Goodness Groceries - Eco-Friendly Mobile Application for iOS

Tuesday 15th December, 2020 - 15:31

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 developing an eco-friendly mobile application for iOS 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 use of evaluation techniques for mobile applications to evaluate the produced technical deliverable. The scientific question treated in this paper is "How can User Experience (UX) be used to evaluate the quality of the user interaction of the Goodness Groceries app". The technical section of the project will present the implementation of the design prototype as an iOS application.**

1. Introduction

Ethical shopping has been a relatively new trend for most consumers around the world. A lot of people don't pay attention to the provenance of the products they buy, as well as the company that hides behind it. Most of these huge companies do not respect peoples values and in fact, use their revenue to actively work on the opposite.

The so-called *conscious consumer* is someone who does not fall in this category of people. This kind of consumer opts to spend more money on products that come from transparent companies that support the same values as they do. They look beyond the label of the product and want to know more about the respective company since by buying a product from a certain company you are also funding and supporting it.

This project comes in handy for people starting to pay attention to what they buy, along with people that already practice this kind of shopping. The goal of this application is to promote ethical and eco-responsible shopping to the consumers by giving them an easy approach to consult the different products they are willing to buy. It provides them with a transparent platform to increase their knowledge of the companies that hide behind certain products. By adjusting and re-evaluating their shopping habits, customers can increase their lifestyle in addition to contributing to better economics through actively supporting fairer and eco-friendlier companies and boycotting the giants of grocery shopping.

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 Human-Computer Interaction (HCI), in particular User Experience (UX). HCI is the study of the way in which technology can possibly influence human work and activities and vice versa. In this project, we will focus on UX, which focuses on what users perceive while using a certain product.

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 with 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 [8] 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 with 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 specification and execution of a User Experience evaluation of the technical deliverable. The deliverable will contain a process which will focus on the evaluation of the latest version of the technical deliverable at the end of the semester. This process should focus on multiple attributes and target each one separately. At the end of the deliverable, the UX evaluation results will be presented and analysed in order to draw a conclusion on whether the app satisfies the set expectations and if it can be considered as successful or not.

2.2.2. Technical. The targeted technical deliverable of this project is the iOS application *Goodness Groceries* following the given requirements. The deliverable will be a functional first version of the app while following the given design and template choices. The developed app should contain all the necessary functionalities for the user to search, scan and look at products overall. This will lead to accomplishing the goal of the app which is promoting eco-friendly and responsible shopping for the customers.

3. Prerequisites

3.1. Scientific

The scientific prerequisite of this project is the knowledge of a simple sophomore Computer Science student (3rd-semester student). All the necessary scientific tasks for this project, such as User Experience evaluation techniques, will be presented and illustrated in the scientific deliverable section. Thus, no knowledge is required in the field of UX to understand the report of the project.

3.2. Technical

The technical prerequisite of this project is the knowledge in the programming language Swift alongside with 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. UX evaluation of the mobile application

4.1. Requirements

The main requirement of this deliverable is to present a process to evaluate the produced technical deliverable, which is the iOS mobile application. The focus here lies on the execution of user-experience evaluation by targeting nine usability attributes used for evaluating mobile applications. We can agree on two different functional requirements for the scientific deliverable.

- FR1: The deliverable shall present a process for the UX evaluation of the application.
- FR2: The deliverable shall present the results of the presented evaluation process applied to the technical deliverable.

4.2. Design

In this subsection, we will discuss how the evaluation process is designed and executed. The best way to present the evaluation process is by dividing it into three parts because it highlights all the major steps done during the production of the deliverable, and those are: the **evaluation process** itself, the **running process** and finally, the **UX result analysis**.

In the first part, the process itself will be described. The structure of the evaluation process is precisely described and divided into the different steps that will be executed in chronological order. This provides a more structured presentation and is more comprehensible for the reader in the end effect.

In the second part, the running process will be explained. Here we focus on how the evaluation process went with the participants and how they felt while using the app and answering the evaluation sheet. This part is especially interesting because it reflects the emotions and opinions of the first few people to test and use the application.

Finally, in the last part, we will present and analyse the obtained results from the evaluation. Moreover, we will discuss the significance of the results related to the app. This part is important because it reflects how effective the app is in the end effect. The results are also interesting because they can be used to improve the app in the future.

4.3. Production

User Experience describes the feelings that users perceive when using a certain app, service or product [4]. According to the official definition of ISO 9241-110:2010, it is *a person's perceptions and responses that result from the use and/or anticipated use of a product, system or service* [5]. This deliverable focuses on presenting an evaluation process for UX. User Experience evaluation can be defined as the investigation and measurement of how users feel when using a particular product, app or system [6].

4.3.1. Evaluation process. The evaluation process was designed in such a way, that we focus on nine of the most used attributes for user-experience evaluation. *Avouris* et al. [2] defined these nine attributes to help evaluating UX on mobile applications. The **nine attributes** are as follows: learnability, efficiency, memorability, user errors, user satisfaction, effectiveness, simplicity, comprehensibility and just as importantly, learning performance.

The evaluation process will be conducted with volunteer participants where each participant will fill out the evaluation sheet. This sheet will be very similar to an exam sheet combined with a survey. There are three different sheets that will be given to the participant: a **missions**, a **questions** and a **useful barcodes** sheet.

The evaluation process is composed of multiple steps.

1) Every participant not yet familiar with the concept and objective of this project will be given a detailed explanation of what the application will be used for and what the goal and motivation of the study is.
2) After the participant feels comfortable and understands the project, they get to use the app for the first time and are given the sheet containing the **useful barcodes** which might come in handy while using the application. Each participant will be given some free time to navigate through the app by itself without help before the process starts. The appropriate time will be around 5-10 minutes, depending on the knowledge the participant has in this field as well as how comfortable they feel about using an unknown app. Clearly, computer scientists have an advantage and most certainly won't need as much time as other people to get familiar with the application. This free usage time allows them to explore by themselves and also possibly already discover some features that will help them accomplish the missions later on.
3) The first sheet with the missions will be given to the participant after these minutes run out and the evaluation process can start.
4) Participants are now expected to fill out the missions sheet by consulting the app. The time required by each participant is very hard to predict as some participants might have more difficulties and take longer to accomplish the missions compared to others.
5) Once the missions are done, the participant can put the phone away and will be given the questions sheet. To fill out the questions sheet, only a few minutes should be required and this should also mark the end of the evaluation process.

Missions sheet
The first sheet is the missions sheet. The missions are not necessarily part of the evaluation itself but were designed to give the participant goals and tasks to complete in order to have an objective while using the app. Without real objectives, participants might only navigate around and in the end, are not able to evaluate the real usability of the app because they did not really use it in a real-world scenario. The missions should simulate actions that the participant would perform when using the app while doing their groceries shopping as well as help them understand the app by showing them what one can do with it..

Questions sheet
The second sheet consists of questions defined to precisely see how well the application performs in different categories. These categories are defined by the nine attributes. This means that for each of these attributes, one question was picked in order to evaluate whether it was achieved or not. Here, the participant can answer the questions with a ranking from 1-5, however not using numbers. The numbers from 1-5 are represented by words, for instance, very bad, bad, normal, good and very good. This helps the participant understand better how to answer a question and perform better results. At the end of the questions sheet, the participant may or may not enter a free comment where he can propose some improvements or ideas for the app. This would help in future development because it would be the first real feedback from people possibly targeted by the application itself, i.e. people that could potentially participate in such a research study.

Useful barcodes sheet
The last sheet contains **useful barcodes** whose purpose is to help the participant to do the missions and use the app overall. This sheet contains an example of a *carte de fidélité*, which the participants can use to authenticate themselves to gain access to the app. Since at the time of the evaluation process the final list of products for the study was not yet defined, five products were created using fictitious data. To achieve a more realistic usage of the application, five different QR-codes corresponding to the products were also added to the useful barcodes sheet which can be scanned by the participant.

4.3.2. Running process. The evaluation process was conducted with five different volunteers. Each evaluation was done individually with the participant and took approximately between 45 and 90 minutes in total. Participants that were not yet familiar with the project also asked questions and showed interest in the study, which usually also increased the evaluation time.

Two of the five participants have a background in computer science. The three other participants are people which could be in the position of a volunteer for the project, i.e., someone which will use the app in its final production state once the study begins. Splitting the participants like this gives us a more technical oriented opinion on the app, as well as a more usability and practicability oriented opinion from the participants that could use the app in a real-world scenario.

The evaluation with the computer scientist participants was organised over a Webex meeting (restrictions related to the Covid-19 virus). The other participants were people close to the student, which allowed for an in-person evaluation. The time taken by both categories of participants was considerably close, as the student had to help the computer

scientists to setup the application on their smartphone. The same time was approximately used to explain the project to the other participants, as no time was lost to setup the app, because they simply used the student's smartphone.

4.3.3. UX result analysis. Each participant filled out a questions sheet, in which each question corresponds to a certain evaluation attribute. Their job was to evaluate how good or bad each specific attribute was fulfilled in the app, according to their personal opinion. The questions were evaluated using a 1-5 ranking, which were represented by words such as, for instance, *very bad*, *bad*, *normal*, *good* and *very good*. The best mark, which is 5, was therefore indicated by *very good*.

Attribute	P-1	P-2	P-3	P-4	P-5	Average
Learnability	4	5	4	5	5	4.6
Efficiency	5	4	5	5	4	4.6
Memorability	5	5	4	5	5	4.8
User errors	5	5	3	4	2	3.8
User satisfaction	5	4	4	4	5	4.4
Effectiveness	5	4	5	5	5	4.8
Simplicity	5	5	4	5	4	4.6
Comprehensibility	5	5	3	5	5	4.6
Learning performance	5	5	5	4	5	4.8

Table 1: Evaluation results for each attribute with 5 participants

As shown in Table 1, each participant evaluated each attribute. The average of each of them is also outstanding, as except for one, all are above 4.0. The overall average (average of the averages) of all the attributes is a remarkable **4.55**, which is a very good result.

The only attribute which is below 4.0 is **user errors**. This attribute is, compared to all the others, significantly low due to **three different problems** encountered during the evaluation:

- When scanning and opening the page of a product, the camera stayed active in the background. The problem associated with this is the following: when the participant moves the phone and the camera captures another QR-code, the page is updated to the new product. This creates confusion and the user doesn't understand what happened and whether they did something wrong or not. One way to solve this issue is improving the technical deliverable by deactivating the camera when the user is viewing a certain product and only resume the scanning when the user closes the view.
- When using the scanning tool, due to an unknown bug, the *go back button* on the top left corner disappeared after looking at the product view page. This problem left the participant with no other choice than closing an reopening the application, as no logs were produced during the incident and the app did not crash. It was also the first and only time that this kind of problem occurred. This will be logged in known errors in the technical deliverable package. This known errors will come with a workaround in case this happens again in the future.
- Multiple participants reported difficulties trying to close the keyboard after using the search function. As the close button is relatively small and only appears when there is text in the search bar, they did not know how to close the keyboard. A possible solution to this issue is to add a close button next to the search bar when it is in use or to close the keyboard automatically when the user taps on a different section of the screen.

These three problems perfectly summarise the comments given by the participants as it were things that stood out to them. This helps us understand where problems still exist in the app, which can be fixed in the future.

To conclude, we can clearly see that the application is a success based on the evaluation results. Each attribute ended with a score between *good* and *very good*, except for one. This attribute however, gave us a clear look at some problems that are still present in the app, which is also a good result in the end effect.

4.4. Assessment

Each requirement set for this deliverable was achieved since we managed to present an evaluation process and the results in the end. The evaluation process could have been better if there were participants with different backgrounds and from different age groups. By including more participants with a background in, among others, computer science and sociology, the results, in the end, could have been more accurate. The participants could have also been divided into different age groups, as the results most likely differ between them. However, in the end, the results are still very good and give a clear vision of the strengths and weaknesses of the app in the current state.

Overall, the participants were very positively impressed by the project and the app itself. This could also be noticed on the fact that most of them asked multiple questions about the project in general and gave very positive feedback about the app. The collected comments given by the participants will be presented and discussed in the next section, which holds the evaluation results.

5. Eco-friendly mobile application for iOS

5.1. Requirements

For the eco-friendly mobile application, there are several functional requirements that need to be met for the application to be successful. These requirements were given by the Pall Center and sociology team members. They summarize the features that need to be implemented in the final application.

- FR1: The first requirement is that each product needs to be displayed in a well-structured view including the corresponding attributes. The app should present the customers with a simple layout which highlights the product indicators whenever they consult a specific product page. This is especially important because the main objective of the app is to promote eco-friendly shopping by showing the product indicators.
- FR2: The app shall provide the customers with a quick way to find products by allowing them to scan QR-codes with the camera of their smartphone. This will help them find the desired product quickly and with low effort. Giving users multiple ways to reach their desired outcome gives them the choice between which path to take. Customers that feel more comfortable using an unknown application might save time and enjoy the process of scanning for products in the supermarket aisles.
- FR3: The third requirement is that the app should be capable of establishing a stable connection to the distant web server. The connection between the two services is important because it will allow customers to send an authentication request to the server. In addition, the app should be able to consult the user status by sending requests to the distant webserver. Thus, it is of great importance that the application can send and receive data from the server.

5.2. Design

The first version of the graphical design was prototyped by a prior student using Figma, which is an interface design tool. The given design was not perfect since it did not entirely follow the **Human Interface Guidelines for iOS** provided by Apple and to improve it the feedback given by the sociologists and Pall Center team members were taken into account. The student could however base himself on the prototype and build the app while correcting and improving the given layout while keeping the general idea behind the design and structure untouched. The design can be divided into 2 main categories: **code architecture** and **graphical navigation design**.

5.2.1. Code architecture. Design patterns usually offer solutions to common problems. They can be used to solve code design problems when designing software. Using a good architecture design pattern for the code structure is very important since it organises the code in a smart and efficient way. The code for the application was designed using the **Model-View-ViewModel** (MVVM) architecture as it is one of the best design patterns created for mobile development since it is very simple to implement, yet very effective in its purpose. It has also been among the favourites of iOS developers in recent years. It was invented by two Microsoft architects **Ken Cooper** and **Ted Peters** [7] and introduced to the public in 2005. The goal of the design pattern is to separate the views (which implement the graphical design part) from the backend logic.

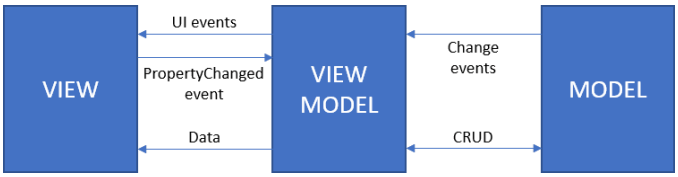


Figure 1: Model-View-ViewModel (MVVM) structure [3]

As we can see in Figure 1, each of the 3 components of this architecture have their own purpose and functionality.

Model
Models contain raw data from the application. **CRUD** (CREATE-READ-UPDATE-DELETE) operations can be performed to manipulate the data by the ViewModels. Usually, applications contain a Model for each type of information that needs to be stored in the database. They provide ViewModels with data and also inform and update them on changes. The only job of a Model is to store data, but not modify it themselves. The business logic is typically separated from the Models.

View
Views hold the user interface (UI) of the application. They hold the design code that forms the output which users get to see. All the data displayed in a view is passed in from the ViewModels. When the user triggers an event in the view, the ViewModel gets informed and performs the desired action by the user in the background.

ViewModel
ViewModels represent the middle man between the Views and Models. They populate Models with data and perform CRUD operations on them. Usually, Views use objects that are bound to variables defined in ViewModels and as a result, when a user changes it by performing a certain action, a *propertyChanged*-event gets fired and informs the ViewModel about the change. This will then handle the change with a custom action, such as storing or validating the given information. The data that gets displayed in a View is requested to the Models by the ViewModel and then passed to the View. Using ViewModels, the Views and Models don't know the state of each other and cannot communicate. They also contain methods and commands that help to manipulate the Models and keeping the Views up-to-date.

5.2.2. Graphical navigation design. The graphical navigation design focuses on presenting the navigation between the views and their graphical design. It can be divided into **3** sub-categories: the **welcome pages**, **categories and products** and the **scanning tool**.

Welcome pages
The welcome pages were designed in such a way that they guide the customer through the app by explaining the goal and functionalities. Therefore, the very first page viewed by the user when opening the app is a large screen welcoming the user to the app. Here the app itself is explained in a relatively short way in order to introduce the user. The next screen invites the user to sign in using their *carte de fidélité* either by typing their ID manually or more conveniently by scanning the card. After successfully authenticating themselves, the user is introduced to a couple of questions to define their shopping habits. Here the user will indicate what type of products they buy and what attributes are more important to them while shopping.

Finally, the user access request will be sent to the server and the user just needs to fill out a form indicated by a link.

Categories and products
Once the customer is successfully authenticated and validated by the research team, he can finally use the app freely. Here customers will be welcomed by a large screen listing all the product categories with a small description and a search bar on the top, where they can search for a specific product faster. When clicking on a specific category, a new view is opened where all the appropriate products are listed alongside the corresponding indicators. Once the user decides on a product a wants to see more, a new view is opened by clicking on it. Here, all the information about the product is displayed. From top to bottom, we have the product name, the image and the description. Then, we have the type of the product as well as the provider. If there are coordinates of the provider available, a small map also displays the location. Finally, at the bottom of the page, we have all the indicators. Only the indicators corresponding to the selected category are listed by default, in order to display all the indicators the user needs to click on *see more*.

Scanning tool
On the menu bar positioned at the bottom of the screen is another tab which is the scanning tool. Here, customers can use the camera of their smartphone to scan QR-codes of different products. When a product is recognised, a small popup appears on the screen and when clicking on it, the detailed product page is opened. This gives the customers a fast way to look up products while doing their groceries shopping.

5.3. Production

This section is split into four parts: **Models**, **ViewModels**, **Handler classes** and the **Views**.

5.3.1. Models. The Models were defined to structure the required data in the app. Since multiple products, indicators and categories are loaded from local static files, a dedicated Model for each was required. Each of these Models contain the necessary attributes to uniquely identify each instance of itself. The data stored in the Models will then be accessed by the ViewModels in order to either pass it on to the Views for display or for direct manipulation of the data. In total there are 6 Models: **Product**, **UserStatus**, **Category**, **Indicator**, **ProductIndicator** and **ProductCategoryWelcome**.

```
3  struct Indicator: Hashable, Decodable {
4      var id: String
5      var name: String
6      var category_id: String
7      var icon_name: String
8      var general_description: String
9  }
```

Figure 4: Structure of the Indicator Model

```
3  struct Category: Identifiable, Hashable, Decodable {
4      var id: String
5      var name: String
6      var icon_name: String
7      var description: String
8      var color: String
9  }
```

Figure 5: Structure of the Category Model

Figures 4 and 5 illustrate the structure of the Indicator and Category Models. Here one can see each attribute that the corresponding Models have, such as name, description, colour, etc.

5.3.2. ViewModels. In this application, the ViewModels were mainly assigned the job of loading the data from the JSON files and populating the Models. The data consists of multiple static JSON files which contain information about the products, indicators and categories. The content of these files is provided by the Pall Center and sociologists team and will not change during the project. 2 different ViewModels were created: *ProductsViewModel* and *CategoriesViewModel*. Both of them are initialised with empty lists of the according type, while the ProductsViewModel is also responsible of loading and parsing the indicators, since they are directly related to the products. These lists are decorated using the **@Published** property wrapper in SwiftUI. This property wrapper automatically creates observable objects, which means that it will systematically monitor the changes. Whenever something changes, the ViewModel will therefore inform all the Views that are using this list about the changes and update the data automatically. This is particularly useful because we load the data into a single list which will then be used in several places across the application. As a benefit, we keep the app's data coherant across all the Views.

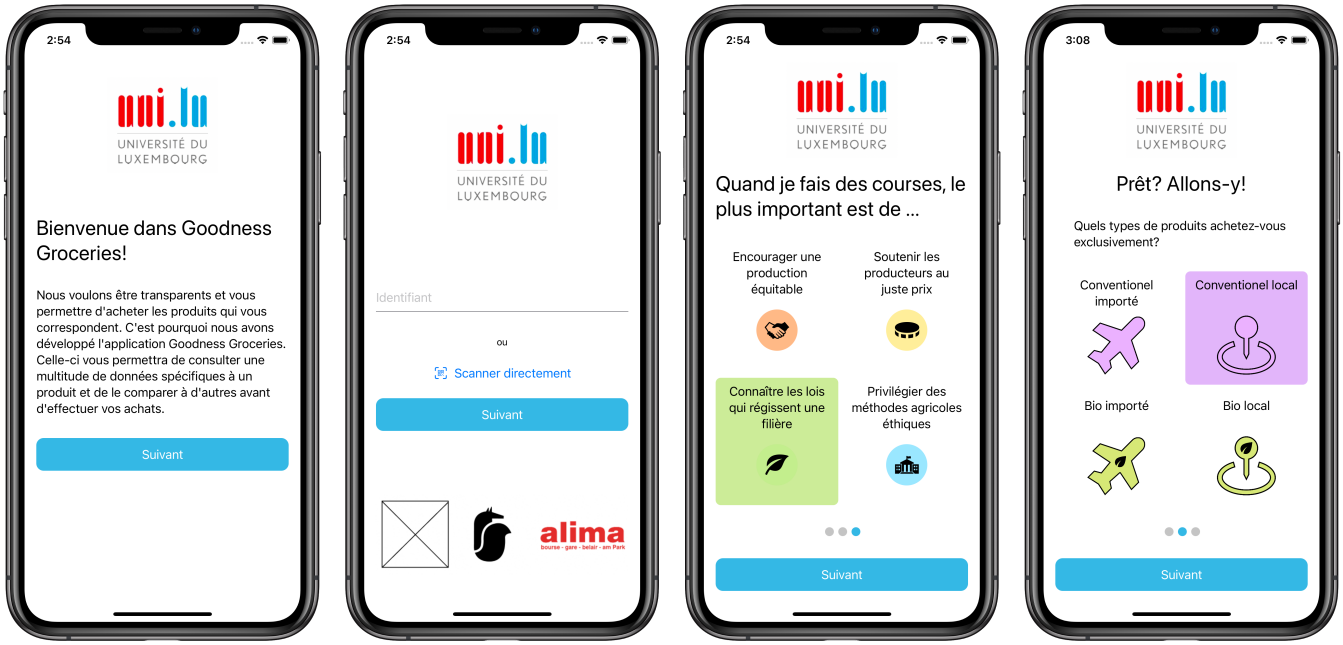


Figure 2: Goodness Groceries user interface design (welcome pages; not complete)

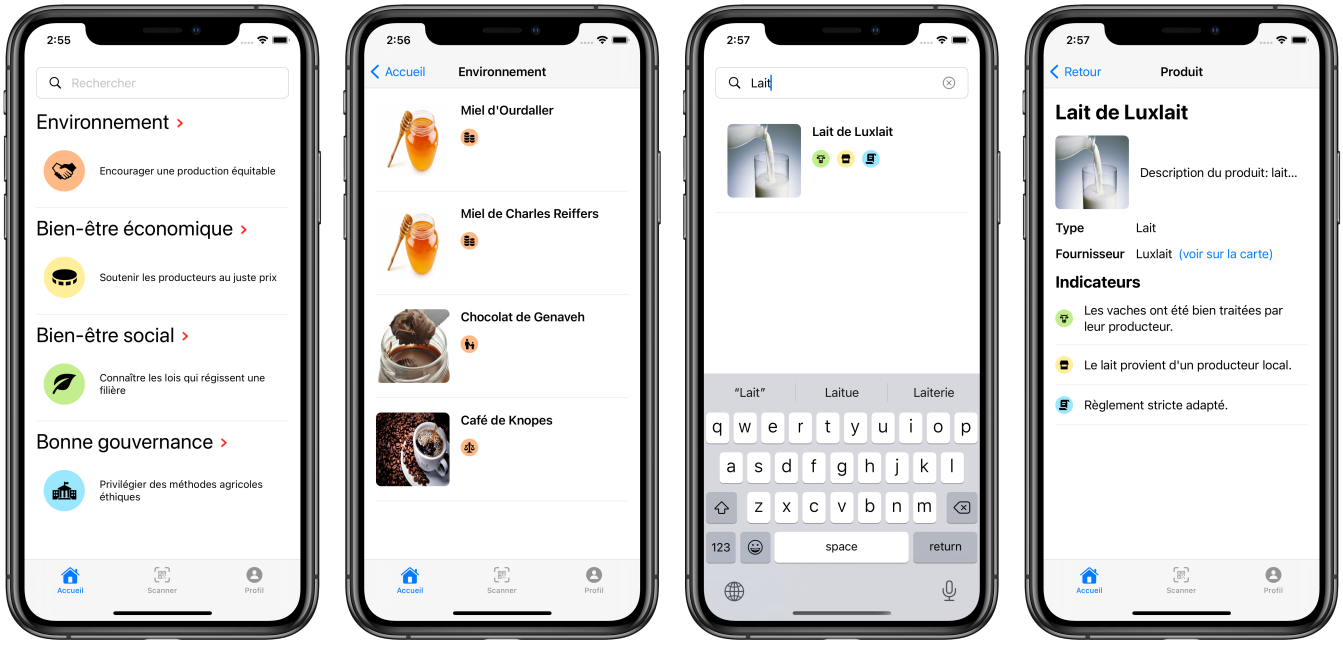


Figure 3: Goodness Groceries user interface design (categories and products)

For the welcome pages the same logic was applied. A new ViewModel called *WelcomeViewModel* was created which also holds an empty list. This list contains the product categories (which are 4) and the data is populated as soon as the class is initialised. Since there are only 4 categories, no need to load in a JSON file. The data is directly defined in the init()-function of the WelcomeViewModel class.

5.3.3. Handler classes. To handle the rest of the application logic, 3 global classes were created: **NotificationsManager**, **NetworkManager** and **UserSettings**. These classes contain fundamental functions for the app which did not belong into the ViewModel logic, therefore separate classes were created to handle these more specific tasks.

NotificationsManager
The NotificationManager is responsible to create and send notifications to the user. It only contains a single function which is sendNotification and has a parameter products which contains the EAN code of the products to be reviewed by the customer. Once the notification is created, it will append the custom list of products before sending. The app then contains a notification listener which gets triggered when the user opens the notification and handles the rest.

NetworkManager
The NetworkManager is obviously in charge of handling all the network-based tasks. Tasks such as consulting the current user status or requesting user access are managed inside this class. We differentiate between 2 different types of network requests: the ones that can be continued at a later point in time and the ones that need to be executed right away. If a network request can be continued later, such as monitoring the user's behaviour on the app or fetching the products for the user to review, and the user does not have an internet connection, the app will memorize and postpone the request. If however, the request is important, such as fetching the user status or requesting access to the app, and the user does not have an internet connection, the app will show an alert message to the user and warn him about the current situation.

UserSettings
The UserSettings class is responsible to handle all the functions and app-behaviours that are specific to the current user. The class holds several @Published properties which track the state of the certain elements, for instance, the customer ID, whether the application is currently loading something, the welcome page the user is currently in, etc. With the help of the NetworkManager class, it fetches and handles the user status (requested or valid) accordingly. The class also handles all the necessary operations to do a user access request to the web server as well as handling the welcome pages and always displaying the correct view to the user.

5.3.4. Views. The Views were created using several SwiftUI elements where **VStack** and **HStack** fall under the most used ones. VStack and HStack allow one to create accordingly vertical and horizontal stacks. These stacks themselves contain other SwiftUI elements and can also hold other stacks, one can nest the elements as desired thanks to the flexibility and freedom SwiftUI offers. To simplify the explanation process it is divided into **3** sub-categories: the **welcome pages**, **categories and products** and the **scanning tool**.

Welcome pages
The key element for the welcome pages is a **switch** statement. As already described, the UserSettings-class holds a certain property which memorizes the current step of the user in the welcome pages. Using a switch statement on this property, it allows us to display the correct view to the user. Since it is a *@Published* property, incrementing the step (by clicking on the blue buttons), will also update the switch statement and therefore display the new view to the user.

As we can see in Figure 2, every page is built on top of a VStack. Every page starts with an image and then some text or a text field. Per default, a VStack (and HStack) aligns its content in the centre of the screen. Putting the SwiftUI component **Spacer()** at the bottom of the VStack, we can push everything to the top of the screen because it will take as much space as it possibly can. This will create an empty area and squeeze all the other components together on top (illustrated in the first welcome page).

Each blue button is a view by itself, where the standard SwiftUI button was redefined. It was defined as a separate view because enhances reusability across other views. The idea here was to stretch the button to the maximum width of the screen and putting the label inside a centred HStack. Finally, the blue background and corner radius was added.

On the last two welcome pages, a Swift package called **QGrid** [9] was used. This package allows one to create easy collection views in SwiftUI. Both pages contain a 2x2 collection view which holds data loaded from the ViewModels. This data was embedded in a SwiftUI button, which turns the cells clickable and gives the user the option to select the desired category.

Categories and products
The categories and products list are also built on top of a VStack and contain two elements: a text field and a **ScrollView**. The ScrollView is an element which is self-explanatory; it lets the user scroll through its content. In this element we loop over each category provided by the CategoriesViewModel and displays a separate view, which represents a row. Each of these rows is a VStack which contains the title of the category and an HStack which contains the icon and description. Each icon has a colour which corresponds to the provided colour palette. These colours are the same ones used for the indicators of each category and can be seen in the first screenshot of the Figure 3.

- **Environnement** → #F5BA8E
- **Bien-être économique** → #FCEDA3
- **Bien-être social** → #CCEB97

- **Bonne gouvernance** → #AAE4FB

Using a **NavigationLink**, one can push a new view when clicking on an element. This allows us to push the view with the list of all the corresponding products whenever we click on a category.

The list of products is similar to the categories: looping over all the products and displaying the product row view, which is structured the same way as the category row views. The difference is that instead of showing the description, we display all the indicators corresponding to the category we clicked on before. The images of the products are however not stored locally and must therefore be loaded from URLs. This is done using the Swift package **SDWebImageSwiftUI** [10]. By clicking on a specific product, a new view is pushed containing all the available information such as provider, type, description and the list of indicators.

The search functionality which can be seen on the third screenshot of Figure 3, is very simple. Whenever the user types something in the text field, the ScrollView content is changed to a list of products corresponding to the search.

Scanning tool

The scanning tool gives users the possibility to scan QR-codes and access product information much quicker. The Swift package that provides this mechanism is called **CarBode** [11]. By using the view modifier **edgesIgnoringSafeArea** on the top edge, the camera view occupies the entire screen except for the navigation bar on the bottom. The barcode scanner is constantly scanning (every 1 second) for QR-codes and when it finds a valid one, it also checks whether it contains a valid product code. If the string contained in the QR-code corresponds to a product, then a small box on the bottom side of the screen appears with some information about the product and the user gets a small vibration on their iPhone thanks to **haptic feedback**. By clicking on this box, the product view page, which can be seen on the fourth screenshot of Figure 3, is pushed using a NavigationLink. When the string found does not match with any product, this box is hidden. This behaviour can be seen on Figure 6.

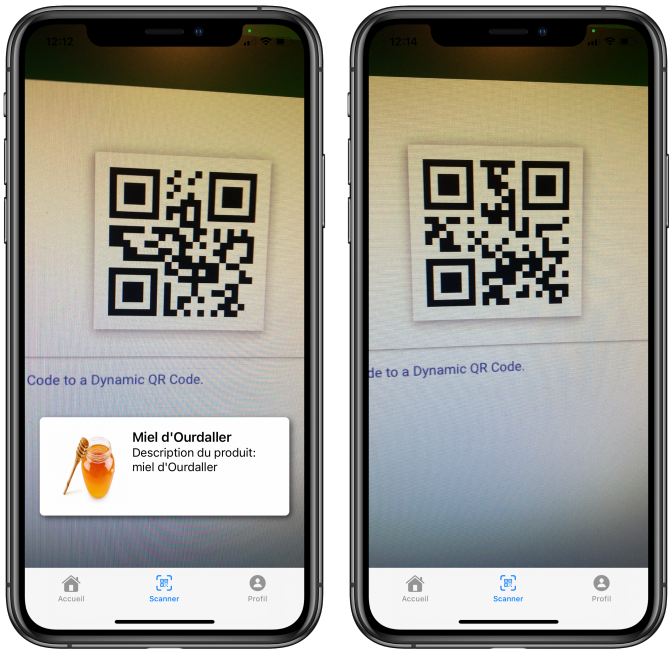


Figure 6: Goodness Groceries scanning tool design (left: valid QR-code; right: invalid QR-code)

5.4. Assessment

Each requirement was successfully achieved and implemented in the application:

- FR1: The app disposes of a product view page which gives the user information about each product. This page also contains a list of all the indicators corresponding to the product.
- FR2: Since the app disposes of a fully functional QR-code scanning tool, with which users can scan their desired product, this requirement was successfully met.
- FR3: The app manages to communicate with the web-server in order to create a user authentication request or fetch the user status. Thus, the requirement was also achieved.

Overall, the end result of the app is very good and can be considered a success. The 3 problems discovered during the evaluation process will be noted and addressed in the future in order to provide a better user experience to the users and create a stable end version for the 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 development of an eco-friendly mobile application for iOS and the execution of a selected UX evaluation technique on it. The scientific part focusses on the UX evaluation of the application, while the technical part is the development of the application itself. In the end, the app achieved every functional requirement set and provides all the expected functionalities. Thanks to the evaluation results, the weaknesses of the application were detected and this allows us to address them in more depth in the future.

After all, this project allowed me to learn a lot about iOS mobile app development while participating in such an interesting project. During the scientific research, I also learned a lot about the different evaluation techniques used for mobile applications and what is important for the end-users.

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] **Nikolaos Avouris, Georgios Fiotakis and Dimitrios Raptis:** On measuring usability of mobile applications https://hci.ece.upatras.gr/wp-content/uploads/publications/2008(c140)On%20measuring%20usability%20of%20mobile%20applications.pdf

[3] **Gunnar Peipman**: Model-View-ViewModel (MVVM) structure graph (modified by the student) https://gunnarpeipman.com/inotifypropertychanged/

[4] **Jennifer Derome**: What is User Experience?, Aug. 2015 https://www.usertesting.com/blog/what-is-user-experience

[5] ISO DIS 9241-210:2010. *Ergonomics of human system interaction* - Part 210: *Human-centred design for interactive systems* (formerly known as 13407). International Standardization Organization (ISO). Switzerland.

[6] Roto V., Vermeeren A., Väänänen-Vainio-Mattila K., Law E. (2011) User Experience Evaluation – Which Method to Choose?. In: Campos P., Graham N., Jorge J., Nunes N., Palanque P., Winckler M. (eds) Human-Computer Interaction – INTERACT 2011. INTERACT 2011. Lecture Notes in Computer Science, vol 6949. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-23768-3$_1$29

[7] **Shashi Prakash Tripathi and Tulika Narang:** Applying Model View View-Model and Layered Architecture for Mobile Applications https://www.researchgate.net/publication/328416556_Applying_Model_View_View-Model_and_Layered_Architecture_for_Mobile_Applications

[8] **Apple Developer:** SwiftUI https://developer.apple.com/xcode/swiftui/

[9] **QGrid:** a Swift package to create collection views, visited on December 10, 2020 https://github.com/Q-Mobile/QGrid

[10] **SDWebImageSwiftUI:** load images from URLs, visited on December 10, 2020 https://github.com/SDWebImage/SDWebImageSwiftUI

[11] **CarBode:** Barcode scanner for SwiftUI, visited on December 10, 2020 https://github.com/heart/CarBode-Barcode-Scanner-For-SwiftUI

7. Appendix

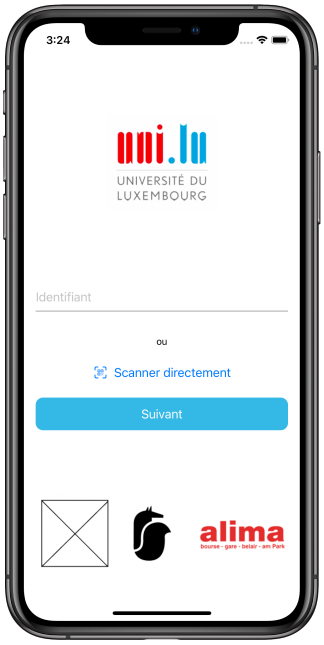
7.1. Application source code

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


3:24
UNIVERSITÉ DU LUXEMBOURG
Bienvenue dans Goodness Groceries!
"Goodness Groceries" Would Like to Send You Notifications
Notifications may include alerts, sounds, and icon badges. These can be configured in Settings.
Don't Allow
Allow
Suivant


3:24
UNIVERSITÉ DU LUXEMBOURG
Bienvenue dans Goodness Groceries!
Nous voulons être transparents et vous permettre d'acheter les produits qui vous correspondent. C'est pourquoi nous avons développé l'application Goodness Groceries. Celle-ci vous permettra de consulter une multitude de données spécifiques à un produit et de le comparer à d'autres avant d'effectuer vos achats.
Suivant


3:24
UNIVERSITÉ DU LUXEMBOURG
Identifiant
ou
Scanner directement
Suivant
alima


3:25
UNIVERSITÉ DU LUXEMBOURG
Quel consommateur êtes-vous?
Dites-nous ce qui compte pour vous afin de personnaliser votre expérience.
Suivant


3:27
UNIVERSITÉ DU LUXEMBOURG
Prêt? Allons-y!
Quels types de produits achetez-vous exclusivement?
Conventionel importé
Conventionel local
Bio importé
Bio local
Suivant

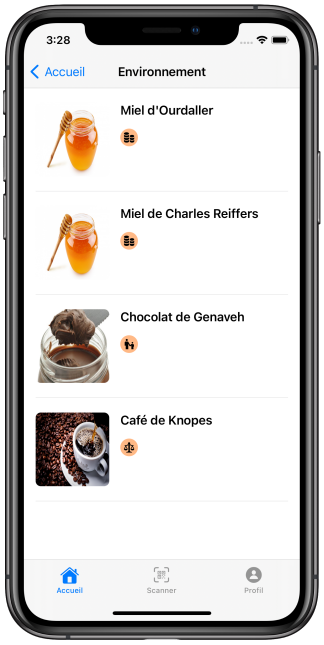

3:27
UNIVERSITÉ DU LUXEMBOURG
Quand je fais des courses, le plus important est de ...
Encourager une production équitable
Soutenir les producteurs au juste prix
Connaître les lois qui régissent une filière
Privilégier des méthodes agricoles éthiques
Suivant


3:27
Dernière étape
Votre demande sera envoyée et traité dès que vous remplissez le formulaire dispobile sur le lien si-dessous. Merci pour votre compréhension.
Ouvrir le formulaire


3:28
UNIVERSITÉ DU LUXEMBOURG
Scannez un produit
Munissez vous de votre Smartphone durant vos courses afin de scanner les QR codes présents sur une sélection de produits.
Suivant

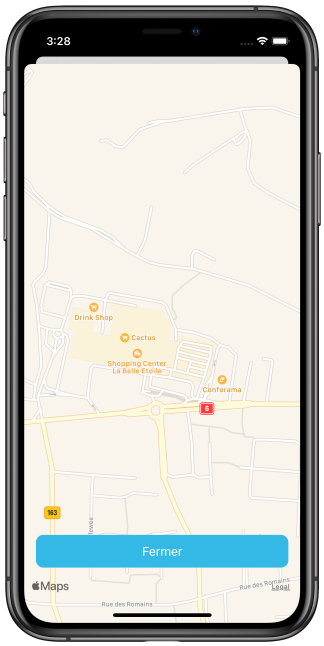

3:28
UNIVERSITÉ DU LUXEMBOURG
Consultez les indicateurs qui comptent pour vous
Pour des achats plus responsables, il ne vous reste plus qu'à profiter des informations disponibles sur chaque produit.
Suivant

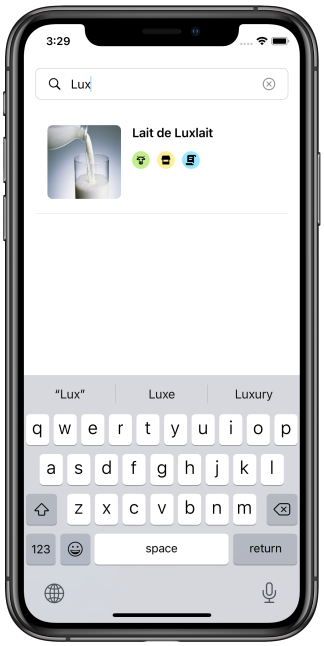

3:28
Rechercher
Environnement
Encourager une production équitable
Bien-être économique
Soutenir les producteurs au juste prix
Bien-être social
Connaître les lois qui régissent une filière
Bonne gouvernance
Privilégier des méthodes agricoles éthiques
Accueil
Scanner
Profil


3:28
Accueil
Environnement
Miel d'Ourdaller
Miel de Charles Reiffers
Chocolat de Genaveh
Café de Knopes
Accueil
Scanner
Profil

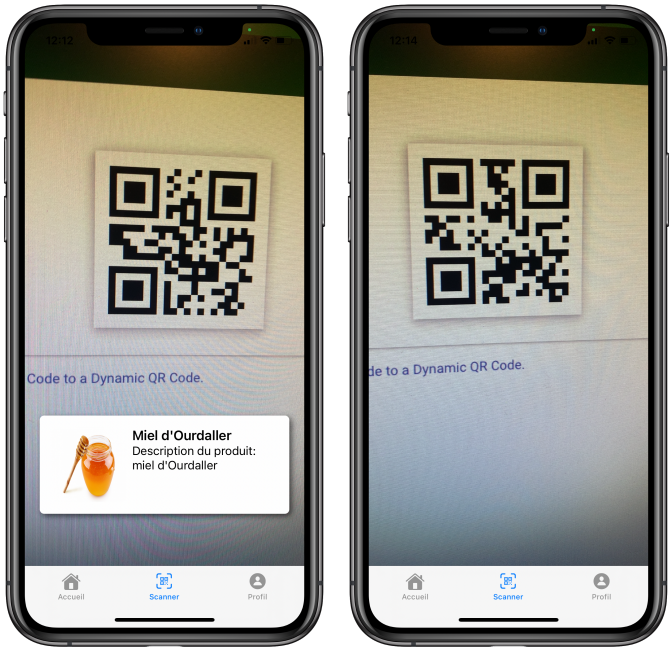

3:28
Environnement
Produit
Miel d'Ourdaller
Description du produit: miel d'Ourdaller
Type
Miel
Fournisseur
Ourdaller
(voir sur la carte)
Indicateurs
Voir plus
Les producteurs ont eu une rémunération adéquate.
Accueil
Scanner
Profil


3:28
Indicateurs
Les abeilles ont été bien traitées par leur producteur.
Les producteurs ont eu une rémunération adéquate.
Le miel provient d'un producteur local.


3:28
Fermer
Maps


3:29
Lux
Lait de Luxlait
"Lux"
Luxe
Luxury
space
return


3:31
Profil
Identifiant
1234567891234
Accueil
Scanner
Profil


12:12
Code to a Dynamic QR Code.
Miel d'Ourdaller
Description du produit:
miel d'Ourdaller
Accueil
Scanner
Profil
12:14
de to a Dynamic QR Code.
Accueil
Scanner
Profil