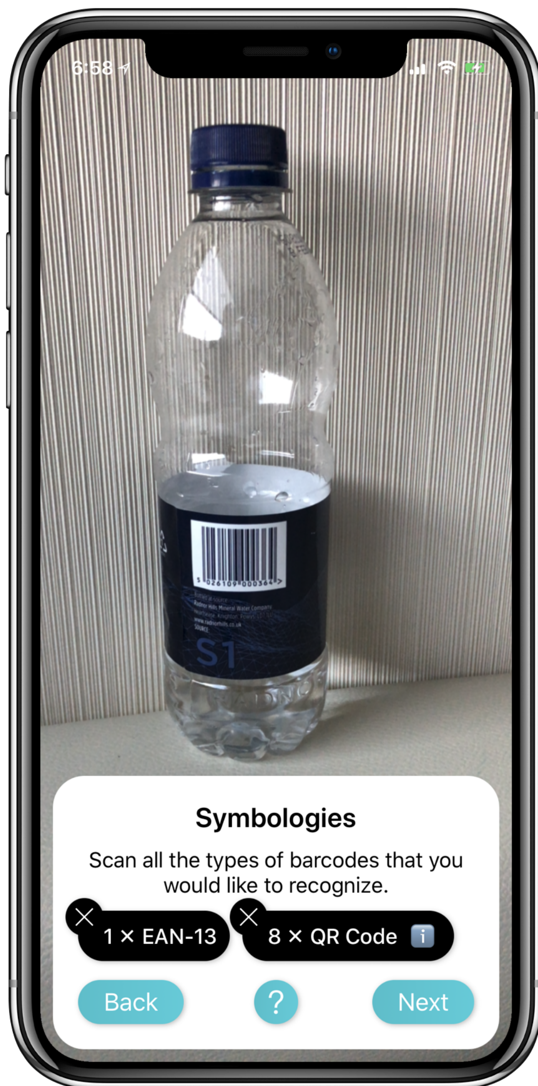
Università
della
Svizzera
italiana

Patrick Balestra

Bachelor Project 2018

Interactive Barcode Scanner Optimizer and Configurator



Advisor: Prof. Michele Lanza
In collaboration with Scandit AG.
May 29, 2018

Abstract

The main product of Scandit is a barcode scanner and data capture SDK that customers can use to recognize many types of machine-readable data in their own products. I have been working at Scandit as a part-time Software Engineer for about a year and customers sometimes struggle in successfully **configuring the barcode scanner**. The library requires reading the documentation and the provided example projects to fully understand. A solution that the team came up with to make it easier for customers to get started is the development of an iOS application that can infer the customer needs. The source code that configures the barcode scanner is then generated and shared with the customer.
In this report I am going to describe the process involved in developing the iOS application.

Intro

About Scandit

This project has been developed in collaboration with Scandit[1], a company headquartered in Zurich that provides the **highest quality mobile barcode scanning solutions** for smartphones, tablets and wearable devices. Some of the clients include Migros, Coop, Macys, Louis Vuitton, DHL, NASA, Verizon Wireless and many other companies around the world.
The main product of the company is a barcode scanner and data capture SDK[2] that customers can integrate into their applications to scan barcodes. The flexibility of the barcode scanner allows for infinite use cases but usually requires the help of an engineer to correctly configure all the required parameters and integrating it with the existing codebase.

Motivation

The customer support team at Scandit receives many support requests everyday consisting in customers not being able to correctly configure the settings of the barcode scanner. Therefore, many of the bug reports received are caused by a misconfiguration of the scanner that results in unexpected behaviors. These bug reports are time-consuming for the whole company, since Scandit engineers have to debug third-party projects and solve different types of issues on their behalf.

Goal

This project has the goal to increase customers happiness by reducing the difficulty of configuring the library and the number of support requests received.
To achieve this goal, I have developed an iOS application that **understands the user needs** and automatically adjusts the barcode scanner based on the environment and features of the recognized barcodes. The user is initially asked to scan barcodes in the same real-world scenario where the barcode scanner will be employed in the final product. By following this guideline, the app can generate a more accurate optimal configuration. The app analyzes the frames and barcodes features recognized by the computer vision algorithm running in the Scandit library. The application also asks the user some questions in order to better

[1] https://www.scandit.com

[2] https://www.scandit.com/products/barcode-scanner/

understand the use case and generate the perfect configuration. The relative source code that configures the Scandit SDK is generated and stored in a file on disk to be previewed and shared with the user for further inspection and integration in his codebase.
Swift and Objective-C source code can be generated to integrate the Scandit library with any iOS application of the customer. A future addition to the project may include support for other popular languages and platforms such as Java, Kotlin and Android.

Structure of the Report

The next section of the report contains more information about barcodes and why I think this project is filling a gap in Scandit's business. The biggest part of this report is an explanation of my approach describing the architecture of the project along with the most important features and technical details. In the end, I also describe how the project was evaluated to make sure it solved the initial problem.

State of the Art

A barcode is a **machine-readable representation** of some data[3]. During the 70s, the UPC barcode symbology became commercially popular when supermarkets started to assign each product a barcode to speed up the checkout process. Nowadays, there are two main types of barcodes:

- **Linear or 1-dimensional code**: it uses vertical bars and spaces to represent a limited amount of data (usually up to 20-30 characters).
- **Matrix or 2-dimensional code**: it uses a pattern of circles, rectangles or other shapes to represent a big amount of data (up to 8000 characters).

Most of the code symbologies incorporate check or error correcting digits to be able to verify the recognized value. For many years, special optical scanners were required to scan barcodes. With the **improvements in portable cameras quality** and computer vision technologies, companies such as Scandit were able to develop mobile solutions to make barcode scanning even more approachable and affordable. Barcode scanners on mobile devices can be very complex to configure given the many variables that can affect the scanning quality, such as camera resolution, focus, zoom, barcode type and content. So far, Scandit has relied on **sample projects and online documentation** to help customers in configuring the barcode scanner. Documentation and sample projects can be insufficient for some customers since these resources are pretty time consuming to browse or don't satisfy everyone. For example, the documentation has a trade-off between its completeness and size. A massive documentation could intimidate developers and encourage them to skip reading it and directly explore the technical details. A short documentation may be more welcoming to developers but will not contain all required details for some specific uses. This is the gap that I have tried to fill with this project. By merging documentation, sample projects and code generation in an interactive way, I hope to provide a tool that improves the experience of discovering and integrating a new library into any project.

[3] https://en.wikipedia.org/wiki/Barcode

Approach

Analysis & Planning

The project started in February 2018 with my colleagues at Scandit who initially created a document outlining the main idea of the iOS application. I traveled to their headquarters in Zurich to meet in person with the team and **clarify the exact features** and align on the timeline of the project.

Generating an optimal configuration requires knowledge of many different details about the use case of the customer. For this reason, we first analyzed the **dependencies** between the data that we should gather from the user. The flow chart in figure 1 is the final dependency graph that shows all steps required to generate a configuration.

Thanks to this initial analysis, we were able to prioritize the parts of the project needed to build a **minimum viable product** (MVP). We built an end-to-end working prototype in order to be able to quickly iterate on the product.

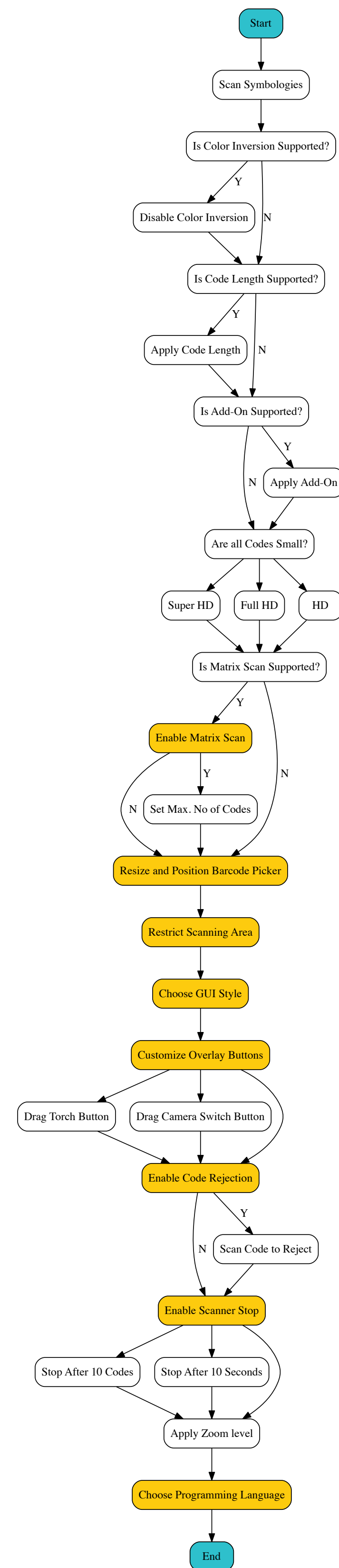


Fig 1. Steps dependency graph.

Project Configuration

I generated the Xcode project and created a git repo hosted on Scandit's GitLab servers to allow my colleague Luca to review and approve my pull requests. I decided to use the Carthage[4] dependency manager to integrate two external libraries. The first one is a micro-library wrote by myself in the past and open sourced on GitHub called *Stryng*[5] which improves the syntax to manage and modify strings in Swift. The second dependency is the *ResizableView* component that allows the user to resize a view and select a specific area of the screen which is described in more details later on.

UI and UX implementation

The AirPods and HomePod setup UI[6] available on all Apple devices since iOS 10 inspired the user interface and user experience of the Configurator app. The live camera feed is displayed in fullscreen and the interactive view with all important information is positioned in the bottom part of the screen. In this way, the interactions with the application such as changing the parameters of the configuration and navigating through the flow are quickly accessible as shown in figure 3. The view can also be temporarily dismissed by

[4] https://github.com/Carthage/Carthage

[5] http://github.com/BalestraPatrick/Stryng

[6] https://support.apple.com/en-us/HT207010

swiping it down, since on smaller screens it may take up more space than what is available for the camera feed where the scanned barcodes appear.

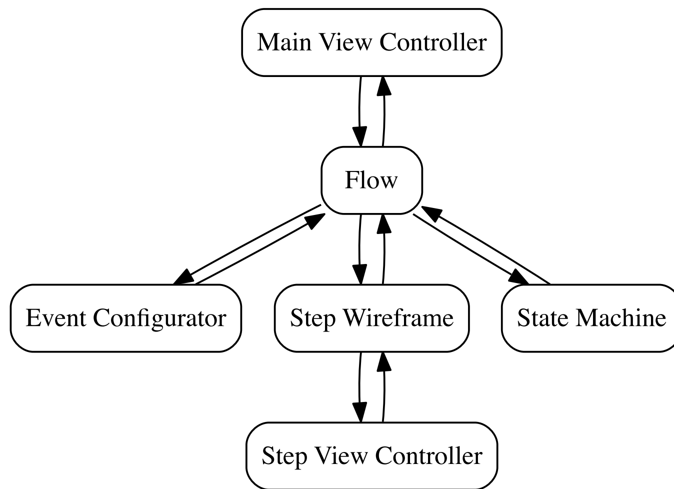


Fig 2. Core application architecture.

Fig 3. Welcome step.

Architecture

The project consists of many similar views and a state machine that uses the ***GameplayKit***[7] framework to manage the presentation of the states during the configuration. To remove responsibilities from the view controller and avoid building a single big object with low cohesion, I've decided to create a separate object called **wireframe**. The wireframe takes care of managing the internal presentation of its subviews and the communication[8] with the flow object.

The **flow coordinator**[9] interacts with the state machine to decide if a state transition is valid and which step to display next. Since the state machine can be influenced by the business logic, to help in the decision of the next state, the **event configurator** object stores all the data gathered from the barcode scanner session. The flow object then simply queries the event configurator to decide what the next state is. This **separation of concerns** allows for a better testability of each component. The project has been developed with testability in mind which in the end yielded a good test coverage. The project has a total of over 240 tests which result in a code coverage of around 75%. The graph in figure 2 shows the architecture of the application.

---

[7] https://developer.apple.com/documentation/gameplaykit

[8] The communication architecture between the wireframe and configurator flow has been inspired by the following article: https://mag.n26.com/creating-flows-f1c2e1bc8108

[9] The flow coordinator architecture was first introduced by Soroush Khanlou on his blog: http://khanlou.com/2015/10/coordinators-redux/

Welcome

The first step of the application is a welcome screen that **explains the application goal**. This is necessary since the application may not be very straightforward to understand for new users. This step offers a short explanation on how to use the app to generate a barcode scanner configuration. The user interface of the app running on an iPhone X can be seen in figure 3.

Scan Symbologies

This is the core step in generating the configuration because **important data are collected** during this scanning session. First of all, the user is asked to scan all barcodes that the final configuration is supposed to recognize in an **environment that matches the final use case** of the app as much as possible. For example, if the final configuration will be used to scan small codes positioned far away from the camera, the configurator should be employed in a similar scenario to allow the algorithms to figure out the combination of settings that result in the best scanning experience.

The barcode scanner session is initialized by enabling every setting to recognize all types of codes. From the data collected in this step, we can then narrow down the use case of the customer to more fine grained settings and at the same time improve the performance of the scanner by restricting its recognition parameters. The **more restricted the settings** of the barcode scanner are, **the faster and better** it will work.

The scanner will try to recognize all types of supported barcodes. The configurator stores symbology, symbols count, color inversion and code count for each scanned code. This is done to preserve every information in order to generate the final scan settings source code. The algorithm then groups barcodes by symbology to allow displaying them to the user as a single entity. More information are provided if needed, such as the complete range of active symbols count (the length of the code's content), the requirement of color inversion and the total number of codes scanned. As seen in figure 4, the scanner in this example recognizes 1 EAN-13 and 8 QR codes.

Matrix Scan

Matrix Scan is a feature that enables the scanner to **track multiple codes over time**. This is very useful in scanning areas that may contain many codes that are tracked simultaneously to give an augmented reality effect.

All 1D symbologies and some 2D symbologies can be tracked with matrix scan. If it is enabled, the following steps will automatically take this information into account to suggest the user the best settings that fit well with matrix scan. This is one of the powerful features of the application. It can adapt to the user needs to facilitate and speed up the creation of the configuration. The customer simply needs to decide if matrix scan is part of its use case or not, as shown in figure 5.

Resize Picker

Some customers may wish to resize and position the picker differently on the screen. The code generated to position the barcode picker in the desired position is not strictly related to the Scandit barcode scanner, but it is helpful for customers to get started in integrating the library in a much easier way. A separate component named *ResizableView* has been developed to allow users to **interactively resize and select a specific area** of the screen. I could not find a similar open-source component that fulfilled all my needs. For this reason, the component has

been developed as a separate project and integrated with the Carthage dependency manager. By decoupling it from the main project, it may be easily open sourced in the future.

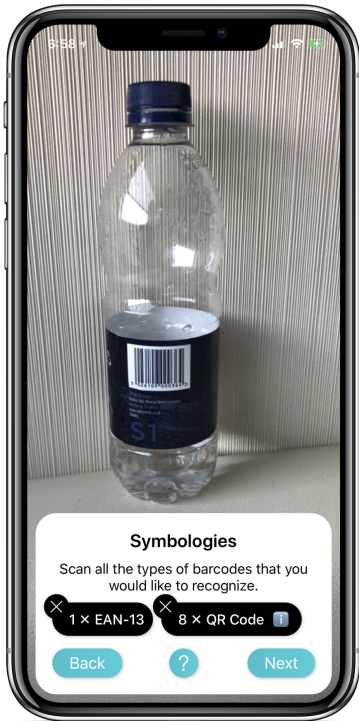


Fig 4. Scan Symbologies step.

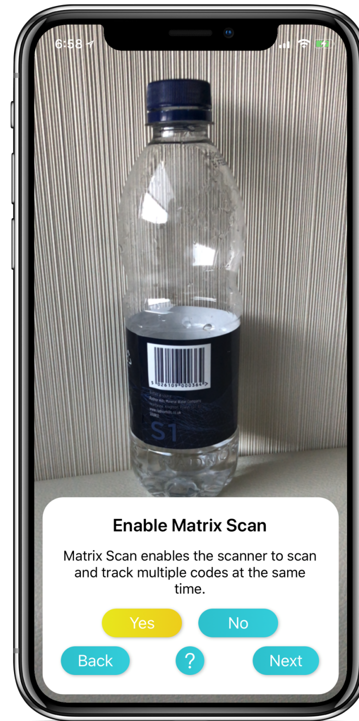


Fig 5. Matrix Scan step.

Scan Area

The barcode scanner may be used in some cases to scan small barcodes that are positioned very close to each other. For this reason, it is possible to **restrict the scan area** to a small specific part of the camera feed. The same *ResizableView* component used in the resize picker step is displayed again (shown in figure 7) to let the user select a specific area of the screen where the scanner should restrict barcodes recognition.

GUI Style

The guy style simply overlays an image to the camera preview to help the user place codes in a specific area for the scanner to recognize them. As it can be seen in figure 8, there are a few different styles to choose from:

- **Frame**: shows a rectangle that helps the user in positioning codes in the camera. When selecting this style, the position and size of the viewfinder must be chosen. For example, in case where small 1D codes are scanned, the viewfinder can be very thin in height but large in width to make it clearer for users which barcodes types and where they are expected in the camera.
- **Laser**: a laser image is shown in the middle of the screen to replicate the physical laser scanner behavior. This is usually displayed in case 1D codes are expected.
- **Locations Only**: codes are highlighted with a temporary rectangle to give feedback about which codes have been recognized.

- **Matrix Scan**: codes are continuously tracked over time with rectangles to give feedback about the recognized codes in every frame.
- **None**: no visual feedback is shown on the barcode picker.

Fig 6. Resize Picker step.

Fig 7. Scan Area step.

Overlay Buttons

The user can interact with two buttons overlayed on the barcode picker to change some features of the scanner while using it. A button to turn on the torch of the device can help scan codes in the dark. The switch camera button gives the user the freedom to change to the front camera if needed instead. Each button can be dragged to a different position on the barcode picker or hidden based on the needs of the customer. As shown in figure 9, the torch and switch camera buttons are activated.

Code Rejection

The scanner tries to recognize all the code symbologies that are enabled in the specified scan settings. Code rejection is a feature that allows restricting the recognized codes and **reject specific codes based on a custom logic** as shown in figure 10. The customer may choose to only scan codes with a specific prefix on weekdays and a different prefix during the weekend for example. This is made possible by enabling code rejection and writing the logic to decide whether accept or reject each code.

This step generates the code to enable code rejection. It first asks the user whether code rejection should be enabled. If the answer is positive, a single code should now be scanned. The content of this code will be rejected as an example in the generated configuration. To make it easier for customers to integrate their logic, the generated configuration contains

everything needed to get started and only requires the user to insert their own logic to decide when to reject codes.

Fig 8. GUI Style step.

Fig 9. Overlay Buttons step.

Stop Scanner

Some customers need to stop the scanner in case a **certain condition has met**. To help them with this task, the user is asked if the scanner should be paused in two conditions:

- After a fixed amount of time (10 seconds).
- After a fixed amount of codes scanned (3 unique codes).

Many support emails received by the team at Scandit are related to customers being unable to integrate a custom logic to stop the barcode scanner. This step tries to reduce the complexity and the number of support requests received by generating the two most common solutions as it can be seen in figure 11. It is also totally optional, therefore it can be skipped and the code won't be generated in the final configuration.

Code Generation

In this step, the customer can choose the programming language of the generated configuration. At the moment of writing, the supported languages as shown in figure 12 are:

- **Swift**: the newest language[10] developed by Apple which is now the default go-to language to write applications for Apple's platforms.

[10] https://swift.org

- **Objective-C**: the language[11] that has been used for almost 20 years to develop applications for Apple's platforms.

Fig 10. Code Rejection step.

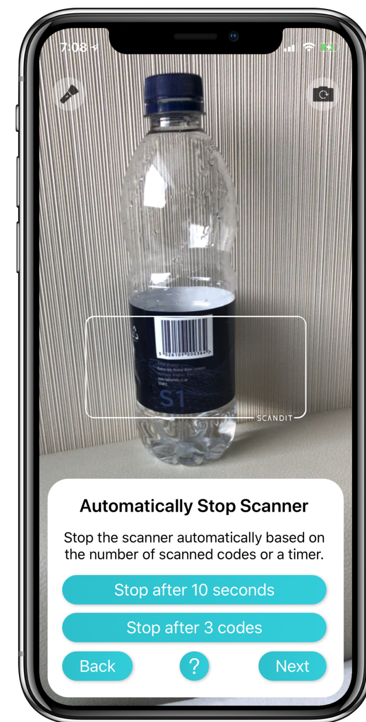


Fig 11. Stop Scanner step.

A big focus since the beginning of the project has been to be as **flexible** as possible in allowing new languages to be added in the future. This is important because the generated code configures a specific version of the Scandit SDK and in case a newer version of the library will be released in the future, we can simply create a new code generator and even let the user choose which one they would like to use at runtime.

My approach to generate the configuration has been to create a **template** file for each code generator. This template has the basic code to set up a single view with the barcode scanner. The file also contains a number of **tokens** that are replaced at runtime with the generated code.

To easily swap and write new code generators in the future, the *CodeGenerator* **protocol** defines all the functions needed to generate a configuration. Each function is responsible to generate a small part of the configuration to enable a better testability and easier code maintenance. Since the application at this moment supports Swift and Objective-C, there are two objects that conform to the *CodeGenerator* protocol, namely *SwiftCodeGenerator* and *ObjectiveCCodeGenerator*. Some features require to generate multiple code snippets placed in different parts of the template. This **modular architecture** allows each code generator to specify multiple private functions to generate complex code.

Because of this architecture, many **unit tests** have been written to make sure that the most important part of the project works as expected. In fact, all objects responsible to generate the

[11] https://en.wikipedia.org/wiki/Objective-C

configuration source code are 100% unit tested. Future engineers working on the project will have an higher level of **confidence** in refactoring and adding new features to the codebase.

Save Configuration

The final configuration source code has now been **generated successfully** and written inside the documents directory of the application. The configuration source code is **well commented** to allow customers to apply their own customizations very easily. There are now three options for the user to browse and inspect it as shown in figure 13:

- The code can be previewed directly on the device inside a web view that provides syntax highlighting. This lets users quickly see the content of the generated code and inspect how a specific method or feature is translated into source code.
- The configuration file can be shared with another device to be integrated directly in the customer codebase for example.
- The configuration can be tested inside the application in order to verify its correctness.

Fig 12. Code Generation step.

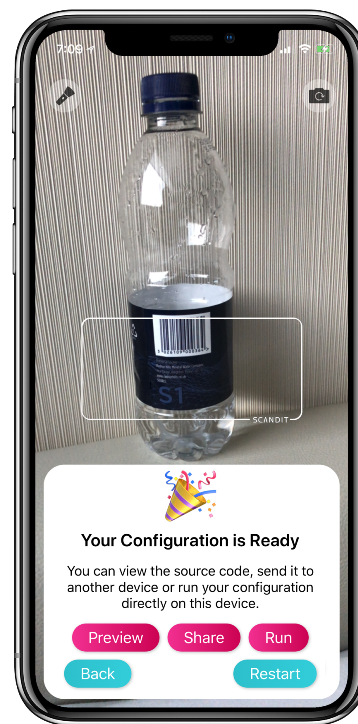


Fig 13. Save Configuration step.

Evaluation

The evaluation of the project has been **ongoing** since the beginning. Since we decided to build a minimum viable product during the first stages of the project, we were able to gather feedback by organizing an **internal user testing session** with two Scandit employees unaffiliated with the project.

This session was very useful to asses the quality and ease of use of the project for new users. The overall feedback was very positive regarding the user experience and utility of the project. A few remarks that testers made were:

- The navigation didn't allow going back to a previous step if a mistake was made. I immediately added the button since it makes sense to be able to modify a previous choice.
- During the session, I noticed that it wasn't very clear how many and which codes the scanner recognized during the scan symbologies step. For this reason, a counter for each code symbology is now displayed.
- The generated code didn't contain any documentation to help the customers in navigating it. Therefore, I documented each part of the generated code.

Thank to the feedback received, I applied the required modifications to improve the user experience. The project was demoed to many Scandit employees who all seemed to really like the final result.

Conclusion

I am very happy with the state of the project and I am confident that customers will find great value in the Configurator app. It is expected to increase customers happiness and reduce the Scandit SDK integration time. During this project I improved my Swift knowledge and learnt to build a robust, testable and flexible architecture while increasing my expertise in regards to barcode scanning.

The project will be extended with some improvements in the future. One of them will consist in a new question that asks the user to select their use case for the configuration. Some common use cases are single barcode scanning, barcode selection (scanning of small barcodes) and multiple barcodes scanning. By requiring this information at the beginning of the flow, the application will then adapt even more and allow the user to only customize specific features of the selected use case. Thanks to this change, the configuration generation will be even more effective and less time-consuming for customers.

In the future, Scandit will decide the best way to deploy the project to customers interested in integrating the Scandit SDK in their own applications.

Appendices

I would like to thank Prof. **Michele Lanza** for allowing me to work on this project and following its development during the semester.
I would like to also thank all my colleagues at Scandit who contributed to the project:

- **Iciar Martinez**, Product Manager: Iciar defined the project requirements from the beginning and she was the point of reference in regards to all product decisions.
- **Luca Torella**, iOS Software Engineer: Luca reviewed the code throughout the project and helped me better understand the behavior of the Scandit barcode scanner.
- **Marco Biasini**, Software Engineer: Marco helped to make the right technical decisions related to the Scandit SDK during the initial phases of the project.