Pedro
Azevedo

The Playground: Plataforma móvel para conectar jogadores casuais de basquetebol

The Playground: mobile platform to connect amateur basketball players

PROPOSTA DE DISSERTAÇÃO

Universidade de Aveiro
2026

Pedro Azevedo

The Playground: Plataforma móvel para conectar jogadores casuais de basquetebol

The Playground: mobile platform to connect amateur basketball players

Proposta de Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à conclusão da unidade curricular Proposta de Dissertação, condição necessária para obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Ilídio Oliveira, Professor associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

palavras-chave

Basquetebol, Desporto Casual, Desporto Amador, Kotlin Multiplatform, Desenvolvimento de Aplicações Móveis, Android, iOS, Organização de Jogos, Disponibilidade de Campos

resumo

Este projeto de dissertação de mestrado procura produzir uma solução para um problema encontrado em jogadores casuais e amadores de basquetebol que jogam em campos públicos. Vários destes jogadores encontram dificuldades em formar equipas quando vão a um campo público, ou porque não existem jogadores suficientes, ou quando existem, podem não estar no mesmo nível de competitividade e experiência. O contrário também acontece, o campo pode estar cheio encontrando dificuldades em partilhar espaço com jogadores que apenas querem lançar bolas. Várias plataformas para resolver este problema foram encontradas, mas parecem não ser utilizadas em Portugal. Desta forma, abre-se uma oportunidade de desenvolvimento de uma aplicação para diferentes plataformas móveis focada na comunidade de basquetebol portuguesa. O projeto segue metodologia Agile e User-Centred Design usando Scrum nas fases desenvolvimento. Na primeira fase foi desenvolvido um questionário com 50 respostas que confirmou os problemas identificados: 22% dos jogadores não tem com quem ir jogar, e 10% menciona falta de jogadores em campo. Quanto aos objetivos, 29% vai jogar com amigos contra outros jogadores, e 17% vai apenas lançar bolas. Deste estudo foram desenvolvidas duas Personas: Miguel (jogador competitivo de 25 anos que procura competição regular) e Carlos (lançador flexível de 33 anos com dificuldade em encontrar espaço para treinar o lançamento). Baseado nas personas, foram recolhidos requisitos em formato de User Stories e divididos pelos Sprints. Optou-se por Kotlin Multiplatform para o desenvolvimento da aplicação, Compose Multiplatform para a interface e, funcionando em Android e iOS, e Supabase para gerir o backend. Esta proposta estabelece as bases para uma aplicação multiplataforma que responde a uma necessidade genuína na comunidade de basquetebol portugues. Jogadores casuais enfrentam barreiras significativas na coordenação de jogos e avaliação da dinâmica dos campos públicos. A solução proposta, que integra localização e ocupação dos campos em tempo real, organização de jogos e sistemas de competição, oferece uma abordagem abrangente a este problema. Embora o sucesso da plataforma dependa criticamente da adoção inicial por parte da comunidade, as metodologias adotadas permitem validação contínua com utilizadores reais e ajuste rápido de funcionalidades. Desta forma, a presente dissertação estabelecerá não apenas um produto funcional, mas também uma framework replicável para plataformas comunitárias similares em contextos portugueses.

keywords

Basketball, Casual Sports, Amateur Sports, Kotlin Multiplatform, Mobile Application Development, Android, iOS, Game Organization, Court Availability

abstract

This master's thesis project seeks to produce a solution to a problem encountered by casual and amateur basketball players who play on public courts. Many of these players find it difficult to form teams when they go to a public court, either because there are not enough players, or because, even when there are enough players, they may not be at the same level of competitiveness and experience. The opposite also happens, the court may be full, making it difficult to share space with players who just want to shoot around. Several platforms to solve this problem have been found, but they do not seem to be used in Portugal. This opens up an opportunity to develop an application for different mobile platforms focused on the Portuguese basketball community. The project follows Agile and User-Centred Design methodology using Scrum in the development phases. In the first phase, a survey with 50 responses was developed, which confirmed the problems identified: 22% of players have no one to play with, and 10% mention a lack of players on the court. As for their objectives, 29% want to play with friends against other players, and 17% just want to shoot around. From this study, two Person's were developed: Miguel (a 25-year-old competitive player looking for regular competition) and Carlos (a 33-year-old flexible shooter who has difficulty finding space to practise pitching). Based on the persona's, requirements were collected in User Stories format and divided by Sprints. Kotlin Multiplatform was chosen for the development of the application, Compose Multiplatform for the interface, running on Android and iOS, and Supabase to manage the backend. This proposal lays the foundations for a multi-platform application that responds to a genuine need in the Portuguese basketball community. Casual players face significant barriers in coordinating games and assessing the dynamics of public courts. The proposed solution, which integrates real-time location and court occupancy, game organisation, and competition systems, offers a comprehensive approach to this problem. Although the platform's success critically depends on initial adoption by the community, the methodologies adopted allow for continuous validation with real users and rapid adjustment of features. In this way, this dissertation will establish not only a functional product, but also a replicable framework for similar community platforms in Portuguese contexts.

acknowledgement of use of AI tools

Recognition of the use of generative Artificial Intelligence technologies and tools, software and other support tools.

I acknowledge the use of Perplexity AI pro (https://www.perplexity.ai/) to search for documents, rephrasing and enrich text, spot spelling and grammar mistakes and structure tables with the content provided. Also the user of TexStudio (https://www.texstudio.org/) to write the document.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

It's Saturday afternoon in Lisbon, and you arrive at your local basketball court ready to play. But instead of the competitive 5v5 game you hoped for, you find only two other players shooting around, not enough for a proper match. Meanwhile, at a court just three kilometres away, twelve players are crowded onto a single half-court, waiting their turn and growing frustrated. Both scenarios represent wasted potential: players eager to play, public courts sitting empty, and no way for these communities to connect.. While digital transformation has revolutionized how individuals track fitness activities like running or cycling, team sports coordination remains stuck in an analog era, relying on chance encounters and informal word-of-mouth.

This project emerged from personal experience as a regular casual basketball player at Lisbon's public courts. Over years of playing, observing common frustrations among fellow players, and discussing the challenges they face, ranging from coordination issues to social problems on the court, it became clear that this problem was systemic and widespread. Rather than accepting these limitations as inevitable, this realization prompted the question: could a thoughtfully designed digital platform solve this coordination problem and enable a community of practice around amateur basketball?

This dissertation proposal aims to design and plan the development of a Kotlin Multiplatform mobile application that supports amateur basketball players in organising casual games, checking court availability, and fostering a community of practice around public courts in Portugal, starting in Lisbon. The remainder of this proposal is organised as follows.

In this chapter, the context of basketball in Portugal is introduced, the problem is stated, and a high-level solution is proposed. The following chapter analyses different options for developing mobile applications, including design and development methodologies, backend strategies, and existing projects related to the problem statement. The subsequent chapter presents the methodology that will guide the dissertation, accompanied by a roadmap and sprint planning (which may be adjusted according to project progress), and highlights the selected technologies and application architecture. Finally, the last chapter reports the initial research and design work already carried out, which will serve as the foundation for the subsequent implementation and evaluation phases.

1.1 Basketball Context in Portugal

Portugal is clearly not known for its basketball, the quality is far behind other European countries. In the International Basketball Federation (FIBA) world ranking[1], Portugal is in the 47th place, and in the middle of European table in 25th place. Comparing with our neighbours Spain that is in 7th in the world ranking. However, in the past years, the Portuguese basketball achieve some marks.

First, Portugal had its first player in one of the best women league in the world, "Ticha" Penicheiro played during 15 seasons in the WNBA in th USA, winning a title with the Sacramento Monarchs, and some individual awards. And in 2019, entered the Women's Basketball Hall of Fame, that has the goal of honouring the most influential players in women basketball.[1]

After her, Neemias Queta was drafted to the NBA in 2021, being the first Portuguese in the NBA, and in the 2023-24 season won his first title with the Boston Celtics. Besides not being the most valuable player in his team in the USA, he played an important role with the National team in the second appearance in the EuroBasket tournament in 2025. In this campaign, the Portuguese team made a surprise performance, passing through the group phase and then, confronting the German team. In this game the Portuguese team was holding up until the last quarter, where the world champions and the winners of the EuroBasket 2025 took the victory. After the first victory in the EuroBasket, Neemias mentioned that it was a great moment for the sport in Portugal and wants the sport to grow more in Portugal.[2]

According to data from the *Instituto Português do Desporto e Juventude* (IPDJ) the Portuguese Basketball Federation registered 31,359 federated athletes in 2024, ranking it as the fifth sport most practised federated sport in Portugal, following football, swimming, volleyball and handball[3]. While the COVID-19 pandemic caused a generalized decline in participation across almost all sports, basketball has shown a steady recovery However, unlike counterparts, it has not surpass its historical peak numbers. The Figure 1.1 illustrates these evolutionary trends from 2013 to 2024 for the top federated sports, excluding football, this remains a distinct outlier with 238,441 athletes in 2024, nearly the double that of swimming, and is omitted to maintain the readability of the comparative scale for the remaining modalities.

[1] https://www.fiba.basketball/en/ranking/men

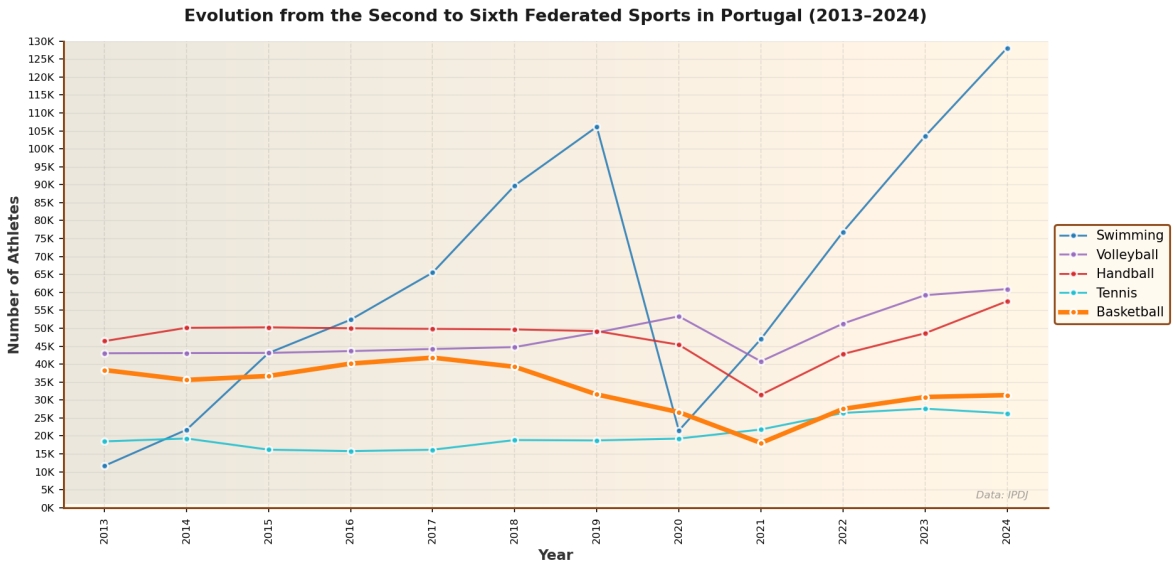

Figure 1.1: Evolution of federated athletes in Portugal's top sports (excluding football) from 2013 to 2024. Data source: IPDJ[3]

1.2 Problem Statement

As a recurrent casual basketball player who frequents public courts with friends to play against other players, I usually encounter issues that degrade the overall experience. The most salient one is the uncertainty regarding how many players will be present at a given court, sometimes there are not enough players to form teams, other times there are too many players for the available court space, and other there is a good number of players. To validate these observations, I conducted a survey for players who frequent attend public courts, gathering 50 responses, which are analysed in 4.1.1.

This revealed similar challenges with the ones I encounter. Players do not go more often to courts because they do not know or find enough people to play with, or because the court is overcrowded. This apparent paradox reflects temporal misalignment, these players may visit at different times without knowing the others intentions or schedules. In practice, court usage fluctuates throughout the day and week, and the occupancy depends strongly when and where each player decides to go. This suggest a lack of initiative or communication mechanism to coordinate games and choose the right place and time to go to a court according to the player's preference. Players have limit information about who is planning to go to which court and when, causing a bad experience and low motivation to go back to the same court.

Players also report poor court conditions, and social problems between players, including aggressive or unpleasant behaviour, individualistic style and mismatched competitiveness (players who do not take the game as seriously as the others and whose skill and attitude creates poor experience for the others).

From this analysis, it can be concluded that basketball players who want to go to public courts and play casual amateur games or practice alone, lack a shared reliable channel to

signal intentions or to observe the occupancy in advance, or even to organize games, leading to individual and independent decisions that collectively result in inefficient usage of public court, missed opportunities for casual games, and fewer motivation to play basketball.

1.3 Proposed Solution

The proposed solution is to develop a digital platform where amateur basketball players can organise casual games in advance and check court availability to plan when and where to play. This platform improves upon the current fragmented coordination by making players' intentions visible, enabling them to find each other and use public courts more efficiently. Beyond functional coordination, the platform facilitates the formation of a Community of Practice (CoP) around amateur basketball. According to Lave and Wenger's social learning theory, communities of practice are groups of people who share a concern or passion for something they do and learn how to do it better as they interact regularly[4]. The platform enables the three essential characteristics that distinguish a community of practice, being the domain (casual basketball), community (player engagement and knowledge sharing), and practice (the regular activity of organising and playing casual basketball together). Through the platform, players engage in joint activities, move from peripheral participation towards fuller engagement in the community, and develop identities as members of the amateur basketball community.

Development and initial maintenance of the platform will be conducted by the author, with institutional supervision provided by the University of Aveiro as the core component of the Master in Informatics Engineering dissertation. Upon completion of this project, the platform is developed with the perspective of future developments and continuity. One of the key objectives for now is to develop a product and after making it available to the community.

To keep user's engagement, the platform will use a *gamification* model using leaderboards, and for different courts to incentive players to go conquer the other courts giving life to them and not overcrowd the others, by spreading players. Also the use of notifications to inform about new games, friends' activity, giving challenges, badges and achievements, rewards if partnerships are achieved, and keep bringing platform updates and novelties.

To prevent misuse and maintain a trust environment, the platform will implement foundational safeguard from the start. First, accountability is established through linking account to a verified email address and phone number, discouraging disposable accounts and anonymous abuse. Second, the platform empowers the community to self-regulate using reporting and endorsement mechanism. A user reporting system allows players to flag inappropriate content or concerning behaviour for review. Users that organise regular games and have positive feedback receive a badge for verification.

Technical safeguards ensure information quality and prevent manipulation. Geolocation verification is using during check-in to confirm physical presence at a court, preventing false occupancy reporting. Players are able to decide who sees their location information, to prevent stalking. Finally, restricting the number of games a user can create per day to prevent spam

and platform manipulation.

The proposed solution will focus on two primary features:

Informal Games "Pickup games"[2] - Enable players to create, discover and join informal matches (pickup games), with features that support different play modes and social organization:

- Players can create, search for and join games listed on the platform;
- The games can be competitive or casual;
- Competitive games contribute to a leaderboard/raking table per court;
- Game formats include 1v1, 3v3, 4v4, 5v5;
- Users can create and mange persistent teams and challenge other teams;

Court Availability and condition awareness Provide users with real-time and information about courts so they can decide where and when to play:

- Persistent court catalogue with attributes - name, location, full/half court, number of courts, has water fountain, etc;
- User can check-in to a court;
- Users can indicate intent to attend a court at a specific time, so others can have an idea of the availability;
- Live occupancy indicators to show how many users intend to go and how many are currently present;
- Court condition reporting by the users, they can submit short status updates to report issues.

These features work with the *gamification*, community-building, and trust mechanisms described above, creating an integrated platform that reduces coordination friction while maintaining the social and competitive dimensions that define casual basketball culture.

[2] "Pickup games" is the name used by basketball players to call a game without a formal organization (not in a league, no referee, no strick rules)

CHAPTER 2

State of the Art & Existing Solutions

This chapter analysis the technologies that exist to develop mobile applications, methodologies and back end strategies, to understand what options exist and which is more suited for the task in hand. Besides that, a study of existing solutions or related projects is conducted to find gaps in them and see how this project can be of value.

2.1 Mobile Application Implementation Technologies

This platform has to reach the maximum basketball players for it to work properly, and should work when the players are in the court, or anywhere. In order to achieve this, the platform should be access from the mobile phones. Statcounter Global Stats reports that in Portugal in September 2025, the Mobile Operating System Market Share has 65.65% Android users and 33.94% iOS[5]. Despite the fact that more than half of people uses Android, the goal is still reach the maximum players possible, so the platform should be accessible from Android and iOS phones.

In order to develop a mobile application for both platforms, there are different approaches that can be selected for the implementation, which are the native, cross-platform, web, hybrid, modeling, cloud-based and merged approaches[6].

2.1.1 Native Apps

Going Native, means that the application will be developed to a specific OS, using its programming language. To build native Android applications, developers typically use Kotlin or Java. Both language are compiled to Java Bytecode that runs on the Android Runtime (ART). As for iOS, developers use Swift or Objective-C, mainly the first one.

Unlike other solutions that add abstraction layers (like React Native or Flutter), native applications compile directly to the platform's native language. This results in better performance and faster execution compare to other solutions, which will be discussed later. In addition to performance, native development offers security benefits, since developers can take advantage of the built-in security features of the operating system. When it comes to User Interface (UI)

and User Experience (UX), each platform provides its own guidelines where Android follows Material Design, while iOS the Human Interface guidelines. This consistency makes applications easier to navigate, and there are no issues in the application adaptation to other devices.[7]

Finally, native applications provide superior direct access to the device's complete hardware and sensor ecosystem, including the camera, microphone, GPS, accelerometer, gyroscope and more[7]. This type of access enables full integration with device features for enhanced functionality, and can give better recognize user activity and environmental conditions, enabling the application to infer user context (for example, distinguishing activities like walking or running) by processing sensor data locally on the device[8].

When the goal is to deliver an application for both Android and iOS using native development, each feature must be implemented twice, once for each platform and in different programming languages. As Robert Nagy explains in *Simplifying Application Development with Kotlin Multiplatform Mobile*[7], the development cost of this approach can be expressed as:

$$Costofdevelopment(n) = n * FC$$

where n represents the number of platforms and FC the *feature complexity*, defined as the sum of all sub-features that compose a feature. Nagy notes that this formula and its graphical representation are approximations that describe reality simplistically, intended to illustrate the cost differences between development approaches rather than provide precise metrics.[7] The core insight is that native development costs scale linearly with the number of platforms, as developing for two platforms costs approximately twice as much as developing for one, and three platforms costs three times as much.

Beyond this, additional factors complicate native development. Each platform has distinct characteristics, and a solution that is straightforward to implement on one platform may be unavailable or significantly harder to achieve on another. This divergence creates extra overhead in aligning features sets across platforms. So, Nagy introduces the additional component, *Synchronization Cost*, which increases in an exponential way as the number of features and its complexity grows.

2.1.2 Web Applications

Any application that is access through *HTTP* requests over a network is called a web based application. These applications are implemented with HTML, CSS and JavaScript. Until Progressive Web App (PWAs) appeared, these applications were browser only, but now, they can be installed in the device, can have offline availability, push notifications and background sync, also able to run on desktop. Developers normally use Angular or React, and other frameworks are StencilJS or Svelte. However, PWAs cannot access key platform's features as the Native apps can, and also, has a possible higher energy consumption.[9]

2.1.3 Hybrid Applications

Another option, also leveraging web technologies such as HTML, CSS and JavaScript, is taking a hybrid approach. The application is developed with as web app and developers wrap the app via cross-platform wrappers, frameworks and tools, as a native app. With this approach, the app has a user experience just as a native app. Some hybrid app frameworks are Cordova, Phonegap, Titanium, Ionic, being the first the most common. Using third-party plugins with JavaScript, it is possible to access features form the device, such as the camera and GPS. In the Maher Gerges and Ahmed Elgalb article[10], where they compare different approaches, it is compared the hybrid apps directly with the native apps, and, in CPU and Memory occupancy, the hybrid cost more, being 106% and 73.0% higher in respectively. Also, not all features are available and the performance is worse than native applications.[10]

2.1.4 Cross-Platform Applications

In order to avoid repeating the same code for different platforms, there are Cross-Platforms approaches. With this method, it is possible to create applications that run on multiple operating systems. In a first look, using cross-platform methods can be an effective solution to develop applications for multiple platforms, when time, cost and basic interoperability is the primary concern. On the other hand, Cross-Platform applications tend to have the worst performance and user experience compared with Native applications, as they are behind the evolution of Android and iOS and cannot take full advantage of each platform's features[7]. In different articles and papers, the categorization of the approach varies, especially, when it comes to the Cross-Platform approach, there are subcategories[11]. The Web and Hybrid approach is normally inserted in this category, alongside with Interpreted approach and Cross-Compiled approach[11], which the most popular used frameworks are React Native and Flutter, respectively.

Interpreted Approach - React Native

Similar to the web and hybrid approach, uses web technologies such as CSS and JavaScript, however, the user interface is not HTML and not render in the browser.[9] The most popular used framework is React Native from 2015, developed by Meta, where developers build applications with React components, and the logic in JavaScript. This framework uses component-based architecture, following the steps of React for the web, promoting code re-usability, modularity and easy debug.[12] The JavaScript code executes on a separate thread and communicates with the native layer through a bridge[7], allowing developers performance-critical parts of the application in native languages[12]. The bridge translates asynchronous, serializable data between the two environments, which enables native rendering of components. However, this approach introduces overhead, resulting in a process generally slower compared to fully native applications.[7]

Cross-Compiled Approach - Flutter

In this approach, the application is developed in a common language, and then, it is compiled to the native code that can be executed on a mobile device, with Flutter and Xamarin popular examples[9]. Flutter is a framework developed by Google, its architecture can be described in three layers[7]. The Framework layer, where developers write the application code and the UI

components, declaratively using the Dart language. The Engine layer renders Flutter widgets to a canvas called Skia Canvas, which is then passed to the final layer, the native platform. The platform displays the canvas and sends back user events to the framework.[7] Compared to React Native, Flutter often achieves better performance by relying on Android's Native Development Kit and iOS's Low-Level Virtual Machine to compile code coming from the engine, Flutter applications deliver higher execution performance. Nevertheless, certain parts of a Flutter application may still need to be implemented in Java/Kotlin and Obj-C/Swift code. Communication between these native components and the Dart code is through a channel, reducing scalability[7].

Cloud-based Approach and Modelling Approach

A Cloud-based application, has its back end in the cloud, instead of running locally, using the cloud features such as flexibility, virtualization and security[11]. Companies like Amazon Web Services, offer that type service called Back end as a Service (BaaS)[6].

As for the Modelling applications there are two subcategories, the Model-Based User Interface Development (MB-UID) and the Model-Driven Development (MDD). Developers use abstract model to describe tasks, data, users and the user interface, which will generate source code. It can transform to code for different platforms. Some tools include XMobile, for MB-UID and JavaScript Application Framework for MDD[6].

Low-Code Development Platforms represent the modern evolution of MDD principles, combining visual with cloud-based infrastructure and automatic code generation. These platforms, including OutSystems, Mendix, Zoho Creator, Lansa, and Salesforce, enable rapid application development through visual drag-and-drop interfaces, pre-built components, and What-You-See-Is-What-You-Get (WYSIWYG) editors.[13]

This approach encounter challenges like scalability, model management issues which are important to increase the efficiency of automation, support scalable operations, and for manipulating, querying, and analysing models. Besides that, optimization of large models and when several dedicated languages are used to carry out different model management strategies, the space and overall complexity increase as the size of the model grows[13]

2.1.5 Kotlin Multiplatform

Although the previous frameworks solve the problem of dealing with different platforms, it is still not optimal if the application demands platform specific features, and keeping up with Android and iOS updates. To address these conditions, developers can choose Kotlin Multiplatform (KMP), also a cross-platform solution.

According to JetBrains documentation, Kotlin Multiplatform is an open-source technology developed by them that enables developers to create applications for multiple platforms efficiently by reusing code across them and still having the native approach benefits[14]. Developers can share code without losing the qualities of native programming, have good user experience, good application performance and having full platform capabilities, alongside the benefits of the cross-

platform, reducing development time, consistency in behaviour in the different platforms[15]. In this approach, the developer has 3 options, it can share and only write a critical feature in Kotlin, or implement the full logic and only develop the UI natively, and finally, with Compose Multiplatform, also share the UI[14].

Jetpack Compose[1] is a tool-kit to develop the User Interface of Android applications. It is implemented in a declarative way, compatible with Kotlin APIs, enabling developers to intuitively build fast UIs. It also supports Google's Material Design, an "open-source design system for building beautiful, usable products"[16]. Since 2021, developers can build shared UIs for Android, iOS, desktop and web, feeling natural on every platform with Compose Multiplatform.[17]

Thompson Carter, refers in his book[18] the same benefits of KMP, code re-usability, faster development, reduce maintenance (as it is only need to fix bugs in one place), flexibility (share code and still access platform-specific features and APIs) and community support. However, he mentions some challenges as:

- Maturity, as the KMP is recent and is still evolving, some libraries and tools may not be fully supported;
- The integration with Platform-Specific APIs can be challenging;
- The transition from single-platform development can be a bit difficult, with the introduction of new paradigms;
- And the integration with an existing project can involve significant re-factoring.

2.1.6 Comparative Analysis

On one hand, using cross-platform frameworks can be an effective solution to develop applications for multiple platforms, when time, cost and basic interoperability is the primary concern. On the other hand, these frameworks tend to be behind the evolution of Android and iOS, and cannot take full advantage of each platform's features. Robert Nagy therefore refines the development-cost equation as:

$$Costof Development(n) = FC * (1 + Costof goingNative)$$

In this equation, n is the number of platforms and FC is the feature complexity. As for the *Cost of going Native*, it reflects the additional effort required when some parts must be implemented with native code. This theoretical model assumes that the *Cost of going Native* is not a fixed value. It is important to note that this is a theoretical model without empirical data; instead, Nagy identifies three factors that contribute to the cost:[7]

First, the level of interoperability that the cross-platform framework provides with native components, which is typically not optimal. When developers need to bridge framework code with platform-specific APIs, they must invest time to understand how to properly connect these two different environments.

[1] https://developer.android.com/compose

Second, the knowledge gap between the framework and native languages adds to the expense. Developers who specialize in the cross-platform framework (e.g., Dart or React) may not be familiar with native languages like Kotlin, Java, or Swift. Learning these languages and implementing features correctly requires additional time and effort.

Third, synchronization costs arise when maintaining consistency between code written in the framework and code written in native languages. As the application grows and more native code is added, keeping both implementations synchronized becomes increasingly complex and time-consuming.[7]

Besides this a roadblock can occurs when a required feature cannot be implemented only with the framework and must instead be written in native code, producing a sudden increase in cost. If no native features are required, cross-platform development can remain less expensive compared to native development, despite the discussed limitations. However, as more features demand native implementation, the cumulative cost eventually exceeds that of a fully native approach.[7]

Table 2.1 provides a comparative analysis of mobile development implementation technologies, synthesizing key characteristics, advantages, and disadvantages.

Table 2.1: Comparison of mobile application development implementation technologies [6], [12], [15]

Approach	**Advantages**	**Disadvantages**
Native apps	Best performance; full access to platform APIs and sensors; highest UX fidelity; mature tooling and debugging. Very large, established communities; extensive third-party libraries.	Two separate code bases for multiple platforms with significantly higher cost; steep learning curve requiring platform-specific language expertise (Kotlin/Java or Swift).
Web apps / PWAs	Single code base; runs on desktops and mobile browsers; easy distribution via web. Massive developer community using standard web technologies; low learning curve building on existing web knowledge; no licensing costs.	Limited access to native features and device sensors; performance and energy use vary significantly depending on internet connection; difficult to monetize; cannot be published in app stores.
Hybrid (wrapped web)	Faster port from existing web applications to installable apps; single web code base; plug-in ecosystems (Cordova, Capacitor) enable device feature access; large, active community; builds on web development skills.	Worse performance than native due to WebView execution; higher CPU and memory usage; some native features unavailable; UX can feel non-native; ecosystem declining in adoption.
React Native (interpreted)	High code reuse across platforms; strong React ecosystem with extensive third-party libraries; UI renders using native components improving responsiveness; good developer velocity; very large and active community with strong backing from Meta; MIT license (no vendor lock-in).	Bridge overhead can reduce performance for heavy UI/CPU work; some advanced features require native modules; platform parity depends on third-party library support; more complex debugging.
Flutter (cross-compiled)	Compiled to native code (via Dart) ensuring high performance; consistent UI across platforms via Skia rendering engine; rich, customizable widget set enabling polished UIs; hot reload for fast iteration; rapidly growing community with Google backing; BSD license.	Larger app binary sizes compared to native apps; occasionally needs platform channels for native feature access; requires learning Dart, a less common language; moderate-to-high learning curve.
Model-Driven / Low-Code	Rapid development using visual, declarative modelling; minimal coding required reducing learning curve; pre-built components and services; strong enterprise ecosystem; integrated deployment pipelines and monitoring.	Moderate performance compared to compiled native approaches; limited flexibility for complex custom logic; vendor lock-in with proprietary platforms (e.g., OutSystems, Mendix); high licensing costs for enterprise features.
Kotlin Multiplatform (KMP)	Share core business logic (networking, storage, domain rules) across platforms while retaining native UIs or using Compose Multiplatform for shared UI; near-native performance and good interoperability with native code; builds on Kotlin knowledge; Apache 2.0 license; JetBrains backing.	Ecosystem still maturing with potential gaps in third-party libraries; some libraries may be platform-limited; complex when integrating with existing native projects; tooling and ecosystem support are improving.

2.2 Mobile Application Development Methodologies

While the selection of implementation technologies determines the technical foundation for mobile application development, the development methodology shapes how the entire development process is organised, executed and iterated upon. The methodology addresses how the work is structured, planned and managed throughout the project life-cycle.

Mobile application development present distinct challenges that differ from traditional software development, due to rapid platform evolution, device fragmentation, high user expectations for performance and usability, frequent update cycles, and the need for continuous adaptation to changing requirements and user feedback. Consequently, development methodologies specifically designed for mobile contexts have emerged, alongside adaptations of established software development approaches.

Agile

Agile Software Development is centred on four core values such as prioritizing individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan[19].

The Agile Manifesto[20] also provides twelve principals in order to follow the four core values. These principles prioritize satisfying customers through early and continuous delivery of valuable software, and embrace changing requirements even late in development. They emphasize delivering working software at frequent intervals with shorter time-scales, supported by daily collaboration between business people and developers. Teams should be built around motivated individuals with proper environment and support, using face-to-face communication as the primary method of information exchange. Working software is the primary measure of progress, and sustainable development pace should be maintained continuously. The principles further stress continuous attention to technical excellence and good design, while maximizing simplicity by minimizing unnecessary work. Finally, self-organizing teams are encouraged to emerge with the best architectures and designs, and teams should regularly reflect on their effectiveness to adjust their practices.[19]

With Agile core values and principals, challenges like the static nature of requirements specifications and the disconnection between initial planning and actual user needs are mitigated. Agile embraces iterative and incremental development. Work is organized into short cycles called sprints or iterations, during which teams deliver working software increments. This approach enables rapid feedback from users and stakeholders, allowing requirements to evolve and be re-prioritized continuously throughout the development process.[19]

Agile has frameworks that help putting this principals to work. There is Scrum, that provides a structured approach with defined roles (Product Owner, Scrum Master, Development Team), time-boxed sprints, and ceremonies (daily, sprint planning, retrospectives) that maintain team organization and accountability. To offer a more continuous flow approach, there is Kanban that

visualises the work on a board with columns representing stages (To-Do, In Progress, Done), allowing teams to prioritize dynamically and limit work-in-progress. Extreme Programming (XP) emphasizes technical practices such as pair programming, test-driven development, continuous integration, and collective code ownership to maintain code quality and enable rapid, frequent releases. There are other methodologies that provide specific practices and focus areas, but all share the core values of the Agile Manifesto. [19]

However, Agile methodologies present a significant challenge regarding usability and user experience. Agile was primarily designed by and for developers, focusing on implementation practices rather than interaction design or user research. Consequently, usability and user experience quality are often treated as secondary concerns, without being planned systematically. Research has shown that Agile teams frequently de-prioritize usability activities such as user testing, assuming such activities consume too much time within tight sprint schedules. Additionally, the role of usability in software development is often unclear within agile teams, leading to instances where teams either neglect usability entirely or implement it incorrectly.[21]

As Mobile development requires a better usability and user experience focus, Agile must work with User-Centred Design.

User-Centred Design (UCD)

User-Centred Design is a development philosophy that positions usability as a primary goal rather than an afterthought. Usability, defined as the degree to which a system can be used by a specific type of user in a specific context to achieve specific goals, is particularly critical for mobile applications. The mobile form factor introduces unique design constraints that fundamentally differ from desktop, like the shape and size of screens restrict how information can be displayed, input methods favour certain interactions over others, and their portability enables use across diverse environments and contexts, including situations where the user's attention may be divided.[21]

UCD is an iterative process focused on the users and theirs needs in each design phase, by actively involving users throughout the process. The goal of UCD is to address the whole user experience, and shape the application to the specific needs of target users[22]. To implement UCD systematically, there are 4 phases defined by the *ISO 9241-210:2019*[23], which establish Human-Centred Design (HCD) principals applicable to UCD. While the standards focus on HCD, UCD applies these principals with explicit focus on a defined target user population rather than broader human contexts. The four phases are:

- **Understand and specify the context of use**: This phase focuses on understanding the target users, their needs, pain points, and the environment in which they will use the system. Research methods such as user interviews, observations, and contextual analysis are employed to build a comprehensive understanding of user contexts;
- **Specify the user requirements**: With a clear understanding of users and their contexts, this phase involves gathering and documenting specific user requirements and desired

features. Requirements are derived from user research findings and articulate what the system must do to address identified user needs. Requirements are a set of guidelines that state what the functionality and capabilities of the platform system, which meet the user's needs and business's objectives. This will guide the development process, ensuring the end product is user-friendly, accessible and valuable[24];

- **Produce design solutions to meet user requirements**: Based on validated user requirements, design solutions are created to address identified needs. This phase includes creating wire frames, prototypes, and mock-ups that translate requirements into tangible design artefacts;
- **Evaluate the designs against requirements**: Prototypes and design solutions are tested with actual users to validate that the designs meet specified requirements and user needs. Feedback from evaluation informs iterative refinements until user satisfaction and requirement fulfilment are achieved.

Agile User-Centred Design Development (AUCD)

The integration of Agile with UCD seeks to leverage the strengths of both methodologies, enabling rapid delivery of working software while maintaining a strong focus on user needs throughout the project life cycle. Integration strategies range from selectively embedding usability practices within Agile work-flows to adopting comprehensive Agile–UCD frameworks. Two main structural patterns are commonly described in the literature: the "one sprint ahead" approach and the "within iteration" approach.[21]

The *one sprint ahead* makes the team responsible for the design to work in a different sprint than the developers. While sprint two is being developed, sprint one is being tested and the sprint three is being design. This potentially can decrease agility as difficult to accepting changes during the development process rises. As for the *within iteration*, as the name implies, the design artefacts are developed in the same iteration they are produced.[21]

Mobile application development using Agile-UCD employs a range of UCD techniques tailored to project needs and constraints. User-testing is the most use following this study [21], different types of prototypes (functional, mock-up, paper, wire-frame) are also included in this Agile-UCD integration.

2.3 Mobile Back end Development Strategies

2.3.1 Back end Fundamentals

Mobile Applications follow a distributed architecture that has 2 components, front end and the back end. The front end, also called the client-side is the part of the application that runs locally in the user's device. This layer contains the user interface and presentation logic and may also include part of the domain or business logic, implemented using Kotlin Multiplatform and Compose Multiplatform on the client devices. The back end, also called the server side, runs on remote infrastructure rather than on the user's device and is not accessed directly by users. It is responsible for operations such as reading, writing, updating, and deleting data in the database,

authenticating users, and enforcing application rules. Although some business logic traditionally associated with the back end can also be implemented within the main application on the device, this work is still considered part of the client-side layer in this context.

De Carli (2023, Chapter 2: "Introduction to Firebase") describes a traditional mobile architecture example[25]. He uses the Figure 2.1 to represent an iOS application that has the front end, the user interface code in Swift, the back end that can manipulate data coded in Ruby or Node.js, and a cloud service, that distributes the data in the cloud (Microsoft Azure, AWS, Google Cloud). He mentions that the application communicates with a virtual machine that can be coded in Ruby, Node.js or even PHP, that is hosted by a cloud service or, less common now, in a physical server. This is a traditional back end, which, in a complex application is usually required to have separate developers for each part of the application, including a back end developer to code the logic and APIs, and a DevOps to distribute in the cloud[25].

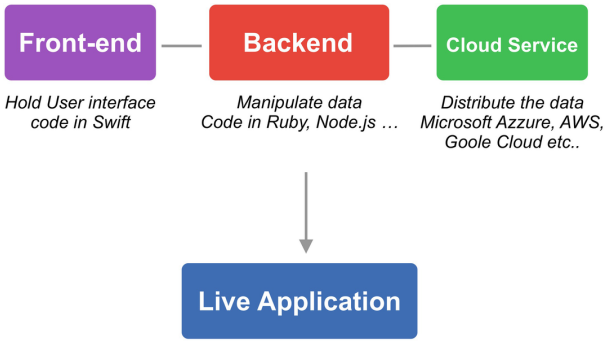

Figure 2.1: An iOS app built with a back end host on a cloud service[25]

However, a different strategy for the back end, that is more recurrent for developers that want to focus on the application development, is the use of Backend as a Service (BaaS) or Mobile Backend as a Service (MBaaS)[26]. BaaS is a cloud-based service model that provides back end features and tools needed for the mobile application that are executed in the server, such as user authentication, database, remote updating, and push notifications, as well as cloud storage and hosting, that speeds up development cycles and decreases the overhead involved in infrastructure management[27], [28]. Firebase[2], is an example of a BaaS, and in the second chapter of Sullivan De Caril' book, he introduces the service and shows the Figure 2.2, that represents the iOS application integrating Firebase. With this, it is possible to see that the back end is managed by the tool and, as Firebase is from Google, it is also built in Google Cloud, having a simpler architecture. The application receives data from Firebase using the APIs provided, without needing to write code on the server side or having another provider to host[25].

The disadvantage of these platforms include the lack of flexibility and fine-grained control, specially when the developer has complex data, custom analytic or there is proprietary infrastructure. Besides that, when choosing a platform, the application developer is subject to vendor lock-in, becoming dependent in the platforms tools, APIs and data formats, if going with the most know ones. With this, developers can also go for a hybrid approach, using features form

[2]https://firebase.google.com/

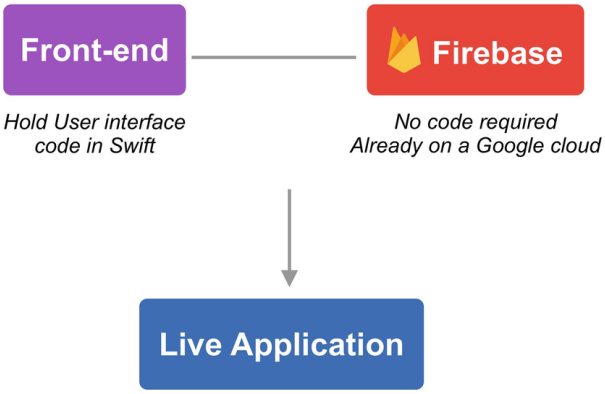

Figure 2.2: An iOS app built with Firebase's back end[25]

the BaaS and developing other back end features with a programming language[28]. In 2.3.4 is discussed some of the options of MBaaS.

2.3.2 Data management strategies

In mobile applications, data management is a critical component that directly influences application performance, user experience, and business logic execution. The ability to store, retrieve, synchronize, and secure data determines the success of most applications. Mobile applications are characterized by specific architectural requirements that differ fundamentally from traditional information systems. These characteristics function as explicit data management requirements rather than mere technical preferences, necessitating consideration when designing data management strategies.

Mobile applications serve multiple users who access the same data from diverse devices and platforms (iOS, Android, Web, Desktop). This distributed multi-device environment requires a centralized Single Source of Truth, an authoritative data repository ensuring all users see consistent, up-to-date information regardless of device or timing of synchronization. Local-only storage on individual devices becomes insufficient when multiple devices must maintain synchronized state, as data inconsistencies would fragment the user experience across platforms.[29]

With multiple users simultaneously accessing, writing, and updating data, applications must maintain real-time synchronization to preserve consistency across all clients. A critical challenge emerges when multiple users edit the same data while off-line which is conflicting changes that must be resolved upon reconnection. The application must therefore implement sophisticated conflict resolution strategies to determine the authoritative state when divergent modifications converge during synchronization, which will be analysed in more detailed in 2.3.3.

Another issue is that network connectivity for mobile users varies significantly across time and environment, ranging from high-bandwidth Wi-Fi to constrained cellular connections to complete off-line states. Mobile applications cannot assume reliable continuous connectivity. Instead, they must adopt offline-first architecture, prioritizing local data availability over network dependency. With this approach, the application will remain usable without a reliable network connection, it will present the user with the local data instead of waiting for the first network

call to complete or fail, and it will fetch data when the optimal conditions are met[30].

Mobile devices operate under significant hardware limitations compared to desktop systems, being constrained with processor speed, limited memory, finite local storage capacity, and battery constraints. These inherent limitations require data management strategies emphasizing minimal memory footprint, efficient access patterns, data compression techniques, and energy-efficient synchronization protocols. Database technology selection must prioritize lightweight solutions, ensuring applications consume minimal computational resources while maintaining reliable data persistence.[30]

Mobile applications have a big volume of multimedia, with users continuously capturing photographs, video recordings, and audio content through integrated device sensors. Since multimedia files are large and users expect access across all devices, local-only storage becomes impractical and Cloud-based object storage solutions become essential, and fast delivery is important.[31]

The convergence of these requirements establishes that mobile applications are naturally suited to cloud-based infrastructure, where external providers offer real-time capable databases, bidirectional synchronization, object storage with CDN integration, background task orchestration, and horizontally scalable infrastructure.[30], [31] These constraints and capabilities are particularly visible in Mobile Back end as a Service (MBaaS) platforms, which will be analysed in detail in 2.3.4.

There are two fundamentally different approaches to databases in mobile application development, the relational and non-relational databases. Relational databases, organize data into structured tables with predefined schemas and utilize Structured Query Language (SQL) for data manipulation, providing strong ACID properties (Atomicity, Consistency, Isolation, Durability) that ensure transactional reliability and data integrity[32]. Non-relational or NoSQL databases employ flexible, schema-less data models that accommodate diverse data types without rigid table structures, offering horizontal scalability and superior performance for unstructured data and high-volume operations [32]. For mobile applications, some examples of databases include, MySQL[3], PostgreSQL[4] and SQLite[5] as Relational databases. MongoDB Realm[6], MongoDB[7], Redis[8], examples for NoSQL, as well as Firebase Realtime Database[9] and Firebase Firestore[10], with this two last being a Google Cloud solution. For local database, usual option are between SQLite, Room[11], that is an abstraction layer over SQlite, Realm or Core Data, Apple's framework for local persistence or cache on iOS[12]. The selection between these database types depends on

[3]https://www.mysql.com/
[4]https://www.postgresql.org/
[5]https://www.sqlite.org/
[6]https://realm.netlify.app/
[7]https://www.mongodb.com/
[8]https://redis.io/
[9]https://firebase.google.com/docs/database
[10]https://firebase.google.com/docs/firestore
[11]https://developer.android.com/training/data-storage/room
[12]https://developer.apple.com/documentation/coredata/

specific mobile application requirements such as data structure complexity, offline capabilities, synchronization needs, and scalability projections, with modern applications often employing hybrid approaches that leverage both relational and NoSQL databases for different functional domains[33].

2.3.3 Data Synchronization

With data in local storage and data in an external database, data synchronization is critical in mobile applications that depend on multi-user interaction, collaboration and shared data. Synchronization ensures that all users and devices have consistent up-to-date information.

Following Zach McCormick and Douglas C.Schmidt paper in data synchronization patters[34], these patterns can be categorized into data synchronization mechanisms patterns (asynchronous vs synchronous), data storage and availability patterns (partial vs complete), and data transfer patterns (full transfer, timestamp transfer, or delta transfer).

Data synchronization mechanisms patterns address how synchronization operations execute relative to the user interface, with asynchronous synchronization performing data transfers in the background without blocking user interactions, thereby maintaining application responsiveness but requiring careful management of concurrent data modifications. While synchronous synchronization blocks the user interface until data transfer completes, guaranteeing immediate consistency at the cost of reduced responsiveness. This synchronous operations are less common, but essential when the users logs in and the system is checking the credentials, for example.

Data storage and availability patterns determine the scope of data maintained on mobile devices, where partial storage retains only a subset of server data based on user needs or device constraints to minimize storage requirements and reduce synchronization overhead. This is the standard for most resource-constrained mobile applications, such as Google Maps downloading only relevant map tiles on-demand, or an e-reader syncing only the specific books a user is currently reading rather than the entire book store inventory[34]. Whereas complete storage maintains a full replica of server data on the device, enabling comprehensive off-line functionality but demanding greater storage capacity and longer initial synchronization times. This pattern is exemplified by applications like Dropbox, which allows users to select specific files or folders for upfront synchronization to ensure availability when network connectivity is lost[34].

Data transfer patterns define what information moves between client and server during synchronization, including full transfer which transmits entire datasets regardless of changes, as one example in [34] is in an application that displays the top ten news articles for a newspaper issued daily (so that the top ten news articles do not change), with this, the dataset changes every time the application is updated, so it uses Full Transfer. Timestamp transfer, which exchanges only records modified since the last synchronization by comparing modification timestamps, commonly used by public transit applications to download only route updates, or by social media APIs to fetch posts created "since" the last check. And delta transfer which computes and transmits only the specific changes or differences between client and server versions, minimizing bandwidth consumption but increasing computational complexity, exemplified by

video conferencing application that transmit only the changed pixels between frames to optimize bandwidth.[34]

When data is written locally while off-line, there will be differences between the on-line data, and there will be a conflict that needs to be resolve before synchronization. There are several conflict resolution strategies, such as Server Wins or Client Wins, where it is decided if the data in the server prevails over the client or the other way, Merge conflict by implementing logic to merge changes when possible with user intervention most likely needed with complex data, and Vector Clocks when there is a complex scenario, helping understand the relationships between changes[35]. Another method that Androids mentions in its documentation is called Last write wins, where the device adds meta data to the written data in the network, and when the network data source receives, any older data will be discard[29]. This can result in data loss if two off-line devices write conflicting updates sequentially, when both reconnect and synchronize, the data written by the first device will be discarded.

For a good user experience on unreliable networks, an offline-first hybrid architecture is a good option. Keeping a local databases for immediate responsiveness, use background workers like Android WorkManager[13] and iOS Background Tasks[14], for scheduled synchronization, and Backend as a Service platforms (Google Firebase, AWS Amplify) for cross-device consistency. Besides that, these background schedulers allow developers to constrain synchronization to unmetered (Wi-Fi) networks only, preventing the application from incurring excessive costs for users in regions where cellular data is expensive or limited[36]. Such architecture balance responsiveness, bandwidth, and storage while allowing application-level conflict-resolution policies to be enforced.

2.3.4 MBaaS Providers

There are several different solutions to choose for MBaaS, from big cloud corporations such has Google's Firebase, Amazon's AWS Amplify, Microsoft Azure, to open-source like Supabase, Parse, Back4App. Below are some relevant MBaaS:

Google Firebase

Firebase was developed by Google and built on Google Cloud infrastructure, that provides various functionalities to help build, deploy and manage mobile a and web applications. Developers do not need to build APIs, as the platform provides its dedicated Software Development Kit (SDK) to interact with its set of tools. Here are the following tools that Google's Firebase provide:

- **Cloud Firestore** - is a flexible, scalable database for mobile, web, and server development. It keeps the data in sync across client apps through realtime listeners and offers offline support for mobile and web so developers can build responsive apps that work regardless of

[13]https://developer.android.com/reference/androidx/work/WorkManager
[14]https://developer.apple.com/documentation/backgroundtasks

network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions.[15]

- **Cloud Functions** - Firebase is a serverless framework that lets developers to automatically run back end code in response to events triggered by background events, HTTPS requests, the Admin SDK, or Cloud Scheduler jobs. JavaScript, TypeScript or Python code is stored on Google Cloud infrastructure and runs in a managed environment. There's no need to manage and scale own servers.[16]
- **Realtime Database** - is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our Apple platforms, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.[17]
- **Authentication** - provides back end services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to the app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more. Integrates tightly with other Firebase services, and it leverages industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with custom back end.[18]
- **Local Emulator Suite** - consists of individual service emulators built to accurately mimic the behavior of Firebase services. This means you can connect your app directly to these emulators to perform integration testing or QA without touching production data.[19]
- **Data Connect** - is a relational database service for mobile and web apps that lets developers build and scale using a fully-managed PostgreSQL database powered by Cloud SQL. It provides secure schema, query and mutation management using GraphQL technology that integrates well with Firebase Authentication. Developers can quickly integrate this product into the mobile and web apps with SDK support in Kotlin Android, iOS, Flutter, and web.[20]
- **Security Rules** - leverage extensible, flexible configuration languages to define what data users can access for Realtime Database, Cloud Firestore, and Cloud Storage. Firebase Realtime Database Security Rules leverage JSON in rule definitions, while Cloud Firestore Security Rules and Firebase Security Rules for Cloud Storage leverage a unique language built to accommodate more complex rules-specific structures. Developers can write simple or complex rules that protect the app's data to the level of granularity that the developer's specific app requires.[21]
- **App Check** - helps protect the application's back end systems from abuse by preventing unauthorized clients from accessing the back end resources. It works with both Google services (including Firebase and Google Cloud services) and custom back end systems to keep resources safe.[22]

[15] https://firebase.google.com/docs/firestore
[16] https://firebase.google.com/docs/functions
[17] https://firebase.google.com/docs/database
[18] https://firebase.google.com/docs/auth
[19] https://firebase.google.com/docs/emulator-suite
[20] https://firebase.google.com/docs/data-connect
[21] https://firebase.google.com/docs/rules
[22] https://firebase.google.com/docs/app-check

- **Cloud Messaging** (FCM) is a cross-platform messaging solution that lets developers reliably send messages. It can notify a client app that new email or other data is available to sync. It can send notification messages to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4096 bytes to a client app.[23]
- **Other Tools** - Firebase provides a several tools for monitoring the app with Firebase Crashlytics and Firebase Performance Monitoring, to engage the user with Google Analytics, Firebase In-App Messaging, Google AdMob, Google Ads and more.[24]

To use Firebase there are two option in terms of Pricing Plans, the *Spark Plan* which is the free plan, and the *Blaze Plan* that is a pay as you go plan, meaning that, when passing a threshold in a specific feature, there is a cost associated. Here are the limits[25]:

- **Authentication** - there is no cost until 50k Monthly Active Users (MAUs), where with the Blaze Plan you pay more for each MAU where the maximum is 0.0025USD (US dollar) per extra MAU if there are more than 10,000,000.
- **Cloud Firestore** - the free version has 1GiB in Store data, 20k document writes per day, 50k document reads and 20k deletions. In the paid plan, when passin the limit the developer has to pay 0.039USD per 100,000 document reads, 0.117USD per 100,000 document writes, 0.013USD per 100,00 document deletions and 0.117 per GiB of storage per month, also having a backup data with 0.039USD per GiB.
- **Cloud Functions** - there is no cost until 2M invocations per month, and there are some limits for the procession power.
- **Cloud Storage** - it has 5GB in the free plan for storage, 1GB per day in download, 20k upload operations per day and 50K downloads. As for the paid plan, it cost 0.026USD per GB extra in storage, 0.12USD per GB extra downloaded and 0.05USD and 0.004USD per 10k upload and download operations respective.
- **Realtime database** - the free tier offers 1GB of storage and in the paid it is 5USD per GB extra.

Supabase

Supabase[26] is an open-source alternative, as it is a combination of different open-source tools that are scalable[37]. Supabase has the following tools:

- **Database** - a Postgres database, available for every project. The relational struture is advantageous for performing complex queries, joins and full-text searches[37].
- **Authentication** - makes it easy to implement authentication and authorization in the app. Supabase provides client SDKs and API endpoints to help create and manage users. Users can use many popular Auth methods, including password, magic link, one-time password (OTP), social login, and single sign-on (SSO).[27]

[23]https://firebase.google.com/docs/cloud-messaging
[24]https://firebase.google.com/docs/run
[25]https://firebase.google.com/pricing
[26]https://supabase.com/
[27]https://supabase.com/docs/guides/auth

- **Storage** - open source S3 Compatible Object Store that makes it simple to upload and serve files of any size, providing a robust framework for file access controls. It can store images, videos, documents, and any other file type. Serve assets with a global CDN to reduce latency from over 285 cities globally. Supabase Storage includes a built-in image optimizer, in order to resize and compress media files on demand[28].
- **Edge Functions** - server-side TypeScript functions, distributed globally at the edge — close to the users. They can be used for listening to webhooks or integrating the Supabase project with third-parties.[29]
- **Realtime** - Supabase provides a globally distributed Realtime service that can send low-latency messages between clients, good for real-time messaging, database changes, cursor tracking, game events, and custom notifications. Track and synchronize user state across clients, ideal for showing who's online, or active participants. And listen to database changes in real-time.[30]

For features not natively provided by Supabase, such as push notifications or built-in analytics, developers can integrate third-party services, webhooks, or direct API integrations[31].

Supabase has 2 main plans, the free option and a pro option where the developer pays 25USD (US dollar) per month[32]. Here what a developer gets with each plan:

- **Database** - The free plan gets 500MB os database size, and the pro gets 8GB with option of paying more 0.125USD per extra GB. With the pro plan, the developer gets automatic backups of seven days, the project is never paused, has point of recovery for 100USD per month per 7 days retention, and more.
- **Authentication** - There are unlimited users, but in the free plan there is the limit of 50,000 Monthly Active Users (MAUs) and 100,000 MAUs for the pro plan, with the option of paying 0.00325USD per MAU extra. The pro plan has extra featuers.
- **Storage** - The free plan has 1GB include of storage while the paid has 100GB then 0.021USD per GB. The free plan has the max file upload size of 50MB and the paid has 500GB.
- **Realtime** - It has the current peak connection where in the free plan has 200 included and the pro plan has 500 with 10USD per 1000 more. The free has 2 Million messages per month included while the pro has 5 Million paing 2.50USD per Million extra.
- **Edge Functions** - The free plan has 500,000 invocations included while the pro plan as 2 Million with 2USD per Million extra.

Parse

Parse is open-source original from Facebook (that decided to shut down and make it open-source), where now, developers have to host the service, either in platforms that can manage

[28]https://supabase.com/docs/guides/storage
[29]https://supabase.com/docs/guides/functions
[30]https://supabase.com/docs/guides/realtime
[31]https://supabase.com/partners/integrations
[32]https://supabase.com/pricing

Parse Server or self-host in a cloud service or in private servers[33]. Parse Platform has similar features has the other services, which are[34]:

- **Database & Storage** - Supports MongoDB and PostgreSQL for structured data, with flexible file storage options including Amazon S3, Google Cloud Storage, or local filesystem storage. GridFS is commonly used as the default large-file storage when MongoDB is chosen[35]. Parse also supports multi-database deployments and can be configured to shard or separate concerns across databases for scale and isolation.
- **Real-time** - Live queries for real-time data updates and built-in push notification support across all platforms, with real time updates.
- **Authentication & Security** - Provides comprehensive user management (signup, login, password reset) with support for OAuth, email verification, and custom authentication.
- **APIs & Integration** - REST and GraphQL APIs with multi-platform SDKs for seamless integration across all major platforms. Supports custom queries and mutations.
- **Cloud Functions** - Allows server-side logic via custom Cloud Functions, database triggers that run on create/update/delete events, and background job processing for async or scheduled tasks.
- **Self-hosting Freedom** - Can be deployed anywhere that runs Node.js (cloud VMs, containers). Full control over infrastructure and data, with a Parse Dashboard for visual management and monitoring, guaranteing complete data ownership.

Parse platform and server are free and open-source, however, the cost lays in the hosting.

Back4App

Back4App is a cloud-hosted solution, that runs and hardened the open-source Parse Platform, exposing the same APIs and SDKs while adding management, scalability, backups and developer tools[36]. With this platform, developers can take advantage of Parse functionalities while simplifying deployment and infrastructure management. In terms of features, they are and work the same as the previous service, Parse, adding hosting, managing and UI features[37]. With Back4App there is no vendor lock-in, meaning that the developer can, any time, export the database, files, Cloud code and run the same Parse Server elsewhere[38].

In terms of pricing, the platform has a free plan, the MVP to "Validate Ideas Quickly — Launch Fast on Our Managed Serverless Back end" (15USD (US dollar) per month paying annually or 25USD monthly), Pay as you go to "Run & Scale Applications on a Serverless Infrastructure" (80USD or 100USD per month) and the Dedicated to "Production-Grade Speed, Isolation & Flexibility on Dedicated Resources"(400USD or 500USD per month)[38]. Here are the differences:

[33]https://www.back4app.com/parse
[34]https://parseplatform.org/#features
[35]https://www.mongodb.com/docs/manual/core/gridfs/
[36]https://www.back4app.com/product/parse-baas
[37]https://www.back4app.com/
[38]https://www.back4app.com/pricing/back end-as-a-service

- **Request per month** - in the free plan is 25k requests, while the MVP is raised to 500k with 5USD per 100k extra request, the Pay as you go 5M with 2USD per 100k extra, and the last one is unlimited.
- **Data Storage** - the free plan has 250MB then is 1GB followed by 3GB and then 8GB, with 15USD monthly for each adicional gigabyte extra.
- **Data Transfer** - the free plan allows 1GB, the MVP is 250GB, the Pay as you go is 1TB and the Dedicated is 2TB, with 0.1USD per each 1GB of adicional transfer.
- **File Storage** - the free plan has 1GB, then 50GB, 250GB and 1TB, with 1USD charge per 10GB extra.

AWS Amplify

AWS Amplify is the Amazon's cloud-powered MBaas and developer tool chain that helps mobile developers build, deploy and scale full stack applications. It offers end-to-end work flows for developing, deploying and managing cross-platform apps, allowing seamless cloud integration.[39] Here are the features[40]:

- **Authentication** - Create seamless on-boarding flows with a fully-managed user directory and pre-built sign-up, sign-in, forgot password, and multi-factor auth workflows. Amplify also supports login with a social provider such as Facebook, Google Sign-In, or Login With Amazon and provides fine grained access control to mobile and web applications. Powered by Amazon Cognito.
- **DataStore** - Use a multi-platform on-device persistent storage engine that automatically synchronizes data between mobile/desktop/web apps and the cloud, powered by GraphQL. DataStore provides a programming model for leveraging shared and distributed data without writing additional code for offline and online scenarios, which makes working with distributed, cross-user data just as simple as working with local-only data. Powered by AWS AppSync.
- **Analytics** - Understand the behavior of the iOS or Android users. Use auto tracking to track user sessions and web page metrics or create custom user attributes and in-app metrics. Provides access to real time data stream and developers can analyze the data for customer insights and build data driven marketing strategies to drive customer adoption, engagement, and retention. Powered by Amazon Pinpoint and Amazon Kinesis.
- **API** - Make secure HTTP requests to GraphQL and REST endpoints to access, manipulate, and combine data from one or more data sources such Amazon DynamoDB, Amazon Aurora Serverless, and the apps custom data sources with AWS Lambda. Amplify enables you to easily build scalable applications that require real-time updates, local data access for offline scenarios, and data synchronization with customizable conflict resolution when devices are back online. Powered by AWS AppSync and Amazon API Gateway.
- **Functions** - Add a Lambda function to the project which the developer can use alongside a REST API or as a datasource in the GraphQL API in the Amplify CLI. The developer can update the Lambda execution role policies for the function to access other resources

[39] https://docs.amplify.aws/android/
[40] https://aws.amazon.com/amplify/features/

generated and maintained by the CLI, using the CLI. Amplify CLI enables developers to create, test and deploy Lambda functions across various runtimes.

- **PubSub** - Pass messages between the app instances and the app's back end creating real-time interactive experiences. Amplify provides connectivity with cloud-based message-oriented middleware. Powered by AWS IoT services and Generic MQTT Over WebSocket Providers.
- **Push Notifications** - Improve customer engagement by using marketing and analytics capabilities. Leverage customer insights to segment and target the customers more effectively. The developer can tailor the content and communicate through multiple channels including email, texts as well as push notifications. Powered by Amazon Pinpoint.
- **Storage** - Store and manage user generated content such as photos, videos securely on device or in the cloud. The AWS Amplify Storage module provides a simple mechanism for managing user content for the app in public, protected or private storage buckets. Leverage cloud scale storage so that is easy to take the application from prototype to production. Powered by Amazon S3.
- **Predictions and Interactions** - Enhance the app by adding AI/ML capabilities and build interactive and engaging conversational bots with the same deep learning technologies that power Amazon Alexa.

AWS Amplify provides a comprehensive tool kit for application development, with open-source libraries, powered by AWS services and can be used with Amplify CLI and Amplify Studio or an existing AWS back end. Amplify Studio brings a visual, collaborative environment for modelling data, managing users and creating back end resources. With Amplify CLI tool chain, developers can configure and maintain back end systems. And the ecosystem, also provides the Amplify UI Components, an open-source set of ready-to-use interface elements for authentication, storage and interactions that integrate directly with back end resources.[39]

- **Data Storage** - No cost up to 5 GB stored on CDN per month, then 0.023USD per GB per month.
- **Data Transfer Out** - Up to 15 GB per month at no cost, then 0.15USD per GB served for additional transfer.
- **Request Count (Server-Side Rendering)** - It covers up to 500,000 requests per month at no cost, then 0.30USD per 1 million requests thereafter.
- **Request Duration (SSR)** No cost up to 100 GB-hours per month, then 0.20USD per GB-hour.
- **Authentication powered by Amazon Cognito** - There are 3 plans, the Essentials, Plus and Lite that provides different features. In the Essencials, the first 10,000 monthly active users (MAUs) are for free, then is 0.0015USD per MAU. For the Plus, is cost 0.02USD per MAU. For the Lite plan, it has a free tier for the first 10,000 raising to 0.0055USD per MAU between 10,001 and 100,000. Then, until 1,000,000 is 0.0046USD per MAU, 0.00325 per MAU until 10,000,000 and greater than that is 0.0025USD.[41]

[41] https://aws.amazon.com/cognito/pricing/

- **Real-time APIs powered by AWS AppSync** - 4.00USD per million Query and Data Modification Operations, 2.00USD per million Real-time Updates, 0.08USD per million minutes of connection to the AWS AppSync GraphQL service. There are different fees for cahing, starting at 0.044USD for 1 vCPU, 1.55GiB of memory and Low to moderate network performance to 6.775USD for 48vCPU, 317.77GiB of memory and 10Gigabit of network performance. The AppSync GraphQL Free Tier offers 250,000 query or data modification operations, 250,000 real-time updates and 600,000 connection-minutes.[42]
- **Data powered by Amazon DynamoDB** - Two pricing options available including on-demand and provisioned. DynamoDB on-demand mode is a serverless option with pay-per-request pricing and automatic scaling, with 0.705USD per million write request units (WCU) and 0.1415USD per million read request units (RCU), the first 25 GB stored per month is free using the DynamoDB Standard table class and 0.283USD per GB-month thereafter. With provisioned capacity mode, the developer must specify the number of reads and writes per second that expects the application to require, with 0.000735USD per WCU, 0.000147USD per RCU and the Data Storage is the same as the other plan.[43]
- **Functions powered by AWS Lambda** Lambda counts a request each time it starts executing in response to an event notification trigger, duration is calculated from the time the code begins executing until it returns or otherwise terminates, rounded up to the nearest 1 ms, with the price depends on the amount of memory the developer allocates to the function.[44]
- **Storage powered by Amazon Simple** - Pay only for what is used. There is no minimum charge. Amazon S3 cost components are storage pricing, request and data retrieval pricing, data transfer and transfer acceleration pricing, data management and insights feature pricing, replication pricing, and transform and query feature pricing.[45]

Microsoft Azure

Microsoft Azure used to have Azure Mobile Services, and in 2016 Microsoft announced the transition to Azure App Service[46]. Now, Azure does not provide a MBaaS like the other providers, but through multiple services in Azure App Service[47]. Azure App Services provides a fully managed platform for building, hosting and scaling web and mobile applications back end systems using multiple programming languages[40]. It allows integrations with the following services:

- **Azure Functions** - Execute event-driven serverless code that responds to HTTP requests, database changes or queue messages, with an end-to-end development experience for scalable APIs[41].

[42]https://aws.amazon.com/appsync/pricing/
[43]https://aws.amazon.com/dynamodb/
[44]https://aws.amazon.com/lambda/pricing/
[45]https://aws.amazon.com/s3/pricing/
[46]https://azure.microsoft.com/en-us/blog/transition-of-azure-mobile-services/
[47]https://azure.microsoft.com/en-us/products/app-service

- **Azure Cosmos DB** - A globally distributed, NoSQL database service that provides access to data with automatic scaling and multi-region replications. It supports several APIs, such asMongoDB, PostgreSQL and Apache Cassandra[48]
- **Azure SQL Database** - A fully managed relational Database as a Service that provides a fast, flexible and elastic SQL database[49].
- **Microsoft Entra ID** - Formerly known as Azure Active Directory (Azure AD), this service manages identity and access, supporting OAuth2, OpenID Connect, and enterprise-grade authentications[50].
- **Azure Notification Hubs** - A cross-platform push notification service that allows the delivery of personalized messages to mobile devices with support for tagging and segmentation[51].
- **Azure API Management** - Enables the creation, protection, and monitoring of APIs used by mobile applications. It provides rate limiting, authentication, analytic and developer portal capabilities[52].
- **Azure Storage** - Offers scalable object, file and queue storage options for unstructured data, media files, and back end integration, forming a common storage layer for mobile applications[53].

For the pricing, each service is billed independently, and has different plans for each services with multiple price variations, including different prices for different regions.

MBaaS Provider Comparison

The table 2.2 presents a comprehensive comparison of the major MBaaS providers currently available in the market. The feature matrix demonstrates that modern MBaaS platforms have converged on a common set of core functionalities, including authentication mechanisms, real-time data synchronization capabilities, serverless computing through Cloud Functions, and comprehensive security rules. All evaluated providers offer APIs (both REST and GraphQL), file storage services, and off-line support, reflecting the maturation and standardization of the MBaaS ecosystem.

However, significant differences emerge in the areas of database technology choices, open-source availability, and infrastructure control. Notably, Supabase distinguishes itself by offering only relational database technology (PostgreSQL), whereas other providers support both NoSQL and SQL databases, providing developers with flexibility in selecting the most appropriate data model for their application. Additionally, the degree of vendor lock-in varies considerably, with open-source solutions like Supabase and Parse offering greater portability and self-hosting capabilities, while proprietary platforms such as Firebase and AWS Amplify provide tightly integrated ecosystems with limited exit strategies.

[48] https://learn.microsoft.com/en-us/azure/cosmos-db/introduction
[49] https://azure.microsoft.com/en-us/products/azure-sql/database
[50] https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id
[51] https://learn.microsoft.com/en-us/azure/notification-hubs/notification-hubs-push-notification-overview
[52] https://learn.microsoft.com/en-us/azure/api-management/api-management-key-concepts
[53] https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction

Table 2.2: Feature Matrix: MBaaS Providers Comparison

Feature	Firebase	Supabase	Parse	Back4App	AWS Amplify	Azure
Auth	✓	✓	✓	✓	✓	✓
Real-time Sync	✓	✓	✓	✓	✓	✓
NoSQL DB	Firestore, Realtime DB	✗	MongoDB	MongoDB	DynamoDB	Cosmos DB
SQL DB	Data Connect	PostgreSQL	PostgreSQL	PostgreSQL	Aurora	Azure SQL
Cloud Functions	✓	✓	✓	✓	✓	✓
Push Notifications	✓	✗	✓	✓	✓	✓
APIs	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓
Offline Support	✓	✓	✓	✓	✓	✓
Analytics	✓	✗	✗	✓	✓	✗
DB Triggers	✓	✓	✓	✓	✓	✓
Security Rules	✓	✓	✓	✓	✓	✓
Free Tier	Yes	Yes	Yes	Yes	Yes	Yes
Pricing Model	Pay-as-you-go	Tiered	Self-host costs vary	Tiered	Pay-as-you-go per service	Pay-as-you-go per service
Observability	Native	3rd Party)	External (self-hosted)	Limited (basic metrics)	Native	Native
Remote Config (A/B testing)	Native	3rd Party	3rd Party	3rd Party	Native	Native
Data Location	Multi-EU	EU	Self-host	EU/US	Multi-EU	Multi-EU
Data Location	Multi-Region (EU)	Single Region (EU)	Any (Self-Host)	Single/Multi (EU/US)	Multi-Region (EU)	Multi-Region (EU)
GDPR Status	Certified	DPA	User Respons.	DPA	Certified	Certified
Open Source	✗	✓	✓	Partial	Partial	✗
Self-Hosting	✗	✓	✓	✗	✗	✗
Vendor Lock-in	High	Low	Low	Low	High	High

2.4 Existing Solutions

To understand the existing solutions available in the Portuguese casual basketball market, it was conducted a search for applications and platforms addressing the challenges identified. This section presents the research methodology and the identified applications.

2.4.1 Search Methodology

By searching the browser using the query "Play basketball in Portugal", the useful links found for someone who just want to go to a public court and play some casual games, or shoot some balls, where from a website called "Courts of the World"[54], "Meetup"[55], a question in Reddit from 2021[56], a blog called "World Baller"[57], and some websites for clubs and schools. With the query "Basketball pickup games Portugal", the results were similar, with the AI from the browser telling to look for games in Facebook groups, in Meetup, and suggested 3 courts in Lisbon.

[54]https://www.courtsoftheworld.com/
[55]https://www.meetup.com/find/portugal/
[56]https://www.reddit.com/r/lisboa/comments/qtokhf/anywhere_to_play_pickup_basketball_in_lisbon/
[57]https://www.worldballer.com/europe/lisbon

Using Perplexity AI, using this prompt in a deep research "I am from Lisbon and I play basketball. I would like to go play on the weekend in a public court. Where can I go? Is there any platform I can find some casual games to join?"[58], the model provided 4 public courts with a description and then information about a group in Meetup, and a Facebook group. As for platforms, it indicated an application that is not available in Portugal, called "Pickup: Connect & Play Sports", another called "Pickup Sports (Adult League)"[59]. Then it also shows "Courts of the World", that has an iOS application, and another application called "Fullcourt: Pickup Basketball"[60].

Using ChatGPT, with the same prompt, with the free version and the option of *Web Search* activated, it answered with 8 public courts, with the Meetup group already mention above, as well as the Facebook group, and the blog "World Baller"[61].

Another query made in the browser to explore ways for players to play casual basketball was "*reservar campos de basquetebol portugal*" [62], where it shows some private places to book, and a platform called "AirCourts"[63], that gathers different courts from different sports to help users book them.

In another search, "*Torneios amadores portugal basketball*"[64], only one result from the first page would be useful for someone who wants to play casual basketball games, the website "Jogabasket"[65].

Going to the Google's Play Store, and searching for "Pickup" applications, several appeared, which Perplexity AI already mentioned, as well as, "Pickup: Play & Host Sports"[66,67], "GoodRec"[68] (it is not available in Portugal), "WOOOBA"[69] and "Pick-Roll"[70].

2.4.2 Identified Solutions

It was found three categories of existing platforms relevant to casual basketball in Portugal: dedicated court discovery platforms, game organization platforms and general community platforms.

Court Discovery Platforms

Courts of the World. Is a FIBA endorsed platform, available in web and in iOS, where users can find courts around the world. The platform enables users to locate basketball courts nearby, see the most popular ones, see court details and weather conditions, players can edit the

[58]https://www.perplexity.ai/search/i-am-from-lisbon-and-i-play-ba-8VmBq1GuQFGTna4LnR3VXg#0
[59]https://pickup-sports.com/
[60]https://play.google.com/store/apps/details?id=com.fullcourt.fullcourt&hl=en_US
[61]https://chatgpt.com/s/t_68ef91bd97b881918a2f64248073502a
[62]book basketball courts portugal
[63]https://www.aircourts.com/
[64]Amateur tournaments Portugal basketball
[65]http://www.jogabasket.pt/
[66]https://www.pickupgames.app/
[67]https://play.google.com/store/apps/details?id=com.testpickup.app
[68]https://play.google.com/store/apps/details?id=com.lewisblack.JustPlay
[69]https://play.google.com/store/apps/details?id=io.woooba
[70]https://play.google.com/store/apps/details?id=com.iperskill.pickroll

court, added to their favourites or check-in, in order to register their present in the court. It has a leaderbord for the top user that contribute for the platform, as they add courts, photos and comments. While testing, the technical implementation of location-based services showed unstable, often failing to load map data or local results, in Lisbon and Porto. The platform also lacks user adherence, resulting in severe data obsolescence, for example, many popular courts show no activity (check-ins or media updates) for several years. This highlights a critical failure in their engagement as the system struggles to maintain the active user base necessary to keep data real-time and relevant.

Jogabasket. From the part of the Portuguese Basketball Federation, this website has the same purpose as the Courts of the World, with the addition of finding clubs. The user can find some courts around Portugal with images, using a map that works fine. There is also a section to find 3x3 tournaments, but, at the time of writing, there is nothing there. This website is good for users to find courts in Portugal.

Game Organization Platforms

Pickup Sports (Adult League). Pickup Sports is a cross-platform mobile application where users can find leagues of more than 15 sports, including basketball. The platform allows users to join competitive leagues or participate in casual games with friends. The application employs a location-based discovery mechanism and implements a monetization model where game organizers earn 75% of fees collected from participants, with the remainder retained by the platform. In Portugal, the application exhibited limited functionality and engagement. The location-based search mechanism did not return results for Portuguese cities, and no basketball games were available. User activity data showed minimal player presence and only in Los Angeles. The application has more than 1,000 downloads in Play Store and less than 5,000 (last check in December 23 2025).

Fullcourt: Pickup Basketball. Fullcourt is a platform designed to organize and manage pickup basketball games. The application captures user profiles through physical attributes (height, weight), playing characteristics (position, skill level, and frequency), enabling users to match suitable players. The core functionality includes game creation and invitation management. The platform provides real-time visibility of game participants, their positions, attendance status, and includes in-game communication features. A courts map displays active player concentration and recurring weekly games ("Runs") at specific locations. User engagement in Portugal is limited, with 10 registered users and a single mapped court location in Lisbon. The application has more than 10,000 downloads in Play Store and less than 50,000 (last check in December 23 2025).

Pickup: Play & Host Sports. Pickup is a multi-sport platform that enables users to host and join games. User profiles are established through sport selection and skill level specification. The platform provides game discovery through search functionality (by sport and location) and game code entry, supplemented by a nearby games dashboard. Game creation allows organizers to configure sport type, skill level, court type (public/private, indoor/outdoor), game format

(individual or team-based), duration, player capacity, and visibility settings (public or private). The platform includes a community feature enabling group formation, public communication, and member-organized games. In the Portugal, the search and nearby games features returned no results, indicating absent game listings. Additionally, newly created games did not appear in the application's discovery interface, suggesting potential functional issues or data synchronization problems. The platform maintains active communities in other regions (such as Hong Kong with 16 communities), but lacks established community presence in Portugal. The application has more than 500 downloads in Play Store and less than 1,000 (last check in December 23 2025).

WOOOBA. WOOOBA is a basketball platform that facilitates game creation and participation. User profiles capture position preference, referee availability, age, and skill level, with configurable profile visibility settings. The platform supports both free and paid games, with the application retaining a commission on paid game transactions. In Portugal, the platform showed no available games in Lisbon and Porto, indicating minimal user adoption and game availability in the region. The application has more than 100 downloads in Play Store and less than 500 (last check in December 23 2025).

Pick-Roll. Pick-Roll is an Italian basketball platform centred on community engagement and competitive ranking. The application implements a seasonal ranking system with leaderboard resets every three months and associated seasonal prizes. User accounts support two roles: individual players or team/minor organization owners. Core features include court discovery through location-based mapping, game and event management (creation and participation), player check-in functionality to broadcast court presence, and peer rating systems. The platform incorporates gamification elements including reputation points ("Fame") earned through fair play, achievement-based medals for platform activities, and customizable player profiles featuring preferred equipment brands, interests, and team affiliations. Seasonal participation requires a subscription fee after an initial two-month free trial period. Game creation offers two modes: standard and ranked games, with ranked participation restricted to active season members. In Portugal, the application shows no signs of use. The application has 50,000 downloads in the Play Store an less than 100,000 (last check in December 23 2025).

General Community Platforms

Meetup. Meetup is a social media platform not only for sports, but where users can find groups for any type of activity, and, inside the group, the host can create events. There is a group called "Lisbon Basketball Club"[71] with 1942 members (last check in December 23 2025), that run indoor games every Monday and Sunday, paying a fee to book the court.

2.4.3 Analysis of Existing Solutions

Portuguese basketball players seeking to organize games currently rely on informal coordination methods or existing mobile platforms. Informal approaches include social media groups and in-person coordination at courts, alternatively, players can turn to dedicated applications designed specifically for organizing sports games. However, for these platforms to function,

[71] https://www.meetup.com/lisbon-basketball-meetup/

including this project in development, they rely on user density, and a player installing one of these applications expects to discover available games organized by other users and to successfully organise games that attract sufficient participants, and if no users are active, the application is going to be deleted. The analysis of existing platforms reveals a consistent pattern which show a minimal to absent user presence in Portugal. Besides that, the majority of courts in Lisbon or Porto (only tested places) does not appear. Another weakness found in the dedicated applications is that some core functionalities does not work properly, like the search mechanisms is inconsistent, and newly created games does not show where it should.

These identified weaknesses in the other platforms translate into design opportunities for this project, such as, adopting a Portugal focus development having Portuguese users evaluating the iterations during development, and mapping as much courts as possible, adding a feature for users to contribute in the process. In order to acquire users, a progressive, phased approach will be carried out, starting by focusing only in one region, having personal demonstrations in high peak hours in different courts in Lisbon, talking in groups and invest in tournaments in different courts where players are required to download the platform. Besides that, the application development will prioritize the core features reliability and user experience, to encourage players to stay active. The competitor platforms have features that can be use in this project to user retention, such as leaderboards, leagues and user achievements.

In the Table 2.3 is a summary of the platforms encounter and their limitations in general and in Portugal.

Table 2.3: Summary of existing platforms for casual basketball in Portugal

Platform	**Primary purpose**	**Overall limitations**	**Limitations in Portugal**
Courts of the World	Global court finder to locate indoor and outdoor basketball courts.	Focuses only on court discovery; depends on user-generated updates, which can become outdated.	Very low visible adoption; many courts lack recent activity and local map loading has been unreliable.
Jogabasket	Federation platform to list clubs, outdoor courts, and 3x3 tournaments in Portugal.	Informational website without user accounts, social features, or game organisation.	Does not support organising casual games or monitoring court activity in real time.
Pickup Sports (Adult League)	Multi-sport app for running leagues and organised games.	Strong focus on structured, often monetised leagues; requires active organisers and communities.	Search returns no basketball leagues or games in Portuguese cities, leaving the app effectively empty.
Fullcourt	Basketball app for finding courts and organising pickup runs.	Needs critical mass of players to keep maps, live traffic, and games meaningful.	Only a handful of users and one mapped court in Lisbon, so players cannot reliably find games.
Pickup: Play & Host Sports	Multi-sport platform to host and join casual games and build communities.	Test games created did not appear in discovery, suggesting technical issues.	No games show up in Portugal suggesting lack of users.
WOOOBA	Basketball app to organise pickup games and manage payments and stats.	Oriented to more structured paid games; value collapses without active organisers.	No games found in Lisbon or Porto, indicating negligible local adoption.
Pick-Roll	Community app for finding courts, organising games, and competing in seasonal rankings.	Ecosystem and community concentrated mainly in Italy; subscription for full ranked play.	No mapped use or localisation for Portugal, so courts and communities are essentially absent.
Meetup	General-purpose platform for groups and events, including sports.	Not specialised for basketball or court data; relies entirely on group organisers.	Only one major group serves Lisbon, limiting variety of locations and formats.

CHAPTER 3

Methodology

Developing a platform to help basketball players organise games and enhance their satisfaction at public courts requires a development process that consistently prioritizes user needs. To achieve this, the project adopts User-Centred Design (UCD), ensuring that users are involved throughout the design process integrated with Agile. The UCD is applied across three phases, each culminating in direct user evaluation. This iterative phases are complemented by Scrum, in the implementation phases, where it promotes adaptive planning and continuous improvement, and Acceptance Test-Driven Development (ATDD), ensuring that the system's features align with user expectations and perform as intended. This chapter focus on explaining how this frameworks are used to develop the project, giving a plan for the implementation, as well as a provisional sprint plan. In the end it is also discussed the architecture that will be implemented as well as the technologies that will be used.

3.1 Iterative Development and Project Plan

This project adopts an iterative and incremental development model structured into three major phases, all guided by UCD principals. Phase 1 emphasizes user research and prototyping, whilee Phases 2 and 3 apply Scrum sprints to iteratively build and evaluate functional features. In this way, the project integrates UCD's research-design-evaluation cycle with Scrum's sprint-based work flow. Figure 3.1 illustrates the time line of the three phases.

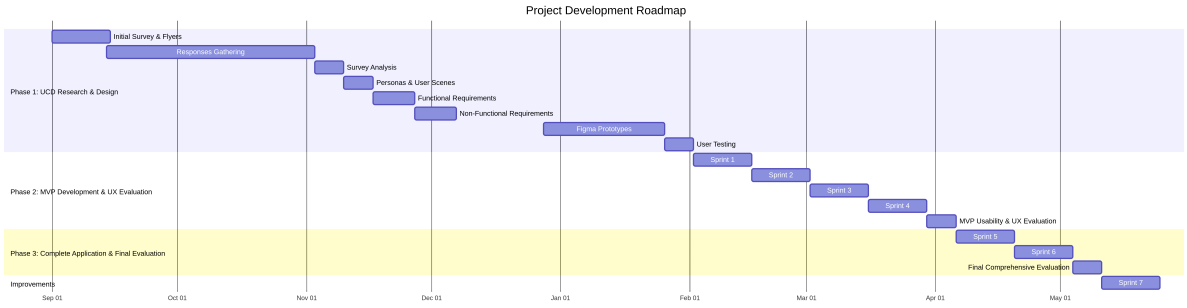

Figure 3.1: Project Development Roadmap

3.1.1 Phase 1: Prototype and Feature Validation

The first phase includes all four steps of UCD (2.2) in order to research the problem, design a proposed solution and evaluate with the possible users. These are the steps that are carried out:

A survey is conducted to understand the basketball players who uses public courts, including their habits, challenges and desired improvements to their experience. With that, *personas* are created representing different user needs and pain points. User scenarios are also developed for different contexts , describing how they use public courts, what problems they encounter and when and how a platform would support them. Through the *personas* and scenarios, requirements are gathered to respond to their needs, documenting them as User Stories (see 3.1.4) and added to the Jira project Backlog. The prototypes of User Stories are created in Figma to provide an interactive and testable design. Finally, users test the interactive prototype and provide feedback on the importance and relevance of the features. This feedback is used to update and prioritize the backlog for the next iterations. In this first phase, the tests that users will perform are Task-based usability testing using interactive Figma prototype, followed by a Feature Prioritization Survey assessing feature importance and relevance;

3.1.2 Phase 2: MVP Development and UI/UX Evaluation

In the second phase, instead of a prototype, a Minimum Viable Product (MVP) is developed, with the main features completed enough that can be usable in early stages and receive feedback. This phase starts applying both Scrum and ATDD methodologies (see 3.1.4 to implement the most critical user stories identified from the first phase's feedback. Each user story follows the ATDD approach, where acceptance criteria are defined, and tests are written to validate these criteria, the feature is developed and then validated ensuring all tests pass. User stories are organized into sprints according to the Scrum framework, enabling iterative and continuous progress, tracking on Jira. With a functional MVP, users evaluate the platform focusing on user experience (UX) and user interface (UI) aspects, through Task-Based Usability Testing to measure task performance, System Usability Scale (SUS) questionnaire combined with think-aloud protocol during task execution, and User Experience questionnaire, providing feedback to guide refinements in the final phase.

The 3.2 repreesents the 5 Epics created that contain the User Stories, as well as the propose sprint planning, where from the Sprint 1 to 4 is the Phase 2 following a break to test, and sprint 5 and 6 represent the Phase 3 (the attribution of the User Stories to each sprint can be adjusted during its development, depending on the its progress). The Epic prioritization follows a dependency-driven and value-driven approach. For this second phase it is more focuses on delivering the minimum feature set required to validate the core value proposition, enabling players to discover courts and organize games.

User Profile and Settings (PLY-9) is developed first in Sprint 1, as authentication and profile management are foundational requirements for all subsequent features. *Court Availability (PLY-6)* is also initiated in Sprint 1, establishing the court discovery infrastructure that is fundamental to game creation. Sprints 2 implements *Game Management (PLY-5)*, which represents the

application's primary value, by creating, browsing, joining, and managing basketball games. This sequence ensures that users can complete the core user journey from registration through game participation, enabling meaningful usability evaluation. In Sprint 3 and 4 is developed features from the different Epics that are not necessary for the main functionality but still adds important value for the application, including adding team and ranking system from the Epic *Ranking and Team System (PLY-8)*.

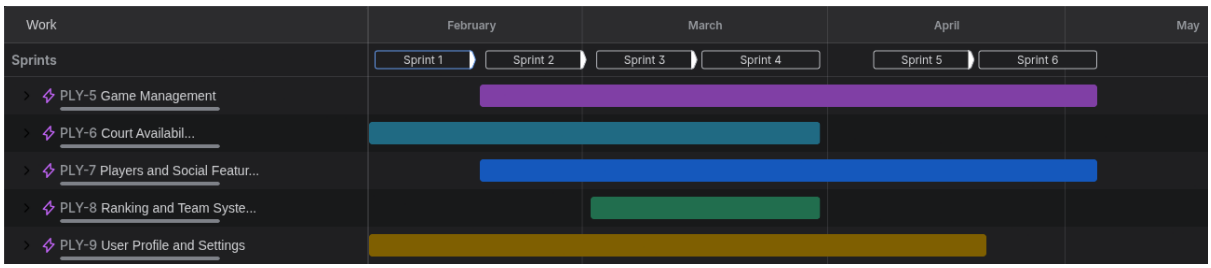

Figure 3.2: Epics and Sprint planning in Jira

3.1.3 Phase 3: Complete Application and Overall Evaluation

Following feedback from the second phase, in the final one it is developed the complete platform, continuing to apply Scrum and ATDD. Improvements from previous feedback are made and all remaining user stories are implemented, developing as well the tests first, ensuring consistent quality throughout. Users conduct a comprehensive evaluation of the complete platform, assessing overall functionality, usability, reliability and satisfaction with the final product, using SUS, task-based testing, and semi-structured interviews.

Continuing the analysis of the Figure 3.2, this phase stars in Sprint 5 and it adds value to the *Game Management (PLY-5)* Epic and a chat function from the Epic *Players and Social Features (PLY-7)*. It will also improve features and bugs detected in the testing in Phase 2, so the Sprints will likely change.

3.1.4 User Stories, Sprints and ATDD

Requirements gathered during the second phase of UCD (Subsection 2.2) are documented as User Stories and integrated into Scrum's Sprint structure. User Stories employ non-technical language written from the user's perspective, clarifying not only what is being built, but also the rationale and value delivered[42]. Normally, each User Story belongs to an Epic as milestones, representing a broader thematic organization, enabling progress tracking against major features. Each story is assigned to a Sprint and marked as complete when users can successfully execute the outlined task, aligning with the project's emphasis on User-Centred Design throughout development. The User Story will be moved through different stages in a board, starting in the To-Do column, moved to the in progress, then testing where is checked if it passes the acceptance criteria initially defined, ending up in the done. Jira[1] serves as the project management tool, enabling systematic planning, task assignment, and progress tracking throughout the development phase.

[1]https://www.atlassian.com/software/jira

ATDD complements the Scrum framework by ensuring that acceptance tests are written for each User Story before implementation[43]. This ensures requirements are precisely defined and verifiable, with tests serving as executable specifications of expected behaviour. The ATDD workflow follows three steps, starting by writing the acceptance tests to define expected behaviour, the feature is then implemented to satisfy these tests, and finally, functionality is validated to ensure all tests pass.This project employs rule-oriented acceptance criteria using a checklist format[44], which integrates naturally with the User Story structure in Jira. It could also be written in a scenario-oriented way, using a Given/When/Then template.

3.2 Architecture

For this project, the Android Developers "Guide to app architecture"[45] is followed, which recommends an architecture, with at least two layers, the UI Layer and the Data Layer, and an optional the Domain Layer. In this structure, the UI layer focuses on presentation concerns while the business logic and data management are encapsulated in the data and, when present, domain layers, clearly separating responsibilities across the application. Besides that, the project will follow the Model-View-Model (MVVM) pattern, where there is a clear separation between the logic and the UI, enabling a better development, testing and a clean code base.[46]

The Figure 3.3 represents the project's architecture and some technologies that will be used, decided after the gathering of requirements. In the Presentation Layer there is the UI components, that will be developed using Compose Multiplatform, and the ViewModel, classes that hold states of the application and exposes them to the View. The ViewModel will call the Use Cases, in the Domain Layer which use entities and the repositories interfaces to connect with the data layer. Here will be use Kotlin Multiplatform taking advantage of the native capabilities while working in both Android and iOS.

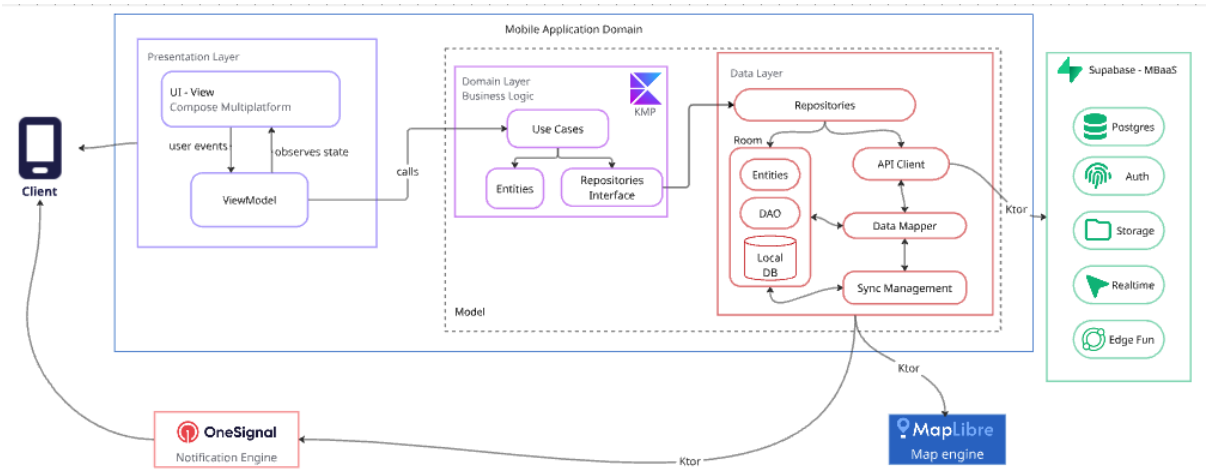

Figure 3.3: Architecture Diagram of the Multiplatform Application

In the Data layer, there are the repositories, as the guide mentions[45] which will work with a local and an external data source. The local data source will be Room[47] that is part of Android Jetpack suite, making a good option for this project, being an abstraction of SQLite database, that integrates naturally with flows and coroutines, providing a annotation-based API.

Compare to SQLDelight, a considerable option, it has slight more boilerplate with entities and DAOs, and less control over raw SQL, making debugging complex queries harder.

For the external data source it will be use the Supabase, an open source MBaaS with several services available that will help with the project, including PostgreSQL Database that can handle complex relationship connections while offering built-in real-time features for live updates. Besides that, it has authentication, storage, and server-side functions, however, lacks native push notifications and mobile analytic, which forces third-party tools integrations. Supabase can be self-hosted or use theirs services that has different pricing options, with the free tier being enough for this project, as it has a limit of 50,000 monthly active users. Comparing to others, it is not a "black box" being easy to debug and has less vendor lock-in, with a PostgresSQL database that can be run anywhere and can be exported.

Using Ktor to communicate with Supabase as it is standard and recommend HTTP client, with multiple examples from Google, Jetbrains and other communities using Ktor as example in Kotlin Multiplatform projects. Another good option would be the Supabase Kotlin Client, however, this would make the application dependable on Supabase and if later the MBaaS was to be replaced, it would be more complex to migrate. With this, a little more code needs to be developed with Ktor, than Supabase Kotlin Client, but ideal to not get vendor lock-in. As some feature of the platform will need to work while the user is off-line or with bad connection, the application will have a synchronization manager that will check what needs to be updated in the local database and resolve conflicts. Finally a data mapper to convert data from Supabase to the Room database is needed.

3.2.1 Maps Engine

An important piece of this project is to have a map identifying the courts for the users to find and visualize. There are several tools and libraries for this, however the most compatible and cheap option is MapLibre Compose[2]. This is a Compose Multiplatform wrapper around the MapLibre[3] SDKs, an organization that provides open-source mapping libraries, which enables to render interactive maps. The MapLibre Compose will work in both Android and iOS, and it is possible to configure gestures, respond to a map click or long click, download off-line images and more other features.

A disadvantage of this library is that the community and support is still growing. An alternative would be Google Maps SDK[4], however it has vendor lock-in, with costs passing the 300 USD (US Dollar) trial credits. Besides that, it is not Kotlin Multiplatform native, needing a platform-specific implementation. Mapbox[5] would be another option, with a free tier and then priced, with no direct support to KMP.

MapLibre Compose is used as a Presentation Layer component, acting as the UI rendering engine responsible for visualizing geospatial data exposed by the ViewModel. It is decoupled from

[2]https://maplibre.org/maplibre-compose/
[3]https://maplibre.org/
[4]https://developers.google.com/maps/documentation/android-sdk/overview?section=start
[5]https://www.mapbox.com/

business logic and data access, ensuring that domain rules related to court management remain independent of the specific mapping technology. In contrast, the MapLibre SDK integrates with the Data Layer, handling the retrieval of map tiles from external MapLibre servers when users pan, zoom, or change location. It also implements caching strategies to store map assets locally for off-line use and manages map styling by fetching and applying the MapLibre style JSON, which defines layers, colours, and other visual properties.

3.2.2 Notifications Engine

To keep user's engagement, it is essential to have notifications, when they are in the application and when they have the application closed, so they get notified when someone has created a game in their favourite court. Supabase have a real-time feature that enables to create in-app notifications, however, when the application is closed there it is no communication between the Supabase back end and the client side. For push notifications, when the application is closed, there are different options. Google's Firebase has the Firebase Cloud Messaging (FCM)[6] that can notify the application about available synchronizations, and send notification messages to single devices, to groups of devices, or to devices subscribed to topics, all this with no costs. Another option is OneSignal[7] that has multiple functionalities including sending notifications when the application is closed. OneSignal has some pricing options, but in the free tier, the push notifications are unlimited. For this project, OneSignal will be used, as the set-up is easier, as it handles user's tokens while in Firebase, it needs more code to handle that part.

OneSignal integrates with the Data Layer, which handles push notification functionality. When a user creates a game, for example, the application makes an API call to OneSignal to send a push notification to users. By isolating notification logic in the Data Layer, the architecture remains agnostic to the specific notification provider, allowing future migration to alternative services without impacting business logic.

[6]https://firebase.google.com/docs/cloud-messaging
[7]https://onesignal.com/

CHAPTER 4

Phase 1 - Prototype and Feature Validation

In the first phase of the project, the main goal it to design the application ending up with a prototype and the users evaluation, to validate the features and their importance. In this chapter it will be describe the implementation of this research and prototyping phase, starting by understanding and specifying the context of use, and then gathering the requirements.

4.1 Understand and specify the context of use

The first part that is important to design the platform is to understand who is the target user and how will the application be used. For this, a user research was conducted through a survey targeting basketball players in Portugal, then two persona where made based on the survey and then user scenarios.

4.1.1 User research

A user research was conducted through a survey in order to understand who goes to public basketball courts, how often they go, why they are not going more often, what gets them annoyed and what would they wish to improve in their experience playing in basketball courts. To collect data, it was used Mircrosoft Forms (see questions in Appendix A) and was shared through *Whatsapp*, between different people and groups and a flyer was made and distributed in a public basketball court in Lisbon (see flyer in Appendix B). A total of 50 responses were gather and here are some takeaways:

- 80% of the responses were from men;
- 27 responses (54%) are from people between 18-25 years old, and 16 (32%) from 26-35 years old;
- The majority live in Lisbon, with 10 (20%) responses from Setúbal, 2 (4%) from Aveiro and 1 (2%) from Porto;
- 27 (46%)people that responded work full time and 21 (36%) are studying in university;

- When asked if they play basketball, 34 (40%) said that they play in public courts causal games, 21 (25%) go to private courts, 15 (18%) just shoot some balls and 6 (7%) play in a competitive team;
- 39 (78%) started playing before they were 16 years old;
- 15 participants go to play in Alcântara (24%) public court and 14 (22%) in Moinho do Penedo in Monsato public court, both in Lisbon;
- When asked the Frequency that they go to public courts, 18 (35%) said that go once a week and 14 (27%) responded with seasonal, with 88% wanting to go more often;
- 24 (24%) do not go more often because of time constraints and 22 (22%) said that they do not known enough people to play with, besides that, 10 (10%) said that nobody is in the court when they go;
- Usually, 30 (27%) participants go with friends to play against each other, 33 (29%) go with friends to play against others, 19 (17%) go alone to shoot some balls, but are open to play some games if it comes up, and 14 (13%) just want to go shoot some balls;
- 58% want more competition, while the other 42% are satisfied;
- When asked to the participants what they find more annoying in the public courts, most complain about the conditions of the court, and other about the players, that some are unpleasant to play with, the level of competitiveness is not match, and lack of organization;

4.1.2 Persona

A Persona is a fictional character created based on the user research that represents the different user types. Personas help to understand the user needs, guiding the next phases of the development process. For this project, 2 types of users were found, creating 2 different Personas.

Persona 1: Miguel — The Competitive Player

Persona - There are basketball players that want to play against other players, have more competition, as seen in the survey with 58%, and as 21 participants do not play more often because they do not know enough people, and 10 complain that nobody is there when they go, it can be assume that they want to find other players to play with. Out of this analysis, the first persona emerges, Miguel the competitive player, that wants to play competitive games more often, the Fig4.1 represents that persona.

Miguel usually goes with friends to play against other players that they find on the court, however they never know if a good amount competitors will be there, at the same time as them, maybe there is someone, but can be there for a long time already and will leave soon, or does not want to play with others, and there is also the possibility the players encounter are not as good as Miguel and is not that fun to play with. Because of this, Miguel does not go more often, as it can be a waste of time. So, he needs a platform to meet other players that want to play a competitive game, or to just see how many players are in the court to play a spontaneous game. Miguel also wants to know if the players are the right match for him, if they play in the same level and if they will dedicate the same as him.

Below are two different scenarios that can happen to players like Miguel.

Figure 4.1: Persona 1 Miguel

Scenario 1: Miguel plays on Sunday Miguel usually goes on Sunday mornings to a court in Alcântara with two friends to play pickup games against other players who might be there. To play a 3v3 game, they need at least three more players. For a 4v4, they need five more, joining someone to complete their team. Typically, Sunday mornings bring older players to the court, and Miguel and his friends often end up joining their games. However, this is not the type of competition they want, so sometimes they try to go at a different time to find players with similar skill levels and physical condition. Occasionally, they encounter another group of friends around their age and level and challenge them to a game. Sometimes they find competitive teams, but other times they end up with players who are just fooling around or with team-mates who do not pass the ball, leaving Miguel and his friends frustrated.

Scenario 2: Miguel plays after work on Summer - During Summer, Miguel likes to play after work. But when he and his friends arrive at the court, no one is there, so they just shoot some hoops and don't have as much fun or get as much running as they hoped. When they are about to leave, a pair of friends sometimes arrives, also hoping to find other players to challenge. However, by then Miguel and his friends are almost ready to go. If they had organized in advance or somehow known that others were coming, they might have adjusted their schedule and been able to play together.

How the platform would help - With a platform, Miguel would proceed as follows: He opens the application and creates a game, a 4v4, set for Saturday at 3 p.m., and invites his two friends, who accept. Miguel now needs five more players, but if he can't find a fifth, he can reduce the game to a 3v3. He makes the game public, so anyone interested in playing a 3v3 or 4v4 receives a notification, or they see it in the list of upcoming games in the application. The first rival joins after seeing Miguel team's ranking. Soon, a team of four expresses intent to join, and so, Miguel invites the first adversary to his team and accepts the other team as opponents. On Saturday afternoon, the group arrives at the available court and checks in via

the application. Other people now know there are eight players present, and the game appears in the application as happening live at that location. Miguel's team wins, and a member from each team records the result in the application. Because the game is ranked, every member of the winning team earns points toward their individual rankings. Miguel enjoyed playing with their fourth team-mate, so they decide to register the team and enter the league table for teams of four. The next time Miguel wants to play, he can create a game with his new team.

Persona 2: Carlos — The Flexible Shooter

Persona -The other type of user, is the player that just goes to the court to practice and shoot some balls, as 29 players out of 50 usually go alone for that, some do not mind playing a game, this type of user wants to find a court or time slot to relax, without competition and space practice. In the Fig 4.2 is the second persona, Carlos the flexible shooter, he goes once a week to a public court to shoot around, and if someone asks him to play, he accepts the invitation. Some times the court is full and Carlos does not have space to shoot and everyone is playing too aggressive. He needs a platform to see which courts have space to just shoot practice.

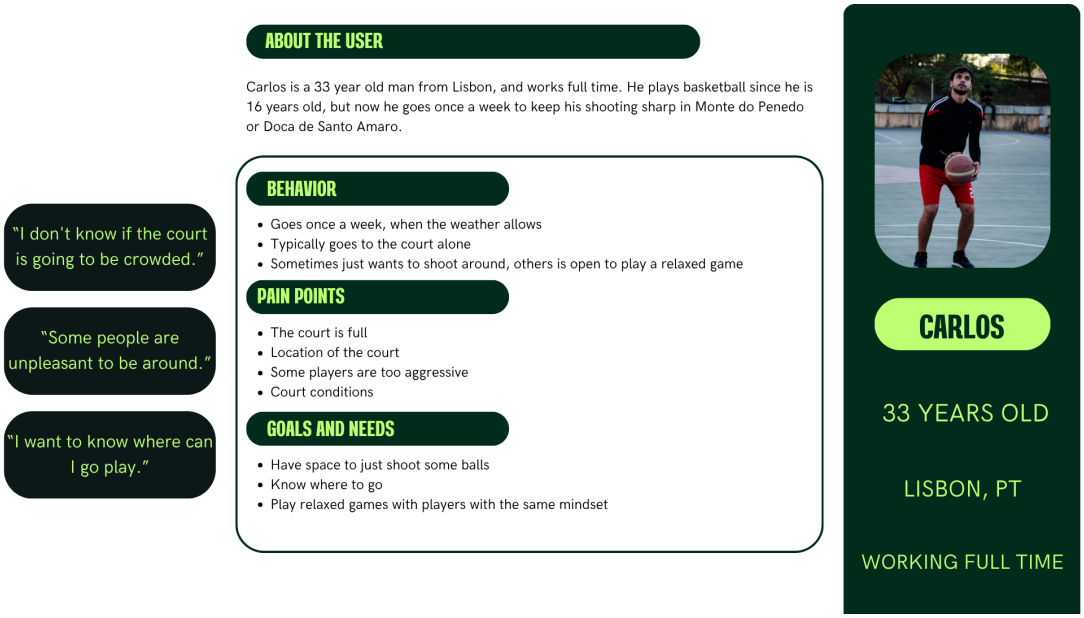

Figure 4.2: Persona 1 Carlos

Scenario: Carlos go practice Saturday morning - These days, Carlos enjoys going to the court to practice his shot. He goes at least once a week, and when other players invite him to join a pickup game, he's always ready for the challenge. Carlos prefers to play on Saturday mornings at Monte do Penedo in Monsanto. However, especially when the weather is good, the court can become crowded, with several games going on at once and multiple people shooting at the same basket. During these times, Carlos often wishes he had gone to a different court.

How the platform would help - With a platform, Carlos could check court availability before leaving home, allowing him to choose a less crowded court. He could also discover courts he did not know about and read reviews from other players about court conditions. Before heading out, Carlos could use the application to indicate his intention to go to a specific court at a certain time, specifying that he plans to shoot around but is open to joining non-competitive

games. Alternatively, he could check in when he arrives, letting others know he's available for practice or casual play.

4.2 Specifying the user requirements

4.2.1 Epics and User Stories

From the personas and the user scenarios requirements where gather. The User Stories were organised into five groups or, as called in Scrum, Epics:

- **PLY-05 Game Management** (Table 4.1) encompasses all features related to creating, discovering, joining, and managing basketball games.
- **PLY-06 Court Availability** (Table 4.2) handles court discovery, management, occupancy tracking, and user check-ins.
- **PLY-07 Players and Social Features** (Table 4.3) encompasses player discovery, friendship management, communication, and player interactions.
- **PLY-08 Ranking and Team System** (Table 4.4) manages competitive ranking, team creation, and team-based competitions.
- **PLY-09 User Profile and Settings** (Table 4.5) handles authentication, profile management, notification preferences, and user data.

Table 4.1: Epic: Game Management - User Stories

PLY-05 Game Management		
ID	**Title**	**User Story**
PLY-10	Create a Game	"As Miguel, I want to create a game with custom parameters, so that I can organize matches with specific settings and invite players."
PLY-41	View Game Details	"As Miguel, I want to view complete details of a game I'm in or interested in, so that I know when to show up, who's playing, and what to expect."
PLY-11	Receive Game Notifications	"As Miguel, I want to receive notifications when games are created in my favourite courts, so that I can quickly join games that interest me."
PLY-12	Respond to Game Invitation	"As a basketball player, I want to accept or decline game invitations, so that I can control which games I participate in."
PLY-13	Browse and Search Games	"As Miguel, I want to browse and filter available games, so that I can find games that match my preferences."
PLY-14	Request to Join Game	"As Miguel, I want to request to join public games, so that I can participate in games organized by others."
PLY-15	Manage Join Requests and Players	"As Miguel and game host, I want to accept or reject players requesting to join my game and manage players in game list, so that I can control who participates."
PLY-17	Start Game	"As Miguel and game host, I want to start the game when all players arrive, so that we can begin playing and tracking the match."
PLY-18	End Game	"As Miguel, I want to end the match and record the final score and winning team, so that results are tracked for rankings and history."
PLY-16	Challenge Teams	"As Miguel and a team member, I want to challenge other teams to a game, so that we can compete against organized opponents."
PLY-19	Spectator Mode	"As Carlos, I want to mark myself as watching a game and record the result, so that I can contribute to game accuracy even if I'm not playing."
PLY-25	Spontaneous Game Creation	"As Carlos, while I am in the court, I want to quickly create an instant game, so that me and other player in the court can play without detailed planning, but still counting in the app."
PLY-24	Tournament Creation	"As Miguel, I want to create tournaments with multiple teams and rounds, so that I can run structured competitive events and include different players and teams."

Table 4.2: Epic: Court Availability - User Stories

PLY-06 The Court Availability		
ID	**Title**	**User Story**
PLY-40	View Court Details and Add to Favourites	"As Miguel, I want to view court details and add courts to my favorites, so that I can easily find courts I like and receive notifications about games there."
PLY-42	Search Courts	"As Carlos, I want to search for courts interactively and easily, so that I can find new courts to try out."
PLY-26	Court Check-In	"As Carlos, I want to check in when I arrive at a court, so that others know I'm there."
PLY-27	Court Occupancy Display	"As Carlos, I want to see how many players are at each court, so that I can decide where to play."
PLY-28	Add New Courts	"As Carlos, I want to add courts that aren't in the system, so that the app includes all court locations."
PLY-29	Rate and Review Courts	"As Carlos, I want to rate and review courts, so that others have accurate and updated information about court quality."
PLY-30	Announce Playing Intentions	"As Miguel, I want to announce when I plan to play at a court, so that others can join me."
PLY-31	View Expected Court Availability	"As Carlos, I want to see how many players plan to be at a court at specific times, so that I can plan my sessions better."

Table 4.3: Epic: Players and Social Features - User Stories

PLY-07 The Players and Social Features		
ID	**Title**	**User Story**
PLY-21	View Player Profiles	"As Miguel, I want to view other players' profiles, so that I can find potential teammates or opponents."
PLY-32	Send friend Requests	"As Miguel, I want to add other players as friends, so that I can easily organize games with them and know when they are at the court."
PLY-33	Respond to friend Requests	"As Miguel, I want to accept or decline friend requests, so that I can manage my friends list."
PLY-36	Search Players and Teams	"As Miguel, I want to search for other players and teams, so that I can connect with them or issue challenges."
PLY-34	Game Chat	"As Miguel, I want to chat with other players in my game, so that we can coordinate details and communicate."

Table 4.4: Epic: Ranking and Team System - User Stories

PLY-08 The Ranking and Team System		
ID	**Title**	**User Story**
PLY-20	Player Ranking System	"As Miguel, a competitive player, I want to earn points for victories and see my ranking, so that I can track my progress and compete with others."
PLY-22	Team Management	"As Miguel, I want to create and manage teams, so that I can play regularly with the same group and compete in team rankings."

Table 4.5: Epic: User Profile and Settings - User Stories

PLY-09 The User Profile and Settings		
ID	**Title**	**User Story**
PLY-37	User Authentication	"As Carlos, I want to register, log in, and recover my password, so that I can securely access my account."
PLY-38	Profile Management	"As Carlos, I want to edit my profile information, so that my details stay updated."
PLY-35	View Game History	"As Miguel, I want to view my past games, so that I can track my playing activity and review results."
PLY-39	Notification Preferences	"As Carlos, I want to customize which notifications I receive, so that I only get alerts that matter to me."

4.2.2 Non-Functional Requirements

For the non-functional requirements, they were aligned with ISO/IEC 25010[48] quality model. Each requirement specifies quality attributes and constraints that apply across multiple functional features. In the ISO there are nine quality characteristics[48]:

- Functional Suitability
- Performance Efficiency
- Compatibility
- Interaction Capability
- Reliability
- Security
- Maintainability
- Flexibility
- Safety

Below is a list of the non-functional requirements created to set guidelines to follow the International Standard guaranteeing software quality.

Table 4.6: Non-Functional Requirements: Reliability

Reliability			
ID	**Title**	**Description**	**Priority**
NFR-01	Offline Availability	The application shall support offline access to previously loaded data, informing when the user is offline and the last updated timestamp.	Must Have
NFR-02	Data Consistency	The application shall not lose data after a write operation. Any data acknowledged to the user as saved shall persist across application crashes, OS termination, or device restart	Must Have
NFR-03	Graceful Degradation	In case of a failed backend request, the application shall: (1) Display cached data when available, (2) Show a clear, user-friendly error message, (3) Never crash or display technical error details, (4) Allow retry when appropriate.	Should Have
NFR-23	Session State Persistence	The application shall preserve the user session state across application restarts using secure authentication tokens. Sessions shall persist until the user explicitly logs out or until tokens expire after a prolonged period of inactivity.	Must Have
NFR-24	Data Sync on App Resume	The application shall automatically synchronize data with the backend when resumed after being backgrounded. Data is considered fresh for 5 minutes, if more than 5 minutes have passed, a sync is triggered.	Should Have
NFR-26	Background Sync	The application may implement periodic background synchronization to keep user data up-to-date even when the app is not actively in use.	Could Have
NFR-27	Form State Recovery	The application shall recover form state when the app is backgrounded during form entry.	Should Have

Table 4.7: Non-Functional Requirements: Security

Security			
ID	**Title**	**Description**	**Priority**
NFR-04	User Authentication	The application shall authenticate users using either social login or email/password. Authentication credentials and session tokens shall be securely stored using platform-provided secure storage mechanisms.	Must Have
NFR-05	Data Privacy (GDPR Compliance)	The application shall comply with GDPR requirements: explicit user consent before data collection, data minimization, user right to access/delete data, transparent privacy policy. Location data used only for real-time search, not stored long term.	Must Have
NFR-06	Secure Data Transmission	All data exchanged between the client and backend shall be transmitted over encrypted channels using HTTPS with TLS 1.3. Invalid or self-signed certificates shall be rejected.	Must Have
NFR-07	Location Privacy	The application shall never expose to other users the user location coordinates; only the location of the court should be shared publicly.	Must Have
NFR-21	Message Encryption	The application shall implement end-to-end encryption for user messages. Message content shall be encrypted on the sender's device and decrypted only on the recipient's device using industry-standard cryptographic algorithms (e.g., AES-256).	Must Have

Table 4.8: Non-Functional Requirements: Interaction Capability

Interaction Capability			
ID	**Title**	**Description**	**Priority**
NFR-08	Learnability	First-time users shall be able to complete core tasks (finding a court, joining a game, sending a message) without prior training or documentation, supported by intuitive UI design and optional onboarding hints.	Must Have
NFR-09	Error Feedback	The application shall provide a clear, descriptive error message when a user action fails, explaining why.	Should Have
NFR-10	Outdoor Readability	The application shall remain readable in outdoor sunlight conditions through high contrast UI and support for system dark/light modes.	Could Have
NFR-25	UI State Preservation	The application shall preserve user interface state (filters, search terms, scroll positions, form inputs) when the app is backgrounded and the process is not terminated. This state shall be restored when the app is reopened during the same session.	Should Have

Table 4.9: Non-Functional Requirements: Compatibility

Compatibility			
ID	**Title**	**Description**	**Priority**
NFR-13	GPS Integration	The application shall integrate with device's native GPS/GNSS hardware for location services.	Must Have
NFR-14	Maps Service Integration	Application shall integrate with device maps services to open maps court location.	Could Have
NFR-20	Camera/Storage Integration	The system shall access native camera to take photographs and allow image upload directly from the device camera and storage.	Must Have
NFR-22	Calendar Service Integration	Application shall integrate with device calendar services to export game events.	Could Have

Table 4.10: Non-Functional Requirements: Flexibility

Flexibility			
ID	**Title**	**Description**	**Priority**
NFR-11	Cross-Platform Support	The application shall run on both Android and iOS platforms, supporting Android 8.0 (API 26) and later, and iOS 13 and later.	Must Have
NFR-12	Screen Compatibility	The application shall function correctly on devices with varying screen sizes and capabilities, for phones only.	Should Have

Table 4.11: Non-Functional Requirements: Maintainability

Maintainability			
ID	**Title**	**Description**	**Priority**
NFR-15	Code Modularity	The application shall follow clean architecture principles with separation between presentation, domain, and data layers. Changes to one module shall have minimal impact on others.	Should Have
NFR-16	Testability	The application shall have unit tests for core business logic. Acceptance tests shall be written for user stories following ATDD methodology.	Should Have
NFR-17	Documentation	The application shall be complemented with documentation for APIs and business logic.	Should Have
NFR-18	Logging	The application shall generate centralized logs for failures and errors, structured for analysis.	Should Have

Table 4.12: Non-Functional Requirements: Safety

Safety			
ID	**Title**	**Description**	**Priority**
NFR-19	User Moderation	The application shall include report features to report players and events to mitigate harassment or unsafe gatherings.	Could Have

Table 4.13: Non-Functional Requirements: Performance Efficiency

Performance Efficiency			
ID	**Title**	**Description**	**Priority**
NFR-28	UI Responsiveness	The application shall respond to user interactions (navigation, button taps, scrolling) within 500 ms under normal operating conditions.	Must Have
NFR-29	Data Load Time	Application data shall load within 2 seconds when network connectivity is available and within 1 second when using cached data.	Should Have
NFR-30	Battery Efficiency	The application shall minimize battery consumption by limiting background processing, using location services only when necessary, and adapting update frequency based on application state.	Must Have
NFR-31	Scalability	The system shall support at least 1,000 registered users and 100 concurrent active users within the Lisbon area without degradation of core functionalities.	Should Have

References

[1] *Ticha penicheiro no women's basketball hall of fame*. [Online]. Available: https://www.fpb.pt/noticia/ticha-penicheiro-no-womens-basketball-hall-of-fame-2/.

[2] *«é um bom momento para o basquetebol em portugal, queremos mais espaço» | maisfutebol*. [Online]. Available: https://maisfutebol.iol.pt/basquetebol/selecao/e-um-bom-momento-para-o-basquetebol-em-portugal-queremos-mais-espaco.

[3] I. P. do Desporto e Juventude, *Estatísticas e dados do desporto*, Excel spreadsheet (v.2025-10-27), Accessed: 2025-12-16, 2025. [Online]. Available: https://ipdj.gov.pt/estat%C3%ADsticas.

[4] M. K. Smith, *Jean lave, etienne wenger and communities of practice - infed.org*, last updated: August 8, 2025; Date accessed: January 10, 2026. [Online]. Available: https://infed.org/dir/welcome/jean-lave-etienne-wenger-and-communities-of-practice/.

[5] *Mobile operating system market share portugal | statcounter global stats*. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/portugal#mobile_os_combined-PT-%5C#yearly-2009-2025-bar.

[6] M. K. Khachouch, A. Korchi, Y. Lakhrissi, and A. Moumen, «Framework choice criteria for mobile application development», *2nd International Conference on Electrical, Communication and Computer Engineering, ICECCE 2020*, Jun. 2020. DOI: 10.1109/ICECCE49384.2020.9179434.

[7] R. Nagy, *Simplifying Application Development with Kotlin Multiplatform Mobile : Write Robust Native Applications for IOS and Android Efficiently*. 2022, p. 184, ISBN: 9781801819657.

[8] A. C. Santos, L. Tarrataca, J. M. P. Cardoso, D. R. Ferreira, P. C. Diniz, and P. Chainho, «Context inference for mobile applications in the upcase project», [Online]. Available: http://www.ist-esense.org/.

[9] S. Huber, L. Demetz, and M. Felderer, «Pwa vs the others: A comparative study on the ui energy-efficiency of progressive web apps», *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12706 LNCS, pp. 464–479, 2021, ISSN: 16113349. DOI: 10.1007/978-3-030-74296-6_35. [Online]. Available: https://www.scopus.com/pages/publications/85111163302?origin=resultslist.

[10] M. Gerges and A. Elgalb, «Comprehensive comparative analysis of mobile apps development approaches», *Journal of Artificial Intelligence General science (JAIGS) ISSN:3006-4023*, vol. 6, pp. 430–437, 1 Dec. 2024, ISSN: 3006-4023. DOI: 10.60087/JAIGS.V6I1.269. [Online]. Available: https://newjaigs.org/index.php/JAIGS/article/view/269.

[11] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, «Taxonomy of cross-platform mobile applications development approaches», *Ain Shams Engineering Journal*, vol. 8, no. 2, pp. 163–190, 2017, ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2015.08.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090447915001276.

[12] D. Zou and M. Y. Darus, *A comparative analysis of cross-platform mobile development frameworks*, 2024. DOI: 10.1109/ISCI62787.2024.10667693.

[13] V. Phalake, S. Joshi, K. Rade, and V. Phalke, *Modernized application development using optimized low code platform*, 2022. DOI: 10.1109/ASIANCON55314.2022.9908726.

[14] JetBrains, *Kotlin multiplatform – build cross-platform apps*, Accessed: Dec.28, 2025. [Online]. Available: https://www.jetbrains.com/kotlin-multiplatform/.

[15] M. Janakaraj, «Architecting shared logic with kotlin multiplatform mobile: Benefits, challenges, and ecosystem positioning», *European Journal of Computer Science and Information Technology*, vol. 13, pp. 121–141, 44 May 2025, ISSN: 20540957. DOI: 10.37745/EJCSIT.2013/VOL13N44121141.

[16] Google, *Material design 3 - google's latest open source design system*, Accessed: Dec.28, 2025. [Online]. Available: https://m3.material.io/.

[17] JetBrains, *Compose multiplatform – beautiful uis everywhere.* [Online]. Available: https://www.jetbrains.com/compose-multiplatform/.

[18] T. Carter, *Kotlin Multiplatform Projects: Sharing Code Between Android, iOS, and the Web*. Lincoln Publishers, Jul. 2025, Kotlin in Depth: Coroutines, Compose, and Multiplatform Apps for Android and Beyond.

[19] S. R. L. de Lemos Meira, E. S. D. De Almeida, M. Goulão, and C. Silva, *Agile manifesto*, 2013. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/agile-manifesto.

[20] K. Beck, M. Beedle, A. van Bennekum, *et al.*, *Manifesto for agile software development*, 2001. [Online]. Available: https://agilemanifesto.org/.

[21] A. C. Fernández, I. A. R. Flores, and A. L. P. Estrella, *Agile user-centered development of mobile applications: A systematic literature review*, 2024. DOI: 10.1109/CONISOFT63288.2024.00012.

[22] I. D. F. .-. IxDF, *What is user centered design (ucd)?*, Accessed: Nov.17, 2025, 2025. [Online]. Available: https://www.interaction-design.org/literature/topics/user-centered-design.

[23] *Iso 9241-210:2019 - ergonomics of human-system interaction — part 210: Human-centred design for interactive systems.* [Online]. Available: https://www.iso.org/standard/77520.html.

[24] Interaction Design Foundation - IxDF, *What are functional requirements?*, Accessed: Nov.19, 2025, 2025. [Online]. Available: https://www.interaction-design.org/literature/topics/functional-requirements.

[25] S. D. Carli, «Introduction to firebase», in *Build Mobile Apps with SwiftUI and Firebase*. Apress, 2023, pp. 17–42. DOI: 10.1007/978-1-4842-9452-9_2. [Online]. Available: https://learning.oreilly.com/library/view/build-mobile-apps/9781484294529/html/533376_1_En_2_Chapter.xhtml.

[26] J.-H. Kuo, H.-M. Ruan, C.-Y. Chan, and C.-L. Lei, «Investigation of mobile app behaviors, from the aspect of real world mobile backend system», Oct. 2017, pp. 1–6. DOI: 10.1109/AEECT.2017.8257762.

[27] P. R. Saraf, «A review on firebase (backend as a service) for mobile application development», *International Journal for Research in Applied Science and Engineering Technology*, vol. 10, pp. 967–971, 1 Jan. 2022. DOI: 10.22214/IJRASET.2022.39958.

[28] R. Shah, S. Jagtap, and V. Jain, «Architecting analytics-driven mobile ecosystems: Scalable backend frameworks for intelligent data flow and real-time user insights», *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, vol. 6, no. 2, pp. 83–91, Apr. 2025. DOI: 10.63282/3050-9262.IJAIDSML-V6I2P109. [Online]. Available: https://ijaidsml.org/index.php/ijaidsml/article/view/148.

[29] *Data layer*, Accessed: Dec.28, 2025. [Online]. Available: https://developer.android.com/topic/architecture/data-layer.

[30] V. R. Guda, «Implementation of offline-first architectures for android internet-based chat systems», ISSN: 2582-7138. DOI: 10.54660/.IJMRGE.2025.6.3.1200-1203. [Online]. Available: www.allmultidisciplinaryjournal.com.

[31] N. Fernando, S. W. Loke, and W. Rahayu, «Mobile cloud computing: A survey», *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013, Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2012.05.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X12001318.

[32] M.-T. Chiriac, «The advantage of nosql databases over sql databases», *Database Systems Journal*, vol. XVI, p. 2025,

[33] T. Taipalus, «Database management system performance comparisons: A systematic literature review», *Journal of Systems and Software*, vol. 208, p. 111 872, Feb. 2024, ISSN: 0164-1212. DOI: 10.1016/J.

JSS.2023.111872. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223002674.

[34] Z. Mccormick and D. C. Schmidt, «Data synchronization patterns in mobile application design, proceedings of the pattern languages of programs (plop) 2012 conference»,

[35] S. Vishvnath, *Designing a robust data synchronization system for multi-device mobile applications*, Aug. 2024. [Online]. Available: https://medium.com/@engineervishvnath/designing-a-robust-data-synchronization-system-for-multi-device-mobile-applications-c0b23e4fc0cb.

[36] Android, *Task scheduling*, Work Manager, Oct. 2025. [Online]. Available: https://developer.android.com/develop/background-work/background-tasks/persistent.

[37] A. Z. Ayezabu, «Ayezabu amanuel supabase vs firebase: Evaluation of performance and development of progressive web apps», 2022.

[38] *2025's best firebase alternatives: Open-source & scalable solutions*, Accessed: Dec.28, 2025. [Online]. Available: https://blog.back4app.com/firebase-alternatives/.

[39] *Aws amplify features*, Accessed: Dec.28, 2025. [Online]. Available: https://aws.amazon.com/amplify/features/.

[40] *Microsoft azure app service*. [Online]. Available: https://azure.microsoft.com/en-us/products/app-service.

[41] *Azure functions*, Accessed: Dec.28, 2025. [Online]. Available: https://azure.microsoft.com/en-us/products/functions.

[42] *User stories with examples and a template | atlassian*, Accessed: Nov.19, 2025. [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories.

[43] A. Alliance, *Acceptance test driven development (atdd)*, Accessed: Nov.19, 2025, 2022. [Online]. Available: https://agilealliance.org/glossary/atdd/.

[44] *Acceptance criteria for user stories in agile: Purposes, formats, examples, and best practices*, Accessed: Nov.19, 2025. [Online]. Available: https://www.altexsoft.com/blog/acceptance-criteria-purposes-formats-and-best-practices/.

[45] A. Developers, *Guide to app architecture | app architecture | android developers*, Accessed: Dec.28, 2025. [Online]. Available: https://developer.android.com/topic/architecture.

[46] M. Fuksa, S. Speth, and S. Becker, «Mvvm revisited: Exploring design variants of the model-view-viewmodel pattern», DOI: 10.5281/zenodo.13350488. [Online]. Available: https://doi.org/10.5281/zenodo.13350488.

[47] Android Developers, *Save data in a local database using room*, Android Jetpack Documentation, Oct. 2025. [Online]. Available: https://developer.android.com/training/data-storage/room.

[48] ISO/IEC, *Iso/iec 25000 - software and data quality - iso/iec 25010*, last access: 28 December 2025. [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010.

APPENDIX A

User Research Survey

Bellow is a print of the Survey conducted.

Casual Basketball Games Opinion

Another day without a basketball game? Your friends missed out again, and you've got no one else to challenge? You go to the court and have to sit on the bench and just watch? If you've ever been to the court and left thinking 'this could be better', I would love to hear from you.

Or, are you satisfied with how you've been organising your games? Then wait, don't leave yet. Share your ideas, because this survey is aimed at all basketball players and enthusiasts who have 5 minutes to share their experience and make casual basketball games more enjoyable and easier to join.

This questionnaire is part of a study developed as part of a Master's thesis in Informatics Engineering at the University of Aveiro, and aims to understand and improve the experience of casual basketball games. Your participation is voluntary and anonymous, and no personally identifiable data will be collected. The responses are intended solely for academic purposes within the scope of the Master's research. Access to individual responses is restricted to the researcher, and the results will only be presented in aggregate and anonymous form (tables/graphs).

The estimated time to complete the survey is about 5 minutes.

Thank you for your participation,Pedro Azevedo

Use and protection of data: in accordance with the General Data Protection Regulation (GDPR), you may request further clarification or ask for your response to be deleted by contacting the researcher responsible: masterthesispedroazevedo@outlook.com. Your personal data will not be collected unless you provide it (by contacting the email address or subscribing to project updates at the end of the form). The data is stored on the University of Aveiro's Microsoft 365 infrastructure. After the project, identifiable responses will be deleted or irreversibly anonymised; only the aggregated analyses will remain in the dissertation.

About you

1. What is your gender? *

- ◯ Woman
- ◯ Man
- ◯ Non-binary
- ◯ Prefer not to say

2. How old are you? *

- ◯ < 18
- ◯ 18 - 25
- ◯ 26 - 35
- ◯ 36 - 45
- ◯ 46 - 55
- ◯ > 55
- ◯ Prefer not to say

3. Where do you live? *

- ◯ Açores
- ◯ Aveiro
- ◯ Beja
- ◯ Braga
- ◯ Bragança
- ◯ Castelo Branco
- ◯ Coimbra
- ◯ Évora
- ◯ Faro
- ◯ Guarda
- ◯ Leiria
- ◯ Lisboa
- ◯ Madeira
- ◯ Portalegre
- ◯ Porto
- ◯ Santarém
- ◯ Setúbal
- ◯ Vila Real
- ◯ Viana do Castelo
- ◯ Viseu

4. Occupation *

- [ ] Student (before university)
- [ ] Student (university)
- [ ] Working (full-time)
- [ ] Working (part-time)
- [ ] Athlete
- [ ] Not working
- [ ] Prefer not to say
- [ ] Other

You and Basketball

5. Do you play basketball? *

- [ ] Yes, in a competetive team
- [ ] Yes, in a professional team
- [ ] Yes, in university
- [ ] Casual games in public courts
- [ ] Casual games in private/payed courts
- [ ] Just shoot some balls
- [ ] No, but I would like to!
- [ ] No
- [ ] Other

6. When did you start playing basketball? *

- ◯ <16
- ◯ 16-19
- ◯ 20-25
- ◯ 25-30
- ◯ >30
- ◯ I want to start!

7. If you are in Lisbon and play at public courts, which ones do you usually go to?

- [ ] Doca de Santo Amaro
- [ ] Campo do Sobralinho (Vila Franca de Xira)
- [ ] Parque das Nações
- [ ] Caxias
- [ ] Moinho do Penedo (Monsanto)
- [ ] Santo Amaro de Oeiras
- [ ] Cascais
- [ ] Alcabideche
- [ ] São Domingos de Benfica
- [ ] Alcântara
- [ ] Conde de Bonfim
- [ ] Lumiar
- [ ] Benfica
- [ ] São Domingos de Rana
- [ ] Mártires da Pátria
- [ ] Santa Clara
- [ ] Chelas
- [ ] Other

8. If you do not play in Lisbon, where do you play? (court name, area, ...)

9. How often do you play in public courts? *

Check all that apply.

Please select at most 2 options.

- [ ] Very frequently (3 or more times a week)
- [ ] Sometimes (once a week)
- [ ] Occasionally (twice a month)
- [ ] Rarely (few times per year)
- [ ] Seasonal
- [ ] Never
- [ ] Other

10. Would you like to go more often *

- ◯ Yes
- ◯ No

11. Why are you not going more often? *

- [ ] Time constrains
- [ ] Nobody is there when I go
- [ ] It is always full
- [ ] I do not know where to go
- [ ] Weather conditions
- [ ] Bad court condition
- [ ] Location
- [ ] Injure
- [ ] I don't have enough people to play with
- [ ] I go enough
- [ ] Other

12. When you go to a public court you... *

- [ ] go with friends to play against each other
- [ ] go with friends to team up and play against or with others
- [ ] go alone hopping to find other players
- [ ] go alone to shoot some balls
- [ ] go alone to shoot some balls, but open to play some games if it comes up
- [ ] I usually go to private courts
- [ ] Other

Your opinion

13. Would you like to have more competition in casual games? *

- ◯ Yes
- ◯ I'm satisfied

14. What do you find annoying or dislike when you play at a public court?

15. Would you improve anything to have a better experience playing in a public court? *

- ◯ Yes
- ◯ No

16. If yes, what would you improve?

17. Would you like to receive updates on the project? Add your email!

APPENDIX B

Flyer

In the Figures B.1 and B.2 are the two versions distributed to get answers to the survey.

Figure B.1: Portuguese flyer

Figure B.2: English flyer