Goodness Groceries - Symmetric key encryption and iOS security measures

Monday 7th June, 2021 - 08:14

Flavio De Jesus Matias
University of Luxembourg
Email: flavio.dejesus.001@student.uni.lu

Benoît Ries
University of Luxembourg
Email: benoit.ries@uni.lu

***Abstract*—This document presents the bachelor semester project of Flavio De Jesus Matias under the tutoring of Benoît Ries. The project consists of the final development of the eco-friendly mobile application *Goodness Groceries* by partnering up with *Pall Center*, a Luxembourgish supermarket, and the Sociology research department of the University of Luxembourg. The app will act as a shopping companion for the customers and provide them with indicators describing the ecologic background of the different products. The scientific part of the project focuses on the exploration and illustration of security measures for iOS and mobile applications in general. The technical section of the project will present the further development of the Goodness Groceries iOS app.**

1. Introduction

Mobile app security has become more and more popular over the years. It is of such great importance that we can't consider it as a feature or benefit anymore. Nowadays, it has become a necessity. Both companies and private people need a certain standard of security in the applications they use every day since a security breach can mean a loss of millions of dollars or the loss of countless sensitive information.

Companies have to make sure they offer the best security standards in their apps to protect their users and customers. Building customer trust takes a very long time, but one mistake can make it turn around very rapidly. Thus, they don't only have a brand and name to protect, but also their customer's information.

In this project, we will introduce several iOS security measures that developers should consider when creating an app. The adequate security measures will be discussed and later implemented in the Goodness Groceries app in order to provide a more secure utilization. Encryption also plays a big role in general security today [3] and the encryption standard used in Apple's operating systems will be later presented in this document.

2. Project description

This section introduces the domain and targeted deliverables of the project.

2.1. Domains

2.1.1. Scientific. The scientific domain of this project is iOS security measures, in particular symmetric key encryption algorithms. iOS security includes a wide range of possibilities and practices, but the report will focus more on the symmetric key encryption algorithm AES used in Apple's Keychain to encrypt user-sensitive data.

2.1.2. Technical. The technical domain of this project is the development of applications for the operating system iOS. Mobile development for iOS requires the latest version of Xcode, which is Apple's Integrated Development Environment (IDE). While Xcode provides the GUI to develop apps, the programming language Swift is also part of the domain alongside its framework SwiftUI.

Swift
Swift is a general-purpose programming language developed by Apple in 2014. The purpose of this programming language was to replace the outdated Objective-C programming language and to create very performant applications for macOS, iOS and other operating systems that belong to the Apple ecosystem.

SwiftUI
SwiftUI is a framework providing innovative and simple [4] ways to build user interfaces across all Apple platforms. It perfectly extends Swift with a set of tools that allows developers to create user interfaces easier and in a more natural way because it reduces the complexity relied upon designing GUIs by providing a simple declarative syntax and leaves developers with more time to focus on the business logic of their application. The SwiftUI syntax is simple to read and written in a natural way, which can help developers to better understand the code and develop graphical interfaces faster.

2.2. Targeted Deliverables

2.2.1. Scientific. The targeted scientific deliverable of this project is the exploration and illustration of security measures for iOS and mobile applications in general. The deliverable will present several security practices used in iOS

mobile apps and identify which can be implemented in the technical deliverable. Furthermore, it will describe and illustrate the chosen security approaches in detail and explain how they are integrated into the technical deliverable. At the end of the deliverable, possible limitations and drawbacks of the chosen security measures will be shown.

2.2.2. Technical. The targeted technical deliverable of this project is the further development of the iOS application *Goodness Groceries* following the given requirements. The deliverable will be a near-final version of the app while continuing to follow the given design and template choices. The developed app should contain all the necessary functionalities such as answering product reviews or comparing products.

3. Prerequisites

3.1. Scientific

The scientific prerequisite of this project is to have the necessary high school algebra knowledge to fully understand the mathematical expressions of the encryption algorithm presented in this document. Although the equations and the algorithm will be explained, the reader might get overwhelmed without the necessary prior knowledge. Finally, no knowledge in cryptography is needed as the algorithms and concepts will be described in detail in this document.

3.2. Technical

The technical prerequisite of this project is the knowledge of the programming language Swift alongside some basics of the framework SwiftUI. Although the report introduces some SwiftUI elements, without some background knowledge the reader might get lost. To better understand the project it is therefore recommended to already have some knowledge in SwiftUI.

4. Symmetric key encryption and iOS security measures

4.1. Requirements

The main requirement of this deliverable is to present the symmetric key encryption algorithm AES [10]. Since this is included in one iOS security measure, which is the Keychain, other iOS security measures should also be presented. Thus, we can agree on two different functional requirements for the scientific deliverable.

- FR1: The deliverable shall present the symmetric key encryption algorithm AES.
- FR2: The deliverable shall briefly present other popular iOS security measures used by iOS mobile developers.

4.2. Design

In this deliverable, we will discuss symmetric key encryption and hashing algorithms, especially AES. The entire section 4.3 is divided into different parts of the production of the scientific deliverable.

In the first part, symmetric key encryption and hashing algorithms are presented and explained. The focus here lies on giving some context about how these work and their purpose.

In the second part, we will discuss the AES encryption algorithm in detail. Here all the steps of the algorithm will be explained in order to provide an understandable view of how the algorithm works. Then, the GCM [6] part and its core function, the GHASH function, is also explained and how it connects together with the AES algorithm.

Finally, in the last part, we will present the link between the AES algorithm and iOS security through Apple's Keychain and how it is used. Then, in the end, other iOS security measures used by developers are briefly presented.

4.3. Production

4.3.1. Symmetric key encryption. Symmetric key encryption is a cryptographic algorithm that encrypts and decrypts data using a single shared key. It is faster compared to asymmetric key encryption and therefore preferred for large amounts of data. Symmetric key encryption itself is a very basic concept which can be seen in Figure 1. There are several widely used algorithms that achieve a high standard of encryption using a symmetric key, such as AES [10], DES [11], and 3DES (Triple DES; uses DES three times to encrypt data). In this report, we will focus on the example of AES.

Figure 1: Symmetric key encryption (from [8])

4.3.2. Hashing algorithms. A hashing algorithm is a mathematical function that converts any input information into a fixed-length string of characters and numbers. Unlike encryption algorithms, hashing algorithms cannot be reversed (ideally) and only take as input some data and output a hash string. Good hashing algorithms share a simple common characteristic: if we hash some data, then change a few bits in the data and hash it again, the 2 hashing outputs should be completely different and there should be no correlation between the 2. There are plenty of hashing algorithms, but some of the most common ones include MD4 [12], MD5 [13], SHA-1 [14], and SHA-2 [15].

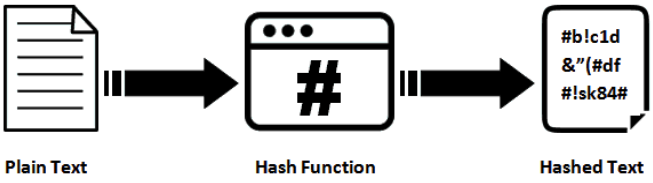


Figure 2: Hashing algorithm (from [9])

4.3.3. AES. AES stands for "Advanced Encryption Standard" and is a symmetric key encryption algorithm that only generates a private key, thus only the key holder can decrypt the encrypted data. A good example where AES is used is to encrypt a personal computer's content when not in use. The computer content is encrypted using the user's password, which can be an arbitrary string of characters because it is processed by a hashing algorithm that outputs the password as the required bit size of the key.

AES, originally *Rijndael*, was invented by the 2 Belgian cryptographers Vincent Rijmen and Joan Daemen in 1998. The algorithm was chosen by the **U.S. NIST** (National Institute of Standards and Technology) in 2001 to replace the old DES algorithm and was adopted by the **U.S. government** to encrypt **top-secret** information. Due to its popularity, the algorithm was also adopted in the private sector.

The bit size of the key is related to the number of rounds Nr that the algorithm will perform. There are 3 possibilities for the key size: 128, 192, and 256 bits, which can be seen in Table 1.

Key size in bits	Number of rounds (Nr)
128	10
192	12
256	14

Table 1: Number of rounds corresponding to the key size

Choosing a larger key in AES can enhance security by adding a few extra rounds of encryption and enlarging the key space size. If we compare the possible key combinations for AES with a 128-bit key with, for instance, the key combinations for a 256-bit key, there is a huge difference:

- 2^{128} combinations for a 128-bit key: **340,282,366,920,938,463,463,374,607,431,768,211,456**
- 2^{256} combinations for a 256-bit key: **115,792,089,237,316,195,423,570,985,008,687,907,853, 269,984,665,640,564,039,457,584,007,913,129,639,936**

Thus, there are more possibilities for the 256-bit key, which makes it harder to break. However, choosing a higher key size requires more performance to execute the algorithm. According to NSA [23], all the bit key combinations are enough to protect information up to the SECRET level. To protect TOP SECRET information, a minimum key size of 192 or 256 bits is required. Thus, it doesn't have a big impact on which key size is chosen for the average user.

As shown in Figure 3, the algorithm is broken down into a series of functions. These functions are executed Nr-1 amount of times and then a slightly modified version for the Nr-th round, which is also called the *final round*. We can therefore divide the algorithm into the following steps/functions:

1) Initial round
   - AddRoundKey
2) Main rounds (9, 11 or 13 rounds)
   - SubBytes
   - ShiftRows
   - MixColumns
   - AddRoundKey
3) Final round
   - SubBytes
   - ShiftRows
   - AddRoundKey

To better explain and illustrate the algorithm, each step and/or function is divided into its own section. The examples shown use a 256-bit key.

Before the algorithm rounds start

Before the algorithm rounds start, the user has to define a password. To goal here is to take as input the password given by the user, which can be arbitrarily long, and hash it to the needs of the algorithm.

To illustrate this, let's say we take the following password for our algorithm:

- Password: **password12345678**

Using a hashing algorithm, for instance, SHA-256, we can hash this password into a short key. A short key is used to derive all the necessary round keys and is 64 characters long, only in hexadecimal values. Since a single hexadecimal value has 4 bits, we end up with a short key of 256 bits for the AES-256 algorithm. The short key in this example is the following:

- Short key: **aafeeba6959ebeeb96519d5dcf0bcc069f81e 4bb56c246d04872db92666e6d4b**

Now that we have the short key, we can separate it into a series of round keys. Round keys are generated for each round of the algorithm and used in the AddRoundKey step. Round keys are different for each round and are generated using the Rijndael key schedule [16]. Furthermore, they are the same size as a block of the algorithm, thus they fit in a 4x4 matrix. Each block in the AES algorithm is 128 bits long, independently of the key size. The first round key is the following:

- First round key: **a567fb105ffd90cb**

AddRoundKey
The AddRoundKey function performs an exclusive or (XOR) on the current matrix with the round key obtained from the key expansion. To illustrate how this function works, we will use the very first round key. This step works the same for all the upcoming rounds, by simply taken the i-th round key, as can be seen in figure 3.

Using the first round key, we can XOR the current matrix with it. Since in our example it is the very first round, the current matrix to be XOR-ed is still the plaintext message. Let's define the plaintext message to be encrypted as: *Here is a secret* and let's convert it to a matrix of hexadecimal values. This is done by converting the ASCII text *Here is a secret* to hexadecimal, which gives the following value: **48 65 72 65 20 69 73 20 61 20 73 65 63 72 65 74**. Each 2 digit number represents a character in the string. This hexadecimal value is then inserted into a matrix from top to bottom, left to right. This gives the following result:

$$\begin{bmatrix} 48 & 20 & 61 & 63 \\ 65 & 69 & 20 & 72 \\ 72 & 73 & 73 & 65 \\ 65 & 20 & 65 & 74 \end{bmatrix}$$

And now the same thing for the current round key, which is the first round key for this example: a567fb105ffd90cb

$$\begin{bmatrix} 61 & 66 & 35 & 39 \\ 35 & 62 & 66 & 30 \\ 36 & 31 & 66 & 63 \\ 37 & 30 & 64 & 62 \end{bmatrix}$$

Once both are in hexadecimal values, we just need to XOR it together. To do so, we take each entry of the current matrix (hex value of plaintext for the very first round) and XOR it with the corresponding entry of the round key (first round key for this example). Thus we obtain the following:

- $61 \oplus 48 = 29$
- $35 \oplus 65 = 50$
- $36 \oplus 72 = 44$
- ...

After doing this for all the 16 entries of the matrix, we obtain the following result:

$$\begin{bmatrix} 29 & 46 & 54 & 5a \\ 50 & 0b & 46 & 42 \\ 44 & 42 & 15 & 06 \\ 52 & 10 & 01 & 16 \end{bmatrix}$$

SubBytes
The SubBytes function is a byte substitution process where each element of the output matrix is replaced by the corresponding entry in the given 8-bit substitution box (S-box), see Figure 4. To read the S-box table, each byte input of the matrix is split into 4 bits. The 2 first bits determine the row, and the 2 last bits determine the column from where to read the new value. Thus, for instance, the value 29 should be replaced by the value located in row 2 and column 9 of the S-box. The new value is therefore a5. This process is done for each entry of the matrix. In our example, by applying the SubBytes function we obtain the following matrix:

$$\begin{bmatrix} 29 & 46 & 54 & 5a \\ 50 & 0b & 46 & 42 \\ 44 & 42 & 15 & 06 \\ 52 & 10 & 01 & 16 \end{bmatrix} \longrightarrow \begin{bmatrix} a5 & 5a & 20 & be \\ 53 & 2b & 5a & 2c \\ 1b & 2c & 59 & 6f \\ 00 & 7c & 7c & 47 \end{bmatrix}$$

ShiftRows
The ShiftRows function shifts the rows in the matrix. The first row does not change, but all the others do. The second row shifts by 1 column to the left, the third by 2, and the fourth by 3. To illustrate this in our current example, we obtain the following matrix:

$$\begin{bmatrix} a5 & 5a & 20 & be \\ 53 & 2b & 5a & 2c \\ 1b & 2c & 59 & 6f \\ 00 & 7c & 7c & 47 \end{bmatrix} \longrightarrow \begin{bmatrix} a5 & 5a & 20 & be \\ 2b & 5a & 2c & 53 \\ 59 & 6f & 1b & 2c \\ 47 & 00 & 7c & 7c \end{bmatrix}$$

MixColumns
The MixColumns function adds diffusion to the cipher together with SubBytes and ShiftRows, by mixing the input around. Diffusion is the idea of hiding the relationship between plain text and ciphertext. By changing a single bit in the ciphertext, the majority of the plain text should also change.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} a5 & 5a & 20 & be \\ 2b & 5a & 2c & 53 \\ 59 & 6f & 1b & 2c \\ 47 & 00 & 7c & 7c \end{bmatrix} =$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

To illustrate how this process works, we will calculate the value of $a_{0,0}$ step by step.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} a5 \\ 2b \\ 59 \\ 47 \end{bmatrix} = \begin{bmatrix} a_{0,0} \\ a_{1,0} \\ a_{2,0} \\ a_{3,0} \end{bmatrix}$$

The MixColumns is a particular kind of multiplication as we don't add up each part of the matrix multiplication, but we exclusive or them together. Let's illustrate this special multiplication for the example of $a_{0,0}$. We obtain the following equation:

$$(2 \times a5) \oplus (3 \times 2b) \oplus (1 \times 59) \oplus (1 \times 47) = a_{0,0} \quad (1)$$

The numbers 1, 2 and 3 each represent the respective polynomials:

- $1 = 00000001_2 = 1$
- $2 = 00000010_2 = x$
- $3 = 00000011_2 = x + 1$

The same applies for all the hexadecimal values from the matrix:

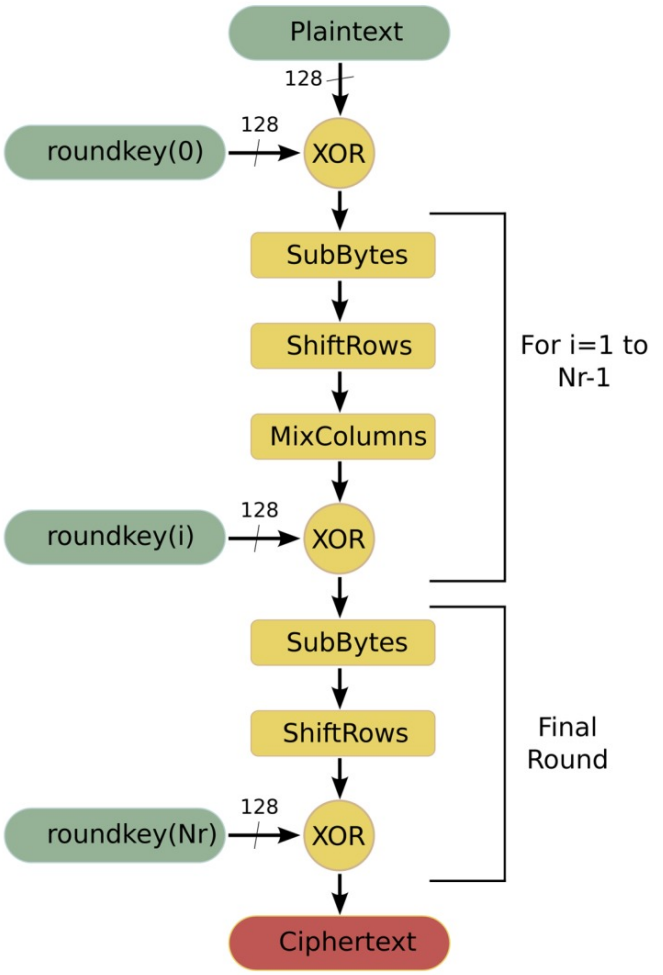


Figure 3: Flowchart of the AES algorithm (from [5]) where the roundkey function corresponds to the AddRoundKey function explained in this report

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 4: S-box (substitution box) for AES

- $a5 = 10100101_2 = x^7 + x^5 + x^2 + 1$
- $2b = 00101011_2 = x^5 + x^3 + x + 1$
- $59 = 01011001_2 = x^6 + x^4 + x^3 + 1$
- $47 = 01000111_2 = x^6 + x^2 + x + 1$

The next step is doing the multiplication for all the 4 parts of the equation 1. Once we have the new polynomial, we can convert it back to binary and do modulo the polynomial $x^8 + x^4 + x^3 + x + 1$, which is 100011011_2 in binary.

- $(2 \times a5) = x \times (x^7 + x^5 + x^2 + 1)$
  $= x^8 + x^6 + x^3 + x$
  $= 101001010_2 \mod 100011011_2$
  $= 01010001_2$
- $(3 \times 2b) = (x + 1) \times (x^5 + x^3 + x + 1)$
  $= x^6 + x^5 + x^4 + x^3 + x^2 + 1$
  $= 001111101_2 \mod 100011011_2$
  $= 01111101_2$
- $(1 \times 59) = x^6 + x^4 + x^3 + 1$
  $= 001011001_2 \mod 100011011_2$
  $= 01011001_2$
- $(1 \times 47) = x^6 + x^2 + x + 1$
  $= 001000111_2 \mod 100011011_2$
  $= 01000111_2$

The second-to-last step before arriving at the value for $a_{0,0}$ is XOR-ing the binary results and converting it back to hexadecimal.

$$\begin{array}{cc} & 01010001_2 \\ \oplus & 01111101_2 \\ \oplus & 01011001_2 \\ \oplus & 01000111_2 \\ \hline & 00110010_2 \end{array}$$

The final step is to convert the binary result 00110010_2 back to hexadecimal 32_{16}. Thus, we end up with the following matrix:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} a5 \\ 2b \\ 59 \\ 47 \end{bmatrix} = \begin{bmatrix} 32 \\ a_{1,0} \\ a_{2,0} \\ a_{3,0} \end{bmatrix}$$

The remaining 15 entries, $a_{0,1}$ to $a_{3,3}$, are calculated using the same operations.

4.3.4. GCM. GCM stands for "Galois/Counter Mode" and is an authenticated encryption algorithm created to ensure the integrity and authenticity of additional authenticated data alongside encryption of plaintext. The additional authenticated data is an optional parameter, used in, for instance, network communications to protect the header information. Integrity refers to the accuracy, completeness, and reliability of data, it ensures that data is not corrupted. The authenticity of data ensures that the data stays in its original form and is not modified by a third party, it ensures that the receiver gets **exactly** what was sent, and not a modified version.

GCM was invented by John Viega and David A. McGrew as an improved version of the *Carter–Wegman* counter mode. It provides authenticated encryption and in 2007 it was adapted by the NIST.

GCM has **4** inputs [6]:

1) **AES key** K: 128, 192 or 256 bits according to the chosen AES algorithm
2) **Initialization vector (IV, 128-bits)**: since using the current block number as an initialization vector does not provide enough randomness, it is concatenation between the number of the current block (32-bits) and a number used once (nonce, 96-bits) which is generated randomly and it must be unique.
3) **Plaintext content**
4) **Optional additional authenticated data**: the authenticated data is not encrypted and therefore not included as an output parameter. It is used to ensure the integrity and authenticity of data which has to be protected but not encrypted.

and **2** outputs:

1) **Ciphertext**
2) **Authentication tag**: integrity check value (ICV) used to confirm the authenticity of the data while decrypting

Figure 5 shows an overview of the GCM algorithm. It can be broken down into various steps:

1) First, the additional authenticated data is separated into blocks of 128-bits and every block is hashed using the GHASH function.
2) Then, once all the additional data blocks are hashed, all the plaintext blocks are encrypted through the AES algorithm. To do so, we pass in the key *K* for the AES algorithm, as well as the initialization vector.
3) Furthermore, we exclusive or the result with the plaintext yielding the ciphertext, which is then again XORed with the output of the GHASH function of the previous step.
4) The result is then passed to the GHASH function again and used on the next step.
5) Once all the plaintext blocks got encrypted, there are 2 more steps until the algorithm is done: perform an exclusive or operation on the result with the bit string concatenation of the length of the additional authenticated data and the ciphertext, and then finally XOR that result with the very first 0 block, which is just a 128 zero bits string encrypted using AES.
6) This result yields the authentication tag, and thus we can output the ciphertext and the authentication tag produced by the algorithm.

This process is illustrated in figure 5 and the GHASH function is explained here below.

GHASH function

The GHASH function is defined as:

$$GHASH(H, A, C) = X_{m+n+1} \tag{2}$$

where:

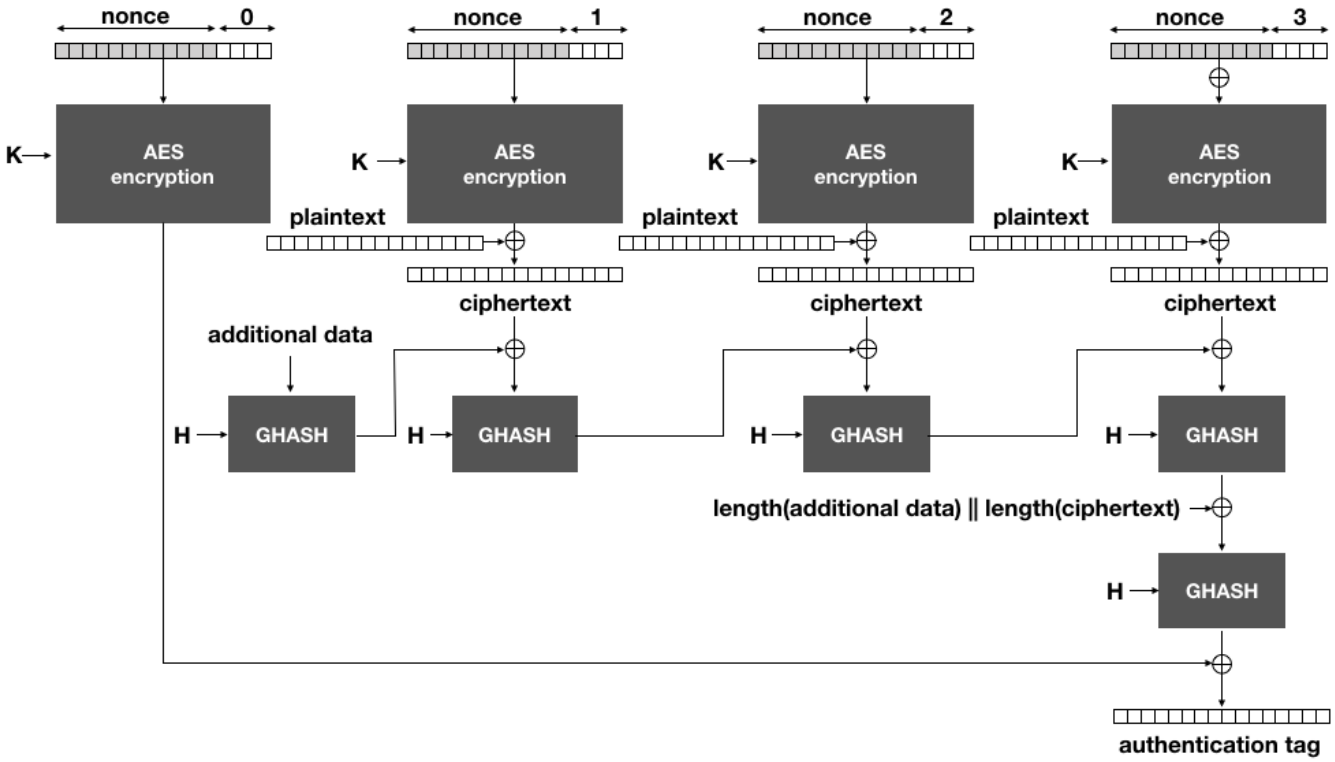


Figure 5: Galois/Counter Mode with AES encryption and 3 plaintext blocks

$H = E_k(0^{128})$ hash key, string of 128 zero bits encrypted
A = additional data (authenticated data, not encrypted)
C = ciphertext output from the AES algorithm
m = number of 128-bit blocks in A
n = number of 128-bit blocks in C

This is the output given by the GHASH function at each step. The output is defined as:

$$X_i = \sum_{j=1}^{i} S_j \times H^{i-j+1} = \begin{cases} 0 & \text{for i = 0} \\ (\mathrm{X}_{i-1} \oplus S_i) \times H & \text{otherwise} \end{cases} \tag{3}$$

In the equation 3 we can see the output of the GHASH function defined for each i up to $m+n+1$. What this function does is XOR-ing the previous block with the current message and multiplying the result with the hash key H. The current message is defined as S:

$$S_i = \begin{cases} A_i & \text{for } i = 1, ..., m-1 \\ A_m^* \parallel 0^{128-v} & \text{for } i = m \\ C_{i-m} & \text{for } i = m+1, ..., m+n-1 \\ C_n^* \parallel 0^{128-u} & \text{for } i = m+n \\ len(A) \parallel len(C) & \text{for } i = m+n+1 \end{cases} \tag{4}$$

where:

v = len(A) $\mod 128$, gives the number of bits in the final block of A, thus $128 - v$ equals the number of 0's to be added to the last block
u = len(C) $\mod 128$, gives the number of bits in the final block of C, thus $128 - u$ equals the number of 0's to be added to the last block

The equation 4 can be split up into different parts:

- Since m represents the number of blocks in the authenticated data, the last block m may not have 128 bits. Thus, for the first $m-1$ blocks of length 128 bits, we simply take the i-th block of the additional data to XOR with X_{i-1}.
- For the last block of the additional data, which is $i = m$, we need to concatenate the missing 0's in order to get a 128 bit block. Then, we proceed the same way.
- The first ciphertext output block from the AES algorithm starts at position $m+1$ and has n blocks. Thus, it goes from the range $i = m+1$ until $i = m+n-1$.
- The last ciphertext block at position $i = m+n$ first needs to be extended with 0's in order to get a 128 bit block.
- The final output of S is the concatenation of the bit strings of len(A) and len(C).

4.3.5. AES-256-GCM in Apple Keychain. Apple's Keychain is implemented as an SQLite database stored locally. As there is only one database, it provides an API that authenticates the request and acts accordingly. The data in the Keychain are stored as key-value pairs and each one of

them has a protection class indicating how the data can be accessed. The different classes are:

- Always
- Passcode enabled
- After first unlock
- While unlocked
- While locked

The link between the AES algorithm, GCM, and iOS security is that the Keychain stores data using the AES-256-GCM algorithm. Each row is encrypted using the AES algorithm with a 256-bit key and authenticated using GCM, thus providing the best possible encryption to user data and passwords.

4.3.6. Other iOS security measures. Finally, there are other iOS security practices used by developers to ensure the best security standards and provide the user a stress-free experience. Some of them are listed below in no particular order.

- **Server connection**: using secure network connections to APIs over the HTTPS [25] protocol
- **SSL-Pinning**: due to possible man-in-the-middle attacks [17] using the HTTPS protocol, one can use certificate validation in order to ensure that data received and sent to are from the correct person
- **Application authorization**: using a PIN with server-side validation and/or biometric authentication such as fingerprint or facial recognition can secure the app and only allow authorized people to access it
- **Anti-fraud system**: when unauthorized access is detected, one can blacklist the device and avoid it from communicating with the server, thus the device is blocked and cannot perform any actions. Another way is by alerting the user when actions from an atypical location are performed and using SMS codes before sending requests to the server to ensure that it is the correct person
- **Data entry**: disable auto-complete and copy-paste and mask the content in password fields and only enter PIN codes through number buttons on the screen instead of the keyboard to bypass potential keyloggers
- **Jailbreak check**: jailbreaks [18] can give other apps unauthorized access to certain features of the device, thus for instance banking apps might want to check whether the device is jailbroken and not allow the user to use the app in that case

4.4. Assessment

Each requirement was successfully achieved and presented in the deliverable:

- FR1: The entire AES encryption algorithm was presented in this deliverable including illustrations and examples. Thus, the requirement is met.
- FR2: The second requirement was also achieved, as the deliverable presents a list with various iOS security measures used by iOS mobile developers. Each security measure is briefly explained, giving an idea of why one should consider these practices while developing a mobile application.

Overall, the end result of the scientific deliverable can be considered very good, as several concepts such as the difference between symmetric and asymmetric encryption algorithms, hashing algorithms, the entire AES algorithm, the GCM extension of AES as well as other iOS security measures were explained. Furthermore, all the requirements set were successfully accomplished.

5. Final version of the eco-friendly mobile application for iOS

5.1. Requirements

For the final version of the eco-friendly mobile application, several functional requirements need to be met for the application to be successful. These requirements summarize the technical requirements that the final app has to fulfill to be fully functional and correspond to the needs of Pall Center and the sociology team. Without these features, the app will not function properly and not achieve its end goal.

- FR1: The first requirement is that the app should be capable of fetching the products bought by the user from the web server and present them in a survey view. The user should then be able to give feedback about each product and explain the reason for their purchase. Finally, the app should post the results back to the server and handle potential internet issues without data loss.
- FR2: The app shall provide the customers with a quick way to compare products by allowing them to simply click on the compare button in a product's view. By doing so, the app should present a new view containing the products to compare in a grid view where each attribute is listed and comparable to the user.
- FR3: The third requirement is that the app should be capable of identifying the user's device language and adapt accordingly. Since the app will be available in French and English, the app should recognize french users and automatically select the correct language when the app is first opened. In all other cases, the app should be presented in the English language. Additionally, the user should be given the possibility to change the language as soon as the authentication is successful and normal app usage is possible. This feature should be available on the profile page of the user.

5.2. Design

The first version of the app design was prototyped before the beginning of the development by a different student. Although the graphical design gave us a clear idea of how the app should function and look, the graphical prototype was not entirely compliant with Apple's Human Interface Guidelines for iOS. Due to this and the appliance of different User Experience techniques, the design of several views was redesigned, even views already created in the prior semester. Since at the end of the semester the app should be in a final version ready to be submitted for review in the app store, various graphical parts of the app were slightly or entirely redesigned. The development of the missing views and features was also imported to guarantee a fully functional app at the end of the semester. We divide this section into **3** subsections: **code architecture**, **graphical redesign of previous views**, and **graphical design of new views**.

5.2.1. Code architecture. The code architecture of the app was presented in the last report [20] in detail. Since the app remains the same and only functionalities and views were added, the code architecture also remains the same. Thus, the app is still based on an MVVM [21] code architecture.

5.2.2. Graphical redesign of previous views. The graphical redesign of the previous views can by itself be split into different categories of views.

Welcome pages
The most noticeable change in the welcome pages is the addition of the *Goodness Groceries* logo and the application of the final color palette [figure 6] as shown in figure 7. The indicator category and product category selection pages were entirely modified. They both apply the same graphical layout, where each category is in its own box including a *select*-button. The motivation behind the change is that it brings a better user experience to the user by making the layout intuitive and simple. For the indicator categories to be more explicit, a popup with its definition is shown when clicking on one of them. This behavior can be seen in Figure 7 (c) and (d).

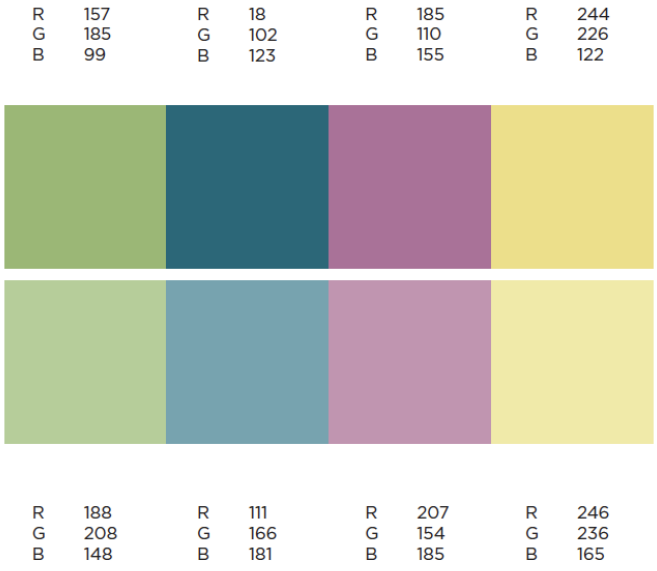


Figure 6: Goodness Groceries final color palette

Categories and products
The first page of this section is the categories view, which is also the home page. On the top left the new Goodness Groceries logo was added as well as a new search bar (see Figure 9 (a)). Furthermore, each category and indicator got a new icon as well as a more meaningful and better description. Before navigating to the products list, there is a new page that lists and organizes all the products into the different product categories (local organic, local conventional, imported organic, and imported conventional)

which can be seen in Figure 9 (b). The product page got small modifications such as adding similar products and the popups with definitions when clicking on each indicator. This can be seen in Figure 9 (c), (d) and (e).

Scanning tool
The scanning tool also got two minor changes.

- One of them is that there is now a popup which opens the first the user wants to use the scanning tool asking for permissions to use the camera. Since the native alert of iOS to ask permissions is extremely limited, it is hard to explain to the user why a certain permission is necessary. Thus using this popup, it is possible to explain to the user why the camera permission is required before prompting them with the native authorization alert. This can be seen in Figure 8 (a).
- The second change is the addition of the torch light button. This allows the user to toggle the torch light of their smartphone when scanning for QR-codes if necessary, directly in the app itself. This can be seen in Figure 8 (b).

5.2.3. Graphical design of new views. The graphical design of the new views can by itself be split into different categories of views and/or functionalities.

Survey page
The survey page was designed such that the user can review all the different products that they have bought. The list of products is fetched by the app regularly using Swift networking libraries which call the web API. For each product in this list, the user should be presented with a view that contains a list of all the indicators for this product. The user can then proceed to select the desired indicators by clicking on checkboxes. Additionally to all the indicators, the user can also select a different checkbox for *price*, as well as indicating a totally different reason for their purchase using a textbox. Whenever the user finishes with a product and clicks on continue, the app should then once again with the help of networking libraries call the web API and post the data to the database. Then, the same view for the next product should appear, or if it was the last product in the list, present a simple popup thanking the user.

Push Notifications
The push notifications do not necessarily represent a view but are rather an important background task of the app. Using the **Apple Push Notification service** (APNs) [19], the app should receive remote notifications from the webserver in the background if the user allowed notifications in the welcome pages. If it is a notification to review products, it should contain a list of products that is decoded by the app and passed to the user. The user then receives a notification and by clicking on it, the survey view page should be opened to start the reviewing process.

Compare page
The compare page is based on a very simple layout style. As each product has 3 similar products of the same type, one can create 4 columns for each of the products. Then, by adding a new column for all the indicators we end up with 5 columns and n+1 rows, where n represents the number of all the indicators. Using this scrollable layout, the user can then see whether a certain product has a specific indicator or not and also compare it to the other products.

Profile page
The profile page is very basic as not much information about the user is provided and/or needed in the app. The only thing known about the user is the client ID of their Pall Center card. This information is shown at the top of the screen. Then, the lower part of the screen contains the settings and products to review. Whenever there are products to review, the user can thus click on the button to start reviewing the products manually. The user can also change language and access the device notifications settings for the app.

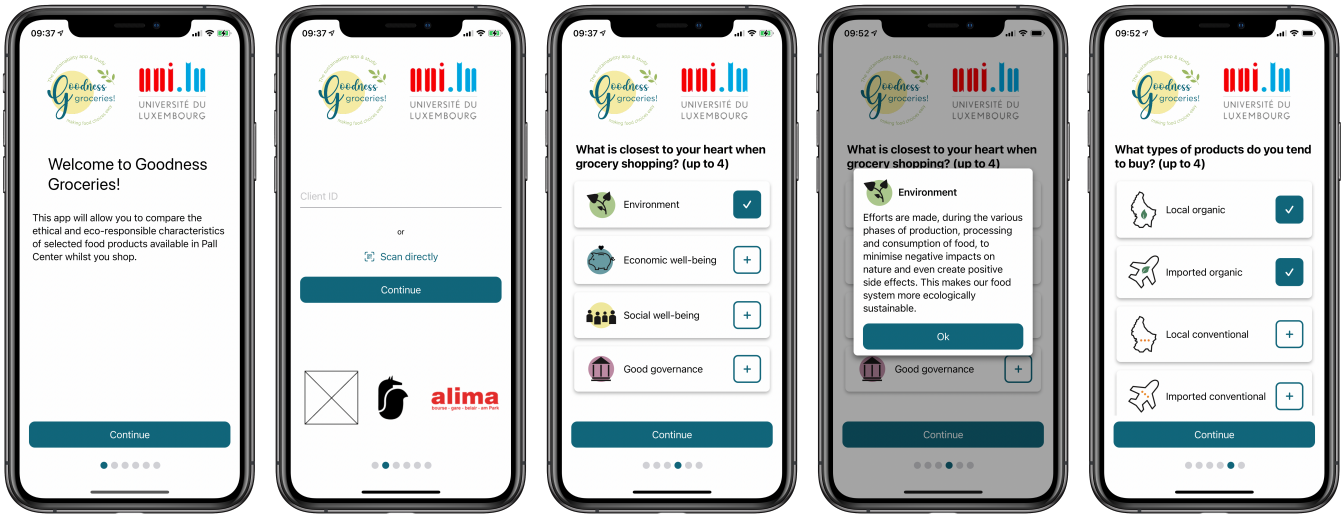
Help page
The help page is also very basic. Here the user can consult all the indicator definitions of the app, as well as access the sociology team website of the university to obtain further information about the study and the app itself.

5.3. Production

As the code structure was already presented in the design section, we may focus on the 2 other sections: **graphical redesign of previous views** and **graphical design of new views**. However, the important new parts of the backend will also be briefly mentioned in between.

5.3.1. Graphical redesign of previous views. The graphical redesign of the previous views can by itself be split into different categories of views.

Welcome pages
The addition of the logo in the welcome pages was done by implementing both images into a HStack, shown in Figure 7, which displays elements in a horizontal view. Then, the colors of the color palette were all added to the **Assets.xcassets** folder of the project by using the respective RGB values. This folder contains all the images and custom colors of the app. By doing so, the colors can be used through the entire app without having to reuse the RGB color codes. The indicator category and product category selection pages were built using the same layout. Each category is implemented inside of a ZStack, where the first layer is simply the color white with rounded corners and a simple box-shadow. Then, on top of the white layer is an HStack containing the icon, name of the category, and the select button. This is shown in Figure 7 (c) and (e). For the indicator categories, there is a special functionality, which is opening a popup with the definition by clicking on it. The popups were also created using another ZStack with a white background and the text on top. These popups are displayed on the top-most layer



(a) (b) (c) (d) (e)

Figure 7: Graphical redesign of the welcome pages

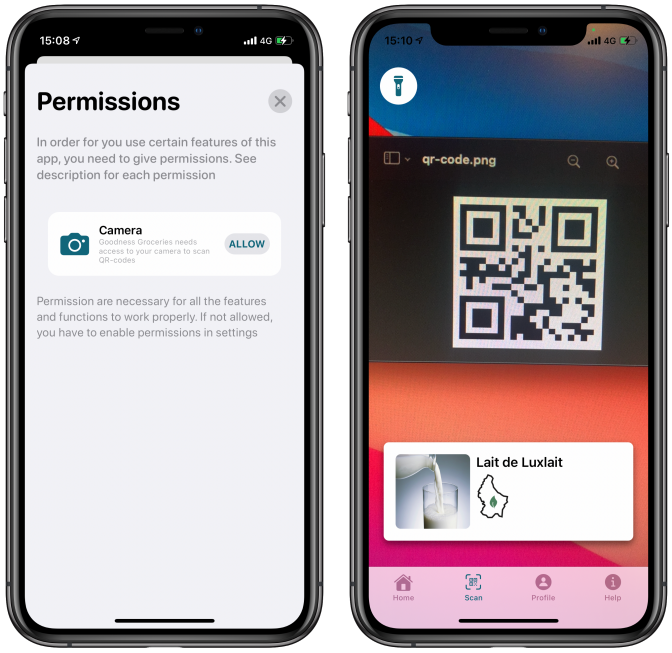
of the app's ZStack such that it is on top of all the other content. The popup can be seen in Figure 7 (d).

Categories and products

In the categories view, the new Goodness Groceries logo was added by implementing the logo image and the search bar inside of an HStack such that they are aligned side by side (see Figure 9 (a)). All the new icons were given by the sociology team and could be replaced in the Assets folder. Thus, the app contains the final version of all the icons and images apart from the product images, which can be seen in Figure 9. The new product categories page in the navigation was created by using a simple ScrollView containing a ForEach and is shown in Figure 9 (b). Each category is then implemented inside of a NavigationLink as an HStack containing the icon and title of the category. Thanks to the NavigationLink, by clicking on the category a new view is pushed and this behavior is therefore done natively by SwiftUI. The new view being pushed is the list of products corresponding to the selected product category. Inside the product view, similar products were added in a horizontal ScrollView and the indicator definitions can be displayed as popups, which is shown in Figure 9 (d) and (e).

Scanning tool

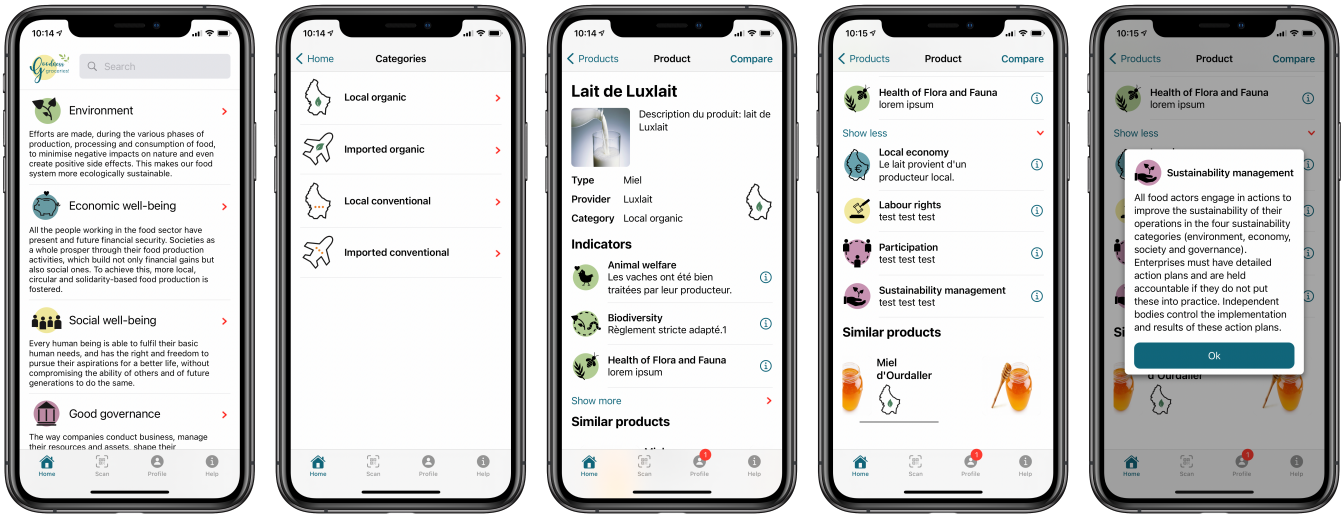
The scanning tool's permission popup was created using a third-party Swift Package called **PermissionsSwiftUI** [7]. This package allows one to ask for specific permissions, in this case only camera, to the user by first presenting them a nice and clean popup as shown in Figure 8 (a). Due to good customization possibilities, one may change the entire text of the popup to adjust it to one's individual needs. The package also incorporates an automatic checking system of each permission and only shows the popup whenever it is needed, which means if there is at least one permission remaining to be answered by the user.



(a) (b)

Figure 8: Goodness Groceries scanning tool

The torch light toggle button was created using a ZStack, which allows to place elements on top of each other. Inside of it, there is only an icon on top of a white circle. The button itself was placed in the top leading corner with an offset of 20 pixels from the top and 20 pixels from the left (see Figure 8 (b)). When the button is triggered, the torch light is toggled and the user is notified with a medium **haptic feedback**.



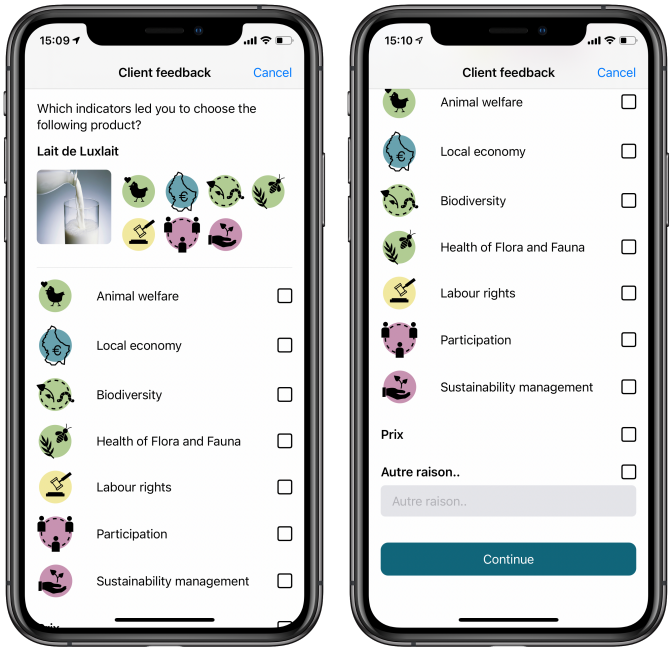
(a) (b) (c) (d) (e)

Figure 9: Graphical redesign of the categories and products pages

5.3.2. Graphical design of new views. The graphical design of the new views can by itself be split the same way as in the design section.

Survey page

The survey page was constructed such that there are 2 VStacks inside of a ScrollView, separated by a divider as shown in Figure 10. The first VStack contains a small introductory text as well as an HStack with the product information and the indicator icons.



(a) (b)

Figure 10: Goodness Groceries survey page

Then, the second VStack contains a ForEach of all the indicators corresponding to the specific product. Each row of indicators is embedded into an HStack with the indicator icon, name, and a checkbox allowing the user to select the indicator. After all the indicators, there is an extra checkbox for the *price* indicator and another checkbox and text field for the *other reason* indicator. Finally, at the end is the continue button. This button calls a function that uses a third-party Swift Package called **Alamofire** [25]. Alamofire is a networking library, allowing one to create simple network requests. Using this library, a POST request is sent to the web application in order to store the information entered by the user.

Push Notifications

To send out push notifications to the users, we need a so-called *device token*. This device token is a unique identifier of the user's device for a specific app. Thus, for every combination of device and app, there is a different unique token. Since each token can uniquely identify the device and app, it can be observed as the *address* to send the notification to (similar to sending emails to an email address).

The device token generation and parsing is shown in Figure 11.

1) Once the user enables push notifications, a request is sent to the APN service in order to register for remote notifications and obtain the device token.
2) APNs will then on their end generate the device token and send it back to the device. An example of a generated device token is: **5b8843b0f29333c0688fddd0bd74b0d624a6a2cd**.
3) The device token is then passed to the client app, which in this case is Goodness Groceries. This means that we now have access to the token in the app.
4) Once we have access to the token in the app, we can send it to the provider, which in this case is the

web application of Goodness Groceries. This token will be stored in the database and be used to send push notifications from the server directly.

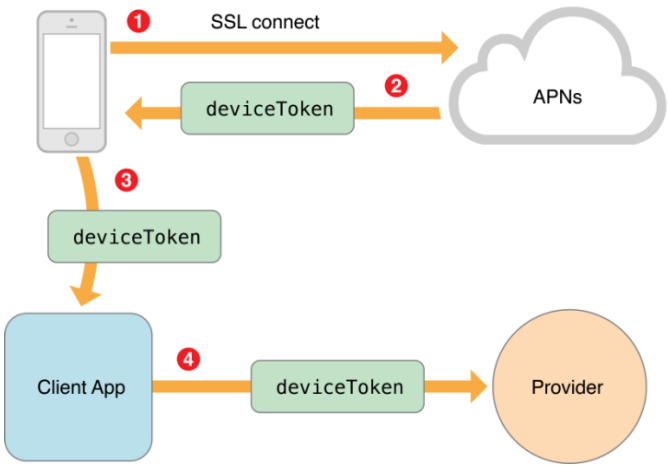


Figure 11: APNs device token generation (from [22])

Once the device token is stored in the database, the app is now capable of sending push notifications to the desired devices. Sending a push notification is shown in Figure 13.

5) The provider, or web application in the Goodness Groceries case, can now send a push notification to a specific device. To do so, we need to send a payload containing the notification data, such as title, body, and sound, to the APN service. An example of such a payload is shown in Figure 12. To identify the receiver of the notification, we also need to send the device token.
6) The notification is then received by the APNs and routed to the device.

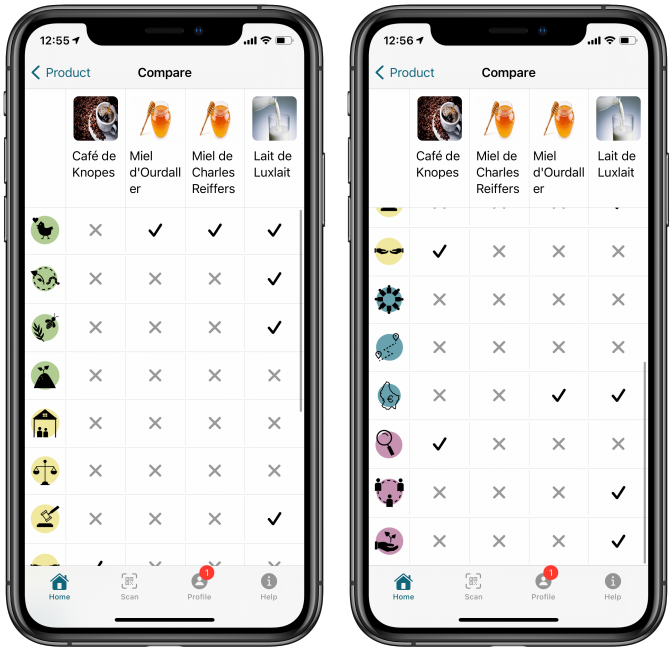
```
'5b8843b0f29333c0688fddd0bd74b0d624a6a2cd',
payload_data={
    'aps': {'mutable-content': 1, 'alert': {'title':
    'Notification title', 'body': 'Notification text'}, 'sound':
    'default', 'badge': 1},
    'products': ['2354896578839']
}
```

Figure 12: Push notification payload example with device token

Figure 13: Sending push notifications through APNs (from [22])

Compare page

The compare page was created in a very basic way. The device width is first divided into 5 columns. All these columns are placed inside a VStack. Apart from the first row containing the product information, all the rows are contained inside of a ScrollView. This allows the user to scroll through all the rows but keeping the product information stuck to the top of the screen as shown in Figure 14. The rows are created using a ForEach of all the indicators and placed inside of an HStack. Finally, depending on whether the current product has the indicator or not, we place a check or a cross icon. All the columns and rows also use horizontal and vertical dividers, which create the borders of the grid. This is achieved by not applying any padding to the dividers, such that they connect with one another.

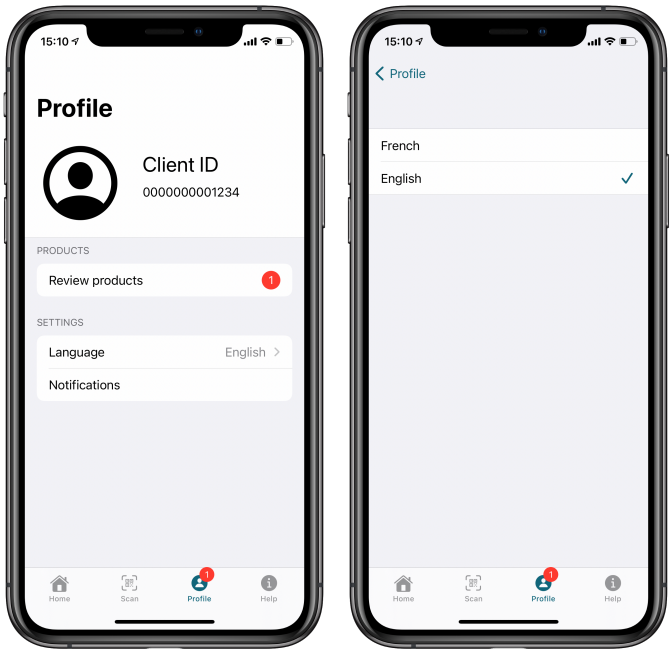


(a) (b)

Figure 14: Goodness Groceries product compare page

Profile page

The profile page is a simple VStack containing an HStack with the client ID information and a SwiftUI Form, which contains the settings at the bottom of the page shown in Figure 15 (a). The Form is a combination of sections and custom buttons, thus we have 2 sections and 3 buttons in total. The first section allows the user to manually start the product review process, which is disabled when there are no products to review. The second section allows the user to change the language settings of the app as shown in Figure 15 (b), as well as accessing the phone settings for the notifications.

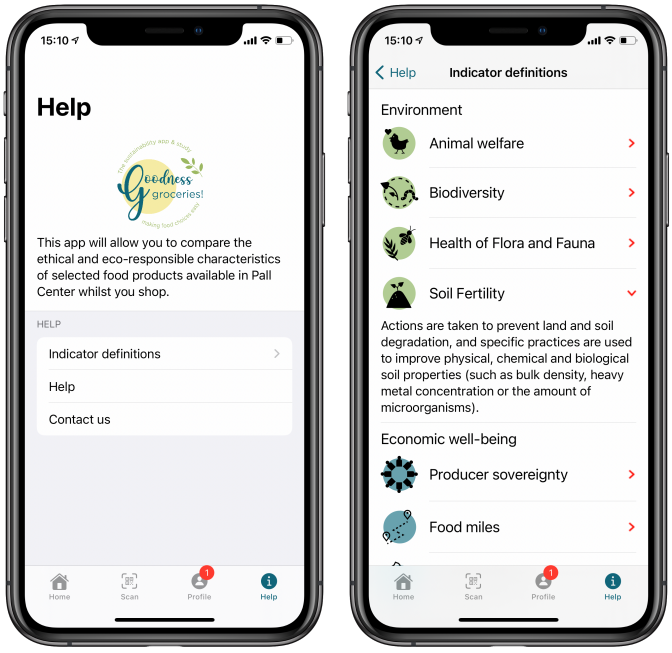


(a) (b)

Figure 15: Goodness Groceries profile page

Help page

The help page uses the same graphical layout as the profile page as shown in Figure 16 (a). However, we only have 1 single section. The first button shows the user the list of indicator definitions.



(a) (b)

Figure 16: Goodness Groceries help page

This list is created using a ForEach for all the indicators, and then using an onTapGesture we can either hide or show the definition. This behavior is shown in Figure 16 (b). The remaining 2 buttons redirect the user to the university website about the project.

5.4. Assessment

Each requirement was successfully achieved and implemented in the application:

- FR1: Every time the app is opened and the user is connected to the internet, the app fetches the products from the web server and updates the products to be reviewed for the user. Every time the user fills out the product survey page, the results are sent back to the server and stored in the database. Thus, the requirement is met.
- FR2: Since the app disposes of a fully functional comparison tool, the user can compare multiple products at once and make their decision based on the provided data. Thus, the requirement is successfully met.
- FR3: Finally, the app stores the preferred language of the user and is able to display everything in the selected language as the entire application was translated into English and French. Thus, the requirement was also achieved.

Overall, the end result of the app is very good and can be considered a success. Each requirement was met and the deliverable is a near-final version of the Goodness Groceries application which is fully functional. The final adjustments and modifications will be done outside of this bachelor semester project.

Acknowledgment

I would like to thank my Tutor, Benoît Ries, for the help and support during the entire semester. His advice and encouragement were fundamental to the success of the entire project.

6. Conclusion

This paper presents a bachelor semester project whose goal is the final development of the eco-friendly mobile application *Goodness Groceries* and the exploration and illustration of encryption algorithms and iOS security measures in general. The scientific part focuses on the encryption algorithms and iOS security measures, while the technical part is the final development of the application itself. In the end, the app achieved every functional requirement set and provides all the expected functionalities. At the end of the semester, the app is in a near-final state, while there only remain small modifications to be done.

After all, this project allowed me to learn a lot about security measures in iOS development, especially encryption algorithms in general such as AES. While doing the technical deliverable I was also able to further improve my iOS development skills in Swift and SwiftUI.

References

[1] BiCS Bachelor Semester Project Report Template. https://github.com/nicolasguelfi/lu.uni.course.bics.global University of Luxembourg, BiCS - Bachelor in Computer Science (2017).

[2] "Understanding iOS App Security Practices — NIX Solutions," NIX Solutions, Jun. 26, 2018. https://www.nixsolutions.com/blog/understanding-ios-app-security-practices/ (accessed Mar. 16, 2021).

[3] "Why is data encryption and protection so important? - TeamDrive - Secure Collaboration," TeamDrive - Secure Collaboration, Jul. 10, 2019. https://teamdrive.com/en/why-data-encryption-important/ (accessed Mar. 16, 2021).

[4] "Xcode - SwiftUI- Apple Developer," Apple.com, 2021. https://developer.apple.com/xcode/swiftui/ (accessed Mar. 16, 2021).

[5] "Flowchart of the AES algorithm. Encryption process.," ResearchGate, Dec. 2012. https://www.researchgate.net/figure/Flowchart-of-the-AES-algorithm-Encryption-process_fig4_233828516 (accessed Mar. 23, 2021).

[6] D. Mcgrew and J. Viega, "The Galois/Counter Mode of Operation (GCM),." [Online]. Available: https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

[7] jevonmao, "jevonmao/PermissionsSwiftUI," GitHub, Apr. 11, 2021. https://github.com/jevonmao/PermissionsSwiftUI (accessed May 06, 2021).

[8] What is Symmetric Key Cryptography Encryption? — Security Wiki, Secret Double Octopus, 2018. https://doubleoctopus.com/security-wiki/encryption-and-cryptography/symmetric-key-cryptography/ (accessed May 14, 2021).

[9] Network Encyclopedia, "Hashing Algorithm — Network Encyclopedia," Network Encyclopedia, Aug. 29, 2019. https://networkencyclopedia.com/hashing-algorithm/ (accessed May 14, 2021).

[10] "What is AES Encryption and How Does It Work? — CyberNews," CyberNews, Dec. 11, 2020. https://cybernews.com/resources/what-is-aes-encryption/ (accessed May 19, 2021).

[11] Edpresso Team, "What is the DES algorithm?," Educative: Interactive Courses for Software Developers, Aug. 29, 2019. https://www.educative.io/edpresso/what-is-the-des-algorithm (accessed May 19, 2021).

[12] Wikipedia Contributors, "MD4," Wikipedia, Apr. 22, 2021. https://en.wikipedia.org/wiki/MD4 (accessed May 19, 2021).

[13] Arnoud Engelfriet, "The MD5 cryptographic hash function (in Technology hashfunctions @ iusmentis.com)," Iusmentis.com, 2021. https://www.iusmentis.com/technology/hashfunctions/md5/ (accessed May 19, 2021).

[14] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," Sep. 2001, doi: 10.17487/rfc3174.

[15] Wikipedia Contributors, "SHA-2," Wikipedia, May 15, 2021. https://en.wikipedia.org/wiki/SHA-2 (accessed May 19, 2021).

[16] "Rijndael key schedule," Crypto Wiki, 2021. https://cryptography.fandom.com/wiki/Rijndael_key_schedule (accessed May 19, 2021).

[17] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-Middle Attack to the HTTPS Protocol," IEEE Security Privacy Magazine, vol. 7, no. 1, pp. 78–81, Jan. 2009, doi: 10.1109/msp.2009.12.

[18] F. Liu, K.-S. Liu, C. Chang, and Y. Wang, "Research on the Technology of iOS Jailbreak," 2016 Sixth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC), Jul. 2016, doi: 10.1109/imccc.2016.178.

[19] "Local and Remote Notification Programming Guide: APNs Overview," Apple.com, Jun. 04, 2018. https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html//apple_ref/doc/uid/TP40008194-CH8-SW1 (accessed May 19, 2021).

[20] F. De Jesus Matias, "Goodness Groceries - Eco-Friendly Mobile Application for iOS," BiCS Semester Project (BSP) Report, Jan. 2021.

[21] C. Anderson, "The Model-View-ViewModel (MVVM) Design Pattern," Pro Business Applications with Silverlight 5, pp. 461–499, 2012, doi: 10.1007/978-1-4302-3501-9$_1$3.

[22] "iOS Push Notification Swift 2 tutorial — code explanation download," Mobile, WordPress Web App Development Company - AlphansoTech, Dec. 16, 2015. https://www.alphansotech.com/ios-push-notification-tutorial-swift-2 (accessed May 20, 2021).

[23] M. Cobb, "Advanced Encryption Standard (AES)," SearchSecurity, 2017. https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard (accessed May 20, 2021).

[24] "AES — cryptology — Britannica," Encyclopædia Britannica. 2021, Accessed: May 20, 2021. [Online]. Available: https://www.britannica.com/topic/AES.

[25] Alamofire, "Alamofire/Alamofire," GitHub, May 13, 2021. https://github.com/Alamofire/Alamofire (accessed May 21, 2021).

[25] "What is HTTPS? - SSL.com," SSL.com, Jul. 13, 2020. https://www.ssl.com/faqs/what-is-https/ (accessed May 26, 2021).

7. Appendix

7.1. Application source code

The entire application source code for this project can be found in this Github repository: **https://github.com/Flavio8699/Goodness-Groceries**.

7.2. Application screenshots

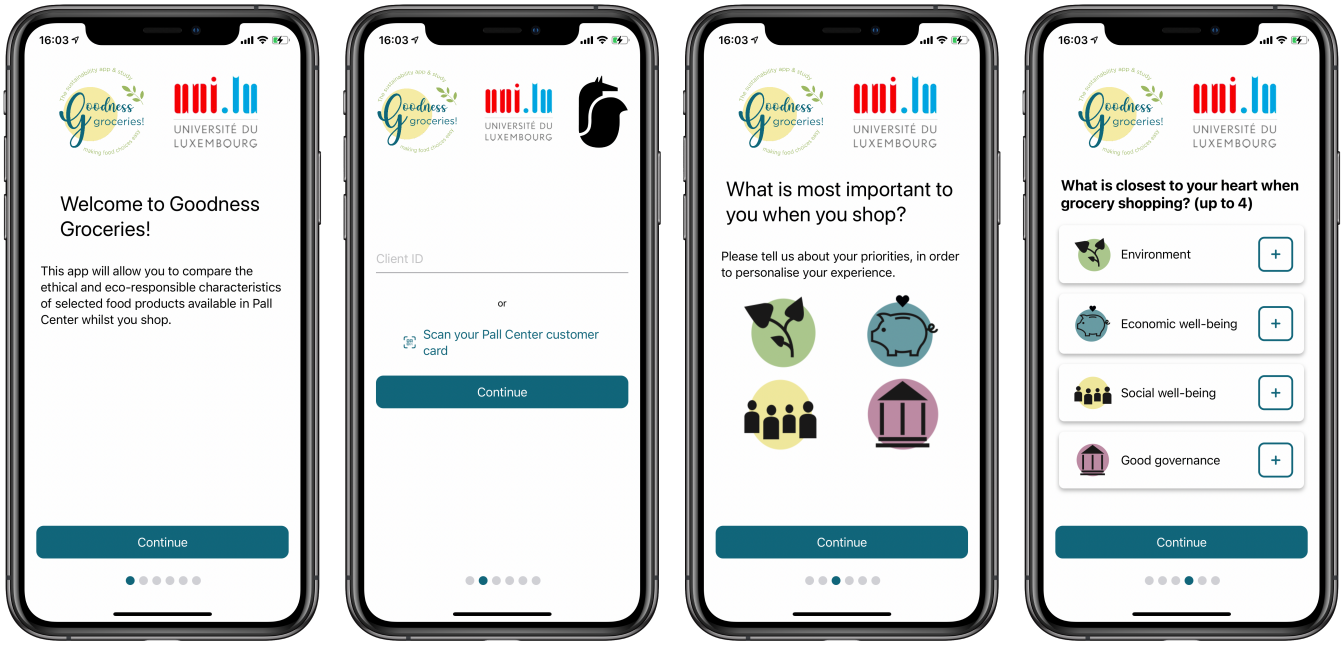


Figure 17: Appendix 1

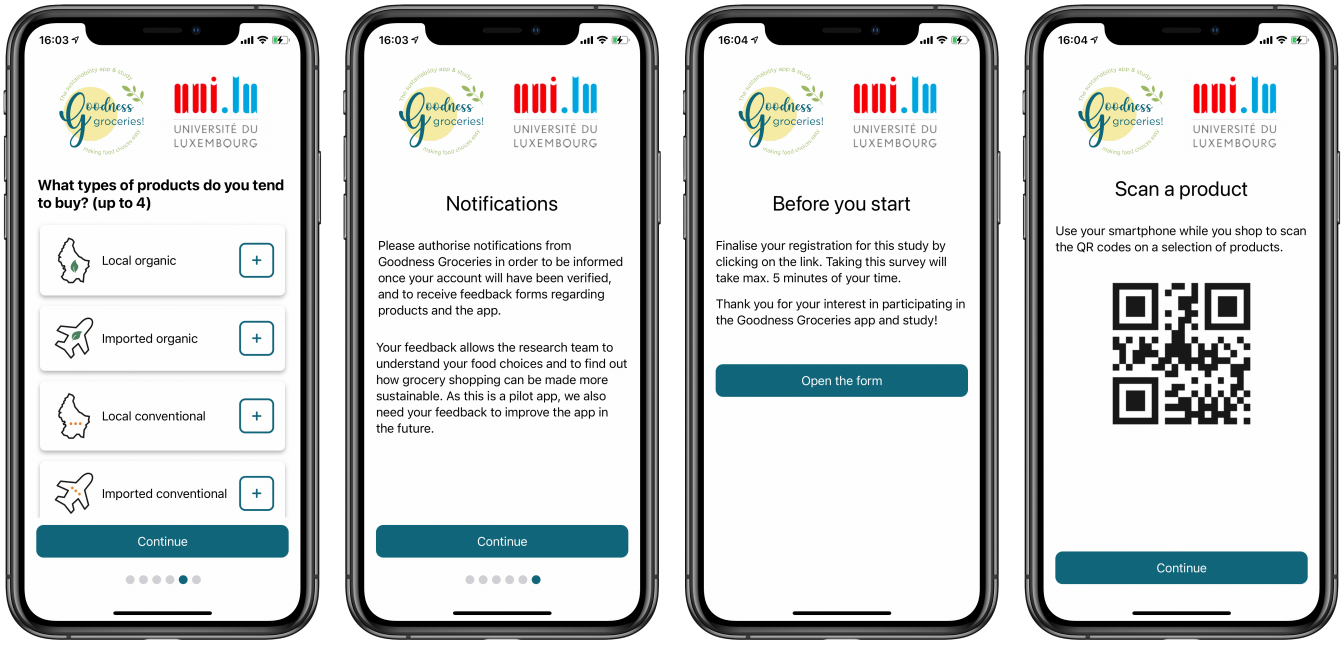


Figure 18: Appendix 2

Figure 19: Appendix 3

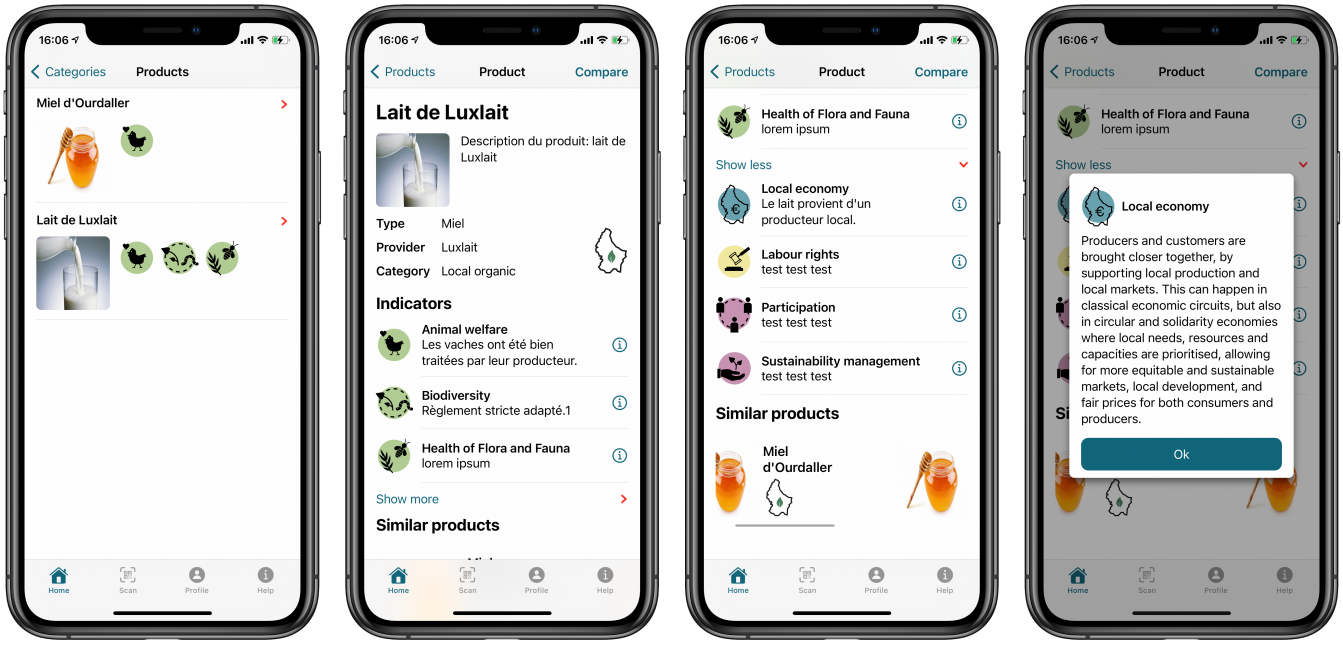


Figure 20: Appendix 4

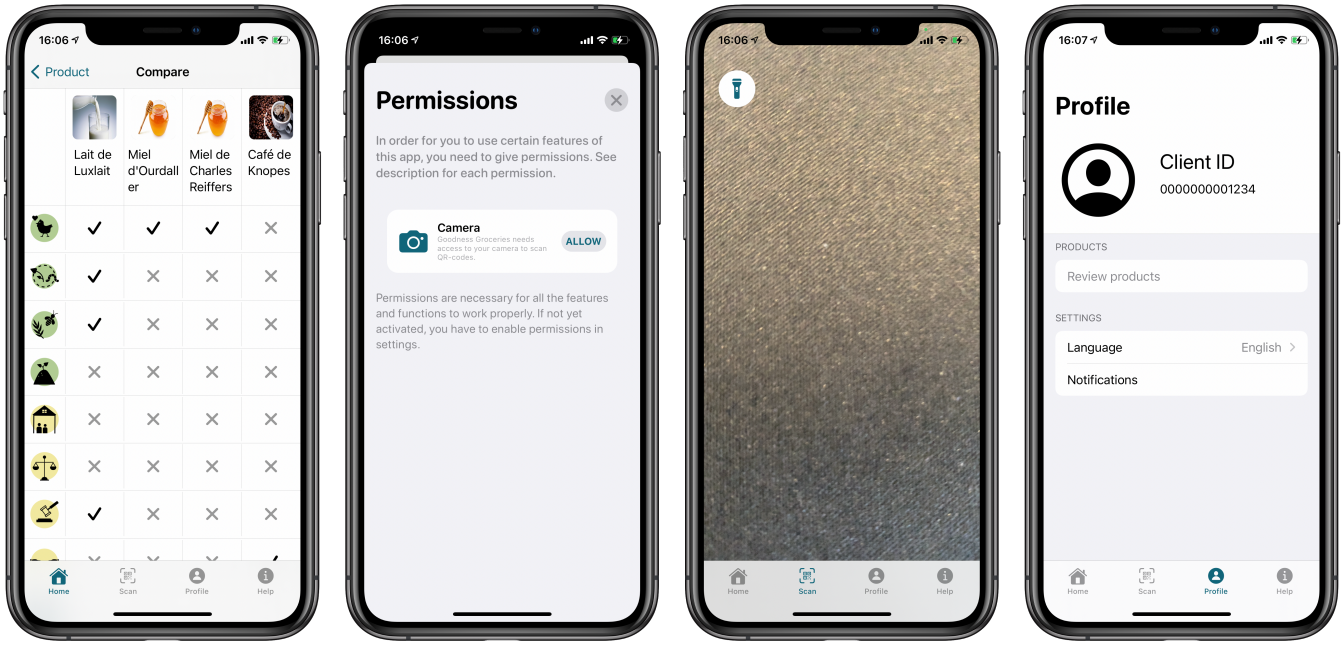


Figure 21: Appendix 5

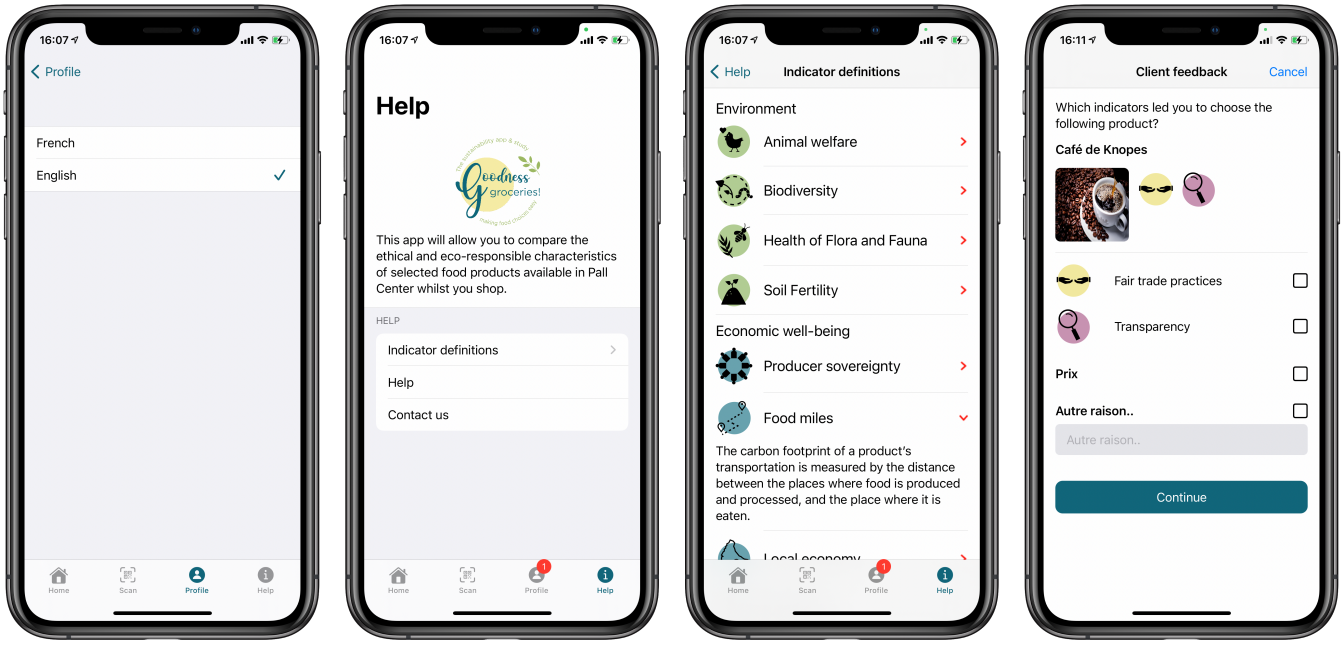


Figure 22: Appendix 6