2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW) | 979-8-3503-3032-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASEW60602.2023.00006

Cross-platform mobile app development: the IscteSpots experience

João Cambaia de Almeida, Fernando Brito e Abreu, Duarte Sampaio de Almeida
ISTAR-IUL, Iscte - Instituto Universitário de Lisboa,
Av. das Forças Armadas, 1649-026 Lisboa, Portugal
Email: {jccga, fba, dsbaa}@iscte-iul.pt

***Abstract*—Cross-platform development frameworks allow producing a single codebase for an app targeting web browsers and native mobile operating systems. However, detractors stress their limitations in accessing platform-specific features or achieving optimal performance compared to native platform development. Although interest in cross-platform development has increased recently, few case studies are published on using them, often on toy examples.**

Therefore, it is important to provide sound evidence on the usage of a cross-development platform for a full-fledged app development case study, from requirements specification to quality assurance, using well-understood standard modeling notations (UML and BPMN).

This case study is about IscteSpots, a gamified app developed in the scope of Iscte's commemoration of its 50th anniversary to promote its heritage and history. Iscte is one of three public universities based in Lisbon, Portugal. IscteSpots provides publicly organized access to a chronological corpus of the university's past and is available on web browsers and on Android and iOS smartphones. IscteSpots is specifically targeted to mobile devices, implementing a contest with gamification strategies, specially targeted to the current community members (students mostly, but also teaching and administrative staff).

Development went through several iterations, including validations with groups of users that were instrumental in the app's continuous improvement. The vast majority of the suggested changes had repercussions at the Graphical user interface (GUI) level, that had to be propagated to the web, Android, and iOS platforms. The agility achieved by generating versions for the three target platforms, without noticeable degradation of execution efficiency and requiring only minor adaptations, amply proved the advantage of using a cross-platform framework.

***Index Terms*—cross-platform development, mobile app development, Flutter and Dart, software quality, UML, BPMN**

I. Introduction

To maximize availability, apps should run in web browsers, as well as in the two leading mobile operating systems: Android and iOS [8]. Creating separate codebases for each platform requires duplicating efforts, resources, and maintenance.

Several approaches have been proposed to write code that can be executed on different platforms with minimal changes, either (i) using platform-agnostic programming languages (i.e. not tied to a specific platform), such as Python, Java, or JavaScript [2], or (ii) by using progressive web app development frameworks [3]. However, although both alternatives offer some portability, they sacrifice the native-like user experience on each platform.

Granting native-like user experience requires using platform-specific User interface (UI) components, design patterns, and guidelines to ensure that an app looks and behaves consistently with other apps on the target platform. This ability to create software applications with just one codebase that can run on multiple operating systems or platforms, thereby reducing development time, cost, and effort, while leveraging native constructs is what is meant by *cross-platform development* [1]. There are generally two main strategies employed by cross-platform frameworks:

i) compilation to native code – some frameworks, such as Microsoft's Xamarin[1]) and open-source NativeScript, use a compilation approach where the code is transformed into native code specific to each platform. These frameworks typically provide bindings to platform-specific Application programming interface (API)'s, allowing developers to access native functionality. The code is compiled ahead of time (AOT) or just-in-time (JIT) during runtime, depending on the framework, to generate native binaries that can be executed on the target platform.

ii) interpretation or runtime execution – other frameworks, like Facebooks' React Native and Google's Flutter, use an interpretation or runtime execution approach. In these frameworks, the code is written using a common language (JavaScript for React Native, Dart for Flutter) and then interpreted or executed by a runtime environment provided by the framework. The runtime environment bridges the code to platform-specific APIs and UI components, rendering native-like interfaces and functionality.

Both approaches aim to provide cross-platform functionality, but there are trade-offs to consider. Compilation to native code often provides better performance and access to low-level platform features but may require additional setup, configuration, and potential platform-specific adjustments. On the other hand, interpretation or runtime execution simplifies the development process, allows for faster iteration, and provides a unified codebase, but it may introduce slight performance overhead and dependency on the framework's runtime. The popularity/interest in interpretation or runtime execution cross-platforms has peaked in the last months, as seen in Figure 1[2].

[1]Xamarin is now part of .NET

[2]NativeScript is not represented since it never rises above 1% in Google Trends' percentile scale

2151-0849/23/$31.00 ©2023 IEEE
DOI 10.1109/ASEW60602.2023.00006

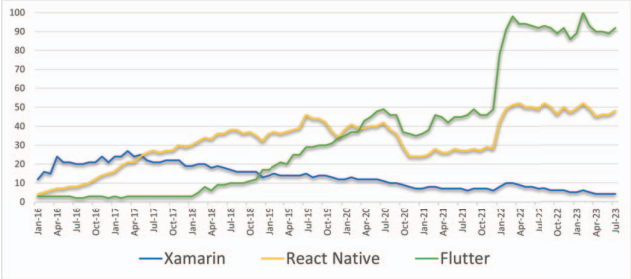

Fig. 1. Popularity of cross-platforms worldwide (source: Google trends)

This paper is organized as follows: Section II holds the related work for this article; in section III our development environment and choice of the Flutter framework in comparison with other frameworks are explained; section VI is where our code quality pipeline is described, in section V a Business Process Model and Notation (BPMN) model is used to describe our internal improvement and deployment processes; in section IV we delve extensively into every conceivable interaction that can be performed with the developed application; in section VII validations with real students are briefly described along with a focus on their feedback, and finally, in section VIII, we draw some conclusions on our case study.

II. Related work

By just focusing on the interpretation or runtime execution frameworks that account for the largest market share, as mentioned in the previous section (React Native and Google's Flutter) we performed some searches on SCOPUS, using the following search string: *"cross-platform development" AND "case study" AND (Flutter OR "React Native")*. We then selected the papers that provided the most interesting insights into the use of Flutter and React Native in real-world scenarios.

Khan et al. [6] compare the performance of Flutter and React Native in terms of efficiency, effectiveness, compatibility, community growth, documentation, architecture, developer productivity, and testing automation support. Zohud and Zein [10], on the other hand, compare the performance of cross-platform development compared with native app development.

Szczepanik and Kedziora [9] discuss state management approaches used for Flutter applications development and propose a combination of two approaches that solve the main problem of existing approaches related to global and local state management.

Galán et al. [5] present a multi-criteria comparison of two mobile applications built with the use of Android and Flutter SDK, showing that a mobile application written using Android SDK is often not only faster and more efficient but also has greater community support and the number of libraries available.

Cheon and Chavez [4] describe a case study of converting an existing Android app written in Java to a Flutter version to support both Android and iOS, discussing technical issues, problems and associated with such a rewriting effort.

Overall, the previous papers provide valuable insights into the use of Flutter and React Native in real-world scenarios, highlighting their strengths and weaknesses in terms of performance, state management, community support, and app conversion.

III. Development Environment

The communities surrounding a language or framework are essential in a software development ecosystem. Therefore, its popularity conveys technology's stability and longevity, since if it is widely used, it is more likely to have ongoing development, updates, and bug fixes. Abandoned or less popular technologies may face a higher risk of becoming obsolete.

According to Google Trends (see Figure 1), Flutter became the most popular cross-platform framework in April 2020. It is a framework built on top of the Dart programming language and the Skia 2D graphics engine with a unique take on front-end development where everything is a widget. These widgets are primarily objects that represent UI as code but can contain functionality, logic, and also hold state.

Both popularity winners – Flutter and React Native – have strong online communities and a wealth of tutorials and package libraries; therefore, popularity should not be the single choice criterion. Performance is also important to consider when choosing a framework or language to develop with. Although React Native supports performant native animations, its performance in most cases is outshined by Flutter, which revealed to have performance "*on par with native solutions for CPU-intensive tasks*" as mentioned in [7].

We then chose Flutter due to its focus on performance, rich ecosystem corroborated by its popularity, and its use of the Dart programming language, similar to the traditional OOP languages like Java that the IscteSpots team was familiar with.

IV. Implementation

Our technology stack consists of a Django web server and a mobile app developed with Flutter and deployed on both Android and IOS devices.

A. Software architecture

Figure 2 describes how IscteSpots components interact with each other. Figure 3 shows how those components were deployed in the intervening computing nodes.

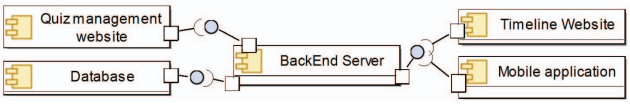

Fig. 2. IscteSpots UML Component Diagram

1) Backend: Our Django server is used as the storage and logic backbone of our solution also providing Content management system (CMS) capabilities, using Django's admin panel we created a web portal for authorized members of Iscte's staff to interact with and manage the contents present in the app (i.e. questions and images). It also provides a

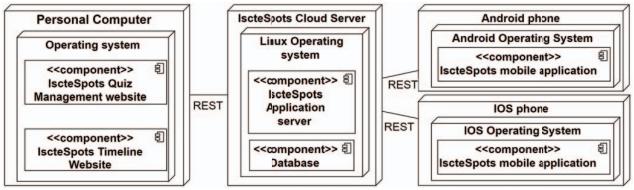

Fig. 3. IscteSpots UML deployment diagram

Restful API programmed by our IscteSpots team that the mobile and web applications consume to provide consistent user progression within our game.

2) Mobile application: As the main consumer of the API provided by our Django Backend, the mobile application published on the app stores for both Android and Ios devices serves as the main platform for the user to interact with our IscteSpots game.

3) Web application: An easier method to consult the timeline curated for the IscteSpots game was developed as a website, which due to Flutter's now great support for the web, could be as simple as copying code from one repository to another. The only reason it was not as straightforward as it seemed, was due to external packages that supported the mobile platforms but not the web. Such an example was sqflite, since it forced us to rewrite the logic for the timeline, which in turn made us rethink and redesign the Rest API provided in the backend for the contents of this page.

To develop the web application, we needed to make modifications to 27 out of 155 ".dart" files which involved changing a total of 2115 lines of code out of the entire project's 14309 lines (i.e. 14.8% of the entire codebase). This number can be further reduced through refactoring.

4) Web Content Management Portal: The Anthropology team, which collaborated with the IscteSpots team, created the corpora on the history and heritage of the University. Building a custom portal to edit these contents was key in building an improved repertoire of questions, images, and events to display on the IscteSpots app and timeline website. For that purpose, we used Django's Admin panel so that the Anthropology team could have editing permissions on the database tables relevant to the contents, such as the one storing the questions for the quizzes and the timeline events.

B. Mobile Application Overview

In this section the complete application flow will be described, using the BPMN diagram in Figure 4 as an anchor. The latter was developed with Signavio academic version, and contains a single pool, named "Flutter Mobile App" which represents the IscteSpots mobile application, containing 14 lanes, each representing a specific page of the app. Whenever tasks were more complex, we used sub-processes instead. The corresponding diagrams cannot be reproduced here due to size limitations.

1) Onboarding: On the initial load of the app, the Onboarding page is displayed, composed of a series of "tabs" that the user can horizontally scroll through to learn the details of the "IscteSpots" game.

2) Application Drawer: The app drawer consists of a vertical list of buttons that navigate the user to the different pages of the app, serving as the main method of navigating the user through the app, represented through the Intermediate conditional event named "App drawer/nav bar" that also encompasses the behavior of the bottom navigation bar of the home page

3) Home page : The home page is composed of three different tabs, present as buttons on the bottom navigation bar. The first button takes the user to the puzzle page, the default page where the app initially loads into, explained in section IV-B3a, the middle button takes the user to the leaderboard screen, where he can see how he compares to the rest of the players using the app, further explained in section IV-B8. Finally, the right button takes the user to the scan page, where he can scan for spots/Quick response code (QR code)s, explained in section IV-B5

a) The Puzzle: is the main page of the application, where the user can drag pieces of the puzzle around the screen while trying to slot them onto their designated place, task number 5 of the diagram. The image used for the puzzle is based on the user's chosen Spot, explained in detail in section IV-B4. When the user **completes the puzzle** he is recommended by the app to perform a scan on a QR code, task number 6 and better explained in section IV-B5. The user can **provide feedback** about the app using a button present on the top bar, which displays an overlay containing a form, task 7 in the diagram.

4) Choose the next spot: Here the user can see a grid of blurred images, each corresponding to a physical location where a "Spot" or QR code can be found and scanned, each of these images is clickable, selecting it as the current spot and changing the puzzle accordingly.

5) Scan a spot: The scan-a-spot page displays what is being captured by the phone's camera and draws an overlay on top that guides the user while pointing his phone at the QR code. This lane in the diagram features a single collapsed task named "Scan spot page Sub-process" number 9 that encompasses all the behavior and interactions present in this Scan Spot page in greater detail.

6) Timeline: The timeline page, lane number 7 in the diagram, presents a horizontal list of years on the top and a vertical list of all the events from the selected year. The top bar features a button to display an explanation dialog and a magnifying glass button that opens the filter overlay where the user can select topics and scopes filtering the displayed years and events. Once filtered, they can either view event details or return to the main timeline page, without the ability to apply additional filters. The Study timeline page is a filtered version of the normal timeline page (sharing the majority of their code), that only contains the events relevant to a Quiz or Spot/QR code and cannot be further filtered.

7) Quizzes: This page contains the list of available quizzes, each one displaying the user's points for each attempt, a button for studying the quiz's content, and one to begin a

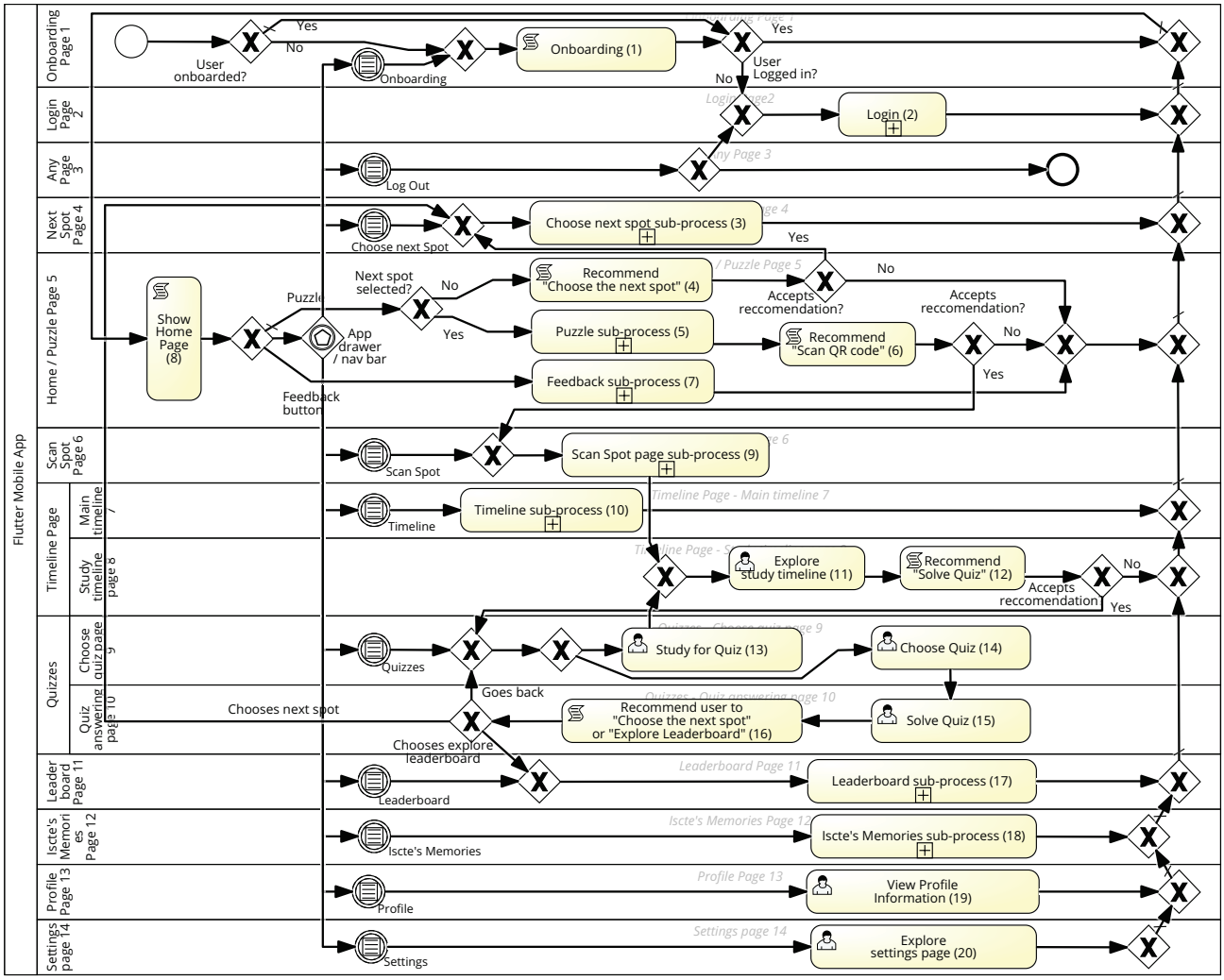

Fig. 4. BPMN diagram describing all the application tasks and navigations possible (tasks are numbered to facilitate descriptions' traceability)

new attempt. Answering the quiz questions, task number 15, the app displays a horizontal timer to display the timed nature of the questions to the user. After answering the questions the user is presented with a score page and offered options to visit the "Leaderboard" or the "Next Spot" page.

8) Leaderboard: The Leaderboard page contains three tabs corresponding to different sub-screens. In them, the user can view a global list of all the players, filter by course or department, and even view a list of the closest players to him.

For further details and screenshots see our source code on the GitHub repository.

V. Iterative Development Process

The iterative development process will be described using the BPMN diagram in Figure 5 as an approximation to the adopted process that contains three distinct process flows, a weekly and sporadic development cycle flows, and one for the interactions between the user and the application.

A. The weekly development cycle

The development cycle encompassed weekly Zoom meetings where a specific set of features were agreed to develop, kickstarting the weekly flow of the diagram in Figure 5.

Firstly the latest version of the project is fetched from our GitHub online code repositories, then the defined changes are fetched from our product backlog hosted on our internal Trello board and implemented through code, testing for bugs or errors and fixing those if they are found, represented in task number 2 and lastly uploading the new version of the project to our GitHub repository through a new commit on the specific branch created for the feature in mind.

B. The deployment cycle

The sporadic cycle that is represented in the lower half of the Android Studio lane in the diagram, Figure 5 represents the deployment process that is triggered when a new version of the app is required on the public application marketplaces.

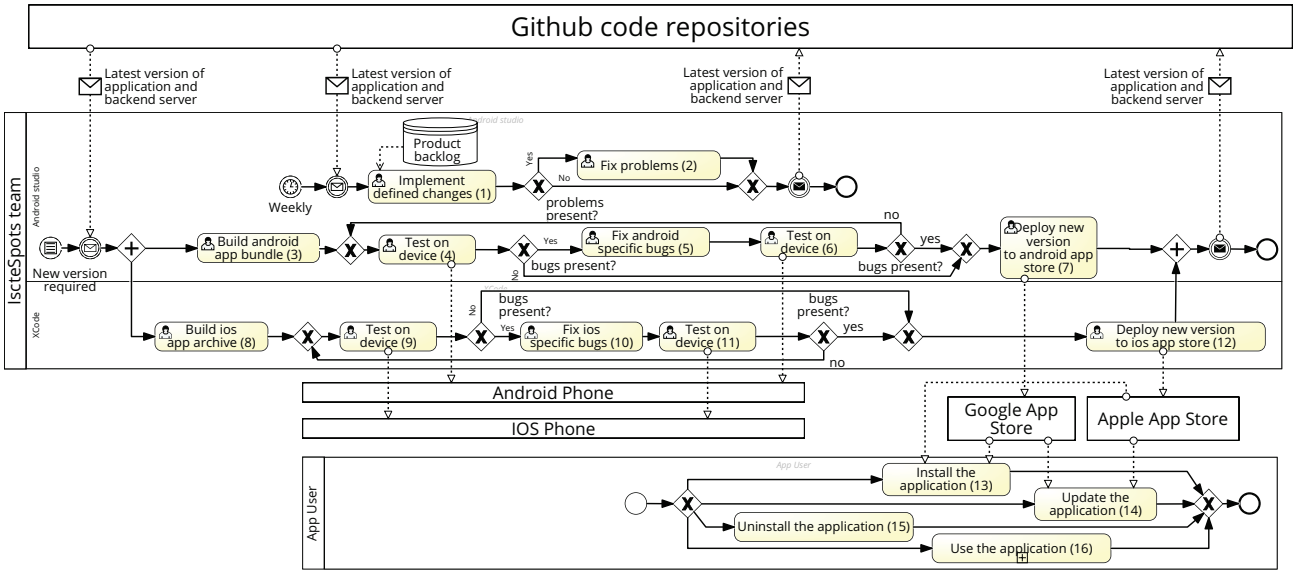

Fig. 5. Iterative development process described in BPMN (tasks are numbered to facilitate descriptions' traceability)

This deployment cycle begins with the "New version required" event node and similarly to the weekly cycle, the first operation required is to download the latest version from GitHub's repository, then the respective app compilations are built, in tasks numbers 3 and 8, represented in different lanes due to the different programs used for each one.

Once the compilations are built, thorough testing of the app is conducted on both platforms, if any issues are found they are resolved and testing is repeated on the devices, represented on Android with tasks 4, 5, and 6 and on IOS with tasks 9, 10, and 11. Otherwise, we skip directly to deploying the new version on the respective app stores, with task number 7 and 12. The process concludes by uploading any code changes to the online repository on GitHub.

VI. Code Quality and CI/CD

Code quality plays a pivotal role in the realm of software development, significantly impacting the reliability of a software product, if executed correctly demonstrates several crucial benefits. First and foremost, it enhances **maintainability**, making it easier for developers to understand and extend the codebase over time. Clean, well-structured code fosters **collaboration**, and improves software robustness and stability, reducing the occurrence of defects. By adhering to coding best practices and standards, developers can create **optimized** and **scalable** code. Ultimately, ensuring great code quality results in a more stable, scalable, and maintainable software product, thereby enhancing customer satisfaction and overall project success.

A. CodeMagic

CodeMagic is a Continuous Integration and Continuous Development (CI/CD) platform specifically designed for Flutter and Dart development, now with support for other frameworks and native development. It is easy to use and understand, and it has a great quantity of documentation and community support specifically for Flutter pipelines, making it a great contender for creating a CI/CD solution for Flutter apps.

CodeMagic provides the option to configure the CI/CD pipeline through a customizable configuration file in YAML allowing the whole pipeline to be declared in code (infrastructure as code) as opposed to a manual configuration which is time-consuming, error-prone, and difficult to replicate.

Such pipelines can include running static code analysis, unit tests, building the app compilation bundles, and publishing the new version on the app stores.

B. SonarCloud

As for the Django web server code, we have chosen SonarCloud, a cloud-based static analysis tool with easy set-up, Python support, the primary language used with Django, and provides a direct connection to our GitHub code repository.

Besides the configurable Python codebase quality checks, including the identification of code smells, maintainability issues, and best practice violations, SonarCloud detects security vulnerabilities and identifies potential weaknesses, thus enforcing the app's robustness and security.

VII. Validation Tests

We conducted two beta testing sessions with students on our campus, each lasting approximately one hour. In total, we had 22 participants, 8 recruited by the IEEE Iscte Student Branch and 14 by the ISCTE-IUL ACM Student Chapter. As we did not establish any preconditions for the profile of the participants, other than their willingness to participate, nor did we interfere in the selection process itself, we can consider that the selection of subjects was random.

By the end of each beta-test, they filled the NASA Task Load Index (NASA-TLX) and the Mobile Application Rating

Scale (MARS) forms to provide feedback about their usage with the app. Overall, this assessment exercise provided very positive feedback regarding the effort involved in using the app, visuals and functionality were regarded as good (4 out of 5), customization was considered fair (3 out of 5) and, frustration levels while using the app were very low (3.5 out of 21). The only less positive aspect was temporal demand (10 out of 21) indicating that the game took up too much time. A compilation of the answers to both feedback forms is presented in Figure 6. Each radar chart represents the multiple sections of both forms as a separate axis with the mean scores in blue, the minimum in red, and the maximum in yellow, the NASA-TLX chart ranging from 0 to 21, and the one of MARS from 0 to 5.

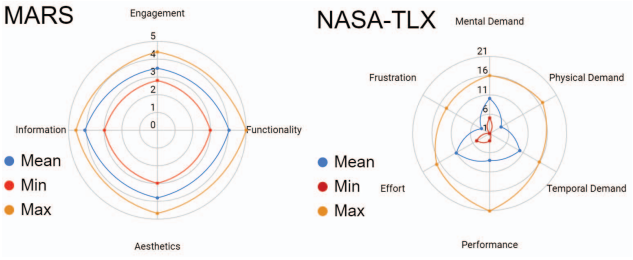

Fig. 6. Feedback from validation tests on IscteSpots app's usability

VIII. Conclusions

Mobile app development presents a multitude of challenges, ranging from platform fragmentation and varying device capabilities to optimizing user experience across different screen sizes and addressing security concerns. Balancing functionality with performance, ensuring seamless offline capabilities, navigating through app store guidelines, and maintaining user engagement through frequent updates further compound the intricacies.

In this paper, we described our experience in the development of the IscteSpots app where cross-platform development allowed us to mitigate the aforementioned challenges. The main benefits that we can report and hope will inspire researchers and professionals when they will need to develop a mobile app are:

Code reusability/resource efficiency – Writing code once and deploying it across multiple platforms reduced our development time and effort, as well as the need for maintaining separate codebases.

Wider audience reach – By targeting multiple platforms we could reach users on different operating systems either using mobile apps (iOS and Android) or using web browsers on Windows, MacOS, or Linux.

Consistent user experience – We were able to offer consistent user interfaces and functionality across the different platforms, providing a seamless experience.

Throughout the development process, Flutter allowed us to rapidly iterate over concepts and features, Dart's flexibility and user-friendliness accompanied by Flutter's vast online community with packages for practically every developer challenge, along with a wealth of tutorials including those from the Flutter team detailing their framework's different widgets and their uses further emphasized its excellence.

We performed IscteSpots validation tests using two instruments (NASA Task Load Index, and the Mobile Application Rating Scale) with 22 subjects as surrogates of the final users. Overall the feedback received on these validation tests was very positive, highlighting the quality of the IscteSpots app.

We believe this was due to a combination of factors: (i) the aforementioned potentialities inherent to Flutter/Dart itself, (ii) the iterative development approach with users' feedback in each cycle, (iii) the shared understanding of the intended app behavior, where the BPMN models were a cornerstone and, last but not least (iv) the adoption of an automated CI/CD pipeline with embedded quality assurance tools.

Acknowledgments

This work was partially supported by the Portuguese Foundation for Science and Technology (FCT) projects UIDB/04466/2020 and UIDP/04466/2020.

References

[1] Amatya, S., Kurti, A.: Cross-Platform Mobile Development: Challenges and Opportunities. In: ICT Innovations 2013, pp. 219–229. Springer International Publishing (2014). doi:10.1007/978-3-319-01466-1_21

[2] Andreeva, J., Dzhunov, I., Karavakis, E., Kokoszkiewicz, L., Nowotka, M., Saiz, P., Tuckett, D.: Designing and developing portable large-scale JavaScript web applications within the Experiment Dashboard framework. Journal of Physics: Conference Series **396**(5), 052069 (dec 2012). doi:10.1088/1742-6596/396/5/052069

[3] Biørn-Hansen, A., Majchrzak, T.A., Grønli, T.M.: Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies. SCITEPRESS - Science and Technology Publications (2017). doi:10.5220/0006353703440351

[4] Cheon, Y., Chavez, C.: Converting Android Native Apps to Flutter Cross-Platform Apps. In: 2021 International Conference on Computational Science and Computational Intelligence (CSCI). IEEE (dec 2021). doi:10.1109/csci54926.2021.00355

[5] Gałan, D., Fisz, K., Kopniak, P.: A multi-criteria comparison of mobile applications built with the use of Android and Flutter Software Development Kits. Journal of Computer Sciences Institute **19**, 107–113 (jun 2021). doi:10.35784/jcsi.2614

[6] Khan, S.M., Nabi, A.U., Bhanbhro, T.H.: Comparative Analysis between Flutter and React Native. International Journal of Artificial Intelligence & Mathematical Sciences **1**(1), 15–28 (sep 2022). doi:10.58921/ijaims.v1i1.19

[7] Oliveira, W., Moraes, B., Castor, F., Fernandes, J.P.: Analyzing the Resource Usage Overhead of Mobile App Development Frameworks. In: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering. pp. 152–161 (2023). doi:10.1145/3593434.3593487

[8] StatCounter: Mobile operating system market share worldwide. StatCounter Global Stats (2023), https://gs.statcounter.com/os-market-share/mobile/worldwide, accessed 2023 Jul 17

[9] Szczepanik, M., Kędziora, M.: State Management and Software Architecture Approaches in Cross-platform Flutter Applications. In: Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering. SCITEPRESS - Science and Technology Publications (2020). doi:10.5220/0009411604070414

[10] Zohud, T., Zein, S.: Cross-Platform Mobile App Development in Industry: A Multiple Case-Study. International Journal of Computing pp. 46–54 (mar 2021). doi:10.47839/ijc.20.1.2091