Flutter and iOS Application Comparison

An Empirical Metric Analysis of Performance and User Experience

Bachelor Thesis

submitted by:	Philip Krück
Date of Birth:	04.11.1998
Matriculation Number:	3938
Company Supervisor:	Jan Jelschen
First Reviewer:	Dr. Oliver Becker
Word Count:	< 12.000 (text + footnotes)
Degree Program:	B.Sc. Business Informatics (A Track 2018)
University:	Hamburg School of Business Administration
Submission Date:	09.04.2021
Partner Company:	apploft GmbH

HSBA HAMBURG SCHOOL OF BUSINESS ADMINISTRATION

Contents

Chapter 1

Introduction

Mobile platforms are dominated by two players - Apple and Google with their respective operating systems iOS and Android. Cumulatively, they form a duopoly in the smartphone operating systems market with a combined usage share of 15.2% for iOS and 84.8% for Android in 2020 according to IDC (2021).
To develop a mobile application for both target platforms, the corresponding development environments and technologies are utilized for each platform. This leads to a doubling of cost, development time, and the need for knowledge of two different application development paradigms. This has resulted in the creation of cross-platform frameworks such as Xamarin (Microsoft Corp. 2021), React Native (Facebook 2021), and Ionic (Ionic 2021).

The fundamental principle behind these frameworks is a specific tech stack operating on a single code base leading to increased development speed while also having the ability to deploy for both operating systems.

Generally, cross-platform frameworks utilize a software bridge to communicate with the underlying host platform plugging into native interfaces. During app runtime, transmission delays may occur leading to reduced execution speed. In addition, cross-platform frameworks provide abstract interfaces over multiple native interfaces leading to a decreased subset of functionality and especially impaired UI customizability. These inherent architecture attributes (further see Section 2.1) explain both impeded performance (Ebone et al. 2018) and user experience (Mercado et al. 2016) compared to native technologies.

Flutter (Google Inc. 2020) is a newly developed open-source UI toolkit developed by Google. It has a nonconventional approach to cross-platform development in that every app ships with the Flutter rendering engine. Thereby, Flutter bypasses host platform communication in terms of

UI rendering and delivers a natively compiled binary which can be executed by the underlying device (see Section 2.2). Furthermore, the framework provides a *"[...] collection of visual, structural, platform, and interactive widgets"* (Google Inc. (2021b)) for UI customization. It seems that Flutter addresses the very issues that are generally criticized about cross-platform solutions.

Unfortunately, since Flutter was first released in March 2018 (Google Inc. 2021a), there are no peer reviewed articles comparing the performance or usability to native apps[1].

1.1 Motivation

As a digital agency specialized on native iOS and Android development, apploft GmbH (the partnering company of this thesis) is highly interested in Flutter. The implications of using this framework could be wide ranging including the extension of the services portfolio to clients with lower budgets. For example, startups which are unsure about product market-fit and held by tight budget constraints are especially keen on reaching the maximum number of potential customers with their app. Flutter could be utilized for a fast iteration of a product deployable to both mobile platforms. Worth noting are the benefits of serving smaller customers like startups which include higher growth potential for long-term cooperation and risk diversification in apploft's client portfolio.

The potential of Flutter goes beyond small clients. Instead of focusing on platform customization with native tooling, more effort could be directed into developing unique custom features. Furthermore, infrastructure setup, package development and app updates would only be necessary for one codebase.

The above stated possible implications are not only relevant for apploft, but mobile application developers at large.

1.2 Thesis Goal

Based on the above stated problems with cross-platform frameworks and the potential business opportunities, the goal of this thesis is to evaluate whether Flutter's claims on performance and usability hold up in practice.

To properly compare Flutter and native, an application will be developed which has typical

1. A search for relevant articles has been conducted using Google Scholar, Sci-hub and IEEE Xplore.

mobile app features including the interaction with a remote API, different means of navigation between screens, user authentication and authorization. The feature selection process is further explained in Section 3.1.

Based on these characteristics, *Kickdown* (Kickdown GmbH 2021) - an online car auction app - was chosen. The app has already been developed for iOS by apploft. To verify laid out claims of the Flutter framework, an app with specific matching functionality and UX design is reproduced.

1.2.1 Performance comparison

To evaluate performance, the typical measures of CPU, GPU and memory usage are chosen in this thesis. On the one hand, these metrics are the underlying causes of more ephemeral metrics such as page load speed. On the other hand, they can be easily measured using software tools (see Section 3.2).

1.2.2 Usability comparison

Expert interviews with employees of apploft are conducted to compare the user experience of the developed Flutter app with the iOS application.

1.3 Scope & Limitations

The feature set of the implemented app is representative for most, but not every type of app. Therefore the results cannot be generalized to every type of possible app. However, the results should be seen as indicators of Flutters value as a cross-platform framework for the archetypal mobile app.

Additionally, the usability study doesn't have a statistical significance due to its qualitative nature. Nevertheless, the depth of detail in expert interviews is much greater compared quantitative methods, and unthought of considerations may be suggested by the interviewees.

This research attribute is especially compelling given the current research state on the Flutter framework as mentioned above.

1.4 Plan of Attack

The following is a list of subgoals of this thesis including accompanying deadlines.

1.4.1 App Development

- complete development of Flutter app* - 26.03.21

1.4.2 Section Writing

- ~~write draft of *Introduction* Section and discuss - 05.02.21~~
- write draft of *Flutter* Section and discuss - 19.02.21
- write draft of *Study Design* Section and discuss - 05.03.21
- write draft of *Application Design* Section and discuss - 19.03.21
- write draft of *Performance Comparison* Section and discuss - 02.04.21
- write draft of *UX Comparison* Section and discuss - 02.04.21
- write draft of *Summary* Section and discuss - 02.04.21
- write draft of *Abstract* Section and discuss - 02.04.21

1.4.3 Performance Measurement

- prepare tooling and conceptualize measurement process - 26.02.
- execute a first iteration of measurement process - 05.03.
- execute a final iteration of measurement process - 26.03.

1.4.4 UX Measurement

- prepare interview process - 26.02.
- conduct a minimum of 3 interviews - 31.03.
- evaluate interviews - 01.04.

1.4.5 Submission

- topic submission - 15.02.
- send out thesis for proof reading - 05.04.21
- submit to examination office + register for colloquium - 09.04.

*The minimum requirement is to complete building out the *offerings* screen. This is the most complex screen of the app and constitutes the main feature. It is sufficient for performance comparison as well as a usability study. However, if time permits, more of the app will be developed and comparatively evaluated.

Chapter 2

Flutter

2.1 Mobile Developement Tiers

2.1.1 Web Apps and Progressive Web Apps

Web apps are applications run and rendered in the browser. The underlying technologis are HTML, CSS and JavaScript with popular frameworks used for the development process such as Angualar (ref), React.js (ref) and Vue.js (ref). Web apps rely on the Browser and an internet connection. They do not have direct access to hardware capabilites such as the camera or the file system.

Progressive Web Apps (PWAs) function similarly to web apps but provide additional capabilities such as offline use, locally cached data, push notifications, and the ability to add them to the home screen.

2.1.2 Hybrid Apps

Hybrid Apps utilize the same technologies as web apps, but they are rendered in a platform web view. Additionally they provide the ability to interact with native APIs through a platform bridge (see 2.1). Popular framework choices for building Hybrid Apps include Ionic and Cordova Hybrid apps unlike web apps are shippable to official stores.

2.1.3 Web Native Apps

Web Native Apps which are built with frameworks such as React Native (ref) or Native Script (ref) utilize the OEM components instead of web views. This is achieved by using JavaScript

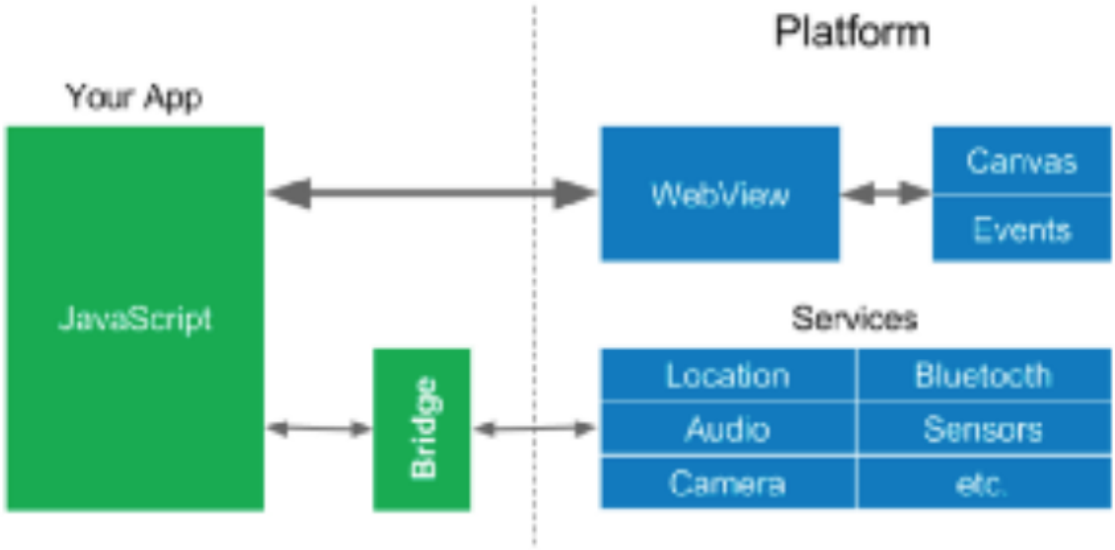

Figure 2.1: Hybrid architecture (image to be replaced).
(ref: https://www.freecodecamp.org/news/a-deeply-detailed-but-never-definitive-guide-to-mobile-development-architecture-6b01ce3b1528/)

calls to a platform bridge which transpiles the JavaScript code into native platform code (see 2.2). The views are thus in Web Native Apps rendered with native technologies.

2.1.4 Cross Compiled Apps

Cross Compiled Apps utilize a compiled programming language such as C# (used for Xamarin) to generate byte or machine code which is then executed on the host platform. This machine code performs similar actions as web native apps in that platform APIs and OEM widgets are called (see 2.2). Flutter can also be classified as a cross compiled architecture. However, Flutter ships its own widgets and rendering engine instead of relying on the host platform (see 2.3). These technicalities will be explained in Section 2.2.

2.1.5 Native Mobile Applications

Native applications directly communicate with the OEM components and APIs which are part of the operating system and thus tightly integrated into the platform. Making full use of hardware and software integration, native apps yield the highest performance and user experience.

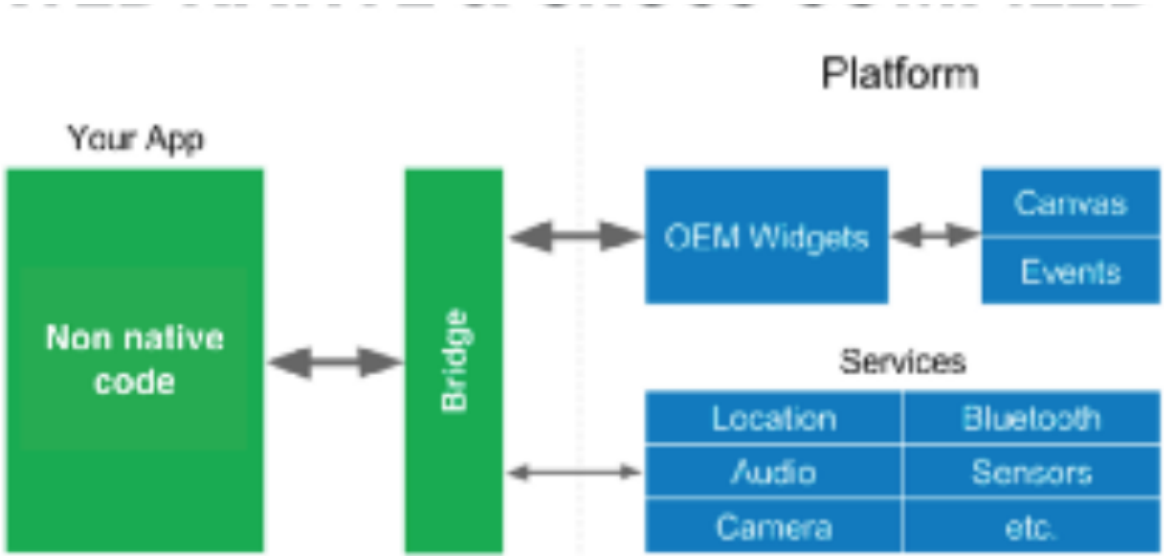

Figure 2.2: Web Native and Cross Compiled Architecture (image to be replaced).

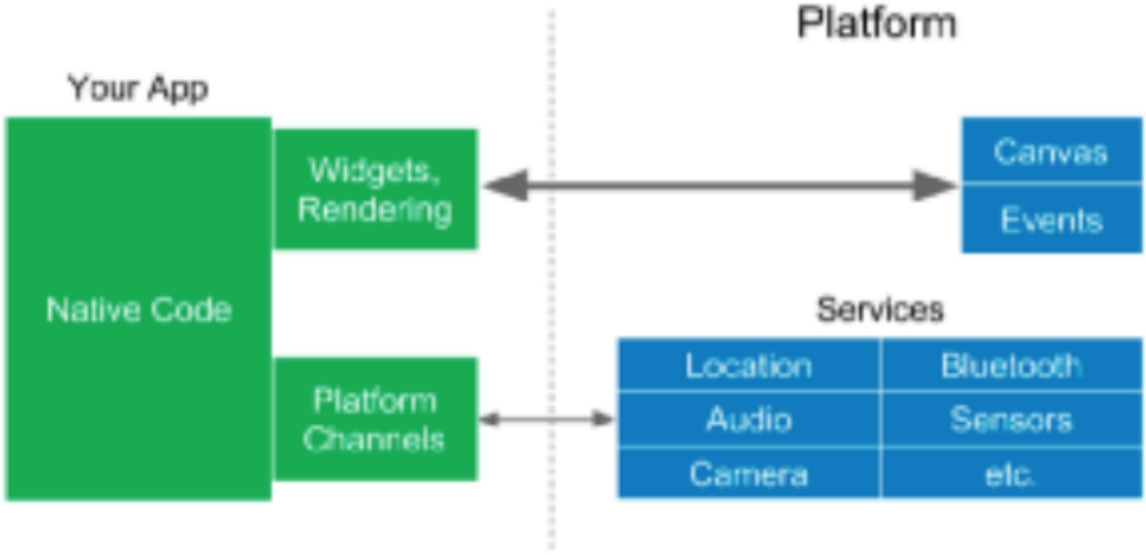

Figure 2.3: Flutter Architecture (image to be replaced).

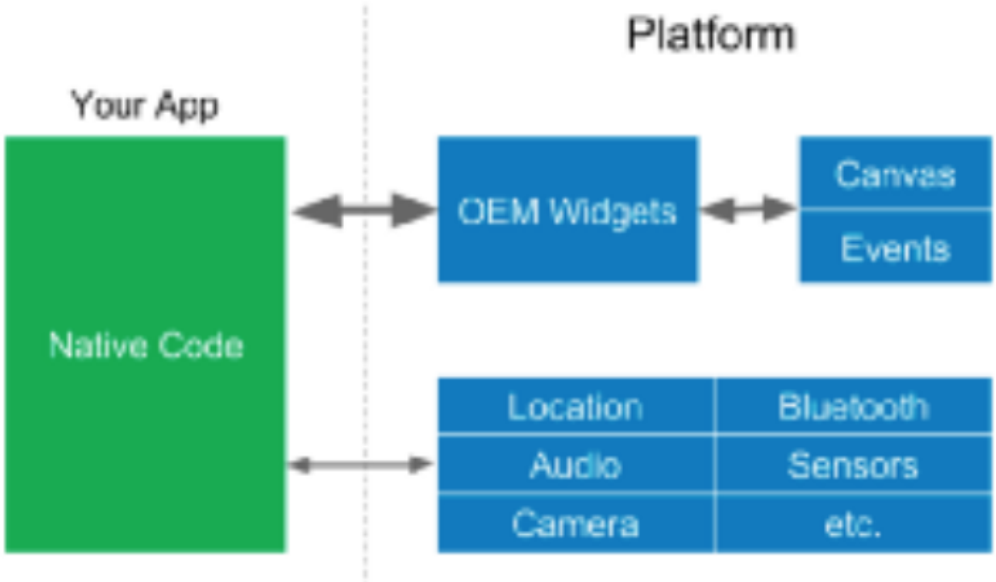

Figure 2.4: Native Mobile Architecture (image to be replaced).

2.2 Flutter Framework Architecture

Flutter's framework architecture is composed of three distinct layers: framework, engine and embedder. The framework acts as the top most layer which app developers interact with. It features the widgets, animation and gestures API as well as platform specific UI components delivered through the Material (Android) and Cupertino (iOS) package.

Below the framework layer lies the engine which unlike the framework isn't written in Dart, but in C and C++. Presumably, this design choice has been made to easily allow the production of native binaries. This act of cross-compilation is Flutter's technical value proposition to increase execution performance as stated in Section 1. The engine layer includes Skia - a 2D graphics rendering engine which is also used in the Chrome web browser.

At the bottom lies the embedder layer. Its sole purpose is to integrate the Flutter application into the platform-specific environment by providing native plugins and event loop interoperation. Additionally, app package formatting is provided in the same way as for native apps. The host operating is thus not able to differentiate between a Flutter and a natively written app.

Flutter aims to minimize application size as it's already shipping the entire framework including the rendering engine with each app which amounts to approximately 4.5 mb (https://flutter.dev/docs/resources/

Common app platform plugin functionality like camera access or webview interaction are extracted in separate packages. Networking, animations or other platform agnostic functionality

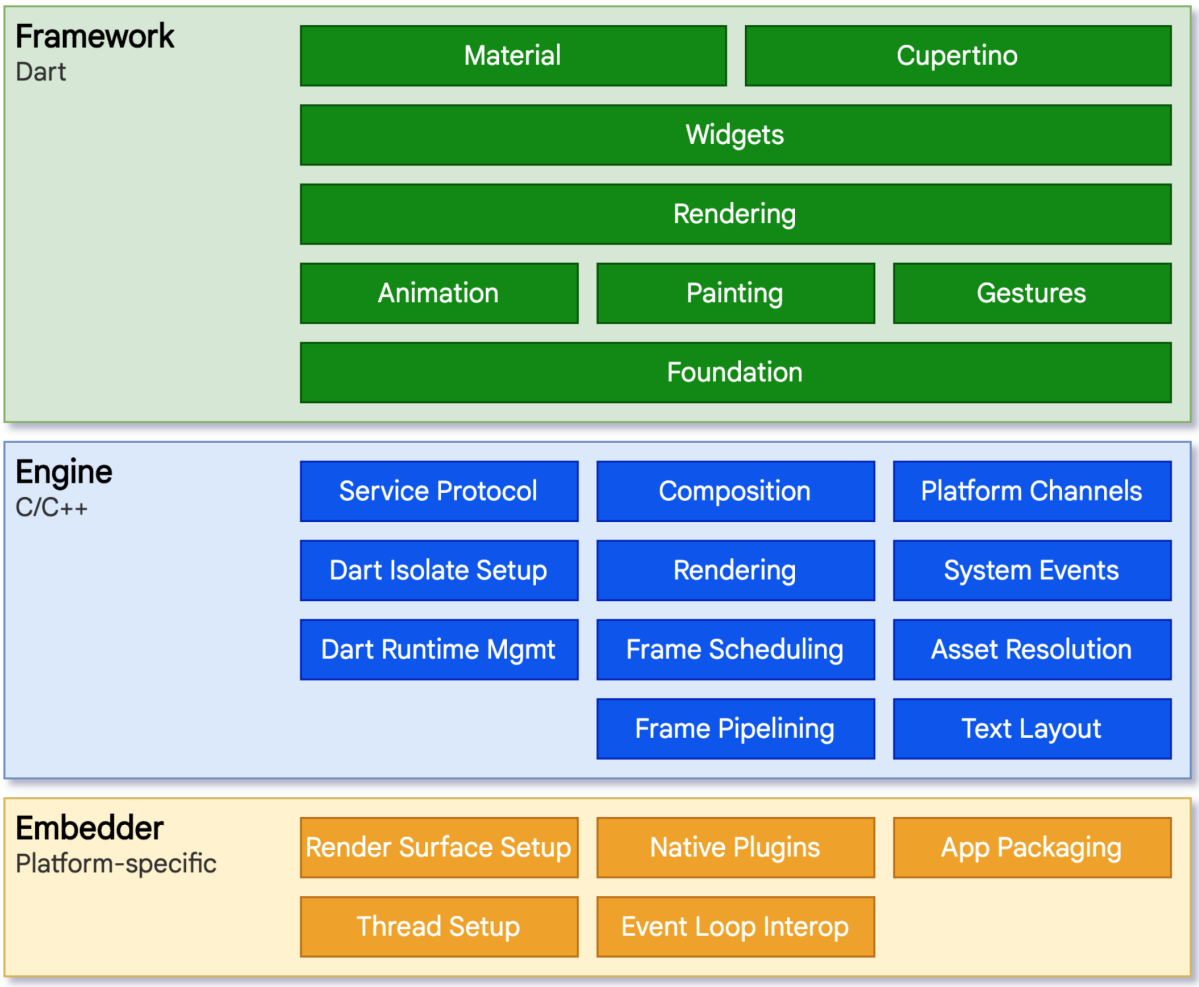

Figure 2.5: Flutter's Layered Architecture (ref to include https://flutter.dev/docs/resources/architectural-overview).

is also bundled in packages.

2.2.1 Programming Language and Compilation

Flutter apps are written in Dart an multiparadigm programming language syntactically similar to Java. (https://dart.dev/overview#language)

During Flutter development, apps run in the Dart Virtual Machine with JIT compilation. This offers stateful hot reload which allows reload changes without needing to fully recompile the app leading faster development iteration. For release purposes, Flutter applications are Ahead-of-Time compiled into native machine code including the Intel x86 and ARM instruction sets. (https://flutter.dev/docs/resources/architectural-overview).

2.2.2 Rendering and UI State

Flutter apps are fundamentally composed of widgets which declare structural, stylistic and layout elements for building the user interface. Each widget may have 0, 1 or multiple children which in turn create a tree structure of parent-child relationships. For example, a Column which has the purpose of laying out its children vertically has both a Text and an Image as its children. Both the Text, and Image widgets have no children and represent the leaf nodes in the tree. The Column is an invisible structural layout element while Text and Image are stylistic. [1] The entry point of every Flutter app is either a 'MaterialApp' or 'CupertinoApp' widget which also marks the root of the tree. Based on this tree structure Flutter can determine where and how elements should be drawn on screen, and instruct its graphics engine accordingly.

This concept alone is not yet sufficient to enable modern mobile applications with complex UI changes or animations based on asynchronous events like user interaction. Widgets are mappings from state to a UI representation. When the state changes during runtime Flutter creates a new widget tree, diffs it against the old one and then redraws only its changes to the screen. This declarative approach simplifies UI development in the sense that the developer does not need to keep track of UI state which can grow exponentially with the increase of UI components on screen.

2.2.3 Method Channels

To utilize platform functionality like camera access, geolocation or other sensor data, Flutter communicates with the platform's native APIs via method channels (see architecture diagram). Common functionality is already provided by Flutter and third party packages, but custom platform channel functionality may be implemented as required. (reference Flutter doucmentation...)

2.3 Flutter's Limitations

Unlike other cross-platform frameworks, Flutter doesn't use OEM components, but its own components instead (widgets).

When the underlying OEM component of the host platform changes with an OS update, Flutters' framework needs to updated to resemble this new functionality. Flutter always has to

1. Technically both Text and Image do have an implicit single child widget which is created by the framework.

continuously replicate these changes in their framework layer. As Google is using Flutter itself in multiple production apps (cite something), the company has an incentive to swiftly replicate OS features. Additionally, since Google also owns Android, coordination may be easier and not as surprising. The benefit though is that once the new widget is rebuilt in Flutter it can even be shipped to older operating systems.

Bibliography

Ebone, A., Y. Tan, and X. Jia. 2018. "A Performance Evaluation of Cross-Platform Mobile Application Development Approaches." *IEEE.*

Facebook. 2021. *React Native.* `https://reactnative.dev/`.

Google Inc. 2020. *Flutter Homepage.* `https://flutter.dev/`.

———. 2021a. *Flutter SDK Releases.* `https://flutter.dev/docs/development/tools/sdk/releases`.

———. 2021b. *Widget Catalog.* `https://flutter.dev/docs/development/ui/widgets`.

IDC. 2021. *Smartphone Market Share,* December. `https://www.idc.com/promo/smartphone-market-share/os`.

Ionic. 2021. *Ionic Framework.* `https://ionicframework.com/`.

Kickdown GmbH. 2021. *Kickdown.* `https://www.kickdown.com/`.

Mercado, I. T., N. Munaiah, and A. Meneely. 2016. "The impact of cross-platform development approaches for mobile applications from the user's perspective." *Association for Computing Machinery Journal.* `https://dl.acm.org/doi/abs/10.1145/2993259.2993268`.

Microsoft Corp. 2021. *Xamarin.* `https://dotnet.microsoft.com/apps/xamarin`.