UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Improving Web Application User Experience with Dynamic Sidebars

Guilherme Emanuel Junqueira Borges

Mestrado em Engenharia Informática

Trabalho de Projeto orientado por:
Prof. Doutor Pedro Alexandre de Mourão Antunes

2023

Acknowledgments

I would like to thank my esteemed supervisor Dr. Pedro Antunes for his invaluable supervision, support and tutelage during the course of my Master's degree. My gratitude extends to the University of Lisbon for the create opportunity to work together with Acro Companion company. Additionally, I would like to express gratitude to Lander, Ruben and Samuel for their mentorship as well as all my colleagues that started the internship with me. My appreciation also goes out to my family and friends for their encouragement and support all through my studies.

Resumo

Acro Companion tem vindo a evoluir gradualmente e a desenvolver a sua gama de produtos nos últimos anos para a sua **aplicação web**. Devido a esta grande pressão no desenvolvimento a **experiência de utilização** e a **interface do utilizador** foram-se degradando. O objetivo deste projeto é **através da implementação de uma solução dinâmica melhorar a experiência de utilização e a interface**.

Para ajudar na organização do projeto usei a metodologia **Action design research** Sein et al. [2011] uma vez que esta é baseada em artefactos, neste caso, uma sidebar dinâmica e permite um design iterativo dentro da Acro Companion. Além disso, ainda ajuda no processo de resolver problemas com o objetivo de organizar o processo de aprendizagem.

Todo o desenvolvimento feito durante os nove meses em que estive no estágio com a Acro Companion está dividido em três grandes iterações:

- Prova de conceito para a sidebar dinâmica (Primeira iteração);
- Sidebar dinâmica de conteúdo de navegação (Segunda iteração);
- Sidebares dinâmica de conteúdo (Terceira iteração).

Cada iteração teve diferentes razões para ter sido criada e com o objetivo de refinar a solução **sidebars dinâmicas**. Com o crescimento da Acro Companion era necessário desenvolver uma solução que primeiro fosse muito escalável e dinâmica e após isso resolvesse todos os problemas de experiência de utilização e da interface do utilizador. Com isto o conceito de sidebars dinâmicas surgiu como uma solução que se adaptava à Acro e não o contrário.

Em cada iteração foram discutidos seis tópicos:

- Problema;
- Design/Construção;
- Intervenção;
- Implementação Prática;
- Avaliação;
- Reflexão.

Primeiramente no capítulo ”Problema” é feita uma análise do problema que leva à criação de uma iteração. Seguida essa análise no capítulo ”Design/Construção” são apresentados conceitos e mockups que tem como finalidade propor uma solução para o problema descrito no capítulo “Problema”. No capítulo da intervenção e onde são apresentados problemas no desenvolvimento da solução, restrições impostas pela Acro Companion, estado da solução e o que falta fazer para completar a solução. No capítulo da ”Implementação Prática” são descritas as decisões tomadas a nível da arquitetura e da estrutura do código. Na ”Avaliação” é feita uma avaliação da solução desenvolvida seguida por uma reflexão segundo todos os pontos descritos anteriormente.

Durante o desenvolvimento da solução ideal para a Acro na Prova de conceito para a sidebar dinâmica existiam duas possíveis soluções principais:

- Titlebar dinâmica;
- Sidebar dinâmica;

Depois de analisar os benefícios e problemas de cada solução, tornou-se claro que a sidebar dinâmica seria a melhor opção para resolver os problemas da Acro Companion, pois a title bar dinâmica não resolveria todos os problemas existentes e em termos de implementação iria aumentar a dificuldade. Alguns inconvenientes da titlebar dinâmica seriam o facto de ter uma dimensão pequena para exibir o conteúdo, o que poderia levar a problemas de UX/UI, como por exemplo a dificuldade em exibir a progressão e hierarquia da navegação ou a impossibilidade de voltar a uma página pai após entrar em uma página filha sem implementar um método para retornar à página anterior.

A sidebar dinâmica, por outro lado, não tinha nenhum dos problemas mencionados anteriormente e tinha outras vantagens. Como a sidebar tem um formato grande, seria mais fácil ajustar e personalizar o conteúdo com ícones e animações que são importantes para a experiência de utilização. Outra consideração é que os monitores são mais largos do que altos, então a barra de título ocuparia muito espaço que já é limitado. Além disso, enquanto a barra de título precisaria estar sempre visível em smartphones, podemos ocultá-la com a barra lateral.

No final, o principal elemento de UI (a sidebar) foi projetada com base nos princípios da teoria de affordances. De acordo com Stendal et al. [2016], ”affordance”refere-se às ”possibilidades de ação”ou capacidades latentes do ambiente, independentemente da capacidade do indivíduo de reconhecê-las, mas sempre em relação aos atores e, portanto, dependentes das suas capacidades.

Ao conceptualizar a sidebar, temos possibilidades de ação e intenções do utilizador. Quando aplicada à solução personalizada da sidebar, a sidebar é dinâmica e exibe apenas as opções necessárias, fornecendo ao utilizador opções relevantes. A sidebar dinâmica pode ser representada por duas principais affordances: a primeira é que a sidebar dinâmica é uma affordance como um todo, aparece apenas para utilizadores específicos e adapta a sua estrutura de acordo com o contexto. A outra affordance é que o conteúdo exibido dentro da sidebar dinâmica tem affordances que adaptam o conteúdo em relação ao tipo de utilizador. Com isto, a Prova de conceito para a sidebar dinâmica do projeto começou com o objetivo de desenvolver uma prova de conceito para testar a solução escolhida.

Durante a Prova de conceito para a sidebar dinâmica, continuei a desenvolver e expandir a solução da sidebar introduzida na iteração inicial. Este capítulo apresenta os ajustes feitos para lidar com os desafios relacionados à flexibilidade e natureza dinâmica da experiência de utilização e interface da aplicação web. O objetivo é fornecer uma solução que atenda à necessidade de um aplicação web mais flexível e ágil, capaz de facilitar o desenvolvimento rápido de serviços e atender às necessidades dos utilizadores, o que não foi alcançado na Prova de conceito para a sidebar dinâmica. Foi necessário mover componentes para a sidebar e introduzir a possibilidade de navegar por páginas através da sidebar. Após esta implementação e através de uma avaliação por parte da equipa da Acro Companion foi descoberto que a experiência de utilização tinha sido prejudicada por apresentar conteúdo de páginas e botões de navegação na mesma sidebar.

O objetivo da Sidebares dinâmica de conteúdo foi abordar os problemas de UX/UI identificados na Prova de conceito para a sidebar dinâmica, refinando o design e a funcionalidade da sidebar dinâmica, com o foco em melhorar a experiência de utilização e garantir que a solução esteja pronta para produção. O objetivo era identificar uma solução estável e adaptável que abordasse os problemas identificados e cumprisse os requisitos do aplicativo da web Acro Companion.

Foi então reestruturada a arquitetura da sidebar,para o novo conceito, em vez de uma sidebar passar a ter duas, tendo cada sidebar uma função diferente:

- Uma das sidebares exclusivamente usada para a navegação entre páginas;
- A outra exclusivamente para a gestão do conteúdo do componente principal.

Além disso, a sidebar de navegação está dividida em três componentes principais para melhorar a estrutura.

- Menu principal;
- Menu de gestão da organização;
- Menu de desenvolvedores.

No menu principal, estão os botões para navegação para as funcionalidades básicas da aplicação. Esta implementação foi mais direta de implementar pois todos os componentes eram individuais e a sua estrutura de navegação era bem estruturada.

No menu de gestão da organização encontra-se a navegação do sistema de gestão de competições que apenas um determinado tipo de utilizador tem acesso. Esta implementação foi a que apresentou um maior grau de dificuldade e levou mais tempo porque a arquitetura desta parte da aplicação era muito antiga e crítica para o funcionamento da Acro Companion. Para mitigar os problemas da alteração da arquitetura do componente foram desenvolvidos testes end to end e testes unitários.

Por fim o menu de desenvolvedores que apenas os desenvolvedores têm acesso que foi a menos cuidada uma vez que o cliente final não tem acesso.

Concluindo, a sidebar dinâmica é um elemento dinâmico que pode ser modificado usando affordances, permitindo maior flexibilidade em termos de conteúdo que pode ser exibido e proporcionando uma interface de utilizador mais uniforme. A implementação das sidebares dinâmicas foi uma implementação com base em várias soluções foram testadas e avaliadas, levando à solução final que cumpriu todos os requisitos pedidos pela Acro Companion.

Palavras-chave: Experiência de Utilização, Interface do Utilizador, Sidebars Dinâmicas

Abstract

This paper demonstrates the role of a **dynamic sidebar** solution as a model to enhance the user experience of a **web application**. Acro Companion has gradually evolved and expanded their product line over the last few years. Because of the emphasis on **product development**, the web application's **user experience** and **user interface** design have become inconsistent and disjointed, threatening the company's growth.

This study, which incorporates evidence from user feedback, demonstrates that implementing a dynamic sidebar solution will result in an improved user experience, a reduction in the amount of time required for implementation, and a project structure that will make future development easier.

By working in an **agile setting**, the development of this study was divided into **iterations**. Each iteration identified and addressed any concerns that arose, allowing the design of the dynamic sidebar implementation and structure to be continuously improved. These concerns were highly evidenced and exaggerated due to the scope and complexity of the project, which required me to adapt and refine the solution in order to overcome **implementation barriers**. As a result, I was able to develop a more **robust** and **scalable** final solution with fewer potential implementation barriers.

Keywords: Agile; Dynamic Sidebars; User Experience; User Interface; Web Application

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Is it possible to enhance the user experience of a web application by using dynamic sidebars? Small companies with a high level of **customer intimacy** may find it challenging to **scale** the project while maintaining a quick development pace and a pleasant user experience and user interface.

For instance, Acro Companion has a complicated application that is difficult to maintain due to the developing pace and complexity of the products, having different ways to navigate and interact with the components throughout the web application can increase the difficulty for user to locate the products or understand how to utilize the web application, causing them to become frustrated with the platform as a whole.

The development of dynamic sidebars will allow for faster integration while maintaining the same user experience and also increasing the **flexibility** of the web application framework.

Managing the integration of a central piece of software in an agile environment is difficult to achieve without disrupting existing products.

So, while the initial implementation may take some time, the balance between a dynamic solution that allows the developer to add any component and structure would solve the main problems that Acro's web application currently have with flexibility.

When compared to the previous version of the web application, the development of the dynamic sidebars provided Acro Companion with a stronger framework for future development as well as a progression in user experience and interface design. This study's solution has the potential to help Acro's products scale.

1.1 Study context: Acro Companion

Acro Companion is a Belgian company that specializes in the development of gymnastic web applications. They have created several of solutions to assist coaches, organizations, and federations in managing all administrative tasks.

The web application has become a growing, complex piece of software that is a game changer in the gymnastics world once it provides products with rates suitable for various sizes of organizations and federations.

Because of the company's diverse customer base, Acro Companion must be agile and flexible in order to meet the varying needs of its clients.

This project addresses the company's commitment to providing the best possible experience and software to the gymnastics community, as well as to address the flexibility challenges that Acro faces as it grows and expands its web application. By addressing these issues, the company will be able to continue to improve and evolve, further solidifying its position as a leader in the industry.

1.2 Organizational setting

During the Acro Companion development process, two distinct teams collaborated: one focused on the development aspects, consisting of five team members, including myself, and another responsible for design, consisting of one team member. My role was on the development team. I consistently provided valuable feedback and insights about the design of the sidebars, which contributed to the design team's continuous improvement.

Acro Companion relies on Azure DevOps to aid with team structure and management. Azure DevOps is a suite of services that cover the entire development life cycle. The use of this platform helps to keep the team organized and the objectives in sync.

The Azure Boards aid in the creation of agile planning and the tracking of work items. We use the reporting tool to help and learn from one another by evaluating each other's code. The Azure Pipelines platform assists us in executing all the tests on the submitted code (DevOps,).

From a high-level perspective, there are numerous types of product development. A federation may request a special product for its case in some circumstances; if this occurs, the federation will pay for the product's development. If more than one Federation requests a product that Acro Companions believes cannot be monetized for the market, funds will be divided equally among those Federations. Furthermore, if a product is desired by one or more customers and an Acro Companion analyst considers it a wise investment, Acro will cover the entire development cost, or the price will be split between the Federations and Acro.

Acro Companion operates on the basis of customer intimacy, and the majority of Acro Companion clients continuously request functionality. The team will then internally debate the best strategy for implementing the feature. Depending on what it is, Acro Companion seeks input internally or from certain excellent clients or test groups once it has been built and deployed as described in the company structure figure 4.1.

Lastly, Acro Companion awaits client feedback on contests. This allows the company to expand by learning as much as possible about their customers and market demands by offering the desired product.

1.3 Document structure

This report's remaining sections will be organized as follows: In the **Methodology** chapter (2), I explain the significance of Design Research to the development of my thesis. A **problem framing** chapter (**??**) in which I describe the issue that Acro Companion's web application is experiencing. In the chapter on **dynamic sidebars** (3), I provide a comprehensive explanation of all theoretical sidebar concepts as well as the current state of the art. To describe the evolution of the dynamic sidebar, I will use three iterations. In the **dinamic sidebar proof of concept iteration** chapter (4), I describe the first design cycle, including all decisions made to enhance the user experience and interface. In the **navigational sidebar iteration** chapter (5), I detail the modifications made to address the challenges associated with the flexibility and dynamic nature of the Acro web application. In the final section of the **content sidebar iteration** (6), I finalize the dynamic sidebar solution and make it production-ready. Each Iteration contains six subchapters:

1. Problem: describes the problem of the iteration;
2. Design/building: describes the design solution;
3. Intervention: describes all the decisions.
4. Practical implementation: context for the practical implementation;
5. Evaluation: where an evaluation is performed to provide additional information about the developed solution;
6. Reflection: where a reflection on the solution implemented in the iteration is provided.

In the **formalization of learning** chapter (7), I explain how the dynamic sidebar evolved from a single design to a dual architecture. I demonstrated the effect of the newly reorganized structure and the benefits it provided to the end user.

1.4 Flexibility difficulties: Acro Companion

As the web application grew, Acro Companion encountered an issue with the title bar's lack of flexibility. To address this issue, a promising solution was sought out in an effort to increase flexibility. During the development of the solution, the team placed an emphasis on flexibility, which can have negative consequences. Due to the need to anticipate and represent narratives with variations an multiple configurations, developing and debugging a flexible solution can be more challenging (Antunes and Tate, 2022).

In the (Thuan et al., 2022a) is referred that the difficulties associated with implementing process flexibility, with a focus on the push and pull factors. Additionally, the concepts of comprehensibility and representationality are introduced as essential elements for achieving process flexibility. Flexibility has the greatest impact during the analysis and design/modeling phases for organizations that require it because of uncertainty, change, and variation as described in Figure 1.1.

Acro Companion aimed for a solution that offered a high degree of flexibility. However, this level of flexibility posed challenges during the development and debugging phases. As a result, I had to advocate for a balanced approach, addressing the trade-off between flexibility and the potential issues it could introduce.

Flexibility and structure must coexist for the optimal solution to be achieved. The solution should be adaptable enough to allow for rapid development while still offering structure.

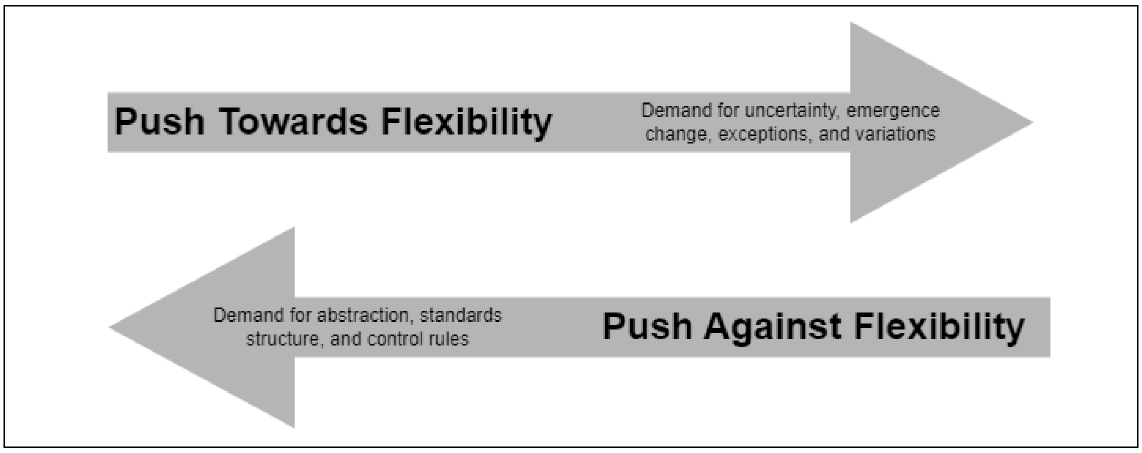

Push Towards Flexibility
Demand for uncertainty, emergence change, exceptions, and variations
Demand for abstraction, standards structure, and control rules
Push Against Flexibility

Figure 1.1: Friction logic

Chapter 2

Methodology

Methodology is an essential part of the research process because it provides a framework for addressing the research problem.

To guide the structure of the research and development for this thesis, I used the **Action Design Research** (ADR) (Sein et al., 2011) as guidance to achieve a solution for the main issue.

My thesis will be based on the ADR that were presented in (Sein et al., 2011), because the ADR is artifact-centered (in this case, the dynamic sidebars), allows for iterative design within Acro Companion, and helps structure the problem-solving process in order to organize the development and learning process. In the following sections, I will provide a more detailed explanation of ADR.

2.1 Action design research

The ADR methodology is part of the **Design Science Research** methodology (Sein et al., 2011) and it's a way to make innovative artifact design inside a company. ADR is a way to do research that leads to **prescriptive** (describe a how to) **design knowledge**. It is a way to share design knowledge that goes beyond instantiations that can be used in a certain context by building and evaluating **artifacts** in an organizational setting. It addresses two issues:

1. Intervening and assessing a **problem scenario** discovered in a given organizational environment;
2. Creating and assessing an innovative artifact that addresses the issue class characterized by the problem scenario.

These two difficulties necessitate a method centered on the **building**, **intervention**, and **evaluation** of an artifact that symbolizes not only theoretical predecessors and researchers' goals but also the impact of users and continuing usage in context.

The stages of the ADR process are shown in Figure 2.1.

The action design research method was chosen because the concept of intervention can explain the relationship with Acro Companion. Intervention research is concerned with discovering which treatments or method are most effective in improving results.

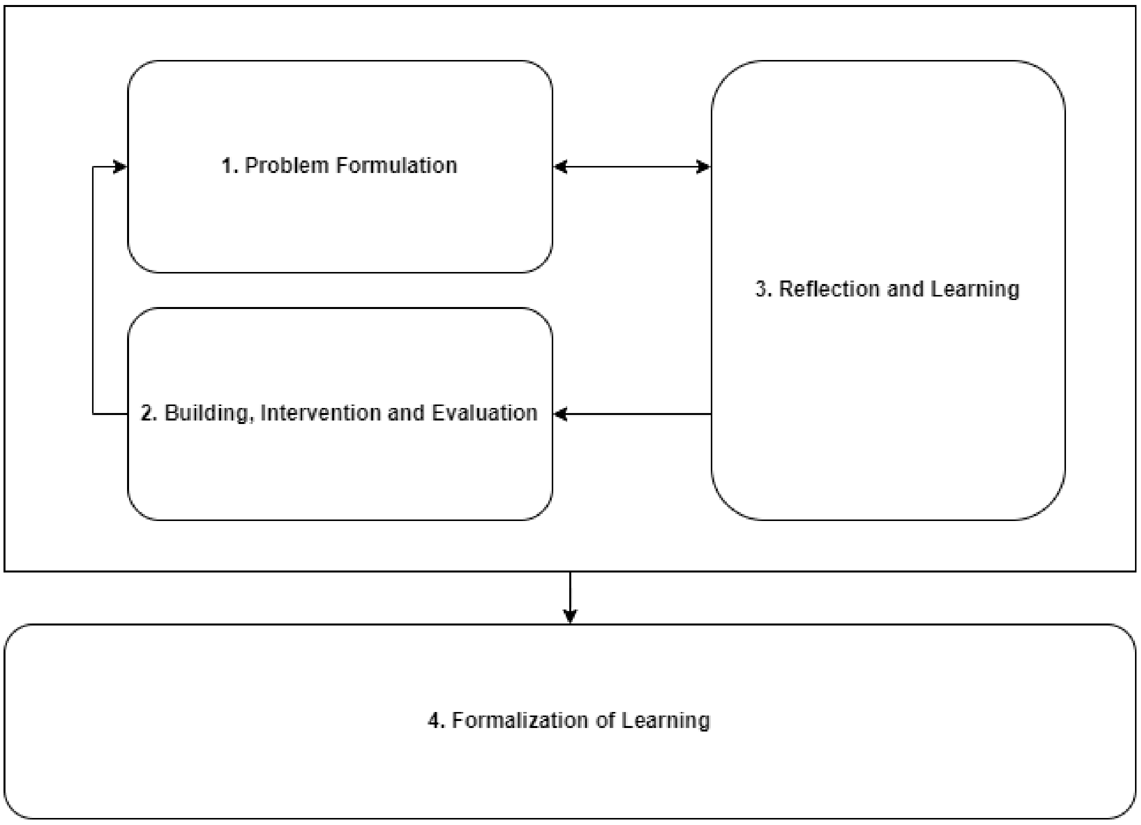

1. Problem Formulation
2. Building, Intervention and Evaluation
3. Reflection and Learning
4. Formalization of Learning

Figure 2.1: ADR Method Stages and Principles, adapted from (Sein et al., 2011)

2.2 Intervention

On the intervention side, I'm going to address my role in the Acro Companion environment and what impact it caused.

Acro Companion identified a flexibility problem that existed due to various conditions, and I proposed a solution that, once implemented, would fix the problem. After multiple iterations, the final solution fixed the problem.

During the intervention, it was my responsibility to not only look at what other businesses had done to address their flexibility problems, but also to find a solution that would make it possible for the company to expand its operations without running into this issue again. Following the completion of each implementation, it was important to determine, with the assistance of Acro Companion, if the solution would satisfy the requirements.

2.3 Theorizing in Design Science Research

Design Science Research and ADR center in a cycle of abstractions, (Lee et al., 2011) the paper provides grounds for building strong design theories in the design science paradigm.

The paradigm describes a circular dependence between the abstraction of the problem and the De-abstraction of the solution. (2.2)

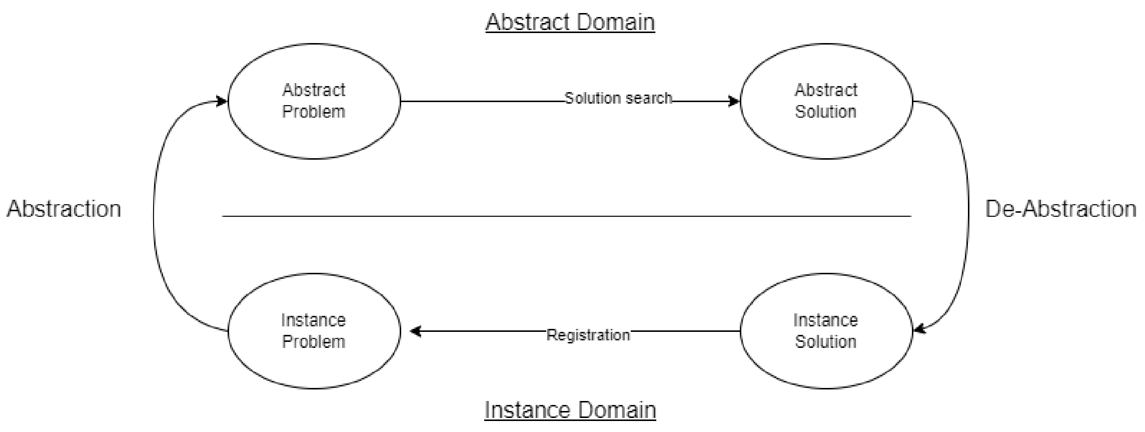


Figure 2.2: Design Theorizing Framework, adapted from (Lee et al., 2011)

A key assumption of this framework is that theorizing for design operates in two distinct domains: an abstract domain and an instance domain. In the abstract domain, a solution search process occurs in which an abstract solution is searched for an abstract problem. In contrast, the instance domain refers to where a particular solution is applied to address an instance or a particular problem. These two domains operate on their own independent grounds, with fewer constraints on how ideas are developed.

There are four activities in the theorizing framework, each represented by an arrow in Figure 2.2. These activities are abstraction, solution search, de-abstraction, and registration. Given that all four of these activities may take place as human thought, it may be possible that these occur not cyclically but simultaneously.

Abstraction - A theory is considered to be generalizable when it may be used in multiple contexts. Abstraction occurs when a researcher extracts common concepts or ideas from an instance problem by eliminating information related to the instance's context. This abstraction process entails a lot of cognitive judgment.

Solution search - Understanding the links between the afferent and efferent is fundamental to solution search theory. The afferent is a sensory world in which the outer environment is seen in terms of its condition, whereas the efferent is the world in which actions are done (e.g., solving the requirements of the problem).

De-Abstraction - Proposed solutions or design artifacts may be speculative or abstract notions throughout the solution search. The abstract notions must be restricted and instantiated before they can be evaluated in a specific situation. This de-abstraction entails the addition of context-specific features.

Registration - The registration activity is part of the theorizing process, but it is not necessarily independent of the design science research assessment process.

The framework proposed in this paper was used to orient toward building strong design theories in the design science paradigm.

To center Acro companion in the Design Science Research I started in the Instance Domain with an **Instance Problem**, that was issues in the user experience of Acro's web application. Through some search on the existing technologies, I arrived to a **Instance Solution** dynamic sidebar, from this I moved to the Abstract Domain, for the **Abstract Solution** was the possibility to divide the application in sections, the **Abstract Problem** that Acro have was that users were having troubles using the products because the interaction with the web application would vary from product to product.

2.4 Evaluating Design Science Research

To assist with the evaluation, I'm going to follow the framework presented and described in (Venable et al., 2012). Venable et al. propose that to address the evaluation gap by developing a Design Science Research (DSR) evaluation framework 2-by-2 (figure 2.3) with clear guidance for how one could design and conduct evaluation within DSR.

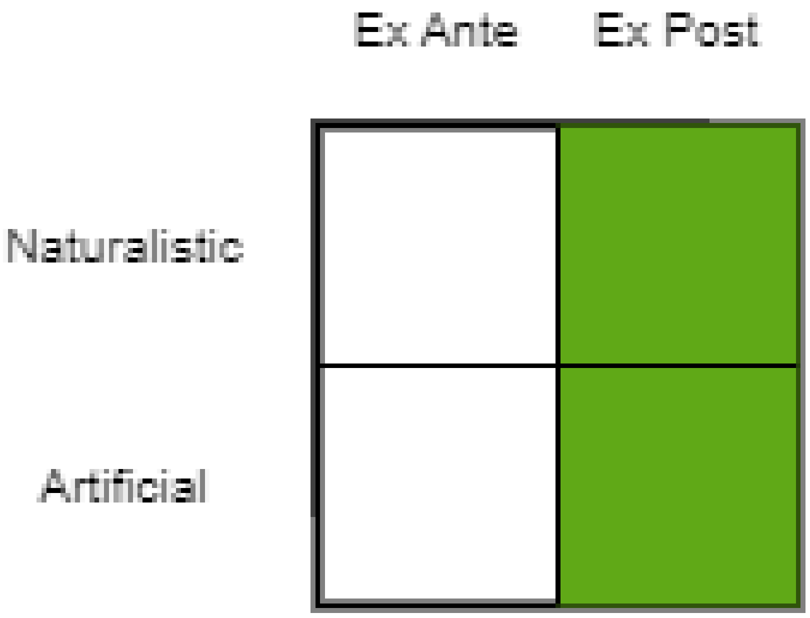


Figure 2.3: A Strategic DSR Evaluation Framework (adapted from (Venable et al., 2012)

This framework combines a dimension that contrasts artificial versus naturalistic evaluation with a dimension that contrasts ex ante and ex post evaluation. Ex post evaluation is referred to the examination of an instantiated item, whereas ex ante evaluation is referred to the evaluation of an uninstantiated artifact, such as a design or model.

According to the 2-by-2 matrix, evaluation in Acro Companion is divided into two significant segments:

1. Artificial Ex Post: This segment encompasses all assessments conducted prior to deploying any changes to the production or test environment. In the context of Acro Companion, this

involved a series of unit tests to validate the logical calculations and end-to-end tests to confirm the correctness of navigation paths.

2. Naturalistic Ex Post: This category pertains to evaluations conducted by users after the modifications have been deployed to the test environment. In Acro Companion's case, all these tests were conducted internally by members of the team. It's worth noting that Acro Companion also presented the new user interface to select clients for feedback. However, the feedback collection process remained under Acro's control, and I did not have direct influence over it.

Chapter 3

Dynamic sidebars

3.1 Evolution of the sidebar

Commonly located on the left or right side of the display, the sidebar is a frequent design element in web applications. It is a container that displays a user-accessible list of navigation items or other information.

The sidebar has evolved over time to become an integral component of contemporary web application layout. With the help of (Chen et al., 2017), I will split the sidebar evolution into phases:

Early web design (1990s to 2000s): In the early days of web design, sidebars served primarily as a separation between content and navigation. They were typically visually simple and static, and they offered only the most fundamental navigational options, such as links to other pages as demonstrated in Figure 3.1.

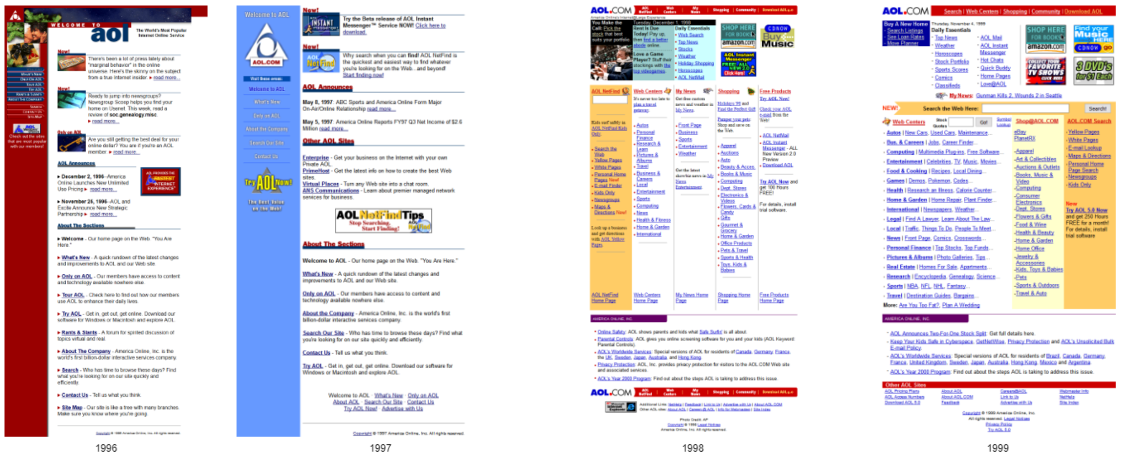


Figure 3.1: AOL 1990-2000

With the rise of social media platforms in the mid-2000s, the sidebar evolved to include social widgets and tools, including feeds, sharing options, and notifications as shown in Figure 3.2.

With the introduction of responsive web design in the 2010s, the sidebar became more flexible and adaptable to various screen sizes (web and mobile view). It became a hamburger menu on

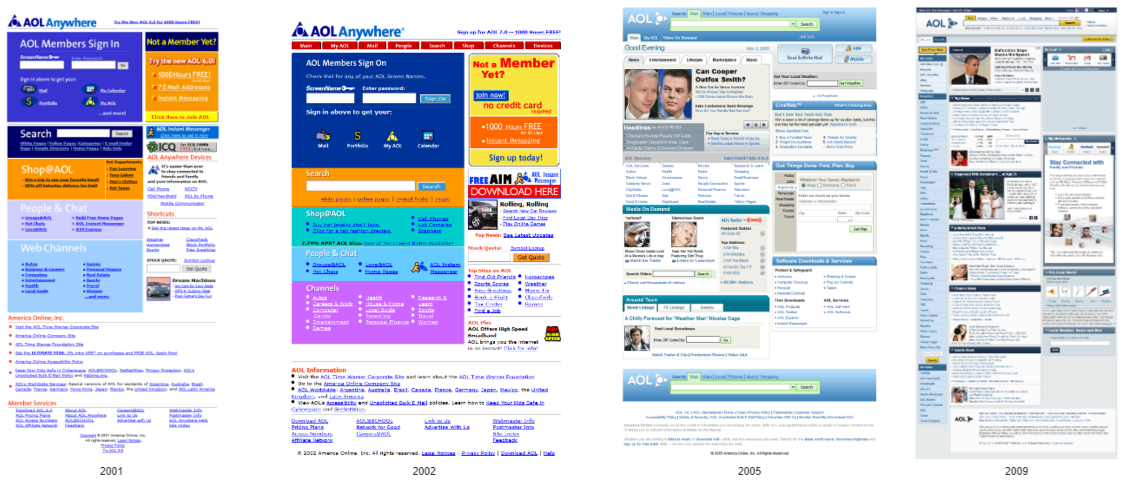


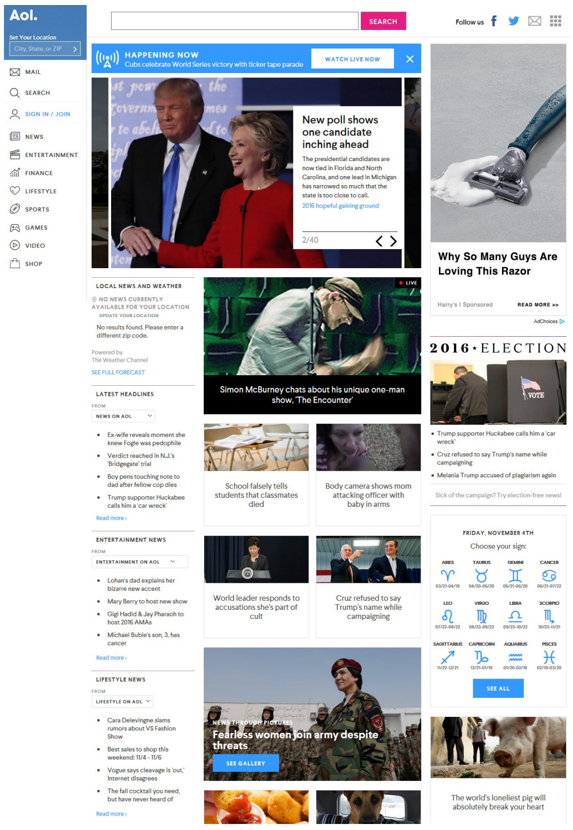
Figure 3.2: AOL 2000-2010

mobile devices and a full-screen menu on larger screens.

With the advancement of web technologies such as HTML5, CSS3, and JavaScript, the sidebar became more interactive and dynamic in the 2010s and 2020s. In addition to interactive widgets such as calendars, chat boxes, and search boxes, animations and transitions were added to enhance the user experience as displayed in Figure 3.3.

Modern web application design (2020s): The sidebar has become an integral part of the user interface in modern web application design. Not only is it used for navigation, but it also displays contextual information, notifications, and user settings. Additionally, the sidebar has become more personalized, with the ability to tailor its layout and content to the user's preferences and behavior as shown in Figure 3.4.

In conclusion, the sidebar has evolved significantly since its introduction to web design. Changes in technology, user behavior, and design trends have contributed to its development. Today, the sidebar is an integral component of the design of contemporary web applications, offering users a personalized and interactive experience.



Cubs celebrate World Series victory with ticker tape parade

New poll shows one candidate inching ahead

The presidential candidates are now tied in Florida and North Carolina, and one lead in Michigan has narrowed so much that the state is too close to call.

2016 hopeful gaining ground

NO NEWS CURRENTLY AVAILABLE FOR YOUR LOCATION
UPDATE YOUR LOCATION

No results found. Please enter a different zip code.

Powered by The Weather Channel

SEE FULL FORECAST

- Ex-wife reveals moment she knew Fogle was pedophile
- Verdict reached in N.J.'s 'Bridgegate' trial
- Boy pens touching note to dad after fellow cop dies
- Trump supporter Huckabee calls him a 'car wreck'

- Lohan's dad explains her bizarre new accent
- Mary Berry to host new show
- Gigi Hadid & Jay Pharaoh to host 2016 AMAs
- Michael Buble's son, 3, has cancer

- Cara Delevingne slams rumors about VS Fashion Show
- Best sales to shop this weekend: 11/4 - 11/6
- Vogue says cleavage is 'out,' Internet disagrees
- The fall cocktail you need, but have never heard of

Simon McBurney chats about his unique one-man show, 'The Encounter'

School falsely tells students that classmates died

Body camera shows mom attacking officer with baby in arms

World leader responds to accusations she's part of cult

Cruz refused to say Trump's name while campaigning

Fearless women join army despite threats

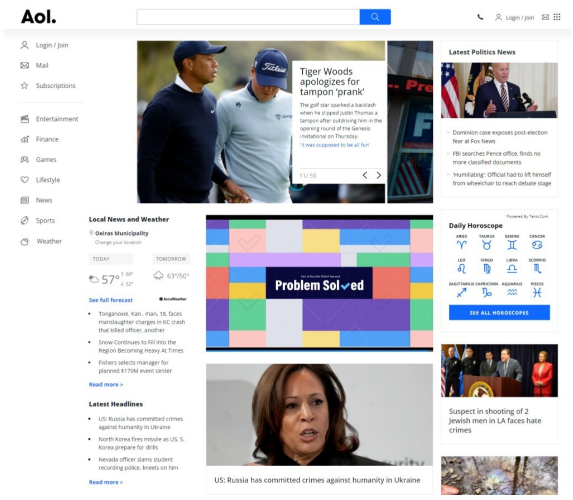
Why So Many Guys Are Loving This Razor

- Trump supporter Huckabee calls him a 'car wreck'
- Cruz refused to say Trump's name while campaigning
- Melania Trump accused of plagiarism again

Sick of the campaign? Try election-free news!

The world's loneliest pig will absolutely break your heart

Figure 3.3: AOL 2010-2020



Tiger Woods apologizes for tampon 'prank'

The golf star sparked a backlash when he slipped Justin Thomas a tampon after outdriving him in the opening round of the Genesis Invitational on Thursday.

'It was supposed to be all fun'

- Tonganoxie, Kan., man, 18, faces manslaughter charges in KC crash that killed officer, another
- Snow Continues to Fill into the Region Becoming Heavy At Times
- Fishers selects manager for planned $170M event center

- US: Russia has committed crimes against humanity in Ukraine
- North Korea fires missile as US, S. Korea prepare for drills
- Nevada officer slams student recording police, kneels on him

US: Russia has committed crimes against humanity in Ukraine

- Dominion case exposes post-election fear at Fox News
- FBI searches Pence office, finds no more classified documents
- 'Humiliating': Official had to lift himself from wheelchair to reach debate stage

Suspect in shooting of 2 Jewish men in LA faces hate crimes

Figure 3.4: AOL 2020-2023

3.2 Sidebar Reactivity

With the development of the dynamic sidebar solution, sidebars were created to be more than purely static screen components. They were designed to be reactive, which allows them to respond to user actions in real time. This is made possible by the deployment of a service that acts as an intermediary between the components and the sidebars.

When a user interacts with an element on the screen, such as by clicking a button or entering text into a form, the interaction generates a response in the service. The service then transmits this action to the relevant sidebar or component, which can subsequently adjust its state to reflect the screen changes as described in Figure 3.5.

This strategy is particularly effective because the sidebars are active players in the user experience, as opposed to merely being inert screen components. They can interact with one another and with the components, providing for a more integrated and smoother experience for the user.

This strategy symbolizes the future of sidebar design, since it permits greater flexibility and customization while retaining a high degree of responsiveness and usability. Acro Companion is able to remain at the forefront of web application development and give its clients with a cutting-edge user experience by using this dynamic, reactive approach to sidebar design.

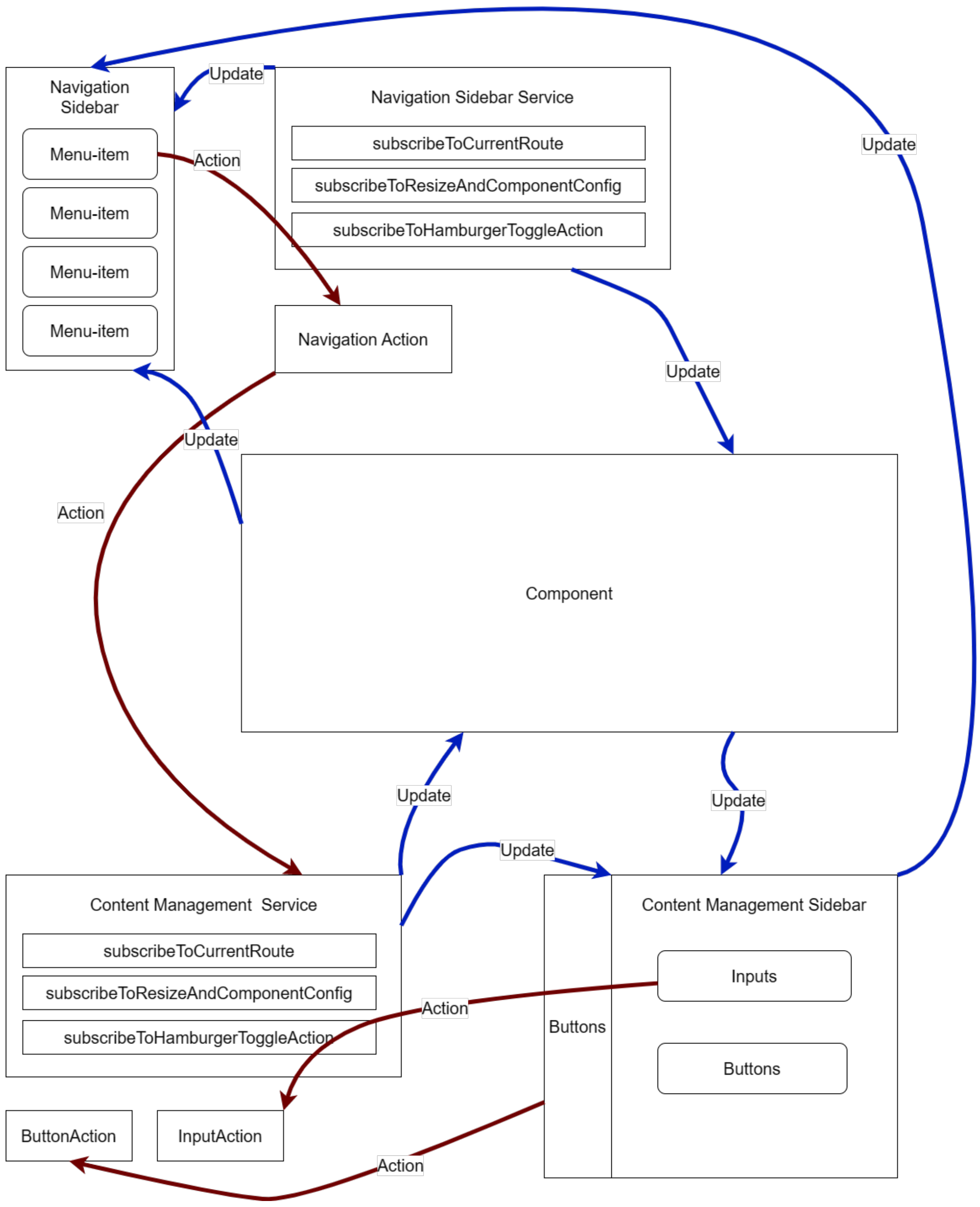


Figure 3.5: Sidebar reactivity

Chapter 4

Dinamic sidebar Proof of concept (First iteration)

In this chapter, I will discuss the first design cycle of the Acro Companion web application, including the iterations and decisions made in the Dinamic sidebar Proof of concept iteration to improve the user experience and interface, as well as address the flexibility issue.

4.1 Problem

Before I get into the specifics of the current Acro companion UX/UI design concerns, I'll explain how they arrived to this problem.

Acro Companion is still a startup, and the main objective has been to be flexible and give as many features as possible in order to attract customers and keep the company viable. All development has been requested by the primary users, who are coaches, clubs, and competition organizers, as described previously in the Organizational Setting section (1.2). When one of these users requires assistance, they contact the business analyst via email or in person during competitions.

If the business analyst authorizes the development, the design team will prepare a design that will be executed by the developers. As shown in Figure 4.1, we will go back and forth between the development team and the business analyst until it is cleared for deployment.

This interaction prompted the need to swiftly build and redevelop, and the easiest choice was to add multiple title bars to present all the content, resulting in the need to reassess and evolve the UX/UI. With this in mind, it's clear to see how the present solution wasn't flexible enough to support the rapid development of services or dynamic enough to only show what the user needed to view at any given moment.

The amount and complexity of Acro Companion's provided products have caused their UI to become inconsistent. As shown in 4.3, certain pages have sidebars, title bars with menus and sub-menus, and others have multiple navigation bars, which may mislead and confuse users.

The Acro Companion web application faced several issues related to user experience (UX). To address these issues, I decided to divide the web application into two sections: one for non-logged-in users and another for logged-in users.

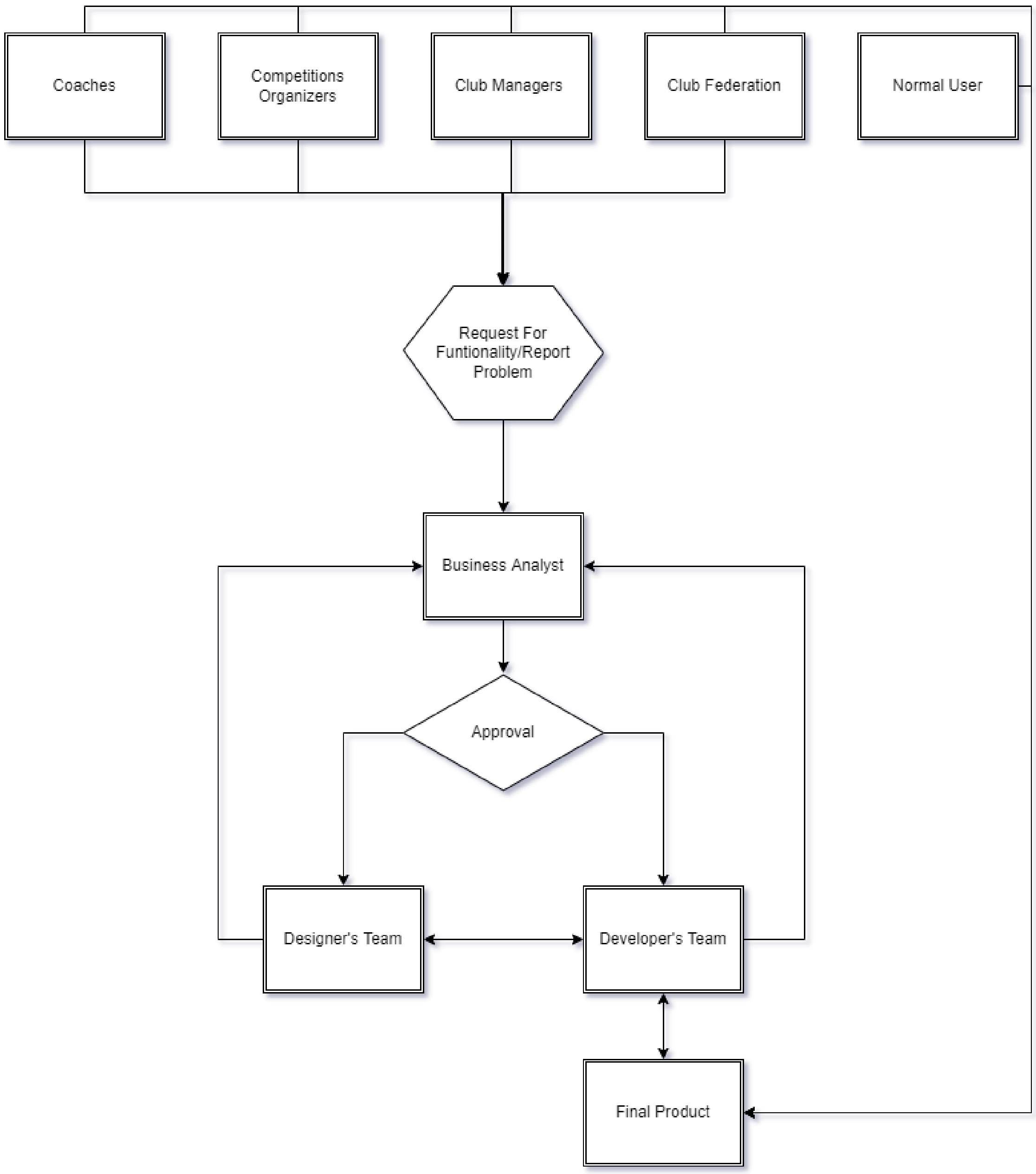


Figure 4.1: Company relation diagram

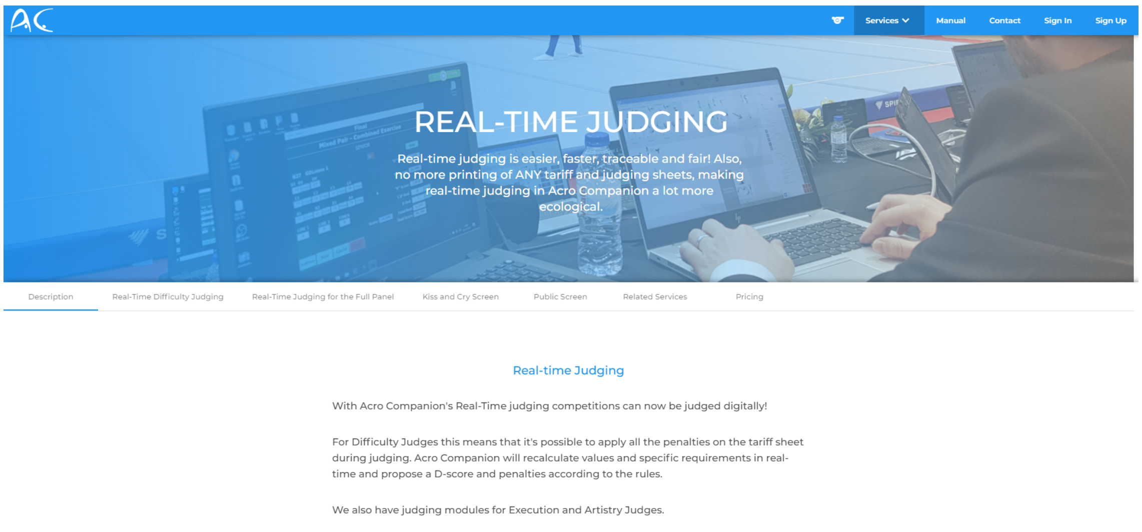
Users who were not logged-in could only view competitions, login, check available services, and view the manual pages. However, logged-in users' access can vary slightly because it is determined by their user level. It has all the features that a non-logged-in user has, plus additional options like profile management, sheet creation, sheet loading, and organization administration.

With this conceptual shift, it was possible to simplify the user interface by making it look like a simple blog and reusing the old title bar to help non-logged-in users navigate between pages.

The intention was to provide non-logged-in users with only the exact information and context necessary to understand the Acro Companion services and the gymnastics world. For the logged-in users, the intention was to provide a far more comprehensive UX for the users who spend more time on the Acro Companion web application.

The non-logged-in user interface was outstanding. The interface had features that were easy to access, comprehend, and use for all the operations that the user required. The UX was also satisfactory, though there were some minor issues with adapting the content to smaller displays, and some pages didn't feel cohesive with the design of Acro's remaining web application.

On the "Services" page, for example, the user must first scroll to see the page content that they previously clicked. As shown in Figure 4.2, the title and description take up half of the page.



REAL-TIME JUDGING

Real-time judging is easier, faster, traceable and fair! Also, no more printing of ANY tariff and judging sheets, making real-time judging in Acro Companion a lot more ecological.

Real-time Judging

With Acro Companion's Real-Time judging competitions can now be judged digitally!

For Difficulty Judges this means that it's possible to apply all the penalties on the tariff sheet during judging. Acro Companion will recalculate values and specific requirements in real-time and propose a D-score and penalties according to the rules.

We also have judging modules for Execution and Artistry Judges.

Figure 4.2: Real-Time Judging Service page

The user interface for logged-in users, on the other hand, was complicated and, at times, unpleasant to use. Figure 4.3 shows how multiple title bars were displayed on some pages to move between distinct content or pages.

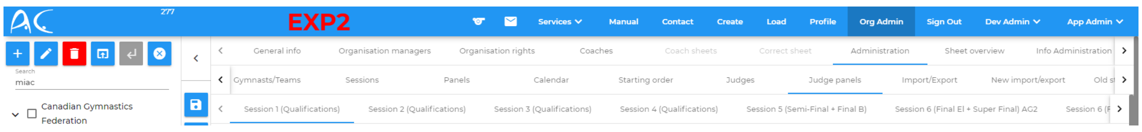


Figure 4.3: Multiple title bars

As a result, the most important problem to solve was the UI/UX for logged-in users; once solved, it may bring more users back to Acro Companion.

4.2 Design/building

To improve the UX/UI issues faced by Acro Companion and described in 4.1, I identified two possible solutions. Both solutions, would involve an assessment of the UX design, including the redesign of certain pages to better fit Acro Companion's new vision. The UI design solution for logged-in users, I could either implement a "Dynamic Title bar" or a "Dynamic Sidebar" solution.

Dynamic Title bar Solution: During the first problem analysis of Acro Companion's web application, it became clear that some pages required more than one title bar so that users could move between sub-pages. The idea presented in the team meeting was a single title bar, and its content would change depending on user context. When the user selects a button from the first title bar, the title bar updates to show the child buttons only visible from that component, as shown in the mockups below (4.4).

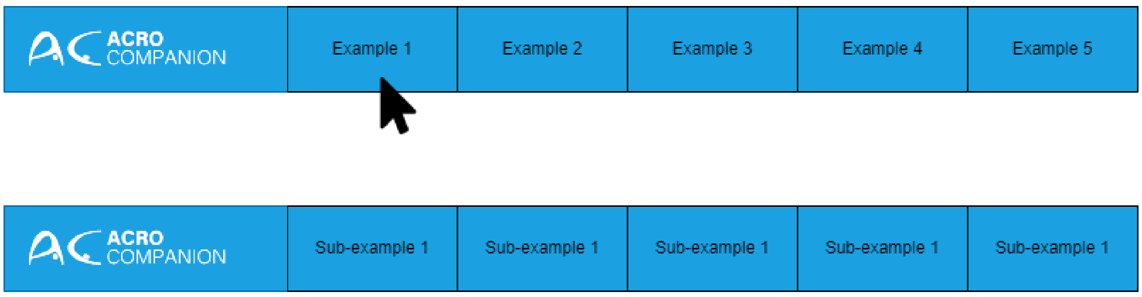


Figure 4.4: Title bar mockup

In comparison to the current solution, the dynamic title bar would address the following issues:

1. Because the dynamic title bar would "host" all the possibilities in a single bar, the title bar stack problem would be eliminated.
2. Because this solution is very similar to the one that was implemented, the period for user adaptation would be much shorter.

Dynamic Sidebar Solution: The concept is the same as the dynamic title bar solution, but with a different approach, we would have a single sidebar whose content changes depending on the user context (e.g., user type, user level, user route). As shown in Figure 4.5, this sidebar would remain on the left side and could be hidden to save space if needed by the user.

In comparison to the current solution, the dynamic sidebar would address the following issues:

1. Because the dynamic sidebar would "host" all the options in a single sidebar container, this solution would eliminate the title bar stacking issue.

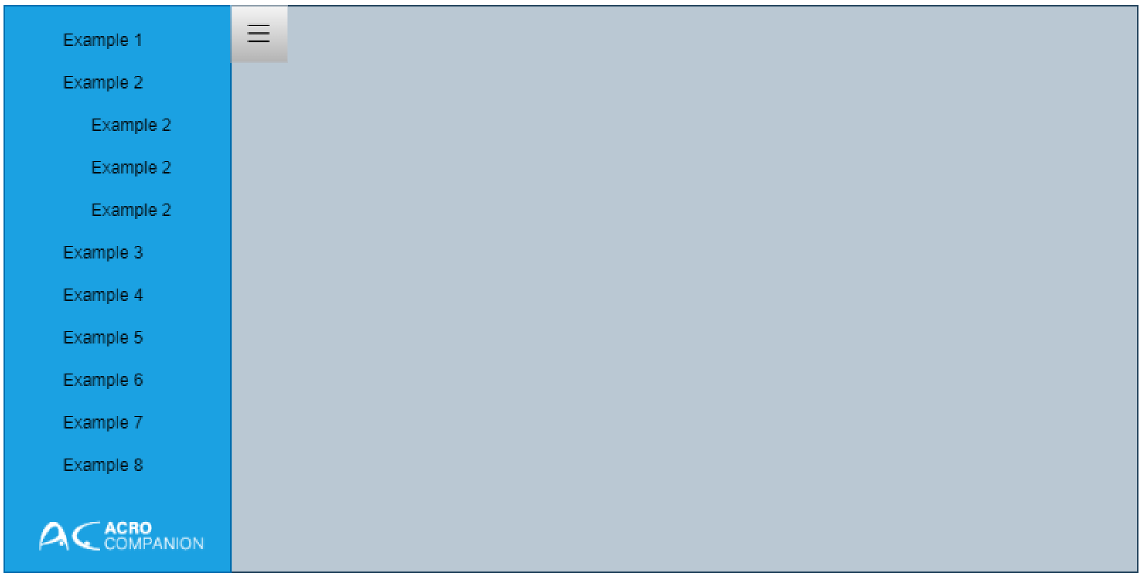


Figure 4.5: Sidebar bar mockup

2. The larger form factor of this solution would make it easier to increase its complexity for future development by simply adding a new component to the sidebar.

3. All product modifications are now centralized in the sidebar for greater consistency and ease of use for the user.

4. The dynamic sidebar can be hidden to save space if desired by the user.

After comparing the two solutions, it became clear that the dynamic sidebar would be the better option for addressing the issues with Acro Companion's UX and UI, because the dynamic title bar would not solve all the existing issues and could cause significant implementation difficulties. Some drawbacks stemmed from the fact that having a small area to display content could lead to some UX/UI problems, such as the difficulty in displaying the navigation's progression or the impossibility of going to a parent page after entering a child page without implementing a method to return to the previous page.

The dynamic sidebar solution, on the other hand, did not have any of the issues mentioned above and had other advantages. Because the sidebar is a large form factor, it would be easier to fit all the content there and customize with icons and animations that are important for the user experience. Another consideration is that monitors are wider than they are tall, so the title bar would take up a lot of space that is already limited. And, while the title bar would need to be always visible on smartphones, we can hide it with the sidebar.

In the end, the main UI element (the sidebar) was designed based on the principles of affordance theory. According to (Stendal et al., 2016), "affordance" refers to the "action possibilities" or capabilities latent in the environment, regardless of the individual's ability to recognize them, but always in relation to the actors and therefore dependent on their capacities.

When we conceptualize the sidebar, we have action possibilities and user intents. When applied to the custom sidebar solution, the sidebar is dynamic and only displays necessary options, providing the user with relevant options. The dynamic sidebar can be represented as two main affordances: the first one is that the dynamic sidebar is a affordance as a whole, it only appears for specific users and adapts it's structure regarding the context. The other affordance is that the content displayed inside the dynamic sidebar has affordances that adapt the content regarding the user type.

Affordances are more important for logged in users because different levels of users have access to different states of the products or restrictions.

All concerns were presented to Acro Companion, and the dynamic sidebar solution was chosen in response.

4.3 Intervention

The development of the new custom sidebar will have a substantial impact on the UI/UX since it will consolidate navigation and content in one location, resulting in a much easier interaction for the user and mitigating the issues depicted in Figure 4.3.

Furthermore, the sidebar delivers a more agile solution. Technically, the sidebar will be able to mimic menus and pages, and it is essential that they are different components. This approach will greatly simplify and structure future development.

This sidebar's sophistication goes beyond the typical sidebar used only for page navigation. The goal is to build a custom component that modifies the user's content based on the path/URL as well as the user's current state and level. Because of the web application's complexity, all sidebar content will be lazy loaded to save network bandwidth and resources and improve the web application's load time.

As a proof of concept, the first interaction was created, followed by the properties listed below:

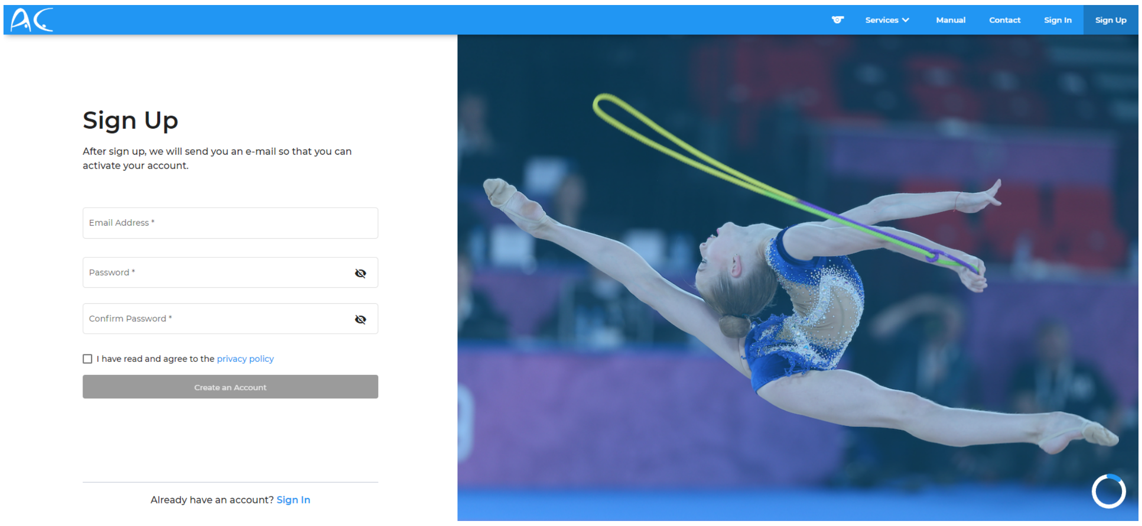
1. Added some extra properties to make fine management and control precisely what happens to the sidebar;
2. Incorporate existential elements of user management to the sidebar;
3. Restructuring web application entry point to competition overview;
4. Integrate the old sidebar from Manual page to the new custom sidebar.

These properties allowed me and Acro to determine whether the solution chosen had everything necessary to be flexible and adaptable to the current Acro Companion web application. Beginning with item 3.

The login component in the Acro Companion web application was originally divided into two parts: one that was a login component that had a picture switcher, and the other that included a router outlet where Sign in, Sign Up, Forgot Password, and Reset Password were loaded. In item

2, the objective was to incorporate all the login components that were loaded onto the router outlet into the sidebar, eliminating the need to have the router outlet load and switch between all the login components.

With this change, the login component would only include the picture switch, while the sidebar would handle loading the necessary login components. As shown in Figure 4.6, the left half of the image represents the sidebar, which is where all possible login pages are lazy loaded, and the right half is the login component.



After sign up, we will send you an e-mail so that you can activate your account.

Figure 4.6: Login Sign-up Redesign

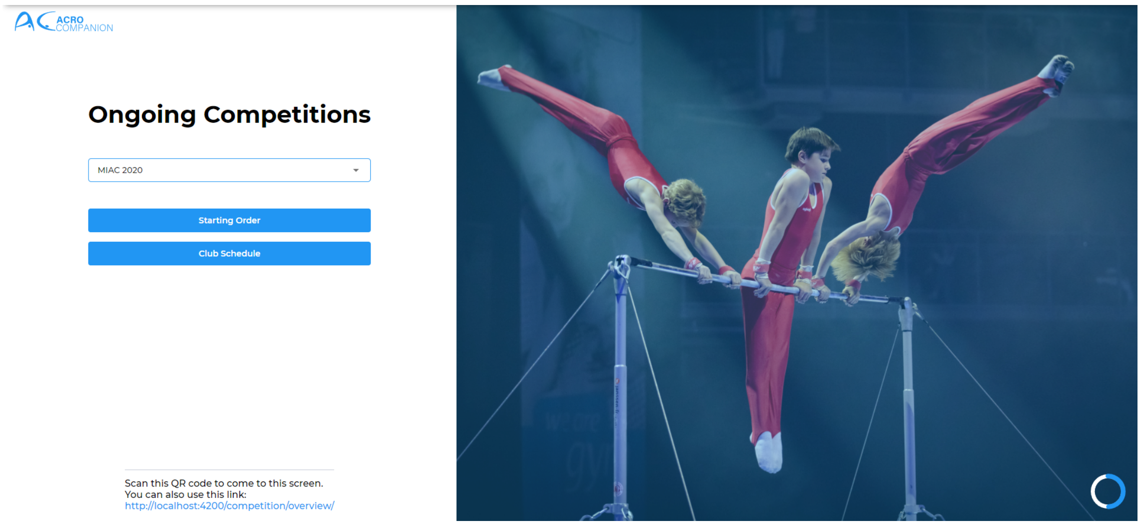
The implementation of the custom sidebar for the competition overview page involved moving the functionality side, previously located on the left of the page, to the sidebar. The picture switcher, which provided the user with a preview and context for the competition, remained on the competition component. This can be seen in Figure 4.7. The goal of this change, together with item 2, was to improve the user experience and make navigation easier by concentrating content and navigation in a single location and increasing the consistency between similar components.

In item 4, we aimed to move the angular material sidebar to the custom sidebar while retaining all its functionality. As shown in Figure 4.8, this relocation necessitated some changes to the properties of the sidebar. Previously, whenever a user selected an element, the URL would change, causing the sidebar to reload. However, in this case, it was required that the router change without triggering a router change.

I was able to determine whether the chosen strategy would work for the Acro Companion web application after implementing these four items, and I concluded that it met all the requirements described in the Design and Building section (4.2).

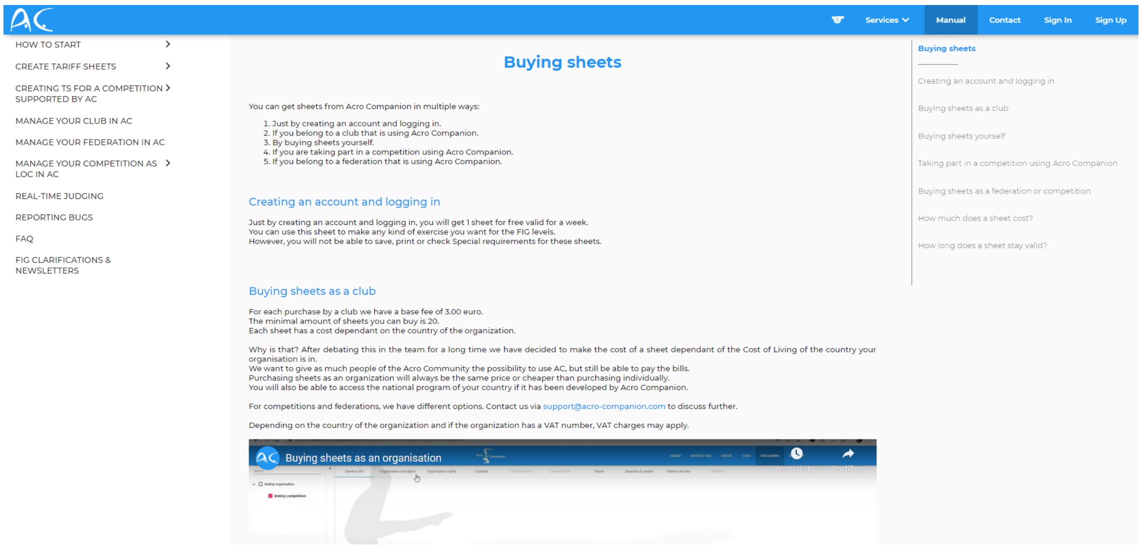
4.4 Practical implementation

The dynamic sidebar's content is intended to change based on the user's current location within the web application. To manage this feature, I built a sidebar service that listens for changes to



Scan this QR code to come to this screen.
You can also use this link:
http://localhost:4200/competition/overview/

Figure 4.7: Competition Overview page with sidebar


Services
Manual
Contact
Sign In
Sign Up
HOW TO START
CREATE TARIFF SHEETS
CREATING TS FOR A COMPETITION SUPPORTED BY AC
MANAGE YOUR CLUB IN AC
MANAGE YOUR FEDERATION IN AC
MANAGE YOUR COMPETITION AS LOC IN AC
REAL-TIME JUDGING
REPORTING BUGS
FAQ
FIG CLARIFICATIONS & NEWSLETTERS
Buying sheets
Creating an account and logging in
Buying sheets as a club
Buying sheets yourself
Taking part in a competition using Acro Companion
Buying sheets as a federation or competition
How much does a sheet cost?
How long does a sheet stay valid?
Buying sheets as an organisation

Buying sheets

You can get sheets from Acro Companion in multiple ways:

1. Just by creating an account and logging in.
2. If you belong to a club that is using Acro Companion.
3. By buying sheets yourself.
4. If you are taking part in a competition using Acro Companion.
5. If you belong to a federation that is using Acro Companion.

Creating an account and logging in

Just by creating an account and logging in, you will get 1 sheet for free valid for a week.
You can use this sheet to make any kind of exercise you want for the FIG levels.
However, you will not be able to save, print or check Special requirements for these sheets.

Buying sheets as a club

For each purchase by a club we have a base fee of 3.00 euro.
The minimal amount of sheets you can buy is 20.
Each sheet has a cost dependant on the country of the organization.

Why is that? After debating this in the team for a long time we have decided to make the cost of a sheet dependant of the Cost of Living of the country your organisation is in.
We want to give as much people of the Acro Community the possibility to use AC, but still be able to pay the bills.
Purchasing sheets as an organization will always be the same price or cheaper than purchasing individually.
You will also be able to access the national program of your country if it has been developed by Acro Companion.

For competitions and federations, we have different options. Contact us via support@acro-companion.com to discuss further.

Depending on the country of the organization and if the organization has a VAT number, VAT charges may apply.

Figure 4.8: Manual page with sidebar menu on custom sidebar

the URL and determines which content to display in the sidebar based on the current path. For example, we will receive an update whenever the URL changes, and depending on the path, we will have the option to show or hide the custom sidebar, and if we show the sidebar, the content will change.

The main challenge was to find the best solution that would not have an impact on the entire web application while remaining as adaptable as possible.

1. Two router-outlets
   (a) One for sidebar
   (b) One for content
2. Sidebar as ngIf
   (a) One for sidebar - Component in app-component called sidebar
   (b) One for content - router-outlet
3. Dynamic ng-content for sidebar
   (a) One for sidebar - Component in app-component called sidebar
   (b) One for content - router-outlet

An Angular router outlet is a placeholder that Angular dynamically fills based on the current router state. It is used to render the component for the current route. In other words, it is a container that is used to render the view for a particular route. It is represented by the "¡router-outlet¿" element in the template. When the router navigates to a new route, it uses the router outlet to place the component for that route into the view.

In item 1, the risks were: whether it was feasible to develop something that worked; the increased complexity of communicating between the sidebar and the content; the need to change the URL paths because the components are loaded into an existing component, and it would be different if we had two router-outlets. This option prevents the end user from bookmarking the Acro Companion Web application link.

The risks in item 2 as ngIf were the number of ngIfs on the sidebar component to determine which component to show, which would be difficult to understand and maintain over time.

In item 3 , the risks were whether it was feasible to build anything that worked due to the complexity of the web application.

The ability to implement lazy loading is the most significant opportunity for all items. After some deliberation about the risks and benefits of each option, the Dynamic ng-content for sidebar approach, would allow the current routing structure to be maintained while only adding a new component to the app component, which would be the custom sidebar. Inside this new component, there would be an ng content with the sidebar options, which would be dynamic, lazy-loaded, and would change based on the router path. I meet all the main requirements for the web application with this option.

The first step was to develop a way to load components in a lazy manner; otherwise, switching to a sidebar component would be a regression to the fluidity of the web application. There were several technical issues, such as the inability to directly load a component into another component dynamically. The workaround method was to load the module and ”build” the component within it.

Following the completion of this process, the next steps in testing the implementation were to incorporate and restructure some of the existing sidebar components, as described in the section 4.2.

Sidebar Specification

In this section, I will be explaining the sidebar component and the service that manages it. The Sidebar Component has an Array type of ”DynamicComponentStructure” objects each object or in other words dynamic component specification is defined by:

Table 4.1: Sidebar Specification

Specification	Description
routerStarts	Defines when that component will be shown
lazyCompoent	Is the Component that will be shown
lazyModule	Used to load in a lazy module the component
sidebarWidth	Specify the sidebar width
sidebarPosition	Specify the default sidebar position
SidebarButtonPosition	Define the default sidebar button position

The array of objects is then used in a function that listens for URL changes and loads the appropriate component into the sidebar component based on those changes. When the component has been created, it must be resized to fit the screen. So I wrote a custom function to handle it, which will make use of the recently populated Observables from the previous function. I adjusted the component using the ”DynamicComponentStructure” properties.

To summarize, there are six major steps in this sidebar component:

1. Listening to the url;
2. Find the correct component;
3. Clear the container reference of the previous component loaded;
4. Populate all Subjects;
5. Load the new dynamic component;
6. Resize the component to adjust to the screen;

The service that handles communication between the sidebar component and the app component is the SidebarService, which hosts all subjects and observables as well as all the setters used to manage all the properties of the sidebar. The subjects defined in this service are:

1. _sidebarPosition;
2. _sidebarWidth;
3. sidebarDisplayBool;
4. _sidebarButtonPosition;
5. _sidebarComponent;
6. _sidebarButtonState;
7. _sidebarState;

4.5 Evaluation

To help explain the Acro Companion evaluation of the Custom sidebar implementation and design changes, the previously described 2-by-2 Matrix in 2.4 will help understand how and why the evaluation was done. To begin, we can divide the evaluation into two major parts:

- In house evaluation, Artificial Ex Post and Naturalistic Ex Post.
- Users evaluation, Naturalistic Ex Post.

The solution was deployed to the experimental environment after the Dinamic sidebar Proof of concept iteration was completed, allowing Acro Companion to begin performing naturalistic ex post evaluations in the sidebar.

According to Acro's team, the core sidebar solution architecture was robust but not yet ready for production. The solution appeared to be flexible enough to address the existing issues, but at the time, multiple areas of the web application were not integrated with the sidebar. So, there was no point in continuing to evaluate the solution in an Artificial Ex Post manner.

4.6 Reflection

To conclude this chapter, the key difficulties assigned were from the Acro Companion web application's logged-in section. The search for a solution and the debates about it all started with this first key UX/UI review, which came up with the principles "need to be flexible and agile."

Chapter 5

Navigational sidebar (Second iteration)

During the navigational sidebar iteration, I continued to develop and expand the sidebar solution introduced in the initial iteration. This chapter presents the adjustments made to address the challenges related to the flexibility and dynamic nature of the web application's user experience and interface design. The objective is to provide a solution that addresses the need for a more flexible and agile web application capable of supporting the rapid development of services and meeting the needs of the primary users, which was not met by the initial iteration.

5.1 Problem

This iteration was prompted by the need to transform the concept (prototype) created in the previous iteration into a production-ready solution.

The first version of the dynamic sidebar solution only adapted six components (sign-in, sign-up, forgot password, reset password, manual, and competition overview). While none of these pages had UX issues, it was easiest to test proof of concept without breaking major functionality on these pages.

To develop a production-ready solution, however, it is necessary to address the vast majority of UX issues in the logged-in portion of the web application.

5.2 Design/building

In this section, the goal was to find a solution that was stable enough for production. There were two primary issues that required to be addressed, robustness and extension:

- Extension of dynamic sidebar components;
- Addition of a navigation sidebar.

A mockup of the dynamic sidebar with two states was designed to demonstrate the functionality and appearance of the updated version of the sidebar. The first state displays buttons for changing the content of the main component, while the second state of the sidebar shows the navigation content.

Besides the hamburger button that was already added in the Dinamic sidebar Proof of concept iteration, The switch button was also added to provide an alternative way for users to toggle between the two states of the sidebar.

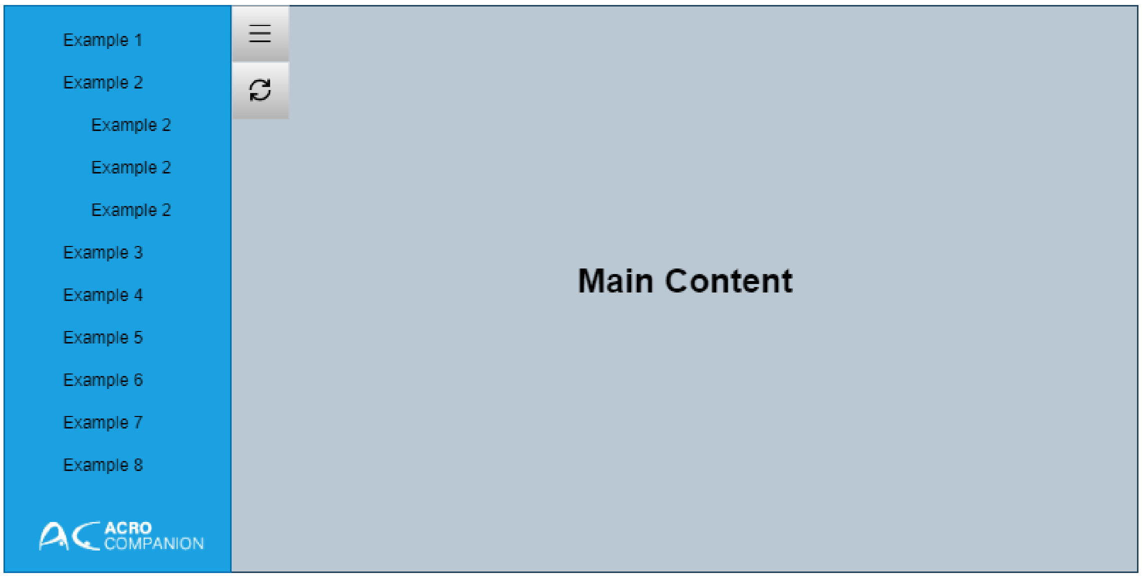


Figure 5.1: Sidebar with navigation state mockup

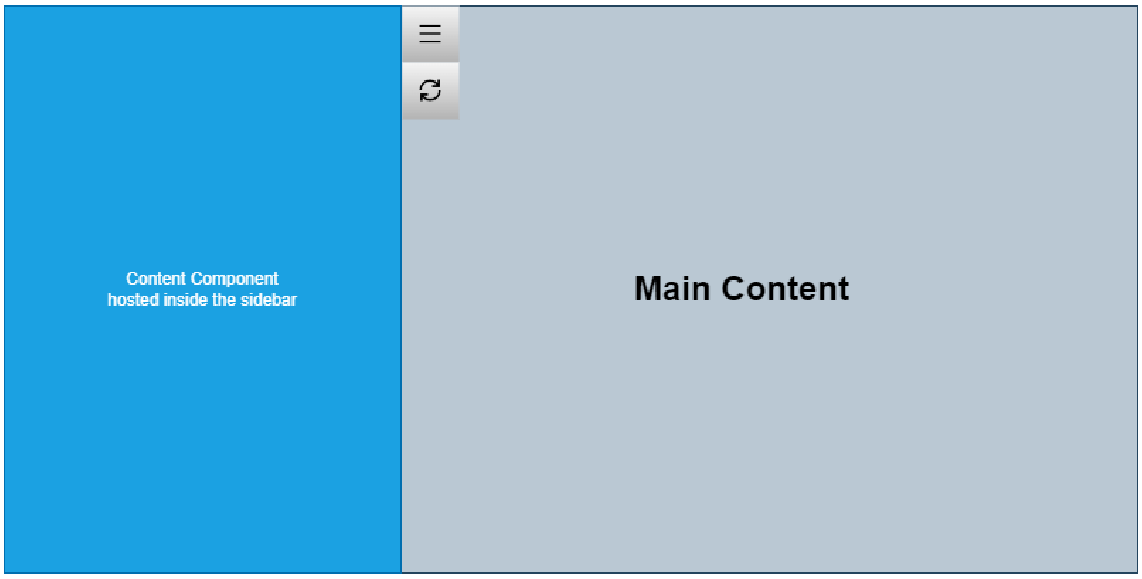


Figure 5.2: Sidebar with component state mockup

Extension of dynamic sidebar components:

The extension of dynamic sidebar components was implemented by evaluating and compiling a list of components that required relocation or modification to the sidebar, including:

- Livestream component;

- Sheet-view component;
- Registrations information component;
- Verify email component.

For each new dynamic sidebar component, four main files were created: a typescript file containing all the HTML and component logic, a SCSS file containing all custom styling, a module containing all necessary modules, and another typescript file containing all the communication between the sidebar and the component.

Changes to the programming paradigm complicated the transition of the dynamic sidebar components. The previous implementation made use of a functional programming paradigm, which resulted in the development of all components within a single component without external communication. To achieve a more dynamic solution, however, a paradigm shift to reactive programming was required. This paradigm, which is predicated on the notion that everything is a stream observer and observable, demanded extensive low-level code development to ensure component compatibility and functionality. This aspect of the development process will not be discussed further in the thesis, as it has no bearing on the final outcome of the sidebar.

Addition of the navigation sidebar:

The addition of the navigation sidebar was accomplished by dividing it into three smaller sidebar components to make it easier for developers to work on specific components without interfering with the user's ability to navigate between pages. The proposed end-user structure aims to improve readability and consistency. As described in subChapter Design/building 4.2 with respect to user level and user permissions, the content would adjust. The three selected components are:

- Main menu (5.4).
- Organisation menu (5.5).
- Developer menu (5.6).

In the top was added a button group to navigate between the different navigation components (Main menu, Organisation menu, Developer menu).

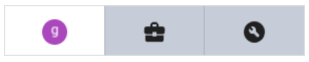


Figure 5.3: Button group

The main menu displays the user's profile picture and name. As shown in Figure 5.4, the tariff sheet buttons and all the buttons that the user has when not logged in have been added to this menu.

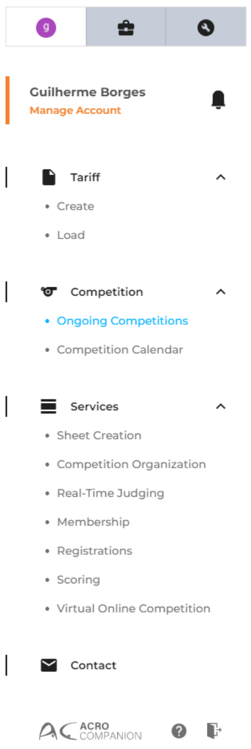


Figure 5.4: Main sidebar (navigational sidebar iteration)

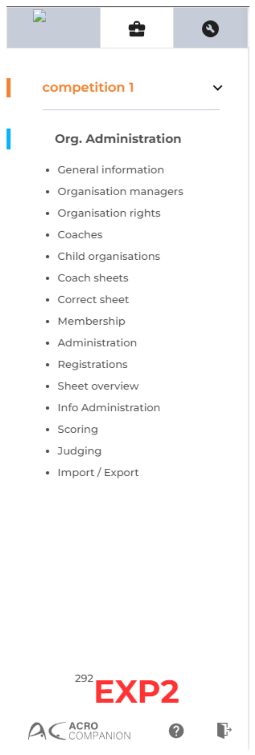


Figure 5.5: Organisation sidebar (navigational sidebar iteration)

The Organisation menu, have all the buttons necessary to manage Federations, Clubs or competitions as we can see in Figure 5.5

As shown in Figure 5.6, the Developer menu contains only buttons accessible to developers and users with user level greater than 55.

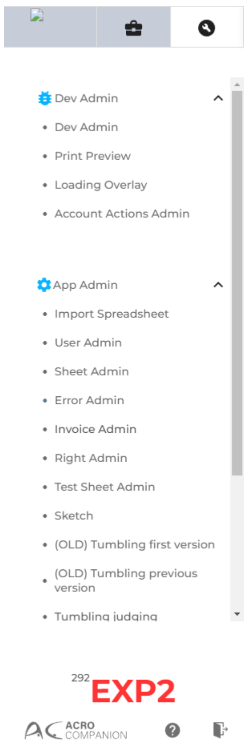


Figure 5.6: Developer sidebar (navigational sidebar iteration)

5.3 Intervention

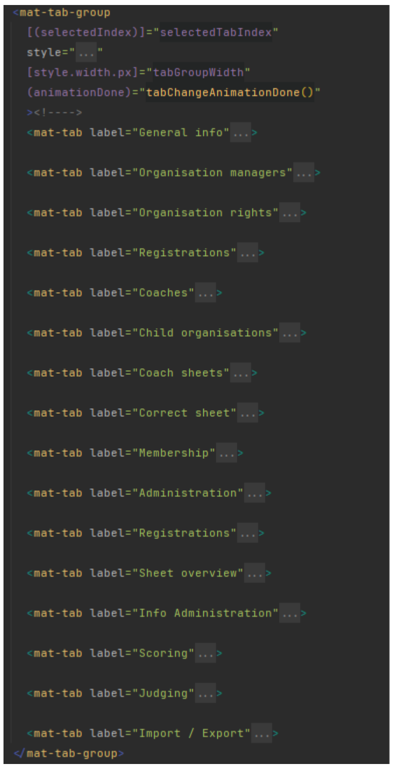
In the intervention, the organization component of the Acro Companion web application had to have its architecture modified.

The previous architecture consisted of a parent component that contained all the organization's components, as well as the logic used to display and conceal each component and calculate user context.

As shown in 5.7, whenever a subcomponent is added to the parent component, it is necessary to accommodate all the logic within the organisation component without breaking what has already been implemented.

The implemented architecture was difficult to scale and complicated to debug when presented with bugs. The architectural transformation had three primary goals and steps:

- Remove child components from main organisation component;
- Fix and update all the routing for the components;



```
<mat-tab-group
  [(selectedIndex)]="selectedTabIndex"
  style="..."
  [style.width.px]="tabGroupWidth"
  (animationDone)="tabChangeAnimationDone()"
  ><!---->
  <mat-tab label="General info"...>

  <mat-tab label="Organisation managers"...>

  <mat-tab label="Organisation rights"...>

  <mat-tab label="Registrations"...>

  <mat-tab label="Coaches"...>

  <mat-tab label="Child organisations"...>

  <mat-tab label="Coach sheets"...>

  <mat-tab label="Correct sheet"...>

  <mat-tab label="Membership"...>

  <mat-tab label="Administration"...>

  <mat-tab label="Registrations"...>

  <mat-tab label="Sheet overview"...>

  <mat-tab label="Info Administration"...>

  <mat-tab label="Scoring"...>

  <mat-tab label="Judging"...>

  <mat-tab label="Import / Export"...>
</mat-tab-group>
```

Figure 5.7: Organisation Admin Component

- Move all the logic to multiple services with observables.

Implementation of the sidebar was significantly delayed and lengthened due to the need to alter the architecture of a major component that hosts all the main products. The enhancement to the codebase's long-term maintainability or extensibility made this change optional, but it was deemed too beneficial to forego.

Because the logic had to be migrated and adapted to other areas of the application, this change impacted all the tests that had already been developed to test that side of the application, as well as introducing numerous failure points.

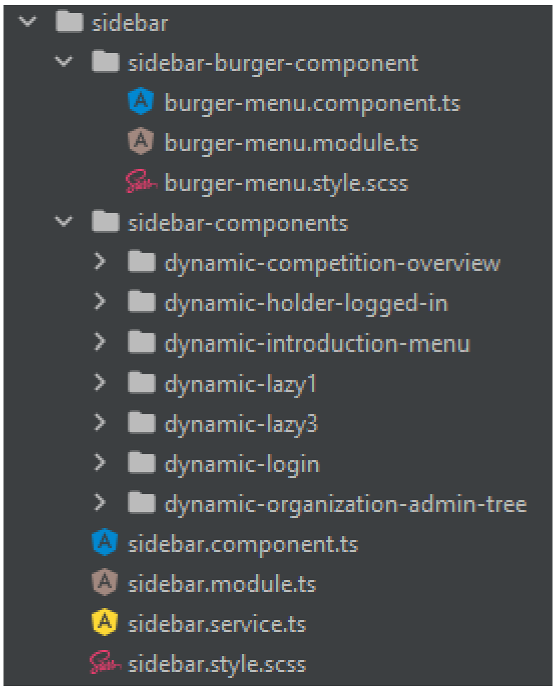
Even though the updated version of the sidebar was complete, it was not deployed due to its impact on all the tests and the risk of introducing failures. During a period of intense competition, this option was not viable.

Despite this, the Acro team welcomed the new architecture of the organization's administration, which featured independent components and services, because it would improve the implementation structure.

5.4 Practical implementation

For this practical implementation, the dinamic sidebar proof of concept iteration's logic was refined and additional logic was added to handle the transition between the sidebar with navigation content (Figure 5.1) and the sidebar with buttons to change the content of the main component (Figure 5.2).

Additionally, a new file structure was defined, which included a directory for the sidebar component and subdirectories for the sidebar button component and all other sidebar components. This new structure provided Acro Companion with the organization it lacked and required to grow.



```
sidebar
  sidebar-burger-component
    burger-menu.component.ts
    burger-menu.module.ts
    burger-menu.style.scss
  sidebar-components
    dynamic-competition-overview
    dynamic-holder-logged-in
    dynamic-introduction-menu
    dynamic-lazy1
    dynamic-lazy3
    dynamic-login
    dynamic-organization-admin-tree
  sidebar.component.ts
  sidebar.module.ts
  sidebar.service.ts
  sidebar.style.scss
```

Figure 5.8: File structure (navigational sidebar iteration)

Two States Sidebar Specification

This section will describe the sidebar component and the service that controls it. Despite the fact that the definition was similar to the dinamic sidebar proof of concept iteration, some variables were necessary to define it more precisely. The Sidebar Component has an array of ”DynamicComponentStructure” objects, with each object’s dynamic component specification defined as follows:

This array of objects is then utilized by a function that monitors URL changes and, based on those changes, loads the appropriate component into the sidebar component. Once the component has been created, it must be adjusted to the appropriate screen size. Consequently, I created a custom function to manage it by utilizing the recently populated Observables from the preceding function. There, I adjusted the component using the ”DynamicComponentStructure” properties.

In conclusion, the six major steps in this sidebar are identical to those described previously in the sidebar definition section (5.4):

SidebarService is the service that manages communication between the sidebar component and the app component. It contains all subjects, observables, and setters used to manage the sidebar’s properties. These are the subjects defined by this service:

Table 5.1: Sidebar Component Specification

Specification	Description
routerStarts	Defines when that component will be shown
lazyCompoent	Is the Component that will be shown
lazyModule	Used to load in a lazy module the component
sidebarWidth	Specify the sidebar width
sidebarPosition	Specify the default sidebar position
sidebarButtonPosition	Define the default sidebar button position
hasNavigation	Define the if page on a specific route had the navigation sidebar
hasDynamicComponent	Define the if page on a specific route had the dynamic component sidebar
hasHamburgerButton	Define the if page on a specific route had a close sidebar button
hasSwitchButton	Define the if page on a specific route had a switch sidebar button

1. _sidebarPosition;
2. _sidebarWidth;
3. sidebarDisplayBool;
4. _sidebarButtonPosition;
5. _sidebarComponent;
6. _sidebarButtonState;
7. _sidebarState;
8. _sidebarHasNavigation;
9. _sidebarHasDynamicComponent;
10. _sidebarHasHamburgerButton;
11. _sidebarHasSwitchButton;

The sidebar component had only three functions: one for handling all route changes and component loading, another for handling resize events, and the last for switching between sidebar contents.

5.5 Evaluation

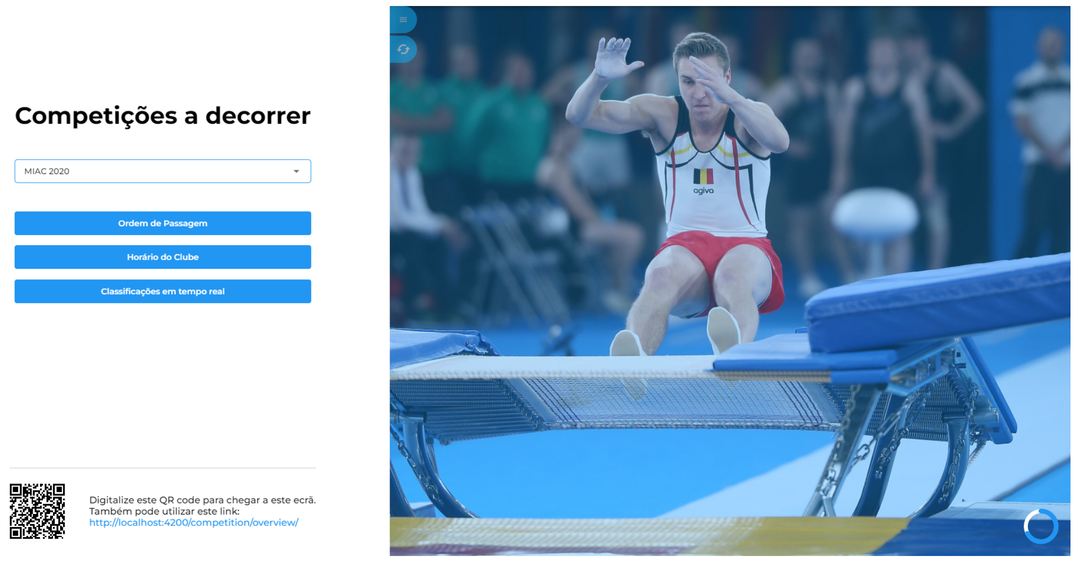
To assist with the evaluation, I'm going to follow the framework presented and described in 2.4.

To clarify, the second internal evaluation, dubbed Artificial Ex Post, will help identify critical errors or bugs that should not reach production and develop end-to-end or unit tests to increase confidence in the sidebar development. Then, the in-house evaluation, known as Naturalistic Ex Post, will assist in identifying significant UI/UX design issues and discussing alterations.

During the second internal evaluation, several issues, including the competition overview sidebar, the placement of buttons across all web applications, and the consistency of the sidebar across all pages, were discussed.

The evaluation at the conclusion of the second sidebar development revealed the following issues:

As illustrated in figures 5.9 and 5.10, the need to toggle between the sidebar with navigation buttons and the content page buttons hindered the fluidity of the user experience (UX) on certain pages of the Acro Companion web application.



Digitalize este QR code para chegar a este ecrã.
Também pode utilizar este link:
http://localhost:4200/competition/overview/

Figure 5.9: Competition Page (Sidebar with page content)

After completing this evaluation, we hit a roadblock and had to continue discussing how to resolve the UX design issue that the implementation of the dynamic sidebar had caused.

5.6 Reflection

After a second evaluation, we discovered issues with the user interface and the user experience. The UX/UI problems that we believed would be resolved by implementing the dynamic custom sidebar were not addressed, and the UX/UI was actually made worse.

Putting navigation and component functions in a single location seemed consistent and organized when discussing the proposed solution. However, after testing the internally implemented solution, it was difficult to comprehend the application's state, and users often required more than double the number of clicks to reach any portion of the logged-in application.

In the default Acro solution, for instance, to access the ”ongoing competitions” livestream,

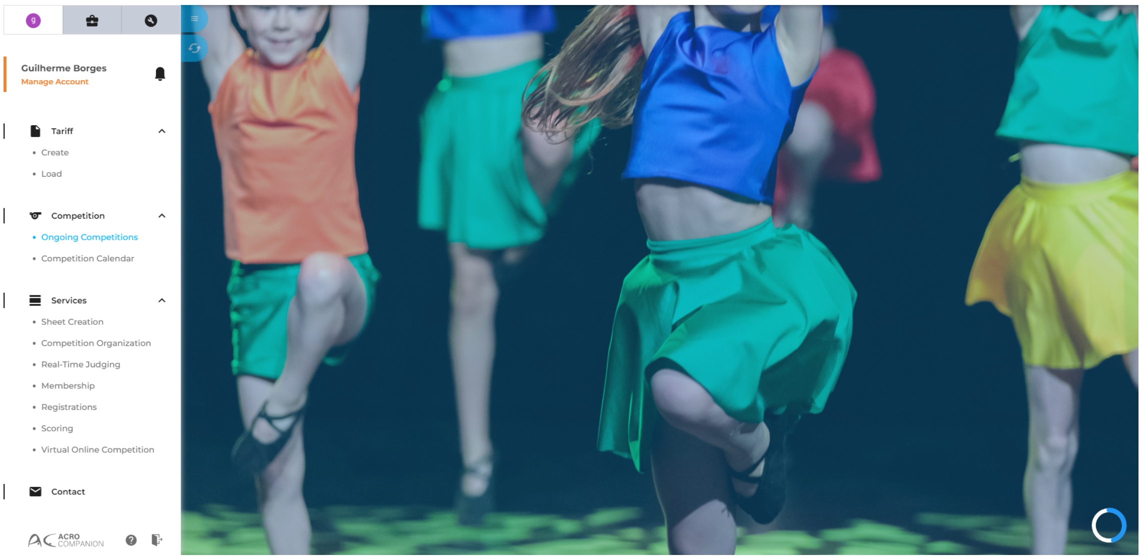


Figure 5.10: Competition Page (Sidebar with navigation buttons)

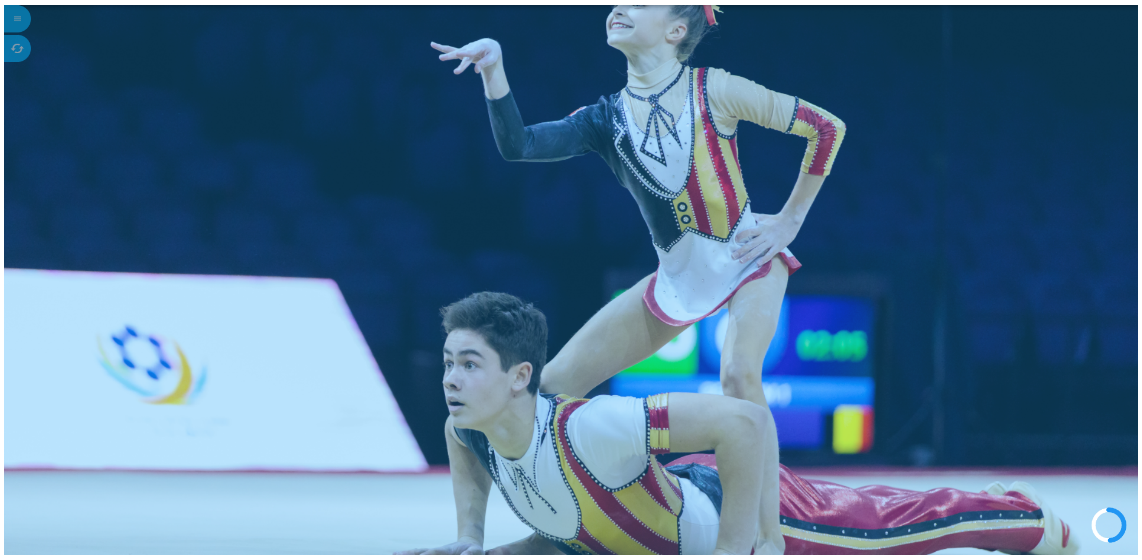


Figure 5.11: Competition Page (Sidebar closed)

it was necessary to click ”ongoing competitions” in the title bar and then click ”livestream.” To utilize the dynamic sidebar solution, it was necessary to open the sidebar, click the ”ongoing competitions” button, change the sidebar to display the ongoing competition sidebar component, and then click the ”livestream” button.

The requirement for users to toggle between navigation and content would result in an unintuitive interface, and because the sidebar’s components would be lazily loaded, the user would experience a slight delay when using it.

Chapter 6

Content sidebar (Third iteration)

In this content sidebar iteration, I will discuss the modifications made to the Acro Companion web application to address these issues from the navigational sidebar iteration and how they were implemented.

6.1 Problem

The objective of the content sidebar iteration was to address the UX/UI issues identified in the navigational sidebar iteration by refining the design and functionality of the dynamic sidebar, with a focus on enhancing the user experience and ensuring the solution is production-ready. The objective was to identify a stable and adaptable solution that addresses the identified issues and fulfills the requirements of the Acro Companion web application.

6.2 Design/building

I proposed a new design solution for the Acro Companion web application by introducing multiple dynamic sidebar containers, similar to those used by popular web applications such as Facebook and Stack Overflow. If we examine Facebook's web application (which is a single page application), for instance, we can see that the application is divided into three main containers:

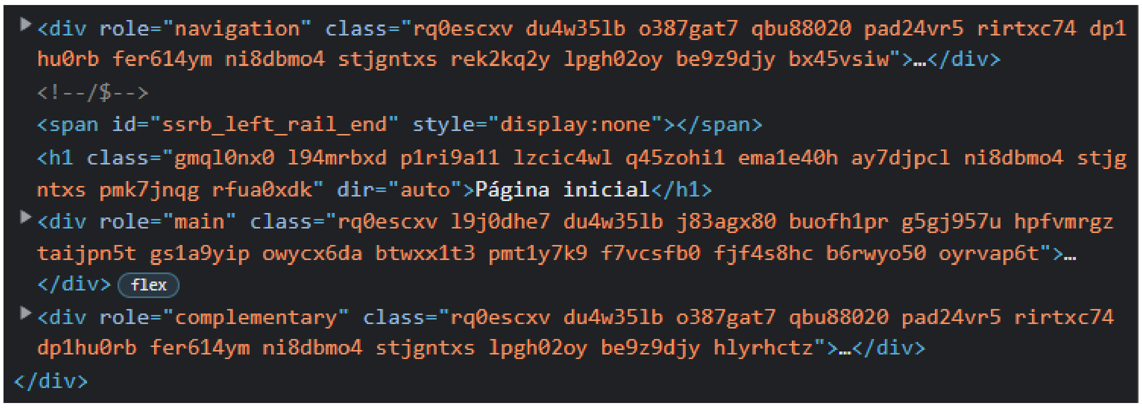
1. Navigation;

2. Main;

3. Complementary;

This method permits the application to display specific content in multiple sidebar container, which can enhance the user experience.

As shown in Figure 6.1 and 6.2, Facebook uses the ”navigation” container for navigation between different ”pages,” the ”main” container to display the content of the selected ”page,” and the ”complementary” container to display additional information and online Friends.



```
▶<div role="navigation" class="rq0escxv du4w351b o387gat7 qbu88020 pad24vr5 rirtxc74 dp1
 hu0rb fer614ym ni8dbmo4 stjgntxs rek2kq2y lpgh02oy be9z9djy bx45vsiw">…</div>
 <!--/$-->
 <span id="ssrb_left_rail_end" style="display:none"></span>
 <h1 class="gmql0nx0 l94mrbxd p1ri9a11 lzcic4wl q45zohi1 ema1e40h ay7djpcl ni8dbmo4 stjg
 ntxs pmk7jnqg rfua0xdk" dir="auto">Página inicial</h1>
▶<div role="main" class="rq0escxv l9j0dhe7 du4w351b j83agx80 buofh1pr g5gj957u hpfvmrgz
 taijpn5t gs1a9yip owycx6da btwxx1t3 pmt1y7k9 f7vcsfb0 fjf4s8hc b6rwyo50 oyrvap6t">…
 </div> flex
▶<div role="complementary" class="rq0escxv du4w351b o387gat7 qbu88020 pad24vr5 rirtxc74
 dp1hu0rb fer614ym ni8dbmo4 stjgntxs lpgh02oy be9z9djy hlyrhctz">…</div>
</div>
```

Figure 6.1: Facebook layout structure

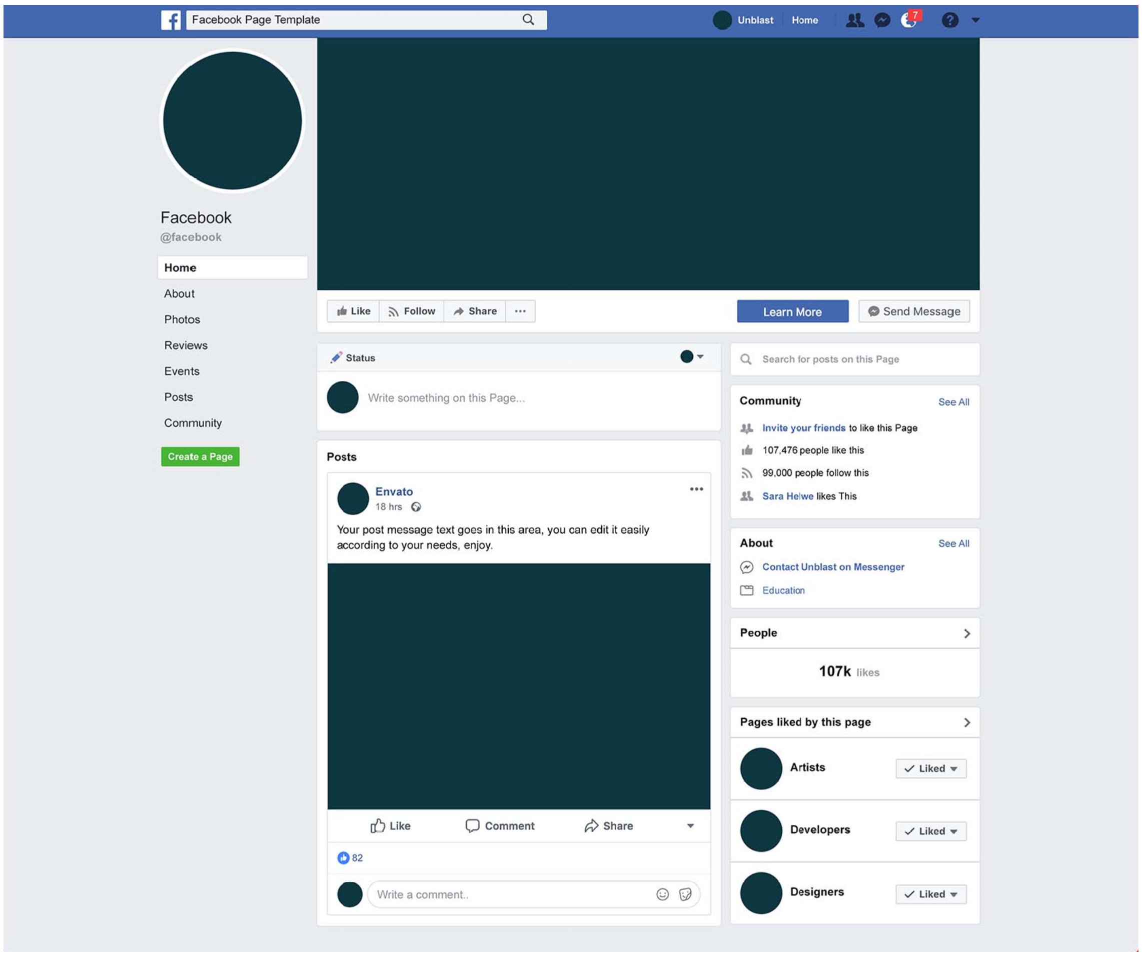
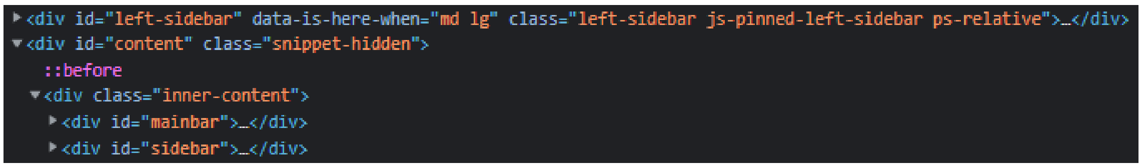


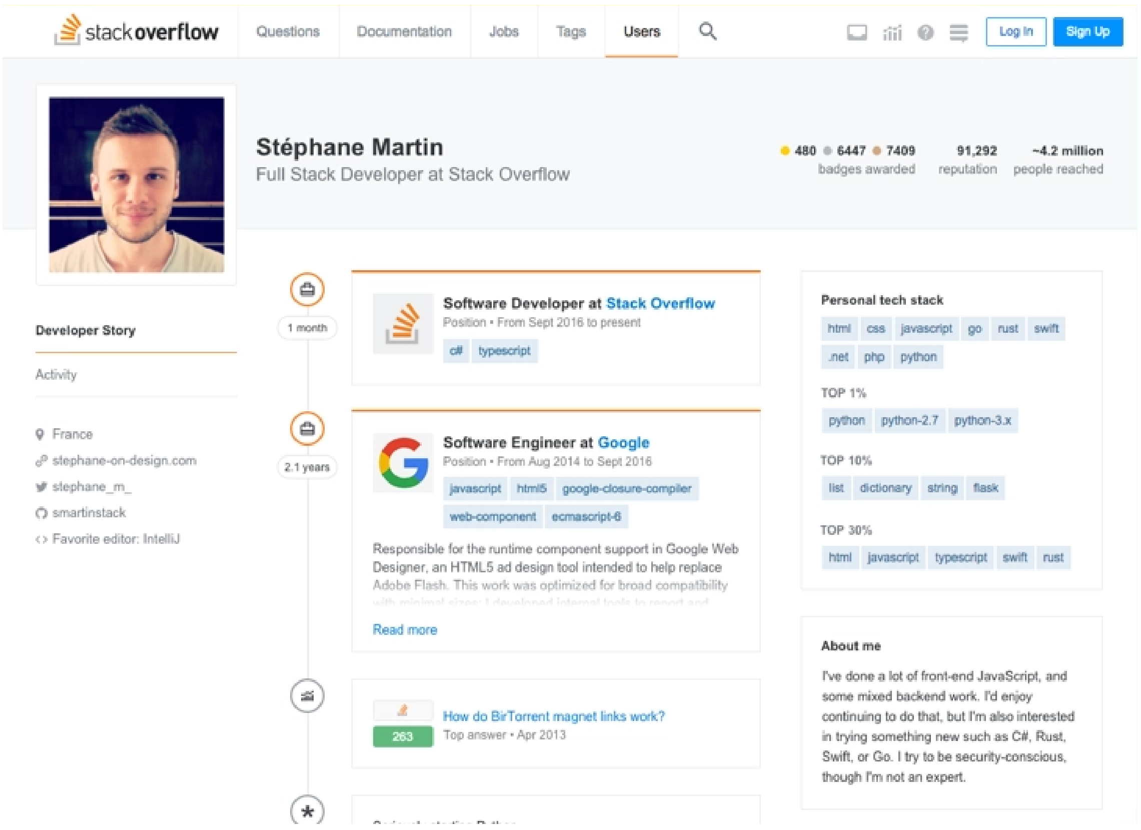
Figure 6.2: Facebook mockup

Another example of a big company that uses the dynamic container concept is Stack Overflow that as we can see in figure 6.3 and 6.4 the left dynamic container has the left-sidebar id, the center container has the mainbar id and the last one has the sidebar id.



```
▸<div id="left-sidebar" data-is-here-when="md lg" class="left-sidebar js-pinned-left-sidebar ps-relative">…</div>
▾<div id="content" class="snippet-hidden">
   ::before
  ▾<div class="inner-content">
    ▸<div id="mainbar">…</div>
    ▸<div id="sidebar">…</div>
```

Figure 6.3: Stack Overflow layout structure



Stéphane Martin
Full Stack Developer at Stack Overflow

● 480 ● 6447 ● 7409 badges awarded | 91,292 reputation | ~4.2 million people reached

Developer Story
Activity

France
stephane-on-design.com
stephane_m_
smartinstack
Favorite editor: IntelliJ

1 month
Software Developer at Stack Overflow
Position • From Sept 2016 to present
c# typescript

2.1 years
Software Engineer at Google
Position • From Aug 2014 to Sept 2016
javascript html5 google-closure-compiler web-component ecmascript-6
Responsible for the runtime component support in Google Web Designer, an HTML5 ad design tool intended to help replace Adobe Flash. This work was optimized for broad compatibility
Read more

263
How do BitTorrent magnet links work?
Top answer • Apr 2013

Personal tech stack
html css javascript go rust swift .net php python
TOP 1%
python python-2.7 python-3.x
TOP 10%
list dictionary string flask
TOP 30%
html javascript typescript swift rust

About me
I've done a lot of front-end JavaScript, and some mixed backend work. I'd enjoy continuing to do that, but I'm also interested in trying something new such as C#, Rust, Swift, or Go. I try to be security-conscious, though I'm not an expert.

Figure 6.4: Stack Overflow mockup

This design solution featured two dynamic sidebars, one for navigation and one for content management, allowing for greater flexibility and user customization, which ultimately led to an enhanced user experience. This new design solution was implemented by including a third container for a second dynamic sidebar, allowing for a more uniform user interface and more layout customization options.

Double Dynamic Sidebar Solution: This solution is similar to the one described in 4.2, but slightly more complicated:

This solution includes two dynamic sidebars, one on the left labeled ”Navigation Sidebar” and one on the right labeled ”Content Navigation sidebar.” On the left sidebar, we will only display content/buttons used for navigation between pages, and on the right sidebar, we will only display buttons to manage the content of the main displayed component and other components, as shown in figure 6.5.

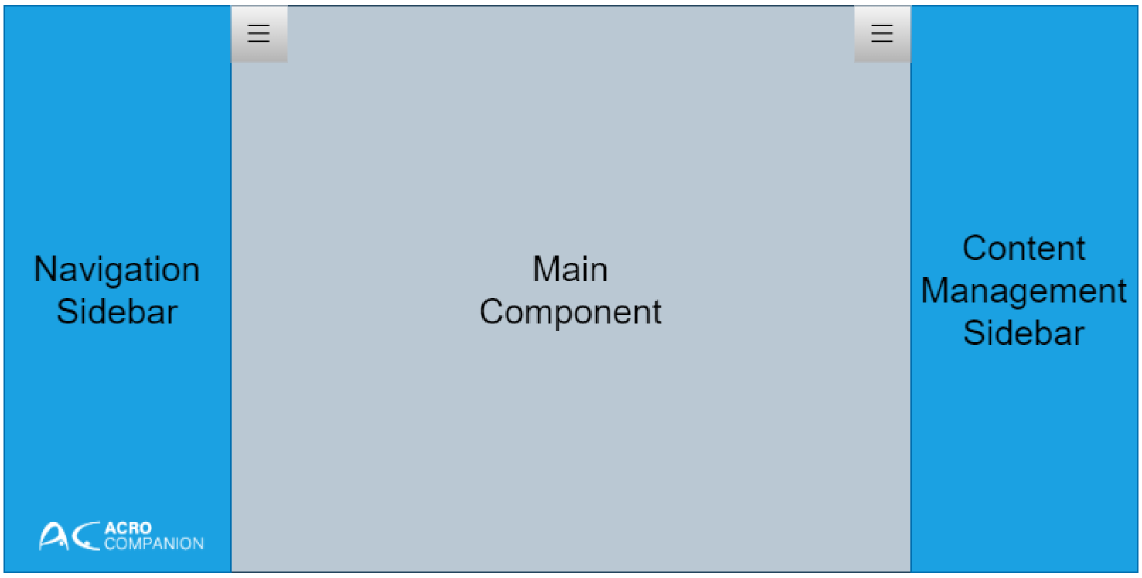


Figure 6.5: Competition Navigation sidebar

This design would provide the user with a greater degree of flexibility when using the web application. They would have several options for tailoring the layout of the interface to their needs and preferences. They could, for instance, leave the left sidebar open while keeping the right sidebar closed, or they could close both sidebars in order to have more space to interact with the primary component. Alternately, they could close the left sidebar and only use the primary component and the right sidebar. This level of customization would enhance the overall user experience by enabling the user to tailor the interface to his or her specific requirements.

6.3 Intervention

In this content sidebar iteration the objective was to finalize all the sidebar remarks from the second evaluation and make all the testing more structured and covering a higher coverage of the project.

The development of the navigation sidebar necessitated a new user interface for the main menu, developer sidebar, and organization sidebar. In addition, it was necessary to restructure and implement existing components used in multiple locations to display different types of buttons. The implemented solution was assigned to the previous organization component 5.7 and had to be relocated to the content management sidebar. This solution had to be implemented within the sidebar component of the content management system, but it had to be activated in a different manner. Each component that required these buttons was required to create them internally. Acro has some pages that only require the content management sidebar, others that only require the buttons, and occasionally both, so it was illogical to always display both, thereby increasing resource consumption. With the implementation of the sidebar, the web application's structure and implementation logic had to be modified to ensure that functionality was not broken. It was necessary to fix all existing tests and add additional ones.

There were three new types of tests added:

- Unit tests
- End-to-End tests
- Integration tests

The business analyst proposed the necessity to add a way to evaluate all the observables added to the organization, as well as all the possible triggers for those observables, as part of the Unit tests. For the End-to-end tests, navigation tests were added to ensure that the sidebar would display the correct navigation sidebar with the correct option and that all sidebar buttons navigated to the correct location. Was added to ensure that all logic in the organization worked and for scoring and evaluating. The integration tests were added to identify existing database request errors.

6.4 Practical implementation

Due to the addition of a second sidebar in this iteration, the file structure and file names for sidebars had to be modified to maintain a readable and understandable structure. As shown in figure 6.6, we have two primary directories, one for the navigation sidebar content and the other for the content management sidebar content.

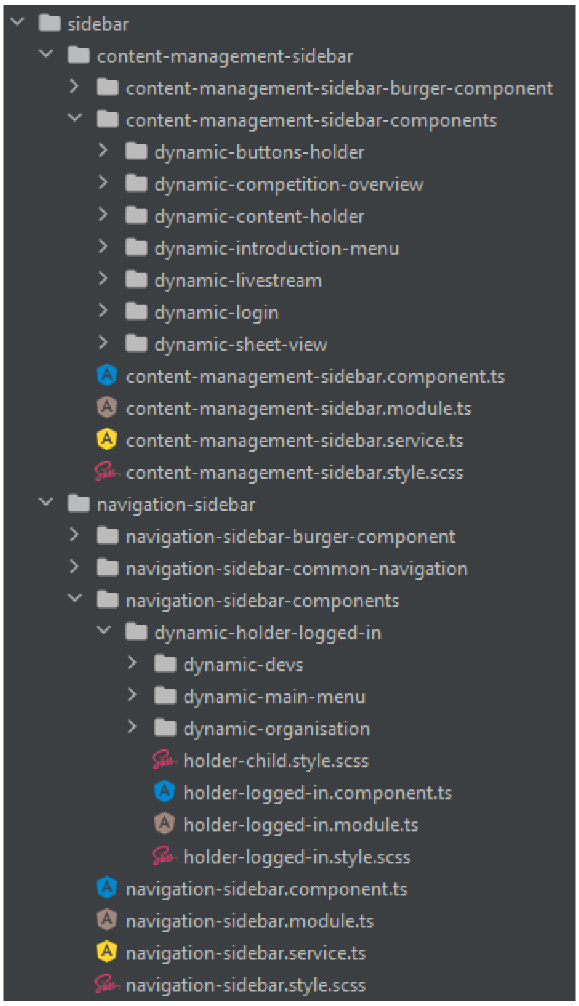
The remainder of the structure is virtually identical; each sidebar has a button to open or close it, as well as a directory containing all the components that can be hosted within it.

In this content sidebar iteration, the sidebar structure of the Navigation Sidebar was streamlined, and performance issues were assessed.

Left Sidebar (Navigation Sidebar):

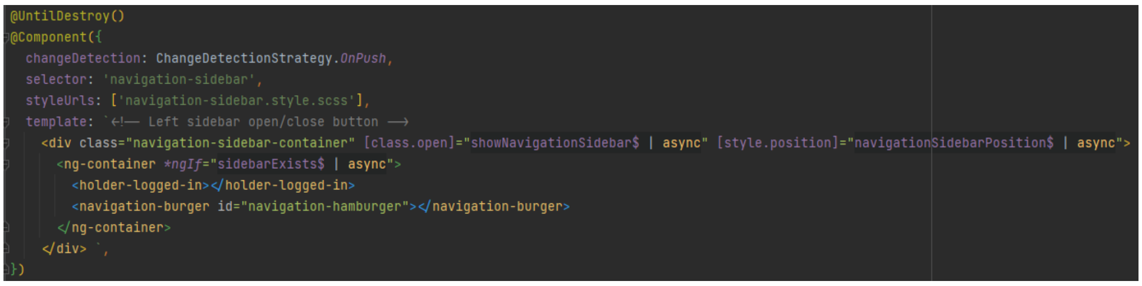
The sidebar template, as depicted in Figure 6.7, consists of the "holder-logged-in" element, which is the navigation structure sidebar, and the "navigation burger" element, which is the open/close sidebar button.

These two elements are surrounded by a "ng-container" in order to utilize structural directives without the need for an additional element, ensuring that the only DOM modifications made are those dictated by the directives themselves. This solution improves performance as a result of the browser rendering fewer elements. In order to dynamically trigger changes in the sidebar's position, state, and animations, the properties "class.open" and "style.position" are overridden on the sidebar styles using observable. The "ngif" property is used to hide the sidebar in a specific component. This is a necessary feature because Acro Companion has pages that are independent from the main user web application; in some cases, these pages are used to display competition information on large screens.



```
sidebar
  content-management-sidebar
    content-management-sidebar-burger-component
    content-management-sidebar-components
      dynamic-buttons-holder
      dynamic-competition-overview
      dynamic-content-holder
      dynamic-introduction-menu
      dynamic-livestream
      dynamic-login
      dynamic-sheet-view
    content-management-sidebar.component.ts
    content-management-sidebar.module.ts
    content-management-sidebar.service.ts
    content-management-sidebar.style.scss
  navigation-sidebar
    navigation-sidebar-burger-component
    navigation-sidebar-common-navigation
    navigation-sidebar-components
      dynamic-holder-logged-in
        dynamic-devs
        dynamic-main-menu
        dynamic-organisation
        holder-child.style.scss
        holder-logged-in.component.ts
        holder-logged-in.module.ts
        holder-logged-in.style.scss
    navigation-sidebar.component.ts
    navigation-sidebar.module.ts
    navigation-sidebar.service.ts
    navigation-sidebar.style.scss
```

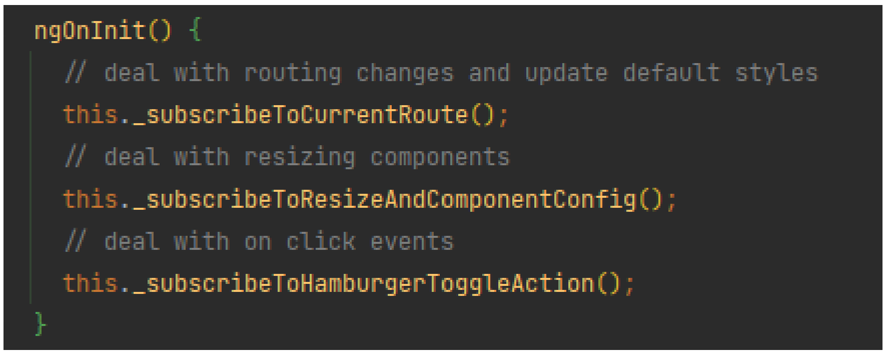
Figure 6.6: File structure (content sidebar iteration)



```
@UntilDestroy()
@Component({
  changeDetection: ChangeDetectionStrategy.OnPush,
  selector: 'navigation-sidebar',
  styleUrls: ['navigation-sidebar.style.scss'],
  template: `<!-- Left sidebar open/close button -->
    <div class="navigation-sidebar-container" [class.open]="showNavigationSidebar$ | async" [style.position]="navigationSidebarPosition$ | async">
      <ng-container *ngIf="sidebarExists$ | async">
        <holder-logged-in></holder-logged-in>
        <navigation-burger id="navigation-hamburger"></navigation-burger>
      </ng-container>
    </div> `,
})
```

Figure 6.7: Navigation Sidebar template

To increase readability, multiple functions were added to split the complexity, thereby dividing it. The primary functionality of the sidebar is defined by three main functions: as depicted in figure 6.8 the function **"_subscribeToCurrentRoute()"** manages all the url event change and retrieves the correct component definition to display. The **"_subscribeToResizeAndComponentConfig"** updates the sidebar styles and resize state because the sidebar has different behaviours depending on the window width. Finally the function **"_subscribeToHamburgerToggleAction"** handles the open/close sidebar state logic because the open/close hamburger is a dumb component.



```
ngOnInit() {
  // deal with routing changes and update default styles
  this._subscribeToCurrentRoute();
  // deal with resizing components
  this._subscribeToResizeAndComponentConfig();
  // deal with on click events
  this._subscribeToHamburgerToggleAction();
}
```

Figure 6.8: Navigation Sidebar ngOnInit

The **"_highlightSidebarMenuItem()"** function is used to highlight the currently selected button in the navigation sidebar. This function is triggered whenever the url changes. The id is inserted into a stream of sidebar button ids, and the current value is utilized within the template.

The figure 6.9 displays all the sidebar component's functions. Every input and output of a function is strongly typed, which is one of the advantages of the Typescript framework, reducing the likelihood of a function call with incorrect arguments.

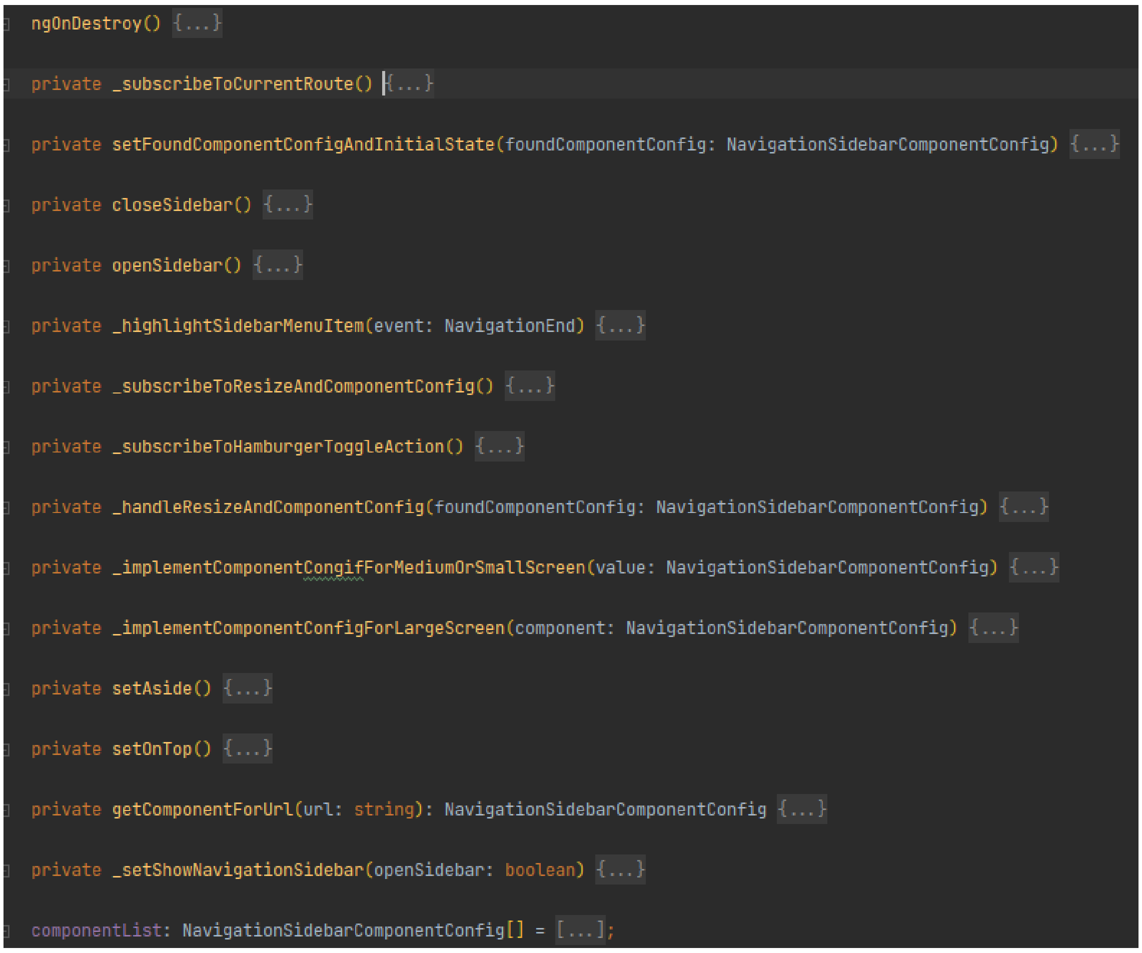
The Navigation Sidebar Component has an array of "NavigationSidebarComponentConfig" objects, with each object's dynamic component specification being defined by:

1. **label** - That defines the name of the component
2. **routerStarts** - That defines when that component will be shown
3. **closedByDefault** - That defines if the sidebar is closed by default
4. **sidebarAndContentStructure** - That will define the structure of the sidebar

A new Sidebar component (content-management-sidebar) and sidebar service were developed (content-management-sidebar-service).

Right Sidebar (Content Management Sidebar):

The sidebar template, as depicted in Figure 6.10, consists of the "content-management-burger," which is the open/close sidebar button, and the "dynamic-content-holder," which is the content management sidebar.

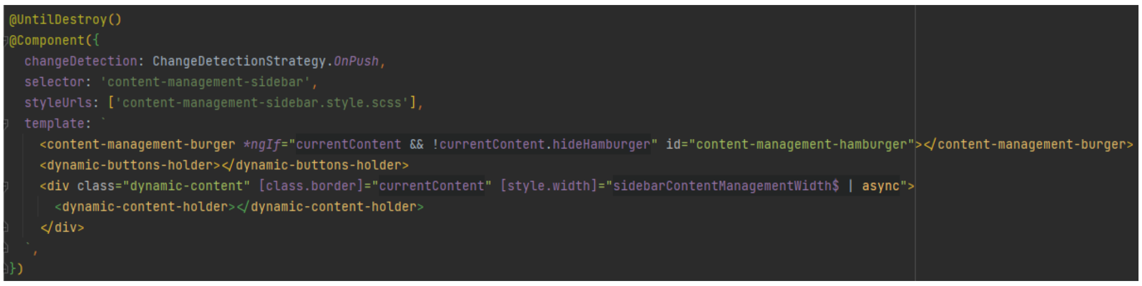


```
ngOnDestroy() {...}

private _subscribeToCurrentRoute() {...}

private setFoundComponentConfigAndInitialState(foundComponentConfig: NavigationSidebarComponentConfig) {...}

private closeSidebar() {...}

private openSidebar() {...}

private _highlightSidebarMenuItem(event: NavigationEnd) {...}

private _subscribeToResizeAndComponentConfig() {...}

private _subscribeToHamburgerToggleAction() {...}

private _handleResizeAndComponentConfig(foundComponentConfig: NavigationSidebarComponentConfig) {...}

private _implementComponentCongifForMediumOrSmallScreen(value: NavigationSidebarComponentConfig) {...}

private _implementComponentConfigForLargeScreen(component: NavigationSidebarComponentConfig) {...}

private setAside() {...}

private setOnTop() {...}

private getComponentForUrl(url: string): NavigationSidebarComponentConfig {...}

private _setShowNavigationSidebar(openSidebar: boolean) {...}

componentList: NavigationSidebarComponentConfig[] = [...];
```

Figure 6.9: Navigation Sidebar functions

The ”dynamic-buttons-holder” presented in the content management sidebar is used to assign a collection of small buttons. This functionality operates differently than the primary content management sidebar; the trigger for this element is a component rather than a change in route.

The sidebar styles’ ”class.border” and ”style.width” properties are overridden with observable to trigger changes in the sidebar’s width, state, and animations. The ”ngif” property is used to hide the sidebar hamburger. This is a necessary feature because Acro Companion has pages that only require the ”dynamic-buttons-holder” and must be always visible.

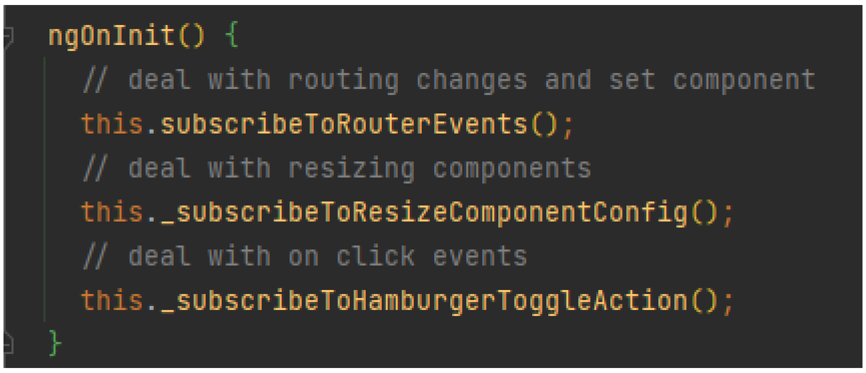


```
@UntilDestroy()
@Component({
  changeDetection: ChangeDetectionStrategy.OnPush,
  selector: 'content-management-sidebar',
  styleUrls: ['content-management-sidebar.style.scss'],
  template: `
    <content-management-burger *ngIf="currentContent && !currentContent.hideHamburger" id="content-management-hamburger"></content-management-burger>
    <dynamic-buttons-holder></dynamic-buttons-holder>
    <div class="dynamic-content" [class.border]="currentContent" [style.width]="sidebarContentManagementWidth$ | async">
      <dynamic-content-holder></dynamic-content-holder>
    </div>
  `,
})
```

Figure 6.10: Content Management Sidebar template

As illustrated in Figure 6.12, multiple functions were added to split the complexity and improve readability. The content sidebar main functionality is defined by the tree main function, as shown in 6.11. The function **”_subscribeToRouterEvents()”** handles all url event changes and displays the correct component definition.

The **”_subscribeToResizeComponentConfig”** updates the sidebar’s styles and resize state, as the sidebar’s behavior varies based on the current component. Lastly, the function **”_subscribeToHamburgerToggleA**
ages the open/close sidebar state logic because the open/close hamburger is a dumb component.



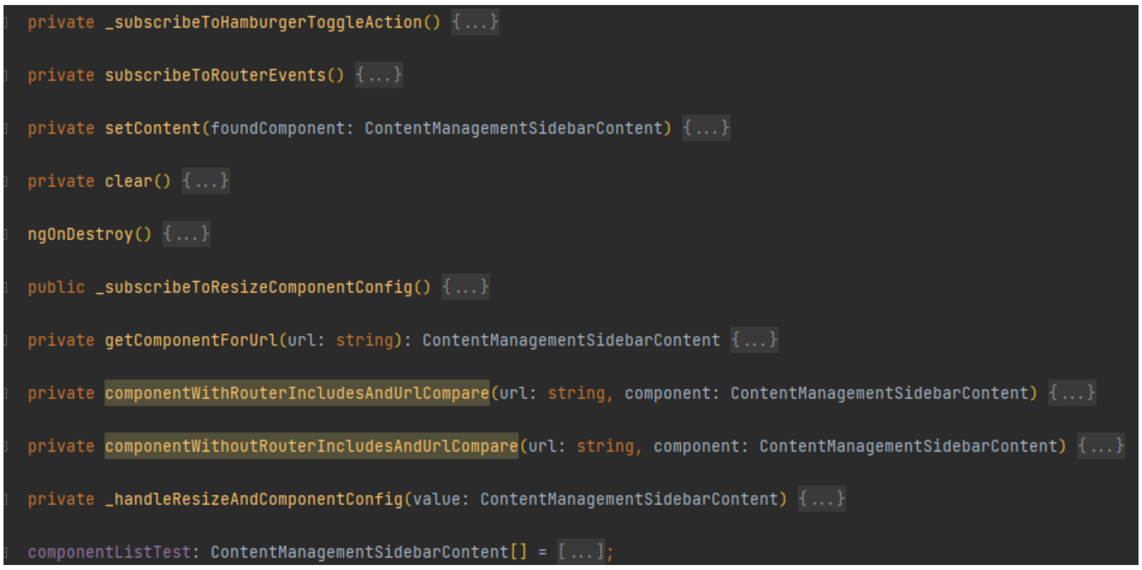
```
ngOnInit() {
  // deal with routing changes and set component
  this.subscribeToRouterEvents();
  // deal with resizing components
  this._subscribeToResizeComponentConfig();
  // deal with on click events
  this._subscribeToHamburgerToggleAction();
}
```

Figure 6.11: Content Management Sidebar ngOnInit

The Navigation Sidebar Component has an Array of ”ContentManagementSidebarContent” objects, each of which is defined by:

1. **label** - That defines the name of the component

2. **routerStarts** - That defines when that component will be shown



```
private _subscribeToHamburgerToggleAction() {...}

private subscribeToRouterEvents() {...}

private setContent(foundComponent: ContentManagementSidebarContent) {...}

private clear() {...}

ngOnDestroy() {...}

public _subscribeToResizeComponentConfig() {...}

private getComponentForUrl(url: string): ContentManagementSidebarContent {...}

private componentWithRouterIncludesAndUrlCompare(url: string, component: ContentManagementSidebarContent) {...}

private componentWithoutRouterIncludesAndUrlCompare(url: string, component: ContentManagementSidebarContent) {...}

private _handleResizeAndComponentConfig(value: ContentManagementSidebarContent) {...}

componentListTest: ContentManagementSidebarContent[] = [...];
```

Figure 6.12: Content Management Sidebar functions

3. **routerIncludes** - That defines when a very specific component will be shown

4. **lazyComponent** - That is the Component that will be shown

5. **contentManagementSidebarWidth** - That will specify the sidebar width

6. **hideHamburger** - That will enable the possibility to not having button

The Content Management sidebar functions identically to the Navigation sidebar, utilizing a different Array but maintaining the same concept; only the resizing logic is altered.

In some components of the content management sidebar, it was necessary to always display a group of components. This collection of components would serve as a holder for the buttons required by those components. As shown in Figure **??**, the small container contains the group of buttons and the large container contains the more complex components.

In addition to the navigation and content management sidebars' underlying logic, additional functionality was required to improve user interaction and comprehension. This included the incorporation of a background and a menu selection option for user convenience.

As mentioned previously, the navigation sidebar's structure includes a "sidebarAndContentStructure" property that can exist in two distinct states. This permits flexibility and adaptability in the sidebar's design and functionality.

1. **SideBySide** - Make that the sidebar is side by side with the main component;

2. **onTop** - Make that the sidebar is on top of the main component.

As depicted in Figure 6.13, the "SideBySide" state is utilized on pages where the user frequently needs to navigate to other pages for convenience. Alternatively, the "onTop" state, as

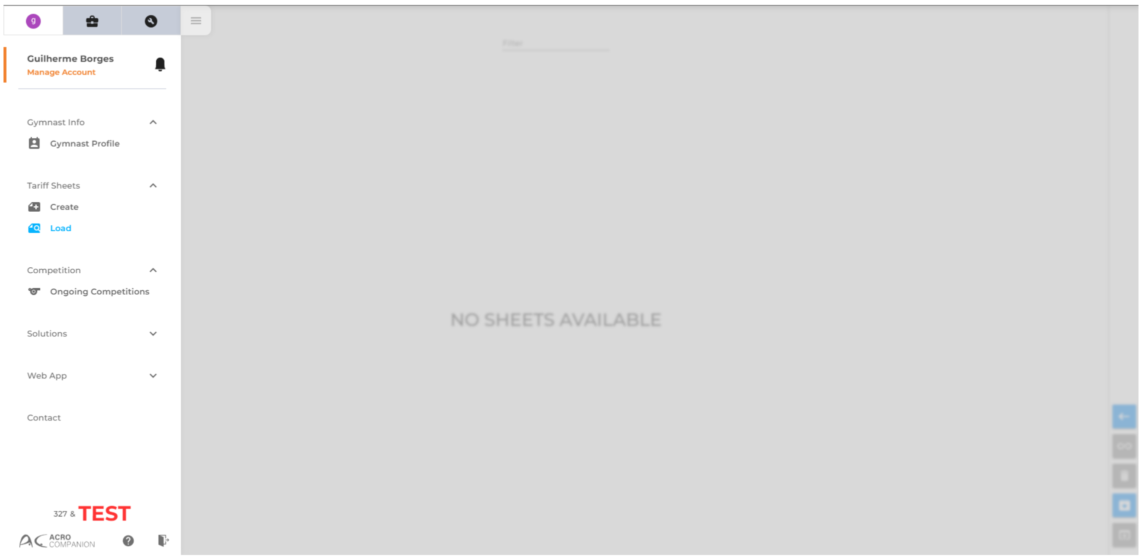


Figure 6.13: Navigation on top structure

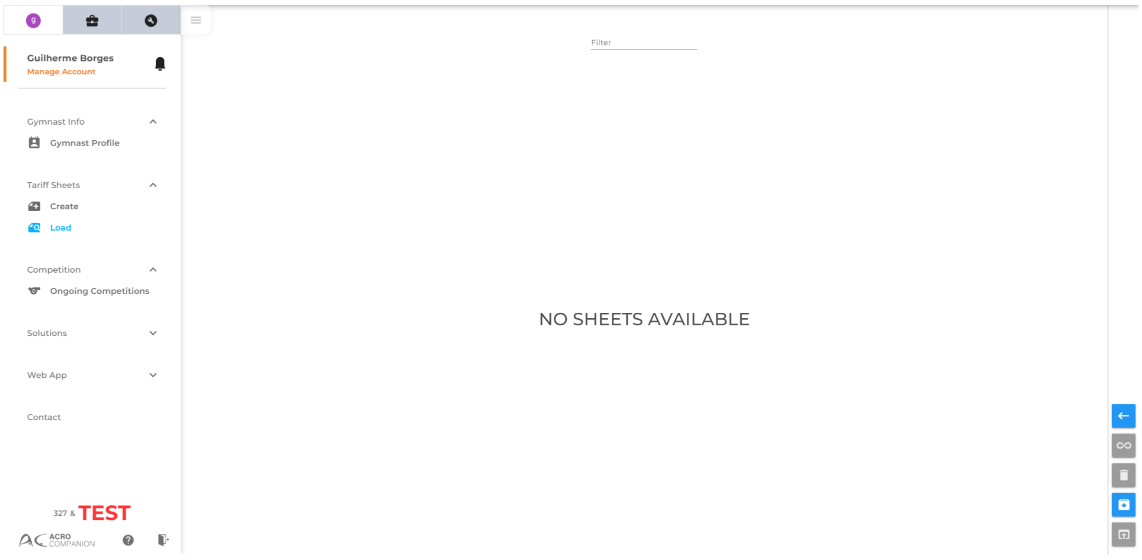
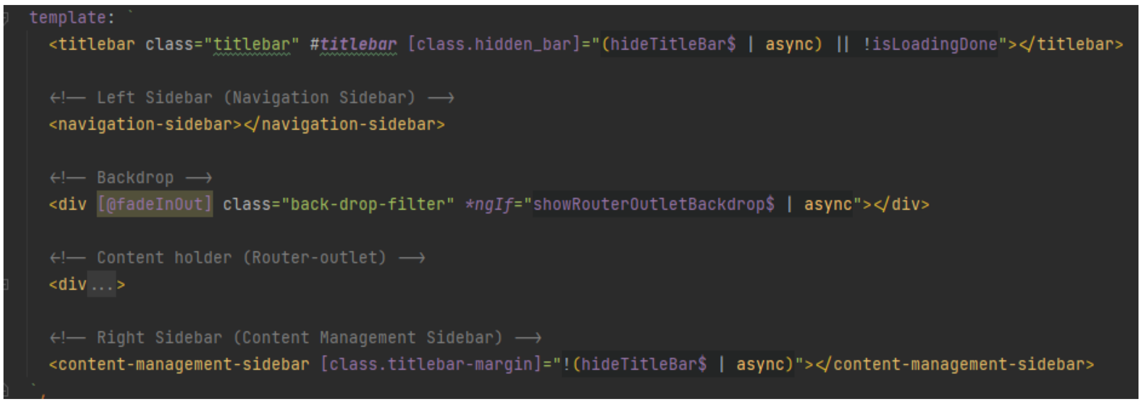


Figure 6.14: Navigation side by side structure

depicted in Figure 6.14, is utilized on pages where the user is engaged in a task and navigations would result in the loss of work.

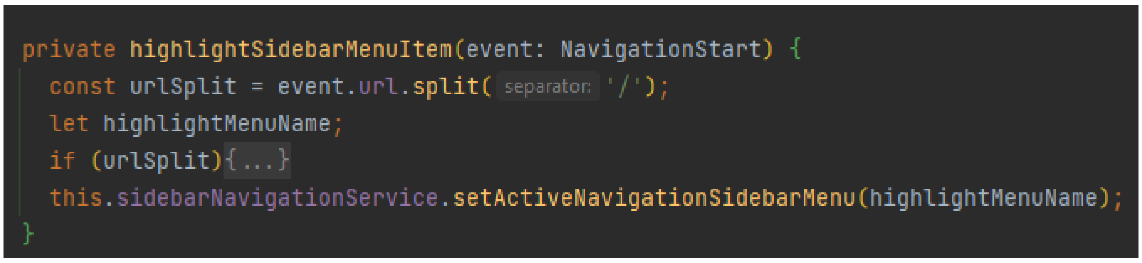
The backdrop was added to the navigation sidebar when it is displayed above the content. This ensures that the user cannot continue editing the page while the sidebar is visible. As shown in Figure 6.15, the backdrop was added to the app.component so that it encompasses both the navigation and content management sidebars. The logic for this feature was implemented by combining the sidebar's state and position observables.



```
template: `
  <titlebar class="titlebar" #titlebar [class.hidden_bar]="(hideTitleBar$ | async) || !isLoadingDone"></titlebar>

  <!-- Left Sidebar (Navigation Sidebar) -->
  <navigation-sidebar></navigation-sidebar>

  <!-- Backdrop -->
  <div [@fadeInOut] class="back-drop-filter" *ngIf="showRouterOutletBackdrop$ | async"></div>

  <!-- Content holder (Router-outlet) -->
  <div...>

  <!-- Right Sidebar (Content Management Sidebar) -->
  <content-management-sidebar [class.titlebar-margin]="!(hideTitleBar$ | async)"></content-management-sidebar>
`,
```

Figure 6.15: App component template structure

I implemented a function that knows the selected menu item regardless of URL changes. Thus, the user is always aware of which option they have chosen and can easily navigate the various menus. As depicted in Figure 6.16, this logic was implemented by monitoring URL changes and updating an observable with the appropriate menu value.



```
private highlightSidebarMenuItem(event: NavigationStart) {
  const urlSplit = event.url.split( separator: '/');
  let highlightMenuName;
  if (urlSplit){...}
  this.sidebarNavigationService.setActiveNavigationSidebarMenu(highlightMenuName);
}
```

Figure 6.16: Highlight sidebar menu items

This value then was used to highlight the menu item, as illustrated in figure 6.17, I implemented as well a hover and a selected hover demonstrated respectively in figure 6.18 and 6.19 styles to improve the user experience.

Figure 6.17: Menu selected

Figure 6.18: Menu hovered

Figure 6.19: Menu selected and hovered

The addition of the backdrop element, and menu selection element all contribute to an enhanced user experience by facilitating the user's interaction with and comprehension of the sidebars. These features were implemented to improve the functionality and usability of the web application and to offer a more streamlined and effective user experience.

6.5 Evaluation

Using the methodology previously described in 2.4, the team discovered some UX issues with the production-ready final solution during the third internal evaluation (Naturalistic Ex Post). The team discovered, for instance, that the logic for the left sidebar was occasionally flawed and inefficient. In addition, the list of organisations that organisation-admin hosts in the sidebar would load every time the page was refreshed, even if the user was not on that page. Additionally, some animation overlays did not function when deep linking to specific pages. This was not the purpose of the iteration, so it was not necessary to address these issues before the solution could be deployed.

For the internal evaluation (Artificial Ex Post), I was able to identify and resolve a few bugs and issues, but this was insufficient because the Web application coverage was insufficient.

Only at the end of the content sidebar iteration was it possible to deploy the solution and collect user feedback (Naturalistic Ex Post). However, most of the feedback was not related to bugs, but rather suggestions for additional enhancements.

6.6 Reflection

Looking back on the development process, it is evident that adding a second sidebar was a major undertaking. Even if the core implementation was complete, it was still essential to rebuild the configuration object, create s for communication and reorganize the project structure in order to reuse a portion of the code used to develop the initial sidebar. Initially, it was feared that this new feature would complicate the user interface, which could have a negative impact on the user experience. However, after careful consideration and testing, it became clear that this approach was required to provide the application with the required level of flexibility. Finding the optimal balance between providing a user-friendly interface and preserving the flexibility of the application was one of the greatest challenges during the development process. To achieve the desired result, a significant amount of testing and design iteration were required. Overall, the incorporation of a second sidebar was a difficult but necessary development step. It enabled a more dynamic and adaptable user interface, which enhanced the user experience overall. The implementation of tests and evaluations contributed to the final solution being both functional and user-friendly. This experience highlighted the significance of careful planning and testing in achieving the desired result.

Chapter 7

Formalization of learning

In the concluding chapter, I will reflect on the lessons learned during the nine-month process of developing dynamic sidebars. I will discuss the key takeaways and insights gained from this experience, including the obstacles encountered and the solutions implemented to overcome them, from the initial concept to the final implementation. In addition, I will investigate the effect of dynamic sidebars on the user experience and the overall efficacy of the proposed solution in enhancing the functionality and usability of the web application. This chapter will provide a comprehensive overview of the development process and the most important lessons learned.

7.1 Dynamic sidebars

The solution design began with a simple concept of hosting components within a dynamic sidebar. The dynamic sidebar would work with affordances with the purpose of adapting to the user context and browser context. This initial iteration served as a proof-of-concept and demonstrated that the concept to enhance the Acro Companion web application was feasible as illustrated in figure 7.1.

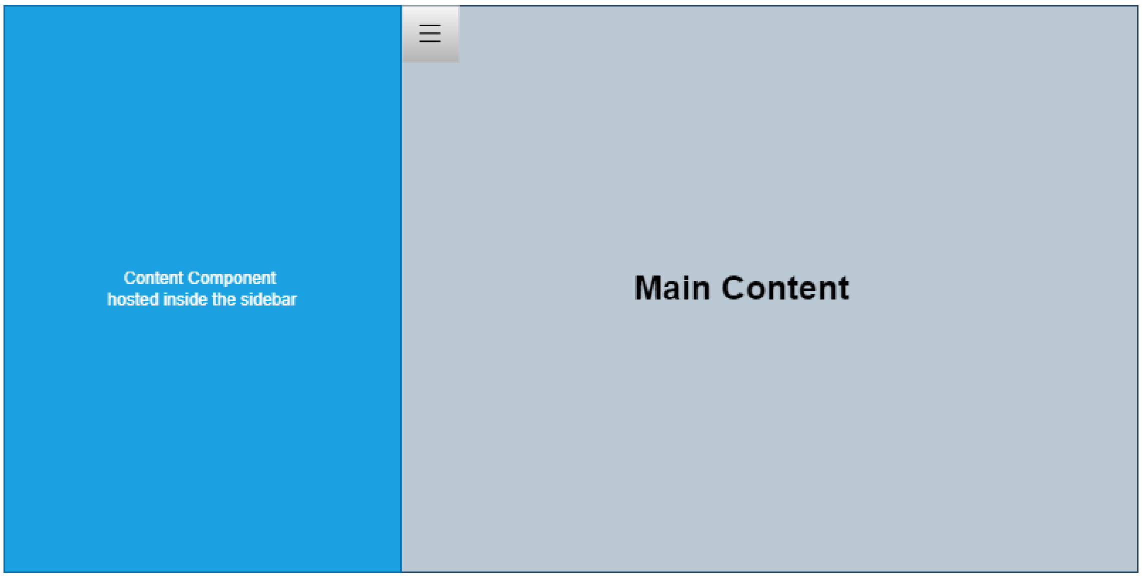


Figure 7.1: First sidebar mockup

In order to completely remove the title bar, it became necessary to implement a navigation sidebar as the project progressed. It was necessary to implement a switch between the sidebar component and the navigation sidebar in order to accomplish this solution. However, this implementation negatively affected the user experience, so a new sidebar design was implemented as illustrated in figure 7.2.

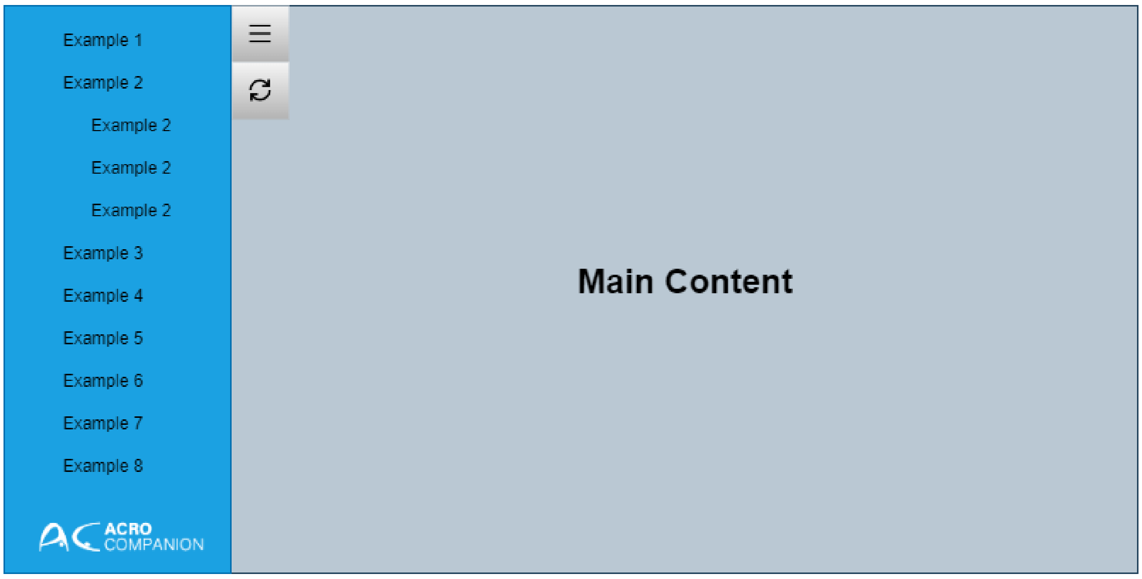


Figure 7.2: Second sidebar mockup

Instead of combining the navigation sidebar and content management sidebar in the same space, the new design separated them into two separate sidebars. In addition, a second dynamic container was added to the Content Management sidebar in order to hold a group of buttons that certain components required. The final solution included two sidebar options: one for navigation and another for content management. The sidebar for content management also featured a second dynamic sidebar as illustrated in figure 7.3.

In conclusion, the dynamic sidebar is a dynamic element that can be modified using affordances, allowing for greater flexibility in terms of the content that can be displayed and providing a more uniform user interface. The implementation of the dynamic sidebar was a journey of trial and error in which various solutions were tested and evaluated, leading to the final solution that met the application's requirements and offered a seamless user experience.

7.2 Impact on user experience

The addition of a second sidebar enabled greater content display flexibility, resulting in a more uniform and understandable user interface. Users were able to customize the interface layout based on their needs and preferences, which enhanced the overall user experience by allowing them to tailor the interface to their specific requirements.

In addition, the development of the dynamic sidebars enhanced the performance of the web

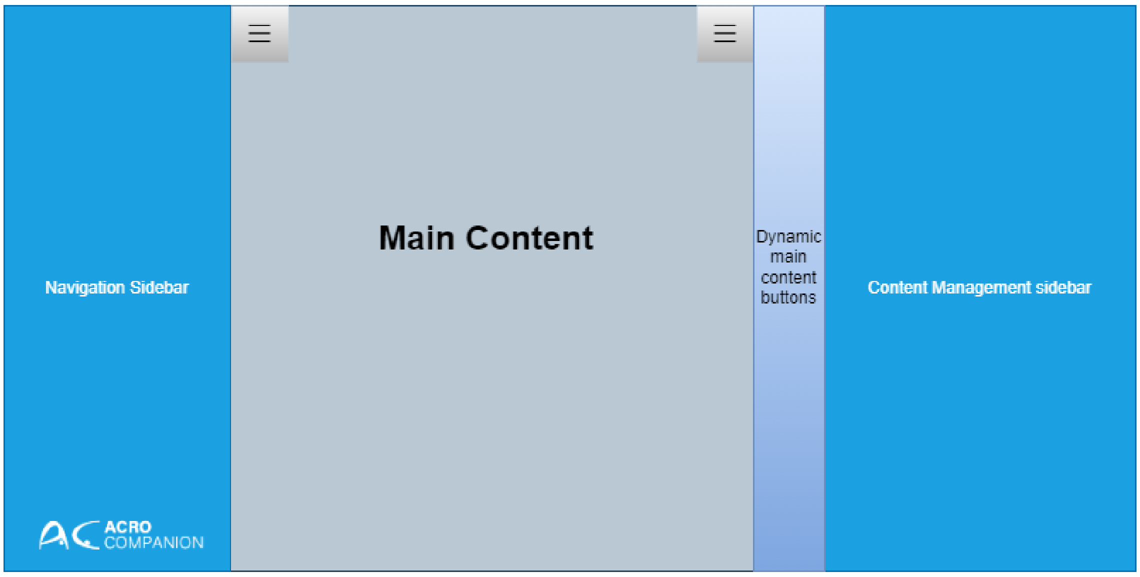


Figure 7.3: Third sidebar mockup

application by minimizing the number of DOM changes and implementing efficient logic for the sidebars. This resulted in a quicker and more responsive web application, which enhanced the user experience even further.

Overall, the development of the dynamic sidebars had a significant impact on the user experience of the Acro web application. It not only improved the overall usability and performance of the application, but it also provided users with a more personalized and intuitive interface, resulting in an ultimately more satisfying user experience.

7.3 Impact on organizational experience

The development of the sidebar was conducted entirely remotely, which presented some communication and collaboration challenges. Despite this, the team was able to effectively communicate and share information using tools including Skype, Azure DevOps, WhatsApp, and Google Docs. In the dinamic sidebar proof of concept iteration, the design team and the development team were not closely aligned, with the design team having already completed the initial design. However, this did not impede the project's progression because the development team was able to communicate any concerns or suggestions for improvement to the design team as needed. The implementation of the dynamic sidebar presented challenges in terms of coordination and collaboration, but the team was able to overcome these obstacles and successfully implement the new feature.

7.4 Future work

This chapter explores the potential for future work based on the nine-month dynamic sidebar project's developments. There are several ways in which the project could be enhanced, despite its solid foundation. Among these are:

Increasing the number of unit tests to ensure complete coverage of all functionalities. Enhancing end-to-end tests to guarantee a consistent user experience across all pages and interactions. Enhancing the user experience by enhancing the sidebar's animations and visual design. Increasing the sidebar's usability and accessibility on mobile devices by optimizing it for mobile devices. Reevaluating the placement of certain components and possibly moving them to the Navigation Sidebar in order to improve organization and usability. Improving the sidebar interactions' logic and decision-making by incorporating an AI algorithm. Each of these areas has the potential to increase the usability and efficacy of the dynamic sidebar for Acro Companion's users.

Chapter 8

Conclusions

In conclusion, Acro Companion's online application has benefited from the deployment of the dynamic sidebar solution. The dynamic sidebar has improved the structure and user experience of the web application by addressing the lack of flexibility and inconsistency in the user experience.

The development of the dynamic sidebar solution was an iterative process, including user feedback and utilizing an agile methodology to continuously enhance the design and structure. By this method, I was able to develop a system that is more resilient, scalable, and flexible, with fewer possible implementation obstacles.

The new dynamic sidebar solution is designed to be reactive, connecting with the other sidebars and the components, and allowing for accelerated development while preserving structure. This solution reflects the future of sidebars, and Acro Companion has received its implementation positively. Overall, the dynamic sidebar solution has effectively solved the company's growth challenges, and its implementation represents a step forward for the Acro Companion web application.

The feedback from Acro companion regarding the dynamic sidebars and project structure improvements was positive.

Bibliography

Angular. The modern web developer's platform. URL https://angular.io/.

Pedro Antunes and Mary Tate. Business process conceptualizations and the flexibility-support tradeoff. *Business Process Management Journal*, 28, 04 2022. doi: 10.1108/BPMJ-10-2021-0677.

Wen Chen, David J Crandall, and Norman Makoto Su. Understanding the aesthetic evolution of websites: Towards a notion of design periods. In *CHI*, pages 5976–5987, 2017.

Lisa Crispin. Driving software quality: How test-driven development impacts software quality. *IEEE Software*, 23(6):70–71, 2006. doi: 10.1109/MS.2006.157.

Azure DevOps. Plan smarter, collaborate better, and ship faster with a set of modern dev services. URL https://azure.microsoft.com/en-us/services/devops/#overview/.

Google. Firebase helps you build and run successful apps. URL https://firebase.google.com/.

Jong Seok Lee, Jan Pries-Heje, and Richard Baskerville. Theorizing in design science research. In *International conference on design science research in information systems*, pages 1–16. Springer, 2011.

Playwright. Playwright enables reliable end-to-end testing for modern web apps. URL https://playwright.dev/.

Rxjs. Reactive extensions library for javascript. URL https://rxjs.dev/.

Maung K Sein, Ola Henfridsson, Sandeep Purao, Matti Rossi, and Rikard Lindgren. Action design research. *MIS quarterly*, pages 37–56, 2011.

Karen Stendal, Devinder Thapa, and Arto Lanamäki. Analyzing the concept of affordances in information systems. In *2016 49th Hawaii international conference on system sciences (HICSS)*, pages 5270–5277. IEEE, 2016.

Nguyen Hoang Thuan, Hoang Ai-Phuong, Mathews Nkhoma, and Pedro Antunes. Using process stories to foster process flexibility: The experts' viewpoint. *Australasian Journal of Information Systems*, 26, 2022a.

Nguyen Hoang Thuan, Hoang Phuong, Mathews Nkhoma, and Pedro Antunes. Using process stories to foster process flexibility: The experts' viewpoint. *Australasian Journal of Information Systems*, 26:1–35, 02 2022b. doi: 10.3127/ajis.v26i0.3479.

John Venable, Jan Pries-Heje, and Richard Baskerville. A comprehensive framework for evaluation in design science research. In *International conference on design science research in information systems*, pages 423–438. Springer, 2012.

Adobe XD. Lifelike in every sense. create stunningly real ui/ux designs and stand out from the rest. URL `https://www.adobe.com/products/xd.html/`.

Appendix A

Appendix

A.1 Technologies

A.1.1 Angular and Typescript

On Acro Companion, Angular is utilized to build the web application's front end. Angular is an HTML and Typescript-based platform and framework for creating single-page client applications. Typescript is a client-side and server-side scripting language that is tightly typed and object-oriented. Modules, components, templates (HTML, SCSS), services, and dependency injectors are all key aspects in the design of an Angular application Angular .

A.1.2 Google Firebase (Cloud Computing)

For controlling user logins and information in Acro Companion, we use Google Firebase as a cloud computing service. We use Firebase Authentication, which offers backend services and authentication based on passwords, Google accounts, and Facebook accounts. Cloud Firestore is a database that is both versatile and scalable (MAYBE COMPARE WITH NORMAL DBs). We construct and link cloud services using Google cloud functions, which is a serverless environment that can scale as the number of users grows and is highly available and fault-tolerant Google.

A.1.3 RXJS

RxJS is a reactive programming framework that uses observables to make writing asynchronous code extremely simple. According to the official literature, this project is a reactive extension to JavaScript that improves efficiency, modularity, and debugability while remaining mainly backwards compatible, with a few breaking changes that limit the API surface. It's Angular's official library for dealing with reactivity, converting pull operations for call-backs into observables Rxjs.

A.1.4 Playwright Tests

The Playwright Test was designed specifically to meet the requirements of end-to-end testing. It accomplishes what is expected of a standard test runner, and more. The Playwright test allows us to Playwright:

1. Run tests across all browsers;
2. Tests should be run in parallel;
3. Right out of the box, you can enjoy context isolation;
4. On failure, record videos, screenshots, and other artifacts;
5. Make POMs extendable fixtures by incorporating them into your structure.

A.1.5 Adobe XD

Adobe XD is more than just a UI/UX design tool, it's a collaborative platform where designers can create magic. Move in the same direction. Gather feedback. Iterate frequently. With tools designed for simple collaboration, you can push your creations even further. The developers use Adobe XD to gather the information they need to implement XD.

A.2 Methodologies

A.2.1 Reactive Programming

Asynchronous programming logic is used to handle real-time updates to otherwise static information in reactive programming. When a user makes an inquiry, it provides an effective way to handle data modifications to content. Building a reactive system entails addressing issues such as separation of concerns, data consistency, failure management, choice of messaging implementation, and so on.

Although reactive programming can be used as an implementation approach to ensure that individual services employ an asynchronous, non-blocking model, designing the system as a whole to be a reactive system necessitates a design that addresses all of these additional considerations.

A.2.2 Design System

In Acro Companion, we use a design system. The capacity to swiftly recreate ideas using preset UI components and elements is the fundamental value of design systems.

The teams can reuse the same pieces, eliminating the requirement to reinvent and hence the danger of unintentional inconsistency.

When it's essential, and if we'll be using a component in a lot of various areas throughout the web application, we'll create a custom component for it, so we only have to customize it once and can use it in several places.

A.2.3 Unit Testing and e2e Testing (End-to-End)

Unit testing is a sort of software testing that examines individual units or functions. Its primary purpose is to thoroughly test each unit or function. A unit is the smallest testable part of an application that can be tested. It mainly has one or a few inputs and produces only one output.

End-to-end testing is a software development lifecycle (SDLC) methodology for evaluating an application's functionality and performance under product-like conditions and data to simulate real-world scenarios. The purpose is to recreate a real-world user scenario from beginning to end.

A.2.4 Test-driven Development (TDD)

"Test-driven" or "test-first" development isn't truly a testing technique, despite its name. TDD, often known as test-driven design, operates as follows: The programmers build unit tests for each small piece of functionality they code. After that, they write the code that allows the unit tests to pass. This forces the programmer to consider a variety of factors before developing the functionality Crispin [2006].