# Groceries delivery

Group H

Anna Hoang
Jakub Jasiński
Mikołaj Kornaś
Piotr Rogulski
Jan Szablanowski

March 30, 2022

## Contents

# 1 Introduction

## 1.1 Aims & Purpose

The purpose of the system is to provide a web application for grocery shopping and delivery. It allows the client to browse available products and place orders on them. They can also create sets of favorite products in order to facilitate future browsing. When the order is placed, the client can view the status of the order and communicate with the courier. The system also includes an interface for shop employees to prepare and dispatch orders.

## 1.2 Modules

The system comprises three modules:

1. Client
2. Delivery
3. Shop

Each module takes part in the process of fulfilling an order. To describe each module's functionality, let us analyze the use cases of the system.

## 1.3 Architecture

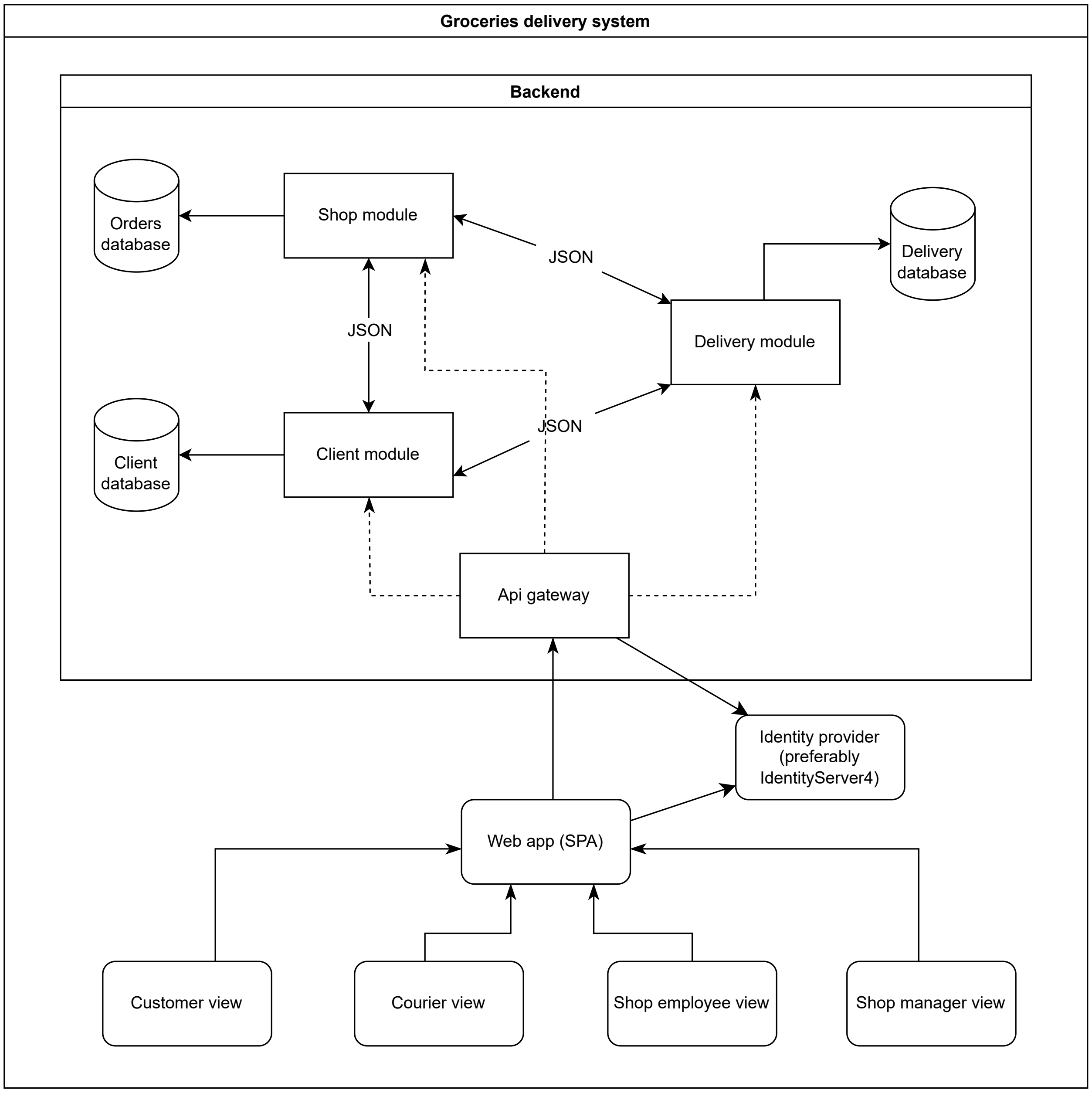
As shown in the next diagram, system has typical multilayered structure with backend, web app and identity provider. Backend consist of three independent modules (each with its own database). Modules communicate with each other using REST protocol (see Attachments section).

Furthermore, communication between backend and frontend runs through Api gateway, which is single entry point for the web application. Api gateway connects also with Identity provider (e.g. IdentityServer4, Google), that is the source of tokens needed to authenticate to the backend.

Web app has four basic views for customer, courier, shop employee and shop manager.

- customer view - browsing product offer and placing orders
- courier view - managing delivery process
- shop employee view - managing order preparation process
- shop manager view - managing product offer

Figure 1: High level system structure.



## 1.4 Technological stack

The backend consists of three independent modules, that implement business logic, one api gateway and three databases. Each module and api gateway should be implemented in a technology, that allows creating REST apis (e.g ASP.NET, Node). Databases should be relational (e.g. SQL Server, PostreSQL).

Web client app should be a Single Page Application (e.g. React, Angular), which implements OAuth 2.0 flow.

The whole system, including databases and client apps should be deployed to the cloud (e.g. Azure) and ideally use some kind of containerization (Docker).

## 1.5 Authorization

Application should use OAuth 2.0 end user authentication (preferably IdentityServer4 with username and password). The auth flow should be Authorization Code Flow with PKCE, because our web app is a Single-Page-Application (most likely written in React or Angular) and secrets can't be used in the source code.

## 1.6 User Stories

In total, there are five actors in the system: the client, the courier, the shop employee, the shop manager (which is also a shop employee) and the algorithm. The system provides each actor with functionalities included in Table 1. Note that the algorithm doesn't have an explicit user interface and, as such, shall be effectively transparent to other users.

| **As a...** | **I want to...** | **So that...** | **MoSCoW** |
|---|---|---|---|
| Client | create an account | I can place orders | must |
| | browse products | I know what to buy | must |
| | make an order | I can inform the shop what I want | must |
| | add favorite sets | I can speed up future shopping | could |
| | choose time of delivery | I can easily collect it | should |
| Courier | register an account | I can work | must |
| | declare my availability | I can work when I can | must |
| | accept orders | I can collect and deliver them | must |
| | send messages to the client | I can communicate with them | should |
| | deliver the order | I can fulfill the client's request | must |
| | query the shop for new orders | I can choose an order to deliver | could |
| | confirm goods received | I can mark the job as finished | must |
| Shop employee | see products ordered by a client | I can complete orders | must |
| | know how to mark orders | I can pack orders | should |
| | change order's status | I can prepare orders and call couriers | must |
| Shop manager | manage the list of products | I can update available products | must |
| | check couriers' availability | I can make a schedule | must |
| | see history of orders | I can generate reports | could |
| Algorithm | I can assign couriers to orders | I can minimize delivery time | could |
| | I can assign shops to orders | | could |
| | analyze couriers' position | | could |

Table 1: User stories

## 1.7 Use Cases

### 1.7.1 Client

The main client functionality is placing orders. It includes:

- viewing products
- adding and removing products from cart
- applying coupons to get discount
- choosing the payment method
- sending messages to courier
- making complaints when their requirements are not fulfilled

Client can also create the account and login so they can list their previous orders, track the current ones and save their contact detail and address data. They can also register loyalty card.

### 1.7.2 Courier

Courier can register an account and login. When logged he can deliver packages, that consists of:

1. querying shop for pending orders
2. accepting orders
3. picking packages from the shop
4. sending messages to client to inform them about package status updates (such as delays)
5. notifying client when package is ready to collect
6. confirming package received

Courier also declare availability to inform shop when he can work.

### 1.7.3 Shop employee

The main role of shop employee is preparing orders. It consists of:

1. accepting or rejecting the order placed by client
2. packing products and addressing package
3. marking when the order is ready to pick
4. notifying courier

Employee can also reports products shortage

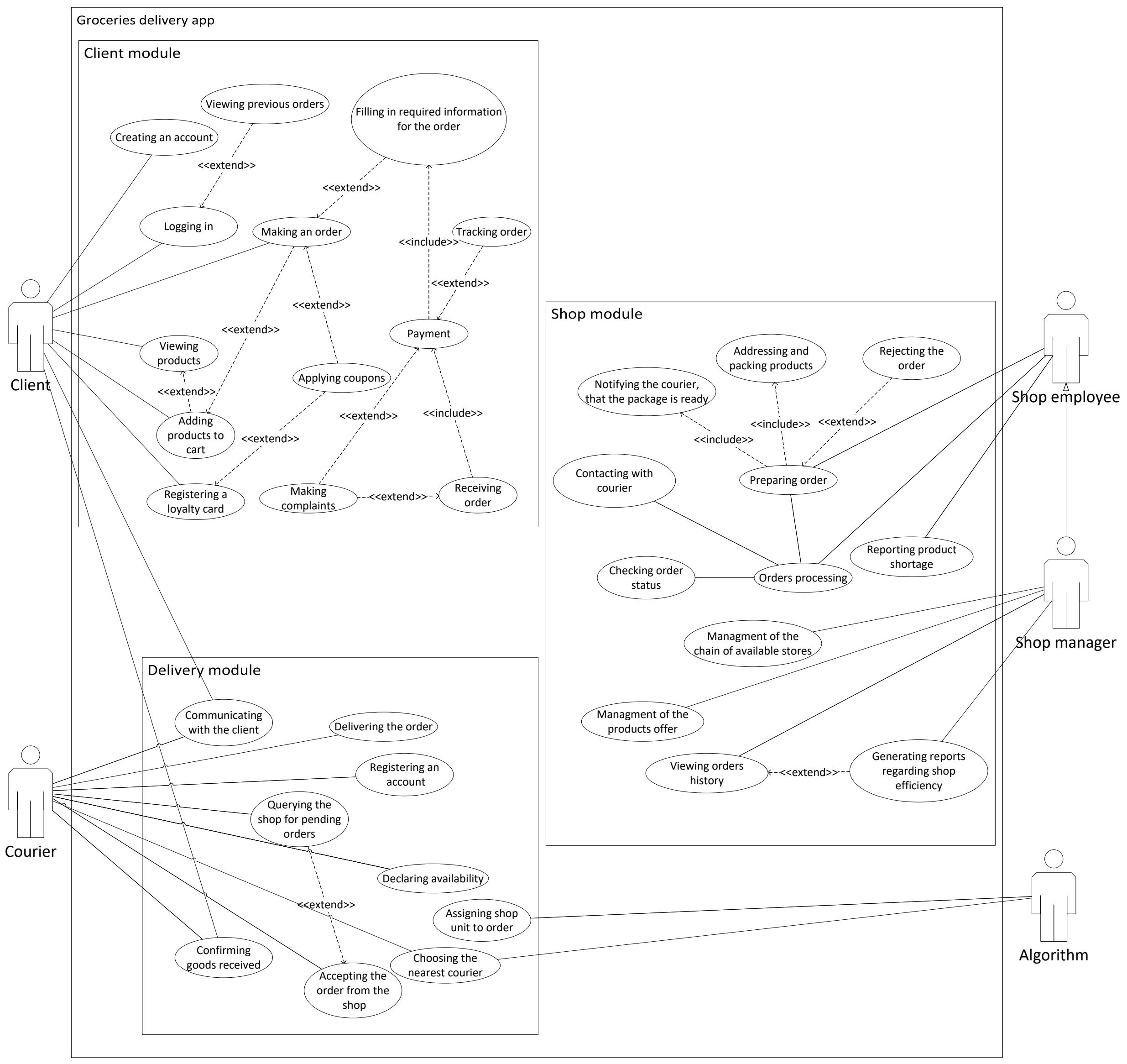
### 1.7.4 Shop manager

Shop manager extends shop employee functionality. He also manages the list of offered products and chain of available stores. He can also view orders history and generate reports regarding shop efficiency.

### 1.7.5 Algorithm

Algorithm is responsible for assigning shop to order and courier to order. It should work in a way to minimize time difference between desired and predicted time of delivery and minimize courier waiting time.

Figure 2: Use case diagram.



# 2 System Specification

## 2.1 Key Functionalities

Main function of the app is to allow users to make grocery products orders from a given supermarket chain (like Carrefour). Orders are then delivered to clients by couriers.

### 2.1.1 Application users

Application has following types of users:

- clients – they can log into web app and place an order
- couriers – they typically use app as a mobile web page to correctly deliver the order
- shop employees – they use web app to check order items, change order status and contacting with the courier
- shop managers – they use web app to manage chain of shop facilities and product offer

### 2.1.2 Application flow

1. Client logs into the web app and places an order.
2. Order is put into the system as pending order.
3. One of the shops accepts the order.
4. Shop employee prepares the order.
5. Shop employee notifies couriers, that the package is ready to be delivered.
6. One of the couriers accepts the delivery request.
7. Courier delivers the order to the customer.
8. If client chose cash payment, courier collects the payment from the client.
9. Order is saved in the system as completed and can be accessed by shop manager from the web app.

### 2.1.3 Acceptance criteria

Below are the key acceptance criteria of the system. Acceptance criteria are defined for main user stories of each app user.

User story: **As a client I want to make an order.**

Acceptance criteria checklist

- Can I observe expected time of delivery?
- Can I choose the time of delivery?
- Can I see, which products are unavailable?
- Can I cancel the order?
- Can I read product reviews?

Non-functional requirements

- Can I sort and filter the list of products?
- Is the expected time of delivery computed in real time?

User story: **As a courier I want to deliver the order.**

Acceptance criteria checklist

- Can I see the destination location?
- Can I see the requested time slot?

Non-functional requirements

- Can I generate the shortest path?
- Can I reassign the order to another courier?

User story: **As a Shop employee I want to prepare the order and change its status.**

Acceptance criteria checklist

- Can I pick up the order from pending orders and change its status to Collecting?
- Can I change order status to WaitingForCourier and notify the courier, that the package is ready?
- Can I see all the products in the order?
- Can I reject the order?

Non-functional requirements

- Does a change of OrderStatus changes it on all modules?
- Is order assigned to another shop, if I reject the order?

- Do I receive the notification, when there is a new pending order?

User story: **As a shop manager I want to manage the list of products.**

Acceptance criteria checklist

- Can I add a new product to the offer?
- Can I remove a product?
- Can I modify the price of the product?

Non-functional requirements

- Are products in the list unique?
- Does the price change have no effect on the current orders?

## 2.2 Non-functional requirements

The requirements that define how our system is supposed to be and correspond to the abbreviation URPS are shown below.

### 2.2.1 Usability

The application has a easy-to-use interface similar to existing online grocery shopping services. The layout is intuitive (i.e. there is a navigation bar on top of the page with login/registration components, a shopping cart button etc.). The customer is presented with a list of products but can also search for items by keywords. The products can be filtered and sorted.

### 2.2.2 Reliability

The application should handle situations like broken connection or unsuccessful payment. Shop module should also inform client, when there is no products or couriers available or the delivery time is getting longer.

When there is temporary no connection to the internet, application shouldn't log out the user. In this scenario, app should also remember the current state (it is easy to achieve in Single Page Application architecture).

Application should also use refresh token to sign in a user without forcing him to enter the credentials.

Lastly, the modules should be independent of each other, so that the crash of the one module doesn't affect other ones.

### 2.2.3 Performance

The app is expected to function properly with around 500 people using it at once. The users should be able to filter products and make their orders in a reasonable amount of time.

The notification mechanism should be well optimized, in order to provide nearly real time communication between modules and system actors.

### 2.2.4 Supportability

The system should collect logs from all modules and store them in a database. Thanks to that, when the bug or crash occurs, system administrators can check causes and report it to the development team.

# 3 System structure

## 3.1 Client module structure

### 3.1.1 Client

This class represents a customer in our system. Required data about the client is collected during the registration, where an instance of this class is created. The class consists of necessary fields for delivering the order and contacting the client. This class also has methods representing their certain activities (**MakeOrder**, **MakeComplaint**). A client can have any number of orders (type **Order**) and can, but is not required to, have a **ShoppingCart** for items.

### 3.1.2 Order

This class describes a shopping order in the context of being created, customized, then monitored by a client. A list of products is represented by one of the class' public fields. When a client wants to buy something immediately (**AddProduct** method) or after finishing his **ShoppingCart** (**AddFromCart** method) the new estimated **Price** is calculated. Once the client proceeds to payment, the **PaymentOpt** is chosen and other fields are set as well. The **CurrentState** changes accordingly to whether or not the client has paid, if the deliverer has gone to deliver the package and if the client received it.

Methods that are used when a client wants to monitor the package after finalizing the order are **Estimat-eDeliveryData**, **NotifyClient** and **TrackOrder**.

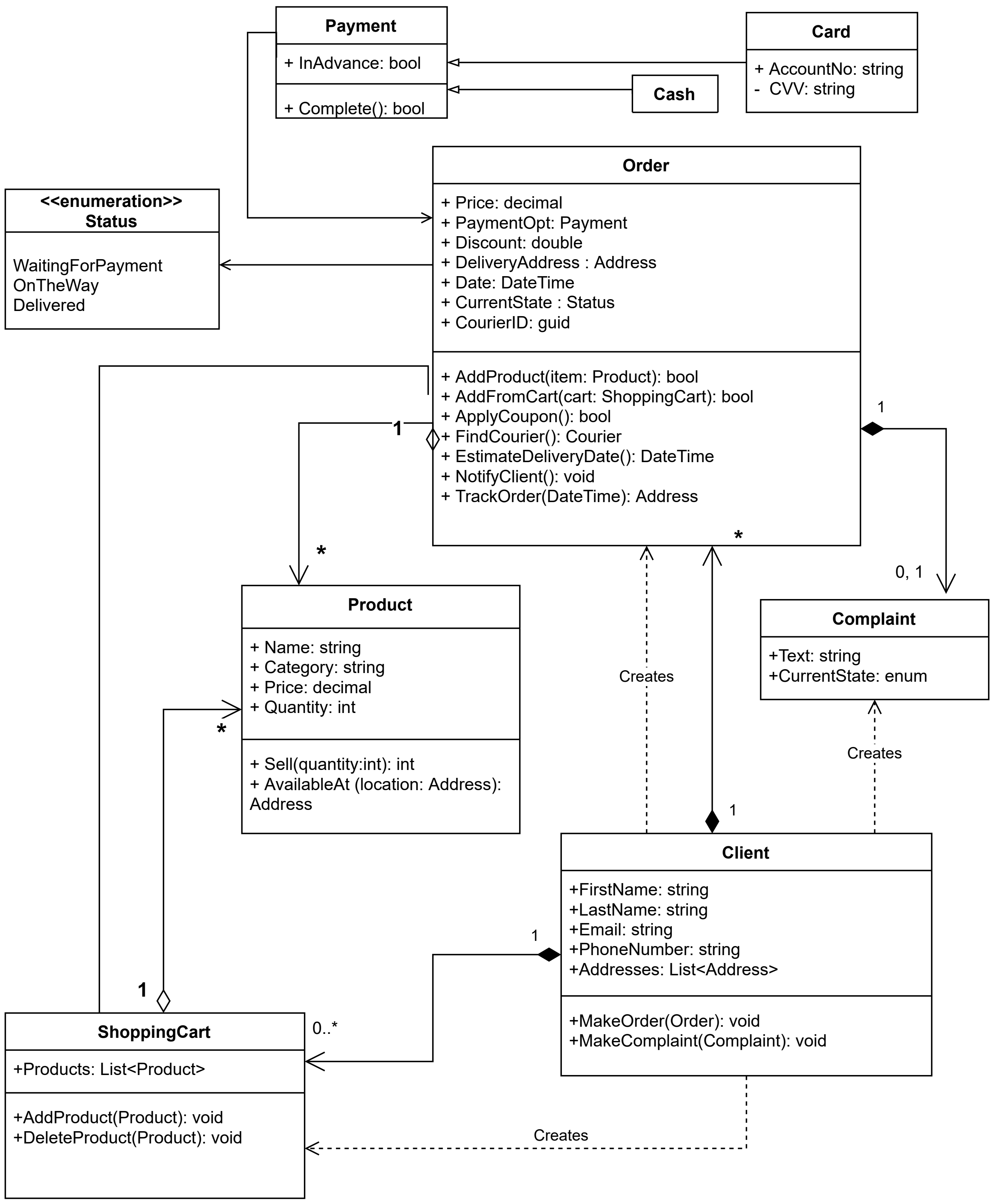
### 3.1.3 Product

This class represents a product, mainly from a client's perspective, as an element in their shopping cart or order. It has a **Price**, can be searched by **Category** and the **Quantity** can be specified. The **Quantity** sold can be recorded by the Shop through **Sell()** and checked for availability with **AvailableAt()** methods.

### 3.1.4 ShoppingCart

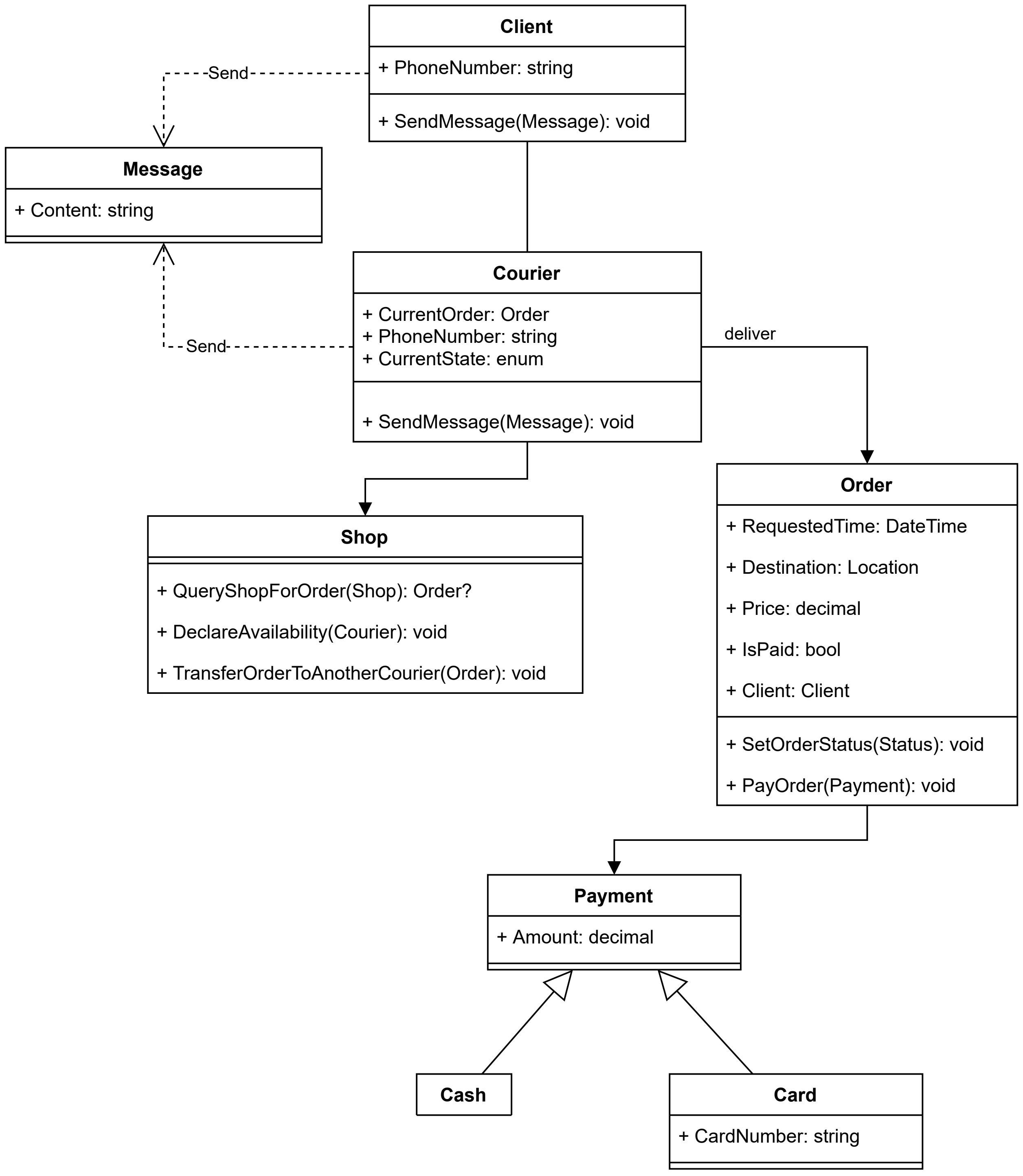
This class describes a shopping cart, to which products can be added to and removed freely. Once the cart is confirmed, the items are all added to the actual order's list of products that will be shipped to the client later on.

Figure 3: Client module class diagram.



## 3.2 Delivery module structure

Figure 4: Delivery module class diagram.



### 3.2.1 Courier

Courier is the main actor in delivery module. The courier is responsible for delivering the order from the shop to a client. That includes setting appropriate order status and collecting payment from clients, if they want to pay with cash. Besides that, courier can also communicate with the client by sending message or calling. If for some reason the courier can't deliver the products, the order can be transferred to another courier.

### 3.2.2 Shop

This class in delivery module represents the shop side in communicating with the courier, preceding the actual delivery. Through that class, courier can query the shop for new order or declare availability. If the courier is available, the shop can notify a courier, that the package is ready to be picked up. There is also a method providing information about a possibility of delegating the task to someone else when justified.

### 3.2.3 Client

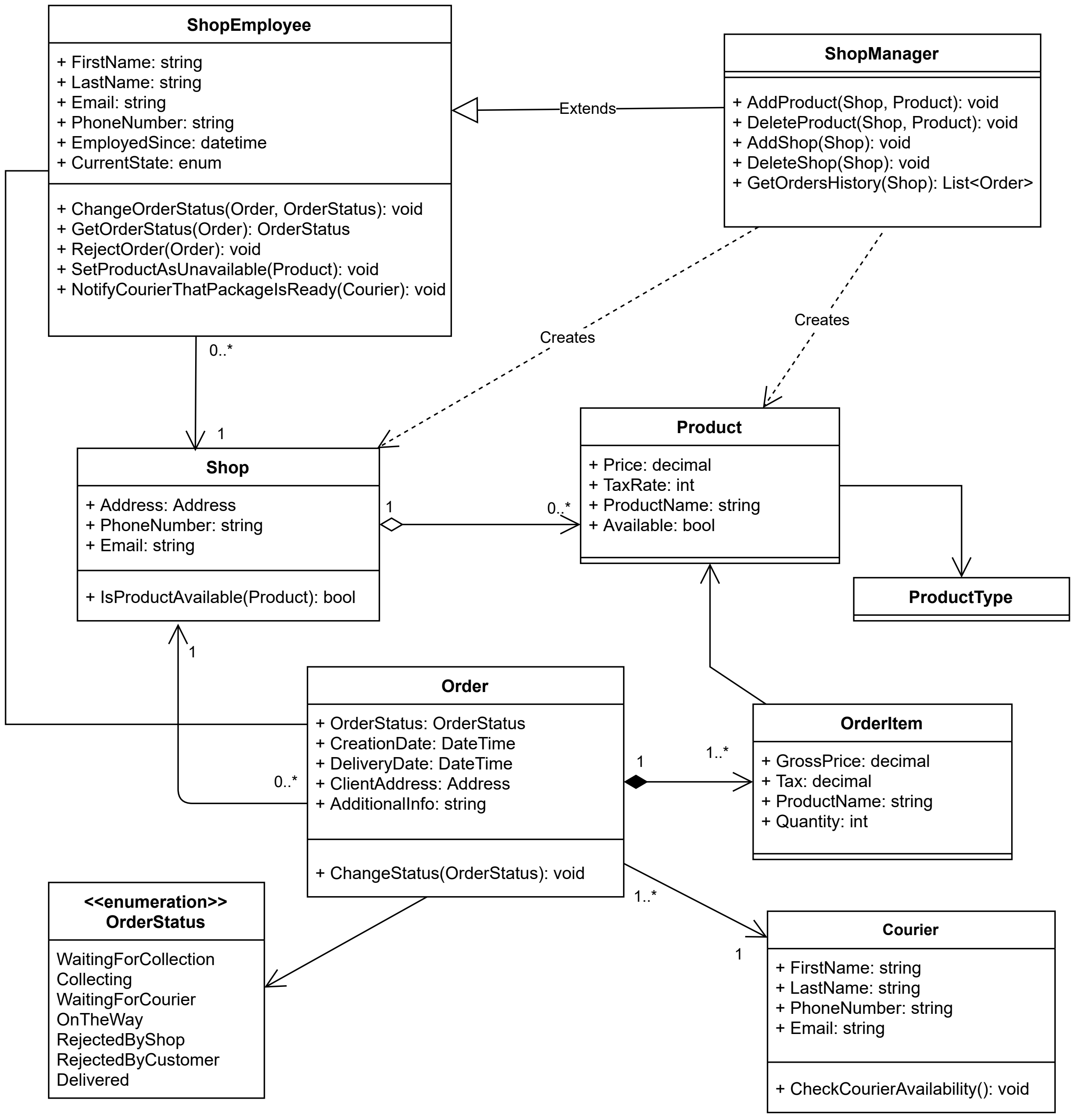
In this module, the client is primarily used for communication between the courier and the client. There are two types of communication: over the phone and via text messages.

### 3.2.4 Order

Order object contains information, that is required for proper delivery of products. The order stores the delivery address (address of a client) and requested time (time, when the client wants to pick up the order from the courier). Additionally, order object has property **IsPaid**, which indicates, whether the Order has been paid (with card) or it will be paid with cash to the courier. Lastly, the order object has also reference to the Client object, which contains client details.

## 3.3 Shop module structure

Figure 5: Shop module class diagram.



### 3.3.1 Shop

Shop object is an individual shop facility of a given retail chain (like Carrefour for example). Each Shop object has many ShopEmployees and at least one ShopManager, who are main actors in this module. Furthermore, each Shop has an offer of available products (the subset of global product offer, which is also handled by the shop module). In addition to this, each shop object contains orders, which were handled or are being processed by a given Shop.

### 3.3.2 Order

After the order is created in client module, it goes to shop module, where it is being prepared and packed. Each order consists of many OrderItems, which are basically Products from shop offer, but with Quantity and GrossPrice (including tax) calculated. Orders are assigned to courier, who is responsible for delivering the order to customer. At any time, ShopEmployee, who is preparing the order can contact with the corresponding courier. Moreover, each order has OrderStatus, which are listed in the diagram. Status can be modified by ShopEmployee during orders processing.

Available order statuses:

- WaitingForCollection – order is waiting to be processed by one of the ShopEmployee
- Collecting – order is being prepared by the ShopEmployee
- WaitingForCourier – order is prepared; it is waiting for the courier
- OnTheWay – order is being delivered by the courier
- RejectedByShop – order is reject by the shop
- RejectedByClient – order is reject by the client
- Delivered – order is delivered to client

### 3.3.3 ShopEmployee

ShopEmployee can preform basic actions regarding Order processing like changing OrderStatus or notifying courier, that the package is ready to pick up. Furthermore, ShopEmployee can also set product as unavailable in the shop facility, where he/she works.

### 3.3.4 ShopManager

ShopManager extends ShopEmployee and has additional actions regarding product offer and shop facility grid management. ShopManager can add/remove product from the global product offer and add/remove shop from the facility grid. In addition, ShopManager can get reports of shop efficiency (number of prepared orders in a given month) and inspect orders history.

### 3.3.5 Product

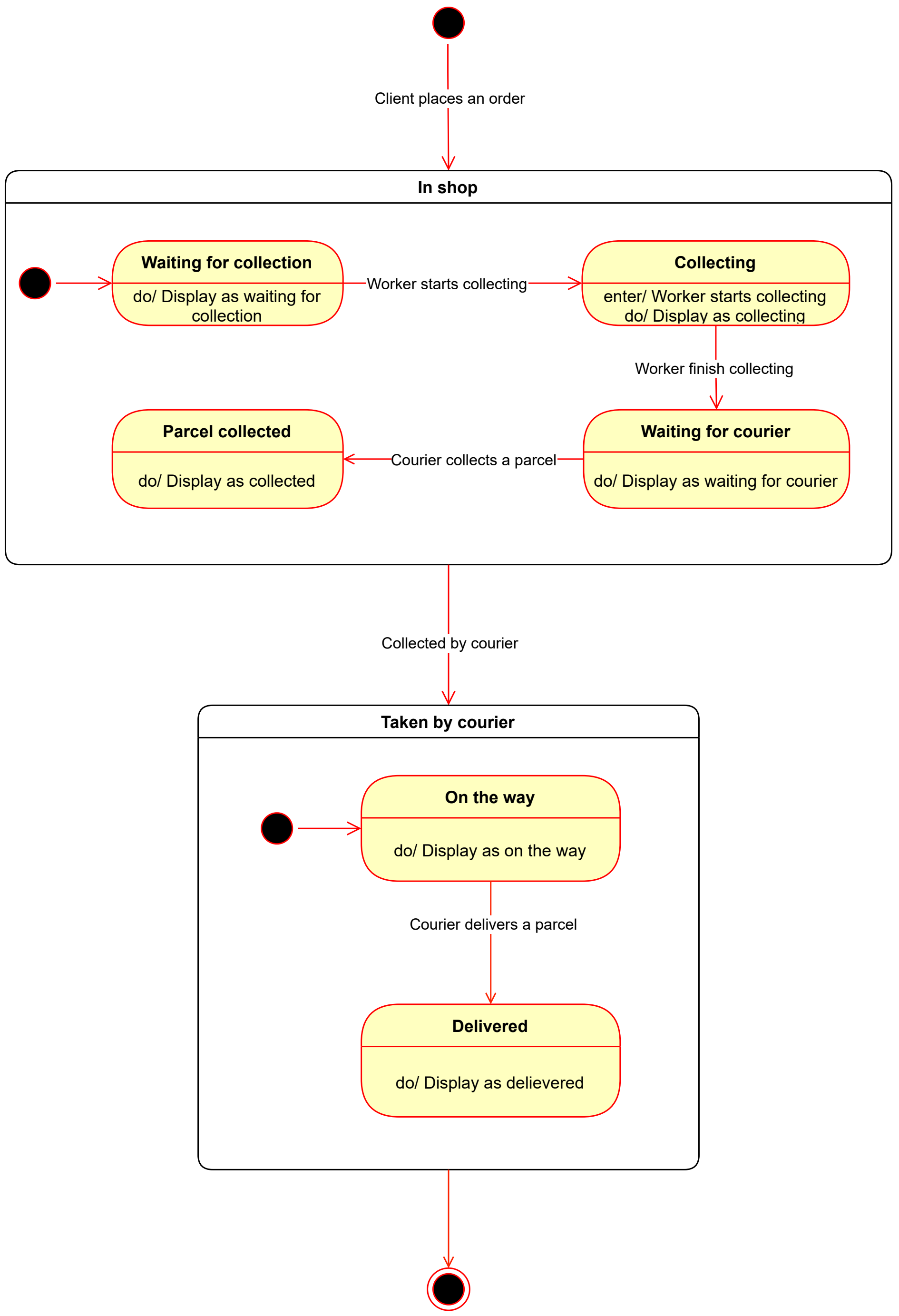
In the shop module workers can also manage the list of products available for the customers. There are three entities, which relate to products:

- ProductType – globally available product in a given retail chain (like 1L bottle of Coca-Cola)
- Product – product in a given shop facility (it is very similar to ProductType, but it only refers to a single shop facility). It contains Available field, which indicates, whether the product is available in a shop facility. For example: Coca-Cola is available in Carrefour in Warsaw, ul. Marszałkowska 22, but it is not available in Carrefour in Kraków, ul. Długa 11)
- OrderItem – product in the order, it has quantity and GrossPrice properties. For example: two bottles of 1L Coca-Cola ordered by Jan Kowalski from Carrefour.

# 4 System states

State diagrams show states of orders (both shop and client module), shop employee, courier and complaints. The other objects did not require such a detailed description of their states due to their obvious action.

## 4.1 Order (shop module)


Client places an order
In shop
Waiting for collection
do/ Display as waiting for collection
Worker starts collecting
Collecting
enter/ Worker starts collecting
do/ Display as collecting
Worker finish collecting
Parcel collected
do/ Display as collected
Courier collects a parcel
Waiting for courier
do/ Display as waiting for courier
Collected by courier
Taken by courier
On the way
do/ Display as on the way
Courier delivers a parcel
Delivered
do/ Display as delievered

Object Order in shop module has a field *OrderState* which can take following values:

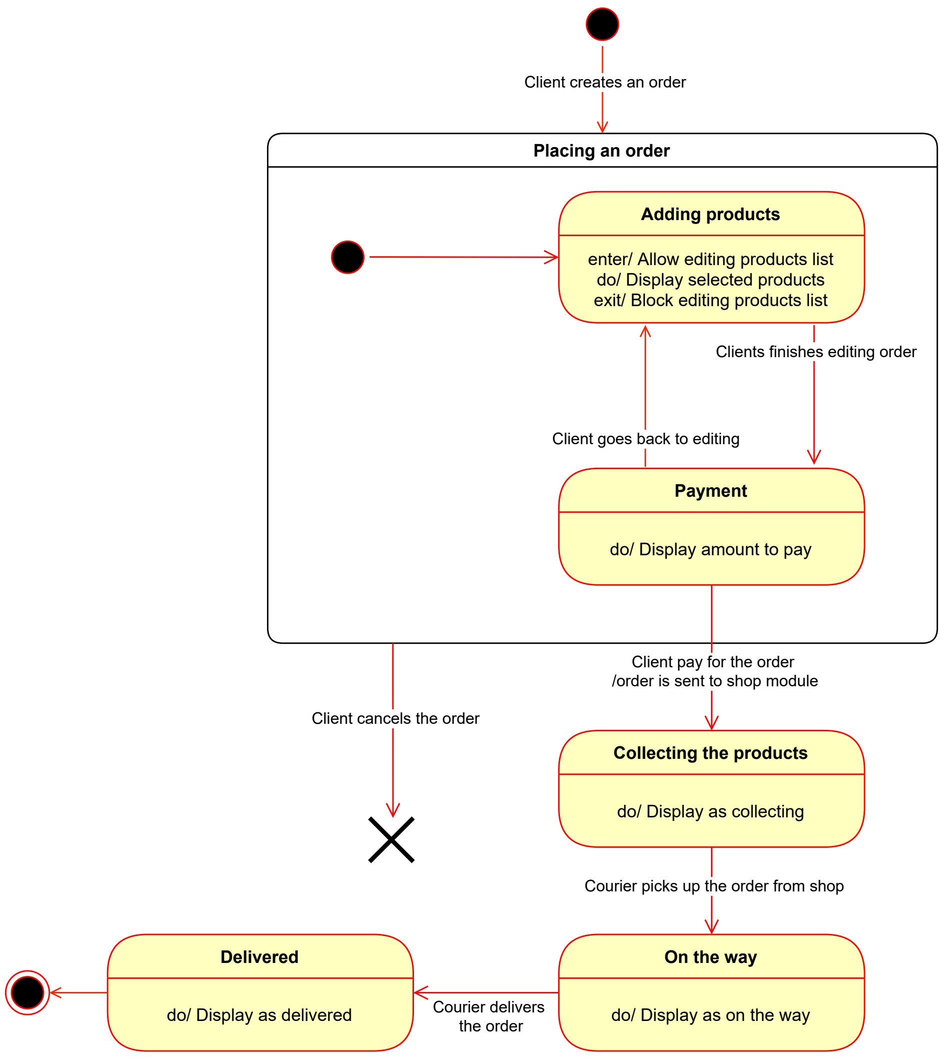
- WaitingForCollection,
- Collecting,
- WaitingForCourier,
- ParcelCollected,
- OnTheWay,
- Delivered

Diagram illustrates how the shop can see an manage the order. It is divided into two parts, first represents states connected with the shop while the second one states connected with courier. First of all orders are made by client and before that time shop doesn't have access to it. Once the order is submitted by a client it gains status as **WaitingForCollection** . It means that the order is already in system and it waits for a shop worker to start collecting it. There is a need for that state because it is a start state and a worker may be busy collecting other orders. There can be a lot of orders in that state in the same time.

Worker starts collecting the order and changes it status to **Collecting**. Now he can see products and their amount he needs to find and collect. After the worker submits completing the order it changes its status to **Waiting for courier**. The order is packed and is waiting for a courier to take it and after that action it changes its state to **Parcel collected**.

In the part connected with courier when courier has parcel is going straightly to the client it changes state to **On the way**. There is a difference between this and previous state because **Parcel collected** means that parcel left the shop but courier can have many parcels so he can go to another client earlier. As soon as courier arrives to client and give him the parcel the state changes to **Delivered** and that is the end state.

## 4.2 Order (client module)



Client creates an order
Placing an order
Adding products
enter/ Allow editing products list
do/ Display selected products
exit/ Block editing products list
Clients finishes editing order
Client goes back to editing
Payment
do/ Display amount to pay
Client pay for the order
/order is sent to shop module
Client cancels the order
Collecting the products
do/ Display as collecting
Courier picks up the order from shop
On the way
do/ Display as on the way
Courier delivers the order
Delivered
do/ Display as delivered

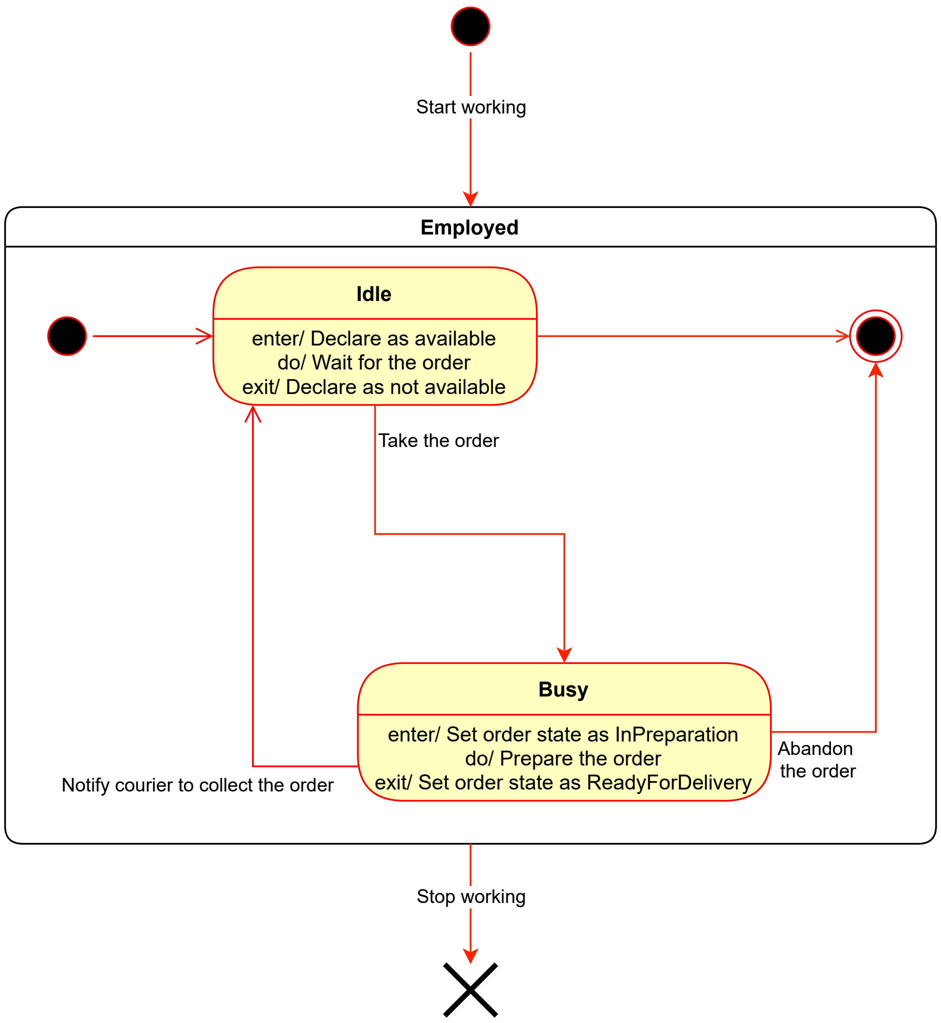
Object Order in client module has a field *CurrentState* which can take following values:

- AddingProducts,
- Payment,
- CollectingProducts,
- OnTheWay,
- Delivered

Diagram illustrates how the shop can see an manage the order. It is divided into two parts, first represents states connected with the placing an order while the second one states connected with courier.In the first part when client decides to make an order then it comes to starting state **AddingProducts**. Client can see list of products, can add products to the list and also delete products from the list. When he decides that he has chosen all products he go to the payment section and state changes to **Payment**. Payment section display amount of money to pay and display available payment methods. Client can still go back to editing the order which makes again **AddingProducts** state. Client can also cancel the order which result in deleting the Order object.

Once a client accepts the order, it is sent to the shop module and the state changes to **CollectingProducts**. It means that shop got notification about the order and the realization is in progress. When the collection of products is completed and courier is straightly on the way to client the order changes its state to **On the way**. When the courier arrives at right address and delivers the order it changes status to **Delivered**.

## 4.3 Shop worker

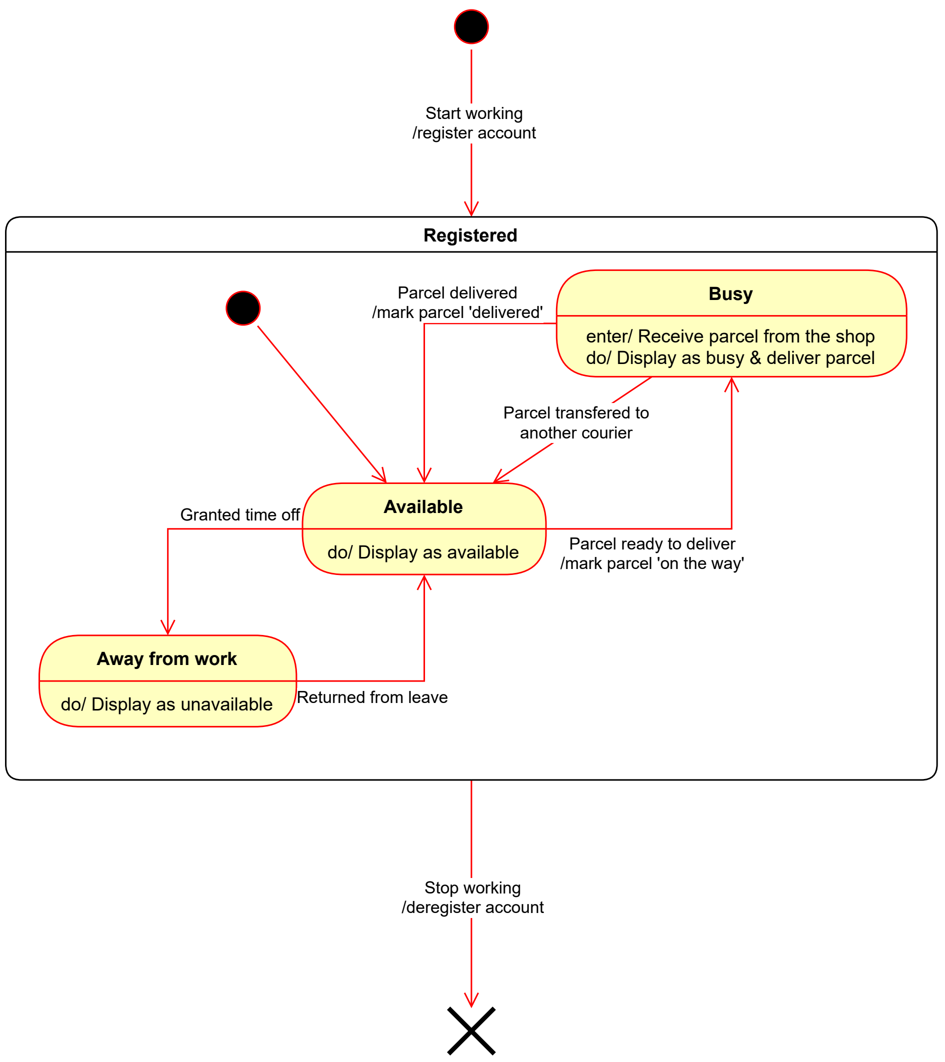

Start working
Employed
Idle
enter/ Declare as available
do/ Wait for the order
exit/ Declare as not available
Take the order
Busy
enter/ Set order state as InPreparation
do/ Prepare the order
exit/ Set order state as ReadyForDelivery
Notify courier to collect the order
Abandon the order
Stop working

Object ShopEmployee in shop module has a field *CurrentState* which can take following values:

- Idle,
- Busy

Shop worker has two states. He can be in **Idle** state which means he is declared as available and waits for the orders. Then he changes status to **Busy** which means he is in progress of completing the order.

## 4.4 Courier

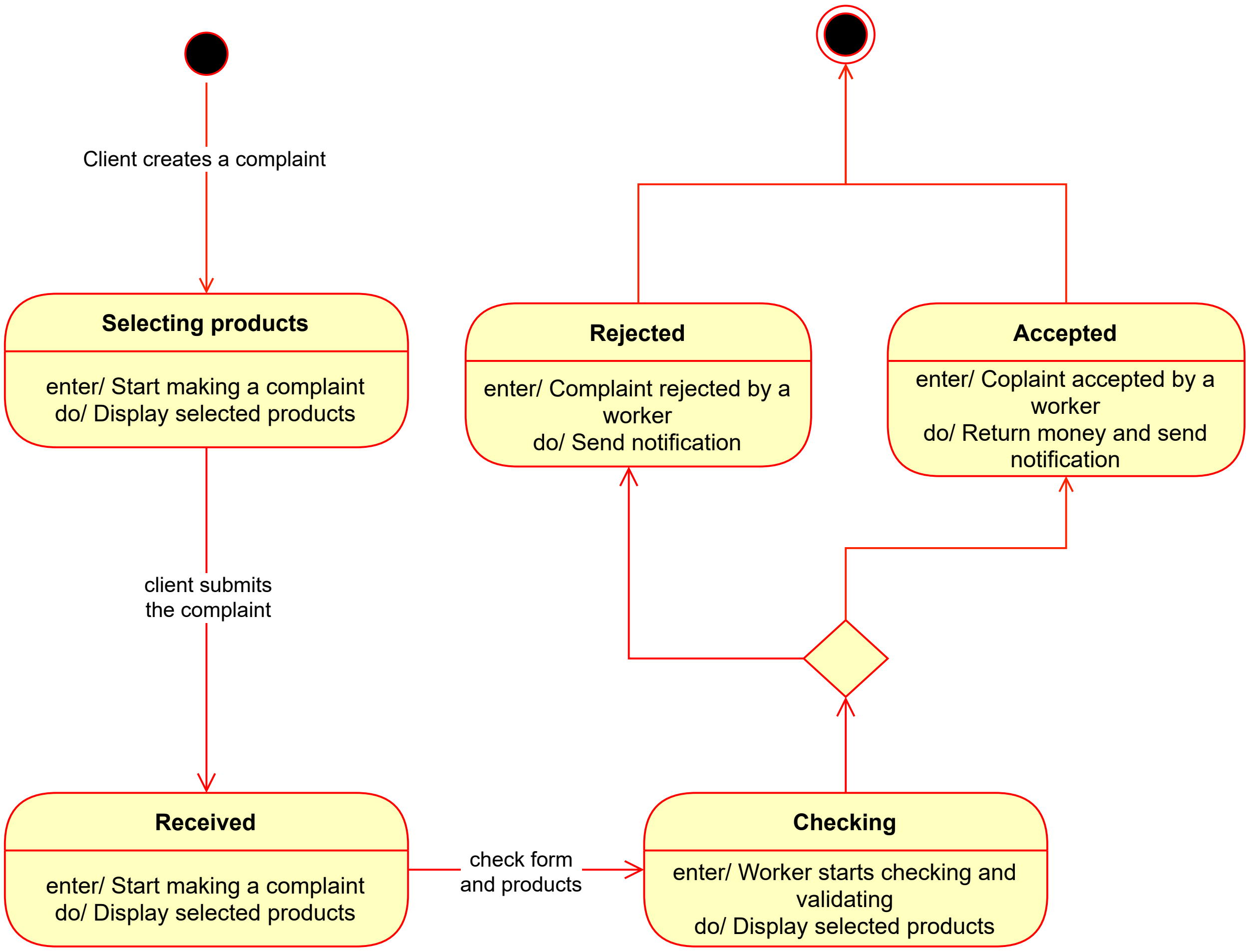


Object Courier in Delivery module has a field *CurrentState* which can take following values:

- Available,
- Busy
- AwayFromWork

The object of Courier is created when someone register his account. The staring state is **Available**. In this state a courier is available to work he is waiting for orders from the shop. Courier can log out or have a break which means that he the state of him changes to **AwayFromWork**. In this state courier can log in or come back from break which result in coming back to **Available** state. When courier is on the way to shop or on the way to client he changes his status to **Busy**. After delivery of package to the client it changes status to **Available**.

## 4.5 Complaint



Client creates a complaint
Selecting products
enter/ Start making a complaint
do/ Display selected products
client submits the complaint
Received
enter/ Start making a complaint
do/ Display selected products
check form and products
Checking
enter/ Worker starts checking and validating
do/ Display selected products
Rejected
enter/ Complaint rejected by a worker
do/ Send notification
Accepted
enter/ Coplaint accepted by a worker
do/ Return money and send notification

Object Complaint in Client module has a field *CurrentState* which can take following values:

- SelectingProducts,
- Received,
- Checking,
- Accepted,
- Rejected

Client can create a complaint to complain products which did not meet the client's expectations. Starting state is **SelectingProducts**. Client has form presenting products with option to select which of them he wants to complain. After the client submit the complaint the notification is send to the shop. When the shop receives the complaint changes state to **Received** and awaits for shop worker to check if the complaint is right and justified. When shop worker is checking the complaint it gets status **Checking**. Worker can see selected products and accept or reject the complaint. If the complaint is rejected the state of complaint changes to **Rejected** and notification to client is send. If the complaint is accepted the state of complaint changes to **Accepted** and notification to client is send.

# 5 System activities

## 5.1 Modules responsibilities and activities

### 5.1.1 Client module

Client module's key responsibilities:

- creating and customizing the order,
- monitoring the order by the client.

Firstly, the **Client** either adds items to a **ShoppingCart** or buys a **Product** immediately. After confirming the products' choice by the customer, they are asked to choose a **Payment** method, then settle it within a deadline set unless it is with a **Card**. This module also allows the client to track their **Order** and check its **Status**. Finally, after receiving the package, in case of any service shortcomings, they can make a **Complaint** to the shop.

Client module is also responsible for the registration of users and managing user accounts.

Lastly, this module contains all the logic related to creating shopping cart, processing payment and notifying client about order status and estimated delivery date.

### 5.1.2 Delivery module

Delivery module key responsibilities:

- delivering orders to clients by the courier
- assigning couriers to the orders
- assigning shops to orders

Delivery module is in-between client module and shop module. Firstly, it is in charge of delivering products to clients. It also includes communication between a courier and the client (over the phone or text messages).

Secondly, delivery module assigns couriers to the orders, by checking couriers availability and order statuses.

Lastly, delivery module assigns shop facilities to the orders using localization and product availability of a given shop.

### 5.1.3 Shop module

Shop module key responsibilities:

- preparing and packaging the order for the courier
- management of product offer
- management of shop facilities grid

Firstly, shop module is used in order processing flow after client places the order (handled by the client module) and before order delivery (handled by the delivery module). Order is prepared by shop employee, who can change order status (**OrderStatus** enumeration type) and notify courier, that the package is ready to pick up.

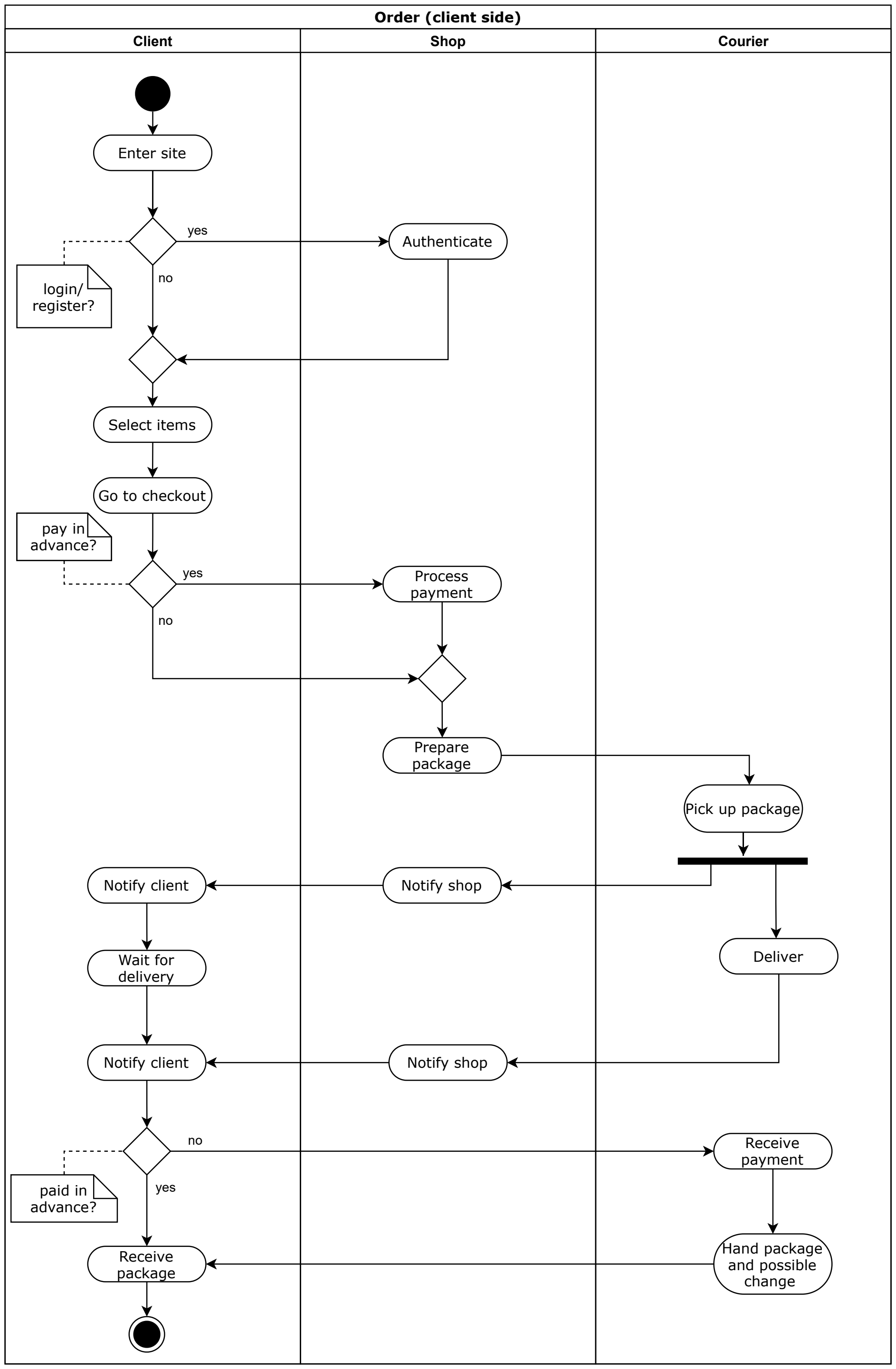
Secondly, shop module is used to modify global product offer (only **ShopManager**) and to change availability of products in certain shops (every **ShopEmployee**).

Thirdly, shop module gives a **ShopManager** an option to add or remove single shop facility from the application.

## 5.2 Key system functionalities

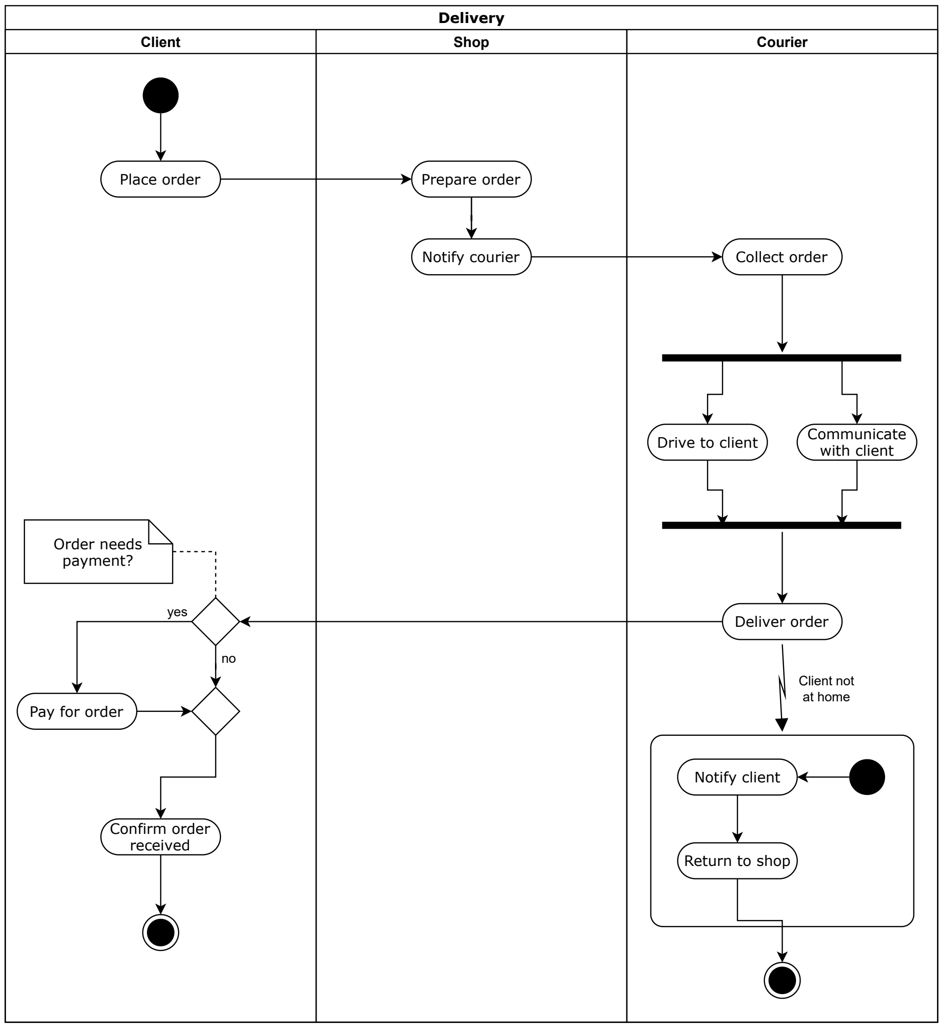
System functionalities are described by activity diagrams, which illustrate what the application will be doing and how it will be done. To select activities for which activity diagrams must be created, user stories have been categorized into complex ones and simple ones. The other objects did not require such a detailed description of their activities due to their obvious action. In the next pages there will be presented diagrams showing activities being taken due to actions like: client making an order, delivery by a courier, making a complaint, shop preparing an order and shop worker.

### 5.2.1 Client making an order



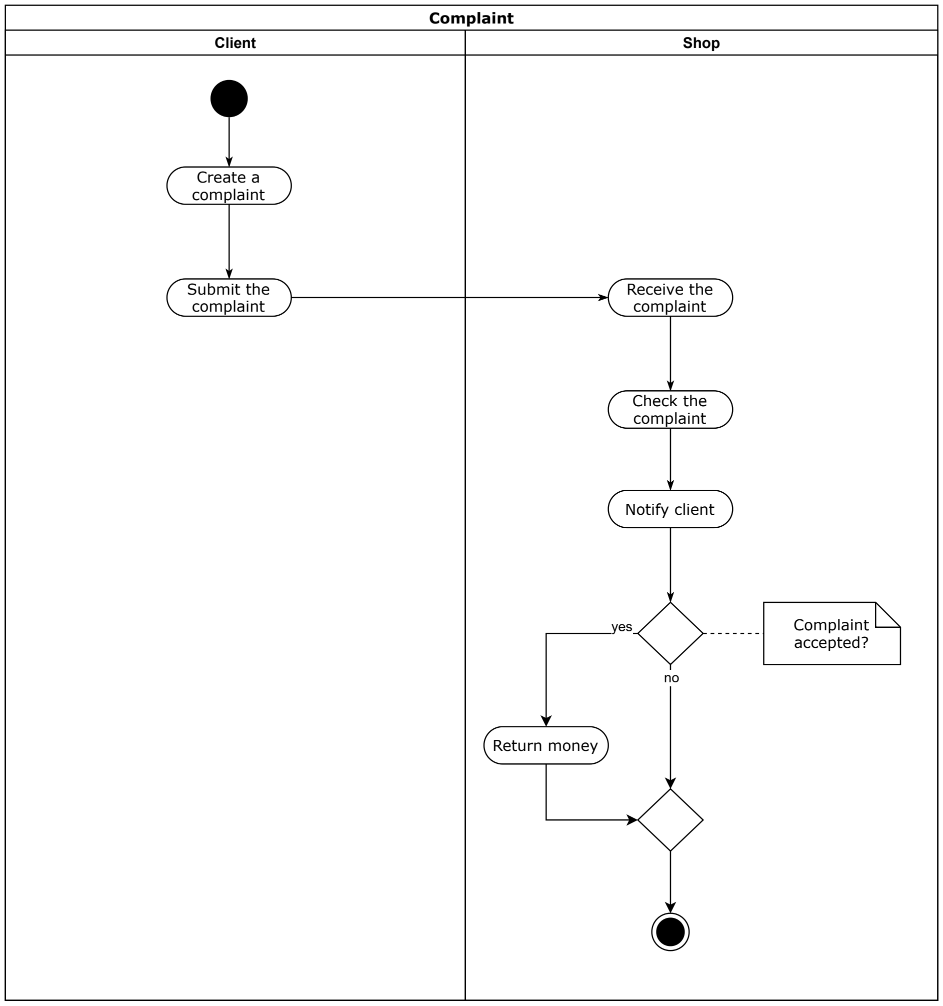
First action taken by a client in process of making an order is entering a website and registering (or logging if registered). After typing credentials then comes veryfing from a shop side. After successful logging client choose products he wants to order. When the list of products is completed client need to submit his choices and then choose method of payment. He can pay in advance or pay exactly to the courier. In the case of the first the payment is processed by the shop. Then after this the package is prepared by the shop and after this is taken by courier. Courier pick up the package and sends notification to shop and client before and after the delivery. After getting package, if it paymant was not in advance client should pay for it to courier and after it the process is done.

### 5.2.2 Delivery by a courier



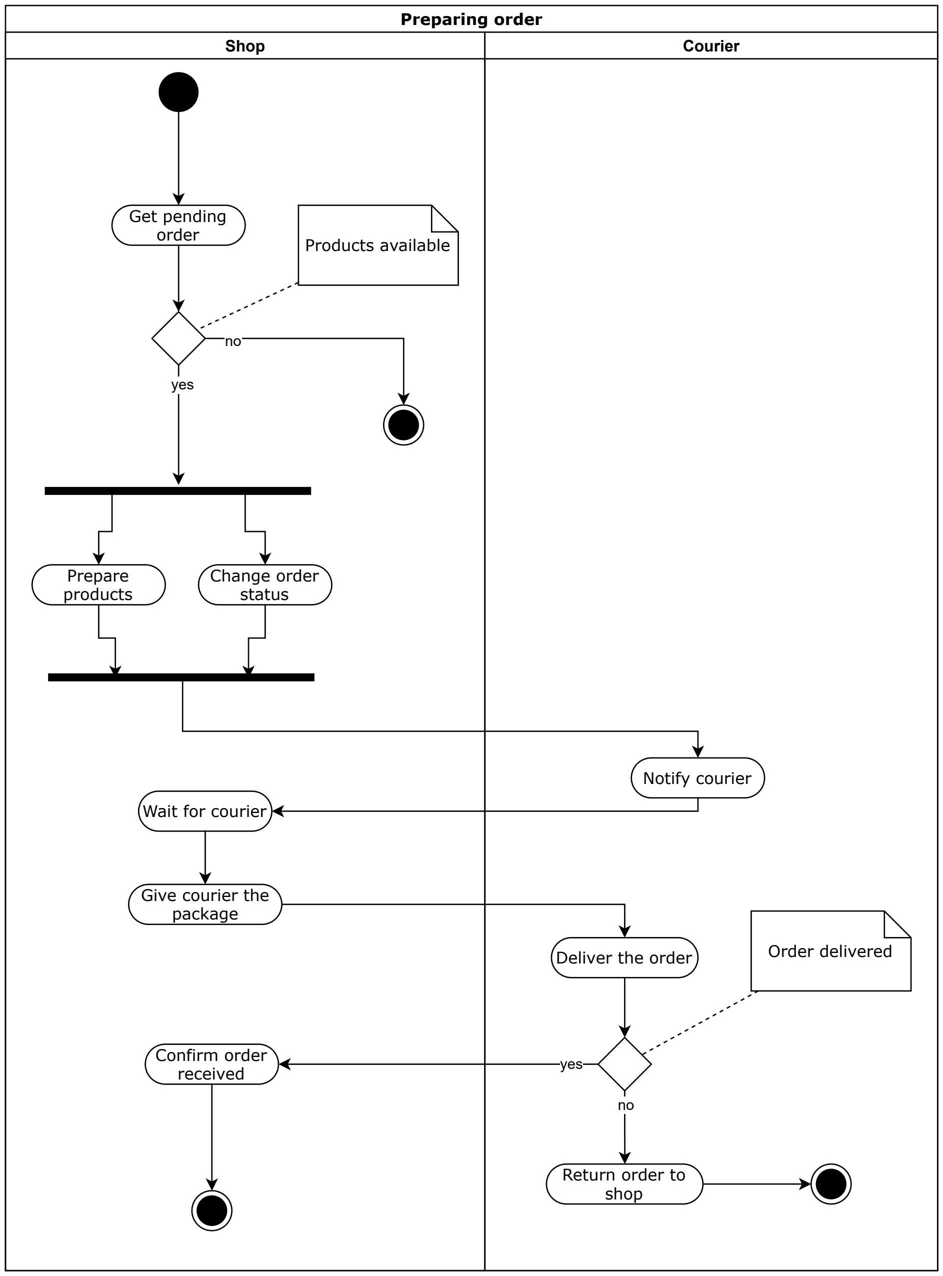
The action of delivering a product starts when a client place an order by paying for the products he has chosen before. Then it is shop job to prepare the order and as soon as they do it they have to mark it as completed and notify the courier. Courier gets notification about the order and comes for it to collect it. Then he is on his way to client and he can communicate with him to tell about his arrival or establish details with an address. When courier delivers product it is for client side to pay for it if he has not paid for it before. After that client confirms receiving products. There may happen situation that the client is not at home, in that situation courier tries to communicate with client and if there is no option to deliver it courier comes with an order back to the shop.

### 5.2.3 Complaint



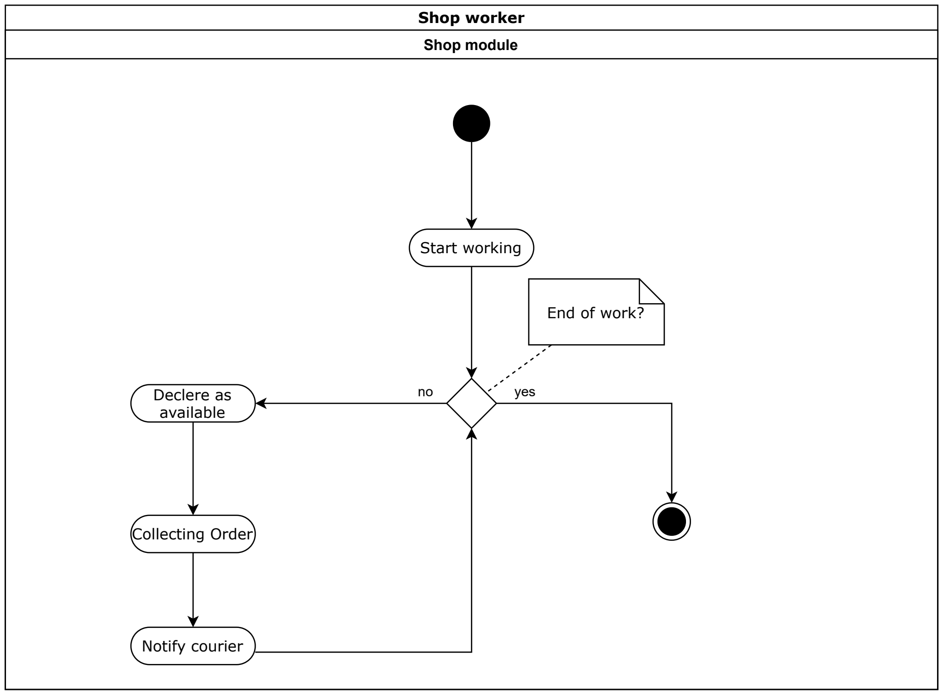
To start making a complaint client after receiving products must click adequate button and then select products which he does not enjoy. After submitting a complaint by the client, the shop receives it and then starts to check if it is right justified. After checking shop sends notification to client whether it was accepted or not. If yes, then money should be returned to a client.

### 5.2.4 Shop preparing an order



To complete an order at first shop worker needs to choose order from pending orders. Then he choose from available products and pack it changing it status. After the package is completed the notification to courier is send that he can come to collect it. Courier delivers the package and in case of success there comes a confirmation of receiving an order. In other case the order is returned to shop.

### 5.2.5 Shop worker



Shop worker is logging to the system to start work. During his shift he declares as available and when he gets an order to collect he does it. After that action he notifies the courier. If it is time to end his shift he logs out, if not he declares as available.

# 6 Communication

## 6.1 Sequence diagrams

### 6.1.1 Preparing order

One of the most important activities in the system is preparing and processing orders by the shop. The process begins, when shop employee sends request **TakeOrder(Order)** to shop module. Then, shop module informs delivery module, that the order is being prepared. At this moment delivery module should find free courier, who will be able to pick up the package.

When the package is ready, shop worker sends request **ConfirmOrderPrepared(Order)** to shop module. Then shop module informs delivery module, that the package is ready to pick up by the assigned courier.

Finally, courier sends request **ConfirmOrderPickUp(Order)** to shop module to communicate, that the package is being delivered to the client.

### 6.1.2 Delivery

Delivery process contains many actions. The most important one is order delivering to the client. After picking up the package from shop, courier changes order status to *NotDelivered* and notifies client.

Second action is processing order payment. If the order requires payment, courier should send payment request to client module, which handles payment in the background. When the payment is finalized, courier should inform shop module, that the order is delivered.

### 6.1.3 Complaint

Groceries delivery system contains also mechanism to make complaints about orders. Firstly, client module submits complaint to shop module. Then, shop worker looks into the complaint and accepts or rejects it. If the complaint is accepted, money is returned to the client's bank account.

Figure 6: Preparing order sequence diagram.

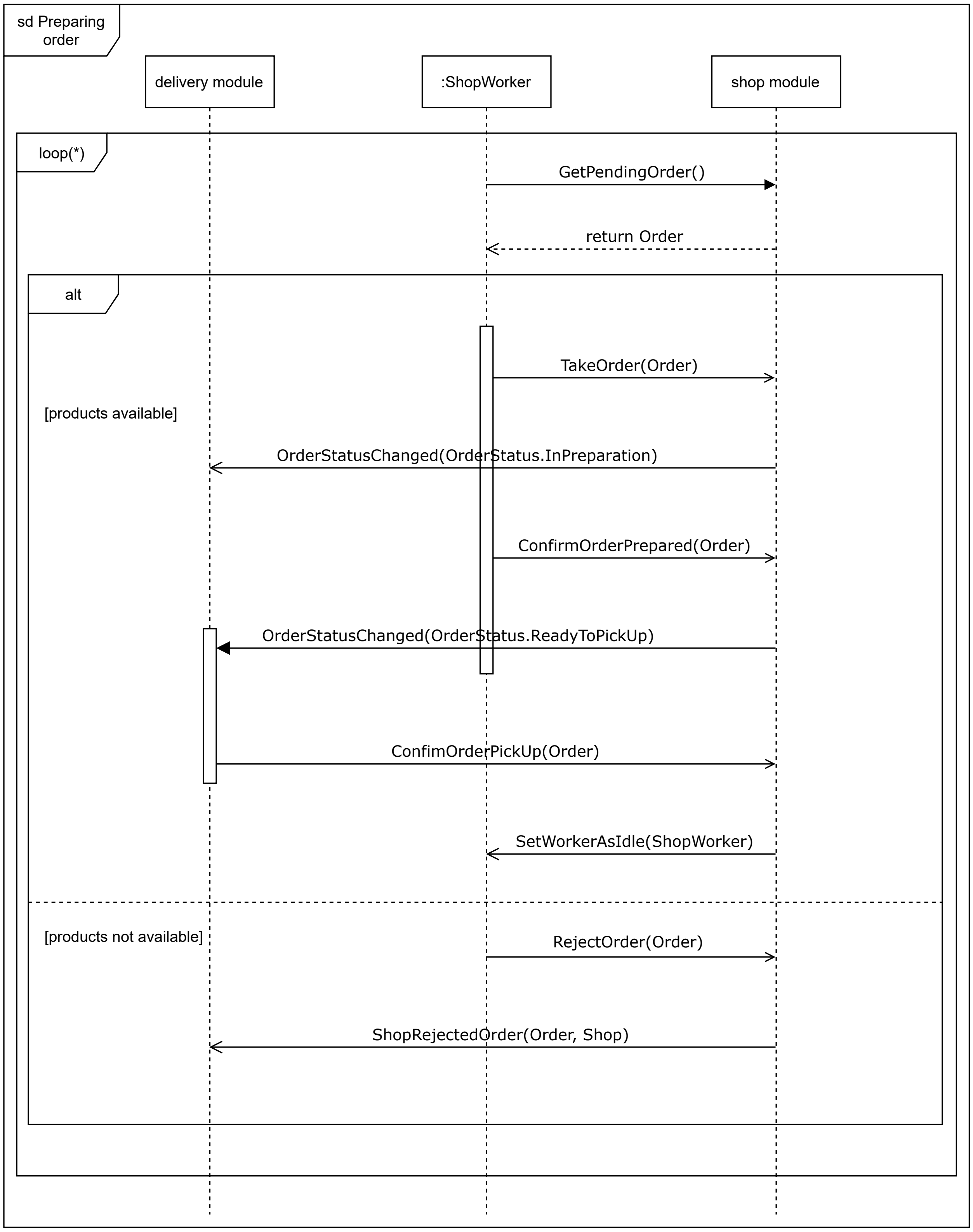


Figure 7: Delivery sequence diagram.

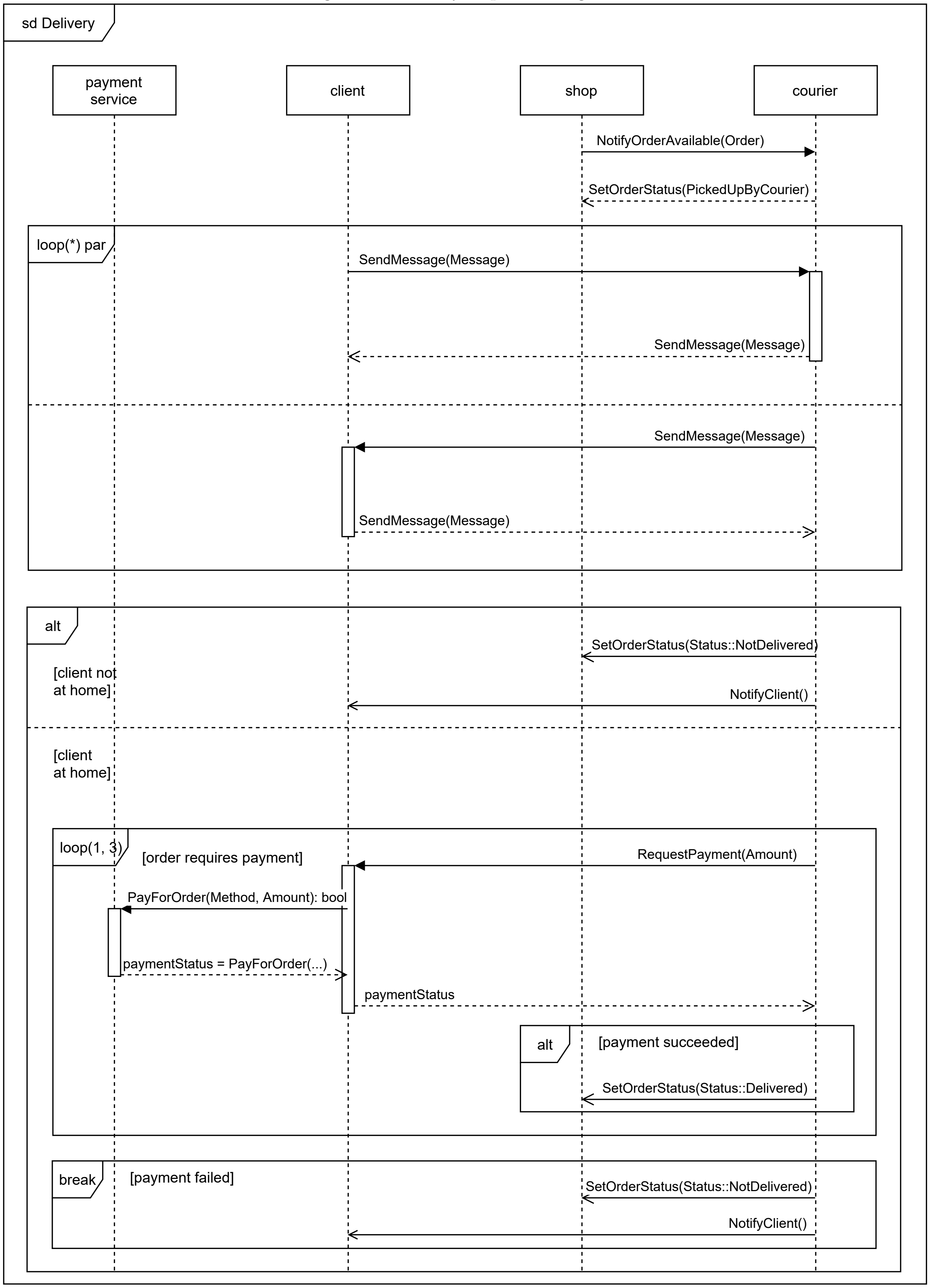
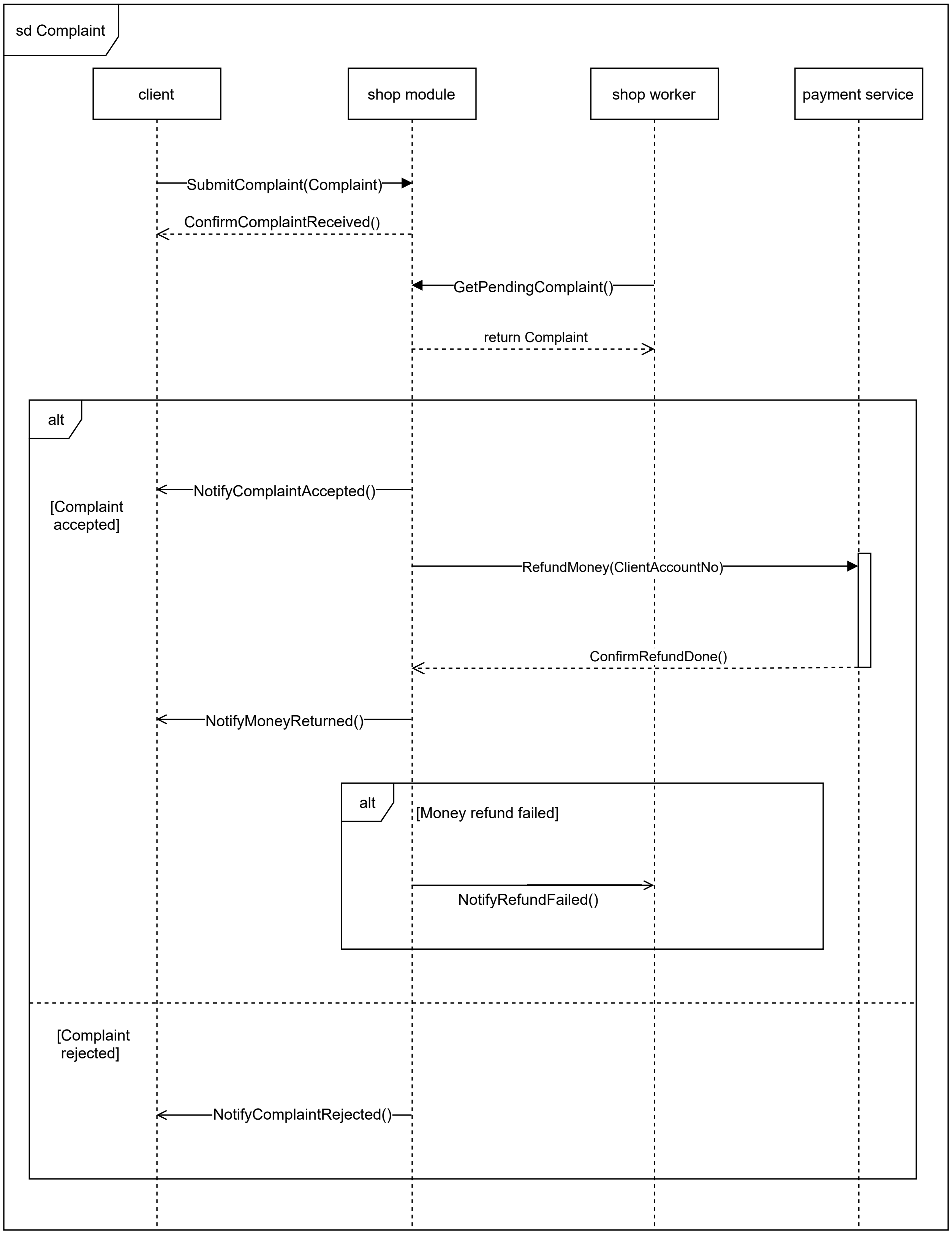


Figure 8: Complaint sequence diagram.



## 6.2 Message structure

This section shows how messages are structured in our system. For further information please refer to our RAML and html files (see: *Attachments* section). Given examples below are represented in a JSON format.

### 6.2.1 Order

The order is represented by an object that consists of a unique id, an array of purchased items, its status and dates of creation and delivery. It also consists of clients data - their address, additional info. When a client makes a new order and later confirms the purchase, the database denotes the new Order. It is done through the `POST` method to `/orders/create/{userId}`. A `GET` request is made to `/orders/pending` when the Shop Worker is looking for all pending orders that are to be handled but when its a specific Order, `GET` is called to `/orders/{orderId}`. The boolean field called 'confirmedPayment' is set to `false` until the accounts are settled. The update is made by calling a `PUT` method to `/orders/{orderId}/payment`.

```json
{
    "orderId": "c2cf4e9e-0393-4189-af47-0aab382ce330",
    "orderItems": [
        {
            "orderItemId": "ca543631-3df7-4d06-8091-179df67c8460",
            "grossPrice": 12,
            "currency": "PLN",
            "productName": "bread",
            "quantity": 3
        }
    ],
    "creationDate": "2021-12-01 12:30:22",
    "deliveryDate": "2021-12-01 13:30:22",
    "clientAddress": {
        "street": "Prosta 12",
        "city": "Warszawa",
        "zipCode": "00-631"
    },
    "additionalInfo": "Info from the client",
    "orderStatus": "Pending",
    "confirmedPayment": false
}
```

Listing 1: Message structure - Order

### 6.2.2 OrderStatus

There are several states in which the Order can be in. The following ones have been considered in the messages: Pending, InPreparation, ReadyForDelivery, PickedUpByCourier, RejectedByShop, RejectedByCustomer, Delivered. This field is modified each time some action is made upon the particular order. The methods `PUT` and `GET` are provided with the chosen Orders id as well as with the adequate action name. For example, by sending these requests to `/orders/{orderId}/pickup` one can update the status to 'PickedUpByCourier'.

```
"ReadyForDelivery"
```

Listing 2: Message structure - OrderStatus

### 6.2.3 Client

The clients information passed in messages are i.e.: phoneNumber and clientAddress. In order to find more details about the client who created an Order in the `GET` method `/clients/{clientAddress}` is used.

```
{
    "userId": "2bcd5428-7bb2-11ec-90d6-0242ac120003",
    "phoneNumber": "123456789",
    "clientAddress": {
        "street": "Prosta 12",
        "city": "Warszawa",
        "zipCode": "00-631"
    }
}
```

Listing 3: Message structure - Client

### 6.2.4 Complaint

Each complaint can be created by a client similarly to how it's in case of an order. It has a unique id, consists of an id of an order it refers to and has a status field that can be modified by shop employees. Similarly to how the client creates a new order, the `POST` call is done to the url that ends with `/create/{userId}` . The `GET` method to `/complaints/pending` retrieves a list of all pending complaints.

```
{
    "complaintId": "061ac70b-e370-40ca-a12e-9ea146ae9429",
    "orderId": "e41d4a1b-e771-4eae-84c8-c598ee60d627",
    "status": "Rejected",
    "text": "Delivery was 5 minutes late"
}
```

Listing 4: Message structure - Complaint

### 6.2.5 ComplaintStatus

When the complaint was pending and is now handled by a given shop, it can either be accepted or rejected, therefore `PUT` method is called to update an appropriate record in the database. The call is made to `/complaints/{action}`, where `action` is one of the two mentioned possibilities.

```
"Rejected"
```

Listing 5: Message structure - ComplaintStatus

# 7 Error handling

During using the application users can face up some problems and errors. There are several kinds of errors along with an exemplary reaction from project application. will be described below.

## 7.1 Errors during handling requests and transactions

Most of error sent from the server during handling requests are only informative. It is required, to show some kind of notification to the user - popup window is preferred, but no hard requirements are specified and form can be different if it will look better. If Frontend is timeouted or it could not establish connection to the server - handling is again the same. For this last case, connection to the server failed, Frontend can create artificial response to display information in the same way as the rest of the errors.

## 7.2 False login credentials

From a user's point of view, a situation may occur when a user tries to log into the website while providing false credentials. In that case, a user will get a single error message stating that the login and password were wrong, and after that he will be able to try to connect one again. However, after fifth trial of bad password of the same login the account connected to that login will be temporarily blocked for 5 minutes. After five minutes the CAPTCHA will appear to check if the user is a human.

## 7.3 Disconnected module

If the module is disconnected user should be taken to the last view of the app and the notification about losing connection should be sent. In shop module it should be list of products and phone numbers to available couriers and clients, in courier module it should be client's address and phone numbers to shop and client as for the client module it should be phone number to shop and courier . Then, if server failed completely, user should see error about the connection provided by this other view, and in case they was disconnected because of lack of authentication - again proper error will be displayed by another view and user will be taken to the login view.

## 7.4 Failure with connecting to database

Adding and updating elements in the database should be transactional - so if server failed before it is finished, it will not be changed. If it finished - new elements will be visible in the Frontend anyway after the server is restarted. An error of disconnected database could occur when trying to either read previous measurements from it, or when trying to save a freshly created one. That problem could easily occur when first checking the connection before even starting the In that case, user should be informed about that in an error message just after single attempt. The server never tries to connect to the database by itself, as it could make our application prone to attacks.

# 8 Simulations of usage

To show that application works properly, there will be described some steps to perform activities and check if they matches expected outcome.

## 8.1 Client making the order on the web app

1. Client logs into the web app.
2. The list of available products with prices shows up.
3. Client can filter and sort products.
4. Client adds some products to shopping cart and chooses product quantity in a modal box.
5. Client chooses the payment method and submits the order.
6. Client module informs the shop module, that there is a new pending order.
7. Client module informs the delivery module, that there is a new pending order.
8. Shop worker gets notification and accepts pending order.
9. Courier gets notification and accepts pending order.
10. Shop worker notifies the courier, that the package is ready.
11. Courier receive the package and delivers it to the client.
12. Client is notified, when the courier is at a given location.

## 8.2 Shop manager adding new product on the web app

1. Shop manager logs into the web app.
2. The list of products offer shows up.
3. Shop manager clicks the ”Add new product” button.
4. The modal box with appropriate text inputs shows up.
5. Shop manager fills required fields and attaches the picture.
6. The new product is visible in the products list.

## 8.3 Shop worker is preparing an order

1. Shop worker starts working by logging in
2. After successful logging he should be declared as available
3. He can check if there are pending orders, if so he choose one and start collecting in
4. He should now have options to abandon the order or accept collecting while sending notification to courier

5. Shop worker can log off

### 8.4 Making a complaint

1. Client makes a complaint after receiving an order
2. Client can add or delete highlighted products which did not meet his expectations
3. Client can submit a complaint
4. Shop worker can see list of complained products
5. Shop worker can mark which products are right justified to complain
6. Shop worker can see amount of money he need to recharge to a client
7. Shop worker can send appropriate amount of money to a client

# 9 Attachments

## 9.1 Message definitions

### 9.1.1 GatewayMessages

See `GatewayMessages.html` in the attachments to explore message definitions as a web page.

GatewayMessages.raml

```
#%RAML 1.0
title: Groceries delivery app - Gateway
version: v1
baseUri: https://mini-delivery.com/
securitySchemes:
    oauth_2_0:
        description: |
            OAuth 2.0 for authenticating API requests.
        type: OAuth 2.0
        describedBy:
            headers:
                Authorization:
                    description: |
                        Used to send a valid OAuth 2 access token. Do not use
                        with the "access_token" query string parameter.
                    type: string
            queryParameters:
                access_token:
                    description: |
                        Used to send a valid OAuth 2 access token. Do not use together with
                        the "Authorization" header
                    type: string
            responses:
                401:
                    description: |
                        Bad or expired token. This can happen if the user or Identity Provider
                        revoked or expired an access token. To fix, you should re-
                        authenticate the user.
                403:
                    description: |
                        Bad OAuth request (wrong consumer key, bad nonce, expired
                        timestamp...). Re-authenticating the user won't help here.
        settings:
            authorizationUri: https://example-identity-provider.com/oauth2/authorize
            accessTokenUri: https://example-identity-provider.com/oauth2/token
            authorizationGrants: [ authorization_code ]
types:
    UUID:
        type: string
        description: UUID
        pattern: ^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}$
    OrderItem:
        description: element of an order
        type: object
        properties:
            orderItemId: UUID
            grossPrice:
                type: number
                description: the final sales price per unit of an item
            currency:
                type: string
                pattern: ^[A-Z]{3}$
            productName: string
            quantity: number
        example:
            orderItemId: ca543631-3df7-4d06-8091-179df67c8460
            grossPrice: 12
```

```
                currency: PLN
                productName: bread
                quantity: 3
        OrderStatus:
            type: string
            enum: [Pending, InPreparation, ReadyForDelivery, PickedUpByCourier,
            ↪  RejectedByShop, RejectedByCustomer, Delivered]
        Order:
            description: Groceries products order
            type: object
            properties:
                orderId: UUID
                orderItems: OrderItem[]
                creationDate: datetime
                deliveryDate: datetime
                clientAddress:
                    type: object
                    properties:
                        street: string
                        city: string
                        zipCode: string
                additionalInfo: string
                orderStatus: OrderStatus
                confirmedPayment: boolean
            example:
                orderId: c2cf4e9e-0393-4189-af47-0aab382ce330
                orderItems: [
                {
                    orderItemId: ca543631-3df7-4d06-8091-179df67c8460,
                    grossPrice: 12,
                    currency: PLN,
                    productName: bread,
                    quantity: 3
                }]
                creationDate: 2021-12-01T12:30:22.52Z
                deliveryDate: 2021-12-01T13:30:22.52Z
                clientAddress:
                    street: Prosta 12
                    city: Warszawa
                    zipCode: 00-631
                additionalInfo: Info from the client
```

```yaml
            orderStatus: Pending
            confirmedPayment: false
    Client:
        description: Client
        type: object
        properties:
            userId: UUID
            phoneNumber:
                type: string
                pattern: ^[0-9]{9}$
            clientAddress:
                type: object
                properties:
                    street: string
                    city: string
                    zipCode: string
        example:
            userId: 2bcd5428-7bb2-11ec-90d6-0242ac120003
            phoneNumber: "123456789"
            clientAddress:
                street: Prosta 12
                city: Warszawa
                zipCode: 00-631
    ComplaintStatus:
        type: string
        enum: [Pending, Accepted, Rejected]
    Complaint:
        description: Complaint
        type: object
        properties:
            complaintId: UUID
            orderId: UUID
            status: ComplaintStatus
            text: string
        example:
            complaintId: 061ac70b-e370-40ca-a12e-9ea146ae9429
            orderId: e41d4a1b-e771-4eae-84c8-c598ee60d627
            status: Rejected
            text: Delivery was 5 minutes late
    Product:
        description: Product
        type: object
        properties:
            name: string
            category: string
            price: number
            quantity:
                type: number
                description: Available quantity. Will always be integral.

/orders:
    /create:
        /{userId}:
            post:
                securedBy: [ oauth_2_0 ]
                description: Add clients order to database
                body:
                    application/json:
                        type: Order
                responses:
                    201:
                        body:
                            application/json:
                                description: |
                                    Successfully created order
                                type: Order
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to create order
                                type: string
    /pending:
        /{shopId}:
            get:
                securedBy: [ oauth_2_0 ]
                description: Get pending orders assigned to shop
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                Successfully got pending orders
                            type: Order[]
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to get pending orders
                                type: string
    /{orderId}:
        get:
            securedBy: [ oauth_2_0 ]
            description: Get chosen order
            responses:
                200:
                    body:
                        application/json:
                            description: |
                                Successfully got chosen order
                            type: Order
                404:
                    body:
                        text/plain:
                            description: |
                                Failed to get this order
                            type: string
        /takeForPreparation:
            put:
                securedBy: [ oauth_2_0 ]
                description: Set order state to inPreparation
                body:
                    application/json:
                        type: OrderStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully taken for preperation.
                                type: OrderStatus
                    404:
                        body:
```

```
                        text/plain:
                            description: |
                                Failed to take for preperation.
                            type: string
        /confirm:
            put:
                securedBy: [ oauth_2_0 ]
                description: Set order state to ReadyForDelivery
                body:
                    application/json:
                        type: OrderStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully taken for delivery.
                                type: OrderStatus
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to take for delivery.
                                type: string
        /pickup:
            put:
                securedBy: [ oauth_2_0 ]
                description: Set order state to PickedUpByCourier
                body:
                    application/json:
                        type: OrderStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully picked up order.
                                type: OrderStatus
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to pick up the order.
                                type: string
        /reject:
            /{userId}:
                put:
                    securedBy: [ oauth_2_0 ]
                    description: The order was rejected by the client.
                    body:
                        application/json:
                            type: OrderStatus
                    responses:
                        200:
                            body:
                                application/json:
                                    description: |
                                        The order was rejected by client.
                                    type: OrderStatus
                        404:
                            body:
                                text/plain:
                                    description: |
                                        Order not found
                                    type: string
            put:
                securedBy: [ oauth_2_0 ]
                description: The order was rejected by the shop.
                body:
                    application/json:
                        type: OrderStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    The order was rejected by shop.
                                type: OrderStatus
                    404:
                        body:
                            text/plain:
                                description: |
                                    Order not found
                                type: string
        /payment:
            put:
                securedBy: [ oauth_2_0 ]
                description: |
                    Update payment status.
                body:
                    application/json:
                        type: boolean
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully updated the payment state.
                                type: boolean
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to update the payment state.
                                type: string

/complaints:
    /create:
            /{userId}:
                post:
                    securedBy: [ oauth_2_0 ]
                    description: Add clients complaint to database
                    body:
                        application/json:
                            type: Complaint
                    responses:
                        201:
                            body:
                                application/json:
                                    description: |
                                        Successfully made a complaint
                                    type: Complaint
                        404:
```

```
                        body:
                            text/plain:
                                description: |
                                    Failed to make a complaint
                                type: string
    /pending:
        /{shopId}:
            get:
                securedBy: [ oauth_2_0 ]
                description: Get pending complaints adressed to the shop
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully got pending complaints.
                                type: Complaint[]
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to get pending complaints.
                                type: string
    /{complaintId}:
        get:
            securedBy: [ oauth_2_0 ]
            description: Get chosen complaint.
            responses:
                200:
                    body:
                        application/json:
                            description: |
                                Successfully got chosen complaint.
                            type: Complaint
                404:
                    body:
                        text/plain:
                            description: |
                                Failed to get this complaint.
                            type: string
        /accept:
            put:
                securedBy: [ oauth_2_0 ]
                description: Accept complaint
                body:
                    application/json:
                        type: ComplaintStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Accepted complaint.
                                type: ComplaintStatus

                    404:
                        body:
                            text/plain:
                                description: |
                                    Complaint not found.
                                type: string
        /reject:
            put:
                securedBy: [ oauth_2_0 ]
                description: Reject complaint.
                body:
                    application/json:
                        type: ComplaintStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: !
                                    Rejected complaint.
                                type: ComplaintStatus
                    404:
                        body:
                            text/plain:
                                description: |
                                    Complaint not found.
                                type: string

```

```
/clients:
    /{clientAddress}:
        get:
            securedBy: [ oauth_2_0 ]
            description: Get details about client
            responses:
                200:
                    body:
                        application/json:
                            description: !
                                Successfully gotten clients info.
                            type: Client
                404:
                    body:
                        text/plain:
                            description: |
                                Client not found.
                            type: string

/products:
    /{productId}:
        get:
            description: Get product information
            responses:
                200:
                    body:
                        application/json:
                            description: |
                                Successfully got product info.
                            type: Product
                404:
                    body:
                        text/plain:
                            description: |
                                Product not found.
    /category:
        /{category}:
            get:
                description: Get all products in the specifies category
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully got products info.
                                type: Product[]
                    404:
                        body:
                            text/plain:
                                description: |
                                    Category not found.
    get:
        description: Get all products
        responses:
            200:
                body:
                    application/json:
                        description: |
                            Successfully got products info.
                        type: Product[]

```

### 9.1.2 ShopMessages

See `ShopMessages.html` in the attachments to explore message definitions as a web page.

ShopMessages.raml

```
#%RAML 1.0
title: Groceries delivery app - Shop module
version: v1
baseUri: https://shop.mini-delivery.com/
securitySchemes:
    oauth_2_0:
        description: |
            OAuth 2.0 for authenticating API requests.
        type: OAuth 2.0
        describedBy:
            headers:
                Authorization:
                    description: |
                        Used to send a valid OAuth 2 access token. Do not use
                        with the "access_token" query string parameter.
                    type: string
            queryParameters:
                access_token:
                    description: |
                        Used to send a valid OAuth 2 access token. Do not use together with
                        the "Authorization" header
                    type: string
            responses:
                401:
                    description: |
                        Bad or expired token. This can happen if the user or Identity Provider
                        revoked or expired an access token. To fix, you should re-
                        authenticate the user.
                403:
                    description: |
                        Bad OAuth request (wrong consumer key, bad nonce, expired
                        timestamp...). Re-authenticating the user won't help here.
        settings:
            authorizationUri: https://example-identity-provider.com/oauth2/authorize
            accessTokenUri: https://example-identity-provider.com/oauth2/token
            authorizationGrants: [ authorization_code ]
types:
    UUID:
        type: string
        description: UUID
        pattern: ^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}$
    OrderItem:
        description: element of an order
        type: object
        properties:
            orderItemId: UUID
            grossPrice:
                type: number
                description: the final sales price per unit of an item
            currency:
                type: string
                pattern: ^[A-Z]{3}$
            productName: string
            quantity: number
        example:
            orderItemId: ca543631-3df7-4d06-8091-179df67c8460
            grossPrice: 12
            currency: PLN
            productName: bread
            quantity: 3
    OrderStatus:
        type: string
        enum: [Pending, InPreparation, ReadyForDelivery, PickedUpByCourier,
        ↪  RejectedByShop, RejectedByCustomer, Delivered]
    Order:
        description: Groceries products order
        type: object
        properties:
            orderId: UUID
            orderItems: OrderItem[]
            creationDate: datetime
            deliveryDate: datetime
            clientAddress:
                type: object
                properties:
                    street: string
                    city: string
                    zipCode: string
            additionalInfo: string
            orderStatus: OrderStatus
            confirmedPayment: boolean
        example:
            orderId: c2cf4e9e-0393-4189-af47-0aab382ce330
            orderItems: [
            {
                orderItemId: ca543631-3df7-4d06-8091-179df67c8460,
                grossPrice: 12,
                currency: PLN,
                productName: bread,
                quantity: 3
            }]
            creationDate: 2021-12-01T12:30:22.52Z
            deliveryDate: 2021-12-01T13:30:22.52Z
            clientAddress:
                street: Prosta 12
                city: Warszawa
                zipCode: 00-631
            additionalInfo: Info from the client
            orderStatus: Pending
            confirmedPayment: false
    ComplaintStatus:
        type: string
        enum: [Pending, Accepted, Rejected]
    Complaint:
        description: Complaint
        type: object
        properties:
            complaintId: UUID
            orderId: UUID
            status: ComplaintStatus
            text: string
        example:
            complaintId: 061ac70b-e370-40ca-a12e-9ea146ae9429
            orderId: e41d4a1b-e771-4eae-84c8-c598ee60d627
            status: Rejected
            text: Delivery was 5 minutes late

/orders:
    /place:
        /{userId}:
            post:
                securedBy: [ oauth_2_0 ]
                description: Add clients order to database
                body:
                    application/json:
                        type: Order
                responses:
                    201:
                        body:
                            application/json:
                                description: |
                                    Successfully created order
                                type: Order
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to create order
                                type: string
    /pending:
        /{shopId}:
            get:
                securedBy: [ oauth_2_0 ]
                description: Get pending orders assigned to shop
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully got pending orders
                                type: Order[]
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to get pending orders
                                type: string
    /{orderId}:
        get:
            securedBy: [ oauth_2_0 ]
            description: Get chosen order
            responses:
                200:
                    body:
                        application/json:
                            description: |
                                Successfully got chosen order
                            type: Order
                404:
                    body:
                        text/plain:
                            description: |
                                Failed to get this order
                            type: string
        /takeForPreparation:
            put:
                securedBy: [ oauth_2_0 ]
                description: Set order state to inPreparation
                body:
                    application/json:
                        type: OrderStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully taken for preperation.
                                type: OrderStatus
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to take for preperation.
                                type: string
        /confirm:
            put:
                securedBy: [ oauth_2_0 ]
                description: Set order state to ReadyForDelivery
                body:
                    application/json:
                        type: OrderStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully taken for delivery.
                                type: OrderStatus
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to take for delivery.
                                type: string
        /reject:
            put:
                securedBy: [ oauth_2_0 ]
                description: The order was rejected by the shop.
                body:
                    application/json:
                        type: OrderStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    The order was rejected by shop.
                                type: OrderStatus
                    404:
                        body:
                            text/plain:
                                description: |
                                    Order not found
                                type: string
/complaints:
    /create:
        /{userId}:
            post:
                securedBy: [ oauth_2_0 ]
                description: Add clients complaint to database
                body:
                    application/json:
                        type: Complaint
                responses:
                    201:
                        body:
                            application/json:
                                description: |
                                    Successfully made a complaint
                                type: Complaint
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to make a complaint
                                type: string
    /pending:
        /{shopId}:
            get:
                securedBy: [ oauth_2_0 ]
                description: Get pending complaints adressed to the shop
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully got pending complaints.
                                type: Complaint[]
                    404:
                        body:
                            text/plain:
                                description: |
                                    Failed to get pending complaints.
                                type: string
    /{complaintId}:
        get:
            securedBy: [ oauth_2_0 ]
            description: Get chosen complaint.
            responses:
                200:
                    body:
                        application/json:
                            description: |
                                Successfully got chosen complaint.
                            type: Complaint
                404:
                    body:
                        text/plain:
                            description: |
                                Failed to get this complaint.
                            type: string
        /accept:
            put:
                securedBy: [ oauth_2_0 ]
                description: Accept complaint
                body:
                    application/json:
                        type: ComplaintStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Accepted complaint.
                                type: ComplaintStatus

                    404:
                        body:
                            text/plain:
                                description: |
                                    Complaint not found.
                                type: string
        /reject:
            put:
                securedBy: [ oauth_2_0 ]
                description: Reject complaint.
                body:
                    application/json:
                        type: ComplaintStatus
                responses:
                    200:
                        body:
                            application/json:
                                description: !
                                    Rejected complaint.
                                type: ComplaintStatus
                    404:
                        body:
                            text/plain:
                                description: |
                                    Complaint not found.
                                type: string
/products:
    /{productId}:
        get:
            description: Get product information
            responses:
                200:
                    body:
                        application/json:
                            description: |
                                Successfully got product info.
                            type: Product
                404:
                    body:
                        text/plain:
                            description: |
                                Product not found.
        delete:
            securedBy: [ oauth_2_0 ]
            description: remove product from shops's offer
            responses:
                200:
                    body:
                        application/json:
                            description: |
                                Succesfully deleted product.
                404:
                    body:
                        text/plain:
                            description: |
                                Product not found.
    /category:
        /{category}:
            get:
                description: Get all products in the specifies category
                responses:
                    200:
                        body:
                            application/json:
                                description: |
                                    Successfully got products info.
                                type: Product[]
                    404:
                        body:
                            text/plain:
                                description: |
                                    Category not found.
    get:
        description: Get all products
        responses:
            200:
                body:
                    application/json:
                        description: |
                            Successfully got products info.
                        type: Product[]
    post:
        securedBy: [ oauth_2_0 ]
        description: Add product to shop's offer
        body:
            application/json:
                type: Product
        responses:
            200:
                body:
                    description: |
                        Succesfully added product.
                    type: boolean
            404:
                text/plain:
                    description: |
                        Failed to add product.
                    type: string
```