CS387 Project Report
E-Commerce Recommender System

Bhaskar Gupta	Niraj Mahajan	Shivam Goel	Tathagat Verma
180050022	180050069	180050098	180050111
180050022@iitb.ac.in	nirajm@iitb.ac.in	180050098@iitb.ac.in	180050111@iitb.ac.in

Requirements

1. Neo4j community edition
2. Nodejs
3. Java 11
4. APOC Core

Introduction - What are recommender systems?

Recommender systems are the systems that are designed to recommend things to users based on several factors. These systems predict the most likely product that the users will purchase based on the history of interactions and purchases done by the user. Companies like Netflix, Amazon, etc. use recommender systems to help their users to identify the correct product or movies for them.

In this project we have developed a low scale e-shopping application using the neo4j graphDB and have deployed the web application using nodejs. Our application provides an interface for Buyers, Sellers, Delivery Personnels and an Administrator, in the web application. The introduction of delivery personnels enhances the transparency of our backend by adding an additional layer to observe the “delivery process”. It also brings our application closer to the real world scenario.

Use cases

We have 4 types of users that will use our application - Administrator, Seller, Buyer, Delivery Agent. In addition to this, we have also provided for a “guest” user, who can access our product catalogues without logging in (quite similar to what amazon offers). Of course the guest user has restricted access - more on this in the use cases.

1. Search Products:
    a. Description: Search the catalogue for a product using a keyword.
    b. Access Allowed: Guest Users, Buyers, Administrators.
    c. Input Parameters: The keyword to match products

   d. <u>Trigger</u>: This is usually the opening page for the relevant users. It can also be triggered from the nav bar.
   e. <u>Ideal Behaviour</u>: Search the products relation to match all entries similar to the input keyword and update the list in the UI
   f. <u>Changes to the DB</u>: No changes.
2. View Product Details:
   a. <u>Description</u>: The list of products returned in the previous use case is a summary of the products. In order to view more details, or to add the product to cart (only for buyers), one has to click on the "View details" button. This will give an in-depth summary of the said product.
   b. <u>Access Allowed</u>: All those who can search for products, since this is an option available while browsing products. So basically Guest Users, Buyers, Administrators.
   c. <u>Input Parameters</u>: The id of the product.
   d. <u>Trigger</u>: On clicking the "View Details" button
   e. <u>Ideal Behaviour</u>: Since this involves a button click, we record the product of the id and send it to a "POST" method as a parameter. On successful execution, this use case will load all the details of the concerned product. For buyers, it will also load an option to add items to cart. This will also show the "Also viewed" and "Also bought" products.
   f. <u>Changes to the DB</u>: In case of a buyer, the view action is recorded to use for recommending "Also Viewed" products.
3. <u>Add to Cart/Remove from cart</u>:
   a. <u>Description</u>: On viewing the details of a product, a buyer can add a quantity x of the products to the cart. This cannot be higher than the number of items present in the database and this is ensured by the UI - the input box will not accept any invalid number. Similarly products can be removed from the cart.
   b. <u>Access Allowed</u>: Only Buyers
   c. <u>Input Parameters</u>: Product ID of the concerned object.
   d. <u>Trigger</u>: By clicking the "Add to Cart" button, provided the input box has a valid number. Similarly by clicking the "remove from cart" button.
   e. <u>Ideal Behaviour</u>: The product will be visible in the user's cart. Also this item will be virtually reserved - For example if the DB has 20 products (say P1), and we add 5 to the cart, the next time the buyer opens the product details, he/she can add utmost 15 items of P1 to the cart. The reverse for remove from cart - the product can re-add the quantity to cart.
   f. <u>Changes to the DB</u>: A "cart" edge will be created for the relevant Buyer-product pair.
4. Place Order:
   a. <u>Description</u>: Buy all elements in the cart. One will need to navigate

    b. Access Allowed: Only Buyers
    c. Input Parameters: None - We already have the buyer id and can “buy” all items in the buyer’s cart.
    d. Trigger: On clicking the buy option when on the “cart” page.
    e. Ideal Behaviour: The balance of the buyer will be checked. If sufficient, the cart will immediately be emptied. The order will be sent to all the respective sellers whose items were added to cart. This will show in their “Order Requests tab”. This will also show in the order history of the buyer as a pending order.
    f. Changes to the DB: The buyer to product cart edge will be deleted. Order nodes are added for all products, and buyer-order, order-product (and eventually delivery personnel-order) edges will be added.

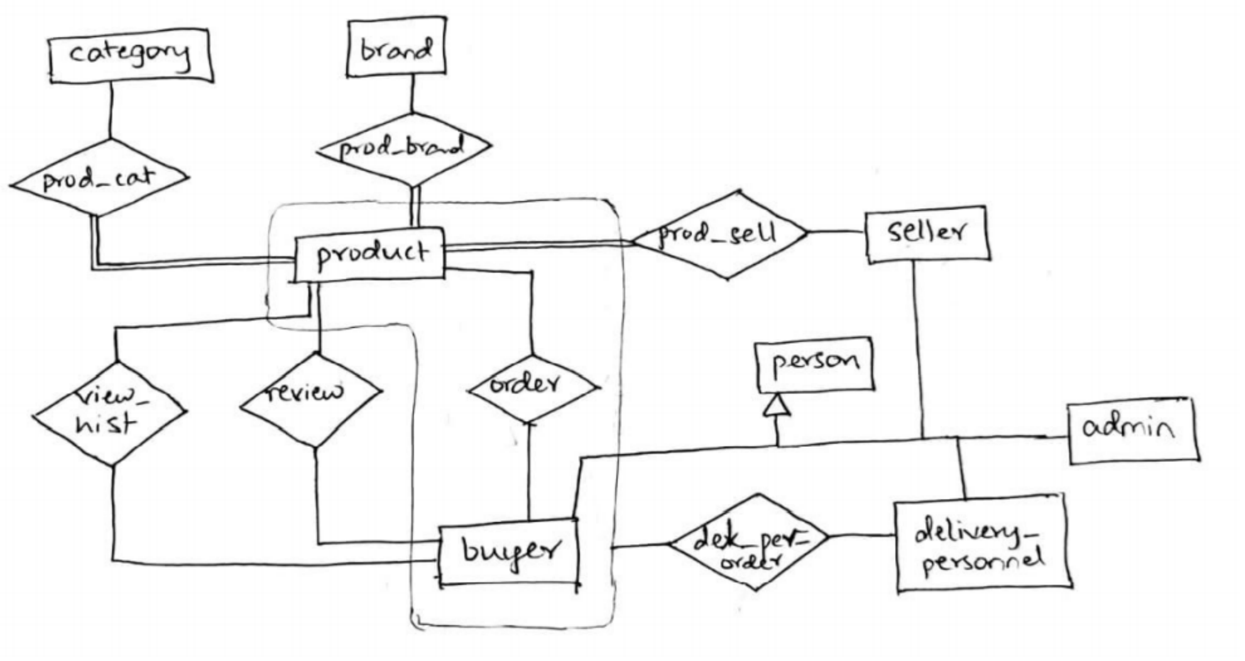
5. Rate a product:
    a. Description: - give review on a product being viewed. Can be accessed from the order history of buyers.
    b. Access Allowed: Buyers
    c. Input Parameters: Buyer id and Order ID.
    d. Trigger: On giving a valid rating, and clicking the “update rating button”
    e. Ideal Behaviour: The rating will get stored for the order product from the user. Each item in the order can be rated
    f. Changes to the DB: the rating for this product, order is updated
6. Log In:
    a. Description: Log in as a User
    b. Access Allowed: Anyone
    c. Input Parameters: Username, password credentials + the mode of login - admin, buyer, seller, delivery agent.
    d. Trigger: On clicking on the login page in the nav bar
    e. Ideal Behaviour: WIll log in if the credentials are correct, will report back if credentials are incorrect.
    f. Changes to the DB: No change.
7. Sign Up:
    a. Description:Add a Buyer
    b. Access Allowed: Anyone
    c. Input Parameters: Username, password, name, mail, address, phone number.
    d. Trigger: On clicking on the login page in the nav bar
    e. Ideal Behaviour: WIll add a new buyer
    f. Changes to the DB: New buyer added..
8. Log Out:
    a. Description: Log out
    b. Access Allowed: Anyone (Not guest ofcourse)

    c. Input Parameters: None
    d. Trigger: On clicking on the login out button in the nav bar
    e. Ideal Behaviour: WIll log out
    f. Changes to the DB: No change.
9. View All User Details:
    a. Description: A high security feature to view all user (seller, buyer, delivery agents) details. (Probably when the FBI asks you :))
    b. Access Allowed: Only admins.
    c. Input Parameters: None
    d. Trigger: When clicked on the designated page on the admin nav bar
    e. Ideal Behaviour: Lists out all the relevant users
    f. Changes to the DB: None
10. Ship an item:
    a. Description: On placing an order, the sellers will get a request to ship the item
    b. Access Allowed: Relevant Sellers
    c. Input Parameters: Object id
    d. Trigger: Situation arises when a buyer sends an order. Object can be shipped by clicking on the ship button
    e. Ideal Behaviour: The object will be shipped out, ie, it will move to the ongoing orders of the seller. A delivery agent with the minimum active deliveries will be assigned for the order and this will be visible in the active orders of the delivery agent. The order history of the buyer will show the order as "shipped".
    f. Changes to the DB: The status attribute of the product is updated. The delivery agent is also assigned to this order product by an edge (delivery personnel-order).
11. Recharge Account:
    a. Description: The buyer can recharge his account here.
    b. Access Allowed: Buyer
    c. Input Parameters: Buyer ID
    d. Trigger: On clicking on the Recharge page
    e. Ideal Behaviour: The buyer balance will get updated
    f. Changes to the DB: Buyer balance will be updated
12. Search Products:
    a. Description: Search the catalogue for a product using a keyword.
    b. Access Allowed: Guest Users, Buyers, Administrators.
    c. Input Parameters: The keyword to match products
    d. Trigger: This is usually the opening page for the relevant users. It can also be triggered from the nav bar.

    e. Ideal Behaviour: Search the products relation to match all entries similar to the input keyword and update the list in the UI
    f. Changes to the DB: No changes.
13. Analytics:
    a. Description: The admin can see analytics for the highest rated seller, most active delivery agent, and the most purchases by buyers. In addition to this, we showed pie charts for the number of purchases per category and have shown the orders placed per year as a line graph..
    b. Access Allowed: Admin
    c. Input Parameters: None
    d. Trigger: On clicking on the analytics page
    e. Ideal Behaviour: This will show the relevant charts and list our the relevant results
    f. Changes to the DB: None
14. Add new product:
    a. Description: Seller can put up new objects on sale.
    b. Access Allowed: Seller
    c. Input Parameters: Object name, quantity, cost, categories, description and product image
    d. Trigger: on clicking the add new product button for seller
    e. Changes to the DB: A new product entry will be created.
15. Mark object as delivered:
    a. Description: Delivery agent can deliver an item..
    b. Access Allowed: Delivery Agent
    c. Input Parameters: Orderid, product id.
    d. Trigger: on clicking on "mark as delivered" in the ongoing orders of delivery agents
    e. Ideal Behaviour: The item will be moved from the delivery agent ongoing deliveries to the past orders page. A similar update will be done for the seller and buyer.
    f. Changes to the DB: The product status for the order will be changed.

Design

We have generated the front end using html and .ejs scripts. For the backend, we have created a graphDB in neo4j - the following figure shows the ER diagram for the database that we have used. For giving recommendations in the recommender system, we have given "Also viewed" and "Also bought" items for every item the buyer opens. This is similar to a form of collaborative filtering. We have also allowed for a keyword for searching while browsing the data. We have used a combination of substring, levenshtein distance, ratings and the length of the string to sort the returned query.



Test results

We have manually tested all the use cases in the database by adding a small test dataset. We have downloaded a subset of data from the Amazon Review Dataset. The original dataset has 22 broad product categories. Since that went upto 12GB and the corresponding size of ratings to 6GB, we used only 2 categories - Books and Electronics. The number of products then is 2.9M(books) and 0.8M(electronics). The overall size of products data is 2GB. The ratings data is available in a separate csv. The number of ratings for Books is 51M and 21M for Electronics. Size of the review data is 2GB(books) and 0.8GB(electronics). The script for downloading data (data_load.py) produces the cypher query language code (2 files - product_data.cypher and user_data.cypher)for inserting the above into the graph data DB. Since the rating data set contains user IDs, we have created buyers as per that. The total number of buyers comes out to be 3.6M. The sellers, admin and delivery_personnel data is generated by us as it is not present in the Amazon dataset. We have kept only 1 admin and 1 delivery personnel in the initial data. Each product is associated with a seller. We have kept only 1 seller initially.

We tested our web application for the backend-frontend link for all the use cases mentioned above. All the linking has been tested positively. But we are getting bad results for the search query use case.

Conclusions

We thus have implemented an e-commerce web application on NODE JS using the neo4j database. We were able to implement a fully functional web application for e-commerce. Although we were able to implement most of the planned deliverables, we had aimed at implementing a combination of collaborative filtering and content based filtering for the recommender system. But we were able to implement just a collaborative filtering method based on “Also viewed” and “Also bought” items. We have successfully linked all the planned UI items and also updated our planned goals by incorporating reviews on our deliverables and have provided Analytics for the admin user, and have tested and demonstrated the entire order process from adding to cart to delivery in a fully transparent manner. Although the UI is linked properly, there are some flaws in the backend for the query filtering. Our search query does not provide an accurate set of results as expected from the dataset, even on trying various substring operations and levenshtein distance (weighted sum with various parameters).

Git link
https://github.com/tathagatv/Recommender-system