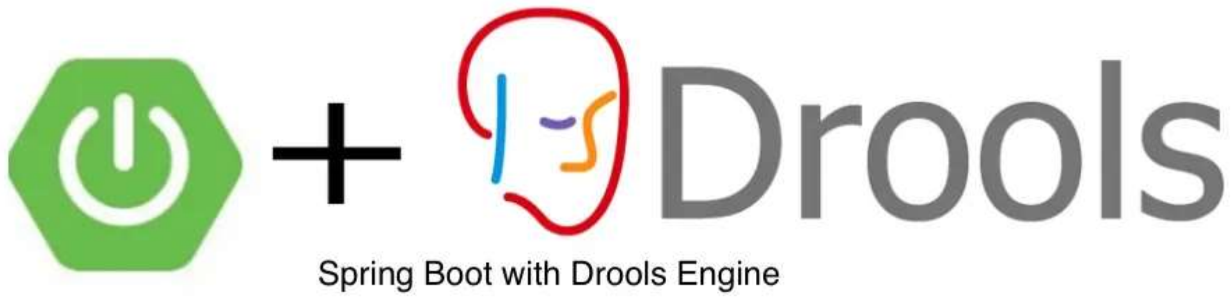
Spring Boot with Drools Rules Engine

Tobin Tom · Follow
5 min read · Jul 3

35



SpringBoot with Drools

A rules engine is used to serve as pluggable software components which execute business rules that a business rules approach has externalized or separated from application code. This externalization or separation allows business users to modify the rules without the need for IT intervention. In cases of large distributed applications that process complex logic, a rules engine can be used to isolate the application code from common rules logic that can be applied commonly across the distributed platform.

In this article we will be building a sample Spring Boot REST service that uses the **Drools Rule Engine** to determine the outcome of various inputs.

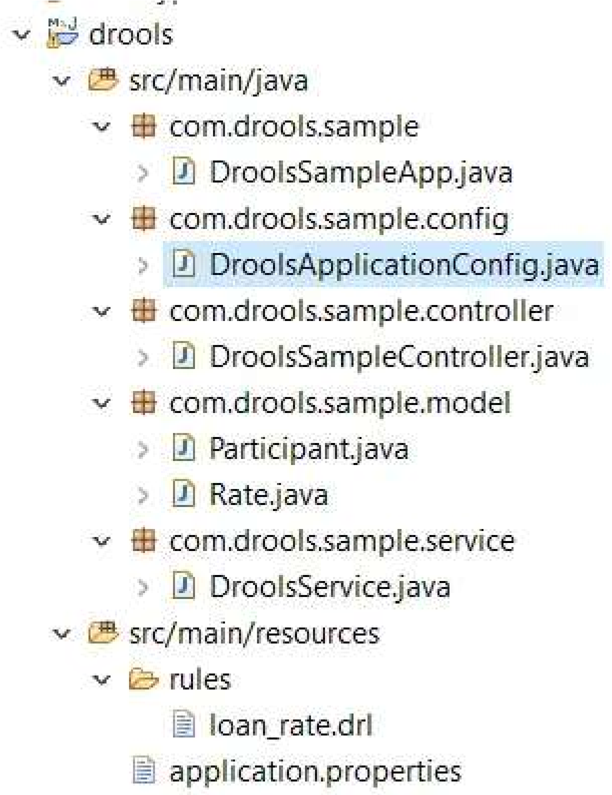
The Drools Rules Engine (https://www.drools.org/) is an open source rules engine implementation from JBoss.

The KIE project (https://www.kie.org/about/) allows integration of the Drools Rules engine with Spring Boot.

Application Setup

We will create a simple REST service application using SpringBoot which will use a Drools Engine to determine output of the service.

We will have a Banking Service that takes an applicant's details and returns a loan rate of interest.



```
drools
  src/main/java
    com.drools.sample
      DroolsSampleApp.java
    com.drools.sample.config
      DroolsApplicationConfig.java
    com.drools.sample.controller
      DroolsSampleController.java
    com.drools.sample.model
      Participant.java
      Rate.java
    com.drools.sample.service
      DroolsService.java
  src/main/resources
    rules
      loan_rate.drl
    application.properties
```

Application Structure

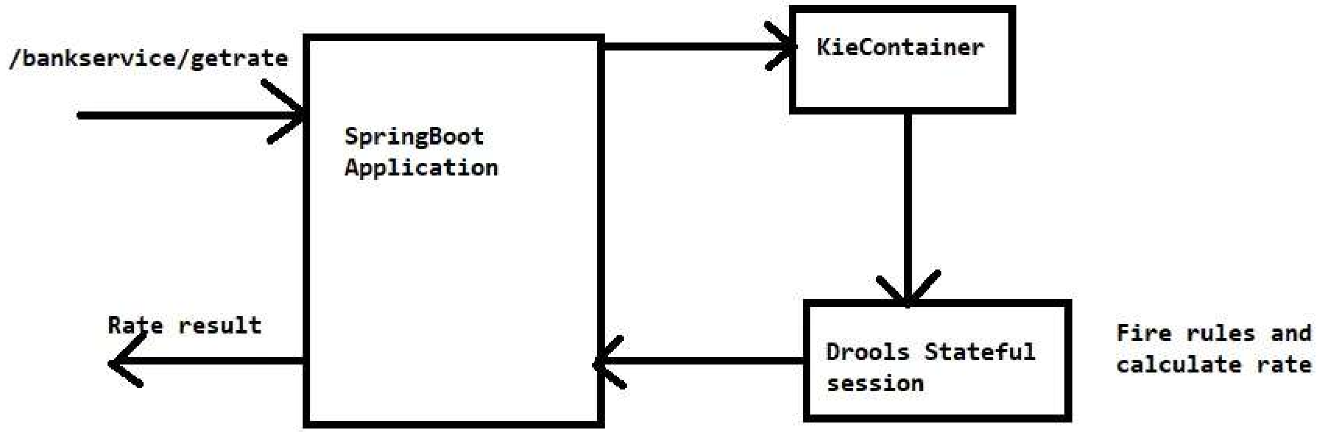
Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.7</version>
  </parent>
  <groupId>com.drools.example</groupId>
  <artifactId>drools</artifactId>
  <version>0.0.1-SNAPSHOT</version>
```

```
      <name>droolsSample</name>
      <description>DroolsSample</description>
      <properties>
            <java.version>11</java.version>
            <maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>
            <drools.version>8.40.0.Final</drools.version>
        </properties>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
            </dependency>

            <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <optional>true</optional>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
            </dependency>

            <dependency>
                <groupId>org.drools</groupId>
                <artifactId>drools-core</artifactId>
                <version>${drools.version}</version>
            </dependency>
            <dependency>
                <groupId>org.drools</groupId>
                <artifactId>drools-compiler</artifactId>
                <version>${drools.version}</version>
            </dependency>
            <dependency>
                <groupId>org.drools</groupId>
                <artifactId>drools-decisiontables</artifactId>
                <version>${drools.version}</version>
            </dependency>
            <dependency>
```

```
                <groupId>org.drools</groupId>
                <artifactId>drools-mvel</artifactId>
                <version>${drools.version}</version>
            </dependency>
        </dependencies>
        <build>
      <plugins>
       <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
       </plugin>
      </plugins>
     </build>
    </project>
```

In the pom.xml, we will need to define the drools dependencies, the drools-core, drools-compiler, drools-mvel and decisiontables.



The service accepts a request payload and invokes the Drools Rules engine to fire all configured rules and returns a response.

DroolsSampleApp.java

```java
package com.drools.sample;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DroolsSampleApp {
 public static void main(String[] args) {
  SpringApplication.run(DroolsSampleApp.class, args);
 }
}
```

DroolsApplicationConfig.java

```java
package com.drools.sample.config;

import org.kie.api.KieServices;
import org.kie.api.builder.KieBuilder;
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieModule;
import org.kie.api.runtime.KieContainer;
import org.kie.internal.io.ResourceFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
public class DroolsApplicationConfig {

 private static final KieServices kieServices = KieServices.Factory.get();
 private static final String RULES_CUSTOMER_RULES_DRL = "rules/loan_rate.drl";

   @Bean
     public KieContainer kieContainer() {
         KieFileSystem kieFileSystem = kieServices.newKieFileSystem();
         kieFileSystem.write(ResourceFactory.newClassPathResource(RULES_CUSTOMER
         KieBuilder kb = kieServices.newKieBuilder(kieFileSystem);
         kb.buildAll();
         KieModule kieModule = kb.getKieModule();
         KieContainer kieContainer = kieServices.newKieContainer(kieModule.getRe
         return kieContainer;
     }
}
```

Here we define the rules being placed in the **/rules** folder under /src/main/resources and we have defined a rule file **loan_rate.drl**. The actual Drools Rules are defined in files with the drl extension.

loan_rate.drl

```
import com.drools.sample.model.Participant;
global com.drools.sample.model.Rate rate;

dialect "mvel"
```

```
rule "Checking Existing Debt Against Loan Amount"
 when
  Participant(loanAmount > (2 * existingDebt))
 then
  rate.setLoanStatus("Rejected - Too Much Debt");
  rate.setLoanRate(999);
end

rule "Checking Annual Salary is greater than 50000"
 when
  Participant(annualSalary <= 50000)
 then
  rate.setLoanStatus("Rejected - Too low salary");
  rate.setLoanRate(999);
end

rule "Checking Credit Score less than 550"
 when
  Participant(creditScore < 550 , annualSalary > 50000,  loanAmount < (2 * exist
 then
  rate.setLoanStatus("Rejected");
  rate.setLoanRate(999);
end

rule "Checking Credit Score less than 650"
 when
  Participant((creditScore < 650 && creditScore >= 550), annualSalary > 50000,
 then
  rate.setLoanStatus("Approved");
  rate.setLoanRate(7.25);
end

rule "Checking Credit Score less than 750"
 when
  Participant((creditScore < 750 && creditScore >= 650), annualSalary > 50000,
 then
  rate.setLoanStatus("Approved");
  rate.setLoanRate(6.25);
end
```

```
rule "Checking Credit Score greater than 750"
 when
  Participant(creditScore >= 750, annualSalary > 50000,  loanAmount < (2 * exist
 then
  rate.setLoanStatus("Approved");
  rate.setLoanRate(5.25);
end
```

Here we define the rules of execution:

1. If the user is asking for a loan amount > (twice the existing debt), the loan process is rejected.
2. If the user has an annual salary of 50000 or less, the loan process is rejected.
3. If the user satisfies the above conditions, then the loan rate is decided by the credit score.

MODEL REQUESTS:

PARTICIPANT REQUEST OBJECT:

```
package com.drools.sample.model;

import lombok.Getter;
```

```java
import lombok.Setter;

@Getter
@Setter
public class Participant {

 private String name;
 private int age;
 private int creditScore;
 private long annualSalary;
 private long existingDebt;
 private long loanAmount;
}
```

RATE RESPONSE OBJECT:

```java
package com.drools.sample.model;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class Rate {

 private String loanStatus;
 private double loanRate;
}
```

DROOLS SERVICE:

```
package com.drools.sample.service;

import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.drools.sample.model.Participant;
import com.drools.sample.model.Rate;

@Service
public class DroolsService {

 @Autowired
    private KieContainer kieContainer;

  public Rate getRate(Participant applicantRequest) {
    Rate rate = new Rate();
         KieSession kieSession = kieContainer.newKieSession();
         kieSession.setGlobal("rate", rate);
         kieSession.insert(applicantRequest);
         kieSession.fireAllRules();
         kieSession.dispose();
         return rate;
     }

}
```

DROOLS CONTROLLER:

```
package com.drools.sample.controller;
```

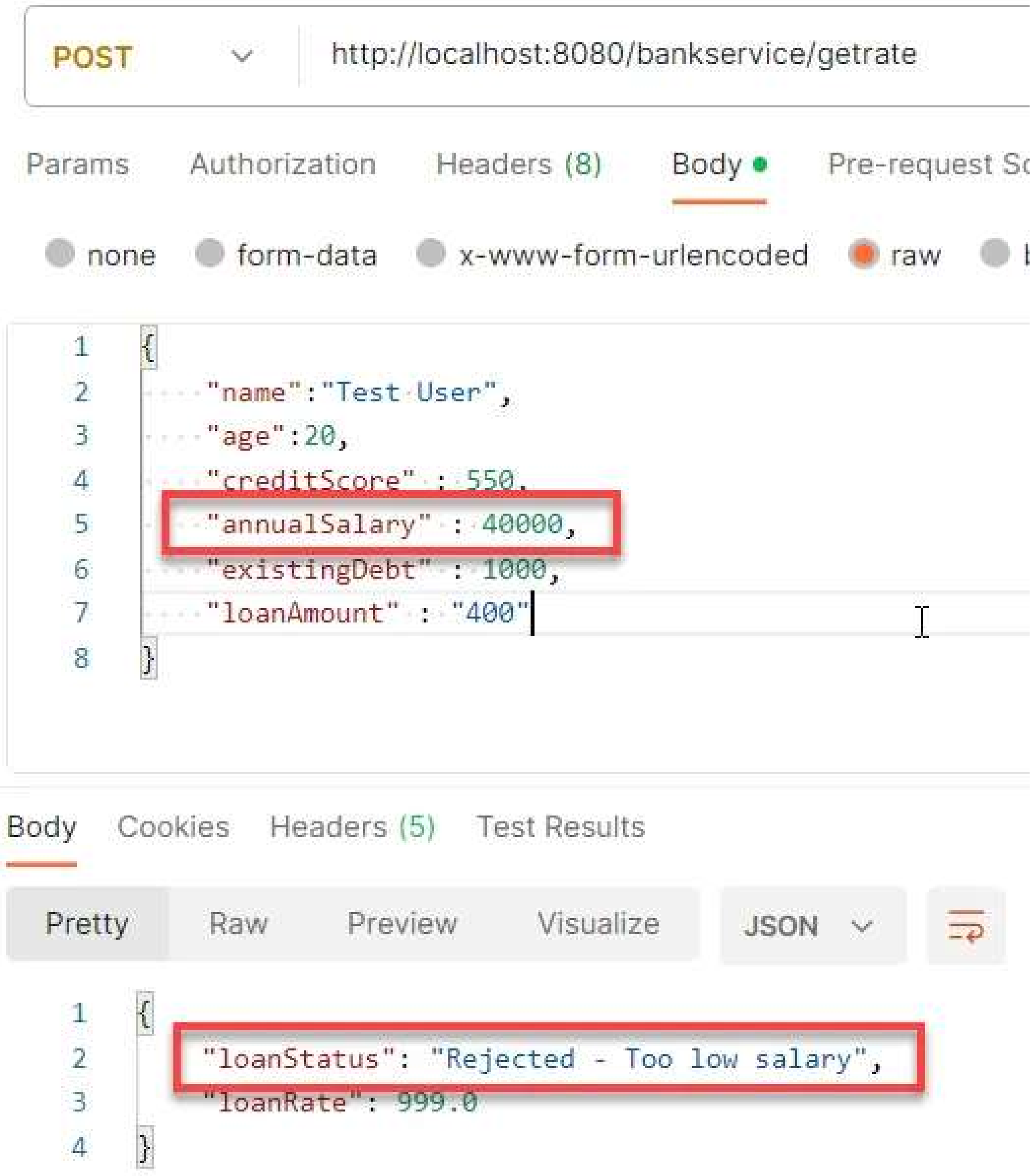
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.drools.sample.model.Participant;
import com.drools.sample.model.Rate;
import com.drools.sample.service.DroolsService;

@RestController()
@RequestMapping("/bankservice")
public class DroolsSampleController {

 @Autowired
 private DroolsService bankService;

 @PostMapping("/getrate")
 public ResponseEntity<Rate> getRate(@RequestBody Participant request){
  Rate rate = bankService.getRate(request);
  return new ResponseEntity<>(rate, HttpStatus.OK);
 }

}
```

The controller takes in a Participant.java object and fires the rules configured in the loan_rate.drl and returns a Rate.java object as response.

Testing the Service

Now we can build the Spring Boot application and run tests against it at: *http://localhost:8080/bankservice/getrate*

1. Testing with Salary less than 50000:

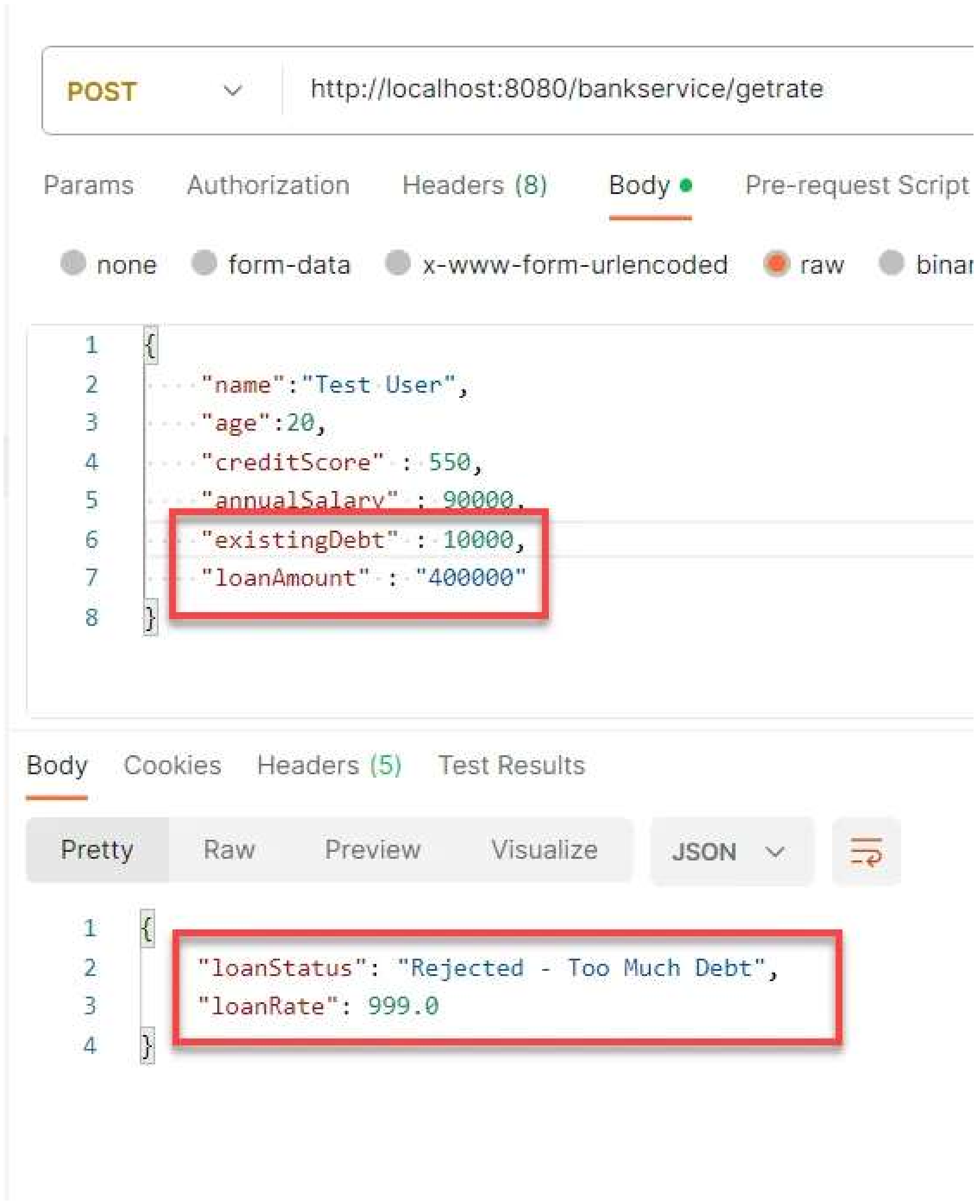


POST http://localhost:8080/bankservice/getrate

```
{
    "name":"Test User",
    "age":20,
    "creditScore" : 550,
    "annualSalary" : 40000,
    "existingDebt" : 1000,
    "loanAmount" : "400"
}
```

```
{
    "loanStatus": "Rejected - Too low salary",
    "loanRate": 999.0
}
```

2. Testing with High salary, but too much debt:

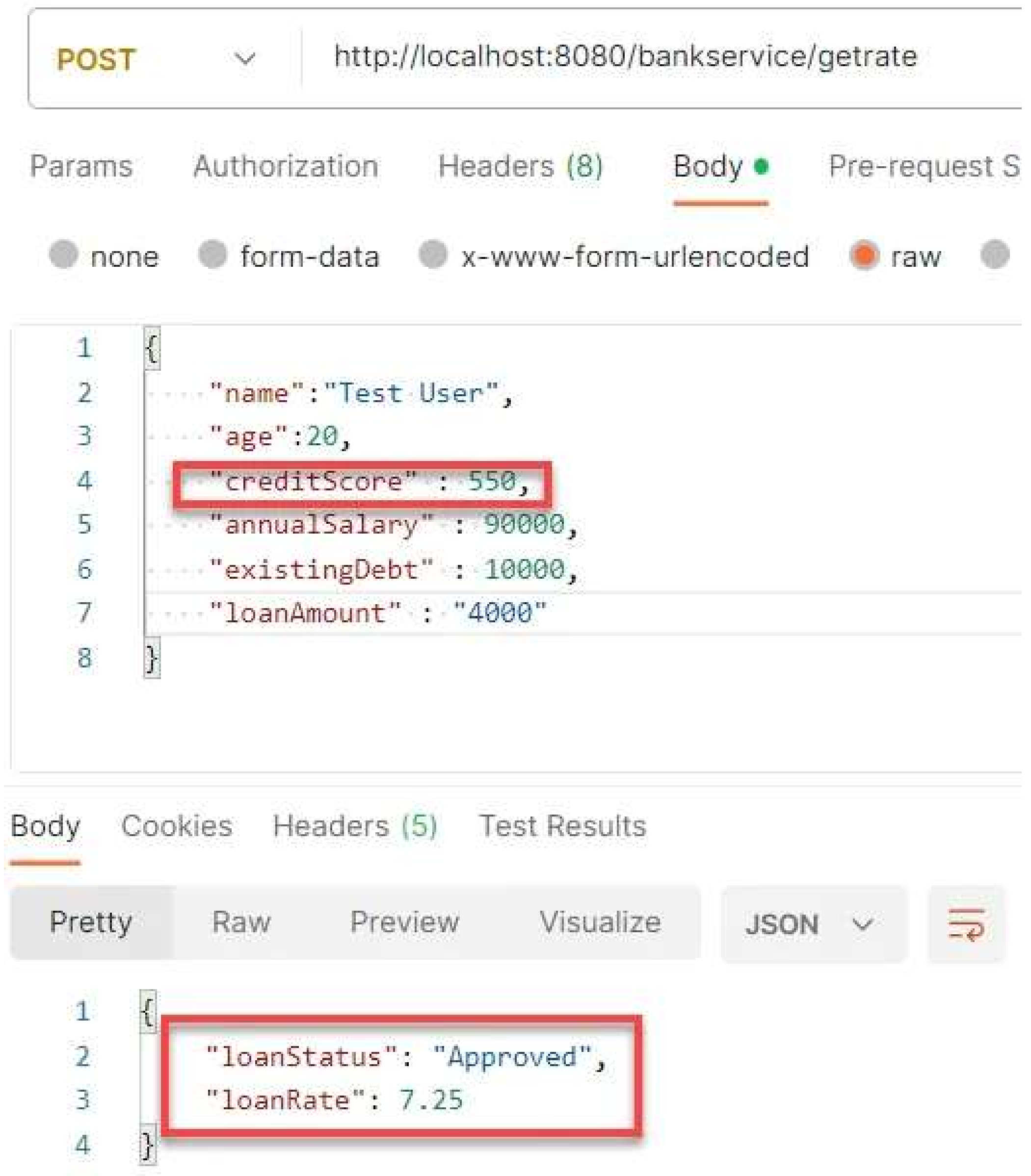


http://localhost:8080/bankservice/getrate

```
{
    "name":"Test User",
    "age":20,
    "creditScore" : 550,
    "annualSalary" : 90000,
    "existingDebt" : 10000,
    "loanAmount" : "400000"
}
```

```
{
    "loanStatus": "Rejected - Too Much Debt",
    "loanRate": 999.0
}
```

3. Testing with Credit Scores:

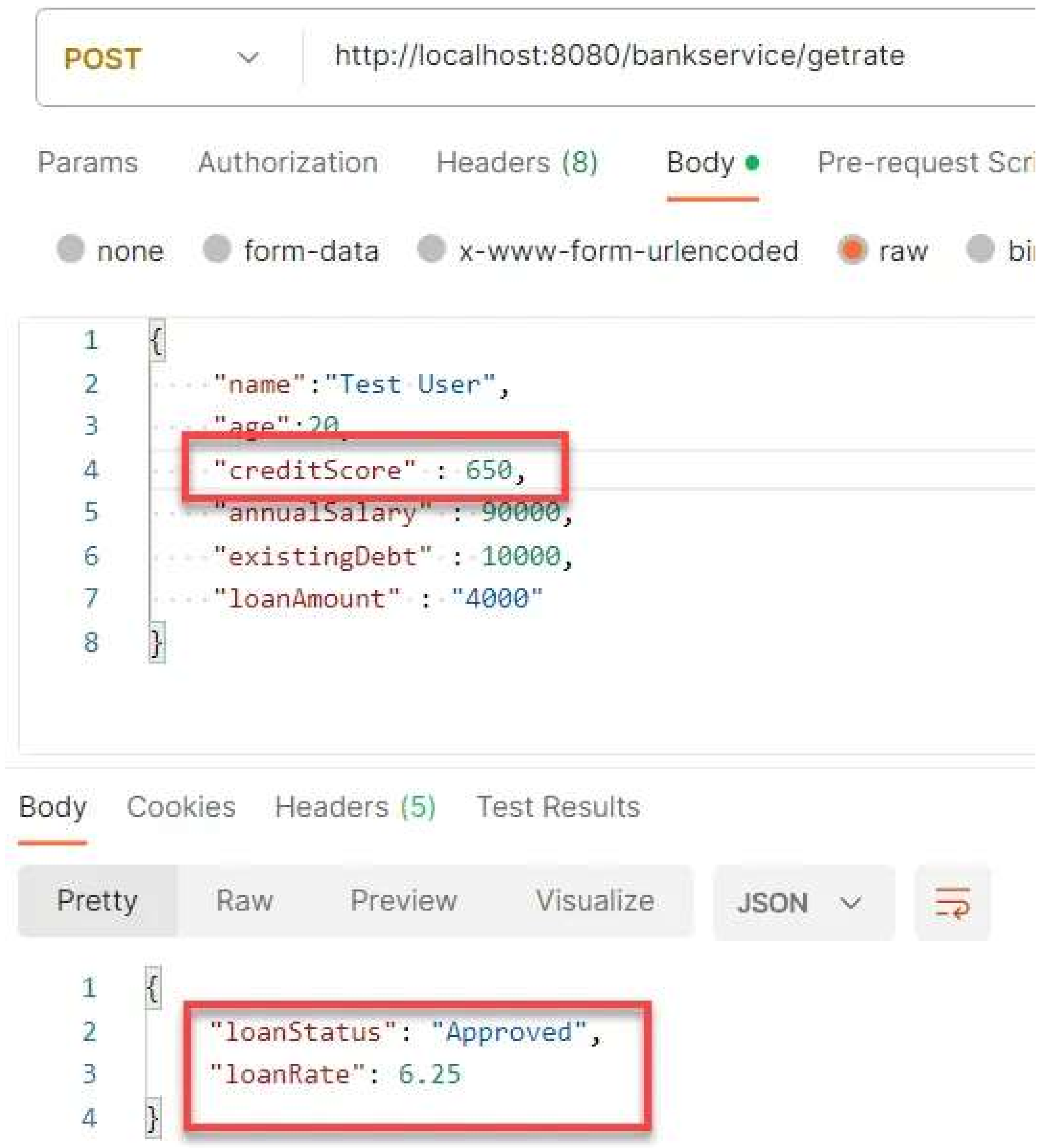


http://localhost:8080/bankservice/getrate

```
{
    "name":"Test User",
    "age":20,
    "creditScore" : 550,
    "annualSalary" : 90000,
    "existingDebt" : 10000,
    "loanAmount" : "4000"
}
```

```
{
    "loanStatus": "Approved",
    "loanRate": 7.25
}
```

Credit Score less than 650

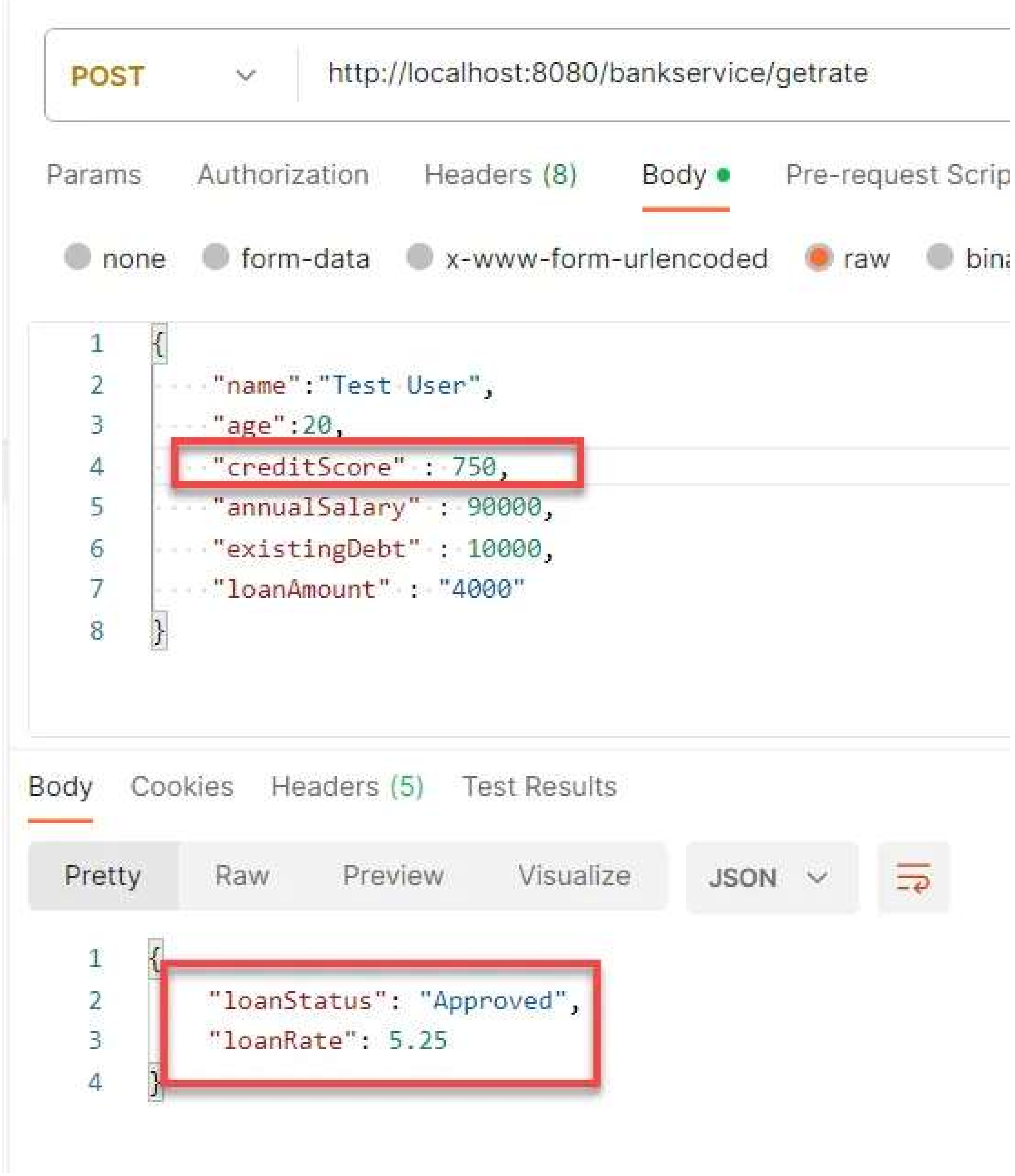


```
http://localhost:8080/bankservice/getrate
```

```
{
    "name":"Test User",
    "age":20,
    "creditScore" : 650,
    "annualSalary" : 90000,
    "existingDebt" : 10000,
    "loanAmount" : "4000"
}
```

```
{
    "loanStatus": "Approved",
    "loanRate": 6.25
}
```

Credit Score <750 and ≥ 650



```
http://localhost:8080/bankservice/getrate
```

```
{
    "name":"Test User",
    "age":20,
    "creditScore" : 750,
    "annualSalary" : 90000,
    "existingDebt" : 10000,
    "loanAmount" : "4000"
}
```

```
{
    "loanStatus": "Approved",
    "loanRate": 5.25
}
```

Credit score greater than or equal to 750

Spring Boot Drools Rules Engine Rest Api Spring

Written by Tobin Tom

Follow

72 Followers

I am a full stack J2EE developer with over 15 years of development experience in various technologies and platforms.

More from Tobin Tom

Tobin Tom

Tobin Tom in DevOps.dev

Introducing Kafka Streams with Spring Boot

Apache Kafka (https://kafka.apache.org/documentation/) i...

10 min read · Nov 29, 2022

64 2

Spring Cloud Gateway OAuth2 Security with Keycloak, JWT...

In this article we will refer to my previous article on building a microservices...

8 min read · Nov 18, 2022

13

Tobin Tom

Documenting OAuth2 secured Spring Boot Microservices with...

When we build APIs, especially when it is a microservices application, it becomes...

8 min read · Nov 19, 2022

11

Tobin Tom in DevOps.dev

Spring Boot Microservices with Consul, Spring Cloud Gateway an...

Microservices are now the norm when building service applications and Spring Bo...

9 min read · Oct 19, 2022

63

See all from Tobin Tom