

Online Order System

IDIR YACINE

March 30, 2022

Contents

I	State of the art	4
1	general introduction	4
1.1	The Context	4
1.2	The Project Objectives	4
1.3	The MEMORY OVERVIEW	5
2	the project introduction	5
2.1	Introduction	5
2.2	Project objectives	5
2.2.1	the existence study	5
2.2.2	the critique of the existence	6
2.2.3	suggested solution	6
2.2.4	Client Section	6
2.2.5	Administrator section	6
II	Project Methologies	6
3	Agile Developement	7
3.1	Continous Integration	7
3.2	GitHub	7
3.3	Code Quality	7
3.4	Test Driven Developement	8
4	Project Architecture	8
4.1	Analysis Models	8
4.2	Domain Driven Design	8
4.3	Design Patterns	9

5	Technologies	9
5.1	TypeScript	9
5.2	Sql	9
5.3	Dart	9
6	TechStack	9
6.1	Flutter	9
6.2	NodeJs	10
6.3	React	10
6.4	Firebase	10
III	Project Implementation	10
7	Mobile client app	11
7.1	UI	11
7.1.1	Product	11
7.1.2	Catalogue	12
7.1.3	Cart	12
7.1.4	Login	13
7.1.5	Gps Location	13
7.1.6	Registration	14
7.2	Application	14
7.2.1	Providers	14
7.2.2	Helpers	15
7.3	Domain	16
7.3.1	Profile	16
7.3.2	Order	16
7.3.3	Cart	17
7.3.4	Catalogue	18
7.4	Infrastructure	19
7.4.1	Authentication	19
7.4.2	Orders	19
7.4.3	Database	20
7.4.4	UserData	20
7.4.5	ServerAccess	20
7.4.6	Exceptions Handling	21
8	Shop react app	22
8.1	UI	22
8.1.1	Login	22
8.1.2	Orders Board	23
8.1.3	Category Manager	24
8.1.4	Product Manager	25
8.1.5	Settings	26
8.2	Application	27

8.2.1	Attribute Cacher	27
8.2.2	Redux Store	27
8.3	Domain	29
8.4	Infrastructure	29
8.4.1	ApiConfig	29
8.4.2	Api	29
9	Shop node app	30
9.1	Customer Database	30
9.1.1	Ban mechanism	30
9.1.2	Optimising requests	30
9.1.3	Explaining some keys	30
9.2	Product Database	31
9.2.1	Explaining some keys	31
9.3	Infrastructure	31
9.3.1	AuthenticationService	31
9.3.2	StorageService	32
9.3.3	OrdersService	32
9.3.4	ProductsDtabase	32
9.3.5	CustomersDatabase	33
9.3.6	Configs	33
10	Installation	34
10.1	Dependencies	34
10.2	Mobile App	34
10.3	Node App	34
10.4	React App	34
IV	Appendix	35

Part I

State of the art

1 general introduction

1.1 The Context

The strengthening of interactions between trade and research institutions such as universities, colleges and research institutes has been increasingly seen as a strategic instrument for regional and national innovation, economic growth and competitiveness. The most recent innovation and research policies place a strong emphasis on interaction and cooperation between business and academia as a key public policy to foster innovation in the economy. Closer interaction is expected to lead to “more relevant research projects, faster uptake of scientific knowledge in the private sector, and better use of scientific knowledge” (NHD, 2003). University-business interaction is highlighted as a tool to boost business research and development capacity building, as well as a tool to make higher education and research more relevant and responsive to business needs. This policy direction is part of the overall political vision of becoming a leading knowledge-based country in the global knowledge-based economy (NHD, 2003).

1.2 The Project Objectives

In our project we are too interested for creating a collaboration between informatic and Commerce fields, so actually we gave our interesting about ”restaurant online orders” , restaurants are searching for Target a larger segment of clients by using , ads , deliveration food , but now we suggest a new way to solve their problem . Our platform is an intermediary that links customers and restaurants’. It helps the customer to reach his favorite restaurant easier. It also helps restaurants to expand their presence, especially in the information and Internet space, which will give the companies phone greater access to customers in terms of quality and quantity, and without Doubt will contribute to increasing sales and this is what companies of a commercial nature seek. Our platform is an exploitation of the scientific knowledge available in the field of informatics within the framework of digitizing trade and enhancing the presence of technology in daily transactions that would facilitate life and move from traditional methods to more modern methods commensurate with what the outside world has reached compared to our situation in Algeria .

1.3 The MEMORY OVERVIEW

2 the project introduction

2.1 Introduction

Our platform in particular, it's an orders management system, so in this chapter we will know what the "orders management system" is, and we will also discuss the goals that we are working to achieve through this work. We are interested in studying the existing criticisms, and suggest possible solutions. Our mission is to create a system that consists of two parts, a client part, which is a mobile application in the Dart language and flutter framework and also Firebase, and there is a server part, which is only available to a store manager who manages orders, the server is a dynamique website that was created Using CSS, HTML and ReactJS and Node Js frameworks , and managing databases using SQL, the role of this platform is to display products for sale via the mobile application, so that the order is managed through the store's manager through the website , the technology of order status tracking and automatic locator technology are available to facilitate the delivery of the ordered items, This platform also provides the ability to open a personal account .

2.2 Project objectives

The objective of the project is to develop a platform that will allow perform the following operations:

- make The ability to order food from the restaurant from anywhere
- organize food delivery operations.
- Manage clients orders.
- show the restaurant products .

2.2.1 the existence study

So that restaurants can solve the problems they may face, including the problems of limited customers and lack of sales, they are forced to choose a suitable place where business is thriving and where people are interesting with food, or they are opening several branches in different places to expand business, which will cost them a lot of money and expenses. (Higher workplace expenses, equipment costs, higher wages for workers, higher taxes) It would run the risk of going bankrupt without making the required profit margin, and on the other hand the customer would have to go personally to the place of the restaurant to order food, which would waste time and effort and might find nothing to look for. Now in our time, there are solutions to this crisis with minimal losses, especially with the presence of the Internet and informations technology, whose exploitation in our daily lives is necessary to facilitate many of the difficulties we faced in the past.

2.2.2 the critique of the existence

The current manual solution is traditional, thus posing different problems, namely:

- It takes a long time to reach the desired goal .
- It requires more effort with the risk of bankruptcy
- Requires a high budget and workers
- A solution that is not in line with the current development of trade methods with the spread of digitization and technology
- A traditional and imperfect solution to the problem at hand, as there are many factors that will directly affect this method

2.2.3 suggested solution

Through information technology, new commerce and service fields development perspectives have emerged, and the creation of a restaurant order management application aims to develop and digitize e-commerce. Using the Internet as a means of dealing in the services and commercial sectors can reduce costs, improve performance and make daily life easier. This project relies on the creation of a customer order management system.

2.2.4 Client Section

By creating a mobile application that is publicly available through which shops display their products and services and allow product demand in the presence of the Internet. It also determines the geographic location of the customer to assist in the delivery process. It also requires the opening of a personal account to create a client's profile to provide greater protection and ease to the customer.

2.2.5 Administrator section

The system also contains a section for store manager , which is a web site that allows for the control of the products offered, the reception and examination of customers' orders and the identification of the customer. the system is Available in offline mode

Part II

Project Methodologies

3 Agile Development

3.1 Continuous Integration

Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration most often refers to the build or integration stage of the software release process and entails both an automation component (e.g. a CI or build service) and a cultural component (e.g. learning to integrate frequently). The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

3.2 GitHub

GitHub is a Version control system that records changes to a file or set of files over time so that you can recall specific versions later. It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.

3.3 Code Quality

Single-Responsibility Principle : A class should have one and only one reason to change, meaning that a class should have only one job.

Open-Closed Principle : Objects or entities should be open for extension but closed for modification.

Liskov Substitution Principle : Let $q(x)$ be a property provable about objects of x of type T . Then $q(y)$ should be provable for objects y of type S where S is a subtype of T .

Interface segregation principle : A client should never be forced to implement an interface that it doesn't use, or clients shouldn't be forced to depend on methods they do not use.

Dependency Inversion Principle : Entities must depend on abstractions, not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions.

3.4 Test Driven Development

Test Driven Development is an approach that allow developers to deploy software features without the fear of breaking existing code which promote teams continous integration cycles.It also enhance softwares lifetimes and development costs on the long run.

How to write Tests Not all the code is created equally, only test the critical parts that contribute to the software sucess . Write tests before production code to ensure software testibility and further enhance code quality . Name tests using their side effects,expected result or the state they leave the system in.

Why write Tests Softwares that has long lifetimes tend to grow exponentially in both complexity and developements costs as such TDD help us break down the complexity by providing live documentations in the form of tests . They also work as a safety net to help future updates and maintenance .

When not to write Tests Tests are not always the right answer as they slow down early development and don't bring much to small,simple,short lived projects .

4 Project Architecture

4.1 Analysis Models

An analysis model is a byproduct of defining conceptual models that capture system behaviors,rules and information flows.Which help identifying the key requirements and obstacles of buidling the software.

Key Points

- Distiling the system into subsystems and capturing the relations between each.
- Capturing the system core components ,events and ressources flows(REA).

4.2 Domain Driven Design

Domain Driven Design is an approach enabling teams to effectively manage construction and maintenance of complex domains software by seperating technical concerns from buisness logic and promoting agile enviroments.

Key Points

- Architecting the software into Ui,Application,Domain Infrastructure.Each layer would be further architected into bounded contexts.
- Setting up knowledge-crunching sessions with domain experts whenever possible to develop a common ubiquitous language.
- Identifying core and subdomains to better spend ressources on what really matters.

4.3 Design Patterns

Design patterns are typical solutions to commonly occurring problems in software design. They are like pre-made blue-prints that you can customize to solve a recurring design problem in your code. **Observer Pattern**
Singleton Pattern

5 Technologies

5.1 TypeScript

A typed superset language that compile to javascript, it supports both commonJs and the latest release of EcmaScript formats.

Why use it It reduces production errors by providing typing and auto-complete features. Enables writing better and cleaner code as it introduces interfaces, classes, inheritance and access modifiers.

Drawbacks Typescript is compiled to vanilla javascript before executing in the browser as such it requires a headless browser or pre-building its modules. Sometimes the compiled javascript is inefficient and ugly which might produce some performance bottlenecks.

5.2 Sql

Structured Query Language (SQL) is a programming language that is typically used in relational database or data stream management systems.

5.3 Dart

Dart is a client-optimized cross platform language, it adopts a declarative paradigm which solves a lot of common problems in imperative languages. Dart is optimised for building UI as it provides hot reloading features.

6 TechStack

6.1 Flutter

A cross-platform framework that enables developers to build visually enhanced apps, the framework uses Dart as its underlying language and promotes a declarative programming paradigm. Projects built by Flutter share most of their codebase throughout the different platforms and provide communication channels to access the platform-specific features.

Drawback : The framework is fairly new and lacks some crucial packages.

6.2 NodeJs

A javascript framework that enable building fast,responsive,scalable backend servers.However it's single threaded which makes it incompatable with intensive/cpu blocking operations.

6.3 React

A javascript framework that enables buidling single page applications,it also enhances websites performances since it render on the client side instead of on the server.

Drawbacks: Since single page applications are rendered and cached on the client browser it produces some security overheads since it exposes all of the logic to the user. The concern is easily dealt with by refactoring the backend logic into a different server and exposing said functionality through an api.

6.4 Firebase

Firebase is a platform and a framework developed by google that provide a range variety of services including authentication,hosting,cloud storage ,realtime database,firestore,functions,analytics and cloud computing.

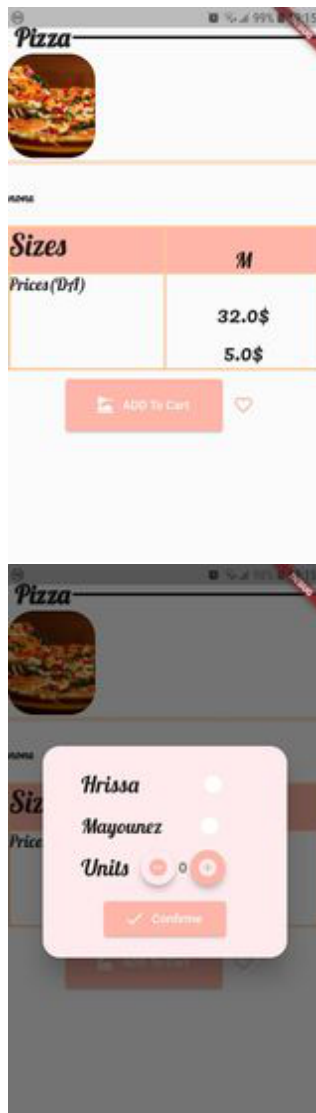
Part III

Project Implementation

7 Mobile client app

7.1 UI

7.1.1 Product



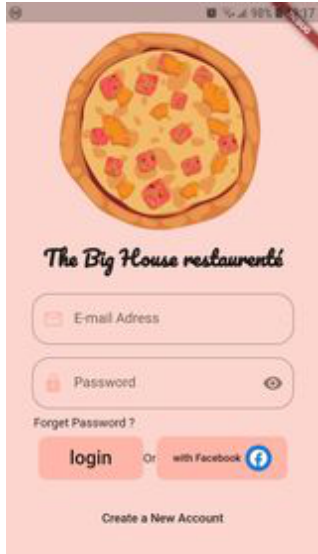
7.1.2 Catalogue



7.1.3 Cart



7.1.4 Login



7.1.5 Gps Location



7.1.6 Registration

"bienvenu dans notre restaurant , nous sommes tres heureux de votre visite a notre service alimentaire"

créé votre new compte , Now !!

premise

nom

l'adresse Email

number phone

...

Sign up


7.2 Application


7.2.1 Providers


C HelpersProvider
<ul style="list-style-type: none"> CatalogueModel catalogueModel CartHelper cartHelper ServicesProvider services ProfileHelper profileHelper
<ul style="list-style-type: none"> HelpersProvider() Future<bool> initApp() Category getCategory(int categoryIndex) int getCategoriesCount() CartHelper getCartHelper() void setUpHelpersContext(BuildContext context) ProfileHelper getProfileHelper() AuthenticationHelper getAuthHelper() DeliveryAddress getAddressHelper()


C NavigationProvider
<ul style="list-style-type: none"> final List<Widget> screens int screenIndex
<ul style="list-style-type: none"> NavigationProvider() Widget getScreen() void navigateToCart() void navigateToCatalogue() void navigateToCategoryproductsScreen() void navigateToProfile() void navigateToSettings() void navigateToLogin(BuildContext context) void navigateToDeliveryAddressScreen(BuildContext context) void navigateToNewAccount(BuildContext context) int getScreenIndex()

7.2.2 Helpers

 AuthenticationHelper
<ul style="list-style-type: none"> □ IAuthenticationService authService □ FacebookAuthentication fbAuthentication □ BuildContext context
<ul style="list-style-type: none"> ● AuthenticationHelper(authService, fbAuthentication) ● void setBuildContext(BuildContext context) ● void signInWithEmailAndPassword(String email, String password) ● void signUpWithEmailAndPassword(String email, String password) ● void signInWithFacebook() ● void signUpWithFacebook() ● void requestPhoneValidation(String phone) ● void resetPassword(String email) ● void sendConfirmationEmail(String email) ● void signOut() ■ void popUpRegisteredBox() ■ void popUpErrorBox(Error error) ■ void setUpNewUserProfile(ProfileHelper helper, String fullName, String phone, String email)

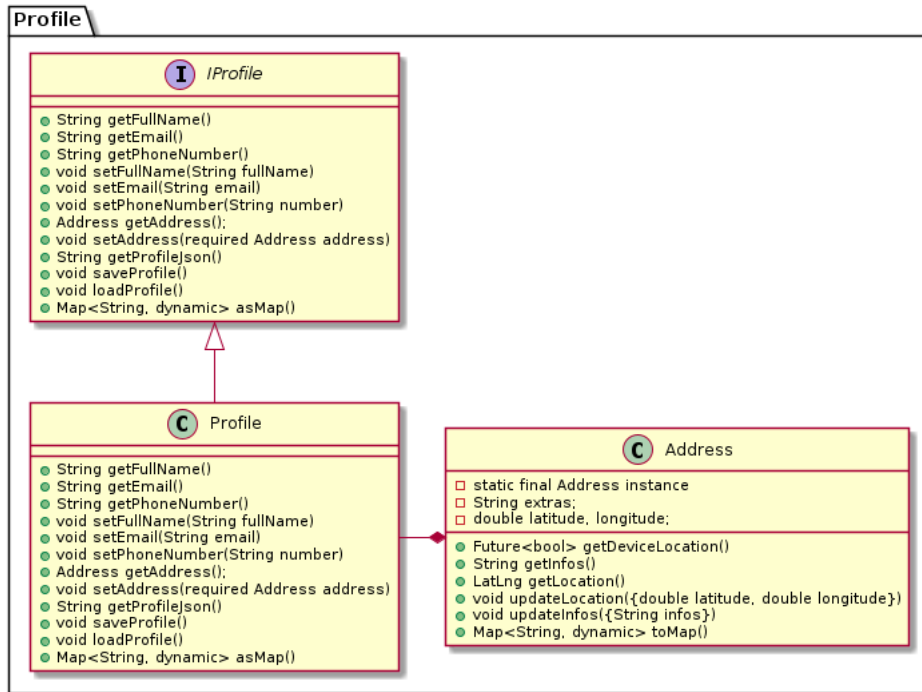
 CartHelper
<ul style="list-style-type: none"> □ Cart cart □ IOrderService orderService □ IAuthenticationService authenticationService
<ul style="list-style-type: none"> ● CartHelper(Cart cart, IOrderService orderService, IAuthenticationService authenticationService) ● void addCartItem(CartItem cartitem) ● int getCartItemCount() ● CartItem getProduct(int productId) ● void placeOrder(BuildContext context) ● void removeProduct(CartItem item) ● void sendOrderToShop(bool isActive, NavigationProvider navigationProvider, BuildContext context)

 ProfileHelper
<ul style="list-style-type: none"> □ final IAuthenticationService authenticationService □ final ProfileModel profile □ final ICustomerDataSynchroniser dataSynchroniser
<p>ProfileHelper(IAuthenticationService authenticationService, ProfileModel profile, ICustomerDataSynchroniser dataSynchroniser)</p> <ul style="list-style-type: none"> ● setDeliveryAdresse(BuildContext context) ● void isLoggedIn(BuildContext context) ● void setProfile(String fullName, String phone, String email) ● void updateProfile() ● void registerProfile(String id)

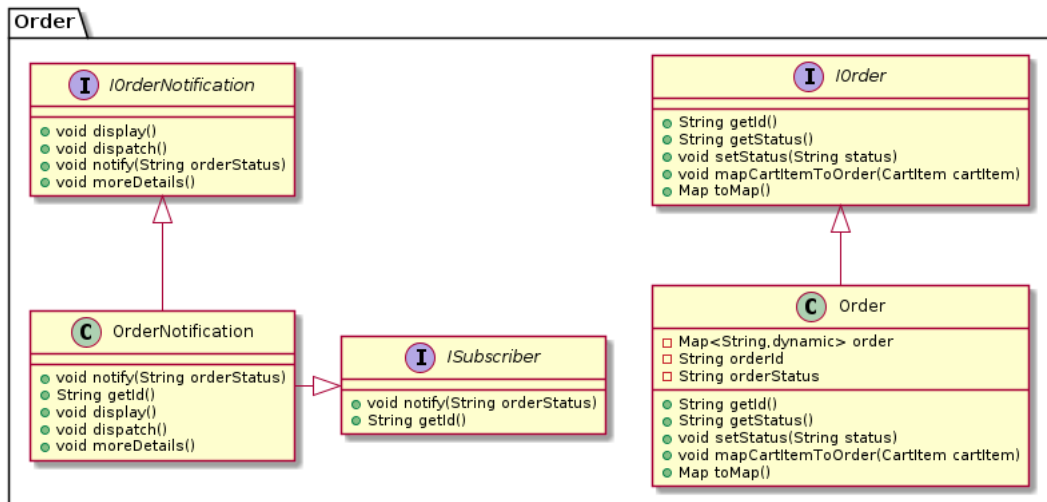
 DeliveryAddress
<ul style="list-style-type: none"> □ Address _address
<ul style="list-style-type: none"> ● Future<bool> initGpsLocation() ● LatLng getLocation() ● void setLocation(LatLng coordinations, String infos) ■ Future<void> requestLocationPermission()

7.3 Domain

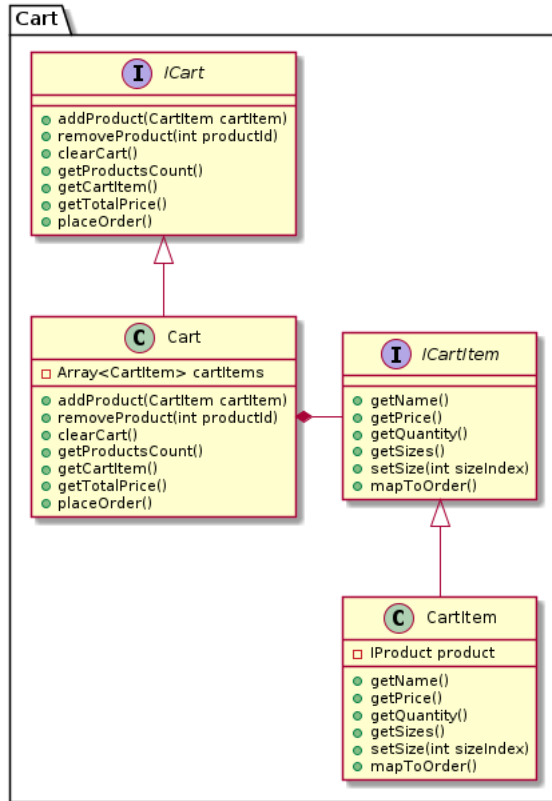
7.3.1 Profile



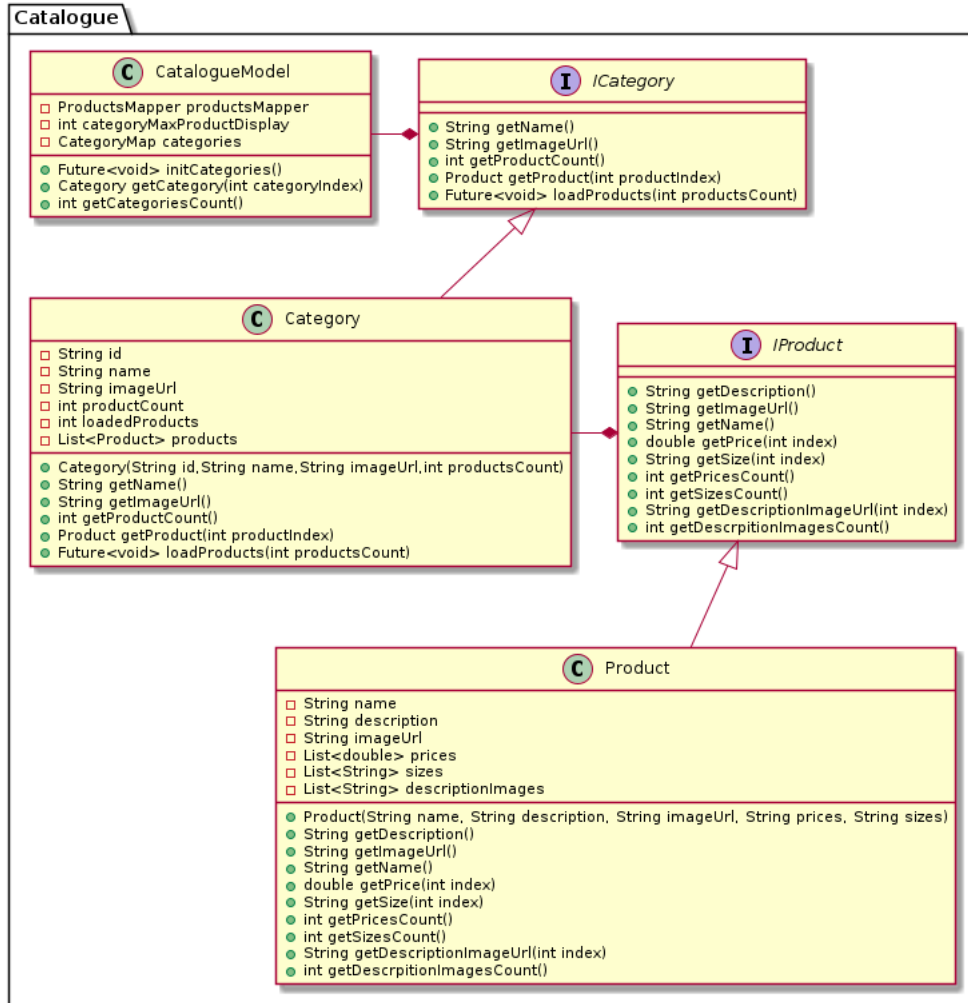
7.3.2 Order



7.3.3 Cart

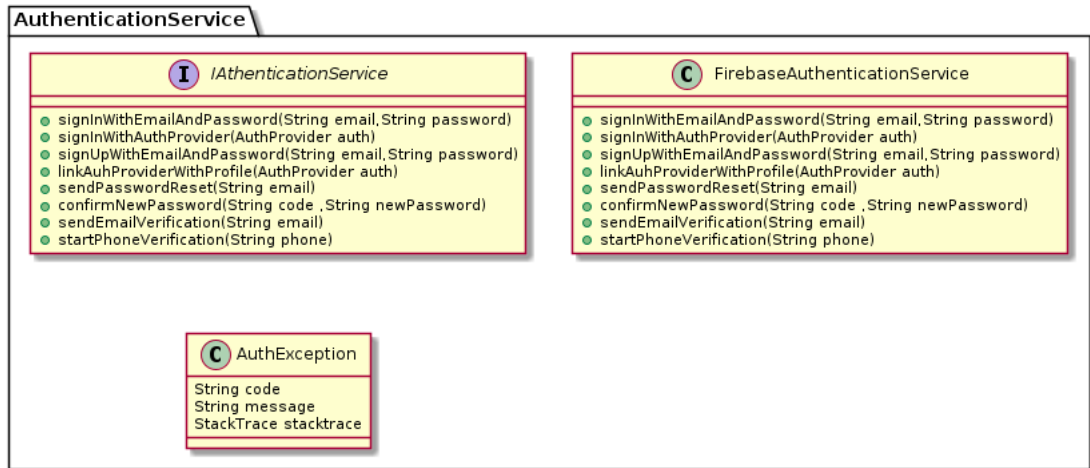


7.3.4 Catalogue

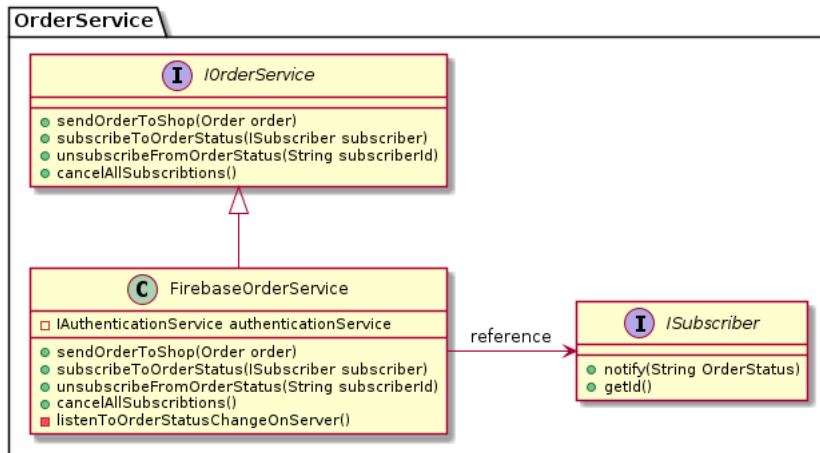


7.4 Infrastructure

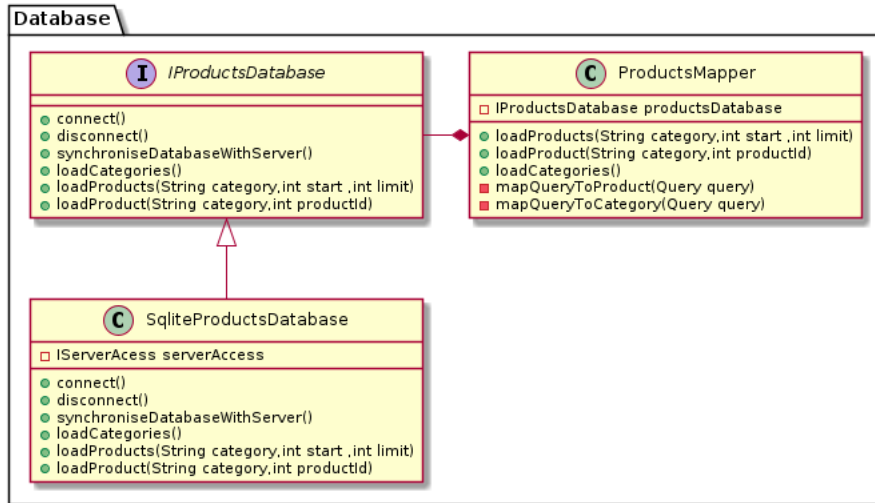
7.4.1 Authentication



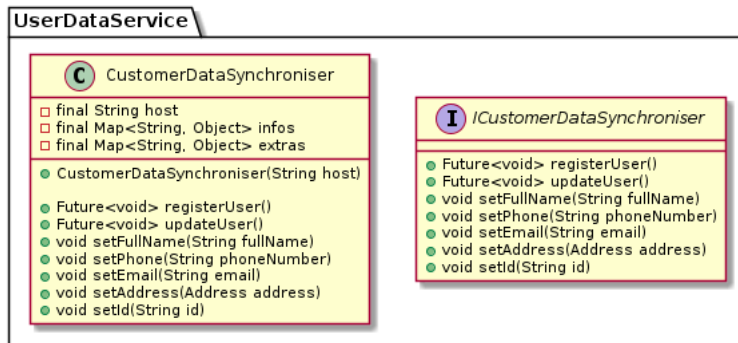
7.4.2 Orders



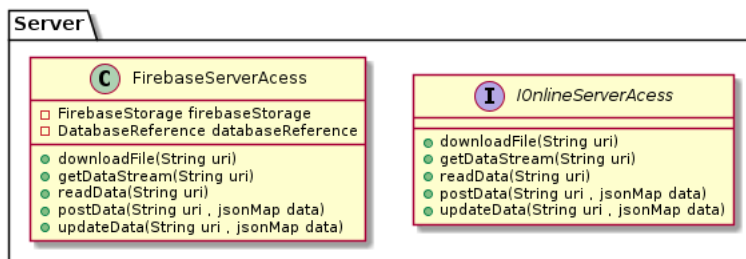
7.4.3 Database



7.4.4 UserData



7.4.5 ServerAccess



7.4.6 Exceptions Handling

Exceptions are encapsulated into an app specific exceptions , in which they indepenandant of the underlying technologies used.

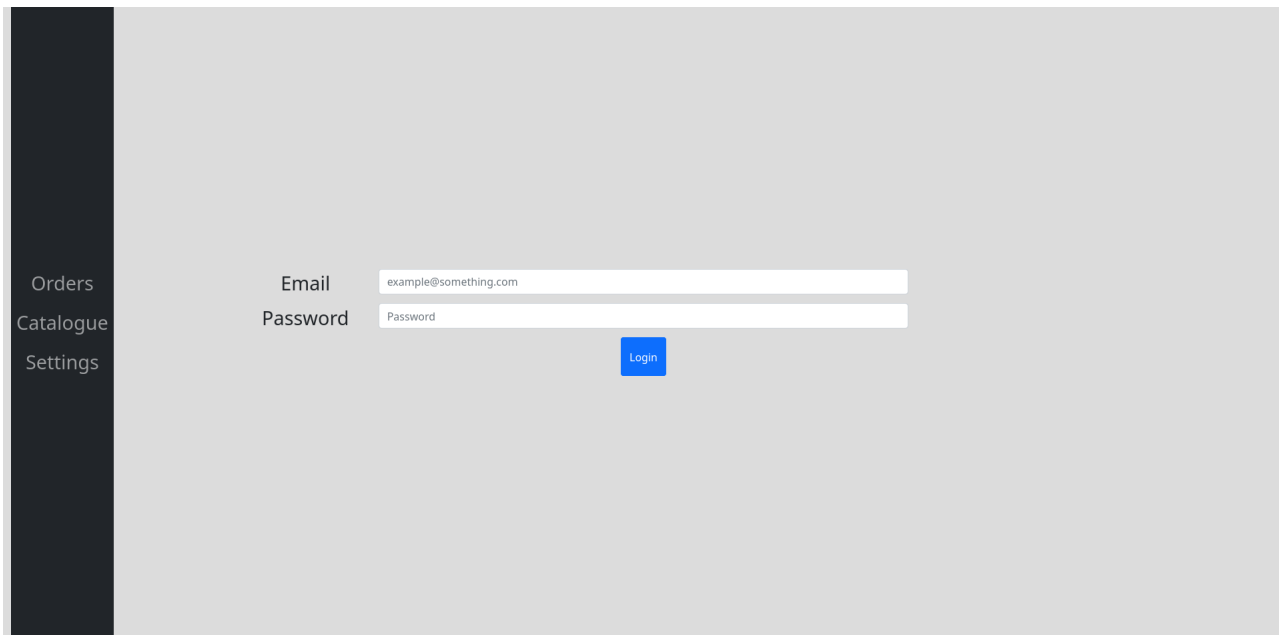
Exceptions codes :

- invalid-login
- invalid-code
- email-used
- user-not-exist
- invalid-email
- invalid-password
- invalid-phone-number
- account-already-linked
- network-error
- order-not-sent
- order-not-found
- local-database-not-found
- remote-database-not-found

8 Shop react app

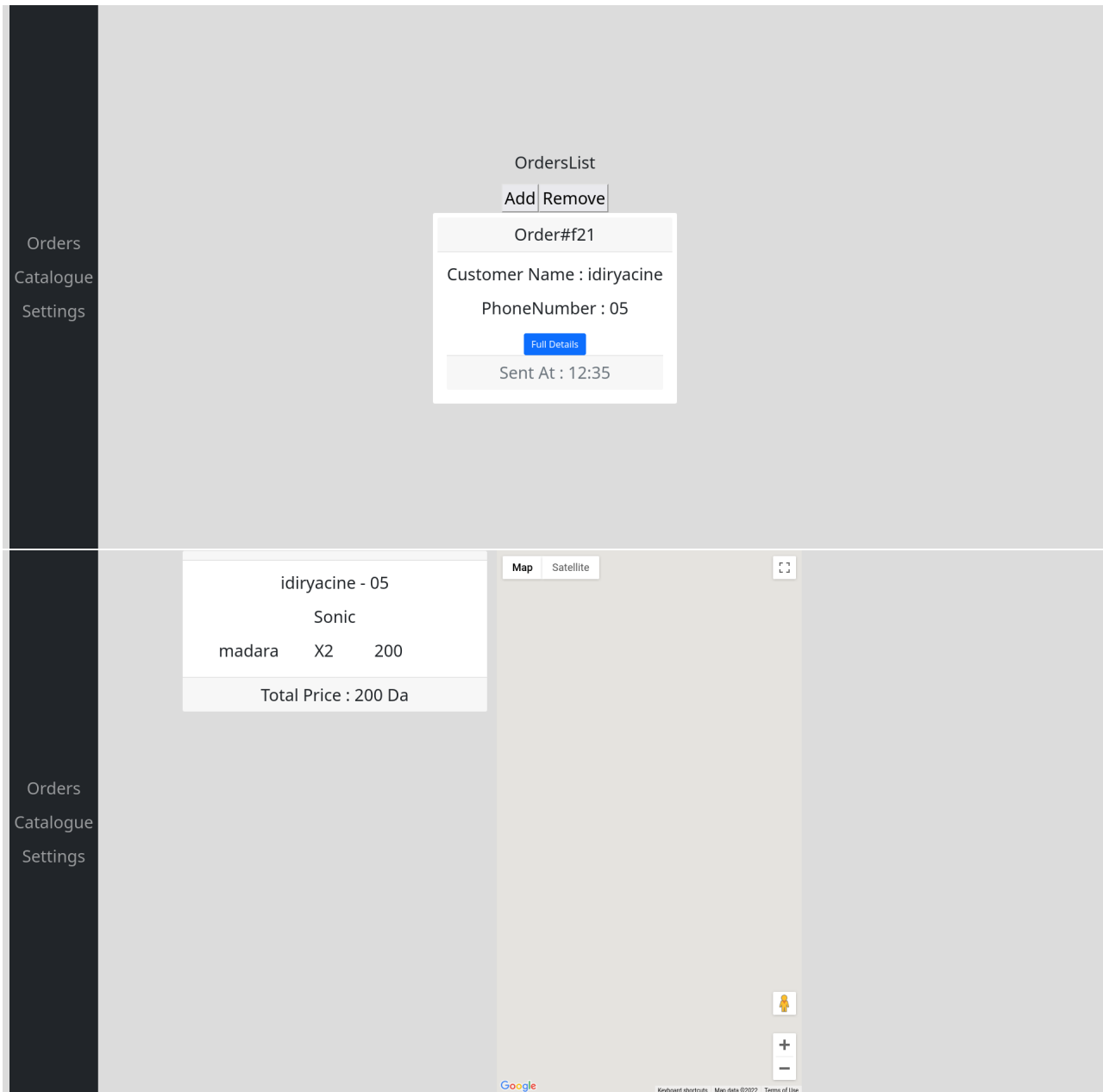
8.1 UI

8.1.1 Login



The screenshot displays a login interface for a web application. On the left, a dark vertical sidebar contains three menu items: "Orders", "Catalogue", and "Settings". The main content area has a light gray background. It features two input fields: the top one is labeled "Email" and contains the text "example@something.com"; the bottom one is labeled "Password" and contains the text "Password". Below these fields is a blue rectangular button with the word "Login" in white text.


8.1.2 Orders Board



8.1.3 Category Manager

Orders
Catalogue
Settings


New CategoryLoad Categories



test

DeleteEditExplore

Orders
Catalogue
Settings



https://static.remove.bg/remove-bg-web/a6eefcd21dff1bbc24

Name

Name

Description

Description

Save

Cancel

24

8.1.4 Product Manager


Orders

Catalogue

Settings

New ProductLoad Product

madara



Default Size : sdd

Price : 32.77

EditDelete

Orders

Catalogue

Settings

Image Url

Name

Description

Price

Size

Add Size

Remove

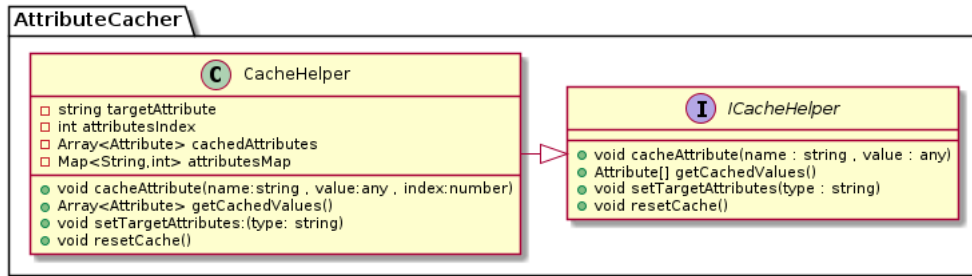
Save

8.1.5 Settings

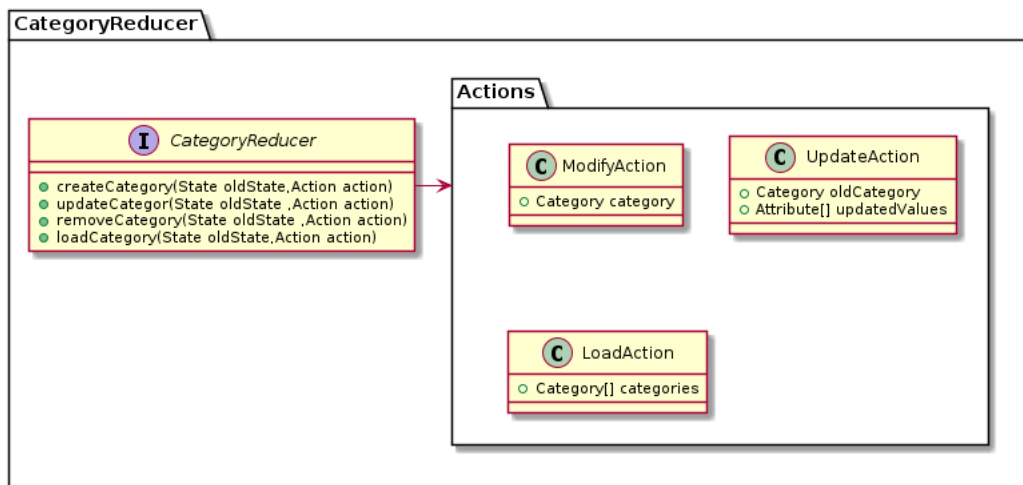
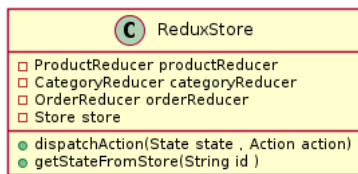


8.2 Application

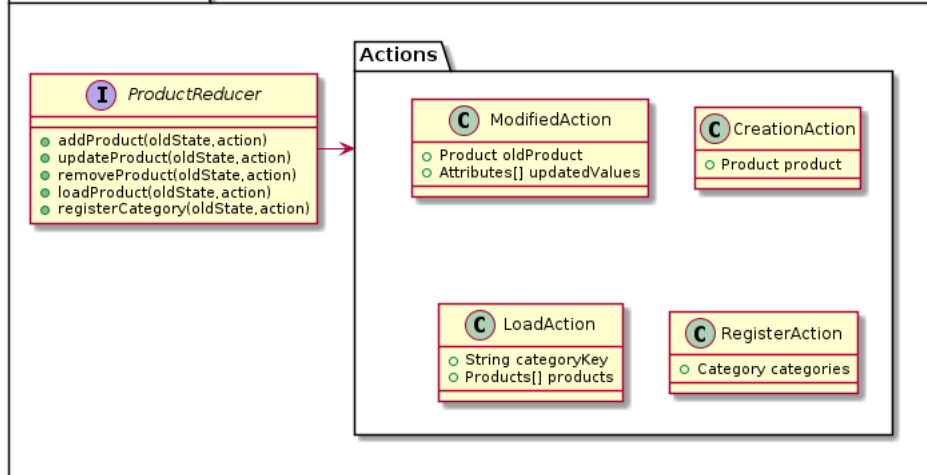
8.2.1 Attribute Cacher



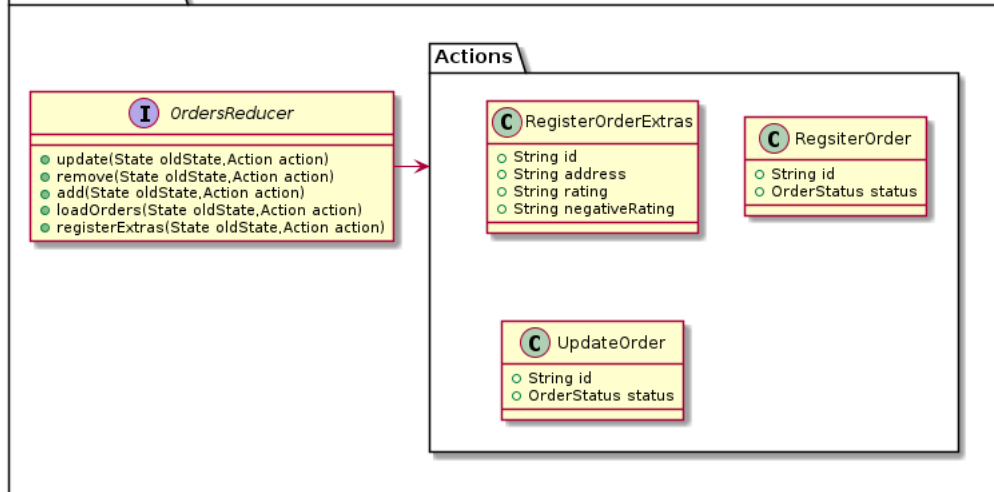
8.2.2 Redux Store



CategoryReducer



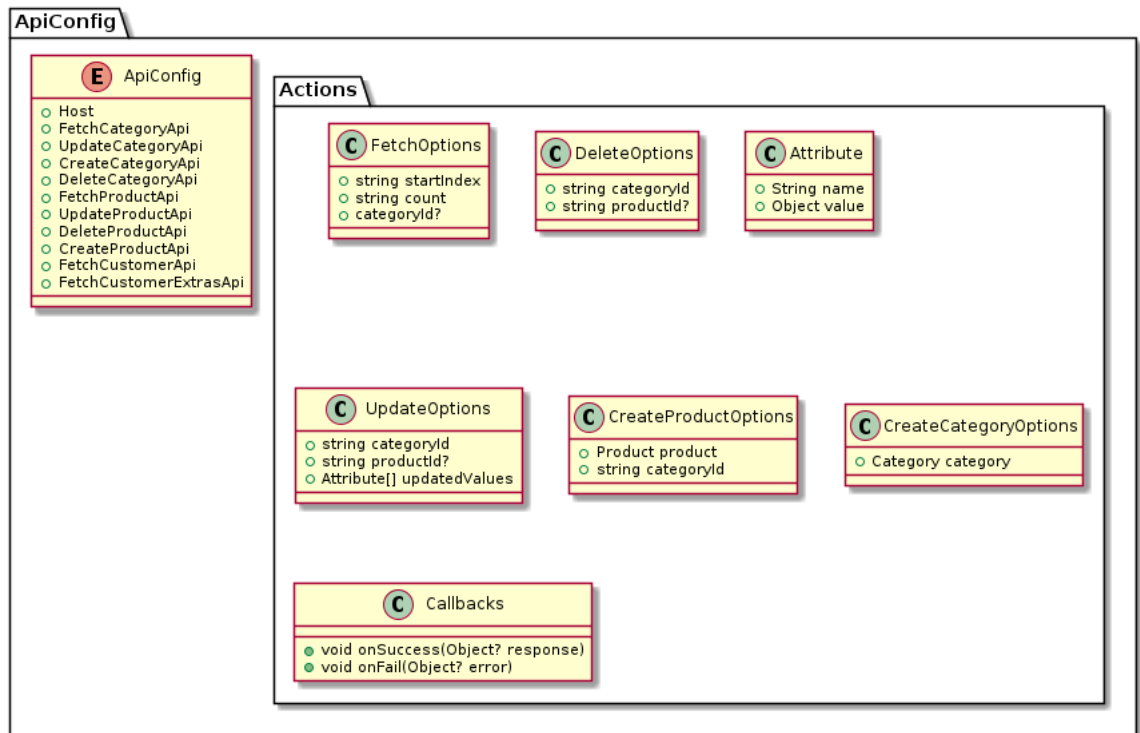
OrdersReducer



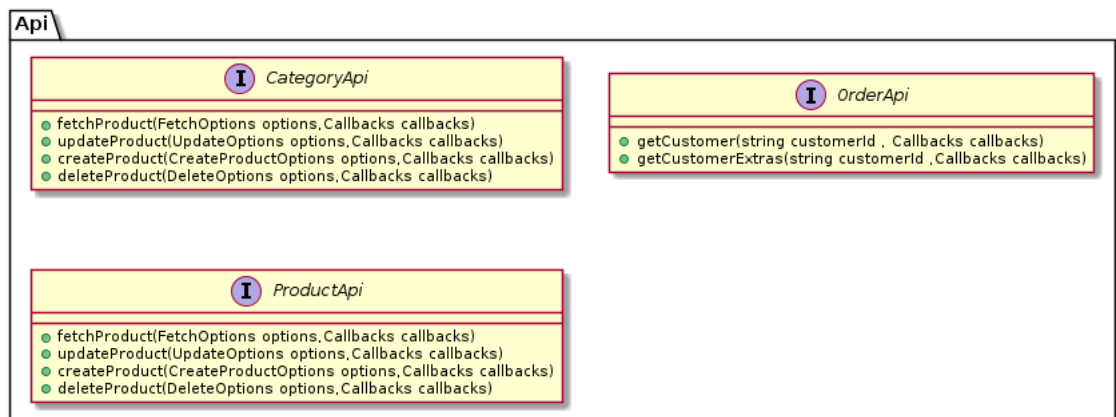
8.3 Domain

8.4 Infrastructure

8.4.1 ApiConfig

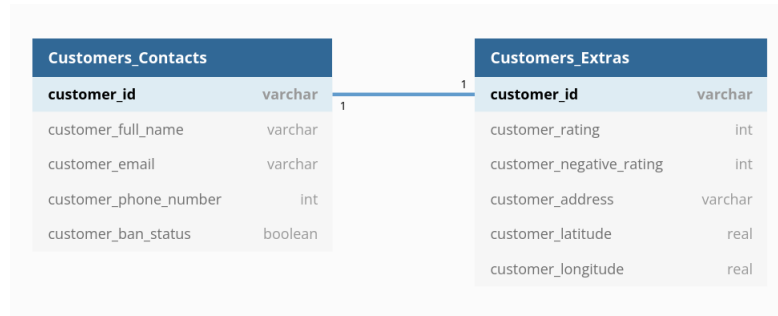


8.4.2 Api



9 Shop node app

9.1 Customer Database



9.1.1 Ban mechanism

In order to avoid bad customers a **ban mechanism** was placed where the owner would be informed of a customer **behavior score** and manage a blacklist to filter out his orders .

The ban mechanism uses a phone **IMEI number** to prevent a spammer from creating new accounts when banned .

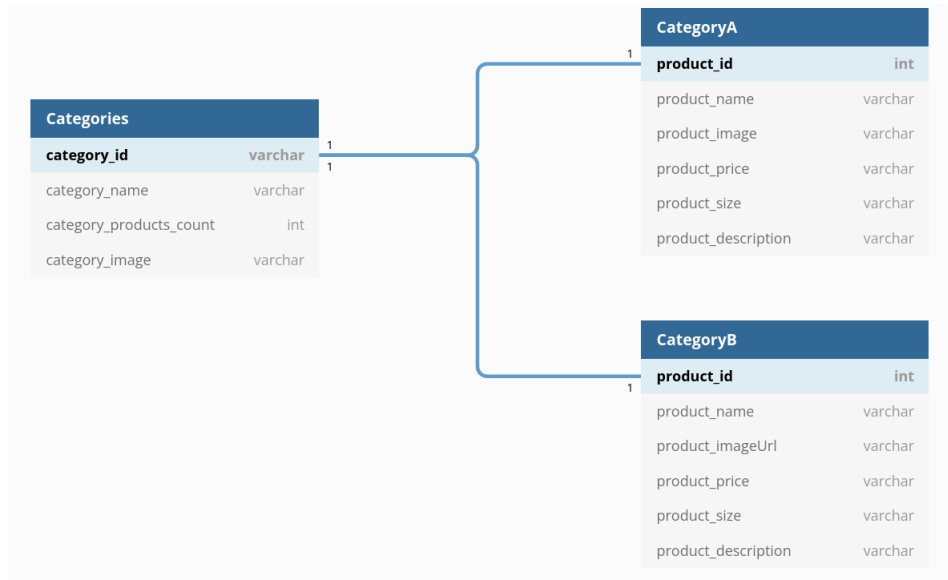
9.1.2 Optimising requests

Embedding delivery data in each order can introduce a **data usage overhead** thus to reduce it a customer delivery data are cached in the database. In which enables the server of retrieving it using only the **customer id** included in the order.

9.1.3 Explaining some keys

Rating represents the number of approved orders without issues while **Negative Rating** represents orders that weren't delivered in cause of a client fault or issue . This separation is important in order to make a ban decision later on.

9.2 Product Database



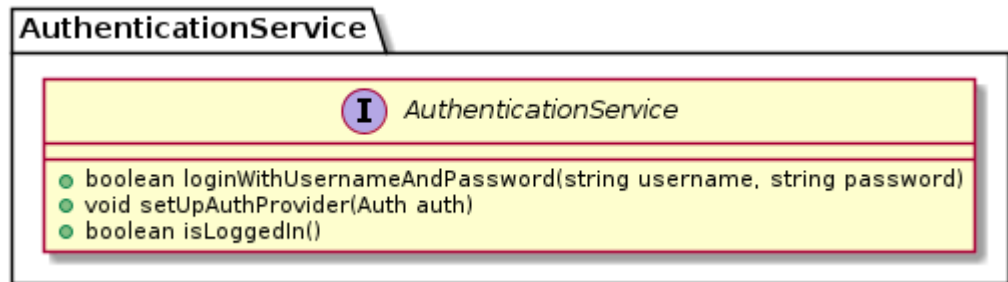
9.2.1 Explaining some keys

Categories Table holds records of categories metadata where each category-entry would have its own **Category Table** that contains its products. Thus each **category id** represents a table name in the database.

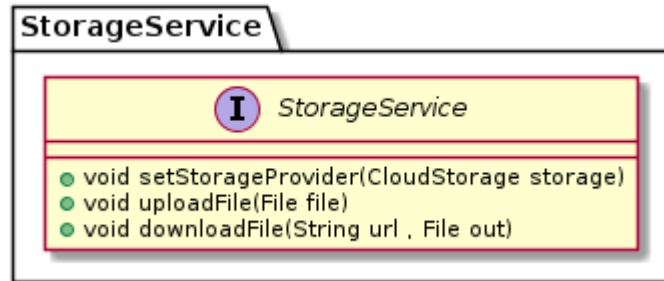
Product Count is important since it's used on the app product loader.

9.3 Infrastructure

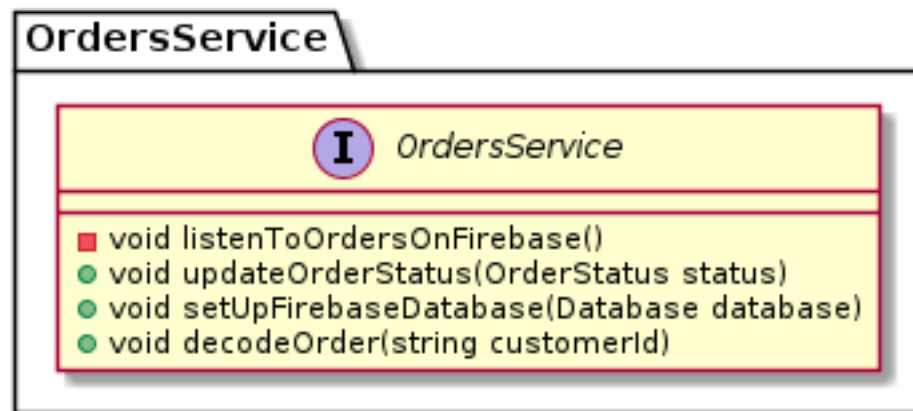
9.3.1 AuthenticationService



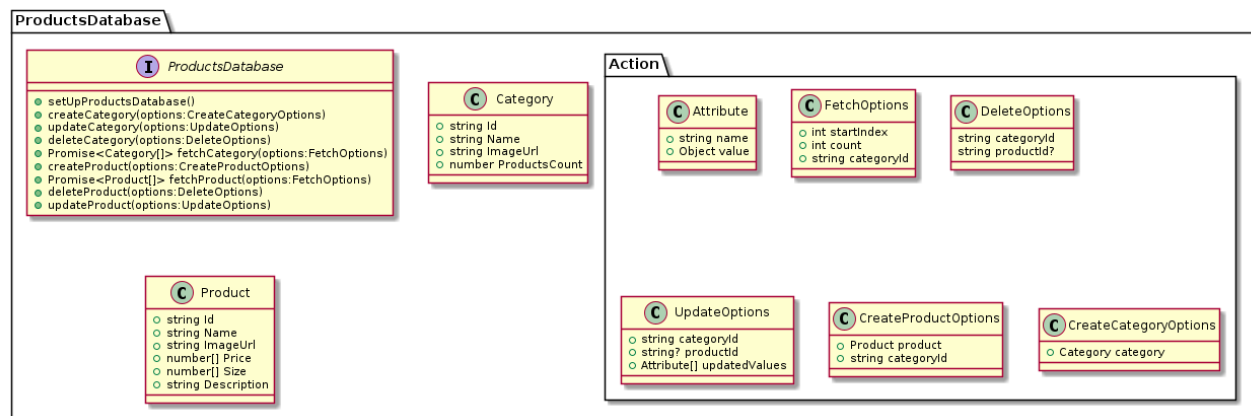
9.3.2 StorageService



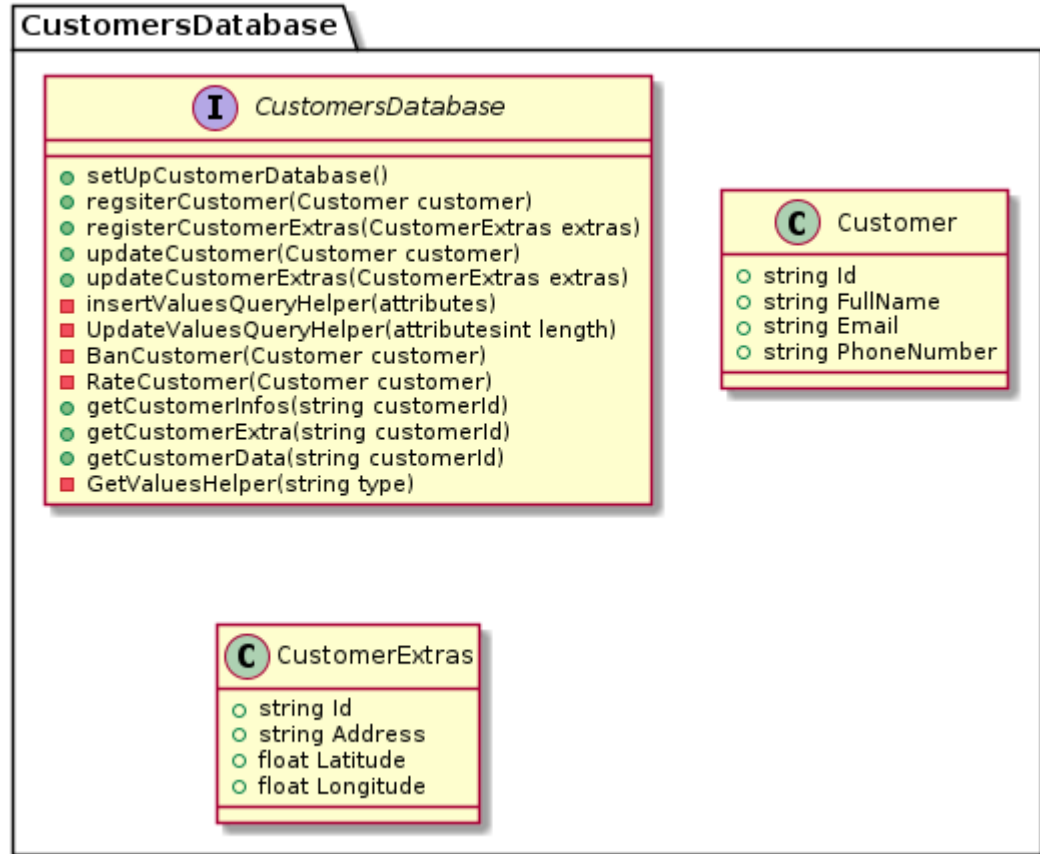
9.3.3 OrdersService



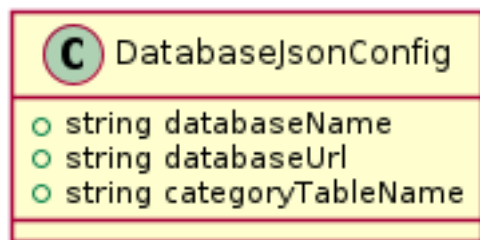
9.3.4 ProductsDatabase



9.3.5 CustomersDatabase



9.3.6 Configs



10 Installation

10.1 Dependencies

Install **Flutter** from the official website [here](#)

Install **Node package manager** from the official website [here](#)

10.2 Mobile App

Clone the project from github :

```
git clone https://github.com/IDIRYACINE/online_order_client
```

Navigate to the project directory and run flutter : **flutter run**

10.3 Node App

Clone The project from github :

```
git clone https://github.com/IDIRYACINE/online_order_server_nodeJs
```

Navigate to the project directory and run : npm start

10.4 React App

Install React by running : npm install react

Clone The project from github :

```
git clone https://github.com/IDIRYACINE/online_order_server_app
```

Navigate to the project directory and run : npm start

Part IV

Appendix

Test Driven Development

Beck Kent - Test-Driven Development

Code Quality

Robert C. Martin - Clean Code

Martin Fowler - Refactoring: Improving the Design of Existing Code

Domain Driven Development

Eric Evans - Domain-Driven Design: Tackling Complexity in the Heart of Software

Project Architecture

Mark Richards - Software Architecture Patterns

Design Patterns

Gang of Four Design Patterns

Alexander Shvets - Dive Into Design Patterns

Analysis Model

Martin Fowler - Analysis Patterns Reusable Object Models

GitHub

Scott Chacon - Pro Git