Fastuga Driver

Ana Martins
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
2201711@my.ipleiria.pt

André Pinto
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
2201723@my.ipleiria.pt

Ivo Bispo
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
2200672@my.ipleiria.pt

Sara Martins
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
2201757@my.ipleiria.pt

Catarina Reis
Departamento de Engenharia Informática
Instituto Politécnico de Leiria
Leiria, Portugal
catarina.reis@ipleiria.pt

Abstract

The development of an Android app using agile methodology allows for a flexible and iterative approach to the project. By prioritizing collaboration, communication, and adaptability, the team can respond quickly to changes and incorporate user feedback throughout the development process.

Using the framework Scrum and methodology Extreme Programming (XP), the development team (we) can break down the app into small, manageable chunks and work on them in short sprints. This allows for frequent checkpoints and adjustments, ensuring that the app is delivered on time and meets the user's needs.

Through the use of agile principles and practices, the development team can deliver high-quality, user-focused Android apps that provide value to the client.

Keywords—Android app, agile methodology, Scrum, XP

I. Introduction

The FasTuga restaurant wants to further increase its business by providing extra value to its customers through a driver mechanism, so they may enjoy their meals in the comfortable environment of their homes.

The idea is to provide a platform to manage a driver's needs regarding the delivery of one or several orders, through a dashboard with assigned and available orders, the presentation of the driver's information, and features that optimize delivery time and allow a view of the driver's performance.

This document is divided into four sections referring to the description (II), development process (III), architecture (IV), and forms of verification and validation (V) of the project, with the support of diagrams and figures to ease understanding.

II. System description FasTuga Driver

The main aim of this project is to develop software to provide a driver feature for FasTuga restaurant in order to expand its business and provide extra value to its customers, which will allow drivers to get involved in the business.

The software is dedicated exclusively to FasTuga restaurant drivers, although anonymous users can use it (in the login and register phases), thus the main functionalities focus on the drivers' delivery and performance needs.

To provide those needs, some main functionalities were implemented, namely a login (Fig. 1) and a register feature (Fig. 2), a dashboard (Fig. 3) with the driver's name and balance and two lists (one for assigned orders and another for available orders), a screen for the order details, (Fig. 4), which includes additional information that may be useful to the delivery - e.g. the customer's name - a map with the directions to the destination and the buttons corresponding to the order status, a screen for the statistics (Fig. 5) in order to give the driver an understanding of his overall performance, a screen for the driver's profile (Fig. 6) in order to consult his information, and a Menu (Fig. 7) with the options "keep me logged in", to avoid excessive logins and to allow direct access to the orders lists, "log out" and "opt-out", in case the driver is unhappy with the application performance and wants to delete his account, a screen to check the notifications history (Fig. 8), a screen to change the name (Fig. 9) and the password (Fig. 10), and a screen to cancel an order (Fig. 11).

The assigned orders can have three states:

- "Preparing": when the order includes hot dishes and is being prepared in the kitchen
- "Ready to Pick": when the order is ready to be picked by the driver
- "Delivering": when the order is being delivered by the driver

The available orders only have one status ("Available").

System's flux diagram:

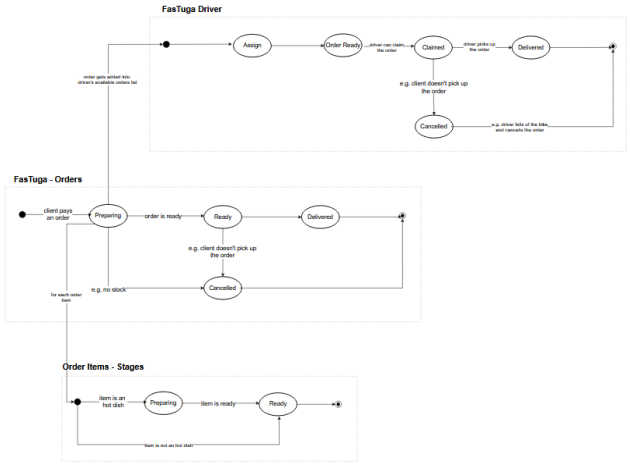


Fig. 1. System Flux Diagram

Status flow diagram:

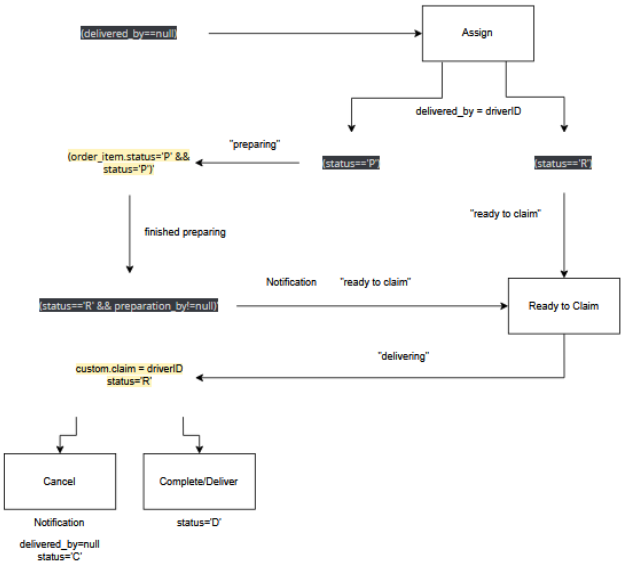


Fig. 2. Status Flux Diagram

Applications' screenshots:

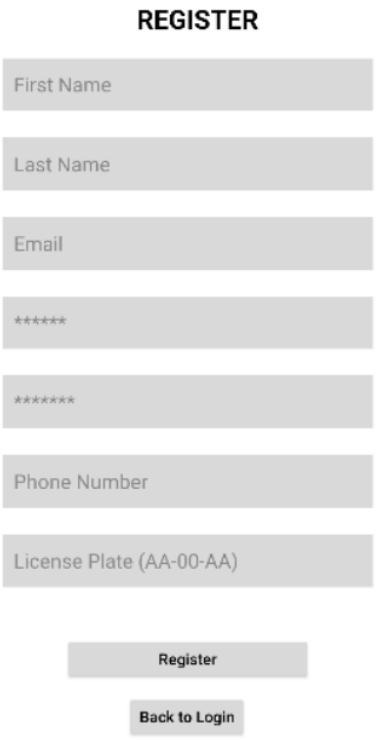


Fig. 3. Register Activity

Fig. 4. Login Activity

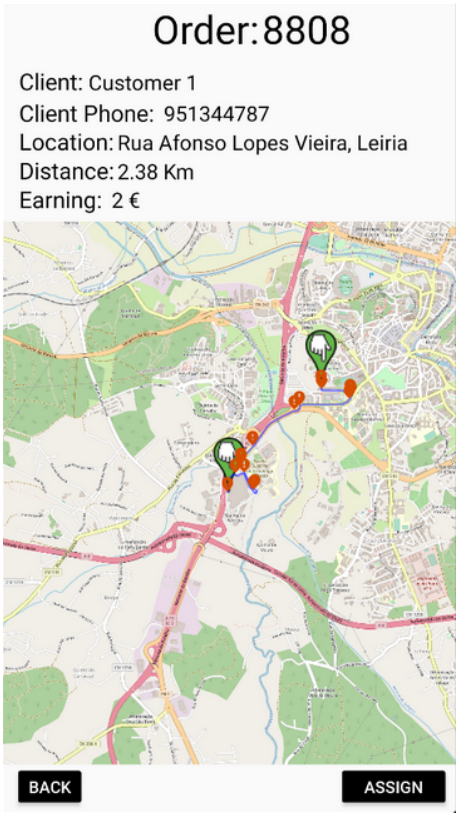


Fig. 5. Order Details Fragment and Map Fragment

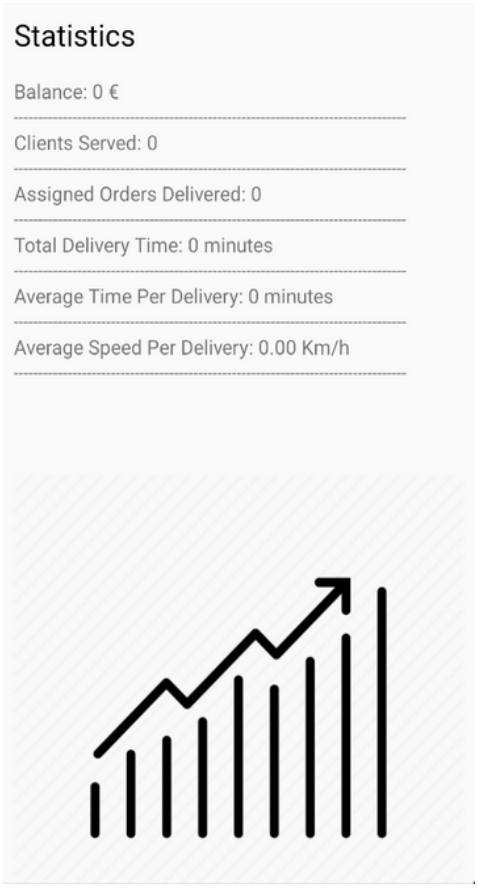


Fig. 6. Statistics Fragment

Fig. 7. Profile Fragment

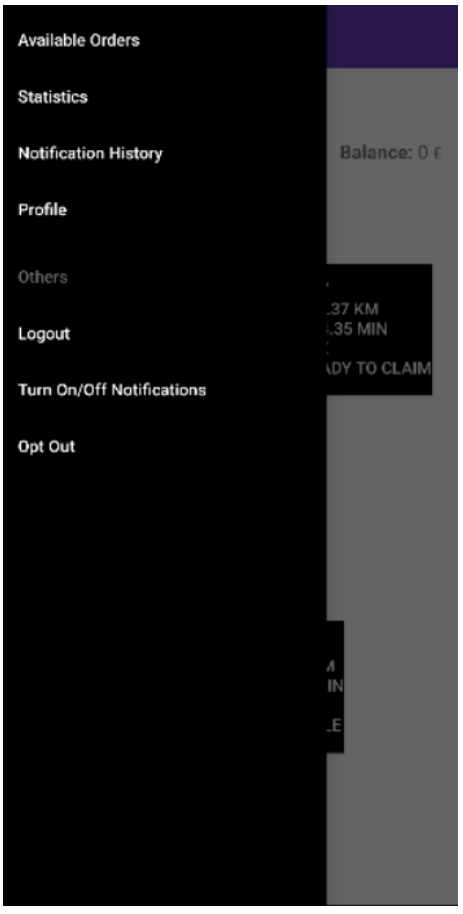


Fig. 8. Menu Fragment

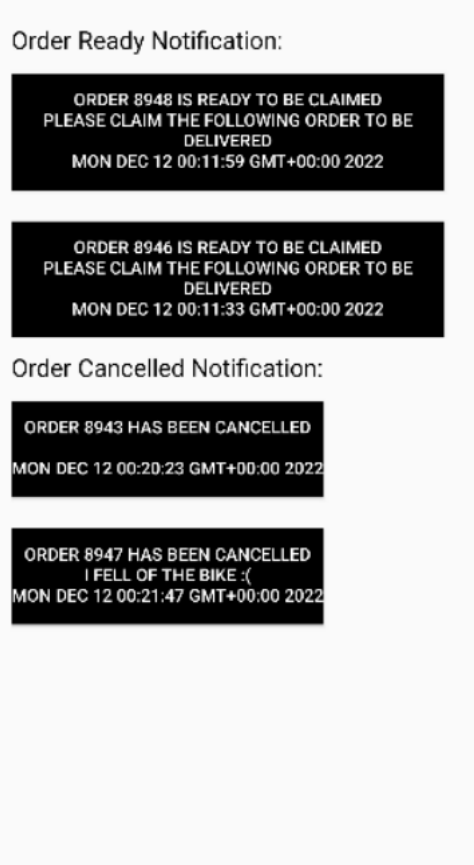


Fig. 9. Notification History Fragment

Fig. 10. Change Name Fragment

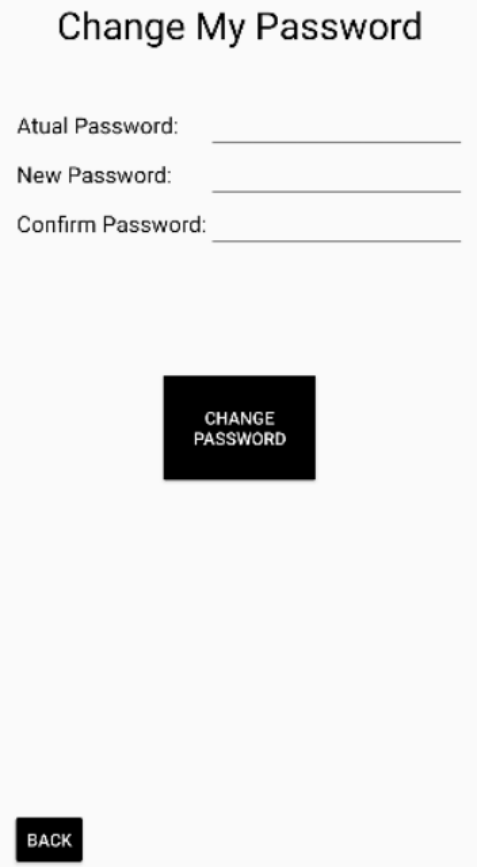


Fig. 11. Change Password Fragment

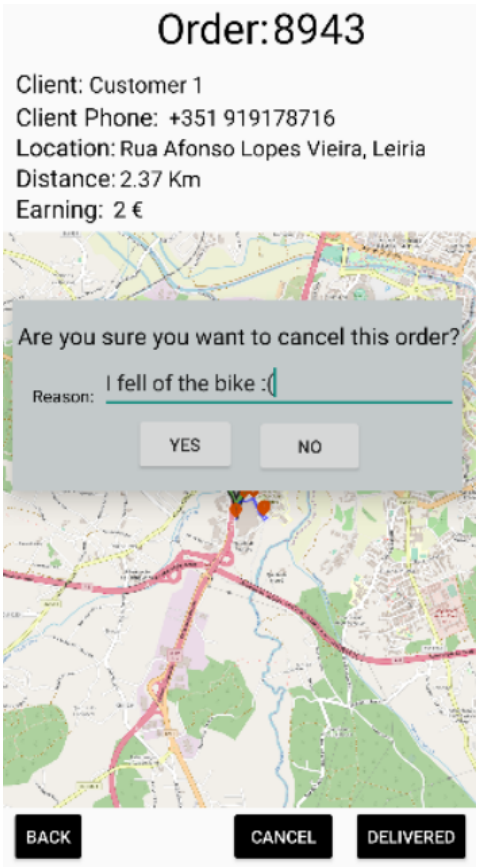


Fig. 12. Cancel Order Pop-up

III. Software Development Process

During the development of the Fastuga Driver application, our team adopted a development process based on agile methodologies, namely the Scrum framework and the Extreme Programming (XP) methodology. Because Scrum is an incomplete framework, we used XP to achieve a more complete, organized, flexible and communicative development process.

At the beginning of each week we planned and started a new sprint with the customer and defined what should be done. After that we prepared and defined what should be done for the following week's sprint, through XP user stories. After defining the user stories, we wrote down the automated tests that should be checked and passed with success. We didn't use daily scrum, but at least 3 times a week, we met remotely and gave feedback on the course of the sprint.

At the end of the sprint, the whole team would meet with the client and present what had been done, the developed functionalities and the working tests, not showing the client everything that was not fully implemented. We also made a retrospective about the work developed during the sprint, where we identified the problems found during the sprint and what we could do differently to improve it in the next sprint.

Regarding the implementation, we started by creating new branches for each user story, developing all the corresponding code in that branch and at the end a pull request was made, so that all the team members could see and approve the code.

Concerning the test repository, we also used pull requests. Sometimes we used pair programming, which helped us understand the code and was very helpful when we got stuck in some part of the development.

In the beginning it was difficult to maintain this whole development structure. However, as the weeks went by, it was easier to follow all the steps to achieve a good result at the end of each sprint.

IV. System Architecture

C4 Model:

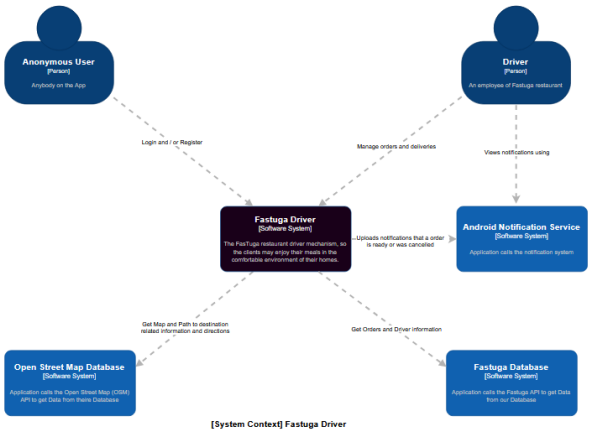


Fig. 13. C4 Model - Level 1: Context

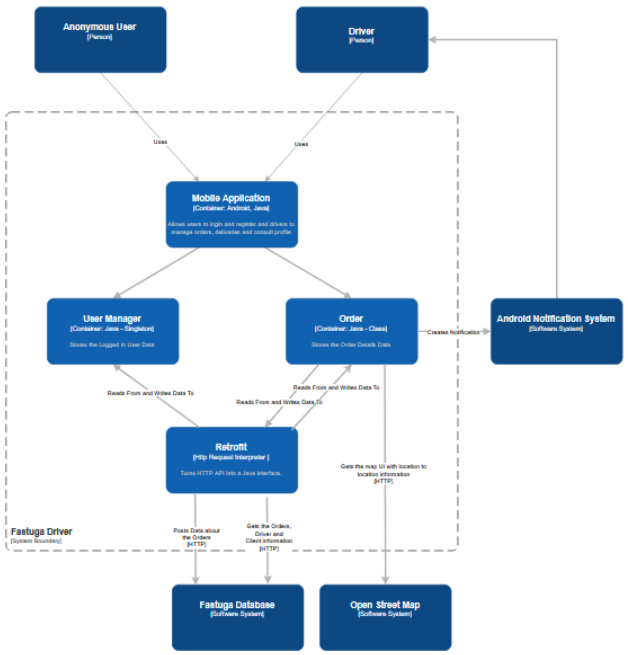


Fig. 14. C4 Model - Level 2: Container

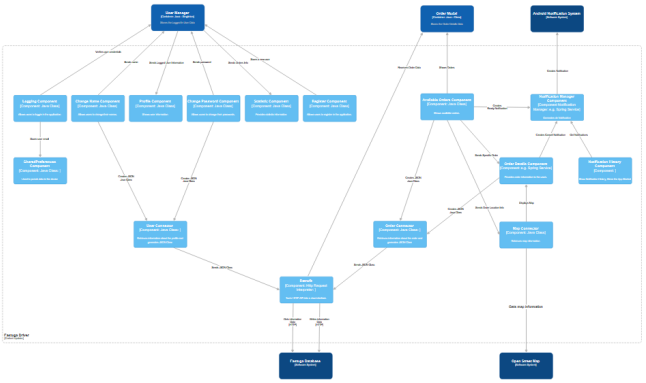


Fig. 15. C4 Model - Level 3: Component

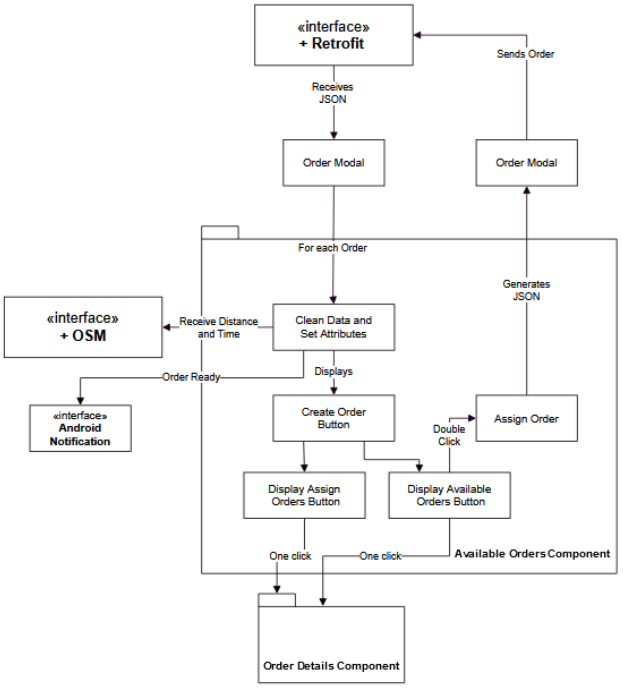


Fig. 16. C4 Model - Level 4: Code - Available Orders

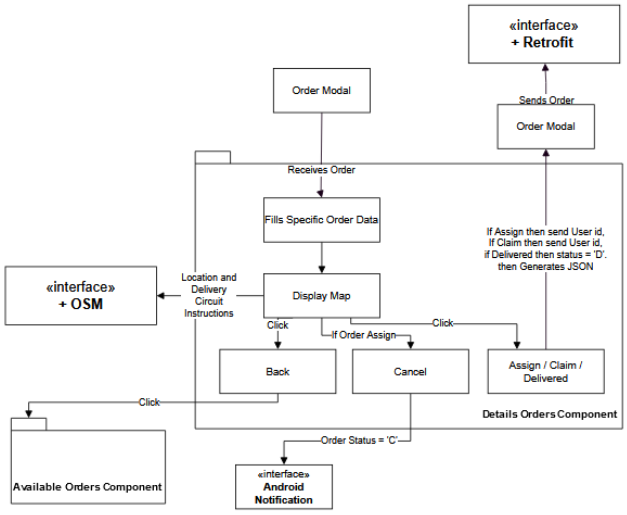


Fig. 17. C4 Model - Level 4: Code - Order Details

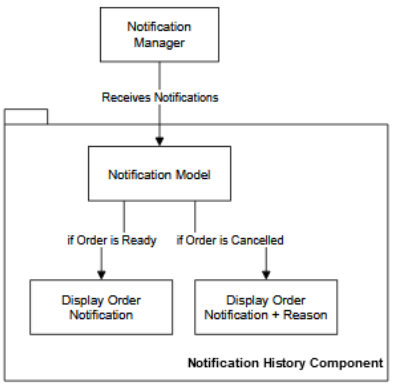


Fig. 18. C4 Model - Level 4: Code - Notification History

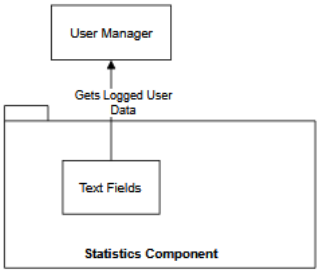


Fig. 19. C4 Model - Level 4: Code - Statistics

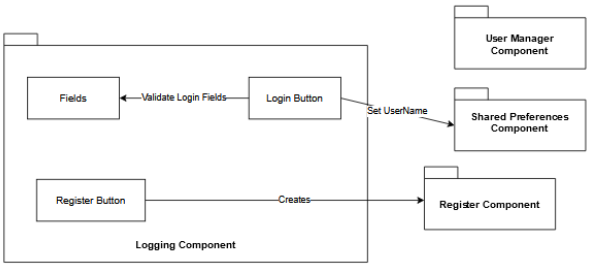


Fig. 20. C4 Model - Level 4: Code - Login

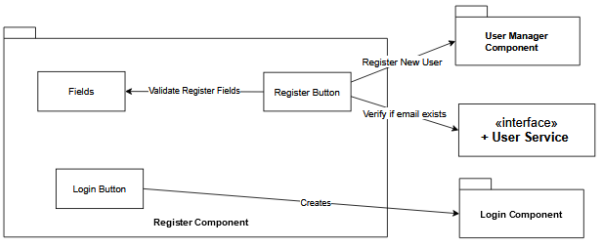


Fig. 21. C4 Model - Level 4: Code - Register

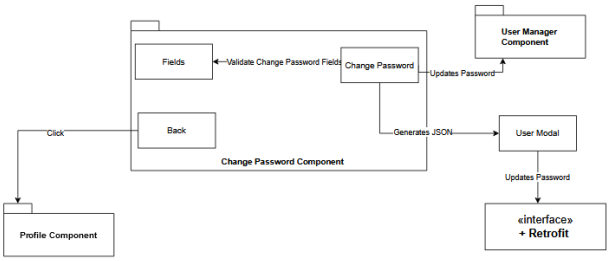


Fig. 22. C4 Model - Level 4: Code - Change Password

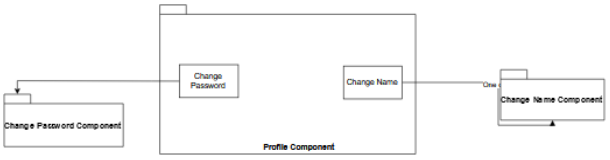


Fig. 23. C4 Model - Level 4: Code - Profile

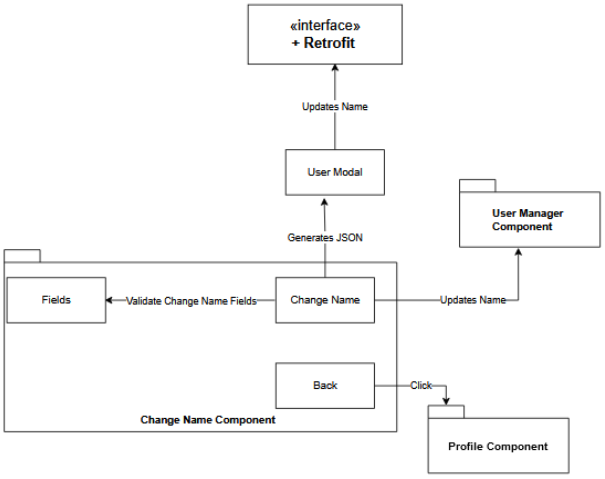


Fig. 24. C4 Model - Level 4: Code - Change Name

3-Layered Model:

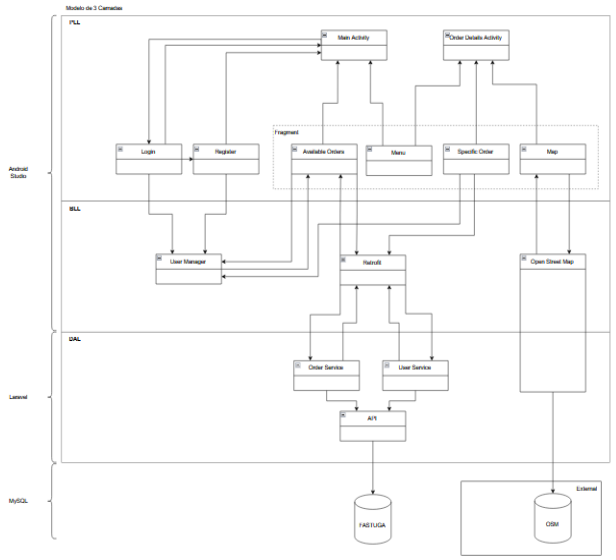


Fig. 25. System Architecture - 3 Layer Model

V. Verification and Validation

For the verification and validation part of our code we, whilst developing the main code, tried to follow a Behaviour-Driven Development (BDD) where we first write down what behaviour a specific user story (US) is supposed to implement and then implement it. This was done for the vast majority of the stories we had. This consisted of using Katalon Studio's Software to capture objects and run tests, test suites and feature files written in Gherkin, with corresponding step definitions written in Groovy.

Our approach was, during an initial phase, to manually generate the test cases to better understand the Katalon Studio software and how it works. This was mainly done for the first user stories (Login and Register), where we implemented all the test cases that were needed. When we finished all the automated tests for these initial user stories, we then made the feature files and their step definitions.

After US1 and US2, the development of these automated tests went smoother as we better understood what we were supposed to do.

Then, for the following user stories and to try to better implement the BDD way, we first wrote the feature file for the user story we were about to develop, in a standard way that we decided amongst ourselves (to take advantage of BDD and reuse most of the step definitions, so that in the later stages we didn't need to write most of the step definitions since they were already done). When the feature file had a good first version, we then used the generated apk from that user story branch and captured the objects needed to implement them, in a common Object Repository, as we felt that it was the best approach, as it allowed us to use these captured objects in various tests for different user stories. Later, for the step definitions written in groovy, we placed each file in a folder named after the user story it was initially made for.

This was the process that was done for the great majority of the user stories and it was the way that we found most suitable for the tasks we had, and overall everything went well.

For most of the feature files written, we used the Background keyword in Gherkin to reuse some common behaviour and also used Scenario Outlines for the Login feature file.

However, we had some difficulties (mainly due to the instability of Katalon Studio on our machines – one of our team members couldn't even run a simple test case). These difficulties were mostly experienced during the initial user stories (Login and Register), where we struggled to find a way to capture the errors placed in the text fields in the login and register forms. Throughout the process, we also sometimes encountered some minor issues like not being able to open a particular Test Suite.

We also would like to note that in order to properly run some automated tests on some feature files (the ones that involve manipulating orders and clicking on them, and those that must have orders in a certain state – e.g. Delivering), we must first configure the database to make sure that we have orders to run that specific scenario (these instructions are written, in a comment at the top of each affected feature file – e.g. in the us10.feature file we must configure the driver to have at least 2 orders in his assigned orders list). Another note worth mentioning is that we needed to add the setting 'appWaitActivity' to our Katalon Studio project.

Finally, for each user story, we ran a Test Suite that runs a Test Case that executes that user story's feature file, which allows us to see reports of these automated tests with several details such as the date, which tests passed, how long it took to execute them...

VI. Software Design patterns

Some of the patterns we used were the Adapter Call (we used an adapter in the User and Orders classes), decorator (Scroll view for available orders and assigned orders), and the Command (with the implementation of the Menu).

VII. Conclusions and Future Work

In conclusion, a food delivery app can provide opportunities for the drivers of Fastuga restaurants to earn income by delivering meals to customers. This can be especially beneficial for individuals who are looking for flexible work opportunities or who want to supplement their existing income. By using a food delivery app, drivers can easily connect with customers who are looking to have

meals delivered from the restaurant they work for. Comprehensive, a food delivery app can be a valuable tool for drivers who are looking to earn money by providing food delivery services.

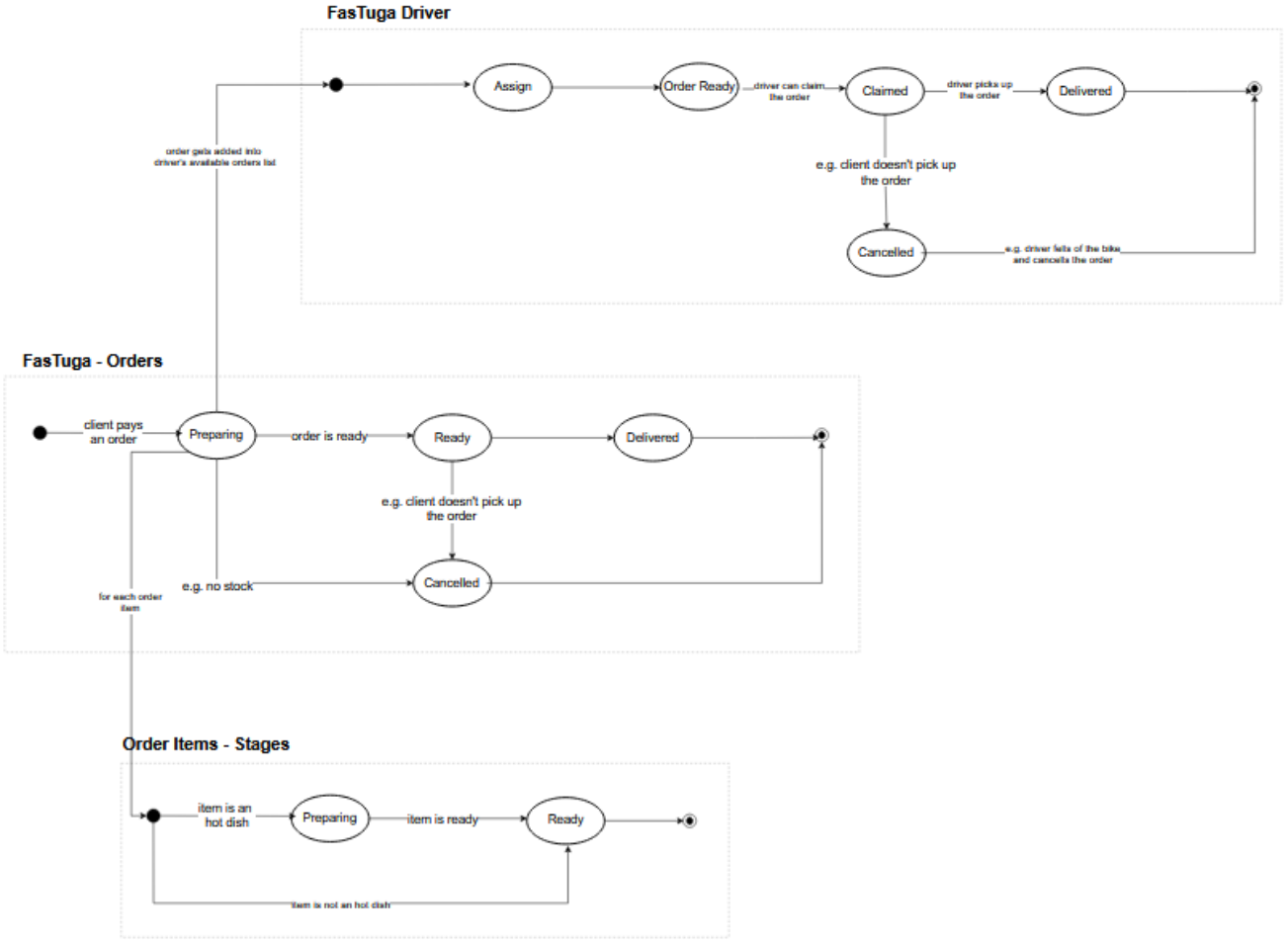
The use of agile in the development of Fastuga Driver App provided several benefits. Agile is a flexible and iterative approach to software development that emphasizes collaboration, continuous feedback, and regular testing and refinement. By using agile, we responded quickly to changing requirements or feedback from the client and delivered high-quality app updates regularly. Additionally, agile helped improve communication and collaboration within the development team (us) and promoted a culture of experimentation and innovation. However, it is important to carefully consider the potential drawbacks and limitations of using agile, such as the need for a high level of coordination and collaboration among team members.

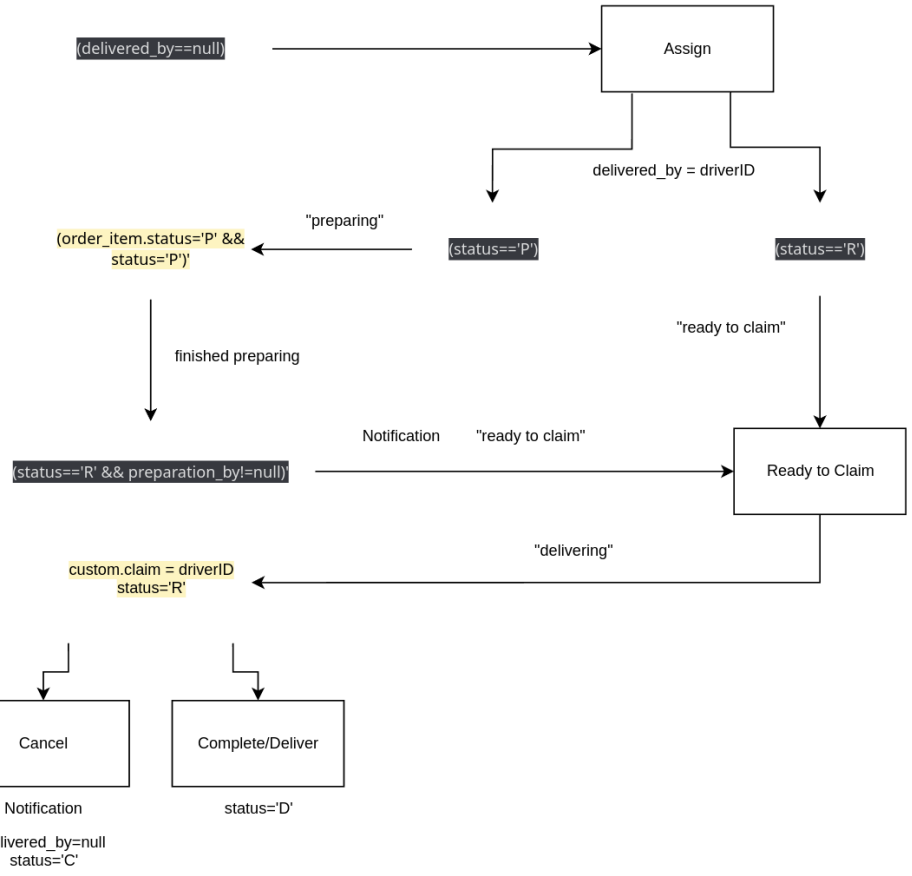
In terms of future work, there are several areas that could be explored in relation to agile app development. For example, further research could be done on how to effectively integrate agile methods with other development approaches, such as waterfall or lean. Additionally, there may be potential for using artificial intelligence and machine learning to automate certain aspects of the agile process, such as testing and feedback gathering.

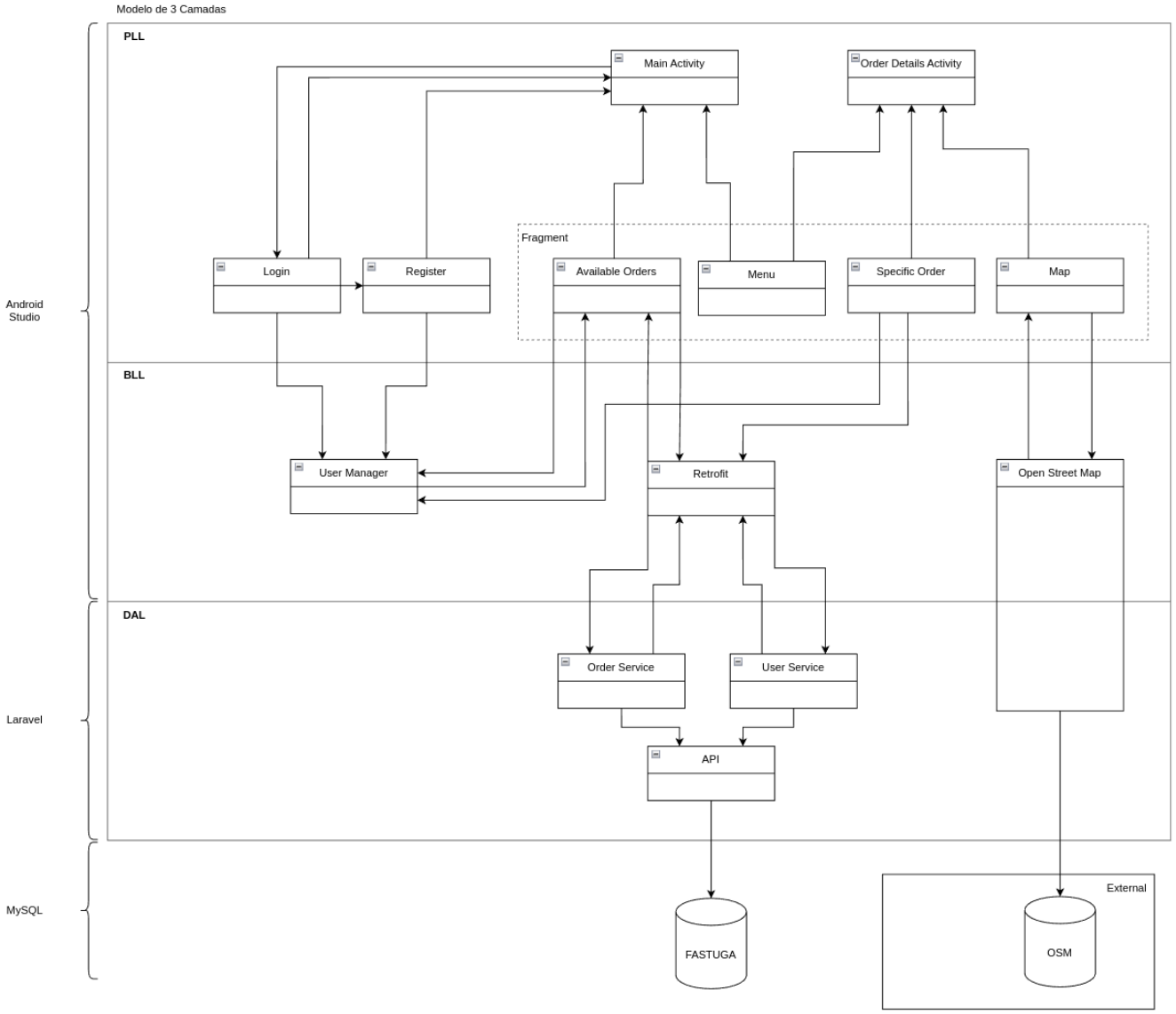
Overall, agile app development has proven to be a valuable approach for creating high-quality software that meets the needs of the client.

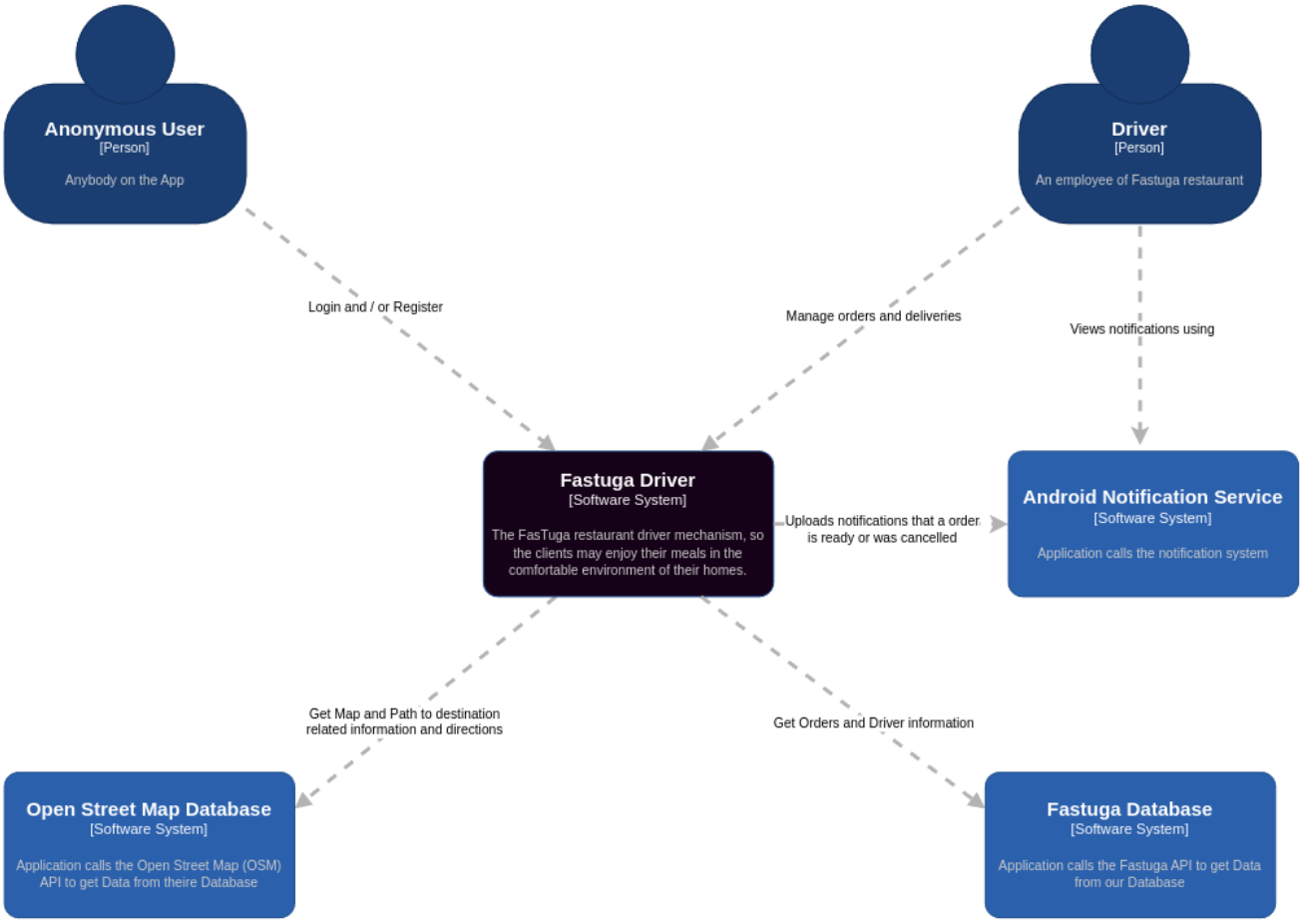
VIII. REFERENCES

[1] https://square.github.io/retrofit/
[2] https://developer.android.com/docs
[3] https://github.com/MKergall/osmbonuspack
[4] https://docs.katalon.com/docs
[5] https://docs.katalon.com/docs/legacy/katalon-studio-enterprise/keywords/mobile-keywords/mobile-start-existing-application
[6] https://developer.android.com/reference/android/content/SharedPreferences

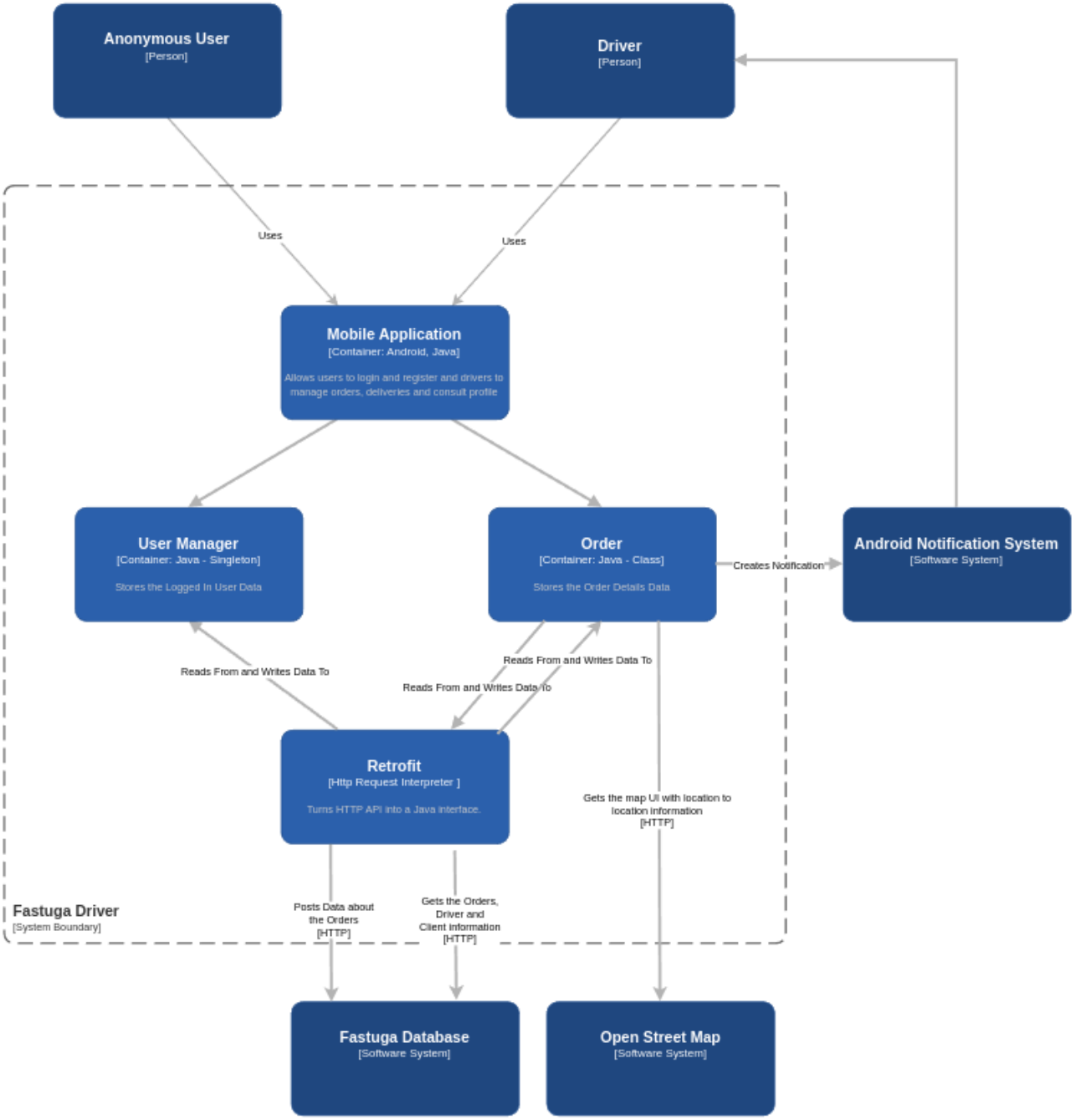

FasTuga Driver
Assign
Order Ready
driver can claim the order
Claimed
driver picks up the order
Delivered
order gets added into driver's available orders list
e.g. client doesn't pick up the order
Cancelled
e.g. driver falls of the bike and cancells the order
FasTuga - Orders
client pays an order
Preparing
order is ready
Ready
Delivered
e.g. client doesn't pick up the order
Cancelled
e.g. no stock
for each order item
Order Items - Stages
item is an hot dish
Preparing
item is ready
Ready
"item is not an hot dish"

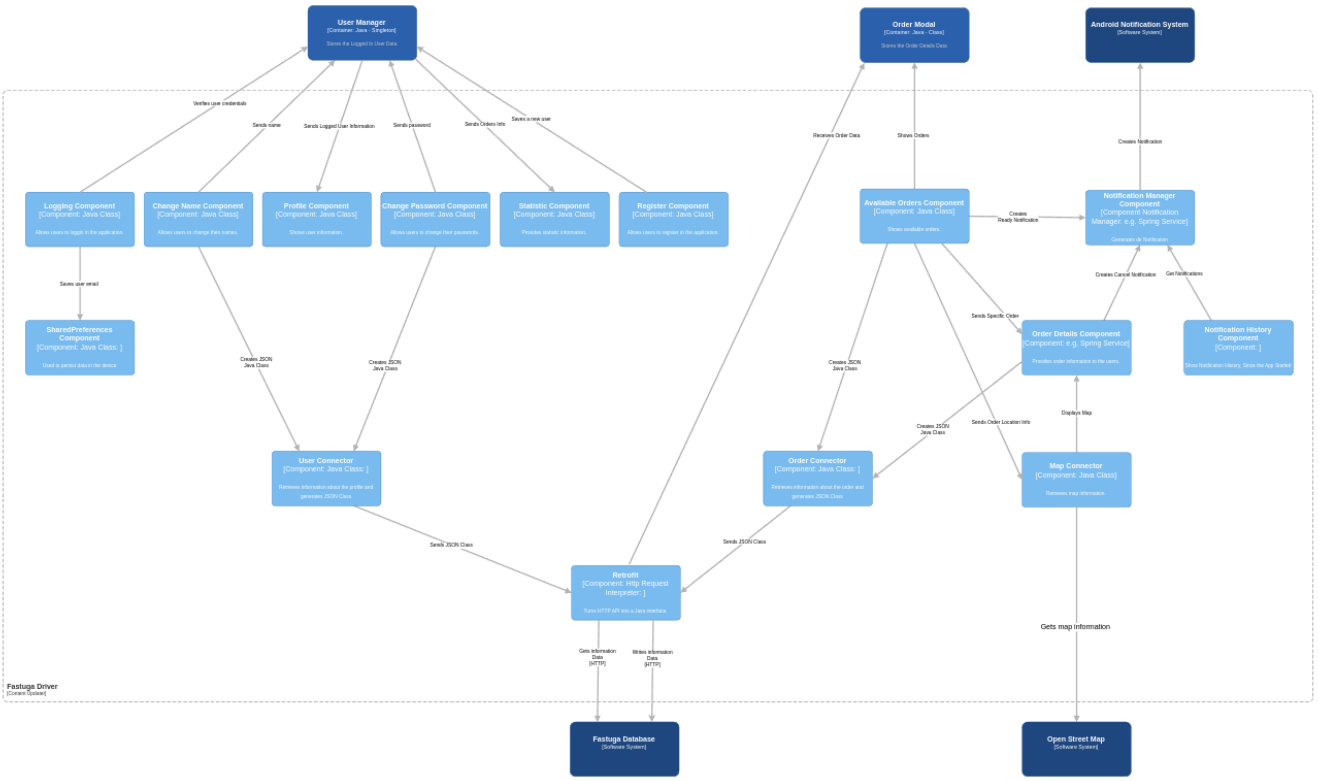

(delivered_by==null)
Assign
delivered_by = driverID
"preparing"
(order_item.status='P' && status='P')'
(status=='P')
(status=='R')
"ready to claim"
finished preparing
Notification
"ready to claim"
(status=='R' && preparation_by!=null)'
Ready to Claim
"delivering"
custom.claim = driverID
status='R'
Cancel
Complete/Deliver
Notification
delivered_by=null
status='C'
status='D'

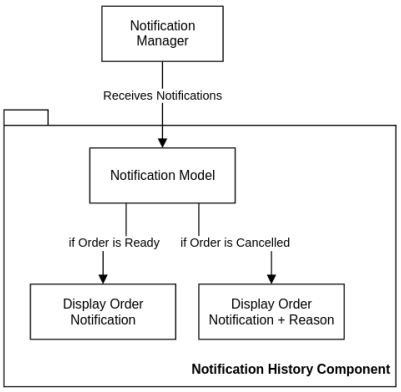

Modelo de 3 Camadas
PLL
Main Activity
Order Details Activity
Fragment
Login
Register
Available Orders
Menu
Specific Order
Map
Android Studio
BLL
User Manager
Retrofit
Open Street Map
DAL
Order Service
User Service
Laravel
API
MySQL
FASTUGA
External
OSM

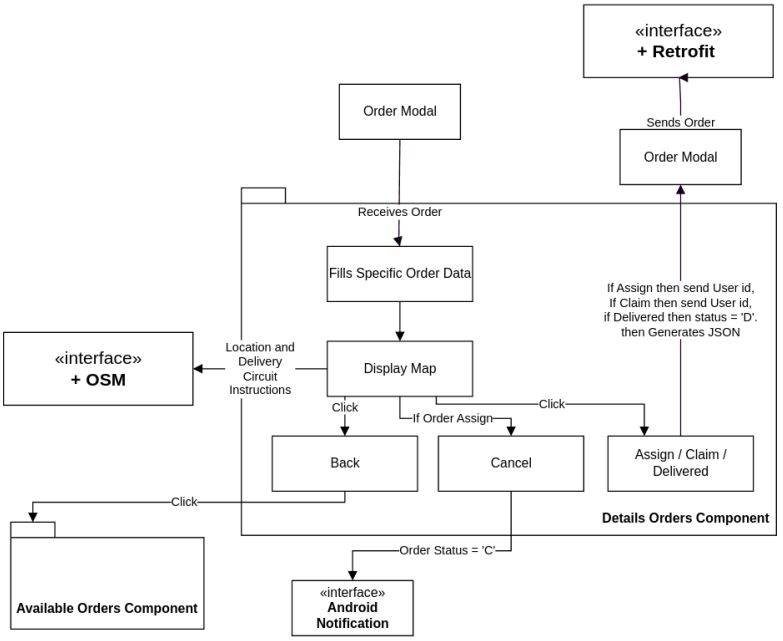


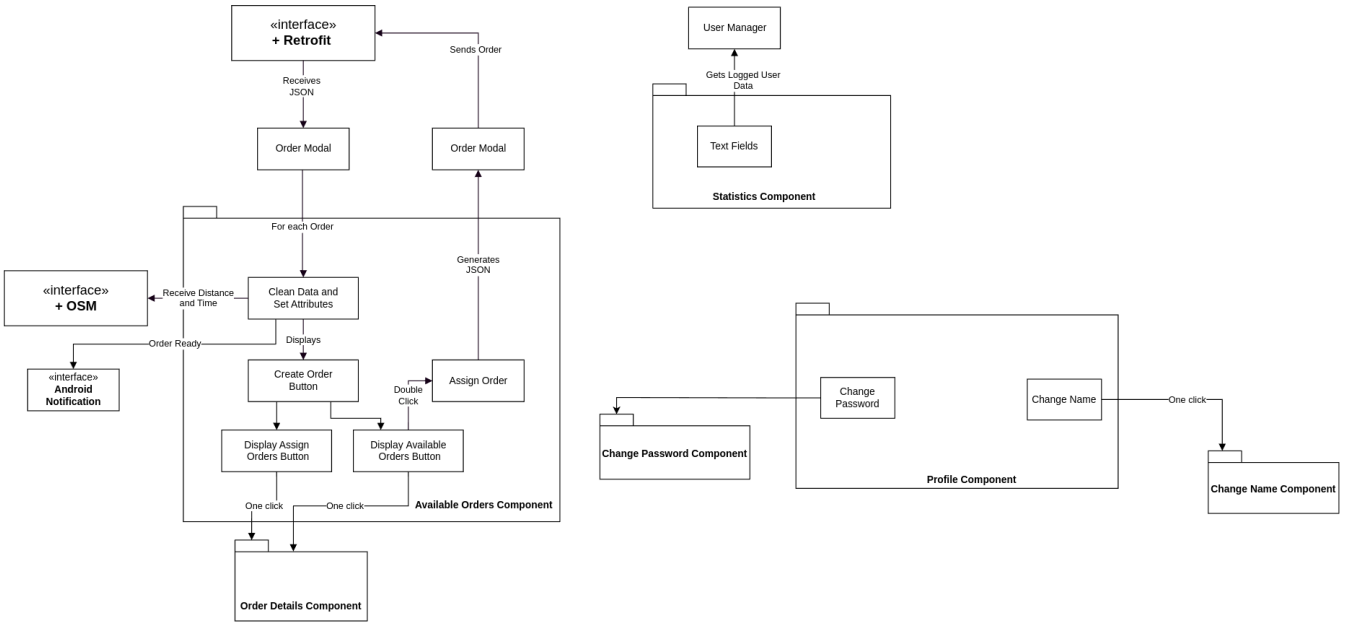
[System Context] Fastuga Driver

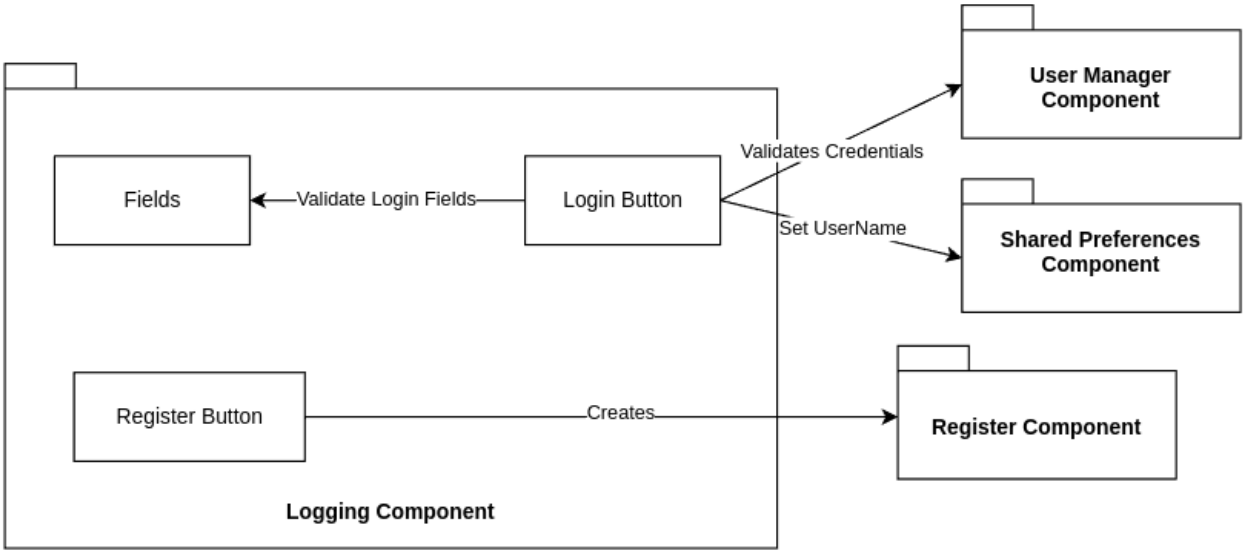

Anonymous User
[Person]
Driver
[Person]
Uses
Uses
Mobile Application
[Container: Android, Java]
Allows users to login and register and drivers to manage orders, deliveries and consult profile
User Manager
[Container: Java - Singleton]
Stores the Logged In User Data
Order
[Container: Java - Class]
Stores the Order Details Data
Creates Notification
Android Notification System
[Software System]
Reads From and Writes Data To
Reads From and Writes Data To
Reads From and Writes Data To
Retrofit
[Http Request Interpreter]
Turns HTTP API into a Java interface.
Gets the map UI with location to location information
[HTTP]
Fastuga Driver
[System Boundary]
Posts Data about the Orders
[HTTP]
Gets the Orders, Driver and Client information
[HTTP]
Fastuga Database
[Software System]
Open Street Map
[Software System]

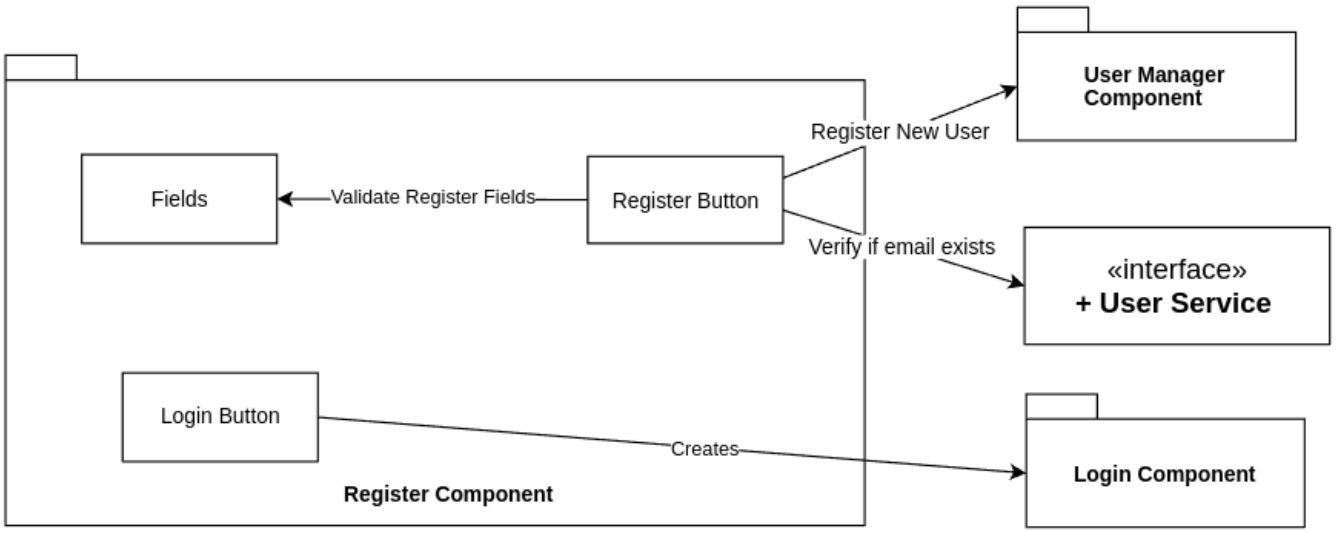

User Manager
[Container: Java - Dagger]
Stores the Logged In User Data
Order Modal
[Container: Java - Class]
Stores the Order Details Data
Android Notification System
[Software System]
Verifies user credentials
Sends name
Sends Logged User Information
Sends password
Sends Orders Info
Saves a new user
Receives Order Data
Shows Orders
Creates Notification
Logging Component
[Component: Java Class]
Allows users to toggle in the application.
Change Name Component
[Component: Java Class]
Allows users to change their names.
Profile Component
[Component: Java Class]
Shows user information.
Change Password Component
[Component: Java Class]
Allows users to change their passwords.
Statistic Component
[Component: Java Class]
Provides statistic informations.
Register Component
[Component: Java Class]
Allows users to register in the application.
Available Orders Component
[Component: Java Class]
Shows available orders.
Creates Ready Notification
Notification Manager Component
[Component Notification Manager, e.g. Spring Service]
Generates de Notification
Saves user email
Creates Cancel Notification
Get Notifications
Sends Specific Order
SharedPreferences Component
[Component: Java Class:]
Used to persist data in the device
Creates JSON Java Class
Creates JSON Java Class
Creates JSON Java Class
Order Details Component
[Component: e.g. Spring Service]
Provides order information to the users.
Notification History Component
[Component:]
Show Notification History, Since the App Started
Creates JSON Java Class
Sends Order Location Info
Displays Map
User Connector
[Component: Java Class:]
Retrieves information about the profile and generates JSON Class
Order Connector
[Component: Java Class:]
Retrieves information about the order and generates JSON Class
Map Connector
[Component: Java Class]
Retrieves map information.
Sends JSON Class
Sends JSON Class
Retrofit
[Component: Http Request Interpreter:]
Turns HTTP API into a Java interface.
Gets map information
Gets information Data [HTTP]
Writes information Data [HTTP]
Fastuga Driver
[Content System]
Fastuga Database
[Software System]
Open Street Map
[Software System]

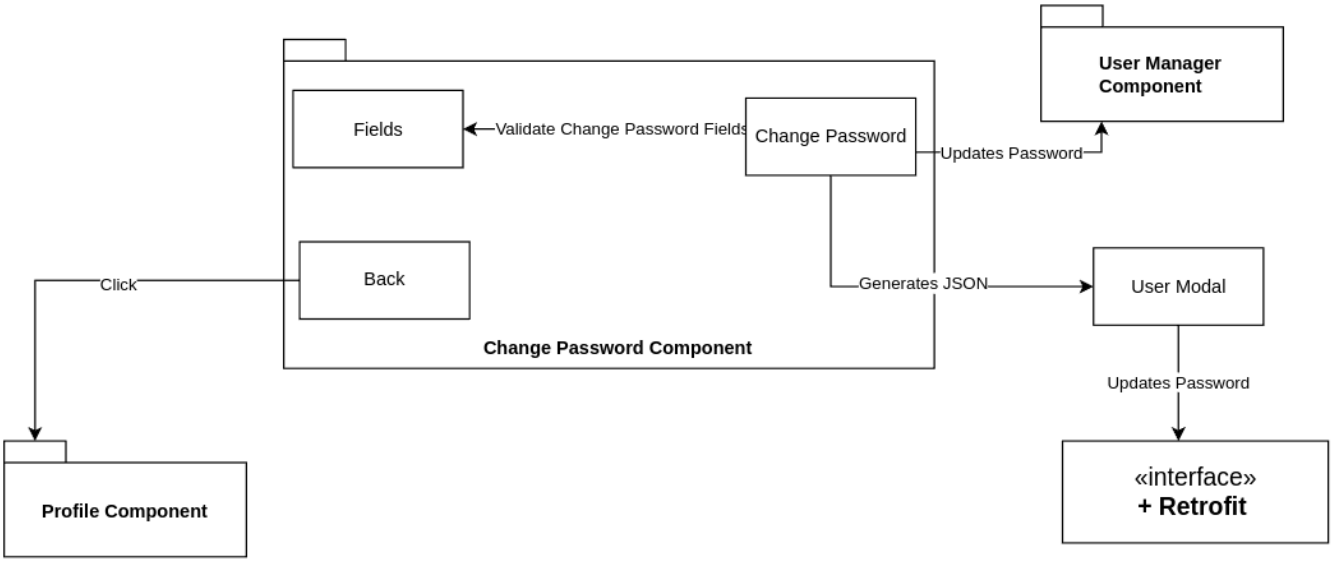

Notification Manager
Receives Notifications
Notification Model
if Order is Ready
if Order is Cancelled
Display Order Notification
Display Order Notification + Reason
Notification History Component

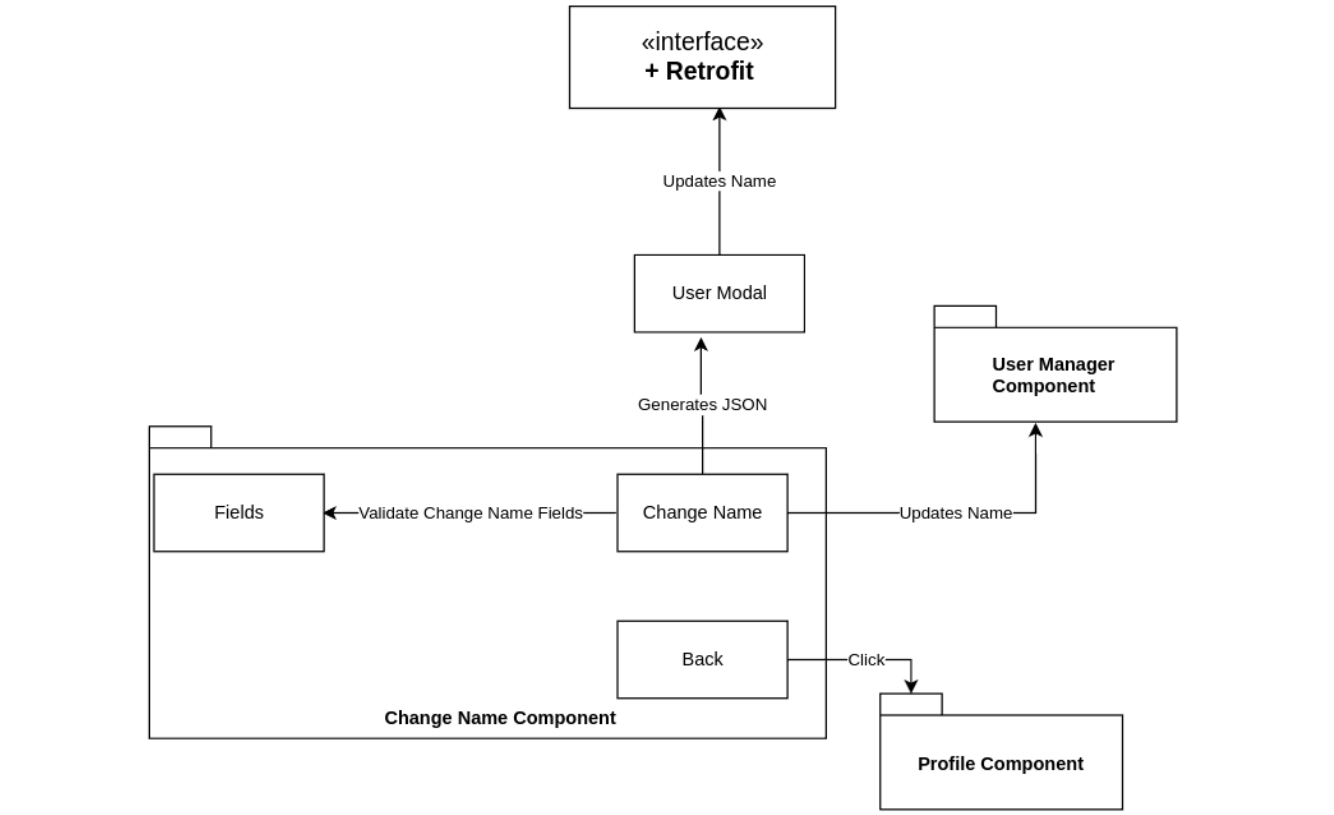

«interface»
+ Retrofit
Order Modal
Sends Order
Order Modal
Receives Order
Fills Specific Order Data
If Assign then send User id,
If Claim then send User id,
if Delivered then status = 'D'.
then Generates JSON
«interface»
+ OSM
Location and Delivery Circuit Instructions
Display Map
Click
If Order Assign
Click
Back
Cancel
Assign / Claim / Delivered
Click
Details Orders Component
Order Status = 'C'
Available Orders Component
«interface»
Android
Notification


«interface»
+ Retrofit
Sends Order
Receives
JSON
Order Modal
Order Modal
For each Order
Generates
JSON
«interface»
+ OSM
Receive Distance
and Time
Clean Data and
Set Attributes
Order Ready
Displays
«interface»
Android
Notification
Create Order
Button
Double
Click
Assign Order
Display Assign
Orders Button
Display Available
Orders Button
One click
One click
Available Orders Component
Order Details Component
User Manager
Gets Logged User
Data
Text Fields
Statistics Component
Change
Password
Change Name
One click
Change Password Component
Profile Component
Change Name Component


Fields
Validate Login Fields
Login Button
Validates Credentials
Set UserName
User Manager
Component
Shared Preferences
Component
Register Button
Creates
Register Component
Logging Component


Fields
Validate Register Fields
Register Button
Register New User
Verify if email exists
User Manager
Component
«interface»
+ User Service
Login Button
Creates
Login Component
Register Component


User Manager Component
Fields
Validate Change Password Fields
Change Password
Updates Password
Back
Click
Generates JSON
User Modal
Change Password Component
Updates Password
Profile Component
«interface»
+ Retrofit


«interface»
+ Retrofit
Updates Name
User Modal
User Manager Component
Generates JSON
Fields
Validate Change Name Fields
Change Name
Updates Name
Back
Click
Change Name Component
Profile Component