UNIVERSITÉ LIBRE DE BRUXELLES
Université Catholique de Louvain

Project report part 2

UCLouvain: LINFO2145 - Cloud Computing
ULB: INFO-Y025 Cloud computing

Samir Azmar

December 19, 2024

Contents

1 What's implemented

- The recommendation does not include products already in the cart.
- There is a back-end service implementing a recommendation engine. It can be found in **project/src/back-end/recommendation**.
- There is an integration of the recommendation feature in the front-end. It can be found in **project/src/front-end/src/interfaces/products/Product.svelte**
- The recommendation engine uses the user's action logs provided by the logs service.

There is a timestamp in the description.
Bonus point: https://www.youtube.com/watch?v=rAgBqbk3Vns

2 Updated list of services

2.1 Logs microservice

What's new?:

The user's logs that concern the endpoint /**checkout**/**save** trigger an event. This event sends the log to the recommendation service and saves it in its database.

Role:

The role of this microservice is to handle logging operations. It provides endpoints to save performance logs for microservices and to save user action logs. This helps in monitoring, auditing, and debugging the application by keeping track of performance metrics and user activities.

Technologies:

The microservice uses the following technologies:

- **Node.js**
- **Express.js**
- **Apache CouchDB**
- **Docker**

How to build the container:

The command to build this container is found in the script **build_containers.sh** that builds all containers.

How to run a swarm of instances of the container:

The file **init_swarm.sh** creates a swarm and deploys all containers of my microservice from **scapp.yml**.

Complete API of the service:

- **Save Performance Logs for a Microservice**
  - **Endpoint:** `POST /logs/microservice/:microservice`
  - **Description:** Saves performance logs for a specified microservice.
  - **URL Parameters:**
    - `microservice` (string, required): Name of the microservice for which logs are being saved.
  - **Request Body:**
    - `logs` (JSON, required): Performance log data to be saved.
  - **Response:**

    - **200 OK:** Logs saved successfully. Returns:
      - `status` (string): `"success"`
    - **500 Internal Server Error:** Saving logs failed. Returns:
      - `status` (string): `"error"`
      - `message` (string): Error details.

- **Save User Action Logs**
  - **Endpoint:** `POST /logs/user-action`
  - **Description:** Saves logs for user actions across microservices.
  - **Request Body:**
    - `logs` (JSON, required): User action log data to be saved.
  - **Response:**
    - **200 OK:** Logs saved successfully. Returns:
      - `status` (string): `"success"`
    - **500 Internal Server Error:** Saving logs failed. Returns:
      - `status` (string): `"error"`
      - `message` (string): Error details.

2.2 Recommendation microservice

Role:

The role of this microservice is to get the user's logs from the logs microservice and use these logs to recommend products.

Technologies:

The microservice uses the following technologies:

- **Node.js**
- **Express.js**
- **Apache CouchDB**
- **Docker**

How to build the container:

The command to build this container is found in the script **build_containers.sh** that builds all containers.

How to run a swarm of instances of the container:

The file **init_swarm.sh** creates a swarm and deploys all containers of my microservice from **scapp.yml**.

Complete API of the service:

- **Save Logs**
  - **Endpoint**: `POST /saveLog`
  - **Description**: Saves log data into the database for recommendation purposes.
  - **Request Headers**:
    - `Content-Type` (string, required): `"application/json"`
  - **Request Body**:
    - `logData` (JSON, required): The log data (action, details, endpoint, timestamp).
  - **Response**:
    - **200 OK**:
      - `status` (string): `"success"`
    - **500 Internal Server Error**:
      - `status` (string): `"error"`
      - `message` (string): Details of the error.
  - **Additional Actions:**
    - Logs the performance of the request for debugging purposes.

- **Get Recommendations**
  - **Endpoint**: `GET /recommendation`
  - **Description**: Retrieves recommendations for the authenticated user.
  - **Request Headers**:
    - `Authorization` (string, required): Bearer token for user authentication.
  - **Response**:
    - **200 OK**:
      - `status` (string): `"success"`
      - `recommendation` (array): A list of recommendation objects.
    - **500 Internal Server Error**:
      - `status` (string): `"error"`
      - `message` (string): Details of the error.
  - **Additional Actions:**
    - Measures and logs performance of the request for debugging purposes.
    - Verifies the user's authentication token.

3 Technology justification

- Node.js
  - **Description**: A JavaScript runtime built on Chrome's V8 engine that allows developers to execute JavaScript on the server side.
  - **Justification**: It enables full-stack JavaScript development, offering non-blocking, event-driven architecture ideal for scalable and high-performance applications.
- Express.js
  - **Description**: A minimalist web application framework for Node.js used to build web applications and APIs.
  - **Justification**: Simplifies the creation of server-side applications by offering robust routing, middleware integration, and efficient request/response handling.
- Docker
  - **Description**: A platform that enables developers to build, ship, and run applications in isolated containers.
  - **Justification**: Provides consistency across environments, improves scalability, and simplifies deployment and dependency management by packaging applications with their runtime environment.
- Apache CouchDB
  - **Description**: A NoSQL database that uses JSON to store data and JavaScript for querying and indexing, with multi-master replication and offline-first support.
  - **Justification**: Stores data in a JSON format, making it easy to work with modern web applications.

4 Map/Reduce querie

You can find this code in **/project/src/back-end/recommendation/views/most_co_purchased.js**

```javascript
const viewDescriptor = {
  views: {
    most_co_purchased: {
      map: function (doc) {
        if (doc.details && doc.details.checkout && Array.isArray(doc.details
            .checkout.items)) {
          const items = doc.details.checkout.items;
          for (let i = 0; i < items.length; i++) {
            for (let j = i + 1; j < items.length; j++) {
              emit(items[i].name, items[j].name);
              emit(items[j].name, items[i].name);
            }
          }
        }
      },
      reduce: function (keys, values, rereduce) {
        const result = {};
        if (rereduce) {
          for (const value of values) {
            for (const product in value) {
              result[product] = (result[product] || 0) + value[product];
            }
          }
        } else {
          for (const product of values) {
            result[product] = (result[product] || 0) + 1;
          }
        }
        return result;
      }
    }
  }
};

module.exports = { viewDescriptor };
```

The `viewDescriptor` defines a CouchDB view named `most_co_purchased`, designed to analyze and find products that are frequently purchased together. Below is a detailed explanation:

Purpose

The `most_co_purchased` view identifies which items are co-purchased in the same checkout session.

Map Function

The map function extracts pairs of items from a document representing a checkout session and emits these pairs as key-value pairs.

Structure of the Map Function

```
map: function (doc) {
    if (doc.details && doc.details.checkout && Array.isArray(doc.details.checkout.items)) {
        const items = doc.details.checkout.items;
        for (let i = 0; i < items.length; i++) {
            for (let j = i + 1; j < items.length; j++) {
                emit(items[i].name, items[j].name);
                emit(items[j].name, items[i].name);
            }
        }
    }
}
```

Explanation

- The function checks if the document contains valid checkout data in `doc.details.checkout.items`.
- For each pair of items in the checkout array:
  - It emits a key-value pair `(itemA, itemB)` to track their co-purchase.
  - A reverse pair `(itemB, itemA)` is also emitted for bidirectional analysis.

Reduce Function

The reduce function aggregates the co-purchase data to count how often each pair of items appears.

Structure of the Reduce Function

```
reduce: function (keys, values, rereduce) {
    const result = {};
    if (rereduce) {
        for (const value of values) {
            for (const product in value) {
                result[product] = (result[product] || 0) + value[product];
            }
        }
    } else {
        for (const product of values) {
            result[product] = (result[product] || 0) + 1;
        }
    }
    return result;
}
```

Explanation

- **Parameters:**
  - `keys`: The emitted keys from the map function (not used here).
  - `values`: The values corresponding to the keys.

  - `rereduce`: Boolean indicating if this is a second-level reduction.

- **Logic:**
  - If `rereduce` is `false`:
    - Count the frequency of each co-purchased item.
  - If `rereduce` is `true`:
    - Merge pre-aggregated results from earlier reductions.
- **Output:** A key-value object where the keys are product names, and the values are their co-purchase counts.

Overall Functionality

The `most_co_purchased` view works as follows:

- **Map Phase:** Identifies all pairs of items purchased together.
- **Reduce Phase:** Aggregates these pairs to determine co-purchase frequencies.

5 Deployment

Before following one of these deployments, make sure to log in with your Microsoft Azure account. If you are not logged in, type in your terminal:

```
az login
```

Make sure to open a terminal in the project folder and that the terminal user owns the files.

5.1 Locally without VMs

1. If you do not possess a **group resource**, execute this command:

   ```
   $ az group create --name linfo2145.weu --location westeurope
   ```

2. Create an **Azure Blob** by executing this shell script `create_azure_blob.sh` with this command and follow the instructions:

   ```
   $ source create_azure_blob.sh
   ```

3. To build and run the docker files of each microservice, execute this command:

   ```
   $ ./build_and_run_containers.sh
   ```

4. Generate a dataset by executing this command: (You can run it multiple time if you want. The IP address is 0.0.0.0)

   ```
   $ ./dataset.sh
   ```

5.2 Locally with 2 VMs + swarm

You can start from step 3 if you already created Azure blob from step 2 in section Locally without VMs

1. If you do not possess a **group resource**, execute this command:

   ```
   $ az group create --name linfo2145.weu --location westeurope
   ```

2. Create an **Azure Blob** by executing the shell script `create_azure_blob.sh` with this command and follow the instructions:

   ```
   $ source create_azure_blob.sh
   ```

3. Start 2 VMs by executing this command:

   ```
   $ ./init_vms.sh
   ```

4. Create a docker swarm by executing this command:

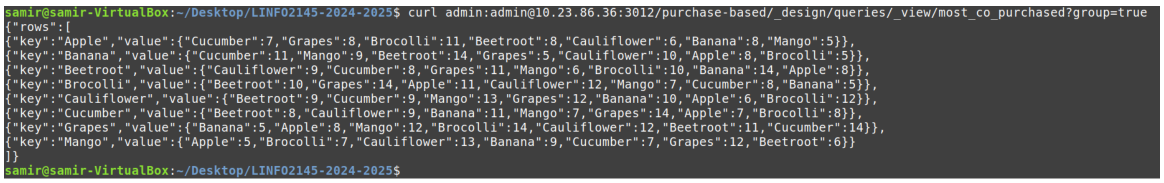
   ```
   $ ./init_swarm.sh
   ```

5. Generate a dataset by executing this command: (You can run it multiple time if you want. The IP address is the manager's IP address.)

   ```
   $ ./dataset.sh
   ```

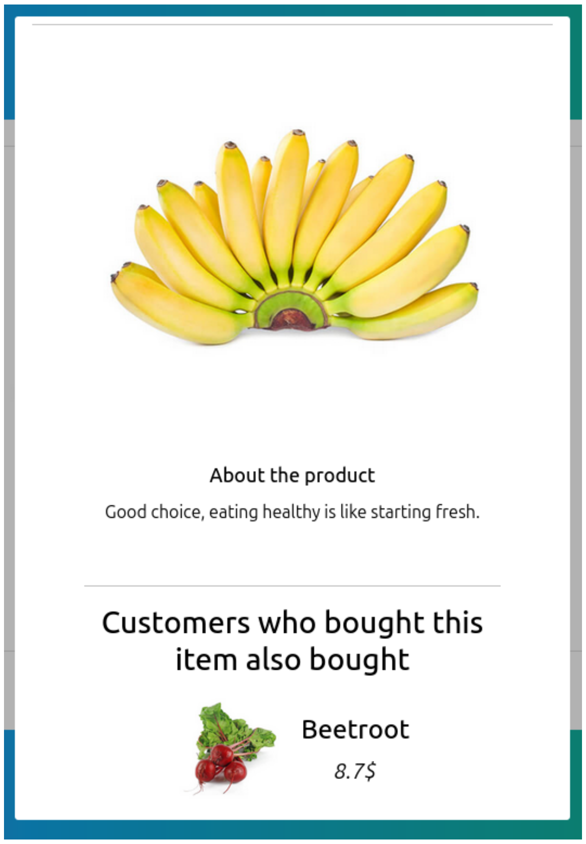
6 Recommendation service in action

After executing the script dataset.sh, we can see for each product the number of time the others products were purchased in the same cart.



```
samir@samir-VirtualBox:~/Desktop/LINFO2145-2024-2025$ curl admin:admin@10.23.86.36:3012/purchase-based/_design/queries/_view/most_co_purchased?group=true
{"rows":[
{"key":"Apple","value":{"Cucumber":7,"Grapes":8,"Brocolli":11,"Beetroot":8,"Cauliflower":6,"Banana":8,"Mango":5}},
{"key":"Banana","value":{"Cucumber":11,"Mango":9,"Beetroot":14,"Grapes":5,"Cauliflower":10,"Apple":8,"Brocolli":5}},
{"key":"Beetroot","value":{"Cauliflower":9,"Cucumber":8,"Grapes":11,"Mango":6,"Brocolli":10,"Banana":14,"Apple":8}},
{"key":"Brocolli","value":{"Beetroot":10,"Grapes":14,"Apple":11,"Cauliflower":12,"Mango":7,"Cucumber":8,"Banana":5}},
{"key":"Cauliflower","value":{"Beetroot":9,"Cucumber":9,"Mango":13,"Grapes":12,"Banana":10,"Apple":6,"Brocolli":12}},
{"key":"Cucumber","value":{"Beetroot":8,"Cauliflower":9,"Banana":11,"Mango":7,"Grapes":14,"Apple":7,"Brocolli":8}},
{"key":"Grapes","value":{"Banana":5,"Apple":8,"Mango":12,"Brocolli":14,"Cauliflower":12,"Beetroot":11,"Cucumber":14}},
{"key":"Mango","value":{"Apple":5,"Brocolli":7,"Cauliflower":13,"Banana":9,"Cucumber":7,"Grapes":12,"Beetroot":6}}
]}
samir@samir-VirtualBox:~/Desktop/LINFO2145-2024-2025$
```

We can see that Beetroot is the most bought item along the Banana.



About the product

Good choice, eating healthy is like starting fresh.

Customers who bought this item also bought

Beetroot

8.7$