PRJ4 - Mobile Cloud Application

Product Price Alert

Group 8: Fotis, Karla, Paul, Daniyal & Nils

Abstract – This article reflects on project 4 of the informatics study program at Fontys Venlo**.** The task was to develop a cloud-based mobile application in a group. Specifically, a Product Price Alert app for iOS involving new technologies such as camera, GPS, REST API, and push notifications. During the project, the goal was to unveil new technology while using an agile approach. In the end, a market-ready system was developed despite the difficulties of the current unordinary situation caused by the Corona crisis.

1. INTRODUCTION

In general, the task was about mobile application and cloud development within a group of a few students. In the beginning the students were free to choose between Android or iOS app development. In our case we decided to develop an iOS application. Furthermore, groups could choose a case study (e.g. taxi, delivery or price alert) or come up with their own app idea. We picked the price alert case study: Potential customers that are interested in certain products, but do not want to buy the product, because of a high price, can get alerted when the price gets lowered or the product is offered with a discount. Additionally, the customer should be able to browse the available products and see product details. All use cases can be seen in *Figure 2 Use case diagram*. During the development of the application, a lot of new technologies had to be used.

2. OVERVIEW OF TECHNOLOGIES

The following technologies have been used:

Camera: The camera has been used to take a profile picture. Either during the registration process or afterwards to change it.

GPS/Map: The GPS sensor has been used to determine the location of customers to set their address and to show the location of sellers on a map.

Push Notification: To get informed about a price change, the customers receive push notifications through the use of APNs.

Loopback 4: Loopback is used to implement the REST API to communicate with the database, to handle authentication and business logic.

IBM Cloud: The Loopback backend is deployed to the IBM Cloud.

SwiftUI: As a UI framework SwiftUI has been used to simplify and speed up the frontend development.

PostgreSQL: To store entities data such as product and customer PostgreSQL database is utilized.

3. OVERALL ARCHITECTURE

As you can see in *Figure 1 Deployment diagram,* the main components are the iOS application, Loopback REST API, PostgreSQL database and the notification relay.

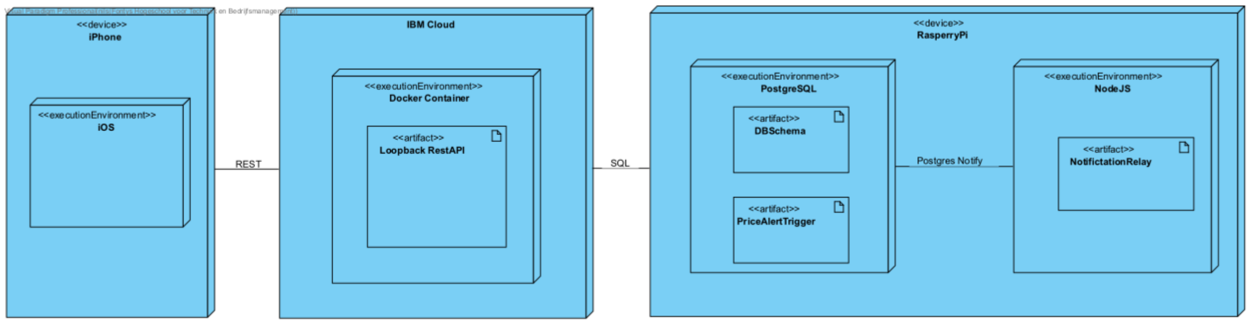


Figure 1 Deployment diagram

iOS Application: In the frontend, the application is running on an iOS device of the customer. It is providing functionalities such as searching for products, seeing product details and setting a price alert. Therefore, it is necessary to communicate with the backend, which is done via a REST API.

Loopback REST API: The backend is deployed in a docker container running in the IBM Cloud. To access data, it sends SQL commands to the database.

PostgreSQL database: It consists of the database schema and the price alert trigger code which is responsible for detecting price change and notify the notification relay via Postgres Notify. The database is running on a Raspberry Pi.

Notification relay: Also running on the Raspberry Pi is a NodeJS application which sends a push notification to the corresponding customer using the Apple Push Notification service (APNs) when it is notified about a price change by the price alert trigger in the database.

4. DISCUSSION

During the project, a few decisions had to be made.

Why did we choose SwiftUI over UIKit?

SwiftUI is a modern declarative UI framework and has got (in our opinion) a few advantages. The declarative coding style leads to more readable and less complicated code. The structure of the UI is described in the swift code instead of in XML, which is hard to read and modify. In general, it speeds up development and leads to fewer bugs.

A disadvantage if SwiftUI is that it only supports iOS version 13 or newer and that it is not very mature and therefore lacking in more advanced areas and in the documentation.

Why did we make use of a Raspberry Pi?

First, we used the cloud service ElephantSQL, but we encountered a problem with the storage limit because we wanted to store images as base64 in the database. To solve this issue, we decided to host our own PostgreSQL database on a Raspberry Pi.

Why did we follow the Scrumban approach?

Scrumban combines Scrum and Kanban components. From the Scrum we took elements such as working in sprints and prioritizing use cases. The Kanban framework added processes like visualizing the workflow and the pull system. The pull system provides a continuous workflow since use cases are pulled into the *In Progress* (as can be seen in *Figure 3 Trello Board*) column in Trello when the team is ready to do the specific task. Furthermore, in kanban team member roles are not clearly defined. Instead of defining, e.g. separate backend and frontend development teams, we split work based on our use cases. Every team member was able to work independently on his or her feature and corresponding branch. The main advantage of scrumban for us was the time-saving aspect. E.g. plans are only made in case there is a demand for the teams.

Why did we use the GitHub Actions and Git feature branch workflow?

GitHub Actions allowed us to continuously integrate and monitor each new functionality added to the project to make sure all the tests pass, and the project builds successfully. Following the Git feature branch approach, a separate branch is created for each feature and only merged to the master branch when it is finished (by our definition of done). This results in fewer merge conflicts and provides the opportunity to decide when to combine features. Especially beneficial when working remotely. Furthermore, a pull request can be used to communicate and review changes. At the beginning of our project, we encountered merging issues because we did not agree on a project structure before starting the development. First, we had to get used to new technologies.

What is the final state of our project?

We finalized all must-haves including the core functionalities such as sending price alters or browsing products and some optional features like editing profile and showing seller details, including a map view. Sellers can interact with the system using the Rest API using seller-specific endpoints to change, for instance, the price of a product or to create new ones.

5. INDIVIDUAL PARTS

Fotis

At the beginning of the project I helped in creating the user groups, user stories and setup the initial database schema which was later changed in order to cover the increasing needs of the project.

My main tasks for this project had to do with the management of a customer's information. This meant that I worked in everything it had to do with the customer storing, updating or looking up information about themselves. The use cases associated with those where Customer and Address Registration, Login Customer, Logout Customer, View Customer Profile, Edit Customer Profile, View Customer MapView and the ability of the customer to save/edit a picture from their storage or by using their camera.

In order to handle all the POST and PUT requests I created the Postman class which can handle any of given entity in terms of the aforementioned request and the OneEntityFetcher which handles the GET requests for any given object. Both are generic and they were used not only to fetch, post or update data but also to handle the customer login/registration request.

At the same time in order to ensure that the user will insert valid inputs to our database and avoid duplicated data or an undesired email format, I created some regex checkers which are used to validate the customers inputs. The regex validation was created as a String extension and were used in editing and creating a customer.

Moreover, user coordinates are vital for our projects as they are used to pinpoint the user's location on the map. For this reason, in order to be able to secure the correct coordinates we did not relied only on Mapkit but we implemented a way to get the coordinates via the inserted address info (as a string) using the LocationManager. Through this we achieve not only credibility but also better user experience due to the fact that when the user updates their information, they don't have to check their location in the map every single time as this is taken care by the system.

One last noteworthy feature which needs to be mentioned is the Picture resize functions which were introduced as extensions to the UImage. Since we store our image in the DB as a string using high quality pictures created a delay in posting/updating our system. Through this picture resize the given picture is shrunken to 10% of its original size making it faster for the application to exchange info with the cloud.

During the project I realized the value of pair programming and complementing each other's knowledge. All of us helped each other along the project and I am really proud that we managed to finish most of the core parts of the application in the two-week project time that were given to us by the University. I worked together with Paul in the back-end part of the user registration, login or update, with Nils in the coordinate retriever or with Daniyal in the GitHub merge problems that we had etc.

The groups mood, dynamics and cooperation level were good as we worked together in many common parts of the project and the atmosphere in every meeting was pleasant and honest.

Karla

At the beginning of the project I have created a wireframe of our project se we can all have the same understanding of how we would like to develop our application. Then we worked all together on developing user stories and user groups. Also, I worked on database schema.

First use case I was working on was displaying product details. From the list of product user can enter specific product and see details such as price, image, seller name and description. List of products is loaded by REST API and specific product is passed with Navigation view. Seller details are fetched using OneEntityFetcher in order to display seller name. Also, every time a user views a product, number of views is updated using PATCH request. From the product details, user can set the price alert or buy product. Since payment was not in the scope of the project, the user gets a pop-up notifying that the product is bought.

After that, I have worked on creating a Menu, a tab bar which would be the main structure for navigating between views.

Then I have worked on Filtering products. The idea was to be able to filter products by category and price range. In GenericFetchList I have added option 'inq' which is an operator that checks if the value of the property matches any of the given values, which in this case were categories that user selected from the list of all categories fetched from database. Also, I have added option 'between' which filtered results to get results between prices entered by user. The result was list of products filtered products by category and/or price range.

Furthermore, I have worked on the poster and created a video.

Difficulties that I have encountered with were mostly SwiftUI issues, but with the help of group members we would find the solution. During the project I have learned how to use various new technologies that I have never worked before with. I have learned how to choose methods and technologies best suited for our needs. It was interesting and challenging project in difficult circumstances, but since our group had great communication and good overall dynamics, everything worked out well.

Daniyal

From the previous experiences in other projects, I was aware of the importance of proper structure and guidelines in contribution to the project. Thus, at the beginning of this project, I set up the Git branching together with the GitHub Actions to achieve continuous integration and increase the overall development productivity. Later on, when merging the first few branches to the master branch, we encountered some merging conflicts, which was related to how XCode creates and maintain the project files and directories.

I was also involved in setting up the LoopBack framework, creating the database schema and editing this article with the domain model diagram.

The first use case I started working on was the searching functionality of the products using the remote endpoints of the REST API without the need for fetching all the products at once in a local variable in the application.

The second use case was handling the Apple Push Notifications (APN) in different states of the application (e.g., foreground, background, etc.) as well as displaying the alert received from the remote server as a modal sheet when a user taps on the notification alert.

Due to the lack of documentation in the SwiftUI package and available online resources, the main challenge was the APN setting it up according to our SwiftUI project to display the received alert. Overall, using the SwiftUI framework as a declarative paradigm language was an enjoyable experience.

In the end, I believe since all of our group members were fully invested in our project, it made every Scrum or pair programming sessions fruitful as well as it created a delightful environment to share our opinions and experiences openly and frequently.

Paul

First, I worked at the use case view popular products. It is about displaying a list of products loaded from the database via the REST API. The products should be ordered by their popularity which is measured by

the amount of times they have been viewed by customers.
For this I implemented a generic service I call GenericFetchList. It can retrieve a list of any entity from the backend. This way it could be reused for other use cases. It also supports additional options such as the ordering by a specified attribute. This was used to order by the amount of views.
Additionally, infinite scrolling is supported by GenericFetchList. This means that the list of entitles is continuously requested from the backend in chunks to improve performance. The next chunk of entities is loaded when the user scrolls to the end of the list.

Another use case I worked on is create price alert. In this one the customer selects a product he wants to get notified about when its price drops below a given price.
For this authentication was needed so that only registered customers can create alerts. For this I had to learn how to work with a highly extendable web framework like loopback which makes heavy use of concepts like dependency injection. This was challenging at first but turned out to be worth the effort since because of it, it was very easy to secure additional endpoints and to add roles like the one for sellers that is used to secure the API endpoints that can be used by sellers to create, delete and update the prices of products.

I also worked on the backend code that detects the price change of products, for which alerts are set, and then sends push notifications to the corresponding customers.
The detection is done in a trigger inside the database which then notifies an external application which sends the request to the Apple Notification Service to deliver a notification to the customers device. During that I temporarily encountered problems regarding the authentication with the APNs and the needed certificates and entitlements.

Furthermore, I worked on asynchronous and synchronous unit tests and on this article and various diagrams.

During the project work I also learned the value of pair programming to combine expert knowledge of multiple areas. An example for this is the implementation of the backend logic for the registration which required knowledge about the registration process that Fotis had and my knowledge about the backend authentication.

Additionally, I learned how to choose technologies and methods by considering advantages and disadvantages for a specific situation e.g. Swift UI for fast development of simple UIs and feature branches for conflict free remote teamwork.

Nils

At the beginning of our project I was involved in design and creating the database. To have a usable database I was I used the online tool *mockaroo.com*. With mockaroo you are able to generate realistic test data to later on fetch those and to display in our app.

Furthermore, I worked for instance on the use cases *view sellers* and *view seller details*. *View sellers* is about displaying all sellers from our database by using the REST API. For this I used our generic fetch service called GenericFetchList to get all the sellers. To get a detailed view of a specific seller I implemented the *view seller details* use case. For each seller in the seller list I linked a second view to retrieve the details in a separate view. This view includes a map view using Apples MapKit framework to embed a map directly into this view. In our database we store for each seller information about longitude and latitude. This data is used to place the seller on the map view. In addition, this view makes use of annotation (including the sellers' address) to highlight the point of interest on the map.

Besides I was involved in a few pair programming sessions. We worked for instance on the use cases *send price alert notification* and *create a new price alert*. Paul has already described them in detail in his section above.

Furthermore, I worked on asynchronous unit tests, on the poster, this article a, a few diagrams (e.g. domain model and deployment diagram) and I created a separate Trello board for our retrospective meetings.

During the project I realised how important good communication (including knowledge transfer during pair programming) is – especially when you think of the remote aspects and special circumstances of this project. This also explains our decision to use the Scrumban approach. Moreover, I learned when and why to use which technology and concepts, e.g. our decision to use SwiftUI over UIKit and the use of Git feature branching.

6. CONCLUSION

Overall the project was a success. We implemented the core functionalities, as mentioned in the previous section. Due to the scrumban approach, we had time to focus on development instead of planning and make well-considered discussions about technologies and methods. During the unusual situation due to Corona, we could gather experience in working in a remote project environment. This was a success because of the methods specifically tailored for the project and for working remotely like the Continuous Integration with GitHub Actions and Git feature branch workflow and the way of distributing work.

Appendix

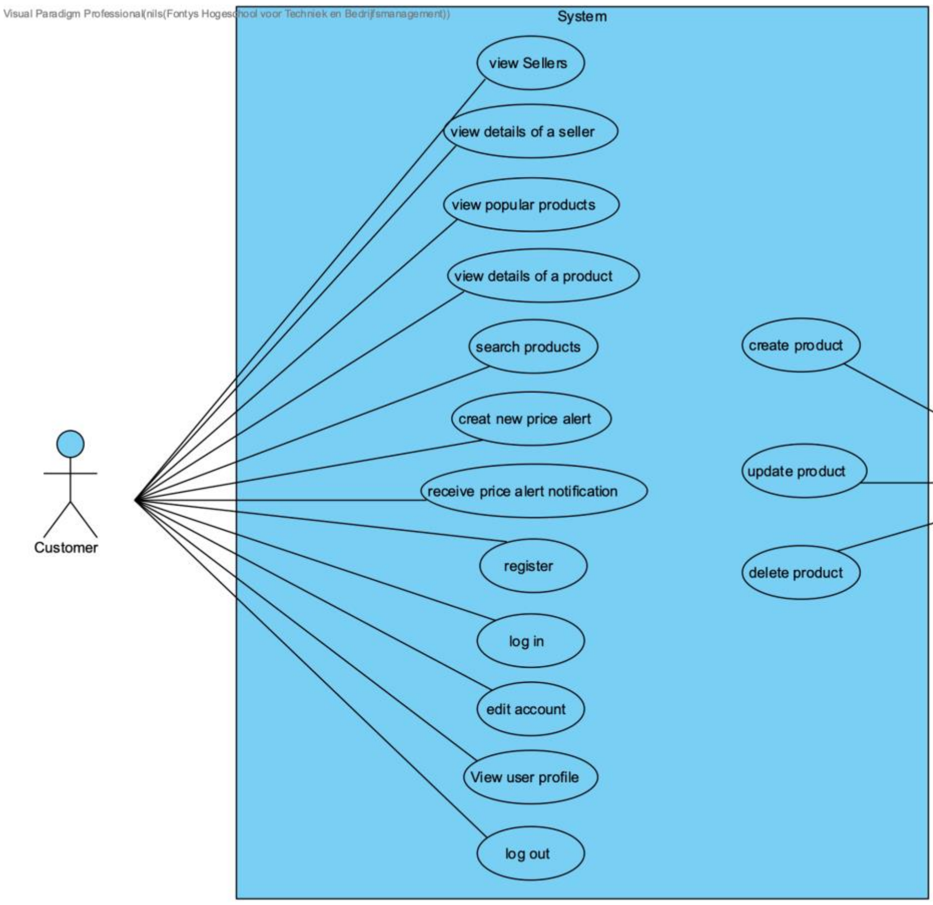
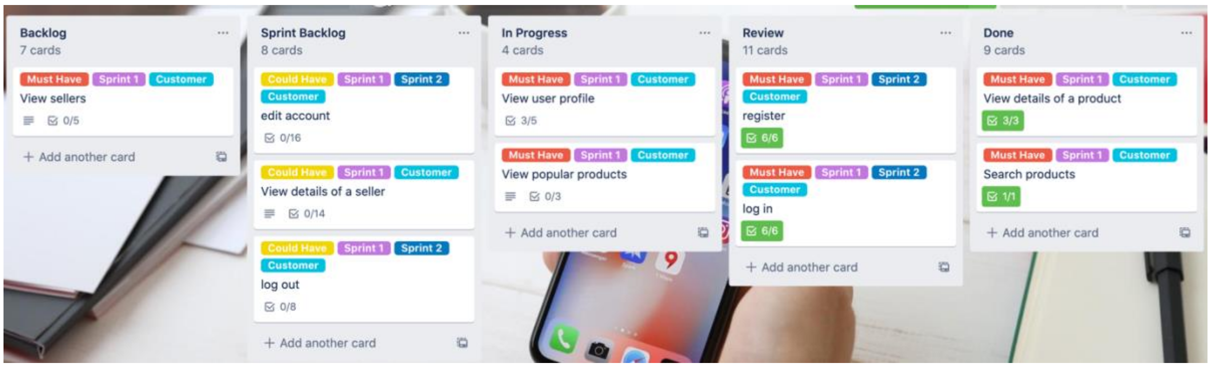


Figure 2 Use case diagram



Backlog
7 cards

Must Have | Sprint 1 | Customer
View sellers
0/5

+ Add another card

Sprint Backlog
8 cards

Could Have | Sprint 1 | Sprint 2 | Customer
edit account
0/16

Could Have | Sprint 1 | Customer
View details of a seller
0/14

Could Have | Sprint 1 | Sprint 2 | Customer
log out
0/8

+ Add another card

In Progress
4 cards

Must Have | Sprint 1 | Customer
View user profile
3/5

Must Have | Sprint 1 | Customer
View popular products
0/3

+ Add another card

Review
11 cards

Must Have | Sprint 1 | Sprint 2 | Customer
register
6/6

Must Have | Sprint 1 | Sprint 2 | Customer
log in
6/6

+ Add another card

Done
9 cards

Must Have | Sprint 1 | Customer
View details of a product
3/3

Must Have | Sprint 1 | Customer
Search products
1/1

+ Add another card

Figure 3 Trello Board