1 Data Source

- I btained the data from text files: `customers.txt`, `ProductData.txt`, `purchase.txt`, `transactions.txt`, `suppliers.txt`, and `user_data.txt`.
- The data is sourced from the website https://www.horozelektrik.com/#/.

2 Data Processing

- The data in the text files is read line by line.
- For each file, there is a method (`readCustomersFromFiles`, `readProductsFromFiles`, etc.) to process the specific type of data and convert it into corresponding Java objects (`Customer`, `Product`, etc.).
- The data is split using the pipe (|) as a delimiter.

3 Node Creation

3.1 Customers

```
for (Customer customer : customers) {
    session.run("CREATE (:Customer {customerID: $customerID,
                  customerName: $customerName,
                  contactInformation: $contactInformation})",
                  Values.parameters("customerID", customer.getCustomerID(),
                  "customerName", customer.getCustomerName(),
                  "contactInformation", customer.getContactInformation()));
}
```

3.2 Products

```
for (Product product : products) {
    session.run("CREATE (:Product {productName: $productName,
                  category: $category,
                  sellingPrice: $sellingPrice, ...})",
                  Values.parameters("productName", product.getProductName(),
                  "category", product.getCategory(),
                  // ... (other product attributes)
                  ));
}
```

3.3 Purchases

```
for (Purchase purchase : purchases) {
    session.run("CREATE (:Purchase {purchaseId: $purchaseId,
                  purchaseDate: $purchaseDate, ...})",
                  Values.parameters("purchaseId", purchase.getPurchaseID(),
                  "purchaseDate", purchase.getPurchaseDate(),
                  // ... (other purchase attributes)
                  ));
}
```

3.4 Transactions

```
for (Transaction transaction : transactions) {
    session.run("CREATE (:Transaction {transactionId: $transactionId,
                  date: $date, ...})",
                  Values.parameters("transactionId", transaction.getTransactionID(),
                  "date", transaction.getDate(),
                  // ... (other transaction attributes)
                  ));
}
```

3.5 Users

```
for (User user : users) {
    session.run("CREATE (:User {username: $username,
                  role: $role, ...})",
                  Values.parameters("username", user.getUsername(),
                  "role", user.getRole(),
                  // ... (other user attributes)
                  ));
}
```

3.6 Suppliers

```
for (Supplier supplier : suppliers) {
    session.run("CREATE (:Supplier {supplierID: $supplierID,
                  supplierName: $supplierName, ...})",
                  Values.parameters("supplierID", supplier.getSupplierID(),
                  "supplierName", supplier.getSupplierName(),
                  // ... (other supplier attributes)
                  ));
}
```

4 Relationships

4.1 Transaction-Customer Relationship

```
for (Transaction transaction : transactions) {
    session.run("MATCH (na:Transaction {transactionId: $transactionId}),
                  (nb:Customer {customerID: $customerID}) " +
                  "CREATE (na)-[:MADE_PURCHASE]->(nb)",
                  Values.parameters("transactionId", transaction.getTransactionID(),
                  "customerID", transaction.getCustomerID()));
}
```

4.2 Transaction-Product Relationship

```
for (Transaction transaction : transactions) {
    session.run("MATCH (na:Transaction {transactionId: $transactionId}),
                  (nb:Product {productID: $productID}) " +
                  "CREATE (na)-[:OF_PRODUCT]->(nb)",
                  Values.parameters("transactionId", transaction.getTransactionID(),
                  "productID", transaction.getProductID()));
}
```

4.3 Purchase-Supplier Relationship

```
for (Purchase purchase : purchases) {
    session.run("MATCH (na:Purchase {purchaseId: $purchaseId}),
                  (nb:Supplier {supplierID: $supplierID}) " +
                  "CREATE (na)-[:FROM_SUPPLIER]->(nb)",
                  Values.parameters("purchaseId", purchase.getPurchaseID(),
                  "supplierID", purchase.getSupplierID()));
}
```

4.4 Purchase-Product Relationship

```
for (Purchase purchase : purchases) {
    session.run("MATCH (na:Purchase {purchaseId: $purchaseId}),
                 (nb:Product {productID: $productID}) " +
                 "CREATE (na)-[:THE_PRODUCT]->(nb)",
                 Values.parameters("purchaseId", purchase.getPurchaseID(),
                 "productID", purchase.getProductID()));
}
```

4.5 Purchase-User Relationship with Attribute

```
for (Purchase purchase : purchases) {
    session.run("MATCH (na:Purchase {purchaseId: $purchaseId}),
                 (nb:User {userId: $userId}) " +
                 "CREATE (na)-[:MADE_BY_USER {purchaseDate: $purchaseDate}]->(nb)",
                 Values.parameters("purchaseId", purchase.getPurchaseID(),
                 "userId", purchase.getUserID(),
                 "purchaseDate", purchase.getPurchaseDate()));
}
```

5 Neo4j Database Schema

5.1 Node Types

5.1.1 Customer

Properties:

- `customerID` (String)
- `customerName` (String)
- `contactInformation` (String)

5.1.2 Product

Properties:

- `productName` (String)
- `category` (String)
- `sellingPrice` (String)
- `image` (String)
- `stockQuantity` (String)
- `description` (String)
- `costPrice` (String)
- `minimumStockLevel` (String)
- `unitOfMeasure` (String)
- `dateAdded` (String)
- `lastUpdated` (String)
- `productID` (String)

5.1.3 Purchase

Properties:

- `purchaseId` (`String`)
- `purchaseDate` (`String`)
- `deliveryDate` (`String`)
- `quantity` (`String`)

5.1.4 Transaction

Properties:

- `transactionId` (`String`)
- `date` (`String`)
- `totalCost` (`String`)
- `discountsApplied` (`String`)

5.1.5 User

Properties:

- `username` (`String`)
- `role` (`String`)
- `password` (`String`)
- `userId` (`String`)

5.1.6 Supplier

Properties:

- `supplierID` (`String`)
- `supplierName` (`String`)
- `contactInformation` (`String`)

5.2 Relationships

- **MADE_PURCHASE:** Between `Transaction` and `Customer`
- **OF_PRODUCT:** Between `Transaction` and `Product`
- **FROM_SUPPLIER:** Between `Purchase` and `Supplier`
- **THE_PRODUCT:** Between `Purchase` and `Product`
- **MADE_BY_USER:** Between `Purchase` and `User` **Property:**
  - `purchaseDate` (`String`)

6 Database Operations

6.1 `getAllTheProduct()`

Retrieves all products from the database and maps them to `Product` objects.

```
MATCH (p:Product) RETURN p
```

6.2 `getAllTheUsers()`

Retrieves all users from the database and maps them to `User` objects.

```
MATCH (u:User) RETURN u
```

6.3 `getUserByUsername(String uName)`

Retrieves a user by their username.

```
MATCH (u:User {username: $username}) RETURN u
```

6.4 `addUser(User newUser)`

Adds a new user to the database.

```
CREATE (:User {username: $username, role: $role,
              password: $password, userId: $userId})
```

6.5 `getAllTheCustomers()`

Retrieves all customers from the database and maps them to `Customer` objects.

```
MATCH (c:Customer) RETURN c
```

6.6 `getAllTheSuppliers()`

Retrieves all suppliers from the database and maps them to `Supplier` objects.

```
MATCH (s:Supplier) RETURN s
```

6.7 `getAllThePurchases()`

Retrieves all purchases from the database and maps them to `Purchase` objects.

```
MATCH (p:Purchase)-[:FROM_SUPPLIER]->(s:Supplier),
      (p)-[:THE_PRODUCT]->(pr:Product),
      (p)-[:MADE_BY_USER]->(u:User)
RETURN p.purchaseId AS purchaseId,
       p.purchaseDate AS purchaseDate,
       p.deliveryDate AS deliveryDate,
       p.quantity AS quantity,
       s.supplierID AS supplierID,
       pr.productID AS productID,
       u.userId AS userId
```

6.8 `getAllTheTransactions()`

Retrieves all transactions from the database and maps them to `Transaction` objects.

```
MATCH (t:Transaction)-[:MADE_PURCHASE]->(c:Customer),
      (t)-[:OF_PRODUCT]->(p:Product)
RETURN t.date AS date,
       t.discountsApplied AS discountsApplied,
       t.totalCost AS totalCost,
       t.transactionId AS transactionId,
       c.customerID AS customerID,
       p.productID AS productID
```

6.9 `searchProducts(String searchTerm)`

Searches for products based on a given search term.

```
MATCH (p:Product)
WHERE p.productName =~ '(?i).*' + $searchTerm + '.*'
   OR p.category =~ '(?i).*' + $searchTerm + '.*'
   OR p.description =~ '(?i).*' + $searchTerm + '.*'
   OR p.unitOfMeasure =~ '(?i).*' + $searchTerm + '.*'
   OR p.costPrice =~ '(?i).*' + $searchTerm + '.*'
   OR p.sellingPrice =~ '(?i).*' + $searchTerm + '.*'
   OR p.stockQuantity =~ '(?i).*' + $searchTerm + '.*'
   OR p.minimumStockLevel =~ '(?i).*' + $searchTerm + '.*'
RETURN p
```

6.10 `searchCustomers(String searchTerm)`

Searches for customers based on a given search term.

```
MATCH (c:Customer)
WHERE c.customerName =~ '(?i).*' + $searchTerm + '.*'
   OR c.contactInformation =~ '(?i).*' + $searchTerm + '.*'
   OR c.customerID =~ '(?i).*' + $searchTerm + '.*'
RETURN c
```

6.11 `searchTransactions(String searchTerm)`

Searches for transactions based on a given search term.

```
MATCH (transaction:Transaction)-[:MADE_PURCHASE]->(customer:Customer),
      (transaction)-[:OF_PRODUCT]->(product:Product)
WHERE transaction.date =~ '(?i).*' + $searchTerm + '.*'
   OR transaction.discountsApplied =~ '(?i).*' + $searchTerm + '.*'
   OR transaction.totalCost =~ '(?i).*' + $searchTerm + '.*'
   OR transaction.transactionId =~ '(?i).*' + $searchTerm + '.*'
   OR customer.customerID =~ '(?i).*' + $searchTerm + '.*'
   OR product.productID =~ '(?i).*' + $searchTerm + '.*'
RETURN transaction, customer, product
```

6.12 `searchPurchases(String searchTerm)`

Searches for purchases based on a given search term.

```
MATCH (purchase:Purchase)-[:FROM_SUPPLIER]->(supplier:Supplier),
      (purchase)-[:MADE_BY_USER]->(user:User),
      (purchase)-[:THE_PRODUCT]->(product:Product)
WHERE purchase.purchaseDate =~ '(?i).*' + $searchTerm + '.*'
   OR purchase.deliveryDate =~ '(?i).*' + $searchTerm + '.*'
   OR purchase.purchaseId =~ '(?i).*' + $searchTerm + '.*'
   OR purchase.quantity =~ '(?i).*' + $searchTerm + '.*'
   OR supplier.supplierID =~ '(?i).*' + $searchTerm + '.*'
   OR product.productID =~ '(?i).*' + $searchTerm + '.*'
   OR user.userId =~ '(?i).*' + $searchTerm + '.*'
RETURN purchase, supplier, user, product
```

6.13 `searchUsers(String searchTerm)`

Searches for users based on a given search term.

```
MATCH (u:User)
WHERE u.username =~ '(?i).*' + $searchTerm + '.*'
   OR u.role =~ '(?i).*' + $searchTerm + '.*'
   OR u.userId =~ '(?i).*' + $searchTerm + '.*'
RETURN u
```

6.14 searchSuppliers(String searchTerm)

Searches for suppliers based on a given search term.

```
MATCH (s:Supplier)
WHERE s.supplierName =~ '(?i).*' + $searchTerm + '.*'
   OR s.contactInformation =~ '(?i).*' + $searchTerm + '.*'
   OR s.supplierID =~ '(?i).*' + $searchTerm + '.*'
RETURN s
```

6.15 totalSalesPerMonth(String startDate, String endDate)

Retrieves total sales per month within a specified date range.

```
MATCH (t:Transaction)
WHERE t.date >= $startDate AND t.date <= $endDate
WITH t.date AS DateString, toFloat(t.totalCost) AS Quantity
RETURN SUBSTRING(DateString, 0, 7) AS Month, SUM(Quantity) AS TotalSales
ORDER BY Month
```

6.16 bestSellingProductsByUnitsSold()

Retrieves the best-selling products based on units sold.

```
MATCH (t:Transaction)-[:OF_PRODUCT]->(p:Product)
WHERE t.date >= '2019-01-01' AND t.date <= $currentDate
WITH p, SUM((toFloat(t.totalCost) + toFloat(t.discountsApplied)) /
          toFloat(p.sellingPrice)) as unitsSold
RETURN p.productName as Product, unitsSold
ORDER BY unitsSold DESC LIMIT 10
```

6.17 bestProfitableProducts()

Retrieves the best profitable products.

```
MATCH (t:Transaction)-[:OF_PRODUCT]->(p:Product)
WHERE t.date >= '2019-01-01' AND t.date <= $currentDate
WITH p, SUM(toFloat(t.totalCost) - toFloat(t.discountsApplied)) as totalNetSales,
     SUM((toFloat(t.totalCost) + toFloat(t.discountsApplied)) /
          toFloat(p.sellingPrice)) as unitsSold
RETURN p.productName as Product, totalNetSales, unitsSold
ORDER BY totalNetSales DESC LIMIT 10
```

6.18 averageDeliveryTime()

Retrieves the average delivery time for products from suppliers.

```
MATCH (p:Purchase)
WHERE p.deliveryDate IS NOT NULL
RETURN AVG(datetime(p.deliveryDate) - datetime(p.purchaseDate)) AS avgDeliveryTime
```

6.19 `getProductStockInfo()`

Retrieves information about product stock.

```
MATCH (p:Product)
RETURN p.productName AS Product, p.stockQuantity AS StockQuantity,
       p.minimumStockLevel AS MinimumStockLevel
```

6.20 `retrieveGraph()`

Retrieves a small graph from the database, including customers, transactions, products, purchases, suppliers, and users.

```
MATCH (c:Customer)-[:MADE_PURCHASE]->(t:Transaction)-[:OF_PRODUCT]->(p:Product),
      (t)-[:FROM_SUPPLIER]->(s:Supplier),
      (t)-[:MADE_BY_USER]->(u:User)
RETURN c, t, p, s, u
```