INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Como é possível melhorar o desempenho da utilização de listas encadeadas em React Native?

DIOGO ANDRÉ PINTO BATISTA
Outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Como é possível melhorar o desempenho da utilização de listas encadeadas em React Native?

Diogo Batista

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Software Engineering

Orientador: Dr. Paulo Baltarejo

Porto, 15 de outubro de 2022

Dedicatória

Se eu fui capaz de chegar até ao mestrado foi graças aos meus pais que sempre me incentivaram a me dedicar aos estudos, sacrificando-se por vezes, em meu favor. Vocês sempre foram minha base na vida, no estudo e a minha verdadeira inspiração. São o exemplo de que se pode chegar a qualquer lado, bastando ter paixão e dedicação pelo objetivo que se quer cumprir. Chegar onde cheguei é mérito meu, mas sobretudo conseguido por ter sido sustentado por serem os dois pilares, que nunca me deixaram vergar.

Com o decorrer da vida académica e pessoal, uma nova pessoa surgiu, que foi o culminar de apoio e motivação para a conclusão deste metrado e deste ciclo de estudos. À minha namorada e companheira de todos os dias, deixo o meu mais sincero obrigado, por me aturares, motivares e ajudares em tudo o que foi (e é) preciso. Por fim, um palavra de apreço pelo orientador Paulo Baltarejo, por transmitir toda a sua sabedoria, e também por toda a sua disponibilidade para o acompanhamento deste estudo.

A vocês, o meu muito obrigado! Dedico-vos esta dissertação e toda a minha vida académica!

Resumo

A Blip, empresa do ramo de desenvolvimento de software, iniciou o desenvolvimento de um projeto multiplataforma para a Betfair. Este projeto surge numa altura de larga expansão do mercado de apostas desportivas, de que a Betfair tenciona fazer parte - promovendo a criação de uma nova aplicação para dispositivos móveis, com um visual moderno, atrativo e de utilização simplificada. Posto isto, a Blip decidiu utilizar a tecnologia React Native, tendo em conta a possibilidade de partilhar parte do código, (sobretudo a lógica de negócio) de forma a desenvolver esta aplicação compatível com navegadores *web* e dispositivos iOS e Android. Contudo, esta tecnologia é bastante recente, existindo alguns problemas de desempenho detetados - em especial na utilização de listas encadeadas, com uma elevada mutabilidade de dados.

Este trabalho tem como objetivo primordial melhorar o desempenho mencionado anteriormente, através da utilização de uma ferramenta capaz de substituir a atual FlatList. Para tal, procedeu-se a uma criteriosa investigação sobre as novas funcionalidades da tecnologia, como o *JSI Modules*. Esta nova ferramenta é desenvolvida em linguagem nativa - podendo constituir uma forma eficaz de aumentar o desempenho de uma aplicação móvel.

De forma a sustentar toda a investigação e desenvolvimento da ferramenta, são mencionados e analisados diversos tipos de aplicações móveis, sendo feita uma abordagem à arquitetura que constitui as plataformas iOS e Android. Também se realiza uma abordagem das tecnologias utilizadas para o seu desenvolvimento, tal como uma análise entre a tecnologia React Native e algumas tecnologias existentes para o desenvolvimento deste tipo de aplicações, destacando-se o Flutter, o Kotlin Multiplatform e o desenvolvimento nativo em iOS com Swift.

O objetivo do trabalho foi concluído com sucesso, existindo uma ferramenta que, após vários testes e métricas tiradas, provou que tem um desempenho bastante melhor que o do FlatList. Para além disso, visto ter um modelo de construção e definição semelhante ao FlatList, torna-se fácil de implementar, caso a necessidade do leitor, seja efetuar uma migração.

Para tal cumprimento do objetivo foi efetuada uma criteriosa investigação, criando uma aplicação de teste, que conseguiu sob as mesmas condições de fluxo de dados, recriar uma página, para cada alternativa estudada. Essas páginas foram testadas utilizando a ferramenta RN Perf Monitor e após várias amostras recolhidas, cálculos estatísticos determinaram qual o componente com melhor desempenho, entre os estudados.

Palavras-chave: desempenho, aplicações móveis, React Native, listas encadeadas, Android, iOS

Abstract

Blip started the development of a cross-platform project for Betfair. This project comes at a time of large expansion of the sports betting market, in which Betfair intends to be a part - promoting the creation of a new application for mobile devices, with a modern, attractive and simplified use. That said, Blip decided to use React Native technology, taking into account the possibility of sharing part of the code (especially the business logic) in order to develop this application compatible with web browsers and iOS and Android devices. However, this technology is quite recent, with some performance problems detected - especially in the use of nested lists, with high data mutability.

The main objective of this document is to improve the performance mentioned above, by using a tool capable of replacing the current FlatList. To this end, carrying out a careful investigation into the new features of the technology, such as JSI Modules, the new tool was developed in native language - since it is an effective way to increase the performance of a mobile application.

In order to support all the research and development of the tool, different types of mobile applications are mentioned and analyzed, and an approach is made to the architecture that constitutes the iOS and Android platforms. An approach to the technologies used for its development is also carried out, such as an analysis between React Native technology and some existing technologies for the development of this type of applications, such as Flutter, Kotlin Multiplatform and native iOS with Swift.

The main objective of this document was successfully completed, because there is a tool that, after taken several tests and metrics, proved that it performs much better than FlatList. Furthermore, since it has a construction and definition model similar to FlatList, it becomes easy to implement, if the reader needs to perform a migration.

To achieve this objective, a careful investigation was carried out, creating a test application, which managed, under the same data flow conditions, to recreate a page for each alternative studied. These pages were tested using the RN Perf Monitor tool and after several samples collected, statistical calculations were made to determine which component had the best performance among those studied.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excerto de Código

Lista de Acrónimos

2D	Two-Dimensional.
3D	Three-Dimensional.
AHP	Analytic Hierarchy Process.
AIR	Additions in Incremental Releases.
API	Application Programming Interface.
ARM	Advanced RISC Machine.
ART	Android Runtime.
BSD	Berkeley Software Distribution.
CPU	Central Process Unit.
CSS	Cascading Style Sheets.
FFE	Fuzzy FrontEnd.
GC	Garbage Collector.
GPS	Global Positioning System.
HAL	Hardware Abstraction Layer.
HTML	HyperText Markup Language.
HTML5	HyperText Markup Language Version 5.
HTTP	Hypertext Transfer Protocol.
JSI	JavaScript Interface.
JSON	JavaScript Object Notation.
LLVM	Low Level Virtual Machine.
MF	Moving Fast.
NCD	New Concept Development.
NDK	Native Developer Kit.
NPD	New Product Development.
RAM	Random Access Memory.
SDK	Software Development Kit.
SMS	Short Message Service.
TSG	Technology Stage Gate.

UI	User Interface.
URL	Uniform Resource Locator.

Capítulo 1

Introdução

Neste primeiro capítulo apresenta-se um breve contexto, assim como a descrição do problema a resolver, tendo como objetivo o enquadramento do leitor no tema em questão. Para além disso, é exibido um resumo da análise de valor e os objetivos pretendidos. Por fim, é apresentada a estrutura do documento.

1.1 Contexto

Cada vez mais caminha-se para uma sociedade tecnológica, em que esta característica se torna indispensável para o quotidiano - seja para atividades pessoais ou profissionais. Esta evolução registou um avanço exponencial, em meados do século XX, acrescentando valor à vida em sociedade (Kohn e Moraes 2007).

Entre as diversas ferramentas tecnológicas que se dispõe, pode-se encontrar o telemóvel, mais propriamente o *smartphone* - seguramente o mais utilizado na atualidade. Mas por que se deve dar destaque a este aparelho tecnológico? Com a inovação no meio tecnológico, um simples *smartphone* possui a capacidade de reunir diversos componentes num único dispositivo, tais como: a máquina fotográfica, a câmara de filmar, o computador, os sensores de temperatura e de pressão, entre outros. Além do mais, também usufrui da capacidade de utilizar aplicações que simplificam o quotidiano, tais como as agendas, as listas de contactos, as mensagens, o(s) navegador(es) de internet, entre outras.

Estas aplicações tornam-se compatíveis com os *smartphones* visto que a sua construção é feita com base no *software* do próprio aparelho. Ou seja, tanto pode ser construído com base num sistema operativo Android, como através do sistema operativo iOS.

Após alguns anos, uma das estratégias utilizadas pela maioria das empresas consiste no uso de aplicações móveis que incluem os seus produtos, ou simplesmente como um meio de gestão de negócios – desta forma potencializam-se mais oportunidades e escolhas aos seus clientes e colaboradores, respetivamente.

Esta estratégia utilizada pelas empresas acarreta custos, e até há pouco tempo, era necessário criar uma aplicação independente tanto para Android, como para iOS, exigindo um esforço maior (por parte da empresa de *software* contratada), para reunir colaboradores que obtivessem conhecimento relativamente aos sistemas operativos de Android e iOS – o que aumentava significativamente os custos mencionados inicialmente.

Posto isto, o Facebook desenvolveu uma tecnologia *cross-platform*, - que partilha o mesmo código base, para a criação de uma aplicação Android e iOS - chamada React Native. Isto permite construir, a partir do mesmo código base, uma aplicação móvel quase nativa para os

sistemas Android e iOS, utilizando como linguagem de desenvolvimento o JavaScript. Este aspeto permite diminuir os custos de mão de obra especializada, as máquinas necessárias para a implementação do projeto e ainda o tempo necessário para o desenvolvimento do mesmo.

Para exemplificar a utilização da tecnologia React Native, e as suas capacidades, menciona-se uma empresa portuguesa, denominada Blip, que está a utilizar esta mesma tecnologia, para desenvolver uma aplicação móvel para Android e iOS, tendo em consideração a partilha da lógica de negócio, utilizando a mesma base de código.

Contudo, não se pode afirmar que é uma tecnologia capaz de construir uma aplicação móvel com um desempenho excecicional, visto que necessita de converter parte do código, para código nativo. Isto não se aplicaria se fosse uma aplicação móvel nativa, pois esta é caracterizada por ser desenvolvida especificamente para uma plataforma. No caso da plataforma Android é desenvolvida com recurso à linguagem Java ou Kotlin, e no caso de iOS desenvolvida em linguagem Swift ou Objective-C.

Este tipo de desenvolvimento tem vantagens, uma vez que são desenvolvidas utilizando o software já presente num dispositivo, logo têm acesso a algumas funcionalidades, como o Global Positioning System (GPS) ou o *Touch ID*. Do mesmo modo, também existe a possibilidade de utilizar Application Programming Interface (API) existentes nessa plataforma, podendo assim desenvolver aplicações com um maior desempenho e usabilidade.

1.2 Problema

Atualmente, a tecnologia de React Native tem vindo a ganhar destaque no mercado tecnológico, encontrando-se em constante evolução. Tal como foi mencionado anteriormente, consegue criar aplicações móveis quase nativas, tanto para o sistema operativo Android, como iOS, utilizando JavaScript como linguagem base.

Posto isto, a empresa é responsável por um projeto, que consiste tanto no desenvolvimento de uma aplicação *web*, otimizada para dispositivos móveis, como de uma aplicação nativa para as plataformas Android e iOS. Para tal, decidiu usar esta tecnologia para o desenvolvimento de uma aplicação. Esta aplicação, que será detalhada posteriormente na Secção 2.2, é parte integrante de uma área de negócio relativa ao mercado de apostas desportivas.

Mas, como é verificável em todas as tecnologias, existe sempre um espaço para melhoria, assim como surgem alguns problemas a resolver. Neste caso, os componentes pré-definidos para a renderização de listas (FlatList e VirtualizedList), em certas situações, podem apresentar problemas de desempenho. Isto acontece quando existem listas muito complexas, tais como as listas encadeadas, como se verifica na aplicação que está atualmente em desenvolvimento. Este problema em específico é bastante impactante, visto que devido às características da aplicação, as listas são dos elementos mais utilizados na sua User Interface (UI).

Nesta última surgem diversos problemas de desempenho, nos componentes que utilizam FlatList ou VirtualizedList (a título de exemplo, ao executar uma navegação numa página da aplicação, quando existe uma lista, os elementos demoram a ser carregados, tornando a experiência do utilizador confusa e pouco acessível).

1.3 Objetivos

O principal objetivo deste trabalho foca-se no melhoramento da solução atualmente existente, que possibilite um aumento de desempenho da atual aplicação da Betfair, aquando da presença de listas encadeadas.

Com o objetivo de tornar a tecnologia React Native mais adequada à criação de aplicações complexas (que necessitam de alto desempenho), esta solução permitirá melhorar o mesmo, com o suporte de um componente externo.

1.4 Análise de Valor

A análise de valor, que será detalhada no Capítulo 3, vai ao encontro do objetivo previamente definido no projeto, ou seja, trazer valor ao produto desenvolvido pela empresa, tendo em conta o desempenho da aplicação – sobretudo em momentos em que se utilizem listas encadeadas.

Esta análise de valor será realizada tendo em conta o modelo New Concept Development (NCD) e o método Analytic Hierarchy Process (AHP). Para além deste aspeto, definir-se-á o valor percecionado, o valor para os utilizadores do produto e a proposta de valor. Por fim, realizar-se-á o modelo de negócio Canvas, de forma a completá-lo.

1.5 Estrutura

Após a análise criteriosa e a construção posterior da estrutura deste documento, a mesma encontra-se dividida em seis capítulos:

- **Capítulo 1: Introdução**, onde se enuncia o problema, os objetivos, um prelúdio da análise de valor e a estrutura do documento.
- **Capítulo 2: Contexto, Problema e Estado da Arte**, onde se realiza uma contextualização mais detalhada do problema a resolver, enunciando os impactos causados na atual utilização em projetos. De seguida, será exposto o objetivo deste documento, que conduzirá as restantes páginas. Por fim, será apresentada a respetiva tecnologia, com destaque para a solução investigada especificamente para melhorar o desempenho da mesma. Além do mais, também se dá a conhecer a importância deste documento para o desenvolvimento futuro de algumas aplicações. Tal como uma avaliação das tecnologias existentes, para o desenvolvimento de uma aplicação multiplataforma.
- **Capítulo 3: Análise de Valor**, onde se realiza uma análise de valor. Como nesta fase o problema em estudo já se encontra bem delimitado, torna-se possível analisar uma solução para combater o mesmo e ir ao encontro do objetivo delineado inicialmente neste documento.
- **Capítulo 4: Análise, Design e Implementação da Solução**, que tem como intuito de apresentar uma análise, *design* e a construção da solução para o problema delimitado, bem como alguns excertos de código relevantes.
- **Capítulo 5: Testes e Resultados**, onde é realizado testes de desempenho, utilizando uma ferramenta da tecnologia de React Native, RN Perf Monitor, possibilitando a avaliação da solução construída. Também serão apresentados os principais resultados deste documento.

- **Capítulo 6: Conclusões e Trabalho futuro**, onde serão apresentadas conclusões retiradas após a realização deste documento, e também sobre trabalho futuro de forma a poder evoluir e analisar com mais detalhe a solução desejada.

Capítulo 2

Contexto, Problema e Estado da Arte

O presente capítulo é dedicado à contextualização detalhada do problema (utilizando um projeto que se encontra atualmente em desenvolvimento pela empresa Blip), conduzindo o leitor à sua melhor compreensão. Para tal, será exposta a área de negócio em que se insere a aplicação em desenvolvimento, mencionada anteriormente, assim como os aspetos que prejudicam o seu funcionamento. Por fim, será realizada uma abordagem das tecnologias utilizadas – bem como das tecnologias existentes – com o intuito de obter uma perspetiva geral sobre o ambiente do projeto e do documento.

2.1 Contexto

Na sequência do que foi referido no Capítulo 1, na Secção 1.1, destacam-se dois pontos importantes: a área de negócio onde se enquadra a aplicação e o produto que a empresa se encontra a desenvolver atualmente, sendo também realizada uma contextualização organizacional.

2.1.1 Contextualização Organizacional

A Blip é uma empresa portuguesa, de desenvolvimento de *software* focado em aplicações *web*, que foi fundada em 2009, no Porto. Esta empresa pertence ao grupo Flutter, que alberga diversas organizações - maioritamente relacionadas com o mercado de apostas desportivas - como a Betfair, a PaddyPower e a FanDuel (*Blip | Blip, a Flutter Company* 2022).

Atualmente está a desenvolver um projeto multiplataforma, que engloba uma aplicação nativa para dispositivos móveis, para a organização Betfair. Esta organização funciona como um cliente da Blip, apesar de ambos pertencerem ao grupo Flutter. A Betfair é considerada uma das maiores empresas de apostas desportivas a nível global, atuando um pouco por todo o mundo, com exceção de alguns locais, como Portugal devido a razões legais impostas pelo governo português (*Betfair » Como apostar na maior Bolsa de Apostas do mundo?* 2022).

2.1.2 Área de Negócio

A aplicação que está a ser desenvolvida enquadra-se no mercado das apostas desportivas. Esta área de negócio esteve sempre presente no mundo do desporto, embora não se constituísse como uma atividade organizada, como é verificado na atualidade.

Desde a Antiguidade, o ser humano não consegue somente assistir a um desporto, tendo a necessidade de se envolver - como parte integrante na competição ou prova que está a decorrer. Em todas as culturas e em qualquer época, encontram-se sinais de apostas em eventos e/ou competições (*Como surgiram as apostas desportivas? - Jornal o Interior* 2022).

No início, as apostas surgiam apenas entre espectadores, sendo o teor da aposta referente ao vencedor da competição. O vencedor da aposta recebia o montante de outro apostador ou outro previamente acordado, nascendo assim o significado de aposta (*Como surgiram as apostas desportivas? - Jornal o Interior* 2022).

Com o decorrer do tempo, as apostas evoluíram de modo a aumentar o público-alvo, criando desta forma novos meios de aposta, com intermediários – o que conduziu ao nascimento das casas de apostas. Como era expectável, a internet teve um grande impacto no mercado das apostas desportivas, visto que a sua crescente popularidade se constituiu como um meio de desenvolvimento distinto para qualquer área de negócio – inclusive a área das apostas desportivas (*Como surgiram as apostas desportivas? - Jornal o Interior* 2022).

Antes deste meio tecnológico, as apostas realizavam-se somente em regime presencial, ou seja, era necessário que o apostador se deslocasse a uma casa de apostas para efetuar a sua previsão. Esta necessidade tornou-se irrelevante com a evolução da tecnologia, uma vez que, atualmente, o apostador pode registar o seu palpite a partir do conforto da sua casa (*Como surgiram as apostas desportivas? - Jornal o Interior* 2022).

Com a entrada das apostas desportivas no contexto tecnológico, tornou-se possível a introdução de apostas em tempo real, ou seja, durante a ocorrência do evento – criando-se desta forma dois grandes mercados: Exchange e Sportbook.

Exchange

O mercado Exchange é definido por ser uma aposta, que um utilizador efetua contra um outro utilizador - que tenha efetuado uma aposta no mesmo evento, mas que a previsão do acontecimento seja contrária, ao do primeiro utilizador. Ou seja, um utilizador faz uma aposta do tipo Lay, de um certo evento, enquanto um outro utilizador faz uma aposta do tipo Back, desse mesmo evento (*What is the Betfair Exchange?* 2022).

O Lay e o Back são os dois tipos de aposta existentes neste mercado, sendo caracterizados da seguinte forma:

- **Lay**: é uma aposta em algo para não acontecer. A título de exemplo, apostar que uma equipa não vai vencer o jogo (*Exchange: Get Started with Betfair Exchange* 2022).
- **Back**: ao contrário do Lay, é uma aposta em algo para acontecer. Por exemplo, apostar que uma equipa vai vencer o jogo (*Exchange: Get Started with Betfair Exchange* 2022).

Este mercado é considerado o mais lucrativo para uma casa de apostas, visto que, como mencionado anteriormente, um utilizador aposta contra outro utilizador, tornando este local num mero intermediário na negociação. Ou seja, a casa de apostas nunca regista valores de prejuízo, uma vez que os valores ganhos pelo utilizador vencedor provêm dos gastos do utilizador perdedor. Posto isto, a casa de apostas recebe uma comissão do valor adquirido pelo apostador vitorioso.

Um exemplo de uma página de um mercado Exchange pode ser observada na Figura 2.1.

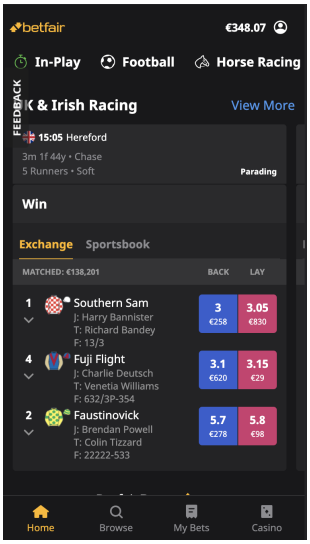

Figura 2.1: Página de um mercado Exchange

Sportbook

O mercado Sportbook (Figura 2.2) caracteriza-se por ser uma aposta tradicional. Ou seja, um utilizador aposta um valor, segundo uma certa *odd*[1], em algo para acontecer, sendo que o valor em caso de aposta ganha, é pago pela casa de apostas, mas em caso de aposta perdida, todo esse valor é adquirido pela casa de apostas (*Sportsbook: How can I use my free bet?* 2022).

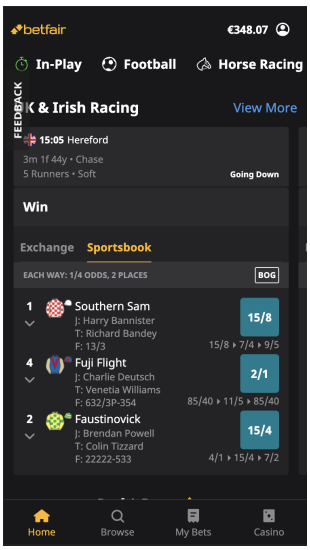

Figura 2.2: Página de um mercado Sportbook

2.1.3 Produto

Atualmente, a Betfair dispõe de duas aplicações distintas para os seus clientes poderem apostar: uma baseada no mercado Exchange e outra no mercado Sportbook.

[1] "probabilidade de ocorrência de um evento, dividida pela probabilidade da não ocorrência do mesmo evento."

Com a necessidade de renovar o seu produto e de atrair novos clientes, a empresa decidiu criar um novo projeto, intitulado Betfair Rebuild, que tem como objetivo a criação de uma nova aplicação móvel, em ambiente *web*, mas também a criação de uma aplicação móvel nativa, tanto para iOS, como Android.

A grande novidade desta aplicação é a junção dos dois mercados, que estavam divididos por duas aplicações, de forma a simplificar a sua utilização. Com diversas alterações visuais, o objetivo da aplicação seria apresentar um aspeto visual moderno, adequado ao quotidiano.

Uma versão beta já foi lançada, podendo observar-se o produto final, na Figura 2.3.

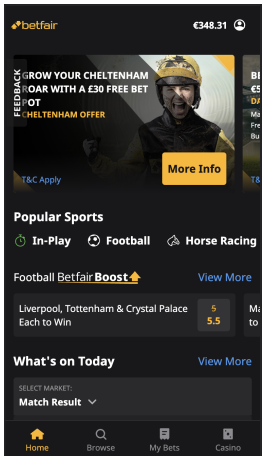

Figura 2.3: Aplicação Betfair Página Principal

2.2 Problema

No seguimento do que foi referido no Capítulo 1, na Secção 1.2, atualmente existe um problema de desempenho considerável em aplicações desenvolvidas em React Native, especialmente quando são utilizadas listas encadeadas.

Tendo como foco principal o problema surgido na empresa, o projeto atualmente em desenvolvimento sofre de uma grande instabilidade a nível de desempenho, visto que os componentes da tecnologia não são otimizados de forma a permitir um uso mais específico de certas listas encadeadas.

Ou seja, como se trata de um projeto realizado para uma área de negócio de apostas desportivas (um mercado bastante volátil, em termos de informações e serviços prestados), a alteração contínua de dados, em tempo real, é bastante elevada – como se pode exemplificar pela instabilidade das *odds*, que sofrem alterações a cada segundo. Posto isto, torna-se necessária a realização de um pedido a um serviço, sempre que esse tipo de alteração ocorre – provocando um consumo elevado dos recursos disponíveis, tanto pela aplicação, como pelo dispositivo onde está a ser acedida.

Estas constantes alterações têm suscitado a produção de efeitos negativos no desempenho da aplicação, tornando-se bastante lenta ao carregar as listas e os seus elementos, como também na tentativa de navegação pelo resto da aplicação. Num mercado destas características, qualquer problema de desempenho é considerado grave, uma vez que a demora na mudança de qualquer informação pode conduzir à aposta de um cliente com valores que já

não se encontram em concordância com o decorrer do evento desportivo, em que pretende registar a aposta.

Para sustentar este problema, foram retiradas métricas da atual aplicação, com uma ferramenta externa - desenvolvida especificamente para React Native - que podem ser analisadas e observadas na Figura 2.4.

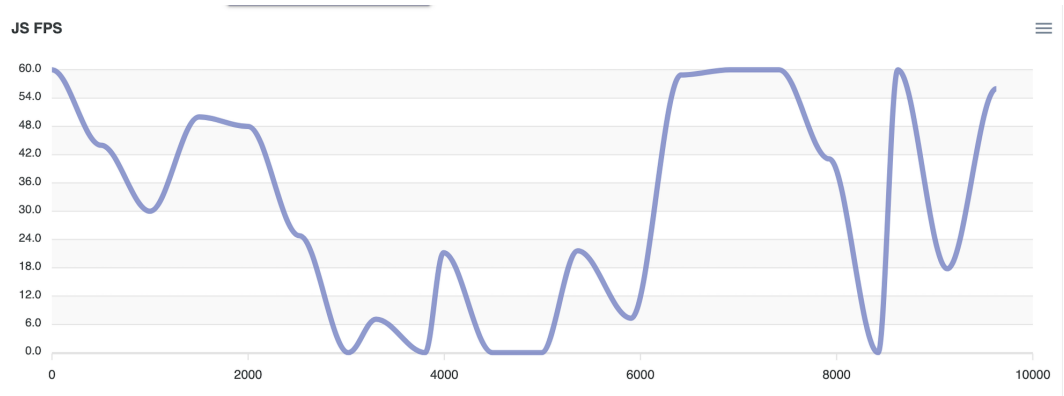

Figura 2.4: Métricas de Desempenho

Como se pode verificar, o *frame rate*, da JS Thread (Figura 2.4), que será abordada posteriormente na Subsecção 2.4.4, revelou uma redução significativa após um ligeiro movimento na aplicação, alcançando valores reduzidos, passando de 60 fps[2], para 0 fps. Também é possível verificar uma inconsistência nos fps, derivado ao movimento previamente referido. Por consequência, realiza novos pedidos a serviços, uma vez que cada novo elemento renderizado pode ser composto por outros elementos secundários, que também necessitam de efetuar pedidos, como demonstrado na Figura 2.5. Isto equivale a uma experiência demasiado limitativa e, por vezes até confusa, para um utilizador.

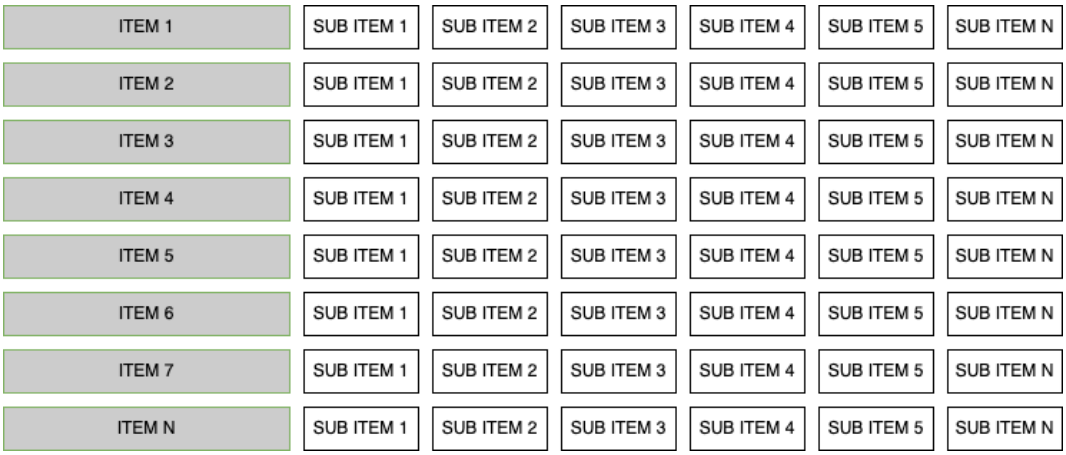

Figura 2.5: Sequência de items

2.3 Objetivo

Tal como já foi abordado na Secção 1.3, o trabalho elaborado no contexto desta dissertação compreende determinados objetivos para alcançar uma solução para o problema existente.

[2] *Frames* por segundo

Ou seja, substituir o atual FlatList, quando utilizada para mostrar listas encadeadas com uma significativa instabilidade de dados, ou seja, dados em constante mutabilidade.

Estes objetivos podem ser subdivididos em determinadas etapas, tais como:

- Investigar alternativas à implementação do FlatList;
- Melhorar o código atualmente em desenvolvimento, seguindo as boas práticas de programação e de utilizaçao de React/React Native;
- Criação de uma aplicação, com o objetivo de testar e analisar as alternativas;
- Teste das alternativas encontradas na investigação;
- Implementação da lista alternativa, com melhor desempenho na execução dos testes, na aplicação da Betfair;
- Teste da lista alternativa na aplicação da Betfair;

Estes objetivos podem ser agrupados num objetivo principal, "Será possível melhorar o desempenho da aplicação, desenvolvida em React Native, quando utiliza listas encadeadas?".

2.4 Estado da Arte

Neste documento, o Estado da Arte tem como objetivo apresentar o estudo das tecnologias associadas à realização do projeto, bem como encontrar abordagens para a resolução do problema. Além disto, serão analisados os diversos tipos de aplicações móveis e tecnologias já existentes, que assumem o mesmo comportamento que a tecnologia utilizada. Por fim, realizar-se-á uma contextualização dos tipos de *software* existentes para os dispostivos móveis.

2.4.1 Tipos de Desenvolvimento de Aplicações Móveis

Serão abordados de seguida, os diferentes tipos de desenvolvimento de aplicações móveis, tais como as aplicações *web*, as aplicações nativas e as aplicações híbridas.

Aplicações web

As aplicações *web* são soluções que necessitam de um navegador para serem executadas, tais como Safari, Google Chrome e Mozilla Firefox - sendo vulgarmente conhecidas como *websites* otimizados para dispostivos móveis. Por não ser necessária a sua instalação no dispositivo do utilizador, como um computador ou um *smartphone*, estas aplicações destacam-se por estarem alojadas num servidor web. Este é acedido, posteriormente, pelos utilizadores, a partir do uso de protocolos , geralmente Hypertext Transfer Protocol (HTTP), que faz parte do vulgarmente conhecido Uniform Resource Locator (URL) - que se caracteriza por ser o identificador da aplicação que se pretende aceder.

O desenvolvimento deste tipo de aplicações recorre a linguagens suportadas pelos navegadores, como HyperText Markup Language (HTML), Cascading Style Sheets (CSS) e JavaScript. Por essa mesma razão, este tipo de aplicação pode ser acedida por qualquer dispositivo, desde que tenha acesso à internet, sem problemas de compatibilidade com o mesmo.

Para além da compatibilidade, uma das vantagens do desenvolvimento de aplicações *web* é a facilidade de manutenção e escalamento, associados ao baixo custo de desenvolvimento. Este último aspeto pode ser explicado, visto que podem ser desenvolvidas com tecnologias bastante estáveis e conhecidas pela maioria dos programadores - não necessitando de mão de obra qualificada para estas tarefas.

Tal como em todas as aplicações, também existem desvantagens. Uma das maiores desvantagens consiste na necessidade constante de estar ligado à internet, pois necessita de estar em contacto direto com o servidor web, externo ao dispositivo. Outra das desvantangens é a limitação de acesso e utilização dos componentes de *hardware* dos dispositivos, como o GPS ou a câmara, o que torna o seu desempenho e a responsividade francamente mais reduzida, em comparação com as oferecidas pelas aplicações nativas (*Usabilidade* 2022).

Atualmente, estão a ser desenvolvidas novas funcionalidades e tecnologias que permitem melhorar o desempenho e a responsividade de aplicações *web*, num meio móvel, como a criação do HyperText Markup Language Version 5 (HTML5). Este permitiu melhorar a experiência de utilização, podendo utilizar a aplicação, mesmo estando *offline*, devido à utilização de cache e ao suporte mais eficaz de conteúdo de vídeo ou geolocalização (Ribeiro 2014).

Aplicações nativas

As aplicações nativas são soluções desenvolvidas especificamente para uma plataforma. No caso da plataforma Android, estas aplicações são desenvolvidas segundo uma linguagem base de Java ou Kotlin e, no caso de uma plataforma iOS, desenvolvidas segundo uma linguagem de Swift ou Objective-C (Martins 2018).

Como são desenvolvidas utilizando o software já presente no dispositivo, estas aplicações têm acesso a todas as funcionalidades dos dispositivos, como a câmara, o *touch* ID (processo de autenticação) e o GPS, tal como a utilização de API's existentes para essa plataforma, podendo assim desenvolver aplicações com um desempenho e usabilidade bastante atrativos para os seus utilizadores.

Tendo em conta que a aplicação está instalada diretamente no dispositivo, a possibilidade de efetuar diversas operações sem ter de estar ligado à internet, consiste numa das grandes vantagens do desenvolvimento deste tipo de aplicações.

Ao serem desenvolvidas especificamente para uma plataforma, os seus custos de desenvolvimento são mais elevados, uma vez que é necessário mais tempo para o seu desenvolvimento, tal como a contratação de mão de obra bastante qualificada, por serem tecnologias com maior especificidade. Na eventualidade, do responsável da aplicação disponibilizar a mesma para ambas as plataformas, revela-se necessário uma duplicação da aplicação, numa outra linguagem, o que acarreta a contratação de um maior número de mão de obra qualificada (*Usabilidade* 2022).

Contudo, com o desenvolvimento da tecnologia, já existem diversas tecnologias que permitem desenvolver aplicações nativas sobre uma mesma linguagem base - sendo posteriormente feita uma conversão para uma linguagem compatível com Android ou iOS. Essa conversão provoca uma diminuição do desempenho, tornando o seu desenvolvimento mais complexo, de forma a ser possível contrariar a mesma.

Aplicações híbridas

As aplicações híbridas são aplicações *web* que incorporam uma base de uma aplicação nativa. Ou seja, são aplicações desenvolvidas segundo uma linguagem de HTML, CSS e JavaScript, que são posteriormente incluídas numa tecnologia de desenvolvimento nativo, como Java, no caso de Android, ou Objective-C, no caso de iOS, a partir da utilização de uma *framework* capaz de fazer essa ligação, como Ionic com Cordova.

Como são desenvolvidas, em parte como uma aplicação *web*, o seu custo de manutenção, escalamento e desenvolvimento tornam-se reduzidos, revelando-se desnecessário a utilização de mão de obra qualificada - pois são utilizadas tecnologias estáveis e do conhecimento geral da maioria dos programadores. Outra das vantagens (esta em particular, em relação às aplicações *web*) é a possibilidade de aceder a diversas funcionalidades nativas dos dispositivos. Este aspeto deve-se à possibilidade de serem usados Software Development Kit (SDK) nativos, o que permite um melhor desempenho dessas funcionalidades, em relação às aplicações *web*.

Contudo, tal como as aplicações *web*, também possuem desvantagens, sendo o seu desempenho mais baixo, assim como a sensação de usabilidade não é tão fluída, comparando com as aplicações nativas (Martins 2018).

Comparação entre Aplicações web, Híbridas e Nativas

Após a contextualização dos diferentes tipos de aplicações móveis existentes, foi efetuada uma comparação entre este tipo de aplicações, como se pode observar na Tabela 2.1.

Tabela 2.1: Comparação entre Aplicações *web*, Híbridas e Nativas

Critério	**Aplicações web**	**Aplicações Híbridas**	**Aplicações Nativas**
Necessária instalação	Não	Sim	Sim
Local de acesso	Navegador	Aplicação	Aplicação
Tipo de Tecnologia utilizada	Tecnologia *web*	Tecnologia *web*	Tecnologia Nativa
Utilização de recursos de um dispositivo móvel	Muito limitado	Alguns recursos disponíveis	Todos os recursos disponíveis
Custo	Pouco dispendioso	Algo dispendioso	Muito dispendioso
Manutenção e Escalamento	Facilitado	Facilitado	Limitado
Compatibilidade Software	Compatível com qualquer *software*	Compatível com a maioria dos *softwares*	Dependendo do *software* do dispositivo

Após a comparação efetuada, é possível determinar que uma aplicações *web* é a mais fácil de ser acedida - pois é compatível com qualquer plataforma - sendo necessário apenas a ligação à internet e a um navegador, tornando-se desnecessária a sua instalação. Apesar das aplicações híbridas, e sobretudo nativas, terem acesso a mais recursos dos dispositivos (e de forma mais eficiente) - uma vez que são acedidas através da linguagem nativa - revelam-se mais dispendiosas, que as aplicações *web*. Por fim, visto que as aplicações *web* e híbridas

são desenvolvidas segundo uma tecnologia *web*, o seu escalamento e a sua manutenção são significativamente mais facilitados que as aplicações nativas - devido à necessidade de uma permissão por parte dos utilizadores dos dispositivos, para efetuar atualizações.

2.4.2 Contextualização de Software

Nesta subsecção, realiza-se uma contextualização dos dois sistemas operativos mais utilizados em dispositivos móveis (iOS e Android), com uma taxa de utilização de cerca de 99,22%, como se pode observar na Figura 2.6.

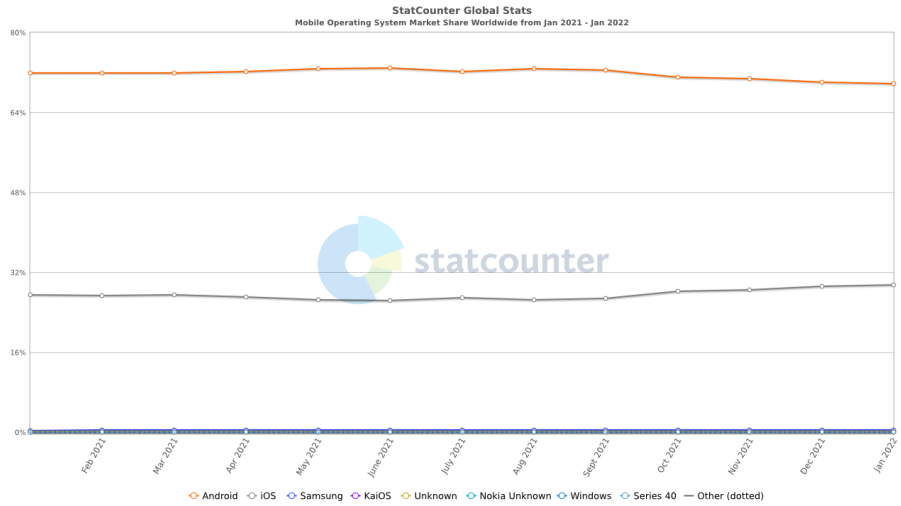

Figura 2.6: Gráfico Sistema Operativo Móvel mais utilizados (*Mobile Operating System Market Share Worldwide* 2022)

Android

O Android é um sistema operativo baseado no *kernel* do Linux, desenvolvido pela Google. Inicialmente para *smartphones*, o sistema Android tem como propósito ser usado para oferecer aos utilizadores uma experiência e alguns recursos que até então só eram possíveis num computador. Atualmente, já expandiu para as televisões inteligentes (Android TV), mercado automóvel (Android Auto) e relógios digitais inteligentes (Android *Wear*).

Este sistema operativo é um *software* open source, ou seja, o seu código base de desenvolvimento está disponível para consulta, assim como para a adição de novas funcionalidades e melhorias – sem que a versão original seja alterada para a restante comunidade tecnológica e respetivos utilizadores (Novac et al. 2017).

A arquitetura de um sistema Android é composta pelos seguintes componentes representados na Figura 2.7.

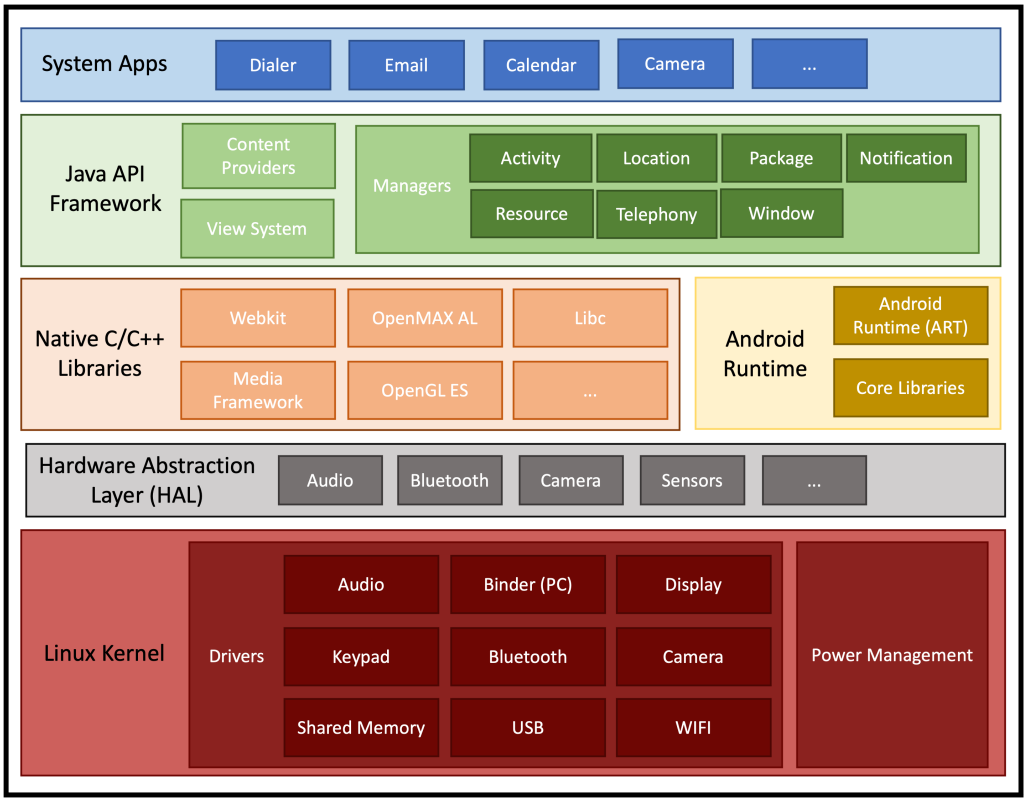

Figura 2.7: Arquitetura do sistema Android (*Arquitetura da plataforma | Desenvolvedores Android* 2022)

- **Linux Kernel**, que é responsável por funcionalidades como o encadeamento e gestão da memória de baixo nível. Para além de permitir o aproveitamento dos recursos de segurança principais, também possibilita que os fabricantes de dispositivos possam desenvolver *drivers* de *hardware* para um *kernel* conhecido (*Arquitetura da plataforma | Desenvolvedores Android* 2022).
- **Hardware Abstraction Layer (HAL)**: caracterizando-se por serem módulos de biblioteca, que permitem implementar uma interface para um tipo específico de hardware, como o *bluetooth* ou a câmara. Quando uma API efetua uma chamada para aceder ao *hardware* de um dispositivo, o sistema carrega um dos módulos de biblioteca para o componente de *hardware* correspondente (*Arquitetura da plataforma | Desenvolvedores Android* 2022).
- **Android Runtime (ART)**: A partir da versão 5.0, todas as aplicações executam os seus processos com uma instância de Android Runtime. Ou seja, o ART é projetado para executar diversas máquinas virtuais em dispositivos de baixa memória, executando arquivos DEX[3], de forma a otimizar o consumo de memória. Alguns dos seus principais recursos são: a Compilação AOT[4] e JIT[5], Garbage Collector (GC) otimizada e uma melhor compatibilidade de depuração (*Arquitetura da plataforma | Desenvolvedores Android* 2022).

[3] Dalvik (nome do autor) executable: formato de bytecode projetado especialmente para Android (Tendo sido a 1ª máquina virtual Java para Android criada).

[4] *ahead-of-time* : antes do tempo, usualmente no processo de build

[5] *just-in-time*: mesmo a tempo, usualmente logo a seguir a um processo dependente

- **Bibliotecas C/C++ nativas**: Alguns dos principais componentes e serviços do Android, como o ART e o HAL, são implementados por código nativo que exigem bibliotecas nativas programadas em C e C++. Para conseguir expor as funcionalidades dessas bibliotecas nativas para as aplicações, o Android fornece as Java Framework API's. Esse aspeto permite que uma aplicação tenha a capacidade de aceder, por exemplo ao OpenGL ES e Java OpenGL API - que permite desenhar gráficos Two-Dimensional (2D) e Three-Dimensional (3D) (*Arquitetura da plataforma | Desenvolvedores Android* 2022).
- **Java API Framework**: Todos os recursos do sistema de Android estão disponíveis a partir da utilização de API's programadas em Java. A sua utilização permite criar as aplicações de Android com mais eficiência, pois simplifica a reutilização de componentes e serviços de sistema modulares e principais - nos quais se destacam o provedor de conteúdo e o gestor de notificações (*Arquitetura da plataforma | Desenvolvedores Android* 2022).
- **System Apps**: O sistema Android já disponibiliza algumas aplicações pré-definidas, como o envio de Short Message Service (SMS), calendários, navegador de internet, lista de contactos, entre outros. Contudo, estas aplicações podem ser substituídas de acordo com as preferências dos utilizadores, por aplicações externas similares (com exceção de algumas aplicações, como as Configurações de Sistema) (*Arquitetura da plataforma | Desenvolvedores Android* 2022).

iOS

O iOS é um sistema operativo desenvolvido pela Apple. Tal como o Android, inicialmente foi desenvolvido para *smartphones* (mais propriamente o *iPhone*). Entretanto, expandiu para outros dispositivos como *iPod Touch*, *iPad* e *Apple TV* (Novac et al. 2017).

Este sistema operativo é uma evolução de um sistema operativo mais antigo, o *Darwin* - uma versão open-source do MacOS, que contribui ativamente para o mesmo (e para o iOS)-, que se baseava na Berkeley Software Distribution (BSD) Unix. Para além disso, também deu origem ao Mac OS X, utilizado nos computadores da marca Apple (*Android and iOS* 2022).

As aplicações desenvolvidas para iOS raramente comunicam diretamente com algum componente de *hardware* de um dispositivo (semelhante ao HAL) - são usadas interfaces que fazem a ligação entre a aplicação e o *hardware* que pretendem aceder (Alexandre Leite e Helena Macedo 2018).

A arquitetura de um sistema iOS é composta pelos seguintes componentes apresentados na Figura 2.8.

- **Cocoa Touch**: Consiste na infraestrutura para as tecnologias como a multitarefa, o serviço de notificação e os diversos serviços de alto nível do sistema, que são determinadas nesta camada. Tal como as principais *frameworks* para o desenvolvimento de aplicações. Esta camada foi desenvolvida em Objective-C e baseia-se na Cocoa Framework.(Alexandre Leite e Helena Macedo 2018)
- **Media Services**: Caracteriza-se por ser a camada responsável pelo sistema multimédia de um dispositivo iOS. Ou seja, a partir desta camada torna-se possível aceder tanto a recursos como áudio, vídeo e gráficos, como *frameworks* para o desenvolvimento de aplicações multimédia (Alexandre Leite e Helena Macedo 2018).

Segundo (Alexandre Leite e Helena Macedo 2018), esta camada foi projetada "para tornar mais prática a implementação de aplicações com o uso do *framework* UIKit (User Interface Kit) que por sua vez disponibiliza recursos como Core Graphics, Core Animation, OpenGL ES, Text Core, Image I/O e Framework Assets Library".

- **Core Services**: Constitui-se como a camada responsável pela gestão dos serviços fundamentais utilizados por uma aplicação. Para além disso, também realiza o suporte de recursos como localização, *iCloud* e rede (Alexandre Leite e Helena Macedo 2018).
- **Core OS**: Esta camada caracteriza-se por ser a responsável pela utilização das *frameworks* Accelerate e External Accessory, de forma a assegurar o controlo de segurança do dispositivo e da comunicação, com o *hardware* que pretende aceder.

  A *framework* Accelerate disponibiliza diversas interfaces para realizar cálculos complexos, de modo a trabalhar com o processamento de sinais digitais, por exemplo. A sua utilização torna-se uma vantagem, pois as suas interfaces são otimizadas para qualquer configuração de *hardware* que se pretenda aceder, de um dispositivo iOS (Alexandre Leite e Helena Macedo 2018).

  A *framework* External Accessory é caracterizada por permitir "a comunicação da parte lógica do dispositivo com acessórios de *hardware* conectados a ele através de uma interface de comunicação"(Alexandre Leite e Helena Macedo 2018).

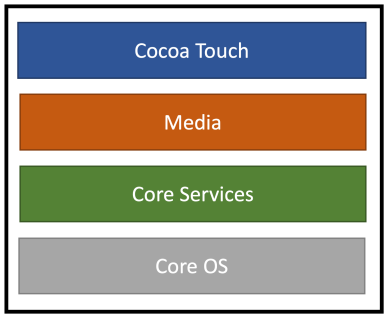

Figura 2.8: Arquitetura do sistema iOS *Funcionamento do Sistema Operacional ios* 2022

Após a contextualização do *software* existente para dispositivos móveis, foi efetuada uma comparação entre este tipo de *software*, como se pode observar na Tabela 2.2.

Tabela 2.2: Comparação entre iOS e Android

Critério	iOS	Android
Desenvolvedor	Apple	Diversos (destacando-se a Google)
Tipo de software	Fechado, com componentes Open Source	Open Source
Segurança	Mais seguro	Menos seguro que iOS
Compatibilidade dispositivos	Dispositivos Apple (iPad, iPod Touch, Apple Tv e iPhone)	Aberto a diversas marcas (Samsung, OnePlus, entre outros), sendo produzido para televisões, relógios inteligentes, automóveis, *tablets* e *smartphones*
Store	Apple App Store	Google Play Store
Base Software	Kernel Darwin OS	Kernel Linux
Família Sistema Operativo	Unix, OS X	Linux

Após a comparação efetuada, é possível determinar o *software* Android como o mais abrangente, em relação à compatibilidade com dispositivos de diferentes marcas, contudo revela-se menos seguro que o software iOS. Isto deve-se à exclusividade deste *software* para dispositivos Apple e também pela forma arquitetural como é construído. O tipo de *software* presente em dispositivos Android é mais diversificado e livre, tendo em conta que é *open source* - logo existe a possibilidade de cada utilizador poder acrescentar funcionalidades próprias ao seu dispositivo. A base do *software* desenvolvido pelo Android é Linux, sendo que o iOS optou por se basear no Darwin OS - antigo sistema operativo desenvolvido pela Apple, porém ainda atual.

2.4.3 Tecnologias existentes

Nesta subsecção, são abordadas algumas tecnologias existentes para o desenvolvimento de aplicações móveis, apesar da tecnologia a utilizar na aplicação já estar previamente definida. Contudo, entendeu-se apresentar outras tecnologias relacionadas com esse tipo de desenvolvimento de aplicações.

Swift

O Swift é a principal linguagem de programação, que usa bibliotecas da Apple para o desenvolvimento de aplicações para o mercado da Apple, tal como a iOS, iPadOS, macOS, tvOS e watchOS, destacando-se na comunidade tecnológica devido à forte adoção pelos utilizadores – sendo a tecnologia mais usada para o desenvolvimento deste tipo de aplicações (Apple 2022d).

Segundo (Apple 2022d), esta tecnologia destaca-se por ser "rápida, segura e interativa". Estes três aspetos podem ser definidos da seguinte forma:

- **Segura**: É considerada uma tecnologia segura, visto que as varáveis são sempre inicializadas. Na tentativa de se aceder a um *array*, a sua posição é sempre validada, de

forma a não ser possível exceder o limite do mesmo. A memória é gerida automaticamente, sendo reforçado o seu acesso - no caso de existirem erros de programação. De igual forma, também tem a capacidade de eliminar classes inteiras, caso este código seja inseguro (Apple 2022c).

- **Rápida**: É vista como uma tecnologia rápida, devido à utilização de um compilador de alta performance denominado Low Level Virtual Machine (LLVM). O seu código é convertido para código nativo, o que permite uma maior capacidade de aceder ao *hardware* do dispositivo (Apple 2022a).
- **Interativa**: É classificada como interativa, pela capacidade de se escrever código genérico (que depois é suportado por diversas extensões), permitindo também uma interação rápida pelas diversas coleções disponíveis. Para além do que foi mencionado anteriormente, esta tecnologia também é capaz de utilizar estruturas que suportam métodos, extensões e protocolos, assim como permitir a utilização de padrões mais orientados à programação funcional (Apple 2022a).

Uma das principais características desta tecnologia é a possibilidade de ser desenvolvida por módulos e integrá-los depois com facilidade, com o suporte de uma ferramenta, denominada Swift Package Manager, que é responsável pelos processos de construção, execução, testes e criação de *packages* das bibliotecas ou executáveis criados (Apple 2022b).

Flutter

O Flutter é uma tecnologia bastante recente, lançada em 2017. Com foco no desenvolvimento de aplicações móveis nativas e aplicações *web*, sob a mesma base de código, é desenvolvida segundo a linguagem de programação Dart (*Flutter documentation* 2022).

Esta tecnologia destaca-se por ter um desempenho semelhante às aplicações nativas. Isto deve-se, essencialmente, à utilização da seguinte abordagem:

- contém um mecanismo nativo de execução, que lida com as responsabilidades de baixo nível, como o desenho da UI e os recursos de interoperabilidade para acesso a bibliotecas e/ou plataformas. Para iOS é utilizado o mecanismo LLVM (como no Swift) e para Android é utilizado o Native Developer Kit (NDK) (Morgan 2019).
- "o código da aplicação é compilado para uma biblioteca Advanced RISC Machine (ARM), que juntamente com o mecanismo, cria a aplicação."(Morgan 2019).

A abordagem utilizada revela-se bastante favorável, uma vez que para além de ser uma tecnologia que permite o desenvolvimento de aplicações nativas e *web* (com a mesma base de código, tal como o React Native e o Ionic/Cordova), consegue compensar o *lag* (atraso) que estas tecnologias sofrem, devido à necessidade de conversão para o código nativo (Morgan 2019).

Contudo, esta tecnologia encontra-se num estado de estabilidade bastante reduzido, visto que o seu processo de *release* é bastante ativo (*Flutter Releases* 2022).

Kotlin Multiplatform

O Kotlin Multiplatform é uma tecnologia, tal como o Flutter e React Native, capaz de desenvolver aplicações móveis nativas e *web*, sob a mesma base de código. Destaca-se, na partilha de código comum, entre Android e iOS, a lógica de negócio e a capacidade de se

conectar a código específico - como o código nativo. Tal é verificado na Figura 2.9 (*Kotlin Multiplatform | Kotlin* 2022).

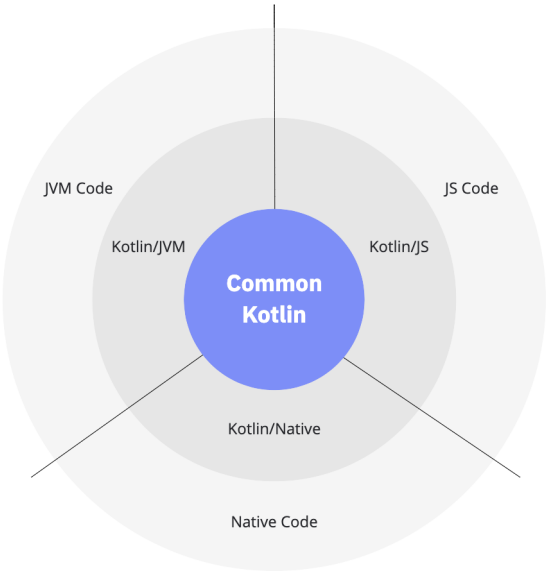

Figura 2.9: Representação Kotlin Multiplatform (*Kotlin Multiplatform | Kotlin* 2022)

Segundo (*Kotlin Multiplatform | Kotlin* 2022), "os projetos multiplataforma estão num estado *alpha*". Ou seja, ainda não é uma tecnologia estável, apesar de ser baseada numa linguagem de programação, já bastante consolidada no mercado tecnológico, denominada Kotlin - bastante utilizada para o desenvolvimento de aplicações Android.

Atualmente, nos diversos projetos que incluem esta tecnologia verificam-se diferentes tipos de estabilidade - entre os quais se destacam o Moving Fast (MF) e o Additions in Incremental Releases (AIR) - visto que são caracterizados por definirem projetos ainda em desenvolvimento, como se verifica na Tabela 2.3.

Tabela 2.3: Projetos Kotlin (*Stability of Kotlin components* 2022)

Componente	**Estabilidade**
Kotlin/JS	AIR
Kotlin/Native	AIR
Kotlin Scripts (*.kts)	AIR
dokka	AIR
Multiplatform Projects	MF

- **Moving Fast (MF)**: "não se deve esperar compatibilidade entre lançamentos mesmo incrementais, qualquer funcionalidade pode ser adicionada, removida ou alterada sem aviso prévio (*Stability of Kotlin components* 2022).
- **Additions in Incremental Releases (AIR)**: "podem ser adicionadas num lançamento incremental, as remoções e mudanças de comportamento devem ser evitadas e anunciadas num lançamento anterior, se necessário."(*Stability of Kotlin components* 2022).

Java

O Java é uma linguagem de programação baseada em classes e orientada a objetos. Esta linguagem tem semelhanças, em termos de sintaxe, em relação ao C e C++, embora organizada de forma diferente e com a introdução de ideias de outras linguagens. Tendo sido desenvolvida para ser uma linguagem de produção, são evitadas a adição de novas funcionalidades e, sobretudo funcionalidades não testadas devidamente (Gosling et al. 2021).

Como se trata de uma linguagem fortemente tipada, consegue distinguir, mais facilmente, os erros de compilação (*compile-time*) - que podem e devem ser detetados na compilação - e erros de execução (*runtime*). Normalmente, o *compile-time* consiste na tradução de programas para uma representção de *bytecode* independente da máquina. Já o *runtime* consiste, entre várias coisas, no carregamento e ligação das classes necessárias para a execução de um programa, o código gerado e a execução real do programa (Gosling et al. 2021).

Sendo considerada uma linguagem de alto-nível, o Java possui gestão automática de armazenamento, utiliza o GC e não permite acessos indevidos, tais como, o acesso a *arrays*, cuja posição não foi verificada (Gosling et al. 2021).

Objective-C

O Objective-C é uma linguagem de programação, usualmente utilizada para o desenvolvimento de *software* para iOS e OS X. É baseado na linguagem de programação C, pois utiliza a mesma sintaxe (acrescentando a sintaxe para definir classes e métodos), tipos primitivos e declarações de controlo de fluxo (*About Objective-C* 2022).

2.4.4 Tecnologias utilizadas

Nesta subsecção, realiza-se uma abordagem das tecnologias utilizadas para o desenvolvimento deste projeto. Para além da tecnologia responsável pelo desenvolvimento do projeto (React Native), também surgem abordagens acerca de outras tecnologias relacionadas, que constituem ou suportam a tecnologia principal.

React Native

O React Native é uma tecnologia *hybrid-native*, desenvolvida pelo *Facebook*, que é utilizada no desenvolvimento de aplicações móveis para dispositivos iOS e Android. Com apenas uma base de código, esta tecnologia é partilhada entre ambas as plataformas, e pode ser utilizada juntamente com o React – de forma a conseguir obter uma aplicação móvel para um ambiente *web* e para os dispositivos físicos, com parte da mesma base de código (*React Native · Learn once, write anywhere* 2022).

Esta tecnologia é baseada em JavaScript, que posteriormente utiliza chamada de componentes nativos, na camada responsável pela UI, em Java, no caso de Android e Objective-C no caso de iOS, de forma a poder renderizar os elementos da UI. Este aspeto é observável na Figura 2.10, em que são executadas chamadas a componentes nativos de Android como o `ImageView` e o `TextView`, e chamadas a componentes nativos de iOS, tais como o `UIImageView` e o `UITextView` (Câmara 2018).

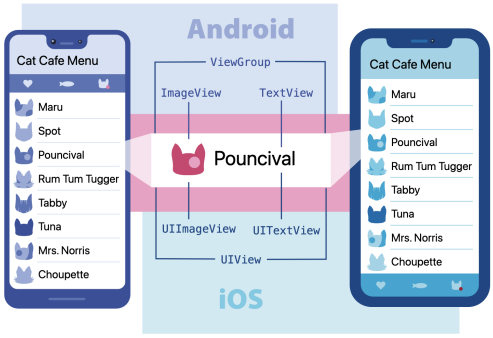

Figura 2.10: React Native | Android e iOS UI (*Core·React Native* 2022)

Os componentes, que são nativos de iOS e Android, são chamados de Componentes Nativos, que posteriormente dão origem a um único componente, dando origem aos Componentes Core. Na Tabela 2.4, é possível verificar alguns exemplos de Componentes Core e os respetivos Componentes Nativos que foram agrupados, tal como a respetiva tag HTML que se assemelha.

Tabela 2.4: Exemplo de ligação entre Componentes Core, Nativos e HTML

Core	**Android**	**iOS**	**Tag HTML**
<View>	<ViewGroup>	<UIView>	
<Text>	<TextView>	<UITextView>	<p>
<Image>	<ImageView>	<UIImageView>	<img>
<TextInput>	<EditText>	<UITextField>	<input type="text">

Visto ser uma tecnologia virada para as aplicações móveis, por vezes é necessário aceder a certas API's nativas que não estão disponíveis no JavaScript, tais como Apple Pay ou Google Pay. Para isso, foram desenvolvidos módulos que permitem a comunicação entre o JavaScript e a API desejada. Estes módulos designam-se de Native Modules (*Native Modules · React Native* 2022).

A comunicação entre o JavaScript e a linguagem nativa (Objective-C/Java) é realizada através de uma camada chamada *Bridge*. Esta camada é responsável pela conversão do código JavaScript para nativo - e o contrário - de forma a conseguir estabelecer uma ligação entre ambos (Câmara 2018).

A Bridge começa a atuar assim que a aplicação é inicializada e a UI Thread é executada. Posto isto, é criada a JS Thread, que é responsável pela renderização dos elementos de JavaScript. Visto que esta camada necessita de fazer diversas ações para proceder à conversão entre JavaScript e o código nativo e a posterior chamada da API desejada. Com isto, o desempenho da aplicação tende a diminuir, tendo já sido criados novos módulos, capazes de estabelecer a comunicação entre ambas as linguagens *bridge* (*Threading Model · React Native* 2022).

JavaScript Interface (JSI) Modules

O JSI Modules, doravante mencionado como JSI, é uma nova camada da arquitetura de React Native, que auxilia a comunicação entre o JavaScript, e a plataforma nativa, de modo a torná-la mais fácil e rápida (*React Native JSI* 2022).

Esta comunicação apresenta melhor desempenho, pois o JSI remove a necessidade da utilização da *bridge*, que estabelece a ligação entre o código nativo e o código de JavaScript. Também remove a necessidade de converter toda a informação , como um objeto JavaScript Object Notation (JSON) para uma *byte stream*, ou o contrário (*React Native JSI* 2022).

Posto isto, os grandes destaques do JSI são:

- Permite registar métodos que podem ser acedidos globalmente, no JavaScript (*React Native JSI* 2022).
- Esses métodos podem ser desenvolvidos em C++ ou podem ser formas de comunicação com Objective-C (iOS) e Java (Android) (*React Native JSI* 2022).
- Qualquer Módulo Nativo, que atualmente seja utilizado através da *bridge*, pode ser convertido num Módulo JSI (*React Native JSI* 2022).
- Para iOS a utilização é direta, pois o código em C++ pode correr diretamente em Objective-C. Já para Android, é necessária uma adaptação (*React Native JSI* 2022).
- Estes métodos podem ser síncronos (*React Native JSI* 2022).

Turbo Modules

O React Native utiliza diversos Módulos Nativos, como `Async Storage` e Informação do Dispositivo como forma do JavaScript invocar funções em Java ou Objective-C, sendo o Turbo Modules uma nova arquitetura desenvolvida para melhorar a rapidez com que esses módulos são inicializados e invocados pelo JavaScript (*TurboModules* 2022).

A razão para a introdução desta nova arquitetura advém de diversos problemas encontrados, quando do estudo do desempenho da tecnologia React Native. Tais como:

- "Atualmente, os Módulos Nativos são especificados num *package* e inicializados cedo". Posto isto, o tempo de inicialização do React Native aumenta com o número de Módulos Nativos, mesmo que alguns nunca sejam utilizados (*TurboModules* 2022).
- Não existe uma forma de verificar, se os Módulos Nativos, invocados pelo JavaScript, estão de facto a ser utilizados na aplicação (*TurboModules* 2022).
- O seu ciclo de vida está diretamente ligado ao ciclo de vida da *bridge*, podendo ter de ser recomeçado diversas vezes, em alguns casos (*TurboModules* 2022).
- "Durante o processo de inicialização, os Módulos Nativos são tipicamente especificados em diversos *packages*. Depois, iterados sobre uma lista de Módulos Nativos diversas vezes, até se conseguir fornecer as informações à *bridge*. Isto não precisa de acontecer num estado de execução (*runtime*)."(*TurboModules* 2022).
- Atualmente, as funções de um Módulo Nativo são calculadas durante o tempo de execução, algo que poderia ser executado no tempo de construção (*build time*), em que a função se chamaria a si própria e seria enviada para uma lista de mensagens (*TurboModules* 2022).

JavaScript

O JavaScript é uma linguagem de programação "leve e orientada a objetos com funções de primeira classe, conhecida como a linguagem de *scripting* para páginas *web*, mas também utilizada em muitos ambientes fora dos navegadores"(*Sobre JavaScript - JavaScript | MDN* 2022).

Esta linguagem permite executar *scripts*, quer do lado do *client*, que permite através do navegador e recorrendo a um mecanismo de comunicação assíncrona, alterar o conteúdo do documento exibido, quer do lado do servidor - recorrendo a ambientes como node.js. A sua sintaxe é semelhante tanto ao Java, como ao C++, tornando mais simples a sua aprendizagem (*Sobre JavaScript - JavaScript | MDN* 2022).

Apesar de orientada a objetos, também pode ser utilizada como uma linguagem de programação de processos. Ou seja, permite criar ações para serem posteriormente executadas, como uma rotina. No contexto de objetos, estes são criados, sem valor (vazios), e posteriormente sendo anexados por métodos e propriedades, tendo a possibilidade de ser utilizado diretamente, assim que construído (*Sobre JavaScript - JavaScript | MDN* 2022).

2.4.5 Análise comparativa entre as tecnologias

Nesta subsecção, é realizada uma análise das diversas tecnologias existentes, com a tecnologia utilizada, segundo alguns critérios previamente estabelecidos. Para estes critérios foram criadas diversas perguntas, que serão respondidas ao longo desta análise - sendo as mesmas apresentadas de seguida:

- **Q1**: Com que rapidez o carregamento e a interação com as ações do utilizador são efetuadas?
- **Q2**: Quanto tempo demora para a funcionalidade entrar em produção? Como se classifica o nível de dificuldade na realização de alterações ao produto? De que forma o código é modular e reutilizável? Qual é a documentação existente? Como funcionam estas tecnologias com certas ferramentas?
- **Q3**: Qual é o conhecimento dos programadores em relação à tecnologia e ao custo de recursos humanos associado?
- **Q4**: As funcionalidades, esperadas nos sistemas operativos/navegadores que são utilizados para aceder à aplicação, estão à disposição do utilizador?
- **Q5**: A tecnologia estará num processo de revolução das suas estruturas? Estarão a ser adicionadas novas funcionalidades à tecnologia, de forma a aprimorar as atuais? Qual é a adesão global do mercado?
- **Q6**: Esta tecnologia pode ser reutilizável para a futura utilização noutras plataformas?

Após a criteriosa escolha de perguntas, segue-se uma análise, baseada na investigação acima descrita, de cada uma das tecnologias (Swift, Flutter, Kotlin Multiplatform e React Native), de acordo com a pergunta em questão.

Q1: Rapidez e Desempenho

As aplicações nativas oferecem uma melhor experiência de utilização, com base na velocidade, em conformidade com a plataforma e a utilização das últimas funcionalidades. As

plataformas que desenvolvem iOS e Android em conjunto preocupam-se em utilizar uma linguagem de programação e um tempo de execução provenientes de uma tecnologia "não-móvel" – tendo como base a ideia de simplificação do custo e o desenvolvimento (Morgan 2019).

- **Swift**
  - É a linguagem oficial do desenvolvimento de aplicações de iOS nativas;
  - É a linguagem de programação onde se conseguem os melhores resultados, em termos de desempenho e rapidez de carregamento da aplicação.
- **Flutter**
  - Desempenho semelhante às aplicações nativas;
  - O desempenho em tempo de execução é muito rápido e resolve o problema do atraso que continua a existir nas soluções desenvolvidas em JavaScript (Ionic/-Cordova ou ReactNative) (Morgan 2019).
- **Kotlin Multiplatform**
  - Compila para nativo, por isso o desempenho e a rapidez permanecem iguais.
- **React Native**
  - Por defeito, possui um pior desempenho que as aplicações nativas;
  - Poderá ter um desempenho semelhante às aplicações nativas, mas é necessário um conhecimento aprofundado dos programadores;
  - Qualquer comunicação entre a parte nativa e o desenvolvimento em JavaScript é completado através de uma *bridge*. Isto conduz a uma maior complexidade, em termos de integração, e por consequência, a uma diminuição de desempenho (Rempel 2019).

Q2: Manutenibilidade e facilidade geral de utilização

- **Swift**
  - Frameworks e documentação oficial da Apple;
  - Grande comunidade, que partilha a resolução de problemas online;
  - Pode ser desenvolvida por módulos e integrá-los depois com facilidade;
  - É compatível com todas as ferramentas de testes automáticos e de envios para produção.
- **Flutter**
  - A rapidez de entrega não se revela tão positiva, visto que as ações como o *debugging* não são tão intuitivas.
  - Os erros e *logs* que aparecem no ecrã podem ser significativamente confusos, pois apontam para linhas de código, que podem estar num nível de abstração bastante abaixo daquilo que está a interagir. Normalmente, nas aplicações nativas, os erros são mais claros para se compreender, e em casos específicos, já existem diversas informações na Internet sobre a resolução e a razão dos mesmos (Bellinaso 2020).

- **Kotlin Multiplatform**
  - A rapidez de entrega é favorável, especialmente porque permite partilhar o código entre plataformas.
- **React Native**
  - A rapidez de entrega não é favorável, porque são necessários recursos humanos qualificados, tanto de JavaScript, como de aplicações nativas;
  - Uma vez que permite a partilha de código entre *web* e as aplicações móveis, é um ponto que torna a sua rapidez de entrega mais eficiente;
  - Como ainda está em desenvolvimento, é necessário verificar as atualizações que vão surgindo com bastante frequência (*When React Native is not a Good Choice for a Mobile Application Development* 2022);
  - Existem algumas incompatibilidades entre algumas ferramentas e a tecnologia, de forma a se conseguir testar a aplicação com segurança (*Flutter Just Might Work* 2019).

Q3: Conhecimento da tecnologia

De acordo com o ranking de popularidade de linguagens de programação do GitHub, pode observar-se, na Tabela 2.5 que JavaScript continua a ser a tecnologia mais utilizada pelos programadores.

Tabela 2.5: Github Ranking popularidade (*Github Language Stats* 2022)

Ranking	Tecnologia	Percentagem
3	JavaScript (React-Native)	10.978%
13	Kotlin	1.151%
16	Swift	0.806%
26	Dart (Flutter)	0.220%

- **Swift**
  - A linguagem de programação mais popular, para o desenvolvimento de aplicações iOS;
  - Lançada em 2014, possui quase 8 anos de existência.
- **Flutter**
  - Baseada numa linguagem que não é tão comum no mercado tecnológico - Dart;
  - Lançada em 2017, ainda existe pouco conhecimento para a maioria dos programadores.
- **Kotlin Multiplatform**
  - Linguagem de programação bastante conhecida, num meio de programação Android.
- **React Native**

  - Desenvolvida em JavaScript, constitui-se como a linguagem de programação mais popular do mundo;
  - Requer algum conhecimento sobre aplicações nativas de iOS e Android.

Q4: Suporte da aplicação em diversas plataformas

- **Swift**
  - Cada nova funcionalidade lançada para dispositivos iOS é lançada, tendo em conta o seu acesso em Swift;
  - Tal como referido anteriormente, é a linguagem de programação oficial da Apple.
- **Flutter**
  - Devido ao seu carácter recente, podem existir alguns problemas ao suportar bibliotecas externas;
  - "95% das funcionalidades que precisei de utilizar estavam lá e disponíveis, apenas uma excepção, alguma integração de terceiros com uma ferramenta de análise popular, mas nada que um simples invólucro HTTP não conseguisse resolver."(Manning 2022).
- **Kotlin Multiplatform**
  - Podem existir alguns problemas ao suportar bibliotecas externas. Mas, neste caso, isso pode ser solucionado com a utilização de bibliotecas nativas.
- **React Native**
  - Em teoria, poderá ter-se acesso a todas as funcionalidades nativas usando React Native. Em alguns casos, será necessário serem construídas de raíz, em Swift (iOS) ou Java (Android), o que diverge da solução base de JavaScript ser integral;
  - "Desenvolvemos o React Native de modo a que seja possível escrever um verdadeiro código nativo e ter acesso a toda a potência da plataforma. Esta é uma característica mais avançada e não esperamos que faça parte do processo habitual de desenvolvimento, no entanto, é essencial que exista. Se o React Native não suportar uma funcionalidade nativa de que necessita, deverá ser capaz de a construir você mesmo."(*Native Modules · React Native* 2022).

Q5: Estabilidade, maturidade, atividade e futuro da tecnologia

De acordo com o ranking de tendências do StackOverflow, observável na Figura 2.11, tanto o Flutter, como React Native e Kotlin estão a surgir mais vezes no StackOverflow, o que indica uma maior utilização, mas também uma menor estabilidade e maturidade, em comparação com o Swift, que já é uma tecnologia bastante estável.

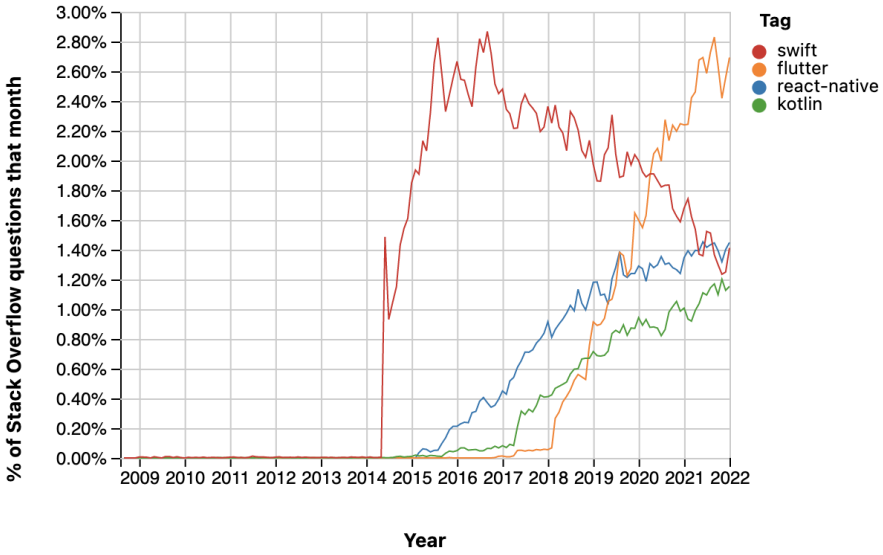

Figura 2.11: Gráfico de tendências do StackOverflow (*Stack Overflow Trends* 2022)

- **Swift**
  - Lançado em 2014. Quase 8 anos de existência e uma enorme adoção pela comunidade tecnológica;
  - Maioria das aplicações iOS são desenvolvidas em Swift.
- **Flutter**
  - Demasiada instabilidade no processo de desenvolvimento;
  - Uma nova versão todas as semanas, praticamente.
- **Kotlin Multiplatform**
  - Ainda não é uma tecnologia estável, pois a maioria dos projetos multiplataforma do Kotlin, ainda estão numa de duas fases de processo de desenvolvimento existente: MF e AIR.
- **React Native**
  - Tem uma adoção relativamente favorável, mas ainda em crescimento;
  - Algumas empresas recuaram na sua utilização (como a Airbnb), mas a comunidade acredita que o seu futuro não está comprometido;
  - Está ligado ao Facebook.

Q6: Oportunidade para desenvolver em conjunto com iOS, Android e web

Tabela 2.6: Oportunidade para desenvolver em conjunto com iOS, Android e *web*

Tecnologia	Web	Android	iOS
Swift	Não	Não	Sim
Flutter	Sim	Sim	Sim
Kotlin Multiplatform	Sim	Sim	Sim
React Native*	Sim	Sim	Sim

*Caso *web* seja desenvolvido em JavaScript, existe a possibilidade de ser partilhada parte do código. Mesmo assim, certas partes necessitam de ser desenvolvidas nas suas linguagens nativas.

Análise

Após uma análise criteriosa às tecnologias estudadas, baseando-se nas questões previamente estabelecidas, é possível afirmar que as tecnologias com compiladores nativos (Swift, Kotlin Multiplaform e Flutter) têm vantagem, em detrimento da tecnologia React Native, respondendo assim à **Q1**.

Em relação à **Q2**, a tecnologia Kotlin Multiplatform e o Swift são as mais capazes para cumprir tais critérios. Neste caso, a tecnologia Flutter é a menos aconselhada, muito por conta da dificuldade de se entender possíveis erros de compilação que surjam, mas também pela ação de *debugging* não ser tão intuitiva.

Analisando a **Q3**, a tecnologia React Native tem um enorme destaque, visto utilizar a linguagem de programação com maior popularidade (JavaScript). A tecnologia Flutter - utiliza a linguagem Dart - e o Swift são as menos aconselhadas, visto que são as menos utilizadas no seio da comunidade tecnológica, em comparação com JavaScript e Kotlin.

No que diz respeito à **Q4**, a tecnologia Swift é a mais capaz, visto ser uma tecnologia já bastante estável e com uma quantidade de aplicações desenvolvidas bastante respeitável. Neste aspeto, e visto serem todas tecnologias relativamente recentes, React Native, Kotlin Multiplatform e Flutter são as menos aconselhadas, segundo este critério.

Escrutinando a **Q5**, é possível declarar a tecnologia Swift como a mais estável, mas destacando a tecnologia React Native, que apesar de ainda estar em crescimento, já tem uma maturidade e atividade bastante razoável. Seguindo as últimas análises, a tecnologia Flutter é a menos aconselhada, uma vez que a sua instabilidade e pouco conhecimento sobre a linguagem Dart, provocam uma interrogação quanto ao seu futuro.

Por fim, analisando a **Q6**, é possível verificar na Tabela 2.6 que as tecnologias multiplataforma (Flutter, Kotlin Multiplatform e React Native) permitem a partilha de código entre plataformas, mesmo com algumas nuances de utilização de linguagem nativa. Contrariando, a tecnologia Swift, atualmente, só permite o desenvolvimento de aplicações iOS.

Em suma, analisando todas as questões, é admissível reiterar a tecnologia React Native como a mais equilibrada. Apesar de não ter um destaque grande, na maioria das questões, revela-se como a tecnologia com a melhor média de aconselhamento, seguida da tecnologia

Kotlin Multiplatform. Contudo, a escolha da tecnologia recaíria sempre em React Native, pois foi a tecnologia escolhida pela empresa, devido ao seu conhecimento, por parte dos seus colaboradores.

2.5 Sumário

Neste capítulo foi efetuado o Estado da Arte deste trabalho, realçando o contexto e o problema, em que este trabalho se insere; os objetivos a cumprir ,e também uma investigação e análise de diversas tecnologias que são utilizadas para o desenvolvimento de aplicações móveis nativas (ou quase nativas, como o exemplo do React Native).

Como tal, foram efetuadas algumas perguntas, de forma a se perceber qual a tecnologia mais capaz, segundo alguns critérios estabelecidos, concluindo-se que a tecnologia React Native, apesar de não se destacar em nenhum dos critérios, foi considerada a mais equilibrada de todas. Esta tecnologia também tem a seu favor o número de utilizadores que conhecem a linguagem em que é desenvolvida, neste caso JavaScript, o que facilita quando é necessário escolher uma tecnologia para a criação de um projeto.

Capítulo 3

Análise de Valor

Neste capítulo, apresenta-se uma análise de valor ao projeto descrito neste documento. A mesma encontra-se direcionada para o valor que provém da contextualização do problema, referido no Capítulo 2.

3.1 Descrição

Uma análise de valor caracteriza-se por ser um processo de análise e avaliação organizado, formal e sistemático. É considerada uma atividade de gestão, que requer planeamento, controlo e coordenação, com o objetivo de aumentar o valor de um produto ou serviço, com o menor custo possível, sem diminuir a sua qualidade (Nicola 2020a).

Esta análise diz respeito à necessidade que um produto ou aplicação possa ter para um cliente. Para tal, é necessário entender o seu propósito, que implica as especificações que possam ser estabelecidas pelo cliente entre o produto e o seu valor, de forma a existir um certo ajuste de ambos os elementos (Nicola 2020a).

Para tal, e com o objetivo de oferecer um benefício à empresa, deve ser efetuado um processo de melhoria de *design*, que consiste na diminuição dos custos do mesmo, mas tendo em conta a sua qualidade. Posto isto, a melhoria de um produto deve compreender três elementos: a sua utilização, o valor estimado e o valor de mercado (Nicola 2020a).

Esse processo pode ser considerado como inovação, o que pode acarretar riscos ao nível económico e de qualidade do produto, para a empresa em questão. Posto isto, de forma a minimizar esses mesmos riscos, pode ser desenvolvido um processo de inovação, que se divide em três fases, como é retratado na Figura 3.1 (Nicola 2020a).

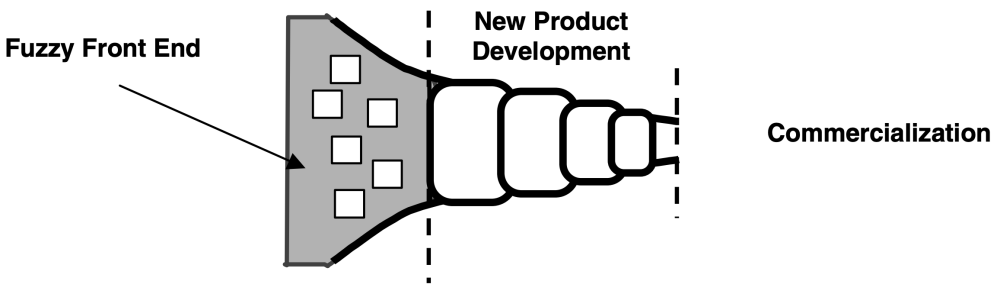

Figura 3.1: Esquema Processo Inovação (Nicola 2020a)

- **Fuzzy FrontEnd (FFE)**
  Fase responsável pela experimentação de ideias, sendo bastante instável, pois numa fase inicial podem surgir diversos projetos/ideias em planeamento, enquanto outros podem precisar de financiamento para continuar (Koen et al. 2002).

  Nesta fase existe um grau de especulação e incerteza elevados, tanto quanto à data de comercialização do produto, como das expectativas de retorno (por exemplo, financeiro). Para tal, são efetuadas diversas pesquisas, tendo em vista a minimização dos riscos associados e da potencial otimização do produto (Koen et al. 2002).

- **New Product Development (NPD)**
  Esta segunda fase, do processo de inovação, é caracterizada por existir um grau de certeza elevado, já com orçamentos definidos e com expectativas de retorno mais previsíveis. Posto isto, nesta fase já se desenvolve o produto, resultando num plano disciplinado e orientado a um objetivo, com um projeto definido (Koen et al. 2002).

- **Comercialização**: A última fase diz respeito à venda e divulgação do produto.

Com o objetivo de apresentar os diversos estágios do processo de inovação - desde o problema à seleção da ideia - será abordado detalhadademente o Modelo NCD, que integra uma parte importante do FFE.

3.2 New Concept Development (NCD)

O Modelo NCD, representado na Figura 3.2, permite uma compreensão e definição dos padrões chave do FFE. Este modelo é caracterizado por começar na Identificação de Oportunidades, bem como na Geração de Ideias, terminando na execução do processo NPD ou Technology Stage Gate (TSG). Posto isto, pode-se dividir em três partes essenciais:

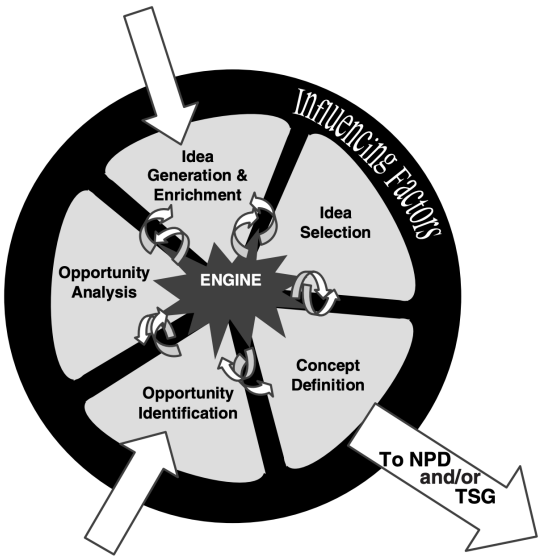

Figura 3.2: Esquema Modelo NCD (Koen et al. 2002)

- **Motor**: O centro do modelo, que consiste na liderança e gestão da estratégia de negócio da organização, responsável por conduzir os elementos de atividade controlável pela empresa (Koen et al. 2002).

- **Elementos de Atividade Controlável**: São caracterizados por cinco elementos diferentes, tais como Identificação de Oportunidades, Análise de Oportunidade, Geração de Ideias, Seleção de Ideias e Definição Conceptual (Koen et al. 2002).
- **Fatores Externos**: Compreendem diversos fatores, não controláveis pela empresa, que possam afetar todo o processo de inovação, tais como: fatores ambientais, fatores organizacionais (políticos, leis, económicos, canais de distribuição, entre outros) e fatores baseados na ciência (Koen et al. 2002).

3.2.1 Elementos de Atividade Controlável

Uma das partes do modelo NCD denomina-se Elementos de Atividade Controlável, que se subdivide em cinco elementos, como mencionado anteriormente (Identificação de Oportunidades, Análise de Oportunidade, Geração de Ideias, Seleção de Ideias e Definição Conceptual). Para melhor enquadrar o leitor, segue-se uma análise de cada elemento pertencente a esta parte:

- **Identificação de Oportunidades**

  O mercado tecnológico relativo às aplicações móveis encontra-se em elevado crescimento, assistindo-se à criação de novas tecnologias capazes de promover uma melhor e mais atrativa utilização de aplicações, em dispositivos móveis. Este crescimento deve-se, em parte, à evolução tecnológica no geral, que permitiu a possibilidade de utilização de recursos numa escala maior que outrora.

  Mas com a evolução da tecnologia, não foi só o mercado de aplicações móveis que se expandiu. Atualmente, o mercado de apostas desportivas está em franca expansão, tanto a nível social, como económico. Só no último ano, em Portugal, as casas de apostas geraram uma receita duas vezes superior, em comparação com o mesmo período no ano anterior, como se pode comprovar na Figura 3.3.

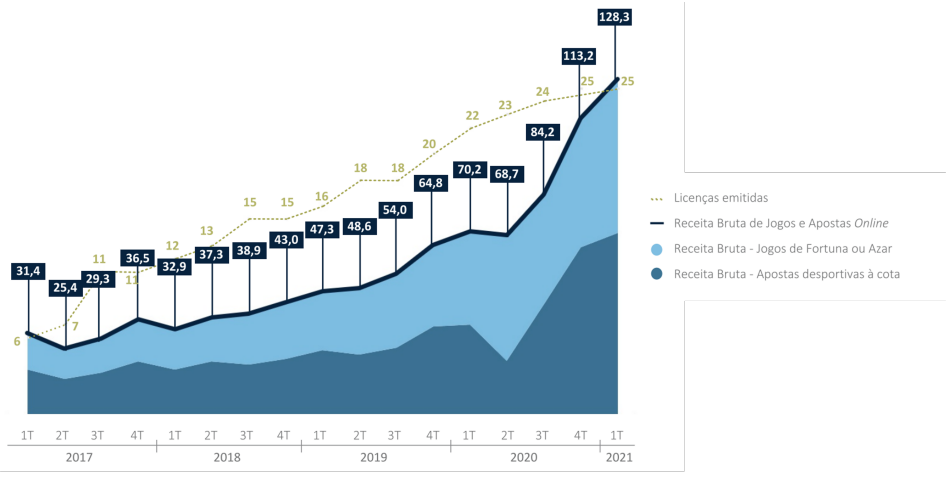

Figura 3.3: Gráfico Receitas Apostas, em Portugal (Milhões €) (SRIJ 2022)

Como mencionado anteriormente, esta expansão também provocou efeitos a nível social, pois registou-se, no último ano, um aumento de quase 400% em transações de apostas, vísível na Figura 3.4, em relação ao mesmo período no ano anterior.

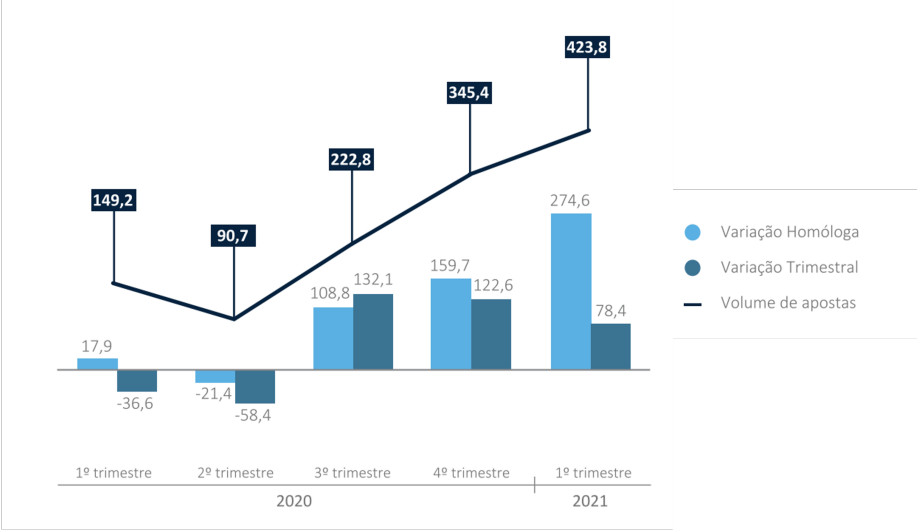

Figura 3.4: Gráfico Volume Apostas, em Portugal (Milhões €) (SRIJ 2022)

Por fim, de forma a sustentar ainda mais o crescimento sócioeconómico, tornando-se uma oportunidade bastante atrativa para se arriscar, é demonstrado na Figura 3.5, o crescimento do número de jogadores registados em casas de apostas, em Portugal, no último ano.

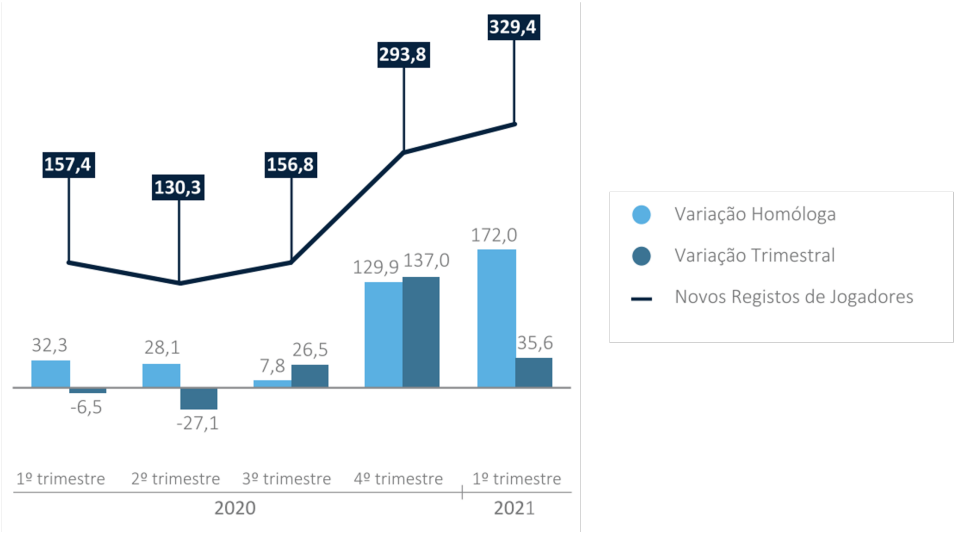

Figura 3.5: Gráfico Jogadores Registados em Casa de Apostas, em Portugal (Milhares) (SRIJ 2022)

- **Análise de Oportunidade**

Para suportar a oportunidade identificada acima, e como comprovado, recorreu-se à análise de dados estatísticos do mercado de apostas, nomeadamente as receitas obtidas, o volume de apostas e o registo de novos jogadores no último ano, em Portugal. O crescimento deste mercado tem implicação indireta no tema deste documento, pois a necessidade de melhorar o desempenho da aplicação (atualmente em desenvolvimento)

pode resultar num aproveitamento da expansão desta área de negócio e também reter os seus atuais consumidores/clientes.

- **Geração de Ideias**
  Após a realização da Análise de Oportunidade ao problema contextualizado, foram geradas algumas ideias, com o intuito de acrescentar valor à oportunidade identificada. A ideia principal é a criação de uma aplicação que esteja disponível tanto para dispositivos móveis (iOS e Android), como para navegadores web, acedidos por um desses dispositivos. De forma a expandir o seu negócio, e como já foi referido anteriormente, surgiu a ideia de se desenvolver uma aplicação multiplataforma, utilizando uma tecnologia com capacidade para esse efeito - particularmente, React Native. No entanto, com o objetivo de não perder a qualidade das aplicações nativas existentes na empresa, foi necessária a realização de um estudo, implicado neste documento.

  Esse estudo deve conceder uma resposta ao problema referido na Secção 2.2, incluindo uma comparação entre o desempenho da atual aplicação, com o desempenho da aplicação com a implementação da nova solução.

  Posto isto, tendo em conta a decisão de se estabelecer o foco na análise de valor às tecnologias multiplataforma, surgiram três tecnologias possíveis para o desenvolvimento do projeto.

  - React Native
  - Flutter
  - Kotlin Multiplatform

- **Seleção de Ideias**
  Em sequência da Geração de Ideias, e com base nos estudos efetuados ao longo do documento, decidiu-se realizar o estudo incidindo sobre a tecnologia React Native, uma vez que foi a escolhida pela empresa para o desenvolvimento do projeto.

- **Definição Conceptual**
  O objetivo do projeto consiste no desenvolvimento de uma ferramenta, capaz de melhorar o desempenho de uma aplicação, aquando da presença de listas encadeadas, desenvolvida com a tecnologia React Native.

3.2.2 Fatores Externos

Os fatores externos foram analisados ao nível económico, social e tecnológico, tendo em conta o discriminado na Identificação de Oportunidades:

- **Fatores Económicos**: A área de negócio, em que o projeto da empresa se enquadra, está em plena expansão, tanto em Portugal, como a nível global, existindo lucros acima dos 100% (SRIJ 2022).
- **Fatores Sociais**: A par do fator económico, o mercado de apostas tem vindo a expandir-se socialmente. Ou seja, nos últimos anos, assistiu-se a um crescente número de clientes deste tipo de mercado (SRIJ 2022).
- **Fatores Ambientais**: A pandemia do vírus SARS-CoV-2 (Covid-19) provocou a abertura de novos mercados, tal como a expansão de outros, que anteriormente não possuíam um impacto tão significativo na sociedade. Um desses mercados foi a expansão das apostas desportivas e dos jogos online.

- **Fatores Tecnológicos**: Nos últimos anos, tem existindo uma evolução massiva da tecnologia, sendo cada vez mais fácil a criação de um produto tecnológico. Com o crescimento das tecnologias para desenvolvimento de *software* multiplataforma, a empresa decidiu criar este novo projeto.

Em relação aos fatores económicos, o mercado onde a empresa se encontra está em crescimento, tanto em território nacional, como ao nível global. Nos fatores sociais pode-se considerar que o estilo de vida tem mudado graças ao avanço tecnológico, quer em termos de educação, quer em termos laborais. No que toca aos fatores tecnológicos, mais precisamente na área móvel, o seu desenvolvimento e respetivo crescimento consistem num dos grandes fatores, como foi evidenciado anteriormente.

3.3 Valor, Valor Percecionado e Valor para o Utilizador

Nesta secção, será analisada a definição de valor e a quem se destina. Além disso, será analisado o valor percecionado, suportado por uma Tabela de Benefícios e Sacrifícios. Por fim, será analisado o valor para o utilizador.

3.3.1 Valor

A definição de valor pode ser considerada como algo impreciso, pois depende de um contexto e de um utilizador. Isto é, algo que possa ter valor para um certo utilizador, num determinado contexto, pode ser algo insignificante num outro contexto, para outro utilizador (Nicola 2020a).

Por exemplo, o desempenho de uma aplicação tem um valor considerável para uma área de negócio de apostas desportivas, mas para uma simples página *web* estática revela-se insignificante.

Segundo (Nicola 2020a), "o valor é uma quantidade, que aumenta a satisfação do cliente ou corta a despesa atribuível ao produto". Posto isto, no caso deste documento, o produto final será efetuado tanto para o projeto da Blip, como uma ferramenta *open-source* - possível de ser acedida por qualquer programador. Este produto irá representar um valor para a aplicação em desenvolvimento pela empresa, mas também um valor para a tecnologia React Native, uma vez que pretende melhorar o seu desempenho - quando da utilização de listas encadeadas. Para tal, tentar-se-á alcançar o valor através da implementação de uma nova solução, capaz de lidar com listas encadeadas com um melhor desempenho e fluidez de movimento.

3.3.2 Valor Percecionado

O valor percecionado varia de cliente para cliente, visto que "diferentes clientes vão interpretar o valor de um produto de formas diferentes"(Nicola 2020a).

Para este documento, o valor percecionado revela-se alto para o cliente, pois a empresa poderá atrair novos clientes, com uma aplicação mais atrativa em termos de desempenho e fluidez de movimento. Da mesma forma que se apresenta alto para um cliente, enquanto utilizador geral - uma vez que se trata de uma ferramenta que será *open source* - poderá acrescentar valor para outras aplicações.

Esta análise de valor percecionado é fundamentada com a Tabela 3.1, através da análise dos benefícios e sacrifícios do ponto de vista do cliente.

Tabela 3.1: Benefícios e Sacrifícios

	Produto	**Serviço**	**Relação**
Benefícios	Solução Alternativa Qualidade Produto	Responsividade Flexibilidade	Confiança Imagem
Sacrifícios	Custo		tempo/esforço

3.3.3 Valor para o Cliente/Utilizador

O valor para o cliente/utilizador pode ser definido pela comparação e análise entre os benefícios e os sacrifícios para o desenvolvimento desta nova solução. Como pode ser usada como uma fórmula aritmética - observável na equação 3.1 - esta calcula o valor do produto, baseando-se nos benefícios e sacrifícios.

$$\chi valor = \alpha benef cios - \beta sacrif cios \tag{3.1}$$

3.4 Proposta de Valor para o projeto

A solução apresentada neste documento procura solucionar o problema apresentado na secção 2.2, cujo objetivo é melhorar o desempenho da aplicação do cliente, quando se utilizam listas encadeadas.

Esta solução criará um valor para o projeto desenvolvido pela empresa, pois existindo um melhor desempenho (assumindo um caso ideal) trará novos clientes, e por consequência, mais receitas. Como se constitui como uma solução *open-source*, também criará valor para qualquer aplicação/projeto que a utilize, uma vez que a sua utilização significa uma procura de desempenho melhorado.

Estas consequências serão abordadas e analisadas na secção seguinte.

3.5 Modelo de Negócio Canvas

Este modelo de negócio, representado na Figura 3.6, é constituído pelos seguintes componentes.

3.5.1 Parcerias Chave

A Blip, enquanto empresa de desenvolvimento de software, e a Betfair, enquanto cliente da empresa, serão as parcerias chave.

3.5.2 Atividades Chave

A atividade chave será a comparação do atual desempenho e fluidez de movimento da aplicação, com o desempenho e fluidez de movimento da aplicação, com a nova solução.

3.5.3 Recursos Chave

O recurso chave deste modelo de negócio é a aplicação já desenvolvida pela empresa.

3.5.4 Estrutura de Custos

Os únicos custos associados a este negócio são os custos com cursos de formação em linguagens móveis nativas, como *Java* e *Objective-C*.

3.5.5 Proposta de Valor

A proposta de valor para o projeto é um melhor desempenho da aplicação, na presença de listas encadeadas.

3.5.6 Relação com os Clientes

A relação com os clientes será direta, tanto com a empresa Blip, como o seu cliente, a Betfair.

3.5.7 Canais

Os canais de comunicação serão reuniões com a empresa e a equipa envolvida no projeto, tal como correspondência eletrónica que seja necessária, com o cliente Betfair.

3.5.8 Segmentos de Mercado

O segmento de mercado é a empresa em questão e qualquer outra aplicação/projeto que necessite de um aprimoramento a nível de desempenho, por ser uma ferramenta *open-source*.

3.5.9 Fontes de Rendimento

Como se constitui um serviço prestado para a empresa, e posteriormente, disponibilizado gratuitamente no mercado tecnológico, não existe qualquer fonte direta de rendimento. A única fonte de rendimento, assumindo que trará o sucesso desejado à empresa, será a angariação de novos clientes e, por consequência, o aumento das receitas.

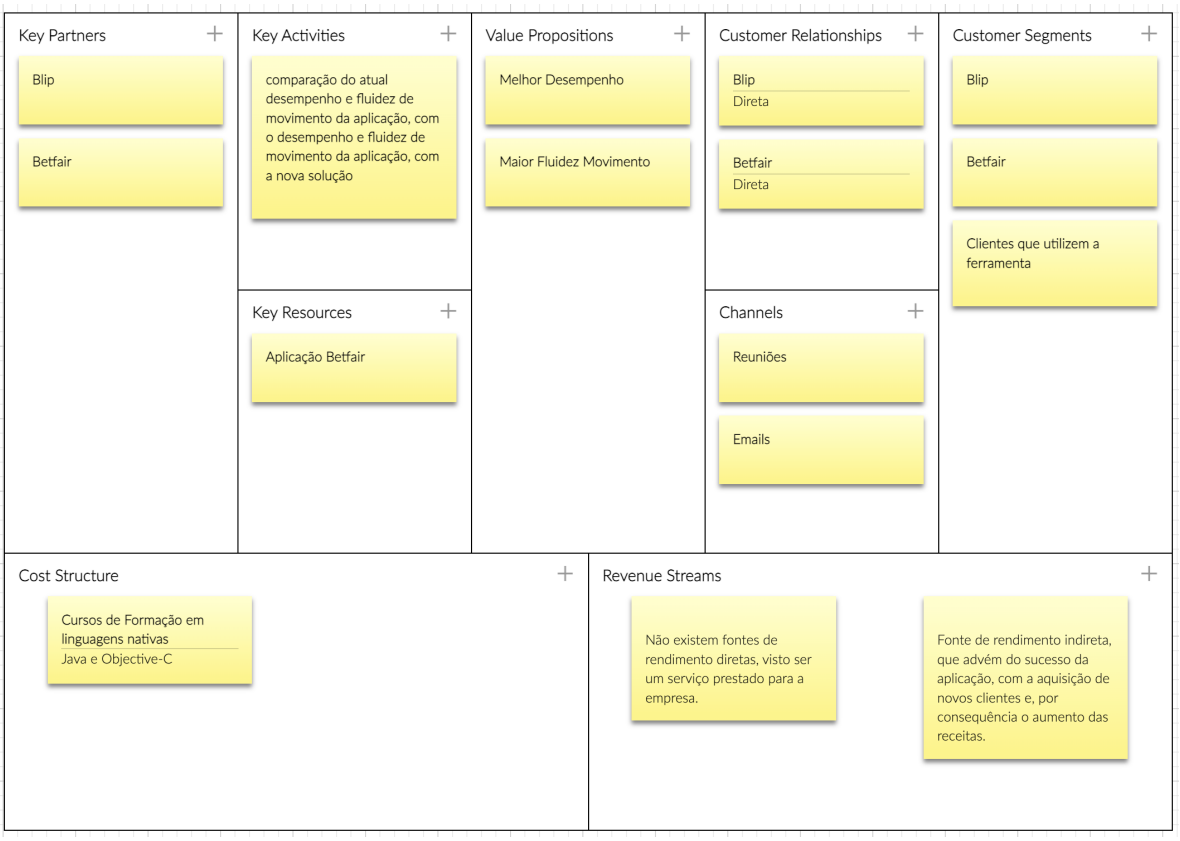

Figura 3.6: Modelo de Negócio Canvas

3.6 Analytic Hierarchy Process (AHP)

Nesta secção será realizado o procedimento de decisão, utilizando as tecnologias existentes para o desenvolvimento de uma aplicação multiplataforma. Este aspeto justifica-se uma vez que a solução planeada para o objetivo deste documento, não é adequada para uma análise de valor, com o método AHP, por falta de soluções alternativas.

O Método AHP, criado por Thomas Saaty, permite o uso de critérios qualitativos, bem como quantitativos no processo de avaliação, para o auxílio na tomada de uma decisão complexa. O propósito deste método é facilitar a compreensão e a avaliação de um problema de decisão, dividindo o mesmo em níveis hierárquicos (Nicola 2020b).

Com o intuito de facilitar a avaliação e a compreensão, a Figura 3.7, representa a divisão, por níveis hierárquicos, do problema de decisão. Esta divisão é composta por quatro partes:

- **Problema**: que consiste na exposição do problema a considerar.
- **Fatores**: que consistem em critérios importantes para a tomada de decisão.
- **Decisão**: que consiste na comparação, par a par, dos fatores, destacando o seu nível de importância.
- **Alternativas**: que consiste nas diferentes alternativas possíveis para a resolução do problema.

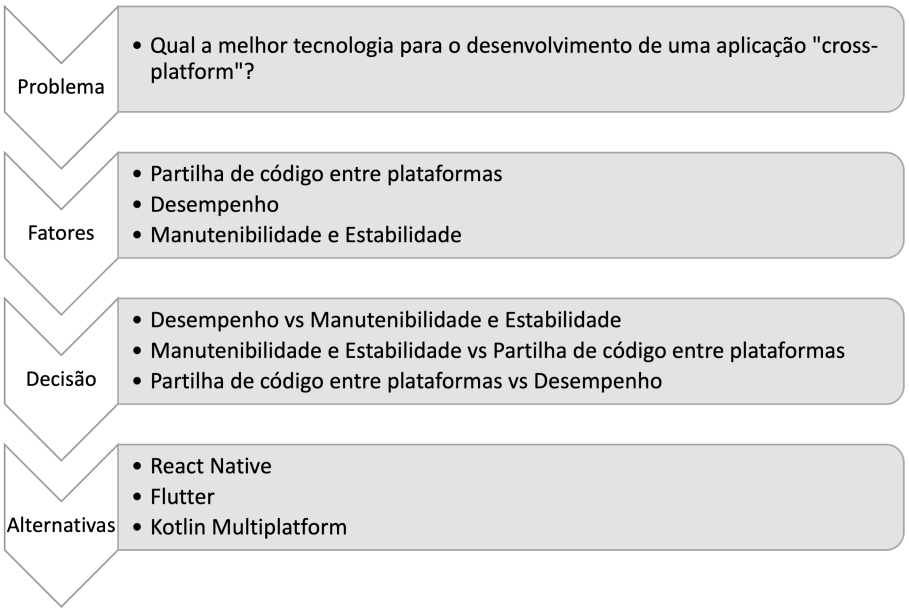

Figura 3.7: Divisão Hierárquica

Segundo uma análise efetuada, na subsecção 2.4.5, foi elaborada a Tabela 3.2, que identifica qual a melhor tecnologia para um determinado fator, numa avaliação numa escala de 1-5, em que 1 corresponde a uma fraca capacidade e 5 a uma ótima capacidade.

Tabela 3.2: Tabela comparação de pares

Tecnologia	**Partilha de código entre plataformas**	**Desempenho**	**Manutenibilidade e Estabilidade**
React Native	4	4	4
Kotlin Multiplatform	4	5	3
Flutter	4	5	2

Após a identificação dos valores das alternativas foi realizada uma análise dos níveis de importância de um fator, em relação a outro, para um determinado problema, sendo estes avaliados numa escala de 1-9, o que é observável na Tabela 3.3 (Saaty 1990).

Tabela 3.3: Escala fundamental - Níveis de importância de comparações

Nível Importância	Definição	Explicação
1	Igual importância	As duas atividades contribuem igualmente para o objetivo.
3	Fraca importância	A experiência e o julgamento favorecem levemente uma atividade em relação à outra.
5	Forte importância	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra.
7	Muito forte importância	Uma atividade é muito fortemente favorecida em relação a outra.
9	Importância absoluta	A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza.
2,4,6,8	Valores intermediários	Quando se procura uma condição de compromisso entre duas definições.

Posteriormente, através de uma organização hierárquica do problema, da consideração de diversos fatores e da exposição de inúmeras alternativas, foi realizada uma árvore hierárquica de decisão, observável na Figura 3.8, com o intuito de se proceder a uma melhor compreensão do problema e da decisão a tomar.

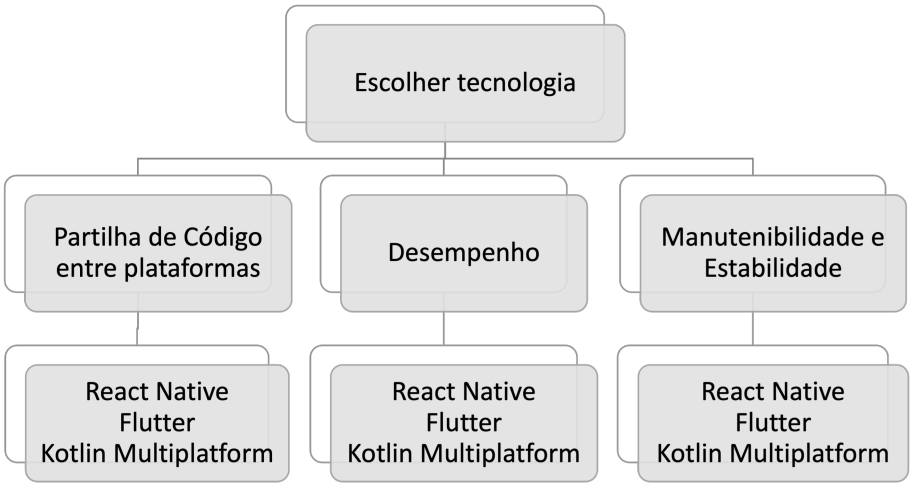

Figura 3.8: Árvore hierárquica de decisão

Tendo em conta a escala dos níveis de importância, definiu-se a seguinte Matriz de Comparação Par a Par, Equação 3.2, visível de seguida.

(As figuras representadas com "P", "D"e "M", são letras que significam "Partilha de Código Entre Plataformas", "Desempenho"e "Manutenibilidade e Estabilidade", respetivamente.)

$$MPP = \begin{array}{c} \begin{matrix} P & D & M \end{matrix} \\ \begin{bmatrix} 1 & 3 & 7 \\ \frac{1}{3} & 1 & 5 \\ \frac{1}{7} & \frac{1}{5} & 1 \end{bmatrix} \begin{matrix} P \\ D \\ M \end{matrix} \\ \begin{matrix} 1.47 & 4.33 & 13 & Soma \end{matrix} \end{array} = \begin{array}{c} \begin{matrix} P & D & M \end{matrix} \\ \begin{bmatrix} 1 & 3 & 7 \\ 0.33 & 1 & 5 \\ 0.14 & 0.20 & 1 \end{bmatrix} \begin{matrix} P \\ D \\ M \end{matrix} \\ \begin{matrix} 1.47 & 4.33 & 13 & Soma \end{matrix} \end{array} \tag{3.2}$$

Através da Matriz, Equação 3.2, foi possível calcular a soma de cada coluna. Posteriormente, após a divisão de cada elemento, pelo valor da soma da coluna correspondente, obteve-se a Matriz de Comparação Normalizada, ilustrada abaixo, Equação 3.3.

$$\begin{array}{c} \begin{matrix} P & D & M \end{matrix} \\ \begin{bmatrix} \frac{1}{1.47} & \frac{3}{4.20} & \frac{7}{13} \\ \frac{0.33}{1.47} & \frac{1}{4.20} & \frac{5}{13} \\ \frac{0.14}{1.47} & \frac{0.20}{4.20} & \frac{1}{13} \end{bmatrix} \begin{matrix} P \\ D \\ M \end{matrix} \\ \begin{matrix} 1.47 & 4.33 & 13 & Soma \end{matrix} \end{array} = \begin{array}{c} \begin{matrix} P & D & M \end{matrix} \\ \begin{bmatrix} 0.68 & 0.71 & 0.54 \\ 0.22 & 0.24 & 0.38 \\ 0.10 & 0.05 & 0.08 \end{bmatrix} \begin{matrix} P \\ D \\ M \end{matrix} \\ \begin{matrix} 1.47 & 4.33 & 13 & Soma \end{matrix} \end{array} \tag{3.3}$$

Após a normalização da Matriz, já é possível saber o Vetor de Prioridade, que indica a prioridade relativa ou peso de cada critério, definido na Equação 3.4. Este vetor é criado através da soma de cada linha da Matriz representada na Equação 3.3.

$$\begin{array}{c} \begin{matrix} P & D & M \end{matrix} \\ \begin{bmatrix} 0.68 & 0.71 & 0.54 \\ 0.22 & 0.24 & 0.38 \\ 0.10 & 0.05 & 0.08 \end{bmatrix} \begin{matrix} P \\ D \\ M \end{matrix} \\ \begin{matrix} 1.47 & 4.33 & 13 & Soma \end{matrix} \end{array} = \begin{bmatrix} 0,64 \\ 0.28 \\ 0.07 \end{bmatrix} \tag{3.4}$$

Após o cálculo da Matriz anteriormente apresentada, procedeu-se ao cálculo do teste de consistência, que engloba a multiplicação de cada elemento de uma coluna, pelo elemento da posição respetiva, da Matriz inicial - não normalizada - verificando-se o resultado na Equação 3.5.

$$\begin{array}{c} \begin{matrix} P & D & M \end{matrix} \\ \begin{bmatrix} 0.64 & 0.85 & 0.52 \\ 0.21 & 0.28 & 0.37 \\ 0.09 & 0.06 & 0.07 \end{bmatrix} \begin{matrix} P \\ D \\ M \end{matrix} \end{array} \tag{3.5}$$

Posteriormente, foi necessário somar cada linha da nova Matriz, sendo definido um novo Vetor de Prioridade, como observável na Equação 3.6.

$$\begin{array}{c} \begin{matrix} P & D & M \end{matrix} \\ \begin{bmatrix} 0.64 & 0.85 & 0.52 \\ 0.21 & 0.28 & 0.37 \\ 0.09 & 0.06 & 0.07 \end{bmatrix} \begin{matrix} P \\ D \\ M \end{matrix} \end{array} = \begin{bmatrix} 2.01 \\ 0.86 \\ 0.22 \end{bmatrix} \tag{3.6}$$

Posto isto, procedeu-se à divisão de cada elemento, do segundo Vetor de Prioridades, pelo elemento na posição correspondente, no primeiro Vetor de Prioridades, obtendo-se um último Vetor de Prioridades, visível na Equação 3.7 cujo valor próprio máximo - média aritmética dos valores do vetor - é utilizado para se calcular a Razão de Consistência (RC).

$$\begin{bmatrix} 2.01 \\ 0.86 \\ 0.22 \end{bmatrix} / \begin{bmatrix} 0.64 \\ 0.28 \\ 0.07 \end{bmatrix} = \begin{bmatrix} 3.12 \\ 3.06 \\ 3.01 \end{bmatrix} \tag{3.7}$$

Por fim, para se calcular a RC, foi necessário calcular-se o Índice de Consistência (IC), que é definido pela seguinte equação 3.8, em que o λmax é o valor próprio do Vetor de Prioridade calculado anteriormente, e o "n", o número de fatores da Matriz, que neste caso são 3.

$$IC = \frac{\lambda max - n}{n-1} = \frac{3,12-3}{3-1} = 0,048 \tag{3.8}$$

Após a obtenção do valor do IC, neste caso 0,048, foi calculado o RC, cujo o procedimento está refletido na equação 3.9, em que para este caso foi selecionado o valor de IR, para matrizes quadradas de ordem 3, com resultado visível na Tabela 3.4.

Tabela 3.4: Tabela Valores de IR para Matrizes quadradas de ordem n

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

$$RC = \frac{IC}{IR} = \frac{0,048}{0,58} = 0,0835 \tag{3.9}$$

Segundo (Nicola 2020b), uma RC menor que 10% "implica que o ajuste é pequeno em comparação com os valores atuais das entradas". Já se esse valor for superior a 90%, significa que os "julgamentos são emparelhados aleatoriamente e são completamente não confiáveis".

Em conclusão, visto que o RC é igual a 0,0835, menor que 10% (0.1), significa que os valores praticados, estão dentro dos limites de consistência.

De seguida, e verificado o último Vetor de Prioridade, foi identificado que a Partilha de Código Entre Plataformas é o que tem maior prioridade, seguido do Desempenho e, por fim, a Manutenibilidade e Estabilidade.

Contudo, é necessário perceber qual a alternativa mais viável, dando resposta ao problema surgido. Para tal, foi necessário calcular a RC para cada critério, de cada alternativa, sendo efetuada por ordem de prioridade de critério, previamente definido.

3.6.1 Partilha de código entre plataformas

Depois de analisada a Tabela 3.2, definiu-se os seguintes níveis de importância entre tecnologias, tendo em comum o critério "Partilha de código entre plataformas", resultando na Matriz visível na Equação 3.10,

(As figuras representadas com "RN", "F"e "KM", são letras que significam "React Native", "Flutter"e "Kotlin Multiplatform"respetivamente.)

$$Mpar = \begin{matrix} \\ RN \\ F \\ KM \end{matrix} \begin{matrix} \begin{matrix} RN & F & KM \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \end{matrix} \tag{3.10}$$

Utilizando o mesmo procedimento, para o cálculo da RC dos critérios, após a criação da Matriz de Comparação Par a Par, relativo à "Partilha de Código entre Plataformas", procedeu-se ao cálculo da Matriz de Comparação Normalizada, com resultado visível na Equação 3.11.

$$Mpar = \begin{matrix} \\ RN \\ F \\ KM \\ Soma \end{matrix} \begin{matrix} \begin{matrix} RN & F & KM \end{matrix} \\ \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 3 & 3 & 3 \end{bmatrix} \end{matrix} = \begin{matrix} \\ RN \\ F \\ KM \\ Soma \end{matrix} \begin{matrix} \begin{matrix} RN & F & KM \end{matrix} \\ \begin{bmatrix} 0.33 & 0.33 & 0.33 \\ 0.33 & 0.33 & 0.33 \\ 0.33 & 0.33 & 0.33 \\ 3 & 3 & 3 \end{bmatrix} \end{matrix} \tag{3.11}$$

Por fim, e de forma a ter os dados necessários para o cálculo do IC, e posteriormente da RC, foi necessário aplicar o processo da divisão dos dois vetores de prioridade, no qual resultou a Equação 3.12.

$$\begin{bmatrix} 0.99 \\ 0.99 \\ 0.99 \end{bmatrix} / \begin{bmatrix} 0.33 \\ 0.33 \\ 0.33 \end{bmatrix} = \begin{bmatrix} 3.00 \\ 3.00 \\ 3.00 \end{bmatrix} \tag{3.12}$$

Após a obtenção do último Vetor de Prioridade, foi aplicado a equação para o cálculo do IC, visível na equação 3.13, e posteriormente o cálculo da RC, visível na equação 3.14.

$$IC = \frac{\lambda max - n}{n-1} = \frac{3-3}{3-1} = 0 \tag{3.13}$$

$$RC = \frac{IC}{IR} = \frac{0}{0,58} = 0 \tag{3.14}$$

Em conclusão, e analisando o resultado da RC, uma vez que os níveis de importância definidos eram todos iguais, ou seja, todos de importância igualada, o resultado da RC é 0. Isto indica que não há necessidade de ajustes, pois qualquer alternativa revela-se favorável.

3.6.2 Desempenho

A Matriz de Comparação Par a Par, exposta na Equação 3.15, foi definida segundo as alternativas existentes, tendo em conta o critério "Desempenho".

$$Mdes = \begin{matrix} \\ RN \\ F \\ KM \end{matrix} \begin{matrix} \begin{matrix} RN & F & KM \end{matrix} \\ \begin{bmatrix} 1 & \frac{1}{3} & \frac{1}{3} \\ 3 & 1 & 1 \\ 3 & 1 & 1 \end{bmatrix} \end{matrix} \tag{3.15}$$

Após a criação da Matriz de Comparação Par a Par, relativo à "Manutenibilidade e Estabilidade", procedeu-se ao cálculo da Matriz de Comparação Normalizada, com resultado visível na Equação 3.16.

$$Mdes = \begin{matrix} \\ \frac{1}{7} \\ \frac{3}{7} \\ \frac{3}{7} \\ 7 \end{matrix} \begin{matrix} \begin{matrix} RN & F & KM \end{matrix} \\ \begin{bmatrix} \frac{1}{7} & \frac{0.33}{2.33} & \frac{0.33}{2.33} \\ \frac{3}{7} & \frac{1}{2.33} & \frac{1}{2.33} \\ \frac{3}{7} & \frac{1}{2.33} & \frac{1}{2.33} \\ 7 & 2.33 & 2.33 \end{bmatrix} \end{matrix} \begin{matrix} RN \\ F \\ KM \\ Soma \end{matrix} = \begin{matrix} \\ 0.143 \\ 0.429 \\ 0.429 \\ 7 \end{matrix} \begin{matrix} \begin{matrix} RN & F & KM \end{matrix} \\ \begin{bmatrix} 0.142 & 0.142 \\ 0.429 & 0.429 \\ 0.429 & 0.429 \\ 2.33 & 2.33 \end{bmatrix} \end{matrix} \begin{matrix} RN \\ F \\ KM \\ Soma \end{matrix} \tag{3.16}$$

Por fim, e de forma a ter os dados necessários para o cálculo do IC, e posteriormente da RC, foi necessário aplicar o processo da divisão dos dois vetores de prioridade, no qual resultou a Equação 3.17.

$$\begin{bmatrix} 0.237 \\ 1.573 \\ 1.573 \end{bmatrix} / \begin{bmatrix} 0.142 \\ 0.429 \\ 0.429 \end{bmatrix} = \begin{bmatrix} 3.12 \\ 3.06 \\ 3.01 \end{bmatrix} \tag{3.17}$$

Após a obtenção do último Vetor de Prioridade, foi aplicada a equação para o cálculo do IC, visível na equação 3.18 e, posteriormente o cálculo da RC, visível na equação 3.19.

$$IC = \frac{\lambda max - n}{n-1} = \frac{3,06-3}{3-1} = 0,03 \tag{3.18}$$

$$RC = \frac{IC}{IR} = \frac{0,03}{0,58} = 0,052 \tag{3.19}$$

Em conclusão, e analisando o resultado da RC igual a 0,052, que é menor que 10% (0.1), significa que os valores praticados estão dentro dos limites de consistência.

3.6.3 Manutenibilidade e Estabilidade

A Matriz de Comparação Par a Par, exposta na Equação 3.20, foi definida segundo as alternativas existentes, tendo em conta o critério "Manutenibilidade e Estabilidade".

$$Mman = \begin{matrix} \\ RN \\ F \\ KM \end{matrix} \begin{matrix} \begin{matrix} RN & F & KM \end{matrix} \\ \begin{bmatrix} 1 & 5 & 3 \\ \frac{1}{5} & 1 & \frac{1}{3} \\ \frac{1}{3} & 3 & 1 \end{bmatrix} \end{matrix} \begin{matrix} RN \\ F \\ KM \end{matrix} = \begin{matrix} \\ 1 \\ 0.20 \\ 0.33 \end{matrix} \begin{matrix} \begin{matrix} RN & F & KM \end{matrix} \\ \begin{bmatrix} 5 & 3 \\ 1 & 0.33 \\ 3 & 1 \end{bmatrix} \end{matrix} \begin{matrix} RN \\ F \\ KM \end{matrix} \tag{3.20}$$

Após a criação da Matriz de Comparação Par a Par, relativo à "Manutenibilidade e Estabilidade", procedeu-se ao cálculo da Matriz de Comparação Normalizada, com resultado visível na Equação 3.21.

$$Mman = \begin{matrix} RN \\ F \\ KM \\ 1.43 \end{matrix} \begin{matrix} F \\ \begin{bmatrix} \frac{1}{1.43} & \frac{5}{9} & \frac{3}{4.33} \\ \frac{0.20}{1.43} & \frac{1}{9} & \frac{0.33}{4.33} \\ \frac{0.33}{1.43} & \frac{3}{9} & \frac{1}{4.33} \end{bmatrix} \\ 9 \end{matrix} \begin{matrix} KM \\ RN \\ F \\ KM \\ 4.33 \end{matrix} \begin{matrix} \\ Soma \end{matrix} = \begin{matrix} RN \\ F \\ KM \\ 1.43 \end{matrix} \begin{matrix} F \\ \begin{bmatrix} 0.65 & 0.56 & 0.69 \\ 0.13 & 0.11 & 0.08 \\ 0.21 & 0.33 & 0.23 \end{bmatrix} \\ 9 \end{matrix} \begin{matrix} KM \\ RN \\ F \\ KM \\ 4.33 \end{matrix} \begin{matrix} \\ Soma \end{matrix} \tag{3.21}$$

Por fim, e de forma a ter os dados necessários para o cálculo do IC, e posteriormente da RC, foi necessário aplicar o processo da divisão dos dois vetores de prioridade, no qual resultou a Equação 3.22.

$$\begin{bmatrix} 1.95 \\ 0.32 \\ 0.79 \end{bmatrix} / \begin{bmatrix} 0.63 \\ 0.10 \\ 0.26 \end{bmatrix} = \begin{bmatrix} 3.09 \\ 3.20 \\ 3.04 \end{bmatrix} \tag{3.22}$$

Após a obtenção do último Vetor de Prioridade, foi aplicado a equação para o cálculo do IC, visível na equação 3.23 e, posteriormente o cálculo da RC, visível na equação 3.24.

$$IC = \frac{\lambda max - n}{n - 1} = \frac{3,11 - 3}{3 - 1} = 0,055 \tag{3.23}$$

$$RC = \frac{IC}{IR} = \frac{0,055}{0,58} = 0,095 \tag{3.24}$$

Em conclusão, e analisando o resultado da RC igual a 0,095, que é menor que 10% (0.1), significa que os valores praticados estão dentro dos limites de consistência.

Após o cálculo dos Vetor de Prioridade, para cada critério, e para cada alternativa, tendo em conta um critério, e comprovada a sua consistência, foi definida uma árvore hierárquica, com os valores próprios obtidos, sendo visível na Figura 3.9.

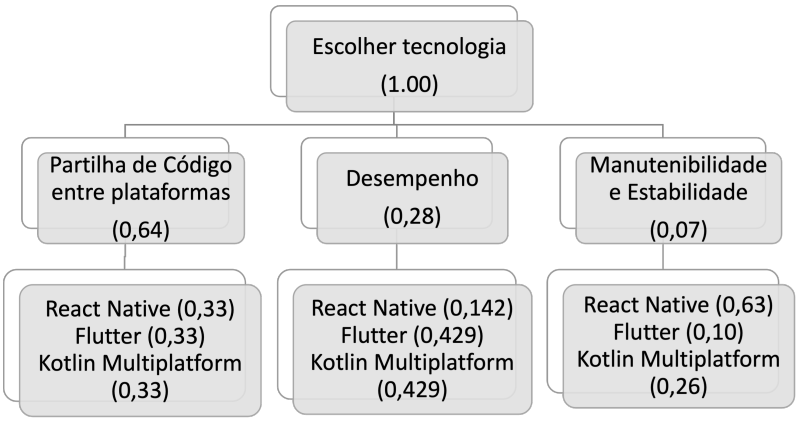

Figura 3.9: Árvore hierárquica de decisão

3.6.4 Matriz de Prioridades

Através da Árvore Hierárquica de Decisão mencionada anteriormente, na Figura 3.9, definiu-se uma Matriz de Prioridades das alternativas existentes, multiplicando-se, posteriormente, pelas prioridades relativas dos critérios, resultando na Equação 3.25.

$$\begin{bmatrix} 0.33 & 0.142 & 0.63 \\ 0.33 & 0.429 & 0.10 \\ 0.33 & 0.429 & 0.26 \end{bmatrix} * \begin{bmatrix} 0.64 \\ 0.28 \\ 0.07 \end{bmatrix} = \begin{bmatrix} 0.211 & 0.040 & 0.044 \\ 0.211 & 0.120 & 0.007 \\ 0.211 & 0.120 & 0.018 \end{bmatrix} \tag{3.25}$$

3.6.5 Classificação das Tecnologias

Por fim, para se proceder à classificação das tecnologias, é necessário somar cada linha resultante da Matriz de Prioridades, fornecendo um Vetor de Prioridade, cujo valor mais elevado, pode ser considerada a tecnologia mais valiosa, para resposta ao problema adjacente.

A Equação 3.26 fornece o resultado final da análise, utilizando o Método AHP. Após a análise, é possível afirmar que a tecnologia Kotlin Multiplatform é a mais indicada para o desenvolvimento de uma aplicação multiplataforma, tendo em conta os critérios estabelecidos previamente. Esta análise rejeita a escolha efetuada pela empresa, contudo ressalvando que existem diversos critérios extra (específicos da empresa) que não foram tidos em conta, devido ao facto de serem fatores não controláveis pelas tecnologias, mas sim pela mão de obra da mesma.

$$\begin{matrix} ReactNative \\ Flutter \\ KotlinMultiplatform \end{matrix} \begin{bmatrix} 0.211 & 0.040 & 0.044 \\ 0.211 & 0.120 & 0.007 \\ 0.211 & 0.120 & 0.018 \end{bmatrix} = \begin{bmatrix} 0.295 \\ 0.338 \\ 0.349 \end{bmatrix} \tag{3.26}$$

3.7 Sumário

Neste capitulo foram estudados razões e oportunidades para a realização do trabalho em questão. Para tal, foram investigadas e definidas oportunidades, como a evolução da tecnologia, o crescimento do mercado de apostas, e também o crescimento de uma sociedade tecnologicamente ativa.

Foi também apresentada a análise de valor ao projeto em questão, de forma a perceber qual o real valor do projeto para o cliente. Verificou-se que os potenciais clientes para este projeto seriam qualquer utilizador de telemóveis e/ou utilizador de plataformas de apostas desportivas. Desta análise surgiu o modelo Canvas.

Posto isto, foram estudadas uma série de tecnologias de desenvolvimento de aplicações móveis, como o React Native, Flutter, Kotlin Multiplatform e a linguagem Swift. Como tal, pode-se concluir que a tecnologia Kotlin Multiplatform é a mais atrativa para o desenvolvimento de uma aplicação móvel nativa, contrariando a decisão da empresa. Essa decisão teve por base fatores extra critérios estabelecidos para o estudo, pois são fatores que não são controlados pelas tecnologias.

Capítulo 4

Análise Desenho e Implementação da Solução

Este capítulo tem como objetivo mostar ao leitor a análise, o desenho e a implementação da solução. Esta solução consiste na tentativa de melhorar o desempenho, das aplicações desenvolvidas em React Native, quando utilizadas listas encadeadas. Para tal, a solução encontrada foi a substituição do componente FlatList (componente utilizado para renderizar listas), por uma alternativa viável e com melhor desempenho.

Para tal, é iniciado pela engenharia de requisitos, ou seja, a documentação da análise e modelação, onde são apresentados, segundo o modelo FURPS+, os requisitos não funcionais, seguidos dos requisitos funcionais, e por fim, o desenho da solução.

4.1 Análise e Desenho da Solução

Esta secção visa oferecer uma visão do processo de engenharia, adjacente à sua implementação.

4.1.1 Engenharia de Requisitos

Nesta subsecção, é apresentada a análise de requisitos não funcionais e funcionais que surgiram no projeto. Os requisitos são classificados segundo as categorias do modelo FURPS+. A utilização deste modelo prende-se com o seu reconhecimento a nível tecnológico e por ter sido bem sucedido em anteriores implementações.

Requisitos não funcionais

Os requisitos não funcionais, segundo o modelo FURPS+, para atestar a qualidade do software são: requisito de usabilidade, confiabilidade, desempenho, suportabilidade, podendo também existir outros, daí a terminação da Desenhoação com um sinal "+". Esses outros requisitos são referentes à qualidade do processo do software, ou seja, os requisitos de desenho, implementação e interface.

- **Usabilidade**: avalia os requisitos baseados na interface com o utilizador, tais como a acessibilidade, estética, consistência (Chung e Prado Leite 2009).

  Neste projeto, existem alguns requisitos de usabilidade associados, tais como:

  - Eficiência na apresentação da interface;
  - Interface agradável e atrativa para os utilizadores a nível de consistência.

- **Confiabilidade**: restringe os requisitos baseados com a disponibilidade, precisão, recuperabilidade e tempo médio entre falhas (Chung e Prado Leite 2009).

  No desenvolvimento deste projeto os requisitos associados à confiabilidade são:

  - Tempo médio de espera para o processamento dos elementos de uma lista;
  - Tempo de execução da página que sofra a atividade do projeto desenvolvido;
  - Caso de alteração de um elemento da lista, o tempo de atualização ser o mais eficiente possível.

- **Desempenho**: restringe requisitos baseados com o tempo de resposta, tempo de arranque e recuperação do sistema e gestão de recursos (Chung e Prado Leite 2009).

  Os requisitos relacionados com o desempenho são um dos principais fatores para garantir a qualidade do projeto, sendo que para tal foram definidos os seguintes fatores:

  - O tempo de arranque do processamento dos elementos da lista ser inferior ao tempo de arranque da página;
  - Aquando de uma atualização, o facto do processamento ser ativado antes da página apresentar defeito, ou seja, não estar em coerência com o produto que deve ser apresentado;
  - Baixo consumo de memória.

- **Suportabilidade**: A suportabilidade restringe os requisitos relacionados com limitações de idioma, testabilidade, adaptabilidade, manutenibilidade, compatibilidade, configurabilidade e escalabilidade (Chung e Prado Leite 2009).

  Neste projeto, os requisitos de suportabilidade presentes são:

  - O projeto deve estar adaptado para permitir a adição de novas funcionalidades no futuro;
  - Deve ser compatível com os diferentes tipos de listas que possam surgir e com as tecnologias que consigam adotar este projeto, como uma biblioteca de apoio.

- **Outros (+)**: Esta secção final cobre os requisitos de desenho, implementação e de interface. Os últimos requisitos não funcionais são:

  - O projeto deve ser desenvolvido através de um processo iterativo;
  - O projeto deve ser testado através de testes de desempenho;
  - No seu desenvolvimento devem ser utilizadas boas práticas de programação.

Requisitos funcionais

Os requisitos funcionais definem as principais características do sistema. Posto isto, pelo carácter arquitetural assumido pelo projeto, estes requisitos são unicamente associados à implementação do projeto.

Para uma melhor organização, seguindo uma metodologia Agile, neste caso Kanban, foram criadas diversas *User Stories*, de forma a registar todo o trabalho necessário para a construção final da solução.

Na Tabela 4.1, pode-se observar as diferentes *User Stories* (US) criadas para o desenvolvimento posterior do projeto.

Tabela 4.1: *User Stories* definidas para a implementação da solução

User Stories	**Descrição**
US01[Investigação]	Investigação de tecnologias que possam melhorar a performance das FlatLists (JSI Modules)
US02[Investigação]	Investigar como otimizar os recursos de React Native, para a utilização de FlatLists
US03[Desempenho]	Otimizar o uso das FlatLists no contexto do projeto da empresa
US04[Funcionalidade]	Criar aplicação de teste
US05[Funcionalidade e Teste]	Utilizar e testar FlatList
US06[Funcionalidade e Teste]	Utilizar e testar FlashList
US07[Funcionalidade e Teste]	Utilizar e testar BigList
US08[Funcionalidade e Teste]	Utilizar e testar RecyclerListView
US09[Funcionalidade e Teste]	Implementar lista com melhor desempenho na app
US010[Teste]	Testar solução, tirando métricas de desempenho com um antes e o depois

No início do análise das aplicações, foram definidos os requisitos funcionais das mesmas. Estes requisitos foram analisados em forma de funcionalidades e transformados em Casos de Uso (UC), como apresentado na sequência abaixo:

- UCs World of Lists:
  - **UC1**: Navegar pela página FlatList;
  - **UC2**: Navegar pela página FlashList;
  - **UC3**: Navegar pela página BigList;
  - **UC4**: Navegar pela página RecyclerListView;
- UC Betfair Rebuild:
  - **UC1S**: Navegar pela página de futebol

Após a análise e identificação dos casos de uso, procedeu-se à criação do respetivo Diagrama de Casos de Uso, para a aplicação World of Lists e Betfair Rebuild, como visível nas Figuras 4.1 e 4.2.

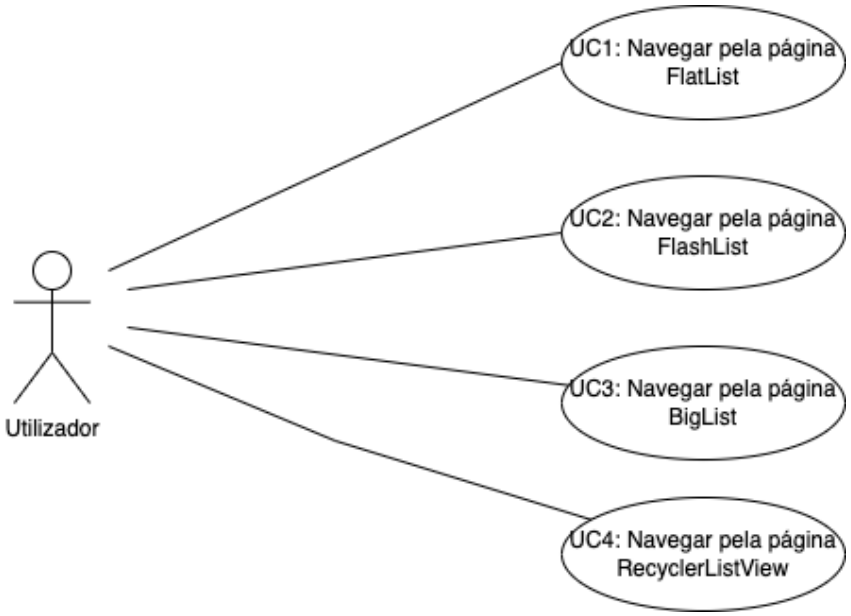

Figura 4.1: Diagrama Casos de Uso: World of Lists

Figura 4.2: Diagrama Casos de Uso: Betfair Rebuild

4.1.2 Desenho da Solução

Findada a análise da solução, segue-se o desenho. Esta subsecção é caracterizada pela explicação da arquitetura geral da aplicação da Betfair e World of Lists, tal como pela a explicação do processo definido para a realização de um caso de uso.

Diagrama de Componentes

Na Figura 4.3 está retratado o diagrama de componentes da aplicação World of Lists. Neste caso, a aplicação interage, através de pedidos HTTP diretos, com a API denominada `pokeapi.co`.

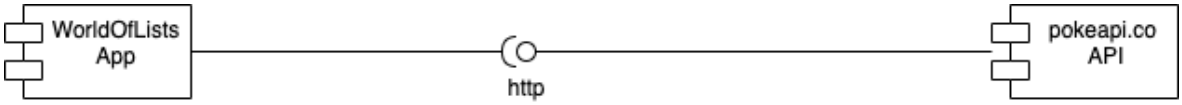

Figura 4.3: Diagrama de componentes da app de teste: World of Lists

Por outro lado, no diagrama de componentes da aplicação Betfair Rebuild, apresentado na Figura 4.4, a aplicação interage, através do nginx[1], o Reverse Proxy, local que realiza um dos redirecionamentos possíveis (GQL[2] ou Access Control). Quando interage com o Access Control, através de pedidos https, isto indica que o Access Control irá validar a autorização do utilizador. Por outro lado, quando interage com o GQL, este irá, através de pedidos REST, interagir com um (ou mais) serviços que contenha a informação necessária, sendo que estes interagem, através de SQL Connectors, com a respetiva base de dados.

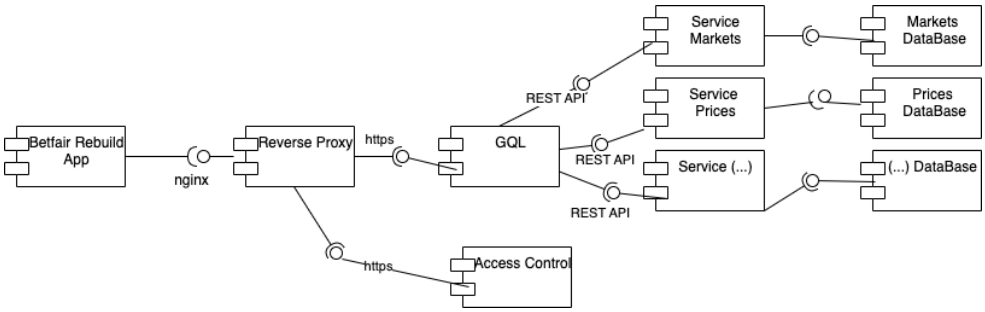

Figura 4.4: Diagrama Sequência UC1: Navegar pela página XList

Desenho Casos de Uso

Tendo em vista a delineação de como cada UC irá funcionar, enfatizando as diferentes interações entre os objetos, diagramas de sequência são apresentados nesta subsecção. Neste caso, como na aplicação World of Lists, o processo de construção dos ecrãs é semelhante, foi desenhado um diagrama de sequência genérico, que alberga todas as UC's criadas para a aplicação World of Lists.

Posto isto será utilizada a UC1: Navegar pela página `FlatListScreen`, com o propósito de servir de exemplo.

[1] nginx: é um servidor HTTP, com funcionalidades como o *proxy* reverso

[2] GQL: Nome não-oficial, que representa uma API da aplicação, cuja função é organizar e fazer pedidos a diversos serviços

UC1: Navegar pela página FlatListScreen Na Figura 4.5 está apresentado o diagrama de sequência das UC's correspondentes à aplicação World of Lists, cujo objetivo é navegar por um ecrã de uma determinada lista. O processo começa pela interação do utilizador com o `HomeScreen`, com o objetivo de clicar num dos 4 botões disponíveis, selecionando o botão com o nome "Test FlatList".

Após isto, o ecrã que contém o `HomeScreen` irá iniciar a página `FlatListScreen`, através do método `navigation.navigate("FlatList")`. O `FlatListScreen` irá pedir informação, à classe `DataCall`, com o método `DataCall.getContent()`, sendo que esta irá fazer um pedido à API, com o método `fetch(url)`. Apesar da API retornar uma lista considerável de elementos, apenas os seis primeiros serão mapeados para o componente, e posteriormente renderizados.

Após a recolha de dados, estes são mapeados para o `FlatListScreen`, que renderiza posteriormente, o componente `Category`, que irá também executar um pedido de informação de dados, à classe `FlatListScreen`, com o método `DataCall.getSubContent()`, e este por sua vez volta a fazer um pedido à API, com um *url* diferente, com o método `fetch(url)`. Tal como o método `DataCall.getContent()`, só os primeiros seis elementos serão mapeados para o componente, e posteriormente renderizados.

Por fim, o componente `Category` ira disponibilizar a informação já totalmente mapeada no `FlatListScreen`, que por sua vez mostra ao utilizador a página totalmente construída e navegável.

Existe também a possibilidade de o utilizador interagir com a página, efetuando diversos *scrolls*. Estes acabam por fazer despoletar novamente todo o processo de pedido de dados. A cada scroll, um novo pedido à API é realizado, através do processo mencionado anteriormente. Como tal, só os seis elementos consecutivos serão mapeados para o componente e posteriormente renderizados, e assim sucessivamente.

A realização de pedidos à API, apesar de só os seis primeiros serem contabilizados, serve como forma de enviar para o componente, o menor número de elementos possíveis, tendo em conta o seu tamanho, e o número de elementos que consegue mostrar no ecrã em simultâneo.

Neste caso, segundos os dispositivos testados são mostrados três elementos, mas com o objetivo de ser uma aplicação transversal a qualquer dispositivo, independentemente do seu tamanho, seis elementos são enviados para o componente, de forma a assegurar que não fica um espaço em branco no ecrã.

Para além da aplicação de testes (World of Lists), também foram desenvolvidas ações na aplicação Betfair Rebuild, aplicação esta que contém o principal objetivo deste documento. Posto isto, o parágrafo abaixo demonstra a interação entre o utilizador a aplicação, na tentativa de navegar pela página de futebol.

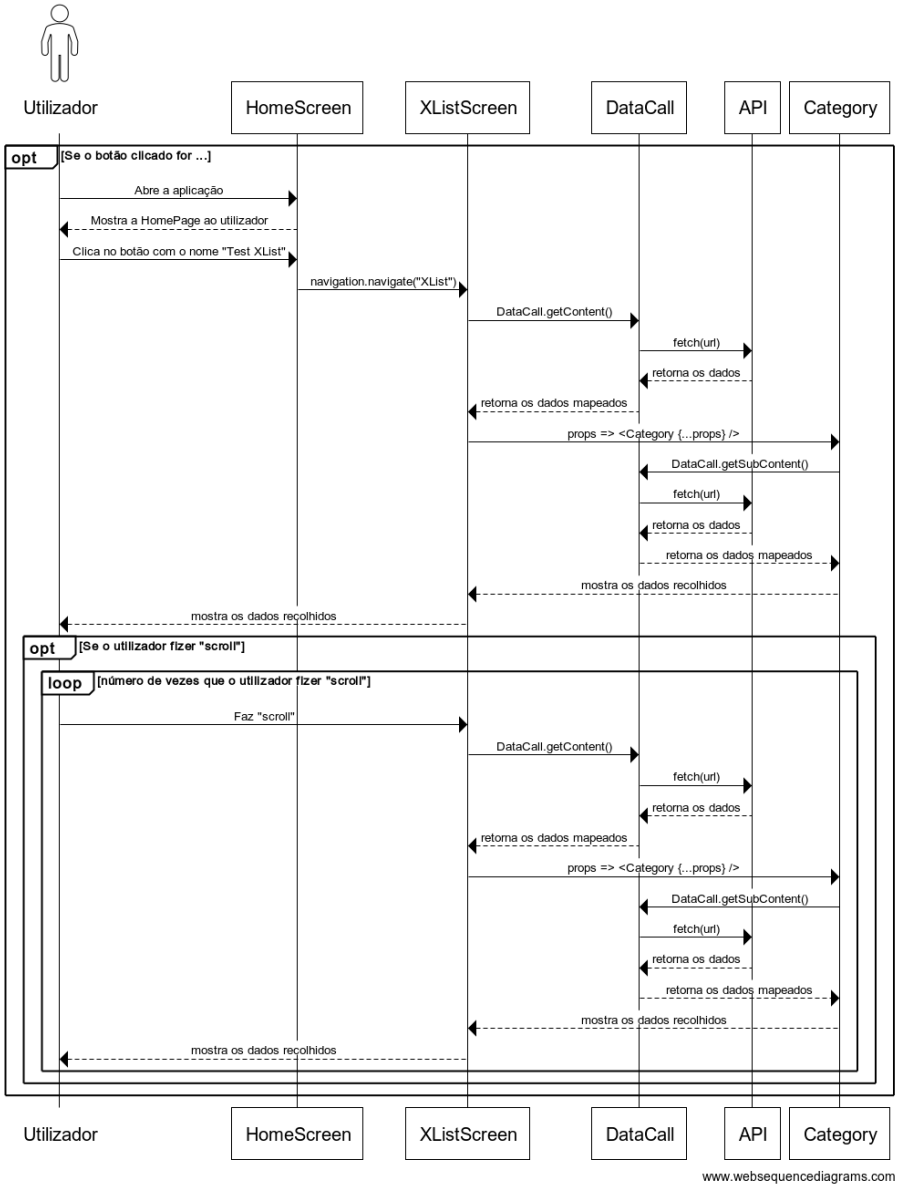

Figura 4.5: Desenho Casos de Uso UC: Navegar pela página XList

UC1S: Navegar pela página de futebol Na Figura 4.6 está apresentado o diagrama de sequência do UC1S. O processo começa pela interação do utilizador com o `UIHomePage`, com o objetivo de clicar num botão com o icon de futebol correspondente. O Utilizador faz o clique e posto isto, o página que contém o `UIHomePage` irá abrir a `FootballPage`, através do método `navigation.navigate("/football")`.

A `FootballPage` irá pedir informação, à `Store`, com o método `dispatch()`, sendo que esta irá pedir a informação ao `BFF`, com o método `getInfo()`. Por fim, este faz um pedido a uma (ou mais) API('s), com o método `fetch(url)`.

Após a recolha de dados, estes são retornados para o `bff`, que por sua vez os envia para a `Store` para serem mapeados. Após isto, a `Store` envia a informação para a `FootballPage`, com o método `createInfoSelector()`, que renderiza a página segundo as informações recolhidas, ficando disponível para interação com o utilizador.

Existe também a possibilidade de o utilizador interagir com a página, efetuando diversos *scrolls*. Estes acabam por fazer despoletar novamente todo o processo de pedido de dados.

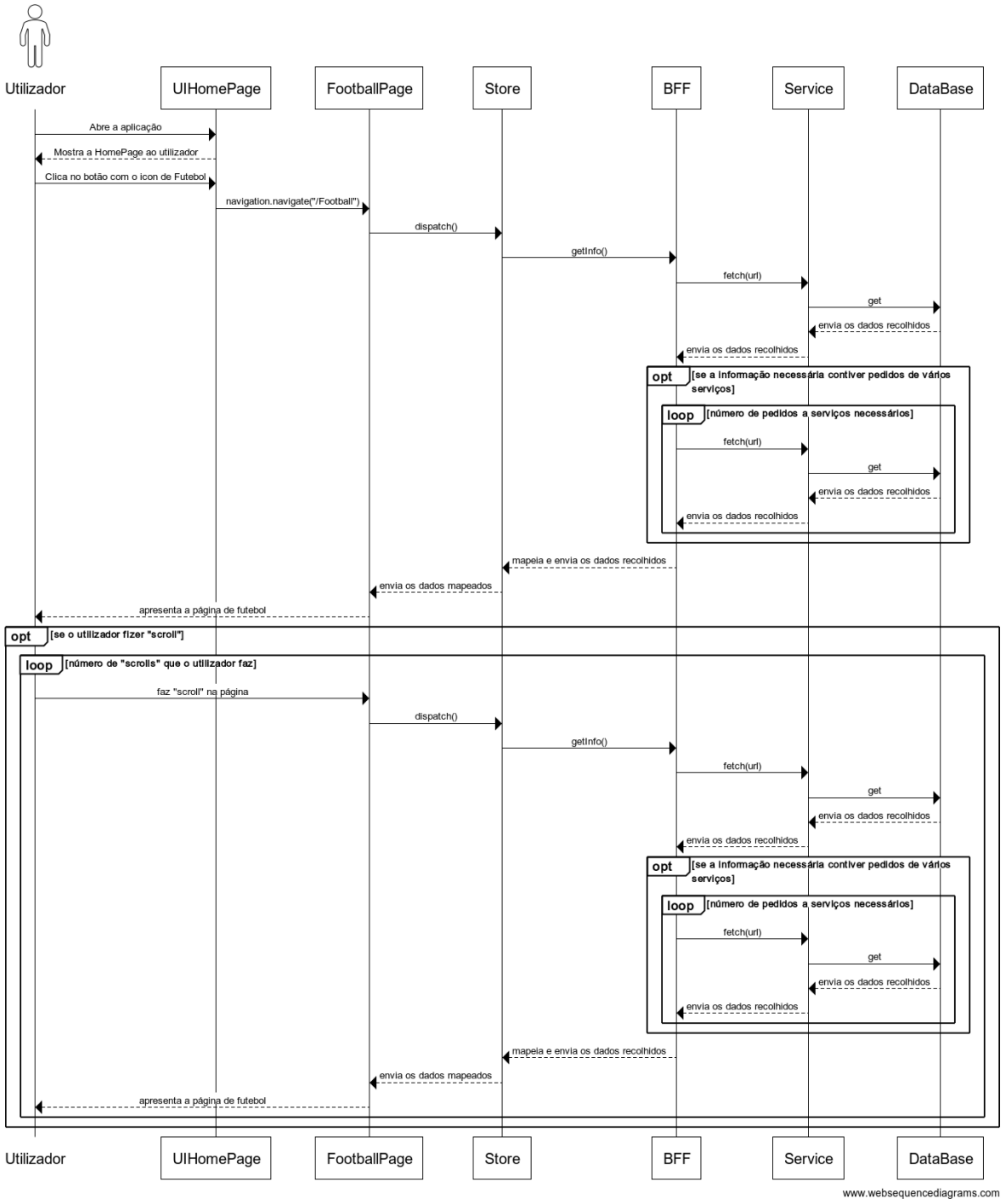

Figura 4.6: Diagrama Sequência UC1S: Navegar pela página de Futebol

4.2 Implementação da Solução

O desempenho é vital em aplicações direcionadas a dispositivos móveis, visto que os recursos como Central Process Unit (CPU) e Random Access Memory (RAM) são escassos em comparação com os computadores, onde estas aplicações são desenvolvidas e testadas.

Observou-se no passado diversas empresas que construíram diversas aplicações de renome em React Native, sendo a Airbnb um dos exemplos mais conhecidos. Os seus programadores focaram-se na tentativa de aperfeiçoar o desempenho, partilhando as suas descobertas, mas acabaram por desistir, pois não tinham pessoas suficientemente especializadas, para

terminar toda a pesquisa e aperfeiçoamento a tempo (Peal 2018a). Para além disso, consideravam demasiada imatura a *framework* para conseguir acompanhar os requisitos do seu projeto/produto (Peal 2018b).

Felizmente, o React Native evoluiu significativamente desde esse momento, em termos de tecnologia e compreensão pela comunidade tecnológica, o que permitiu um melhor suporte aos problemas de desempenho que atualmente existem.

No contexto do projeto Betfair Rebuild, os testes de desempenho efetuados concluíram que a interação com listas contribuíram para a degradação do desempenho, principalmente na aplicação Android.

Os componentes com listas são amplamente utilizados nas aplicações móveis para listar a informação. A quantidade de informação a ser mostrada e a taxa de *scroll* pode, por vezes, ter impacto na experiência do utilizador e criar *jank*[3]. Exemplos destes cenários podem ser facilmente encontrados quando é feito um *scroll* rápido e "infinito"com informação dinâmica.

A comunidade tecnológica desenvolveu algumas soluções baseadas em conceitos como Infinite Scroll, Virtualized List e Recycler List, de forma a tentar corrigir esses comportamentos descritos previamente. Posto isto, nas próximas secções serão abordadas as diversas alternativas implementadas - de forma a resolver o problema de desempenho - utilizando os principais projetos da comunidade tecnológica sobre listas otimizadas (RecyclerListView e BigList), um projeto desenvolvidos pela Blip (SnapshotList), uma atualização da atual FlatList e, por fim uma lista desenvolvida pela Shopify - denominada de FlashList.

Estas tentativas de implementação e estudo, foram desenvolvidas numa aplicação - criada para o efeito, em que sob as mesmas condições (com a mesma consistência de dados), fossem capazes de revelar métricas o mais fidedignas possíveis.

Posteriormente, são comparados os prós e os contras com a contraparte React Native (que é a FlatList), destacando também essas métricas obtidas e avaliando-as. Por fim, é utilizada a FlashList na aplicação do Betfair Rebuild, uma vez que se constitui como a lista que melhor satisfaz os condicionalismos impostos pela aplicação, de forma a comparar o seu desempenho com a implementação atual (com recurso à FlatList).

4.2.1 Implementação aplicação de teste: World of Lists

Como mencionado anteriormente, para testar as alternativas encontradas, para melhorar o desempenho de uma aplicação - desenvolvida em React Native - quando confrontado com listas encadeadas, foi criada uma aplicação. Denominada World of Lists, com o propósito de testar sob a mesma consistência de dados, esta aplicação é composta por um ecrã inicial (Figura 4.7), que tem quatro botões diferentes, cada um com o objetivo de navegação, pelo utilizador, para uma página idêntica, mas desenvolvida com o suporte de componentes de listas diferentes (FlatList, FlashList, BigList e RecyclerListView).

Após o clique num dos botões, a aplicação efetua um pedido a uma API gratuita `pokeapi.co`, com o objetivo de recolher o maior número de dados possíveis para uma lista. Após, esse pedido, um outro é efetuado, por cada item obtido no pedido anterior, criando assim uma lista encadeada, com diversos pedidos a serem executados em paralelo. De forma a simular o comportamento mais correto, e seguindo as boas práticas de programação, só são

[3] Jank é o resultado de *frames* que demoram muito para a criação no navegador, e isso impacta negativamente os seus utilizadores e a sua experiência

mapeados seis elementos do pedido de cada vez, sendo os restantes despoletados após o movimento de *scroll* (de seis em seis elementos). No final das ações internas da aplicação, o resultado deverá constituir-se com uma página com uma lista, com diversos elementos, com a capacidade de fazer *scroll* vertical e horizontal (exceto no caso do BigList, pois o mesmo não tem essa capacidade), como demonstrado na figura abaixo (Figura 4.7).

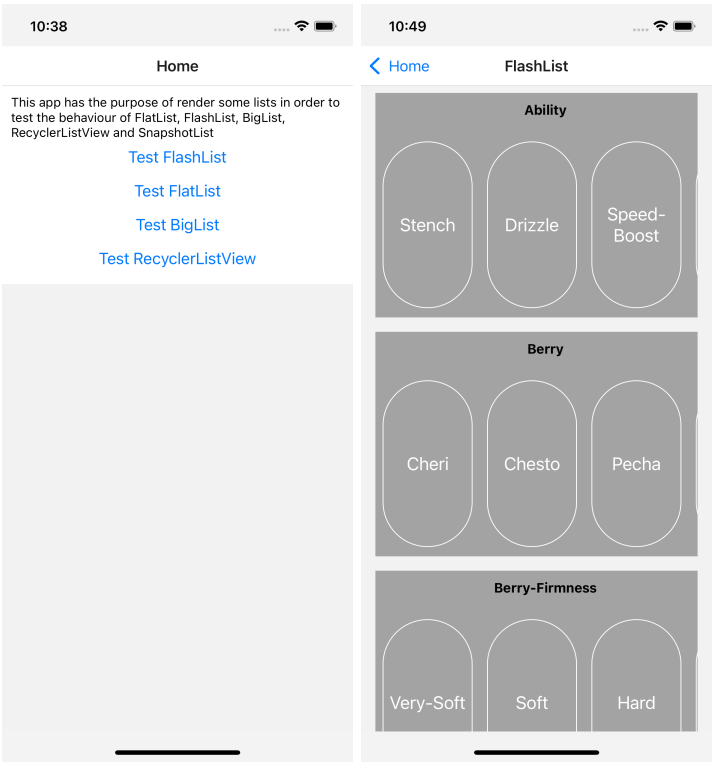

Figura 4.7: World of Lists App: Ecrã Inicial e Ecrã com informação

Como mencionado anteriormente, foi simulado o mesmo comportamento, para a recolha de dados ser o mais consistente possível. Para tal, foi criada uma classe (Excerto de código 4.1), que faz um pedido asíncrono à API `https://pokeapi.co/api/v2`, com o propósito de obter os dados existentes neste serviço. Esta classe é utilizada por todas as listas, de forma ao pedido ser mapeado sempre de igual forma. As suas funções são caracterizadas por retornarem apenas seis elementos de cada vez, de forma a não sobrecarregar a renderização dos componentes na UI, apesar de ser efetuado sempre um pedido à API, quando são chamadas.

```
1  export class DataCall {
2    static async getContent(start, count) {
3      const pokeResp = await fetch('https://pokeapi.co/api/v2');
4      const fullData = await pokeResp.json();
5      const mappedData = Object.values(fullData);
6      const filteredData = mappedData.slice(
7        start,
8        Math.min(mappedData.length, start + count),
9      );
10     return filteredData;
11   }
12
13   static async getSubContent(url, start, count) {
14     const pokeResp = await fetch(url);
```

```
15      const fullData = await pokeResp.json();
16      const mappedData = fullData.results;
17      const filteredData = mappedData.slice(
18        start,
19        Math.min(mappedData.length, start + count),
20      );
21      return filteredData;
22    }
23  }
```

Excerto de código 4.1: DataCall classe

A utilização das funções da classe `DataCall` foi criada segundo as boas práticas do React e React Native, sendo chamadas após a montagem do componente (`componentDidMount`). Esta função chama uma função assíncrona denonimada `fetchMoreData`, que espera pela conclusão da recolha de dados por parte da função da classe `DataCall`, e que posteriormente altera o estado do componenete, com esses dados recolhidos. Após isto, também altera o contador `count`, com o intuito de na próxima vez que for executada, receber os seis items posteriores aos atualmente recebidos. Isto é observado no Excerto de código 4.2.

```
1  export class XListScreen extends Component {
2  ...
3   componentDidMount() {
4      this.fetchMoreData();
5    }
6
7    async fetchMoreData() {
8      if (!this.inProgressNetworkReq) {
9        this.inProgressNetworkReq = true;
10       const categories = await DataCall.getContent(this.state.count, 6);
11       this.inProgressNetworkReq = false;
12       this.setState({
13         categories: this.state.categories.concat(Object.values(categories)
     ),
14         count: this.state.count + 6,
15       });
16     }
17   }
18 ...
19 }
```

Excerto de código 4.2: fetchMoreData

Tendo em vista a melhor otimização possível para a renderização de listas, algumas `props` (Excerto de código 4.3), definidas para ajudar na otimização dos componentes de FlatList , foram ajustados com valores pré definidos. Quatro `props` foram ajustadas, tais como:

- `initialNumToRender`: ajustado com o valor 4, pois indica o número inicial de items a serem renderizados;
- `windowSize`: definido com o valor 11, pois indica ao componente, o número de *viewports* (5 acima, 1 visível, 5 abaixo). Com valor pré definido de 21, foi decidido diminuir o valor para 11, pois assim menos items são montados em simultâneo, o que reduz o consumo de memória;
- `maxRenderPerBatch`: estabelecido com o valor 4, pois indica o número de items renderizados em cada *scroll*. O valor definido teve em conta a quantidade de items

que são mostrados no *viewport*, três, mais um item de reserva, para não se notar uma quebra quando um *scroll* ocorre;

- `removeClippedSubviews`: especificado com *false* para plataforma iOS e *true* para plataforma Android, esta *prop* tem como objetivo reduzir o risco de *frames* serem reduzidas. No caso de iOS não é ativada, visto que contém alguns *bugs* na sua utilização.

Também se definiu o valor 90 para a *prop* `itemVisiblePercentThreshold`. Isto indica ao componente quando é que ele deve renderizar o item a seguir. Ou seja, quando estiver 90% visível.

```
1  import {Platform} from 'React Native';
2
3  export const FLAT_LIST_DEFAULTS = {
4    initialNumToRender: 4,
5    windowSize: 11,
6    maxToRenderPerBatch: 4,
7    removeClippedSubviews: Platform.select({ios: false, android: true}),
8  };
9
10 export const VIEWABILITY_CONFIG = {
11   itemVisiblePercentThreshold: 90,
12 };
```

Excerto de código 4.3: Excerto de código *props default (Otimização)*

De forma a recriar as páginas sob as mesmas condições e de forma mais otimizada possível, foram definidas um conjunto de *props*, específicas de cada componente.

FlatList: Uma vez que todo o trabalho se desenvolve a partir das falhas de desempenho deste componente, utilizou-se as práticas de otimização existentes, como o conjunto de `props` mencionado anteriormente no Excerto de código 4.3, juntamente com outras definidas, como exemplificado no Excerto de código 4.4. Essas `props` são:

- `onEndReached`: quando executado (no momento em que o scroll chega ao fim da atual lista de elementos disponíveis), é feita uma nova chamada da função `fetchMoreData`. Posto isto, o componente renderiza os novos elementos resultantes da chamada da função.
- `data`: onde é especificado os dados a serem mostrados pelo componente;
- `keyExtractor`: onde é definido uma *key* única de forma a identificar cada item;
- `renderItem`: onde é renderizado o item conforme o estilo desejado. Por exemplo, neste caso o item é transformado num componente denonimado Category;
- `ListFooterComponent`: tem o intuito de renderizar um componente com a capacidade de mostrar que o componente pai está a carregar.

```
1  <FlatList
2      style={{flex: 1}}
3      viewabilityConfig={VIEWABILITY_CONFIG}
4      {...{
5          FLAT_LIST_DEFAULTS,
6      }}
7      contentContainerStyle={{margin: 3}}
```

```
8     onEndReached={this.handleListEnd}
9     data={this.state.categories}
10    keyExtractor={item => item}
11    renderItem={this.rowRenderer}
12    ListFooterComponent={this.renderFooter}
13 />
```

Excerto de código 4.4: FlatList props

FlashList: Seguindo para o componente FlashList verifica-se, no Excerto de código 4.5, que tem bastantes semelhanças com o FlatList, em termos de atribuição e definição de `props`. A grande diferença reside na a introdução, opcional, de uma *prop* denominada `estimatedItemSize`, que ao ser definida indica ao componente o tamanho estimado de cada item, conseguindo o mesmo renderizar de forma mais otimizada.

```
1  <FlashList
2      data={this.state.categories}
3      keyExtractor={item => item}
4      renderItem={this.rowRenderer}
5      ListFooterComponent={this.renderFooter}
6      onEndReached={this.handleListEnd}
7      viewabilityConfig={VIEWABILITY_CONFIG}
8      style={{flex: 1}}
9      estimatedItemSize={250}
10     contentContainerStyle={{margin: 3}}
11 />
```

Excerto de código 4.5: Flashlist props

BigList: Quando comparado com o componente BigList, observável no Excerto de código 4.6, é possível de se afirmar, que existem `props` comuns ao FlatList e FlashList, como `data`, `keyExtractor` e o `renderItem`. Por outro lado, a *prop* `renderFooter` tem o mesmo comportamento que a respetiva `ListFooterComponent`, presente no FlatList e no FlashList. Contudo, este componente tem três `props` específicas, com o intuito de o otimizar o máximo possível. São elas:

- `itemHeight`: onde é definida a altura do item;
- `headerHeight`: que tem o dever de especificar a altura do cabeçalho do componente;
- `footerHeight`: cujo objetivo é indicar a altura do rodápe do componente existente.

```
1  <BigList
2      data={this.state.categories}
3      keyExtractor={item => item}
4      renderItem={this.rowRenderer}
5      renderFooter={this.renderFooter}
6      onEndReached={this.handleListEnd}
7      itemHeight={250}
8      headerHeight={0}
9      footerHeight={60}
10     style={{flex: 1}}
11     contentContainerStyle={{margin: 3}}
12 />
```

Excerto de código 4.6: BigList props

RecyclerListView: Por fim, quando comparado com o componente RecyclerListView, apresentado no Excerto de código 4.7, verifica-se que a sua construção é totalmente diferente. As `props` que o constitutem são bastante características do componente, sendo que apenas uma é comum a todos os outros componentes, o `onEndReached` e o `renderFooter`, semelhante ao BigList. As restantes `props` obrigatórias para a construção otimizada deste componente são:

- `dataProvider`: responsável por indicar ao componente qual os dados a serem renderizados;
- `layoutProvider`: que indica ao componente como devem os items serem distribuídos na lista, ou seja, se horizontal ou vertical, ou se alguns em modo horizontal e/ou outros em modo vertical;
- `rowRenderer`: onde é definido como o item deve ser mostrado, sendo que neste caso o item é alterado para ser mostrado como um componente Category.

```
1 <RecyclerListView
2    style={{flex: 1}}
3    contentContainerStyle={{margin: 3}}
4    onEndReached={this.handleListEnd}
5    dataProvider={this.state.dataProvider}
6    layoutProvider={this.state.layoutProvider}
7    rowRenderer={this.rowRenderer}
8    renderFooter={this.renderFooter}
9 />
```

Excerto de código 4.7: RecyclerListView props

Como é possível observar, a maioria dos componentes têm `props` em comum, o que facilita qualquer migração de uma lista para a outra, excetuando o RecyclerListView, que é composta por `props` bastante características, o que torna bastante complicado e demorado o processo de migração, caso seja necessário.

4.2.2 Conceptualização dos componentes utilizados

Esta subsecção tem o propósito de dar a conhecer ao leitor, como são desenvolvidos e as características de cada componente já mencionado anteriormente.

FlatList

O componente FlatList foi desenvolvido para colmatar algumas deficiências do componente outrora utilizado, ScrollView, como por exemplo a renderização de listas mais longas com eficiência. VirtualizedList é o componente por detrás da FlatList e é onde se encontra a implementação React Native do conceito de virtualização.

O problema com o componente ScrollView consistia no ato de renderizar todos os itens de uma vez. Embora visualmente seja mais atrativo, a memória, o tempo, a JS Thread e a UI Thread têm menor desempenho, levando a falhas na aplicação e uma experiência mais lenta.

Quando ajustado com critério, a FlatList é capaz de renderizar apenas os itens visíveis dentro da *viewport*, permitindo um significativo ganho de memória consumida e tempo de processamento, mas comprometendo o desempenho visual, especialmente em listas dinâmicas e mais complexas, que é o caso do projeto Betfair Rebuild.

Embora o React Native tenha feito algum progresso com a introdução da FlatList, ainda se encontra longe da maturidade e do desempenho do `RecyclerView` nativo, no Android ou `UICollectionView`, no iOS, principalmente devido à natureza *single-thread* e assíncrona.

RecyclerListView

O componente RecyclerListView foi desenvolvido, em JavaScript, sob o mesmo conceito de Recycler List que os componentes nativos `RecyclerView` e `UICollectionView`, ou seja, sob o conceito de reutilizar o componente, quando este volta a aparecer no *viewport*, em vez de o destruir, e posteriormente voltar a construir.

Este componente foi desenvolvido com o objetivo de colmatar as deficiências da FlatList. Em vez de criar visualizações para conteúdos no ecrã e destruir as visualizações fora do ecrã, o RecyclerListView recicla de forma inteligente as visualizações criadas anteriormente.

Este componente utiliza técnicas semelhantes às versões nativas, como *Layout* e *Providers*, podendo-se especificar os dados, o tipo de *layout* e as dimensões estimadas por meio desses *providers*. Posto isto, usa os tipos de *layout* selecionados para decidir se deve criar novas visualizações ou reutilizar as visualizações existentes que não estão visíveis na *viewport* (*RecylerListView* 2022).

BigList

O componente BigList foi desenvolvido tendo em conta o RecyclerListView, mas colmatando algumas deficiências que os seus programadores argumentam que existem: como a falta de cache, a estabilidade e a simplicidade - apresentando riscos para a permanência do cabeçalho e dos cálculos de deslocação dos itens.

Este componente também foi inspirado no conceito de Recycler List, para mostrar um largo número de conjunto de dados de forma eficiente. Assim, quando um item se desloca para fora do ecrã, o *recycler* não destrói totalmente a visualização, mas reutiliza a visualização para novos itens que se deslocam para o ecrã. Esta reutilização permite melhorar o desempenho visual, a capacidade de resposta ao toque e o consumo de energia, de acordo com os seus autores (*react-native-big-list* 2022).

No entanto, este componente tem algumas desvantagens. Quando a lista não consegue renderizar os elementos com rapidez suficiente, os componentes não renderizados aparecem com um espaço em branco, podendo dificultar a experiência visual do utilizador, e também só suporta listas de orientação vertical. Caso a aplicação necessita de uma lista, com orientação horizontal, o BigList não poderá ser utilizado.

SnapshotList

Ao longo da investigação, foram encontradas alternativas aos componentes mencionados acima, e por isso, tentou-se implementar uma versão experimental da FlatList, baseada num processo de tirar um *snapshot* aos elementos, antes de saírem do *viewport*, para posteriormente os mostrar, enquanto o componente não carrega esses elementos novamente.

A API da FlatList expõe várias configurações para otimizar a renderização e o conteúdo carregado. Estas otimizações vêm com algumas desvantagens, nomeadamente a baixa taxa de preenchimento dos itens da lista enquanto se deslocam rapidamente pelo ecrã, porque as visualizações são destruídas e recriadas preenchendo a *bridge* do React Native, com eventos

de *layout* e deixando o *Thread JS* sem capacidade para responder ao retorno da chamada para interações de toque.

Este componente foi desenvolvido com o intuito de resolver essas deficiências, solicitando ao serviço de backend os dados a serem exibidos para todos os itens e renderizando apenas as árvores de *layout* dos itens que estão visíveis no ecrã (*viewport*). Os itens que estão fora do ecrã também são renderizados, só que em vez de renderizarem totalmente a árvore de *layout*, é utilizado um *snapshot* com a representação visual.

Embora este componente tenha sido investigado, não foi destacado para tirar métricas e resultados, pois a sua implementação, aliada aos testes previamente efetuados, não foram capazes de superar o até então já alcançado por outras listas.

FlashList

O componente FlashList foi desenvolvido tendo em conta o componente FlatList, mas com um melhor desempenho (cerca de 5 vezes, no que diz respeito aos FPS da UI thread, e 10 vezes, no que toca aos FPS da JS thread) - segundo os seus programadores.

Este componente, tal como o BigList, também foi inspirado no conceito de Recyler List, para mostrar um largo número de conjunto de dados de forma eficiente. Assim, quando um item se desloca para fora do ecrã, o *recycler* não destrói totalmente a visualização, mas reutiliza a visualização para novos itens que se deslocam para o ecrã. Esta reutilização permite melhorar o desempenho visual, a capacidade de resposta ao toque e o consumo de energia, de acordo com os seus autores (*FlashList* 2022).

Para além das similiaridades com a FlatList, no que toca à implementação, também tem configurações extra, que oferecem um desempenho ainda melhor às listas criadas (*FlashList* 2022).

4.2.3 Análise da Implementação das soluções

Atualmente não há uma solução perfeita para a resolução dos problemas existentes. Cada abordagem utilizada tem vantagens e desvantagens. Com o objetivo de atingir a melhor solução possível, é necessário considerar especialmente as desvantagens de cada solução, a que é capaz de oferecer uma melhor experiência ao utilizador e também a que seja mais adequada ao nível do ponto de vista tecnológico do projeto em questão.

FlatList

A utilização da FlatList tem algumas vantagens como: a renderização dos itens de uma lista lentamente (conceito de *lazy*), reduzindo o uso de memória e o tempo de processamento; a renderização do conteúdo de forma assíncrona fora do ecrã, para restringir a memória e permitir um *scroll* mais suave; e a utilização da propriedade *getItemLayout* que remove a necessidade de se calcular sempre de forma dinâmica o *layout* da lista, pois a largura e altura de cada item é conhecida à *priori*.

Contudo, a sua utilização tem algumas desvantagens como: no caso da renderização assíncrona (para os itens que estão fora do ecrã) em que o *scroll* é mais rápido que a taxa de preenchimento, o ecrã mostra momentaneamente conteúdo em branco; apesar de criar e destruir as visualizações dos itens no ecrã, e fora dele ajudar a reduzir o uso da memória, é computacionalmente caro; o coletor de lixo (*garbage collector*) pode ser chamado inúmeras

vezes, pois as visualizações são destruídas à medida que o utilizador desliza pela aplicação; muitas das limitações são difíceis de se ultrapassar devido ao processo de *threading*; e os dados do *data adapter* não podem ser acedidos de forma síncrona, portanto é possível ver as visualizações a "piscar"à medida que são renderizadas de forma assíncrona durante um *scroll* rápido.

RecyclerListView

A utilização do RecyclerListView conta com algumas vantagens, como por exemplo, a implementação da estratégia de reciclar do JavaScript - para reutilizar as visualizações e evitar a recriação de novos itens que estão a ser mostrados no ecrã; em vez de desmontar, ele recupera a visualização fora do ecrã para renderizar novos itens; e suporta *providers* de dados e de *layout*, que através deles é possível manipular dados, o tipo de *layout* e as dimensões estimadas ou exatas.

Todavia, a sua utilização tem desvantagens, como por exemplo, quando a lista não puder renderizar itens com rapidez suficiente, os itens não renderizados irão aparecer como um espaço em branco no ecrã; não é fácil de migrar através da FlatList; pode ter conflitos com os cabeçalhos fixos que usam a API *Animated*; e a posição dos itens, durante o *scroll*, pode ser calculada erradamente.

BigList

A utilização do Biglist possui algumas vantagens, como por exemplo, tal como o RecylerListView, implementa a estratégia de reciclar do JavaScript; o algortimo do *recycler* cria dinamicamente os itens quando eles são necessários; e a reutilização da visualização melhora muito o desempenho e a capacidade de resposta.

Porém, a sua utilização também possui algumas desvantagens como: a propriedade *itemLayout* não permite itens com tamanhos dinâmicos, pois têm todos de ter as mesmas dimensões obrigatoriamente. Esta é uma enorme desvantagem, considerando o projeto Betfair Rebuild, ao qual este documento faz referência.

SnapshotList

A utilização do SnapshotList tem algumas vantagens, como por exemplo, a estratégia de virtualização tem em consideração os itens visíveis no ecrã, e não um tamanho fixo (por exemplo: 5 itens por ecrã); não há necessidade de criar e destruir visualizações à medida que ocorre um *scroll*, reduzindo o balbuciar da lista na aplicação; o conteúdo dinâmico é montado mais abaixo, em comparação com outras abordagens; e a taxa de preenchimento de itens no ecrã é mais alta, em comparação com as outras abordagens, porque os itens já são renderizados como *snapshots* no *scroll*.

Apesar disso, também tem algumas desvantagens, como por exemplo, tirar os *snaphots* da visualização pode sobrecarregar e congelar a *thread* principal, especialmente se for feito de uma só vez; e não é adequado para listas longas, devido ao maior uso de memória.

FlashList

A utilização da FlashList tem algumas vantagens, como por exemplo, tal como o RecylerListView e o BigList, implementa a estratégia de reciclar do JavaScript; o algortimo do *recycler*

cria dinamicamente os itens quando eles são necessários; e a reutilização da visualização melhora muito o desempenho e a capacidade de resposta.

Contudo, apresenta algumas desvantagens, como a necessidade de conhecer o tamanho de um item, para os resultados de desempenho serem melhorados, e não tendo todas as funcionalidades criadas na FlatList, o que em casos bastante específicos, pode não ser uma boa solução para uma migração.

4.3 Sumário

Neste capitulo foi descrita, numa abordagem técnica, a implementação de uma aplicação de testes (World of Lists), com o intuito de verificar, sob as mesmas condições, se os componentes tinham de facto mais desempenho que o FlatList. Foi também apresentada a arquitetura da aplicação Betfair Rebuild e World of Lists, sob forma de dar a conhecer ao leitor a razão de ser necessário a procura de uma melhor solução.

Por fim, foi realizada uma conceptualização dos componentes a investigar/testar, analisando as suas vantagens e desvantagens, tanto para a sua aplicação na aplicação Betfair Rebuild, como em qualquer aplicação desenvolvida em React Native.

Capítulo 5

Testes e Resultados

Este capítulo tem como objetivo avaliar e testar a solução investigada ao longo deste trabalho. Para tal, são definidas grandezas a avaliar, tal como testes estatísticos para comparar e analisar os diferentes resultados obtidos.

5.1 Avaliação da Solução

Nesta secção, é feita uma avaliação da solução proposta. Para tal, serão definidas as grandezas utilizadas na avaliação, as hipóteses a testar, a metodologia de avaliação e o teste estatístico que irá ser utilizado.

5.1.1 Solução Proposta

No contexto deste documento, a abordagem a realizar e a solução do problema já foram definidas pela empresa, no momento da criação do projeto. Foi definido que para encontrar uma solução para o problema seria necessário investigar sobre novas formas de melhorar o desempenho da aplicação, mais propriamente das listas encadeadas que fazem parte da mesma.

Posto isto, a solução proposta passa por otimizar a utilização das atuais listas encadeadas, com recurso a tecnologias já presentes no React Native, e posteriormente, utilizar um componente externo, que seja capaz de processar e apresentar uma lista, encadeada ou simples, de uma forma mais rápida, com maior fluidez de movimento e com menor consumo de memória.

Apesar de terem sido identificadas na Subsecção 2.4.3, há diversas tecnologias que poderiam ter um melhor desempenho, para o tipo de projeto que está em desenvolvimento. A escolha da tecnologia já tinha sido realizada, e por isso, é necessário adaptar essa escolha a uma nova solução capaz de contrariar esse mau desempenho.

Em suma, é necessário avaliar esta nova solução num ambiente, sob as mesmas condições que o atual, de forma a ser possível retirar métricas de desempenho e comprovar o seu sucesso. Para tal, será usado o React Native Profiler, assim como uma ferramenta denominada RN Perf Monitor (com o suporte da aplicação Flipper), de forma a conseguir-se retirar métricas, o mais eficientemente possível.

5.1.2 Ferramentas de Suporte

Para auxiliar na tiragem de amostras de teste, para os componentes em estudo, é necessário recorrer a ferramentas de suporte. Existem diversas ferramentas disponíveis, como o React

Native profiler. Mas, visto que são necessário informações mais detalhadas das métricas em análise, decidiu-se pela utilização da ferramenta RN Perf Monitor, como identificada abaixo.

RN Perf Monitor

O RN Perf Monitor é uma ferramenta criada com o propósito de auxiliar nos testes de desempenho das aplicações desenvolvidas em React Native. De forma a executar o seu processo de avaliação de desempenho, uma das métricas chave é o *frame rate*, que num cenário ideal deve rondar os 60fps (*Measuring and improving performance on a React Native app* 2022).

Esta métrica é avaliada em duas ocasiões, JS Thread e UI Thread, ou seja o consumo de memória executado pelas ações de JavaScript, e o consumo de memória executado pelas ações de UI (*Measuring and improving performance on a React Native app* 2022).

Tendo em vista que para um bom desempenho, o *frame rate* deve ser constante e não ser reduzido, esta ferramenta analisa, durante uns milissegundos previamente definidos, a taxa de *frame rate* existente na aplicação. Após essa análise, de forma a ser similar ao Google Lighthouse (ferramenta para avaliar desempenho de aplicações web), uma fórmula foi definida para obter um valor concreto do desempenho da aplicação, como demonstrado na Equação 5.1 (*Measuring and improving performance on a React Native app* 2022).

$$Score = \frac{(UIfps + JSfps) * 100}{120} * (1 - JSThreadlockpercentage) \qquad (5.1)$$

5.1.3 Grandezas a avaliar

De forma a cumprir o objetivo, estabelecido para este documento, é relevante analisar algumas grandezas que são consideradas de vital importância para um melhor desempenho da aplicação.

Para tal, o conjunto de grandezas a avaliar são:

- A variação de fps renderizados pela JS Thread;
- A variação de fps renderizados pela UI Thread;
- JS Threadlock;
- Valor médio de desempenho recebido.

5.1.4 Métricas

Nesta subsecção, serão apresentadas as metodologias que vão ser utilizadas no decorrer da avaliação da solução, relativamente às grandezas anteriormente referidas.

A variação de fps renderizados pela JS Thread

Desta forma, com o suporte da ferramenta RN Perf Monitor, torna-se necessário analisar o valor médio, ao fim do limite de tempo definido anteriormente (10000 ms), dos fps renderizados pela JS Thread, assim como analisar a variação observável no gráfico gerado, pela mesma ferramenta. Com o intuito de avaliar esta grandeza, será necessário testar sob as mesmas condições, após as otimizações realizadas e a introdução da solução escolhida.

A variação de fps renderizados pela UI Thread

Neste contexto, com o suporte da ferramenta, revela-se uma necessidade de analisar o valor médio, ao fim do limite de tempo estipulado previamente (10000 ms), dos fps renderizados pela UI Thread, tal como inspecionar a variação visível no gráfico gerado. Deste modo, para avaliar esta grandeza, será necessário testar sob as mesmas condições, após as otimizações realizadas e a introdução da solução escolhida.

JS Threadlock

Deste modo, com o auxílio da ferramenta, surge a necessidade de verificar o valor médio, ao fim do limite de tempo estabelecido previamente (10000 ms), dos valores da JS Threadlock. Com o propósito de avaliar esta grandeza, torna-se necessário testar sob as mesmas condições, após as otimizações realizadas e a introdução da solução escolhida.

Valor médio de desempenho recebido

Desta forma, com o suporte da ferramenta, é necessário analisar o valor médio de desempenho, ao fim do limite de tempo definido previamente (10000 ms), que será o valor resultante da resolução da equação 5.1. Com o intuito de avaliar esta grandeza, será necessário testar sob as mesmas condições, após as otimizações realizadas e a introdução da solução escolhida.

5.1.5 Teste de Hipóteses

Um teste de hipóteses é um "É uma metodologia estatística que nos auxilia a tomar decisões sobre uma ou mais populações baseado na informação obtida da amostra".(*Teste de Hipóteses* 2022)

Para um teste de hipóteses, existem duas Hipóteses:

- **Hipótese Nula**: quando é registada uma igualdade na comparação.
- **Hipótese Alternativa**: quando é registada uma diferença entre os resultados das amostras. Esta hipótese é dividida em dois tipos:
  - **Hipótese Bilateral**, quando a amostra registada é diferente.
  - **Hipótese Unilateral** (Superior), caso a amostra registada seja superior, ou (Inferior) caso a amostra registada seja inferior.

Para este caso de avaliação, serão consideradas a Hipótese Nula (apesar de ser muito pouco provável de acontecer), e as Hipóteses Alternativas Unilaterais Superior e Inferior. Posto isto, considera-se que $\mu 0$ representa o desempenho atual da aplicação e $\mu 1$ o desempenho da aplicação, com a implementação da solução escolhida.

Desta forma, pretende-se rejeitar, a Hipótese Nula 5.2 e a Hipótese Alternativa Unilateral Inferior 5.4, pois signifca que a implementação da solução escolhida não obteve o sucesso esperado. Caso se verifique a Hipótese Alternativa Unilateral Superior 5.3 pode-se assumir o objetivo principal deste documento cumprido.

$$h0 = \mu 0 = \mu 1 \tag{5.2}$$

$$h1 = \mu 0 < \mu 1 \tag{5.3}$$

$$h2 = \mu 0 > \mu 1 \tag{5.4}$$

5.2 Teste Estatístico

De forma a testar as métricas acima mencionadas, será realizado um teste estatístico. O teste escolhido foi o t-test, pois consiste num teste paramétrico que permite a comparação de duas amostras diferentes. Este teste permite avaliar o sucesso da nova solução.

Para a realização deste teste, serão recolhidas 10 amostras consecutivas, dos valores médios de desempenho, pois este valor já tem em consideração a variância de fps na JS Thread, UI Thread e, também tem em consideração a percentagem de JS Threadlock, como indicado na equação 5.1.

Estes testes são divididos em duas fases: análise e teste das métricas retiradas da aplicação de teste, World of Lists, com a utilização de um simulador android de baixa qualidade; e análise e teste das métricas retiradas da aplicação Betfair Rebuild, com a utilização do simulador do aparelho iPhone 13. Para o primeiro caso são tiradas métricas aos quatro componentes de listas existentes, num aparelho de baixa qualidade, para se perceber melhor os resultados. Para o segundo caso são tiradas métricas ao componente que está a ser utilizado atualmente, o FlatList, e a lista com o melhor desempenho obtido na aplicação de testes, neste caso o FlashList.

5.2.1 Aplicação World of Lists

A aplicação World of Lists foi desenvolvida com o propósito de testar, sob as mesmas condições, as diferentes listas escolhidas e investigadas como alternativas, para melhorar o desempenho de uma aplicação de React Native. Posto isto, métricas foram tiradas, com o suporte da ferramenta RN Perf Monitor, sendo expostas e analisadas com testes estatísticos, nos parágrafos abaixo.

FlatList

Iniciando pelo componente FlatList, componente que serve como amostra de comparação, com os outros componentes, foram tiradas 10 amostras, conforme o planeado, verificando-se, na Tabela 5.1, os resultados das mesmas.

Após o registo dos resultados obtidos, segue-se uma análise matemática para se encontrar a média ponderada e o desvio padrão, de forma a existirem meios para uma melhor análise e comparação de resultados.

Tabela 5.1: Resultados Métricas: FlatList (Anexo A)

Nº Teste	Média JS fps	Média UI fps	JS threadlock	Média desempenho
1	34.0 fps	46.1 fps	0.000s	66
2	33.7 fps	49.9 fps	0.000s	69
3	24.9 fps	54.5 fps	0.999s	60
4	17.2 fps	47.9 fps	1.982s	44
5	15.1 fps	42.5 fps	1.016s	43
6	19.4 fps	44.2 fps	0.483s	50
7	38.7 fps	46.5 fps	0.000s	71
8	38.4 fps	56.9 fps	0.500s	75
9	20.9 fps	54.5 fps	0.982s	57
10	20.3 fps	51.2 fps	0.965s	54

Média ponderada

$$d = \frac{d1 + d2 + ... + dn}{nAmostras} = \frac{66 + 69 + 60 + 44 + 43 + 50 + 71 + 75 + 57 + 54}{10} = 58.9 \tag{5.5}$$

Desvio padrão

$$o = \sqrt{\frac{(d1 - d)^2 + (d2 - d)^2 + ...(dn - d)^2}{nAmostras - 1}} = 11,259 \tag{5.6}$$

Após observação e análise da Tabela 5.1, verifica-se que o componente FlatList apresenta uma média de fps bastante reduzida e inconstante na JS Thread, sendo que na UI Thread os valores são bastante satisfatórios e equilibrados. Contudo, as médias de desempenho observadas declaram que a sua capacidade está além do desejável. Segundo a Equação 5.5, a média das 10 amostras equivale a 58.9% de média. Isto demonstra uma fragilidade quando comparado com listas mais complexas. Por outro lado, o seu desvio padrão, Equação 5.6, indica uma alta volatibilidade de valores, o que indica que este componente não é fiável em termos de resultados.

FlashList

Seguindo para o componente FlashList, um dos componentes em estudo, foram tiradas 10 amostras, conforme o planeado, verificando-se, na Tabela 5.2, os resultados das mesmas.

Após o registo dos resultados obtidos, segue-se uma análise matemática para se encontrar a média ponderada e o desvio padrão, de forma a existirem meios para uma melhor análise e comparação de resultados.

Tabela 5.2: Resultados Métricas: FlashList (Anexo B)

Nº Teste	Média JS fps	Média UI fps	JS threadlock	Média desempenho
1	50.7 fps	57.1 fps	0.000s	89
2	54.1 fps	57.5 fps	0.000s	93
3	49.3 fps	51.9 fps	0.000s	84
4	49.7 fps	52.7 fps	0.000s	85
5	47.4 fps	50.7 fps	0.000s	82
6	45.5 fps	47.4 fps	0.000s	77
7	49.0 fps	51.8 fps	0.000s	84
8	48.8 fps	51.7 fps	0.000s	83
9	46.7 fps	48.4 fps	0.000s	79
10	49.5 fps	56.4 fps	0.000s	88

Média ponderada

$$d = \frac{d1 + d2 + ... + dn}{nAmostras} = \frac{89 + 93 + 84 + 85 + 82 + 77 + 84 + 83 + 79 + 88}{10} = 84.4 \tag{5.7}$$

Desvio padrão

$$o = \sqrt{\frac{(d1 - d)^2 + (d2 - d)^2 + ...(dn - d)^2}{nAmostras - 1}} = 4.719 \tag{5.8}$$

Após observação e análise da Tabela 5.2, verifica-se que o componente FlashList apresenta uma média de fps bastante interessante e praticamente constante na JS Thread, sendo que na UI Thread os valores são bastante elevados e equilibrados. Por outro lado, as médias de desempenho observadas declaram que a sua capacidade é bastante atrativa. Segundo a Equação 5.7, a média das 10 amostras equivale a 84.4% de média. Isto demonstra uma capacidade notável quando comparado com listas mais complexas. Por outro lado, o seu desvio padrão, Equação 5.8, indica uma ligeira volatibilidade de valores, o que poderia indicar que este componente não é fiável em termos de resultados. Após análise, é observável que isso se deve a duas amostras díspares, que criam um desvio ligeiro nas amostras recolhidas.

BigList

Avançando para o componente BigList, um outro componente em estudo, foram analisadas 10 amostras, verificando-se os seus resultados, na Tabela 5.3.

Após o registo dos resultados obtidos, segue-se uma análise matemática para se encontrar a média ponderada e o desvio padrão, de forma a existirem meios para uma melhor análise e comparação de resultados.

Tabela 5.3: Resultados Métricas: BigList (Anexo C)

Nº Teste	Média JS fps	Média UI fps	JS threadlock	Média desempenho
1	18.2 fps	35.2 fps	1.033s	40
2	19.5 fps	40.0 fps	0.000s	49
3	32.3 fps	49.5 fps	0.000s	68
4	35.8 fps	54.5 fps	0.000s	75
5	39.3 fps	57.4 fps	0.000s	80
6	37.0 fps	55.4 fps	0.000s	77
7	38.3 fps	56.1 fps	0.000s	78
8	25.2 fps	42.8 fps	0.500s	53
9	36.8 fps	53.3 fps	0.000s	75
10	34.2 fps	51.7 fps	0.000s	71

Média ponderada

$$d = \frac{d1 + d2 + ... + dn}{nAmostras} = \frac{40 + 49 + 68 + 75 + 80 + 77 + 78 + 53 + 75 + 71}{10} = 66.6 \tag{5.9}$$

Desvio padrão

$$o = \sqrt{\frac{(d1 - d)^2 + (d2 - d)^2 + ...(dn - d)^2}{nAmostras - 1}} = 14.073 \tag{5.10}$$

Após observação e análise da Tabela 5.3, verifica-se que o componente BigList apresenta uma média de fps bastante reduzida, mas constante na JS Thread, sendo que na UI Thread os valores são bastante satisfatórios, mas com alguns desvios consideráveis. Por outro lado, as médias de desempenho observadas declaram que a sua capacidade fica além do desejável. Segundo a Equação 5.9, a média das 10 amostras equivale a 66.6% de média. Isto demonstra uma capacidade satisfatória quando comparado com listas mais complexas. Por outro lado, o seu desvio padrão, Equação 5.10, indica uma alta volatibilidade de valores, o que indica que este componente não é fiável em termos de resultados.

RecyclerListView

Por fim, conclui-se a tiragem de amostras com o componente RecyclerListView. Este componente tem uma forma de construção bastante diferente dos outros em estudo, mas foram analisadas 10 amostras, verificando-se os seus resultados, na Tabela 5.4.

Após o registo dos resultados obtidos, segue-se uma análise matemática para se encontrar a média ponderada e o desvio padrão, de forma a existirem meios para uma melhor análise e comparação de resultados.

Tabela 5.4: Resultados Métricas: RecyclerListView (Anexo D)

Nº Teste	Média JS fps	Média UI fps	JS threadlock	Média desempenho
1	50.8 fps	59.0 fps	0.000s	91
2	47.4 fps	57.1 fps	0.000s	87
3	45.7 fps	53.0 fps	0.000s	82
4	46.0 fps	51.4 fps	0.000s	81
5	36.9 fps	45.0 fps	0.000s	68
6	37.0 fps	43.3 fps	0.000s	66
7	39.1 fps	46.5 fps	0.000s	71
8	35.6 fps	45.8 fps	0.000s	67
9	40.4 fps	50.7 fps	0.000s	75
10	46.3 fps	57.7 fps	0.000s	86

Média ponderada

$$d = \frac{d1 + d2 + ... + dn}{nAmostras} = \frac{91 + 87 + 82 + 81 + 68 + 66 + 71 + 67 + 75 + 86}{10} = 77.6 \tag{5.11}$$

Desvio padrão

$$o = \sqrt{\frac{(d1 - d)^2 + (d2 - d)^2 + ...(dn - d)^2}{nAmostras - 1}} = 9.180 \tag{5.12}$$

Após observação e análise da Tabela 5.4, verifica-se que o componente RecyclerListView apresenta uma média de fps bastante satisfatória, mas muito volátil na JS Thread, sendo que na UI Thread os valores são bastante satisfatórios, com pequenos desvios. Por outro lado, as médias de desempenho observadas impossibilitam definir a sua capacidade, pois o seu desvio padrão, Equação 5.12, é demasiado elevado, o que indica que este componente não é fiável em termos de resultados. Segundo a Equação 5.11, a média das 10 amostras equivale a 77.6% de média. Isto demonstra uma capacidade interessante quando comparado com listas mais complexas.

5.2.2 Betfair Rebuild

Após análise dos componentes na aplicação World of Lists, o que obteve melhor média de resultados, aliado ao desvio padrão revelado - FlashList -, foi escolhido para ser comparado com o FlatList, que atualmente está implementado na aplicação Betfair Rebuild.

FlatList

Iniciando pelo componente FlatList, componente que serve como amostra de comparação, com os outros componentes, foram tiradas 10 amostras, conforme o planeado, verificando-se, na Tabela 5.5, os resultados das mesmas.

Após o registo dos resultados obtidos, segue-se uma análise matemática para se encontrar a média ponderada e o desvio padrão, de forma a existirem meios para uma melhor análise e comparação de resultados.

Tabela 5.5: Resultados Métricas: FlatList (Anexo E)

Nº Teste	**Média JS fps**	**Média UI fps**	**JS threadlock**	**Média desempenho**
1	30.3 fps	59.8 fps	2.500s	56
2	19.3 fps	60.0 fps	5.000s	33
3	22.4 fps	59.9 fps	4.000s	42
4	12.5 fps	69.2 fps	5.500s	27
5	10.4 fps	59.9 fps	5.000s	28
6	20.2 fps	59.8 fps	4.500s	36
7	16.5 fps	59.4 fps	5.000s	33
8	31.0 fps	59.7 fps	5.000s	31
9	24.0 fps	59.4 fps	6.000s	24
10	16.3 fps	59.4 fps	3.500s	40

Média ponderada

$$d = \frac{d1 + d2 + ... + dn}{nAmostras} = \frac{56 + 33 + 42 + 27 + 28 + 36 + 33 + 31 + 24 + 40}{10} = 35 \tag{5.13}$$

Desvio padrão

$$o = \sqrt{\frac{(d1 - d)^2 + (d2 - d)^2 + ...(dn - d)^2}{nAmostras - 1}} = 9.274 \tag{5.14}$$

Após observação e análise da Tabela 5.5, verifica-se que o componente FlatList apresenta uma média de fps bastante deficitária, e com enorme volatibilidade na JS Thread, sendo que na UI Thread os valores são bastante elevados e constantes. Contudo, as médias de desempenho observadas demonstram uma capacidade basta reduzida, quando a sua utilização na aplicação. Segundo a Equação 5.13, a média das 10 amostras equivale a 35% de média. Isto demonstra uma capacidade muito aquém do desejável. Por fim, o seu desvio padrão, Equação 5.14, é bastante elevado, o que indica uma inconsistência de dados, e tornando este componente muito pouco fiável.

FlashList

Por fim, analisando o componente FlashList, um dos componentes em estudo, foram tiradas 10 amostras, conforme o planeado, verificando-se, na Tabela 5.6, os resultados das mesmas.

Após o registo dos resultados obtidos, segue-se uma análise matemática para se encontrar a média ponderada e o desvio padrão, de forma a existirem meios para uma melhor análise e comparação de resultados.

Tabela 5.6: Resultados Métricas: FlashList (Anexo F)

Nº Teste	Média JS fps	Média UI fps	JS threadlock	Média desempenho
1	54.4 fps	60 fps	0.000s	95
2	50.4 fps	60 fps	1.000s	82
3	57.5 fps	60 fps	0.000s	97
4	57.1 fps	60 fps	0.000s	97
5	56.7 fps	60 fps	0.000s	97
6	57.4 fps	60 fps	0.000s	97
7	56.1 fps	60 fps	0.000s	96
8	53.2 fps	60 fps	0.500s	89
9	47.6 fps	60 fps	1.000s	81
10	56.7 fps	60 fps	0.000s	97

Média ponderada

$$d = \frac{d1 + d2 + ... + dn}{nAmostras} = \frac{95 + 82 + 97 + 97 + 97 + 97 + 96 + 89 + 81 + 97}{10} = 92.8 \tag{5.15}$$

Desvio padrão

$$o = \sqrt{\frac{(d1 - d)^2 + (d2 - d)^2 + ...(dn - d)^2}{nAmostras - 1}} = 6.443 \tag{5.16}$$

Após observação e análise da Tabela 5.6, verifica-se que o componente FlashList apresenta uma média de fps bastante promissora, e com enorme consistência de valores na JS Thread, sendo que na UI Thread os valores são bastante elevados e constantes. Por outro lado, as médias de desempenho observadas demonstram uma capacidade basta impressionante, quando a sua utilização na aplicação. Segundo a Equação 5.15, a média das 10 amostras equivale a 92.8% de média. Isto demonstra uma capacidade impressionante e promissora para a sua utilização. Por fim, o seu desvio padrão, Equação 5.16, indica alguma inconsistência. Porém, após análise, verifica-se que isso se deve a duas amostras, que se revelaram deficitárias.

5.2.3 Análise dos Resultados

De modo a analisar os resultados de forma criteriosa e imparcial, são analisados os valores médios de desempenho de cada componente (quanto maior a média, melhor é o desempenho do componente), o desvio padrão (quanto menor o desvio padrão, mais estável e fiáveis são os resultados) e o resultado do teste de hipóteses, entre duas amostras independentes, a amostra que se pretende contrariar (FlatList) e as amostras concorrentes (FlashList, BigList e RecyclerListView).

Para calcular as amostras independentes, é necessário recorrer à fórmula presente na Equação 5.17, e posteriormente utilizar a tabela t (Tabela 5.7), com o valor de significância escolhido, neste caso 1%, para validar os resultados. Posto isto, é necessário analisar o Teste de

hipóteses, mencionado na subsecção 5.1.5, podendo existir 3 resultados possíveis: Hipótese Nula, quando os resultados não sofreram variações, Hipótese Bilateral, quando os resultados são diferentes, ou Hipótese Unilateral Superior ou Inferior, caso os resultados das amostras sejam superiores ou inferiores respetivamente.

$$t = \frac{d1 - d2}{\sqrt{(\frac{nAmostras}{\frac{o1^2+o2^2}{2}}}]} \tag{5.17}$$

Tabela 5.7: Tabela T: (Nivel Significância 1%)

gl	**20%**	**10%**	**5%**	**1%**
1	3.078	6.134	12.706	63.656
2	1.886	2.920	4.303	9.925
5	1,476	2,015	2,571	4,032
10	1,372	1,813	2,228	3,169
...	...	...	...	...
18	1,330	1,734	2,101	2,878

World of Lists

Assumindo o componente FlatList, como a hipótese a ser debatida, doravante será denominada de $\mu 0$. Por outro lado, os componentes FlashList, BigList e RecyclerListView, doravante serão denominados de $\mu 1$, $\mu 2$ e $\mu 3$, respetivamente.

Analisando a Tabela 5.8, verificam-se as seguintes condições:

Tabela 5.8: Análise dos resultados: World of Lists

Componente	**Média (d)**	**Desvio Padrão (o)**
FlatList	58.9	11.259
FlashList	84.4	4.719
BigList	66.6	14.073
RecyclerListView	77.6	9.180

FlatList($\mu 0$) vs FlashList($\mu 1$) A amostra $\mu 0$ tem como valor médio 58.9 e desvio padrão de 11.259. Já a amostra $\mu 1$ tem como valor médio 84.4 e desvio padrão de 4.719. Posto isto, procede-se ao seguinte cálculo:

$$t = \frac{d1 - d2}{\sqrt{(\frac{nAmostras}{\frac{o1+o2}{2}}}]} = (58.9 - 84.4).\sqrt{\frac{10}{\frac{11.259^2+4.719^2}{2}}} = 6.605 \tag{5.18}$$

Neste caso, verificando a Equação 5.18, ressalva-se o t-test de valor 6.605. Com isto, quer dizer que h0=$\mu 0 < \mu 1$. Posto isto, podemos afirmar que a hipótese nula é rejeitada, e que a

hipótese unilateral superior é aceite, logo prova-se que o FlashList tem melhor desempenho que o FlatList.

FlatList(μ0) vs BigList(μ2) A amostra μ0 tem como valor médio 58.9 e desvio padrão de 11.259. Já a amostra μ2 tem como valor médio 66.6 e desvio padrão de 14.073. Posto isto, procede-se ao seguinte cálculo:

$$t = \frac{d1 - d2}{\sqrt{(\frac{nAmostras}{\frac{o1^2+o2^2}{2}}}] = (58.9 - 66.6).\sqrt{\frac{10}{\frac{11.259^2+14.073^2}{2}}} = 1.351 \quad (5.19)$$

Neste caso, verificando a Equação 5.19, ressalva-se o t-test de valor 1.351. Com isto, quer dizer que h0=μ0<μ2. Posto isto, podemos afirmar que a hipótese nula é rejeitada, e que a hipótese unilateral superior é aceite, logo prova-se que o BigList tem melhor desempenho que o FlatList.

FlatList(μ0) vs RecyclerListView(μ3) A amostra μ0 tem como valor médio 58.9 e desvio padrão de 11.259. Já a amostra μ2 tem como valor médio 77.6 e desvio padrão de 9.180. Posto isto, procede-se ao seguinte cálculo:

$$t = \frac{d1 - d2}{\sqrt{(\frac{nAmostras}{\frac{o1^2+o2^2}{2}}}] = (58.9 - 77.6).\sqrt{\frac{10}{\frac{11.259^2+9.180^2}{2}}} = 4.071 \quad (5.20)$$

Neste caso, verificando a Equação 5.19, ressalva-se o t-test de valor 4.071. Com isto, quer dizer que h0=μ0<μ2. Posto isto, podemos afirmar que a hipótese nula é rejeitada, e que a hipótese unilateral superior é aceite, logo prova-se que o RecyclerListView tem melhor desempenho que o FlatList.

Conclusão Os dados da Tabela 5.8 revelam que o componente com melhor média de desempenho é o FlashList, tal como o componente com menor desvio padrão. Também é revelado, a partir dos cálculos efetuados pelo t-test, que todos os componentes alternativos são mais capazes que o FlatList. No entanto, o componente FlashList é o que obteve maior valor. Isto indica que este componente é o que tem mais capacidade para ser utilizado na aplicação World of Lists, pois é o que tem melhor desempenho e com valores mais constantes e fiáveis.

Betfair Rebuild

Assumindo o componente FlatList, como a hipótese a ser debatida, doravante será denonimada de μ0 e o componente FlashList μ1.

Analisando a Tabela 5.9, verificam-se as seguintes condições:

FlatList(μ0) vs FlashList(μ1) A amostra μ0 tem como valor médio 35 e desvio padrão de 9.274. Já a amostra μ1 tem como valor médio 92.8 e desvio padrão de 6.443. Posto isto, procede-se ao seguinte cálculo:

Tabela 5.9: Análise dos resultados: Betfair Rebuild

Componente	**Média (d)**	**Desvio Padrão (o)**
FlatList	35	9.274
FlashList	92.8	6.443

$$t = \frac{d1 - d2}{\sqrt{\left(\frac{nAmostras}{\frac{o1+o2}{2}}\right)}}] = (58.9 - 84.4).\sqrt{\frac{10}{\frac{11.259^2+4.719^2}{2}}} = 16.186 \tag{5.21}$$

Neste caso, verificando a Equação 5.18, ressalva-se o t-test de valor 16.186. Com isto, quer dizer que h0=μ0<μ1. Posto isto, podemos afirmar que a hipótese nula é rejeitada, e que a hipótese unilateral superior é aceite, logo prova-se que o FlashList tem melhor desempenho que o FlatList.

Conclusão Os dados da Tabela 5.9 revelam que o componente com melhor média de desempenho é o FlashList, tal como o componente com menor desvio padrão. Também é revelado, a partir dos cálculos efetuados pelo t-test, que o componente FlashList é mais capaz. Isto indica que este componente é o que tem mais capacidade para ser utilizado na aplicação Betfair Rebuild, pois é o que tem melhor desempenho e com valores mais constantes e fiáveis.

5.3 Sumário

Neste capitulo foram apresentados os testes a efetuar às métricas recolhidas pela ferramenta RN Perf Monitor, de forma a aferir o desempenho de cada componente. Para atingir este objetivo foi necessário definir as grandezas a avaliar, bem como as hipóteses a testar. Na secção Teste Estatístico foram demonstrados, os resultados das testes especificados e chegou-se à conclusão que o componente FlashList é o que tem melhor desempenho, seja na aplicação World of Lists, seja na aplicação Betfair Rebuild.

Posto isto, define-se o objetivo a que se propôs este documento com sucesso, pois encontrou-se uma alternativa válida à atual implementação de listas usadas em React Native.

Contudo, poderiam ter sido realizados mais testes, principalmente recolhendo mais métricas, de forma a se obter um número de amostras superior, e também realizar-se testes, sob as mesmas condições, em diversos dispositivos, sob forma de comprovar que o componente é eficaz em qualquer dispositivo.

Capítulo 6

Conclusões e Trabalho Futuro

Com o desenvolvimento deste trabalho foi abordado, investigado e analisado um problema bastante impactante no desenvolvimento e utilização de aplicações móveis, baseadas na tecnologia React Native. Mais propriamente, no fraco desempenho demonstrado, quando utilizadas listas encadeadas. O objetivo principal deste estudo foi encontrar uma possível solução para o problema em questão, com base na migração de um componente alternativo, capaz de superar o atual componente utilizado. Para testar a viabilidade deste estudo foi construída uma aplicação, denonimada World of Lists, na qual foram efetuados os respetivos testes.

Por fim, neste último capítulo serão apresentadas as conclusões finais do estudo efetuado neste trabalho. Será apresentada uma contextualização do problema e, de seguida, uma avaliação do trabalho efetuado relativamente aos objetivos definidos. Os pontos que não foram realizados serão deliberados, com o intuito de incorporar em desenvolvimentos futuros. Por fim, é feita a apreciação final sobre o estudo.

6.1 Resumo

A Blip constitui-se como uma empresa dedicada ao desenvolvimento de *software* para aplicações *web*, destacando-se um novo projeto (atualmente em desenvolvimento) para a Betfair. Este projeto inovador é gerado em React Native - uma tecnologia multiplataforma - tendo em vista a redução de custos associados ao seu desenvolvimento e à possibilidade de partilha de código (no que diz respeito à lógica de negócio) entre aplicações.

Para além da utilização de uma tecnologia inovadora, este projeto destaca-se pelo seu enquadramento numa área de negócio como o mercado de apostas desportivas. Este mercado encontra-se em franca expansão, no qual a Betfair não quer perder a oportunidade de se integrar, visto que consiste numa das casas de apostas mais conhecidas ao nível global.

Contudo, esta tipologia de mercado necessita de aplicações significativamente otimizadas em termos de desempenho, uma vez que existe um elevado fluxo de dados em constante alteração. Por sua vez, esta tecnologia ainda possui diversos problemas - destacando-se o problema resolvido neste documento.

A tecnologia React Native, aquando da presença de listas encadeadas, demonstra uma enorme fragilidade em termos de desempenho - tornando a experiência dos seus utilizadores bastante limitada, e por vezes, vaga. Tendo em conta a área de negócio em que o projeto se insere, um desempenho otimizado é vital para o sucesso da aplicação e do negócio em si.

Na utilização de FlatList, como componente utilizado para renderizar listas encadeadas, (com um fluxo de dados em constante mutação), a aplicação mostra-se bastante lenta, com a probabilidade relativamente elevada de uma amostra com dados incoerentes, e ainda, uma navegação pouco fluída pela aplicação. Este aspeto pode revelar alguma dubiedade para os utilizadores, assim como pouca recetividade à sua utilização.

Após análise e teste das diversas alternativas consideradas, foi percetível uma análise comum, tanto na aplicação de testes World of Lists, como na aplicação da Betfair. Essa análise comum foi a distinção do componente FlashList, como o mais fiável e com melhor desempenho, quando presenteado com listas encadeadas.

Este componente obteve as melhores médias de desempenho, chegando por vezes a valores de cerca 96% de desempenho, o que retrata uma capacidade face ao problema bastante atrativa. Também se comprovou que este componente era o mais constante, entre os investigados, tendo valores de desvio padrão mais reduzidos, quando comparados com os outros componentes. Isto pode indicar uma adpatação do componente, seja qual for a ação executada pelo utilizador na aplicação em questão.

De todos os componentes, o FlashList também é o componente mais similar ao FlatList, na forma de construção e definição das suas *props*, o que o torna bastante vantajoso no caso de ser utilizado numa migração com o FlatList.

6.2 Objectivos Concretizados

Para o desenvolvimento deste trabalho, e posterior procura de uma solução, foram definidos certos objetivos, que serão deliberados como realizados no decorrer deste capítulo. Esses objetivos tinham o foco na investigação de alternativas à implementação da FlatList; melhorar o código atualmente em desenvolvimento, seguindo boas práticas de programação; a criação de uma aplicação, com o objetivo de testar e analisar as alternativas; e posterior implementação da alternativa e contrada e teste na aplicação da Betfair. Todos os requisitos não funcionais foram averiguados, sendo que alguns, por opção do autor, apenas foram analisados visualmente, sem o retirar de amostras.

Todos os objetivos foram cumpridos, sendo que poderiam ter sido aprofundados alguns pontos, como a criação de uma lista em código nativo, utilizando a nova arquitetura do React Native e o JSI Modules, tal como a criação de uma aplicação em linguagem nativa, de forma a comparar o seu desempenho com a aplicação de testes criada.

6.3 Trabalho Futuro

Apesar de o objetivo ter sido cumprido com bastante sucesso, ficou por analisar e testar a implementação de uma lista nativa, visto que a nova arquitetura do React Native assim o permite. No entanto, fica uma breve investigação efetuada para essa implementação, como forma de deixar em aberto uma futura continuação deste estudo.

6.3.1 JSI Investigação

Esta abordagem tem como critério principal a utilização de código nativo, para a criação de um componente capaz de obter um desempenho bastante elevado, em comparação com os componentes atualmente produzidos no React Native. Para tal, foi pensada uma solução

desenvolvida em Java/Kotlin e Objective-C/Swift. Mas, após uma investigação, concluiu-se, que a melhor abordagem, seria o desenvolvimento em C++, pois assim conseguiriam resolver algumas falhas da solução previamente pensada.

Ou seja, a atual arquitetura do React Native Bridge entre Native e o JavaScript funciona de forma assíncrona e transfere dados apenas em JSON. Com isto, seguem alguns problemas como (*JSI and JavaScript Core Discussion* 2022):

- **Chamadas assíncronas**
  - Muitos *threads* e saltos entre eles: JavaScript, Shadow, Main, Native...
  - Os *JS threads* e Main não comunicam diretamente entre si (resulta uma renderização lenta da interface do utilizador)
- **JSON**
  - Não há partilha de dados entre JS e *threads* nativos.
  - Transferência de dados lenta devido à serialização JSON

O React Native JSI fornece a API para o mecanismo JS Runtime e permite expor o código nativo diretamente ao JavaScript, sem necessidade de utilizar uma Bridge.

Posto isto, fornece as seguintes vantagens:

- Sincroniza as chamadas da *JS Thread* para Nativo e vice-versa.
- Partilha de dados entre threads
- Faz chamadas diretas para a thread principal da interface do utilizador, o que promove uma renderização mais rápida.

Pode-se utilizar C++, apenas para trabalhar com o JSI, porque o JS Runtime possui uma API de C++, onde é possível criar uma camada de C++, entre o JSI e o código base de Java ou Swift presente (*JSI and JavaScript Core Discussion* 2022).

6.4 Apreciação Final

O estudo desenvolvido foi bastante benéfico para o seu autor, que teve a oportunidade de investigar e aprofundar os seus conhecimentos relativo a aplicações móveis e à tecnologia React Native. Contudo, fica a sensação de querer saber mais, deixando em aberto a investigação incompleta sobre o JSI Modules e a sua utilização para a implementação de um componente nativo, capaz de superar qualquer dificuldade provocada por uma lista encadeada.

Bibliografia

About Objective-C (2022). url: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html (acedido em 21/02/2022).

Alexandre Leite e Helena Macedo (mai. de 2018). «COMPARATIVO ENTRE SISTEMAS OPERACIONAIS MÓVEIS – ANDROID X IOS». Em: *SIMTEC - Simpósio de Tecnologia da Fatec Taquaritinga* 4.1. Section: Tecnologia em Informática. url: https://simtec.fatectq.edu.br/index.php/simtec/article/view/253 (acedido em 15/02/2022).

Android and iOS (2022). url: https://home.ubalt.edu/abento/315/android-ios/index.html (acedido em 15/02/2022).

Apple (2022a). *Swift - Fast and Interative*. en. url: https://developer.apple.com/swift/#fast (acedido em 23/02/2022).

– (2022b). *Swift - Package Manager*. en. url: https://developer.apple.com/swift/#package-manager (acedido em 23/02/2022).

– (2022c). *Swift - Safety*. en. url: https://developer.apple.com/swift/#safety (acedido em 23/02/2022).

– (2022d). *Swift.org*. en. url: https://swift.org (acedido em 18/02/2022).

Arquitetura da plataforma | Desenvolvedores Android (2022). pt-BR. url: https://developer.android.com/guide/platform?hl=pt-br (acedido em 14/02/2022).

Bellinaso, Marco (jul. de 2020). *Flutter: the good, the bad and the ugly*. en. url: https://medium.com/asos-techblog/flutter-vs-react-native-for-ios-android-app-development-c41b4e038db9 (acedido em 16/02/2022).

Betfair » Como apostar na maior Bolsa de Apostas do mundo? (2022). url: https://sitedeapostas.com/guias/bolsa-de-apostas (acedido em 26/02/2022).

Blip | Blip, a Flutter Company (2022). url: https://blip.pt/about-us/ (acedido em 26/02/2022).

Câmara, Rafael (abr. de 2018). *O que você deve saber sobre o funcionamento do React Native*. en. url: https://medium.com/tableless/o-que-voce-deve-saber-sobre-o-funcionamento-do-react-native-7e3c610aa268 (acedido em 21/02/2022).

Chung, Lawrence e Julio Cesar Sampaio do Prado Leite (2009). «On Non-Functional Requirements in Software Engineering». Em: *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos*. Ed. por Alexander T. Borgida et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 363–379. isbn: 978-3-642-02463-4. doi: 10.1007/978-3-642-02463-4_19. url: https://doi.org/10.1007/978-3-642-02463-4_19.

Como surgiram as apostas desportivas? - Jornal o Interior (2022). url: https://ointerior.pt/publicidade/como-surgiram-as-apostas-desportivas/ (acedido em 11/10/2022).

Core·React Native (2022). en. url: https://reactnative.dev/docs/intro-react-native-components (acedido em 21/02/2022).

Exchange: Get Started with Betfair Exchange (2022). url: https://support.betfair.com/app/answers/detail/a_id/6490/ (acedido em 16/02/2022).

FlashList (2022). en. url: https://shopify.github.io/flash-list/ (acedido em 20/09/2022).

Flutter documentation (2022). en. url: https://docs.flutter.dev/ (acedido em 22/02/2022).

Flutter Just Might Work (mai. de 2019). en-US. Section: Development. url: https://www.donnfelker.com/flutter-just-might-work/ (acedido em 16/02/2022).

Flutter Releases (2022). pt. url: https://twitter.com/FlutterReleases (acedido em 16/02/2022).

Funcionamento do Sistema Operacional ios (2022). en. url: https://silo.tips/download/funcionamento-do-sistema-operacional-ios (acedido em 14/10/2022).

Github Language Stats (2022). url: https://madnight.github.io/githut/#/pushes/2021/4 (acedido em 16/02/2022).

Gosling, James et al. (2021). «The Java® Language Specification». en. Em: p. 848.

JSI and JavaScript Core Discussion (2022). en. url: https://github.com/react-native-community/discussions-and-proposals/issues/91 (acedido em 20/09/2022).

Koen, Peter A. et al. (2002). «1 Fuzzy Front End : Effective Methods , Tools , and Techniques». Em.

Kohn, Karen e Cláudia Herte de Moraes (2007). «O impacto das novas tecnologias na sociedade: conceitos e características da Sociedade da Informação e da Sociedade Digital». pt. Em: p. 13.

Kotlin Multiplatform | Kotlin (2022). en-US. url: https://kotlinlang.org/docs/multiplatform.html (acedido em 18/02/2022).

Manning, Nick (2022). *What It Was Like to Write a Full Blown Flutter App | HackerNoon*. en. url: https://hackernoon.com/what-it-was-like-to-write-a-full-blown-flutter-app-330d8202825b (acedido em 16/02/2022).

Martins, Carlos (out. de 2018). «Xamarin vs. Android». pt-PT. Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia. Porto: ISEP - Instituto Superior de Engenharia do Porto.

Measuring and improving performance on a React Native app (2022). fr. url: https://www.bam.tech/article/measuring-and-improving-performance-on-a-react-native-app (acedido em 12/10/2022).

Mobile Operating System Market Share Worldwide (2022). en. url: https://gs.statcounter.com/os-market-share/mobile/worldwide (acedido em 12/02/2022).

Morgan, Jasper (fev. de 2019). *Multi vs Cross Platform in the age of Flutter*. en. url: https://medium.com/snapp-mobile/multi-vs-cross-platform-in-the-age-of-flutter-6e76920028b6 (acedido em 16/02/2022).

Native Modules · React Native (2022). en. url: https://reactnative.dev/docs/0.62/native-modules-android (acedido em 16/02/2022).

Nicola, Susana (nov. de 2020a). *Análise de Valor*. en. presentantion. ISEP. url: https://moodle.isep.ipp.pt/pluginfile.php/147880/mod_resource/content/1/Aula%201_18_Nov_2020.pdf.

– (nov. de 2020b). *Análise de Valor - Método de Análise Hierárquica*. en. presentantion. ISEP. url: https://moodle.isep.ipp.pt/pluginfile.php/187507/mod_resource/content/1/An%C3%A1lise_Valor_Aula_4_21NOV_2018_1hora_AHP.pdf.

Novac, Ovidiu Constantin et al. (jun. de 2017). «Comparative study of Google Android, Apple iOS and Microsoft Windows Phone mobile operating systems». Em: *2017 14th International Conference on Engineering of Modern Electric Systems (EMES)*, pp. 154–159. doi: 10.1109/EMES.2017.7980403.

Peal, Gabriel (jun. de 2018a). *React Native at Airbnb*. en. url: https://medium.com/airbnb-engineering/react-native-at-airbnb-f95aa460be1c (acedido em 05/06/2022).

– (jun. de 2018b). *React Native at Airbnb: The Technology*. en. url: https://medium.com/airbnb-engineering/react-native-at-airbnb-the-technology-dafd0b43838 (acedido em 05/06/2022).

React Native · Learn once, write anywhere (2022). url: https://reactnative.dev/ (acedido em 18/02/2022).

React Native JSI (2022). *React Native JSI: Part 1 - Getting Started*. en. url: https://blog.notesnook.com/getting-started-react-native-jsi/ (acedido em 19/02/2022).

react-native-big-list (2022). en. url: https://www.npmjs.com/package/react-native-big-list (acedido em 05/06/2022).

RecylerListView (2022). en. url: https://www.npmjs.com/package/recyclerlistview (acedido em 05/06/2022).

Rempel, Jeremy (mar. de 2019). *Why we need Kotlin Native*. en. url: https://medium.com/android-things/why-we-need-kotlin-native-adacc03e988c (acedido em 16/02/2022).

Ribeiro, André (nov. de 2014). «Development of Mobile Applications using a Model-Driven Software Development Approach». pt-PT. Information Systems and Computer Engineering. Lisboa: Universidade Técnico Lisboa.

Saaty, Thomas L (1990). *How to make a decision: the analytic hierarchy process*. en. article. European journal of operational research 48.1, pp. 9–26.

Sobre JavaScript - JavaScript | MDN (2022). en-US. url: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript (acedido em 21/02/2022).

Sportsbook: How can I use my free bet? (2022). url: https://support.betfair.com/app/answers/detail/a_id/2 (acedido em 16/02/2022).

SRIJ (2022). *Registo da atividade de jogo online em Portugal*. url: https://www.srij.turismodeportugal.pt/fotos/editor2/estatisticas/Estat%C3%ADstica_online_1T_2021.pdf.

Stability of Kotlin components (2022). en-US. url: https://kotlinlang.org/docs/components-stability-pre-1-4.html (acedido em 16/02/2022).

Stack Overflow Trends (2022). url: https://insights.stackoverflow.com/trends?tags=react-native%2Cflutter%2Ckotlin%2Cswift (acedido em 16/02/2022).

Teste de Hipóteses (2022). url: https://www.inf.ufsc.br/~andre.zibetti/probabilidade/teste-de-hipoteses.html (acedido em 12/10/2022).

Threading Model · React Native (2022). en. url: https://reactnative.dev/docs/threading-model (acedido em 21/02/2022).

TurboModules (2022). en. url: https://github.com/react-native-community/discussions-and-proposals/issues/40 (acedido em 21/02/2022).

Usabilidade (2022). pt-PT. url: http://usabilidade.gov.pt/ (acedido em 12/02/2022).

What is the Betfair Exchange? (2022). en. url: https://betting.betfair.com/how-to-use-betfair-exchange/beginner-guides/what-is-the-betfair-exchange-010819-51.html (acedido em 26/02/2022).

When React Native is not a Good Choice for a Mobile Application Development (2022). en. url: https://www.netguru.com/blog/is-react-native-good (acedido em 16/02/2022).

Apêndice A

Anexos

A.1 FlatList: World of Lists

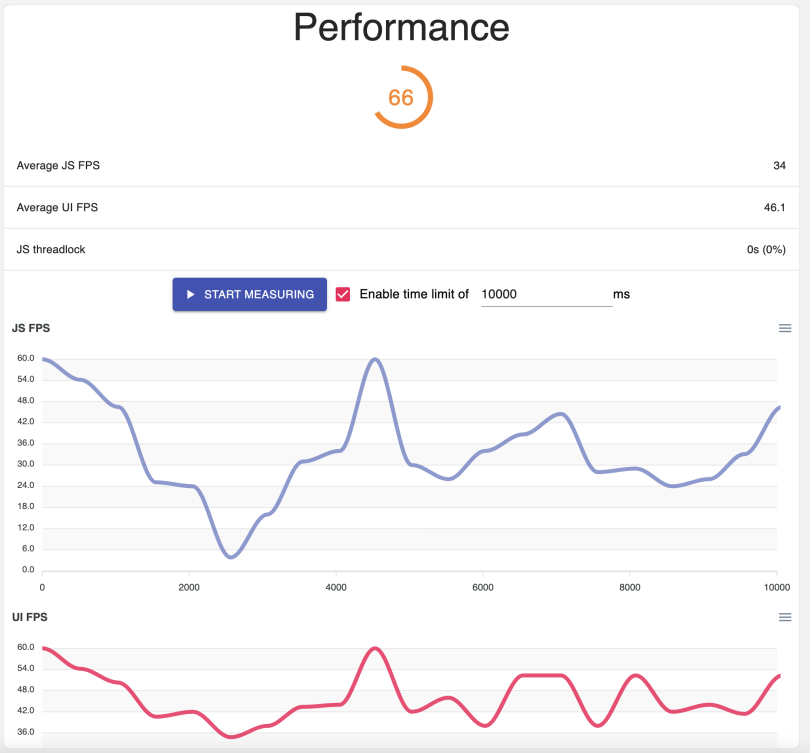

Figura A.1: FlatList Teste 1

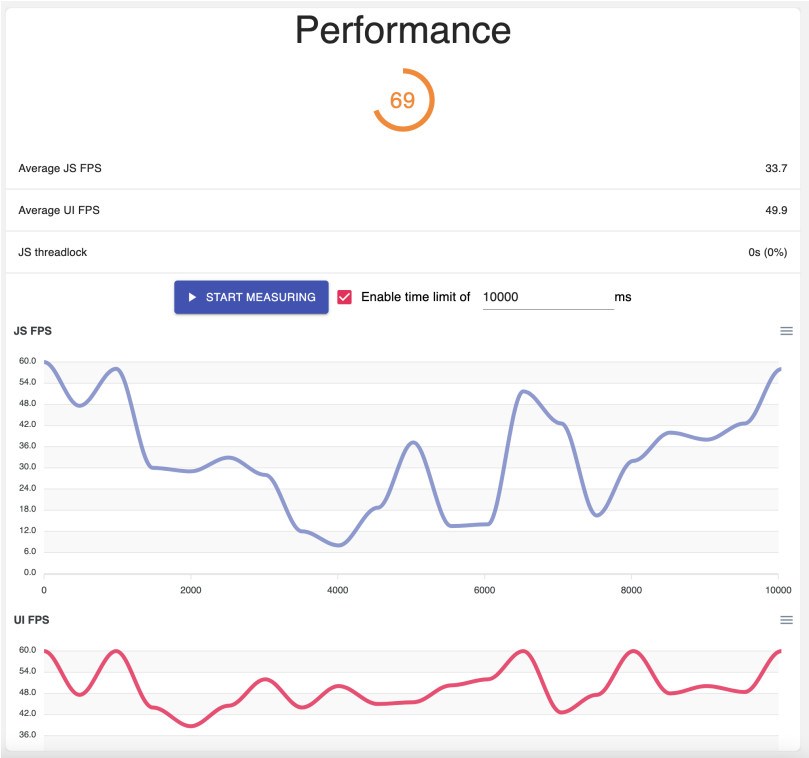

Figura A.2: FlatList Teste 2

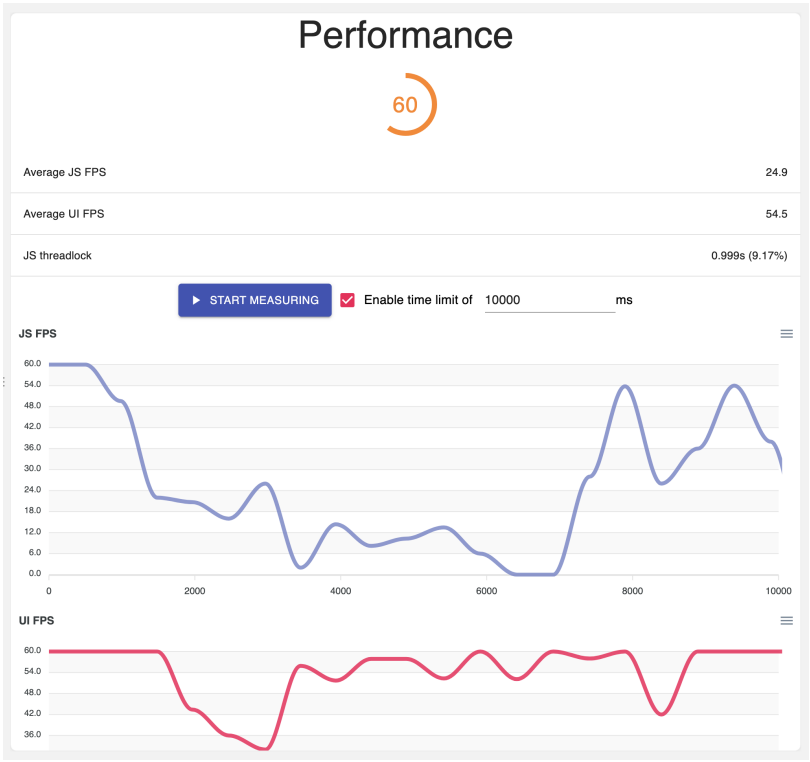

Figura A.3: FlatList Teste 3

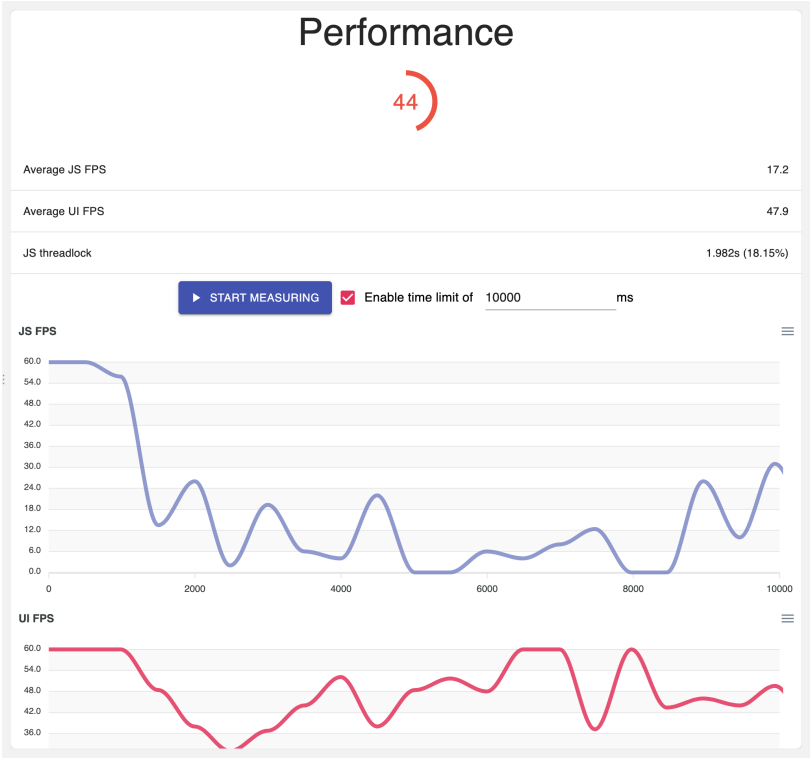

Figura A.4: FlatList Teste 4

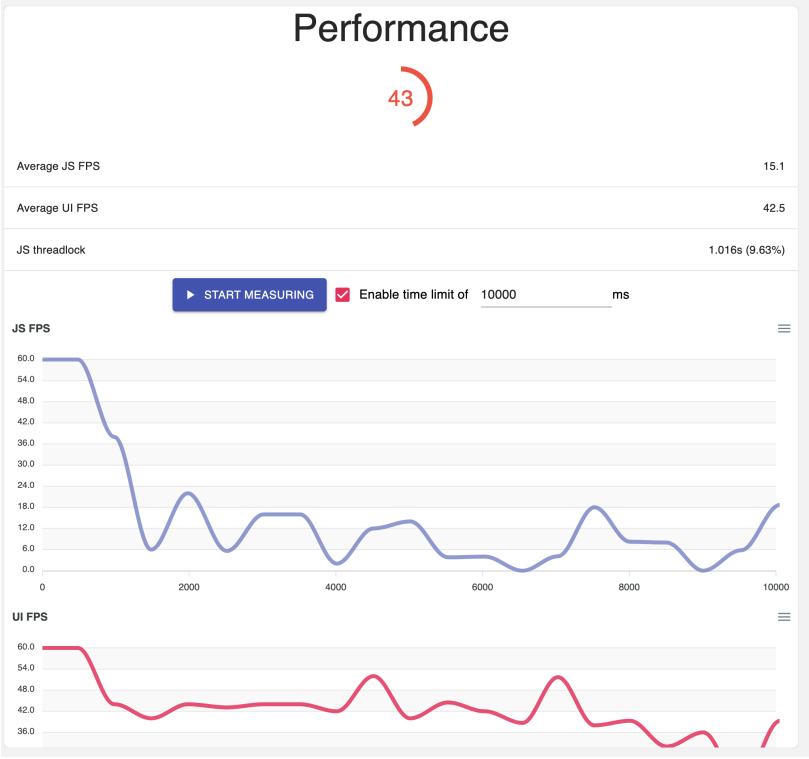

Figura A.5: FlatList Teste 5

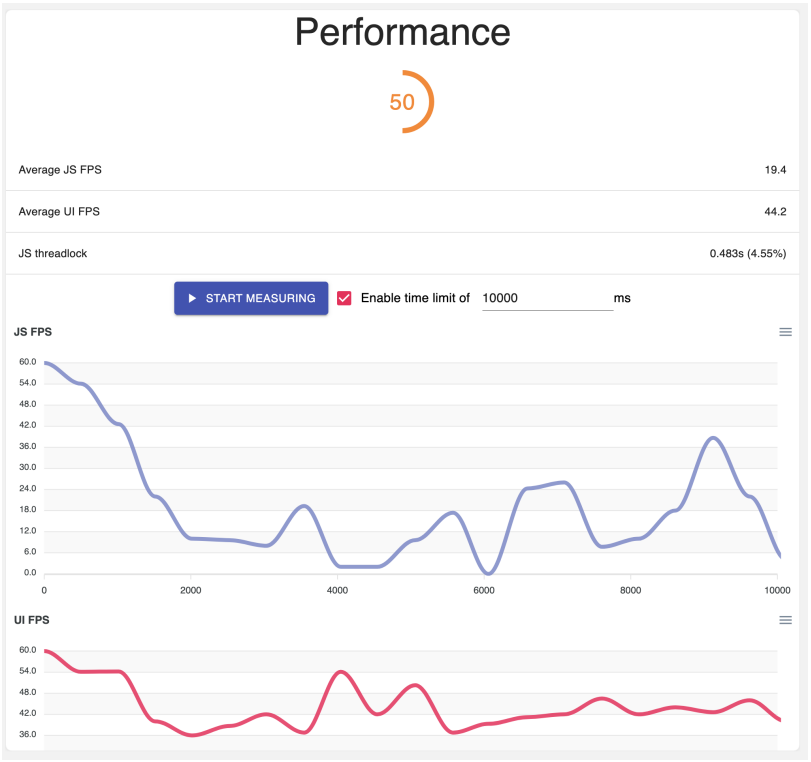

Figura A.6: FlatList Teste 6

Figura A.7: FlatList Teste 7

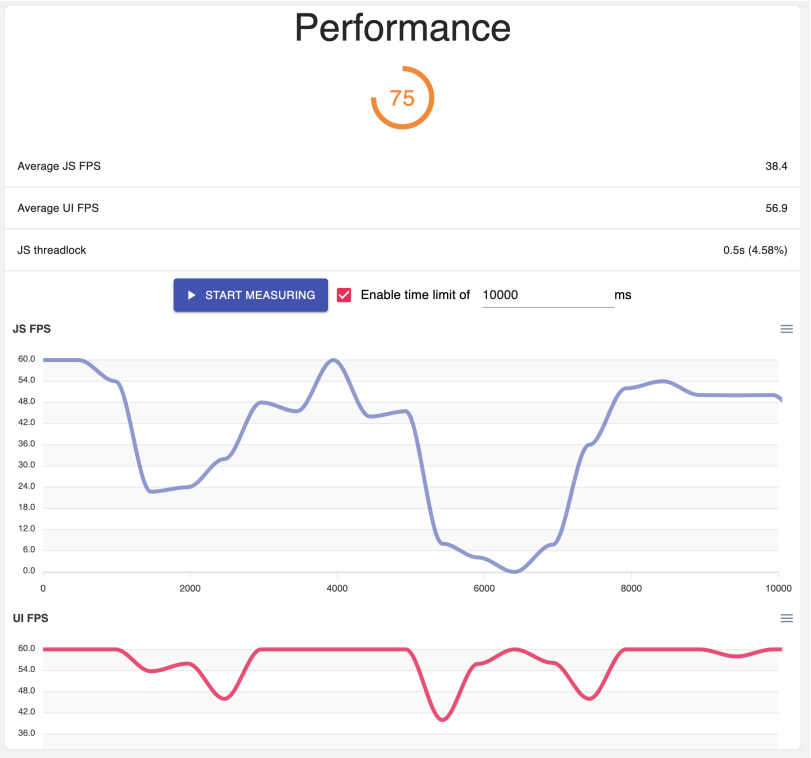

Figura A.8: FlatList Teste 8

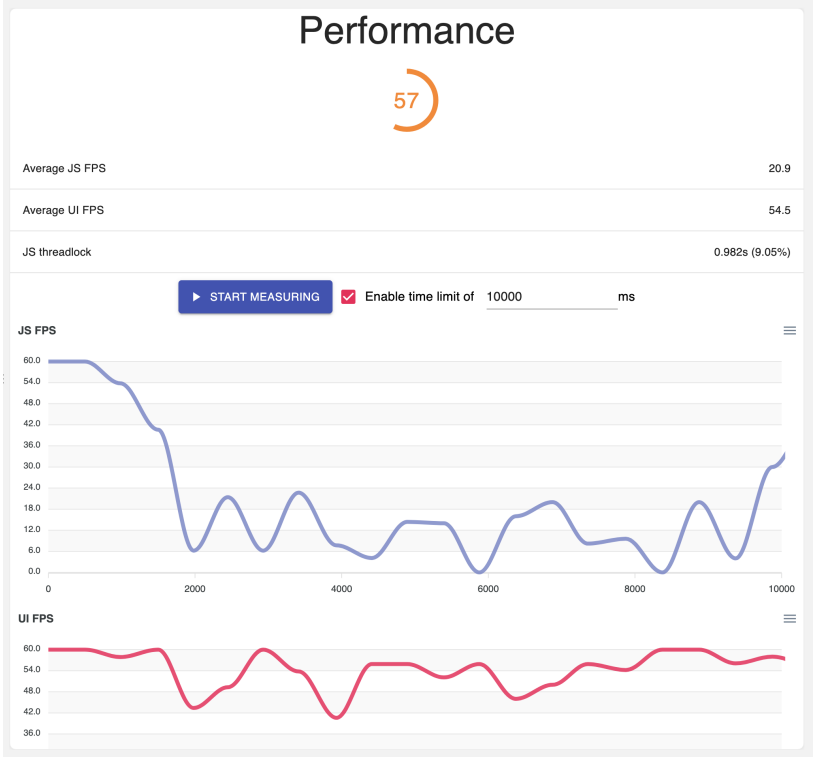

Figura A.9: FlatList Teste 9

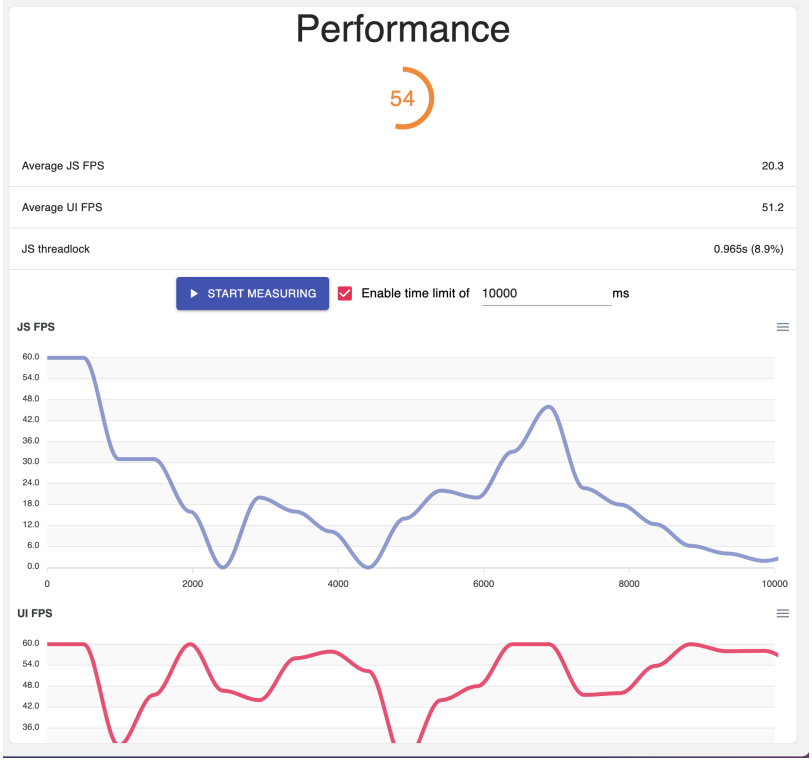

Figura A.10: FlatList Teste 10

Apêndice B

Anexos

B.1 FlashList: World of Lists

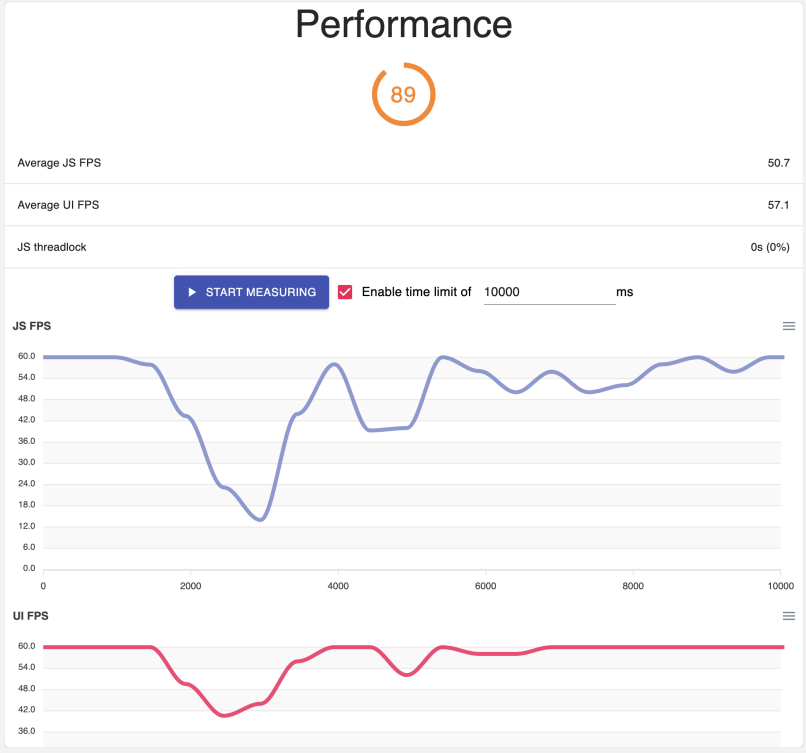

Figura B.1: FlashList Teste 1

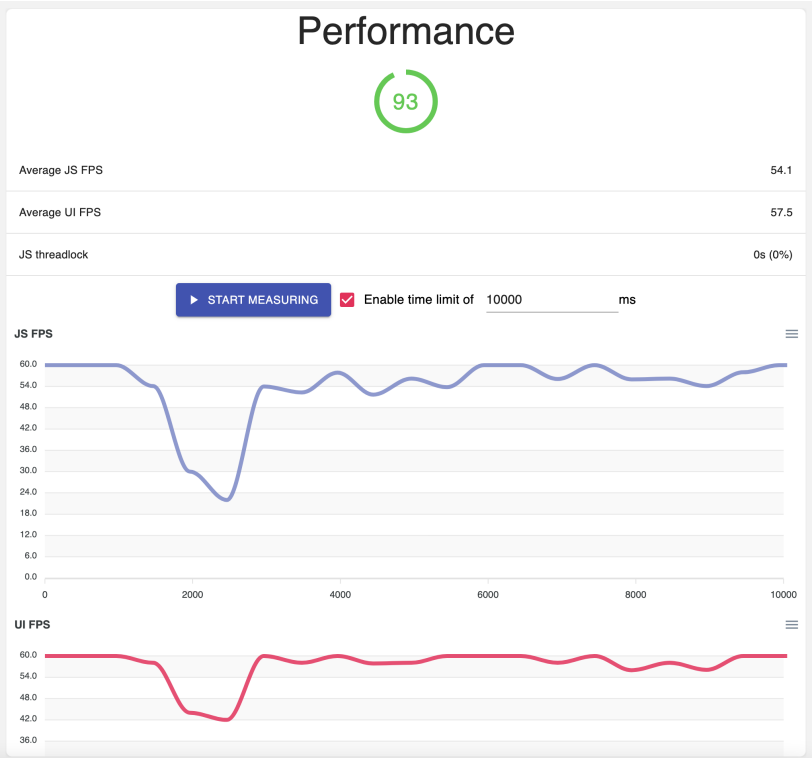

Figura B.2: FlashList Teste 2

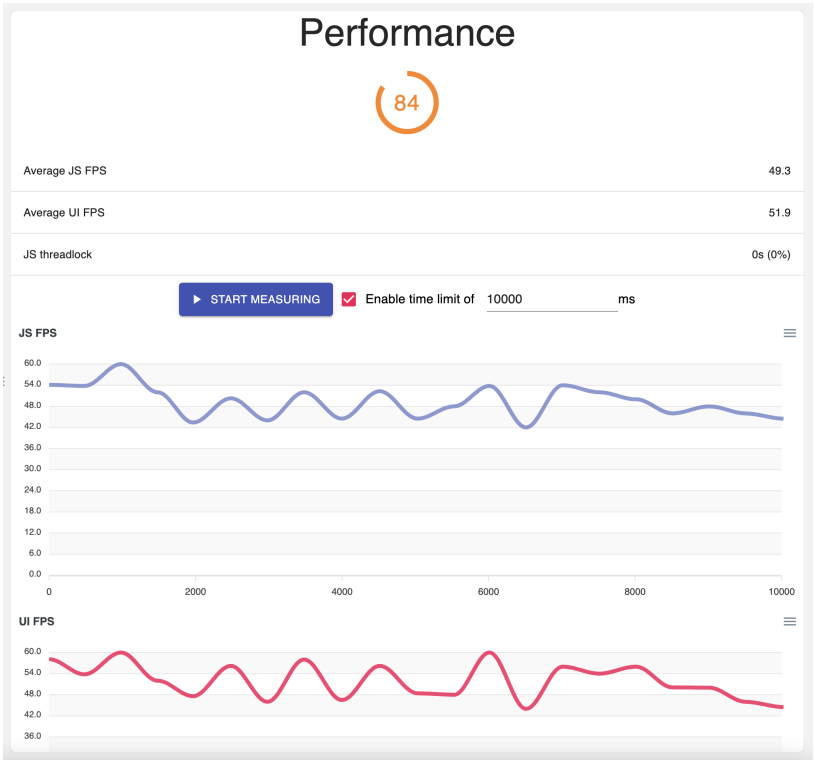

Figura B.3: FlashList Teste 3

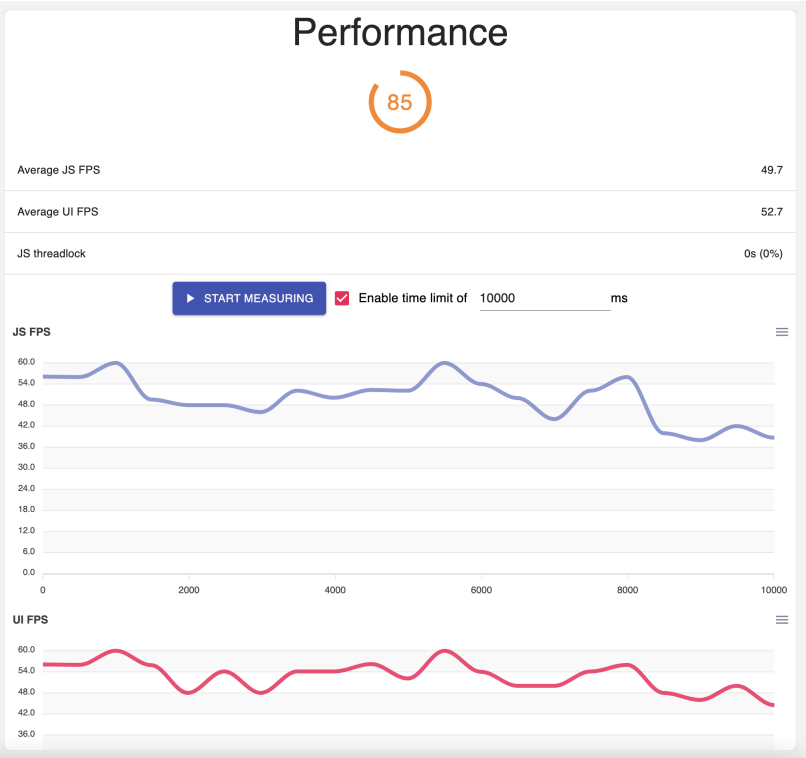

Figura B.4: FlashList Teste 4

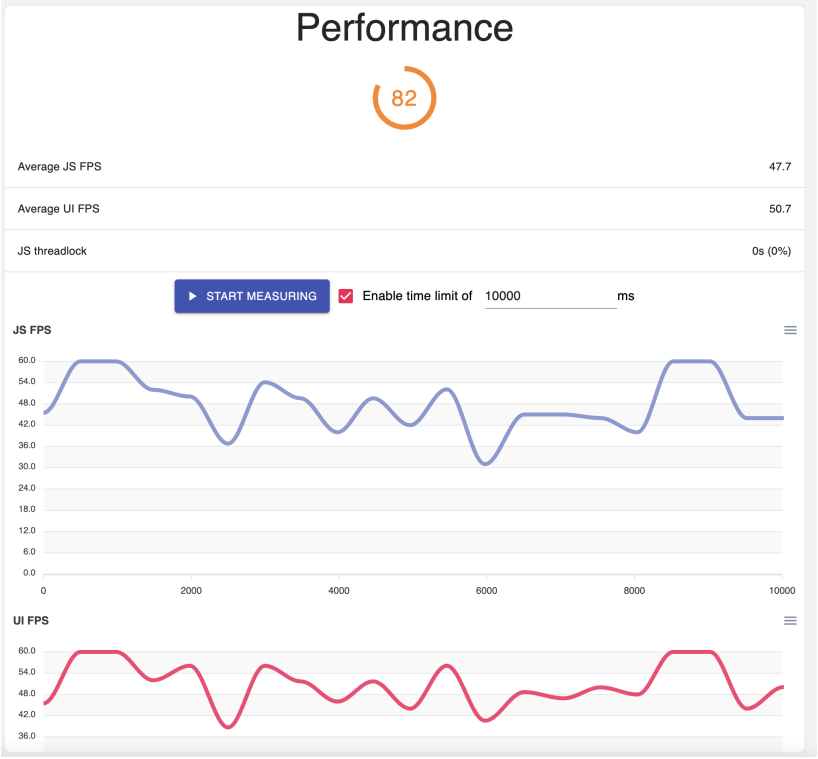

Figura B.5: FlashList Teste 5

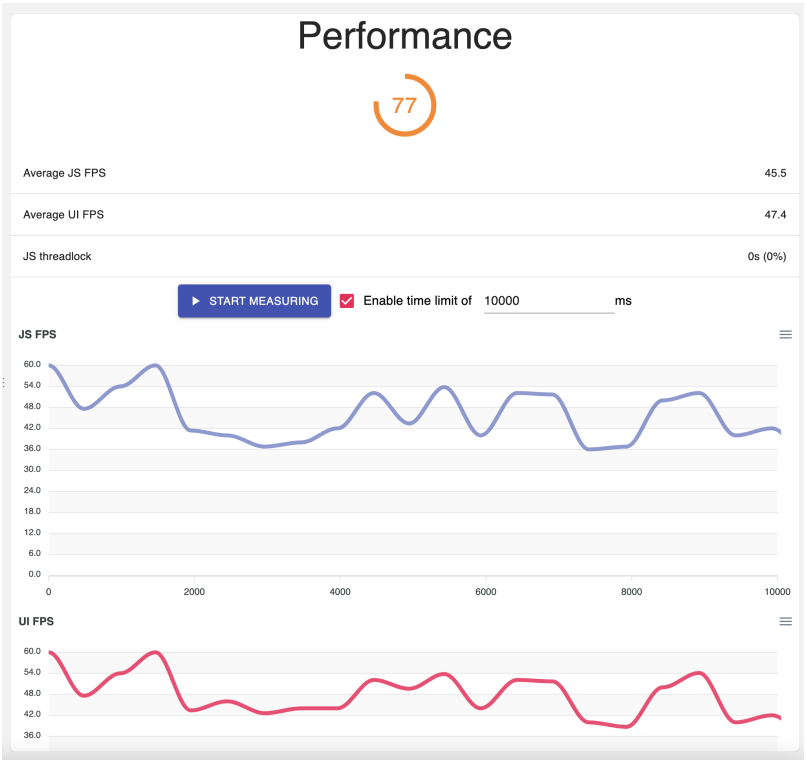

Figura B.6: FlashList Teste 6

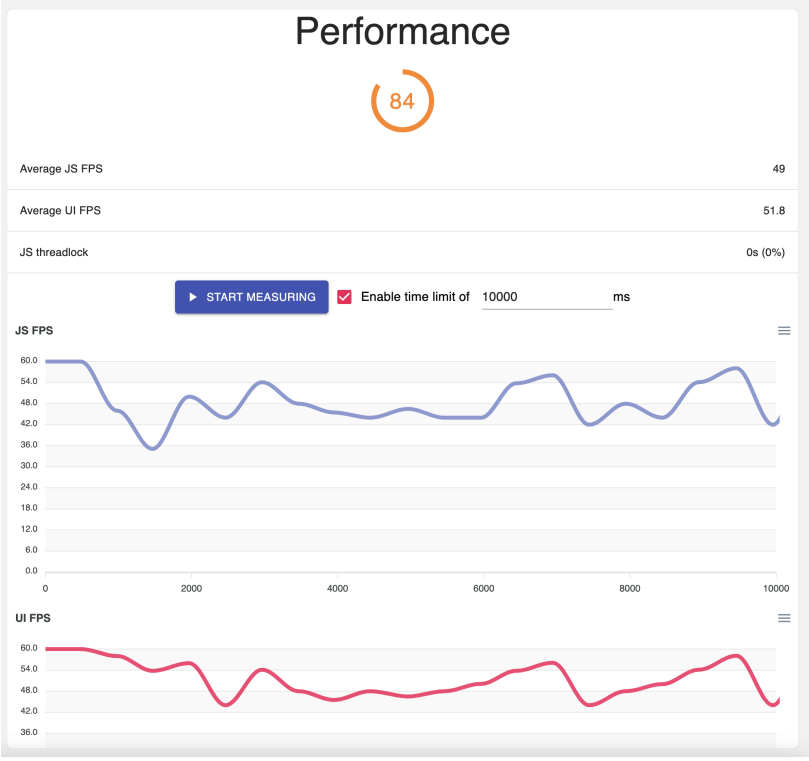

Figura B.7: FlashList Teste 7

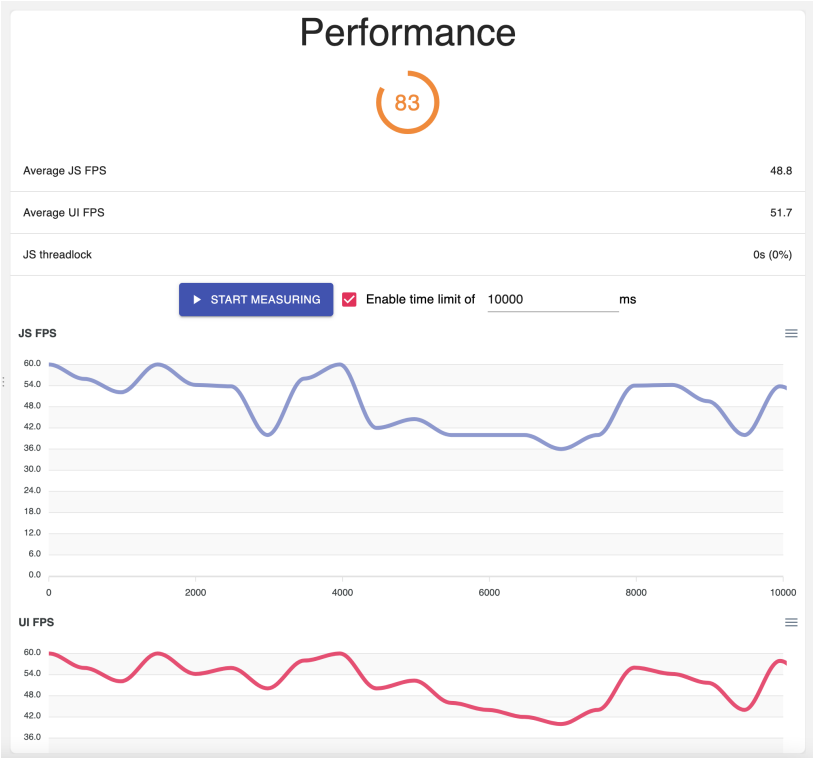

Figura B.8: FlashList Teste 8

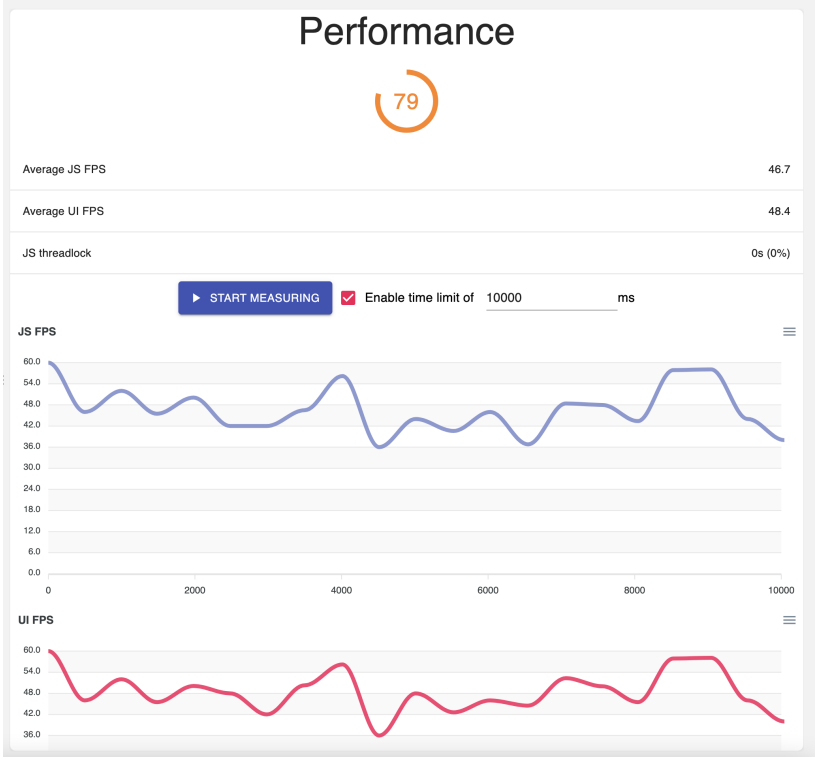

Figura B.9: FlashList Teste 9

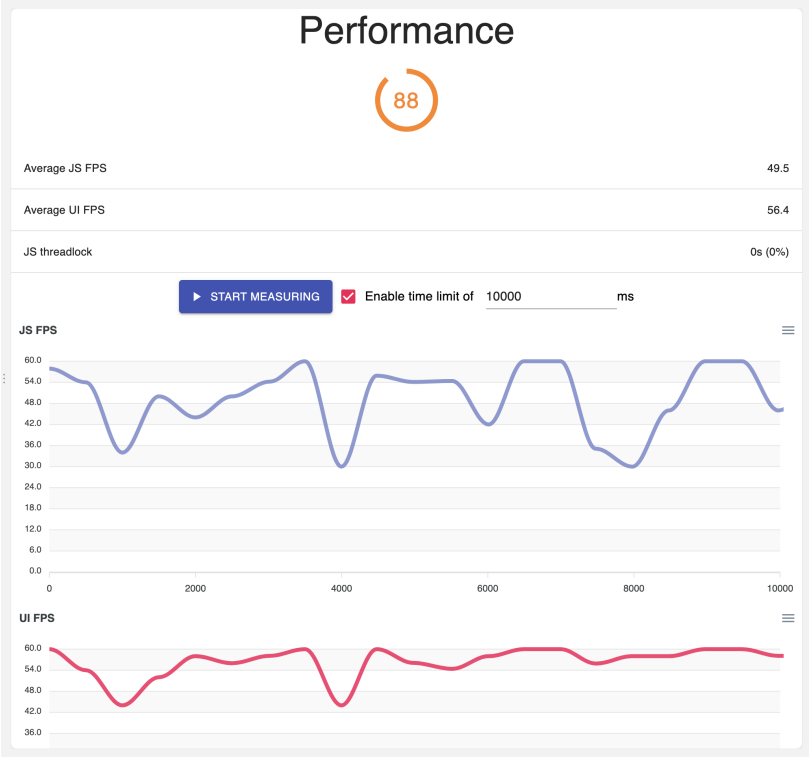

Figura B.10: FlashList Teste 10

Apêndice C

Anexos

C.1 BigList: World of Lists

Figura C.1: BigList Teste 1

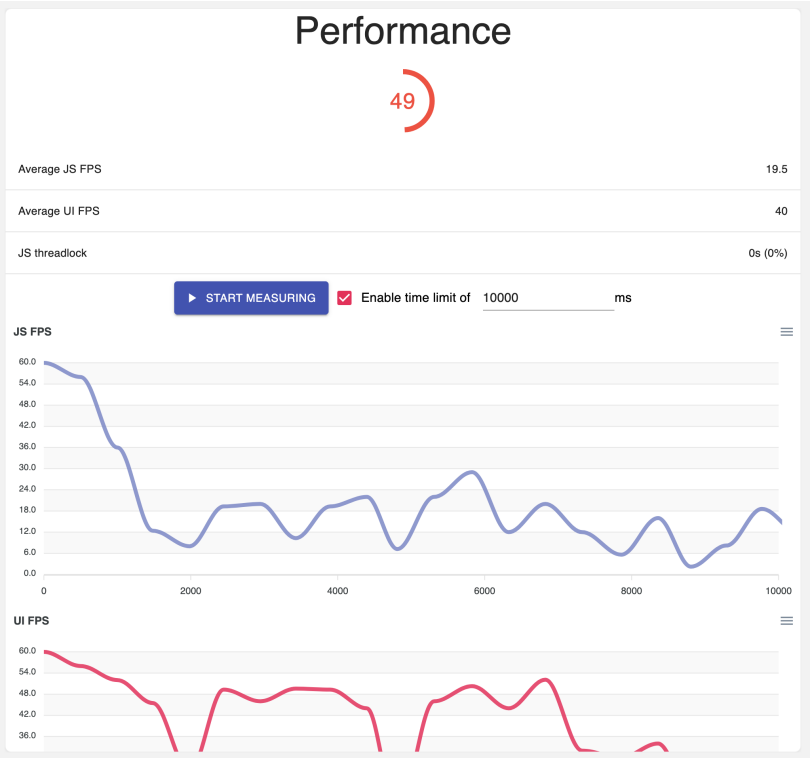

Figura C.2: BigList Teste 2

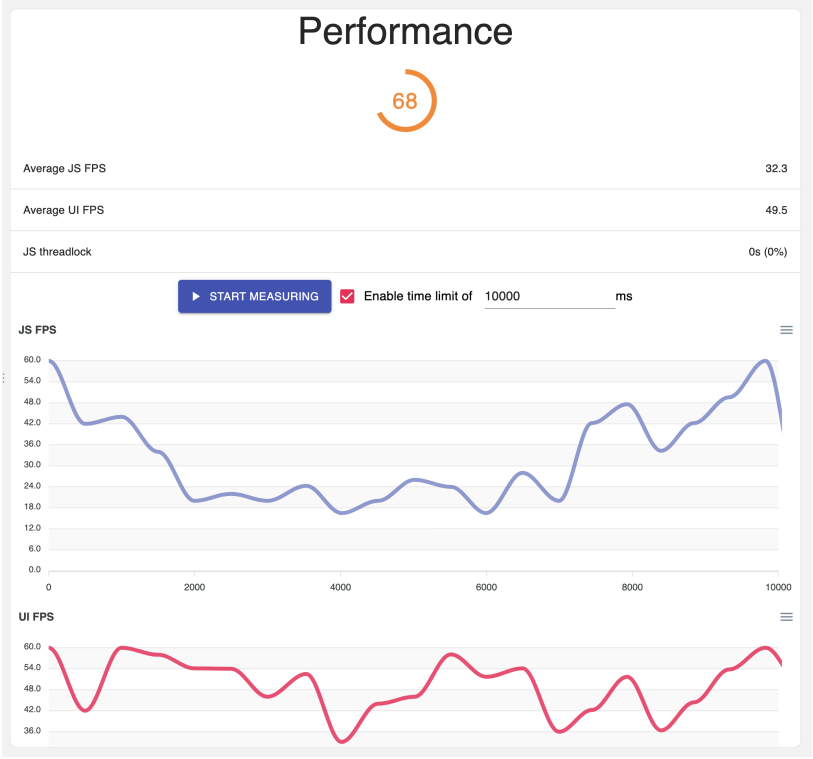

Figura C.3: BigList Teste 3

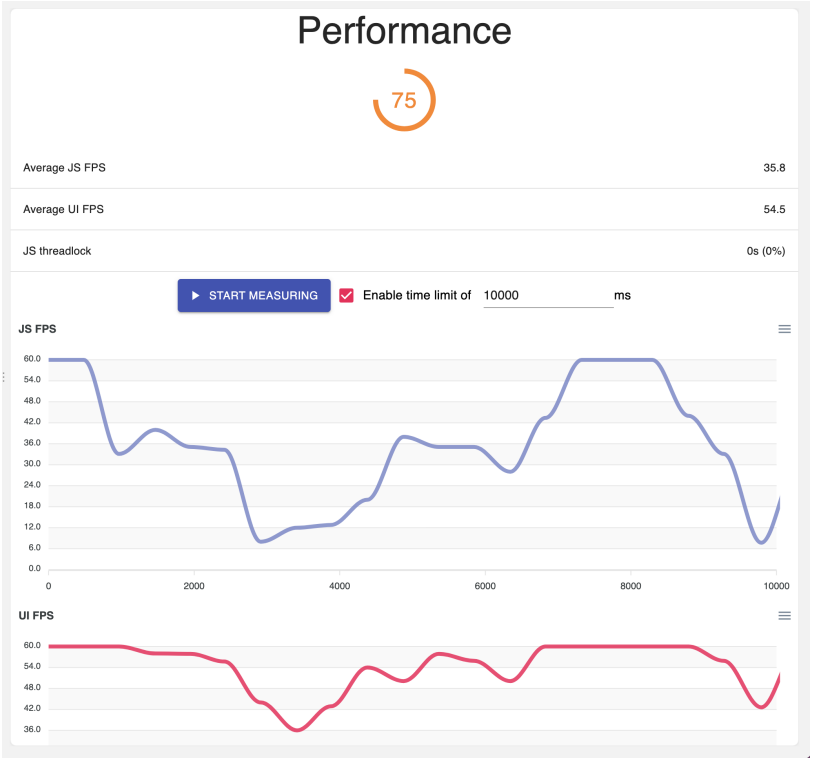

Figura C.4: BigList Teste 4

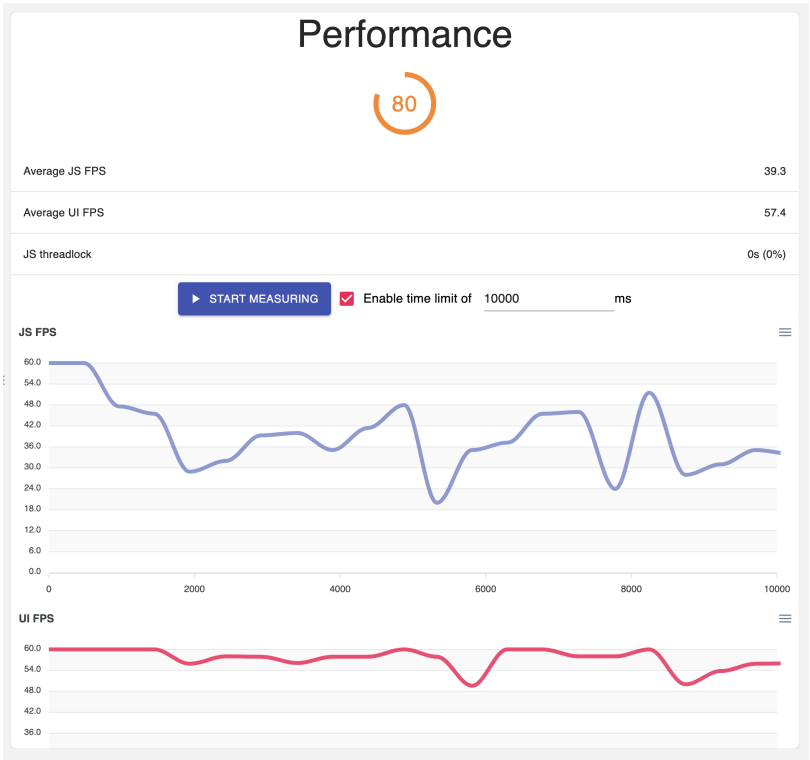

Figura C.5: BigList Teste 5

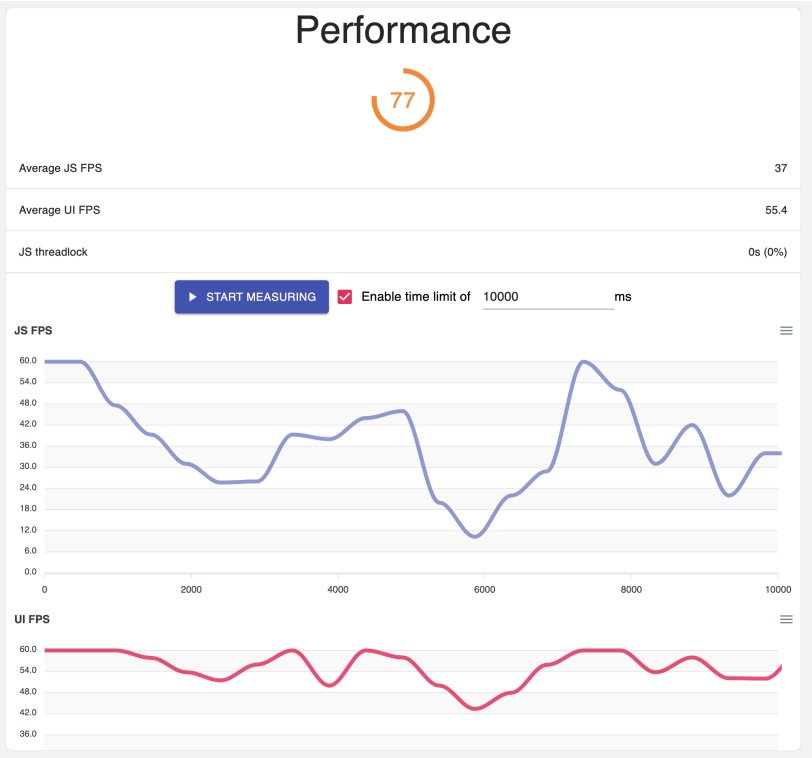

Figura C.6: BigList Teste 6

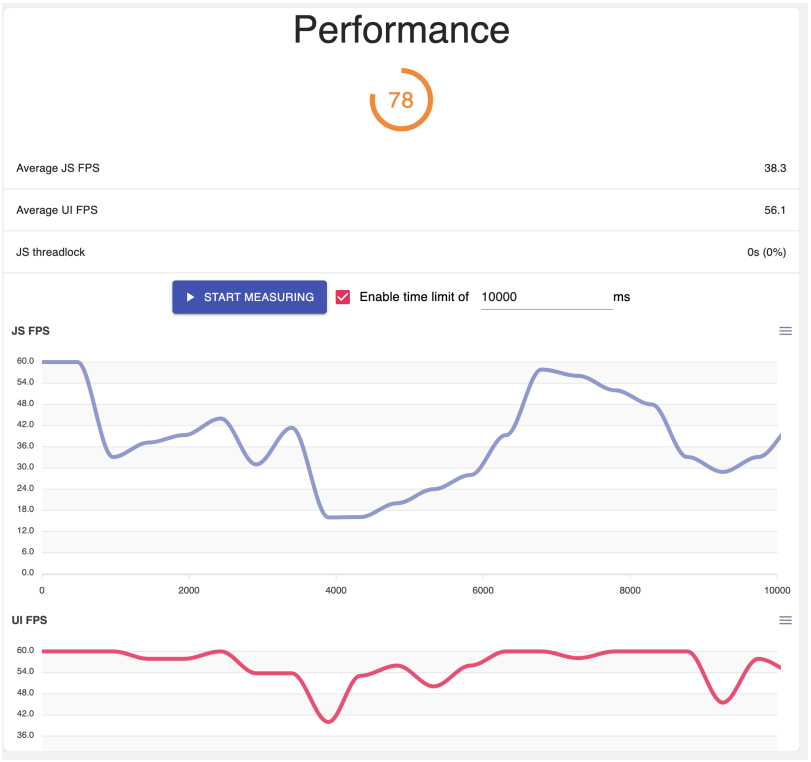

Figura C.7: BigList Teste 7

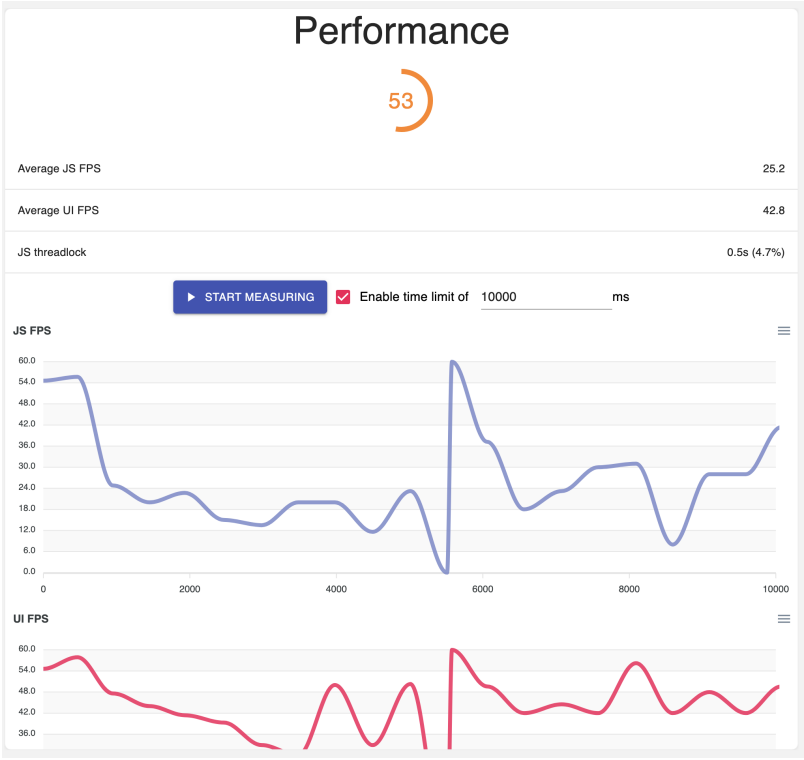

Figura C.8: BigList Teste 8

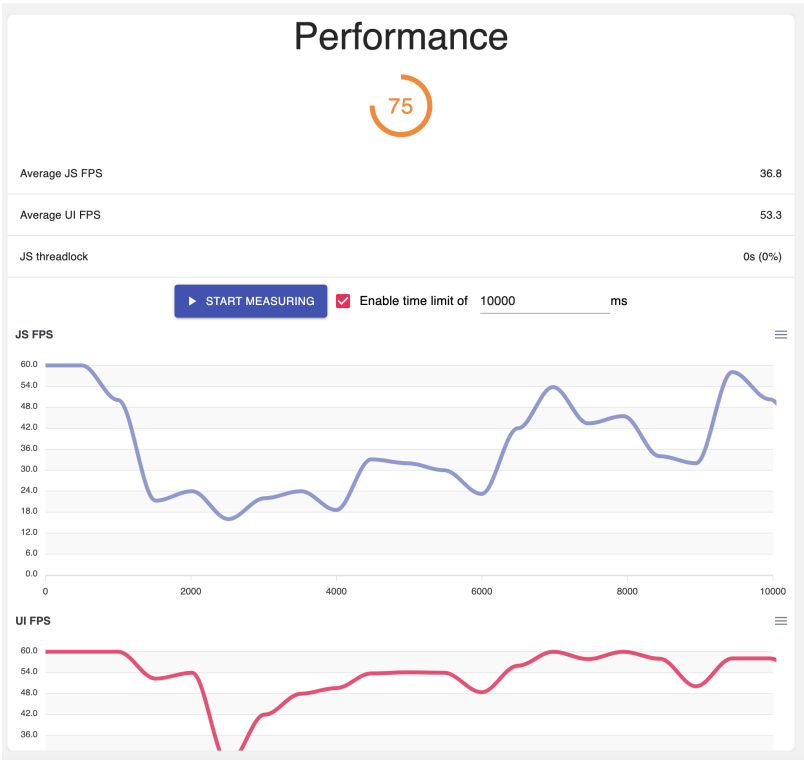

Figura C.9: BigList Teste 9

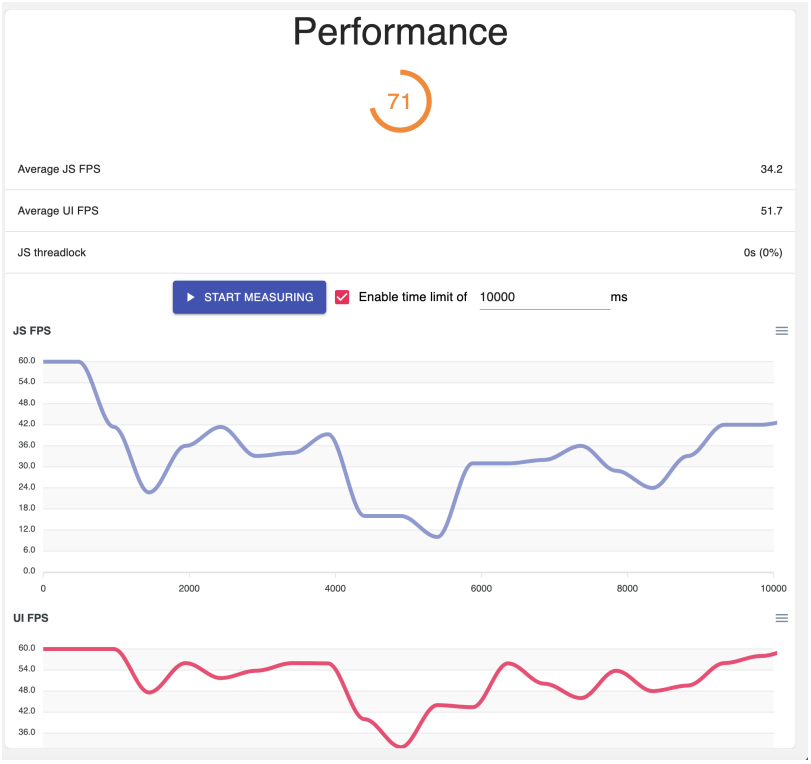

Figura C.10: BigList Teste 10

Apêndice D

Anexos

D.1 RecyclerListView: World of Lists

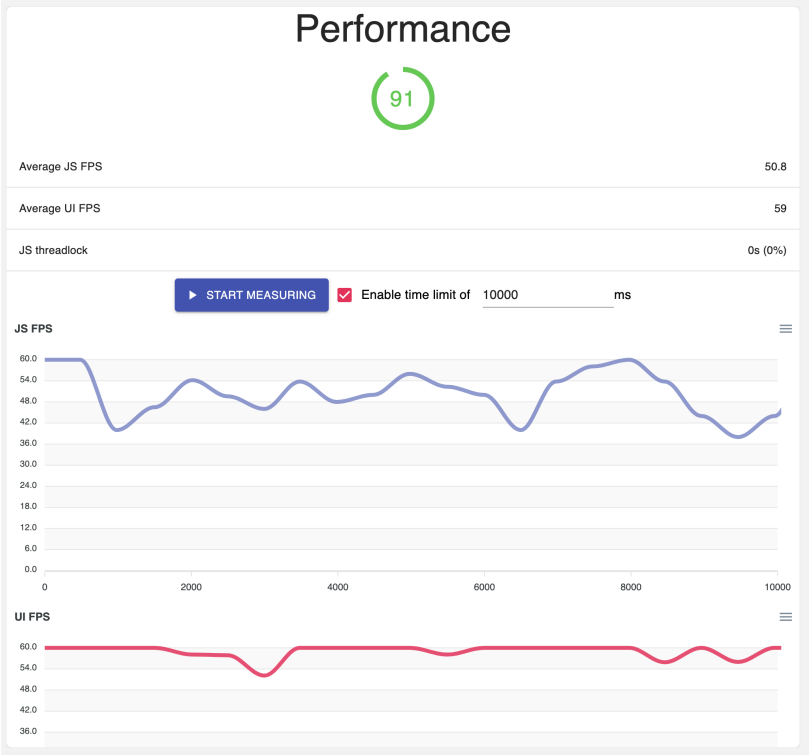

Figura D.1: RecyclerListView Teste 1

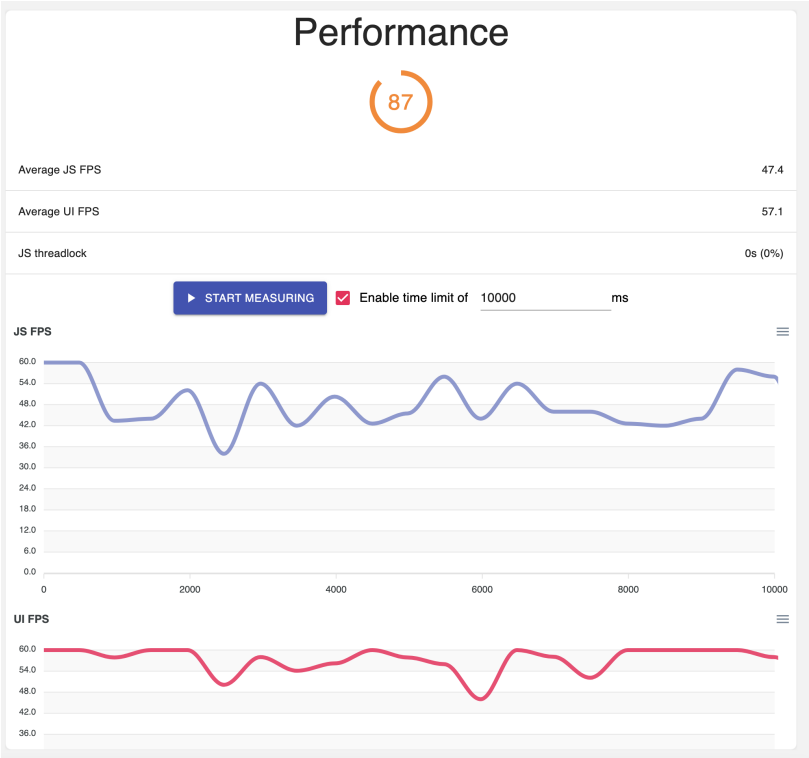

Figura D.2: RecyclerListView Teste 2

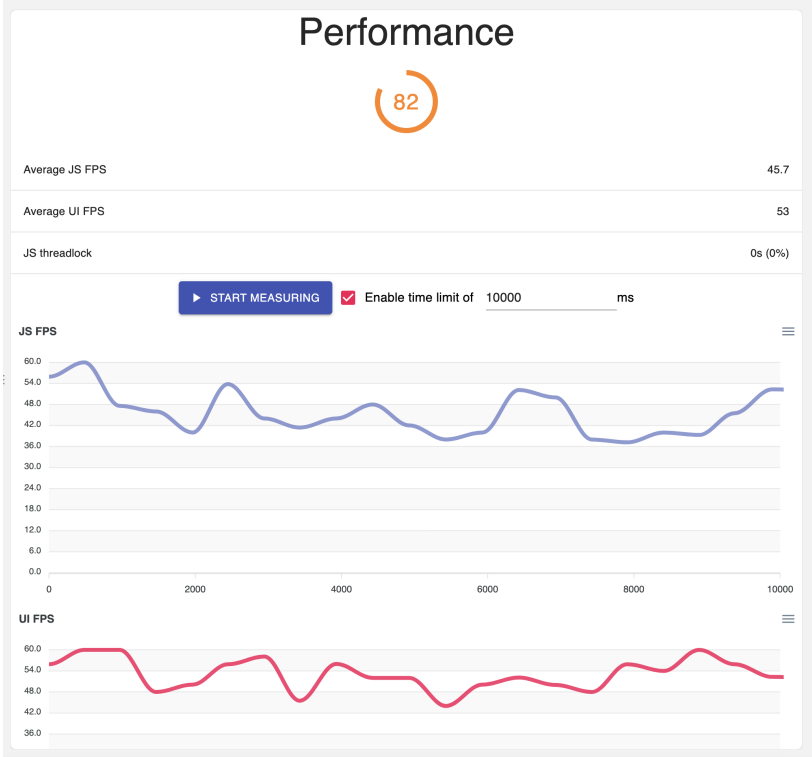

Figura D.3: RecyclerListView Teste 3

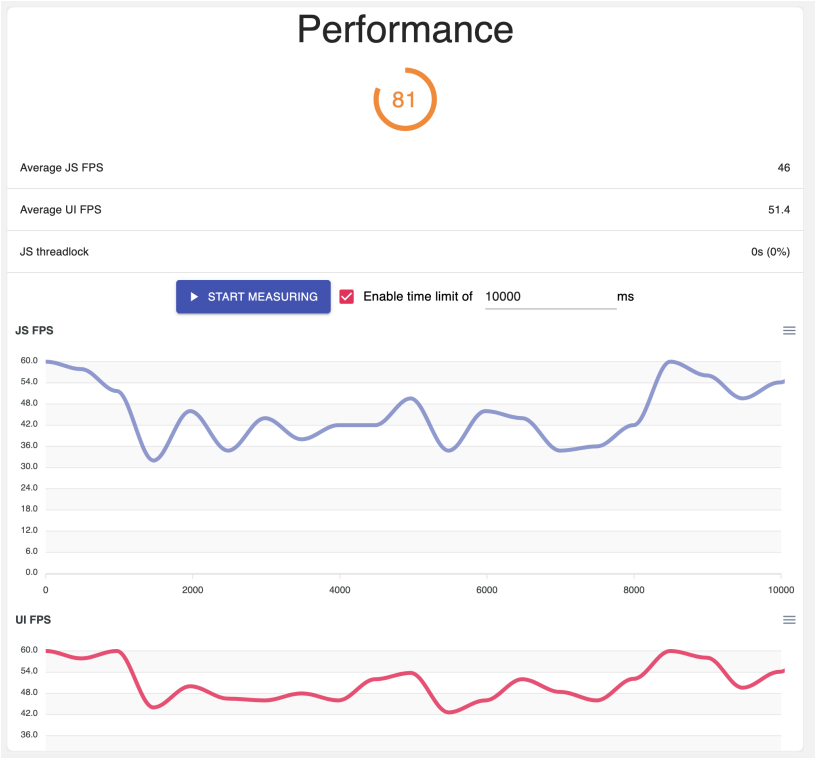

Figura D.4: RecyclerListView Teste 4

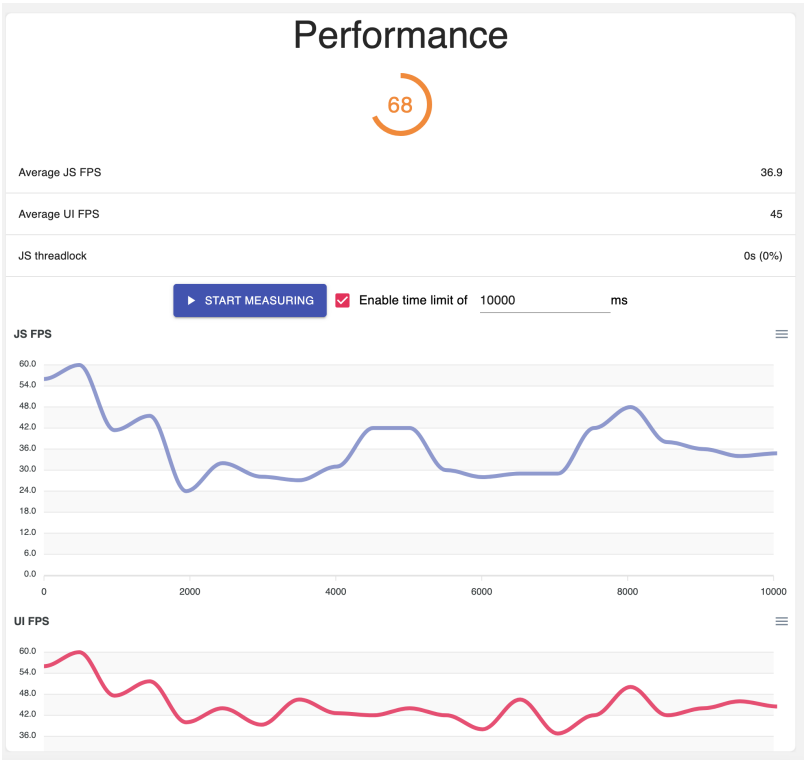

Figura D.5: RecyclerListView Teste 5

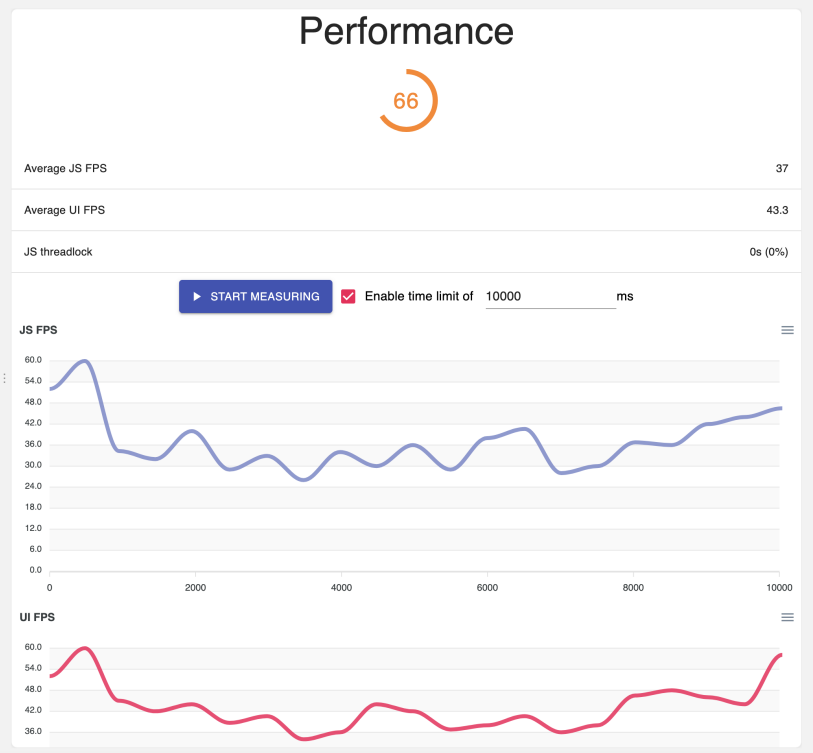

Figura D.6: RecyclerListView Teste 6

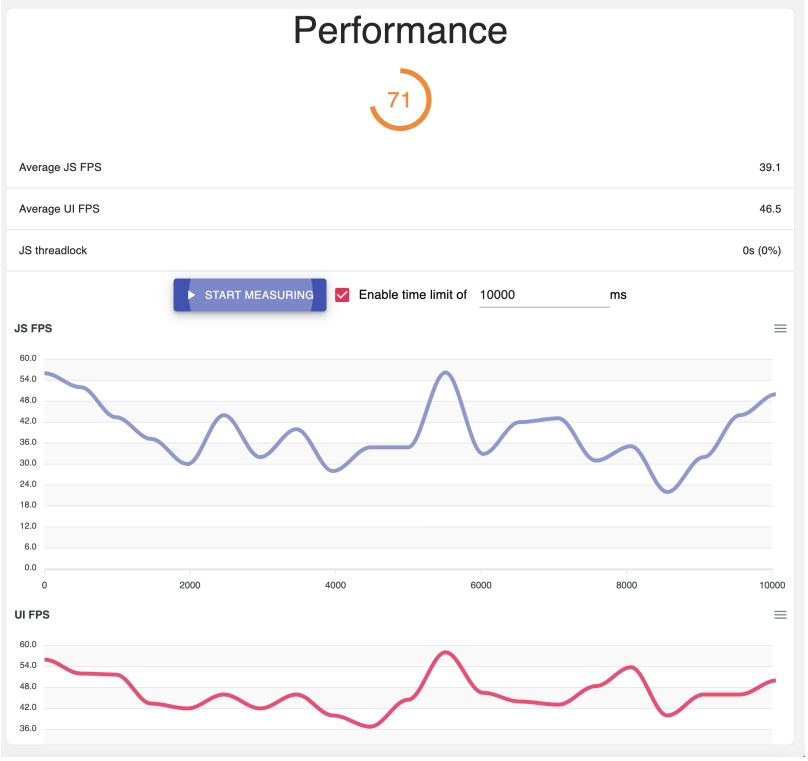

Figura D.7: RecyclerListView Teste 7

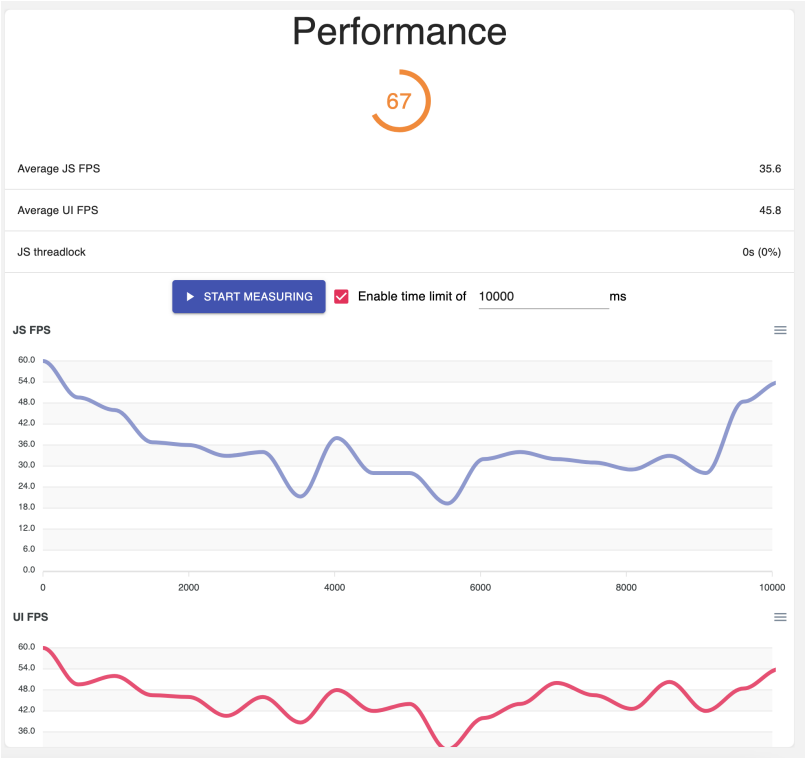

Figura D.8: RecyclerListView Teste 8

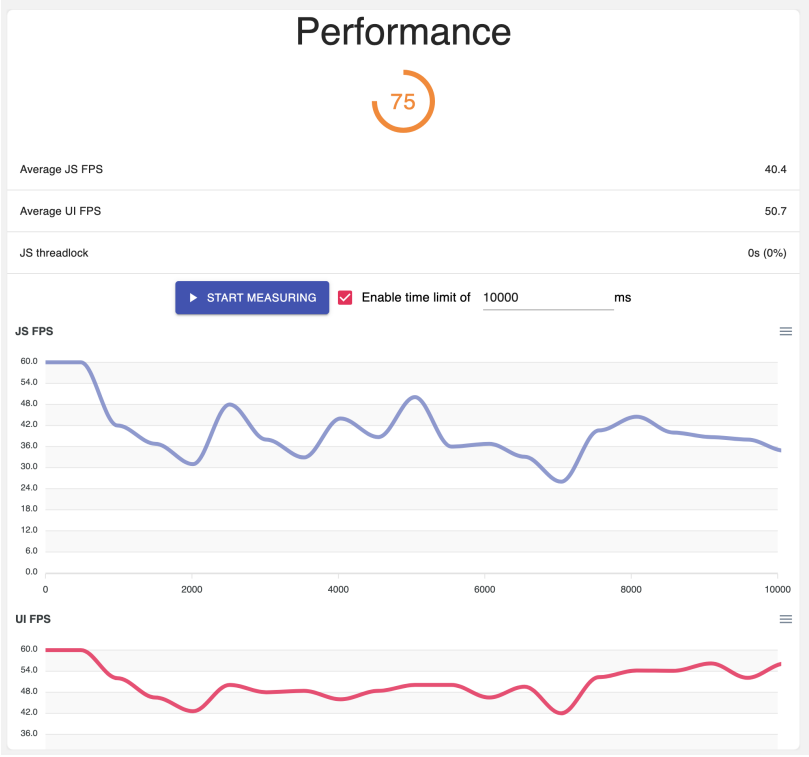

Figura D.9: RecyclerListView Teste 9

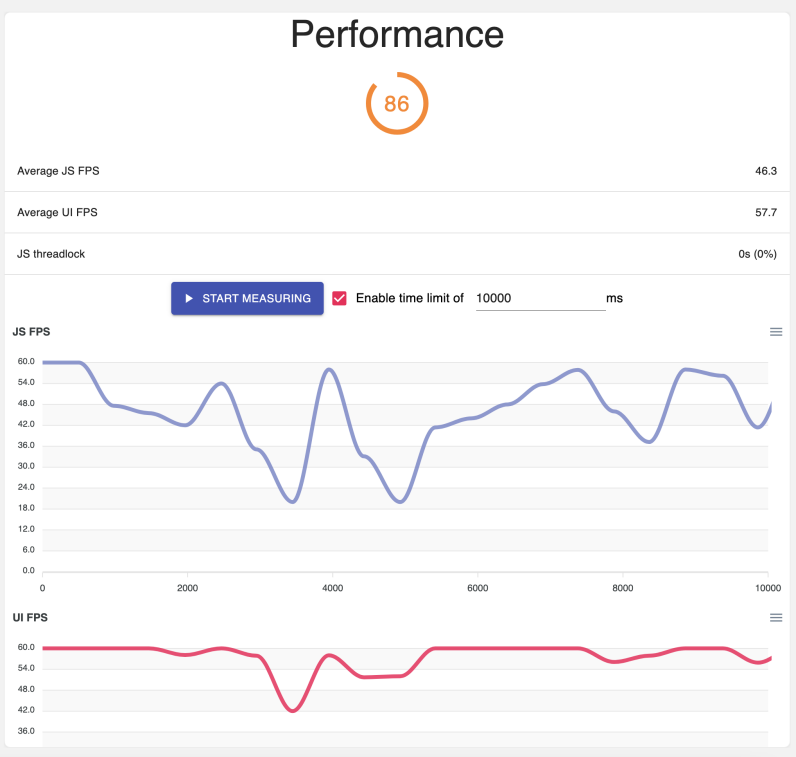

Figura D.10: RecyclerListView Teste 10

Apêndice E

Anexos

E.1 FlatList: Betfair Rebuild

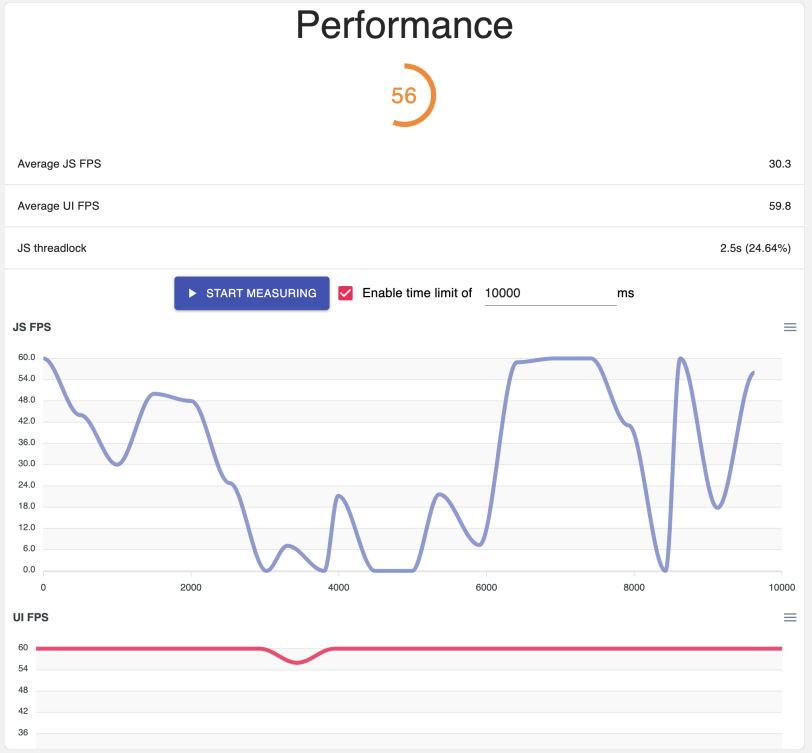

Figura E.1: FlatList Teste 1

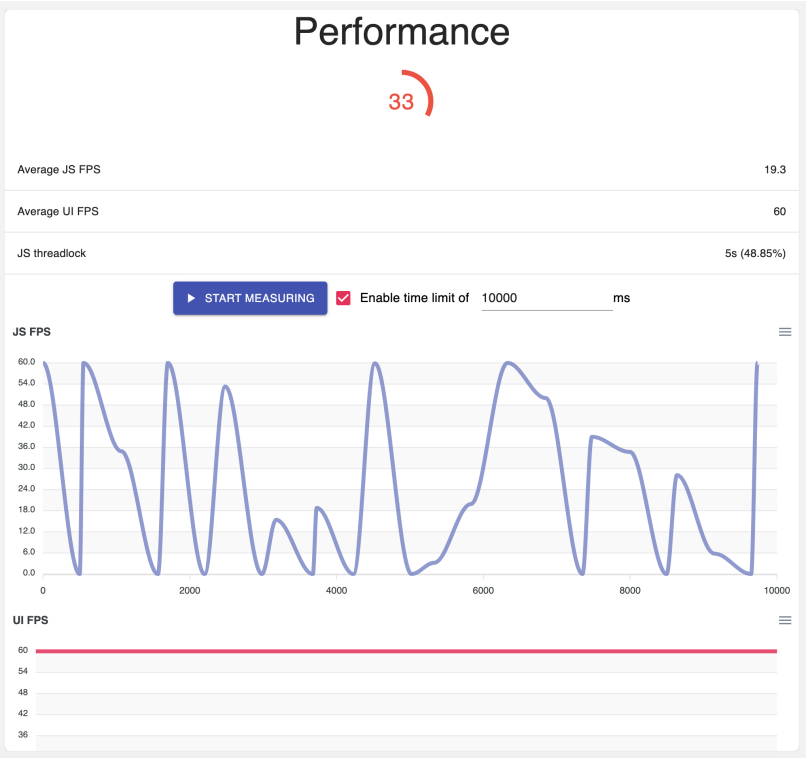

Figura E.2: FlatList Teste 2

Figura E.3: FlatList Teste 3

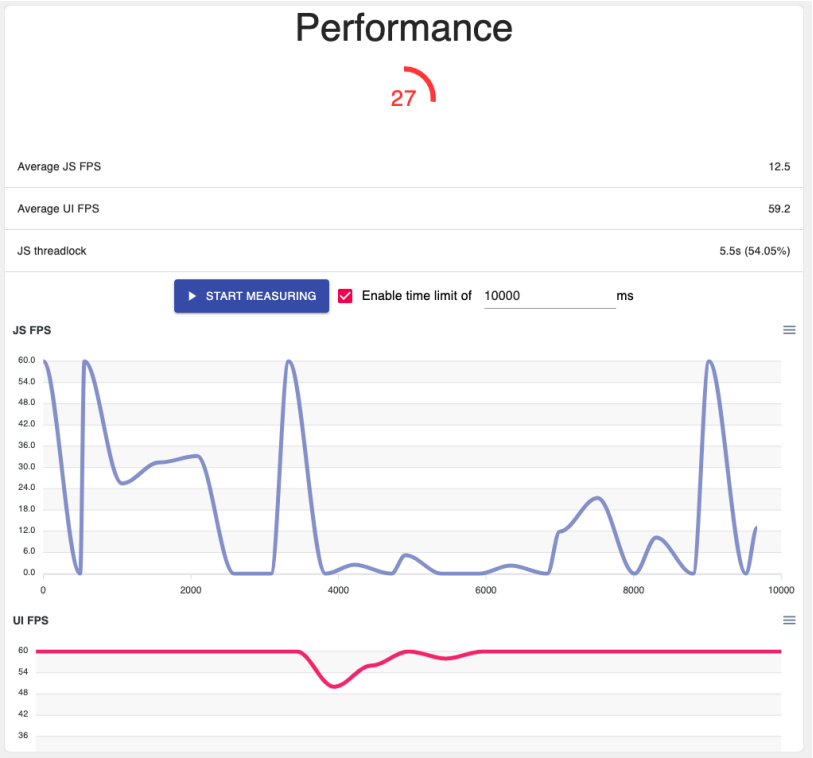

Figura E.4: FlatList Teste 4

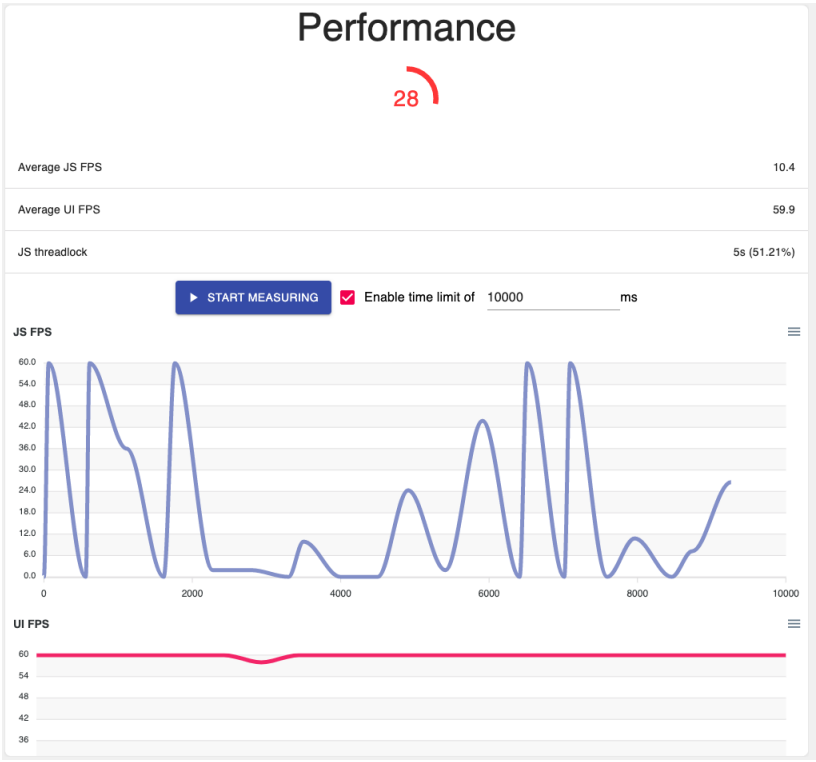

Figura E.5: FlatList Teste 5

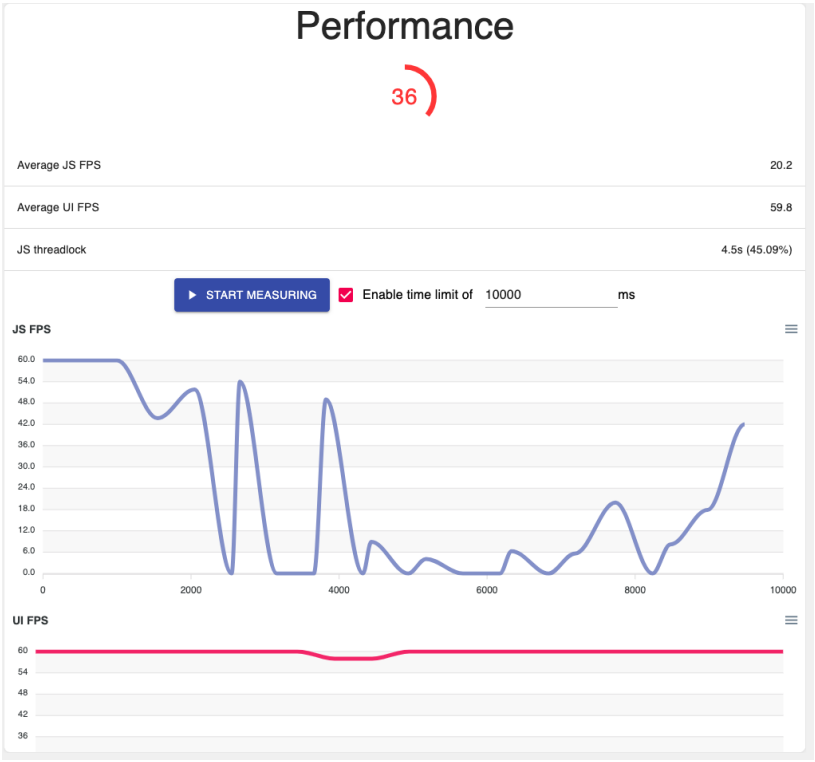

Figura E.6: FlatList Teste 6

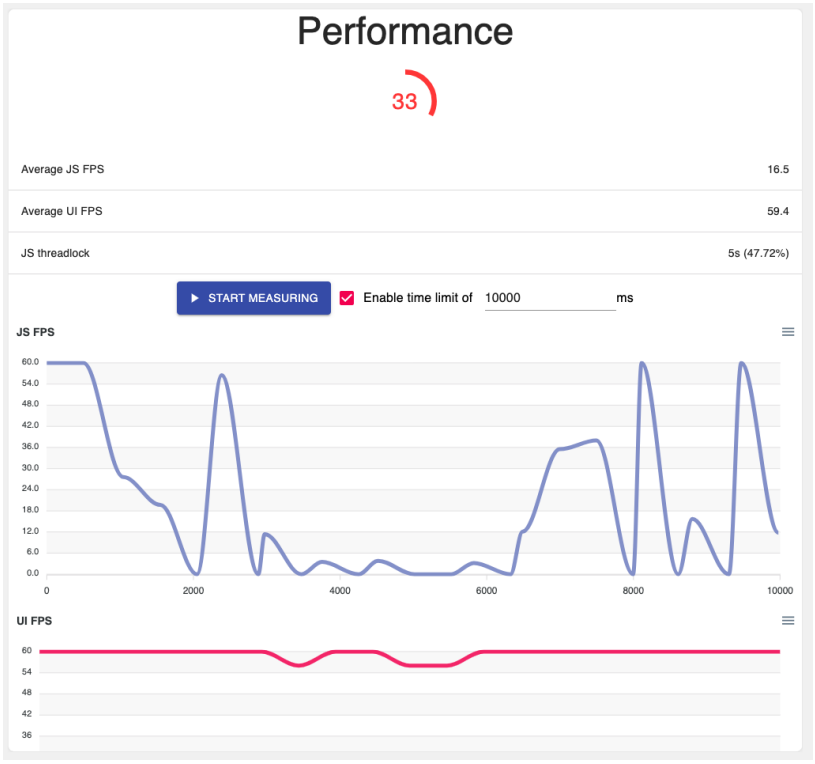

Figura E.7: FlatList Teste 7

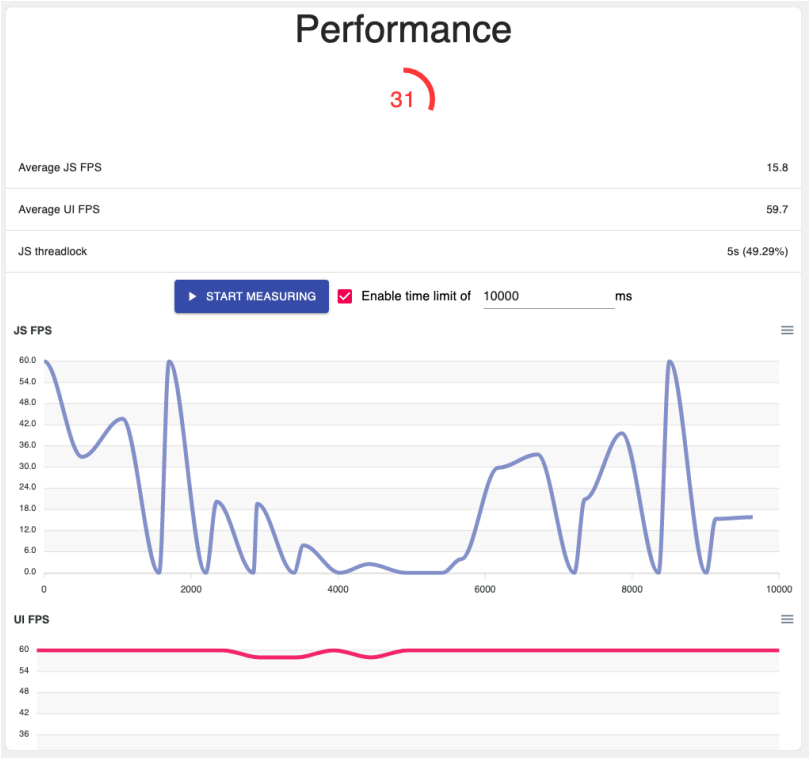

Figura E.8: FlatList Teste 8

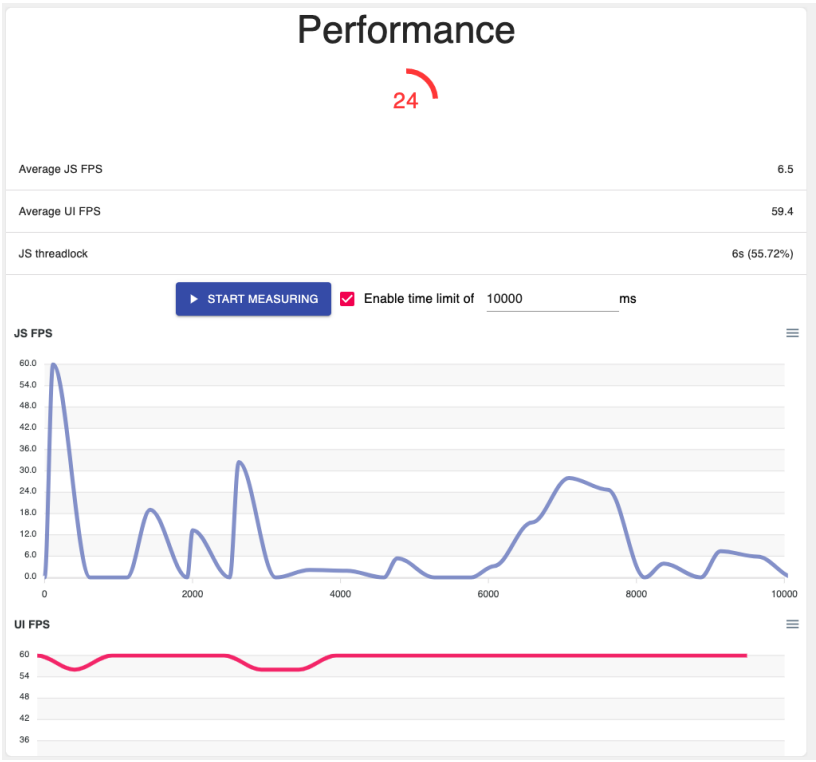

Figura E.9: FlatList Teste 9

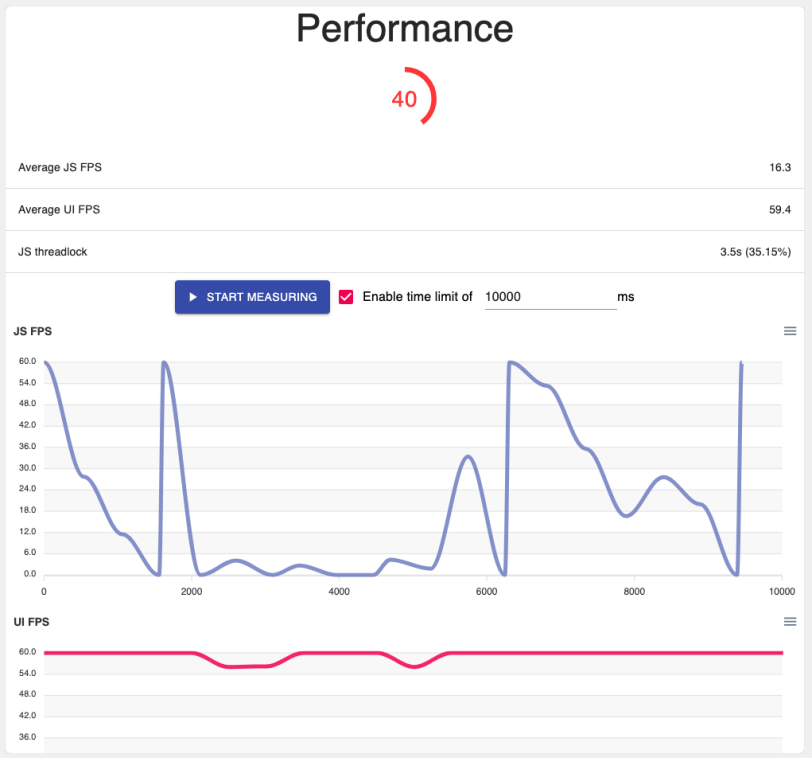

Figura E.10: FlatList Teste 10

Apêndice F

Anexos

F.1 FlashList: Betfair Rebuild

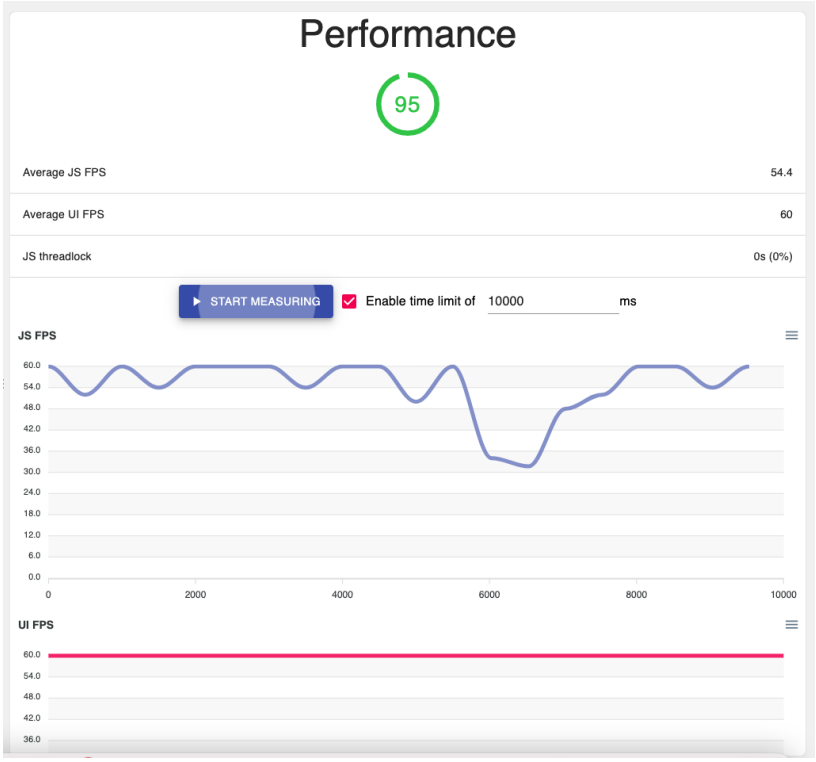

Figura F.1: FlashList Teste 1

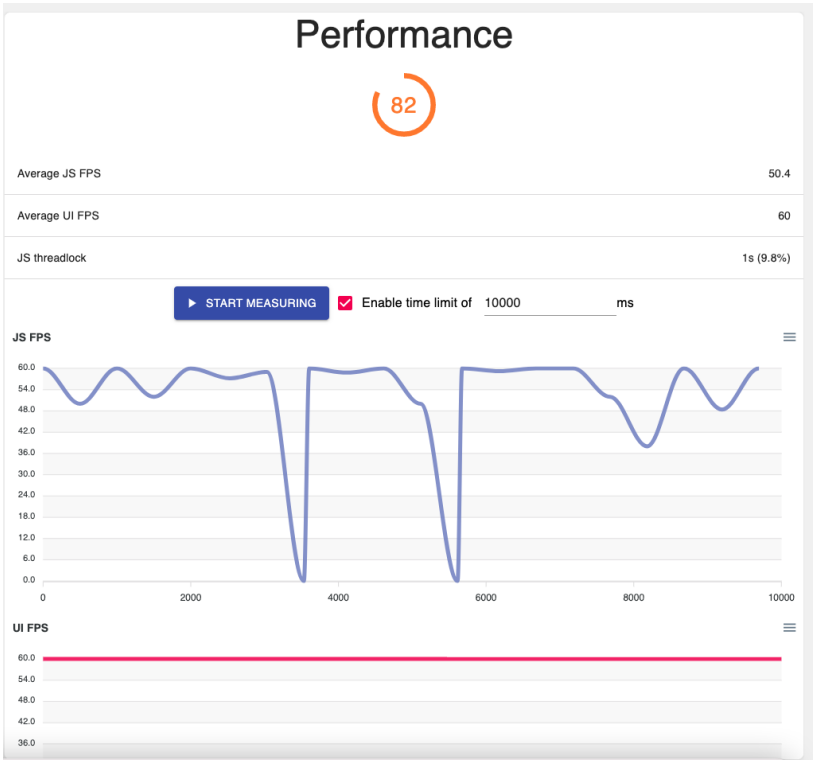

Figura F.2: FlashList Teste 2

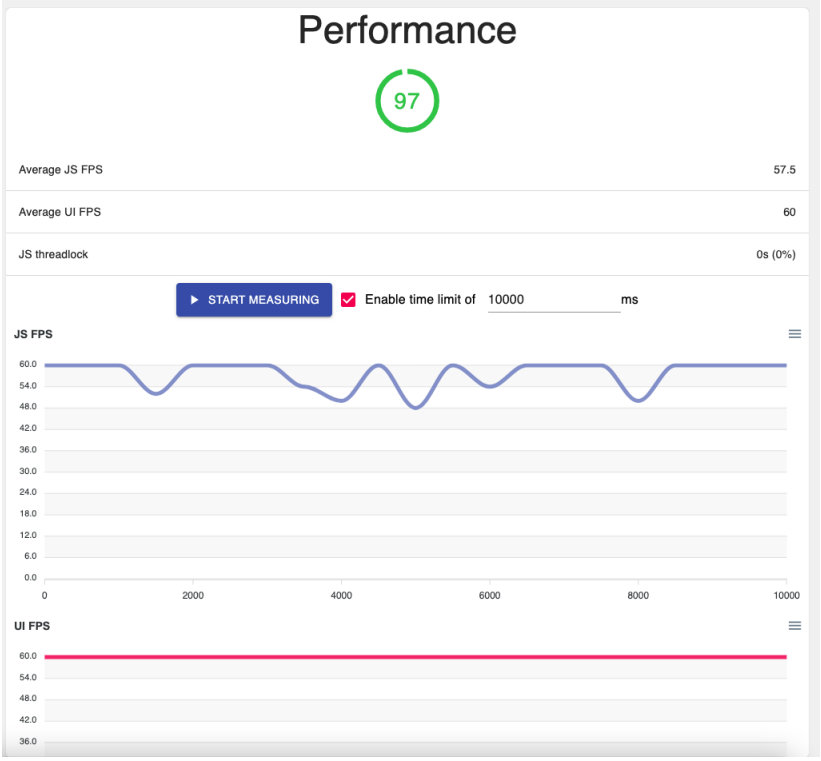

Figura F.3: FlashList Teste 3

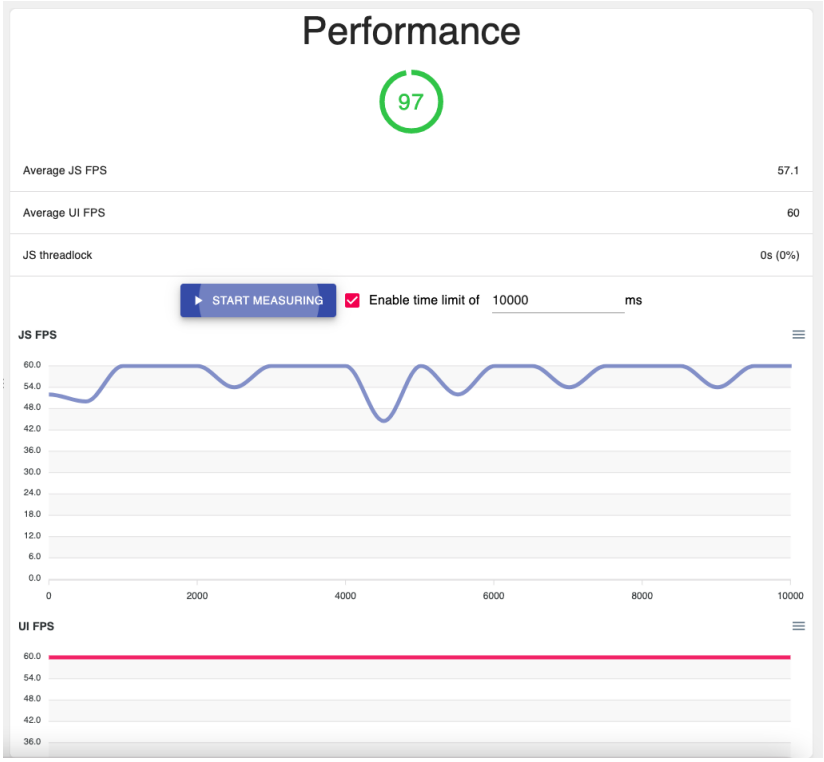

Figura F.4: FlashList Teste 4

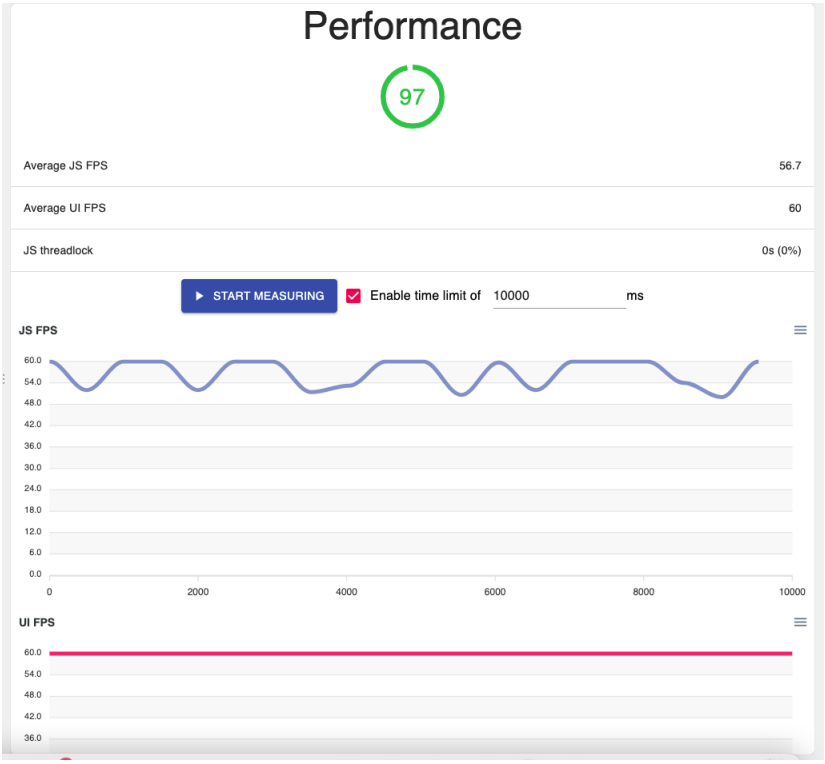

Figura F.5: FlashList Teste 5

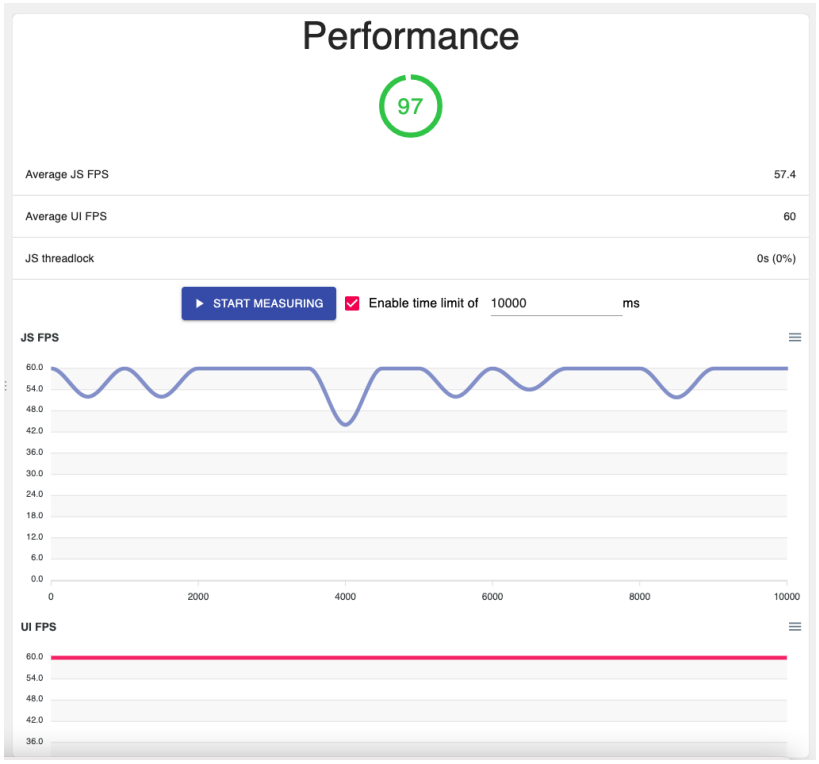

Figura F.6: FlashList Teste 6

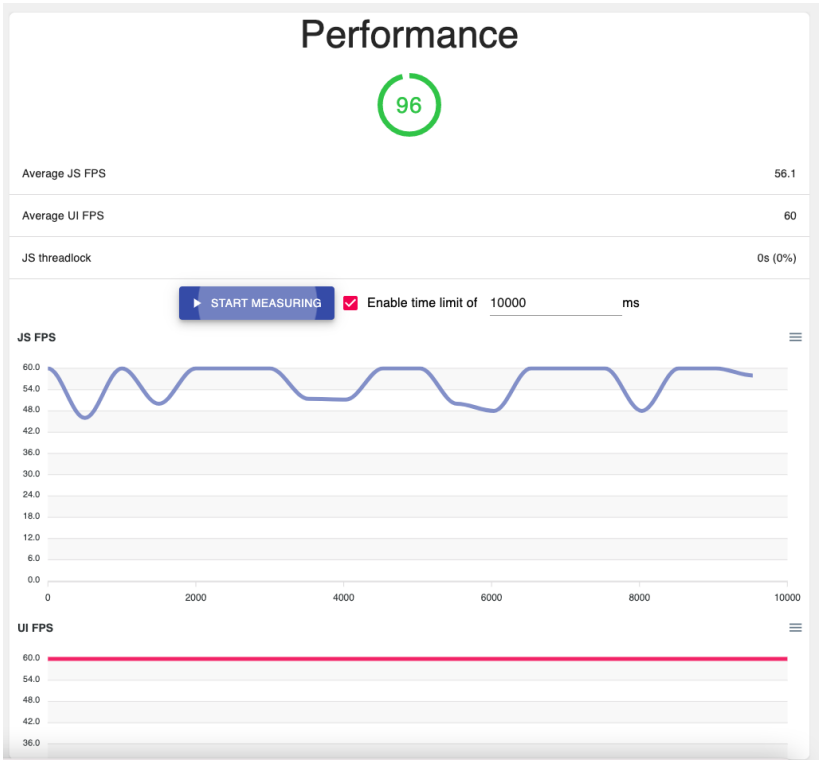

Figura F.7: FlashList Teste 7

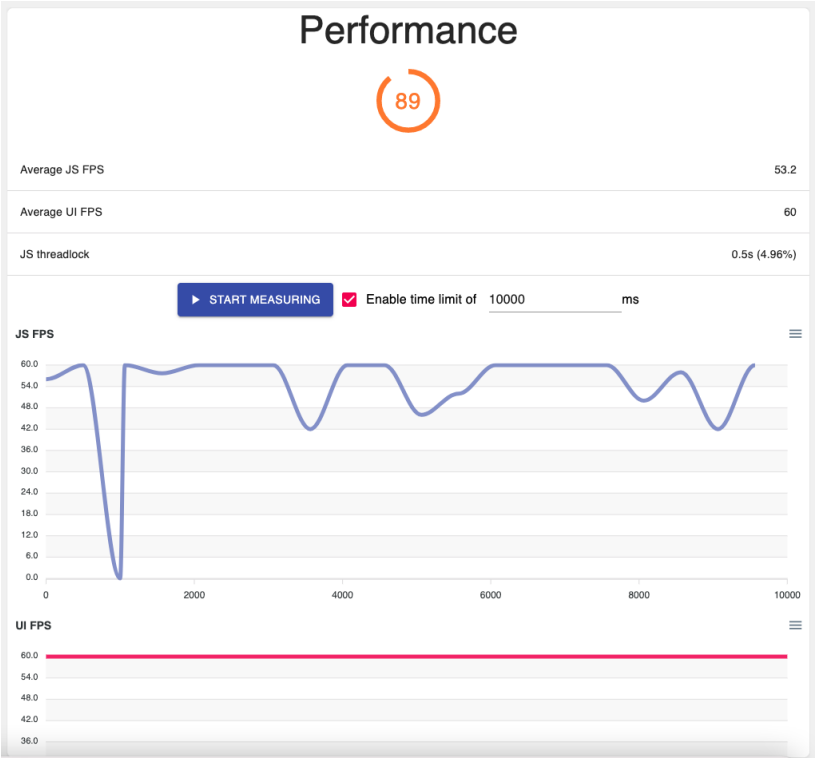

Figura F.8: FlashList Teste 8

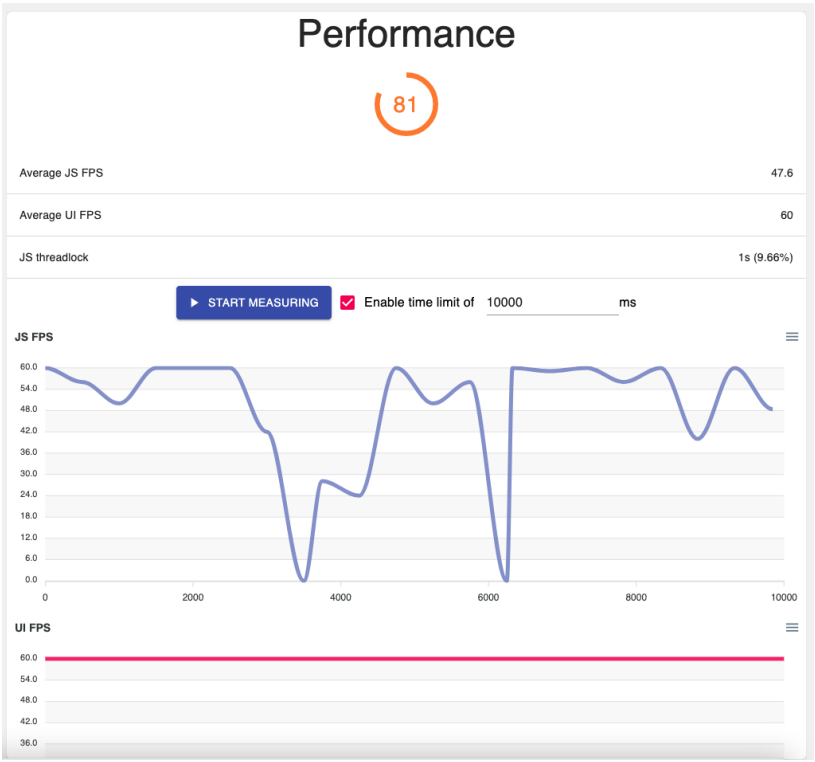

Figura F.9: FlashList Teste 9

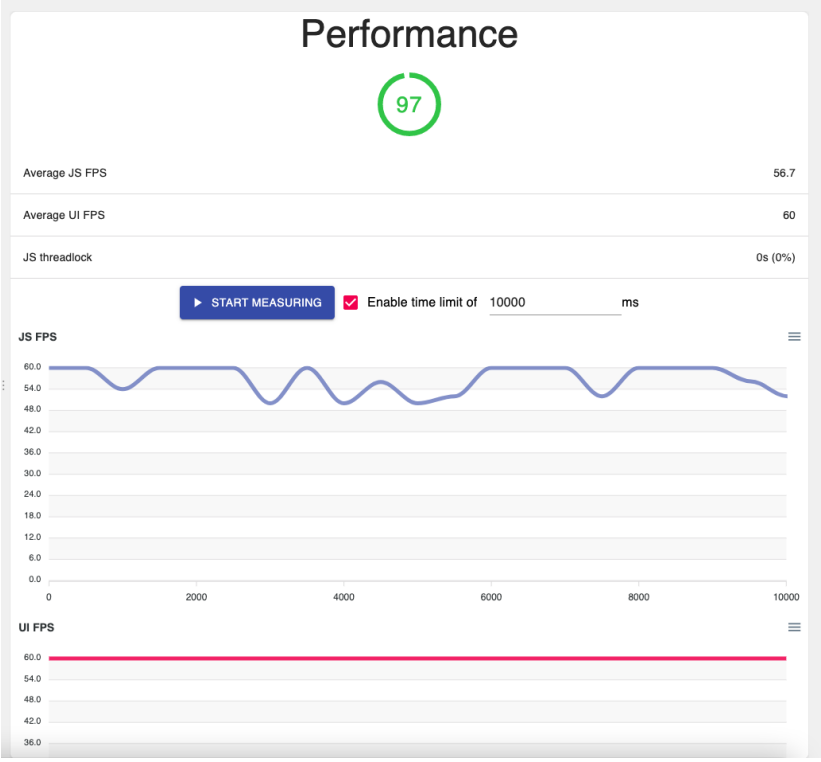

Figura F.10: FlashList Teste 10