Design Document - Menu-Service

Technology Stack

Current Implementation

1. **Architecture:** Monolith (from the assignment perspective)
2. **Framework:** Java 17, Spring Boot 3.4.8
3. **Persistence:** MySQL (Dockerized for local)
4. **Local Caching:** Caffeine
5. **Distributed Caching:** Redis (Dockerized)
6. **API Layer:** REST APIs (versioned under /api/v1/...)
7. **Container:** Docker and Docker Compose
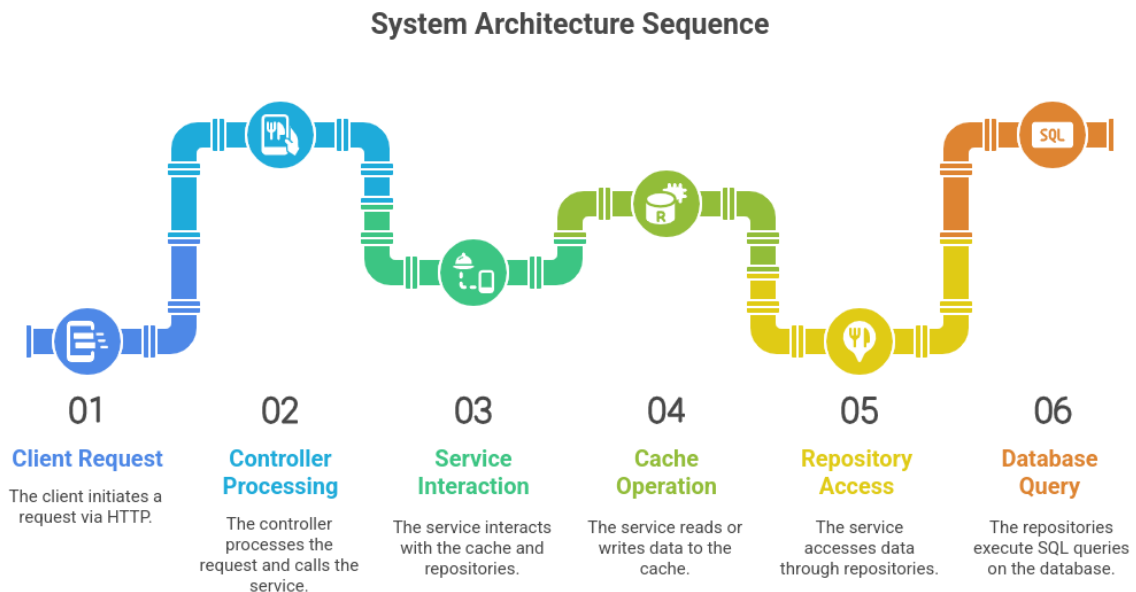
Responsibilities

1. **Spring Boot - Menu-Service:** REST API, business logic, caching, rate limiting
2. **MySQL:** Persistent storage for restaurants and menu items
3. **Redis:** Menu data caching, rate limiting counters

High-Level Architecture

Current Implementation Architecture

System Architecture Sequence

01	02	03	04	05	06
Client Request	**Controller Processing**	**Service Interaction**	**Cache Operation**	**Repository Access**	**Database Query**
The client initiates a request via HTTP.	The controller processes the request and calls the service.	The service interacts with the cache and repositories.	The service reads or writes data to the cache.	The service accesses data through repositories.	The repositories execute SQL queries on the database.

Made with ⧨ Napkin

- Spring Boot Menu Service: Exposes REST APIs, handles business logic, and manages caching and DB access.
- Redis: Distributed cache for menus and menu items.
- MySQL: Persistent storage for restaurants, menus, and menu items.

Component Responsibilities

- Controller Layer: REST API endpoints, request validation
- Service Layer: Business logic, transaction management
- Repository Layer: JPA-based data access
- Mapper Layer: DTO ↔ Entity conversions
- Cache Layer: Redis (future: add Caffeine for application level)

Data Modeling

Database Schema (MySQL)

Restaurant Table

```
CREATE TABLE restaurant (
  id BIGINT NOT NULL AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  address VARCHAR(255),
  city VARCHAR(255),
  pincode VARCHAR(20),
  PRIMARY KEY (id)
);

CREATE INDEX idx_restaurant_city ON restaurant (city);
```

MenuItem Table

```
CREATE TABLE menu (
  id BIGINT NOT NULL AUTO_INCREMENT,
  availability BIT(1) NOT NULL,
  created_on DATETIME(6) DEFAULT NULL,
  dish_name VARCHAR(255) NOT NULL,
  price DOUBLE NOT NULL,
  type ENUM('NON_VEG','VEG') NOT NULL,
  updated_on DATETIME(6) DEFAULT NULL,
  restaurant_id BIGINT DEFAULT NULL,
  PRIMARY KEY (id),
  KEY FK_menu_restaurant (restaurant_id),
  CONSTRAINT FK_menu_restaurant FOREIGN KEY (restaurant_id) REFERENCES restaurant (id)
);

CREATE INDEX idx_menu_dish_name ON menu (dish_name);

CREATE INDEX idx_menu_restaurant_type ON menu (restaurant_id, type);
```

API Data Models

MenuItemRequest

```json
{
  "dishName": "Margherita Pizza",
  "price": 299.99,
  "availability": true,
  "type": "VEG"
}
```

CreateRestaurantRequest

```json
{
  "name": "Pizza Haven",
  "address": "123 Food Street, Koramangala",
  "city": "Bangalore",
  "pincode": "560034",
  "menuItems": [
    {
      "dishName": "Margherita Pizza",
      "price": 299.99,
      "availability": true,
      "type": "VEG"
    },
    {
      "dishName": "Chicken Tikka Pizza",
      "price": 399.99,
      "availability": false,
      "type": "NON_VEG"
    }
  ]
}
```

PaginatedMenuItemResponse (Paginated)

```json
{
  "restaurantId": 1,
  "restaurantName": "Pizza Haven",
  "menuItems": [
    {
```

```
      "id": 1,
      "dishName": "Margherita Pizza",
      "price": 299.99,
      "availability": true,
      "type": "VEG"
    },
    {
      "id": 2,
      "dishName": "Chicken Tikka Pizza",
      "price": 399.99,
      "availability": false,
      "type": "NON_VEG"
    }
  ],
  "currentPage": 0,
  "totalPages": 1,
  "totalElements": 2
}
```

RestaurantMenuResponse

```
{
  "id": 1,
  "name": "Pizza Haven",
  "menuItems": [
    {
      "id": 1,
      "dishName": "Margherita Pizza",
      "price": 299.99,
      "availability": true,
      "type": "VEG"
    },
    {
      "id": 2,
      "dishName": "Chicken Tikka Pizza",
      "price": 399.99,
      "availability": false,
      "type": "NON_VEG"
    }
  ]
}
```

RestaurantResponse

```
{
  "id": 1,
  "name": "Pizza Haven",
  "address": "123 Food Street, Koramangala",
  "city": "Bangalore",
  "pincode": "560034"
```

```
}
```

MenuItemBasicResponse

```
{
  "id": 1,
  "dishName": "Margherita Pizza",
  "price": 299.99,
  "availability": true,
  "type": "VEG"
}
```

RestaurantMenuResponse

```
{
  "dishName": "Margherita Pizza",
  "price": 299.99,
  "availability": true,
  "type": "VEG"
}
```

Caching Strategy

- **L2 Cache (Redis)**: Caches frequently accessed data with configurable TTL. Invalidates on data updates/deletes.
- **L1 Cache (Caffeine, planned)**: For high-frequency data to reduce latency.
- **Cache-Aside Pattern**: Check cache first; on miss, query database, cache result.
- **Invalidation**: Clears cache on data modifications for consistency.

Rate Limiting, Pagination & Versioning

- Rate Limiting: (TO-DO) Redis-based, per-client/IP, configurable limits ensure fair resource usage and prevent abuse.

- Pagination: Supported in menu fetch APIs (page, size params) for efficient data retrieval.
- API Versioning: URL-based (`/api/v1/restaurant/...`) for backward compatibility and smooth updates.

Assumptions and Trade-offs

Assumptions

- Menu changes are infrequent compared to reads.

- Each restaurant has at least one menu.

- Restaurant data changes less frequently than menu items.

- Eventual consistency is acceptable for menu reads.

Trade-offs

- Cache vs. Consistency: Prioritise read performance and accept eventual consistency.
- Performance vs Resource Usage: Caching increases memory usage but

  improves latency.

Monitoring & Observability

- Spring Boot Actuator: Exposes endpoints (health, metrics, info, caches, loggers) for real-time application monitoring and management.

- Logging: Structured logs, correlation IDs, and error stack traces.

- Metrics: Tracks response times, error rates, cache hit/miss ratios, and DB/Redis health

  for performance insights.

- Future: Integrate Prometheus for metrics collection, Grafana for visualization, ELK for log analysis, and distributed tracing for request flow tracking.