Dedicated Device Deployment Essentials

Developing Applications on Dedicated Devices for End Users

RADIVOJE OSTOJIC
PRINCIPAL SOFTWARE ENGINEER, BRIGHTMARBLES

CONTENTS

As industries continually change and evolve, there is a need for devices that can meet specific requirements, which classic consumer devices are unable to support. Such devices are called dedicated devices and are built based on the required functionality. And similar to consumer devices, they consist of two components — hardware and software — and both have a minimum set of features needed to make the use case work flawlessly.

This Refcard covers the end-to-end essentials of dedicated devices — from considerations in choosing a hardware solution and a development technology stack to processes and approaches for developing and maintaining applications on these devices. In most cases, dedicated devices use the Android OS, so these concepts will be explored using this open-source platform as an example.

WHAT ARE DEDICATED DEVICES?

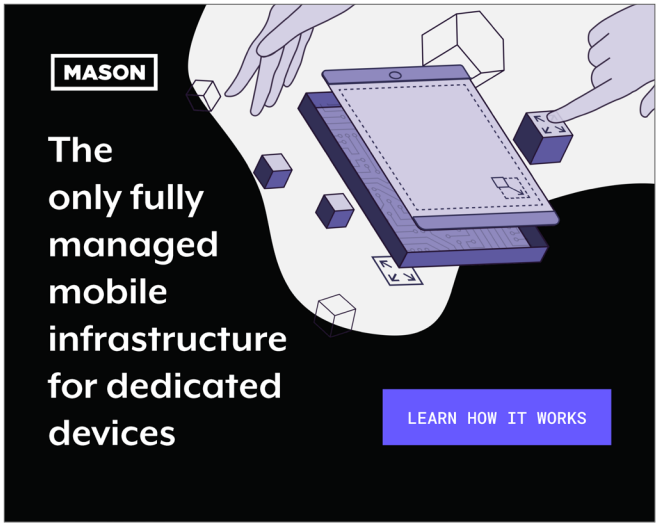
Dedicated devices are also called corporate-owned-single-use (COSU) devices. They represent single-purpose enterprise units, which were developed with a focus on configuration (e.g., display, ports, peripherals). These devices were first widely used during the early 1980s in the retail sector for credit card machines, but today, they are used across various markets — from finance and healthcare to autonomous vehicles and drones.

If we compare dedicated devices with widespread consumer devices, we can see that dedicated devices today can be as powerful as consumer devices. However, they serve a different purpose. Effective dedicated devices give enterprises more control and allow them to design the perfect product and user experience to meet the expectations of their use case. They can also be set up to run multiple tasks simultaneously, although this isn't always the case. When looking at the big picture of dedicated devices, we can conclude that they have several main features:

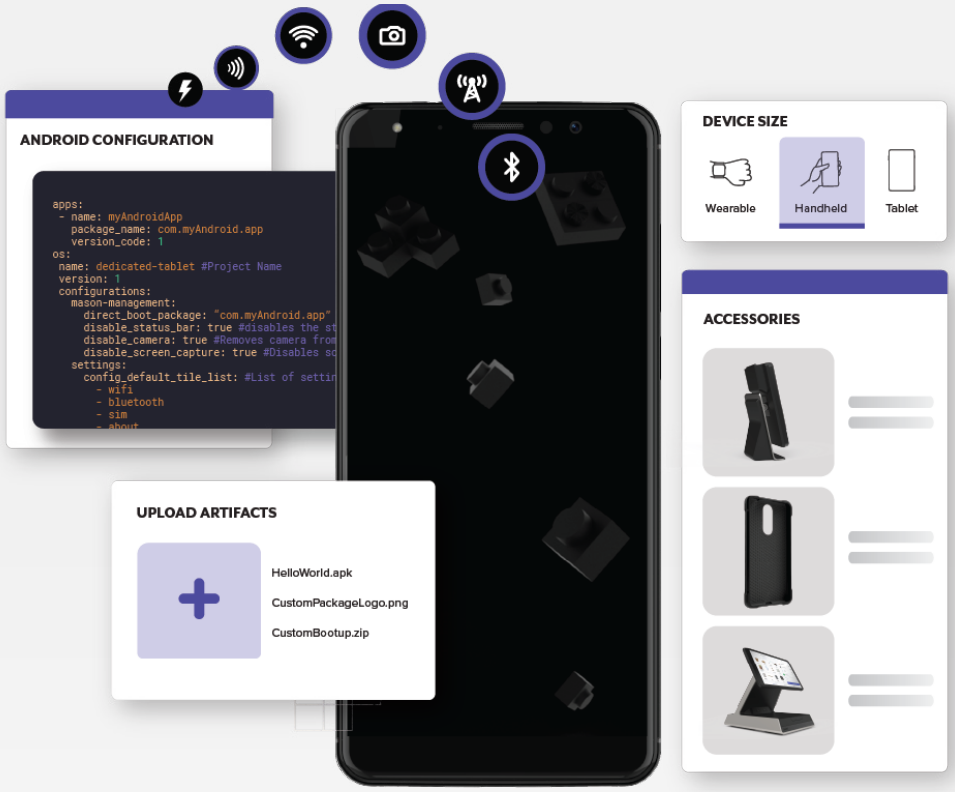
- Owned by the enterprise
- May have one or more applications that are selected by the enterprise and often have restricted interactions, such as running in kiosk mode
- Designed to deliver specific product and user experiences, normally delivering functionality that relates to this goal
- Reliable, secure, and can remain in **always-on** mode

MOBILE DEVICE MANAGEMENT

To make the product complete, it needs to contain hardware, an operating system (OS), and a cloud platform — although it is possible to run devices with no cloud platform, as older providers may still



MASON



Tired Of Having A Fragmented Mobile Infrastructure Stack?

Want Full End To End Control Of Your Dedicated Devices?

Hardware - OS - Fleet Management - Logistics

Sign Up For An Account Today

CREATE AN ACCOUNT

Fast, Fearless Development

Build a custom Android OS in seconds. Use a simple YML file to standardize the OS across your fleet.

Control Through Code

Control every aspect of your development with code — from OS configuration to hardware functionality.

Speed Through Automation

Use automation to decrease time spent testing, releasing and provisioning. Spend your time building great features and products.

Predictable Hardware

Consistent, reliable infrastructure. Stop spending cycles refactoring and testing every time consumer hardware changes.

do. In that case, devices are provisioned and maintained in person, and periodic site visits or other touchpoints are used to monitor it. There is no communication between the device and remote support or IT; however, this is an outdated model that has high personnel and logistical costs with low capabilities to respond to problems or evolving needs. The majority of mobile device management (MDM) platforms do not have hardware. In most cases, hardware is bought separately, and then an MDM is selected to manage these devices. Depending on the OS (e.g., Android, Linux, etc.), it is necessary to create application code that enables adequate use of hardware functionality.

Each of the devices should have predefined application software connected to the enterprise back end and cloud-based device management. There is also a set of well-defined rules by which devices are managed. In addition to being a set of consumer applications with different configurations, back-end infrastructures, certificates, and licenses, MDM has several important functionalities that are crucial for the entire system:

- Remote, real-time analysis, diagnosis, verification, and monitoring of errors
- Regular update of applications, functionalities, etc.
- Clear and secure connections between the various system components

ESSENTIALS TO DEDICATED DEVICE DEPLOYMENT

After the decision to build a dedicated device that is widely available to users, it is necessary to make a few key decisions when it comes to hardware, the application development technology stack, customized device UX, deployment, and remote debugging and logistics. Each of these concepts represents an important segment of the entire system, and if one is selected incorrectly, it can affect the quality of the entire product.

CHOOSING A HARDWARE SOLUTION

Hardware can impact the quality, user experience, and stability of the entire system. There are many factors to consider when choosing the right hardware, including:

- Does the device support all use cases of the project?
- What is the price-performance ratio?
- How easy is it to establish a setup of devices for development and distribution to users?
- Does the device have all the right certifications, if needed, based on industry?
- Most importantly, do these all fit into the project budget?

COTS VS. DIY

Dedicated devices can be divided into two groups: off-the-shelf devices (COTS) and custom devices (DIY). COTS devices are mainly developed by original equipment manufacturers (OEMs), and they are usually available to a larger number of users. Some prominent OEMs are Samsung, Apple, LG, Mason, and others.

DIY devices are custom devices created exclusively depending on the needs and specifications of the project. They may be customized versions of modular off-the-shelf options, or something designed specifically for the purposes of the project. They may rely on an operating system like Android OS or may use only microcontrollers and low-level firmware to manage their processes. Some solutions include Raspberry Pi, Mason, and OEM/original design manufacturer (ODM) catalog devices. DIY or out-of-country options typically require evaluating the device for certifications, checking for regulatory compliance, performing your own quality assurance, etc. Also, import/export laws and the certifications required may change.

SELECTING AN APPLICATION DEVELOPMENT TECHNOLOGY STACK

In most cases, dedicated devices use an Android operating system; therefore, the natural choice for the mobile application stack is Android's SDK. Native Android development requires a basic understanding of how the Android OS works and how the connection between software and hardware is established. When it comes to the language for developing Android applications, Java or Kotlin, among other possible languages, can be used. Java is a proven programming language with many open-source tools and libraries available that can facilitate the development process. Also, most of the Android SDK code is written in Java.

On the other hand, Kotlin is increasingly taking the lead in Android mobile application development. Almost all the latest and greatest Android apps use Kotlin or largely transfer their Java code to Kotlin. Kotlin is designed to improve the flaws and shortcomings of Java and is currently clean, lightweight, and less verbose. The Android team recommends Kotlin for the development of Android applications, and Kotlin is many times the first choice of languages.

KOTLIN VS. JAVA OVERVIEW

FEATURE	KOTLIN	JAVA
Lightweight	✓	X
Null safety	✓	X
Extension functions	✓	X
Smart casts	✓	X
Checked exception	X	✓
Lambda expressions	✓	X

The table above shows some primary offerings of Java and Kotlin. For more details, see how Kotlin compares to Java: https://kotlinlang.org/docs/comparison-to-java.html

In addition to native development, it is possible to use cross-platform technologies — the most represented include Flutter, ReactNative, Xamarin, and the increasingly popular, relatively new Kotlin multi-platform (KMM). This approach makes sense if you need to develop the application with one source code on several platforms, such as on an Android device and on a web platform (this option is offered by Flutter) to save money and development time. The major difference between native and cross-platform development is that native is much safer, has faster performance, and is very responsive, which leads to a generally enhanced user experience. The disadvantages are that it can run only on one platform, the cost of software development is higher, and it requires more developer resources.

In contrast, cross-platform development is far cheaper, the code can be used on more platforms, and in most cases, it requires fewer developer resources. The disadvantages are mainly related to limited device feature support and performance delays. Additionally, features that are not supported in the cross-platform must be developed in the native platform, and this requires the mandatory presence of a native mobile developer on each team.

CUSTOMIZING THE DEVICE UX

The appearance of the application plays a decisive role in what kind of experience the user will have. Often, the user interface (UI) influences whether the user will like the product or not, as well as whether they will continue to use it. Due to these factors, it is crucial that the device offers maximum flexibility and freedom regarding its design.

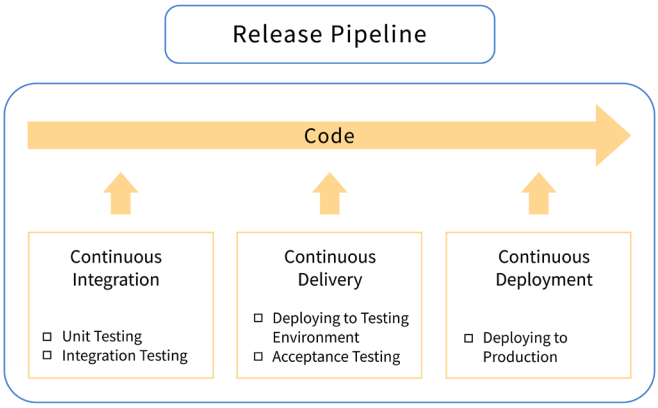
With custom OS development, it is possible to change the entire UI system to meet the project requirements. A concrete example of this is Android Automotive within the Android Open Source Project (AOSP), where a separate UI system was created (Android Automotive System UI) using components and core applications for OEMs, users, and developers. You can explore this example further here: https://source.android.com/devices/automotive

Some OEMs make it easy to configure the base OS of their device through a YML configuration file, starting from the boot-up animation to turning hardware components on and off (such as a camera). This flexibility simplifies the customization process and allows developers who may not have deep system knowledge to easily change the OS.

In addition to this approach, there are EMM/MDM options available, primarily offered by large manufacturers. These providers have devices with predefined UI that can also be changed depending on the needs of the client via their managed services. If, for example, a client wants a specific design for the Android Launcher, modifications can be made in the config file by the provider. A simple use case is a company that provides devices to their employees with a specific login method, custom Android Launcher design, and limited number of applications. Another example is kiosk mode on top of the Android OS layer, which can generally have only one application that supports a single function, or a set of applications that are locked by the IT admin. Also, the UI can be modified inside custom read-only memory (ROMs) using rooting.

CREATING A DEPLOYMENT PIPELINE

Deploying software is one of the most important and complex processes of the SDLC. This step includes selecting the appropriate deployment method, automating the deployment process to reduce the risk of human errors, increasing the deployment speed, and streamlining the process overall. There are several platforms on the market that cover and automate many parts of this process, such as AppCenter, AWS CodeDeploy, and CircleCI, as well as tools that can be used in conjunction with these and integrated into any deployment pipeline or workflow, enabling automation — for example, Mason with Jenkins and Kubernetes.



As shown in the image above, there is a difference between continuous delivery and continuous deployment. Continuous delivery refers to the delivery of code to the testing team, and continuous deployment refers to the deployment of code to the production environment.

REMOTE DEBUGGING AND SUPPORT TOOLS

It is completely common for minor or major errors to occur in production; in extreme situations, it is even possible for the entire system to stop. To avoid such situations, devices running Android OS often offer options for remote debugging and support tools. A dedicated Android device is debugged via the development machine on which the DevTools instance is located.

From there, the content on the Android device is screencast to the DevTools instance. It is then possible to use Team Viewer, Remote ADB tool — which allows developers to connect to the ADB shell

service of an Android dedicated device over the network and execute necessary terminal commands — or a number of enterprise devices with MDM included (e.g., VMware Workspace One). This approach requires interaction on both the client and server side, but there are often devices for which it is not possible to have someone on the client side run the debugging tool. In these situations, some manufacturers offer touchless support and debugging. Such systems themselves can recognize the error or abnormal behavior of the software, and this can be implemented as a separate feature, add-on, or a special application that controls the system.

LOGISTICS

There are different types of operations and logistics services that certain manufacturers and companies offer, including product flow control from the manufacturer to end user, product storage, production, packaging, warehousing, provisioning, certifications, shipping, and security. Depending on the level of service, there are three basic types of logistics for managing dedicated devices:

TYPE	COVERAGE	PURPOSE
In house	All services in house	An in-house logistics team is ready to prepare, provide, and ship software/hardware to end users.
Individual third party	Partial in-house services	Services may include warehousing and in-house IT work but do not involve processes such as shipping software/hardware to the end user.
Full third party	Every process relies on third-party services	All logistics are handled externally, including shipment directly to a fulfillment service without seeing the product.

It is especially important to note that involving a third party may introduce compliance issues as well as reduce visibility into and control over processes. These matters should be weighed if you are considering any level of managed services.

CONCLUSION

The entire ecosystem around the single-purpose Android device, which aims to provide hardware, software, and even logistics, is a unique response to the growing demands of specific industries like healthcare, automotive, and consumer electronics. For these industries, devices must meet a variety of requirements, such as safety, licenses (which can often vary from country to country), an intuitive user experience, and simple modifications of the OS and UX.

Above are the key fundamentals for developing, deploying, and managing single-purpose dedicated devices, offering different levels of service — from full to partial service. The benefits that such devices offer include ease of handling, security, logistics, and an excellent user experience on the client side due to a customizable UX. Although challenges exist, if the concepts explored here are taken into account, any potential setbacks can be limited. Finally, it is important to note that the demand for such devices is increasing, and we can expect the expansion of existing solutions, further innovation across industries, and ultimately, continued growth in the widespread use of dedicated devices in the future.

ADDITIONAL RESOURCES

- Android Open Source Project (AOSP) – https://github.com/aosp-mirror
- Android Basics in Kotlin – https://developer.android.com/kotlin/androidbasics
- Dedicated Devices Cookbook – https://developer.android.com/work/dpc/dedicated-devices/cookbook
- https://solutionsreview.com/mobile-device-management/understanding-the-difference-between-mdm-mam-emm-and-uem/
- https://devops.com

WRITTEN BY RADIVOJE OSTOJIC,
PRINCIPAL SOFTWARE ENGINEER, BRIGHTMARBLES

Radivoje Ostojic is a principal software engineer at BrightMarbles with extensive experience in mobile development, ranging from Android and Kotlin to Flutter and cross-platform development. He has architected complex software systems, led teams, written technical articles, and spoken at conferences. He currently partakes in developing software for some of the most influential start-ups across various industries.

DZone, a Devada Media Property, is the resource software developers, engineers, and architects turn to time and again to learn new skills, solve software development problems, and share their expertise. Every day, hundreds of thousands of developers come to DZone to read about the latest technologies, methodologies, and best practices. That makes DZone the ideal place for developer marketers to build product and brand awareness and drive sales. DZone clients include some of the most innovative technology and tech-enabled companies in the world including Red Hat, Cloud Elements, Sensu, and Sauce Labs.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC 27709

888.678.0399 | 919.678.0300

Copyright © 2021 Devada, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means of electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.