**Project Name:** Recipe Management System

**Members:**

1. Chinta Vishnupriya
2. Pavani Arra
3. Prince Panjiyar
4. Udaya R

**Table of Contents**

1. **Introduction:**

This document details the development of a simple Recipe Management System. This system allows users to create, read, update, and delete recipes from a database. It utilizes Java for development and connects to a MySQL database for data persistence.

1. **Features:**

* CRUD Operations: Supports Create, Read, Update, and Delete (CRUD) operations on recipes in a MySQL database.
* Recipe Caching: Implements a cache using a HashMap to store frequently accessed recipes by ID, improving performance.
* User Interaction: Provides a console-based menu for users to manage recipes interactively.
* SQL Integration: Utilizes JDBC to connect and interact with a MySQL database for data persistence.
* Multithreading: Implements multithreading using ExecutorServices to handle concurrent operations, ensuring efficient and scalable management of recipe-related tasks.

1. **Technical Requirements:**

* **Programming Language:** The project is built using Java for its core functionality.
* **Database: Requires** MySQL for storing recipe data. The database connection is managed using JDBC.
* **MySQL JDBC Driver:** Ensure that the MySQL JDBC Connector is included in the classpath to interact with the MySQL database.

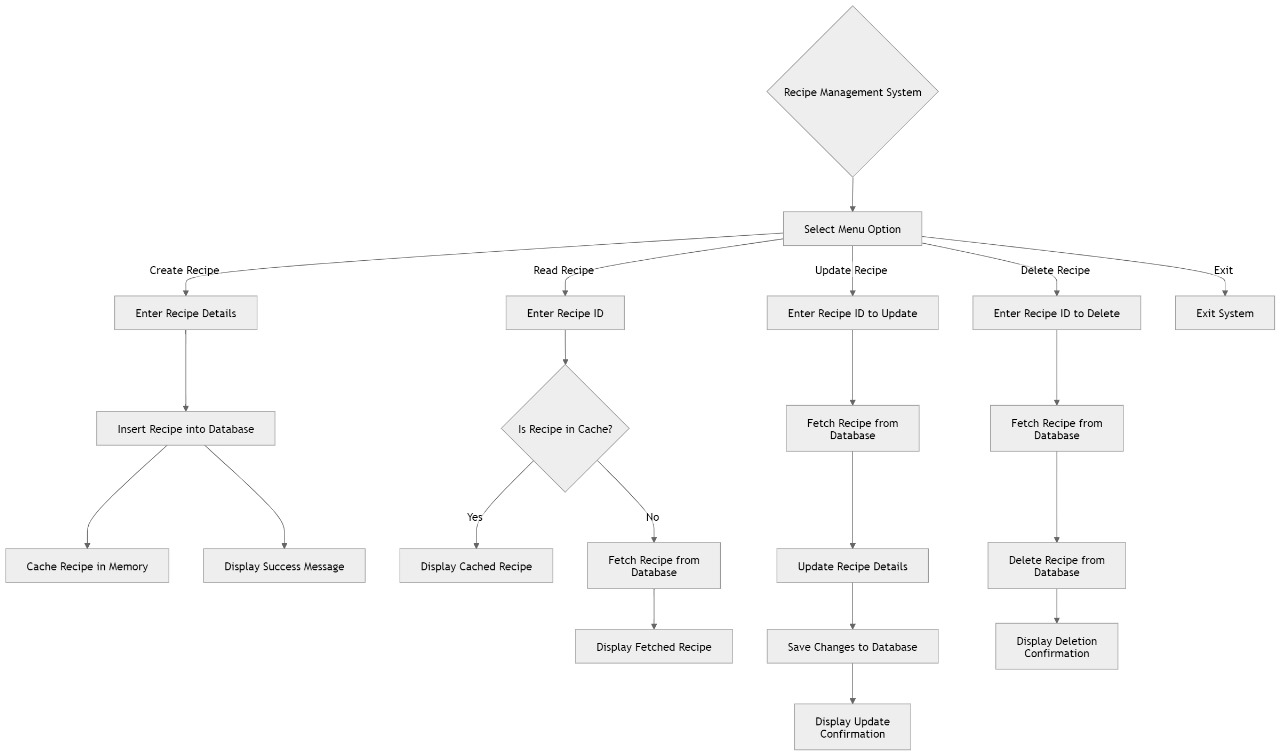
1. **System Architecture:**

This system architecture leverages a simple three-tier architecture with the following components:

* Presentation Layer: Command-line interface to interact with users.
* Business Logic Layer: Core logic for managing recipes, caching, and ensuring efficient operations.
* Data Access Layer: Handles communication with the MySQL database to perform CRUD operations.

The caching mechanism optimizes performance by reducing database queries for frequently accessed recipes.The Executor Service is employed in tasks such as reading from the cache and database concurrently or handling deletion of recipes from both the cache and the database in parallel.

1. **Design and Implementation:**

****

1. **Database Schema:**

| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| --- | --- | --- | --- | --- | --- |
| id | INT | NO | PRIMARY | NULL | auto\_increment |
| name | VARCHAR(255) | NO |  | NULL |  |
| ingredients | TEXT | NO |  | NULL |  |
| instructions | TEXT | NO |  | NULL |  |
| prep\_time | INT | NO |  | NULL |  |

* id: INT, auto-incremented, serves as the primary key.
* name: VARCHAR(255), stores the recipe name, cannot be null.
* ingredients: TEXT, stores the ingredients, cannot be null.
* instructions: TEXT, stores the cooking instructions, cannot be null.
* prep\_time: INT, stores the preparation time in minutes, cannot be null.

1. **Caching Strategy:**

This caching strategy helps reduce the number of database queries by storing frequently accessed recipes in memory. It also limits memory usage by only storing a small number of recipes and evicts older entries using a LRU mechanism.

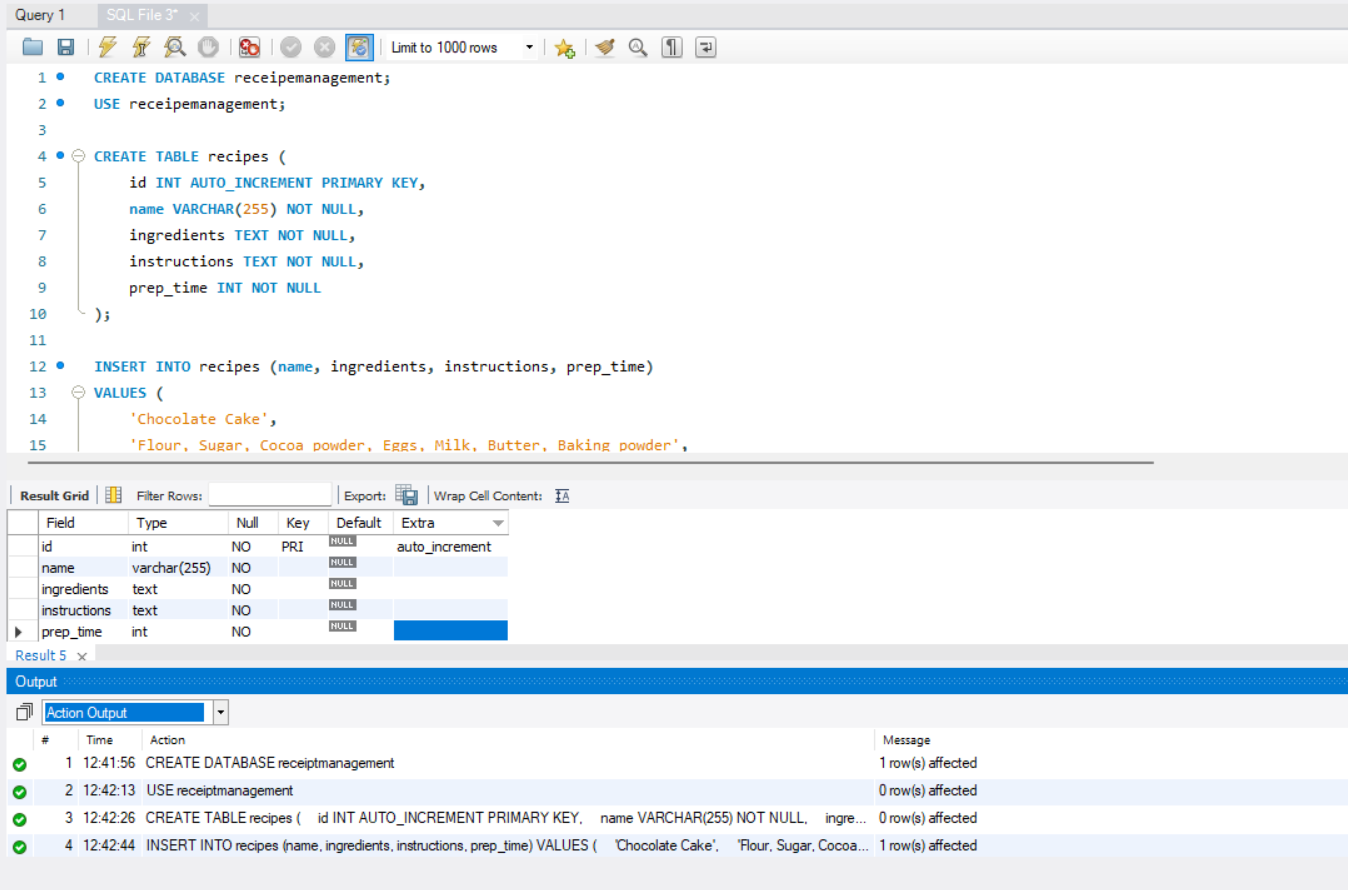
**Eviction Policy :** When the cache reaches its maximum size, the least recently used recipe is evicted.

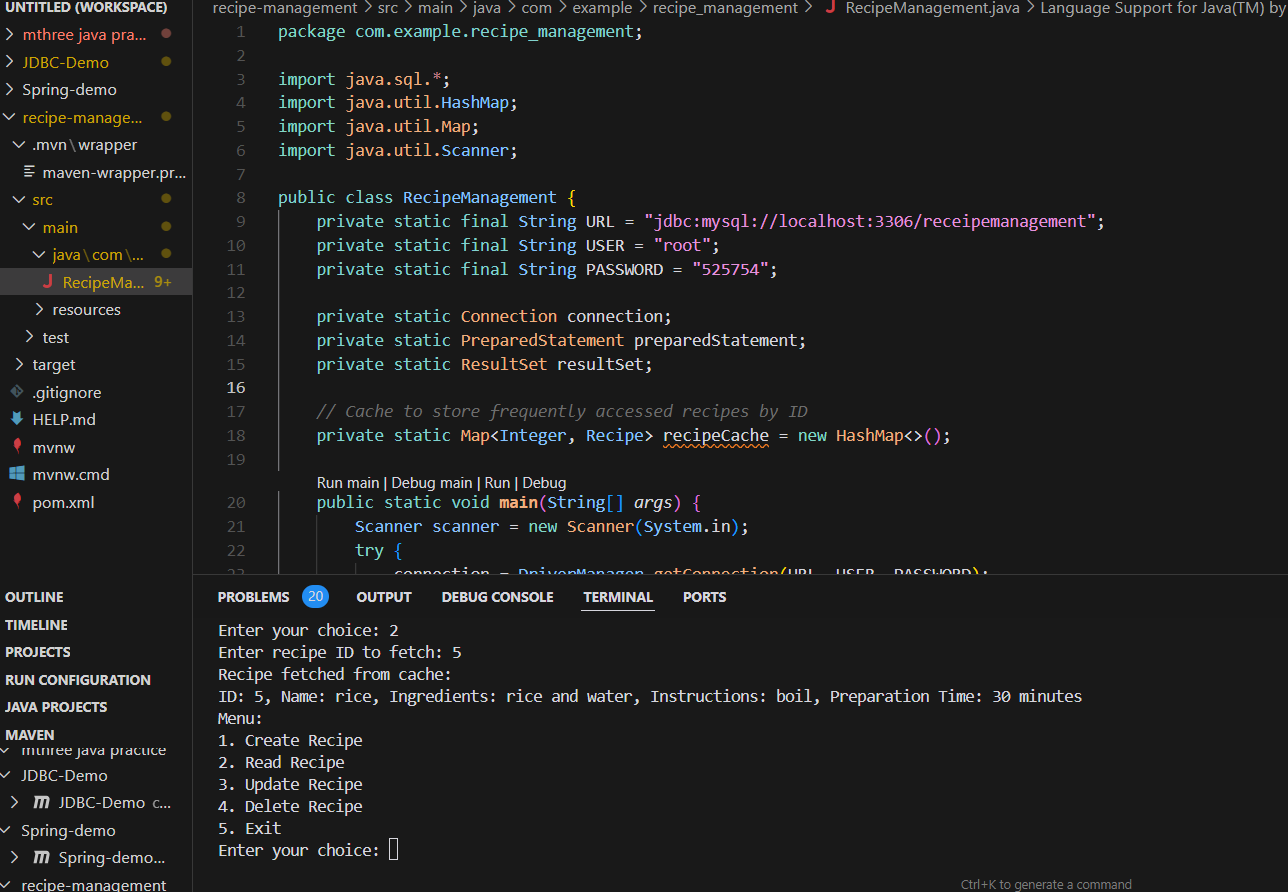
The cache uses the keySet().iterator().next() approach to identify the oldest recipe entry (the first inserted recipe in the map). This oldest entry is then removed to make room for the new entry.

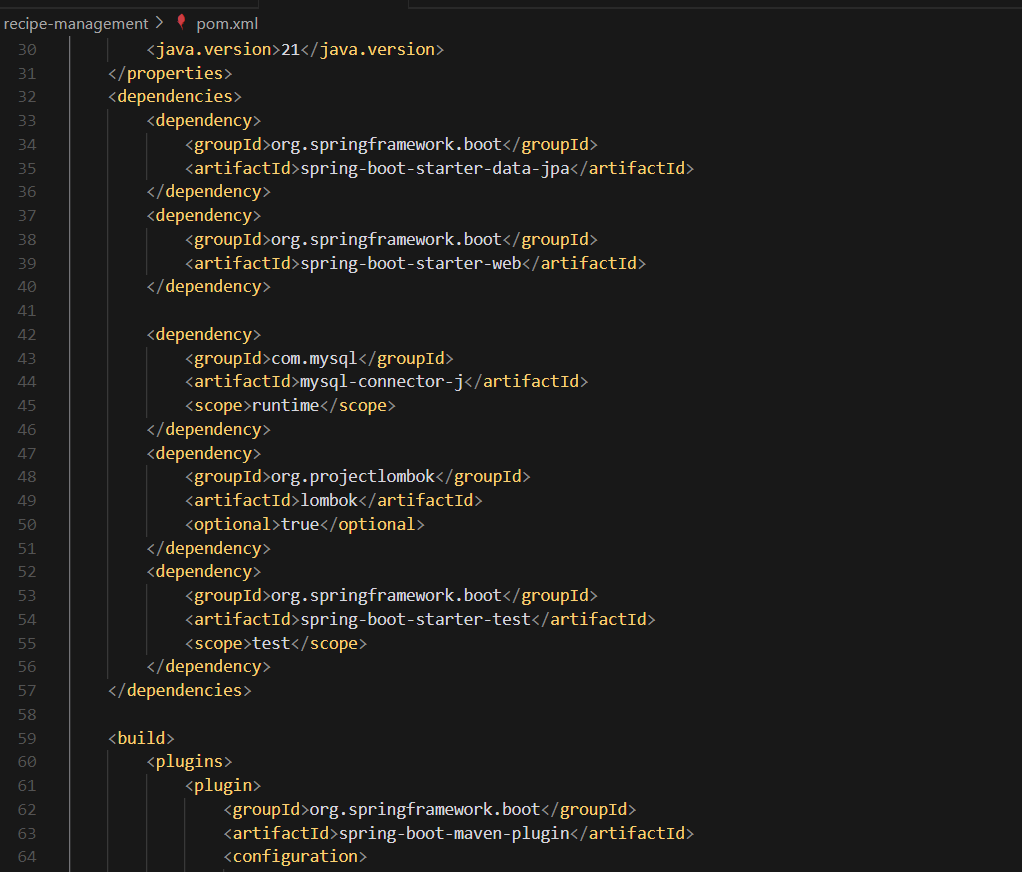
1. **Dependencies**

* Spring Boot Starter Data JPA: Provides integration with Java Persistence API (JPA) for working with relational databases using Spring Data.
* Spring Boot Starter Web: Supports building web applications, including RESTful services using Spring MVC.
* MySQL Connector/J: JDBC driver for connecting to MySQL databases at runtime.
* Spring Boot Starter Test:Provides testing support, including libraries like JUnit, Mockito, and Spring TestContext Framework for unit and integration testing.

1. **Implementation and results:**

****

****

****