

**BDMA Joint Project: Semantic Data Management**

**SPROUT**

**Ariston Lim**

**Hung Nguyen**

**Julio Candela**

**Valdemar Hernández**

**June 2020**

**Table of Contents**

[**M1. Purpose Statement**](#_ean7lllogo86) **3**

[**M2. Graph Family Justification**](#_7qj0tknt4a8z) **4**

[**M3. Graph Design**](#_wnfx04i6gd6a) **5**

[**M4. Data Flow**](#_bu7wfjferai6) **6**

[**M5. Exploiting the graph**](#_le3n6y610mny) **7**

[**M6. Metadata**](#_4tvp9jwo120y) **9**

[**M7. Proof of Concept**](#_hldqykklhanv) **10**

[**Appendix 1: Flow Diagrams**](#_uhubdnkfmp6n) **12**

[**References**](#_pcua1kx8rh1h) **13**

# M1. Purpose Statement

Sprout is a startup company that works with a “Farm-To-Table” concept to provide local, organic products of the highest quality to its customers. To achieve this, Sprout is planning to partner with those farmers close to Barcelona that practice sustainable farming and promote ecological balance in order to offer their products through a mobile application developed by us.

For a business that operates as a marketplace, where a lot of dynamic components are involved (e.g. suppliers, customers, market prices), it is important to implement data management and analysis tools to ensure the company has the capability to adapt in this changing environment and make the business thrive.

This document aims to describe how the Data Management project helps Sprout achieve this goal with a specific implementation of graph analytics, where data generated from inside the company, such as orders placed by customer and customer feedback for specific products, will be modeled as a graph so it can be later analyzed to obtain insight on what products are preferred by our customers depending on their purchase history and their feedback to always offer the best selection of products to our customers.

For a startup company, it is important to establish itself as a reliable option for its customers, and these results will help us in that regard in two fronts; first, the company will be able to maintain steady sales by offering the best set of products to each customer, and second, the company will be able to take action to cover the demand for its best selling products, always staying on top of the customers’ orders.

# M2. Graph Family Justification

Sprout firmly believes that Analytics will bring organic food business to the next level. As a consequence, one of our competitive advantages is to become a smart company applying graph algorithms in order to recommend bigger and more varied baskets of products for customers and suggest the best prices for organic food. The best option to deal with these kinds of analysis is property graphs.

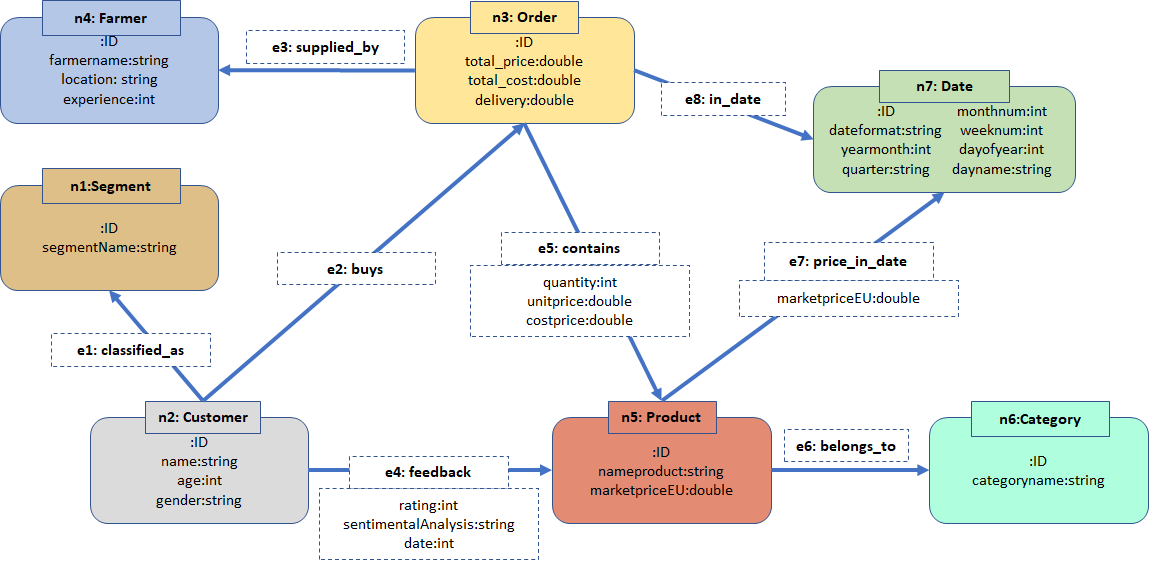
Regarding the core of our project, data analysis, property graphs offer a high level representation of the data model utilizing nodes, relationships and properties as well as traversal navigation to speed-up graph patterns and path queries in order to traverse the graph looking for recommendations of similar customers. Moreover it provides graph algorithms to perform recommendation tasks which are relevant issues of the company. These features let business users understand the data and perform much more analytical tasks with intuitiveness, simplicity and flexibility. The chosen language is Cypher since it is the de-facto option for property graphs, human-readable [1] and allows us to perform all the tasks described in Neo4j which can be connected from different endpoints as Python.

On the other hand, Sprout’s data integration processes are simple since most of the input files are already part of the internal systems such as orders and feedback. In other words, there is no need to integrate external and complex sources. Furthermore, the integration process is already done with Spark in order to prioritize the forthcoming Analytics. Thus, the use of property graphs seems convenient for simpler tasks in contrast to knowledge graphs, which can be used for convoluted data integration.

In conclusion, property graphs fit quite well for our analytical purposes providing the right tools to make analysis, reports and machine learning tasks easier, faster and deeper. In this way, Sprout would be benefited by insights from graph analytics and foster customers to increase the size of their baskets.

# M3. Graph Design

The graph model can be seen in the following image in addition to the metadata of the nodes and relationships:



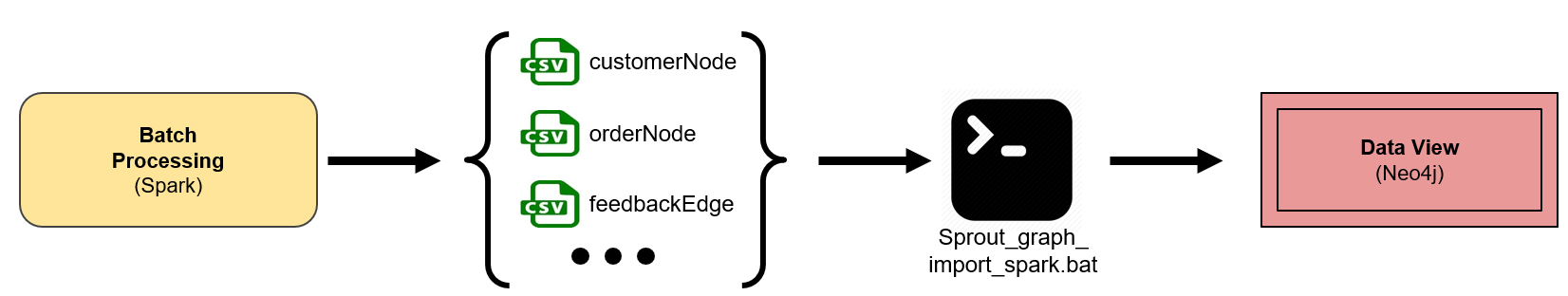
Some of the functional requirements for the analysis are below:

* Customers are classified in segments depending on the frequency of orders (This task is performed in the Spark pre processing).
* Customers can buy many orders or baskets which contain different products storing the quantity, unit price and cost of each product bought in the relationship “contains”. This information is important for basket recommendations. In the orders node, the aggregate numbers of price and cost are also stored for simpler analysis.
* In addition, the graph also contains the selected farmer and the date of transaction for each order in order to generate historical reports for business analysts.
* The graph also stores the information of the EU pricing of organic products per date in the relationship “price\_in\_date” in order to generate estimated prices for Sprout’s baskets.
* Products keep the current value in the EU market as reference and they belong to a category.
* Customers give feedback about products with information such as rating and sentimental analysis performed by Spark Streaming. This will allow us to validate the recommendations we are showing to customers and offer better baskets to them.

# M4. Data Flow

The original source for the data used for the graph analytics comes from Sprout’s data lake, which contains different types of raw files with details such as order history, market prices from different countries of the EU for selected products [2-7], and the customer feedback.

This data is processed by Spark to produce a set of CSV files with the appropriate format to simplify the population of the graph through the use of a Batch file. By executing the *Sprout\_graph\_import\_spark.bat* file, all the instances for the nodes and edges are uploaded at once to Neo4j, where the graph analysis can be performed. The following diagram provides a simplified view of the process described previously:



Additional details on the complete data flow process (including the BDM implementation) can be observed in [Appendix 1: Flow Diagrams](#_uhubdnkfmp6n). The complete list of files used as input to populate the graph is shown below:

| categoryNode.csv | customerNode.csv | dateNode.csv |
| --- | --- | --- |
| farmerNode.csv | orderNode.csv | productNode.csv |
| segmentNode.csv | belongstoEdge.csv | buysEdge.csv |
| classifiedAsEdge.csv | containsEdge.csv | feedbackEdge.csv |
| inDateEdge.csv | priceInDateEdge.csv | suppliedbyEdge.csv |

It should also be mentioned that the files *categoryNode.csv*, *dateNode.csv*, and *SegmentNode.csv* were considered as fixed data, and as such, did not require any preprocessing from Spark before they could be uploaded into Neo4j.

All the files and the script used for this implementation can be found in the following Github repository: <https://github.com/JulioCandela1993/BDM_Sprout>, inside the ***DataIntegration/*** and ***DataIntegration/Spark/src/main/resources/*** folder.

# M5. Exploiting the graph

In this section, we are going to use graph analytics to build a products recommender which uses Neo4j to perform **collaborative filtering [8]** to output recommended products. The result will be filtered with real-time feedback retrieved from ElasticSearch before showing to our customers. Our recommender will give recommendations of products to a specific user based on two strategies explained below.

1. **Based on orders:** For each product which has been bought by the user, we find other orders which contain that product and recommend other products found in those orders to the user. The Neo4j query below is used to link top 10 recommended products to the user with userID=212 based on orders data.

MATCH (c:customer)-[:buys]->(o1:order)-[:contains]->(p1:product)<-[:contains]-(o2:order)-[:contains]->(p2:product)

WHERE ID(c) = {userID}

WITH p2 as RecommendedProduct, count(o2) as Score

ORDER BY Score DESC LIMIT 10

WITH RecommendedProduct

MATCH (c:customer) WHERE ID(c) = 212

MERGE (RecommendedProduct)-[:recommended\_for]->(c)

1. **Based on feedback and rating:** For each product which has been given a positive feedback from the user A, we find another user B who also likes that product. Since they might have the same preferences, we suggest top 10 preferred products of user B to user A based on the rating.

MATCH (c1:customer)-[f1:feedback]->(p1:product)<-[f2:feedback]-(c2:customer)-[f3:feedback]->(p2:product)

WHERE ID(c1) = 212

AND f1.sentimentalAnalysis = "POSITIVE"

AND f2.sentimentalAnalysis = "POSITIVE"

AND f3.sentimentalAnalysis = "POSITIVE"

WITH p2 as RecommendedProduct, f3.rating as Score

ORDER BY Score DESC LIMIT 10

WITH RecommendedProduct

MATCH (c:customer) WHERE ID(c) = 212

MERGE (RecommendedProduct)-[:recommended\_for]->(c)

We are also using Neo4j to find our bestsellers by using a Cypher query. Top-selling products will have a property named bestseller which is set to *true*. They are updated daily by resetting all the values to *false* before running the following query.

MATCH (p:product)<-[:contains]-(o:order)

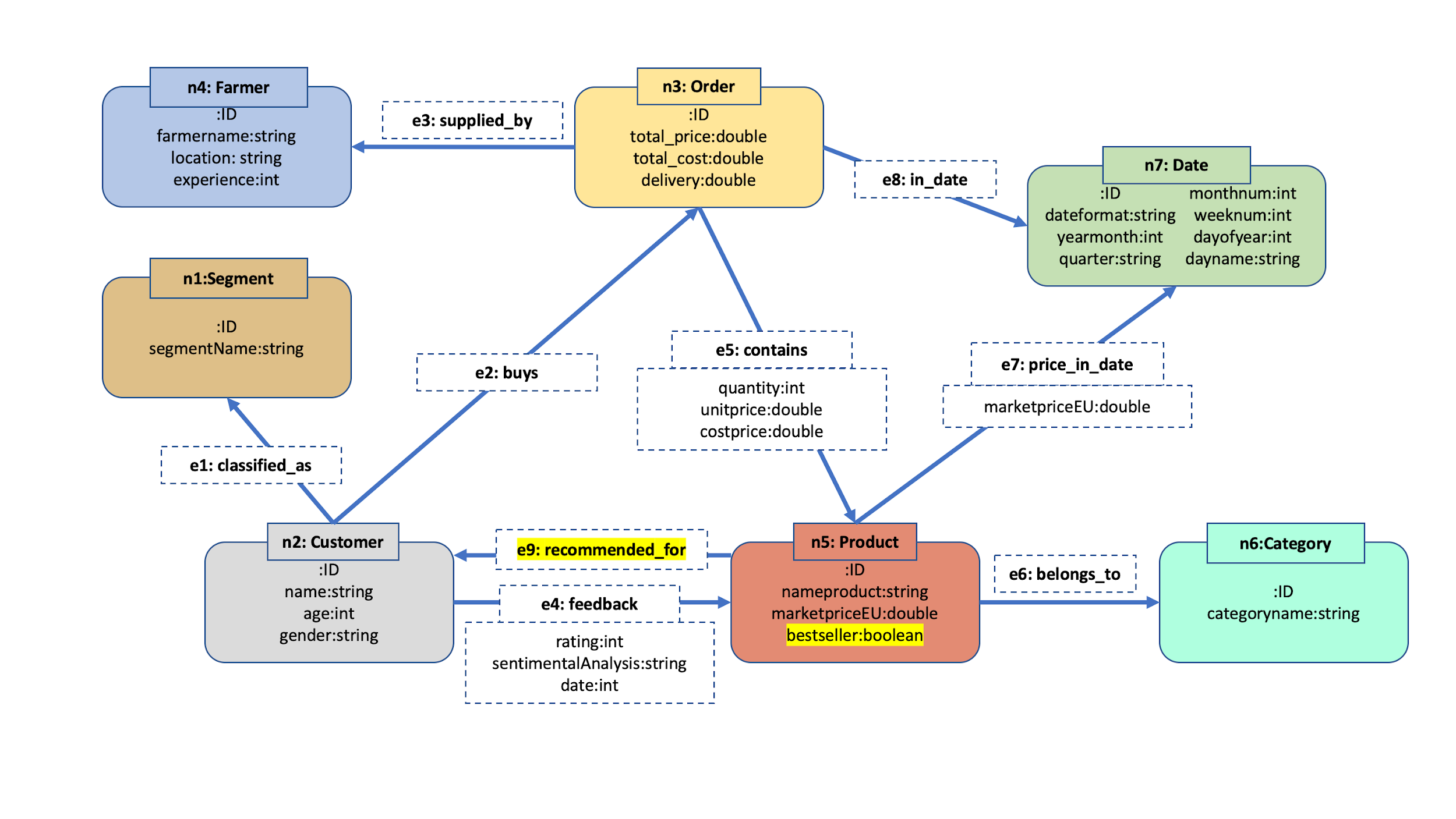
WITH p, count(o) as count

ORDER BY count DESC

LIMIT 10

SET p.bestseller = true

# M6. Metadata



The diagram above shows the updated metadata after doing the analysis in M5. Specifically, the new generated metadata (**highlighted in the diagram**) are:

1. (:product)-**[:recommended\_for]**->(:user)
2. New property “**bestseller**” inside the (:Product) node

The [:recommended\_for] edge is generated after running the collaborative filtering algorithm as explained in M5. We will run this algorithm *daily* to generate a list of recommended products for each user. Everyday, there might be new products connected to the customer via this new edge. We still keep the “old” edges and we will do filtering later on based on user feedback. This “recommended\_for” edge will be stored as part of the new metadata in the graph database.

Having this edge will simplify finding a product recommendation for users. We can run a simple query:

MATCH (c:customer)<-[:recommended\_for]-(p:product)

WHERE ID(c) = {userID}

RETURN p.nameproduct as recommendedProduct

LIMIT 10

We will also update the “bestseller” feature *daily.* At the beginning of each day, we will set the boolean as False for all products. After running the query, the top 10 best selling products will be flagged as True. This data will be used as part of our front page of the app to dynamically update the Bestselling Product tab.

# M7. Proof of Concept

In this section, we are going to demonstrate a use case of our recommendation system using Neo4j and ElasticSearch. Since sales data and feedback are processed and stored in Neo4j at the end of the day, we need to take into account real-time feedback coming from our application to be able to suggest products to an user in real-time. With Neo4j, we can easily perform a collaborative filtering with a simple query and we are planning to use the Louvainalgorithmlater to detect strongly connected products to form a basket of products which usually go together. ElasticSearch gives us a powerful platform to visualize our sales statistics and also a fast search engine.

In this example, we will be using those two tools to output a list of recommended products for the user with *userID = 2962.* With these two lines of code below, we are executing two Neo4j queries to generate [:recommended\_to] relationships from products to users based on two strategies explained in the M5 section.

# Execute recommender to generate some [:recommended\_to] relationships

session.write\_transaction(recommend\_from\_order)

session.write\_transaction(recommend\_from\_rating)

Next, we are executing a Neo4j query to get recommended products that we generated by above queries

# Query recommended products for the user

recommendedProducts = session.read\_transaction(get\_recommended\_product)

for record in recommendedProducts:

id = record["recommendedProduct"].id

listRecommendedProducts[id] = record["recommendedProduct"]

The real-time feedbacks are updating on ElasticSearch and we can get all feedbacks have been provided by the user

# Query feedbacks from the user from ElasticSearch

query\_body = {"query": {"bool": {"must": {"match": {"userid": userID}}}}}

feedbacks = es.search(index="cust\_index")

Some recommended products might have been given negative feedback from the user, which has not been updated in Neo4j graph (daily updated). That is the reason why we need to filter the result with the user’s feedback updated real-time in ElasticSearch.

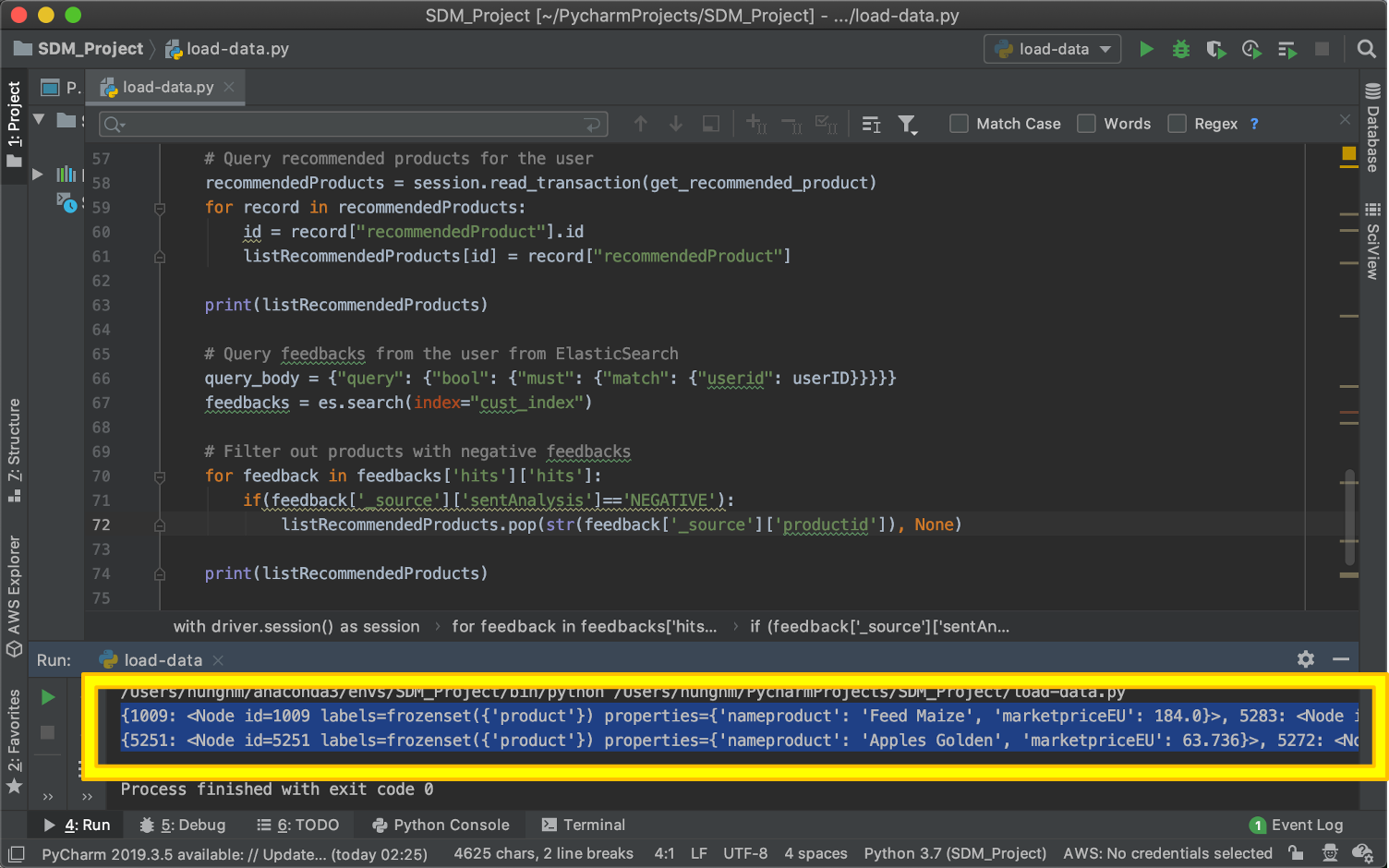
# Filter out recommended products with negative feedbacks

for feedback in feedbacks['hits']['hits']:

if(feedback['\_source']['sentAnalysis']=='NEGATIVE'):

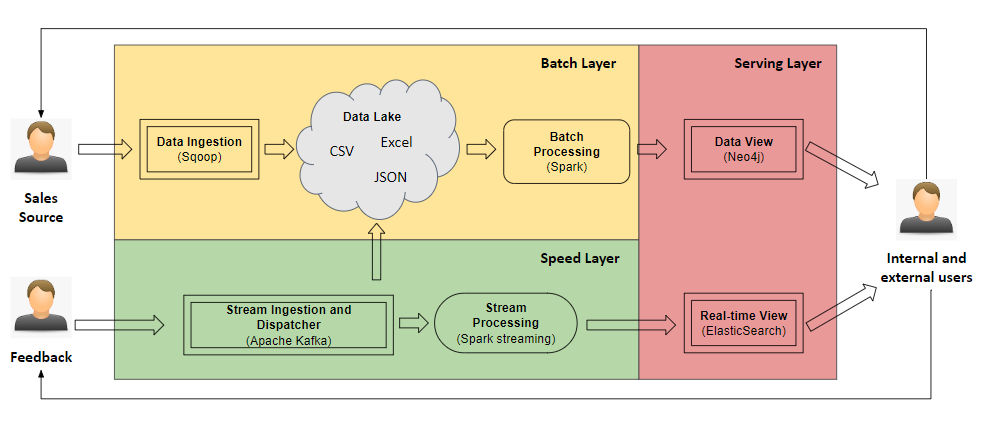
listRecommendedProducts.pop(str(feedback['\_source']['productid']), None)

The list of recommended products for the user before and after the filtering are shown in the below figure (highlighted in the console).

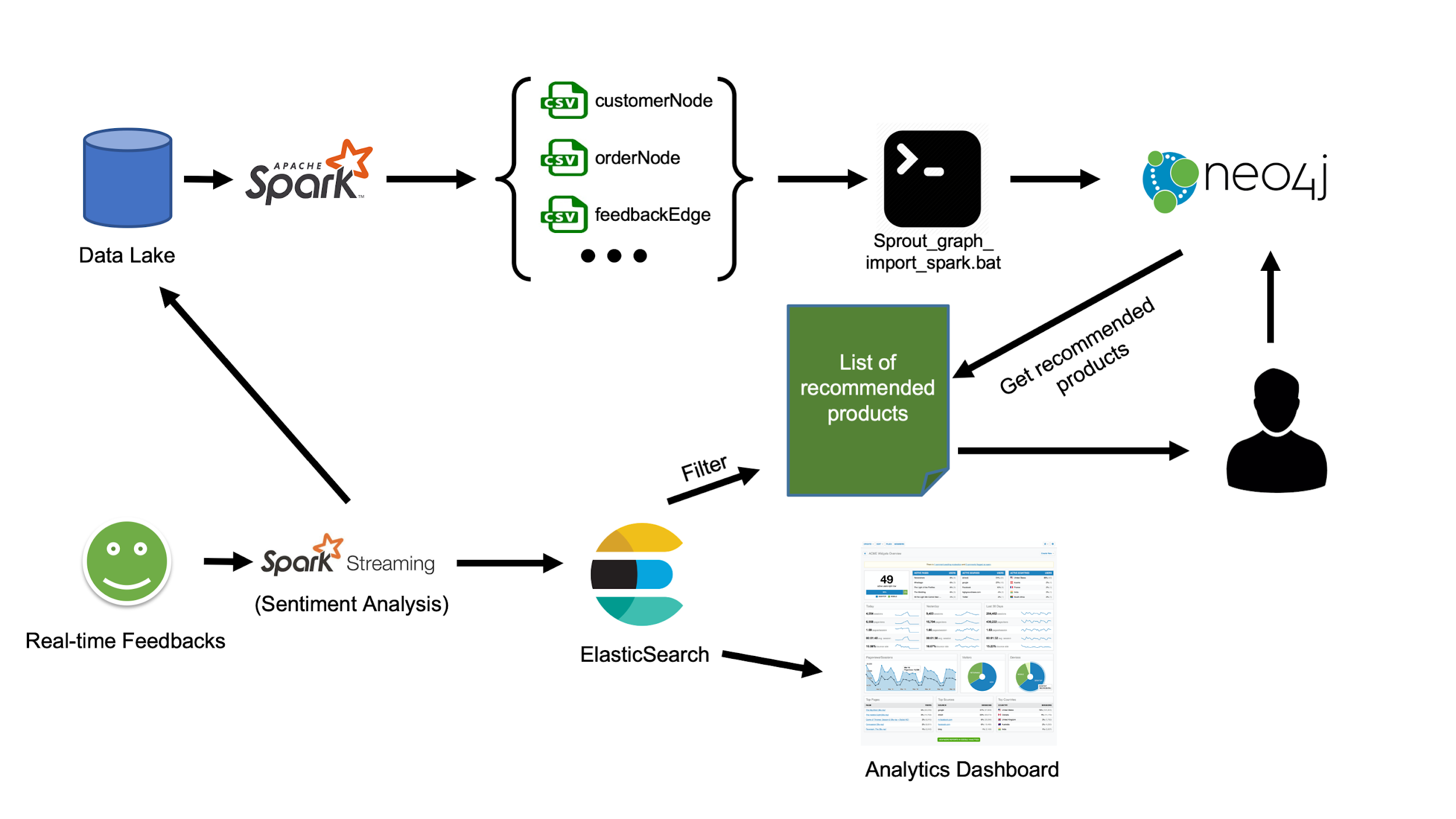


# Appendix 1: Flow Diagrams

The following diagram shows the final Big Data architecture for Sprout. The graph analytics is performed in the Serving Layer, once the data has been processed by Spark.



The next diagram shows how we built our product recommender. It is possible to observe how all the tools used for the project interact together between them.

****

# References

1. *(2020). Cypher Basics I. Retrieved from* <https://neo4j.com/developer/cypher-basics-i/>
2. (2020). *Open Data Portal Home. EU Open Data Portal.* Retrieved from <https://data.europa.eu/euodp/en/home>
3. (2020). *EU prices for selected animal products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/ef78194c-e94c-47dd-85f7-44ba98b6746c>
4. (2020). *EU prices for selected fruit products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/eu-prices-for-selected-fruit-products>
5. (2020). *EU prices for selected dairy products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/eu-prices-for-selected-dairy-products>
6. (2020). *EU prices for selected vegetal products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/eu-prices-for-selected-vegetal-products>
7. (2020). *EU prices for selected vegetable products*. EU Open Data Portal. Retrieved from: <https://data.europa.eu/euodp/data/dataset/b8c13971-e315-470e-a125-40d67fe7067b>
8. J. Ben Schafer. Dan Frankowski. Jon Herlocker. Shilad Sen. (2007). *Collaborative Filtering Recommender Systems.* LNCS. Retrieved from: https://link.springer.com/chapter/10.1007/978-3-540-72079-9\_9