No worries! I'll guide you step-by-step on how to approach the "Kirana Register" backend service problem and help you with the code. Let's break it down in a structured way:

**1. Setting Up Your Development Environment**

* **Step 1**: **Install Java** – Make sure Java is installed. You can download the JDK from [here](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html) if you haven't already.
* **Step 2**: **Install Maven** – Spring Boot uses Maven to manage dependencies. You can get Maven [here](https://maven.apache.org/download.cgi).
* **Step 3**: **Install MongoDB** – Since you will use MongoDB as the database, install it from [here](https://www.mongodb.com/try/download/community).
* **Step 4**: **Set Up Spring Boot** – You can use [Spring Initializr](https://start.spring.io/) to quickly bootstrap your Spring Boot project.
  + Select **Java** as the language.
  + **Spring Boot Version**: Choose the latest stable version.
  + **Dependencies**: Add the following:
    - Spring Web
    - Spring Data MongoDB
    - Spring Security (for authentication/authorization)
    - Lombok (for reducing boilerplate code)
    - DevTools (for development convenience)

**2. Creating the Spring Boot Application**

* **Step 1**: Open the project in an IDE (like IntelliJ IDEA or Eclipse).
* **Step 2**: Spring Boot uses an application.properties or application.yml file to configure the database connection.

Add the MongoDB configuration in src/main/resources/application.properties:

properties

Copy code

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=kirana\_db

**3. Define the Models (Entities)**

You need to define the transaction model (how data will be stored in MongoDB).

Create a class Transaction.java inside the model package:

java

Copy code

import lombok.Data;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import java.util.Date;

@Data

@Document(collection = "transactions")

public class Transaction {

@Id

private String id;

private double amount;

private String currency;

private String transactionType; // credit or debit

private Date date;

}

**4. Create the Repository**

MongoDB repositories make it easy to interact with the database.

Create an interface TransactionRepository.java inside the repository package:

java

Copy code

import org.springframework.data.mongodb.repository.MongoRepository;

import java.util.List;

public interface TransactionRepository extends MongoRepository<Transaction, String> {

List<Transaction> findByTransactionType(String transactionType);

}

**5. Create the Service Layer**

This layer will handle business logic like saving transactions and converting currencies.

Create a service class TransactionService.java inside the service package:

java

Copy code

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class TransactionService {

@Autowired

private TransactionRepository transactionRepository;

public Transaction saveTransaction(Transaction transaction) {

return transactionRepository.save(transaction);

}

public List<Transaction> getTransactionsByType(String type) {

return transactionRepository.findByTransactionType(type);

}

}

**6. Create the Controller Layer**

This layer will expose the API endpoints for users to interact with the system.

Create a controller class TransactionController.java inside the controller package:

java

Copy code

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/transactions")

public class TransactionController {

@Autowired

private TransactionService transactionService;

@PostMapping

public Transaction recordTransaction(@RequestBody Transaction transaction) {

return transactionService.saveTransaction(transaction);

}

@GetMapping("/type/{type}")

public List<Transaction> getTransactionsByType(@PathVariable String type) {

return transactionService.getTransactionsByType(type);

}

}

**7. Adding User Authentication and Authorization**

You need to secure the APIs with Spring Security.

**Step 1**: Define user roles (read-only, read-write) in a SecurityConfig.java file:

java

Copy code

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

@Configuration

public class SecurityConfig {

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http.authorizeRequests()

.antMatchers("/api/transactions/\*\*").hasRole("USER")

.anyRequest().authenticated()

.and()

.httpBasic();

return http.build();

}

@Bean

public InMemoryUserDetailsManager userDetailsService() {

UserDetails user = User.withDefaultPasswordEncoder()

.username("user")

.password("password")

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

}

**8. Handling Currency Conversion**

To handle real-time currency conversion, make a call to the FX Rates API and cache the results.

**Step 1**: Create a CurrencyService.java to call the FX Rates API:

java

Copy code

import org.springframework.stereotype.Service;

import org.springframework.web.client.RestTemplate;

@Service

public class CurrencyService {

private final String API\_URL = "https://api.fxratesapi.com/latest";

public double convertCurrency(String from, String to, double amount) {

RestTemplate restTemplate = new RestTemplate();

String url = API\_URL + "?base=" + from + "&symbols=" + to;

CurrencyResponse response = restTemplate.getForObject(url, CurrencyResponse.class);

return amount \* response.getRates().get(to);

}

}

**Step 2**: Define a CurrencyResponse.java class to handle API response:

java

Copy code

import java.util.Map;

public class CurrencyResponse {

private String base;

private Map<String, Double> rates;

public Map<String, Double> getRates() {

return rates;

}

public void setRates(Map<String, Double> rates) {

this.rates = rates;

}

}

**9. Running the Application**

* Open your terminal/command prompt.
* Navigate to the root of your Spring Boot project and run:

bash

Copy code

./mvnw spring-boot:run

* The server will start at http://localhost:8080. You can test your APIs using Postman.

**10. API Rate Limiting**

You can use a library like Bucket4j to limit the number of requests per minute.

**Step 1**: Add the dependency to pom.xml:

xml

Copy code

<dependency>

<groupId>com.github.vladimir-bukhtoyarov</groupId>

<artifactId>bucket4j-core</artifactId>

<version>7.3.0</version>

</dependency>

**Step 2**: Implement rate limiting logic inside the controller (you can set limits like 10 requests per minute).