Polyglot Project

Czech Technical University in Prague

Databázové systémy 2 (B4M36DS2)

Marek Jagoš

7 January 2025

Contents

1 Polyglot project

The purpose of this student project is to develop application that uses multiple database systems for different tasks (**polyglot persistence**). It aims to demonstrate how different database systems can work together effectively to provide high performance, scalability, and usability.

1.1 Subject area

The application concerns itself with an online store that sells used and refurbished laptops. The store offers two main types of products: used laptops and refurbished laptops. For refurbished laptops, the company either upgrades existing models or repurposes components from multiple laptops to create a new one. These refurbished laptops are sold under new SKU numbers and come with a short warranty. The company sells these laptops through a web application that requires appropriate features.

1.2 Functional requirements

The functional requirements for the application are the following:

- Product browsing and filtering
- User activity logging
- Sessions
- Caching
- User accounts
- Shopping cart
- Purchase history

1.3 Database system utilization

- Product inventory: PostgreSQL
- User accounts: PostgreSQL
- User sessions: Redis
- Shopping carts: Redis
- Purchase history: MongoDB
- Activity logs: Cassandra
- Personalized recommendations: Neo4j
- Caching frequent accessed data: Redis

PostgreSQL is well-suited for managing tasks that demand data consistency, complex querying, and a rigid schema. It will handle:

- Product Filtering: The structured nature of product data, combined with the need for advanced filtering queries, makes PostgreSQL an ideal choice.
- User Accounts: Constraints enforcement and data consistency are critical for user authentication, authorization, and personal data management.

Redis excels at handling simple key-value operations with fast in-memory performance. It will be used for:

- Shopping Carts: Shopping carts are inherently simple key-value data structures that do not require complex queries.

- Session Management: Session data typically involves rapid access and does not necessitate querying the value itself.
- Data Caching: Redis enables efficient caching for frequently accessed data, reducing database load and improving response times.

MongoDB is well-suited for semi-structured data and scenarios that require occasional value querying without frequent updates. It will manage:

- User Purchase Statements: MongoDB will store and retrieve user purchase history, offering flexibility for semi-structured document storage.

Cassandra is optimized for high-volume write operations and time-series data, with minimal reliance on complex queries. It will be responsible for:

- Logging: Logging systems often involve heavy sequential write operations, and read queries are infrequent. Cassandra's architecture handles these requirements efficiently.

Neo4j specializes in managing graph-based data and efficiently handling relationship queries. It will be utilized for:

- User Relationship Tracking and Recommendations: Neo4j will enable identifying user relationships and recommending products based on shared purchase patterns. For example, a user who buys a laptop may receive recommendations based on other users' purchasing behavior.

2 Implementation

2.1 Application implementation

The application is built using **Python** and the **Flask Web Application Framework**. Database instances are managed through **Docker Compose**, with each database instance being assigned a distinct localhost port for accessibility. Python scripts establish connections to these databases using appropriate libraries tailored for each database type.
Each database connection is encapsulated within its own **microservice**, which is responsible for providing database-specific operations. These microservices expose their functionality via API routes and run locally on ports ranging from 5001 to 5005. The main console application runs on port 5000 and interacts with these microservices by sending HTTP requests to their respective endpoints, enabling it to perform necessary actions without directly interfacing with the databases.

2.2 Application execution

The application is installed by creating and activating Python environment through requirements file. The application is executed as:

```
run_app.sh
```

2.3 Test pipeline

Corresponding test pipeline was created for this project. It consists of

- Unit tests - Test each microservice and its functions
- Integration tests - Test interaction between microservices
- User tests - Test higher level user-exposed functions

Testing pipeline is executed as (application must be running):

```
run_tests.sh
```

3 Console application

3.1 Interactive session

User interacts with the application through a interactive Python session. Some user-facing application commands are exposed to the user, which he can freely execute to interact with the application. These commands are:

- fetch_products
- login
- logout
- update_cart
- clear_cart
- purchase = app.purchase
- get_statements
- read_statement
- read_log

There are, of course, many other methods executed inaccessible to the user that streamline the application functionality, but these will not be listed here.

3.2 Example interaction

```
fetch_products({"vendor": "HP", "product_condition": "Excellent", "Screen Size": "15.6"})
> [
   [
      5,
      "HP",
      "Refurbished Laptop",
      "Excellent",
      "RFBSHLPT-5",
      2,
      1,
      829.0,
      {
         "Disk Storage Size":256,
         "Disk Type":"SSD",
         "Operating system":"Windows 11",
         "Processor Name":"Intel Core i9-9900X",
         "RAM Size":8,
         "Screen Size":15.6
      },
      "Fri, 03 Jan 2025 23:45:40 GMT",
      "Fri, 03 Jan 2025 23:45:40 GMT"
   ],
   [
      73,
      "HP",
      "Refurbished Laptop",
      "Excellent",
      "RFBSHLPT-66",
      1,
      3,
      759.0,
```

```
      {
         "Disk Storage Size":128,
         "Disk Type":"HDD",
         "Operating system":"Windows 11",
         "Processor Name":"Intel Core i3-8300",
         "RAM Size":4,
         "Screen Size":15.6
      },
      "Fri, 03 Jan 2025 23:45:41 GMT",
      "Fri, 03 Jan 2025 23:45:41 GMT"
   ]
]
login("testuser1", "password")
> 'Incorrect credentials'
login("testuser1", "password1")
> 'Welcome back testuser1'
update_cart(5, 1)
> {5: 1}
purchase()
> 1
read_statement(1)
> {'_id': 1, 'creation_date': 'Fri, 03 Jan 2025 23:31:12 GMT', 'purchase': {5: 1}, 'user_id': 7}
update_cart(5, 2)
> {5: 2}
logout()
> 'User testuser1 logged out'
login("testuser1", "password1")
> 'Welcome back testuser1'
read_cart()
> {5: 2}
read_log(7)
> [
   {
      "action":"Read Logs",
      "parameters":{
         "filter":"{'user_name': 7}"
      },
      "tags":[
         "MONGODB"
      ],
      "timestamp":"Fri, 03 Jan 2025 23:48:15 GMT",
      "userId":7
   },
   ...
]
follow(7)
> True
get_recommendation()
> [1] # Laptops that followed users have purchased recently
```