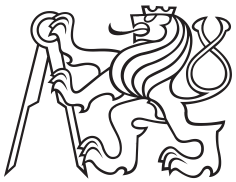
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Digital shopping cart platform with facial recognition

Xuan Anh Nguyen

Supervisor: Ing. Jan Hauser
Field of study: Software Engineering and Technology
May 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Nguyen** Jméno: **Xuan Anh** Osobní číslo: **488036**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Digitální nákupní košík s rozpoznáváním obličeje

Název bakalářské práce anglicky:

Digital Shopping Cart Platform with Facial Recognition

Pokyny pro vypracování:

1. Porovnejte existující přístupy k tvorbě multiplatformních mobilních aplikací a vyberte ten nejvhodnější pro realizaci scénáře elektronického nákupního košíku.
2. Seznamte se s problematikou rozpoznávání obličeje a využívání kamer mobilního telefonu pro skenování obličeje zákazníka.
3 Navrhněte architekturu platformy elektronického nákupního košíku včetně subsystému pro rozpoznávání obličeje.
4. Naimplementujte MVP elektronického nákupního košíku, který bude umožňovat přidávání zboží pomocí skenování čárového kódu a spravování obsahu košíku.
5. Vyhodnoťte řešení s pomocí uživatelského testování a survey na adekvátním vzorku zákazníků, minimální velikost testovaného vzorku je 10 zákazníků.

Seznam doporučené literatury:

[1] P. Harrington, Machine Learning in Action, Manning Publications, 2012
[2] A. Boduch and R. Derks, React and React Native: A complete hands-on guide to modern web and mobile development with React.js, 3rd Edition, Packt Publishing, 2020
[3] S. Z. Li and A. K. Jain, Handbook of Face Recognition, Springer, 2011
[4] K. P. Murphy, Probabilistic Machine Learning: An introduction, MIT Press, 2022
[5] K. P. Murphy, Machine Learning: a Probabilistic Perspective, MIT Press, 2012
[6] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, Third Edition, Addison-Wesley Professional, 2012
[7] F. Schroff, D. Kalenichenko and J. Philbin,FaceNet: A Unified Embedding for Face Recognition and Clustering, IEEE,2015
[8] inVerita, Flutter vs React Native vs Native: Deep Performance Comparison, Medium, 2020

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jan Hauser katedra řídicí techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **08.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Jan Hauser podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

Acknowledgements

I want to thank my thesis advisor Ing. Jan Hauser, for his support, advice and time despite his work schedule. Additionally, I want to also thank Ing. Martin Ledvnika for his feedback in the previous semester. Lastly, I want to thank my coworkers and Applifting for their help and its great culture. Special mention to my family and friends for their continuous endurance and encouragement.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 20, 2022

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 20. května 2022

Abstract

The aim of this thesis is to create a solution to Applifting's food and drink purchase process by developing a hybrid mobile application. A hybrid application allows developers to build software supporting Android and IOS platforms with a single codebase. This application enables users to authenticate with their faces and purchase products with barcodes. The target users are employees of Applifting company, but it can also be used in other mid to large-sized companies that offer products to their employees.

At the beginning of this work, we analyse requirements and technologies used to improve the shopping process. After the analysis, we describe the design and implementation of the application. The end result is evaluated by user testing sessions and conducting user surveys on ten people. The result showed the employees' satisfaction with the effortless solution, and their willingness to use the product.

Keywords: shopping cart system, face recognition, barcode scanning, hybrid mobile application, React Native

Supervisor: Ing. Jan Hauser

Abstrakt

Práce se zabývá vytvářením hybridní mobilní aplikace, která řeší problém nákupu nápojů a jídla firmy Applifting. Hybridní aplikace umožňuje vývojářům vytvářet software podporující Android a IOS platformy z jediného zdrojového kódu. Účelem aplikace je poskytnout uživateli přihlásit se pomocí obličeje a skenování produktů pomocí čárového kódu. Cílovou skupinou jsou zaměstnanci firmy Applifting, ale řešení je také možné uplatnit ve středně velkých až velkých firem, kteří nabízí svým pracovníkům prodej produktů.

Úvodem práce jsou analyzovány požadavky a technologie potřebné k vylepšení nákupního procesu. Po té popisujeme návrh, implementaci a testování aplikace. Zhodnocení úspěchu je získáno pomocí uživatelských testů a dotazníků na vzorku deseti lidí. Výsledek práce ukázal zájem o budoucí použití a spokojenost s rychlejším řešení.

Klíčová slova: systém nákupního košíku, rozpoznávání obličeje, skenování čárových kódů, hybridní mobilní aplikace, React Native

Překlad názvu: Digitální nákupní košík s rozpoznáváním obličeje

Contents

Figures

Tables

Chapter 1

Introduction

Every aspect of our lives is getting more digitised. In the past, we used simple tools such as paper and pen to record information, and as the years passed, we replaced those with digital versions that can accomplish more. The same is true for the company Applifting, which started by using paper and pen to record the purchases of foods and drinks for its employees.

As the company grew, so did the need to improve the tools to manage the inventory better. The first solution was a shopping cart system embedded in the company's internal web application, followed by an Android [1] and later IOS application. Nevertheless, issues occurred when the company found a significant discrepancy between the available products and registered purchases during the monthly inventory check. Another problem happened after the release of the Android application when several employees who owned IOS devices couldn't use the app and had to wait for the release of the IOS version. Later, there were inconsistencies in features offered by the app on the two operating systems, where one platform was ahead with some parts over the other.

This thesis aims to implement a digital shopping cart for mid to large-sized companies like Applifting that offer food and drinks to their employees. The finished product is a mobile application installed on a device located near the food and drinks bar. Any employee can access the device to open the application where they either register a new account or log in to an existing one to purchase a product. During the registration process, it's necessary to connect the user's identity from the company's internal system to our application. After the authentication, the employee takes out their desired product and scans the barcode from the device's camera, adding it to the cart. Later the worker completes their purchase in the checkout, while the pending transactions get registered for later payment.

This thesis starts by analysing the existing technologies and alternatives for application development and facial recognition, where we look at their advantages and disadvantages. Next, we design the architecture of the

whole application, from mockup prototype to visualisation of the design of mobile app, the frontend, to the backend, which will serve as a microservice communicating with the company's internal systems and distribution solution. Following the design, the next section is the implementation focusing on efficiently solving specific problems or discussing possible inefficiencies. Lastly, we evaluate the solution through user surveys and user testing.

Chapter 2

Analysis

In the introduction chapter, we described the background of the solution and mentioned the necessary features that the application should have. As a next step, this chapter specifies the requirements in more detail. It then analyzes the technologies used in the implementation, looking at existing solutions and their tradeoffs.

2.1 Requirements

The requirements were based on the company's previous solutions and our suggested improvements. Those will be later used when evaluating if it satisfies our initial goals.

2.1.1 Functional requirements

In software development, functional requirements define specific actions that a user can accomplish [2].

1. **Face authentication**

   The system allows the user to login with their face.

2. **Connect identity to account**

   The system allows the user to connect their identity from the company's internal system with their account.

3. **Barcode scanning**

   The system allows the user to find a product by scanning its barcode.

4. **Product search**

   The system allows the user to search for a product.

5. **Shopping cart**

   The system allows the user to add and modify products in their cart.

6. **Register purchase**

   The system allows the user to purchase products in the cart into the company's internal system.

2.1.2 Non-functional requirements

Non-functional requirements, on the other hand, define the behaviour of the system [3].

7. **Multiplatform support**

   Works on Android and IOS.

8. **Simpler development**

   It does not take too much resource to develop.

9. **Usability**

   It's simple to use and understand.

2.2 Use cases

From the user's perspective, a use case it's a list of activities that end to a particular goal, heavily reliant on functional requirements but more in detail.

1. **Registering the user face**

   The user binds their face with the company's internal account, registering their face and profile picture to the system, resulting in account creation.

2. **Shopping products**

   The user scans a product or searches it from the list of products and adds it to the cart. He can then modify the cart quantity, which he checks at the checkout screen at the end of the shopping flow.

2.3 Mobile applications

Mobile app development is a process of developing software applications for smartphones (mobile or tablet) [4], smartwatches, and even cars. The majority of the market is divided into two platforms, IOS and Android [5].

Historically development for each platform was enabled by their dedicated native language. To create a successful application, you need to have enough capable developers to support both platforms. But with the rise of demand for a cheaper alternative came hybrid apps, which solved this issue. This section analyzes and compares different approaches to mobile development.

2.3.1 Native mobile development

Native applications are written in the native language used for the device platform. For developers, on the Android platform, it is Kotlin and Java [6], while on the IOS platform, it is Swift and Objective-C [7]. Apps are primarily developed using a dedicated SDK (software development kit) provided by specific tools. They are released on app stores such as Google Play or App Store. Those stores serve as a distribution channel for users to allow them to install the apps.

Generally, the main advantage of developing native applications are:

- Faster and efficient utilization of performance since it has the best integration with the OS hardware.
- Usage of newest features, that the OS newly introduced.
- Ease of development, because the platform provides specific components.
- Consistent UI that matches with the user experience on the platform.

2.3.2 Native mobile development drawbacks

While native mobile applications give you the best solution that can fully maximise all OS features, it requires more development resources because of platform customisation. Further disadvantages include:

- Multiple codebases.
- Time-consuming and costly to develop, especially for smaller businesses at the beginning.
- Managing the consistency between the feature set.
- The necessity of owning a device that has Xcode for IOS development.

2.3.3 Hybrid mobile development

Hybrid applications combine the approaches of native and web technologies. Like native applications, they can utilize some of the OS features, being distributed through an app store and still function offline without access to the internet. But the most significant difference is that they offer a common programming language and development environment with the "write once run anywhere" concept in mind [8], similarly to the web applications. In the following sections, we will explore existing hybrid mobile development frameworks.

Ionic

Ionic is an open-source framework targeted for building hybrid mobile applications with Cordova. It was created with belief that HTML5 would become the new standard of native applications in the future, just as it is now on the desktop [9].

At its core, it's just a web page running in a native app shell like iOS's UIWebView or Android's WebView, which Cordova wraps. Cordova enables the application to use numerous mobile device services like permissions or hardware capabilities via the Cordova Javascript API [10]. For this reason, web developers shouldn't have much trouble building Ionic apps with the knowledge of CSS, HTML and Javascript.

React Native

React Native is an open-source framework created by Meta that lets developers build mobile applications using ReactJS, a well-known Javascript framework in the web ecosystem [11]. Additionally, it has been one of the most popular cross-platform solutions with a mature library ecosystem for the past years (viz. Figure 2.1).

Looking at how React Native works internally, it sends asynchronous messages to the native widgets APIs [13] in order to render the appropriate user interface from the React component.

In an example (Listing 2.1), while using a React Native app, two threads are communicating through a bridge with each other under the hood. The main thread represents the native side that handles displaying elements and user interactions. The second one runs on the Javascript VM used to manage the business logic and declare the user interface [14].

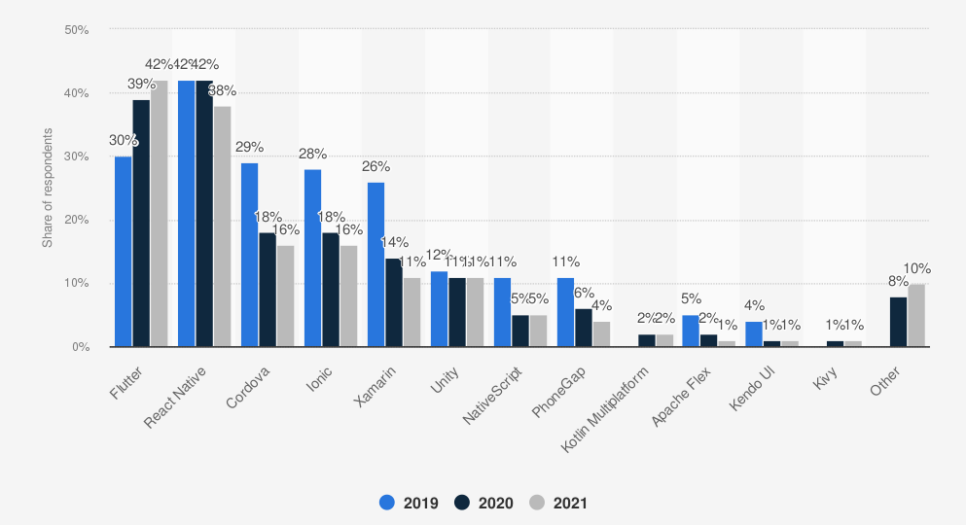


Figure 2.1: Visualisation of most popular hybrid applications solutions [12]

```
import React from 'react';
import { View, Text } from 'react-native';

export const Home: React.FC = () => (
 <View style={{
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center'
 }}>
   <Text>Example text</Text>
 </View>
);
}
```

Listing 2.1: Code example showing a text on a screen

We defined a Text user interface inside our code example using the Text React component. The Javascript codebase relays a serialized message through the bridge to the native thread instructions on what view and text to load. Similar logic applies if a user is interacting with a button, the native thread registers an event which is again serialized and sent to the Javascript thread so it can respond with an appropriate message.

Flutter

Flutter is an open-source project created by Google [15]. It consists of the Flutter framework, a UI library containing various reusable UI elements, and

the Flutter SDK, a set of tools to develop an application and compile code into the native code of IOS or Android.

Unlike other frameworks, there isn't any bridge to convert components into their corresponding native version or a need to render the UI on a Webview. Instead, Flutter is responsible for drawing pixels directly on the screen, making UI updates faster than Flutter's alternatives.

Flutter works similarly to a game engine. At a high level, a Flutter application is composed of widgets (a part of the user interface) that are rendered onto a Skia canvas, a graphic engine written in C++ that sends commands to the CPU and GPU [16]. The platform then takes the modified canvas and shows it to the user while sending events back [17].

2.3.4 Hybrid mobile development drawbacks

While there are many solutions on how one should implement a cross-platform application, each of the previously mentioned ones has its advantages and disadvantages. Nevertheless, all of them share some same pitfalls.

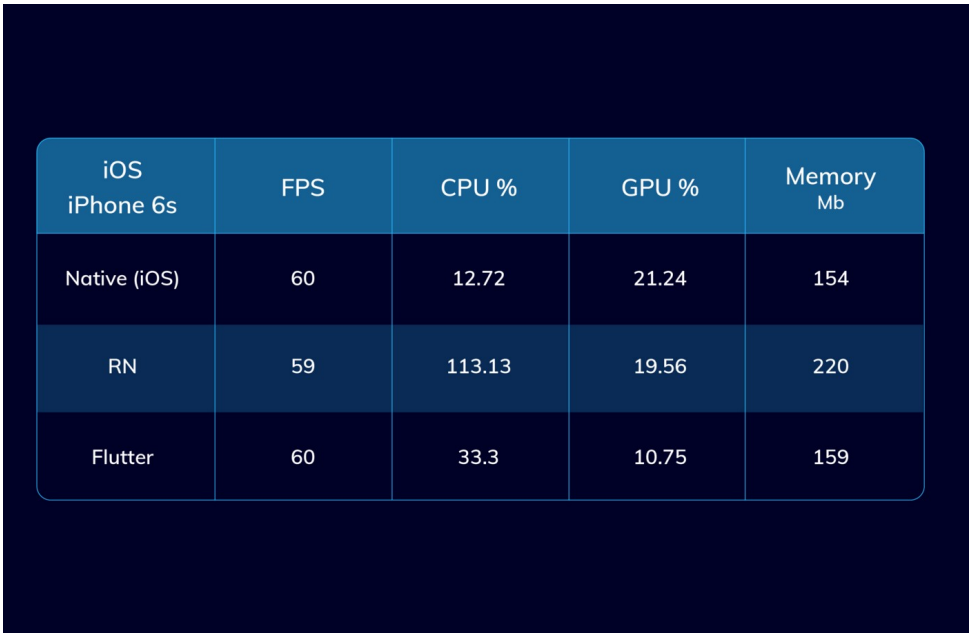
Here are some potential problems occurring in hybrid applications:

- New OS APIs are not available until maintainers of the frameworks support it.
- Possible problems with particular Native functionalities like Payment and FaceID.
- Battery usage is not optimized.
- Sometimes heavily relies on the ecosystem for needed features that are usually already available in the native development.

2.3.5 Our choice

After understanding the core concepts of each solution for hybrid applications, we are going to pick the best one based on certain criteria:

- Performance
  - Because our application does not have heavy animation requirements, a simple list view benchmark will be appropriate. Figure 2.2 contains the test results for the performance of scrolling inside a listview on iPhone 6. All solutions show the same FPS, but a clear difference can be seen in React Native, where the percentage of CPU operations is higher, resulting in a bigger memory consumption [18].



iOS iPhone 6s	FPS	CPU %	GPU %	Memory Mb
Native (iOS)	60	12.72	21.24	154
RN	59	113.13	19.56	220
Flutter	60	33.3	10.75	159

Figure 2.2: iPhone 6 results on a ListView test comparing a Native, React Native and Flutter app [18]

- Testing experience
  - When releasing an application outside the development environment, one would need to publish it on an app store, which requires a lengthy acceptance procedure from the provider. But for testing purposes, exporting a codebase in a fast and simple manner is crucial during the iterative development. That's why testing tools are used.
  - In the native application ecosystem, developers use TestFlight for IOS and Google Play Testing for Android. For Hybrid applications, there is only the Expo library on React Native [19]. It offers an app that can pull the newest codebase without a need to plug the device into the computer, and you can share a QR code with the users so they can install and test the app without any complicated setup.
- Development adaptation
  - Except for Flutter, only React Native and Ionic can be written in a language well-known to web developers, which makes hiring developers that can work with this easier.

Reflecting on those points, I choose React Native as a solution for hybrid application development, even though Flutter has a better performance score.

2.4 Usage of device camera in face authentication

Nowadays, mobile devices offer their user face authentication to access the device. This phenomenon is most noticeable for iPhone users with the introduction of FaceID. To implement this functionality, two different approaches could be used:

1. **Normal camera**

   With the recent improvements in face detection technology, applying machine learning algorithms to images is one of the simplest way to solve this problem. But this comes with a limitation by relying on the device's camera quality. In addition, the light source of the surroundings can influence its results.

2. **Depth camera**

   A particular type of camera which can extract 3D information from the image by projecting infrared light. An example of this is the TrueDepth camera used in FaceID on iPhone devices [20]. This gives more detailed information about the face, inputted into a machine-learning algorithm to verify the user. The benefit of this approach is that it's less susceptible to light and image noise.

For our application, we considered using the normal device camera to support both Android and IOS devices.

2.5 Face authentication

Authentication is a process of confirming if a user is who they claim to be [21]. It's a crucial entry point before the user can interact with any service. The most common type we use is email and password-based authentication, but it's becoming more frequent to have biometric authentication above it. Part of the reason is that they are more convenient and secure [22]. Instead of remembering the password, it relies on the user's biometric information such as face or fingerprint.

2.5.1 Introduction to face authentication

This section describes how a face authentication system can work and be implemented. We have an application with a video stream that authenticates a registered user after showing their face on the camera. Firstly, the camera takes an image shot and sends it to the system to evaluate it. Then it applies

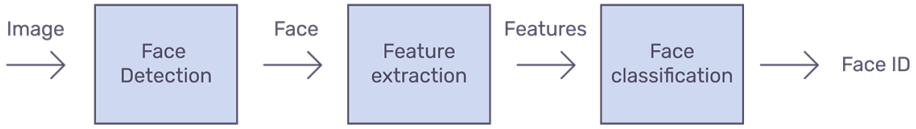


Figure 2.3: A face authentication pipeline [24]

face detection, a process where the face position is detected to improve the accuracy in the extraction part. This can be implemented with various pattern recognition algorithms [23]. The resulting cropped face image is attributed to another process called face recognition, where we extract critical features of the face. Those key features define the uniqueness of each person, which can be in the format of an N-dimensional vector. The last step determines who the person is from the face embedding by comparing it with stored faces, and returning their identification if found. An illustration of the whole process is shown in Figure 2.3 .

2.5.2 Solutions to face authentication

Commercial

- Microsoft Azure [26].
- AWS recognition [27].
- Kairos [28].

Open-source

- Google FaceNet [25].
- Insight Face [29].
- DeepFace [30].

2.5.3 Our choice

After consideration, we dismissed the option of commercial solutions. Even though they offer great scalability and data protection, the cost per usage didn't look acceptable to us. The open-source options have excellent accuracy on the LFW dataset, a face recognition benchmark tested on live face

images. Moreover, it has REST API support. In the end, we decided to use Google FaceNet, a machine learning model, because of its accuracy score and popularity in the ecosystem.

2.6 Barcode scanning

Barcode scanning is a simple problem. Since it's an old technology used in the past in the retail industry, many devices exist that can fastly and reliably scan products. Those principles are used in software libraries for mobile applications.

The implementation of barcode scanning works by applying digital image processing techniques to the image. This involves some noise filtering, sharpening and contrast-enhancing. Finally, the data is extracted and then evaluated [31].

As a solution, we chose to use the Expo barcode scanner [49], which is part of the Expo Camera library.

Chapter 3

Design

The following chapter describes the implementation design. From the used technology, the architecture, which gives us an overview of how each part will look like, communicate and work together, to the user interface prototype.

3.1 Selected technologies

Several factors need to be considered while choosing a technology for the implementation. In this section, we describe a summary and reasons on which and how the technologies were chosen.

Flask

Flask is a lightweight web framework written in Python [32]. Compared to Django, another popular Python framework, it requires less knowledge and effort to use. Because of our requirement to implement facial recognition, it's common to have server technology that can utilize Python libraries.

Tensorflow

One of the most popular libraries used in machine learning projects, Tensorflow offers comprehensive and flexible tools for working with machine learning models [33]. In our case, it simplifies the development with the Google FaceNet model.

CV2

Also known as OpenCV, it is an open-source computer vision library. OpenCV was built to provide a common infrastructure for computer vision applications [34]. The library is very helpful for image processing while being very efficient. Some of the most notable usages of the library's algorithms are tracking moving objects, recognising faces and extracting 3D models from video.

Pickle

It is an object serialisation library accessed from the Python library. The model works by converting a Python object into a byte stream stored in a Pickle file [35]. In the implementation chapter, we will describe how we use this file format to save face embeddings.

Expo Camera

Part of the Expo ecosystem, Expo Camera provides a React component that renders a preview for the device's front or back camera. Moreover, the component can also detect faces and barcodes appearing in the preview [36]. The implementation of face detection is based on the Firebase ML kit.

Nginx

It is a web server that allows the user to serve static content and manage request proxying. With Nginx, you can configure and manage SSL certificates. This is very useful in securing the communication between the client and server through HTTPS [37].

Gunicorn

Gunicorn is a WSGI server that can run any web application. WSGI is a protocol to standardise communication between a web server and a web application [38]. For this reason, we will need Gunicorn to bridge the communication between Nginx and Flask server.

3.2 Architecture

The project gains better modality and efficiency in achieving its outcome with good architecture planning. We will better understand it by breaking it into

smaller parts and defining how they are structured together. The following section describes the system components and used architecture patterns.

3.2.1 3-Tier architecture

A three-tier architecture is a client-server architecture in which the functional process logic, data and user interface are developed as independent components. These well-defined boundaries give developers more flexibility and freedom in application design. As a result, long-term software maintenance improves [39].

The architecture consists of these parts:

3.2.2 Application server

The application server is implemented with the Flask framework written in Python language. The purpose of an application server is to process incoming HTTP requests from the client. Moreover, it has no ORM or auth guards, only a representation of a DB layer and some business services. In the following section, we introduce the company's internal system and describe available services.

Applifting internal system

It is an external system with a product and purchase API with a REST interface. Accessing its resources requires an authorization token passed in the header of each request. Only the employee of Applifting can retrieve such a token by requesting it on the company's internal website. We will use this behaviour during the user registration process on our mobile application. We will ask users to read the authorization token from the website and pass it to your system, where it's saved together with the face feature in our database as a key-value object with the token being the key.

Product service

The job of this service is to access products and modify purchase data. Because of the company's internal system conditions, an authorization token will be required. The user can acquire this token after the registration and login. The server retrieves the token from its database service. It sends it to the client, where he stores it internally until the shopping is completed. For example, when the mobile application needs to display some product information, an HTTP request is sent with the stored token in the header,

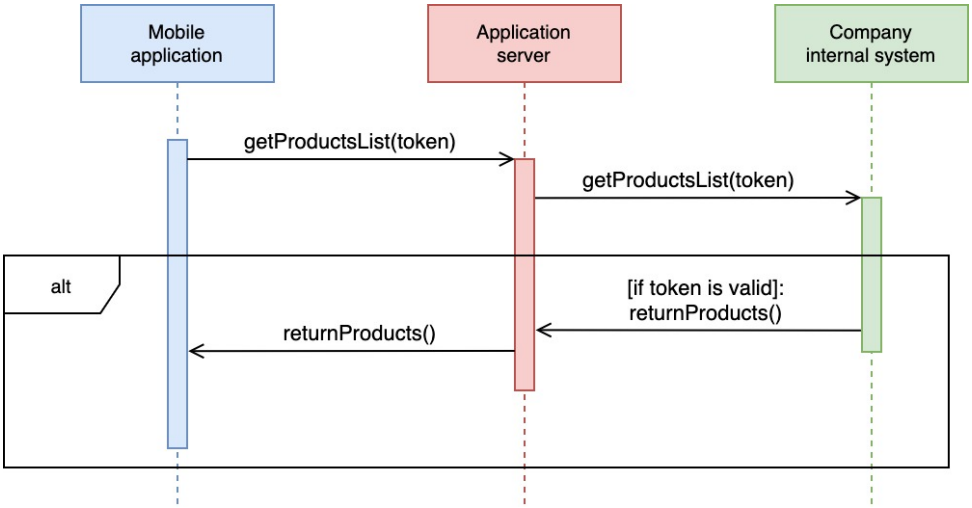


Figure 3.1: Sequence diagram of product list display

which is set again in another HTTP request to the company's internal service. An example of retrieving products and displaying them to the client is shown in Figure 3.1 .

Face recognition service

Its responsibility is to execute facial recognition during authentication with the help of the Google FaceNet model. The model is preloaded and accessed as a singleton during the server initialisation.

3.2.3 Mobile application

The client-side of the system is a React Native mobile application written in Javascript. The user interacts with the mobile application that communicates with the application server to retrieve and process information. The following text clarifies the React architecture and the face recognition process.

React Native architecture

Typically some native applications use an MVC pattern to organise their codebase and functionality, but with React Native, it's different. Because it's part of the React JS library, its structuring is also similar, mainly component-based. The data model, business logic, and interface can be found together in components but with a difference in the division. React components were meant to be reusable, so it's usually organised by functionality.

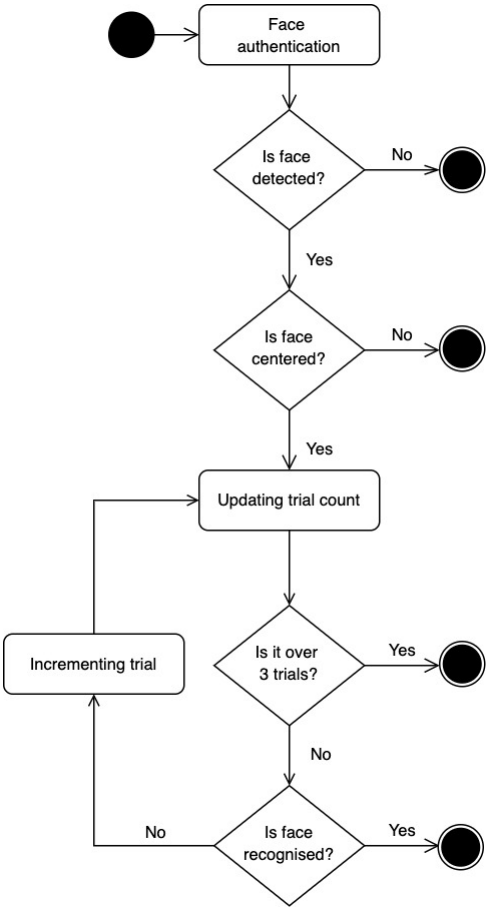


Figure 3.2: Process diagram of face authentication on the client before an image is sent to the server

Face recognition process

Although the face extraction can be done in the backend, sending images frequently from the camera to the server to evaluate a face isn't an optimal solution, so a better approach was considered. Using the face detection API from the Expo camera library, we can pass this task to the client-side while sending the best picture to reduce the server work.

There are three mechanisms when the mobile application sends an image during face authentication (viz Figure 3.2). Firstly, only on face detection is an image sent to the server. Secondly, the face position is checked if the location is in the center of the preview, making the sent images more consistent. Finally, after three unsuccessful trials, the user cannot further progress in the authentication flow.

3.2.4 Database

A database is an organized collection of structured information stored in a DBMS (database management system) [40]. Databases allow us to retrieve or store data consistently and efficiently. The most popular databases are SQL databases, which are structured into a group called tables, and NoSQL databases like Redis, and MongoDB, which are optimized for different problems that SQL primarily does.

Since we are working with machine learning, we needed to consider the problem of storing face recognition results and the underlying process of identifying the user. As mentioned in Chapter 2, during the sign-in operation, the first two face recognition steps are executed, outputting the face features in the format of an N-dimensional vector. For the third step, there are two possible ways to implement it, either we store the images of the registered users and apply face extraction for each person when comparing with the user trying to login. Or the other approach is to directly store the face recognition results, which removes the need to reapply the extraction process per registered user. For this reason, we decided to use a Pickle file as a storage solution.

3.3 Data model

Our application doesn't directly hold any data. It instead uses the company's internal system to read and modify product and purchase data. Nevertheless, we can describe a representation of the data model from the product service and database service. The data model is represented with an UML diagram in Figure 3.3 .

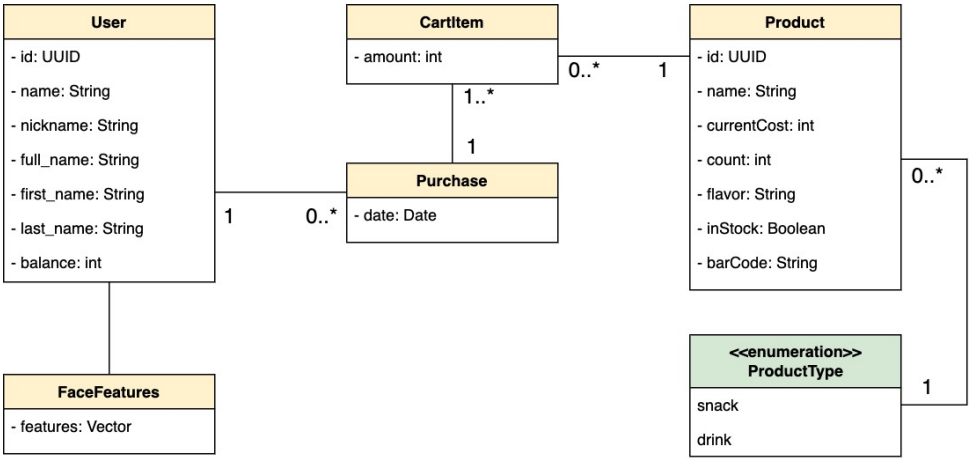


Figure 3.3: UML diagram of the application data model

3.4 Deployment diagram

A deployment diagram is a UML diagram used to visualize the devices of a system, the communication protocols between them and the placement of sub-components inside those devices [41]. This helps you to understand the physical deployment of the hardware machine.

Figure 3.4 shows a deployment diagram of our digital shopping cart system. A request is sent to our backend starting from the React Native mobile application. Behind it, an Nginx service reverse-proxies this request to a dedicated port via a UNIX socket, separating our application from the world for security reasons. After catching this request, the Gunicorn service invokes the correct code of the Flask server, returning a response.

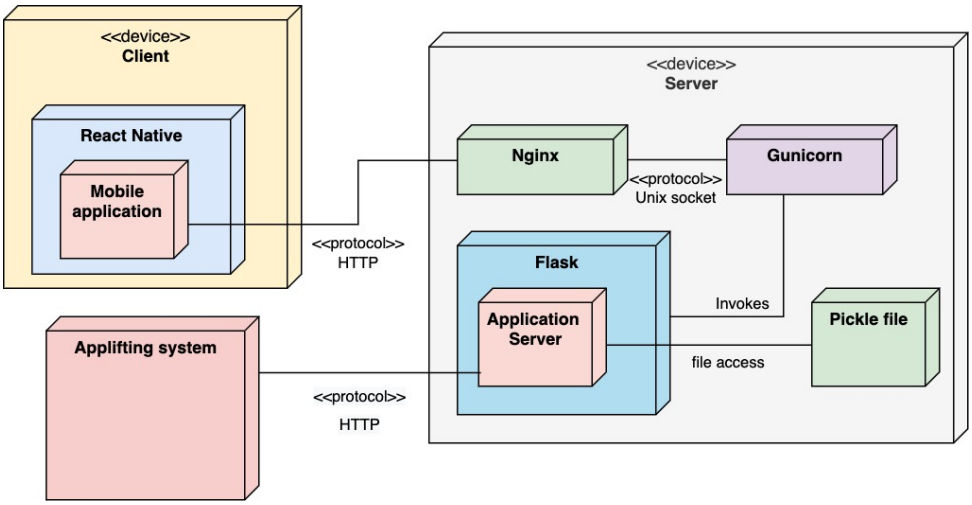


Figure 3.4: Deployment diagram

3.5 User interface

3.5.1 Navigation

The following Picture x describes the navigation between screens in the mobile application.

3.5.2 High-fidelity prototype

A prototype is a visual representation of our product. There are two types of prototypes, high-fidelity and low-fidelity. Each one of them differs by the level of detail, making high-fidelity the most refined and closest to the actual

application. This section presents several actions accompanied by high-fidelity prototype screens and descriptions. A prototype is a visual representation of our product. There are two types of prototypes, high-fidelity and low-fidelity. Each one of them differs by the level of detail, making high-fidelity the most refined and closest to the actual application. This section presents several actions accompanied by high-fidelity prototype screens and descriptions.

Face authentication

The user looks at the device camera and tries to position his head to the circle on the screen and center it.

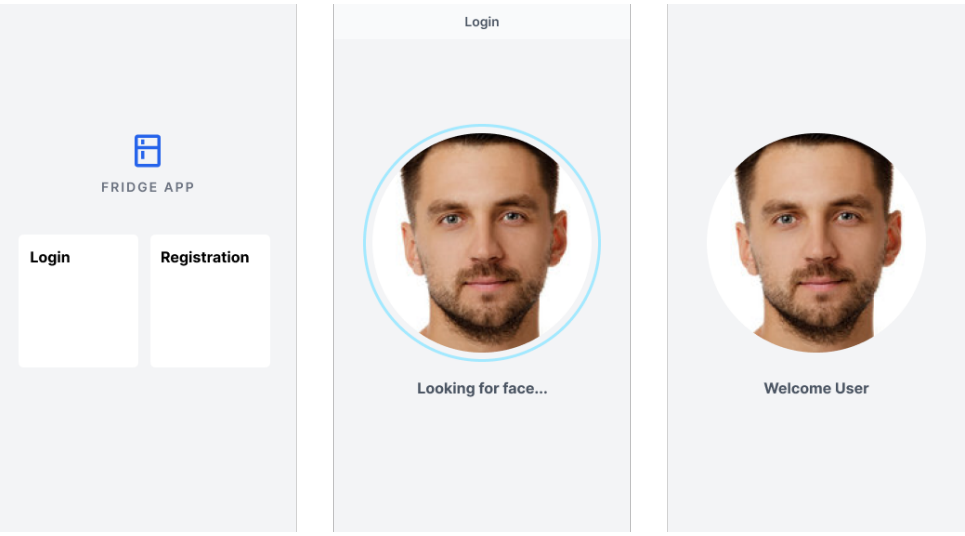


Figure 3.5: Screen for authethication

Registration

The user goes through the registration walkthrough to understand how to connect his company account to the digital cart shopping system. After passing the required steps, the system will take a user picture to complete the registration process.

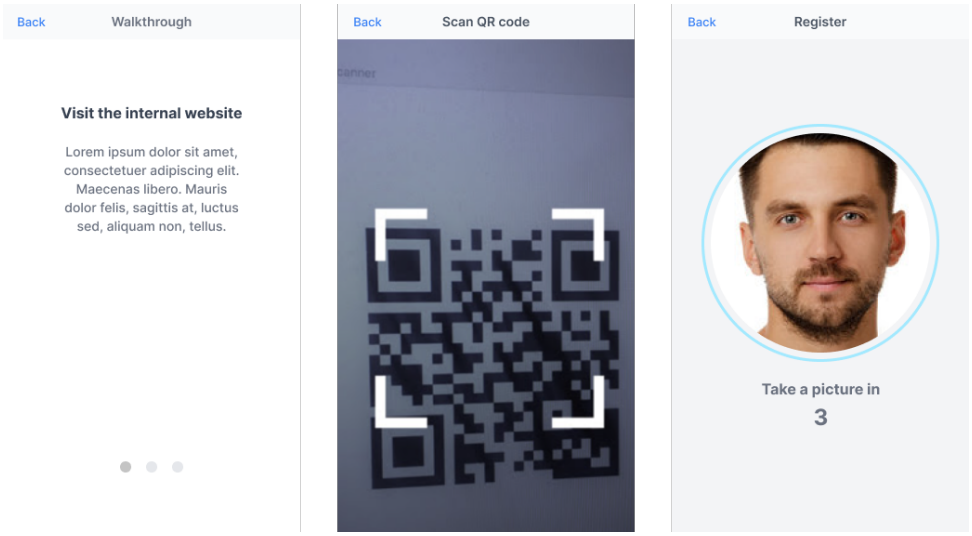


Figure 3.6: Screen for user registration

Buying products

The user scans the barcode of the desired product. A product preview is displayed with a quantity counter so the user can adjust the amount. After navigating to the checkout, the user views a summary of his current purchases and clicks on the complete purchase button, making this purchase registered.

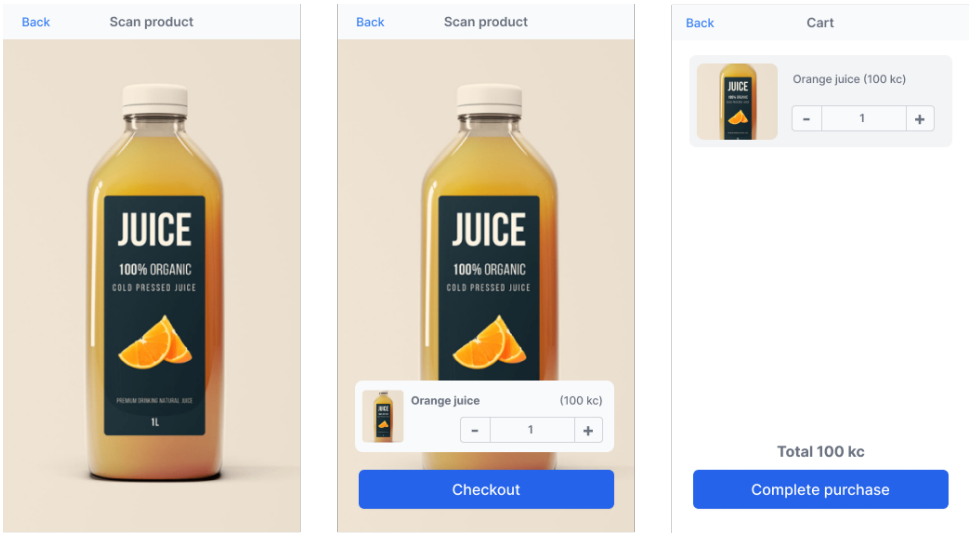


Figure 3.7: Screen for product purchase

Chapter 4

Implementation

This chapter details the main obstacles encountered during the implementation and project deployment process. Especially the problem of how to process an image and verify a user.

4.1 Face authentication

One of the main functionalities of your application is to enable face authentication for the users. As mentioned in the Analysis chapter, Google FaceNet is used in the implementation. There are still two crucial steps needed to make it fully functional. An image processing before the extraction and face verification.

4.1.1 Image processing

The Google FaceNet model requires a specific type of image because it was trained with photos having a square size of 160x160 pixels and pixel values standardized across all three channels [43].

When we receive an image from the mobile application, we first convert the color scheme to RGB if it is different.

```
def read_image(img):
   img = cv2.imdecode(numpy.fromstring(img.read(),
   numpy.uint8), cv2.IMREAD_COLOR)
   img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
   return img
}
```

Listing 4.1: Reading image and setting the RGB color scheme

Next, the converted result gets cropped with MTCNN, a face detection library, to output an image with only a face.

```
def crop(img):
   try:
       det = mtcnn.detect_faces(img)[0]
       margin = int(0.1 * img.shape[0])
       img = crop_bb(img, det, margin);
       return img
   except Exception as e:
       raise Exception("No face found")
}
```

Listing 4.2: Cropping the image with the MTCNN library

Finally, the image resizes and applies pixel standardisation, called zero centring [44]. This technique processes the image so that the mean of the image lies at zero.

```
def process_face(img):
   img = cv2.resize(img, (160, 160))
   img = img.astype('float32')
   mean, std = img.mean(), img.std();
   img = (img - mean) / std
   return img
}
```

Listing 4.3: Image resize and pixel standardisation

4.1.2 Face verification

When the face gets extracted, the result will be used to find the most familiar face of a registered user that satisfies a requirement. The model was trained so that the output vectors of similar looks are close to each other and at the same time, the vector location of different faces is far away [25]. To determine face similarity, a Euclidean distance function is applied on each face. If the distance between those two vectors is lower than our specified threshold, it results in a match.

The specific value for the threshold was chosen by trial and error on a custom sample (viz Figure 4.1). We tried to compare two persons with similar features during this test to determine a suitable threshold. The number seven was chosen, which worked well on future tasks.

```
dist between faces: 7.3649764
```

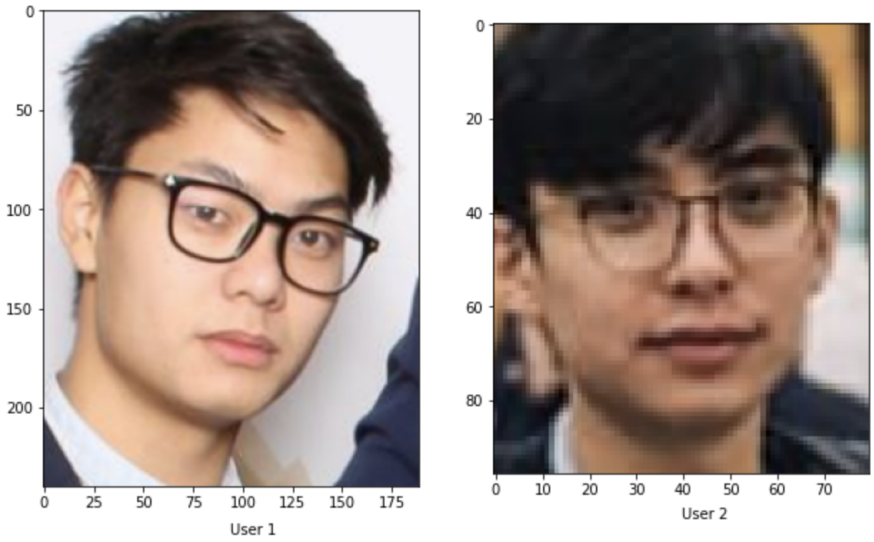


Figure 4.1: Test result of comparing the distance between two sample faces

4.2 Deployment

Deployment is the process of setting the environment in a machine to make it able to serve its functionality to other devices. This section describes which cloud providers we used and a summary of steps needed to make the application functionality for the client.

4.2.1 Digital Ocean

One way to host a server is to use dedicated cloud providers that simplify scalability and other aspects of machine hosting. Digital Ocean is one of those, an American company offering cloud tools and platforms to developers [45]. We used their product, Droplet, a Linux based virtual machine,[46], because it was simpler to set up than Amazon's EC2 from our experience.

4.2.2 Flask in production mode

Using the default running script for our Flask application in a local development environment is enough, but it's not recommended to use it in production mode. The reason is that Flask isn't designed in a scalable and secure way without the help of dedicated WSGI servers. For this reason, we use Gunicorn [42].

4.2.3 Deployment summary

After choosing a cloud platform, the next step was to set up the environment and services that will host our application:

1. Create a user group that will run our application server, giving it only access to specific folders like our Pickle file and image folder.
2. Configure Nginx and Gunicorn to run our application server.
3. Purchased a domain, and configure DNS to make it point to Digital Ocean.
4. Integrate SSL certificate with the Nginx plugin, securing our request with HTTPS.

Chapter 5

Evaluation

Evaluation is an integral part of software development. It aims at finding defects and checking the fulfilment of the requirements. If conducted incorrectly, users may stop using the application or misinterpret its usage before they can use the crucial part. There are many ways to do an application evaluation. Some of those techniques are user testing and surveys. This chapter outlines the result of our evaluation produced by the employees of Applifting. Our evaluation goals are to figure out if the requirements were met and find out potential improvements.

5.1 User testing

User testing is a qualitative evaluation method in which users directly participate [47]. A typical way to conduct user testing is to invite users to do some task on a product while observing and recording their actions and behaviour in order to identify design flaws.

The implementation of User testing requires specific preparation like defining the user group, test objectives, the testing medium and selections of tasks. In our case, the test will be conducted on employees of Applifting regardless of age or gender. Because a high fidelity prototype cannot fully showcase the functions of our application, we decided on the implemented application as a testing medium. Regardless of the tasks and objectives, two tasks were selected that reflect our thesis goals. The first one is to find out if our facial recognition is simple and reliable enough. The second one is product purchase which is the main activity.

5.1.1 Chosen user testing method

There is a specific method for user testing that is quick and simple, it's called the "Quick-and-Dirty usability Test". It focuses on rapid feedback on design instead of trying to create a routine and perfect test [48]. Little preparation is needed. It's done on anyone available, but ideally on people who represent our end-users. We choose this framework because it is simple, quick and efficient when one iterates the design frequently.

5.1.2 Results

Ten participants were given the same tasks and time duration. Most of them successfully purchased a product without any complications. Some users even complimented how the solution was faster and simpler than the previous ones. But there were several issues in the registration part. The following points were collected:

- Registration walkthrough was not clear enough: The users didn't know what to do after they attained the QR code that represents a token to access resources to the company's internal system. A rethinking of content will be considered and tested.
- Users didn't read the walkthrough text: Part of the reason was that the text size was small and significant sentences weren't noticeable to the users. This mistake was the easiest to fix.
- Users didn't notice the face enrollment countdown: After scanning the QR code, some users changed their focus to the surroundings when the face enrollment countdown started. We may add some more significant visual clues or sound to notify the user.

There were some suggestions from the users. The following points summarise some of them:

- Change the swipe component to multiple screens instead.
- Indicate steps during the registration walkthrough.
- Show a summary of transactions after purchase.

5.2 User surveys

In this quantitative analysis, the survey had two goals in my mind. Firstly, to measure the application usability (viz Figures 5.1, 5.2, 5.3). Participants

were asked to answer the questions on a scale from 1 (Very Bad) to 5 (Very good). The results for this first category were overall positive, but not without any negative answers. We expected the registration process to be the most complicated part from the user's perspective. A further rethinking of the registration walkthrough will be considered in the future.

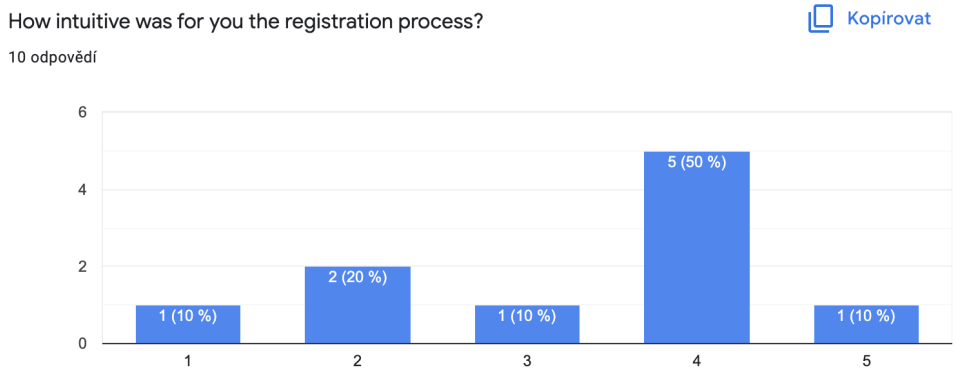


Figure 5.1: Registration usability evaluation

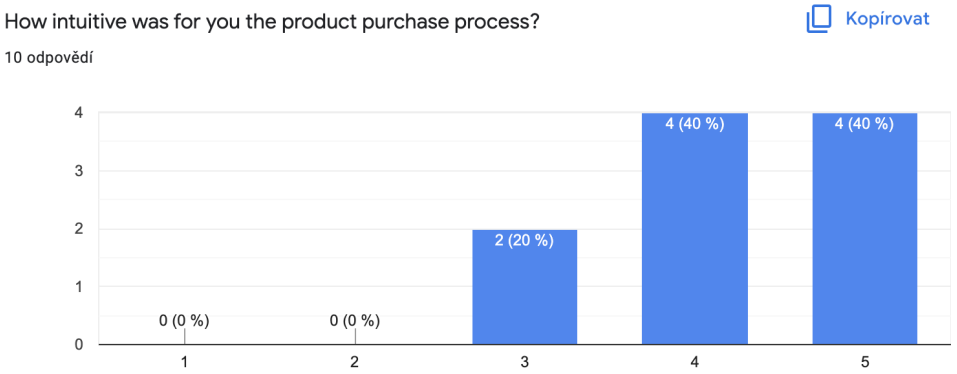


Figure 5.2: Purchase usability evaluation

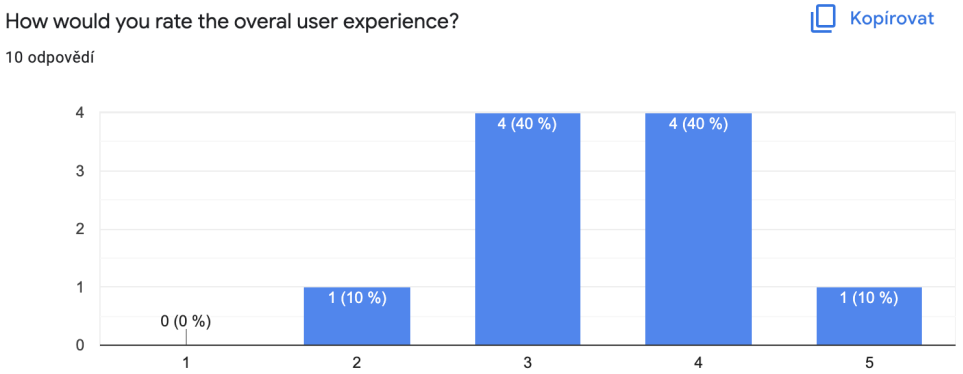


Figure 5.3: Overall usability evaluation

The second goal was to determine if the user would be willing to use our solution in the future or potentially replace the current one (viz Figures 5.4, 5.5). The answers were also on a scale from 1 (Never) to 5 (Definitely). Even though a high number of users were willing to use this application in the future, a smaller majority wanted to replace the previous solutions with this one.

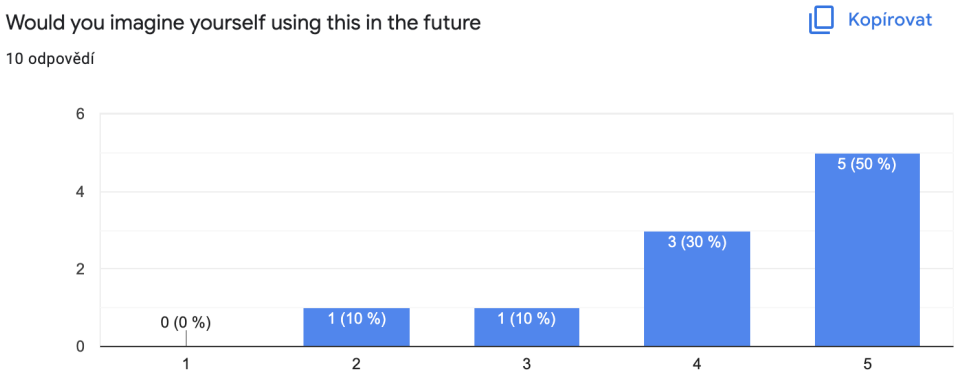


Figure 5.4: Future usage evaluation

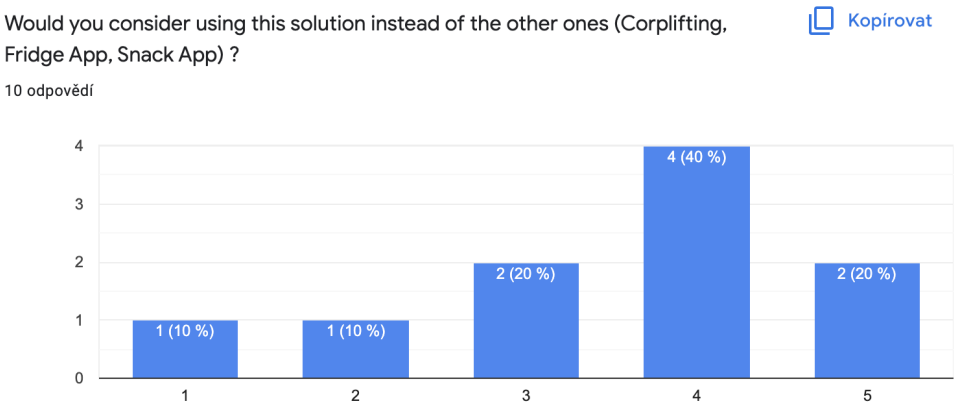


Figure 5.5: Replacement of competitor evaluation

The survey questions were structured in such a way that participants could complete them within 5 minutes.

5.3 Results

From the questionnaire, we concluded that there was a demand from the participants to use our solution in the future, even possibly replacing the current ones. Even with average results in usability, we are pretty confident that a future improvement on the registration part can change the results to complete satisfaction.

Chapter 6

Conclusion

The aim of this project has been to design and implement a technological solution to the problem of logging and purchasing food and drink items at Applifting, and it has been successfully accomplished. As demonstrated throughout this thesis, we followed the steps of software development, namely analysis, design, implementation and testing. The final product makes use of facial recognition technology to ease the users' shopping experience.

The analysis part defined the application's requirements, from which use cases were derived. Then different technologies for mobile applications, face authentication and barcode scanning were analyzed and compared. As for mobile application development, we choose React Native because of its familiarity to React JS and available testing tool that enables fast iterations for the developer. An open-source approach was considered for face authentication, using the Google FaceNet, a machine learning model with a solid score on the LFW dataset and broad adoption.

In the design and implementation part, we described the chosen architecture and the functionality of each of the components. This resulted in the implementation of a React Native mobile application and Flask server running together with Gunicorn and Nginx. The solution was deployed into the cloud on Digital Ocean.

From the evaluation part, a user survey was conducted showing the app's appeal to the potential users. A high percentage of those surveyed indicated their willingness to use the app in the future, as it made the purchase experience much quicker and easier. A potential issue concerned the accessibility of the app and the user guidelines. However, this can be remedied based on the feedback received from users.

Although the design and implementation of this app were trialled at Applifting, it is believed that it can be refined and adapted to the needs of other similar organizations. This is since the underlying technology used in the app is widely available to use and only certain aspects of the app such as the identity management system would require an update.

Bibliography

[1] Applifting s.r.o. (2022). *The Fridge*. (Version 3.0). [Online]. `https://play.google.com/store/apps/details?id=cz.applifting.fridge` (visited on 16. 5. 2022)

[2] Fulton R, Vandermolen R (2017). *Airborne Electronic Hardware Design Assurance: A Practitioner's Guide to RTCA/DO-254*. CRC Press

[3] Chen, Lianping and Ali Babar, Muhammad and Nuseibeh, Bashar (2013). *Characterizing Architecturally Significant Requirements*. IEEE Software - SOFTWARE

[4] Erfani, Mona and Mesbah, Ali and Kruchten, Philippe (2013). *Real Challenges in Mobile App Development*. International Symposium on Empirical Software Engineering and Measurement

[5] Maradin, Dario and Malnar, Ana and Đipalo, Ena (2020). *The Market Structure of the Smartphone Operating Systems Industry in the EU*.

[6] B. Abazi (2017). *Android Development with Java: Step by step guide to build applications*. CreateSpace Independent Publishing Platform

[7] Apple Inc. (2022) *Xcode*. [Online]. `https://developer.apple.com/documentation/xcode`

(visited on 16. 5. 2022)

[8] Enihe, Raphael and Joshua, Jimmy (2020). *HYBRID MOBILE APPLICATION DEVELOPMENT: A BETTER ALTERNATIVE TO NATIVE*.

[9] IONIC (2022) *All About Ionic*. [Online]. `https://ionicframework.com/docs/v1/guide/preface.html`

(visited on 16. 5. 2022)

[10] Apache (2022) *All About Ionic*. [Online]. https://cordova.apache.org/docs/en/latest

(visited on 16. 5. 2022)

[11] Meta Platforms Inc. (2022) *React Native*. [Online]. https://reactnative.dev (visited on 16. 5. 2022)

[12] Statistica (2022). *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021*. [Online]. https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours

(visited on 16. 5. 2022)

[13] A. Boduch and R. Derks (2020). *React and React Native: A complete hands-on guide to modern web and mobile development with React.js, 3rd Edition*. Packt Publishing

[14] Medium, Blagoja Evkoski (2017) *React Native: What it is and how it works*. [Online]. https://medium.com/we-talk-it/react-native-what-it-is-and-how-it-works-e2182d008f5e

(visited on 17. 5. 2022)

[15] Alphabet Inc. (2022) *Flutter*. [Online]. https://flutter.dev

(visited on 17. 5. 2022)

[16] Adam (2018) *How Flutter Works*. [Online]. https://buildflutter.com/how-flutter-works

(visited on 17. 5. 2022)

[17] Alphabet Inc. (2022) *Flutter architectural overview*. [Online]. https://docs.flutter.dev/resources/architectural-overview

(visited on 17. 5. 2022)

[18] Medium, inVerita (2020) *Flutter vs React Native vs Native: Deep Performance Comparison*. [Online]. https://medium.com/swlh/flutter-vs-react-native-vs-native-deep-performance-comparison-990b90c11433

(visited on 17. 5. 2022)

[19] Expo (2022) *Expo*. [Online]. https://expo.dev (visited on 17. 5. 2022)

[20] Apple Inc. (2022) *About Face ID advanced technology*. [Online]. https://support.apple.com/en-us/HT208108 (visited on 17. 5. 2022)

[21] TechTarget, Mary E. Shacklett (2021) *Authentication*. [Online]. https://www.techtarget.com/searchsecurity/definition/authentication (visited on 17. 5. 2022)

[22] Krishna, Dr and Talukdar, Fazal and Laskar, Rabul (2013). *Study on Biometric Authentication Systems, Challenges and Future Trends: A Review.*

[23] Gürel, Cahit (2011). *DEVELOPMENT OF A FACE RECOGNITION SYSTEM.*

[24] Luka Dulčić (2020) *Face Recognition with FaceNet and MTCNN.* [Online]. `https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/`

(visited on 17. 5. 2022)

[25] Schroff, Florian and Kalenichenko, Dmitry and Philbin, James(2015). *Facenet: A unified embedding for face recognition and clustering.*

[26] Microsoft Corporation (2022) *Face API.* [Online]. `https://azure.microsoft.com/services/cognitive-services/face`

(visited on 18. 5. 2022)

[27] Amazon.com Inc. (2022) *Amazon Rekognition.* [Online]. `https://aws.amazon.com/rekognition` (visited on 18. 5. 2022)

[28] Kairos AR. (2022) *Kairos.* [Online]. `https://www.kairos.com`

(visited on 18. 5. 2022)

[29] Jiankang Deng and Jia Guo (2018). *insightface.* [Online]. `https://insightface.ai` (visited on 18. 5. 2022)

[30] Sefik Ilkin Serengil, Github (2019). *deepface.* [Online]. `https://github.com/serengil/deepface` (visited on 18. 5. 2022)

[31] Jeff Brown (2010). *ZBar About.* [Online]. `http://zbar.sourceforge.net/about.html` (visited on 18. 5. 2022)

[32] Unknown author (2021). *What is Flask Python.* [Online]. `https://pythonbasics.org/what-is-flask-python` (visited on 18. 5. 2022)

[33] Google Brain Team (2022). *Tensorflow.* [Online]. `https://www.tensorflow.org` (visited on 18. 5. 2022)

[34] OpenCV team (2022). *OpenCV About.* [Online]. `https://opencv.org/about` (visited on 18. 5. 2022)

[35] Python Software Foundation (2022). *pickle — Python object serialization.* [Online]. `https://docs.python.org/3/library/pickle.html` (visited on 18. 5. 2022)

[36] Expo (2022). *Camera.* [Online]. `https://docs.expo.dev/versions/latest/sdk/camera` (visited on 18. 5. 2022)

[37] Igor Sysoev (2022). *Configuring HTTPS servers*. [Online]. http://nginx.org/en/docs/http/configuring_https_servers.html

(visited on 18. 5. 2022)

[38] apirobot (2021). *What is WSGI and Why Do You Need Gunicorn and Nginx in Django*. [Online]. https://apirobot.me/posts/what-is-wsgi-and-why-do-you-need-gunicorn-and-nginx-in-django

(visited on 18. 5. 2022)

[39] SANG-WOO HAN (1997). *THREE-TIER ARCHITECTURE FOR SENTINEL APPLICATIONS AND TOOLS: SEPARATING PRESENTATION FROM FUNCTIONALITY*.

[40] Oracle Corporation (2022). *What Is a Database?*. [Online]. https://www.oracle.com/cz/database/what-is-database

(visited on 18. 5. 2022)

[41] Amanda Athuraliya (2021). *The Easy Guide to UML Deployment Diagrams*. [Online]. https://creately.com/blog/diagrams/deployment-diagram-tutorial

[42] Pallets (2010). *Deploy to Production*. [Online]. https://flask.palletsprojects.com/en/2.1.x/tutorial/deploy

(visited on 18. 5. 2022)

[43] Jason Brownlee (2019). *How to Develop a Face Recognition System Using FaceNet in Keras*. [Online]. https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-using-facenet-in-keras-and-an-svm-classifier

(visited on 18. 5. 2022)

[44] Giuseppe Bonaccorso (2018). *Mastering Machine Learning Algorithms*. Packt Publishing

[45] DigitalOcean LLC. (2022). *DigitalOcean*. [Online]. https://www.digitalocean.com

[46] DigitalOcean LLC. (2022). *Droplets*. [Online]. https://www.digitalocean.com/products/droplets

[47] Bastien, J. (2009). *Usability testing: A review of some methodological and technical aspects of the method*. International journal of medical informatics

[48] Leah Buley (2013). *The User Experience Team of One*. Rosenfeld

[49] Expo (2022). *Barcode scanner*. [Online]. https://docs.expo.dev/versions/latest/sdk/bar-code-scanner (visited on 18. 5. 2022)

Appendix A

Development guide

Frontend github repo `https://github.com/alexnguyen98/fridge-face-frontend`

Backend github repo `https://github.com/alexnguyen98/fridge-face-backend`

A.1 Frontend

A.1.1 Requirements

- Node v15.2.0, < v17.0.1
- Yarn/npm
- Expo Go app

A.1.2 Changing backend host

In the file `fridge-face-frontend/src/constants/index.ts` set the constant `SERVER_URL` to your desired backend host (defaults to `https://nguyexu.tech`).

A.1.3 Usage

```
# Install dependencies
yarn install

# Running dev mode
```

```
yarn start
# Test the project on a emulator or
by scanning the QR code from the terminal

# Releasing on production
expo publish
}
```

A.2 Backend

A.2.1 Requirements

- Python 3.7.2
- pip3

A.2.2 Usage

```
# Activate python venv
python3 -m venv .venv
source .venv/bin/activate

# Install dependencies
pip3 install -r requirements.txt

# Running dev mode
python3 run.py

# Running prod mode
gunicorn run:app -b 0.0.0.0:3000
}
```

Appendix B

Screenshots from production application

The following appendix shows screenshots from the mobile application in production. The software was running on a tabled device (iPad 7th generation).

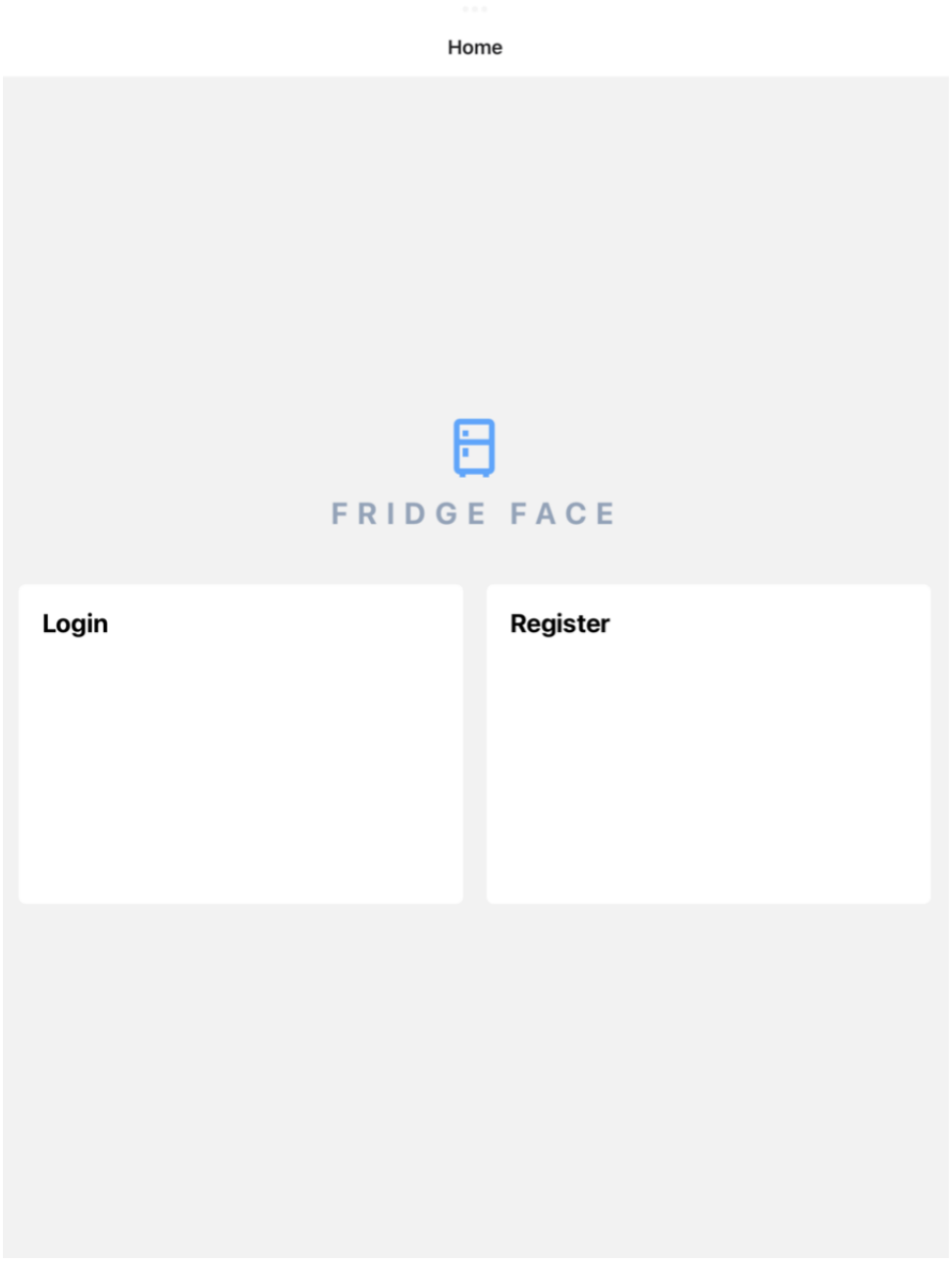


Figure B.1: Home screen

Figure B.2: Walkthrough screen

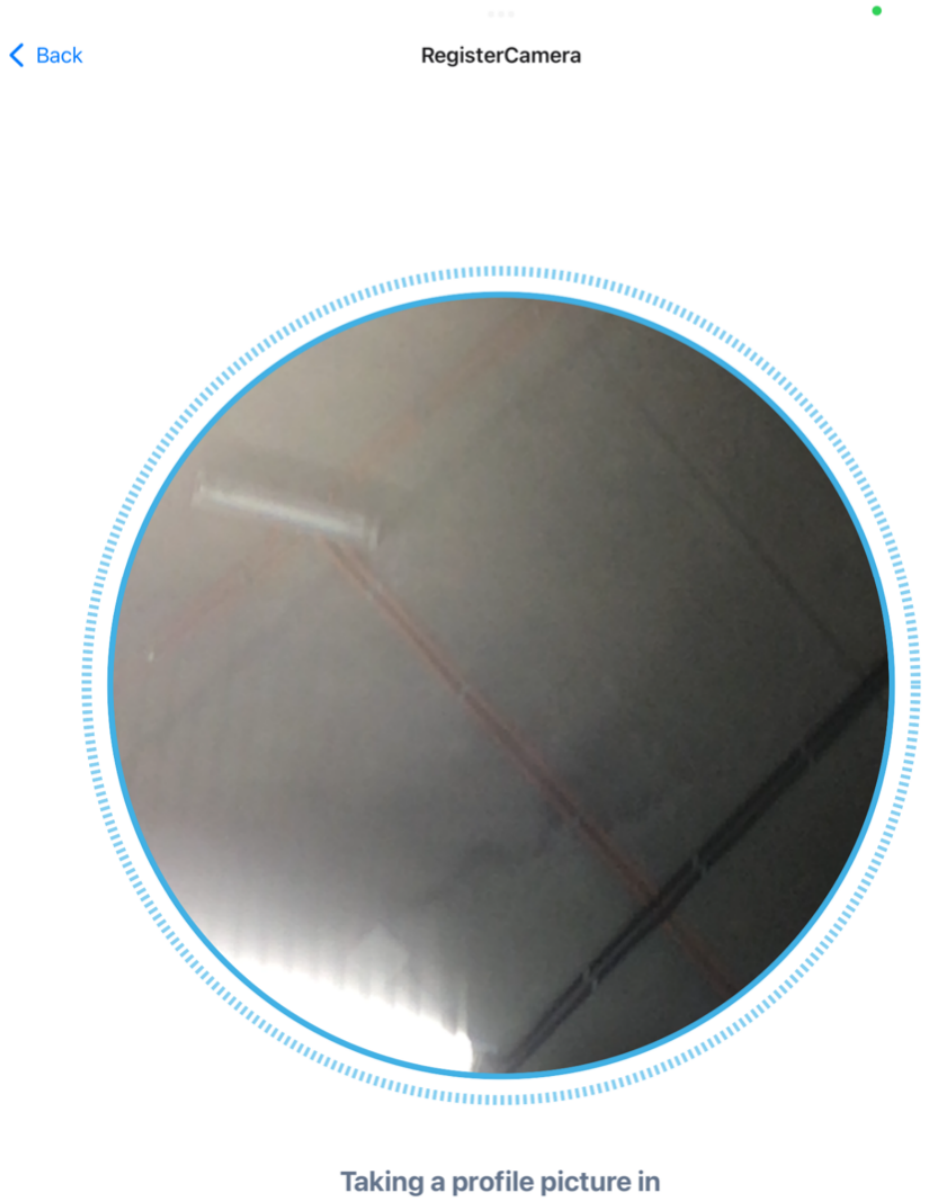


Figure B.3: Face enrollment screen

Figure B.4: Login screen

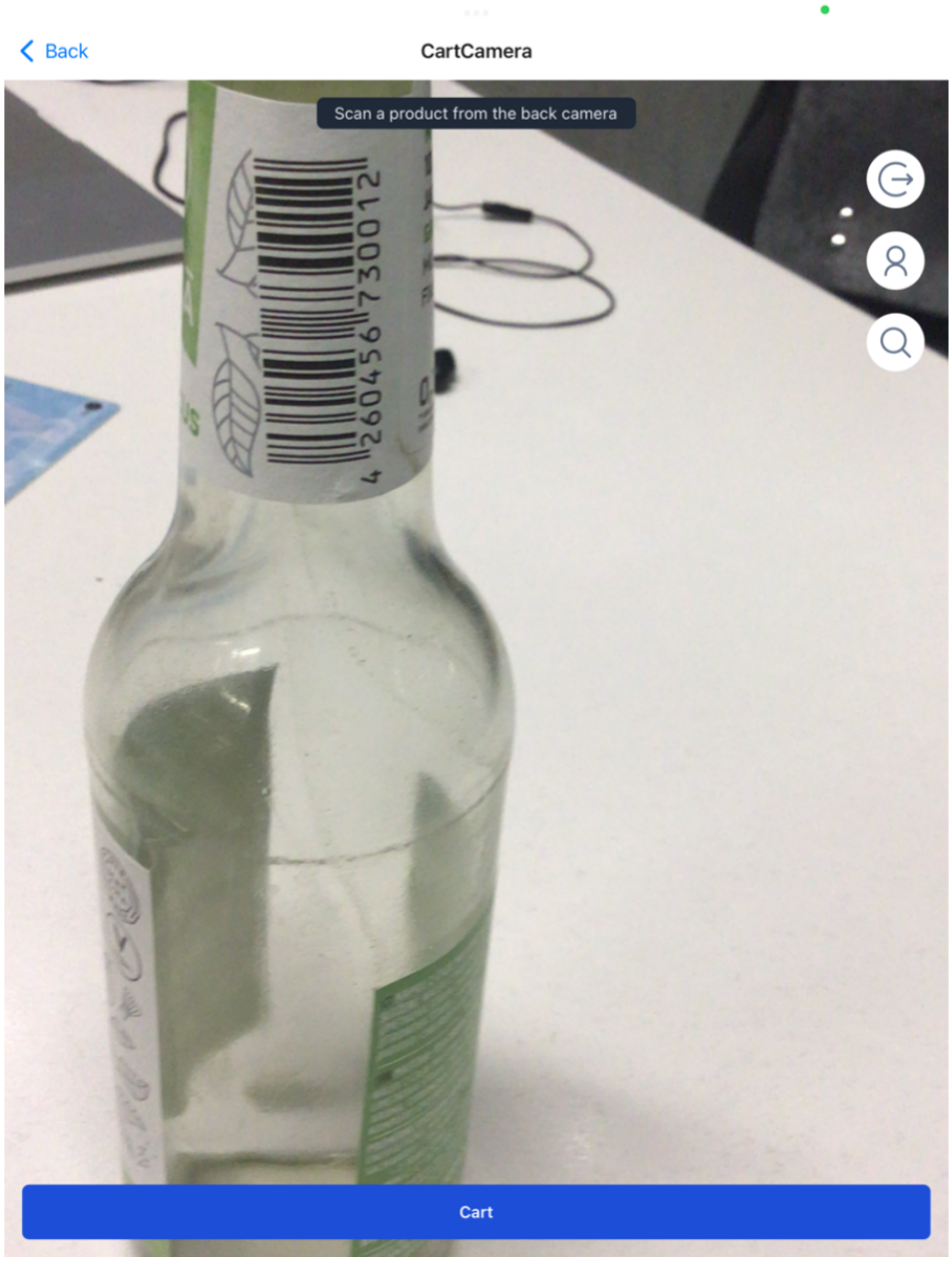


Figure B.5: Scan product barcode screen

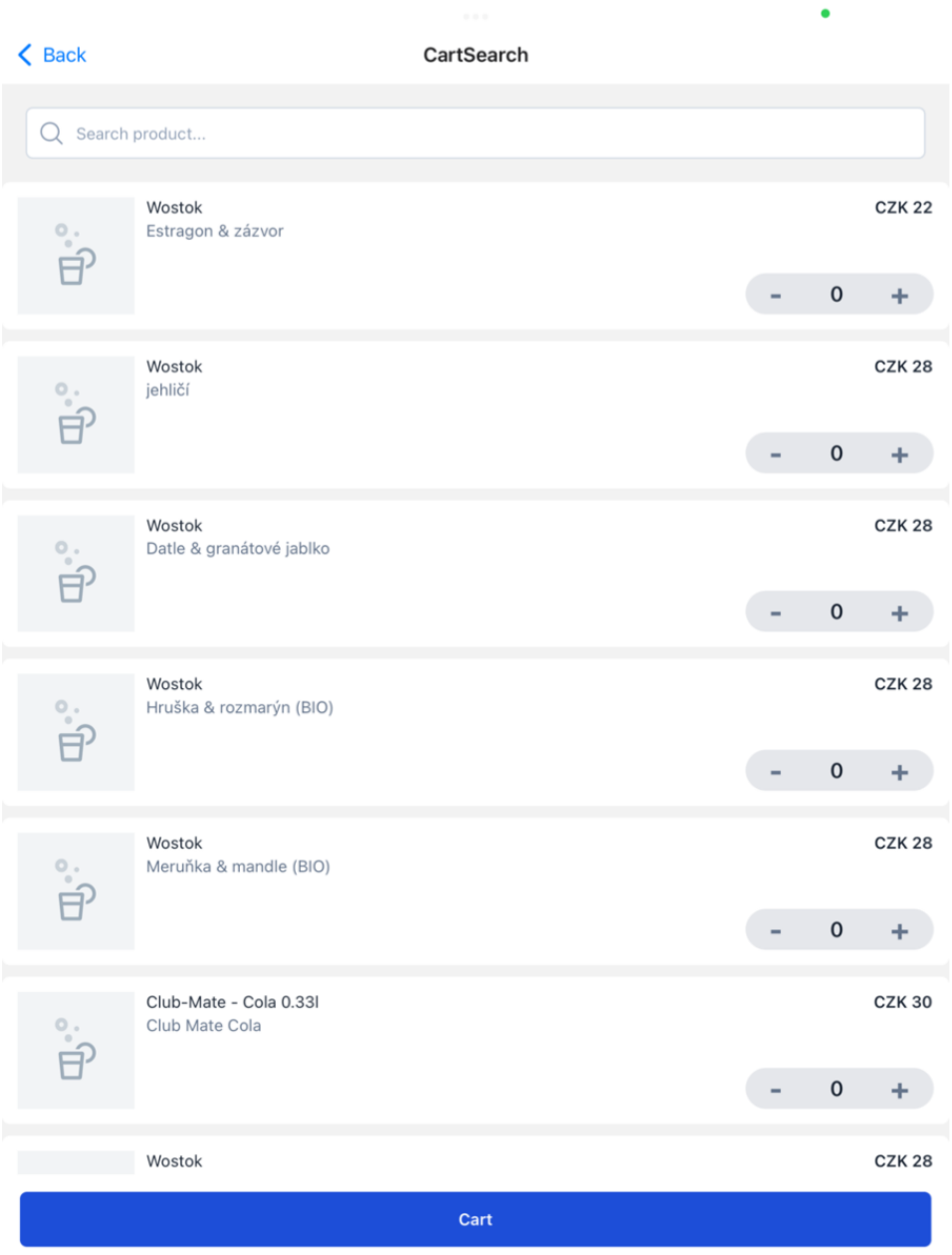


Figure B.6: Product search screen