilla. Todos los permisos son clases que heredan de BasePermission, y básicamente lo único que se requiere es implementar un método `has_permission`, que retorna un booleano para otorgar el permiso. Este permiso luego se pasa al viewset a través del campo `permission_classes`.

En el Código 4.1 se puede ver esto ejemplificado. Se observa que SportStandingViewSet tiene un `permission_class` asignado a `[IsCoordinatorForSport|ReadOnly]`, y se presentan las implementaciones de ambos permisos. El operador `|` puede usarse como un OR lógico entre los permisos. Esto quiere decir que las consultas a SportStandingViewSet solo se aceptarán si la persona tiene un rol de coordinación en ese deporte, o si la solicitud de la request es de solo lectura.

```python
# SportStanding/view.py
class SportStandingViewSet(ModelViewSet):
    permission_classes = [IsCoordinatorForSport|ReadOnly]
    ...

# SportStanding/permissions.py
class IsCoordinatorForSport(BasePermission):
    def has_permission(self, request, view):
        sport = request.data.get('sport')
        user = request.user
        role = SportCoordinatorRole.objects.get(sport=sport,person__user=user_id)
        return role.exists()

# backend/permissions.py
class ReadOnly(BasePermission):
    def has_permission(self, request, view):
        return request.method in SAFE_METHODS
```

Código 4.1: SportStandingViewSet.py con permisos asignados.

Actualmente no todas las viewsets cuentan con permisos específicos.

Se dejó establecido como permiso por defecto a nivel de todo el proyecto el permiso `IsAuthenticatedOrReadOnly`, que chequea que el usuario de la request esté autenticado en el sistema, si no, solo otorga permisos de lectura.

4.2.2. Endpoints de la API

En la Tabla 4.1 se listan los endpoints o enlaces de consulta de la API. Estos son los puntos de acceso a sus funcionalidades, y se acceden añadiendo la dirección asociada al endpoint a la siguiente URL:

`https://jing.dcc.uchile.cl/`

Por ejemplo, si se quisiera acceder a la información en la base de datos relacionada específicamente al Evento con el identificador 8, se debe consultar a la URL:

`https://jing.dcc.uchile.cl/api/events/8/`

Estos endpoints en general tienen la siguiente estructura:

- Inicia con el texto "`/api/`". Como se explica en la Sección 4.4, esto apuntará la consulta al servidor de la API.
- Luego sigue el nombre de la entidad asociada a la consulta, escrito en plural. Por ejemplo, "`events/`".
- Si la consulta es sobre una instancia específica de dicha entidad, se agrega su identificador único. Por ejemplo, "`8/`"
- Para las consultas asociadas a acciones se añade el nombre de la acción. Por ejemplo, "`/matches/132/close/`" solicita que se cierre el partido número 132.
- Las consultas terminadas en "`filters/`" retornan las opciones de filtrado de cierta entidad. Por ejemplo, "`/matches/filters/`" retornará una lista de todas las posibles universidades, deportes y lugares que podría tener un partido en cierto evento. Esto se usa en el frontend para dar las opciones al usuario.

Los endpoints de consulta masiva en general admiten filtros en parámetros GET de la siguiente forma:

`/api/matches/?event=8&location=5&participants=3,7`

En esta consulta de ejemplo, se están solicitando todos los partidos del evento 8 que se jueguen en la locación 5 y que participen simultáneamente las organizaciones 3 y 7.

4.2.3. Cálculo de posiciones y puntajes

En el sistema original, como los puntajes estaban directamente asociados al equipo o institución a nivel del modelo de datos, en general no había un cálculo de puntajes per se, sino que se sumaban los puntos correspondientes al finalizar cada partido o declarar un deporte cerrado. Luego de la reestructuración del modelo, se cambió este enfoque a uno más modular.

Ahora cada relación MatchTeam, que indica la participación de un equipo en un partido, tiene un atributo `score` y uno `is_winner`. Notar que un mismo partido podría tener múltiples MatchTeams declarados como ganadores, permitiendo empates, que no eran posibles en el modelo anterior.

Al hacer una request de tipo GET a:

`/api/standings/sport/calculate/[s_id]/?event=[e_id]`

donde `s_id` y `e_id` son los identificadores de un deporte y un evento, respectivamente, se contarán todas las victorias que cada institución haya tenido en ese deporte y evento, además del número de partidos totales, el número de partidos asistidos y la suma de su marcador en cada partido.

Luego, se ordenarán las instituciones según la cantidad de victorias que tengan, con la suma de marcador como segundo criterio, y se les asignará un lugar en la tabla de posiciones, admitiendo empates. Después se les asignará un puntaje a cada uno dependiendo de los puntajes asignados al tipo de deporte que sea. Si una universidad no tuvo ningún partido en ese deporte, o si tuvo pero no asistió a ninguno, se considera que no participa y no se le asigna puntaje.

Esta tabla de posiciones y puntajes **no** se guarda en la base de datos, sino que se envía al usuario. La intención de esto es que se envíe al frontend el cálculo automático de los puntajes, pero que la última decisión sea de los coordinadores, que podrían tomar esta sugerencia calculada y confirmarla, o editarla antes de enviarla como puntaje final.

Para confirmarla, se espera una request POST a:

`/api/standings/sport/set/`

con la tabla de posiciones y puntajes aprobada por los coordinadores. Esta request marcara el deporte como cerrado y creará los registros de SportStanding en la base de datos.

Nótese que ni los puntajes ni las posiciones están en ningún momento asociados directamente ni con los equipos, ni con los partidos, ni con los deportes, sino que viven modularmente en la base de datos como un registro de SportStanding.

Un protocolo similar se sigue para EventStanding, la diferencia es que en lugar de contar victorias en partidos y marcadores, para ese caso se calculan todos los SportStandings para cada deporte del evento, se suman los puntajes obtenidos en cada uno y luego se ordenan de la misma forma que el caso anterior. Se envía el cálculo al usuario, pero no se guarda nada hasta que explícitamente se recibe la request POST en la URL:

`/api/standings/event/set/`

El servidor chequea que todos los partidos de un deporte estén cerrados para poder cerrarlo, y análogamente chequea que todos los deportes estén cerrados para poder cerrar un evento. Si no se cumple, se rechaza la request.

URL	*Nombre*	*Detalle*
/api/events/	events-list	Listar todos los eventos
/api/events/[id]/	events-detail	Obtener evento específico
/api/locations/	locations-list	Listar todas las locaciones
/api/locations/[id]/	locations-detail	Obtener locación específica
/api/logs/	logs-list	Listar todos los registros de cambios
/api/logs/[id]/	logs-detail	Obtener registro específico
/api/matches/	matches-list	Listar todos los partidos
/api/matches/[id]/	matches-detail	Obtener partido específico
/api/matches/[id]/close/	matches-close	Cerrar partido
/api/matches/[id]/finish/	matches-finish	Finalizar partido
/api/matches/filters/	matches-filters	Obtener filtros de partidos
/api/messages/	messages-list	Listar todos los mensajes
/api/messages/[id]/	messages-detail	Obtener mensaje específico
/api/news/	news-list	Listar todas las noticias
/api/news/[id]/	news-detail	Obtener noticia específica
/api/news_categories/	news_categories-list	Listar todas las categorías de noticia
/api/news_categories/[id]/	news_categories-detail	Obtener categoría específica
/api/persons/	persons-list	Listar todas las personas
/api/persons/[id]/	persons-detail	Obtener persona específica
/api/persons/filters/	persons-filters	Obtener filtros de personas
/api/player_teams/	player_teams-list	Listar todas las relaciones persona-equipo
/api/player_teams/[id]/	player_teams-detail	Obtener persona-equipo específico
/api/sports/	sports-list	Listar todos los deportes
/api/sports/[id]/	sports-detail	Obtener deporte específico
/api/standings/event/	event_standings-list	Listar todas las posiciones de evento
/api/standings/event/[id]/	event_standings-detail	Obtener posición de evento específica
/api/standings/event/calculate/[event_id]/	event_standings-calculate	Calcular las posiciones para un evento
/api/standings/event/set/	event_standings-set	Definir posiciones de un evento
/api/standings/sport/	sport_standings-list	Listar todas las posiciones de deporte
/api/standings/sport/[id]/	sport_standings-detail	Obtener posición de deporte específica
/api/standings/sport/calculate/[sport_id]/	sport_standings-calculate	Calcular las posiciones para un partido
/api/standings/sport/set/	sport_standings-set	Definir posiciones de un partido
/api/teams/	teams-list	Listar todos los equipos
/api/teams/[id]/	teams-detail	Obtener equipo específico
/api/teams/filters/	teams-filters	Obtener filtros de equipos
/api/universities/	universities-list	Listar todas las instituciones
/api/universities/[id]/	universities-detail	Obtener institución específica
/api/users/	users-list	Listar todos los usuarios
/api/users/[id]/	users-detail	Obtener usuario específico

Tabla 4.1: Listado de endpoints

4.3. Frontend

En esta sección se describirán los métodos seguidos al implementar las vistas diseñadas previamente. En general, para cada subsección se dará un acercamiento a la estrategia adoptada para la implementación, agregando código solo para las secciones nucleares de la plataforma, y luego se mostrará el resultado visual del proceso.

Los fragmentos de código de esta sección se encuentran editados por simplicidad, omitiendo líneas y borrando información irrelevante a lo que se busca explicar.

4.3.1. Navegación

Para poder utilizar React correctamente como una single-page app, o aplicación de página única, se utiliza la librería React Router, que permite navegar entre componentes y modificar la URL e historial del explorador para simular una navegación estándar, cuando realmente solo se están montando y desmontando componentes en un mismo sitio, evitando recargas del explorador y manteniendo el sitio siempre en foco.

Para lograr esto, se construye un mapeo de rutas en la raíz del árbol de componentes, la componente App.js. Esta mapeo asociará cada ruta con una componente raíz, y este tipo de componentes se encuentra organizada apropiadamente en su subdirectorio propio, dentro del directorio `pages/`.

```
1  function App() {
2    ...
3    return (
4      <EventContext.Provider value={{ event, setEvent }}>
5        <AlertProvider template={Alert} timeout={10000} position='bottom center'>
6          <div className={styles.root}>
7            <NavBar />
8            <TransitionGroup component={null} exit={false}>
9              <CSSTransition key={location.pathname} classNames="fade" timeout={0}>
10               <Routes location={location}>
11                 <Route index element={<News />} />
12                 <Route path="personas" element={<Persons />} />
13                 <Route path="partidos" element={<Matches />} />
14                 <Route path="equipos" element={<Teams />} />
15                 <Route path="mensajes" element={<Messages />} />
16                 <Route path="administracion" element={<Admin />} />
17                 <Route path="resultados" element={<Results />} />
18                 <Route path="eventos" element={<Events />} />
19                 <Route path="mapa" element={<Maps />} />
20
21                 <Route path="*" element={<Navigate to="/" replace={true} />} />
22               </Routes>
23             </CSSTransition>
24           </TransitionGroup>
25         </div>
26       </AlertProvider>
27     </EventContext.Provider>
28   );
```

Código 4.2: App.js. Se configuran las rutas del sitio y contextos globales.

Cabe notar en la línea 7 del Código 4.2, se encuentra la componente NavBar. Está no pertenece al sistema de rutas, ya que está presente en todas las páginas.

Esta componente NavBar se construye principalmente a base de la librería Navbar de Bootstrap, y se organiza en tres subcomponentes: NavBarLeftSection, NavBarMidSection y NavBarRightSection, que representan las secciones descritas en la Subsección 3.4.1. Éstas son

relativamente estándar, y utilizan NavLinks de React Router, que permiten la navegación single-page descrita anteriormente.

4.3.2. Tablas filtrables

Dado que la app busca extender las funciones de una planilla, su núcleo son las vistas con tablas de datos que pueden filtrarse bajo ciertos parámetros, haciendo consultas a la API en tiempo real.

Los Partidos, Equipos, Personas, Deportes y Universidades usan estas tablas. Para efectos de ejemplificar, en esta subsección se describirá cómo se genera la página de Partidos. Las demás páginas de tablas funcionan de la misma forma.

Teniendo en cuenta que esta vista es crucial y repetitiva a lo largo del sitio, se buscó implementarla de la manera más reutilizable posible. Para esto, inicialmente se hizo una componente tipo *wrapper*[1], que cumpliera la función de estructurar una página con una barra lateral, pudiendo entregarle distintos contenidos tanto para la barra como para el cuerpo de la página. Se creó entonces la componente `pages/wrapper/SidebarPage.jsx`, cuya única función es separar el sitio en dos partes, usando las componentes de grilla de Bootstrap. (Código 4.3)

```
function SidebarPage({sidebar, children, rootRef, ...props}) {
  return (
    <Container fluid className={styles.root} ref={rootRef} {...props}>
      <Row className={styles.wrapper}>
        <Col xs="2" className={styles.sidebar}>
            {sidebar}
        </Col>

        <Col className={styles.content}>
          {children}
        </Col>
      </Row>
    </Container>
  );
}
```

Código 4.3: SidebarPage. Divide el sitio en una barra lateral y contenido.

Al usar este wrapper, se le puede pasar otra componente en el prop `sidebar`, y esta se renderizará en la barra lateral. Los elementos que se inserten como hijos de este wrapper serán renderizados en la sección de contenido principal.

[1]Un *wrapper*, o envoltura, cumple la función de envolver una o varias sub-componentes, con el fin de estructurarlas de cierta manera o de asignarles propiedades. No es una componente útil por sí sola.

```
<SidebarPage sidebar={<h2>Esto va en el sidebar</h2>}>
    <h1>Esto va en el contenido</h1>
</SidebarPage>
```

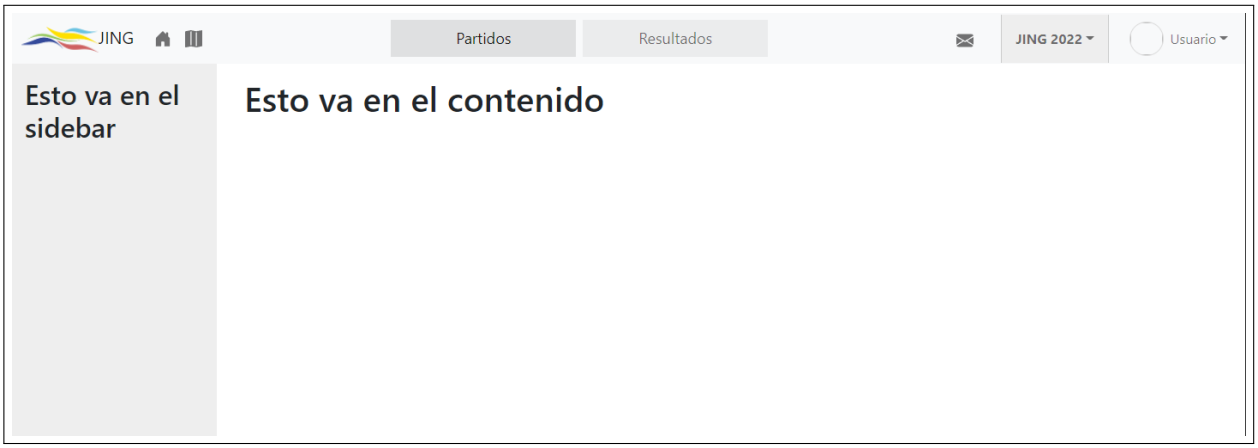

Figura 4.2: Ejemplo de uso de SidebarPage. Código en la parte superior de la imagen.

Teniendo este wrapper estructural, se agrega por encima un wrapper funcional. Esta nueva componente se llama `pages/wrapper/TableSearchPage.jsx`, y representa una página con una tabla y filtros de búsqueda. En los estados de esta componente estarán guardados los filtros activos, y a través de `useEffect`, se encarga de chequear si hay cambios en los mismos. De ser así, efectuará la consulta con los parámetros definidos. Esta componente se describe en el Código 4.4.

TableSearchPage retorna solamente un SidebarPage al que le incrusta la componente para la barra lateral con filtros, y en el cuerpo genera un título, un indicador de carga e incrusta la componente correspondiente a la tabla de datos. Estas componentes de barra lateral y de tabla las recibe a través de los props SidebarComponent y TableComponent.

Cabe destacar que ambos `useEffect`s generan cambios circulares. Si se cambian los filtros se cambia la URL, y si se cambia la URL se cambian los filtros. Este comportamiento se mantiene controlado mediante un estado interno, y es necesario para procurar la consistencia entre filtros y URL, que se discutirá más adelante.

```jsx
function TableSearchPage({SidebarComponent, TableComponent, apiUrl, title}) {
  function fetchData() {
    axios
      .get(`https://jing.dcc.uchile.cl${apiUrl}${location.search}`)
      .then((response) => {
        ...
        setRows(response.data.results);
      });
  }
  ...
  // Si cambia la URL, se actualizan los filtros y se hace la consulta.
  useEffect(() => {
    ...
    setCurrentPage(pageParam);
    setFilters(filtersParam);
    fetchData();
  }, [location]);

  // Si cambian los filtros, se actualiza la URL.
  useEffect(() => {
    ...
    navigate("?" + objectToParamsString(searchFilters));
    setAlreadyChanged(true);
  }, [event, currentPage, filters]);

  return (
    <SidebarPage
      sidebar={<SidebarComponent filters={filters} setFilters={setFilters} />}
    >
      <h1 className=styles.title>{title}</h1>
      {isLoading && (<LoadingIndicator />)}
      <h4>{event?.name}</h4>

      <TablePagination ... />
      <TableComponent rows={rows} fetchData={fetchData} />
      <TablePagination ... />
    </SidebarPage>
  );
}
```

Código 4.4: Componente TableSearchPage. Hace requests al cambiar los parámetros.

Con este wrapper funcional, envolviendo un wrapper estructural, podemos crear la componente para cualquier página de tablas, en particular la página de Partidos. Esta componente será `pages/Matches/index.jsx`. Las componentes de nombre `index` pueden refenciarse usando el nombre del directorio que la contiene. Para usar TableSearchPage es necesario darle un SidebarComponent y un TableComponent. Para eso se crea MatchesSidebar y MatchesTable.

MatchesSidebar define el contenido de la barra lateral, que serán principalmente inputs de distintos tipos para filtrar. Recibirá de TableSearchPage los props necesarios para leer y actualizar los filtros aplicados.

MatchesTable define cómo se renderizarán las filas de data. Recibe desde TableSearchPage

las filas obtenidas de las requests, y la función `fetchData()`, para que desde la misma tabla se pueda invocar una recarga de la información.

Resumiendo, con la ayuda de TableSearchpage, para crear la página de partidos es necesario crear 3 componentes:

- `pages/Matches/index.jsx`: Solamente genera una instancia de TableSearchPage con los props adecuados. (4.5)
- `pages/Matches/MatchesSidebar.jsx`: Contiene los inputs para ingresar filtros. (4.6)
- `pages/Matches/MatchesTable.jsx`: Contiene la tabla para mostrar los resultados. (4.7)

Esto se puede hacer de la misma forma para cualquier otra página de tabla. Podría ser posible abstraer más los componentes de XSidebar y XTable, pero cada entidad tiene sus distintos tipos y condiciones de filtrar, además de diferentes formas de mostrarse en una tabla, por lo que se prefirió que se implementara ad hoc para cada una.

```
function Matches() {
  return (
    <TableSearchPage
      SidebarComponent={MatchesSidebar}
      TableComponent={MatchesTable}
      apiUrl="/api/matches/"
      label="partidos"
    />
  );
}
```

Código 4.5: Matches/index. Crea una instancia de TableSearchPage

```
function MatchesSidebar({ filters, setFilters }) {
  useEffect(() => {
    const fetch = axios
      .get(`https://jing.dcc.uchile.cl/api/matches/filters/?event=${event.id}`)
      .then((response) => {
        setParticipantsOptions(response.data.participants ?? []);
        setSportOptions(response.data.sport ?? []);
        setLocationOptions(response.data.location ?? []);
      })
  }, [event]);
  ...
  return (
    <div>
      <Form.Check
        name="my_matches"
        label="Mostrar solo mis partidos"
        onChange={handleCheckbox}
        checked={filters.my_matches}
      />

      <Form.Label htmlFor="participants">Participantes</Form.Label>
      <Select
        isMulti
        name="participants"
        options={participantsOptions}
        onChange={handleSelect}
        value={participantsOptions.filter((o) => filters.participants.includes(o.value))}
      />

      <Form.Label htmlFor="sport">Deporte</Form.Label>
        <Select
        isClearable
        placeholder="Sin filtrar"
        name="sport"
        options={sportOptions}
        onChange={handleSelect}
         value={sportOptions.find((o) => o.value === filters.sport) ?? null}
      />
      ...
    </div>
)}
```

Código 4.6: MatchesSidebar. Renderiza los filtros de partidos.

```
function MatchesTable({ rows, fetchData, ...props }) {
  const alert = useAlert();
  function requestAction(matchId, action) {
    axios
      .post(`https://jing.dcc.uchile.cl/api/matches/${matchId}/${action}/`)
      .then((response) => {
        fetchData();  // Cambió un partido, actualizar data.
        alert.show(`El partido ha sido ${action}`);
      })
      .catch((error) => {
        alert.error(error.message);
       }
      });
  }

  return (
    <Table className={styles.table} ...>
      <thead>
        <tr>
          <th>Fecha</th>
          <th>Hora</th>
          <th>Lugar</th>
          ...
          <th>Cerrado</th>
          <th></th>
        </tr>
      </thead>
      <tbody>
        {rows.map((row) => (
          <tr key={row.id}>
            <td>{moment(row.date).format("ddd DD-MM-YY")}</td>
            <td>{moment(row.date).format("HH:mm")}</td>
            <td>{row.location.name}</td>
            ...
            <td>{row.closed ? <Check /> : <XLg/>}</td>
            <td className="text-center">
              <Button className={styles.action} variant="secondary"
                      onClick={() => handleClickFinish(row.id)}>
                Finalizar
              </Button>
              <Button className={styles.action}
                      onClick={() => handleClickFinish(row.id)}>
                <InfoCircleFill/>
              </Button>
              <Button className={styles.action} variant="danger"
                      onClick={() => handleClickDelete(row.id)}>
                <TrashFill />
              </Button>
            </td>
          </tr>
        ...
```

Código 4.7: MatchesTable. Renderiza la tabla de partidos.

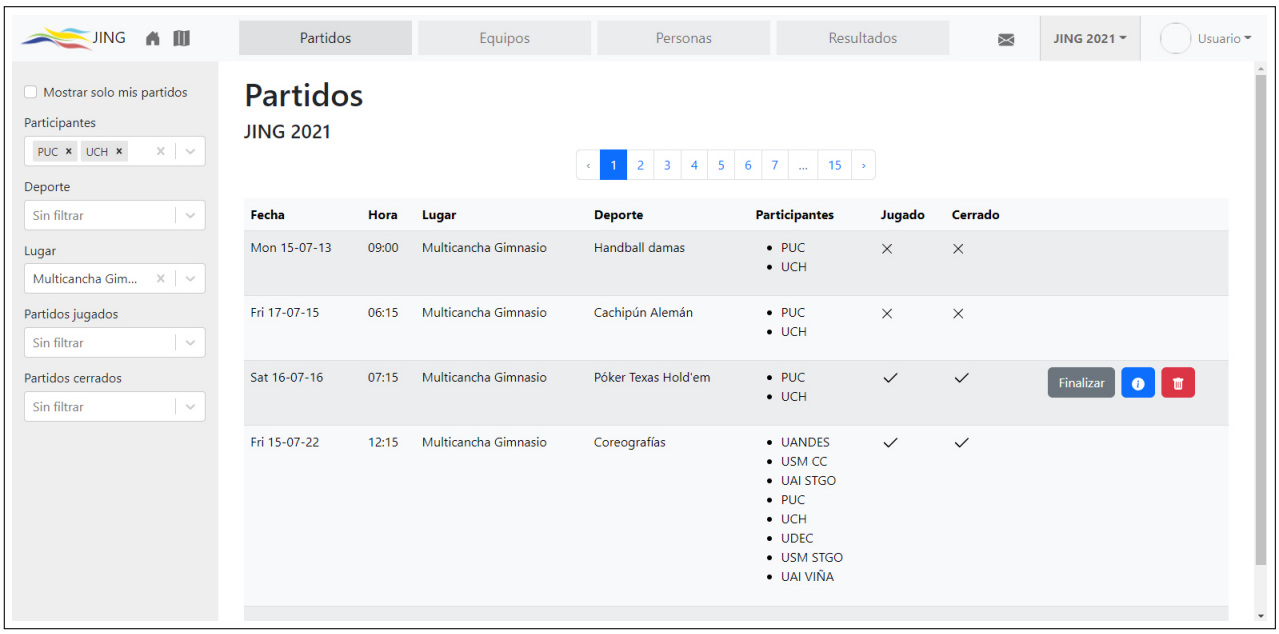

Figura 4.3: Vista de búsqueda de partidos

4.3.3. Selección de evento

Tal como se diseñó, se agregó un botón en la barra de navegación que mostrara constantemente el evento activo actual. Al presionar este botón, se despliega un menú que permite acceder a la vista para cambiar evento.

Originalmente, el menú desplegable contenía directamente los otros eventos, pero generaba algunos problemas al tener que consultar los eventos seleccionables, y por simplicidad y tiempo se decidió cambiarlo a una vista aparte. De todas formas, mucha de la data y consultas dependen del evento activo actual, por lo que poder cambiarlo globalmente desde cualquier sitio abría la posibilidad de fallos y provocaba que todas las páginas tuvieran que considerar el caso en que cambiara el evento en cualquier momento. En ese sentido, se considera mejor solución forzar al usuario a una vista específica antes de poder cambiar el evento.

La vista de selección de eventos no constituye gran complejidad, pero se estilizó más exageradamente que el resto de las páginas, solo para evitar que fuera una página en blanco con un selector o una lista corta.

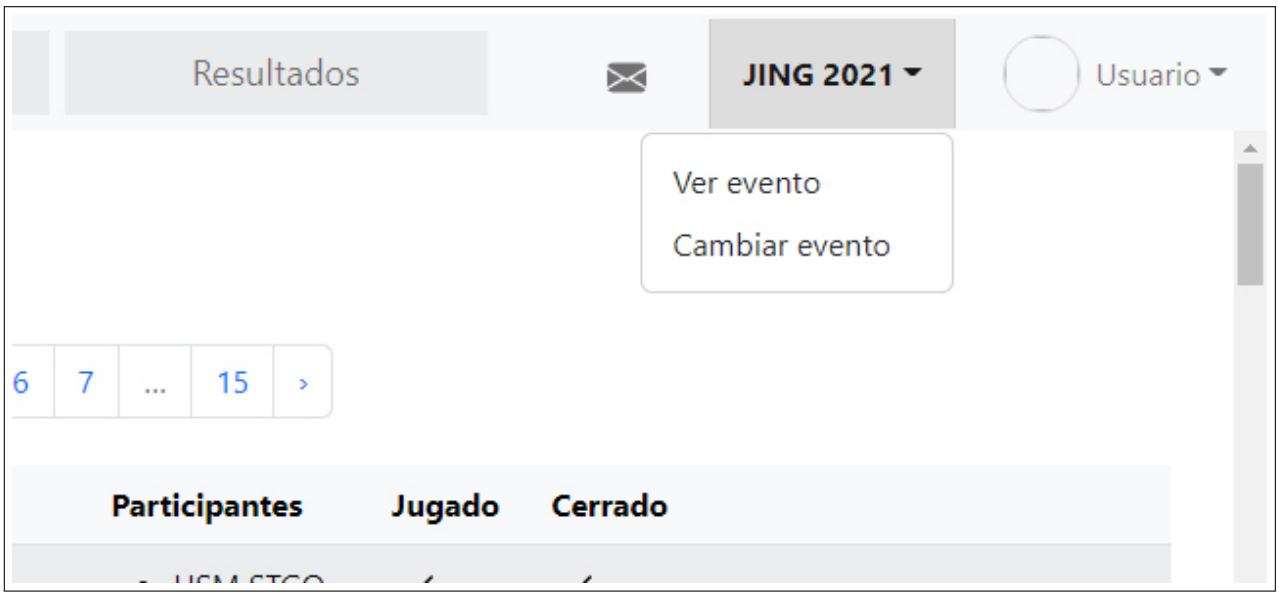

Figura 4.4: Menú desplegable para evento

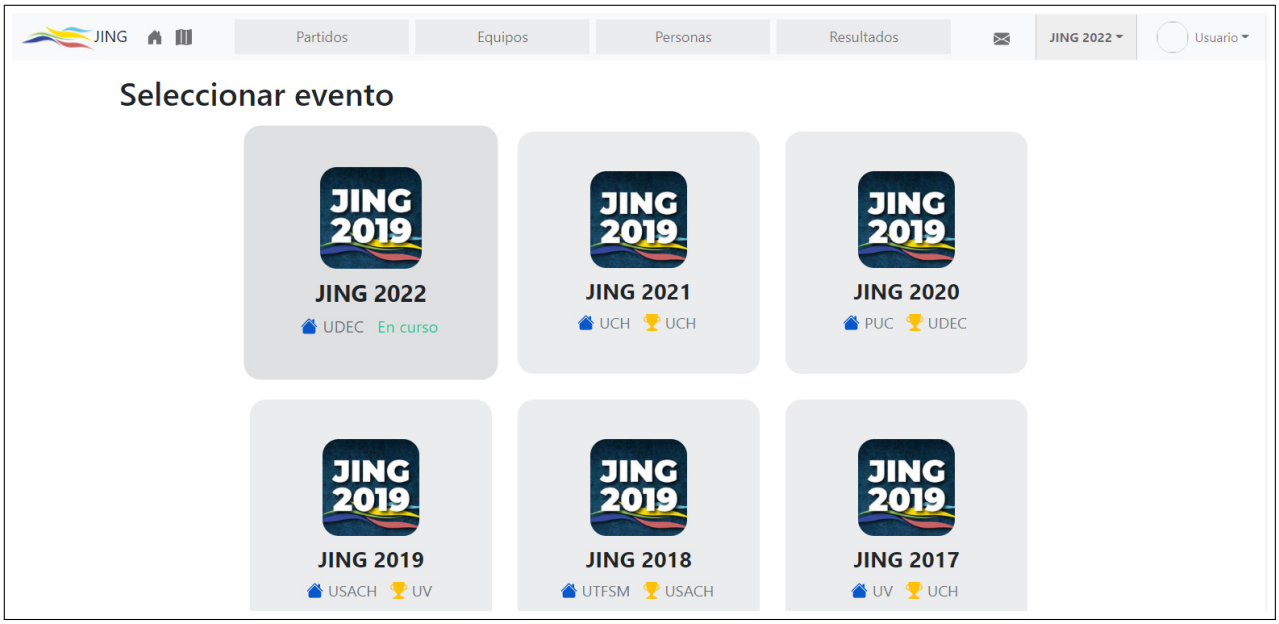

Figura 4.5: Vista de selección de evento

4.3.4. Resultados y puntajes

Las interfaces para mostrar los resultados, si bien tienen una estética ligeramente más estilizada, no son particularmente complejas. Constan de un selector desplegable para elegir qué deporte se quiere ver, o ver los resultados generales del evento. Al cambiar este selector, se consulta al backend y se reciben todas las estadísticas de cada equipo precalculadas desde el servidor, como se explica en la Subsección 4.2.3

#	Universidad	Victorias	Marcador	Participa	Puntos	Partidos	Jugados
1	Universidad Técnica Federico Santa María - Santiago	3	13	✓	1300	13	4
2	Universidad Técnica Federico Santa María - Casa Central	2	24	✓	800	7	3
3	Universidad Adolfo Ibáñez - Santiago	2	18	✓	480	8	3
# 4	Universidad de los Andes	1	20	✓	240	10	4
# 4	Universidad Adolfo Ibáñez - Viña	1	20	✓	240	10	3
# 6	Universidad Católica	0	4	✓	60	9	1
# 7	Universidad de Concepción	0	0	✓	30	12	1
# 8	Universidad de Chile	0	0	×	0	9	0

Figura 4.6: Tabla de resultados para un deporte

#	Universidad	Puntos
1	Universidad Técnica Federico Santa María - Santiago	33380
2	Universidad de Concepción	32120
3	Universidad de los Andes	27700
# 4	Universidad Técnica Federico Santa María - Casa Central	27500
# 5	Universidad de Chile	25420
# 6	Universidad Católica	23840
# 7	Universidad Adolfo Ibáñez - Viña	21950
# 8	Universidad Adolfo Ibáñez - Santiago	18750

Figura 4.7: Tabla de resultados para el evento completo

4.3.5. Consideraciones de usabilidad

Filtros

Se destacan los filtros como punto fuerte en términos de mejoras orientadas a la usabilidad. Al implementar librerías de inputs se consiguió una experiencia flexible que permite, por ejemplo, seleccionar varias universidades usando un solo input de Select, o buscar resultados

con autocompletado. Además, se dio bastante dedicación en definir las funciones de TableSearchPage, procurando implementar de forma estable la sincronización entre las acciones del usuario, el estado de los filtros y los resultados mostrados. El primer acercamiento a esta implementación resultó en una experiencia con bastantes bugs al momento de hacer búsquedas veloces con modificaciones rápidas a los filtros, pero se mejoró iterativamente hasta asegurar una buena experiencia.

Transiciones y carga

Si bien el sistema de single-page app es llamativo por su velocidad de navegación inmediata, esto puede ser un problema cuando las vistas entre las que se está transicionando son tablas de muchos datos. Si el cambio entre una y otra es inmediato, es fácil perderse o producir ceguera al cambio [21]. Utilizando la librería React Transition Group, se logró hacer transiciones suaves entre distintas secciones y en la aparición de elementos en pantalla.

Otro problema generado por el estilo single-page, es que todas las consultas de datos son asíncronas, por lo que cada vez que se hace una, por defecto la interfaz no hace nada hasta que llega la respuesta. Si bien estos tiempos de espera suelen ser bastante pequeños, ocurren en casi todas las interacciones, pudiendo acumular una sensación de falta de control [22]. Para combatir esto, es útil añadir elementos que indiquen al usuario que su acción fue registrada y que el sistema la está procesando [23], por lo que se añadieron pequeños elementos giratorios, o *spinners*, que dan esta retroalimentación sin ser intrusivos en la atención del usuario.

Figura 4.8: Elemento giratorio de carga (*spinner*) a la derecha del título

Alertas

En la misma línea que el ítem anterior, las alertas entregan retroalimentación crucial al usuario para conocer el resultado de sus acciones.

Se implementó un sistema de alertas que puede ser accedido fácilmente desde cualquier parte del sitio, como se muestra en las líneas 2, 8 y 11 del Código 4.7. Basta con inicializar la alerta con `useAlert()` y luego `alert.show()` mostrará la alerta desde la raíz de la aplicación.

Figura 4.9: Sistema de alertas en el borde inferior de la pantalla

Enlaces consistentes Algo en lo que se tuvo particular consideración al implementar las tablas filtrables, fue que el estado de los filtros se viera reflejado fielmente en la URL del sitio, y por el otro lado, que si se modificaba la URL, se actualizaran los filtros en concordancia. Esto indirectamente provoca mejoras en la experiencia de la navegación, por ejemplo, al garantizar que al desplazarse por el sitio usando el historial o los botones *Atrás* y *Adelante* del explorador, la plataforma responderá de manera consistente, cosa que no ocurre en la mayoría de los sitios single-page.

Otro beneficio de esto es que permite poder compartir enlaces de búsqueda entre personas. Si alguien ingresa a esta dirección del sitio:

```
/partidos/?event=5&participants=1,3,5,6&sport=1&page=1
```

se garantiza que el sitio iniciará buscando los partidos según los parámetros entregados.

4.3.6. Funcionalidades no implementadas

Debido a la excesiva extensión de las etapas de análisis, diseño y desarrollo del backend, la implementación del frontend se inició tardíamente y no se pudo completar.

La principal funcionalidad faltante en este desarrollo fue un sistema apropiado de usuarios, roles y permisos. No existe un login ni registro, ni comunicación con el backend en términos de autenticación. A nivel de frontend, esto implica que todos los usuarios tienen acceso a todas las funcionalidades, y esto a su vez compromete la definición de un flujo adecuado de acciones, que como se mencionó en la Subsección 3.4.4, tampoco tuvo un diseño preliminar.

Otra característica clave del proyecto que no se pudo implementar, fue la interfaz del sistema de mensajería. No se alcanzó a iniciar un desarrollo, ni se diseñó preliminarmente, por lo que solo existe a nivel de backend. Este sistema está funcionalmente enlazado con el sistema de autenticación y roles, que al no estar presente también dificultaría su confección.

Finalmente, una vista menos crítica que no alcanzó a hacerse fue la de noticias. Se considera como menos grave ya que por su naturaleza no implica una logística compleja ni muy atada a la autenticación de usuarios. Es principalmente una visualización de los datos recibidos, con algún formulario simple de envío, similar a la vista de eventos, por lo que finalizar

esta vista de forma minimalista no supondría una inversión mayor de tiempo.

4.4. Despliegue de la aplicación

La aplicación se encuentra en los servidores del Departamento de Ciencias de la Computación de la Universidad de Chile.

El backend en Django está configurado con la opción *DEBUG=True*, cuya principal implicancia es que al tener un error o excepción en el servidor, éste se desplegará con el seguimiento completo del código involucrado en la falla (i.e. *traceback*). Entregar dicha información al cliente es innecesario y genera riesgos de seguridad en la aplicación al informar sobre el funcionamiento y estructura interna del código y servidor. También este traceback incluye todas las variables de entorno y configuración de Django que, además de ser información interna, podrían contener datos sensibles como correos de administración y datos de encriptación.

Un beneficio de este modo DEBUG es que automáticamente se sirven los archivos multimedia, tanto estáticos del sitio como subidos por los usuarios, sin necesidad de mayores configuraciones en el servidor. Sin embargo, esta forma no es eficiente ni segura, por lo que no es apropiada para un ambiente en producción [24] y debe replantearse a futuro.

La razón para desplegar el backend en esta configuración es que para esta etapa del proyecto, en la que no existen datos reales y aún se están probando funcionalidades, el riesgo de una eventual vulnerabilidad y las posibles consecuencias de ella no justificarían la inversión de tiempo necesaria para aprender y configurar un escenario más robusto, además de las facilidades que dicho modo entrega para probar y cambiar cosas rápidamente.

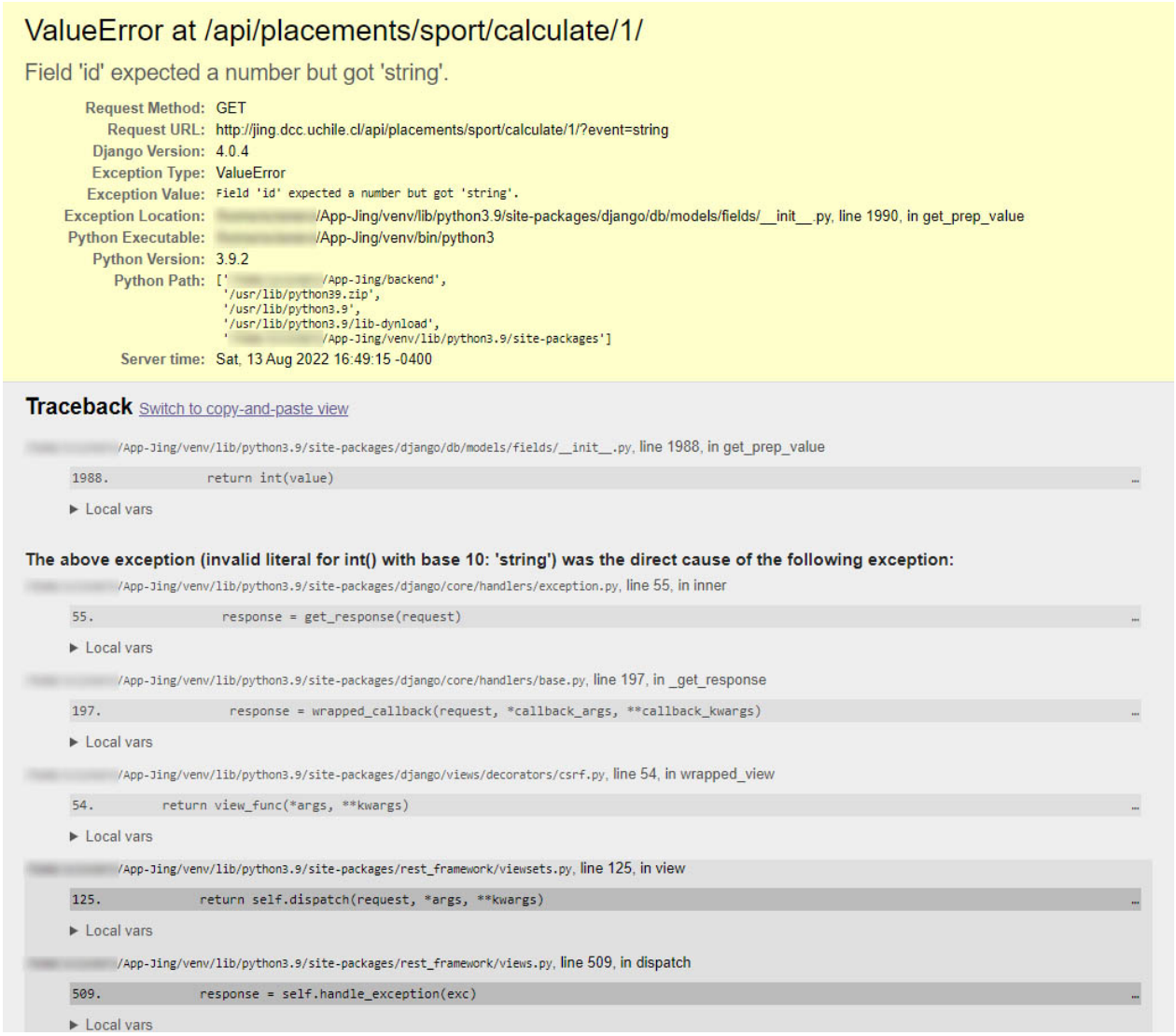

Figura 4.10: Error en Django usando DEBUG=True

Settings Using settings module backend.settings

Setting	Value
ABSOLUTE_URL_OVERRIDES	{}
ADMINS	[]
ALLOWED_HOSTS	['*']
APPEND_SLASH	True
AUTHENTICATION_BACKENDS	['django.contrib.auth.backends.ModelBackend']
AUTH_PASSWORD_VALIDATORS	'********************'
AUTH_USER_MODEL	'auth.User'
BASE_DIR	PosixPath('[redacted]/App-Jing/backend')
CACHES	{'default': {'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'}}
CACHE_MIDDLEWARE_ALIAS	'default'
CACHE_MIDDLEWARE_KEY_PREFIX	'********************'
CACHE_MIDDLEWARE_SECONDS	600
CORS_ORIGIN_WHITELIST	['http://localhost:3000']
CSRF_COOKIE_AGE	31449600
CSRF_COOKIE_DOMAIN	None
CSRF_COOKIE_HTTPONLY	False
CSRF_COOKIE_NAME	'csrftoken'
CSRF_COOKIE_PATH	'/'
CSRF_COOKIE_SAMESITE	'Lax'
CSRF_COOKIE_SECURE	False
CSRF_FAILURE_VIEW	'django.views.csrf.csrf_failure'
CSRF_HEADER_NAME	'HTTP_X_CSRFTOKEN'
CSRF_TRUSTED_ORIGINS	[]
CSRF_USE_SESSIONS	False
DATABASES	{'default': {'ATOMIC_REQUESTS': False, 'AUTOCOMMIT': True, 'CONN_MAX_AGE': 0, 'ENGINE': 'django.db.backends.sqlite3', 'HOST': '', 'NAME': PosixPath('[redacted]/App-Jing/backend/db.sqlite3'), 'OPTIONS': {}, 'PASSWORD': '********************', 'PORT': '', 'TEST': {'CHARSET': None, 'COLLATION': None, 'MIGRATE': True, 'MIRROR': None, 'NAME': None}, 'TIME_ZONE': None, 'USER': ''}}
DATABASE_ROUTERS	[]
DATA_UPLOAD_MAX_MEMORY_SIZE	2621440
DATA_UPLOAD_MAX_NUMBER_FIELDS	1000
DATETIME_FORMAT	'N j, Y, P'
DATETIME_INPUT_FORMATS	['%Y-%m-%d %H:%M:%S',

Figura 4.11: Variables de configuración en Django usando DEBUG=True

Por el lado del frontend en React, también se encuentra en un estado de desarrollo en vez de producción, sin embargo éste es menos trascendental en términos de seguridad. La principal diferencia es que en una estructura o *build* de producción, los archivos necesarios se minifican y ofuscan para compactarse lo más posible y se envía una versión final optimizada al cliente. Esto por supuesto hace la navegación más eficiente en términos de tiempo y espacio. Por otro lado, el build de desarrollo, aunque menos eficiente, facilita el debugging manteniendo los nombres semánticos de las componentes y revelando más información interna al explorador, además de permitir un refresco automático del sitio cada vez que se detectan cambios en el código, función conocida como *hot reloading*.

Por las mismas razones expuestas al hablar del despliegue del backend, por defecto se mantiene el frontend en este modo de desarrollo. De ser necesario, generar y servir un build de producción con la versión más reciente de la aplicación puede hacerse fácilmente en un par de minutos.

Los enlaces para acceder al sitio son los siguientes:

- Aplicación frontend, que dirige al puerto 3000 del servidor:
  `https://jing.dcc.uchile.cl/`
- API backend, que dirige al puerto 8000 del servidor:
  `https://jing.dcc.uchile.cl/api/`

Cabe mencionar que actualmente el despliegue no cuenta con un auto-reinicio adecuado y los servidores del DCC a veces tienen reinicios o caídas, por lo que los enlaces podrían no estar disponibles en algunas ocasiones. Se procurará estabilizar esto luego de entregar este informe para garantizar que la aplicación pueda ser revisada de ser necesario.

Capítulo 5

Validación

Para procurar y comprobar el funcionamiento correcto de la aplicación, se siguieron metodologías de desarrollo y verificación interna para validar la integridad del sistema.

Por temas de tiempo y mala planificación en el desarrollo del frontend, no fue posible tenerlo estable oportunamente para poder validar la experiencia de usuario, sin embargo, se diseñaron metodológicamente las pruebas necesarias para realizar esta validación en el futuro cercano.

En este capítulo se mencionarán los métodos usados para procurar un funcionamiento correcto del servicio, en particular del backend, y se presentará la planificación de las pruebas de usabilidad que no lograron concretarse.

5.1. Tests unitarios

Estos tests se separaron y enfocaron en cada *app* de Django, probando las funcionalidades esperadas de cada una.

El caso base para cada conjunto de tests fue cubrir las cuatro operaciones de un sistema CRUD, es decir, creación, lectura, edición y borrado. Dado que toda la funcionalidad de la API se encuentra implementada en ViewSets, era esperable que todos los modelos contaran con estas cuatro operaciones básicas a través de la API.

Sobre esa base, se agregan las acciones particulares esperadas para cada ViewSet, como por ejemplo, las funciones `calculate` de SportStandingViewSet y EventStandingViewSet, cuya función es calcular los puntajes de un deporte o evento, según los partidos o competiciones ganadas y su respectiva asignación de puntaje.

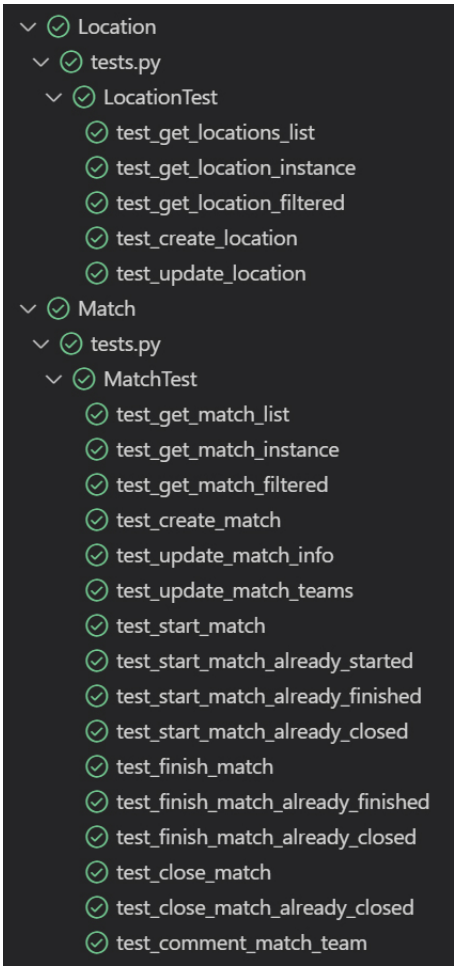

Figura 5.1: Ejemplos de tests básicos de Location y de funciones de Match

Cada conjunto de tests se construye como una clase que extiende la clase APITestCase, de Django REST Framework, que a su vez extiende TestCase, de Django, y entrega facilidades para realizar los tests. Las herramientas más importantes que otorga son:

- Poder realizar consultas locales al backend a través de un cliente virtual.
- Instalar en una base de datos temporal los datos de prueba contenidos en fixtures, que se reinician entre cada test y se eliminarán al finalizar el conjunto.
- Funciones de aserción o validación que permiten comparar los datos esperados con los recibidos, y definir la aprobación de cada test en base a ellas.

Recorriendo el Código 5.1, se observa que inicialmente se define el fixture que incluye un conjunto de datos de prueba minimalista generado manualmente, con lo necesario para probar las funcionalidades unitarias. Luego, en la función `setUp`, se establece un inicio de sesión utilizando el cliente de testing. Esta sesión se usará para todos los tests de esta clase.

Luego, se define el primer test, llamado `test_get_locations_list`, cuyo objetivo es probar la consulta de lectura de múltiples datos a modo de lista. Se describe manualmente el

resultado esperado de la consulta, que corresponde a los 2 Locations incluidos en el fixture. Luego, en la línea 23, se realiza la consulta de tipo GET la dirección `/api/locations/`, y posteriormente se valida que la respuesta tenga un código HTTP 200 (i.e. "OK") y que el contenido de la respuesta sea igual al contenido esperado. Si se cumplen estas dos condiciones, entonces el test se declara aprobado.

El segundo test del fragmento, `test_get_location_instance`, sigue el mismo protocolo de definir resultado esperado, hacer la consulta y verificar la respuesta. En este caso se consulta la dirección $/api/locations/1/$, que debiera retornar la Location con $id = 1$.

Otro ejemplo relevante se observa en el Código 5.2, que intenta iniciar el partido con $id = 3$, sabiendo que este partido ya está iniciado, por lo que el resultado esperado es que la consulte falle, que la respuesta contenga el código de error `already_started`, y que el código HTTP de la respuesta sea 409 (i.e. "Conflict", que indica que la solicitud no es incorrecta per se, pero entra en conflicto con el estado actual del objetivo).

De forma similar se arman el resto de los tests. El total de tests del sistema es de 75, repartidos entre las distintas apps de Django y módulos, como se indica en la Tabla 5.1.

Para obtener la cobertura de los tests se utilizó la librería *coverage.py*, que se integra fácilmente con la suite de testing de Django [25, 26], entregando reportes de cobertura de código. Para efectos de este reporte, se omitieron los archivos de migraciones y otras configuraciones de Django, además de los archivos de tests mismos. La cobertura final entregada es del 92 %, con falencias principalmente en el código de cálculo de puntajes, que fueron las últimas añadidas en el proyecto y para las que no se siguió una metodología TDD tan estricta.

```python
class LocationTest(APITestCase):
    fixtures = ['test_fixture.json']

    def setUp(self):
        self.client.login(username='scisneros', password='123')

    def test_get_locations_list(self):
        url = '/api/locations/'
        expected = [
            {
                "id": 1,
                "name": "Cancha de Tenis",
                "address": "Linden Street 2189",
                "university": 1
            },
            {
                "id": 2,
                "name": "Piscina",
                "address": "Cedar Lane 4248",
                "university": 2
            },
        ]
        response = self.client.get(url, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertJSONEqual(response.content, expected)

    def test_get_location_instance(self):
        url = '/api/locations/1/'
        expected = {
            "id": 1,
            "name": "La Cancha",
            "address": "qwe 123",
            "university": 1
        }
        response = self.client.get(url, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertJSONEqual(response.content, expected)
```

Código 5.1: Ejemplo de tests para obtener Locations.

```python
def test_start_match_already_started(self):
        url = '/api/matches/3/start/'
        expected_code = 'already_started'
        response = self.client.post(url, format='json')

        self.assertEqual(response.status_code, status.HTTP_409_CONFLICT)
        self.assertEqual(response.data['code'], expected_code)
```

Código 5.2: Ejemplo de test para iniciar un Match con error.

module	*tests*	*statements*	*missing*	*coverage*
Administration	4	34	1	97 %
Authentication	0	19	0	100 %
backend	0	34	0	100 %
Event	4	28	0	100 %
Location	5	23	1	96 %
Match	18	234	16	94 %
Message	4	29	1	97 %
News	8	45	2	96 %
pagination	0	6	0	100 %
Person	6	81	2	98 %
serializers	0	20	5	80 %
Sport	4	49	1	98 %
Standing	10	240	51	82 %
Team	6	89	6	94 %
University	4	42	2	95 %
utils	2	19	0	100 %
Total	**75**	**992**	**88**	**92 %**

Tabla 5.1: Reporte sobre cobertura de tests unitarios

5.2. Generación de datos de prueba

Durante las entrevistas realizadas en la etapa de análisis, se conversó la posibilidad de obtener datos reales de alguna iteración anterior de los JING. La entrevistada accedió a esto y se hizo el acuerdo de que eventualmente coordinaría con el memorista para definir qué datos conseguir. Sin embargo, debido a retrasos de ambas partes e indisponibilidades de agenda, a la fecha de este informe no fue posible obtener los datos.

Inicialmente en la implementación se generaron algunos datos aleatorios sencillos para poder ir probando informalmente la plataforma. Luego de evaluar que era posible que no se recibieran los datos reales de los JING, se decidió expandir estos datos aleatorios generados para simular de mejor manera la magnitud y condiciones de un evento real, con el fin de usarlos más formalmente para probar tanto la integridad de la plataforma como la capacidad de carga y velocidad de las consultas.

Para ingresar la data se utilizó el sistema de fixtures de Django. Para generar los fixtures se confeccionó un programa sencillo en Python, con un algoritmo "ingenuo" y secuencial. Se investigaron alternativas más formales para la generación de datos ficticios en Django, como las librerías *model_bakery*[1], *Factory Boy*[2] y *faker*[3], pero considerando que la generación de esta data sería algo puntual para efectos de validación y no necesariamente mantenible

[1] `https://model-bakery.readthedocs.io/en/latest/`
[2] `https://factoryboy.readthedocs.io/en/latest/`
[3] `https://faker.readthedocs.io/en/master/`

ni eficiente, y que ya existía un código que generaba algunos modelos, además de ya haber retrasos importantes en el tiempo de implementación, se descartó la necesidad de hacer un mecanismo más robusto, con el tiempo de aprendizaje que ello implicaría, y se mantuvo el acercamiento simplista del algoritmo.

El programa generador construye la data en el formato de fixtures usando diccionarios de Python, y luego la imprime en un archivo `.json` que podrá ser consumido por Django. La estructura más sencilla y menos condicional de generar son los Events, como se aprecia en el Código 5.3. El resto de los modelos sigue el mismo patrón con distintas complejidades, haciendo uso de la librería `random` para generar data aleatoria y relacionar entidades aleatorias entre sí.

Cada modelo tiene sus propias condiciones de generación para asimilar lo más posible las condiciones de un dataset real. Por ejemplo, al generar las relaciones entre un Match y Team, se procura que todos los Teams que compitan sean de instituciones distintas, que un partido no jugado no pueda tener puntajes, o que un equipo que se ausentó no haya marcado puntos, entre varias otras condiciones. También se asigna una probabilidad de provocar que ciertos equipos no asistan, con el fin de procurar que hayan deportes en los que una institución no participó. Todas estas condiciones se aplican ineficientemente de forma iterativa y/o por fuerza bruta.

Recopilando información de las redes sociales de los JING, junto con lo conversado en las entrevistas, se consideraron los siguientes números:

- ∼30 deportes (e.g. Handball), con ∼60 subdivisiones (e.g. Handball Damas).
- 8 instituciones participantes.
- Entre 8 y 16 equipos por cada división.
- Entre 12 y 24 partidos en fases de grupos.
- Entre 4 y 16 partidos en fases eliminatorias.

Con esos números, se calcularon los siguientes estimados:

- ∼2.000 partidos por evento.
- ∼500 equipos por evento.
- ∼2.000 personas por evento.

Se decidió simular 10 eventos, para generar un número considerable de registros que emulara 10 años de uso anual para los JING. Finalmente, se generaron aproximadamente 20.000 partidos, 10.000 personas, 4.500 equipos, 130 locaciones, 60 deportes y 8 universidades, junto con los registros relaciones, que rondan entre las 14.000 relaciones jugador-equipo, 60.000 relaciones partido-equipo y 80 universidad-evento.

Las universidades participantes y los nombres y tipos de deportes se obtuvieron de iteraciones pasadas de los JING. Para los nombres, apellidos y direcciones, se descargaron listados aleatorios del sitio `www.randomlists.com`.

```python
data = []

# Events
data_events = []
for i in range(10):
    event = {
        "model": "Event.event",
        "pk": i+1,
        "fields": {
            "name": f"JING {2013+i}",
            "year": 2013+i,
            "logo": ""
        }
    }
    data_events.append(event)
data.extend(data_events)
```

Código 5.3: Generador de fixture para Event.

5.3. Planificación de validación con usuarios

Dentro de los planes estaba realizar una validación de usuarios con pruebas de usabilidad, sin embargo, por falta de tiempo y retrasos en el inicio de la implementación del frontend, no fue factible llevarlas a cabo. Originalmente se planificó realizar pruebas con al menos 5 usuarios, según los estándares definidos por Nielsen-Landauer [27]. Ya se contaba con la predisposición para participar de las 2 organizadoras entrevistadas, además de 2 personas que han organizado los juegos La Mona, y al menos 1 persona deportista estudiantil. Se pretendía darles un conjunto de tareas a realizar en la plataforma, con el método *thinking aloud testing* [28], o de pensamiento en voz alta, donde se pide a los usuarios describir en tiempo real lo que van pensando al usar el sitio, siendo facilitados por las técnicas de conversación descritas por Pernice [29]. Luego, se les pediría responder la encuesta estandarizada *System Usability Scale (SUS)* [30], que entrega ciertas métricas cuantitativas exploratorias respecto al nivel de usabilidad del sistema, además de cualquier retroalimentación cualitativa que ofrezcan los usuarios.

Suponiendo que se tendrían suficientes sujetos de prueba, se definieron conjuntos de tareas diferenciadas para personas que tienen experiencia en organización deportiva y personas que no. Estas tareas atómicas buscarían replicar casos de uso de un usuario real y simular una navegación natural. Los conjuntos de posibles tareas secuenciales que se planearon preliminarmente son los siguientes:

Usuarios no-organizadores:

- Averigua cuándo y dónde es el siguiente partido que te corresponde jugar.
- Averigua en qué lugar es el siguiente partido de Tenis Varones donde juegue la Universidad de Concepción.
- Averigua cómo llegar a ese lugar.

- Averigua qué institución va ganando el deporte Tenis Varones.
- Ve cuántos goles hubo en algún partido de Rocket League que ya se haya jugado.
- Encuentra qué universidad ganó los JING 2018.
- Revisa tus últimos mensajes recibidos.
- Averigua si recientemente han habido cambios de horario para algún juego.

Usuarios organizadores:

- Mismas tareas de usuarios no-organizadores.
- Crea un partido de Natación 200m Mixto, en la Piscina A, para un día y hora específicos. En este partido deben competir 4 instituciones distintas.
- Marcar que finalizó el partido de Fútbol Varones de cierto día y hora específicos, donde ganó 3 a 1 la Universidad de Santiago.
- Revisa las objeciones levantadas respecto a los resultados del partido de Vóleibol Mixto de cierta fecha y hora. Acepta las objeciones y cierra el partido con el puntaje actualizado.
- Envía un mensaje a todos los capitanes de Vóleibol Mixto, informando la actualización.
- Corrige la numeración en la dirección de la sede. Debería ser 851 en vez de 850.

Para la confección de las tareas se consideraron los lineamientos de Schade [31]. Los deportes y lugares específicos se mencionan para reducirle al usuario la presión de elegir sobre datos que aún no conoce. Intencionalmente se evita lo más posible usar palabras que estén en la interfaz. Por simplicidad, para organizadores se omiten intencionalmente algunas tareas de creación/edición de entidades, que tienen un flujo muy similar a las otras.

5.4. Discusión

La creación de endpoints para cada modelo importante abre puntos de input/output que requieren estabilidad, coherencia y resiliencia a lo largo del desarrollo del proyecto, por lo que fue crucial tener la validación constante que entrega una metodología TDD. Los tests permitieron revelar inconsistencias generadas antes de que llegaran a la capa de frontend, que generalmente implica un debugging más costoso.

Gracias a la generación de datos aleatorios se logró detectar dos consultas relevantes que tenían un tiempo de carga inesperado. La primera era la consulta para obtener todos los partidos en los que participara un subconjunto de instituciones. Por ejemplo, los partidos en los que participaran **al menos** la Universidad de Valparaíso y la Universidad Autónoma. Inicialmente se implementó esta consulta como una cadena de `.filter()` que se construía iterativamente con la lista de instituciones, generando filtros de este estilo:

```
queryset.filter(university=id1).filter(university=id2).filter(...
```

donde `idX` son los id de cada institución. Esto funcionaba correctamente con los datos minimalistas, pero al utilizar porte sobre coberturla data aleatoria masiva y hacer una consulta de 4 o más instituciones, el servidor se quedaba procesando por un tiempo indeterminado. Se desconoce exactamente por qué ocurría esto, pero obligó a cambiar el acercamiento a esta consulta. Finalmente se utilizaron agregaciones que hicieron la consulta más escalable, quedando similar a la siguiente forma:

```
queryset.filter(university__in=id_list).annotate(
  num_participants=Count('university')).filter(num_participants=len(id_list))
```

Otra consulta que aumentaba considerablemente su tiempo de ejecución al agregar más datos era el cálculo de puntaje total de un evento. Aún cuando demoraba aproximadamente 4 segundos y no era un tiempo crítico, se revisó la implementación y se descubrieron operaciones muy lentas en torno a la creación de muchos serializadores individuales para cada deporte, en lugar de un serializador de lista que manejara todo a la vez. Luego de refactorizar esa sección, el tiempo de consulta se redujo a aproximadamente 60 milisegundos.

Por el lado del frontend, al tener data realista se pudo ajustar los tamaños de las tablas y la cantidad de información mostrada en ellas para evitar contaminación visual y disminuir la carga cognitiva [32]. Al mismo tiempo, se pudo cuantificar de mejor manera los tiempos de respuesta de las acciones al tener una cantidad de data similar a la que habría en un evento real. Esto es muy importante tomando en cuenta que la reestructuración hacia un sitio tipo Single-Page App implica consultas en prácticamente cada acción del usuario, por lo que tener control de los tiempos de respuesta es crucial para una experiencia fluida en un sistema con alto volumen de información.

El punto de este análisis es resaltar que estos y otros problemas no se hubieran podido detectar usando datos de prueba sencillos o en cantidades poco comparables a las del mundo real. Si bien lo ideal hubiera sido contar con data efectivamente real, la data generada se constituyó como una fuente importante de validación interna.

Respecto a la validación con usuarios, por supuesto su ausencia impide desprender información valiosa respecto de la usabilidad del frontend. No obstante, la confección de tareas de prueba y la necesidad de considerar casos de uso realistas para su elaboración, permitió tener una percepción distinta sobre el flujo y navegación del sitio. Una consecuencia relevante de este análisis centrado en el usuario, es que se logró detectar que muchas de las acciones atómicas de búsqueda de información están conectadas, como por ejemplo, al buscar mis partidos es muy probable que inmediatamente después necesite ver la ubicación del lugar de juego, o que al revisar los resultados finales de un deporte, querría ver los partidos y marcadores individuales de ese deporte. De esta forma, se diagnostica la necesidad de interconectar mejor las distintas secciones, permitiendo al usuario saltarse la verticalidad de la jerarquía del sitio para navegar entre información relacionada.

Como conclusión de este capítulo, se logra apreciar que las metodologías de validación interna fueron fructíferas y beneficiosas para propender a la construcción de un sistema

robusto y en línea con los cambios y mejoras planteadas en el análisis del problema. Sin perjuicio de esto, se reconoce como falencia importante en la realización de este proyecto, la ausencia de una validación externa que permita confirmar estos planteamientos e hipótesis. Si bien el sistema está construido sobre un esquema de diagnósticos, estándares y metodologías que conceden cierta certeza en la persecución del objetivo, es difícil confirmar su validez sin una perspectiva fuera de este esquema.

Capítulo 6

Conclusiones

Para concluir este informe, se hará una evaluación del trabajo realizado, contrastándolo con los objetivos y soluciones propuestas al inicio del proyecto. Posteriormente se explorarán formas en que este trabajo podría ser continuado a futuro para mejorar el sistema y que eventualmente sea utilizable. Para finalizar, se elaborarán reflexiones y aprendizajes técnicos y personales, obtenidos en la experiencia de este trabajo de título.

6.1. Cumplimiento de objetivos

Contrastando el trabajo expuesto en este informe con los objetivos específicos planteados en la Subsección 1.4.2, se detecta que el objetivo número 1 no se logró cumplir a cabalidad. Aunque sí se logró dotar de mayor estabilidad a la plataforma y se logró ejecutar el sitio nuevamente arreglando los problemas que tenía para inicializarse, detectados en una exploración profunda de su estado, no se cumplió la propuesta de completar el sistema para alcanzar la versión mínima viable del desarrollo inicial. La falta de un sistema de mensajería, de una distinción funcional de los roles y de un flujo de uso claro para la plataforma, la dejan en un estado no viable para ser usada en un contexto real.

El objetivo específico 2 se considera cumplido. Al haber realizado diagnósticos del desarrollo y conciliarlos con la información obtenida a través de las entrevistas y análisis de usabilidad, se logró reconstruir una base con sustento metodológico de los requisitos y necesidades de los usuarios, y se pudo vislumbrar de mejor manera las intenciones del primer desarrollo. Este informe de memoria sirve como recopilación de las conclusiones obtenidas y puede usarse como hoja de ruta para dar continuidad al proyecto, en contraste con la inexistencia de lineamientos y direcciones en el estado anterior.

Para el objetivo 3, sí se rediseñó y se reimplementó por completo la interfaz de usuario, con un mayor enfoque en la usabilidad y con la investigación teórica y metodológica adecuada. Sin embargo, a pesar de tener una base teórica más robusta, no es posible confirmar o rechazar con certeza el cumplimiento de este objetivo, dado que no se logró realizar las validaciones de usabilidad con usuarios reales.

Finalmente, el objetivo 4 sí se cumple, al tener desplegado el sitio públicamente en los servidores del DCC.

En resumen, contrastando los cumplimientos específicos con el objetivo general, éste se puede declarar como parcialmente cumplido.

Se reconoce que el planteamiento del objetivo 1 pudo haber sido más granular, separándolo en una etapa exploratoria, una de estabilización y una de finalización. De esta forma, hubiera quedado más claro el cumplimiento de cada etapa, en lugar de juntar varios procesos en un solo objetivo, que finalmente no queda claro hasta que nivel se cumplió.

6.2. Problemas solucionados

Respecto a los problemas presentes en el software original que fueron solucionados con el trabajo realizado, se mencionan los siguientes avances a nivel abstracto:

1. El proyecto se encontraba en un estado de abandono. No existía documentación de ningún tipo sobre el código, no había sostén para las decisiones tomadas, no había una hoja de ruta para continuar el proyecto y la relación con la contraparte se vio perjudicada debido al incumplimiento de los compromisos. Todo esto hacía muy difícil darle continuidad al proyecto, y eventualmente, cuando las contrapartes dejaran el contexto universitario, se perdería toda posibilidad de retomarlo. El trabajo realizado en esta memoria permitió darle una segunda oportunidad al proyecto, y si bien no alcanzó a finalizarse como esperaba, sienta las bases para ser abordado nuevamente con mayor facilidad.

2. Se actualizó el proyecto a estándares de la industria más apropiados para el tipo de plataforma que se espera. Se introdujeron tecnologías que mejoran tanto la experiencia de usuario como la mantenibilidad del código. El código anterior, al ser HTML plano con múltiples clases CSS utilitarias en lugar de semánticas, hacía compleja la legibilidad y dificultaba mantener un estilo coherente a lo largo del sitio, teniendo que agregar manualmente el conjunto de clases necesarias y precisas para cada caso. La arquitectura actual por componentes con nombres semánticos permite un mejor entendimiento a primera vista de las estructuras, y facilita la reutilización de funcionalidad. Por ejemplo, todas las páginas de tablas se construyen con un mismo componente, permitiendo reusar operaciones complejas como lo es realizar consultas en tiempo real al cambiar cada filtro.

3. Estas tecnologías aportan también a implementar más fácilmente elementos de interfaz que mejoren la experiencia de los usuarios. Por ejemplo, el sistema de alertas puede ser importado y llamado desde cualquier parte del sitio, sin necesidad de repetir código o implementar alertas ad hoc a cada página. Este mismo acercamiento puede replicarse para otros elementos, como ventanas modales e indicadores de carga.

4. La división entre frontend y backend separa completamente las responsabilidades, permitiendo que, en adelante, el grueso de las modificaciones se realicen solo en el frontend, mientras que el núcleo del backend se mantiene relativamente estable.

5. La reestructuración del modelo de datos permite proyectar funcionalidades que previamente no eran posibles. Al estar todo muy acoplado en el sistema original, se hacía difícil pensar en cada entidad como un objeto independiente y conjugable con otras partes. La independización de entidades en el sistema actual no solo ayuda a reducir la repetición de datos y el acoplamiento, sino que abre las puertas a funcionalidades futuras, como por ejemplo, el cálculo de estadísticas de una persona/deporte/universidad a lo largo de varios eventos.

6.3. Trabajo futuro

Dado el tiempo acotado de desarrollo y los contratiempos, aún hay varias secciones del proyecto que se encuentran inestables o con soluciones temporales sub-óptimas. Es la intención del memorista continuar trabajando en él luego de finalizar la memoria, en coordinación con la profesora guía, para estabilizarlo lo más posible y dejarlo en un estado abordable por un/a eventual siguiente desarrollador/a.

Aparte de esa estabilización, se consideran las siguientes tareas mínimas para poder utilizar la plataforma en una competencia real, que son básicamente trabajo planificado para esta memoria, pero que no alcanzaron a realizarse.

1. **Validación en entorno real.** Validar con usuarios organizadores, data de competencias pasadas y con flujos completos. Se puede aprovechar la próxima instancia ya confirmada de los JING 2022 en octubre para armonizar el sistema con el mundo real.

2. **Roles en frontend.** Dado que actualmente no hay una diferenciación entre los distintos usuarios a nivel de interfaz y de permisos, todos los usuarios podrían acceder a todas las funcionalidades, lo que por supuesto hace la página inutilizable en una competencia.

3. **Carga de data masiva.** La magnitud de información que se debe cargar al sistema al inicio de cada evento, entre miles de personas, partidos, equipos y sus relaciones, hace inviable que se haga de manera individual, siendo necesario un mecanismo de carga masiva.

4. **Robustez en validación de datos.** La alta flexibilidad que entrega una estructura de API REST conlleva también la apertura de varios puntos de entrada y salida de información. Esto, junto al desacoplamiento de entidades que se efectuó a nivel de datos, puede comprometer la integridad de la data, y las validaciones y medidas de seguridad actuales no satisfacen un nivel apropiado de certeza y resiliencia.

5. **Interfaz de mensajería.** Era una de las principales funciones esperadas por las usuarias entrevistadas, debido a que el problema de la comunicación sobresaturada era uno de los más engorrosos. A nivel de backend, existe la capacidad de obtener y generar mensajes con la API, por lo que la principal carencia es a nivel de interfaz.

Las siguientes características son no-críticas, pero fueron requeridas o propuestas por las usuarias durante el desarrollo:

1. **Registro de asistencia por QR.** Esta funcionalidad estuvo en los planes desde la aplicación original, pero por temas de tiempo en ninguna de las dos se ha podido implementar. La idea es poder registrar presencialmente la asistencia de cada persona, usando códigos QR desde sus celulares, como se menciona en las entrevistas. Aún siendo una petición explícita de las usuarias, el análisis de las entrevistas y casos de uso da a entender que ésta no es realmente una funcionalidad necesaria para la plataforma (ver Subsección 3.2.2).

2. **Interfaz de cambios recientes.** Existe un registro en el backend de los cambios que hace cada usuario, pero la única forma de ver las filas es a través de la página de administración de Django. Tener una vista consultable en el frontend con esta información aportaría enormemente a la transparencia y responsabilidad de la organización.

Finalmente, se plantean las siguientes propuestas que no son necesarias ni requeridas hasta el momento, pero que podrían mejorar la experiencia en el sitio y ser desafíos técnicos interesantes:

1. **Interfaz orientada a dispositivos móviles.** El usar React ya entrega una responsividad base a la plataforma, pero ésta está completamente pensada para una vista en escritorio. Es muy probable que el sistema se use en terreno durante las competencias, por lo que se beneficiaría considerablemente de un diseño optimizado y enfocado para teléfonos y tablets.

2. **Fase de grupos. Llaves de eliminatoria.** La plataforma tiene como objetivo llevar un registro de los partidos individuales y sus resultados generales, pero no tiene gran capacidad de llevar un seguimiento más detallado de las distintas etapas del campeonato, por ejemplo, una posible fase de grupos y eliminatorias, con sus respectivas interfaces que permitan visualizar el progreso de las mismas. Probablemente la funcionalidad más grande que se propone.

3. **Perfiles y estadísticas.** La reestructuración del modelo de datos le da independencia a cada entidad respecto a los eventos. Esto permite que sea posible hacer agregaciones de estadísticas generales de personas, equipos, deportes e instituciones a lo largo de distintos eventos, y construir páginas de perfil asociadas a cada uno.

4. **Estética de la interfaz.** Los estilos de Bootstrap permiten una estética basal apropiada, pero el sitio no cuenta con ninguna estilización ni paleta particular, llegando a ser inconsistente en algunas cosas y pudiendo afectar su usabilidad, además de la ausencia de una imagen institucional representativa.

5. **Optimización de base de datos.** Si bien los tiempos de respuesta actuales son bastante aceptables, el memorista no contaba con mucha experiencia en bases de datos, por lo que estudiar la optimización tanto de las consultas como de las tecnologías de la base de datos podría ser útil para la escalabilidad del proyecto, considerando que su uso es puntualmente intenso en un lapso de 3-4 días de competencia.

6.4. Reflexiones finales

Habiendo entregado las conclusiones técnicas respecto a este trabajo de título, se procederá en esta última sección a expresar reflexiones personales relacionadas a la labor como ingeniero de software.

En un ámbito metodológico, se destaca la utilización de estrategias como Test-Driven Development al desarrollar. Tiene sus propias complejidades y requiere una claridad temprana del esquema completo del proyecto para aplicarse, lo que lo hace difícil de implementar en un proyecto que parte desde cero, pero en situaciones como esta, en que el dominio funcional ya estaba relativamente establecido, fue una herramienta útil para construir un sistema más robusto, incluso cuando se usó de forma menos estricta.

Así mismo, también se destaca la aplicación más metódica de técnicas de usabilidad y estudios de usuarios. Gran parte de la bibliografía de este informe trata sobre estos temas, ya que se dedicó bastante tiempo a la investigación al respecto. Aplicar estas técnicas no solo procura mejores resultados en la ejecución, sino que el solo hecho de aprender sobre ellas y su teoría entrega al desarrollador perspectivas sobre el usuario que difícilmente puede obtener en sus estudios regulares, y que le permiten hacer mejores estimaciones fundamentadas.

En una lectura más reflexiva, un trabajo de título de este estilo tiene muchas aristas distintas involucradas, en un tiempo relativamente corto. Al requerir una etapa de análisis, recopilación de información, confección de modelo de datos, implementación del backend, diseño de interfaces, implementación del frontend, despliegue de la aplicación y validación del sistema, en un tiempo equivalente a tres meses de trabajo full-time, se hace crítico no solo tener una muy buena capacidad de planificación, sino más importante, tener la capacidad de ser realista con las expectativas. Es fácil dejarse llevar por la motivación de resolver el problema, al punto de abarcar más de lo factible. En ese sentido, se considera que este proyecto pudo haberse beneficiado de haber usado metodologías más iterativas y más ágiles, en lugar de intentar una reestructuración completa y una versión final en una misma iteración larga. Se subestimaron los tiempos de una migración tecnológica y todo el aprendizaje que ello conlleva.

La concreción del trabajo puede variar a medida que se obtiene información. Hacer planificaciones a largo plazo sin una exploración adecuada conlleva casi inevitablemente a resultados insatisfactorios, o a redireccionar a medio camino. Por esta razón es que en este informe inicialmente se plantea una solución preliminar en abstracto, que se logra concretar solo después de haber hecho las recolecciones y análisis pertinentes. Aún así, la ejecución terminó siendo más compleja de lo esperado.

Por otro lado, se desprende que cada arista del proyecto tiene sus pros y contras, y no hay solo una decisión correcta al momento de desarrollar. En el caso de este trabajo, al dar enfoque a la robustez y mantenibilidad, se perdió mucho tiempo de implementación funcional, lo que impidió tener el producto esperado. Quizás se podría haber solo reparado y construido sobre el proyecto original, pero eso hubiera traído sus propios problemas, como un producto inestable y con baja mantenibilidad futura, que fue justamente una de las razones que motivó este trabajo.

Por esto y como se menciona en la sección de trabajo realizado, a pesar de no lograr el objetivo de generar una plataforma utilizable, se considera que este trabajo fue un éxito en el objetivo de recuperar un proyecto de difícil continuidad, al darle más enfoque a los ámbitos previamente postergados y abriendo la posibilidad de construir sobre ellos.

Bibliografía

[1] *Juegos Deportivos de Ingeniería (JING) 2019*. https://www.ing.uc.cl/eventos/jing-2019/, visitado el 2021-10-06.

[2] *JING*, Agosto 2016. https://www.facebook.com/JuegosDeportivosIngenieria/, visitado el 2021-10-06.

[3] Miranda, Pablo, Zambra, Javier y Simmonds, Jocelyn: *App-Jing*. https://github.com/jsimmond/App-Jing, 2019.

[4] *Templates — Django documentation — Django*. https://docs.djangoproject.com/en/3.2/topics/templates/.

[5] *Material Design for Bootstrap*, 2013. https://mdbootstrap.com/, visitado el 2021-12-20.

[6] *DoLeague*. https://www.doleague.com/, visitado el 2021-12-10.

[7] *Tournify*. https://www.tournify.uk/, visitado el 2021-12-10.

[8] *Playinga*. https://playinga.com/, visitado el 2021-12-10.

[9] *Competize*. https://www.competize.com/, visitado el 2021-12-10.

[10] Makai, Matt: *Object-relational Mappers (ORMs)*. https://www.fullstackpython.com/object-relational-mappers-orms.html, visitado el 2022-10-10.

[11] Gupta, Lokesh: *What is REST – Learn to create timeless REST APIs*, Abril 2022. https://restfulapi.net/, visitado el 2022-10-10.

[12] Services, Amazon Web: *What is an API? - API Beginner's Guide - AWS*. https://aws.amazon.com/what-is/api/, visitado el 2022-10-10.

[13] Christie, Tom: *Home - Django REST framework*, 2011. https://www.django-rest-framework.org/, visitado el 2021-12-19.

[14] Team, Codecademy: *What is CRUD?* https://www.codecademy.com/article/what-is-crud, visitado el 2022-10-10.

[15] Facebook: *React – A JavaScript library for building user interfaces*, 2019. https://reactjs.org/, visitado el 2021-12-19.

[16] *Bootstrap*, 2000. https://getbootstrap.com/, visitado el 2021-12-20.

[17] Pernice, Kara: *User Interviews: How, When, and Why to Conduct Them*, Octubre 2018. https://www.nngroup.com/articles/user-interviews/, visitado el 2021-12-14.

[18] Nielsen, Jakob: *Interviewing Users*, Julio 2010. https://www.nngroup.com/articles/interviewing-users, visitado el 2021-12-14.

[19] Rosala, Maria: *The Critical Incident Technique in UX*, Enero 2020. https://www.nngroup.com/articles/critical-incident-technique/, visitado el 2021-12-14.

[20] *Acerca de U-Cursos*. https://www.u-cursos.cl/dev/paginas/acerca, visitado el 2022-10-07.

[21] Budiu, Raluca: *Change Blindness in UX: Definition*, Septiembre 2018. https://www.nngroup.com/articles/change-blindness-definition/.

[22] Nielsen, Jakob: *Website Response Times*, 2010. https://www.nngroup.com/articles/website-response-times/.

[23] Katie, Sherwin: *Progress Indicators Make a Slow System Less Insufferable*, Octubre 2014. https://www.nngroup.com/articles/progress-indicators/.

[24] *How to manage static files (e.g. images, JavaScript, CSS) — Django documentation — Django*. https://docs.djangoproject.com/en/4.0/howto/static-files/, visitado el 2022-08-13.

[25] Batchelder, Ned: *coverage: Code coverage measurement for Python*. https://pypi.org/project/coverage/, visitado el 2022-08-14.

[26] Django: *Advanced testing topics — Django documentation — Django*. https://docs.djangoproject.com/en/4.1/topics/testing/advanced/#integration-with-coverage-py, visitado el 2022-08-14.

[27] Proceedings ACM/IFIP INTERCHI'93 Conference: *A Mathematical Model of the Finding of Usability Problems*, 1993.

[28] Nielsen, Jakob: *Thinking Aloud: The #1 Usability Tool*, Enero 2012. https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/.

[29] Pernice, Kara: *Talking with Users in a Usability Test*, Enero 2014. https://www.nngroup.com/articles/talking-to-users/.

[30] Brooke, John: *SUS: A quick and dirty usability scale*. Usability Eval. Ind., 189, Noviembre 1995.

[31] Schade, Amy: *Write better qualitative usability tasks*, Abril 2017. https://www.nngroup.com/articles/better-usability-tasks/.

[32] Whitenton, Kathryn: *Minimize Cognitive Load to Maximize Usability*, Diciembre 2013. https://www.nngroup.com/articles/minimize-cognitive-load/.

[33] Nielsen, Jakob: *Enhancing the Explanatory Power of Usability Heuristics.* página 210, Enero 1994, ISBN 0897916514.

[34] W3Techs: *Usage Statistics and Market Share of Bootstrap for Websites, October 2020.* `https://w3techs.com/technologies/details/js-bootstrap`.

ANEXOS

Anexo A

Transcripción de Entrevistas

En el presente anexo se entrega la transcripción de las entrevistas realizadas a las ex-organizadoras de los JING y los juegos La Mona.

El texto en negritas corresponde a las preguntas y comentarios del memorista, mientras que el texto regular son las respuestas de la entrevistada.

Dado que el foco de la entrevista es obtener información del problema y no sus participantes en sí, en la redacción se suprimieron muletillas, coloquialismos y comentarios personales irrelevantes al dominio de esta memoria.

A.1. Entrevista a Fabiola Mariqueo

Fabiola Mariqueo, actual integrante del Centro Deportivo de Ingeniería, FCFM. Tiene experiencia organizando La Mona y participó de varios JING. Entrevista realizada el viernes 8 de abril de 2022, via videollamada.

Esta es la continuación de un trabajo dirigido que había hecho Jocelyn Simmonds del DCC, en el 2019. - ¿Sabías sobre este trabajo? ¿Habías hablado con Jocelyn o con los que hicieron la plataforma?

Yo, específicamente, no. En ese tiempo estaba Sofía Baeza, que era la presidenta. Estaba mucho más metida con respecto a este proyecto, porque al final era un proyecto que se diera para los JING ese año, pero por distintas barreras que tuvieron no se logró llegar con la aplicación. Entonces, estuvo medio complejo el trabajo en ese momento, pero sí sé en qué

estaba el proyecto, más o menos.

- Este proyecto lo estaban trabajando dos alumnos del DCC. Como era un trabajo dirigido, no alcanzó a terminarse al 100 %, y tampoco hay nada documentado. Ahora me tocó recibir este proyecto sin estudios metodológicos y un código incompleto. Entonces, mi idea con esta reunión es volver a retomar estos requerimientos, para saber cuáles son las necesidades que tienen ustedes para organizar torneos, ya sea La Mona, los JING, de distintas envergaduras, y para poder generar un poco más de insumo y de sustento para esta memoria, para que no sea programar una cosa que en verdad al final nadie va a usar.

- ¿Has organizado alguno de estos torneos (Mona, JING, JOE)?

Sí. O sea, los JOE nosotros no organizamos mucho porque son parte de la DDAF, que es el área de deportes de la Universidad, pero sí la Mona y la Copa CDI. Varios años, de hecho; pero siempre en baby fútbol, que es como el deporte que a mí me gusta.

- ¿Cuál es la estructura del campeonato (La Mona)? ¿Lo hacen por equipo, por departamento? ¿Es muy estricto el requerimiento de departamento?

De la Mona, en particular, efectivamente la gente tiene que ser del departamento, excepto por las personas que van en Plan Común, en Segundo Año. Muchas veces hay gente que va atrasada, gente que tiene ramos de distintos departamentos. A ellos se les permite "parchar", por así decirlo; pero, en el fondo, son como agentes libres que pueden decidir en qué departamento juegan. El tema no es que puedan jugar en distintas competencias por varios departamentos, sino que deciden. Por ejemplo, si están en Plan Común todavía, en Segundo Año, pueden decidir representar a Eléctrica, pero tienen que jugar solo por Eléctrica; no pueden jugar voleibol por Eléctrica y baby fútbol por Industrias.

- ¿Cómo funciona el sistema de puntaje? Explícame un poco cómo funciona esa estructura, de cómo se arma la competencia en sí.

A priori, se da cupo como para que todos los departamentos tengan un equipo por sí solo, pero de repente no todos los departamentos inscriben una cantidad suficiente de jugadores o jugadoras para que puedan hacer un equipo y puedan jugar bien. Entonces a veces se hace una fusión de departamento. El puntaje que se otorga es a los primeros tres lugares: Se otorgan 400, 200 y 100 puntos, respectivamente, del primer al tercer lugar. Hay categorías hombres y mujeres. Cualquier persona no-binaria puede jugar en el equipo que le plazca; no tenemos restricciones para eso. Y, como te dije, son cupos para todos los departamentos. Creo que son 13 departamentos, o 14 contando Plan Común, si lo dividimos. Son 14 cupos. Ahí, se arma un grupo dependiendo de cuantos equipos haya finalmente, considerando las fusiones. En cada torneo, por ejemplo, si son 8 equipos en total, se divide en 4 grupos. Siempre es intentando generar la menor cantidad de partidos, sacando la combinatoria.

- Entonces, tengo entendido que la arquitectura de todo esto sería como: Está el torneo La Mona, en La Mona hay varios deportes, cada deporte tiene divisiones (como hombre-mujer), y cada una de esas tiene sus propios partidos. Dependiendo de quien gane esos partidos, va sumando puntaje. Ese puntaje se suma como a un total por departamento.

Sí. Eso mismo.

- Respecto a los jugadores en sí. ¿Cuál es el proceso actual para postular o registrar a los jugadores?

Nosotros lo hacemos a través de un formulario, donde tienen que inscribir sus datos.

- ¿Como Google Forms?

Sí, Google Forms. Que es como nombre, departamento al que pertenecen, su número de contacto; y tienen que adherir un pantallazo de su IA, que son los ramos que inscribieron ese semestre.

- ¿Y todo eso lo manejan entonces después en una planilla de Google?

Sí.

- ¿Cómo funciona eso? ¿Muy enredado el llevar el registro de los jugadores? ¿Se confunden las cosas? ¿Mucho registro falso? ¿Hay algún problema particular con usar una planilla Google?

No tanto. En general, intentamos revisar todas las BIA, que es como, a priori, el factor determinante de que si una persona puede jugar en un departamento o no; pero, lo que nos pasa a veces, es que mucha gente se equivoca: De repente, hay gente que manda cuatro veces el formulario, entonces hay que estar depurando esos datos; pero no es tan engorroso tampoco.

- Respecto a la progresión del campeonato (ir anotando qué equipo ganó acá, tener tantos puntos, dónde guardar el total, . . .), ¿eso cómo lo hacen? ¿También lo manejan en planillas de Google sincronizadas?

Sí. Por eso igual sería interesante tener una plataforma donde podamos ir subiendo los puntajes, y que la gente pueda ver mucho más rápido.

- ¿Han tenido algún problema importante porque se les descoordinó la planilla, porque alguien no anotó algo, etc.? ¿Recuerdas algún problema real que les haya pasado a ustedes en algún momento?

Que me venga a la mente rápido, no; pero siempre nos pasa que hay gente que se demora mucho en subir los puntajes a la planilla, por ejemplo. Ese tipo de problemática. Nunca nos ha quedado la embarrada en el Excel.

- ¿Qué tan flexible es este Excel? ¿Alguna vez han tenido que cambiar la estructura de alguna competencia durante el torneo (que un partido ya no vaya, o juntar partidos)? ¿Ocurren ese tipo de cambios durante la competencia?

Sí nos ha pasado que de repente sacaban partidos, y se intenta sacar si es que sobra algún horario de cancha disponible. Que alguno de esos equipos juegue con otro si es que el otro equipo con los que tienen que jugar tiene disponibilidad. Al final, todo depende mucho de la disponibilidad de los equipos.

- A nivel de planilla, ¿Es muy enredado cuando pasan esas excepciones? ¿Hay que modificar mucho o cambiar algo muy importante?

No, eso no es problemático.

- ¿Cómo se comunican ustedes con los jugadores de la competencia?

A través de Whatsapp. Eso, igual es bastante engorroso a veces tener demasiados grupos de Whatsapp. Al final, lo que pasa es que siempre hay un encargado por categoría, pero siempre hay una persona que está supervisando que esté todo funcionando. Entonces, esa persona que se encarga de cerciorar de que esté funcionando todo tiene como 8 grupos de Whatsapp, y en La Mona hemos tenido hasta 12 categorías y hay que estar en los 12 grupos de Whatsapp.

- En estos grupos de Whatsapp, ¿hablan con todos los jugadores o tienen representantes de cada equipo?

No, solo con los capitanes.

- Para comunicarse rápido, si cambiaron la cancha a última hora, o se cambió un partido a última hora, ¿todo eso funciona por Whatsapp? ¿Les ha traído problemas en algún momento comunicarse de esa forma?

Que yo sepa, no. De repente, hay capitanes que avisan muy encima que no pueden, que se les baja gente; pero creo que es más por problemas de coordinación de ellos como grupo, más que por la plataforma.

- ¿Cuánta gente participa en la organización completa? Entre organizadores, encargados de deporte, árbitros; cualquier persona que tenga que ver con la organización, ¿más o menos de cuánta gente estamos hablando?

Lo que pasa es que nosotros como CDI somos entre 10 y 14 personas, y los árbitros, por tema de tiempo y recurso salen de integrantes de otros equipos. Por ejemplo, si el equipo A y B tienen que jugar, salen del equipo C, y después cuando el equipo C y D tengan que jugar, salen del equipo A y B los árbitros.

A veces, hacemos llamado a gente que quiera organizar, y llegan como 5 a 6 personas. Entonces serían como 20 personas que están organizando.

- En tu experiencia, ¿cuánto tiempo te ocupa *[organizar]* a la semana o en el día a día? ¿Son varias horas que tienes que dedicarle a esto? ¿Te ocupa mucho tiempo de la U, o te complica mucho con los ramos?

Depende. Para la persona que está supervisando, no es tanto. Si bien es más engorroso por la cantidad de grupos de Whatsapp, no hay que invertir tanto tiempo. Pero, por ejemplo, para organizar baby fútbol, siempre ahí es un poco más de tiempo porque siempre damos 32 cupos de equipo; entonces, si salen todos los partidos, son por lo menos 16 partidos a la semana, más la categoría de mujeres. En general, la persona que organiza baby fútbol organiza baby hombres y baby mujeres. El tema es que hay que estar cateteando harto para que manden la disponibilidad. Pero una vez que ya contamos con toda la disponibilidad,

estamos una hora haciendo calzar los partidos.

- A lo que voy con esta pregunta, es porque algo que considero importante para esta plataforma es que tiene que ser, dentro de todo, bien sencilla y rápida de usar, y que se pueda, por ejemplo, usar en el celular. Considerando el contexto estudiantil, como ustedes son estudiantes también, entonces están ocupados durante el día: No tienen horario de trabajo fijo, digamos, horario de oficina, para organizar estas cosas. Probablemente la experiencia del usuario es que se va a meter medio apurado en la página, o va a tener que estar a distintas horas del día trabajando en esto.

Ajá *[asiente]*

- Respecto al traspaso de información a medida que pasan los años, ¿ustedes conocían a la organización anterior, o eran ustedes mismos? ¿Fue muy complicado ese proceso de traspasar ese conocimiento de un año a otro?

Como cada 3 o 4 años hay una generación de recambio que suele estar, no sé si un poco más a la deriva, pero siempre hay poca gente antigua y mucha gente nueva (lo que nos está pasando este año). De las 14 personas que somos, somos 2 que llevamos hartos años, y el resto solo están del año pasado o de este año.

- En pandemia, ¿cómo les afectó la organización de esto? En general, ¿cambió mucho la organización del torneo en sí?

La verdad es que sí, porque el primer año solo fueron E-sports. Yo lo vi como una oportunidad igual. En ese tiempo me tocó a mí ser presidenta en 2020, entonces obviamente fue todo muy doloroso el proceso, porque al final estábamos todos muy encerrados, había que dedicarle tiempo cuando, en realidad, no había ganas. Pero fue una oportunidad de abrirnos mucho más, porque antes los E-sports que hacíamos eran Mario Kart y LoL. Ese año incluimos una cantidad enorme de juegos que son muy competitivos y otros que son más didácticos, como Ludo, o Age of Empires, Rocket League; muchas competencias que en realidad al beauchefiano sí le llegan. Hay mucha gente que juega un montón de E-sports.

Fue una oportunidad porque, al final, La Mona es este torneo grande que aúna todas las competencias, y la copa CDI; pero en el fondo, es para que la gente pueda compartir con las personas que están en su departamento. Está enfocado sobre todo para la gente que está entrando a la especialidad, y la copa CDI, que es una versión donde la gente puede hacer equipos con las personas que ellos quieran. Ahí se pueden mezclar de todos los departamentos, de todas las edades. Esa es la diferencia con el propósito.

La cosa es que la copa CDI nunca ha incluido E-sports. Dado que en la pandemia pudimos incluir estas cosas, ahora está esta parte de la copa CDI, donde estamos organizando los deportes colectivos más tradicionales, y se va a solapar al final de la copa CDI, el inicio de lo mismo que una copa CDI (va a tener un nombre específico), pero versión E-sports. Ahí hay una inclusión súper grande de una cantidad de deportes que antes no incluíamos.

- Agregar los E-sports, ¿cambia en algo el sistema de puntajes, de equipos, divisiones? ¿O funciona igual que los deportes físicos?

Es lo mismo. A veces, lo que cambia es que se hacen llaves. Lo que vi en LOL, por ejemplo, que era como un sistema de llaves, y después los que ganaban tenían un winner bracket y los que no loser bracket. Quizás, por ese lado, sí pueda cambiar. Quizás ya no hay fase de grupos. Como son enfrentamientos, hace mucho más sentido pasarlo a un sistema de llaves, y no de fase de grupos.

- ¿Todos los deportes son de 1 versus 1, o puede haber algo distinto?

Creo que todos son 1 versus 1. Excepto el Ludo, cuando ahí las partidas eran de 4 personas.

- Respecto a los JING, ¿tienes alguna experiencia organizándolos?

Sí. En 2019 estuve un poco metida, iba a las reuniones, pero no veía temas logísticos. Solo dar nuestro punto de vista respecto a las decisiones que se tomaban. Pero, el 2020, estuve organizando los JING online, que también fueron solo de E-sports.

- En tu experiencia, ¿es muy distinto de La Mona?

La verdad, creo que es un trabajo súper parecido, eso sí que es más engorroso, en el sentido que, al final, son 8 universidades dando su opinión, o teniendo su opinión respecto a distintas cosas, entonces es un trabajo de coacción mucho más grande; pero en la práctica el torneo funciona de la misma forma: O existen fases de grupo o llaves, y eventualmente se va desarrollando el torneo, hasta que se les otorga un puntaje al primero, segundo y tercer lugar.

- O sea, estructuralmente el campeonato es lo mismo, pero como estamos hablando de una envergadura distinta de personas, la comunicación y toda la parte social es un poco más complicada.

Lo que es distinto en los JING a La Mona es que nosotros no hacemos la distinción entre el puntaje que se le otorga a los deportes tradicionales y a los E-sports. Para nosotros, no nos hace sentido hacer esa jerarquización. En los JING hay deportes tipo A, tipo B y tipo C. Respecto a eso se van otorgando los puntajes.

Los deportes tipo A, que al final son los tradicionales, tienen mucho más puntaje a los de tipo C, que son, por ejemplo, Ludo, Catán o Cachos, que son más tipo juego que una competición tradicional.

- ¿En qué estado están los JING ahora? ¿Quién los organiza?

Esto se empezó a hacer en 2014-2015, pero la cosa es que hay un grupo de Whatsapp donde todos los años se va cambiando la directiva de quién se va a hacer cargo, pero siempre se decide como todos los años, a principio, quién va a ser Casa. La persona que es Casa es quien pone todo, pone el establecimiento, y es quien toma la batuta de organizar. Entonces, esa persona lidera las reuniones, y tiene mucha más injerencia en las decisiones.

- Entonces es algo que se decide año a año entre los centros deportivos de cada universidad. No es que haya una organización fija de los JING.

Es muy democrático. Se decide entre todos.

- ¿En qué está eso ahora? Porque el año pasado no hubo JING.

No. Los primeros JING online participó poca gente (no todas las universidades), y después el año pasado había muchos centros de estudiantes que no estaban: Había muchos CDEs que no estaban conformados. Entonces, se decidió no hacerlos. Pero, como ahora se viene la presencialidad, ese tema cambió.

Se supone que se van a hacer, y la voluntad de todas las universidades es que se va a hacer, pero siempre está la carta trampa de que cuando ninguna otra Casa pueda (porque la idea es que todas las universidades puedan ser anfitrionas), cuando nadie más puede, la UC. Como están casi todas las carreras ahí, y está incluido ingeniería en el mismo campus, para ellos es mucho más sencillo poner el campus.

Dadas las disposiciones de las universidades, es muy probable que los JING vayan.

- Para los partidos, ¿registran las asistencias (quién llegó, ¿quién no llegó, ¿quién está jugando, . . .)?

Sí, pero de los equipos. No hacemos asistencia por persona.

Creo que, para los JING, sí tiene funcionalidad *[tomar asistencia por persona]*, en el sentido de que, de repente, hay gente que juega muchos deportes; para saber dónde está una persona y dónde tienen que ir a buscarlo para competir. Eso nos pasa mucho, hay gente que compite en cosas y hay que ver si es que sigue compitiendo o no, para ver si es que puede llegar al otro evento.

Como es un evento que está muy espalda tras espalda, los partidos se atrasan y se alargan, asumo que quizás para saber que una persona terminó su partido. Eso me hace sentido.

A.2. Entrevista a Sofía Baeza

Sofía Baeza, ex presidenta del Centro Deportivo de Ingeniería, FCFM. Tiene experiencia organizando los JING y La Mona. Entrevista realizada el lunes 2 de mayo de 2022, presencialmente.

[Previo a iniciar la grabación, se dio una contextualización similar a la otra entrevista]

- ¿Algún comentario sobre el proyecto?

[. . .] Creo que la idea era muy buena, porque como éramos hartos (éramos 7 universidades, nosotros llevábamos 1000 personas), era mucho volumen de gente, y esto facilitaba toda la organización, sobre todo la parte de los resultados. Yo creo que el hecho de ir sabiendo los resultados a medida que pasaban (porque tenías que ver con quién tenías que jugar, si era un partido de fútbol tenías que ver como avanzaba la otra persona), te servía para ir contando puntos: si vamos bien en esto, vamos a salir primero en lo otro, etc. Esa información se daba al final del día, nosotros terminábamos a las 12 de la noche, a la 1 de la mañana, y recién ahí decíamos en esto ganaste tú, en esto gané yo. Lo otro que resolvía bien era la transparencia:

Obviamente todas las cosas tienen distintas versiones, entonces yo digo como, "no yo gané y te gané 3-2" y tú dices que yo gané 3-1. Y hay una planilla que es literal un papel, que pasa de un tipo a otro, y en un campus enorme tienes que ir de un edifico a otro. Todo eso, digitalizarlo, facilitaba toda la gestión.

- Entonces, ¿los puntajes los anotaban en papel y después los pasaban a una planilla online?

Sí, al final del día, como a la 1 de la mañana, pasando las cosas a internet. Y también, si lo metes a un Drive, que tiene 20 personas, no es como que uno vaya a pensar que están alterando [los resultados], pero podría pasar; y si a alguien se le acaba el espacio, perdemos el Drive.

- Lo que hablé con Fabiola, sobre el tema de La Mona, en ese sentido, las planillas en Excel no eran gran problema, porque no es tanto la escala de gente que hay, pero me imagino que, si estás tratando con 7 universidades de distintos lados, que quizás ni se conocen, debe ser problemático.

Es mucha gente distinta. Había un encargado de cada deporte, y ese encargado de cada deporte recuperaba las cosas de todos los árbitros, de todos los partidos. A veces había árbitros externos, a veces era buena gente que te ayudaba. Tenías que estar persiguiendo la planilla. No funcionaba bien.

- Entonces sería bueno que haya un registro de quién hizo cada cambio, para que sea transparente.

Me acuerdo de que eso se registraba en la aplicación, como que un coordinador tenía ese poder, y todo lo que tú quisieras hacer tenías que verlo con ella. Y, por ejemplo, me acuerdo de que habíamos puesto un sistema, en que había cierta cantidad de minutos (como 10 minutos) para que alguien pudiera objetar el resultado que se había subido. Entonces, mandaba una alerta, y eso se revisaba en este comité, que somos las 7 universidades. Yo era representante de la Chile, y cada vez que pasaba algo grave, nos juntábamos y decíamos "¿Qué vamos a hacer con esto?" y decidíamos; pero terminábamos haciendo eso para todo. No era que pasara algo grave, sino que cada cosa que se tenía que hablar, teníamos que hablarlo todos. Y eso también se vuelve un poco tedioso, porque son muchos deportes, muchas cosas al mismo tiempo; entonces este tipo de alertas, que nos llegaba al celular (se supone), y tú ahí veías "ya, tenemos que revisar esto", lo veíamos en dos segundos y listo.

- Me imagino que igual, todo esto se hace durante el semestre, entonces están todos también con sus propios ramos, sus propias carreras y cosas que hacer. Me imagino que comunicarse inmediatamente no es muy factible.

Igual, cuando no ha sido acá en Santiago, es más fácil, porque estamos todos allá. Fue en Viña una vez, en Valparaíso otra; ahí iba a ser en Concepción, pero la pandemia.

Generalmente es viernes, sábado y domingo; o jueves en la tarde, viernes, sábado y domingo.

El tema es que son solo 7 personas (porque es un representante por universidad) que deciden todo. Entonces, dejar estos registros en la aplicación también nos servía para guiar

un poco la conversación y no estar hablando cualquier cosa; dejar un registro, como dices tú, de quién cambió, quién ingresó el puntaje. Si hay que preguntar algo, ya sabes más o menos quién lo vio.

- Hablando más del torneo en sí, ¿qué tipo de deportes se hacen en general? Porque ahora también en pandemia se han sumado deportes online.

Siempre los tuvimos. Había 3 categorías de deportes: El A eran los más típicos. También estaba LOL ahí, porque era un furor en las universidades. Y después empezamos con cosas más básicas. El B creo que es como un híbrido; como handball, por ejemplo, que no todas las universidades tenían. Cosas que no estaban tan institucionalizadas quedaban en el B. Y, en el C, había cachipún, y cualquier cosa.

- ¿Esas categorías eran más o menos fijas? ¿Siempre eran 3 categorías y con X puntaje, que se mantenía fijo siempre? De repente cambiaba de un año a otro; pero raro. Por ejemplo, si al año siguiente todas las universidades tenían handball, no tenía sentido dejarlo en la B, porque ya había gente que ya había entrenado.

- Pero ¿el deporte cambia de categoría, pero la categoría en sí sigue con la misma ponderación?

Nosotros la mantuvimos los 3 años que estuve.

- Había visto en la aplicación que hay ahora, que está ese sistema de categorías, pero está dentro del código mismo escrito el puntaje, entonces no sería algo que podrían cambiar ustedes fácilmente.

Si se pudiera ingresar cosas, adaptarla a La Mona, ser más flexible; eso sería mejor.

Cuando armamos este proyecto, lo pensaba para la universidad también. Esto sigue siendo planilla, nadie tiene un sistema que de verdad funciona y que esté pensado para el deporte. [...]

- Sobre la postulación y el registro de jugadores, ¿cómo funcionaba eso en general (en 2019)?

Teníamos un tope. Teníamos una fecha tope para entregar todas nuestras planillas.

- Pero, el postulante mismo, el deportista, ¿cómo se enteraba de esto?, ¿cómo se registraba? ¿Era en Google Forms?

No era Google Forms, no éramos tan modernos en ese momento. Yo creo que era un Excel nomás.

- Entonces, ¿los mismos centros de deportes se contactaban e iban llenando las inscripciones?

Cada universidad llenaba su inscripción en un Excel. Cómo, tú, por ejemplo, si querías jugar algo, creo que te acercabas al CDI, y nosotros te inscribíamos. Ahora, Google Forms, mucho mejor. No sé si la última vez lo usamos, habría que ver como meterse al Drive. Pero esa era la idea, y ese igual era un tema, porque mucha gente quería ir en algunas cosas, y

poca gente quería ir en otras. Entonces, yo creo que igual ahí la difusión, el 1-a-1, después buscabas gente para que fuera. Nosotros necesitábamos varios datos, por el tema del seguro de la universidad, teníamos que saber que era alumno.

- Lo comparo con La Mona, que, como es un poco más abierto, quizás "más informal". Usan un Google Forms y se inscriben todos los que quieran.

Creo que eso funciona, porque en el Google Forms puedes subir documentos, como el boletín, o la BIA. Yo creo que eso está bien.

- Pero como funcionaban ustedes, ¿era más jerarquizado quizás? ¿Iba el deportista, hablaba con su centro de deportes, y el centro de deportes lo inscribía?

Sí, eso lo hacíamos; pero también aceptábamos estudiantes por sí solos.

Ahora, yo creo que ahí, lo importante sería el tema de cómo se inscribían en la app, porque tenías que tener un usuario; entonces nosotros ingresábamos los usuarios que teníamos, pero cualquier cambio que hubiera, no teníamos cómo cambiarlo. Quizás la inscripción podría ser directamente en la página.

- Siguiendo con los deportistas, ¿tomaban asistencia? ¿Es importante tomar asistencia?

Lo hacíamos en los buses, porque necesitábamos que no se quedara nadie abajo, y necesitábamos saber que iban, porque íbamos a otro lado. Acá no. Acá llegaban no más a su partido (porque el último fue en la Católica), y ahí mismo veías si tenías a la gente.

No sé si uno podría pasar asistencia antes, pero sí marcar si vino o no vino. Lo cual le da información al organizador más que al usuario. Sería bacán saber como "oye esta persona faltó a todas las cosas que se inscribió"; no lo llamamos.

- En ese sentido, la asistencia era más importante para saber que estuvieran y que no se hubieran quedado.

Cuando estábamos afuera, eso era importante. Una vez se nos perdió un tipo y fue horrible. Pero también para los deportes, porque, si no, no puedes jugar. Si te faltan 3 y no tienes banca, perdiste por walkover, y esa no es la idea si estás viajando y en este evento.

- ¿Podía pasar que llegara alguien que no estaba inscrito y que tomara el lugar de otra persona?

No. La idea es que se mantenga, no solamente el nivel de que eres bueno, sino que la organización. Porque para otro equipo que sí se preparó y que sí estaba todo motivado y acá llegan puros parches, da lata. Se nota.

Yo creo que, en algunos casos, lo dejamos. Pero los JING son muy competitivos. Las universidades quieren ganar. [...]

- El registro de los puntajes dijiste que era bien complicado. ¿Tuvieron problemas graves alguna vez? ¿Alguna situación particular, en algún momento, por el problema de la planilla?

Sí, porque se perdían a veces. Entonces, cuando se pierden, tienes que juntarte con los equipos, y preguntarles a los dos cómo les fue, y eso logísticamente no es bueno, porque tienes que buscar a la persona que estuvo en ese momento, y que ambos se pongan de acuerdo [...]. No recuerdo que nos haya pasado así como que tenemos que volver a hacer un partido. Aparte que no podíamos, porque los espacios están súper copados. Nos prestan generalmente las instalaciones, pero son tantos partidos de tantas cosas, que es uno tras otro, y no es como que tengas tiempo para meter algo extra; aparte que debes tener al equipo ahí, preparado. Dependíamos demasiado de eso, y por eso ahí nos quedábamos hasta tan tarde cuando había discrepancias. No teníamos otra forma: Éramos 7 personas tratando de ponernos de acuerdo. Era horrible.

Planillas que se pierdan, yo anoté una cosa y tu viste otra, y todo eso; cuando no está en un mismo lugar, es difícil transparentar.

- Hablando de los espacios, ¿en general es muy complejo llevar la logística de los espacios?

Era un tema, porque generalmente los JING eran en invierno. No invierno, pero septiembre. Está muy rico el día, y al otro hace mucho frío. Igual dependíamos harto de eso.

Como te digo, eran muchos espacios distintos. El cachipún también lo tienes que hacer en algún lugar, y tienes que explicarle a la gente dónde es eso. Entonces, si yo digo "ya va a ser aquí", cómo le explico a una persona qué es esto, cómo llega, dónde está, a qué hora tenía que estar. Era difícil hacer calzar los horarios con las cosas más tradicionales, como las canchas de fútbol; pero, además, era difícil explicar o ilustrar dónde eran los otros eventos. Por ejemplo, en la Santa María que está en Valparaíso, había salas que estaban súper escondidas, y el LOL se jugaba en una sala. Teníamos mapas, entregábamos muchos mapas en volantes. Y esa vez que hicimos la app pusimos una foto del mapa, pero cambia una sala y ¿cómo avisas?; entonces empezamos a avisar por Whatsapp [...]. Entonces, igual la logística era "ojalá lo lean, ojalá lleguen". Llegaban atrasados, se corría la hora, ese tipo de cosas.

- Ese es otro tema. La mensajería. ¿Era muy complejo coordinar todos esos Whatsapps?

Cada universidad funcionaba distinto. Me acuerdo que la Católica tenía una lista en Whatsapp y los tenía a todos. Todo lo que decían, le llegaba a todo el mundo, y había una forma en que tú podías hablar y no respondieran, que yo no conocía eso en Whatsapp; y era lo que queríamos hacer en esta app, que cierta gente pudiera mandar mensajes, no que se generara una conversación. Como más anuncio. Eso es lo que habíamos visto, y eso es más importante, porque estos cambios repentinos son más fáciles de anunciarlos.

Yo funcionaba con millones de Whatsapp, pero igual dependía de que el capitán bajara la información, y ahí quién sabe. Era difícil.

Me acuerdo que pusimos un lugar de novedades. Había unas novedades, y un chat, pero ya no me acuerdo.

- ¿Qué fue lo principal de la aplicación que no funcionó cuando intentaron usarla?

No podían ingresar. No hacía match con el usuario. No podían entrar, y nadie sabía que en verdad funcionaba bien.

- ¿Habían probado antes?

Sí, lo probamos aquí 3 usuarios, 4 usuarios. [...] Ese fue el problema: que la gente no podía ingresar. Ni siquiera sé si funcionó el resto. Quizás había otras cosas que nos habría funcionado, pero ni siquiera llegamos hasta ahí. [...]

- ¿Cómo es la transición entre directivas de JING entre un año y otro? ¿Suele ser mucha gente repetida? ¿Cómo se traspasa la información, el conocimiento de cómo se hacen los JING?

Ahí te puedo hablar de mi experiencia. Bueno, yo estuve tres años. Siempre íbamos de a dos. Siempre había dos encargados de cada universidad, pero uno era el representante, cosa de que no me voy yo y se pierde toda la información. Entonces, todas las universidades tienen dos personas. Nosotros tenemos esta figura del CDI, pero en otras universidades era el encargado de deportes del centro de estudiantes, o el mismo presidente del centro de estudiantes. Dependía de la universidad de como estaba organizada internamente; pero, para nosotros, siempre era alguien del CDI; no necesariamente el presidente o presidenta, sino como alguien que estuviera a cargo de este proyecto.

Como se pasa, yo lo que hice fue ir con otra persona, que esta segunda persona fuera quien tomaba la cosa después. No me funcionó. La segunda persona se fue. Pero yo al menos era súper transparente con todo. Como toda la gente del CDI estaba metida [...], en realidad yo compartía todo con el CDI mismo. Teníamos una carpeta JING, yo tenía acceso, la otra persona tenía acceso, y los demás podían ver; o yo ponía el correo del CDI, entonces toda la gente que tenía acceso podía.

Lo otro bueno es que, como nos dividíamos por equipo (o sea, fútbol tenía su capitán, las otras cosas tenían su capitán), a veces ellos tomaban los JING después. Igual ya saben como funciona y todo.

Tenemos toda la información en una carpeta Drive, no sé si tengo acceso todavía, pero estaba todo bastante ordenado; los permisos eran acotados igual, no era información abierta.

- Más que nada, pensando en que llegue un año en que cambie el CDI o quien sea que esté organizando, y se olvide de esta aplicación porque nadie sabe usarla, o no saben que existe.

Creo que ahí, lo que siempre pasa: En las organizaciones uno tiene que dejar un informe. [...] Un instructivo, cómo usar la aplicación, [...]. No solo para nosotros, [...], pero ponte tú para la Universidad de Valparaíso, quizás sí les sirve un instructivo de cómo usar la aplicación.

Me preguntaste si cambiaban mucho. En los tres años que estuve yo cambiaron un par nomás. [...]

- Me dijiste que esto es normalmente fin de semana. Entonces, ¿No es como que estaban entremedio de controles, de pruebas, o de tareas?

Me acuerdo que hubo un año que pedimos que cambiaran un control de inglés, y lo hicieron, pero cada estudiante habló con los profes. Ahora, eso era cuando ibas para afuera, cuando fue acá en la Católica el último año te ibas a tu casa, pero allá estabas con la gente todo el día. Entonces, creo que depende del lugar, pero sí, es un fin de semana; dedicarle todo tu fin de semana a eso.

- Más que nada, pensando en cuanto tiempo le dedican en usar esa plataforma. Si van a estar apurados viendo entre sus tareas, o que van a estar corriendo en la micro viendo el celular, claro, quizás tiene que ser un poco más rápida la aplicación. Menos instrucciones, más botones [...] Si me dices que normalmente le dedican 100 % o 90 % de su tiempo, ¿quizás no es tan prioritaria la velocidad?

Yo creo que igual, cuando te metes a U-Cursos, te metes porque quieres saber algo. En este caso, es donde me toca jugar, e igual necesitas tener una respuesta más o menos rápida. Pero sí, te familiarizas con la aplicación. No es como que alguien va a llegar y quiere saber algo rápido y nunca ha visto la aplicación. Hay un uso muy intensivo en esos 3 días.

- ¿Cuánta gente trabaja en esto al mismo tiempo?

A mí nunca me tocó organizarlo, porque la sede es la que más trabaja, obviamente. Queríamos que fuera JGM, el Poli, pero cuando yo estaba todavía estaba muy nuevo; entonces no lo pedimos. [...]

Nuestra organización, eran cerca de 20 deportes, había un coordinador por cada deporte (20 personas), más los 10 del CDI, más algún staff que nos ayudara porque igual era mucha gente. Éramos 40 personas, 30 personas; de mí universidad no más. Después todas las universidades tienen su gente. [...] pero la sede tenía harta gente, porque era un coordinador por cada deporte, tenían gente de comunicaciones, de logística. Como si fuera una empresa. Encargados de sala, a veces teníamos alojamiento.

- O sea, ¿fácilmente 150 a 200 personas en total?

Sí.

- ¿Y los que gestionan los puntajes y los participantes?

[...] Hay un coordinador por deporte, serán 20, más los 7 que somos nosotros, 14 si nos dan poder a los dos. Serán 40 o 50 personas.

- Me imagino coordinar a todas esas personas en planillas de Excel... todos editando al mismo tiempo.

Y muchas veces físicas! Porque de repente la señal no era tan buena... Tenías una montaña enorme de papeles...

¿La idea era usar esto in situ? ¿Terminaba el partido y registrar el resultado ahí mismo en la aplicación?

Y ahí se activaba este tiempo, para hacer reclamos.

- ¿En qué están los JING?

[...] Creo que hay muchas ganas de hacerlos. [...] Si la pandemia lo permite, de todas maneras. ¿Dónde? No tengo idea qué universidad se va arriesgar a hacerlo ahora. [...] Conce quería hacerlo, pero ya con la pandemia y todo no les dieron permiso, así que creo que lo único difícil sería el lugar.