AutoBasket

"With AutoBasket, it's never been easier to get everything you need for a great meal!"

Dhiraj Kumar Sah
Dhru Sanjay Prajapati
Harshil Bhavsar
Keerat Singh

Reporting Date: 2nd February2024

TABLE OF CONTENTS

ABSTRACT

This week, we explored system architecture and database solutions for large datasets. We discussed various architectures, including monolithic, client-server, service-oriented, microservices, peer-to-peer, event-driven, and layered architecture, comparing their pros and cons. We also explored database designs for large datasets, such as relational databases, NoSQL databases, column-store databases, in-memory databases, cloud databases, NewSQL databases, and document databases.

We chose MongoDB as our database for the project, given its flexibility in handling structured and unstructured data, scalability, and efficiency in handling large datasets. We designed the database schema to match our dataset, making data storage convenient and suitable for analysis.

INTRODUCTION

AutoBasket is a company founded by Larry and Veronica Smiles with the purpose of simplifying the grocery shopping experience for individuals and families. Situated in the heart of Toronto, the company's headquarters pulsate with the energy of a city known for innovation and diversity. The company's app automates grocery and recipe lists, linking recipes to required products, and streamlining the shopping process. Today, AutoBasket assists households across the world to save time and energy on their weekly grocery runs, making it easier for busy families and individuals to get everything they need for a great meal.

AutoBasket has hired us as an intern to focus on all areas of development for the organization and to help create solutions to common issues within the industry. Our role as an intern is not only focused on the knowledge you have learned in school but also on developing your soft skills, including presentations, teamwork, and leadership. This internship will provide us with valuable real-world experience and an opportunity to contribute to the continued success of AutoBasket.

SYSTEM ARCHITECTURES

In the world of software and information technology, various system architectures are commonly employed. These include:

1. **Monolithic Architecture**: The Monolithic Architecture comprises a single, self-contained unit that encompasses the entire application. While suitable for smaller applications, it can become challenging to maintain and scale as the system grows and complexity.
2. **Client-Server Architecture**: This architecture divides the system into two distinct parts: clients and servers. Clients are responsible for interacting with the user, while servers handle the processing and management of data. This separation allows for distributed processing, scalability, and a clear distinction between client-side and server-side logic.
3. **Service-Oriented Architecture (SOA)**: SOA focuses on designing systems as a collection of interoperable services that can be reused and combined to meet various business needs. By promoting loose coupling, flexibility, and interoperability, SOA enables the development of complex systems that can adapt to changing requirements.
4. **Microservices Architecture**: In this style, it breaks down a system into small, independent services, each handling a specific task, allowing for flexibility, scalability, and fault isolation.
5. **Event-Driven Architecture (EDA)**: A system design that emphasizes the detection, production, and consumption of events to trigger actions or communicate changes between components, promoting scalability and loose coupling, enabling real-time response to changing conditions.
6. **Layered Architecture**: A network structure with hierarchical layers, each with its own responsibilities. The bottom layer handles data transmission, the middle layer manages data routing, and the top layer provides services and interfaces. Organized, hierarchical, responsible, data transmission, routing, services, and interfaces.

7. **Peer-to-Peer Architecture (P2P)**: A decentralized network structure where equal peers communicate directly without a central server. Nodes can request and provide resources, dynamically discovering and communicating with other nodes. Decentralized, equal, distributed, dynamic, and self-organizing.
8. **Event-Driven Messaging Architecture**: A system that sends and receives messages in response to events. When an event occurs, a message is sent to the appropriate module, which then processes the message and triggers further actions like decoupling, event-based, message passing, modular, scalability.
9. **Hybrid Architectures**: This combines different architecture styles, such as centralized and decentralized, or peer-to-peer and client-server. This approach allows for the benefits of multiple architectures, increased flexibility, and better adaptability.

For our project, we have opted for a client-server architecture, a widely used and proven system design that aligns with the features and requirements commonly found in successful applications. The client-server model divides the system into two distinct components: the client, responsible for user interaction and presentation, and the server, handling data processing and storage. This architecture offers several benefits such as centralized data management, scalability, and efficient resource utilization. By centralizing data on the server, we can ensure consistency and integrity, while also facilitating easier updates and maintenance. Scalability is achievable by adding more servers to meet the increased demand. However, we acknowledge the dynamic nature of technology and its rapid advancements. Should we identify a better alternative that suits our project needs, we remain open to exploring and adopting it to ensure the most effective solution for our goals.

COMPARISION CHART

Architecture	Scalability	Complexity	Fault Tolerance
Monolithic	Limited horizontal scalability	Easier to develop and test	Single point of failure
Client-Server	Allows for distributed processing, scalability	Moderate complexity	Can be designed with fault tolerance mechanisms
Service-Oriented (SOA)	Depends on specific implementation	Moderate complexity	Promotes reusability, flexibility, and interoperability
Microservices	Easier to scale	Complex development and testing	Allows for fault isolation

Event-Driven (EDA)	Promotes loose coupling and scalability	Moderate complexity	Decoupling components through event-based communication
Layered	Limited horizontal scalability	Moderate complexity	Limited fault isolation
Peer-to-Peer (P2P)	Depends on specific implementation	Moderate complexity	Depends on specific implementation
Event-Driven Messaging	Enables scalability, fault tolerance, and loose coupling	Moderate complexity	Enables fault tolerance through decoupled communication

Table-1: Comparison Chart

PROS AND CONS FOR SYSTEM ARCHITECTURE

Architecture	**Pros**	**Cons**
Monolithic	• Simple to design and implement. • Easier to debug since the entire system is in one place. • Single codebase, making it easier to maintain.	• Scalability can be a challenge as the system grows. • Difficult to adopt new technologies or languages. • Changes in one module may affect the entire system.
Microservices	• Scalability and flexibility, as each service can be developed, deployed, and scaled independently. • Easier adoption of new technologies in specific services. • Fault isolation, as a failure in one service doesn't necessarily affect the entire system.	• Complexity in managing and coordinating multiple services. • Increased communication overhead between services. • Challenges in maintaining consistency and data integrity.

Client-Server	• Centralized control and management of resources. • Easier to implement security measures in a controlled environment. • Scalability by adding more servers or upgrading existing ones.	• Single point of failure (the server). • Potential bottlenecks if the server becomes overloaded. • Increased complexity in managing server-client communication.
Peer-to-Peer	• No single point of failure, as each node in the network can act as both a client and a server. • Scalability, as adding more nodes can enhance the network's capabilities. • Decentralization, which can improve fault tolerance and reduce dependence on a central authority.	• Difficulty in managing and coordinating peer interactions. • Security concerns, as each peer needs to be responsible for its security. • Potential for inconsistent data if peers have different versions.
Service-Oriented Architecture (SOA)	• Reusability of services across applications. • Interoperability between different systems and platforms. • Easier maintenance and updates, as changes in one service do not affect others.	• Initial setup and development can be time-consuming. • Complexity in managing service dependencies. • Potential for performance bottlenecks due to service calls.
Event-Driven	• Responsiveness to events and real-time processing. • Loose coupling between components, promoting flexibility. • Scalability by adding event processors as needed.	• Increased complexity in managing and tracking events. • Debugging can be challenging due to asynchronous nature. • Potential for event order issues and maintaining consistency.

Event-Driven Messaging:	• Events are processed as they occur, enabling real-time response to changes. • Easily scalable by adding more event processors to handle increased event load. • Components are decoupled, promoting flexibility and easier maintenance.	• Managing and tracking events can be complex, especially in large and dynamic systems. • Debugging asynchronous systems can be more challenging than synchronous systems. • Ensuring the correct order of events can be crucial and may require additional mechanisms.
Layered Architecture:	• Components are organized into layers, promoting modularity and ease of maintenance. • Each layer has a specific responsibility, making the system more understandable and maintainable. • Layers can be reused in different parts of the system or in different projects.	• Each layer introduces some level of overhead in terms of communication and processing. • Changes in one layer may require adjustments in other layers, potentially making the system less flexible. • The communication between layers can introduce latency and affect performance.

Table-2: Pros & Cons of different System Architecture

DATABASE SOLUTIONS FOR LARGE DATASETS

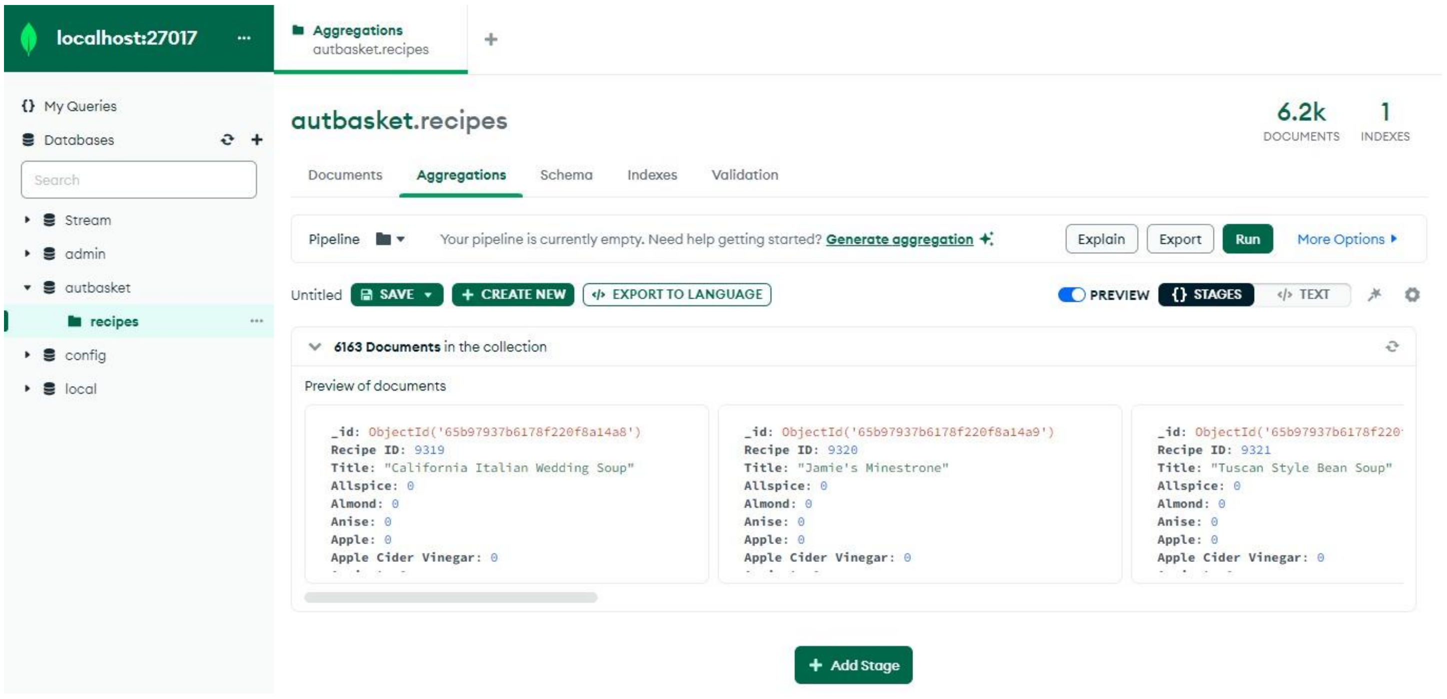
Multiple database designs can be utilized to manage large datasets, each catering to the specific needs of the project. These designs include:

1. **Relational Database**: Such as PostgreSQL, MySQL, and Oracle, are efficient for handling large datasets due to their use of Structured Query Language (SQL), allowing for fast querying and manipulation of data.
2. **NoSQL Databases**: Like MongoDB, Cassandra, and Riak, are designed to handle large amounts of unstructured or semi-structured data. They are horizontally scalable, meaning they can handle increasing data by adding more servers to the cluster.
3. **Column-store Databases**: Such as Apache Cassandra, Amazon Redshift, and Google Bigtable, are optimized for fast query performance when handling large datasets by storing data in columns instead of rows.
4. **In-Memory Databases**: Like SAP HANA, Oracle Times Ten, and Volt DB, store data in the main memory (RAM) instead of on disk, providing faster data access and querying times, often used for real-time analytics and reporting.

5. **Cloud Databases**: Like Amazon Aurora, Google Cloud SQL, and Microsoft Azure Database Services, offer fully managed database services in the cloud, providing scalability, high availability, and security for large datasets.
6. **NewSQL Databases**: Such as Google Spanner, Amazon Aurora, and CockroachDB, provide a balance between the scalability of NoSQL databases and the ACID transactions of relational databases, suitable for large datasets requiring strong consistency and high availability.
7. **Document Databases**: Like MongoDB, Couchbase, and RavenDB, store data in flexible JSON or XML documents, allowing for easy data modeling and flexible schema design, often used for content management, inventory management, and IoT data storage.

DATABASE SCHEMA

We have opted to implement MongoDB as the database for the project. MongoDB is a kind of NoSQL database that doesn't need a fixed structure for data, making it flexible. It stores data in a format like JSON and is known for being fast, scalable, and easy to work with for different types of data.



6163 Documents in the collection

Preview of documents

```
_id: ObjectId('65b97937b6178f220f8a14a8')
Recipe ID: 9319
Title: "California Italian Wedding Soup"
Allspice: 0
Almond: 0
Anise: 0
Apple: 0
Apple Cider Vinegar: 0
```

```
_id: ObjectId('65b97937b6178f220f8a14a9')
Recipe ID: 9320
Title: "Jamie's Minestrone"
Allspice: 0
Almond: 0
Anise: 0
Apple: 0
Apple Cider Vinegar: 0
```

```
_id: ObjectId('65b97937b6178f220
Recipe ID: 9321
Title: "Tuscan Style Bean Soup"
Allspice: 0
Almond: 0
Anise: 0
Apple: 0
Apple Cider Vinegar: 0
```

Figure -1: Database schema for AutoBasket

We chose MongoDB because our project deals with both structured and unstructured data. This flexibility is important because users might not always enter details perfectly. For our analysis, we'll likely have a lot of data, and MongoDB is known for handling large amounts of data quickly and efficiently – exactly what we need for our machine-learning project.

To make things easier for analysis, we've set up the database schema to match the dataset we have. This way, the data storage is convenient and fits well with our analysis goals.

CONCLUSION

In conclusion, we have made significant progress in our project this week by finalizing two crucial components: the architectural framework and the database technology. We have chosen to implement a client-server architecture, which is renowned for its scalability and ability to handle distributed computational loads. This will enable our project to efficiently manage extensive training or inference tasks and provide quick responses to real-time inference requests.

Additionally, we have selected MongoDB as our database technology due to its flexibility, scalability, and ability to handle unstructured and semi-structured data efficiently. MongoDB's schema flexibility feature, which allows for adapting to different data styles and formats, makes it an ideal choice for machine learning projects like ours. This decision will enable us to effectively manage the complexities of our project and ensure efficient data management and retrieval.

However, we also recognize that our project may require other architectural frameworks and database schemas in the future. For instance, we may need to adopt a microservices architecture to accommodate the scalability and flexibility requirements of our project. Additionally, we may need to utilize other database schemas, such as SQL or cloud-based databases, to handle diverse data types and ensure optimal data management.

Therefore, we will continue to monitor our project's evolving needs and adapt our architecture and database schema accordingly. By doing so, we can ensure that our project remains agile, scalable, and efficient in handling large datasets and complex machine learning tasks.

REFRENCES

[1] Lentreo. (2023, 5 22). *Types of System Architectures*. Retrieved from Medium: https://medium.com/@abhijitgunjal1648/types-of-system-architectures-eb37dd496971

[2] *MongoDB*. (2024, 2 2). Retrieved from MongoDB: https://www.mongodb.com/

[3] Terra, J. (2023, 8 7). *What is Client-Server Architecture? Everything You Should Know*. Retrieved from SimpliLearn: https://www.simplilearn.com/what-is-client-server-architecture-article

[4] *What are the pros and cons of different software architectures?* (2024, 2 2). Retrieved from Educative: https://www.educative.io/answers/what-are-the-pros-and-cons-of-different-software-architectures