JAVA TEST

Alberto Gárate Gútierrez

GAMING INNOVATION GROUP St. Julian's, Malta

29/01/2024

Java Test

Design and implement a RESTful API (including the data model and the backingimplementation) for money transfers between accounts.

Explicit requirements:

- Keep it simple and to the point (e.g. no need to implement any authentication,assume the API is invoked by another internal system/service)
- Please use Java as a programming language and Maven for dependency injection, butdon't forget about requirement #1
- The final result should be executable as a standalone program (should not require a pre-installed container/server) and uses Docker
- Demonstrate with tests that the API works as expected and can automatically be run by a bash script
    - Build the application using Maven plugins
    - Maven will produce a FAT-Jar
    - Trigger the application by using *java -jar xxx.jar*
    - Use curl to execute requests to the API
    - Printing results in the console is enough to demonstrate that the applicationworks as expected
- Please include a README in any format about decisions you made along the way,what you focused on, what you didn't focus on and why

Implicit requirements:

- The code produced by you is expected to be of high quality
- There are no detailed requirements, use common sense
- Use of Spring Data and storing data in PGSQL or MySQL
- Use of BitBucket, GitLab or GitHub

1) DEFINE THE DATA MODEL

```
import lombok.Data;
import javax.persistence.Entity;
import javax.persistence.Id;
import java.math.BigDecimal;

@Data
@Entity
public class Account {
    @Id
    private Long id;
    private BigDecimal balance;
}
```

- Defined an Account class with the @Entity annotation from JPA to represent the entity in the database.

- Lombok is used to reduce code verbosity.

2) IMPLEMENT THE TRANSFER SERVICE

```
import org.springframework.stereotype.Service;
import java.math.BigDecimal;

@Service
public class TransferService {
    public void transferMoney(Long sourceAccountId, Long targetAccountId,
BigDecimal amount) {

    }
}
```

- Created a TransferService class annotated with @Service that handles transfer operations between accounts.

3) IMPLEMENT THE REST API

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

```
@RestController
@RequestMapping("/api/transferencias")
public class TransferController {
    @Autowired
    private TransferService transferService;

    @PostMapping
    public ResponseEntity<String> transferMoney(@RequestBody
TransferRequest request) {
        transferService.transferMoney(request.getSourceAccountId(),
request.getTargetAccountId(), request.getAmount());
        return ResponseEntity.ok("Transferencia exitosa");
    }
}
```

- Implemented a controller (TransferController) with Spring annotations to handle REST requests related to money transfers.

- Used TransferRequest to represent the transfer request.

4) CONFIGURE MAVEN

```
    <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.yourcompany</groupId>
    <artifactId>your-application</artifactId>
    <version>1.0.0</version>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <!-- Spring Boot dependencies -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- Other dependencies as needed -->
    </dependencies>
```

```
    <build>
        <plugins>
            <!-- Plugin to build a Fat-Jar -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>

            <!-- Docker plugin -->
            <plugin>
                <groupId>com.spotify</groupId>
                <artifactId>dockerfile-maven-plugin</artifactId>
                <version>1.4.13</version> <!-- Version may vary -->
                <executions>
                    <execution>
                        <id>default</id>
                        <goals>
                            <goal>build</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <imageName>your-docker-image-name</imageName>
                    <!-- Other configurations as needed -->
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

- Configured the POM.xml file for Maven, including Spring Boot dependencies and plugins to package a Fat-Jar.

5) CONFIGURE DOCKER

```
FROM openjdk:11-jre-slim AS builder

WORKDIR /app
COPY pom.xml .
COPY src src
RUN mvn package

FROM openjdk:11-jre-slim
COPY --from=builder /app/target/your-application.jar /app.jar
```

```
EXPOSE 8080
CMD ["java", "-jar", "/app.jar"]
```

- Configured a Dockerfile to build and run the application in a Docker container.

6) SPRING DATA AND PostgreSQL

```
spring.datasource.url=jdbc:postgresql://localhost:5432/your-database-name
spring.datasource.username=your-username
spring.datasource.password=your-password
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL
ialect
```

- Configured properties to connect the application to a PostgreSQL database

7) IMPLEMENT STORAGE LOGIC (IMPLICIT)

```
    //1) Spring Data Configuration:

    java
    Copy code
    @Entity
    public class Account {
        // Attributes and getter/setter methods
    }

    public interface AccountRepository extends JpaRepository<Account,
Long> {
        // Custom methods if needed
    }

        //2) Database Configuration:
    properties
    Copy code

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

        //3) # PostgreSQL Configuration
    spring.datasource.url=jdbc:postgresql://localhost:5432/your-database-
name
```

```
    spring.datasource.username=your-username
    spring.datasource.password=your-password
    spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Postgre
SQLDialect
```

- Mentioned Spring Data configuration, including the @Entity annotation in the entity class and the creation of a repository.

8) GIT

```
    Initialize the Git repository:
    bash
    Copy code
    git init
    git add .
    git commit -m "Initial commit with the base structure"

    Create a repository on GitHub:

    bash
    Copy code
    git remote add origin https://github.com/your-username/your-repo.git

    git push -u origin master
```

- Initialized a Git repository and created a repository on GitHub