November 6th 2022

Quicc

Faster groceries-shopping experience

Group 6, Group Members:
Liran Neta s2596415
Robert Ignat s2565609
Chirita Maria Simona s3098508
Coremans Sören Willem 3104672
Craciun Alexandru-Andrei s3103579

November 6th 2022

Introduction

Have you ever felt like that girl in the picture above?
In other words - Have you ever sensed as though the time in the supermarket is being wasted by looking for a specific product?
Maybe a Basmati Rice? And what about the tomato sauce your girlfriend is asking you to buy for more than a month?
Will you come back home with the same excuse of "Sorry, I could not find it and I didn't want to disturb the employees"?

To understand the problem more deeply, let us introduce you to Tom, who is about to host a dinner party for some Friends. He already has a great meal planned and a long list of all ingredients he needs to prepare it. To acquire everything, he needs to go shopping at the local Supermarket. He quickly finds the most common ingredients, but has trouble with some of the more unusual ones. Finding most of these takes him so long, he is now behind schedule and even worse there is still one item missing. To locate it he asks an employee currently stocking the shelves, but even he cannot find it and afterwards he asks one of his colleagues, they explain to him that the item he is looking for is currently out of stock. He gets home far too late, and his rushed meal is missing an important ingredient. Now he has disappointed his guests and they leave slightly annoyed.

Tom's story and the issues faced during it is something almost everyone has had to deal with sometime in their lives. Many people spend hours in a grocery store only to come home without all the ingredients, either because they missed them or because they forgot to buy them after the frustration of time-wasting in the store. Quicc aims to fix this common problem, by creating a mobile-app with the corresponding features (explained later in this report).

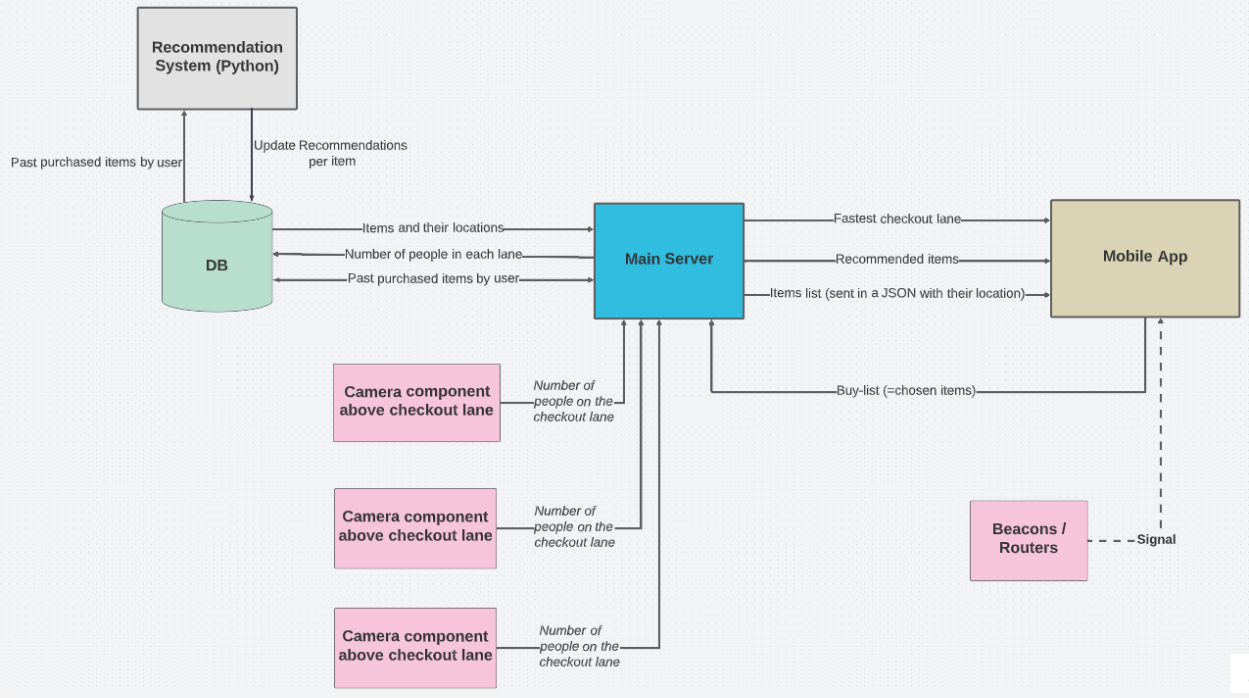
In this report, we will describe in detail how Quicc is structured, functioning, and serves the user the most suitable way. We will start by introducing the use cases and the functionalities of the system, in addition to its specific algorithms. Later, we will discuss the development process of the component and their integration. We will elaborate on the test plan and evaluation metrics we used to evaluate the fitness of the system to the initial requirements. You can also find more elaborative details about -how the work has been completed eventually, and what adaptations we came with, along the way.
Furthermore, the test results are shown,and discussed. Finally, a further analysis and suggested improvements are concluded as well.

Enjoy Your Reading!

Tasks division:

- **(Sören)** Calculate the most efficient route from entrance to checkout, for the user to pick up all items on the list (**frontend**)
- **(Liran)** Report (except for the individual reports)
- **(Liran)** Central Server with (**backend**):
    - Location of items in the shop
      Map the user's desired items locations in the store (hard-coded locations).
    - Number of people in each checkout lane
- **(Simona & Robert)** Develop mobile app capable of getting the list from the user and showing the (custom) map
- **(Andrei)** Use Bluetooth beacons to localize user and show location on the (custom) map, together with optimal route
- **(Robert)** Use top-down camera to estimate how many people are at each checkout (route the user to most empty checkout place)
- **(Andrei)** (backend) Use past purchase data combined with what the user is currently buying to recommend items that are already on the path

System Architecture



Functionalities

- **User requirements specification**
    - As a user, I want to be able to choose the items I want to buy from the store
    - As a user, I want to see the most efficient route from the entrance to the checkout point, based on my chosen items. The route will be shown on a map of the store.
    - As a user, I want to see the products location on the map as well
    - As a user, I want to see my location on the map, while navigating in the store and grabbing the items (the localization is done using trilateration)
    - As a user, I want to see recommended items I might be interested in, in accordance with the items that are already on my route.
      An elaborative explanation of the algorithm behind the recommendation system is described later here.
    - As a user, I want to know the fastest checkout lane

- **Mobile App (Android, Java)**
    - Get a list of all the items in the store, with their relative location (e.g. the banana is located on the aisle between nodes 3 and 5, 40% from node 3 [and 60% from node 5]).
    - Get a list of recommended items for the user, based on our algorithm (to be explained later) and based on the current route.
    - Get the fastest checkout lane
    - Send to the main server the items the user plans to buy ("Buy-list")
    - Receive signals from the beacons, to detect the location of the use

- **Server (Node.js)**
    - Get the items and their relative locations from the DB
    - Get the number of people on the checkout lanes, from each camera, and store it in the DB
    - Store past purchased items of the users in the DB (for the recommendation algorithm)

- **Camera component (Kotlin, TensorFlow)**
  Each camera will be located above one specific checkout lane
    - Estimates the number of people that are waiting on the checkout lane, using computer vision* ; and
    - Sends it to the main server

      * In our MVP, the fastest checkout is determined based on the number of people on the checkout lane. The more accurate calculation of the fastest lane is based on the number of products of people. Due to time and resources limitations, we decided to use a number of people instead of products.

- **Recommendation System (Python)**

  The recommended items are determined using a recommendation algorithm, which is divided into 2 parts:

  1. A Python script that fetches all the purchases made by users and utilizes collaborative filtering to match items based on their rate of being purchased together (inspired by this article).
     To keep the results up-to-date, the script runs every couple of minutes.
     For each of the items, the script builds an ordered list of items, from the most-recommended to the least-recommended item, as shown here:

     ```
       item: "Apple"
     ▾ recommendation_list: Array
         0: "Beer"
         1: "Chocolate"
         2: "Eggs"
         3: "Banana"
         4: "Milk"
     ```

     That list is saved on the DB, and then fetched by the user.

  2. The displayed recommended items are those which fulfill the following "recommendations-criteria":
     a. Not in the user's buy-list
     b. Found as highly recommended, based on the recommendation_list of an item the user is currently buying
     c. Located on the current route in which the user is going

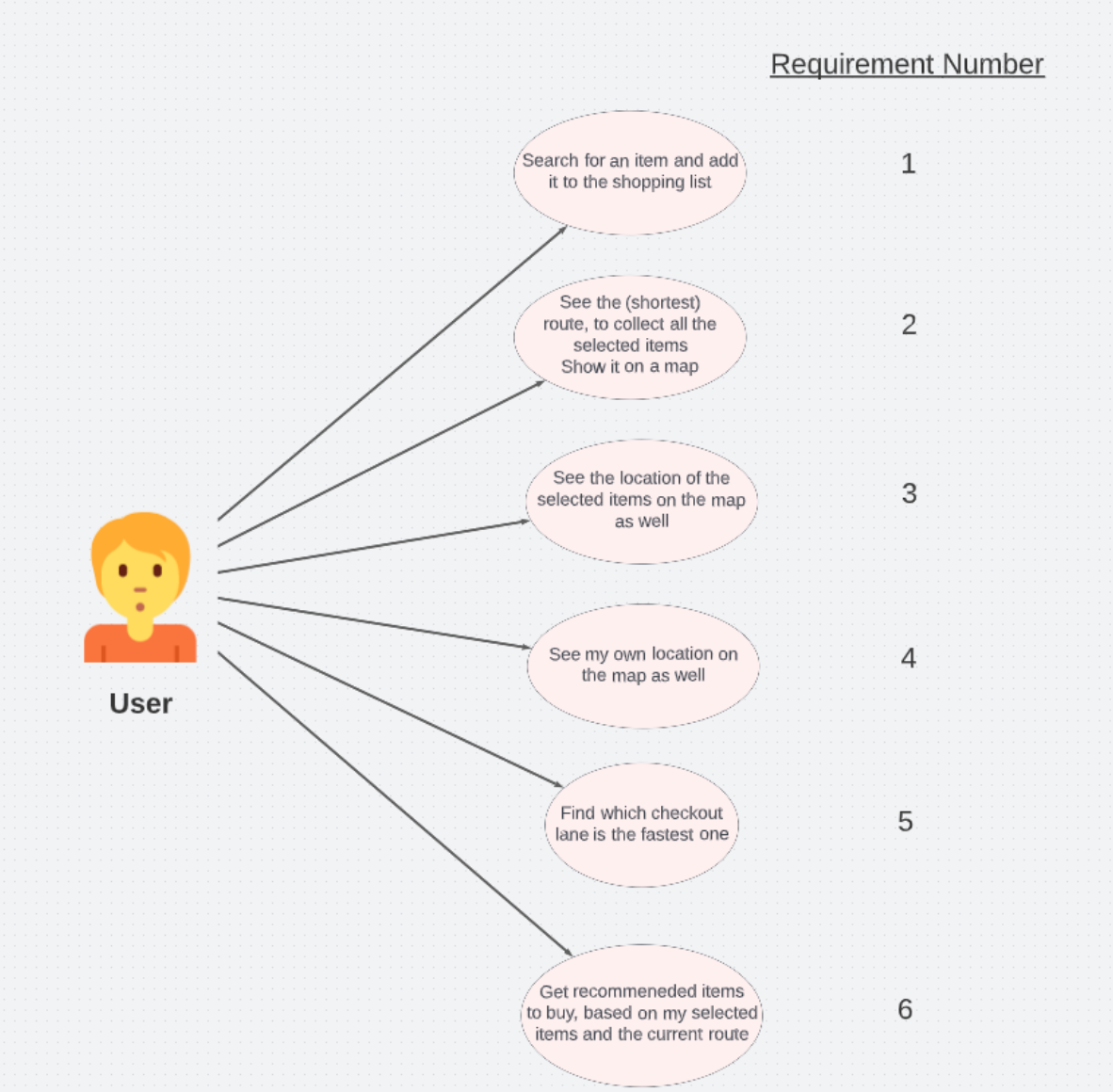
- **Most-efficient route algorithm (implemented as part of the Mobile Application)**
  1. The Algorithm must calculate the fastest route between the entrance and the checkout lanes while also passing all items as specified on the shopping list (the location of the items is shown as a relative location*)
  2. Return this route in a useful way that can be easily used to be drawn on a map and shows where each product is located.
  3. Must be able to calculate all products on a given route, excluding the ones on the shopping list as the route was calculated (to be later used in the recommendations algorithm)

     * Relative location: between which 2 nodes ("from" and "to") is the item located?
     Where is the item on the line that connects between these 2 nodes (in percentage, beginning from the "from" node)

November 6th 2022

Use case Diagram

This diagram demonstrates the user interaction with the interface (the main requirements)



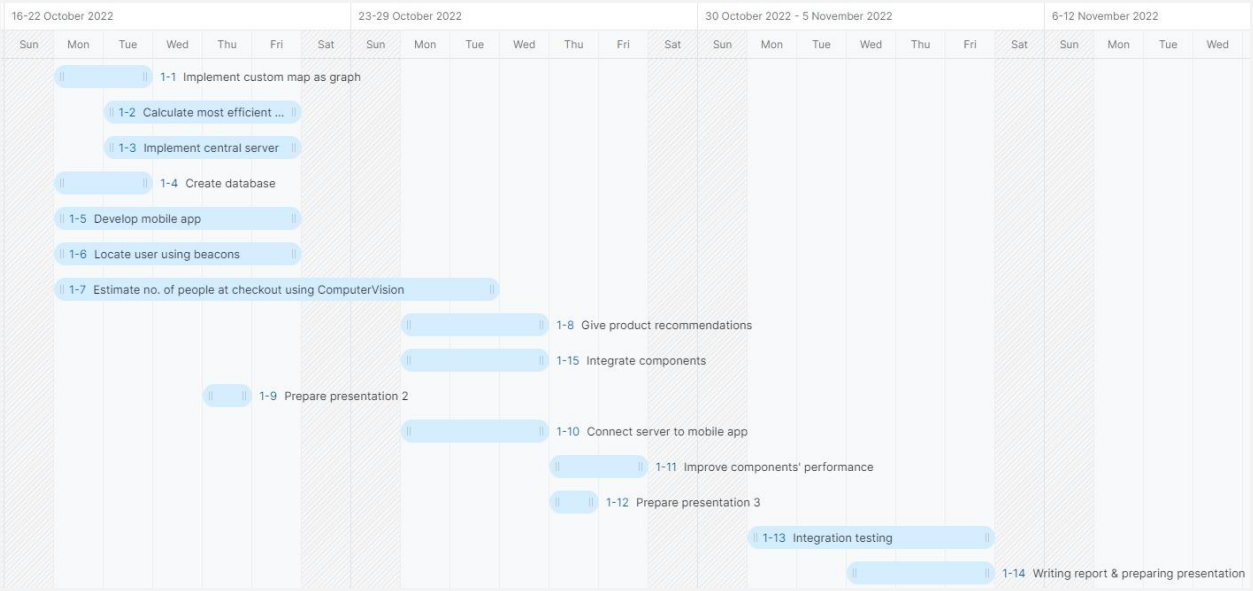
Interaction between users and the system

- The user starts building a new shopping list and the system will open a search bar. The user will then be able to type in the name of a desired item and the system will suggest items available based on the user's input.
- The user sets the shopping list and the system will create a path on the map that (1) goes through all the items on the list and (2) shows all the items on the map.

- After the system calculates the route it will give the user a list of recommended items on the same route.
- The user is able to ask the system what is the fastest checkout lane and get the indication back from the system.

Development planning

The project is planned to be completed within 3 weeks. For that purpose, we have built a Gantt chart that helps us understand the tasks to be done, their order, and their division among the group members.



Test plan

Even a well-designed system can be crushed. To avoid unexpected bugs, there is a need to test the system prior to its release. Hereby is a detailed plan of our system testing, consisting of the components (=units) and their integration testing.

In this section, we mention the test plan for all the use cases (requirements) of Quicc system:

<u>Requirement 1 - Search for an item and add it to the Buy-List</u>

1. Create a collection of Items and their relative location in the **DB**
2. Sent a GET request from the user to the **server** to fetch all the items from the DB (API URI /items)
3. Verify, using **Postman**, whether the GET request indeed returns the desired outcome (all the items)
4. On the **Mobile App** itself, search for an item and add it to the buy-list. Check on the app itself if the user buy-list is updated when items are added.

Requirement 2 - See the (shortest) route to collect all the selected items + Show it on a map

1. Design a route with a couple of items (their relative location is predetermined) and check manually what is the shortest route to collect all of them.
2. Call the function that calculates and returns the shortest route to collect all the selected items. Send to the function the selected items as a parameter.
3. Display the route on the map, and check whether it is equivalent to the expected outcome.

Requirement 3 - See the location of the selected items on the map as well

1. Select a couple of items with a predetermined (relative) location.
2. Evaluate where on the map the items should be located, based on the nodes locations and the distance from them (in percentage).
3. Display them on the map, and compare against the expectations

Requirement 4 - See my own location on the map as well

1. Get the user location using the trilateration technique (in a space where there are beacons around)
2. Predict where on the map the user should be shown, based on the current location
3. Use the function we have built to convert the GPS coordinates into X Y location on the phone display.
4. Compare the result against the initial prediction

Requirement 5 - Find which checkout lane is the fastest one

1. Run one of the Camera Components and aim the camera at a couple of people in the room
2. Check whether the **POST** request that is sent from the **camera component** to the **server** contains the correct number of people.
3. Verify on the **server** side that indeed the correct number of people and "lane ID" were sent
4. Verify whether the **DB** is modified accordingly (the server updates the DB when it receives the POST request from the camera component)
5. Repeat steps 1-4 on 3 other Camera Component, such that the number of people differs every time
6. Check what is the return value of the **GET** request that returns the **fastest lane**. Verify whether this is the correct and expected result or not.

Requirement 6 - Get recommended items to buy, based on my selected items and the current route

1. Add a couple of **Dummy-Purchases** into the DB by sending a **POST** request to the **server** (with the help of **Postman**)
2. Verify whether the data is kept correctly on the **DB** ("purchases" collection)
3. Run the **Python Script** for generating the recommendations for each item

4. Check the recommendations as they appear now in the **DB**
5. Ensure they make sense, based on the stored purchases
6. On the **Mobile-App** - add a new buy-list and see whether the recommended items are matching the "recommendations-criteria" (described in more detail in the Functionalities section).

Evaluation Metrics

The system may pass the tests, but there is still a need to measure and evaluate the system's and components' fitness to the requirements.
To determine the succession of the system and its components, we use the following table:

User Requirement	Number of tests performed (each test has different data)	How many of the tests resulted in the expected outcome?
Requirement 1	3	For instance: 2 / 3
Requirement 2	3	For instance: 3 / 3
...	...	...

Each of the requirements tests (as described here) will be performed 3 times. In each of the tests, the data would differ, to test the reliability of the system in different cases.
The evaluation is all about comparing the outcome with the expected results. The higher the accuracy is - the better the system fits the requirements.

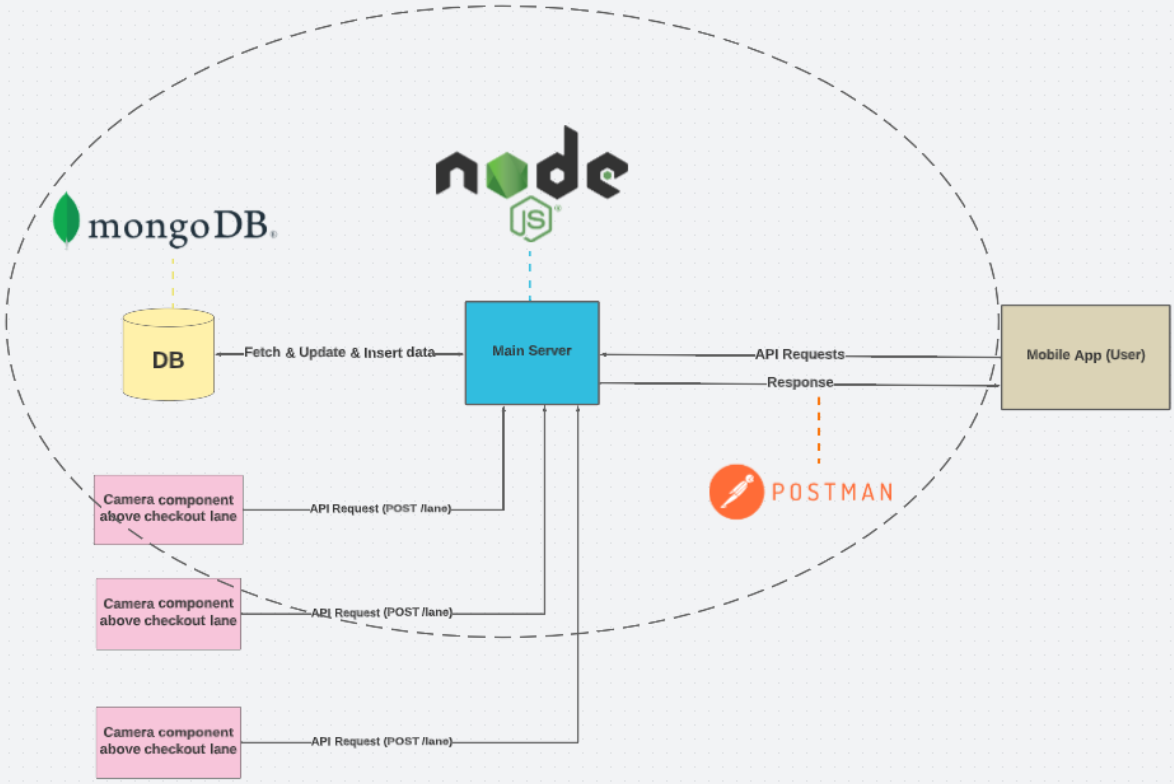
November 6th 2022

Component Development

Main Server & DB (Liran Neta)

The Quicc system consists of multiple components - the cameras above the lanes, the user interface and the Database. All of these should be connected in a way to transfer data to and from each other.
This is the **main functionality** of the server - listening to new requests (on an API-basis), manipulating the DB accordingly, and sending a response in return.
The **Database** (=DB) stores all the relevant data of the system. The data is fetched from the main server based on the user's actions.

Architecture



NodeJS (Express)

The framework being used to build and run the main server is Express JS (based on NodeJS).
The reason behind using this framework was fast growth and ease-of-use, due to the familiarity of the developer with the technology.

MongoDB

All the data in the system should be easily stored and accessed. There is no need for a rigid schema. Instead, there is a need for flexibility and scalability.
In MongoDB, data is stored in collections, which are like tables, and documents, which are like rows. Each document is a JSON representation of the data.

In our system, we have designed the following schemas:

Users collection - all the users of the system:*

```
{
        Id: "...",
        Fullname: "..."
}
```

Items - all the items in the store:

```
{
        Id: "...",
        Name: "...",
        Location: "..."
}
```

Lanes - all the checkout-lanes in the store:

```
{
        Id: "...",
        Name: "...",
        Location: "..."
}
```

Purchases - all the purchases made by Quicc users (in order to build the recommendation-system):

```
{
        customer: a reference to the purchasing user (not used currently in our product),
        itemsPurchased: [item1, item2, ...]
}
```

** We do not implement an authentication system for the current project due to its prototypical nature.*

API Requests

The server listens on a specific IP and port number.
Once a request is received from a user, the server launches the corresponding function, based on the request URI (for instance: 192.168.1.1:3000/items).

Here is the table detailing all the API requests the server responds to:

Request	Method	Name in Postman	Use
/items	GET	getItems	Fetch the store's items-list (which the user can choose from)
/item	POST	addNewItemToDB	Add a new item to the store's items-list (by the admin)
/lanes	GET	getLanes	Return all the checkout lanes (the lane-number and the number of people waiting on the line
/lane	POST (Update / Insert)	upsertLane	Update the number of people near a checkout lane, based on the lane-number. Insert a new lane-number in case it is yet not stored on the DB
/fastestLane	GET	fastestLane	Return the checkout-lane number, which has the least amount of people on the line
/purchases	GET	getPurchases	Return all the purchases done by all the users combined
/purchase	POST	addPurchase	Add a purchase done by a user to the Purchases-Collection on the DB
/recommendations	GET	getRecommendations	Fetches the list of recommended-items per item

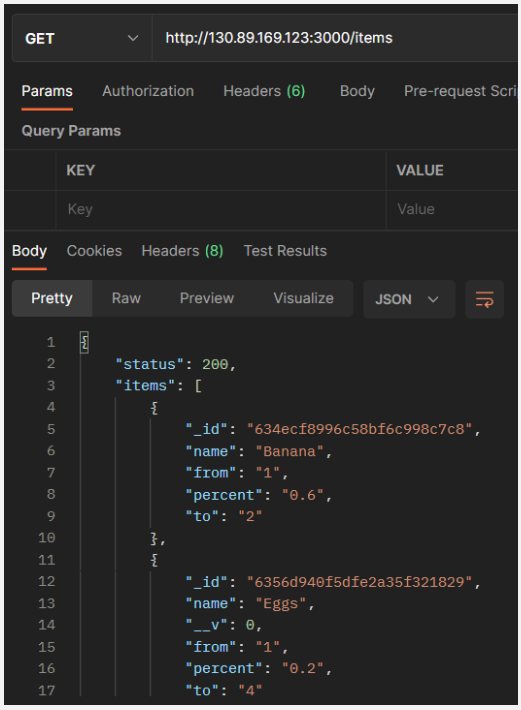
Test results (using Postman)

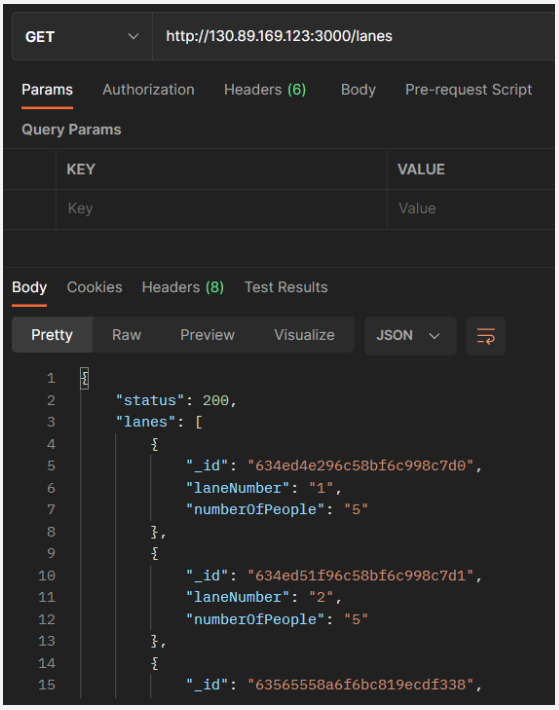
To ensure that all requests are completed successfully, an API request program such as Postman is utilized.

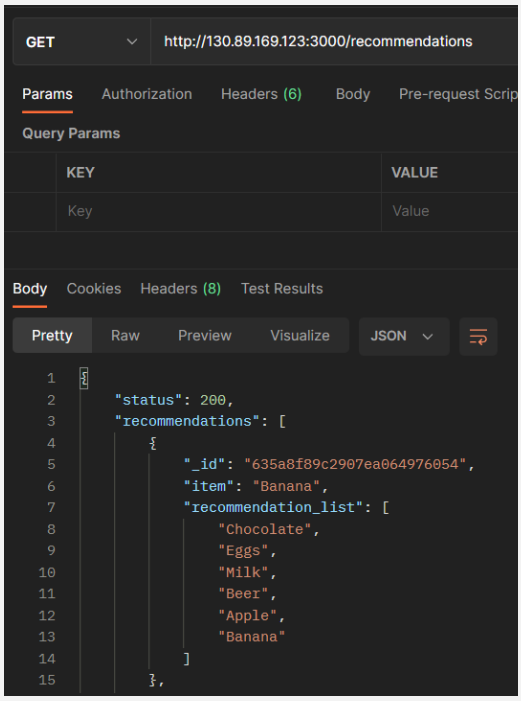
November 6th 2022

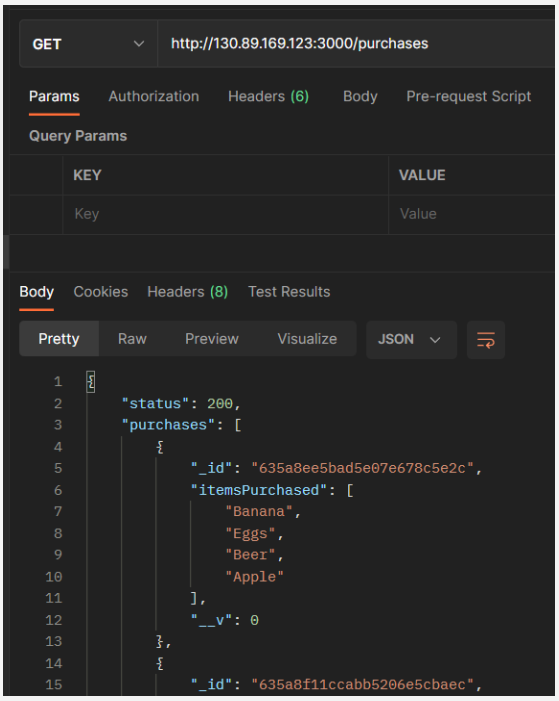
Postman is a software used to test out API requests, as long as the server is running.
By entering the URI (and the request body if needed), we can send the API request to the server.

For instance:

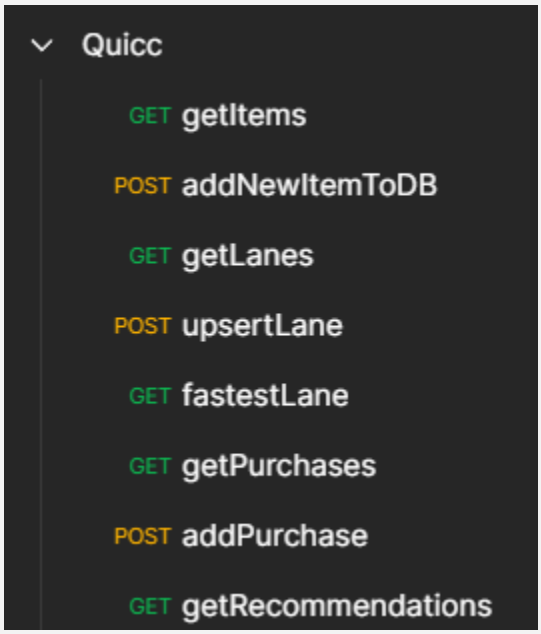








All the requests are saved and can be used immediately later:



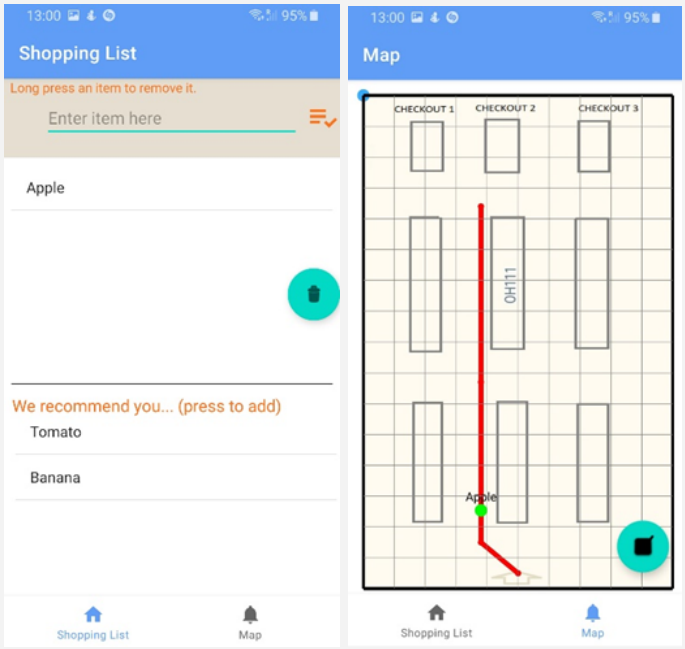
All the API requests resulted in the desired functionality!
In other words - the server and the API requests were successfully tested and produced the expected outcome.

Report (Liran Neta)

To demonstrate the entire contribution that has been made by the team members, we have decided to mention the fact that I have been spending a couple of days on writing this entire report, excluding the individual sections. It took a lot of thinking over the entire process, understanding the problem to solve, and the way we designed and implemented the solution. The process of writing was made in coordination with the rest of the team's work, updating me about the progress, test results and findings.

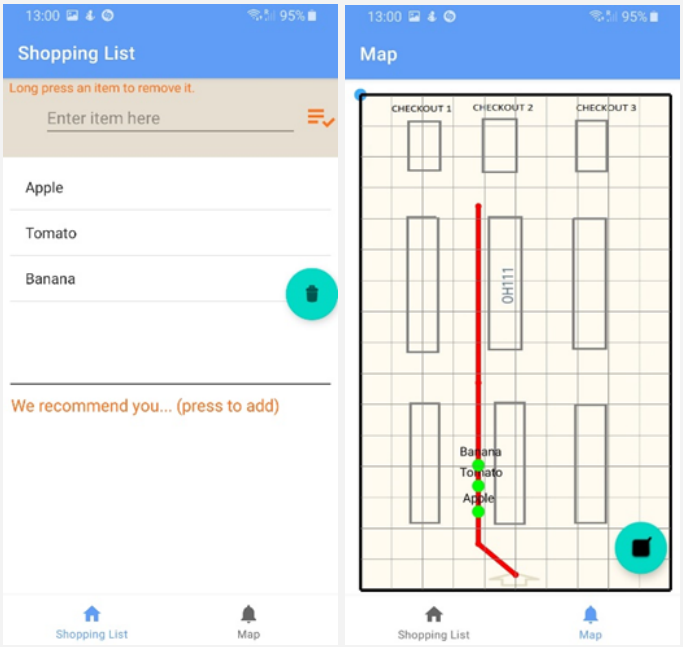
Mobile App (Simona)

In the third week of working on the project, my individual task was to develop the graphical user interface. I chose to develop the mobile app using Android Studio and the chosen programming language was Java.
The mobile app has two activities. The main activity allows the user to create a shopping list by entering items using an EditText and submitting them to the list using an ImageView. The items will be shown in a ScrollView. The user is told that they have the possibility of removing items from the list by long pressing them. Right under the shopping list, a list of recommended items will be shown, which will be created based on the items that the user has in the shopping list, using a recommendation algorithm. If the user finds a recommended item to be useful for them, they can press the item to add it to the shopping list. The main activity also has two buttons: one that deletes the shopping list (as well as the recommendations list) and one that will take the user to the second activity.
The second activity has the purpose of showing the user the custom map, as well as the fastest route (that will allow the user to pick up all the desired items within the shop) and the location of the user. I tested the mobile app using a virtual device, as well as a physical device.
This is how the mobile app looks so far:



In the fourth week, what I did was to implement the GET and POST requests to the main server.
As the diagram presented in the beginning shows, the mobile app posts to the main server the list of chosen items that are currently in the shopping list and requests from the server the fastest checkout lane, the recommended items and the whole list of items available in the store.
In the final week, the custom map has been added as well as the graph and the button from the bottom of the page have been removed and replaced with tabs.

The final look of the mobile app is this one:



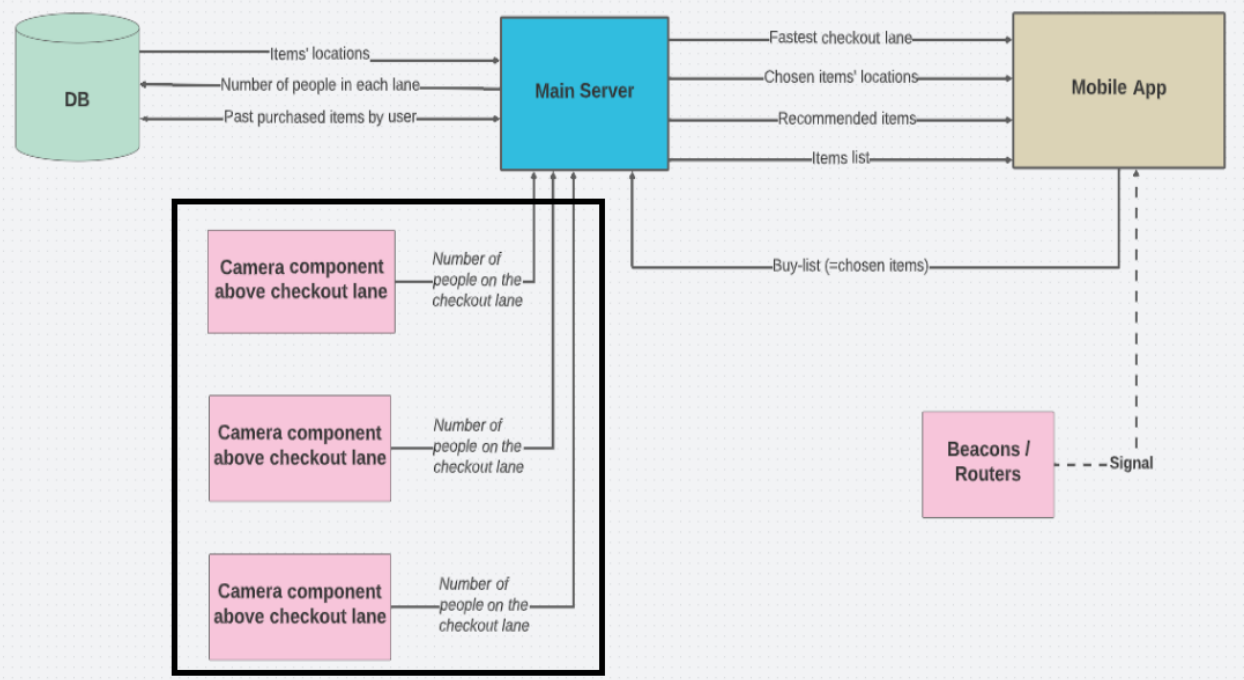
Let's say we add "Apple" to our shopping list; then the recommended items will be on the same path as the apple.

The first testing that I performed was the functionality testing, that stands for testing the different functions throughout the application (like testing the input field for items, the navigation buttons , delete button). One of the first defects that I encountered in the beginning of the app development was the button alignment, but that was later solved and replaced with tabs.

November 6th 2022

Camera Component (Robert Ignat)

Architecture



First week development process

The component that I worked on this week was the array of camera systems that would detect how long queues were at checkout aisles. For the final prototype, we intended to measure the exact number of people at each aisle. We decided to use android phones as cameras instead of Raspberry Pis (mainly because it is not that practical to procure multiple Pis with a camera for each).

However, this came with some challenges. When it comes to data science and machine learning, Python is the most popular language. As a consequence, many machine learning libraries or frameworks are designed for Python, but Android Studio apps run on Java/Kotlin. Therefore, I started searching for libraries or models that would work with android studio.

Sidenote: from a system architecture point of view, the images should be processed on the camera system rather than the central server, both because it might not be a good idea to require the server to constantly process images from all the checkout lanes (can be over 20), and because just sending the results requires less bandwidth .

Android does have an easy to use library called MLkit, but its limitations made it unusable for our project - it can only detect up to 5 objects at a time (which is not restricted to humans only, so it would often detect store items instead of humans). Also, most libraries that I found were optimized for image classification rather than detection.

Eventually, I found a set of Tensorflow models that were trained on the COCO dataset (300 000 + images, 1M+ object instances). I also discovered an example basic app that showcased the model on android. This app was written in Kotlin, so I spent some time learning the language.

After having a decent understanding of it, I studied the sample app until I had a good grasp of how it worked. I then began to modify it, tweaking the detection parameters, removing unnecessary features and processing the detected objects so that the number of persons are counted and http post requests are periodically sent to the server (at the time of writing, the request was just tested using httpbin).

Interaction with the system

The sending of these POST requests (to ./lane as mentioned in the previous section), containing the camera's ID and the number of people in the queue to the server, is the one and only way in which this component interacts with the system, so it is the only "interface".

Initial performance

The model's performance is quite consistent, mainly due to the large database that it was trained on. The site mentioned it has a mAP (mean average precision) of 34% which might sound low, but that is the metric for detecting all possible objects (80 classes).

From all the tests that I made, I noticed that detecting persons is much more reliable than the metric would suggest, and the certainty of the detections is also over 65% most of the time. It is also worth mentioning that the observed false positive rate (people being detected where there aren't any) is very low, as we only encountered it once during manual testing.

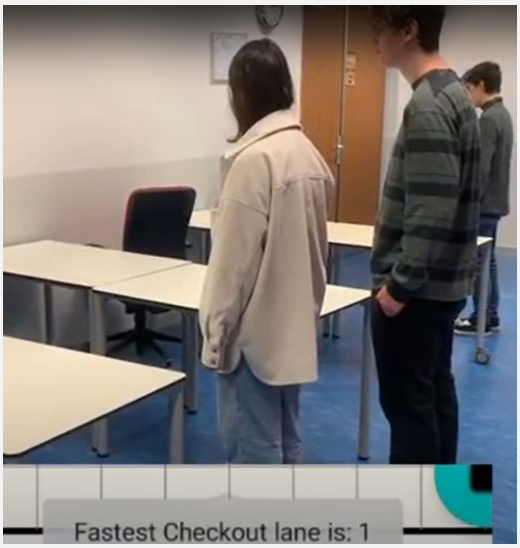
These results make it so that getting the max count from a few detections and sending that to the server every few seconds is a good method to get an accurate estimation of the queue lengths.

Integration

The integration of this component was simple, as it only required the camera app to send a POST request periodically to the server. This was completed on the first day of the second week (the "integration" week) and the camera app did not change since then. Instead, I focused on integrating different components and aiding my colleagues with some of the more complex tasks.

Test results (after integration)

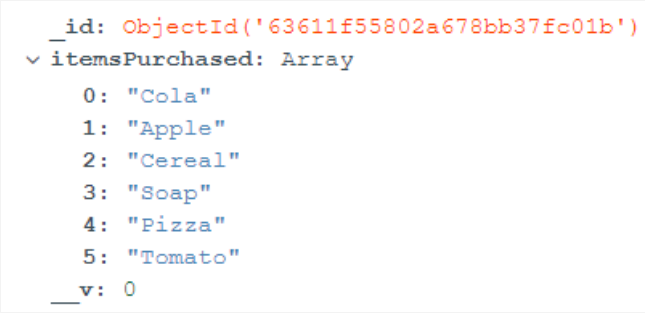
The following pictures show the accurate calculation of the fastest checkout lane (taken from the demo video). During manual testing, we observed that the database receives the correct number of people seen by a camera, with virtually no errors.

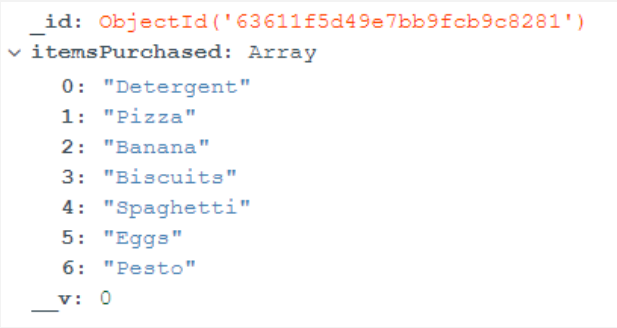


Recommendation System (Andrei)

The component I worked on this week is the Recommendation System.It is based on an algorithm developed by Amazon in 1998, but ours uses old shopping lists to determine which item goes well with others items instead of using user ratings.

Firstly, we create a matrix that has on columns all the items in the database and on the row all the past shopping lists. Then we assign 1 if the current item is in the old shopping list or 0 if it's not.





For example, other users had these shopping lists. The matrix looks like this:

	Cola	Apple	Cereal	Soap	Pizza	Tomato	Banana	Chocolate	Spaghetti	M&M	Pesto	Fanta	Detergent	Biscuits	Eggs
sl_1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
sl_2	1	1	0	0	1	1	1	1	1	1	1	1	0	0	0
sl_3	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1

Secondly, we calculate for each combination of products the similarity value. Let's say we want to find the similarity value between Cola and Apple. First we multiply the value of Cola and Apple in the matrix for each shopping list. After that we add all the values.

A = 1 * 1 + 1 * 1 + 0 * 0 = 2

Then for each object we sum the squared values in the matrix and then calculate the square root of them. After, we multiply the results.

$$B = \sqrt{1^2 + 1^2 + 0^2} * \sqrt{1^2 + 1^2 + 0^2} = 2$$

The final step is to divide A and B so we can have the similarity value between Cola and Apple.

$$sv_{ColaApple} = \frac{A}{B} = 1$$

After calculating the similarity values for each combination of items we get the similarity matrix.

For example, let's take only Cola, Apple and Cereal.

	Cola	Apple	Cereal
Cola	-1000*	1	0.707107
Apple	1	-1000*	0.707107
Cereal	0.707107	0.707107	-1000*

*If we try to calculate the similarity value for the same item we assign the value -1000.
**If we try to calculate the similarity value, but we get 0 for the value B, we assign the value -100.

For the recommendation list, all we need to do is to order each item based on the similarity values.
For Cola the best item is Apple, for Apple the best item is Cola and for Cereal both Apple and Cola are good because, in our example, the one time someone bought Cereal, also bought Cola and Apple.

Reference

Indoor localization (Andrei)

During the third week, I worked on getting the Indoor localization using bluetooth Beacons and trilateration. Using the knowledge I have gathered during the second challenge of this module, I implemented this task with ease.

For the detection, finding the distance and finding different properties of the bluetooth beacons I used a popular library called AltBeacon.

For the trilateration part, I used a library developed by lemmingapex (link). Using both of the libraries I can easily detect the location of the user inside the shopping center.

Problems encountered:

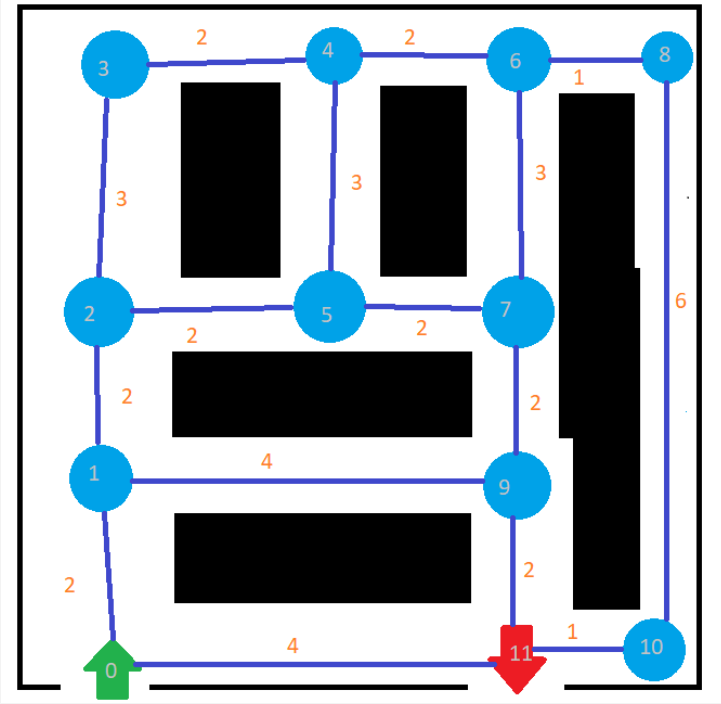
1. Because I didn't have access to beacons to test my task, I had to use different applications to simulate bluetooth beacons. The precision of the simulated beacons wasn't exact, therefore during the testing, the indoor position got a little fuzzy. To be more precise, we did not have the exact coordinates of the beacons, therefore we had to approximate them on the map.
2. Because of the layout of the room, containing lots of steel bars and other people, the signal got noisy, therefore the distance estimation wasn't precise.
3. Because of the limitations of the bluetooth signal, the distance estimations can have errors up to several meters.

November 6th 2022

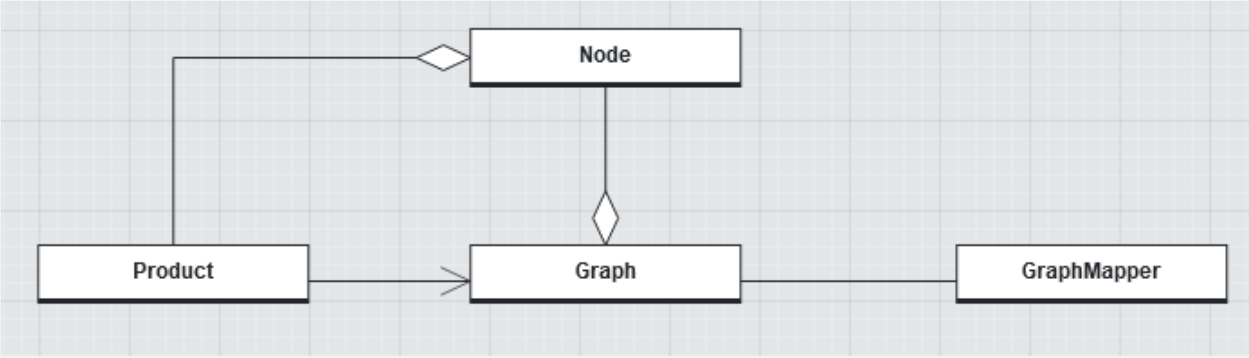
Most-Efficient Route Algorithm (Soren)

General Idea:

Simplified, a store can be described as a graph, with each intersection between aisles being a node and the aisles themselves being edges between the nodes, with the length in meters being the weight of each edge. You must also include the products on the shopping list and make them nodes, with the corresponding neighbors depending on their aisle. If you then make one node the entrance and another the exit and calculate the fastest route between them that also visits all nodes resulting from products, the result is the fastest possible route through the supermarket that passes by all required items.



Architecture



GraphMapper:

The job of this class is to calculate the fastest path on a graph from one node to another, with the additional requirement that certain nodes have to be visited. That all seemed simple enough and it also sounds like a problem that many other people will have solved before. So I started searching the internet for an efficient algorithm I could implement. Unfortunately, while I was correct that this was by no means a new problem, it even has a catchy name: Traveling salesman problem, it earned this name for being one of the most famous NP problems. As I was not confident in my ability to solve one of maths greatest unsolved problems, I had to settle for an inefficient algorithm. The Method I came up with is relatively simple: I used the Dijkstra Algorithm on all the "must visit" nodes and the start node to find the fastest route between them (and to the end node). I then started at the start node, arrayed all the must visit nodes and then added the end node at the end. Using the fastest paths as previously calculated, I then added the resulting path and its length to the possible paths. Do this for all possible combinations of "must visit" nodes and pick the shortest path from all the possible paths.

Node:
The node is mostly a collection of information, which includes:

- Its location
- Its node neighbors
- If it has a product and if yes the product
- The products located between itself and its neighbors
- A unique index
- If it is the exit or not

In terms of functions the node only consists of setters, getters and a clone function.

Product:
The product similar to the node also has little in the way of functionality and is mostly used to store information, including:

- Its name
- On which aisle it is and where exactly on that aisle it is

Graph:
The Graph is the class that binds all others together and serves as the intersection for all who wish to use the algorithm. Its main job is to integrate a list of given products into the graph and execute the algorithm implemented in GraphMapper on said graph. However it also provides a function to calculate all other products on a specified graph. It also translates the index based return value of GraphMapper to a far more usable node based value and is designed to be reusable on its given graph. The graph takes a list of nodes,

which represent the graph based floor plan of the store, and its algorithm takes a list of products and returns a list of nodes in the order in which they are to be visited.

Problems Encountered:
While numerous bugs crept into the code and had to be fixed, the exclusively code based nature of the task meant most problems could be easily fixed. The integration of the products into the graph was quite difficult as multiple products may be on one lane and therefore after every integration, all remaining products had to be checked and if necessary updated to their new edge and position on said edge. The only unfixable problem I faced was the inefficiency of the GraphMapper algorithm. While I did try to improve it slightly and this did certainly make a difference, the nature of the problem, as previously stated, ensures it will always be exponentially slow depending on its input.

Integration Process

During weeks 8 and 9, we have been integrating all the developed components into one system.

The process started by integrating the **mobile-application with the server** (checking whether requests are sent and responses are received).
We also integrated the **camera component** into the system, by ensuring that requests are sent and responses are received to and from the server.
The next step was integrating the **recommendation system** by communicating with Quicc DB.
Afterwards, it was the time of the **"Most-Efficient-Route"** algorithm code to be joined with the mobile-application. In that phase, the Java code for the route-algorithm has been added to the main Android project of the mobile-application. After some trials, we managed to successfully get the shortest-route after adding items to the buy-list - exactly as desired!
Displaying the **route on a map** was a complex operation - google maps was not a possible solution, as it is more complex to draw obstacles (the aisles) manually on the map. Therefore, we had to create a custom map, draw the calculated route on that map and also translate GPS coordinates (of the room and the nodes) to exact X,Y points on our map.
Lastly, we ensured that the **location of the user** is displayed correctly, by using the beacons (and their preknown GPS coordinates) in the room OH 111, which is structured similar to the expected room where the live-demonstration takes place (OH 113).

After the integration procedure, we tested whether the system meets the initial requirements.
The outcomes and evaluation are presented in a subsequent section.

Adaptations

Along the development process, we realized that the initial architecture of the system is missing some important elements.
It is important to mention that the architecture shown here is the finalized version. In this section, we are going to describe which of the system elements were added during the development:

Recommendation System - at the beginning, we planned to have the recommendation system as an additional code of the server. By the time, we realized that a separated recommendation system would be our best choice, due to its time-consuming algorithm.
The use of Threads was also discussed. However, Node.JS does not allow multi-threading programming. In other words, our server cannot run multiple threads simultaneously.
Consequently, we have decided to build a python script that runs periodically and updates the recommendations

accordingly (based on the updated purchases). That way, the mobile-application and the server are not disturbed, and responsible for their most important job - to make sure the user has a satisfying experience of use!

Camera Component Development - We decided to use Android phones as cameras, instead of Raspberry Pis (mainly because it is not that practical to procure multiple Pis with a camera for each).
However, we had to deal with a couple of challenges. When it comes to data science and machine learning, Python is the most popular language. As a consequence, many machine learning libraries or frameworks are designed for Python, but Android Studio apps run on Java/Kotlin. Therefore, we found out what libraries or models are being used when working with Android Studio.
In addition, the developer of the component (Robert) had to learn Kotlin for the accomplishment of the task.

We did not have to adapt any other significant changes to our system. We believe that the good planning at the beginning of the project helped us overcome more difficulties such as ongoing adaptations.

Performance Analysis

After the tests were being performed as described in the aforementioned section, we evaluated the results based on the metrics shown here.

The results:

User Requirement	Number of tests performed (each test has different data)	How many of the tests resulted in the expected outcome?
Search for an item and add it to the Buy-List	3	3 / 3
See the (shortest) route to collect all the selected items + Show it on a map	3	3/ 3
See the location of the selected items on the map as well	3	3/ 3
See my own location on the map as well	3	1/ 3

Find which checkout lane is the fastest one	3	3/ 3 With short delays caused by the update speed
Get recommended items to buy, based on my selected items and the current route	3	3/ 3

The evaluation metrics show that our solution was successfully tested and meets the initial criteria and requirements.

Future Improvements

The first thing we have noticed that should be improved is the indoor localization. That is, of course, as a result of the lack of beacons far from each other, to demonstrate the store.

We also think that the UI/UX should be improved, to give a better experience of use for the user.

More interesting features we thought of adding are:

1. Have the location of the nodes and the map stored in the database (instead of being hardcoded) and make the server send that information to the client. This would make the app work for multiple stores with different layouts
2. Implement audio feedback and guidance to also help visually impaired people
3. Allow users to import their shopping list from a document or a picture
4. Make use of Bluetooth signal fingerprinting in order to improve location accuracy

In addition, the performance analysis shows that the indoor localization should be tested on an environment with a stable Bluetooth scanning, so that the given results are accurate and precise.