Mobile application client for a cloud based software system

Joshua Schnabel

Supervisor: Dr. rer. nat. Robert Heinrich

Abstract. This work deals with the design of a hybrid app. The goal is to create a second sales channel next to the existing PickUpShop in the CoCoME system. In order to implement this, first the required Use-Cases were modeled, second the needed pages were designed. Afterwards, a number of frameworks, such as Cordova and OnsenUI, were used to design the multi OS app. By programming a prototype it was proven that such an app would work.

1 Introduction

CoCoME stands for Common Component Modeling Example and is a modular software experiment. The software simulates a trading system with enterprises and stores. Each store has an inventory and a cash desk line. Each cash desk has some components like a bar code scanner or a cash box.

This paper documents the development process of a sales app for the CoCoME Systems. This app is designed to attract more customers. In today's world, it is very important to gain customer's loyalty. High customer loyalty generates higher sales.

But an app also carries risks. This can lead to a security problem for the whole system. Also the performance of the whole application can be affected if many customers order via the app.

2 Requirements

In this chapter, the use cases are listed.

2.1 The app's use cases

UC 14 - Process App Sale

Brief Description A Customer selects the product items s/he wants to buy and the payment by credit card is performed.

Involved Actors AppCustomer, Bank

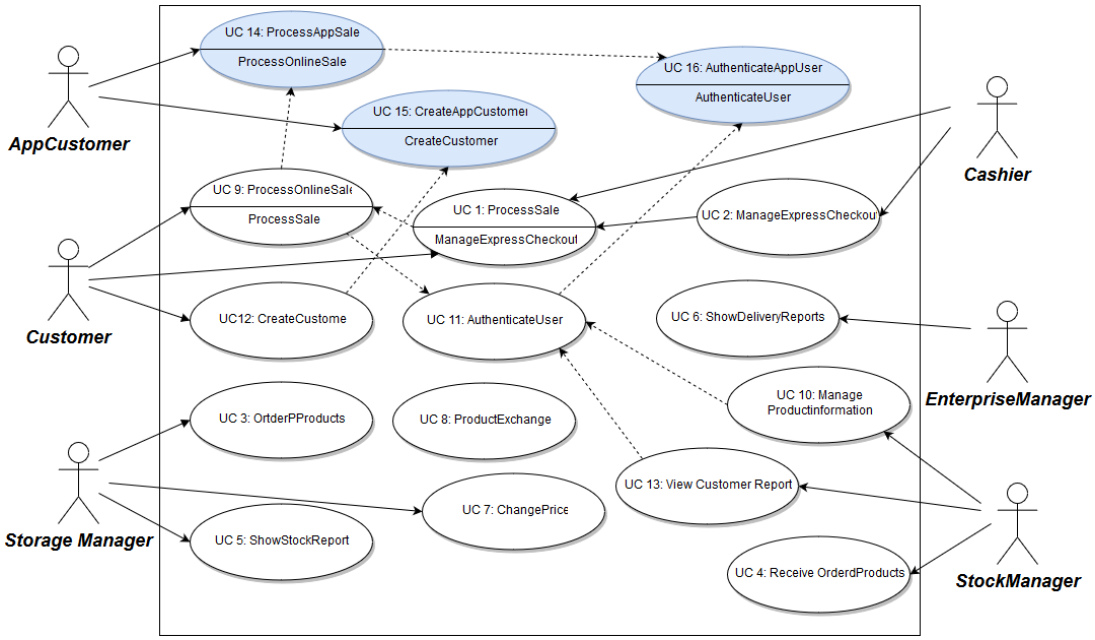

Fig. 1. Use Cases

Precondition The App is ready to process a new sale and the Customer already has an account registered in the System.

Trigger The Customer opens the app and wants to buy product items.

Postcondition The Customer has paid and the sale is registered in the inventory.

Standard Process

1. The AppCustomer searches products provided by the App.
2. The AppCustomer can see details for each product on a separate site.
3. The AppCustomer adds the product items s/he wants to purchase to the Shopping Cart by clicking the Add to Cart button. Step 1-3 is repeated until all items are added to the cart.
4. The System presents the Customer with the item names, prices and the running total.
5. Denoting the end of adding items, the Customer presses the Proceed to checkout button.
6. The AppCustomer selects the Store where s/he wants to pick up her/his purchased product items.

7. The AppCustomer is presented with a login form and is required to complete the „Authenticate user “use case.
8. In order to initiate card payment, the Customer selects a credit card used for the purchase.
9. The AppCustomer enters her/his PIN in the designated field presented by the System.
10. The System presents the Customer with an overview of the purchase, the AppCustomer confirms the purchase and waits for validation. Step 9 is repeated until the validation is sucessful or the Customer decides to cancel the purchase.
11. Completed sales are logged by the System and sale information are sent to the Inventory in order to update the stock.
12. A success message is presented to the AppCustomer and the product items are being prepared to be picked up by the customer.
13. The AppCustomer closes the app.

Alternative or Exceptional Processes

- In step 8: No Card available
  1. In order to add a new credit card the Customer clicks the Add Card button.
  2. The Customer enters the card number of the new credit card and saves the card.
- In step 10: Card validation fails
  1. The Customer tries again and again.
  2. Otherwise, the Customer can decide to cancel the purchase.

UC 15 - CreateAppCustomer

Brief Description The app provides the opportunity to create a new Customer account.

Involved Actors AppCustomer

Precondition The Customer does not have a Customer account yet and the app is started.

Trigger A new AppCustomer wants to create an account.

Postcondition The User is authenticated.

Standard Process

1. The app presents the Customer a 3 step wizard with forms to fill out, requesting all necessary information to create a new Customer account.
   (a) Form for name, email and password
   (b) Form for address

   (c) Summery of the informations and a submit button
2. The Customer fills out the forms, checks the information and submits the information.
3. The app checks the provided information and creates a new Customer account in the Inventory.

Alternative or Exceptional Processes

- In step 3: Provided information is incorrect or not valid. The Customer is notified of the problem and enters the information again until it passes the check.

UC 16 - AuthenticateAppUser

Brief Description The app provides the opportunity to authenticate a User.

Involved Actors AppCustomer

Precondition The app is started.

Trigger A User wants to authenticate himself.

Postcondition The User is authenticated.

Standard Process

1. The AppCustomer is presented with a login form and enters her/his email and password.
2. The app checks the provided credentials and logs the User in.

Alternative or Exceptional Processes

- In step 2: Wrong credentials
  1. The User is presented with an error message.
  2. The User may try again until the authentication succeeds.

2.2 Quality-of-service requirements

2.2.1 Required criteria

- The app should response in under 1 second.
- Date from the backend should return in under 3 seconds.
- The app should not be vulnerable to attackers.
- The user data should be secured from illegal access.

2.2.2 Delimitation criteria

- The application should support Android API-Level 14 and higher.
- The application should support OS X version 10.9 and higher.

2.3 Implementation requirements

2.3.1 Required criteria

- The application should be implemented with a multi OS framework.
- The application should use the CoCoMe platform.
- The application should be runnable on IOS and Android.

2.3.2 Desired criteria

- The application should be tested with automatic Unit and UI tests.

3 Technology-Stack

This chapter describes the technology stack.

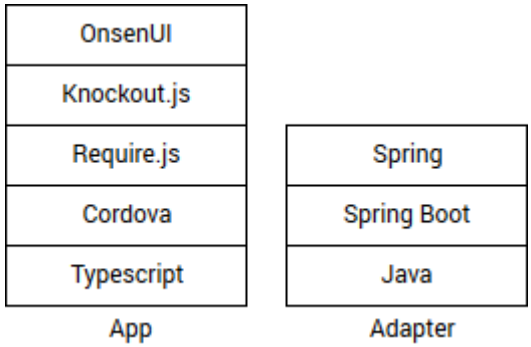

Fig. 2. Technology-Stack

3.1 Cordova

Apache Cordova is an open-source framework for mobile development. It allows to use standard web technologies like HTML5, CSS3, and JavaScript for cross-platform app development. Applications execute within wrappers targeted to each platform and rely on standards-compliant API bindings to access each device's capabilities such as sensors, data, network status, etc [Foun15].

3.2 Onsen UI

Onsen UI provides UI framework and tools for creating fast and beautiful HTML5 hybrid mobile apps based on PhoneGap / Cordova. Having common core with no framework dependencies, app development with Onsen UI is easy with any of the ever-changing JavaScript frameworks [xOns16].

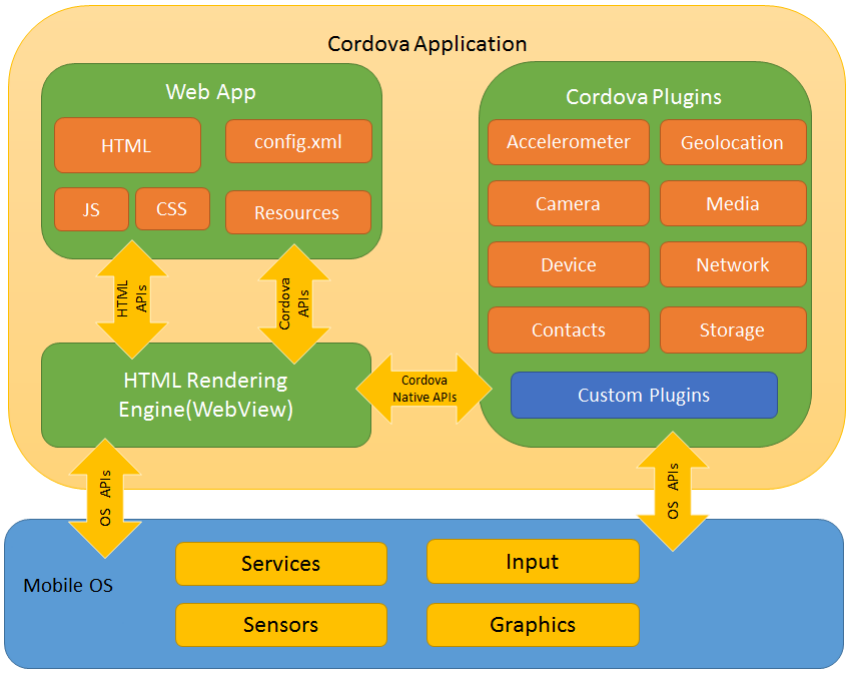

Fig. 3. Cordova app architecture [Foun15]

3.3 Knockout.js

Knockout is a JavaScript library for rich, responsive display and editor user interfaces with a clean underlying data model. It provides a simple way to connect a HTML frontend with a javascript model. The binding can be done by HTML tags. [Knoc17]

3.4 Require.js

RequireJS is a JavaScript file and module loader. It is designed to load modules at runtime. Modules may have detachments that can be resolved by require.js. [Chun17]

3.5 Typescript

Typescript is a free language developed and maintained by Microsoft. Typescript is an extension of javascript and adds optional static typing and class-based object-oriented programming to the language. The Typescript compiler transcompiles the code to JavaScript. This JavaScript code is runnable in all browsers. [Micr17]

3.6 SpringBoot

„Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration. “ [Soft17]

4 Architectural-Decisions

The app will be designed with OnsenUI to provide a multi OS UI. Furthermore the Cordova framework will be used to provide a multi OS compatible backend. This app backend will be programmed in Typescript. This combination provides the best combination of performance, compatibility and effectiveness.

This chapter describes the app's design from the pages to the classes.

4.1 Pages

This chapter deals with the use cases and the resulting pages.

4.1.1 UC 14: ProcessAppSale This use case calls for a page to search for products. Also a page to see product details is required. The customer can add products to the card. So a page to see a the products in the cart is required. The user must be able to select a store, this should be done on the main page. The last page of this use case is the checkout page. On this page the user can see his products and s/he can buy them.

4.1.2 UC 15: CreatAppCustomer The user needs a page to create himself an account. This should be done on a separate page.

4.1.3 UC 16: AuthenticateAppCustomer To authenticate a user a login page should be provided.

4.1.4 Graph of pages Figure 5 shows the pages of the app and their relationships. This graph provides a reference for later tests.

4.1.5 Mockups After defining the pages, this chapter shows the mockups of the described pages. The figures 5 to 11 are showing the mockups of each page.

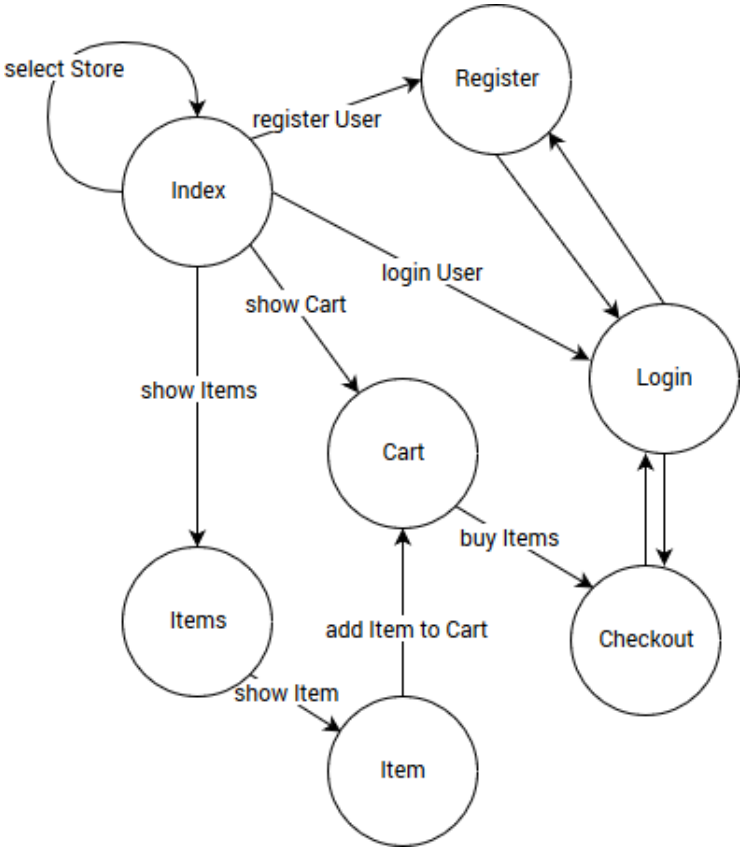

Fig. 4. Pages and possible navigation ways

4.2 Classes

Figure 12 shows the app's basic classes and their relationships.

The Navigator class is the primary class of the app. These class manages the pages. Each page consists of two components: the page itself and its pagestate. The pagestate provides the page's current status and can be used to transfer this status.

The page can use components. These components are abstract descriptions of UI elements. These abstract descriptions are connected to the HTML throw knockout.js. Knockout.js updates the HTML elements if the values of the abstract descriptions are changed. The values of the abstract descriptions are also updated if the HTML is changed.

The page can also use the services. These services are provided by the ServiceHolder. Services offer access to the informations of the CoCoMe system.

Because of the incompatibility of javascript with SOAP and WSDL an adapter is required between the app and the CoCoMe system. This adapter will be pro-

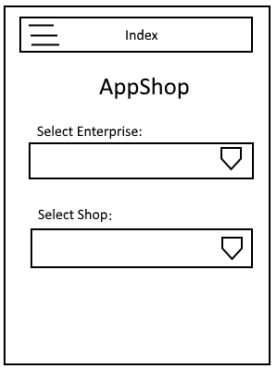

Fig. 5. The index page mockup

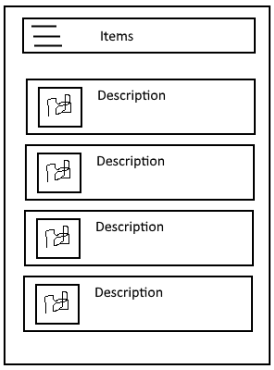

Fig. 6. The items page mockup

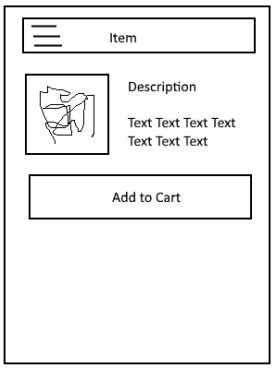

Fig. 7. The item page mockup

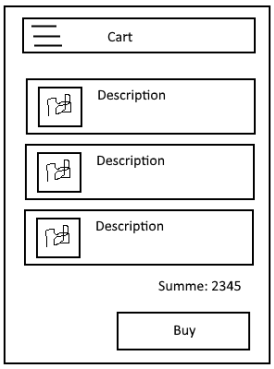

Fig. 8. The cart page mockup

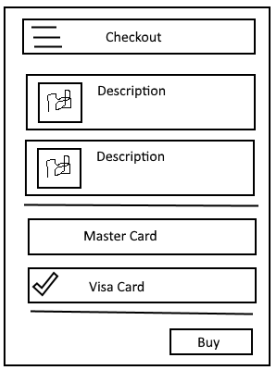

Fig. 9. The checkout page mockup

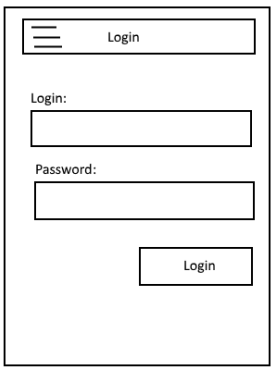

Fig. 10. The login page mockup

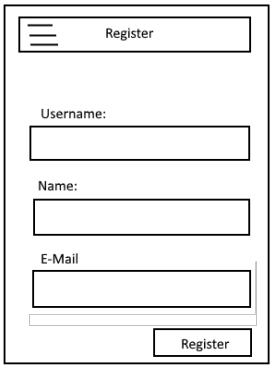

Fig. 11. The register page mockup

grammed in Java with the SpringBoot framework. It converts the REST requests from the app to SOAP-Requests.

4.2.1 Sequence diagram Figure 13 shows the way from opening a page to calling the CoCoMe system. A page can be opened by an user event. For this, the component must address the navigator. The navigator creates a pagestate object and passes the object to the page. Then the page creates the HTML page and presents this page to the user.

To fill the page with information, the page can use the provided services. These services are managed by the service holder. A service calls a REST-Service of the adapter. This adapter calls the WSDL services of the CoCoMe service.

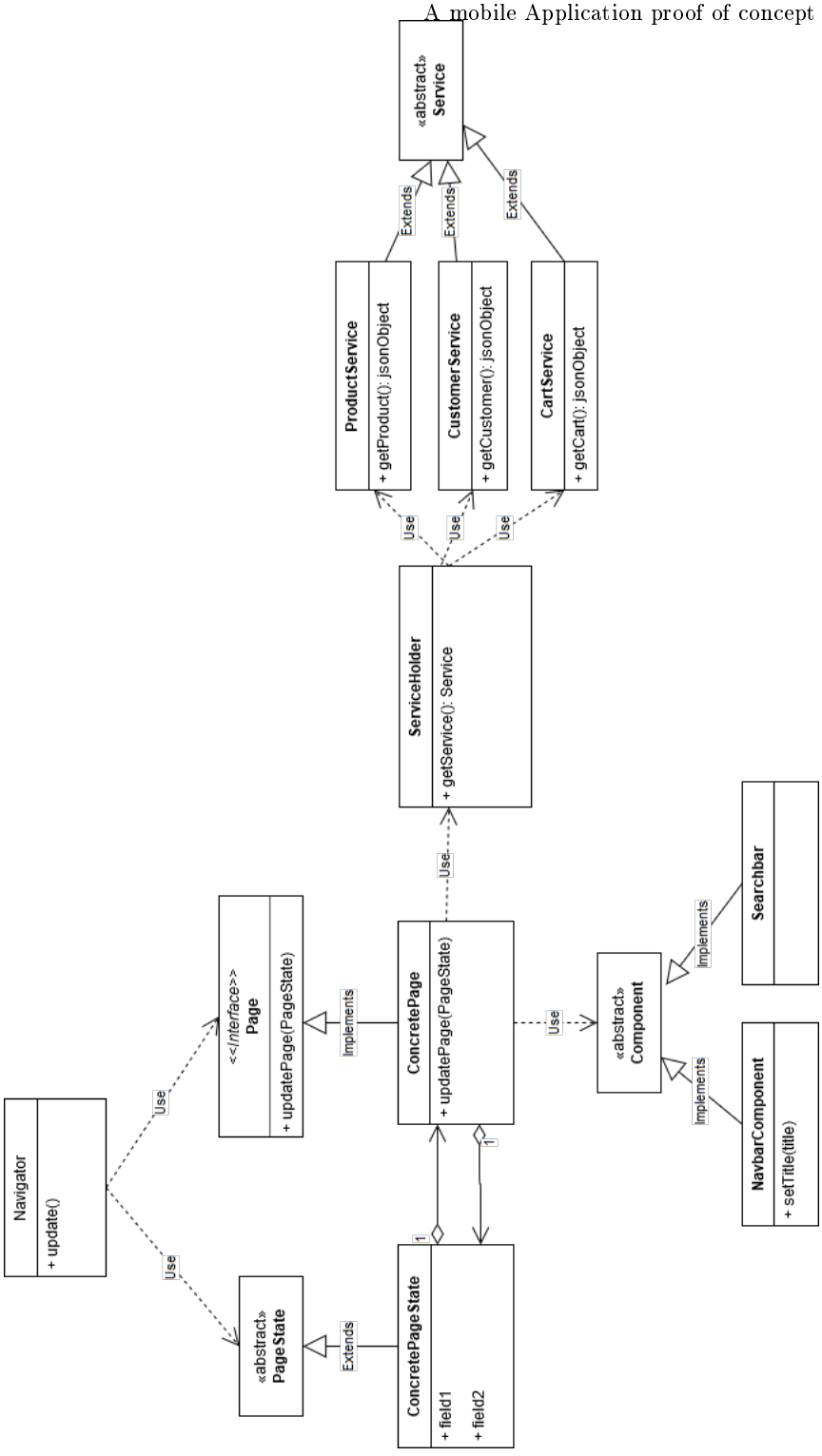

Fig. 12. Primary classes of the app

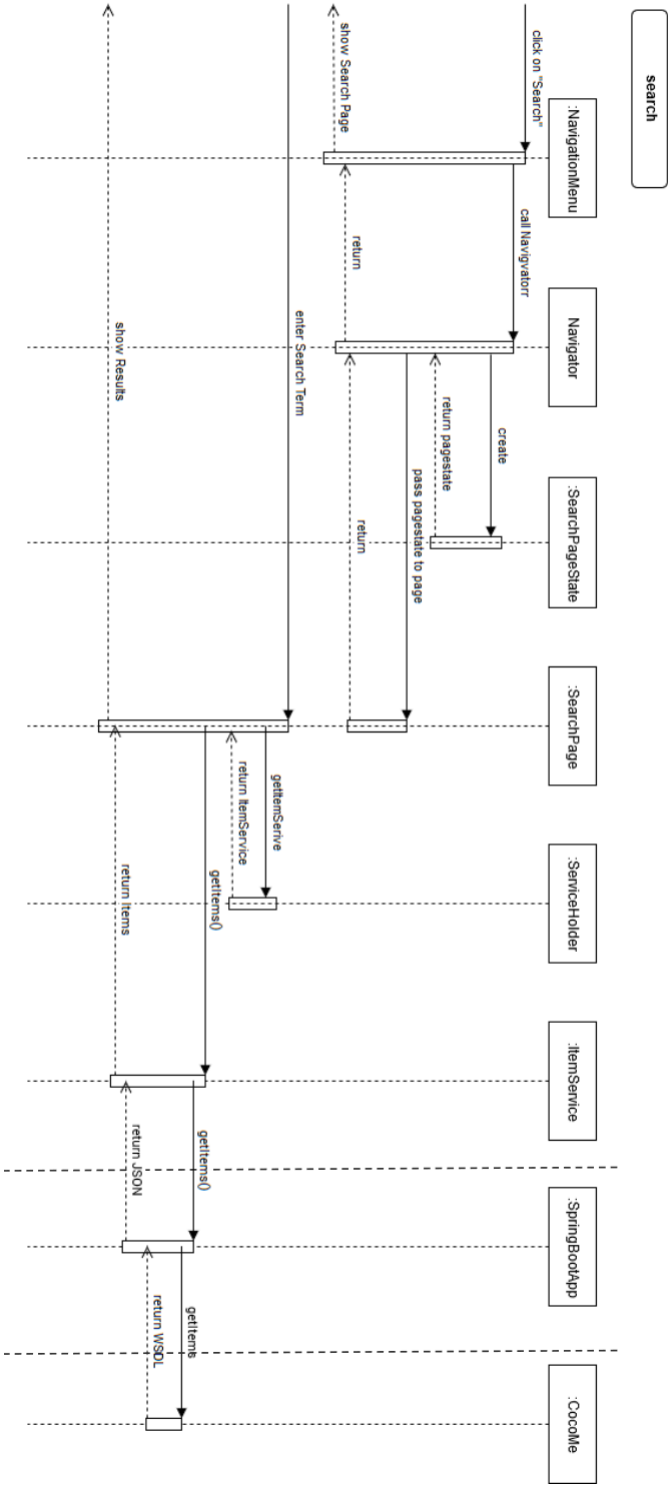

Fig. 13. Sequence diagram of the communication to the backend

5 Test

The tests will be separated in two tasks: the app tests and the adapter tests.

5.1 App

5.1.1 Integrationtest Figure 14 shows the app's current test status. The graph shows the working and the not working paths. This test was manually tested by the developer.

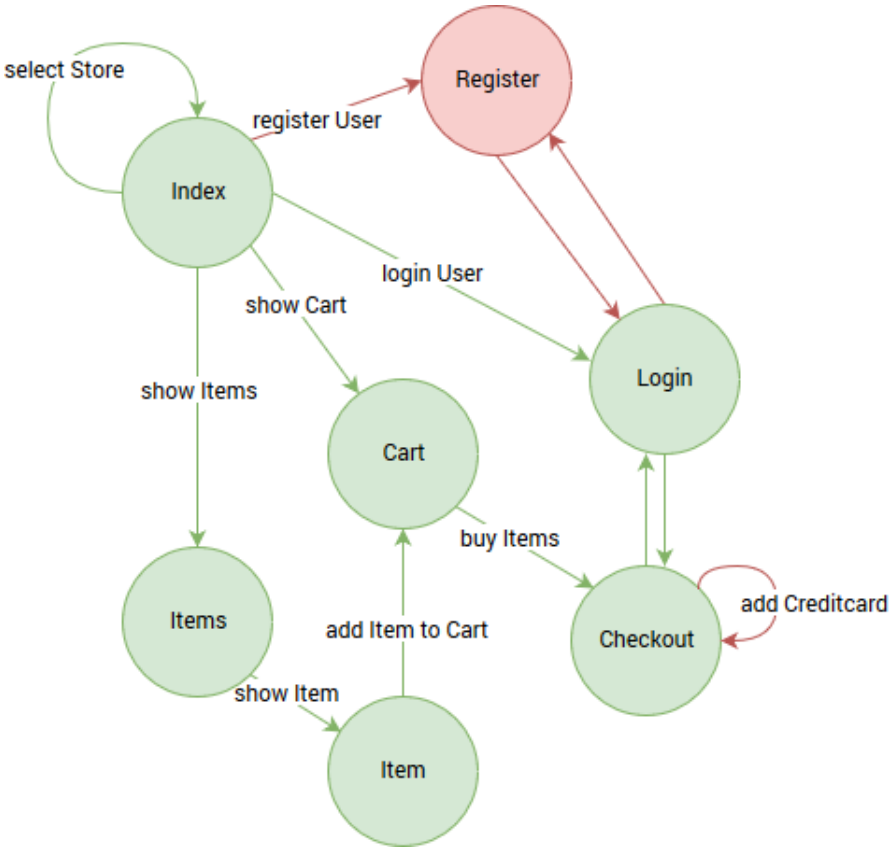

Fig. 14. App Tests Status

Currently the usecase "CreateAppCustomer" (UC 15) is not implemented and therefore the matching paths are still red.

Test protocol:

Step 1: Open the app in the browser.

Step 2: Select an enterprise and a store (Data depends on the CoCoME data)
Step 3: Click on an item in the now open page.
Step 4: Click on the "Add to cart" button.
Step 5: Click on the cart icon to open the cart.
Step 6: Click on the "Checkout" button.
Step 7: Confirm the error message and open the login page over the menu.
Step 8: Go to the cart again.
Step 9: Click on the "Checkout" button.
Step 10: Add a creditcard by clicking on the button.
Step 11: Enter your creditcard informations.
Step 12: Select a card and click on the "Buy" button.
Step 13: Enter your Pin.

5.1.2 Unit test The app's unit tests are realized with jasmin.js. Jasmin.js is a javascript framework for unit tests.

Fig. 15. Jasmin results

Test 1:
Description: Test if the app is initialized
Preconditions: None
Successful condition: App is loaded
Possible failures: App is not loaded
Test 2:
Description: Test if the app has loaded the pages
Preconditions: None
Successful condition: Pages are callable
Possible failures: Pages not found
Test 3:
Description: Test if the index page can be loaded
Preconditions: Pages are callable
Successful condition: Index page is callable
Possible failures: Index pages not found

5.2 Adapter

The adapter is tested by JUnit tests. Figure 16 shows the current status of these tests. All services in the adapter are tested.

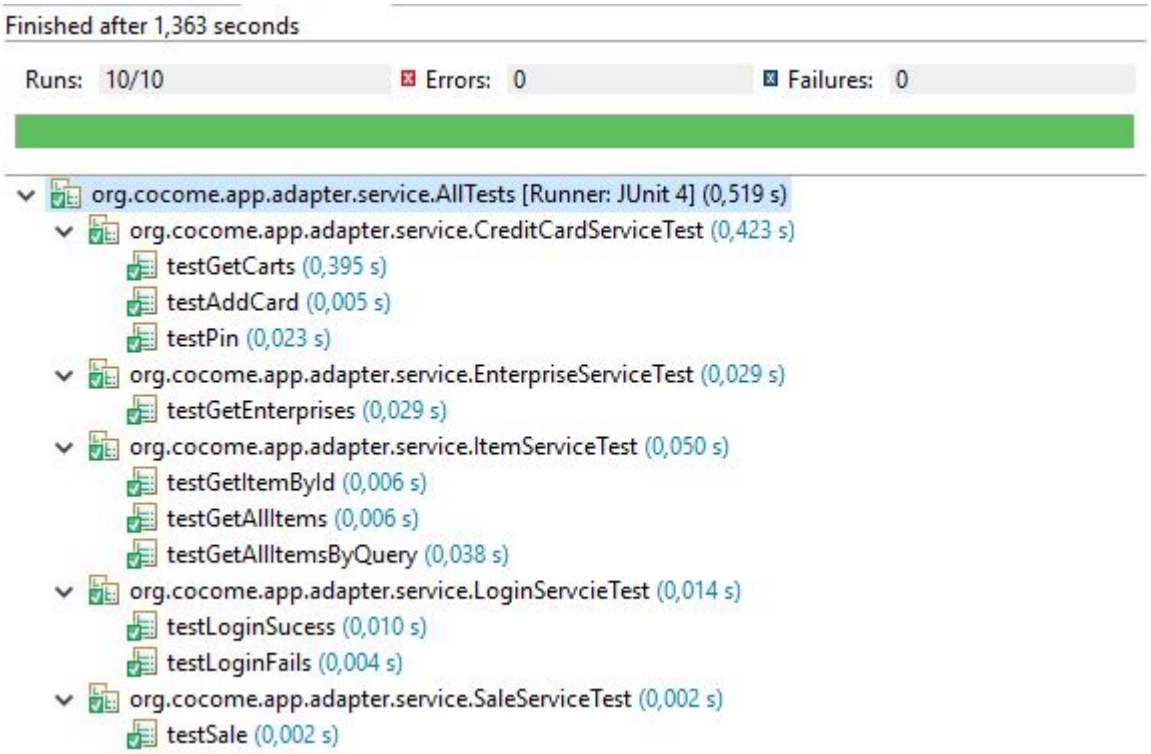

Fig. 16. Adapter Tests

Test 1:
Description: Test the creditcard service
Preconditions: none
Successful condition:

- Stored cards are accessible
- New Cards can be added
- Pin of card can be checked

Possible failures:

- Stored cards are not accessible
- New cards cannot be added
- Pin of card cannot be checked or give wrong result

Test 2:
Description: Test enterprise service
Preconditions: none
Successful condition: Get a list of all enterprises

Possible failures: Get no or incomplete list
Test 3:
Description: Test Item service
Preconditions: none
Successful condition:

- All items can be accessed.
- Single item can be selected by its id.
- Single item can be selected by its name.

Possible failures:

- List of items is not complete.
- Single item cannot be selected by its id.
- Single item cannot be selected by its name.

Test 4:
Description: Test login service
Preconditions: none
Successful condition:

- Valid user can be logged in.
- Invalid user gets rejected.

Possible failures:

- False login is possible.
- Existing user cannot log in.

Test 5:
Description: Test sale service
Preconditions: none
Successful condition: Sale can be processed.
Possible failures: Sale cannot be processed.

6 Summary

This chapter shows screen shots of the working app. The development process from the mockups to the finished pages is very clear.

The use cases have not been fully implemented yet. But the current state shows the app's potential. All important functionalities were implemented and were tested extensively.

It has turned out to be a good choice to program the app in typescript. With the using of Cordova and Onsen UI the app can be deployed on several operating systems.

But there were also some problems. The communication between javascript and wsdl services was impossible. On the one hand there is no working library

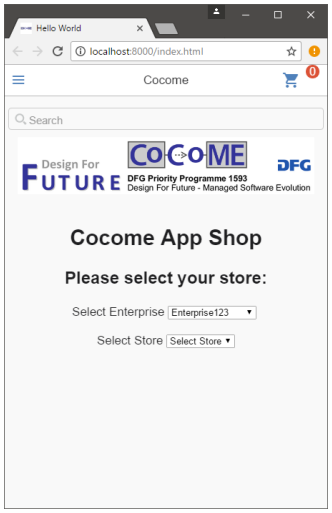

Fig. 17. The final index page

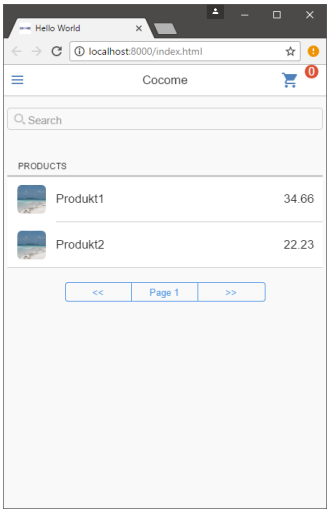

Fig. 18. The final items page

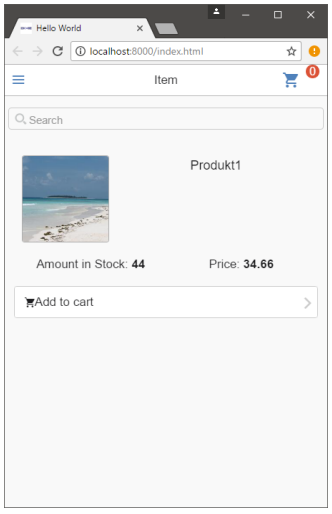

Fig. 19. The final item page

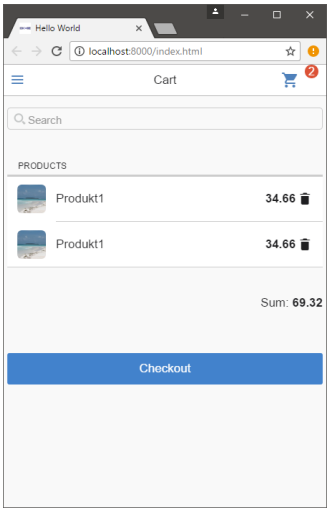

Fig. 20. The final cart page

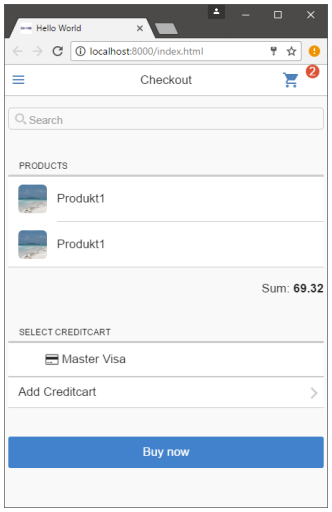

Fig. 21. The final checkout page

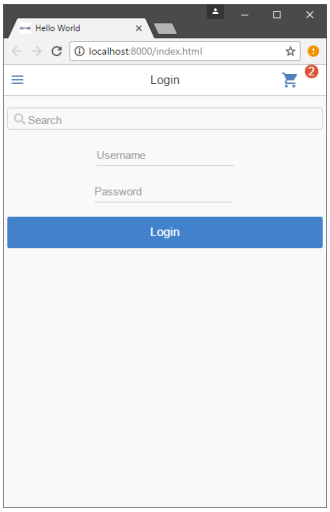

Fig. 22. The final login page

which supports the communcation to wsdl, on the other hand there was not enough time to implement this.

Another difficulty was the administration of JavaScript packages. However, this could be well implemented by using requriejs. This manager loads the required packets at runtime from the file system or webserver.

The compilation for iOS is still pending and needs to be explored. This was also not possible due to the lack of time.

7 Outlook

The integration test should be automated which could be done with selenium. This would cause errors to be recognized early. Also the app's unittests should be extended. These are not completely implemented yet.

The missing use case should be implemented. Furthermore the app should provide more information about the items and a user rating system would be very promising.

8 Conclusion

This work has shown that the incorporation of a new component brings many dangers, for example, the incompatibility between existing technologies and new technologies. As shown in this example, this must be solved by an adapter. The use of several frameworks takes many tasks, but this requires a longer processing time.

Programming this app has been interesting and a lot of fun. It has shown many new aspects of multi OS programming.

References

Chun17. Andy Chung. RequireJS. http://requirejs.org/, 2017.
Foun15. The Apache Software Foundation. Overview. https://cordova.apache.org/docs/en/latest/guide/overview/index.html, 2015.
Knoc17. Knockout. Knockout.js. http://knockoutjs.com/, 2017.
Micr17. Microsoft. Typescript. https://www.typescriptlang.org/, 2017.
Soft17. Pivotal Software. Spring Boot. https://projects.spring.io/spring-boot/, 2017.
xOns16. Monaca x Onsen UI Team. Onsen UI 2: Beautiful HTML5 Hybrid Mobile App Framework and Tools. https://onsen.io/, 2016.