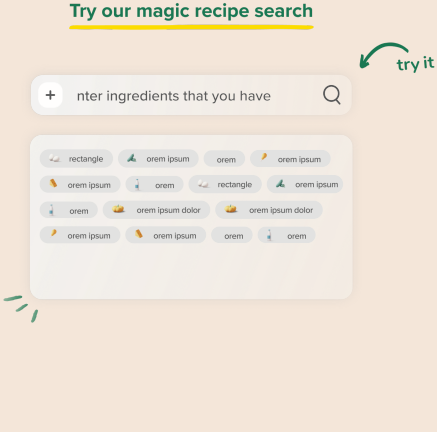
MY REAL-TIME BACKEND ARCHITECTURE PROJECT

mealit

Project Overview:

My project is a service that enables searching and filtering through a big data of recipes and presents them to users.



Real-Life Expectations:

Expected Number of Users:

- Initial Launch: Approximately 5,000 users.
- Active Users: Around 70%.

Operations per User:

- Browsing recipes, saving favorites, and creating meal plans, with an estimated average of 5-10 operations per session.

Bandwidth Considerations:

- Average Data per Operation: 500 KB - 2 MB, depending on content size (e.g., images, recipe details).

Update Schedule:

- **Feature Releases:** Monthly or bi-monthly.
- **Content Additions:** Weekly updates with new recipes.
- **Bug Fixes:** Weekly or as needed.
- **Major Updates:** Every 3-6 months for significant enhancements.

Traffic Patterns:

- **Regular Traffic:** Consistent user activity throughout the day.
- **Peak Traffic:** Increased usage during meal planning times (mornings and evenings).

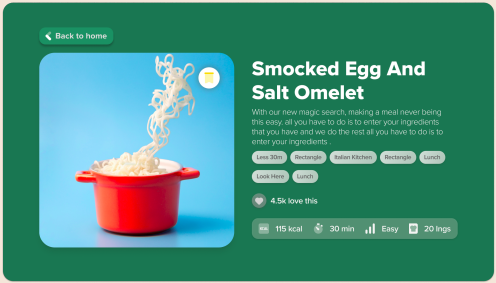
STORAGE ARCHITECTURE SOLUTIONS

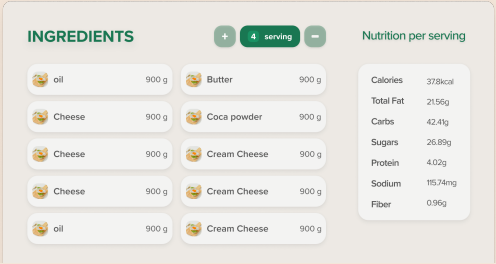
Before diving into the solutions,

let's first look at the structure of the recipes data.

```
{
"name": "string",
"description": "string",
"ingredients": ["string"]
}
```

Recipe schema





The primary goals :

Availability

Ensure that the database is accessible and operational as much as possible.

Efficiency

Optimize data retrieval and manipulation for speed and resource efficiency.

Scalability

Designing the system to handle increasing amounts of data and traffic seamlessly.

Text Search Solution (e.g., Elasticsearch)

for text search functionality on the data.

Configuration:

- **Data Indexing**: Index data with frequently searched words.
- **High Availability**: Use multiple nodes.
- **Scalability**: Implement sharding and replication.
- **Caching Layer**: Improve response times and reduce load.

- **Data indexing:**

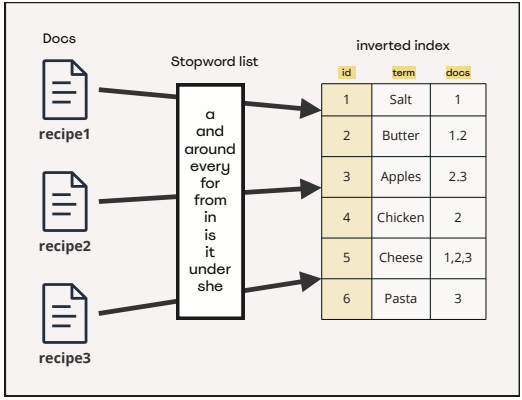
  It is a data structure technique used to locate and quickly access data in databases.

- **How Data indexing may works ?**

The **inverted index** is a database index storing a mapping from content, such as words or numbers, to its locations in a database, or in a document or a set of documents.

To **build the inverted index**, we have to split those sentences into tokens, in our case, the tokens are words. After this, tokens will be saved with links to documents.

Sometimes we can transform tokens before saving and searching such as Dropping of the **stop words**, ***Lemmatization***..



- # Replication and Partitioning:

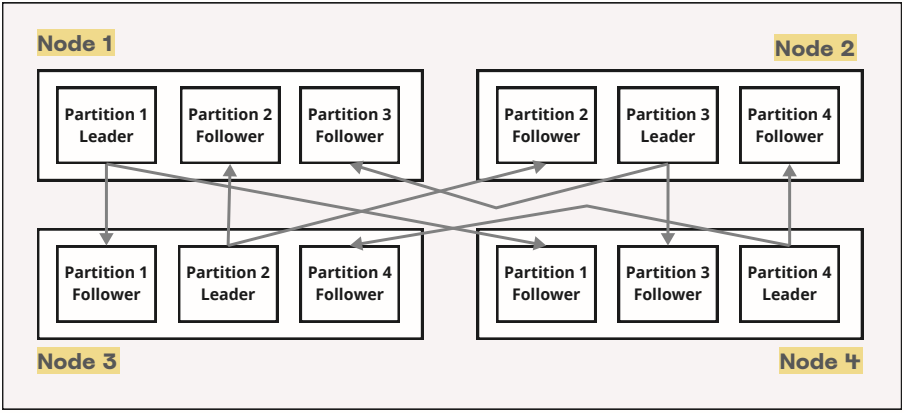
Replication means keeping a copy of the same data on multiple machines that are connected via a network.

Partitioning is a way of intentionally breaking a large database down into smaller ones.

Why do we need Replication and Partitioning ?

- To keep data geographically close to your users (and thus **reduce latency**)
- To allow the system to continue working even if some of its parts have failed (and thus **increase availability**)
- To scale out the number of machines that can serve read queries (and thus **increase read throughput**)

Example for Replication and Partitioning

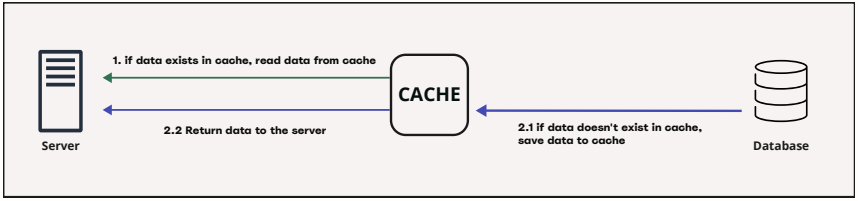


Cache :

A cache is a temporary storage area that stores the result of expensive responses or frequently accessed data in memory so that subsequent requests are served more quickly.

Use Cases:

- Store user-specific data such as favorite recipes and past searches.
- Store search results to reduce repeated searches.
- Cache popular and trending recipes for quicker access by users.



Managing Requests Per Second (RPS)

What Does RPS Mean?

Requests per second (RPS) is a crucial metric for understanding a server's capacity to handle concurrent requests. It helps in assessing the server's performance and scalability, ensuring it can manage the expected load without compromising the user experience.

When will the RPS be higher or lower in the Mealit search service?

In the Mealit search service, RPS will vary based on user activity patterns. During peak times, RPS will be higher, while during off-peak times, such as late at night or early morning, user activity might decrease, leading to a lower RPS.

Example

Higher RPS (up to 150-200)	Lower RPS (around 20-30)
Morning (7 AM - 9 AM): Users planning breakfast or looking for lunch ideas before work.	**Late Night (11 PM - 6 AM):** Most users are likely to be asleep, leading to fewer searches.
Afternoon (11 AM - 1 PM): Users searching for lunch recipes.	**Early Morning (3 AM - 6 AM).**
Evening (5 PM - 8 PM): Users planning dinner, which is typically the busiest time for meal planning.	**Mid-Morning (9 AM - 11 AM):** Users might be at work or busy with their daily routines, resulting in fewer recipe searches.
Weekends and Holidays: more users might be cooking at home, leading to increased searches.	**Afternoon (2 PM - 4 PM)**

Managing RPS:

- **Auto Scaling OR Scheduled scaling:**
  - Minimum: 2 instances, Maximum: 5 instances.
  - Implement automatic scaling based on predictable load changes.
  - Create scheduled actions to increase or decrease capacity at specific times.
- **Proxy, Load Balancer (e.g., NGINX):**
  - A proxy load balancer is a solution that acts as a traffic proxy and distributes network or application traffic across multiple servers.
- **Rate Limiter :**
  - Control the rate at which incoming requests or actions are processed or served by a system. rate limiter imposes constraints on the frequency or volume of requests from clients to prevent overload, maintain stability, and ensure fair resource allocation.
  - Limit to 10 requests per second per user.

That's All, Thanks!