university of groningen

Final Design Document

Problem Analysis And Software Design

Authors:
Andreea Cristina Zelko - s4311833
Tudor Roscoiu - s4308492

Lecturer:
dr. Jan Salvador van der Ven

Contents

1 Team Contract

Group 54 is composed of Andreea Cristina Zelko and Tudor Roscoiu. We have decided to design an advanced software system for the RETHINK supermarket using some of the latest technologies such as AI, blockchain and many more, that is easy to use, has a friendly layout and that takes care of customers' needs in a really innovative way.
Our goals as a team are to provide a well-put together project that meets all requirements, to make sure that the work is distributed evenly and to aim for a high grade while also enhancing our knowledge with every assignment that we submit.

1.1 Commitments

As a team we will:

- Distribute the tasks evenly.
- Work hard on successfully completing our assigned part of the project.
- Focus on what is best for the project as a whole.
- Review the other teammates' part of the project.
- Be open to suggestions/improvements/comments coming from the other team members.
- Submit the assignments on time.
- Aim for a high grade.

1.2 Communication

As a team we will:

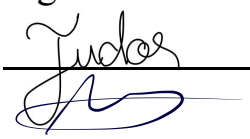
- Communicate through the Whatsapp group chat (text) and Blackboard Collaborate (audio & video).
- Respond to each other's texts in no more than 6 hours on *working days* and 12 hours on the *weekends*.
- Communicate serious matters that might arise and that would result in the delay of completing our assigned part or the project overall.

1.3 Participation

As a team we will:

- Participate in the tutorials in our designed time-slot every week.
- Be active and engage in discussions with our team.
- Participate in a weekly Blackboard audio call to work together and talk about our improvements to the project as a whole.

Signatures:



2 Scope

2.1 Overall

Our focus is on offering an assisted shopping experience tailored to customers' needs that will improve satisfaction by making it easy to shop and pay for items.

This will be achieved by creating an online supermarket system that allows clients to browse through a list of available products, add them to a basket, and place orders. This list will contain basic details about every product and the number of products in stock as the only limitation of how many of a single product one can order. The bill and the order history will be easily available to the customer in their user profile.
Additionally, since the target audience is international, the user will have a wide range of languages to choose from when accessing the shop.
Lastly, the system will handle both manual and automated restocking of the store inventory.

It has been decided that the system will not include handling of deliveries to customers, as the focus is on stock management. Facial and fingerprint authentication will not be used, and the software will not give any financial information like an analysis of the profit gained. Employee management is not within the scope of the system either.

2.2 Minimum Viable Product

The MVP of this project will focus on the backend and database elements of the system.

It will contain the following functionalities: restocking (manual and automatic), viewing product details and placing orders, which will be accessible through a simple RESTful API.
The frontend and UI of the system will not be the focus of the MVP, but a simple interface for accessing the backend through the API will be necessary for testing. Well built and nicely designed UI will be added later in the development process as implementing the basic functionalities takes priority.

2.3 Proof of Concept

The scope of the proof of concept is a smaller version of the overall scope of the whole system.

We will build a simple backend that has access to a database of products. The user will be able to place products in a basket, view information about the stock of the supermarket and place mock orders to the system. If possible, we will also demonstrate how the language translation can be implemented into the UI.
This choice was made because the PoC will be used to demonstrate that all the different components of the system (payment API, supplier API, translation API, etc.) can work together smoothly.

3 Stakeholders

The stakeholders of our RETHINK Supermarket system are grouped by users, customers, negative stakeholders and others as follows:

3.1 Users

- *Cashiers*
  Their job is to scan the products at the checkout, take the payment from the client in multiple forms and ultimately, to print the receipt of the transaction.
- *Restaurants*
  They order large quantities of products from the supermarket, mostly in bulk.
- *Students*
  They order products that are cheap, convenient and/or on sale.
- *Elderly people*
  They need a simple-to-read receipt, the products and their quantities in the order history.
- *Foreigners*
  They need a system that is offered in their language of choice, including the products cart and the receipt.

3.2 Customers

- *RETHINK Supermarket*
  The company that the system is designed for, according to their needs and requirements.
- *Franchise owners*
  These are the people that own the RETHINK Supermarket at a particular location.

3.3 Other stakeholders

- Security experts
  They are the people that maintain the integrity of the system and prevent data breaches, cyber-attacks and other criminal activities that could compromise it.
- Product managers
  They oversee the products that are currently for sale in the supermarket and deal with the possible issues regarding a specific product and the lot that it belongs to.
- Developers
  They are the ones that are responsible for building the supermarket system according to the customer’s requirements.
- Testers
  They work with the developers and their job is to test for possible bugs and errors that might occur when using the system, in order to offer a smooth experience for the users.

- Law Firm
  They take care of the legal aspects of the RETHINK Supermarket system and the way people use it, such as: copyrights and licenses.

- Online payment methods providers
  They are the most reliable and trusted card payment companies such as PayPal, Apple Pay and Google Pay.

- Suppliers
  Suppliers are a wide range of people: from local farmers that sell their fresh produce in our supermarket to large distributors that bring in a wide range of products.

3.4 Negative stakeholders

- Self-checkout supermarkets
  These are stores and establishments that compete with the RETHINK supermarket for offering similar time-efficient options for checking out.

- Hackers
  They are the people whose goal is to compromise the security of the system and the data stored in the databases for their own benefit or others'.

- Shoplifters
  They affect the sale of products in the supermarket and have a negative impact on the stock count in the system.

4 Requirements

The requirements for the MVP are represented in the form of user stories together with acceptance criteria, ordered by priority: from the most crucial to less important ones.
This choice was made because we believe that in order to define the system's behavior in a way that is easily understood by both clients and developers we should write user stories as they have a predefined, easy-to-read and easy-to-understand form, they can be easily prioritized and managing them in long-term can be done in a faster and more reliable way than use cases.

1. As a customer, I want to have a list of products and their details at my disposal so that I can see what is available in the supermarket.
   Acceptance test:
   - Show products in the form of a list.
   - Products contain: a name, price in cents, VAT rate, stock number.

2. As the store manager, I want to have a list of products that need to be restocked so that I know what I need to order.
   Acceptance test:
   - Ability to check the number of products in stock.
   - Update store stock when items are bought and when they are restocked.

3. As a customer, I want to be able to manage my cart while I'm in the store so that I can organize my shopping better.
   Acceptance test:
   - Ability to add products to the shopping cart.
   - Ability to remove products from the shopping cart.

4. As the store manager, I want to have products automatically restocked so that I don't have to worry about the products running out.
   Acceptance test:
   - Automatically order and restock a product when its product count drops below 5.

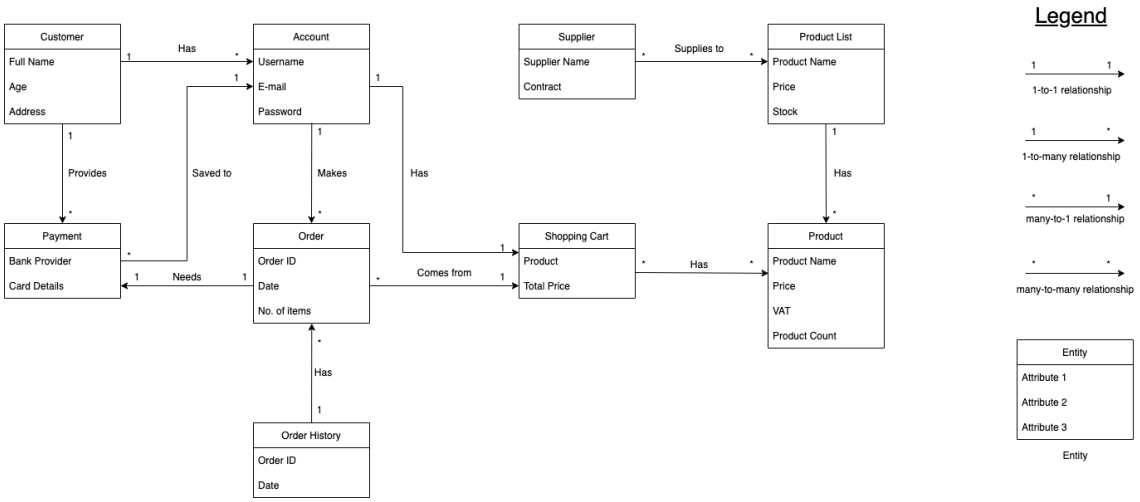
5. As a customer, I want to be able to pay for my own goods so that I spend less time in the store.
   Acceptance test:
   - Ability to enter card payment details and initiate a transaction.

6. As a customer, I want to see my order history so that I keep track of the products that I bought and want to buy again.
   Acceptance test:
   - Show order history in the form of a list.

7. As an international customer, I want to find information about products that I add to the cart in the language that I desire so that I don't have to struggle with the language barrier when I shop in the supermarket.
   Acceptance test:
   - Display a "translate" button in the app.
   - Ability to choose from every language available.

5 Domain model

The domain model was created with the requirements in mind and can be observed in the following illustration.



5.1 Entity descriptions

Customer - user of the supermarket system. A customer is defined by a full name, an age and an address.

Account - user profile in the supermarket system. An account is defined by an username, an email address and a password.

Product - item that is for sale in the supermarket. A product is defined by a name, a price, a VAT (Value Added Tax) and a count representing the number of products in stock at a given time.

Product List - a list of products in a specific order. A product list contains names, prices and stock numbers.

Shopping Cart - virtual cart where products that a customer wishes to purchase are added or removed. A shopping cart is defined by the products that it contains and the total price for them.

Payment - means of paying for products in the shopping cart. A payment is defined by a bank provider and the card details.

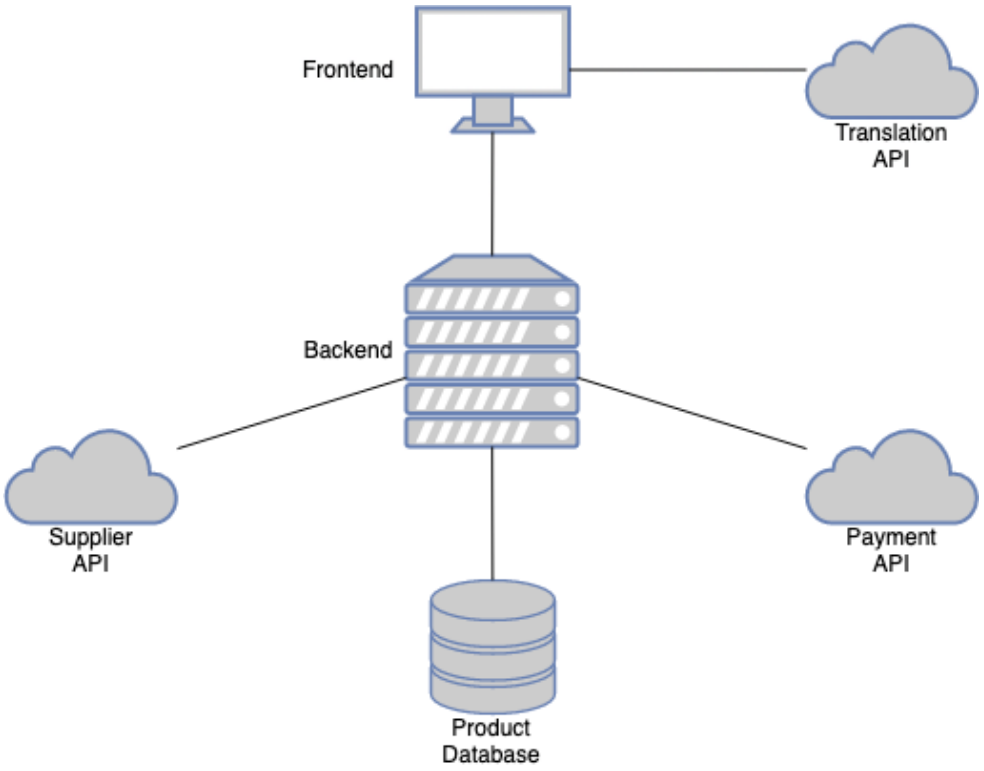
Order - indication of the products that have been purchased. An order is defined by an ID, a date and the number of items that it contains.

Order History - record for products purchased in a supermarket order. An order history is defined by an order ID and a date.

Supplier - provider of products for the supermarket. A supplier is defined by a name and a contract.

6 Architecture Overview

Our RETHINK Supermarket system consists of the database where the products are stored, the backend, the frontend, the connection to the RETHINK Supply API, the payment API and the translation API. The main system will allow customers to add and remove products to their carts and purchase them. The transactions will be stored in the payment API and can be monitored by the person assigned to the task. Data in the database can be manually or automatically altered, according to the conditions written by us in code, and will be updated regularly, according to the products supplied or restocked. The aforementioned will be implemented, connected and managed with Django and its tools.



The system will make use of the following architectural styles and patterns.

- MVT[1]: This is the architectural framework of Django, and thus it is what the system will follow as well.
- REST[2]: This design principle will be used for communicating with third party APIs such as the ones used for payment, translation and ordering supplies.

7 Components

7.1 Frameworks

Backend and Database management

Django has been chosen for the project's backend and database management functionality.

Why? Being on the lookout for a web framework that is simple but complete, we found Django to be the most well-suited choice for our RETHINK supermarket. The syntax is easy to write and understand so no matter what programming language the developers are most acquainted with, they can all work on the project and divide the tasks evenly.

Another advantage that Django offers is the data models. Most aspects of the supermarket system can be designed as separate models, so the different functions of the system would not

[1] MVT: Model-View-Template architectural framework
[2] REST: REpresentational State Transfer architectural style

interfere with others. This allows us to organize the functions better, have a clean environment and find errors with ease.

Ultimately, Django has a tight relationship with databases. It offers support for MySQL, along with others and handles the database all on its own, so there is no previous knowledge required in order to operate a database. By default, Django works with SQLite so there are few things left to be added in order to have a completely working database tied to our system. Therefore, it provides a platform for communication between a user and a database.

Frontend and UI

ReactJS will be used to construct the frontend and design the user interface.

Why? We want to build the frontend using a framework that many developers are familiar with and that is easy to use. ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components.

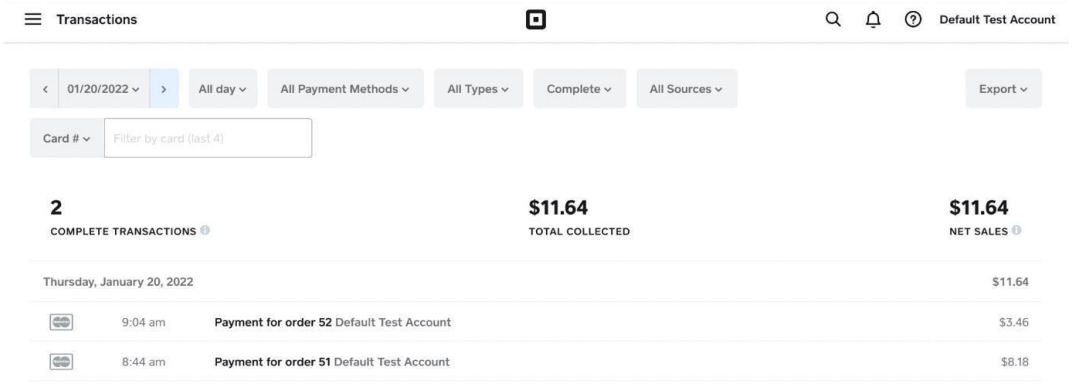
7.2 APIs

Payment API

Our payment API of choice for the RETHINK Supermarket system: Square API.

Why? Square offers an API that can be easily integrated into websites through Django and is made to simplify the payment process. It is a free to integrate service with minimal fees deducted per transaction, it is business-oriented and accepts cards: credit or debit such as VISA, Mastercard, Maestro, but also digital wallets and gift card payments. A big advantage that Square offers is the security through their tokenization scheme which ensures only encrypted payment data goes through.

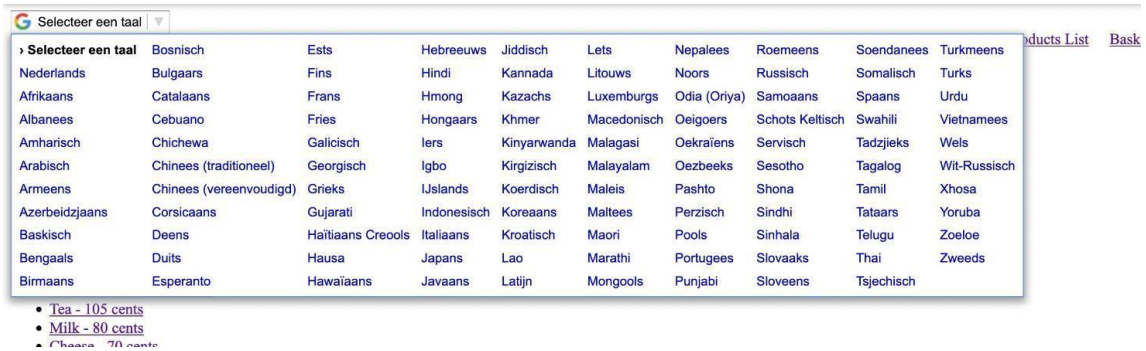
In regards to our system, Square provides customizable checkout pages for both card and POS transactions through their Web payments SDK, making it easy for the developers to choose a layout that is convenient for customers to use. Once a customer places an order, the checkout page where the payment details should be entered pops up and after the transaction is successful, it can be viewed in the Square interface where it contains all the relevant details such as: amount, date and transaction number. This is illustrated in the picture below.



How was the payment API implemented in the system?
We used both the payment API and the checkout page provided by the Web payments SDK of Square. Once the user wishes to checkout, they are redirected to the checkout page. All the order information is sent inside the redirect request. Once the client chooses a payment method and enters their details, the Web payments SDK connects to the Square payments API which charges the client's card with the correct amount. The Web payments SDK is used to ensure that customer payment information is sent and processed securely.

Translation API

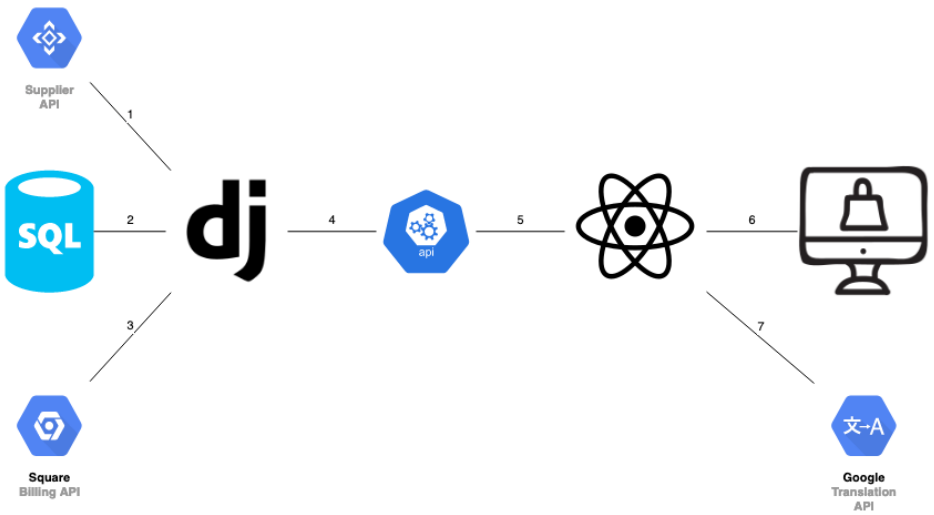
Our translation API of choice for the RETHINK Supermarket system: Google Translate API. Why? One of the most important features of our system that aims to attract customers from all around the world and provide an inclusive environment is the language change feature, provided by Google's API. Google Translate API offers easy implementation, support for 27 languages and a simple interface. It appears on every page of the system in the form of a small button, meaning it is not distracting customers from the most important tasks that the supermarket offers. Below is a picture showcasing the language selection list implemented in the PoC.



How was the translation API implemented in the system?
For the implementation of the language change we used a client that accesses Google Translate API. With this, we were able to add an element into the base.html file of our system that allows the user to select any language to translate into. Placing the element in the base.html file meant that all of our views can be translated, thus making our system accessible for international users.

Other API
RETHINK SUPPLY API
How was the RETHINK Supply API implemented in the system?
The Supply API is used in the restocking procedure of the system. Everytime an item is removed from stock (usually because a client has purchased it), the system checks the remaining available stock for the product.
If the remaining stock is below a predefined threshold (in the case of our PoC, we set the limit to 5, but in an actual use case, the limit would probably be closer to 50), then the system begins a restocking procedure for that item. The Supply API is accessed, a new order is opened, a new orderline containing the desired number of items is added to the order, and finally, the order is placed.

7.3 Interactions between components



1. **Django & the Supplier API**
   The Django backend communicates with the product supplier through the given Restful API. Calls towards the supplier are made when the store stock is automatically refilled. The backend opens a new order, adds the required products, then places the order.

2. **Django & SQLite**
   Django uses SQLite for database management by default, making use of the Model class to easily interact with the contents of the database. In the case of this project, the database contains information on the stock of the supermarket (products that are in stock, products that are in a shopper's backet, products that have been ordered, etc.)

3. **Django & Square API**
   The Django backend uses the Square payment API for handling transactions with customers. Square offers an SDK that is used to save credit / debit card information, create orders with unique IDs and send orders to be billed to the API through calls that follow the REST principle.

4. **Django & the RETHINK API**
   RETHINK functions using the client - server architecture so Django is used to run the backend of the project, creating a RESTful API for the frontend to access.

5. **RETHINK API & ReactJS**
   The ReactJS frontend of the system places calls to the backend through the API.

6. **ReactJs & Google translate API**
   The frontend places calls to the Google translate API in order to offer support for many languages for the UI

7. **ReactJS & the UI**
   ReactJs is used to build the UI displayed to the customer when they browse the online supermarket

8 Proof of Concept

The PoC was designed to show that all the components mentioned above can function well together and produce a working, usable product. For this purpose, all the components of the system are present in the PoC. Below is an explanation of what was done.

Django was used to manage a simple SQLite database which contained all the products available from the supplier, a basic stock of the supermarket, basic information about a user, as well as keeping track of a user's basket contents. This was achieved through the use of the models.Model class, which allows database information to be treated as objects, making it very easy to connect new function definitions to each type of object. With this, we were able to implement the automated restocking functionality, where a new order is placed to the Supplier API every time an item in the inventory is running out of stock (less than 5 products remaining). This is done by sending requests to the RESTful API through which we communicate with the supplier, and ordering the necessary products.

Next, we used the Square Web payments SDK and implemented calls to their API, making it possible to place orders to the RETHINK supermarket and pay for them. Using mock debit card information, we were able to see transaction histories for the shop, as well as very basic financial information, like gross profit.

The functionality explained above was then implemented behind a RESTful API for the frontend to use. Since our focus was on basic connections and usability of the components, we did not invest a lot of effort into the aesthetics of the UI. We created a simple user interface using both ReactJS and Django (ReactJS was used for showing the checkout pages provided by Square, while Django presented the store inventory, basket contents, and user information).

The last component that needed to be included was the translation API. We added a ‘Translate’ button to the UI and connected it to the Google Translate API, making it possible to now switch between over 50 languages. With that, our PoC demonstrates that the design of the RETHINK supermarket system is workable and capable of creating a usable final product.

9 Learnings from the Proof of Concept

In implementing the proof of concept we learned how to access and use third party APIs as well as gaining experience with using the Django framework and working with python. We also found ourselves tackling challenges we did not anticipate and having to make additional design decisions as we went along. One example of such a situation is when we had to decide how to handle the multilingual aspect of our system. We were unsure at what level the translation should take place, but in the end we chose to translate the final displayed views since it was the most straightforward approach.
We also learned that we should look for services that offer REST APIs as they are easy to implement into our system.
Lastly, we learned that the Django framework is well suited for this project, as it provides easy adaptations from database entities to model classes, which can in turn contain functions for easy manipulation of the database. This way, we are able to manage stock items with ease and place restock orders to our suppliers.

9.1 Requirements vs PoC

When looking back at the system requirements after implementing the PoC, we realized that certain aspects of the project might take more time and effort than first estimated.

For example, we anticipated that implementing the automated restocking would be more difficult than just implementing a way for store workers to manually place orders to the supplier. However, we came to the conclusion that manual restocking involves allocating users with permissions different from the ones regular customers have, as well as creating a UI for the managers to use when ordering. This is contrasted by the ease with which automated restocking was implemented in the backend of the PoC.

On the other hand, now that we have hands-on experience with the project, we came up with ideas that could become requirements for future iterations of the system. The fact that Django models were easy to work with gave us the idea of adding lists to a user profile (e.g. ‘Favorites’ or ‘Wish List’). This is not currently in our list of requirements, but we are considering adding it at a later stage of development as it would improve customer satisfaction.

9.2 Decisions

Below is a list of the decisions that we have documented during the process of building the PoC.

- Our initial plan was to use only Django for all parts of the system, but we decided to add ReactJS to our components because we wanted to offer a good customer experience through great UI and UX.
- We considered the possibility of making our product an app using Java or Python, but we concluded that a web-based store which is built using the REST principles fits better with the requirements of the RETHINK supermarket.
- We tried to keep the models of the system as simple and as straightforward as possible. Therefore, we are not yet storing delivery information from the supplier API.
- We decided to implement a checkout page for a card transaction because Square provides a dummy card that we could use during the checkout process for the presentation.

- We decided to have the restock process be triggered when product stock runs lower than 5 items since we do not find having a large database only for the purposes of demonstration necessary.
- We decided to implement the translation at the top most layer as we figured there is no need to store product information in different languages.

10 Teamwork Evaluation

10.1 Andreea’s opinion

I am happy with how me and Tudor were able to work together. He was available when we needed to discuss something or work together. He was also understanding and respectful of my time and availability.
I believe the work was spread evenly between the two of us and that we both followed the team contract that was set at the beginning of the project.

10.2 Tudor’s opinion

Andreea and I got along very well during the entire project. She is very hardworking and helped us focus on the most important aspects of our system. The workload was divided evenly since both of us worked on the same things simultaneously and I am really happy with the rapidness and accuracy that she showed when answering my questions. We encountered small misunderstandings but Andreea and I managed to solve them immediately so that we would achieve our goals. The team contract was followed entirely so we worked in a very efficient manner.