Below is a complete example of how to combine Java Spring Boot with GraphDB to execute SPARQL queries. You need to change some parts such as the GraphDB endpoint URL and SPARQL query result handling according to the needs of your specific application.

Step 1: Add Dependency to pom.xml

xml
Copy code

```
<dependencies>
    <!-- ... other dependencies ...                    -->
    <dependency>
        <groupId>com.ontotext.graphdb</groupId>
        <artifactId>graphdb-free-runtime</artifactId>
        <version>INSERT_VERSION_HERE< /version>
    </dependency>
</dependencies>
```

Step 2: Configure Connection to GraphDB in application.properties or application.yml

properties
Copy code

```
# application.properties

graphdb.url=http://localhost:7200/repositories/your-repository-name Replace your-
repository-name with the name of the repository you are using.
```

Step 3: Create Configuration Class For GraphDB

java
Copy code

```
import com.ontotext.graphdb.repository.GraphDBRepository; import
org.springframework.beans.factory.annotation.Value; import
org.springframework.context.annotation.Bean; import
org.springframework.context.annotation.Configuration;

@Configuration
public class GraphDBConfig {

    @Value("${graphdb.url}")
    private String graphDBURl;

    @Bean
    public GraphDBRepository graphDBRepository() { return
        new GraphDBRepository(graphDBURl);
    }
}
```

In the code above:

@Configuration marks the class as a configuration class.

@Value("${graphdb.url}") is used to annotate the value of the property from the configuration file (application.properties).
@Bean is used to create a GraphDBRepository bean to manage connection and interaction with GraphDB.

Step 4: Create Service Class for RDF Data Processing and SPARQL Query

java
Copy code

```
import com.ontotext.graphdb.repository.GraphDBRepository; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Service;

@Service
public class RDFService {

    private final GraphDBRepository graphDBRepository;

    @Autowired
    public RDFService(GraphDBRepository graphDBRepository)
        { this.graphDBRepository = graphDBRepository;
    }

    public String executeSparqlQuery(String query) { // Use
        GraphDBRepository to execute SPARQL query return
        graphDBRepository.executeSparqlQuery(query);
    }
}
```

In the code above:

@Service marks the class as a Spring service.
@Autowired is used to inject GraphDBRepository into RDFService. executeSparqlQuery is a method to execute a SPARQL query using GraphDBRepository.

Step 5: Create Controller Class to Handle HTTP Requests

java
Copy code

```
import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.RequestParam; import
org.springframework.web.bind.annotation.RestController;

@RestController
public class RDFController {

    private final RDFService rdfService;

    @Autowired
    public RDFController(RDFService rdfService)
        { this.rdfService = rdfService;
```

```
    }

    @GetMapping("/sparql")
    public String executeSparqlQuery(@RequestParam String query) { // Call
        service to execute SPARQL query return
        rdfService.executeSparqlQuery(query);
    }
}
```

In the code above:

@RestController marks the class as a Spring controller.
@Autowired is used to inject RDFService into RDFController.
@GetMapping("/sparql") defines an API endpoint for executing SPARQL queries.

==========

Below is the build.gradle equivalent of the pom.xml snippet and configuration in the previous example:

```
groovy
Copy code
plugins { id
    'org.springframework.boot' version '2.5.4' id
    'io.spring.dependency-management' version '1.0.11.RELEASE'
}

repositories
    { mavenCentral()
}

dependencies
    { implementation 'org.springframework.boot:spring-boot-starter' implementation
    'org.springframework.boot:spring-boot-starter-web' implementation
    'com.ontotext.graphdb:graphdb-free-runtime:INSERT_VERSION_HERE'
}

configurations
    { compileOnly {
        extendsFrom annotationProcessor
    }
}

springBoot
    { buildInfo()
}

// Replace INSERT_VERSION_HERE with the version of GraphDB you are using Note that
you need to replace INSERT_VERSION_HERE with the specific version of GraphDB you are using.
```

You can also use the org.springframework.boot plugin to create applications that can run independently. If you want to create an executable JAR file using java -jar, add the following configuration:

groovy
Copy code
springBoot
{ buildInfo()
executable = true
}
After creating a new build.gradle file, you can use the following command to download dependencies and build the project:

bash
Copy code ./
gradlew build The
above command will create the file build/libs/your-application-name.jar. You can run the application with the command:

bash
Copy code
java -jar build/libs/your-application-name.jar Note that you
need to install Gradle on your computer to use the above commands.

========== Python

If you want to use FastAPI in Python to create an application that interacts with GraphDB, you also need to use a Python RDF library to perform SPARQL queries and interact with RDF data. One of the popular libraries is rdflib.

Below is an example of how a simple FastAPI application can be set up to perform SPARQL queries and interact with GraphDB. First of all, you need to install the necessary libraries:

bash
Copy code
pip install fastapi uvicorn rdflib Then create
a main.py file with the following content:

python
Copy code
from fastapi import FastAPI, HTTPException from rdflib
import Graph, URIRef

app = FastAPI()

```
# Path to GraphDB endpoint
graphdb_endpoint = "http://localhost:7200/repositories/your-repository-name"

# Create Graph object from path graph =
Graph()
graph.load(graphdb_endpoint, format="turtle")


@app.get("/sparql")
async def execute_sparql_query(query: str):
    try:
        # Execute SPARQL query results =
          graph.query(query)

          # Convert results to a Python list to return result_list =
          [dict(zip(results.vars, binding)) for binding in results.bindings]


          return {"results": result_list} except
    Exception as e: raise
          HTTPException(status_code=500, detail=str(e))
```

In the code above:

FastAPI is used to create API web applications. rdflib is
used to interact with RDF data and perform SPARQL queries.
The SPARQL /sparql endpoint receives a SPARQL query via the query parameter and returns the results as JSON.

Run the FastAPI application with the following command:

bash

Copy code

```
uvicorn main:app --reload Then
```

you can visit http://127.0.0.1:8000/sparql?query=YOUR_SPARQL_QUERY in the browser or use other tools like curl or Postman to send queries query SPARQL and get results.

Note: You need to replace your-repository-name and YOUR_SPARQL_QUERY with your repository name and SPARQL query.