# Inside a Recipe Discovery System

BY ABDULRAHMAN WAHEED

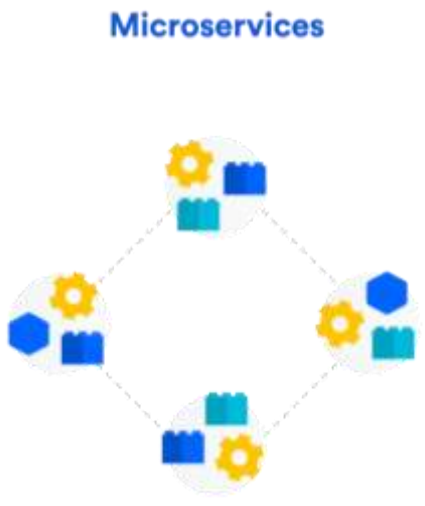
**Table of Contents**

**Introduction**

We are all burdened by the never-ending wondering about what to eat and how to prepare it. Traditional recipe search methods are often marred by intricacies and time-consuming processes, leading to a less than-optimal user experience .This project develops a recipe helper to help you find something to eat. In this report we will be discussing The Architectures that we might be utilizing  to crate this project.we will describe them and compare to see which one will be the most suitable to create out recipe discovery system.

**Microservices Architecture**

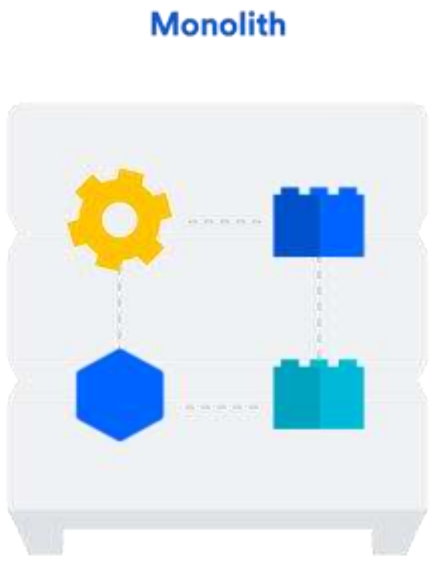
The first architecture we will be discussing is microservices. In Microservices each service can undergo development, deployment, operation, and scaling independently, without impacting the operations of other services. There is no requirement for services to share code or implementation, with communication occurring exclusively through clearly defined APIs. Moreover Every service within this architecture is tailored to fulfill distinct functionalities, concentrating on addressing specific issues. In our case we have Recipe retrieval Service which manages storage of recipes and provide endpoints for searching recipes based on ingredients, dietary preferences, cooking methods, and cuisines. if a service becomes complex over time and additional code is written it can be subdivided into smaller, more specialized services in our case we can have a nutrition service which has nutrition information about our recipes . This approach offers scalability and flexibility but requires careful coordination and management of multiple services.



Source of Image : https://divante.com/blog/wp-content/uploads/2020/01/Frame-1.png

**Monolithic Architecture**

Monolithic architecture is a software architectural style in which a whole application is constructed as a single unit. This design connects and interdepends all of the application's components. Monolithic programs are often built from a single codebase, with all the modules tightly coupled together. This means that any changes or upgrades to the system must be deployed in its entirety, which can be time-consuming and might cause the risk of breaking the whole system, especially for large and complicated systems. using a monolithic architecture in our app all aspects would be handled within a single application. For example, the system could include a database of recipes with details on cooking time, difficulty level, nutritional information, ratings, and reviews all in one component . While this approach may offer simplicity



Source of Image : https://divante.com/blog/wp-content/uploads/2020/01/Frame-1.png in development and deployment, as the application grows, managing and scaling could become challenging. deploying updates such can be risky because a change in one part of the application can potentially affect other parts.

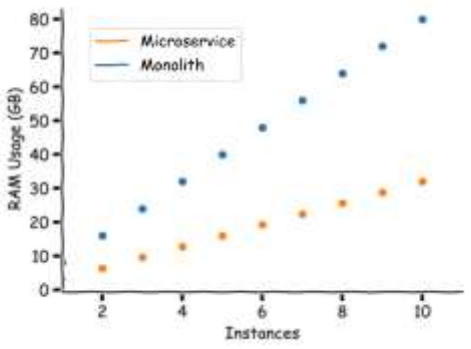
**Representational State Transfer (REST)**

Another architecture is rest (representational state transfer) an architectural style for designing networked applications. In REST software applications communicate with each other over the internet using standard HTTP methods such as GET, POST, PUT, DELETE, etc, and exchanging representations of resources such as in our case recipes, users, ratings, etc.  In REST a client-server structure is used with HTTP handling the requests however the Communication between client and server must be stateless, meaning no client data is stored between requests, and each request stands alone. Using a REST architecture, the recipe system could use APIs for each functionality, such as searching for recipes, retrieving nutritional information, fetching ratings and reviews, and generating a shopping list. Clients could make HTTP requests to these APIs to interact with the system. For instance, a client could request recipes with specific ingredients and dietary preferences, and the server would respond with matching recipes along with details like cooking time, difficulty level, and nutritional information. This approach offers simplicity in communication but requires careful design to ensure discoverability and security.

Event-Driven Architecture (EDA) is a modern architecture pattern built from around the integration of small, loosely connected services designed to publish, consume, or route events. Events denote major occurrences or change in the state of a system or its surrounding environment. Each event encapsulates a transformative moment, representing a modification in the system's state or an update. For our case, this could involve actions such as adding an ingredient to our shopping list, a user submitting a rating to the system, or when a user requests recipes with specific ingredients, an event could be triggered to search for matching recipes. Events possess the capability to convey comprehensive state information, summarizing details like item names, prices, or quantities in an order. Alternatively, events can function with identifiers only, for example the recipe #8942 has been added. EDA philosophy of loose coupling between producer and consumer services facilitates seamless scalability, effortless updates, and the autonomous deployment of individual components within a system. As a result, EDA not only enhances the agility of the system but also provides a robust foundation for managing complex workflows in a more adaptable manner.

## Implementation

Microservices architecture seems to be the most suitable architecture to use. Since Each service would handle a specific aspect of the functionality, such as recipe retrieval, nutritional information retrieval, ratings and reviews, and ingredient shopping list generation which is more flexible and less risky.more over it consumes less ram which is better for optimization.



Source of Image : https://thenewstack.io/microservices/microservices-vs-monoliths-an-operational-comparison/

To do build the system we have to create the following services:

Recipe Service: This service manages the retrieval and storage of recipes. It would provide endpoints for searching recipes based on ingredients, dietary preferences, cooking methods, and cuisines. Each recipe entry would include cooking time, difficulty level, nutritional information, health benefits, and possibly ratings and reviews.

Nutrition Service: This service is responsible for fetching and calculating the nutritional information for each recipe. It could integrate with external APIs or databases to gather data about the ingredients used in each recipe and calculate the overall nutritional content.

Rating and Review Service: This service handles the storage and retrieval of ratings and reviews for each recipe. Users can submit ratings and reviews, which are stored in a database and associated with the corresponding recipe.

Shopping List Service: This service generates a shopping list for each recipe based on the required ingredients. It could calculate the estimated price of the ingredients by querying external APIs or databases for current prices.

API Gateway: This Acts as a single entry point for clients to access the system's functionality and Routes requests to the appropriate microservices based on the request type.

In conclusion, the microservices architecture is the best solution after considering a variety of architectural options for creating a recipe discovery system. This architecture ensures flexibility and reduces risk by enabling autonomous service creation, deployment, and scalability. We can efficiently manage complexity and improve scalability by segmenting the system into discrete capabilities managed by independent microservices, such as recipe retrieval, nutritional data, ratings and reviews, and shopping list creation.

References:

1. Amazon Web Services. (2022). What is EDA? - Event Driven Architecture Explained. Retrieved from https://aws.amazon.com/what-is/eda/
2. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.
3. Richardson, C., & Fowler, M. (2018). Microservices vs. Monoliths: Which architecture is right for your project? ThoughtWorks.
4. Hohpe, G., & Woolf, B. (2003). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley.
5. Pautasso, Cesare; Wilde, Erik; Alarcon, Rosa (2014), REST: Advanced Research Topics and Practical Applications, Springer.