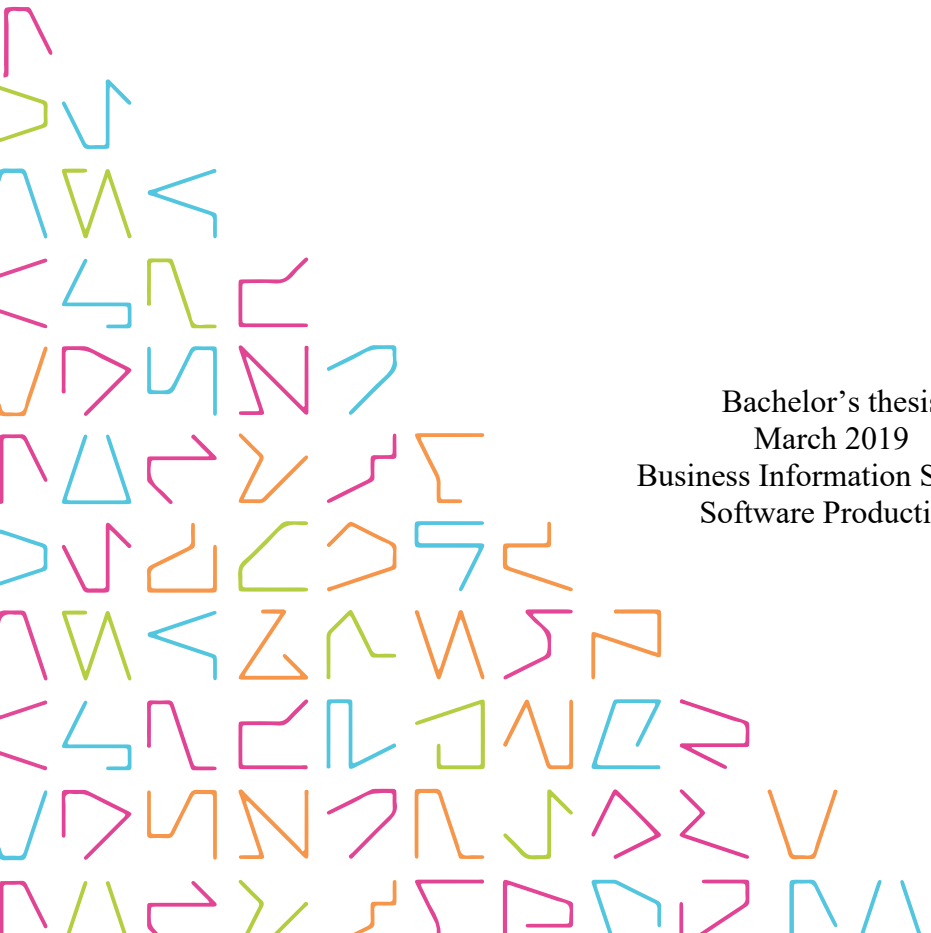
TAMPEREEN
AMMATTIKORKEAKOULU

DYNAMIC BRANDING IN MOBILE APPLICATIONS

Joni Tuominen

Jyri Virtaranta



Bachelor's thesis
March 2019
Business Information Systems
Software Production

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

TUOMINEN, JONI & VIRTARANTA, JYRI:
Dynamic Branding in Mobile Applications

Opinnäytetyö 40 sivua, joista liitteitä 0 sivua
Maaliskuu 2019

Opinnäytetyö työ tutki ja kartoitti mahdollisuutta luoda dynaaminen tyylittely mobiiliapplikaatiolle. Tyylittelyn mahdollisuutta tutkittiin Android- ja iOS-alustoille. Tavoitteena oli kehittää molemmille mobiilialustoille mahdollisimman samankaltainen loppuratkaisu, jota on helppoa käyttää sekä ylläpitää. Tarkoituksena oli luoda menetelmä, joka tarjoaa applikaation loppukäyttäjälle mahdollisuuden vaihtaa applikaation tyyliä ajon aikana.

Opinnäytetyö käsitteli molempien alustojen taustoja ja tarkasteli niiden tarjoamia tyylittelyratkaisuja. Työssä tarkasteltiin kehittäjien tarjoamien mallien lisäksi myös muiden kehittämiä malleja applikaation tyylittelyyn. Opinnäytetyö myös esitteli mallin dynaamiseen tyylittelyyn ja yksinkertaisen prototyypin havainnollistamaan sen toteuttamista.

Selvitystyön seurauksena kehitettiin menetelmä, jolla voidaan vaihtaa applikaation tyyli dynaamisesti. Menetelmää mukaillen toteutettiin ohjelmointikehysprototyyppi, jota voidaan hyödyntää erilaisten tyylittelyjen lataamiseen ja käyttöönottoon applikaatiossa.

Asiasanat: dynamic branding, application styling, mobile application

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business information systems
Software production

TUOMINEN JONI & VIRTARANTA JYRI:
Dynamic Branding in Mobile Applications

Bachelor's thesis 40 pages, appendices 0 pages
March 2019

This bachelor’s degree examined the possibility of creating a dynamic styling for a mobile application without predetermined style settings. This process was examined on both Android and iOS platforms. The goal was to create a solution that was similar as possible on both platforms. Purpose of the similarity was to ease usability and maintainability. Solution were to enable the end-user to change styles during the runtime without requiring multiple versions of the application.

The thesis examined the backgrounds of both platforms and looked at the solutions offered by the platforms for styling. Third-party frameworks for styling the application were also examined. The thesis introduced a model for dynamic styling and a simple prototype to illustrate the implementation.

As a result of the research, a method was developed which allows dynamic style changing for the application. In accordance with the method, a prototype framework was implemented which can be utilized for downloading and applying various styles for the application.

Key words: dynamic branding, application styling, mobile application

SISÄLLYS

ABBREVIATIONS AND TERMS

XML

XML, or Extensible Markup Language, defines a set of rules or guidelines for data that is both human and machine readable. XML is designed to be simple, general and usable across the internet. XML is a string that contains tags and content. Tags normally start with < and end with >. Tag types contain starting tag <string>, ending tag </string> and empty element tag <element/>.

XML files are also used to define styles, values, colors, attributes and other kinds of resources. Components in user interface or layout XML can be assigned to use resources from different XML files containing colors or dimensions for example.

JSON

JSON stands for JavaScript Object Notation and is designed to be a lightweight format for storing and transporting data. JSON is self-describing. Syntax rules are:

- Data is in key/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

(json.org. n.d.)

API

Application Programming Interface (API) is a set of subroutines, communication protocols, and tools for software development. In short, it is a way to get resources or communicate in a predetermined manner. The primary purpose of an API is to make programming easier by abstracting the underlying resource.

UI

User Interface (UI) is a term used to describe the way for humans to interact with computers.

Java

Java is an Object-oriented, class-based programming language that is run on over 3 A billion devices worldwide. According to the TIOBE index of March 2019, Java is the most popular programming language (TIOBE. 2019).

Swift

Swift is a programming language developed by Apple Inc. It's designed to work with Apples Cocoa and Cocoa Touch frameworks, and Objective-C code. It uses the Objective-C runtime library which allows C, Objective-C, C++ and Swift code to run in one program. According to the TIOBE index of March 2019, Swift is the 17th most popular programming language (TIOBE. 2019).

Objective-C

Objective-C is a predecessor to Swift. According to the TIOBE index of March 2019, Objective-C is the 10th most popular programming language (TIOBE. 2019).

Kotlin

Kotlin is a cross-platform general-purpose programming language. It's designed to interoperate fully with Java. Kotlin mainly targets Java Virtual Machine but also compiles to Javascript or native code. Kotlin is sponsored by JetBrains and Google. According to the TIOBE index of March 2019, Kotlin is the 39th most popular programming language (TIOBE. 2019).

IDE

Integrated Development Environment (IDE) is a software application that provides comprehensive tools to software developers. Most modern IDEs have smart code completion, syntax highlighting, automatic linting of supported languages and various debugging tools. IDEs contain tools that are very often used by developers and IDEs try to automate things that are usually done manually.

Runtime

Describes the program state after compilation where the program is running or executing.

APK

Android Package (APK) is a software package used by Android to distribute software.

SDK

Software Development Kit (SDK or devkit) is a set of tools that are used to create a certain kind of application.

URL

Uniform Resource Locator (URL) more commonly known as a web address. Most commonly refer to web pages although they are also used for email, file transfer, database access, and many other things. (World Wide Web Consortium n.d.)

Compiler/Compilation

A compiler translates high-level language into a low-level language — for example, translation of source code into bytecode or machine code.

Inheritance

Inheritance is a mechanism in object-oriented programming where objects or classes are based on other objects or classes. The original is usually referred to as parent object or superclass, while the new one is referred to as child object or subclass. In most cases, the child inherits the behavior and attributes of the parent, with some exceptions.

A common way to picture this is that class Human and class Dog are subclasses of a class Mammal. In this case, both inherit things like sleeping and eating from the Mammal class, but Dogs specifically implement the ability to wagtail that Human does not.

1 INTRODUCTION

1.1 Background

Branding is an essential thing for the recognizability of companies. It usually builds around names or logos of products, companies or people. Brands help us to connect products to companies or product families. Brands can contain different kinds of attributes, such as color themes, shapes and other kinds of styles. They also create a notion of quality for people. This applies to both physical and digital products.

This bachelor's thesis aims to describe research for coming up with an optimal solution to style mobile applications during runtime. One of the goals is to implement a prototype for iOS and Android platforms to manage the brand of the application. Issues, challenges, and suitability of existing methods for styling are examined. Deviation of bases in terms of styling between iOS and Android platforms is one of the focuses of the research. Existing methods include default styling methods for both platforms and alternative third-party libraries if such are discovered. The research aims to discover the most optimal way to implement a prototype for dynamic branding and styling.

The goal of the prototype is to create a method for changing application styling during runtime. The prototype should have as similar functionality and architecture on both platforms. Since it should be usable on as many devices as possible, device independence becomes one of the center points for development and testing. Other critical aspects of the implementation are sustainability, modularity, and the possibility for further development. Since prototype needs to be supported by many devices as possible, operating system versions become critical for research and development.

Client for the prototype is Piceasoft LLC. Piceasoft develops solutions for managing the lifecycle of mobile devices. These solutions are, for example, software to support recycling and reselling of mobile devices. The purpose of this thesis is to produce a branding prototype based on the research to manage the styling of user interface components of the mobile application during runtime. Piceasoft develops applications for iOS and Android platforms. Therefore the development of the prototype is targeted for those platforms as

well. During the research, the goal is to find or produce an optimal solution for both mobile platforms.

Summarily, this thesis aims to discover an effective way for dynamic branding that could be used to change styling for application during runtime. The user should be able to use the branding feature with small effort, and it should not require any technical knowledge. Users should be able to create styling or brand for application matching their own brand.

1.2 Client

Piceasoft LLC is founded 2012 at Tampere. Piceasoft develops software and application solutions to manage the lifecycles of mobile devices. Piceasoft's mainline products are Diagnostics, Switch, Verify, Volume, Report, Trade-In, and Eraser. These products offer ways to ease recycling and reselling of mobile devices. Customers of Piceasoft are companies such as operators and recycling centers.

1.3 Assignment

Assignment and focus of the thesis are to research dynamic branding and produce a prototype that could be used to style an application during runtime. The prototype could be used by clients to apply the brand of their company for styling the application.

The architecture on both iOS and Android platforms should be similar. It should also be implemented in a way, that applying branding does not require different releases of the application.

The application should be able to download branding data from the server using specific arguments. Arguments could contain identifying information about the data.

Users of the application should be apple to use different kinds of parameters to create styles for UI components. Parameters could be key-value pairs that define colors, locations, fonts and backgrounds of UI components. Parameters would be included in branding data that could be passed for the application using JSON format.

In implementation different versions of operating systems should be taken into consideration. Implementation should also be usable on all Android phones and tablets that use minimal SDK required by the application and the prototype.

2 BRANDING

The practice of branding is thought to have begun with the ancient Egyptians who were known to have engaged in livestock branding as early as 2,700 BCE (Wheeler, H., *The Miracle Of Man,* London, Longacre, 1946, p. 84.). This was used to differentiate people's cattle from one another. Along the years the term has evolved to represent a company's values and promises they extend to the consumer.

In the past, when talking about a product or a company, things were quite simple. Chucks nails & hammers made nails and hammers. They might have been the only company in the town, so everyone knew who they were and what they made and most likely you had to buy their product at their workshop. Then things got bigger. Now you might have thousands of similar products that are made on the other side of the planet by someone you have never heard about. These companies might not be able to advertise to you locally, and that's where a brand comes in. Branding is the process of creating a recognizable image — a way for the end user or customer to connect different products in their mind. You as a customer see the same colors and logos, and you start putting one and one together.

Branding is also used to differentiate products from rivals' products in cases where there are set physical requirements for the said product. For instance, two nails might look the same even though their composition might be different. This is where the brand on the package comes in place. Alternatively, in some cases, there might be something like an etched logo on the surface of the product if the packaging is not possible or commonly used.

It is a well-known fact that consumers are willing to pay for a brand. That is to say, the actual product might not differentiate too much from a competitor but if a person has had positive experiences with brand A's other products they assume quality from this brand, even when their other products might be produced in a completely different country by entirely different people from entirely different materials.

Ingredient branding is the process of selling a product based on what it is made of. For example, a company might sell computers, and their brand might not be that well known.

By marketing their products with sentences like "our PCs contain the newest Intel processors" they effectively use Intel brand to sell their products. (Kotler & Pfoertsch 2010)

With software, this sort of Ingredient branding is a bit harder. You don't see anyone selling their product as "made with that one SDK or library that Google provides". With software, we can take branding to a whole new direction. Instead of advertising a product that looks the same for everyone as made with X. We can dynamically change the whole app to use the colors and logos of a particular brand.

3 DYNAMIC BRANDING

The core idea of dynamic branding is the same as the usual branding. Companies want their applications to look like they are connected to the company. The difference is that all the parts that you would use for displaying your brand need to be changeable. Therefore, things like the product shape or logos carved into the product cannot be used. With conventional products, this can be expensive as you might have to repaint the product or remove sticker logos and apply new ones. With digital products such as mobile applications, there are not nearly as many limitations. With a digital product like an application, this means that users can redefine the styling of the application to match their own brand.

The issue arises when an application is developed by some company selling licenses for other companies. If the application is developed for one client only then, there won't be issues, because there is no need to develop multiple user interfaces with different stylings. The issue could be solved by developing different versions of the application with user interfaces that matches the brand of the client company. Maintenance of multiple versions can be time and resource consuming method. It would be much easier to further develop and maintain one version with a feature that could dynamically change the styling of the application. Dynamic branding feature could serve most of the customers without the need to develop multiple versions of the application.

Another issue with branding arises with the default styling methods of iOS and Android. Both platforms require styles to be created before compilation. After compilation resources that are used in the styling of the user interface are read-only resources and they cannot be modified during runtime.

The use case for the dynamic branding could be the following: Company A buys a license to an application and uses this application as part of their services. With dynamic branding, the outlook of this application can be changed, so it matches the company’s other products, while the functionality of the application stays the same. Maintainability and development of possible client specific implementation becomes simpler, if styling functionality is detached from rest of the implementation.

4 MOBILE OPERATING SYSTEMS

4.1 Android

Android is a mobile operating system developed by Google LLC. Android development was started by Android Inc, but Google bought the company in 2005 and has been developing Android since. In the first quarter of 2017, 86.1% of the mobile devices sold were using Android (Gartner 2017). Android has held at least 50% market share every year since 2014 (StatCounter GlobalStats 2019).

Multiple different versions have significantly changed how UI is defined or composed on Android. Mostly this comes in the form of new Views although some old functionality has also been updated at times. This somewhat limits development as some newer functionality might not be available on older versions, and some old functionality might be deprecated on newer versions. The Android API support library somewhat helps as this is designed for cases where maximum API support is needed.

Android is well known for using snacks as code names for different release versions (table 1) (Android Police 2012).

TABLE 1. Android operating system versions

Code Name	Version number	Release Date	API level
(no name)	1.0	2008-09-23	1
Petit Four	1.1	2009-02-09	2
Cupcake	1.5	2009-04-27	3
Donut	1.6	2009-09-15	4
Eclair	2.0 - 2.1	2009-10-26	5 - 7
Froyo	2.2 - 2.2.3	2010-05-20	8
Gingerbread	2.3 - 2.3.7	2010-12-06	9 - 10
Honeycomb	3.0 - 3.2.6	2011-02-22	11 - 13
Ice Cream Sandwich	4.0 - 4.0.4	2011-10-18	14 - 15
Jelly Bean	4.1 - 4.3.1	2012-07-09	16 - 18
Kit Kat	4.4 - 4.4.4	2013-10-31	19 - 20
Lollipop	5.0 - 5.1.1	2014-11-12	21 - 22

Marshmallow	6.0 - 6.0.1	2015-10-05	23
Nougat	7.0 - 7.1.2	2016-08-22	24 - 25
Oreo	8.0 - 8.1	2017-08-21	26 - 27
Pie	9.0	2018-08-06	28

Since Android is open source, it is modifiable by anyone to suit their needs. This kind of openness has given the possibility for Android device manufacturing companies to implement additional features for the Android operating system. Some of these new features may have an impact on the development of user interfaces, for example, mobile devices with a curved screen could come with Android version that has specific API developed by the device manufacturer to handle user interface components of the application that utilizes the curved screen feature.

Android has 169 certified partners that manufacture Android devices (Google LLC n.d.a).

All these certified partners could add additional features to Android that could have some impact on the user interface handling and styling on Android application.

4.2 iOS

iOS (formerly known as iPhone OS) is a mobile operating system developed by Apple Inc. Apple released the first version of iOS in 2007. It has the second highest market share of mobile devices after Android. Of all the devices sold in the first quarter of 2017 13.7% of the devices were using iOS as their operating system (Gartner 2017). iOS held a market share of 22.85% of mobile operating systems in January 2019 (StatCounter GlobalStats 2019).

iOS has 12 major versions to current date (table 2). Some of those versions have changes greatly how the operating system looks and functions. From user interface design standpoint, iOS versions 5 and 7 are important. iOS 5 introduced storyboards for user interface design and iOS 7 changed the whole look and feel of the operating system. Along with major changes of iOS7 to the operating system's user interface, it brings changes to user interface development. iOS 7 changed most of the user interface elements by modifying their properties and functionality. (Napier & Kumar 2014, 51.)

TABLE 2. iOS operating system versions (Lifewire 2019)

Name	Version number	Release date	Support ended (year)
iPhone OS 1	1.0 – 1.1	2007-06-29	2010
iPhone OS 2	2.0 – 2.2	2008-07-11	2011
iPhone OS 3	3.0 – 3.2.	2009-06-17	2012
iOS 4	4.0 – 4.3	2010-06-22	2013
iOS 5	5.0 – 5.1	2011-10-12	2014
iOS 6	6.0 – 6.1	2012-09-19	2015
iOS 7	7.0 – 7.1	2013-09-18	2016
iOS 8	8.0 – 8.4	2014-09-17	n/a
iOS 9	9.0 – 9.3	2015-09-16	n/a
iOS 10	10.0 – 10.3	2016-09-13	n/a
iOS 11	11.0 – 11-4	2017-09-19	n/a
iOS 12	12.0 – 12.1	2018-09-17	n/a

iOS is a closed source operating system, therefore it does not have any version with third-party modifications. iOS development is done only by Apple, and they are also the only manufacturer of iOS devices.

5 STYLING METHODS

5.1 Android

Native Android development is done with Android Studio. Android Studio is the official IDE based on IntelliJ IDEA. In addition to IntelliJ developer tools, Android Studio has other essential features such as mobile device emulator. Android studio uses the Gradle framework as a foundation. This enables you to create multiple APKs or build variants for the application which can be useful if, for instance, you plan on releasing a free version and a paid version of the application (Google LLC n.d.b).

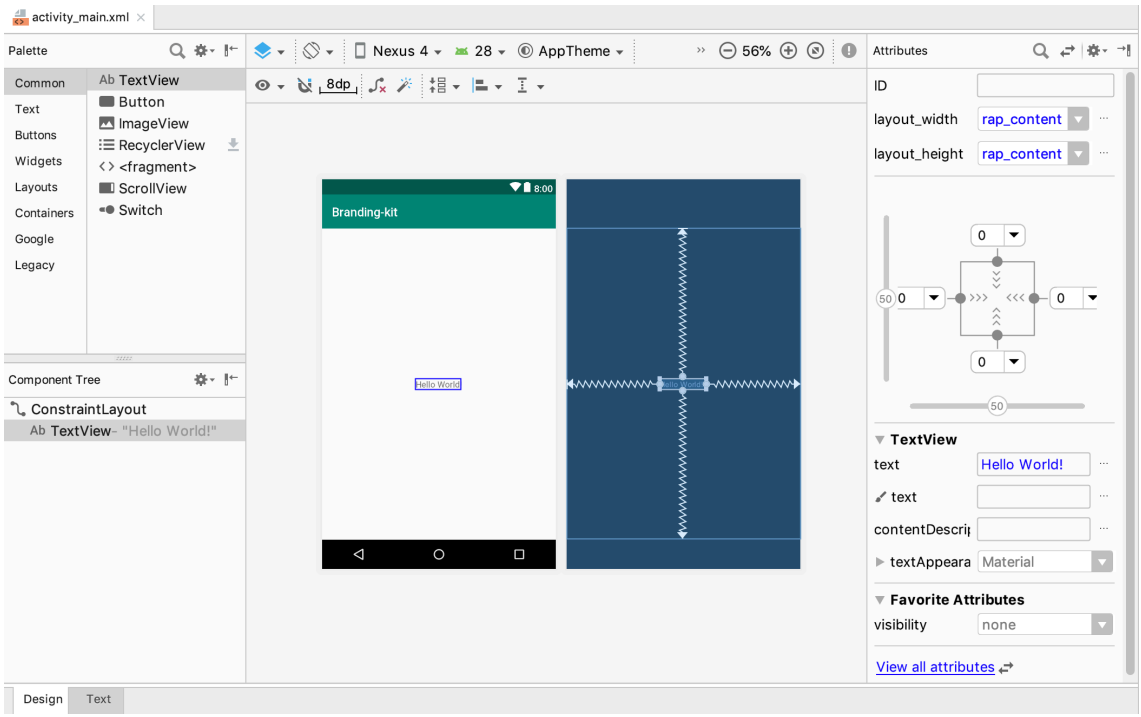


IMAGE 1. Android studio user interface editor

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

IMAGE 2. An XML representation of the view (image 1)

For UI development android studio also has a graphical interface where you can emulate the UI and adjust parts of it just by clicking and dragging with the mouse (image 1). These changes are reflected in the XML files (image 2). This, however, is a very crude way to create UI Layouts as you cannot fine-tune some of the settings.

The second and most popular way of manipulating the UI is by manually adjusting the XML files mentioned above. This gives the developer pixel-perfect precision with the elements and a way to copy paste recurring elements.

A third way would be to create and edit the actual components through the code, this usually is something that happens entirely automatically based on your XML files, but it is also possible to create the UI with pure Java or Kotlin although this would require a lot of completely unnecessary work.

Android SDK provides different kinds of user interface components by the default (Google LLC n.d.c). Such components are, for example, text fields, buttons, and sliders. The developer may create their components by extending the existing component classes or their superclasses. Some values of the properties and attributes of the components can be accessed or mutated via default methods of the class. Component attributes including but not limited to color, shape, and dimensions. Properties vary by the type of the components. These properties and attributes control the behavior and functionality of components. Some of the attributes are assignable and editable via user interface editor.

Android also contains a style and theme system that can be used to style Android applications. These styles are contained in XML files that are created explicitly for styling. A

style is a collection of attributes that specify the appearance for a single view, such as font color, font size, and background color. A theme is a type of style that is applied to the entire app, activity or view hierarchy. (Google LLC n.d.d.)

Attributes mentioned above for styles can also be stored to XML files without creating a style by using a different tag. These attributes include but are not limited to colors, strings, and dimensions. Attributes are usually separated by type into different files. File names for these attributes usually correspond their contents, for example, a file containing colors is named colors.xml, and it contains only entries with color tag. These attributes are referenced from the XML by using unique key assigned to them. Attributes can be used with different kinds of views.

Styles can be set as version-specific by creating a correctly named folder, res/values/styles.xml would be for all versions while res/values-v21/styles.xml would be for API level 21 and up. These version specific styles can inherit from the ones that are available for all versions, and inheritance can reduce duplicate code.

Android developer guide strongly suggests that the developer should extend an existing style when creating new ones. This helps you maintain compatibility with platform UI styles. This can be achieved by using the parent attribute. You should also always inherit core app styles from the Android Support Library to provide compatibility with API level 14 and higher:

```
13      <style name="GreenText" parent="@TextAppearance.AppCompat">
14          <item name="android:textColor">#00FF00</item>
15      </style>
```

IMAGE 3. Android style extending AppCompat TextAppearance style

When extending your styles. You can also use the dot notation:

```
19      <style name="GreenText.Large">
20          <item name="android:textSize">22dp</item>
21      </style>
```

IMAGE 4. Android style extending GreenText style (image 3)

Themes are created in the same way. However, themes and styles are applied differently. Styles are applied to views while themes are applied to either application or activity in

the manifest. From API level 21 forward you can specify a theme attribute to a view. This is then inherited by any child views. (Google LLC n.d.d.)

Styles are limited by the fact that you can only apply one style to a View. You can also apply a TextAppearance attribute that works similarly to a Style. This allows you to use the same text style everywhere while changing other layout styles.

Style attributes can also be applied programmatically, and in most cases, this overrides any style or theme settings. In the case of the developer setting the same attribute in multiple places, the higher row in the list below overrides the lower row.

1. Character or paragraph styling with text Spans to TextView derived classes.
2. Programmatically set attributes.
3. Individual attribute settings in a View.
4. Styles set to a View.
5. Default styles.
6. Themes.
7. Certain View specific styling such as TextAppearance on a TextView.

(Google LLC n.d.e.)

5.2 iOS

iOS development is mostly done by using the Xcode IDE. Xcode is developed by Apple. Xcode is an IDE made for developing applications for Apple's products, that included iPhones, Apple computers, Smartwatches and other devices made by Apple. Xcode offers all the needed tools for development, such as code editor, user interface designers, compiler and all kinds of supporting tools. (Apple Inc. 2019a.)

iOS platform has two official programming languages for development. These languages are Objective-C, and Swift, they are developed by Apple. Using C/C++ on iOS application development is also possible, although using them requires the developer to implement their memory management. Objective-C and Swift objects are managed by Automatic reference counting (ARC), which provides automatic memory management (Apple Inc. 2013; Apple Inc. 2018a).

User interface designer tool uses a simple drag-and-drop model for development. File containing user interface views is called Storyboard which is used as file extension as well. Storyboards can also contain transitions between different views. Ultimately storyboards are XML files that contain views or layouts with their child components and their properties. The compiler compiles these Storyboard files into Nib files that are loaded and instantiated at runtime (Apple Inc. 2018b). (Apple Inc. 2018c.)

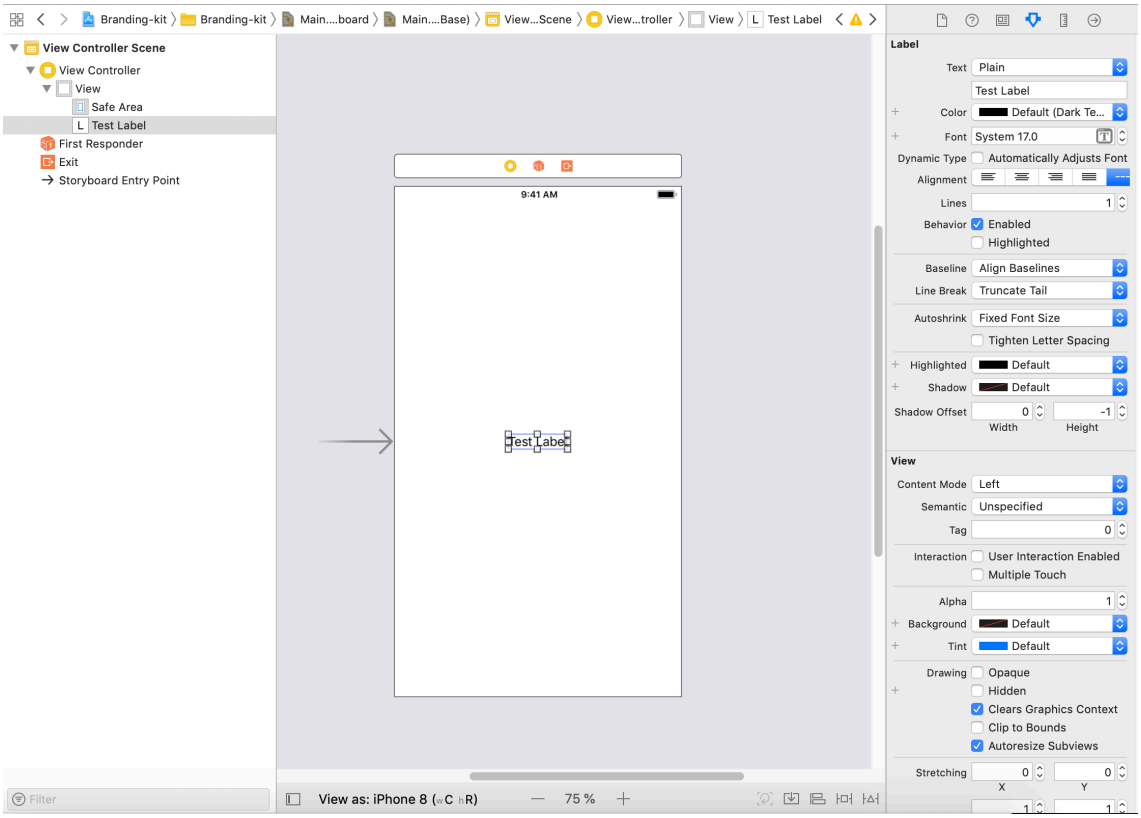


IMAGE 5. Xcode storyboard editor

While editing storyboard XML files is possible, Apple does not seem to provide any tutorials or guides for such development. On archived documentation, Apple recommends using Storyboards as a way to design user interfaces on iOS platform.

View controllers are the foundation of iOS application's user interfaces, and they contain super view and sub-views or user interface components such as buttons and text fields (labels) (Apple Inc. 2019b). They have a corresponding class file that manages the functionality of components within its main view. Various properties of components defined in view controllers can be accessed and manipulated using Storyboard user interface editor.

All UI components offered by Apple's UIKit has a class representation that manages properties and functionality of these components. These components can be accessed or mutated during runtime by using accessor/mutator methods implemented by Apple. Since they are standard Objective-C or Swift classes they can also be extended, giving developer possibility to add their custom functionality and properties for these user interface components.

Storyboard approach differs somewhat from the approach that Android use on styling. On the iOS platform, there are no specific XML files that are designed to contain resources such as colors, dimensions or shapes. Colors, dimensions, and shapes are all manipulated by assigning these properties for UI components via Storyboard editor. Some functionality may require the developer to extend the class and define behavior for component programmatically.

For some resources, there is a way to use resource bundles. However, these resource bundles are intended for resources such as images and sounds (Apple Inc. 2017). Resources from resource bundles can be assigned for UI components by using Storyboard editor.

IMAGE 6. Xcode property list editor view

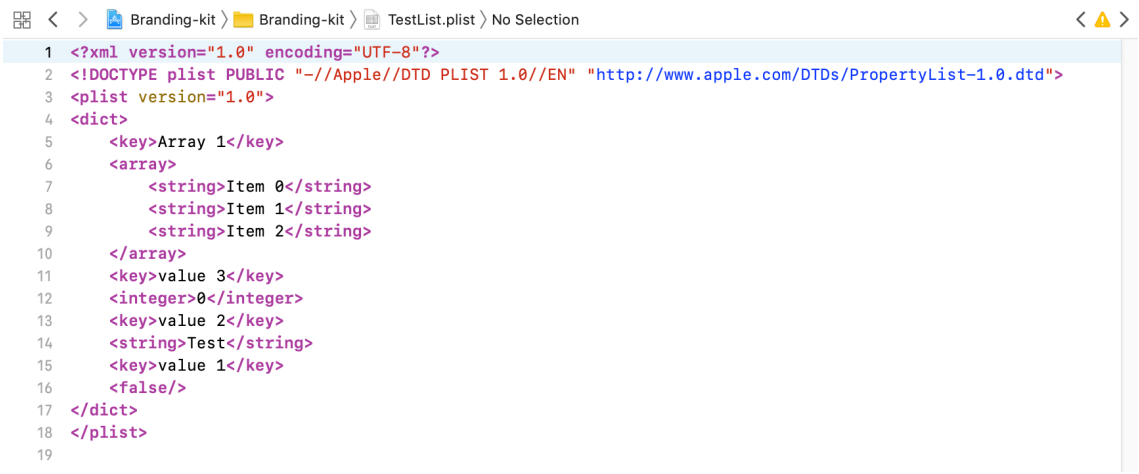


IMAGE 7. Property list XML file contents

Even though iOS does not have specific XML files to store attributes for components, it is possible to create property lists to store values for styling. Property lists can be used to store strings, Booleans, numbers, dictionaries or arrays (Apple Inc. 2018d). Property lists are XML files that can be modified by using editor provided by Xcode IDE. The downside of these property files is that values defined in them cannot be used directly in Storyboard editor. Values in property lists have to be accessed programmatically. Property lists can be mutated during runtime unlike XML attribute files on Android.

5.3 Issues with native methods

Native styling methods for both platforms offer few ways to implement styling for the application. These styling methods do have few issues though when the application should be implemented in a way that offers the user the option to style the application by themselves.

One of the most significant issues are the resources that are used in styling since they cannot be modified after compilation. The developer could define a massive amount of different kinds of resources that the user could use during runtime of the application, but this kind of approach would be extremely time-consuming.

Styling methods do not have natively any functionality to get and store new styling resources for styling. New resources should be accessible from any instance of the application. Getting resources for the application could be done by using an internet connection.

Linking new resources to the application's user interface is also an issue. For example, how to tie the text field to use a specific color resource when the resource is not linked to the component during the development.

Other smaller issues may arise from the differences of the operating system versions or mobile devices. Android and iOS version may have API changes for UI components that could alter their methods behavior. Devices can also have differences in how they behave handle method calls.

6 THIRD PARTY STYLING SOLUTIONS

Third-party frameworks offer an alternative way to style applications. Frameworks have a different approach for styling user interface as the native way has. UIKit Style Sheets or UISS library for iOS utilizes JSON for styling the components (Wijas 2015). A similar approach can be used in the implementation of the dynamic branding prototype. For UISS the JSON can be provided locally or via the internet connection. While UISS provides an extensive and simple way to style iOS application, it has some limitations. One of the limitations in case of dynamic branding is the possibility to create identifiers for the user interface components. This becomes an issue when a single super view needs to have multiple user interface components of the same type with different stylings. There is no way to identify which component uses which style since styling is tied to the type of the component.

```
"UISSDemoFirstViewController": {
    "UIButton": {
        "backgroundImage:normal": ["button-background-normal", [0,10,0,10]],
        "backgroundImage:highlighted": ["button-background-highlighted", [0,10,0,10]],
        "titleColor:normal": ["white", 0.8],
        "titleColor:highlighted": "white",
        "titleEdgeInsets": [1,0,0,0],
        "UILabel": {
            "font": ["Copperplate-Bold", 18]
        }
    }
}
```

IMAGE 8. Styling sample from UISS demo application

As seen in the sample the UISS can have multiple different styles for same type components, but they are differentiated by their super view which in this case is UISSDemoFirstViewController. Therefore, every button used in the super view is drawn with the same attributes. Another issue with the library is that it has no longer active development and it uses a lot of deprecated methods. shouldAutorotateToInterfaceOrientation is one example of these deprecated methods (Apple Inc. 2019c).

Some other frameworks for application styling are Material Design Android Library and NUI. Material Desing Android Library does not explicitly provide functionality to style applications, but it introduces new components to use in the user interface (Navas 2015).

NUI is styling framework for iOS, and it is similar to UISS in the way of defining styling attributes. Styling in NUI is done by using CSS like files (Benner 2015). All of the frameworks and libraries presented in this thesis does not have received commits at least in two years. None the less, they provide a view for different implementation approaches for styling an application.

7 DYNAMIC STYLING METHODS

Since both platforms have their issues with the native styling methods, implementation of application requires an additional implementation to handle certain situations with the styling. Implementation is required to handle branding delivery, brand storing and linking to the component.

There are a few approaches for different parts of the implementation. Brand delivery could be done by using internet connection, a cable connection or with other means of data delivery such as NFC or QR code. While using NFC tags or QR code is a possibility, though it would be limiting due to smaller storage spaces. Internet and cable connections do not have such limitations as the NFC tags, and QR codes have. Delivery could use XML or JSON format for the branding data, as the data could be ideally structured to correspond with properties of the user interface components.

```
{
    "navbar" : {
        "font" : "comic_sans",
        "font_size" : "14px",
        "font_color" : "green",
        "background_color" : "white",
        "height" : "15%",
        "width" : "100%",
        "icon" : "logo.png"
    },
    "view" : {
        "background_color" : "red"
    }
}
```

IMAGE 9. Example JSON for component styling

XML or JSON format could also be used for storing the branding data. Storing the branding data is preferable so that the user does not have to download the branding data for every instance of the application. Java, Kotlin, Objective-c and Swift programming languages used in native development, contains tools to parse both formats.

Dynamic branding requires an object to handle data files, and it needs to check for already downloaded data for every instance of the application. Another requirement is an object that would handle brand data downloading and possible version checks for the data. Implementation requires to have the functionality to download different kinds of resource files such as audio and images in addition to XML or JSON files.

Linking the resources from XML or JSON to user interface components can be done by extending the existing component classes. Extended component classes require a property that can be used to identify the correct resource. For example, if the text field component could have some identifier which would be used to fetch the correct resource from brand data.

An identifying property could be simply coded into the extended class as a string or integer. Since the application can have multiple components of the same type, an extended component class could be extended even more to provide styling for different components with different identifiers. For example, if button class is extended and that class defines its styling with the identifier from the brand data, but the application requires two buttons which will both have a different color, in such case the extended button would be linked to one specific styling if not extended more.

A more dynamic way to implement extended component class would be to define a custom attribute for it, that attribute could be edited in the user interface editor or by editing the XML source file. That class could be used for multiple components by assigning a different identifier for every component. Adding custom attributes for user interface components on both platforms is supported by default. Custom components would inherit all the properties of the superclass, and some of its attributes can be assigned by editing the XML file, custom attributes can also be assigned the same way. On iOS these attributes are IBInspectables, and they have their specific input field accessible via Storyboard editor in Xcode.

Since these extended user interface classes have identifier property for brand data, they can be programmed to get entries from XML or JSON containing the brand data by using identifier property's value as a key. These entries in the brand data could contain colors, icon or sound file names or dimensions that can be used to define the behavior and appearance of the user interface component. Values from brand data can be applied to the component via methods inherited from the superclass.

Some of the components of the application may require special handling that may not be implementable by extending classes. For example, such a component is the status bar on android.

Different versions of Android and iOS may also require alternative approaches to accomplish some of the functionality of dynamic branding.

8 PROTOTYPE

8.1 Overview

Prototype's purpose is to test and demonstrate the dynamic branding methods. The prototype is developed as external branding kit. Branding kit is a library that can be embedded into any mobile application project. Embedding can be done for existing or entirely new projects.

The aim is to create a branding kit that enables the user to apply user-defined brand for the application. Brand data is downloaded from the server using a specific URL with an identifier for the branding data. Branding data can be applied to the application during runtime. The brand can be changed or modified at any time and modifications does not require new versions of the application.

The end user can create their branding for the applications that use branding kit. Branding of the application is based on JSON that the user fills with values of their choice. Brand JSON is then uploaded to the server which provides it for the application. After receiving the JSON, implementation of the user interface components accesses values from the JSON and applies those values on themselves.

Server software implementation is not part of the mobile-end implementation and it will not be documented in this thesis.

8.2 Technology stack

Programming languages used for the implementation are Java and Swift. Java is used in Android application development and Swift for the iOS development. JSON is used to transfer and store branding data.

On Android the branding kit target version is 16 and newer. iOS implementation will be targeted for iOS 9 and newer.

8.3 Architecture

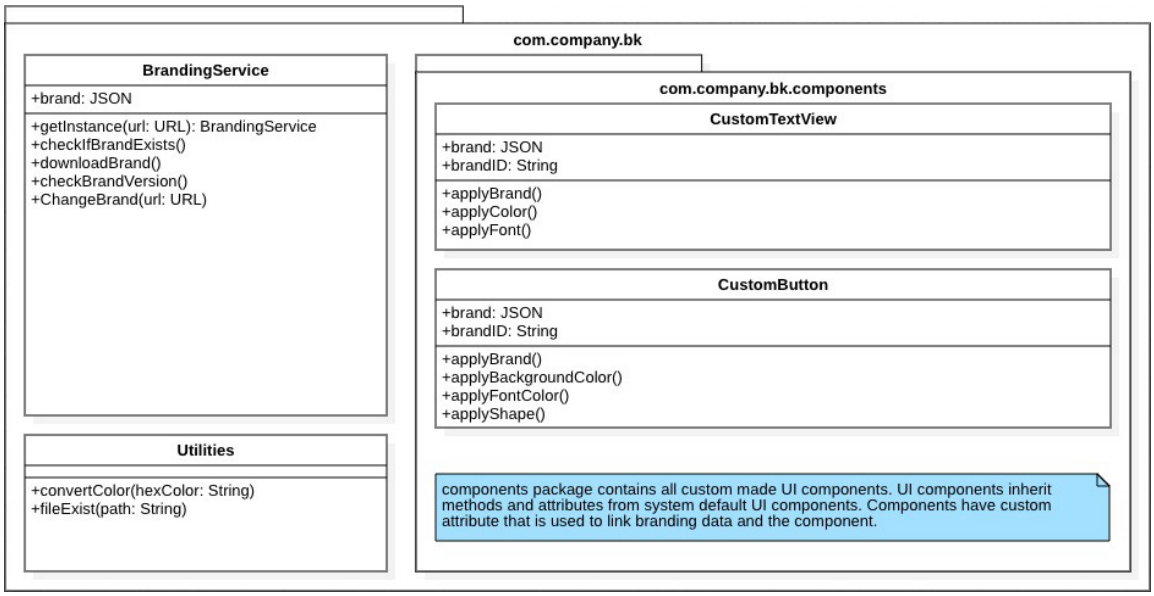


FIGURE 1. Directional architecture for branding kit prototype

Branding kit has BrandingService named class to provide functionality that manages the branding data. It contains methods to download the branding data from the URL, check the version of the branding data and storing the branding data. BrandingService contains an instance of the downloaded brand data JSON. Branding service works as an interface between the developer and the branding kit.

BrandingService uses the singleton pattern. Therefore only one instance of it exists during an instance of the application. Singleton pattern is meant to ensure that there are not multiple different download instances or that the branding kit does not have different multiple different branding data JSON. It also provides simplicity for the component implementation, since all components can get their resources without instantiating another branding service.

Branding kit offers developers a possibility to implement their own brand selection system. This brand selection could be based on user name or predefined brand names. User name can be tied to some specific URL that can be used to download certain brand data from server using BrandingService.

Branding kit will also have Utilities class to provide supporting functionality for other classes. For example, file handling needed by BrandingService is provided by the Utilities

class. Also color conversions for different kinds of user interface components are provided by the Utilities class.

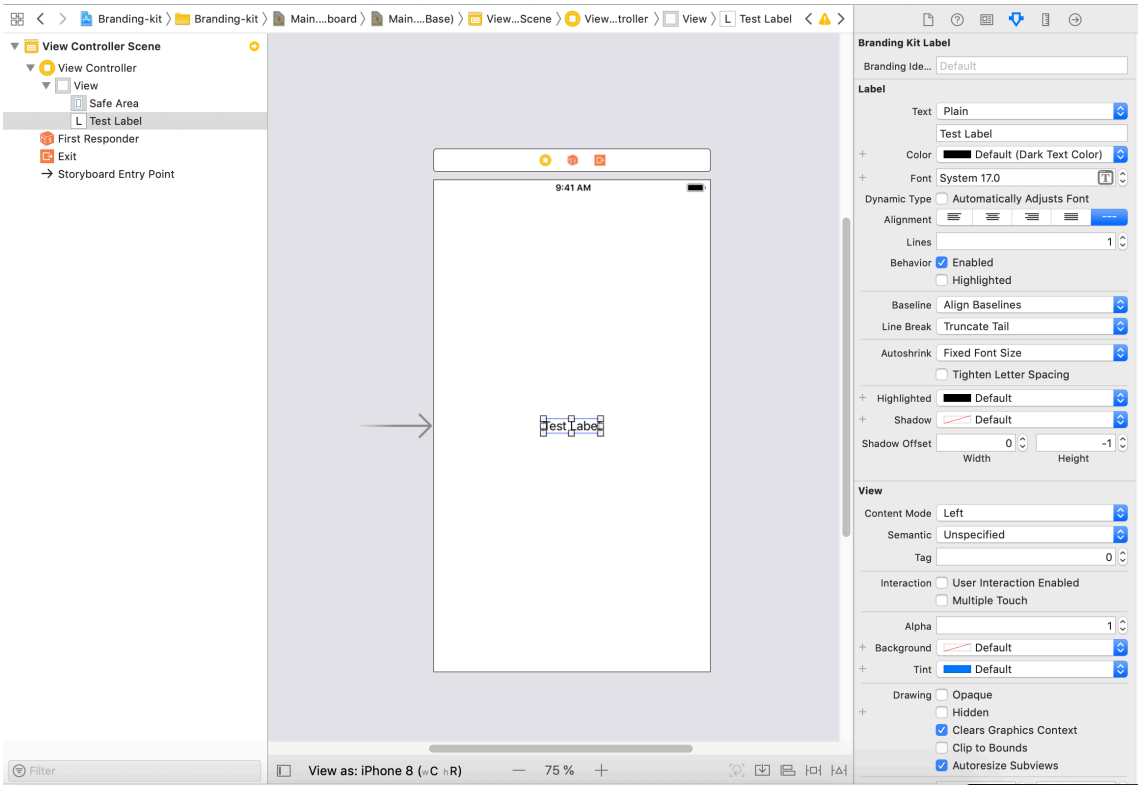


IMAGE 10. Properties of custom UILabel with branding identifier attribute

Custom user interface components extended from the default components provided by the SDKs of the Android and iOS. These components have different support method implementations based on their types and needs. Such methods can, for example, manipulate the dimensions, shape or coloring of the component. Implementation of the support methods ultimate use of the mutator methods of their superclasses. All of the custom components have reference to the branding data which is used to get their specific attributes. Every custom component has an identifier property to identify the component's attributes from the branding data JSON. The identifier is assignable for every component by using a custom attribute in the layout XML file on Android or with IBInspectable in Storyboard on iOS. If no identifier is assigned for the component, then the styling defined in the XML/Storyboard is applied.

```swift
import UIKit

class BrandingKitLabel: UILabel {

    var brandID: String = ""

    @IBInspectable var brandingIdentifier: String = "" {
        didSet {
            brandID = brandingIdentifier
        }
    }

    /**
     * Retrieves brand JSON/Dictionary from branding service
     *
     * Get brand JSON/Dictionary and use the brandID as a key to get
     * attributes for the BrandingKitLabel. Object that is returned
     * for the brandID key can be JSON/Dictionary that contains attributes
     * for the styling of that specific type of component.
     */
    func getBrand() {
        // TODO: Implementation
    }

    /**
     * Applies the brand for the component
     *
     * Get attribute with some attribute key from JSON/Dictionary that was
     * retrieved with brandID and set the attribute for the this class
     * by using methods inherited from UILabel (super).
     * Key for font color attribute could be named as "font_color"
     */
    func applyBrand() {
        // TODO: Implementation
    }

    override func draw(_ rect: CGRect) {
        // Change attributes for component before drawing
        getBrand()
        applyBrand()

        super.draw(rect)
    }
}
```

IMAGE 11. Directional structure of custom UILabel with branding identifier

8.4 Flow

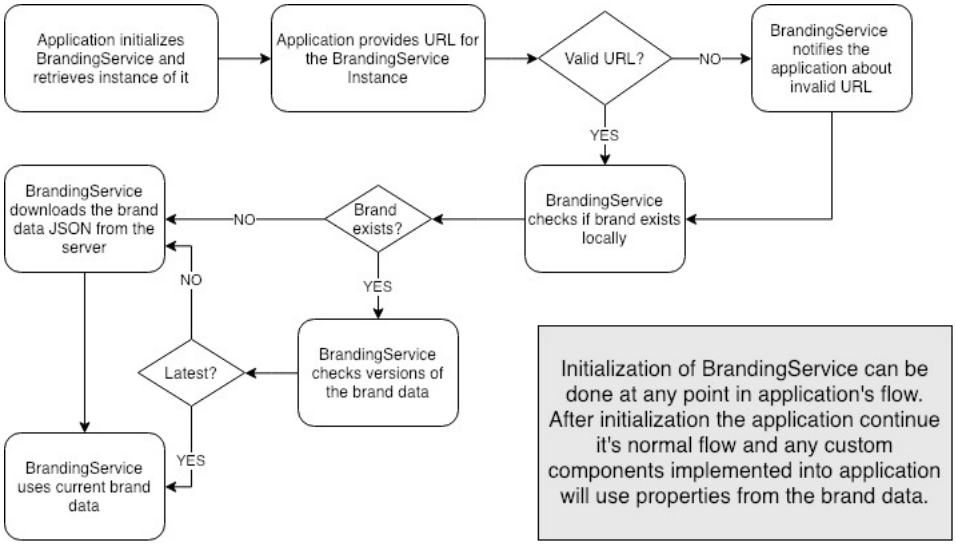


FIGURE 2. Flowchart describing the branding data processing

Branding kit can be taken into use at any time during the application's instance. However, because potentially large resource downloads, it is recommended to create loading activity or view controller and launch the actual application after the download. Download finish event could be implemented by creating and using listener interfaces. Branding service starts to download the branding JSON from the server after it has received a valid URL and has performed a local brand check. If brand exists locally, it performs version validation for the branding JSON to determine if a newer version of that specific brand data is available for download. After the download, the values of the brand data can be used to style components.

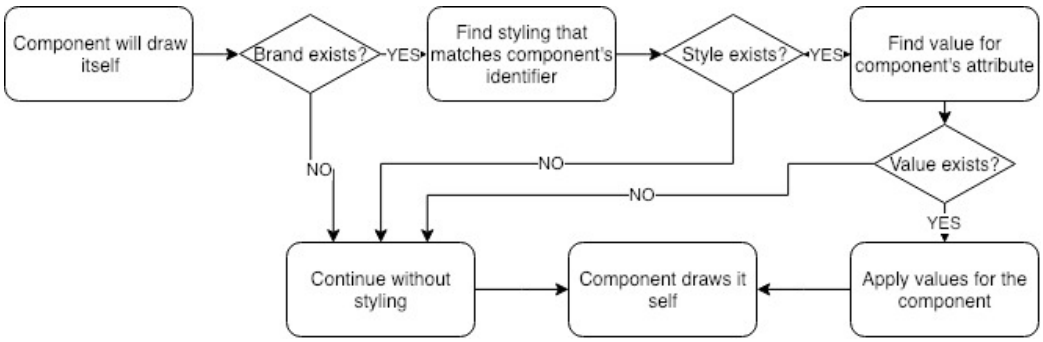


FIGURE 3. Flowchart describing brand data usage for custom component

Components try to automatically apply values from the branding data if any is available for their branding identifiers. If a component does not have a style in the brand data that

matches components identifier or there are no values for the attributes. Then the component draws itself without applying any specific styling. If components have styling implemented in the user interface editor, then the component uses those values if values from the branding data are not available.

If device does not have internet connection, then the branding cannot be downloaded. In such case the branding kit performs the local brand check and if local brand is found then the branding kit applies that branding data.

If developer decides to have branding as an optional feature, the application can be implemented in a way that does not instantiate branding service at all, therefore it will never try to apply any branding data.

8.5 Advantages of the Branding kit

Branding kit provides dynamic styling possibility for the application. Branding kit is usable on Android and iOS platform. Architecturally the implementation is similar for both platforms. Component and JSON approach makes it modular and the modular approach improves maintainability and extensibility. Implementation of the components is fault tolerant since in case of missing values components are draw using their default attributes.

8.6 Issues with the Branding kit

In the absence of an internet connection, the branding kit is not usable, and that may have an impact on user experience. Dependency on server implementation and deployment may prove to be an issue. Server-side maintainability becomes crucial, and any issues with the server-side are reflected in the usability of the branding kit. While the branding kit approach works on most of the user interface components on both platforms, some components require distinctive implementations that does not use the same pattern as the others.

9 CONCLUSION AND DISCUSSION

Dynamic branding for mobile applications is mostly untouched subject, and there is little data that discusses it. Some existing frameworks and libraries have an implementation that could be applied to achieve dynamic branding for the mobile application to some degree. However, these frameworks and libraries are outdated and contain deprecated implementation. They also lack the possibility to handle multiple styles for the same type of user interface components.

While the dynamic branding is not available natively or by third-party frameworks, it still is implementable with native development kits. Such implementation is applicable for some basic attributes of the user interface components. But positioning of these components dynamically would require large-scale implementation which could impact negatively on application performance and user experience. Some of the user interface components like navigation bars require a distinct implementation to handle their styling.

For now, the selection of native user interface components is small. Since there are not many components available, the implementation and maintainability of dynamic branding remain simple. This could, however, change in the future if Apple and Google decide to add more components or change the functionality of components. Additional user interface components are not an issue if their implementation is similar to the existing ones. It would require little effort to add support for the new components if dynamic branding implementation is modular and follows a similar pattern that is used in the branding kit prototype.

Dynamic branding may require downloading of additional resources such as images and sounds, this can grow transferable data amount. If the device relies on mobile data to provide internet access, the data amount may cause additional costs depending on the plan. Support for alternative branding data transfer methods could be implemented in the future to reduce the data amount. Alternative transfer methods could include USB-cable connection, NFC, QR codes or memory cards.

One thing to take into consideration is saving multiple branding data sets to the device memory. On one hand this would lead to require less bandwidth used to download resources like images, but on the other having multiple image sets might take quite a lot of space.

Branding based on JSON opens options to improve user experience and usability. Instead of requiring user to understand mobile application development, a simple picture of the default UI could be offered to the user that could have simple text mapping in the form of: “Label_1_color : white, Label_1_font_size : 14px". These mappings might be limited to only specific options, for instance, too big or small font sizes might cause problems.

In the future, implementation of web application could be created to help user to style the application. Application could display example of user interface with its styling. Application could provide tools to change styling values and those values could be illustrated to the user with changes to the example.

There is also possibility that Apple and Google decide to develop functionality to support dynamic branding, therefore third-party implementations could become obsolete.

REFERENCES

Android Police. 2012. A History of Pre-Cupcake Android Codenames. Accessed 09.02.2019. https://www.androidpolice.com/2012/09/17/a-history-of-pre-cupcake-android-code-names/

Apple Inc. 2013. Apple Developer Archive: Transitioning to ARC Release Notes. Accessed 19.03.2019. https://developer.apple.com/library/archive/releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html#//apple_ref/doc/uid/TP40011226%20luettu%2019.3

Apple Inc. 2017. Apple Developer Archive: Bundle Structures. Accessed 19.03.2019. https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/BundleTypes/BundleTypes.html#//apple_ref/doc/uid/10000123i-CH101-SW1%20luettu%2019.3

Apple Inc. 2018a. Swift Documentation: Automatic Reference Counting. Accessed 19.03.2019. https://docs.swift.org/swift-book/LanguageGuide/AutomaticReferenceCounting.html

Apple Inc. 2018b. Apple Developer Archive: Using Interface Builder. Accessed 01.03.2019. https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/UsingInterfaceBuilder.html

Apple Inc. 2018c. Apple Developer Archive: Nib file. Accessed 01.03.2019. https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/NibFile.html

Apple Inc. 2018d. Apple Developer Archive: About Information Property List Files. Accessed 19.03.2019.

https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html

Apple Inc. 2019a. Apple Developer: Xcode 10. Accessed 19.03.2019. https://developer.apple.com/xcode/

Apple Inc. 2019b. Apple Developer: UIViewController. Accessed 19.03.2019. https://developer.apple.com/documentation/uikit/uiviewcontroller

Apple Inc. 2019c. Apple Developer: shouldAutorotateToInterfaceOrientation. Accessed 19.03.2019. https://developer.apple.com/documentation/uikit/uiviewcontroller/1621459-shouldautorotatetointerfaceorien

Benner. 2015. Github repository: NUI. Accessed 19.03.2019. https://github.com/tombenner/nui

Gartner. 2017. Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017. 23.05.2017. Accessed 08.02.2019. https://www.gartner.com/en/newsroom/press-releases/2017-05-23-gartner-says-worldwide-sales-of-smartphones-grew-9-percent-in-first-quarter-of-2017

Google LLC. n.d.a. Certified Partners. Accessed 09.03.2019. https://www.android.com/certified/partners/

Google LLC. n.d.b. Android Developers: Configure your build. Accessed 19.03.2019. https://developer.android.com/studio/build

Google LLC. n.d.c. Android Developers: User Interface & Navigation. Accessed 19.03.2019. https://developer.android.com/guide/topics/ui

Google LLC. n.d.d. Android Developers: Styles and Themes. Accessed 08.02.2019. https://developer.android.com/guide/topics/ui/look-and-feel/themes

Google LLC. n.d.e. Android Developers: Resource types overview. Accessed 08.02.2019.
https://developer.android.com/guide/topics/resources/available-resources

json.org. n.d. Introducing JSON. Accessed 26.03.2019.
https://www.json.org

Kotler, P. & Pfoertsch, W. 2010. Ingredient Branding. Germany: Springer-Verlag

Lifewire. 2019. The History of iOS, from Version 1.0 to 12.0. Accessed 09.03.2019.
https://www.lifewire.com/ios-versions-4147730

Napier R. & Kumar M. 2014. iOS 7 Programming. Pushing the Limits. West Sussex, PO19 8SQ, United Kingdom: John Wiley & Sons Ltd.

Navas. 2015. Github repository: MaterialDesignLibrary. Accessed 19.03.2019.
https://github.com/navasmdc/MaterialDesignLibrary

StatCounter GlobalStats. 2019. Mobile Operating System Market Share Worldwide. Accessed 08.02.2019.
http://gs.statcounter.com/os-market-share/mobile/worldwide/

TIOBE. 2019. TIOBE Index for March 2019. Accessed 26.03.2019.
https://www.tiobe.com/tiobe-index/

Wijas. 2015. Github repository: UISS. Accessed 19.03.2019.
https://github.com/robertwijas/UISS

World Wide Web Consortium n.d. URL. Accessed 26.03.2019.
https://www.w3.org/Addressing/URL/url-spec.html