|  |
| --- |
| Bachelor Degree Project |

Automating & visualizing tedious tasks within mobile application development pipeline

*- Automating complexities of application development*

|  |
| --- |
| *Author:* Ahmadreza Vakilalroayayi  *Supervisor:* Jesper Andersson  *Semester:* VT/HT 2018  *Subject:* Computer Science |

**Abstract**

Before deploying a mobile application, a complex of manual efforts must be fulfilled excluding the development of the source code [1]. In this thesis, regardless of the mobile application’s code which can be a game, application or e-book, we will investigate how to automate, visualize and simplify the following manual procedures:

1. Modifying/initializing the resource directory in the source code directory.
2. Push all changes to the Git repository simultaneously.
3. Deploythe app from the Git repository after each push.

One of the effort of mobile application development is to generate it’s configurations data together with its graphical contents which is included in the application resource directory. Therefore, Initializing the resource directory itself is really time consuming [2] while there is a possibility to generate, modify and interact with resource directory easier, push its changes automatically into its respective Git repository and finally trigger deployment automatically and simultaneously. In this scenario with changing the resource directory we are actually publishing the new version of the application.

**Keywords:** Automating deployment from GIT, Visualizing resource data of mobile apps, Software architecture, Automating manual efforts, Generate Android resource data, Use of Node.js in mobile app development, Customer understanding.

Preface

Before I begin, I would like to mention that I am indebted to my wife *Nazanin* for supporting me throughout the years in Sweden and I am thankful for her endurance while I was spending most of my time not for her but for my career. Also, I would like to pay my special regards to Linnaeus University and to my supervisor *Jesper Andersson* for all his vital feedback. I wish to express my deepest gratitude to my program coordinator *Ola Flygt and* it is wholeheartedly appreciated that all your great advice for my study proved monumental towards the success of this study.

Contents

[List of figures V](#_Toc1378569943)

[List of tables VI](#_Toc803087692)

[Glossary VII](#_Toc2039962643)

[1 Introduction 1](#_Toc2058763099)

[1.1 Background 1](#_Toc1573397595)

[1.2 Related work 2](#_Toc520675279)

[1.3 Problem formulation 2](#_Toc1919165039)

[1.4 Motivation 2](#_Toc2125437703)

[1.5 Objectives 3](#_Toc1218260835)

[1.6 Scope/Limitation 3](#_Toc425362336)

[1.7 Target group 4](#_Toc1790864743)

[1.8 Outline 4](#_Toc563644281)

[2 Method 5](#_Toc1343118731)

[2.1 Identifying the problem 5](#_Toc1324769509)

[2.1.1 Change graphical contents manually 5](#_Toc833917179)

[2.1.2 Modify configuration files manually 6](#_Toc323498085)

[2.1.3 Perform Git commands and deployment manually 6](#_Toc1413891040)

[2.2 Solutions 7](#_Toc1360740634)

[2.2.1 Edit graphical contents automatically 7](#_Toc965199004)

[2.2.2 Modify configuration files automatically 8](#_Toc1295633387)

[2.2.3 Perform Git commands & deployment automatically 9](#_Toc968120019)

[2.3 Design and development 10](#_Toc923027003)

[2.4 Reliability and Validity 10](#_Toc1665139684)

[3 Implementation and demonstration 11](#_Toc2144648633)

[Current manual approaches 11](#_Toc1251610951)

[Automating manual approaches 11](#_Toc1889584436)

[Server-side principles and modules 12](#_Toc1884024413)

[3.1 Image processing 13](#_Toc1615826912)

[3.1.1 Crop icon launchers automatically 13](#_Toc1435675279)

[3.1.2 Change background automatically 14](#_Toc1696524831)

[3.1.3 Applying shadows automatically 15](#_Toc683133466)

[3.1.4 Apply padding automatically 15](#_Toc666761574)

[3.1.5 Create in-app buttons automatically 16](#_Toc352128875)

[3.1.6 Apply transparency automatically 17](#_Toc575612461)

[3.2 Configuring configuration files using Node.js 17](#_Toc578041025)

[4 Results 22](#_Toc1925526470)

[5 Discussion & Analysis 23](#_Toc1096287741)

[6 Conclusion & future work 27](#_Toc349722416)

[References 29](#_Toc1903480525)

**List of figures**

**List of tables**

# **Glossary**

|  |  |
| --- | --- |
| Resource directory |  |
| Source code |  |
| **GIT** |  |
|  |  |
|  |  |
|  |  |

Note that all the variable names, file names and directory names mentioned in this project are just for demonstration and is used only as examples and has no affect on the authenticity of this project.

# **1 [Introduction](#_1 Introduction)**

In order to develop a mobile application from scratch, a software developer has to perform numerous tasks. Beside developing the source code, it is mandatory to have some resources available locally on the mobile device. A mobile device is a computer small enough to hold and operate on the go which should also run an application if we go offline. An application can run offline only if we have all required data that the app needs on our mobile device storage which is called our resource data. Resources are all unique data, values and configurations that our source code uses to initialize application modules, variables and graphical interfaces. These data will specify the designs and configurations of a mobile application and if we manipulate these data, we might have changed the application functionality or its graphical contents.

To change these configurations, the developer of the application must modify the resource data and publish it by deploying a new version Therefore, deploying new version of a mobile app with even a minor change require repetitive and time consuming manual efforts to adjust resource data, update Git repository of the mobile app and deploy it again. Example of such changes are, changing the font, the icon or even a simple color in the mobile application. The aim of this report is to investigate possibilities that can bypass manual tasks and how deep we can automate time consuming approaches in developing and deploying a mobile applications.

## **1.1 Background**

This project is representing a software architecture that can provide variability and flexibility. To design this architecture, we need to plan a software system that can give us the relevant requested output which is the deployed mobile application.

Software architecture is strategic planning with design methodologies that can be used to create or modify software products [3]. The preferred architecture in this report should corresponds to manual tasks but in an automated way. For example, modifying many configuration files (e.g., passwords, dynamic values, cloud URL, hexadecimal values, API keys, and paths, etc.). Also, automating the design process or user interface of the applications appearances can be quite challenging if we want to use image processing [4] to produce appearances (e.g., app icons for Android & iOS, notification icons, in-app colors, transparencies, and buttons, etc.).

The complexity of this project will take place when we need to combine all manual processes as a single automated software system with both server and client side which will automate the creating of a resource directory and once the resource directory is updated or initialized, all changes will automatically pushed to its Git repository using shell scripts, and finally deploy the mobile app from the Git repository using a CI/CD tool.

## **1.2 Related work**

The app resource overview of Android documentation [2] is related to this thesis except automating its Git and deployments feature. As indicated in Android documentation, all manual works to create a resource folder for application are categorized and discussed. All values in resource folder will be used by source codes, such as bitmaps, layout definitions, user interface strings, animation instructions, and more. The documentation is related to several parts of this project except it is only discussed problems and solved them manually which needs manual effort to be accomplished.

The other related work to this project is producing appearances for mobile application by using image processing in our model. To create the resource folder for our apps, we will require preparing at least application icons or ic\_launcher for both Android and IOS. This procedure has been automated as a web application by Roman Nurik [9] the front-end engineer on Firebase at Google, and his application is called "Launcher Icon Generator" [10].

In January 2015 an article was published [5] with the title of “Continuous Delivery” about the difficulty and challenges in continues delivery but also the benefits of this software engineering approach. This approach can be used by many companies since customers may order many applications within a short time and ask for changes repeatedly in future. As an example, this can be done with automated deployment of Bitrise [**24ref**]. Bitrise is a Continuous Integration and Continuous Delivery (CI/CD) Platform as a Service (PaaS) with a main focus on mobile application development (iOS, Android, React Native, Flutter). It is a collection of tools and services to help us with the automation of our software projects which we will use it as a part of this project.

## **1.3 Problem formulation**

We want to investigate how to automate all required manual tasks into an integrated pipeline or a single understandable activity that can communicate with all end-users. If we consider the human involvement into manual cycles, then we need to investigate a possible way on how to combine all these cycles into a single automated sequence with flexibility behavior. So, how we can automatically create the resource directory as user desires? How we can automatically push the changes to Git repository after modifying the resource directory? And finally, how to automatically deploy a new version of the mobile application when changes has been pushed?

## **1.4 Motivation**

Automating manual tasks itself can be tough and time taking. There are lots of details that needs consideration and lots of algorithms that we need to use instead of manual efforts. Although, to automate every manual task we might put forth substantial efforts but on the other side it take effort once for all to develop from the scratch and it will eliminate all tedious efforts in the future. Automating can be done differently depending on our demand and how deep we need to perform the automation. Therefore, our demand in this thesis is to automate the following manual tasks:

1. Creating resource directory with its sub directories [21]
2. Create all relevant appearances [2]
3. Create all configuration files that the source code uses [2]
4. Place all data into their appropriate directories where the developer has addressed within the source code [2]
5. Push the project into its Git repository and finally deploy the application [22]

With designing a new flexible generic model and by applying the model into the current system we can ensure that the variability of the model can take away all repetitive tasks which can be also supportive for unknown future expectations with an output of a deployable product.

With this project, we will see how we can unify all cycles as a single adaptive software system that can get some input from a user interface and produce all relevant configurations, appearances, and directories depending on any expectations in the server side. The time that a developer needs to spend on producing source code will be saved for other duties and they no longer need to use several applications to modify appearances or to modify any configuration files. On the other hand, this idea can be used in any software development company while the cost of hiring a developer will be more efficient since the software developer can spend more time on developing a software instead of reskinning or publishing an application.

## **1.5 Objectives**

In this section we have categorized our objectives that need to be achieved.

|  |  |
| --- | --- |
| **O1** | **Problem identification**, define the current method/solution that is used in updating and deploying a mobile application |
| **O2** | **Solutions**, develop a method that meets the requirements |
| **O3** | **Design & development**, use related works to provide a conceptual basis for all requirements and design the relevant architecture |
| **O4** | **The demonstration**,use prototyping and programming toprovide a proof-of-conceptby implementing the expected software architecture. |

At the end of this project, the expected result will be a general solution within software development industries. Except for object 4 or demonstration which is the implementing the work in a specific company with specific requirements.

## **1.6 Scope/Limitation**

The intention of writing this thesis is to automate and simplify software development pipeline. The size and complexity of a pipeline can be varied in several companies, and this variation can cause limitation within thesis work and is wholly related to kind of time consuming tasks that should be accomplished in order to deploy a mobile application. For this matter, there should be tons of different pipelines with their different subsequent tasks that can be automated as algorithms.

After starting the project we realized that the current project can be also generally used within software development industries. This idea convert certain and specified manual activities into automated algorithms. By visiting company Infomaker Scandinavia AB, we have discussed their situation and we came up with many manual tasks that are required to be done in order to publish an application manually. These manual tasks are totally different in each company and of course some companies would not let their sensitive information due to the risk of data breach. So, because of this matter, we failed to get information from other companies to evaluate them in this section.

In software development companies there are pipelines that used to accomplish tasks [23] and we are only focusing on automating a pipeline which is a general way for all companies that develop software. In order to use this idea in other companies we first need to interview developers at that specified company, second evaluate their current pipeline which they are using to develop software, and third we need to automate their tasks as much as possible and integrate them together as a software with converting most of the manual duties into automated algorithms.

Although we planned to automate manual procedures, Git integration [11] and deployment process [12] that is uniquely requested by a company, but in some situation it might be possible if a company asks for more features which requires more time for interviews, demonstrations, discussions and feedbacks that might exceed the scope and of course the time limit of this thesis project.

## **1.7 Target group**

The primary target group are software development companies that produce or update multiple applications in less time, especially those companies that get reskinning orders from customers. Customers play the most critical role in the industry [14] and the more they order, the company need to do more development and they definitely need more software developers [13].

## **1.8 Outline**

Section 2 will identify the problem and then we chose a method to answer the problem. In section 3 implementation and demonstration, we demonstrate the idea. In section 4 results, we discuss several solutions and challenges. In section 5 discussion and analysis, we discuss the output of our result. In section 6 conclusion and future work, we discuss the conclusion and what can we do if we had more time.

# **2 Method**

We have recognized manual tasks as the major problem and to solve the problem, we need to plan and perform our objectives. There are three essential steps to solve the problem and also to provide reliable answers we need to demonstrate the solution by prototyping our method which takes place as the last or fourth objective. In 2.1 we identify the problem and in its subsections we provide each problem with its answers. Furthermore, in 2.2 we have discussed these problems as challenges.

* 1. **Identifying the problem**

The problem is identified as performing many time taking and tedious manual tasks that need to be done before the application is ready for delivery. These time taking efforts are caused by three problems which we discussed in 2.1.1, 2.1.2 and 2.1.3.

**2.1.1 Change graphical contents manually**

This is a general problem for any user when it comes to edit an image, and basically, image editing can be done with some applications or software which also these software use image processing to modify an image. The company is currently using several tools to design the appearance of an application, such as Adobe Photoshop or GIMP which are tools that can give transparency to images, crop images, resize images and other related tasks to edit an image. Some of these applications also need a license to be activated which are also costly. This image editing is a routine task that the company has to accomplish every day to reskin a mobile application for customers who intend to have a new user interface. This problem is causing waiting time and takes lots of efforts which can be solved with an application that can accomplish the same routine algorithms.

As we discussed, the image must be opened and modified with a raster graphics editor such as GIMP or Adobe Photoshop. So, how we can edit an image without using the mentioned editors? The answer to this problem is to use programming and image processing instead of using other software and applications. With image processing, even without opening an image we can edit and save our changes. No matter what changes should be done, the image can be stayed unopened and our changes will applied once we pass the image as a parameter to our program. In this situation, the estimated time to edit an image or icon can be also decreased if we pass what needs to be changed to our code as a parameter together with the path of the target image or a directory. An example of such is to make hundreds of icon launchers darker or brighter very fast at the same time. In different circumstances when user needs to uniquely edit each image the system should provide a GUI environment that can be interacted for such purpose. With integrating an image processing module to our software architecture our system will be supportive to accept an image and ask the user to apply changes by providing a list of filters, various cropping sizes, and several filters.

**2.1.2 Modify configuration files manually**

The other general and time consuming problem is creating many configuration files. A configuration file can be opened and modified by an editor such as Vim or Notepad++. A mobile application has many configuration files that our source code uses to function and there are many configuration files in the resource directory of an application. These configuration files need to be updated every time that the customer request changes or even when the company wants to start developing an application. In this case many configuration files need to be changed and saved or initialized in their exact related path by text editor which is too time consuming. So, how we can initiate many configuration files in their related paths in less than few seconds? Or how we can update them in less time? How we can understand if our inserted value is incorrect while we use text editors? Or how we can validate them?

In order to initialize a resource directory we can use a template resource folder with all required subdirectories and generate it whenever the company needs to create a new mobile application. This resource directory has already all the configuration file with temp values and can be changed. In order to change all configuration files in already existing resource directory is to fetch all configuration files from all paths for the end users and present them as edited or not edited, also there are values in configuration files which not need to be changed and can be used for long time, such as some constant URL or cloud credentials.

Without using an accurate algorithm to produce configuration files, if an employee attends to modify many configuration files in short time manually, she/he might make a mistake in inserting values because there is no validation process. See *Figure 3.10* which show a configuration file that consists of many attributes. The validation of user input will help to avoid deploying a misbehaving mobile application.

**2.1.3 Perform Git commands and deployment manually**

At the time that the application and the resource directory is ready for publishing, developers should have all changes pushed to the Git repository to make the app ready for customer satisfaction and then the deployment process. Before the deployment, they provide a test or pre-deployed version to the customer to see if they experience any difficulty working with their application and if the customer found it undesirable, the company have to redo the "problem 1" and/or "problem 2" to provide another test version again until the customer become satisfied from the outcome. So, how is it possible to also involve customers in this pipeline to avoid reproducing Problem 1 and 2? From the graphical modification perspective, is it possible to let customers to choose their in-app graphical desires and deploy the application themselves by updating the resource directory?

In this step, besides having trouble of deploying an application, we also have to make the customer satisfied which instead we can provide them a tool that help them choose graphical properties for their own application and deploy it. We have discussed a tool that can update the resource directory, such as configuration file and graphical contents and if the company and customer can use that tool, then how we can deploy the application once the resource directory is updated?

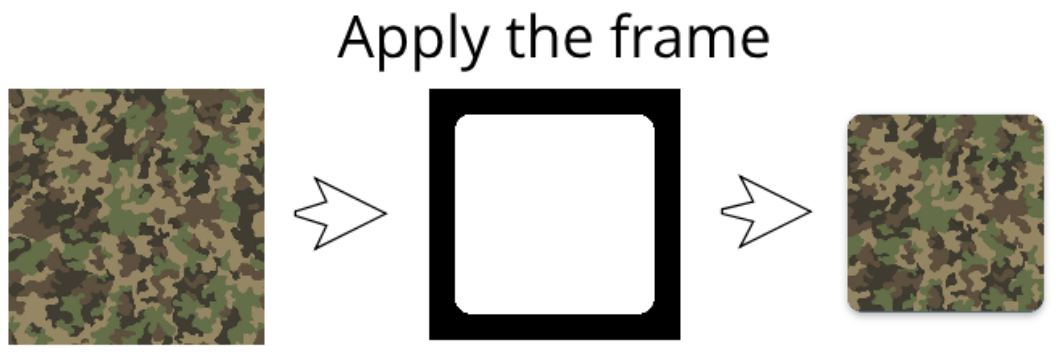
The above three problems are problems within the current pipeline, and these are creating a major issue which is too time-consuming and costly for the company. In the next sections we will go through each problem to discuss the solution respectively.

* 1. **Solutions**

A software developer should be able to design app icon and modify all configuration files at the same time of developing the source code and finally the app should be deployed after each changes. All of these efforts can be done with an application. Therefore, instead of manual effort this application can solve all the problems in 2.1.1, 2.1.2 and 2.1.3.

**2.2.1 Edit graphical contents automatically**

With digital image processing we can crop images, resize images, apply shadows, change colors, apply transparency, mask an image, cut an image, and tons of other options. As an example, we can use image processing to round the corner of an icon launcher. In order to solve this problem we use an algorithm on the client side and this can be also done with midpoint circle algorithm [20]. The other solution is to use frames and image masking, in this scenario the application will use an algorithm to read the frame's pixels and on the other side mask the main icon based on the pixels that occur in the frame as shown in *Figure 2.1.*

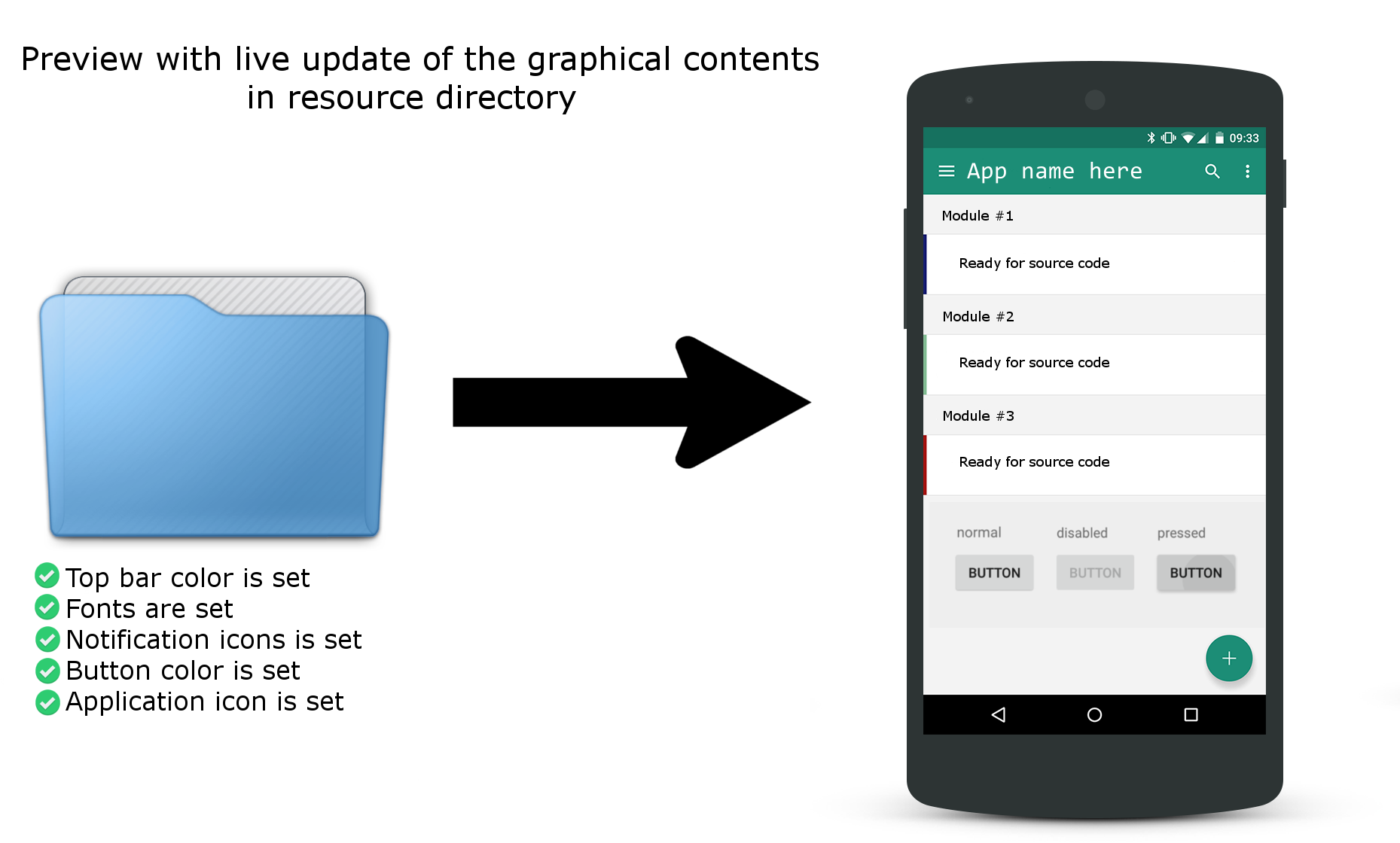
*Figure 2.1 – Cropping images with frames*

Once the image is cropped, we require to have the largest icon launcher which is **mipmap-xxxhdpi** and also the smallest icon launcher which is **mipmap-hdpi** in the resource directory. Refer to *Figure 3.4* for the other cropped icons and see *Figure 3.2* for the belonging directory for each size.

**2.2.2 Modify configuration files automatically**

The other challenging part is to initialize or modify a configuration file in its belonging path that is addressed from the source code. In order to solve this problem, we need a template resource directory that is most familiar to all applications resource directories. We have investigated and found the most common resource directory that is used in the most of the mobile applications and kept it in the server as the primary sample. The mentioned directory has the most useful configuration files which can be later edited within the scope of our automation system.

Whenever a user wants to initialize or modify a resource data for the mobile application, a new copy of the template resource directory which is located on the server will be provided to the user for modification. Therefore, this automation will iterate through all the configuration files to provide them to the end-user. This also applies when a user wants to change the graphical contents. Further more, the emulator will show the changes on the virtual device. Example of such emulator is Android Emulator which simulates Android devices on the screen so we can test our applications and interact with its features. When the resource directory is ready we can go one step further to deploy the application.



*Figure 2.2 – Emulator views the changes of resource data*

**2.2.3 Perform Git commands & deployment automatically**

In order to begin with automating the deployment, first we have to automate the Git commands. Therefore, Git add, Git commit, and Git push should trigger automatically once the resource directory is changed. Once the Git repository is updated, our CI/CD tool will publish the new application from its Git repository.

* 1. **Design and development**

As we defined our solutions in the previous section, we need to write algorithms that can execute and apply the expected solutions. Together these algorithms design an architecture that can accomplish the challenges in 2.2.1, 2.2.2, and 2.2.3. Therefore, we decided to manipulate each functionality in a web application that can perform all the challenges and be approachable for remote access. In this case both the developer and the customer has the ability to modify the resource data and publish the mobile application.

This web application can be counted as a positive value from the perspective of the customer which is a customer understanding and is a good business perspective and planning [19]. Customer version should be asynchronous client-side web applications that fetch all required data from the allocated resource directory on the server and represent them at the front-end. The customer can have access to the front-end web application while the developer has access to both front-end and back-end. On the server side, the developer prepares a template resource folder for the specific customer's mobile application.

## **2.4 Reliability and Validity**

For few years the usage of automation is increased in IT industries. With the use of automation in software development industries, we can create applications faster, deploy our application straightly to production, by pass manual testing of application, maximize code coverage and minimize the code reviewing duration. Mozilla is a good example of a company that uses continues delivery and automated deployment which they say for most of their projects “once a code change lands on a master branch it is shepherded to production with little-to-no human intervention”[25]. Although the outcome of an application that uses automation is much reliable than rest of the manual development techniques but in today’s technology we assume it as a good approach which might not be true in the future as technology is growing.

# **3 Implementation and demonstration**

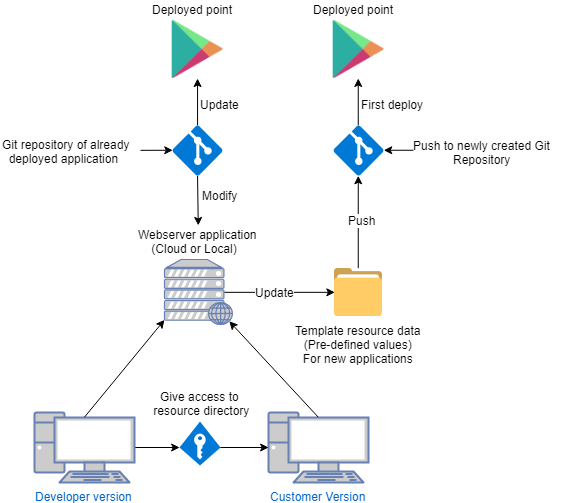
In the previous chapter we have discussed 3 problems and to solve these problems we have discussed 3 challenges involved in each problem. This chapter will implement and demonstrate the solutions by doing these challenges in more details.

**Current manual approaches**

Manual solution to deploy a mobile application follows many manual tasks that a developer has to do repeatedly such as edit graphical contents and/or configuration data and finally deploy the mobile application after changes pushed to its related Git repository.

**Automating manual approaches**

With programming we are able to automate manual tasks that are time consuming, such as edit graphical and/or configuration data if needed in a visualized environment which can be a web application with both back-end, front-end and integrated Git commands to push changes automatically as shown in Figure 3.1.



*Figure 3.1 – The expected architecture*

**Back-end application**

To manipulate the resource data of a mobile application we must execute codes on the server side and the first step toward this objective is to choose a framework such as NodeJS and also we need shell scripts to automate terminal commands such as Git commands and bash commands in particular.

**Front-end application**

The front-end application is used to send requests to the server and to receive responds. For instance, a common respond is a JSON data consist of what needs to be changed in the resource directory of a mobile application and a request is the list of available resource directories that are located on the server.

**Bidirectional communication method**

The back-end application should establish a connection to the front-end web application in order to send and receive request and responds. Accordingly we have to choose a method of communication between the server and the browser such as Web Sockets which enables bidirectional, event-based and real-time communication between the server and the web application. All requests to the server are meant to fulfill some manual efforts and on the server side these requests are handled with implementation of the following functionality in this chapter.

**Running the web application**

To run our web application we must make sure that we have [NodeJS](https/nodejs.org) and node package manager or [npm](https://www.npmjs.com) installed on our system. Once installed, we verify the installation using node --version and npm --version commands. Basically, we run our web application with running the server and also the client side service. To run the server and client side we use the following commands respectively once we navigate to the root directory of the web application. Since the application change the files and folders on the storage we suggest running the commands as administrator privilege.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Linux | Windows | Description |
| 1 | ~$ sudo npm install | $ npm install | Install dependencies |
| 2 | ~$ sudo node server.js | $ node server.js | Run server |
| 3 | ~$ npm start | $ npm start | Run client side application |

We assume that we have cloned our mobile application repositories from Git in the path **Cloned\_Git\_Repositories** on the server side. After running both server and client side web application server will read the sub directories of the mentioned location and fetch all the names of available repositories and send them to client side web application. The following code snipped shows how we can send such data on successful connection.

**const fs = require('fs')**

**io.sockets.on('connection', function(socket){**

**const clonedResources = './Cloned\_Git\_Repositories/'**

**fs.readdir(clonedResources , (err, files) => {  
 socket.emit('Repositories', files)  
})**

**})**

On the client we receive repository names and it’s time to add them to our <select> element on the fly using DOM. Here is an example of using HTML DOM to manipulate our <select> element.

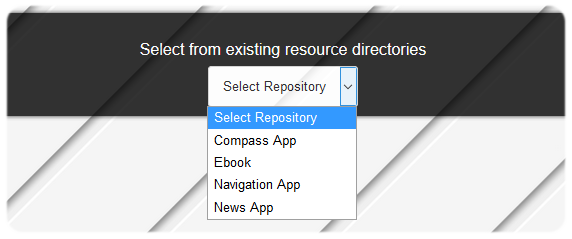
**<select id="repositories"></select>**

**<script>**

socket.on('Repositories', function (data){  
 var repoList = document.getElementById("repositories")  
 var defaultOption = document.createElement('option')  
 defaultOption.innerHTML = "Select Repository"  
 repoList.appendChild(defaultOption)  
 for (var i=0; i<data.length; i++){  
 var tempOption = document.createElement('option')  
 tempOption.innerHTML = data[i]  
 repoList.appendChild(tempOption)  
 }  
})

**</script>**

In this case the user can see what are the available resource data to manipulate. According to the **following Figure 3.1NEW** now we have all available repositories on the client side as a drop down menu. Therefore, the user is able to first choose the repository, make changes and then send the changes together with the repository name to the server so server knows where these changes belongs to.

****

Further more, the <select> element has an event listener which it will send the selected repository name to the server once it is changed in order to get more data about the selected repository from the server which we will demonstrate later in this section.

**3.1 Image processing**

From the moment that we have our repository’s name on the client side, we can send our changes to the server and server will modify our mobile application resource data. One of this changes is changing graphical contents such as changing the icon of the mobile application. In this section 3.1 we will describe two main approaches when it comes to image processing. The first approach is more automated which can change multiple images at the same time using parameters in the program and the second one is an integrated GUI image editor on the client side which provide the user more features. One of the Node modules that we have implemented for the automation on the server side is JIMP to ease workloads of editing our graphical contents in the resource directory. Example of these workloads are, crop icon launchers, change icon background color, applying shadows, apply padding, manipulate in-app buttons and modify transparency of an object. In addition to what mentioned we also provided an integrated image editor on the client side for the user to be able to uniquely edit any images.

**3.1.1 Crop icon launchers automatically**

To crop an icon launcher, user should first upload the image to the browser and send it to the server using a “submit” button. Following is a simple example of sending an image file to the server.

**var options = { "repositoryName" : repoName , "cropFrame" : frameName}**

**var submit = document.getElementById("submitButton");**

**submit.addEventListener('click', function(){  
 var socket = io.connect('serverAddress:Port')  
 socket.on('connect', function(){  
 var delivery = new Delivery(socket)  
 delivery.on('delivery.connect',function(delivery){  
 var file = $("input[type=file]")[0].files[0]  
 delivery.send(file, options)  
 })  
 delivery.on('send.success',function(fileUID){  
 message.innerHTML = 'File is now sent to the server!'  
 })  
 })**

**})**

**Code snippet 1**

Our file in the delivery of the above example also contains a JSON object named “options” which indicates the destination repository and also specify which frame should be used to crop the image. We can add as much as details we require into this JSON object using text fields, drop down menu and return all values into our JSON object and later handle them on the back-end. Here is an example of receiving the icon file on the server side.

**const fs = require('fs');**

**var io = require('socket.io').listen(5001)**

**var dl = require('delivery')**

**io.sockets.on('connection', function(socket){**

**var delivery = dl.listen(socket);**

**delivery.on('receive.success', function(file){  
 fs.writeFile(file.name, file.buffer, function(err){  
 if(err) {  
 console.log('Error: Icon file could not received by the server.');  
 } else {**

**console.log('Icon file received by the server.');**

**changeAppIcon(file.name, cropFrame)  
 };  
 });  
 });**

**});**

**Code snippet 2**

Another alternative to the above solution is Base64 encode and decode for sending images which we discuss later in this section. To crop images, we have all the frames located on the server and by using two for loop on both frame and the target icon we can change the icon based on the pixels that presents on the frame. Here is a code snippet as an example of such approach.

**function changeAppIcon(appIconName, cropFrameType){**

**Jimp.read(appIconName, function (err, appIcon) {  
 if (err) throw err;  
 Jimp.read("frames/"+cropFrameType+".png", function (err, masker) {  
 if (err) throw err;  
 masker.quality(100)  
 masker.resize(1024, 1024)  
 appIcon.resize(1024, 1024)  
 appIcon.write("Cloned\_Git\_Repositories/"+TargetRepoName+"/Resources/App Icon/ios/Icon.png")  
 appIcon.resize(1024, 1024)  
 for (var y=0; y<1024; y++){  
 for (var x=0; x<1024; x++){  
 pixelColor = masker.getPixelColor(x, y)  
 if (pixelColor > 0){  
 appIcon.setPixelColor(0xFFFFFF00, x, y)  
 }  
 }  
 }**

**})  
}**

**Code snippet 3**

Another example of this automation is resizing multiple images at the same time using JIMP in a directory which is the fastest way possible to modify multiple images. See appendix 1.1

The figure 3.4 shows the icons that already cropped with the application. Evidently, the figure show that for each of these three images we used different frame and thus the output is based on our **cropFrame** variable in code snippet 2

*Figure 3.4 – Cropping with frames*

**3.1.2 Change background automatically**

**Solution:** Using two for loop starting from **X=0** and **Y=0** and ends at **X=Max** and **Y=Max**. We change X and Y pixels colors into desired colors to create a background color at the time that the loop iterating through our pixels, and then we add the icon on top of the background color. The icon must be trimmed which the cropping feature can be used to solve the problem.

Figure 3.5 shows three different background color that we used for a same icon. The Android icon that is on top of the background is a PNG icon that has transparent property and after applying the background we merge the icon to the colored background.



*Figure 3.5 – Changing background colors of icons*

**3.1.3 Applying shadows automatically**

**Solution:** We can quickly increase and decrease shadows of the frame and add it behind the main icon. The **JIMP** module in node package manager can simplify this action. Figure 3.6 shows that the right side icon has more shadows than the left side icon. This can apply to the image while we have a shadow frame for each relevant frame that we have, and the shadow frame will be the first layer that we save on the file, then we change crop the image and put it as the second layer and the rest of the layers will be our subsequent layers.

*****Figure 3.6 – Applying shadow to app icons*

**3.1.4 Apply padding automatically**

**Solution:** When it's time to add the icon on top of a background or any other image we can use JIMP module to resize the image (Padding).

Basically, the end user sends the size to the server, and on the server, we have implemented functions to simplify this action.

*****Figure 3.7 – Padding of icons*

**3.1.5 Create in-app buttons automatically**

**Solution**: The arrow icon which is merged on each of the buttons can be selected or chosen by both company and customers. By using image processing, we can resize and add the PNG icon (Up and down icons) on top of a rectangle with a specific background color to produce a button image with the largest size first. Then we need to resize the large button image into the other sizes*.* Produce all in-app appearances with all required sizes. Each of these sizes can be used for different aspect ratio and resolutions. Figure 3.8 is an example of a button to open and close overview in front of an article in the mobile app.



*Figure 3.8 – Example of an in-app button in several sizes*

**3.1.6 Apply transparency automatically**

Create all in-app buttons or notification icons with transparency property. As we see in the picture below, the application must have buttons that can show it's behind activities. This is one of the most time taking task when it comes to producing a lot of buttons in the application. The amount of transparency can be selected on the client side.

*Figure 3.9* – *Applying transparency to in-app buttons if needed*

**3.1.7 Integrating image editor**

We disscused JIMP as an image processing library and to provide a more understanding environment for user we need to develope a user interface. \*\*\*\*\*\*\*<DONE UNTIL HERE>\*\*\*\*\*\*

**3.2 Configuring configuration files using Node.js**

repos.addEventListener('change', myfunction2)

function myfunction2(){  
 if (repos.value != "Select Repository"){  
 socket.emit('openThisRepo',{ repoName:repoList.value, configFile:mainConfig});  
 }  
}

The server will now respond with the main mobile application configuration data as a JSON object.

socket.on('openThisRepo', function (data) {  
 jsonfile.readFile(**clonedResources+**'/'**+**data.repoName, function(err, obj) {  
 socket.emit('CurrentConfigs', obj);  
 })  
})

Now that we have the JSON configuration on the client side its time to visualize it for the user by creating on the fly HTML elements and provide them for manipulation.

The other most important step to automate these tedious tasks is to automate configuration modifications. A configuration file within a mobile application, consist of many key attributes and values. Each value belongs to its key attribute and will be called when the application is running. This can also be considered as the primary data resource that helps us decrease our hard-code in the source code. By keeping all required data/variables in the configuration file, we can write fewer lines of codes, and we can inherit values that we need anytime and anywhere within the source code.

For example, in the last section, we have discussed "**Image processing for application appearances,**" and we have discovered that the end-user can change colors. Some of these colors are statics values which our project will produce them and save them automatically into their related paths *Figure 3.2*, and the rest color values should be set in the configuration files.

Figure 3.10 is a configuration file that called "**global\_config.json**" and values in this file are categorized as shared configurations for both iOS and Android version of the mobile application:

Path address: Resources\Modules\shared\configuration

*Figure 3.10* – *An example of a configuration file*

The source code of the application will call these values when it is running and if we, for example, change the "**secondary color**" which is a hexadecimal color, the whole colors that source code uses as "**secondary color**" will be transformed into our hexadecimal color.

As we see the complexity of a configuration file in *Figure 3.10,* any mistake can end up in bug within the code or failure in compiling process. Our solution will be used many exceptions to handle mistakes that a user might make. For example, if the hexadecimal color changed inappropriately or changed into a wrong hexadecimal color number, the end-user will be notified.

The developer has to address all values that they need from configuration files in their source codes. Sometimes source code uses its important values from many configuration files in several directories, and the path to these directories are entirely different with each other. The current situation or the current company solution follow manual steps to re-configure an application. Besides manual image editing, this procedure is also too time consuming and might end up in human mistakes if many configuration files need to be changed. In this case, the developer or the person who need to re-configure the configurations, have to go through these steps one by one for each file:

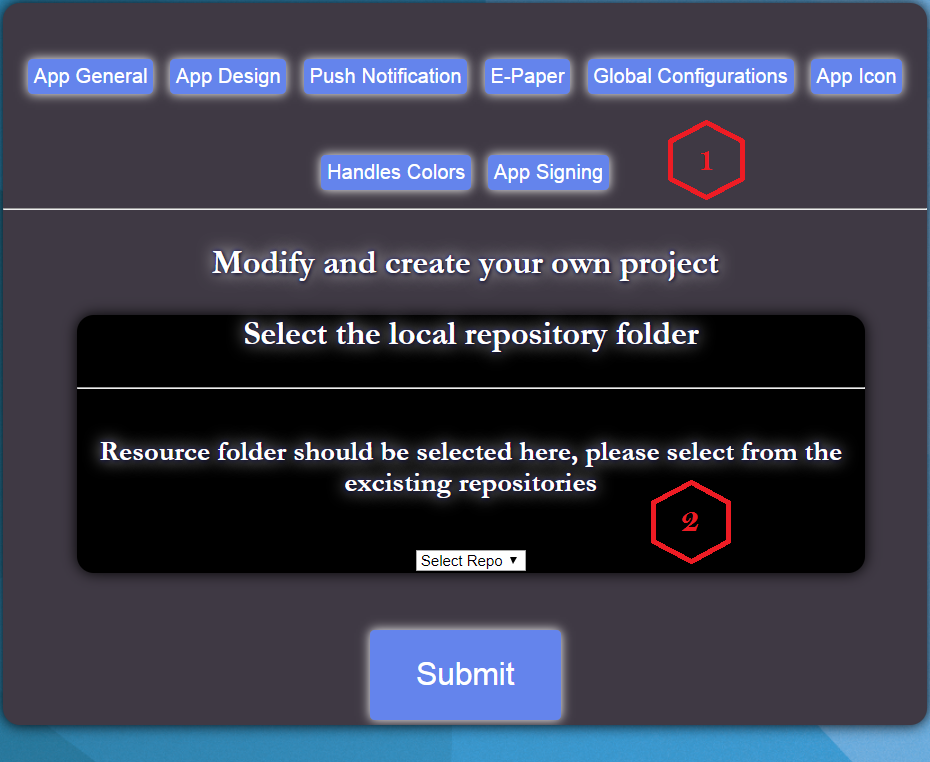
1. Navigate to each path/directory of the configuration file
2. Open configuration file in that directory
3. Find the key attribute in that configuration file and edit it
4. Save the file

How about if we need to re-configure many files? In such scenario, the estimated time is beyond the expected time to deliver an application.

This thesis will solve this problem by representing or visualizing all configuration files at the end-user. So, any available configuration file in the resource folder will be opened automatically and will be presented in a much-categorized style. The end-user will be able to re-configure a file without navigating to directories and also they will be notified which file has been left unsaved or unchanged. This solution will take less time, and less effort and even a person with no experience can change configuration files.

**Client-side demonstration**

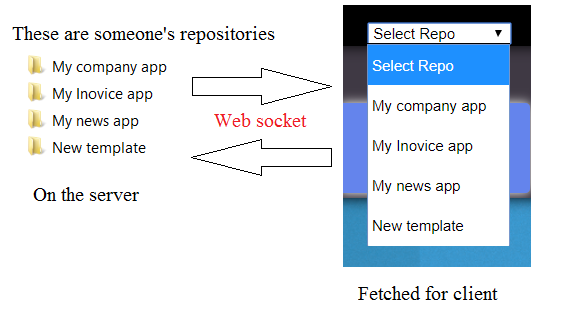
The client-side application will interact with server-side and send/receive requests/responds using web socket.

Figure 3.11 is the screenshot of the client-side application that demonstrates this idea.

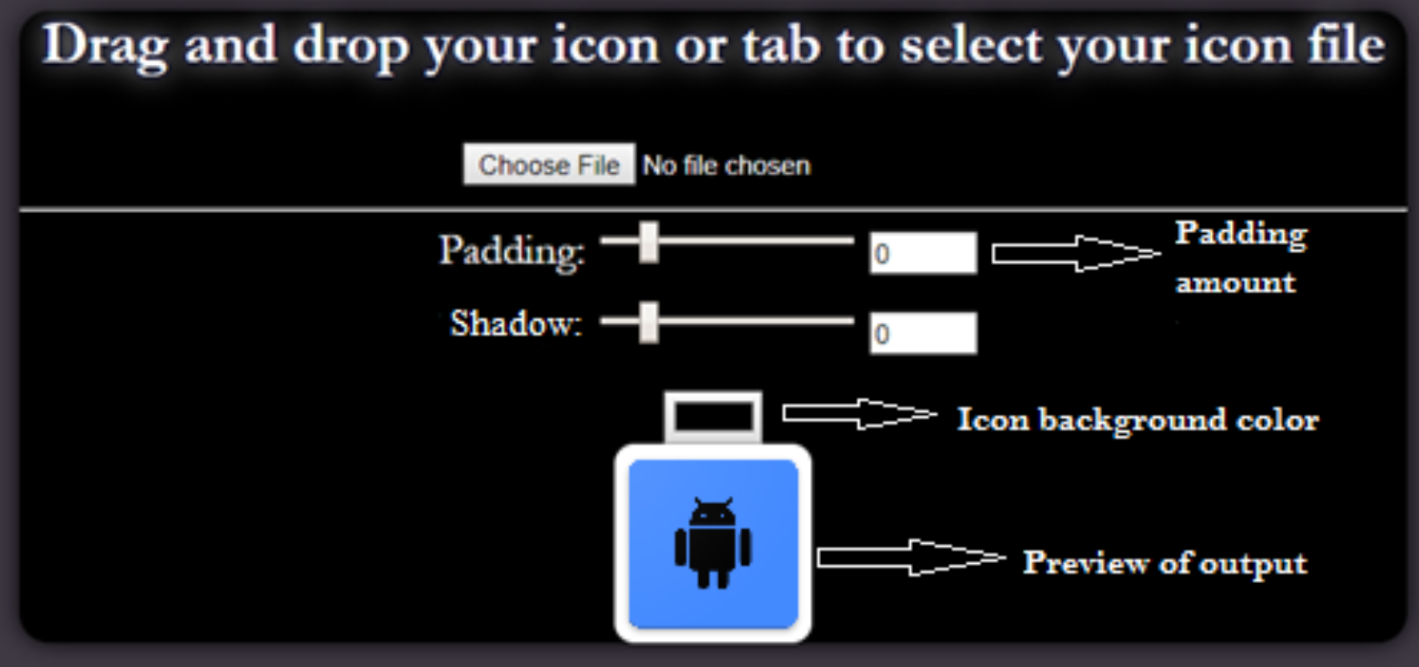
*Figure 3.11- An Implemented tool to automate the pipeline*

The number **1** in the *Figure 3.11* is the place that all configuration files and designs are represented by their names. Also, the **App Icon** and **Handles Colors** buttons are the tool that can design appearances.

The number **2** in the *Figure 3.11*, is the place that users can choose repositories. By selecting a repository, all configuration files will be fetched from the server. All repositories are on the server and are cloned from Git or initialized as template repository to start a new project. So, template repository has all the necessary configuration files, and they need to be modified with this tool.

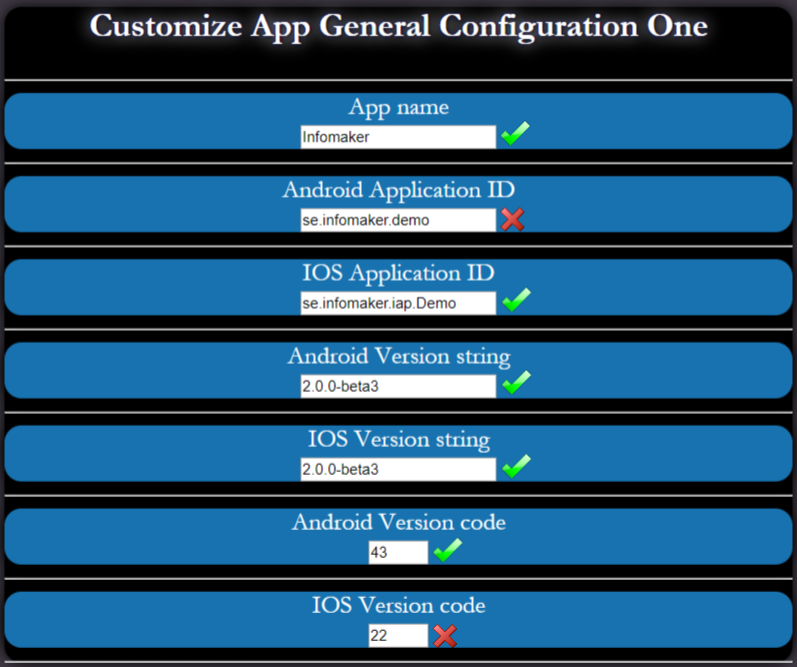
As *Figure 3.12* indicates*,* if a customer needs to access a repository to change their icons, the company can put their repo in their belonging path on the server, and they will be able to see their repository on the list below.

*Figure 3.12 – Fetching from repositories*

In *Figure 3.13,* by selecting the **App Icon**, an element (div or section element) will be added to the front-end which allows the user to modify the icon launcher of the application.

*Figure 3.13 – Tool to edit application icons*

Below is the same element that created for configuration at the front end. All values are fetched from files, and if the user inserts an incorrect value, the front-end will notify the user and handles the exceptions.



*Figure 3.14 – Modifying configuration files with validations feature*

As an example, in *Figure 3.14* if the version number inserted incorrectly and instead of increment number the user insert decremented number, they will be notified. This is the overall example for handling configuration files. The template resource folder has version 0, a template icon launcher, and all values in its configuration files are empty, except those values that are always in common with another repository, for example, a password that is similar in other repo or a key.

# 

# **4** **Results**

We have described four objectives in section 1.5 and to clarify each of them, in this chapter we describe the result of each objective.

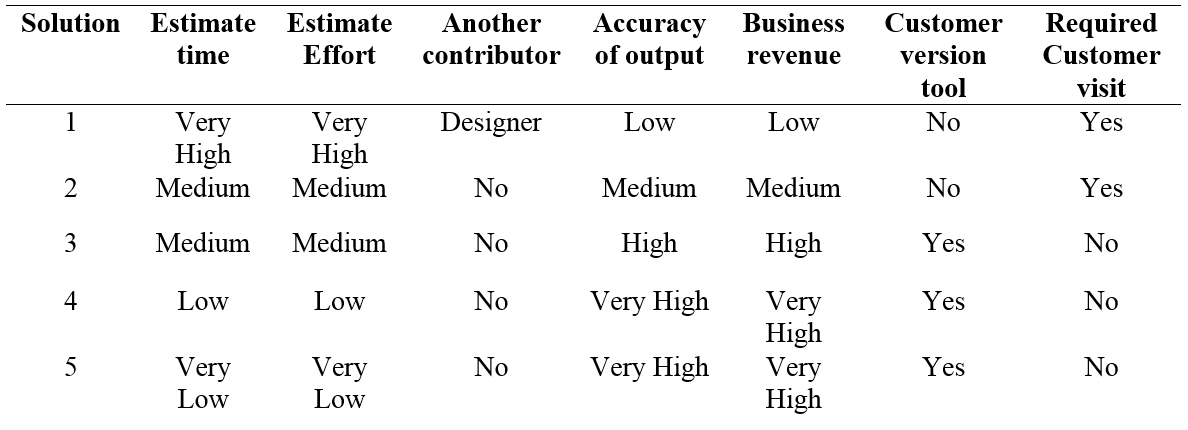
4.1 Problem identification

By identifying the problem we have notified that the current solution that the company is using to develop a mobile application can be counted as the main problem. The current solution is time consuming and not efficient since many employee should spend time for contribution and many resources should be used at the same time to produce a mobile application.

After we identified the problem, we need to start investigating different solutions for the problem which is also the result of problem identification.

4.2 Solutions

We have discussed 5 primary solutions to solve the problem. These solutions are:

1. Current company's solution
2. Offline tool for the company only to produce resource data for mobile apps
3. Offline tool for both company and customer to produce resource data for mobile apps.
4. An available online tool with customer and developer version to produce resource data for mobile apps.
5. **Online available tool to also automate deployments and Git integration.

*Table 4.2.1- Evaluating our several challenges*

The *Table 4.2.1* will show the benefits and differences between each solution. We have to go through many tasks from starting an application until it is completely deployed. These results are based on estimated cons and pros that involved in each solution. All values provided in the table can be varied from company to company and these values are based on the interviews with the specific company and also discussions that we had with business owner of the company.

4.3 Design and development

By evaluating several solutions and interviews with the company, now we have the opportunity to choose a discussed solution and to design an architecture to solve the problem. The result of this objective is that we use related works to provide a conceptual basis for all requirements and design the relevant architecture.

4.4 The demonstration

The result of this objective is proving that the selected solution can solve the problem in the way that the company is expected. We demonstrate and prove that the newly designed architecture will bypass all manual tasks. Also, we demonstrate how the application can be used and what requirements does it need to be implemented. In this demonstration we can also present the software by user manual and presentation which can help new employee at the company to use the software.

# **5 Discussion & Analysis**

In the result section, we have provided a table to estimate cons and pros of each solution. If the company or any other companies want to start producing and deploy an application from 0% to 100% progress, what is the estimated data for each of the following factors and how we estimate them?

**Estimated time to deploy an application**

How much time does it take to develop and deploy an application?

In the process of developing a mobile application, we need to do several tasks, such as, working on the source code and do coding, prepare the user interface for the mobile application with good graphical content, prepare configuration files, push the project to Git repository and finally deploy the application from Git.

Without an automated system, these tasks need to be done manually which needs several involvements. On the other hand, if we want to do these tasks manually, some of these tasks need to be done with the help of other tools or applications, such as, producing user interface contents using Adobe Photoshop or GIMP which the company already use these tools to develop user interface of the mobile application.

The other time taking task belongs to modifying configuration files which can take time to find many configuration files and modify them in several subdirectories in the resource directory. Also, if our works are depended on someone else tasks, the required time will be increased. For example, waiting for the designer to change the color of the mobile application icon. If the designer is not available or busy with other tasks, there will pending time, and the task cannot be done on time even if the developer wrote the code. In this case, the customer should wait for the application to be deployed.

This excludes customers if they decide to design the application with the automated tool. For example, the customer wants to design the application, in this case, the developer will continue on his/her primary responsibility which is just preparing the source code within the pipeline, and the company will not be responsible for the delay.

**Estimated efforts to develop and deploy an application**

How much effort does it need to get an order from the customer until we deploy the application?

If we need to go through all steps that discussed in chapter 3 for design and configuration files, then we need more effort to accomplish them without an automated pipeline. In this case, we need to use several tools to edit image files and also modifying configuration files manually. The effort that we spend to deploy an application can be decreased with an automated tool. For example, using image processing instead of using Adobe Photoshop will bypass the effort that we need to spend on some applications to edit our user interface contents.

The digital image processing is much faster than manual image editing because we use an algorithm to modify image's pixels and can be compiled and edit the image in less than few seconds. Besides digital image processing, we have written an algorithm to find all configuration files within a resource directory to present them for the end-user by visualizing them. In this case, the end-user will no longer need to navigate into several subdirectories in resource directory to find configuration files.

**Another contributor in the development pipeline**

Can a developer or customer design the user interface content without experience in image editing?

We defiantly need to believe that we can finish the job ourselves if we want to stay tuned for the remaining time of the project. The developer has to push the project forward since everything is related to the source code that she/he is developing. The source code cannot compile without the resource directory and vice versa. On the other hand, if developers or customers use the automated tool to edit images using image processing then they no longer need a designer to change colors, background, and icons, etc.

**Accuracy of output**

Human involvement can cause mistakes if many tasks have to accomplish within the pipeline. The computer and processing are much more accurate than manual commitment to process a data [17]. This is also true that in our automated tool we believe that the accuracy of algorithms is higher than human accuracy.

When it comes to modifying configuration files or edit the graphical contents, the processed data by CPU can be tested. Also, a human might forget to edit some attributes within the configuration files or might make a mistake when there are no exceptions to compare the inserted data with a correct format. As we indicated in the demonstration, the person who is working with the tool will be notified if they insert illegal values or wrong strings. On the other hand, in the manual modifying without the automated tool, there will be no exception for wrong inserted values, and if the user saves the configuration file with a wrong attribute inside, then the application will misbehave after the deployment as long as the source code uses these attribute for functioning.

With the automated pipeline, every exceptions and mistake can be targeted logically in short time which will help the company to finish a project on time with no mistakes.

**Business revenue**

As we discussed in previous sections, by spending less effort and less time for an application the company might no longer need to pay the salary for repeatedly and time taking tasks. With the automated pipeline, the company can deliver applications much faster to their customers. On the other hand, if the company keep their customers on a long waiting list, then they are driving away their customers [18], and the company cannot sell products to them. This will negatively affect both community and the network of the company.

The other factor that can charge the company is buying a license for image editing applications, such as, Adobe Photoshop and other applications that need a license. With the automated pipeline, the company will no longer need to accept these charges.

**Customer version tool**

Customers want their desired outcome, and by allowing them to decide their expected outcome, we comply with customer understanding which is also the key to expanding the business [19]. The automated tool will help customers to edit the content of their application and get their desired outcome. On the other hand, they will also be able to deploy their new version of the application with no knowledge of the deployment process.

**Required customer visit**

Can a customer modify their application contents from their home?

With the automated tool that belongs to customers, they can modify and edit their application. Also, they will help the company as a contributor to the project, and this is both beneficial for the customer and the company. The customer can contribute using the online tool or customer version tool from anywhere as long as we have mobility idea to support them.

# **6 Conclusion & future work**

The findings within this thesis are gathered by general investigations and interviews with a software company. As we have seen, the results are mainly focusing on simplifying the manual process by automating techniques and in many software industries time is highly valuable while the customers might be disappointed because of waiting time [18].

This thesis also suggests optional customer involvements in their application deployment pipelines, and with this, we can ensure that the customer will be satisfied if we keep them close to their desired outcome [19].

Results of this thesis can be found useful within business and software industries while we have discussed the revenue and we conclude that time taking tasks are not so beneficial for a company.

If we could have more time within this thesis, then we could continue this idea to design a self-driven architecture with more beneficial outcomes. This could be done by more interviews with more companies to evaluate and compare their experiences with each other in mobile application development pipeline.

From the demonstration side, this report can be upgraded to demonstrate the automatic deployment feature. This feature can be added to the current implementation as a trigger, and by pressing the trigger, the resource data will be pushed to the Git and then deployed automatically.

Although this idea has been applied to one company, but it is possible to follow the objectives again in other companies to identify the problem, find a solution based on the problem and then design an architecture to solve the problem within those company.

We have introduced four objectives within this thesis and each of these objectives has their relevant conclusion and experienced that we have achieved which we discuss them in this chapter.

6.1 Problem identification

We have experienced that the problem can be identified in several perspectives and more investigation can cause more problem to show up and to be identified. These problems will be occurred as designing architecture and what architecture to use or for example how to apply image processing to the architecture. All of these problems can be solved easier and faster in future projects if we have experienced them for one time.

6.2 Solutions

When it comes to find a solution for a problem, all problems can be solved with several solutions and we have experienced that each solution that we chose can affect the next step of the software functionality negatively or positively. As an example, one solution for the image processing can target many problems, such as cropping the image, applying shadows and changing colors to mobile application design in one algorithm instead of separating the solution which can also minimize the lines of code. As we have experienced, for future work it is really important that we write the code and develop our methods in a way that it can be flexible with adding new algorithms, functionalities, and future requirements.

6.3 Design and development

In the process of designing an architecture we have experienced that the design can be changed based on the company's expectation and can be also challenging while we should meet all the requirements with one design. Sometimes it is not possible to design an architecture that can automate all manual tasks and instead we should consider that the application might have human involvement at the end. Some of these involvements can be only automated with artificial intelligence and in that case we should consider that it might affect the accuracy of the result when it comes to drawing a PNG icon manually which needs human actions to be accomplished.

6.4 The demonstration

With demonstration, we have the chance to prove our work and everything rely on the completion of the work and if it is done by a certain time. Although demonstration can transform idea into reality, but on the other hand it is difficult to prove everything within demonstration while we have a limited time to develop the idea. In future work we should consider that the demonstration is totally related to the progress of the project and we should avoid demonstrating until the application is developed entirely.

# **References**

[1] Weebly. (2016) The pipeline of mobile application development [Online]. Available: <https://carolcausey.weebly.com/blog/the-pipeline-of-mobile-application-development>. [Accessed March.11, 2018]

[2] Android. (2018) Providing resources [Online]. Available: https:// developer.android.com/guide/topics/resources/providing-resources. [Accessed March.13, 2018]

[A] 25hoursaday. (2007) The Difference Between Hard Coding and Magic Numbers [Online]. Available: <http://www.25hoursaday.com/weblog/2007/04/11/TheDifferenceBetweenHardCodingAndMagicNumbers.aspx>

[3] Wikipedia. (2018) Software architecture [Online].Available: https:// en.wikipedia.org/wiki/Software\_architecture. [Accessed March.13, 2018]

[4] Wikipedia. (2018) Digital image processing. [Online]. Available: https:// en.wikipedia.org/wiki/Digital\_image\_processing. [Accessed March.15, 2018]

[5] IEEExplore. (2015) Continuous Delivery: Huge Benefits, but Challenges Too [Online]. Available: [https://ieeexplore.ieee.org/abstract/document/ 7006384/?part](https://ieeexplore.ieee.org/abstract/document/%207006384/?part)=. [Accessed March.16, 2018]

**[6deleted] ACTA Press. (2012) Model Driven Development of Cross-Platform Mobile Applications [Online]. Available: [http://www.actapress.com/ PaperInfo.aspx?PaperID=454180&reason=500](http://www.actapress.com/%20PaperInfo.aspx?PaperID=454180&reason=500). [Accessed March.17, 2018]**

**[7deleted] J.-M. Favre, "Towards a Basic Theory to Model Driven Engineering." Workshop on Software Model Engineering, Oct. 2004**

**[8deleted] Android. (2018) Launcher Icon. [Online]. Available: https://developer. android.com/guide/practices/ui\_guidelines/icon\_design\_launcher\_archive [Accessed March.20, 2018]**

[9] Roman Nurik. (2018) Front-end engineer. [Online]. Available: https:// roman.nurik.net. [Accessed February.8, 2018]

[10] Android Asset Studio. (2018) Online tool. [Online]. Available: https:// romannurik.github.io/AndroidAssetStudio. [Accessed February.5, 2018]

[11] Wikipedia. (2018) Git. [Online]. Available: [https://en.wikipedia.org/ wiki/Git](https://en.wikipedia.org/%20wiki/Git). [Accessed February.5, 2018]

[12] Wikipedia. (2018) Software deployment. [Online]. Available: https:// en.wikipedia.org/wiki/Software\_deployment. [Accessed March.11, 2018]

[13] If you want to grow your company. (2018) [Online]. Available: https:// www.entrepreneur.com/article/272027. [Accessed March.22, 2018]

[14] 5 Reasons Why Customer Satisfaction Is Important. (2018) [Online].

Available: <https://survicate.com/customer-satisfaction/importance-customer-satisfaction>. [Accessed March.22, 2018]

[15] Mozilla. (2018) Websocket. [Online]. Available: https://developer .mozilla.org/en-US/docs/Web/API/WebSockets\_API. [Accessed March.25, 2018]

[16] Wikipedia. (2018) Configuration file [Online]. Available: https:// en.wikipedia.org/wiki/Configuration\_file. [Accessed March.26, 2018]

[17] Data and process advantage. (2010) Is computer analysis accurate? [Online]. Available: <https://www.dpadvantage.co.uk/2010/03/09/is-computer-analysis-accurate>. [Accessed March.28, 2018]

[18] INC. (2014) 5 Things That Drive Away Customers [Online].

Available: <https://www.inc.com/bill-murphy-jr/5-things-that-drive-customers-away.html>. [Accessed March.28, 2018]

[19] Sixteen ventures. (2014) Understanding Your Customer’s Desired Outcome [Online]. Available: <https://sixteenventures.com/customer-success-desired-outcome>. [Accessed February.25, 2018]

[20] Wikipedia. (2018) Midpoint circle algorithm [Online].Available: https:// en.wikipedia.org/wiki/Midpoint\_circle\_algorithm [Accessed January.20, 2018]

[21] Android. (2018) Add app resources [Online]. Available: https:// developer.android.com/studio/write/add-resources. [Accessed June.14, 2018]

[22] Atlassian. (2018) Bitbucket deployments [Online]. Available: https:// confluence.atlassian.com/bitbucket/bitbucket-deployments-940695276.html. [Accessed June.18, 2018]

[23] Bugwolf. (2018) Understanding the software delivery pipeline [Online]. Available: <https://bugwolf.com/blog/understanding-the-software-delivery-pipeline>. [Accessed June.20, 2018]

[24] Medium. (2018) How we automate deployments and testing with bitbucket pipelines [Online]. Available: <https://medium.com/monplan/how-we-automated-deployments-and-testing-with-bitbucket-pipelines-bb478c12c55f>. [Accessed June.25, 2018]

1. Dzone. (2018) 9 benefits of continuous integration [Online]. Available: <https://dzone.com/articles/9-bene-ts-of-continuous-integration>. [Accessed June.25, 2018]

Appendix

An example of modifying multiple images at the same time using NodeJS and JIMP.

const Jimp = require('jimp');const glob = require('glob'); // yarn add glob

/\*\* \* Resizes images in the directory. Only operates on .png, .jpg, and .bmp. \* @param {string} dirPath - Path to directory. Can be relative or absolute. \* @param {Object} options \* @param {int|Jimp.AUTO} [options.width=Jimp.AUTO] \* @param {int|Jimp.AUTO} [options.height=Jimp.AUTO] \* @param {boolean} [options.recursive=false] - Whether or not to also resize recursively. \* @return {Promise} \*/function resizeDirectoryImages(dirPath, { width = Jimp.AUTO, height = Jimp.AUTO, recursive = false }) {

return new Promise((resolve, reject) => {

glob((recursive ? "\*\*/" : "") + "\*.@(png|jpg|bmp)", { nocase: true, nodir: true, realpath: true, cwd: dirPath}, (err, files) => {

if(err) {

reject(err);

} else {

resolve(files);

}

});

}).then(files => {

return Promise.all(files.map(path => {

return new Promise((resolve, reject) => {

return Jimp.read(path).then(image => {

image

.resize(width, height) // You may want to change this.

.write(path, (err) => {

if(err) {

reject(err);

} else {

resolve(path);

}

});

})

}).then(console.log)

}));

});}

resizeDirectoryImages('./example', { width: 500 }) // Resize all png, jpg, and bmp images in the example directory to be at max 500 wide

.then(() => {

console.log('Done!');

});