UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Redevelopment of the competition administration system

Daniel Alexandre Pires Correia Cambinda

Mestrado em Informática

Trabalho de Projeto orientado por:
Prof. Doutor Luís Manuel Ferreira Fernandes Moniz

2024

"We are what we overcome."

Acknowledgments

I would like to express my heartfelt gratitude to my college advisor for the support and guidance throughout the project. A special thank you to Lander for giving me the opportunity to work on this project in Acro Companion, which has been invaluable for my growth.I am also deeply thankful to Filipe, Tiago and Ruben for their assistance, expertise, and the knowledge they have generously shared with me. Your encouragement and collaboration have made a significant impact on my learning experience.

I would also like to extend my appreciation to my friends, whose support and camaraderie have made this journey more enjoyable and memorable. Thank you for being there through the highs and lows.

Lastly, I want to dedicate this acknowledgment to my family, whose love has been my foundation. In particular, I want to honor my grandmother, who unfortunately passed away last year. Her strength and wisdom will forever inspire me every day.

Resumo

A ginástica, como um dos desportos olímpicos mais antigos, combina precisão, habilidade e coordenação, elementos que também se aplicam à administração de suas competições. A gestão de competições de ginástica é uma tarefa desafiadora, que exige uma abordagem sistemática para garantir que todas as regras e regulamentos sejam seguidos de maneira precisa. Para enfrentar essas dificuldades, surgem soluções tecnológicas especializadas, como o projeto Acro Companion, que visa melhorar a administração de competições de ginástica. A presente dissertação documenta os esforços de desenvolvimento e melhorias realizadas na plataforma Acro Companion, focada em aprimorar a eficiência e simplicidade do processo de gestão de competições.

A Acro Companion é uma empresa belga que oferece soluções para a administração de competições de ginástica desde 2016. Inicialmente, a plataforma era voltada exclusivamente para competições de ginástica acrobática, permitindo que os administradores de competições criassem, importassem e gerissem dados de maneira mais prática em comparação com ferramentas tradicionais, como o Microsoft Excel e o Google Sheets. Com a pandemia da COVID-19, a plataforma foi adaptada para realizar competições online, possibilitando que vídeos das apresentações fossem enviados em vez de apresentações ao vivo, mantendo o fluxo das competições durante um período de restrições sociais.

O objetivo desta dissertação é relatar o processo de desenvolvimento realizado durante o estágio na Acro Companion, que teve como foco a modernização e expansão das funcionalidades do sistema de administração de competições. O projeto visa aumentar a estabilidade do sistema, garantir maior precisão e consistência dos dados, além de diminuir a dívida técnica acumulada ao longo do tempo. A implementação de novas funcionalidades como a formatação de nomes e permitir copiar administração de competições genéricas e a correção de problemas encontrados na solução atual como a falta de validação na importação de dados são centrais para este trabalho, garantindo que o sistema se mantenha robusto e eficiente.

A administração de competições envolve a coordenação com federações de ginástica nacionais e internacionais, como a Federação Internacional de Ginástica (FIG), para assegurar que os eventos ocorrem dentro dos calendários e regulamentos previamente estabelecidos. Isso inclui desde a definição de categorias etárias e níveis de competição até a alocação de juízes e a organização de painéis de avaliação. Uma das dificuldades enfrentadas pelos organizadores é a garantia de que todas as etapas de uma competição – como qualificatórias, semifinais e finais – ocorram de maneira fluida e sem erros.

Diante dessas necessidades, a Acro Companion oferece uma plataforma especializada para gerir essas complexidades, fornecendo uma interface gráfica que facilita a criação e o controle de competições de ginástica. Ao longo desta dissertação, exploram-se as dificuldades encontradas na versão atual da solução, bem como as melhorias introduzidas para resolver esses problemas e adicionar novas funcionalidades.

Os objetivos centrais do projeto foram:

- **Estabilizar o sistema atual** - Isso incluiu a identificação e correção de bugs e falhas que comprometessem o funcionamento do aplicativo.
- **Aumentar a precisão e consistência dos dados** - Foram implementados mecanismos de validação de dados para garantir que entradas incorretas ou incompletas fossem evitadas, minimizando erros no sistema.
- **Desenvolvimento de testes automatizados** - Para garantir que as novas funcionalidades não causassem a quebra das funcionalidades já existentes, foram criados testes automatizados que asseguram a estabilidade do sistema.
- **Redução da dívida técnica** - Este conceito refere-se à necessidade de reestruturar ou corrigir partes do código que foram implementadas de maneira subótima para atingir objetivos de curto prazo. A melhoria do código e a eliminação de componentes desnecessários foram essenciais para garantir a escalabilidade e a manutenção futura do sistema.

Durante o projeto na Acro Companion, várias tecnologias foram empregadas para a modernização do sistema. Entre as principais estão:

- **Angular**: Framework de front-end baseado em TypeScript, utilizado para criar uma interface gráfica interativa e escalável. A arquitetura modular do Angular facilitou a criação de componentes reutilizáveis e a divisão do código em módulos, o que tornou o sistema mais eficiente e organizado.
- **RxJS**: Biblioteca para programação reativa, integrada ao Angular para facilitar a gestão de operações assíncronas. O uso de RxJS permitiu a implementação de atualizações em tempo real e fluxos de dados reativos, essenciais para o funcionamento dinâmico do sistema.
- **Angular Signals**: um wrapper em torno de um valor que notifica consumidores interessados quando esse valor muda. Os Angular Signals podem conter qualquer tipo de valor, desde primitivos até estruturas de dados complexas.
- **Firebase**: Serviço de back-end oferecido pela Google Cloud Platform, utilizado para gerir bases de dados em tempo real, autenticação de usuários e armazenamento de arquivos. O Firebase foi escolhido por sua integração eficiente com o Angular e pela capacidade de realizar atualizações de dados instantâneas.

- **Playwright**: Ferramenta de testes end-to-end (E2E) desenvolvida pela Microsoft, utilizada para automatizar a execução de testes no navegador. Essa ferramenta foi essencial para garantir que as funcionalidades do sistema funcionassem conforme o esperado após a adição de novas funcionalidades.

Durante o processo de desenvolvimento, várias melhorias significativas foram introduzidas na plataforma Acro Companion. Entre elas:

- **Aplicação do Patcher ao Restaurar Backups** - Foi implementado um sistema de patch (atualização de modelos) que garante que, ao restaurar dados antigos de backup, as estruturas de dados e as lógicas de negócio sejam atualizadas para o modelo mais recente do sistema. Essa melhoria foi fundamental para evitar inconsistências e garantir que os dados mais antigos continuassem compatíveis com o sistema atualizado.

- **Trim de Strings** - Implementou-se uma função que remove automaticamente espaços em branco no início e no final de caixas de texto, garantindo que os dados sejam inseridos de forma consistente e evitando problemas de pesquisa e comparações erradas na base de dados.

- **Melhoria das Combinações de Especificações** - A criação de combinações de especificações para competições foi simplificada. Agora, os gerentes de competição podem selecionar múltiplos níveis e categorias em uma única especificação, o que reduz o tempo necessário para configurar uma competição. Além disso, foram adicionadas validações para impedir que combinações impossíveis fossem criadas, como um exercício de ginástica que exigisse uma dupla, mas fosse atribuído a um atleta individual.

- **Melhoria da Navegação e Importação de Dados** - Para melhorar a experiência do utilizador, foram feitas alterações no sistema de navegação por guias e na seção de importação/exportação de dados. Guias desnecessárias foram ocultadas em contextos onde não se aplicavam, tornando o sistema mais simples e fácil de usar.

- **Formatação de Nomes** - Criou-se um recurso para padronizar a formatação dos nomes de ginastas e juízes. O sistema oferece duas opções de formatação: uma com a primeira letra de cada nome em maiúscula e outra em formato oficial, com o sobrenome em letras maiúsculas, conforme utilizado em documentos formais.

- **Copiar Administração de Competições Genéricas** - Essa funcionalidade permite que os gerentes copiem as configurações de competições anteriores, economizando tempo ao evitar a necessidade de configurar manualmente todos os detalhes de cada nova competição.

- **Refatorização do Sistema de Comunicação das Competições Genéricas** - A dessincronização entre o modelo de administração de competições e o modelo de organização/competição é um grande problema no sistema de administração de competições. Nesta secção é abordada a transição do uso da biblioteca Rxjs para os novo Angular Signals como primeiro passo

para a resolução deste problema. A utilização de Signals permite uma melhor reatividade e sincronização entre os modelos, garantindo que as operações de gravação de dados sejam realizadas apenas quando ambos os modelos estiverem devidamente alinhados.

- **Migração do Sistema de Gestão de Competições de Módulos para Componentes Autónomos** - Nesta secção é abordada a migração para os novos Angular Standalone Components removendo assim o uso dos NgModules.

O projeto realizado durante o estágio na Acro Companion resultou em melhorias notáveis na solução de administração de competições, tornando o sistema mais consistente e funcional. A integração de tecnologias modernas como Angular, RxJS e Firebase, combinada com a implementação de novas funcionalidades, garantiu que o sistema atenda às necessidades dos gerentes de competições de ginástica, simplificando o processo de organização de eventos complexos. Além disso, as melhorias implementadas em termos de validação de dados e usabilidade destacam-se como importantes avanços na experiência do usuário.

As mudanças implementadas não serviram apenas para melhorar o desempenho do sistema atual, mas também para estabelecer uma boa base para futuras atualizações e expansões na administração de competições de ginástica. O próximo passo será explorar novas funcionalidades, como a automação da criação de exercícios para fases subsequentes das competições, buscando otimizar processos e expandir as capacidades do sistema.

Palavras-chave: Angular, Adminitração de competições, Programação Reativa, Web Application, Firebase

Abstract

Gymnastics, a sport that demands precision, skill, and coordination, requires a meticulous approach to competition administration. The project undertaken with Acro Companion aims to enhance current solutions for managing gymnastics competitions, focusing on improving the efficiency and simplicity of competition administration through the refinement of existing features and the introduction of innovative tools. This initiative seeks to streamline various aspects of competition management, significantly reducing the complexity and time required to organize and execute gymnastics events.

This report provides a comprehensive overview of the project's progress, highlighting key accomplishments, challenges encountered, technologies used, and outstanding tasks. Substantial advancements have been made toward developing a more functional and user-friendly administration tool specifically designed to address the unique demands of gymnastics competitions. By examining both the successes and obstacles faced during the development process, the report offers valuable insights into ongoing efforts to create a solution that not only meets but exceeds the expectations of gymnastics professionals.

Ultimately, the goal of this project is to deliver a more robust and tailored competition administration platform that enhances the overall experience for all stakeholders involved in gymnastics events. Through continuous improvement and innovation, Acro Companion aspires to set a new standard in the management of gymnastics competitions, ensuring that the focus remains on athletes and the sport itself rather than the complexities of administration.

Keywords: Angular, Competition Administration, Reactive programming,Web Application, Firebase

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Gymnastics is an enduring Olympic sport with a long history, that finds itself at the crossroads of tradition and technological innovation. While its physical prowess has captivated audiences for years, the digital landscape has ushered in a new era for competition management. Organizing a gymnastics competition, regardless of the scale, is not easy. There are innumerous rules and constraints that need to be fulfilled with high precision for a competition to run smoothly.

Competitions, especially in disciplines like gymnastics, are often planned months or even years in advance. Organizing such events requires careful coordination with governing bodies such as the International Gymnastics Federation (FIG) or national federations to ensure that competitions align with broader schedules, including regional, national, and international events. This coordination includes securing venues, arranging transportation, accommodations, meals, and setting schedules for athletes, coaches, judges, and officials.

An essential aspect of gymnastics competitions is the adherence to strict rules and regulations, which govern various aspects of the event to ensure fairness and consistency. One key component of these regulations is the division of athletes into appropriate age groups. Typically, gymnasts are divided into categories such as juniors and seniors, based on their age and experience. These divisions help create a competitive balance by ensuring that athletes compete against others of a similar skill level and physical development. The age eligibility is often determined by specific cut-off dates set by the competition's governing body, ensuring a fair and structured approach.

Another crucial aspect of gymnastics competitions is the classification of event types. Gymnastics can be contested either as individual events or team-based competitions, with both formats having distinct structures. Men's and women's gymnastics feature different apparatus, reflecting the historical and technical evolution of the sport. For men, the events typically include six apparatus: floor exercise, pommel horse, still rings, vault, parallel bars, and horizontal bar. Each apparatus tests different aspects of strength, flexibility, and technical skill. Women's gymnastics, on the other hand, consists of four apparatus: vault, uneven bars, balance beam, and floor exercise. These events emphasize grace, precision, and control, with a strong focus on artistic presentation, particularly in the balance beam and floor exercise.

The overall structure of a gymnastics competition is divided into multiple stages, often beginning with qualifying rounds, where athletes or teams must meet specific scoring criteria to advance to the next stage. The qualifiers are followed by semi-final or final rounds, where the top performers compete for medals and titles. Depending on the competition, all-around champions may also be crowned based on cumulative scores across multiple apparatus. In team competitions, scores from individual gymnasts are often combined to determine the overall team score. The progression of the competition, from qualifiers to finals, is meticulously organized to ensure that each phase runs smoothly and according to schedule.

Acro Companion[7] is a Belgian startup established in 2016, with the goal of facilitating gymnastics competitions with its specialized web application. As part of my Master's Degree in Informatics, I will be remotely working with Acro Companion as an intern, tasked of enhancing their competition administration solution.

The current report serves as a comprehensive documentation of my internship with Acro Companion, reporting efforts aimed at creating a fully functional competition administration solution. The primary objective is to provide users with the ability to seamlessly create and manage competitions through an intuitive graphical user interface (GUI).

1.2 Motivation

The primary goal of Acro Companion is to digitalize all gymnastics-related processes of a competition into a single application. The competition management side of a competition is a process that often receives less attention from companies in this field, resulting in existing solutions being simple and missing a lot of key features like creating custom competition phases. To gain a competitive edge, Acro Companion has decided to enhance its current solution by making it simpler to use, but still capable of addressing the majority of client's requested features.

The development of a competition management system also involves integrating the system with other services that are necessary in a competition, such as the scoring and real-time results viewing.

To successfully improve the company's proposed solution, it was essential to become familiar with the technologies and workflows use by Acro Companion, ensuring a more efficient design and development process.

Lastly, it's important to highlight that this project is focused on web development, and that use of the selected technologies were a particular interest of mine, influencing their decision to work on the project.

1.3 Goals

Acro Companion's current solution was facing some difficulties related to stability, accuracy, functionality, and maintenance. This project focuses on addressing these key issues, aiming to stabilize, modernize, improve and expand the current competition management system. The objective is to

transform the application into a more reliable, efficient, and robust tool for competition organizers and future developers.

The main goals in this project include:

- **Stabilizing the Current System** - This involves diagnosing and fixing any issues, bugs, or failures that may disrupt the application normal functioning.
- **Enhancing Data Accuracy, Consistency and Validation** - This involves identifying areas where user-entered data lacks proper validation and implementing mechanisms to handle invalid inputs, thereby minimizing errors. By ensuring the application processes data correctly, data can maintain it's consistency and accuracy.
- **Test Development** - To make sure that the new added functionalities are not breaking the previously implemented features new tests will be created to ensure the system stability.
- **Decrease technical debt** - Technical debt refers to the future cost of rework or maintenance caused by taking shortcuts or suboptimal solutions in software development to achieve short-term goals. Technical debt can be decrease by eliminating unused code, avoiding quick fixes, and taking into consideration code scalability then developing new features.

To effectively measure the achievement of the project goals, a comprehensive set of metrics and evaluation strategies could be implemented. For system stability, monitoring the frequency of bugs, crashes, and system failures before and after the enhancements could reveal a notable decrease in reported issues, signifying improved stability. To evaluate data accuracy, consistency, and validation, thorough data audits could be conducted alongside tracking the occurrence of invalid inputs or inconsistencies; the implementation of robust validation mechanisms is expected to result in fewer data-related errors and greater consistency across datasets.

Test coverage could be assessed using a combination of unit tests, and end-to-end(e2e) tests. An increase in test coverage percentage, along with consistent successful test results, would indicate enhanced system stability and reliability following the introduction of new features. Furthermore, the reduction of technical debt could be quantified through key code quality metrics, including cyclomatic complexity, lines of unused code, and code duplication. Regular code reviews and the use of static analysis tools could help ensure that technical debt is effectively minimized.

By systematically comparing pre and post-implementation metrics, the success of the project and its impact on the competition management system could be assessed both quantitatively and qualitatively.

1.4 Structure of the document

This document is organised as follows:

- Chapter 2 - A brief context is provided on the company's competition management solutions, as well as other competing applications.

- Chapter 3 - Describes the workflow used in the company, the development ad application architecture, and the state of the application at the beginning of the project.
- Chapter 4 - A description of the technologies used to develop the project is provided, along with the purposes for which each was used.
- Chapter 5 - Describes all the implementation process of the project.
- Chapter 6 - In this chapter, a comprehensive reflection on the project was made, along with reflection for potential future improvements.

Chapter 2

State of the art

2.1 Acro Companion solution for competition administration

Acro Companion's first competition administration solution was initially developed specifically for acrobatic gymnastics competitions. This solution provided users with the ability to create, import, and manage data in a much simpler and faster way than traditional tools like Microsoft's Excel or Google Spreadsheets, significantly streamlining the competition management process.

With the onset of the COVID-19 pandemic, Acro Companion adapted to the new reality by introducing a new type of competition—online competitions. This feature allowed users to upload videos of each exercise, replacing the traditional in-person format. Instead of live performances, the videos were presented in the same order as they would be in a standard competition. While this feature offered a creative solution to continue hosting competitions during the pandemic, it is now rarely used as most competitions have returned to their original in-person format.

The success of this adaptation led to increased demand from Acro Companion's clients, who requested the inclusion of additional gymnastics disciplines such as Trampoline, Women's Artistic Gymnastics (WAG), Men's Artistic Gymnastics (MAG), and Rhythmic Gymnastics. Given that these disciplines differ from acrobatic gymnastics but share similar competition structures, Acro Companion developed a more flexible, competition solution that could accommodate all of these disciplines that was named Generic competition administration. This new system allows for easier scalability and versatility for future expansion. However, since this solution was developed rapidly to meet client expectations, there are still some issues to address and features that can be improved.

Despite these challenges, Acro Companion continues to stand out by offering unique features that competitors do not, while placing a strong emphasis on user experience. This focus ensures that creating and managing competitions across various gymnastics disciplines remains simple and intuitive, aligning perfectly with their mission: "Gymnastics made easy."

2.2 Other competition administration solutions

Acro Companion stands on top of competition as the definitive solution for gymnastics competition administration, offering features and versatility that address a wide range of competition man-

agement needs. While competitors like KSIS[12] and Swiss Timing[19] are recognized names in the gymnastics industry, particularly for scoring, they offer more limited functionalities when it comes to competition management. These systems provide essential features such as event location, date, discipline types, staff and athlete level configurations, and exercise creation based on competition entries. However, they do not cover all the capabilities included in Acro Companion, as shown in Table 2.1.

The table displays a direct comparison between Acro Companion and other competitors, highlighting their differences across multiple categories. While KSIS and Swiss Timing excel in real-time scoring and multi-discipline support, their functionalities diverge significantly in other areas. For example, Swiss Timing lacks registration and event management capabilities, which are essential for handling participant data and scheduling. Acro Companion integrates these functionalities into a cohesive system, enabling managers to streamline competition setup and execution.

Livestream integration is another area of distinction. Systems such as GymData[10] and Uplifter[11] provide live result streaming but lack integration for livestream broadcasting. This separation can create inconsistencies in the presentation of results and live coverage. Acro Companion addresses this by combining live result streaming and livestream integration, ensuring synchronized and reliable updates for spectators, officials, and participants.

Scalability is also a significant differentiator. While KSIS and Swiss Timing primarily focus on international events, and systems like Uplifter and GymData are tailored for regional or club-level competitions, Acro Companion adapts to competitions of any size. This flexibility allows it to handle events ranging from small local competitions to large-scale international tournaments, such as the World Championships.

In summary, the table not only shows the presence or absence of specific functionalities across systems but also emphasizes how Acro Companion integrates these features into a single platform.

Gymnast Competition administration software	**Real-time scoring**	**Multi-discipline support**	**Registration and Event Management**	**Live Result Streaming**	**Detailed reports**	**Livestream integration**	**Target Audience**
Acro Companion	✔	✔	✔	✔	✔	✔	Any competition size
KSIS	✔	✔	✔	✔	✔	✔	International competitions
Swiss Timing	✔	✔	✖	✔	✔	✔	International competitions
GymData	✔	✔	✔	✔	✔	✖	Regional/National competitions
Uplifter	✔	✖	✔	✔	✖	✖	Club/Regional competitions

✔ - has feature ✖ - does not have feature

Table 2.1: Gymnastics competition administration systems comparison

Chapter 3

Application state and technologies

3.1 Acro Companion's workflow

Before detailing the work I have completed during my internship at Acro Companion, it's essential to outline the company's Software Development Life Cycle (SDLC) to provide proper context.

The development cycle typically begins with one of two scenarios: either addressing a bug or system crash discovered in the application, or implementing a new feature requested by a client or a senior representative of Acro Companion. Once an issue or feature request is identified, a ticket is created in Azure DevOps Board, which contains key information about the bug or the new functionality. For bugs, the ticket includes instructions on how to replicate the issue, while for new features, it outlines the specific requirements. Additionally, the ticket specifies who will be responsible for handling the task and assigns a priority level, which ranges from 1 (highest priority) to 10 (lowest). A priority of 1 indicates that the problem or feature must be addressed immediately.

Once the ticket is created, the assigned developer begins working on the task. The developer creates a new branch from the main codebase, ensuring that the work is isolated from production while the solution is being developed. This approach helps to maintain code stability, as the main branch is typically the production-ready version that will eventually receive the solution.

After completing the implementation, the developer submits a pull request (PR), initiating the code review process. The review is done in a hierarchical, iterative manner. First, an intern reviews the code and provides feedback. Next, a more experienced member of Acro Companion's core development team takes over, followed by a final review by the team manager. Throughout this process, reviewers can request changes to ensure that the solution aligns with the ticket's objectives and adheres to best coding practices, ensuring clarity, maintainability, and efficiency in the code.

Once all requested changes are addressed, and the reviewers are satisfied with the implementation, the code is approved by each level of review. Before merging into the main branch, the solution must also pass the company's end-to-end (e2e) tests to verify that it does not break existing functionality or introduce new issues. Only after passing these automated tests and receiving final approval is the code merged into the main branch, becoming part of the production-ready

application.

This workflow(see Figure 3.1) not only ensures that bugs are fixed and features are implemented efficiently but also guarantees a high level of quality and consistency in the codebase, while providing a clear structure for collaboration and feedback throughout the development process.

3.2 Development architecture

The development architecture for modern web applications is a multi-layered framework that integrates various tools, platforms, and services to ensure a seamless development workflow. This architecture is designed to support the entire Software Development Life Cycle (SDLC), from writing and testing code to deploying and maintaining the application in production. By utilizing a combination of cloud-based services, local development tools, and browser utilities, developers can effectively collaborate, manage resources, and build robust, scalable web applications.

In this architecture, cloud services such as Azure DevOps and Google Cloud Platform (GCP) are central to managing code repositories, automating deployment pipelines, and providing back-end services like real-time databases, file storage, and authentication. The local environment complements these cloud services by offering essential tools for code writing, testing, and debugging, while emulators allow developers to simulate cloud environments for testing purposes without needing to connect to live production systems.

This layered architecture, with both cloud and local components, enables developers to build, test, and deploy applications efficiently, ensuring both scalability and maintainability throughout the development process.

The cloud environment is categorized into two primary platforms: Azure DevOps and Firebase from Google Cloud Platform (GCP). Azure DevOps is utilized for version control, project management, and continuous integration/continuous deployment (CI/CD) processes, this services are better explained in Section 3.4.1. Google Cloud Platform (GCP) provides back-end services via Firebase for handling several server-side tasks, this services are better explained in Section 3.4.6.

The local environment includes tools and frameworks used directly by developers. It consists of three main parts: the browser and Integrated Development Environment (IDE) and a database emulator for the end-to-end(E2E) tests.

Browser In the browser, there are three main tools used:

- **Firebase SDK**: The Software Development Kit used to connect the web app to Firebase services like the database, authentication, and storage.
- **Angular Framework**: The front-end framework used to build the web application's user interface.
- **Browser DevTools**: Tools such as Chrome DevTools, used for debugging, testing, and performance inspection.

Integrated Development Environment (IDE) The IDE is the primary space where developers write and manage their code. It includes:

- **Code Editor**: Where developers write the application code (e.g., Visual Studio Code).
- **Git**: Version control system for tracking changes and collaboration.
- **Terminal**: Command-line interface for running commands, scripts, and managing Git.
- **Playwright**: A tool for automating browser testing to ensure the web app behaves as expected.

3. Emulator Emulators allow developers to simulate cloud services in a local environment for testing, eliminating the need to connect to live cloud infrastructure. These emulators include:

- **Authentication Emulator**: Simulates Firebase Authentication for testing user login processes.
- **Storage Emulator**: Simulates Firebase Storage for testing file upload and download functionality.
- **Database Emulator**: Simulates the Firestore Database for testing database interactions locally.

This development environment(see Figure 3.2) provides a full-stack architecture for building a web application using Angular for the front end and Firebase services for the back end, all managed with Azure DevOps for version control, project management, and CI/CD. The environment supports cloud-based services and includes local development tools, such as browser-based debugging tools and emulators for testing, ensuring that the application can be developed and tested effectively before deployment.

3.3 Application architecture

Acro Companion's front-end application is built using the Angular framework, which uses TypeScript to create a dynamic and interactive graphical user interface (GUI). Angular's robust features provide a structured and scalable way to develop web applications, particularly well-suited for large-scale projects.

In addition to Angular, Acro Companion utilizes Firebase to manage critical back-end operations. Firebase is a suite of services within Google Cloud Platform(GCP) that plays a key role in real-time database management, user authentication, file storage, and serverless functions that are essential for specific back-end tasks. Firebase integrates seamlessly with Angular, enabling real-time data updates and simplifying complex operations like authentication and cloud storage management.

Angular[1] is a robust TypeScript-based framework used to build Single Page Applications (SPAs). It is responsible for rendering the GUI, handling routing, and managing client-side state.

Angular follows a component-based architecture this means that each feature or view of the application is broken down into reusable components. These components are bundled into modules to organize the app structure.

- **Components & Modules**: Angular follows a component-based architecture this means that each feature or view of the application is broken down into reusable components. These components are bundled into modules to organize the app structure.
- **Routing**: Angular's built-in router enables the application to load different views without reloading the entire page, improving user experience.
- **Service Layer**: Angular services manage business logic and make HTTP requests to Firebase. They ensure that components remain focused on UI concerns while services handle data interaction and processing.
- **State Management**: For handling complex state, Angular can integrate with libraries like NgRx or Angular Signals, although Firebase's real-time database often minimizes the need for elaborate state management solutions in simpler apps.
- **Forms and Data Validation**: Angular provides robust support for building and validating forms, whether reactive or template-driven.

Firebase[8] is a comprehensive cloud-based platform that provides several backend services, including authentication, real-time database, Firestore (NoSQL database), and cloud functions. It eliminates the need for traditional servers and databases.

- **Authentication**: Firebase Authentication provides a variety of user authentication methods like email/password, Google, Facebook, and others. This is managed directly from Angular services that interact with Firebase's SDK.
- **Firestore/Real-time Database**: Firebase offers two database options:
  - **Firestore**: A NoSQL, document-based database that provides rich querying capabilities and real-time updates.
  - **Real-time Database**: A simpler, tree-structured database with real-time synchronization, ideal for live updates.
- **Cloud Functions**: For server-side logic, Firebase Cloud Functions can be used to write backend logic that responds to database changes, HTTP requests, or authentication events. This provides a way to run backend code without managing servers.
- **Cloud Storage**: Firebase also supports file uploads and management through Cloud Storage, which integrates with Firestore for metadata and with Firebase Authentication for access control.

Angular interacts with Firebase through Firebase's SDK, which provides APIs for authentication, database operations, and cloud functions. Angular services make HTTP requests or use WebSockets to communicate with Firebase, ensuring real-time synchronization between the client and the backend.

This combination(see Figure 3.3) of Angular's powerful front-end capabilities with Firebase's cloud-based services allows developers to focus more on implementing new features and solving user problems, without being bogged down by infrastructure management. Together, they provide a scalable and efficient development environment for Acro Companion's evolving needs.

3.4 Tecnologies

In today's fast-paced software development environment, leveraging effective tools and frameworks is essential for project success. This document examines key components that facilitate modern application development, including work management tools, integrated development environments (IDEs), testing frameworks, programming languages, and backend solutions.

Azure DevOps serves as a cornerstone for managing the application lifecycle, supporting continuous integration and delivery. Visual Studio Code offers developers a versatile IDE for efficient coding and debugging. Testing is streamlined through frameworks like Playwright, while reactive programming with libraries such as RxJS enhances the handling of asynchronous data.

The choice of TypeScript as a programming language brings added reliability through its static typing system, and Angular is highlighted for its component-based architecture that supports scalable applications. Additionally, Firebase provides a powerful backend solution with real-time data synchronization capabilities, further enriching the development ecosystem.

3.4.1 Work management tools

To manage the work in progress, Acro Companion uses Azure DevOps[21]. Azure DevOps is a platform created by Microsoft that allows developers and project managers to easily manage an application lifecycle. This platform's biggest strength is the support for continuous integration (CI), continuous delivery (CD), and continuous testing (CT), that allows for developers to continuously improve and or fix problems in the deployed application without the need to shut it down every time a change is made.

- **Azure Boards** - used to create tickets that provide important information about the bugs and features that need to be implemented by the developers.
- **Azure Repos** - cloud-based repository with a version control system that enables development teams to store and manage the application source code.
- **Azure Pipelines** - Continuous delivery(CD), continuous integration(CI) and continuous testing (CT) service that helps developers to automate the building, testing and deployment

of an application. This service makes this possible by defining a series of steps and tasks in a pipeline that every code change has to successfully pass.

Integrated Development Environment

An Integrated Development Environment (IDE)[5] is a software application that provides developers with comprehensive tools to write, test, and debug code efficiently. IDEs typically include a code editor, a debugger, and build automation tools all within a unified interface, streamlining the development process. By offering features like syntax highlighting, code completion, version control integration, and error detection, IDEs help developers write cleaner, more maintainable code while reducing the time spent switching between different tools.

The IDE I used was Visual Studio Code (VS Code)[18], developed by Microsoft. VS Code is lightweight, open-source, and supports a wide range of programming languages through extensions. It offers features like IntelliSense for smart code completions, built-in Git[9] integration for version control, and an integrated terminal for running commands. Additionally, its customizable interface and extensive library of extensions make it a versatile choice for developers working on everything from web applications to system programming.

3.4.2 Testing tools

Acro Companion has developed a custom framework for writing its end-to-end (E2E) test suites, which are executed using Playwright[14], a robust testing framework developed by Microsoft specifically for E2E testing and browser automation. Playwright supports multiple programming languages, including JavaScript, Java, and Python. However, as Acro Companion's platform is built using Angular, the test suites are primarily written in TypeScript to maintain consistency with the application's codebase. Playwright's powerful cross-browser capabilities and its seamless integration with modern web technologies make it an ideal choice for ensuring that Acro Companion's web application functions reliably across different environments, while automating complex testing scenarios with ease.

3.4.3 Reactive programming

Reactive programming[13] is a programming paradigm that focuses on the flow of data and the propagation of change. It allows developers to work with asynchronous data streams and react to changes as they occur. For Angular applications RxJS[15] is the most used library to manage asynchronous operations and handle reactive programming patterns. RxJS provides a comprehensive set of tools for working with Observables, allowing Angular developers to seamlessly implement features such as real-time updates, user input handling, and HTTP request management. The Observables in RxJS empower developers to compose complex asynchronous workflows in a more readable and maintainable manner.

Recently, Angular introduced a new reactive primitive called Signals[2], which provides a simpler and more intuitive approach to handling reactivity within Angular applications. Unlike

RxJS Observables, which are ideal for complex asynchronous workflows, Signals offer a more declarative way to manage state and reactivity. Signals track and automatically update values when dependencies change, removing the need for manual subscriptions. This reduces boilerplate code, especially for managing local component state. While RxJS excels in managing event streams and asynchronous operations, Signals focus on simplifying the reactive state management within Angular components, making it easier to keep the UI in sync with underlying data changes. This makes Signals a powerful addition to Angular's reactive programming toolkit, especially for applications where state updates are frequent and performance optimization is critical.

3.4.4 Programming language

TypeScript[20], a superset of JavaScript developed by Microsoft in 2012, was designed to overcome some of the key limitations of JavaScript, particularly the lack of a strong type system. JavaScript's dynamic typing allows variables to hold values of any type, offering flexibility but also introducing the risk of runtime errors that are difficult to detect during development. TypeScript addresses this by introducing static typing, which helps developers define explicit types for variables, functions, and objects. This type system significantly improves code reliability, enabling errors to be caught at compile time rather than during execution.

In addition to enhancing error detection, TypeScript promotes the development of more maintainable and scalable codebases. The static type definitions make code easier to understand, reducing ambiguity and making large projects more manageable. By combining the flexibility of JavaScript with the rigor of a type system, TypeScript strikes a balance that empowers developers to write more robust, readable, and self-documenting code. This makes it a valuable tool for teams building complex applications that require long-term maintenance and collaboration.

3.4.5 Framework

Angular[1] is one of the leading front-end development frameworks, widely recognized for its powerful features and comprehensive ecosystem. Unlike simpler approaches using just HTML, JavaScript, and CSS, Angular stands out through its adoption of a component-based architecture. This architecture promotes the development of modular, reusable code, which enhances scalability and maintainability in large applications.

Angular's structured framework is particularly well-suited for building single-page applications (SPAs), providing developers with tools to organize their code efficiently and manage complex UI interactions. At the heart of an Angular application are components and services. Components serve as the building blocks of the user interface, responsible for rendering and handling the presentation layer, while services manage shared functionalities such as business logic and data manipulation. This separation of concerns allows for a clear division between UI and logic, making the codebase easier to maintain and extend over time.

Additionally, Angular's robust dependency injection system and built-in tools for state management further enhance code reusability, while its ecosystem supports testing, routing, and HTTP

client services, making it a highly efficient choice for developing modern, feature-rich web applications

3.4.6 Firebase

Firebase[8] is a cloud-based NoSQL database provided by Google Cloud Platform[17], offering a robust back-end solution that goes beyond simple data storage. As part of the Firebase platform, it enables real-time synchronization of data across clients, making it ideal for building collaborative, dynamic applications that require instant updates. Firebase's flexible NoSQL structure allows developers to design and manage data in a non-relational format, accommodating diverse data models without the rigid constraints of traditional databases. One of Firebase's standout features is its real-time synchronization, which ensures that any changes in the data are instantly propagated to all connected clients, maintaining a seamless user experience.

However, Firebase is more than just a database. It offers a comprehensive suite of back-end services, including user authentication, cloud functions for executing server-side logic, static and dynamic hosting, and even push notifications. This all-in-one platform simplifies back-end development by handling server infrastructure, scaling, and security, allowing developers to focus on building core features and improving user engagement. By removing the complexities of back-end management, Firebase accelerates the development process, making it an invaluable tool for both startups and large-scale applications.

3.5 Competition administration solution

With the competition administration system being the focus of this project, it is important to explain how this system works, who are the target users and what are the use cases.

The competition administration system is destined to be used by competition managers so that they can set up the competition based on their needs. It is also important to reference that a competition itself is created by Acro Companion developers to reduce the risk of mistakes when creating the competition features, so only managers insert data into the competition.

Figure 3.4 represents the use cases and some of functionalities the system currently offers. To better understand these use cases there are a couple of concepts that must be explained.

- **Specification combinations**: Area of the application where managers can setup combinations between levels, categories, competition phases and discipline specific exercises to later help in the creation of exercises. In Figure 3.5 it is possible to see the Specification Combinations area for a trampoline competition.
- **Competition zones**: A competition zone is a designated area where gymnasts or teams perform their exercises during a competition.
- **Panels**: As competition zones can be divided into multiple sub zones, each sub zone needs a group of judges to give a score to the gymnast or team, and that group of judges is called

a panel or judge panel.

- **Gymnast and teams**: Gymnast are the athletes that perform the exercises created by the competition managers. Some exercises need to be performed by more than one gymnast, so teams of gymnasts are created to perform those exercises.
- **Judges**: Judges will give a scores to the gymnasts or teams performing exercises.
- **Judges seats**: A panel has various types of judges and when judges are assigned to a panel they are given a specific role in that panel that is represented by a judge seat, as can be see on Figure 3.6.
- **Exercises**: An exercise is defined by its discipline, category and whether it is performed by a gymnast or a team. Gymnasts and teams can perform multiple exercises.
- **Starting order**: After exercises are created they are performed in the order they are assigned when they are created. This order can be changed in the starting order page. Figure 3.7 represents a set of exercises designed for the hoop category in a rhythmic gymnastics competition.
- **Save and restore data**: Whenever a manager feels that they are done setting up the administration part of the competition they can save the changes that were made. If for some reason the manager feels that they need to go back to a version of the administration that was previously saved it can be done in the backup page.

With these concepts in mind, it's important to consider the following use cases:

- **Create/Edit Competition Specification Combinations, Competition Zones, and Panels:** This functionality allows competition managers to define specification combinations, ensuring they align with discipline-specific rules. Managers can also configure competition zones where performances occur and assign judge panels to these zones for evaluation.
- **Create/Edit Gymnasts, Teams, Judges, and Exercises:** Managers can add or update information about individual gymnasts, teams, and judges participating in the competition. Additionally, they can define exercises, specifying the discipline, category, and whether they are performed individually or in teams.
- **Assign Gymnasts and/or Teams to Exercises:** This feature enables managers to link gymnasts or teams to specific exercises, ensuring proper alignment with the competition structure and rules.
- **Assign Judges to Judges' Seats:** Judges are assigned to specific seats within panels, each corresponding to a predefined role, such as execution or difficulty judge. This ensures clarity in judging responsibilities during performances.

- **Edit Exercises Starting Order:** Managers can adjust the sequence in which gymnasts or teams perform their exercises. This functionality helps address scheduling issues and ensures a smooth competition flow.
- **Save Competition Administration to the Database and Restore Data from Previous Saves:** Managers can save the current competition configuration to the database, preserving all settings and data. If necessary, they can restore previous versions, preventing data loss or reverting to a stable state after errors.
- **Import and Export Data from the Competition:** This feature facilitates data management by allowing managers to import relevant information, such as gymnast details or exercise configurations, and export data for reporting, backup, or further analysis.

The competition administration system provides essential tools for managing gymnastics events, focusing on the tasks required by competition managers. It covers everything from configuring specification combinations, competition zones, and panels to managing gymnasts, teams, judges, and exercises. The system also allows assigning gymnasts and teams to exercises, assigning judges to seats, adjusting the starting order of exercises, and saving or restoring competition data. Additionally, the ability to import and export data simplifies managing and sharing information. These functionalities ensure that the competition setup process is organized, consistent, and aligned with the requirements of gymnastics events.

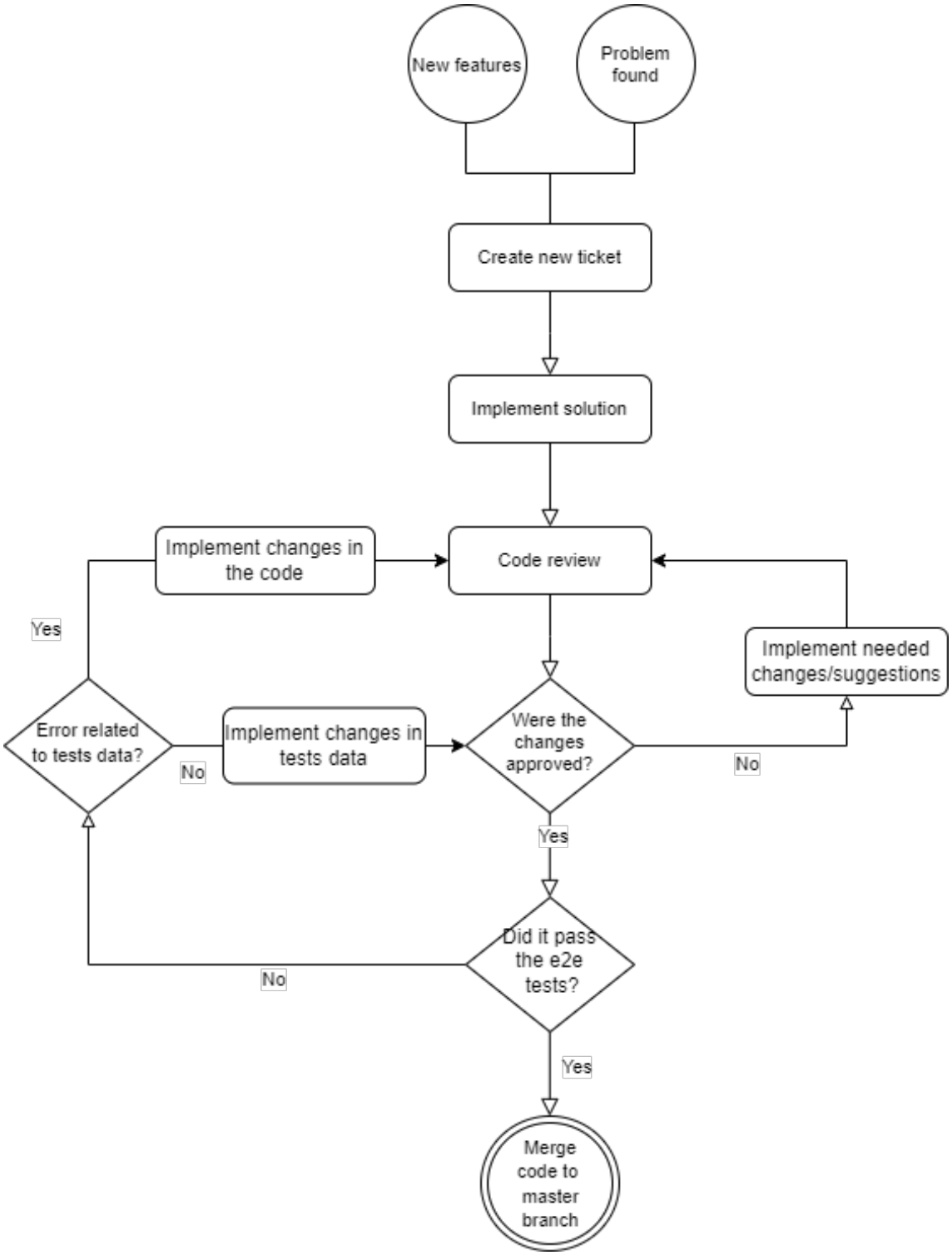

Figure 3.1: Acro Companion workflow

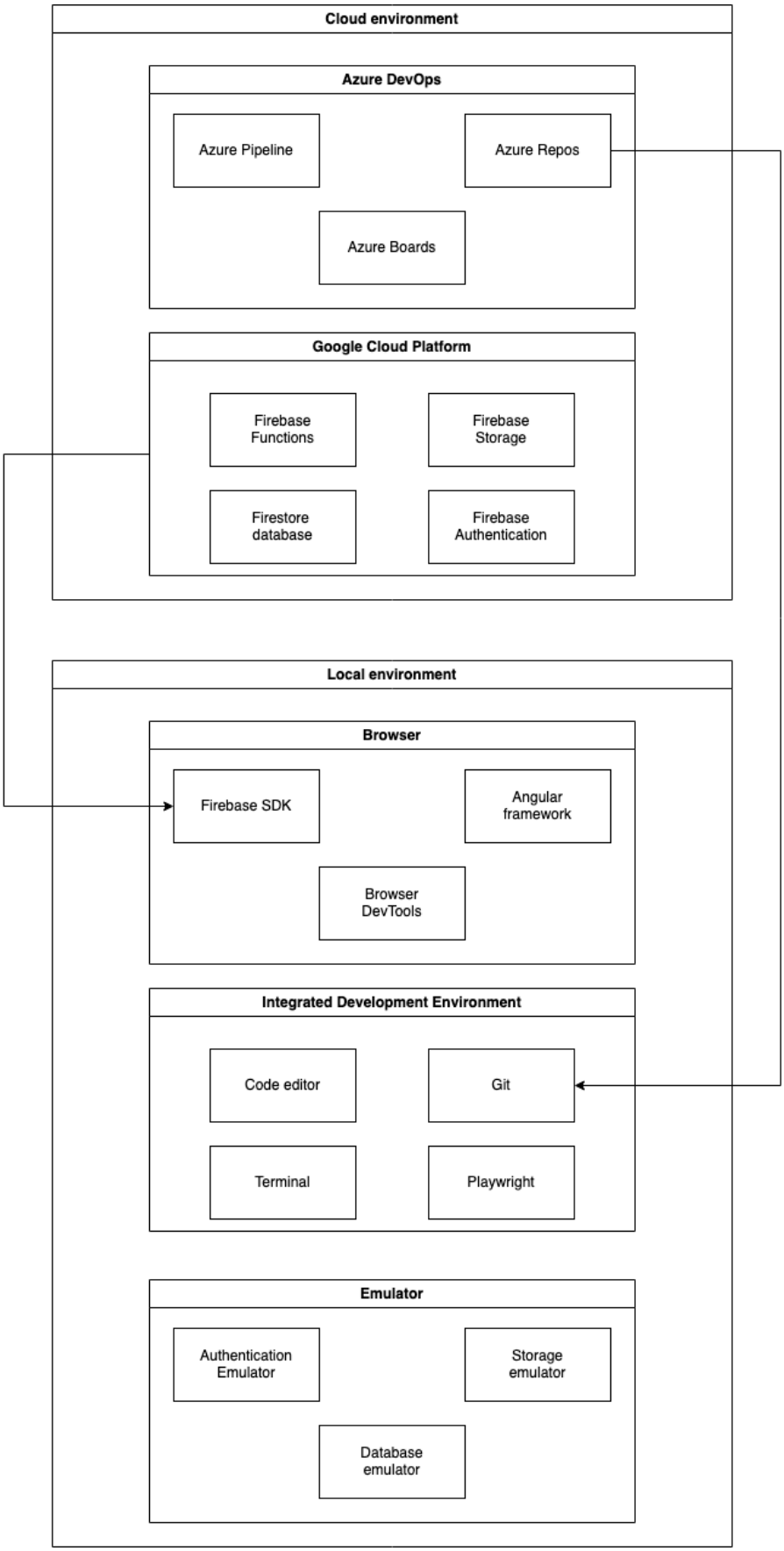

Figure 3.2: Development architecture

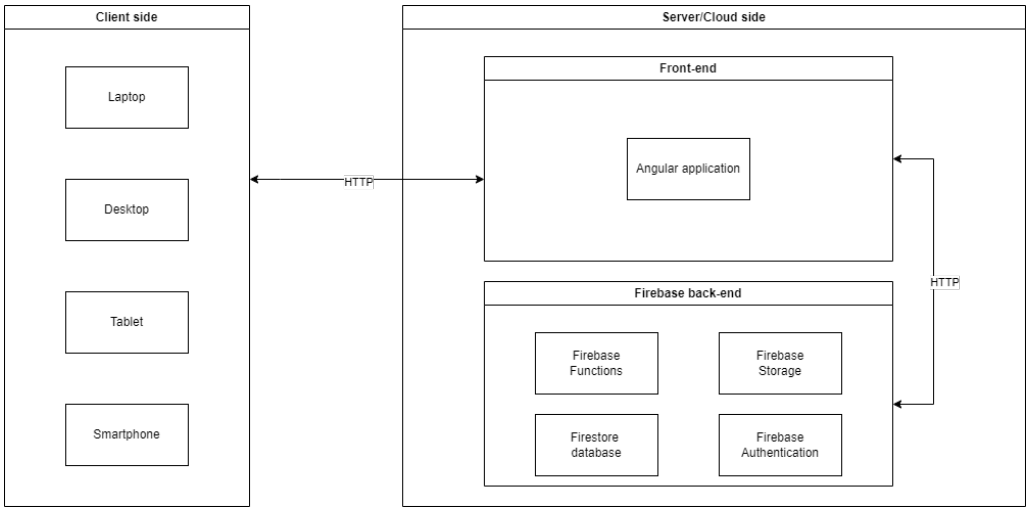

Figure 3.3: Application architecture

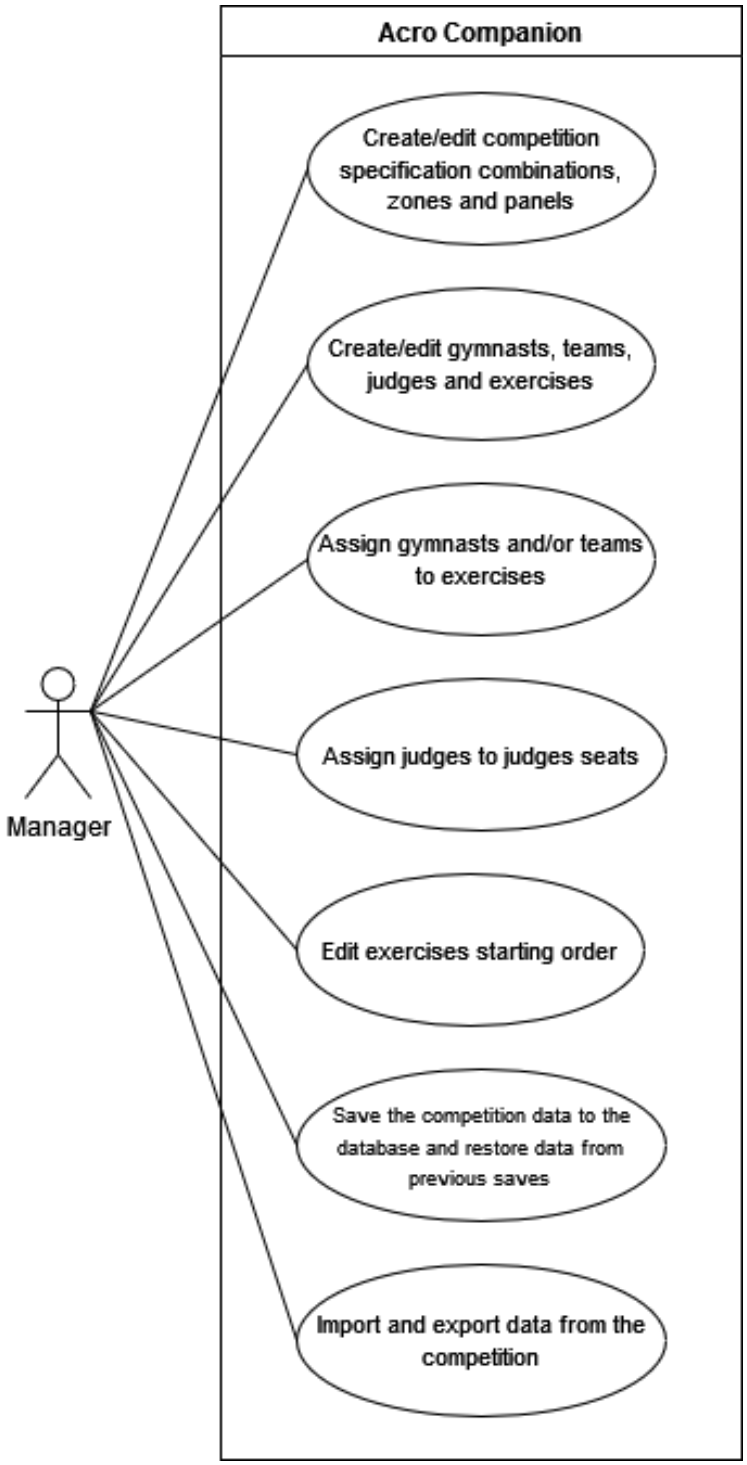

Figure 3.4: Gymnastics Competition Administration use cases

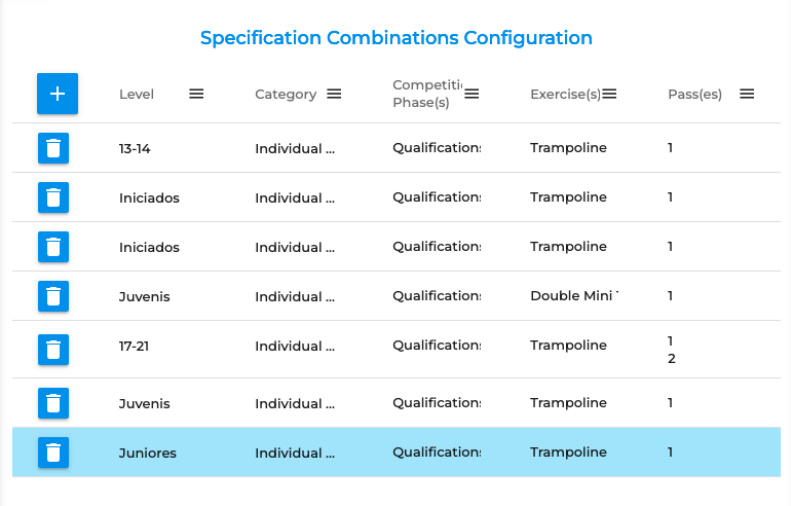

Figure 3.5: Specification combination configuration example

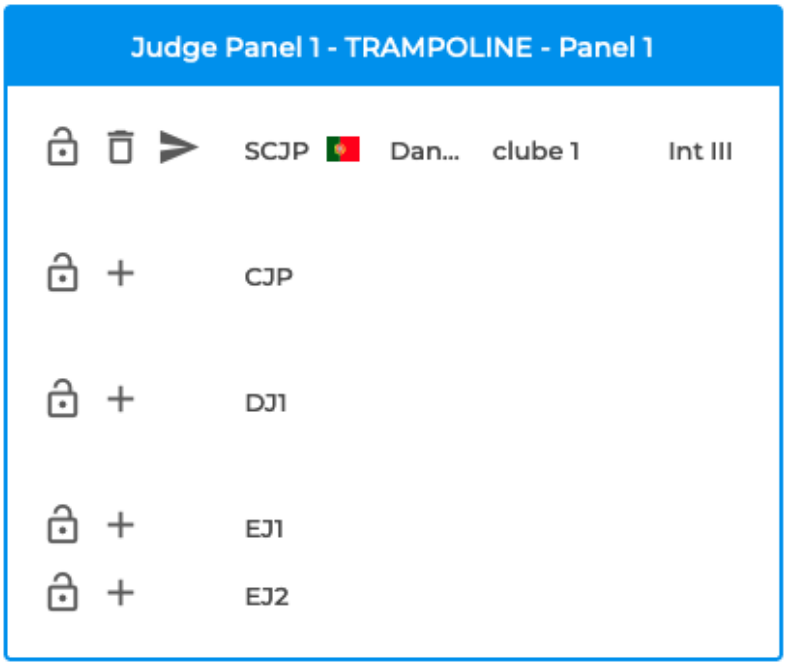

Figure 3.6: Judges seats example

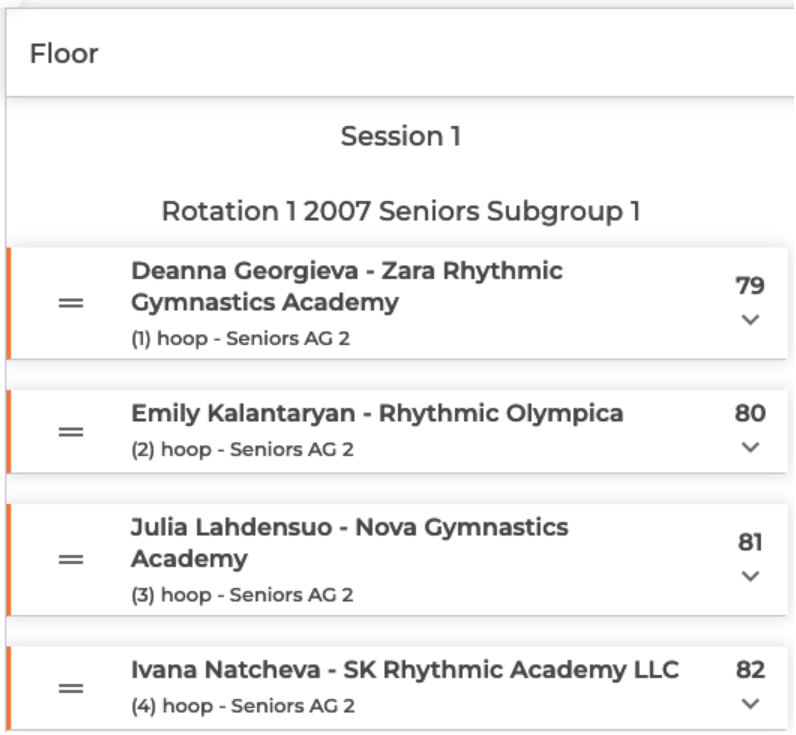

Figure 3.7: Competition exercises example

Chapter 4

System enhancements and features implementation

This chapter provides a more detailed view of the improvements made to the competition administration system throughout this project. These contributions address challenges related to system stability, data integrity, user interface usability, and overall performance. The modifications cover improvements to existing features, as well as the addition of new ones. Each change was implemented with the goal of simplifying user workflows, ensuring data accuracy, and improving the system's scalability. Where relevant, this chapter highlights the reasoning behind these changes, their implementation details, and their expected impact on the Acro Companion system. Additionally, it is worth mentioning that new end-to-end (E2E) tests were created to ensure that all existing features remain functional, even if some now require more advanced configuration to operate correctly.

4.1 Improvements to existing solution

4.1.1 Applying patcher when restoring backups

When restoring database backups, if the backup model version is older than the latest version, patches created after the backup model version must be applied so that data can stay consistent.

In applications that use Angular and Firebase together, models (data structures) evolve over time due to new features, changing requirements, or performance optimizations. Failure to update or patch these models can lead to issues like data inconsistencies, application crashes, and poor user experiences.

A critical reason for patching models is ensuring data consistency between the Angular application and Firebase. As the application evolves, the data model may need to be updated to reflect changes in how data is structured or used, such as adding new fields or modifying how timestamps are stored. The data model in Angular (TypeScript interfaces, data objects, etc.) must align with the structure in Firebase. If the client model is outdated, it may not properly process data from Firebase, leading to errors or data corruption. Outdated models can result in undefined fields or incorrect data types, causing the application to write incorrect or corrupted data into Firebase.

Using Firebase with a NoSQL database, such as Firestore or Realtime Database, allows for flexible schema design, but this flexibility can create challenges when schema changes occur. For example, if a new field is added to a Firestore document, the Angular model must be patched to accommodate the new structure. If a new field like a phone number is added to a user's profile, both the Angular model and Firebase queries must be updated to handle this new data correctly. If the application still interacts with older data versions, the model patcher ensures backward compatibility, preventing issues with deprecated fields and outdated data.

Patching outdated models can lead to performance improvements by optimizing data structures or eliminating unused fields. A patcher can update models to only fetch the required data, reducing bandwidth usage and improving load times. Keeping models updated also ensures that queries in Firebase are efficient and reflect recent optimizations, such as indexing or field changes, resulting in smoother application performance.

Firebase uses Security Rules to control access to data, and if the data model changes without updating these rules, vulnerabilities can arise. For instance, adding a sensitive field like user roles requires an update to the security rules to prevent unauthorized users from accessing that data. A model patcher ensures that new fields or model changes are reflected in Firebase Security Rules to prevent unauthorized access and avoid potential data leaks.

Neglecting to patch models can lead to technical debt, making the system harder to maintain over time. The longer models remain outdated, the harder it becomes to patch them without introducing errors. When model updates are skipped, the Angular application's data objects can become out of sync with the actual data in Firebase, leading to bugs and inconsistencies. Regular patching helps maintain alignment between the models and the real-world data structure, reducing technical debt.

When a Firebase model changes, existing data often needs to be migrated to match the new schema. For instance, if a "birthdate" field is added to user profiles, all existing profiles may need to be updated with default or computed values. A model patcher can automate this process, reducing manual effort and preventing human errors. During migration, model patchers can safely handle complex data transformations, ensuring that no data is lost or corrupted.

Several best practices are essential for implementing a model patcher in Angular-Firebase applications. Using TypeScript's strong typing system ensures that changes in Firebase data structures will cause compile-time errors in Angular, catching issues early. Versioning models helps manage backward compatibility, allowing old data to be handled while new models are patched in. Automating data migrations using Firebase Cloud Functions or scripts helps maintain database consistency with updated models. Testing model changes and migrations in a staging environment ensures compatibility with both new and old data. Updating Firebase Security Rules when patching models that involve sensitive data ensures that the application remains secure as the data model evolves.

In an Angular-Firebase application, model patching is essential to ensure data consistency, performance, security, and long-term system integrity. Regularly applying patches helps prevent

issues and reduces the complexity of future updates.

4.1.2 Trimming strings when saving to the database

In today's modern web era, ensuring the robustness and integrity of data is crucial, particularly in applications that handle large volumes of user-generated content. One of the most common yet often overlooked issues in such scenarios is the presence of leading and trailing spaces in text inputs. These extra spaces can significantly compromise data consistency, search accuracy, and overall system performance. For example, in the context of competition management, a simple leading space in a name field could prevent a manager from finding a gymnast or judge in the competition administration system, leading to failed searches and administrative inefficiencies.

The root of this problem lies in how text data is handled before it is inserted into the database. Inconsistent handling of spaces can create mismatches between what users input and what is stored in the system, causing confusion and unexpected behavior. A name entered as "Daniel Cambinda " (with a trailing space) will not match "Daniel Cambinda" during a search, even though the two values appear identical to the user. This can lead to search failures, inaccurate results, and difficulties in managing large datasets.

To solve this issue, there are two common approaches:

- Trim Text Fields Before Database Insertion: By automatically removing any unnecessary leading or trailing spaces from all text inputs before they are stored in the database, you ensure data consistency and avoid errors related to extra spaces.
- Restrict Input Fields to Prevent Leading and Trailing Spaces: This approach involves setting input field restrictions, so users are prevented from entering data with leading or trailing spaces in the first place.

However, given the size and complexity of Acro Companion's application and the vast number of text input fields it manages, the most practical and scalable solution is to trim all text fields automatically before they are inserted into the database. This approach provides a clean, consistent dataset without requiring manual input restrictions on each individual field, ensuring smooth operation at scale.

Implementing this solution involves iterating through every array and object in the document being saved and applying the trimming operation to every field of type string. By removing unnecessary spaces before the data is committed to the database, you can significantly improve search performance, data retrieval accuracy, and overall system efficiency. This not only helps in preventing issues like failed searches but also ensures that the data remains clean, consistent, and easy to manage over time.

Moreover, trimming input data before database insertion leads to additional benefits, including:

- **Improved Query Performance** - Consistent data ensures that searches, queries, and comparisons operate more efficiently. The database can retrieve results faster, improving user

experience and reducing system load.

- **Optimized Storage Usage** - Removing extra spaces reduces the overall data size, which can accumulate over time in large applications. While the savings may seem minor on a per-entry basis, they add up in databases with millions of records, leading to better storage optimization and potentially lower costs.
- **Better Data Validation** - When text fields are trimmed, validations based on character count, format, or other constraints function more accurately. For instance, if a field is limited to 50 characters, an extra space won't cause a validation failure.
- **Increased Data Integrity** - Trimming strings helps prevent data duplication due to minor differences like leading or trailing spaces. This reduces the risk of storing redundant or erroneous data, which can be especially problematic in applications with large, interconnected datasets.

Additionally, trimming helps ensure that the data stored in the database remains secure. By eliminating unwanted spaces and hidden characters, you reduce the likelihood of certain types of input-based security vulnerabilities, such as SQL injection or manipulation attacks, which can exploit poorly handled user inputs.

In summary, trimming text fields before inserting data into the database is a critical practice for maintaining data integrity, improving search efficiency, optimizing storage, and preventing user frustration. It ensures that your system operates smoothly and that your database remains clean, consistent, and performant over time. This simple, yet effective, solution is key to building a reliable and scalable application, especially one like Acro Companion, which deals with a significant amount of user-generated data.

4.1.3 Improvements to generic competitions specification combinations

As stated before specification combinations is an area in the generic competition administration system where competition managers can setup the combinations between levels, categories, competition phases and disciplines specific exercises that are defined for a competition. The process of creating a specification combination can be challenging at first, because it involves multiple variables and settings that must be carefully aligned to meet the competition's structure and rules.

This complexity can sometimes result in errors during the setup process, such as incorrect category assignments or mismatched levels and phases, which can cause confusion and disrupt the flow of the competition. To address these challenges and enhance the user experience, several improvements have been made to simplify configuration and reduce the likelihood of mistakes.

In order to reduce of these configurations mistakes and help managers setup their competition in a simpler and faster way some improvements were added to this area of the application:

- **Add a warning notifying that recent changes contain impossible combinations** - For each discipline, there are some specifications that are considered impossible. For exam-

ple, in the Trampoline, Double Mini-Trampoline, and Tumbling (T&T&DMT) discipline, specifications that have the ”Duo Man” or ” Duo Woman” categories and the ”Double Mini-Trampoline” or ”Tumbling” exercises are considered impossible since these exercises have to be performed by one athlete at a time. After filling all the fields in a specification combination if there is any of these impossible combinations the system displays a message with the text: ’The most recently edited specification combination is not possible. It will not be possible to create exercises for this combination(see Figure 4.1).

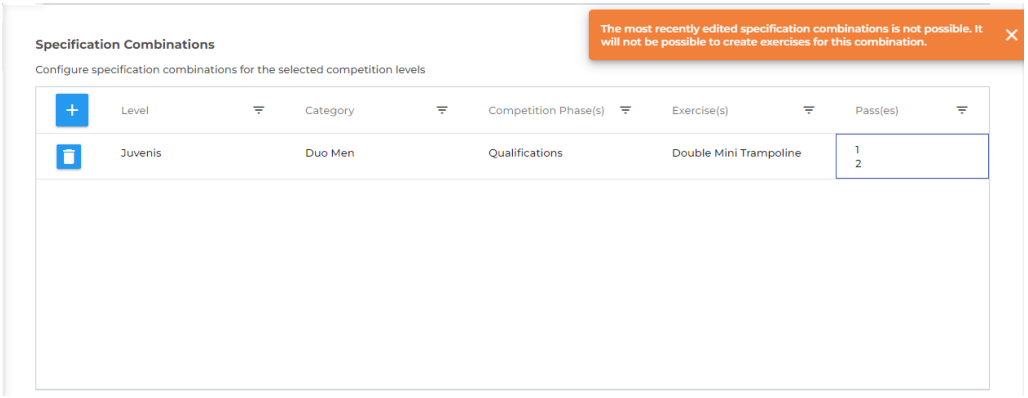

Figure 4.1: Impossible combination warning

- **Auto-assign category when only one category is available** - In order to reduce this area’s complexity and and to make the creation of specifications faster and more enjoyable for managers the category field was hidden for disciplines that only have 1 category available for selection. The disciplines where this happens are Women’s Artistic Gymnastics (WAG) and Men’s Artistic Gymnastics (MAG) that only have the discipline ”Women” and ”Men” respectively. This happens only for these disciplines because they are gender specific. Hiding this field does not happen to all gender specific disciplines because some of have individual and group categories. In Figure 4.2 it is possible to see a specification combination table with no ”Category” field.

- **Allow selection of multiple levels and categories in a single specification** - Another improvement made to decrease the specification creation time was making the fields ”Level” and ”Category” multi-select (see Figure 4.3. This change reduces the amount of specifications that need to be created making it possible managers to create all the desired combinations in only one specification.

  This change takes into account that there are some rules for the creation of specification combinations. Some disciplines have a set of combinations that are considered impossible, for example exercises for a pair or group can not be performed by a single athlete individually. So taking this into account these impossible combinations can be selected but are ”ignored” and it is only possible to create exercises for valid combinations.

  Because this is a change that also implicated changes in the Specification Combinations data

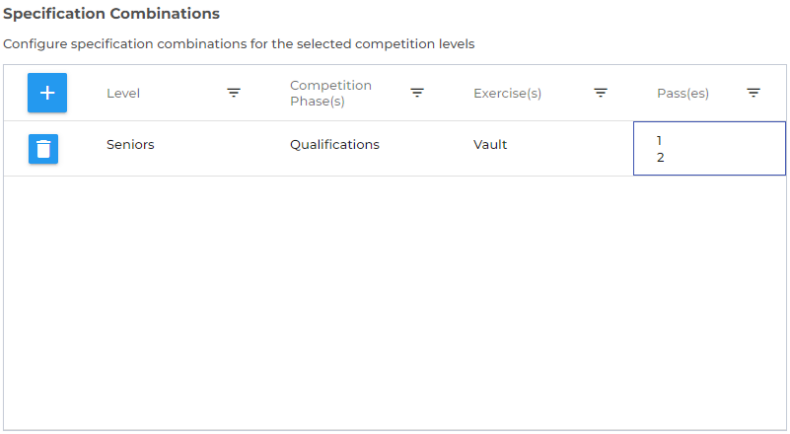

Figure 4.2: Specification combinations with no category

model, a patcher had to be created to adapt older model versions. This patcher transformed the previous simple string fields into arrays of string with the old level or category value being added to the new array.

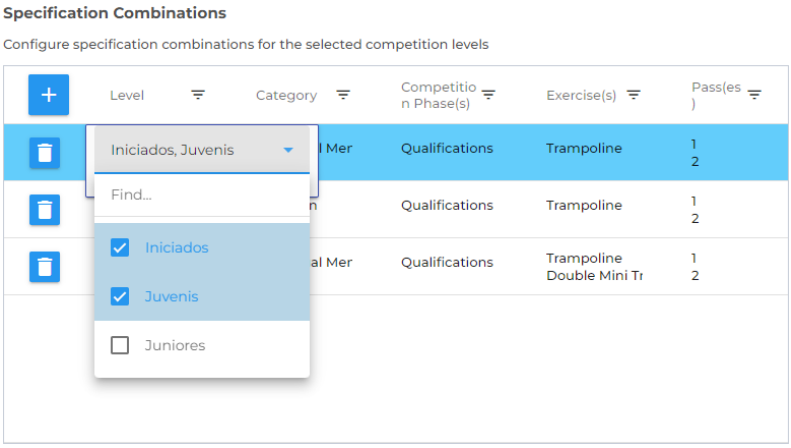

Figure 4.3: Multi-select for the level field in specification combinations

- **Do not allow specifications with only a pass 2** - One of the more recurrent mistakes when defining specification combinations is when it comes to selecting the passes. The ”Passes” field allows the selection of what passes are going to exist for that specification but many competition managers thought that this field is to select the amount of passes for the specification. This mistake leads many specification combinations to being incorrect but since there was no validation for this field exercises could be created with only a pass 2 and no pass 1, which makes no sense in gymnastics since a pass 2 can only exist if a pass 1 exists. To solve this issue a field validator was added to this field. This validator checks if the specification that is being added/edited has the pass 1 selected or if there is any other specification combination with the same level, category, competition phase, and exercises combination that has the pass 1 selected since some managers prefer to create a specifi-

cation for each level, category, competition phase, exercise and pass combination. If the validator does not find any similar specification with a pass 1 selected the passes field of the current specification becomes red and if the cell is hovered with the mouse display the message ”Specification combination has pass 2 with no pass 1”. A visual representation of this validator can be seen in Figure 4.4.

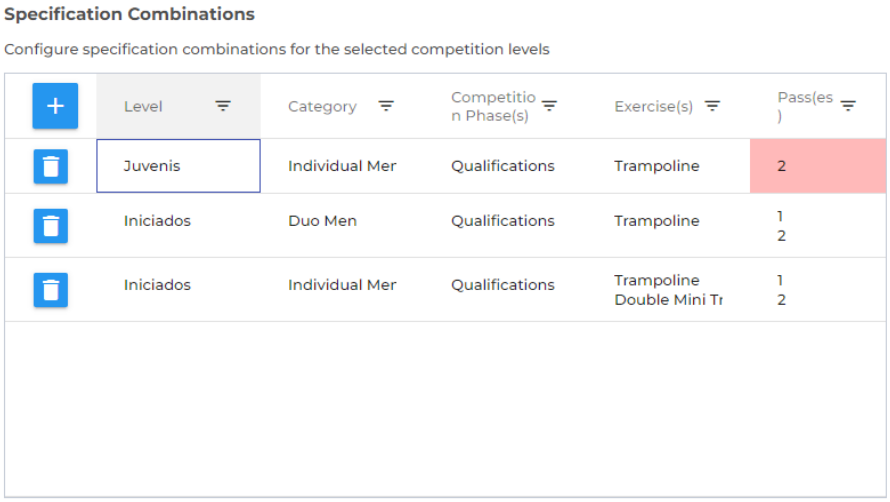

Figure 4.4: Error in specification combination with no pass 1

4.1.4 Improvements to tab navigation and import/export section

In the realm of software design and user experience, a fundamental challenge lies in balancing complexity with simplicity. As applications grow in functionality, they often introduce features, tabs, and options that cater to a wide range of use cases and configurations. However, many of these features may not always be relevant to the user’s immediate task or context. One effective solution to this challenge is the strategic hiding of non-essential tabs, columns, and input fields—particularly when they pertain to configurations or settings that are not applicable in a given scenario.

This refactor delves into the practical application of this approach within the context of a gymnastics competition management system, specifically focusing on the advantages of hiding configuration-related tabs and validators in forms where certain settings are not necessary. Through case studies of dynamically concealed elements and improved error validation processes, this refactor argues that this approach not only simplifies the user interface (UI) but also enhances overall usability without introducing significant drawbacks.

The gymnastics competition management system, as many enterprise-level applications, needs to accommodate a wide variety of configurations depending on the structure of a given competition. These configurations might include distinctions between team-based and individual categories, various gender-specific disciplines, and hierarchical levels within the sport. As a result, the system needs to provide forms and tabs that handle these diverse scenarios. However, not all configurations are relevant at any given time. For example, if no team categories are specified in the

competition settings, displaying the Teams tab becomes unnecessary. Similarly, in competitions that focus solely on individual categories, the Category and Level columns in the Gymnasts tab serve no purpose. The presence of irrelevant tabs and fields not only clutters the interface but can also confuse users, leading to unnecessary navigation and a higher likelihood of errors.

This refactor proposes hiding these non-relevant UI elements, such as the Teams tab when no team categories are configured, or the Category and Level columns when only individual categories apply, is a powerful means of simplifying the interface while maintaining the full range of functionality. By implementing dynamic visibility rules, where certain elements only appear when they are contextually relevant, users are guided toward the most pertinent actions and information. This practice of progressive disclosure—where advanced or less-frequent options are only shown when needed—ensures that novice users are not overwhelmed by complexity, while advanced users still have access to deeper functionality when applicable.

A key example of this is the decision to hide the Teams tab entirely if no team categories are specified in the competition's Specification Combinations(see Figure 4.5). In cases where a competition involves only individual gymnasts and no team events, the existence of a Teams tab would not only be unnecessary but could also mislead users into thinking that team configuration is required. By dynamically hiding this tab in such scenarios, the system remains focused on the task at hand, allowing users to proceed through the interface with greater confidence and fewer distractions. In Figure 4.6 we can see the the Teams tab becomes visible when a team category is selected in the specification combinations.

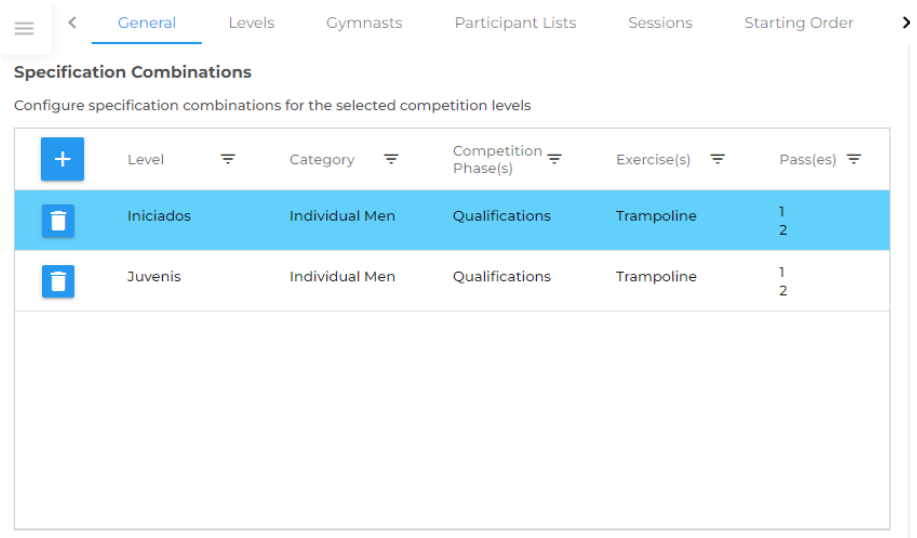

Figure 4.5: No team on menu

Similarly, within the Gymnasts tab, the visibility of certain columns such as Category and Level is directly tied to the competition's structure. If the competition settings do not include individual categories or levels, these columns are in this context superfluous, cluttering the interface with unused input fields(see Figure 4.7). Hiding these columns (see Figure 4.8) when no individual categories exist streamlines the data entry process for users, ensuring that only the most

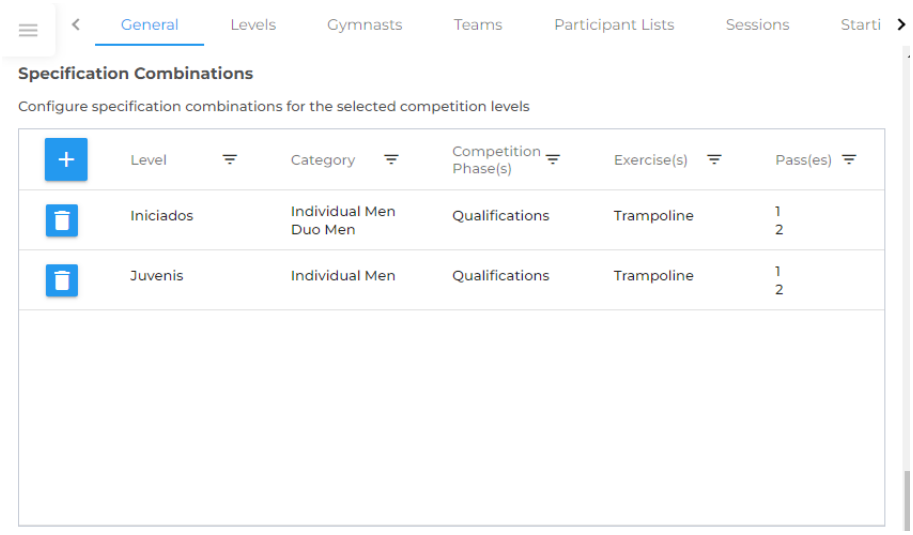

Figure 4.6: Menu with teams on menu

relevant information is presented at any given time. This also reduces the potential for user errors, as irrelevant fields that could lead to confusion or incorrect data entry are simply not shown.

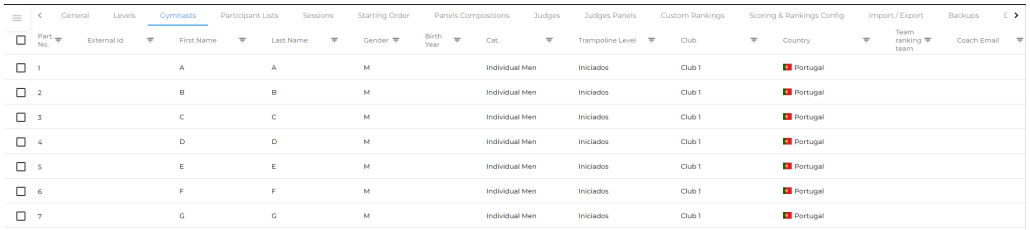

Figure 4.7: Gymnast tab with unnecessary fields not hidden

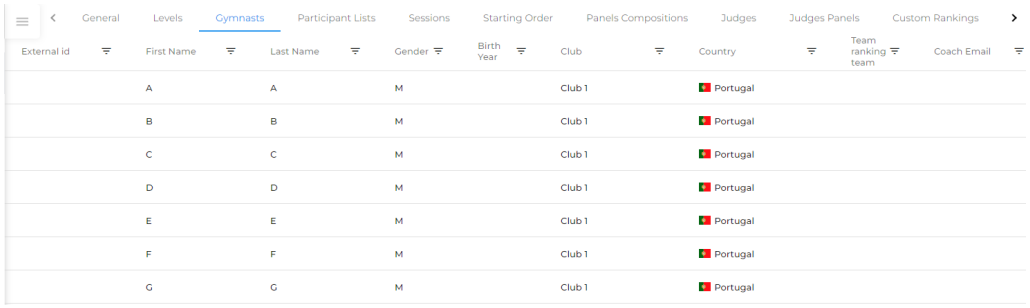

Figure 4.8: Improved gymnast tab when there are no individual categories in Specification Combinations

Another critical case relates to gender-specific disciplines in gymnastics competitions. Certain events are strictly male or female, and therefore, the inclusion of a Gender field in the Import and Gymnasts tabs for those disciplines becomes redundant. By hiding the Gender field in these instances, the system simplifies the data entry process, removing unnecessary steps and minimizing the cognitive load on users. This conditional visibility of the Gender field ensures that the system

remains intuitive and efficient, adapting to the specific needs of each discipline without cluttering the interface with irrelevant options. In Figure 4.9 we can see that the import table has the gender column because it is not gender-specific and in Figure 4.10 the same column is hidden because WAG is strictly for female athletes.

Figure 4.9: Import gymnast table for trampoline

Figure 4.10: Import gymnast table for Women's Artistic Gymnastics (WAG)

In addition to the dynamic hiding of tabs and columns, this refactor also examines the importance of improving error validation mechanisms in forms. While hiding unnecessary UI elements enhances clarity and reduces confusion, it is equally important that the system provides clear and actionable feedback when users encounter errors. Error tooltips, for example, play a vital role in guiding users toward correct data entry by offering immediate, context-specific feedback. Improving these tooltips in both the Gymnasts and Teams tabs ensures that users are not only made aware of errors but also understand how to resolve them. Enhanced error tooltips reduce frustration, shorten the learning curve for new users, and ultimately lead to higher user satisfaction.

Beyond error tooltips, this refactor also explores the need for better validation of Level and Category combinations for both gymnasts and teams. The system must rigorously check these combinations to ensure that the data entered is both accurate and consistent with the competition's specifications. For example, if a gymnast is assigned a category that is not available for their level, or if a team is configured with incompatible categories, the system should provide immediate feedback and prevent incorrect data from being saved (see Figure 4.11). By implementing more robust validation checks, the system not only reduces the potential for user errors but also ensures data integrity, improving overall reliability and trust in the system.

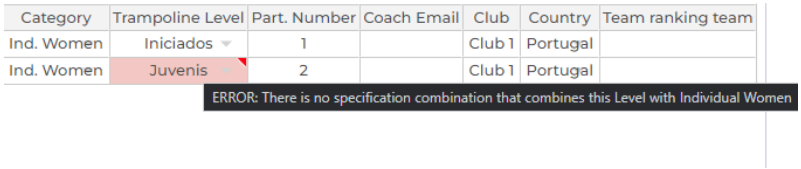

Figure 4.11: Import level and category validation error

Ultimately, this refactor argues that the selective hiding of tabs, columns, and validators in gymnastics competition management software results in a more efficient and user-friendly experience. By reducing visual clutter and preventing users from interacting with irrelevant options, the system becomes easier to navigate and less prone to error. Moreover, these benefits come with virtually no negative impact on the functionality of the system. Advanced users still retain access to all necessary configurations when applicable, and the dynamic nature of the UI ensures that the system remains flexible enough to handle a wide range of competition formats.

In conclusion, the practice of dynamically hiding non-essential configuration options and improving error validation processes represents a significant improvement in the usability of complex software systems like gymnastics competition management platforms. This approach not only streamlines the user experience by removing irrelevant options but also enhances data accuracy and reduces user frustration through better feedback and validation mechanisms. As such, this refactor demonstrates that hiding configuration and validation elements in this context leads to a more efficient, user-friendly, and error-resistant system, with minimal downside.

4.2 Implementation of new features

4.2.1 Name formatting for gymnasts and judges

In competition administration, the inconsistency in the formatting of gymnast and judge names is a recurring issue that can lead to confusion and misinterpretation of official documents. This lack of standardization in name formatting is particularly evident in the capitalization of names, which often varies widely. For example, some names may appear with no capitalization at all, while others might feature inconsistent capitalization, such as the first name in lowercase and the last name fully capitalized. These discrepancies can cause issues in record-keeping, reporting, and in the presentation of competition results.

To address this problem and ensure uniformity in the display of names, a reusable dialog was developed. This dialog (also known as popup) allows managers to apply a consistent formatting style to the names of gymnasts or judges, ensuring clarity and professionalism in competition documentation. The tool provides two distinct formatting options to accommodate various needs and preferences:

- **Normal Formatting**: This option capitalizes the first letter of each name, following a standard title case format (e.g., “Daniel Cambinda”). It is appropriate for general use in most

competition documents, including athlete profiles, score sheets, and event listings. This format enhances readability without altering the conventional presentation of names.

- **Official Formatting**: In addition to capitalizing the first letter of each name, this option capitalizes the entire last name (e.g., "Daniel CAMBINDA"). This format is typically used in official or formal documents, such as certificates, awards, and rankings, where the distinction of the last name is crucial for identification or emphasis.

The introduction of these formatting options eliminates the need for manual corrections and ensures that all names are presented consistently across different platforms. By applying one of these predefined formats, competition managers can maintain a professional appearance in all communications and documents related to the event.

Additionally, the dialog is designed to be highly user-friendly and reusable, allowing administrators to apply formatting quickly and efficiently to multiple names in a single operation. This not only saves time but also reduces the potential for human error, which is particularly important in large-scale competitions where hundreds of names may need to be formatted.

Figure 4.12 illustrates the layout and functionality of the dialog. It also highlights the ease of use, with clear options for selecting the desired formatting style. Users can switch between "Normal Formatting" and "Official Formatting" with a simple click, and the changes are immediately applied after confirming the selected option.

Additionally formatting the names benefits the application by:

- **Improving Data Integrity**: Standardized formatting ensures that names are consistently recorded across databases, reducing the likelihood of duplicate entries or mismatches caused by variations in name presentation.

- **Enhancing Searchability**: A uniform naming format makes it easier to search for and identify specific individuals within a database, particularly in systems that rely on exact matches for queries.

- **Giving a Professional Presentation**: Consistent formatting contributes to a polished and professional appearance in all printed and digital materials associated with the competition. This is particularly important in high-profile events where attention to detail reflects on the credibility of the managers.

The implementation of a reusable dialog for name formatting in competition administration represents a significant improvement in both the efficiency and professionalism of competition administration. By offering flexible, yet standardized, options for name presentation, this tool ensures that all individuals are accurately and consistently represented in competition records, enhancing both the clarity and quality of administrative documentation.

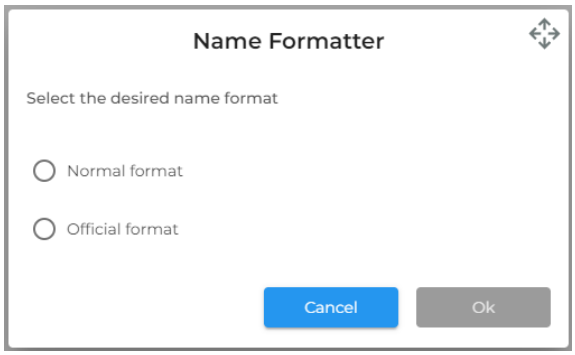

Figure 4.12: ”Name formatter” dialog

4.2.2 Copy generic competition administration

Creating a competition from scratch can be a time-consuming task, given that most competition managers handle multiple events a the same time. With the need to make this process faster and more efficient I was assigned the task of creating a feature that allows competition managers to copy data from previous competitions organized by them.

The first step to implement this new feature was creating a dialog so that managers can select the competition they want to copy from as well as the data they want to copy. To achieve this the dialog was transformed into a two step process. In the first page(as can be seen in Figure 4.13) managers can select the competition they wish to copy from. This competition list is limited to competition that are also managed by the current user and those need to have the same discipline and country, ensuring that the copied data remains relevant and compliant with local regulations.

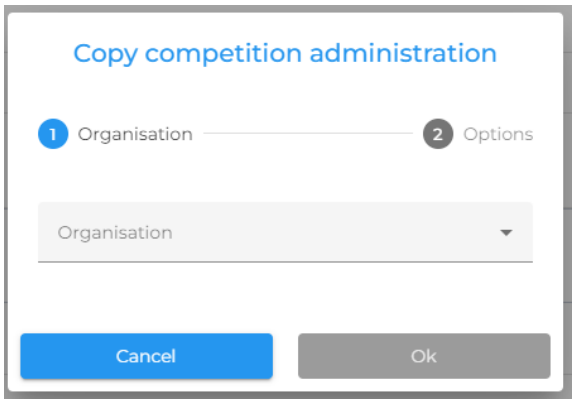

Figure 4.13: Step 1 - Select competition you want to copy from

The second page (represented in Figure 4.14) of the dialog is where the managers can selected what options that are going to be copied. In this page managers can also confirm what competition was selected in the first dialog page as well as select if he wants to remove the existing data of the current competition (in case there is any). The copy options are divided in 3 main groups:

- **Levels, Competition phases, specification combinations and sublevels** - This group allows managers to copy settings related to competition structure and age divisions. Competi-

tion phases help to divide the competition into steps, ensuring participants move through the event in a clear and systematic way. The levels correspond to the age division for the selected country. As previously explained in Section 4.1.3, specifications combinations allow managers to setup combinations between levels, categories, competition phases, and discipline specific exercises for the competition to easily create exercises for gymnasts. Sublevels is an option that allows managers to split a specification combination level and category combination into multiple independent levels (e.g Seniors A, Seniors B, etc.).

- **Competition zones, sessions and panel compositions** - In this group it is possible to select options that are related to the competition venue and scheduling. Competition zones are designated areas in the competition venue where gymnasts or teams are going to perform their exercises during a competition. Sessions refer to a period of time during a competition in which a group of athletes or teams perform their routines. Panel compositions are configurations contain the amount of judges per judge type that are going to judge for each competition zone and session combination. This last option is only available if the other two options are selected because it depends on them.

- **Other** - With this option it is possible to copy gymnasts and judges from the selected competition.

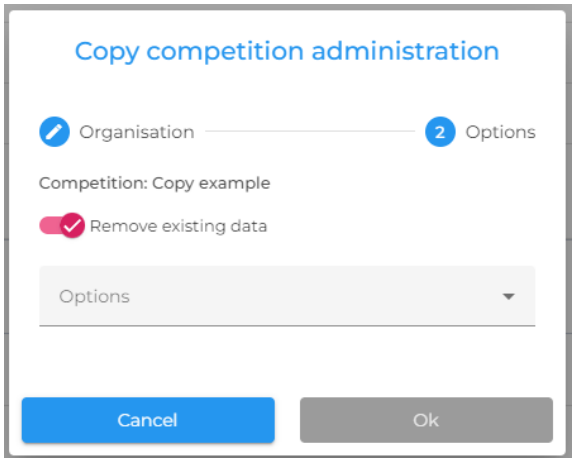

Figure 4.14: Step 2 - Select data to be copied

The second step of the implementation consists in creating a service that processes all the data and adds it in the model of the current competition. After the data is processed a new dialog is presented to users displaying the feedback confirming if the data successfully copied or if there was any error during the data process, as can be seen in Figure 4.15.

The final step of the implementation was adding a feature explanation in the manual section of the application(as can be seen in Figure 4.16).

This new feature not only saves time but also ensures uniformity across competitions, which is crucial for maintaining standards in recurring or closely related events. Furthermore, managers

Figure 4.15: Copy Feedback Dialog

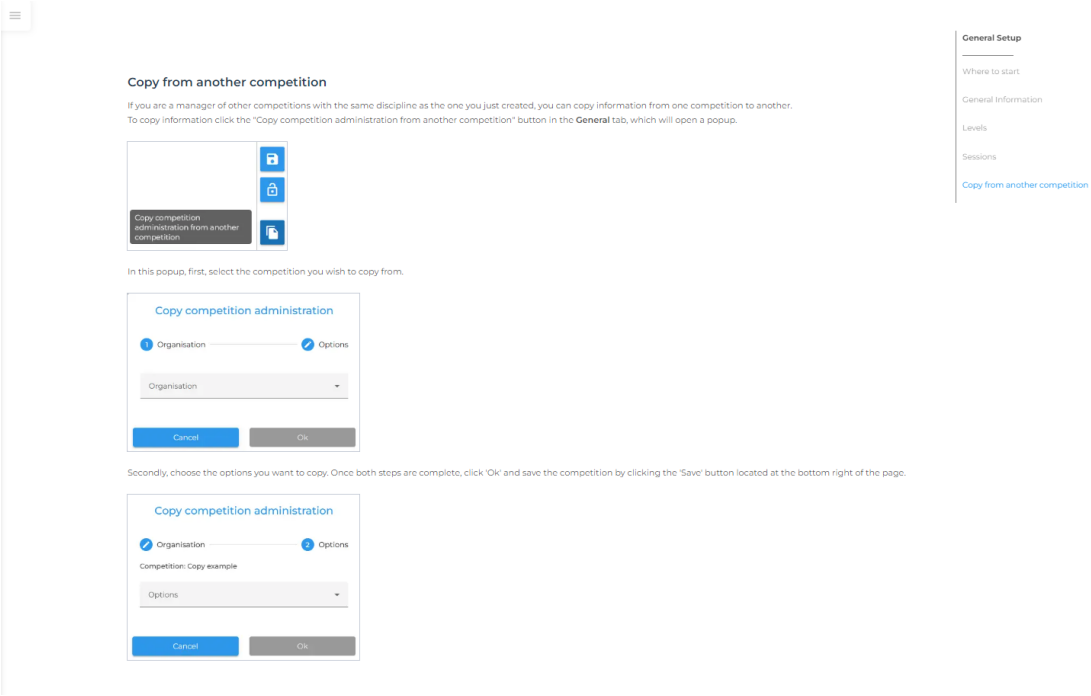

Figure 4.16: Manual

have the flexibility to modify copied data as needed, allowing for customization while retaining the foundational structure of the original competition.

4.2.3 Refactoring the generic competition administration communication

As Angular evolves, it consistently introduces innovative concepts designed to enhance developer productivity and optimize application performance. Among these advancements are RxJS and the newly introduced Angular Signals. Both of these tools play crucial roles in managing reactivity and state within Angular applications, but they approach these tasks in distinct ways.

RxJS, a library for reactive programming using Observables, has long been a cornerstone for handling asynchronous data flows and state management in Angular. Its rich set of operators allows developers to compose complex data streams and manage side effects with precision. However, RxJS's complexity can sometimes introduce significant boilerplate code and a steep learning curve.

In response to these challenges, Angular has introduced Angular Signals, a new reactive primitive designed to simplify the management of state and reactivity. Signals provide a more declarative approach to state changes, automatically updating the UI when dependencies change without the need for manual subscription management. This new paradigm aims to reduce boilerplate code, enhance performance, and offer a more intuitive development experience.

In this section, the key differences between RxJS and Angular Signals will be exposed[23], examining their respective approaches to reactivity, change detection, and performance. I will also explain the benefits of Angular Signals and the steps defined to make this transition possible.

- **Usage Pattern**: While both Angular Signals and RxJS Observables are used for reactive programming, they serve different purposes and offer distinct usability. Angular Signals are more streamlined and efficient for managing state within Angular components, providing a simpler and more direct approach compared to the more complex and flexible RxJS Observables. They provide a clean, value-based reactivity model that feels natural for developers familiar with regular JavaScript variables. With Signals, developers can directly access and manipulate the state, and the UI automatically responds to changes, offering a more straightforward approach to handling local component state. This pull-based reactivity model is simpler and more efficient for most cases of UI updates.

  In contrast, RxJS Observables follow a more complex, push-based event stream model, suited for asynchronous data streams like HTTP requests, user interactions, or event handling. While Observables are powerful in handling continuous streams of events over time, they add unnecessary complexity when used for simpler, localized state management tasks. For most component-based reactivity in Angular, Signals provide a better, more developer-friendly pattern with fewer moving parts compared to the often verbose RxJS Observables.

- **Performance Implications**: Angular Signals are notable for their performance, especially when dealing with local state changes inside Angular components. Signals are optimized for fine-grained reactivity, meaning they only trigger updates in the parts of the UI that depend on the state that has changed. This eliminates unnecessary re-renders and ensures that the application remains highly performant, particularly as the size of the application grows. Signals are natively integrated into Angular's reactivity model, which makes them incredibly efficient for managing UI state.

  On the other hand, while RxJS Observables are reliable tool for handling complex asynchronous workflows, they can impose significant performance overhead when used for state management. Observables emit data over time, and each emission can trigger multiple

change detections across the component tree if not carefully optimized. This can lead to performance degradation, particularly in large applications with frequent state updates. By contrast, Signals provide a more lightweight, focused solution for managing state changes, offering a better out-of-the-box performance experience without requiring developers to manually optimize reactivity.

- **Unsubscription Handling**: One of the major benefits of Angular Signals is that they do not require manual unsubscription, making them inherently safer and easier to manage than RxJS Observables. Signals are automatically tracked and cleaned up by Angular, so developers don't need to worry about memory leaks or complex unsubscription logic. This feature makes Signals an obvious choice for reactive state management, especially in long-lived components, where unsubscription management can become cumbersome.

  In contrast, RxJS Observables require manual subscription and unsubscription. Developers must carefully manage subscriptions to ensure that they are properly cleaned up, especially for continuous or long-running streams. Failure to do so can result in memory leaks, leading to degraded performance over time. Developers need to utilize patterns such as the `takeUntil()` operator or manually call `unsubscribe()` to avoid these issues. Signals, by automatically handling this lifecycle aspect, clearly have an advantage here, simplifying memory management and reducing the risk of errors.

- **Learning Curve**: The learning curve for Angular Signals is significantly lower than that of RxJS Observables, making them a more accessible and beginner-friendly choice for developers working within the Angular ecosystem. Signals operate with a straightforward, value-based API that resembles standard JavaScript state management patterns, meaning developers can quickly understand and implement them without needing to learn new programming paradigms. This ease of use is further amplified by the minimal boilerplate required to integrate Signals into Angular components, allowing developers to focus more on business logic rather than dealing with complex reactivity patterns.

  In comparison, RxJS Observables present a much steeper learning curve due to their reliance on functional programming concepts, streams, and operators. Understanding how to work with observables requires familiarity with a wide range of RxJS operators, subscription patterns, and techniques for handling async events over time. While Observables are powerful for certain use cases, they often introduce excessive complexity, especially for developers unfamiliar with functional programming or reactive streams. Signals, by comparison, provide a much smoother learning experience, making them the better choice for most Angular projects where simplicity and ease of understanding are key.

- **Change Detection Behavior**: Angular Signals are deeply integrated with Angular's change detection system, making them better suited for reactive state management in components. When the value of a Signal changes, Angular automatically triggers change detection for the affected parts of the component, ensuring that only the necessary parts of the DOM

are updated. This fine-grained control reduces the amount of manual optimization required from developers, leading to better overall performance and a more responsive UI. Signals make Angular's change detection process more efficient by minimizing the scope of updates, which is crucial in maintaining fast, scalable applications.[24][22]

In contrast, RxJS Observables do not automatically trigger Angular's change detection. Developers must manage this process manually, often by using the `async` pipe in templates or calling `ChangeDetectorRef.markForCheck()` to trigger reactivity. This introduces additional complexity and makes it easier to inadvertently cause inefficient change detection, as entire components may be unnecessarily rechecked upon every observable emission. While RxJS Observables can be made to work with Angular's change detection, the process is more cumbersome and less intuitive than the built-in reactivity offered by Signals. For state-driven UI updates, Signals are clearly the better choice, as they provide seamless, automatic integration with Angular's reactivity model without requiring extra manual intervention.

- **Conclusion**: In most typical Angular use cases, Angular Signals offer a better, more efficient, and easier-to-use solution for reactive state management compared to RxJS Observables. Signals provide optimized performance for fine-grained state changes, a simpler learning curve, and eliminate the need for manual subscription management. They fit naturally into Angular's reactivity model, providing a streamlined experience for developers. While RxJS Observables remain valuable for handling complex asynchronous data streams, they are overkill for most component-level reactivity tasks. Angular Signals, by offering a more direct, intuitive, and high-performance approach, are the preferred tool for most scenarios, making them a superior choice for reactive programming in Angular.

To transform this system into using Angular Signals the workload was divided into five main steps:

- **Step 1** - Create a signal version of the generic competition administration and transform all the model modification to update that signal instead of the observable - Data model observable is still updated when the signal is updated as a security measure for database saves
- **Step 2** - Total remove of the Rxjs Observables in the generic competition administration model service
- **Step 3** - Go over every generic competition administration related component and transform from Rxjs to Angular Signals
- **Step 4** - Transform inputs and outputs from decorators to signal input and output function
- **Step 5** - Change local state of generic competition administration components to signals

This transition into using Angular Signals is an essential first step in addressing a larger issue within the application: the desynchronization between the competition administration model and the organization/competition model. This problem arises during competition switches, where the organization model tends to load faster than the competition administration model.

As a result, if data is saved while these models are out of sync, it can lead to incorrect data being recorded under the previous competition. By implementing Signals, we aim to enhance the reactivity and synchronization between these models, ensuring that data operations are performed only when both models are in alignment. This improvement will help prevent potential data integrity issues and streamline the competition management process.

4.2.4 Migrate competition administration from modules to standalone components

One of the most significant new Angular features is the introduction of Angular Standalone Components [3]. Standalone components allow developers to create self-contained components that manage their own dependencies, making them more reusable and easier to integrate across different applications. This approach significantly reduces the coupling between components and modules, leading to a more streamlined and modular development process. The evolution toward standalone components reflects broader industry trends, including modular design, lightweight architecture, and micro frontend adoption.

Historically, Angular applications have relied heavily on the NgModule system — a foundational concept that organizes and encapsulates related components, directives, pipes, and services into cohesive modules. This module-centric architecture has long been instrumental in structuring Angular applications, enabling developers to group related functionality and manage dependencies effectively. However, it has also introduced additional layers of abstraction, which can lead to increased complexity and boilerplate code, particularly in large-scale applications.

This new Angular feature also addresses several pain points that developers have encountered with the traditional module system, such as the overhead of managing large module hierarchies, the difficulty of sharing components across different modules, and the complexities of dependency management.

When comparing Standalone Components to the traditional Angular module system, several key benefits emerge, each contributing to a more efficient and maintainable development process. The benefits of using standalone components in Angular are:

- **Simplified Component Development**: Standalone components significantly reduce the complexity associated with Angular component development by eliminating the need for ‘NgModule‘ declarations. In the traditional module-based architecture, every component, directive, or pipe had to be declared inside a module, even for small or isolated components. This added unnecessary boilerplate code and increased the learning curve for developers, particularly those new to Angular. Standalone components simplify this by allowing developers to create components without the overhead of managing modules.

No Need for NgModules: Standalone components remove the necessity of defining an ‘NgModule‘ for every component. This makes the development process more straightforward, particularly for smaller projects or isolated components where defining a module feels like overkill. Developers can now focus on building individual, self-contained components without having to think about managing modules. The lack of dependency on ‘NgModule‘ simplifies the entire process.

For smaller projects or prototyping scenarios, where speed of development is essential, standalone components drastically reduce the initial setup. Developers no longer need to generate or manage unnecessary ‘NgModules‘, making the project structure leaner and more manageable.

- **Enhanced Modularity and Flexibility**: Standalone Components encourage a more modular approach to application architecture. Each component, directive, or pipe can operate as a self-contained unit, with its own dependencies explicitly defined. This contrasts with the traditional ‘NgModule‘ approach, where components and their dependencies are declared in a centralized module, often leading to tightly coupled code.

  The standalone model allows developers to build and import components in a more granular and flexible manner. For instance, if a component depends on certain Angular features, such as ‘CommonModule‘ for structural directives like ‘ngIf‘ and ‘ngFor‘, these dependencies can be imported directly within the component itself. This enables true modularity, where components are less dependent on the broader application’s module structure and are easier to reuse across different parts of an application or even in entirely separate applications. This decoupling also makes it easier to maintain and refactor components without worrying about breaking other parts of the application.

- **Improved Tree Shaking and Bundle Size Reduction**: Tree shaking refers to the process of eliminating unused code from the final JavaScript bundle, which directly impacts application performance by reducing the size of the bundle that is delivered to the client. Standalone components have a positive impact on tree shaking and overall bundle size optimization. With standalone components, tree shaking is more granular because components and their dependencies are clearly defined at the component level, making it easier for the Angular build process to shake off any unused code. This results in smaller bundle sizes and improved performance, especially in applications that have a large number of rarely-used components or libraries.

- **Improved Performance and Bootstrapping Time**: Standalone components can lead to faster bootstrapping and improved performance, particularly in applications with a large component hierarchy. In traditional Angular applications, the bootstrapping process involves compiling and initializing the root module (‘AppModule‘) and all of its dependencies. With standalone components, the bootstrapping process is more direct because the component itself can serve as the entry point, bypassing the need for a root module.

- **Easier Testing and Maintainability**: Standalone components simplify unit testing by removing the need to mock or configure entire modules. Each standalone component can be tested in isolation, along with its declared dependencies. This leads to faster, more focused tests that are easier to write and maintain. Furthermore, the decoupling of components from modules enhances maintainability, as it becomes easier to refactor or update components without needing to worry about the module structure.

- **Compatibility with Micro Frontends and Modern Architectures**: Standalone components are ideal for micro frontend architectures, where different parts of a larger application can be developed and deployed independently. Each micro frontend can be composed of standalone components that are self-contained and manage their own dependencies, making it easier to integrate different parts of the application. This modular approach also facilitates better scaling of development teams, as each team can focus on a specific part of the application without needing to worry about the broader module structure.

- **Backward Compatibility and Incremental Adoption**: One of the best aspects of standalone components is their backward compatibility with existing Angular applications and the ability to adopt them incrementally. This allows developers to take advantage of standalone components' benefits without needing to refactor their entire application all at once. Since standalone components are fully compatible with the existing 'NgModule'-based architecture, developers can mix and match the two approaches within the same application. This ensures a smooth transition path and reduces the risks involved with migrating to a new architecture. Developers can begin by gradually converting specific parts of the application to standalone components while leaving the rest of the application unchanged. This makes it possible to adopt the new architecture in an incremental and flexible manner, ensuring that teams can fully explore the benefits of standalone components without interrupting the ongoing development processes.

The steps I followed to transform the competition administration system from module-based into standalone components were:

- **Identify the existing components and their dependencies** - In this step I started by reviewing the existing modules to identify all components, directives, and pipes currently grouped within each module in the system. Angular modules are composed of a decorator provides crucial details about the module's configuration(can be seen in Figure 4.17), as well as a class definition for the module. The "declarations" options contains a list of the components, directives and pipes that were created for that modules, "imports" specifies other modules needed by the module and "bootstrap" defines the component which component is used as as entry point for that module. Other option that are worth mentioning are the "exports" which allows to define what components can be used when the module is imported and "Providers" that defines the services that are available for dependency injection in the model.

```
@NgModule({
  imports: [BrowserModule,RouterModule],
  declarations: [AppComponent],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Figure 4.17: Angular module file structure

- **Isolate Components** - The second step consisted into converting each component to a standalone component by setting standalone property value to true in the component decorator and add the component dependencies to the imports property(see Figure 4.18).

```
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss'
})
export class AppComponent {}
```

Figure 4.18: Angular component file structure

- **Update Routing** - The last step of this process was to replace the deleted module with a component setting it new route entry-point.

Angular Standalone Components represent a significant improvement in the framework's architecture, offering a simpler, more modular, and more performant way to build Angular applications. By eliminating the need for 'NgModule' declarations, standalone components reduce complexity, while also improving the performance and maintainability of Angular applications. With their backward compatibility and ability to be adopted incrementally, standalone components also provide a flexible and scalable solution for modern web development.

Chapter 5

Conclusion

In collaboration with Acro Companion, this project aimed to elevate the competition management solution, ensuring it not only became more stable and fair but also integrated features that provide a comprehensive management experience, effectively distinguishing it from competing applications. All the implementations completed during this project are now fully integrated into the production version of the application, reflecting our commitment to delivering a high-quality solution tailored to the needs of gymnastics competitions.

Through the achievement of our defined objectives, it is evident that the solution now incorporates more rigorous data handling practices, which significantly enhance both system stability and credibility. This not only ensures that users receive valuable insights throughout the management process but also instills a greater sense of trust in the application.

The project also involved a thorough identification and resolution of various issues within existing system functionalities, with some challenges uncovered through extensive research while others emerged from direct user feedback. This comprehensive approach resulted in a more accurate and complete application compared to previous iterations. Continuous testing of both existing and new functionalities has been pivotal in ensuring that future enhancements do not compromise the integrity of the solution.

Although direct feedback from end-users was not provided, it is significant to note that all changes were rigorously reviewed and accepted by the superiors at Acro Companion, demonstrating their confidence in the improvements made. During the configuration of competitions, they reported a notable improvement in user experience, highlighting that the overall process was faster and more intuitive, which reflects the effectiveness of the enhancements implemented.

While the development process was largely successful, it was not without its challenges; navigating a new project alongside unfamiliar technologies presented obstacles that required resilience, adaptability, and innovative problem-solving. Ultimately, this project has substantially improved the competition management solution, positioning it as a robust tool that effectively addresses the evolving needs of its users and sets a new standard in the field of gymnastics competition management. The advancements made not only enhance the operational aspects of organizing competitions but also contribute to a more positive experience for athletes, coaches, and event organizers alike, allowing them to focus on the sport rather than the complexities of administrative

tasks.

Looking forward, future work could focus on enhancing system automation, particularly in the creation of exercises for subsequent competition phases based on real-time scoring. Additionally, refining the user interface to better accommodate different competition formats and expanding test coverage would help ensure reliability in more complex scenarios. This would allow the platform to continue evolving and addressing the growing demands of competition management.

Bibliography

[1] Angular. Angular website. https://angular.io/guide/what-is-angular.

[2] Angular. Signals guide. https://angular.dev/guide/signals.

[3] Angular. Standalone components guide. https://v17.angular.io/guide/standalone-components.

[4] Angular and Firebase Community. Integrating angular with firebase: Best practices and considerations. https://angular.io/guide/firebase.

[5] Codecademy. What is an ide? https://www.codecademy.com/article/what-is-an-ide.

[6] Software Engineering Community. Understanding software patching. https://www.techtarget.com/searchenterprisedesktop/definition/patch.

[7] Acro Companion. Acro companion wbbsite. https://www.acro-companion.com.

[8] Firebase. Firebase website. https://firebase.google.com/products-build.

[9] Git. Git website. https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.

[10] Gymdata. Gymdata official website. https://www.gymdata.co.uk/.

[11] Uplifter Inc. Uplifter official website. https://www.uplifterinc.com/.

[12] Ksis. https://rgform.eu/menu.php?akcia=KS.

[13] Maxmilián Otta. What is reactive programming? https://www.baeldung.com/cs/reactive-programming.

[14] Playwright. Playwright website. https://playwright.dev.

[15] RxJS. Rxjs website. https://rxjs.dev/guide/overview.

[16] Elevien Team. Elevien. https://elevien.com.

[17] Google Cloud Team. Google cloud documentation. https://cloud.google.com/docs.

[18] Visual Studio Code Team. Visual studio code documentation. https://code.visualstudio.com/docs.

[19] Swiss Timing. Gymnastics timing solutions. https://www.swisstiming.com/sports/gymnastics/.

[20] Typescript. Typescript website. https://www.typescriptlang.org.

[21] Azure DevOps Website. What is azure devops? https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops.

[22] Eugeni Yoz. Angular signals, reactive context, and dynamic dependency tracking. https://medium.com/@eugeniyoz/angular-signals-reactive-context-and-dynamic-dependency-tracking-d2d610056

[23] Eugeni Yoz. Angular signals vs observables: Differences. https://medium.com/@eugeniyoz/angular-signals-observables-differences-4a0aa7a13bc.

[24] Eugeni Yoz. Dependency graph in angular signals. https://medium.com/@eugeniyoz/dependency-graph-in-angular-signals-53ee47f75e21.