Università di Catania

Uni ct MATEMATICA E INFORMATICA

Spring Boot JPA

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

alessandro.midolo@unict.it

https://www.dmi.unict.it/amidolo/

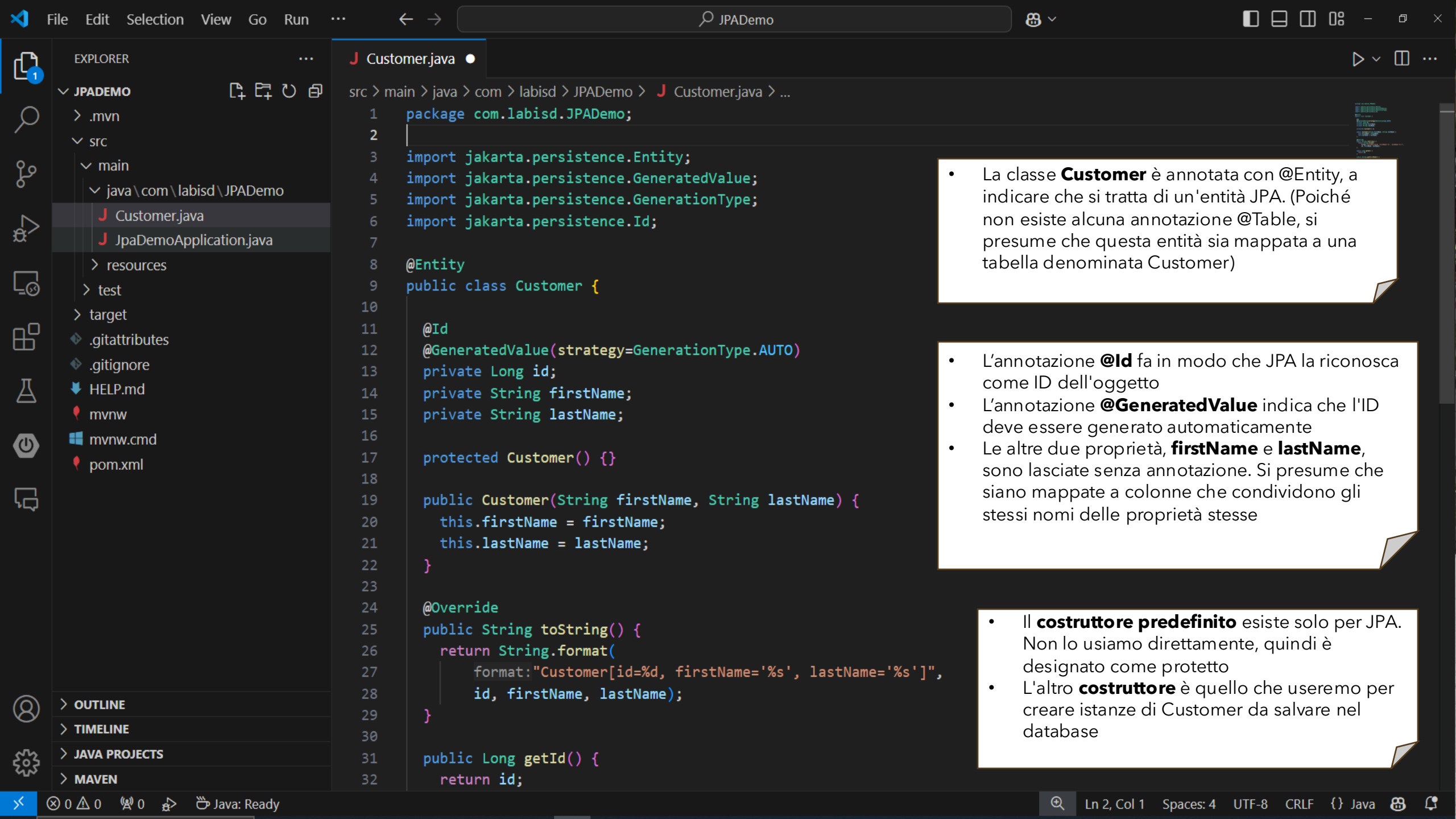
A.A. 2024/2025

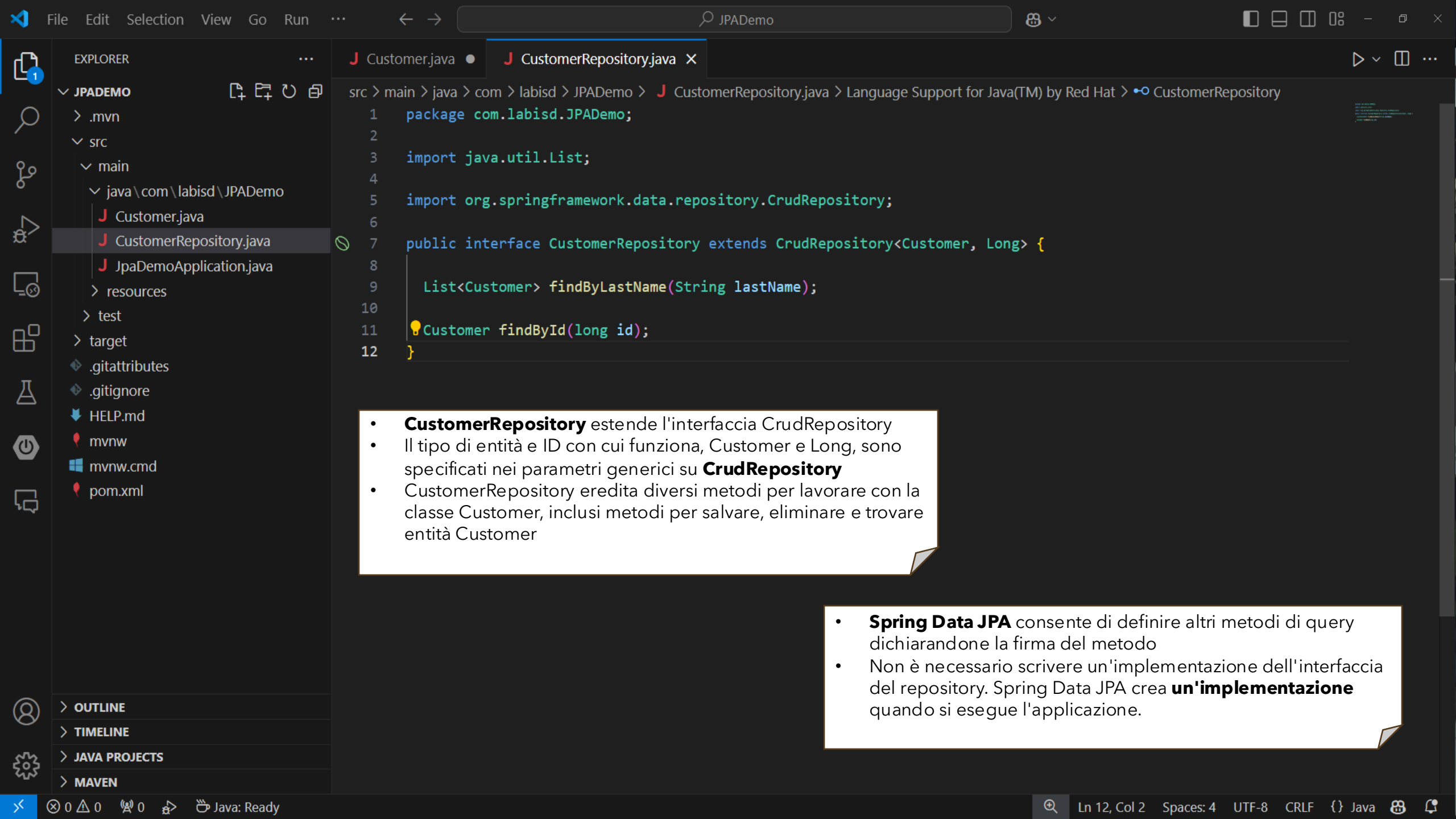
Spring Data JPA

- Creare un'applicazione che utilizza **Spring Data JPA** per archiviare e recuperare dati in un database relazionale
- Creiamo un'applicazione che memorizza i POJO (Plain Old Java Objects) dei clienti in un **database**
- Il codice completo del progetto lo trovate già pronto sul repository github

Inizializziamo il Progetto

- Navigate all'url https://start.spring.io/ per inizializzare il progetto
- Scegliere Maven e java come linguaggio
- Selezionare le dipendenze **Spring Data JPA**, **H2 Database** e **Spring Web**
- Generare il progetto
- Scaricare il file zip ed estrarlo all'interno del proprio IDE


File Edit Selection View Go Run
JPADemo
EXPLORER
JPADEMO
.mvn
src
main
java\com\labisd\JPADemo
Customer.java
JpaDemoApplication.java
resources
test
target
.gitattributes
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
Customer.java
src > main > java > com > labisd > JPADemo > Customer.java > ...
1 package com.labisd.JPADemo;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 @Entity
9 public class Customer {
10
11 @Id
12 @GeneratedValue(strategy=GenerationType.AUTO)
13 private Long id;
14 private String firstName;
15 private String lastName;
16
17 protected Customer() {}
18
19 public Customer(String firstName, String lastName) {
20 this.firstName = firstName;
21 this.lastName = lastName;
22 }
23
24 @Override
25 public String toString() {
26 return String.format(
27 format:"Customer[id=%d, firstName='%s', lastName='%s']",
28 id, firstName, lastName);
29 }
30
31 public Long getId() {
32 return id;
La classe Customer è annotata con @Entity, a indicare che si tratta di un'entità JPA. (Poiché non esiste alcuna annotazione @Table, si presume che questa entità sia mappata a una tabella denominata Customer)
L'annotazione @Id fa in modo che JPA la riconosca come ID dell'oggetto
L'annotazione @GeneratedValue indica che l'ID deve essere generato automaticamente
Le altre due proprietà, firstName e lastName, sono lasciate senza annotazione. Si presume che siano mappate a colonne che condividono gli stessi nomi delle proprietà stesse
Il costruttore predefinito esiste solo per JPA. Non lo usiamo direttamente, quindi è designato come protetto
L'altro costruttore è quello che useremo per creare istanze di Customer da salvare nel database
Java: Ready
Ln 2, Col 1
Spaces: 4
UTF-8
CRLF
Java


File Edit Selection View Go Run
JPADemo
EXPLORER
JPADEMO
.mvn
src
main
java\com\labisd\JPADemo
Customer.java
CustomerRepository.java
JpaDemoApplication.java
resources
test
target
.gitattributes
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
Customer.java
CustomerRepository.java
src > main > java > com > labisd > JPADemo > CustomerRepository.java > Language Support for Java(TM) by Red Hat > CustomerRepository
1 package com.labisd.JPADemo;
2
3 import java.util.List;
4
5 import org.springframework.data.repository.CrudRepository;
6
7 public interface CustomerRepository extends CrudRepository<Customer, Long> {
8
9 List<Customer> findByLastName(String lastName);
10
11 Customer findById(long id);
12 }
• CustomerRepository estende l'interfaccia CrudRepository
• Il tipo di entità e ID con cui funziona, Customer e Long, sono specificati nei parametri generici su CrudRepository
• CustomerRepository eredita diversi metodi per lavorare con la classe Customer, inclusi metodi per salvare, eliminare e trovare entità Customer
• Spring Data JPA consente di definire altri metodi di query dichiarandone la firma del metodo
• Non è necessario scrivere un'implementazione dell'interfaccia del repository. Spring Data JPA crea un'implementazione quando si esegue l'applicazione.
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
Java: Ready
Ln 12, Col 2
Spaces: 4
UTF-8
CRLF
Java

Interface CrudRepository<T,ID>

All Superinterfaces:
`Repository<T,ID>`

All Known Subinterfaces:
`ListCrudRepository<T,ID>`

https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html

---

```
@NoRepositoryBean
public interface CrudRepository<T,ID>
extends Repository<T,ID>
```

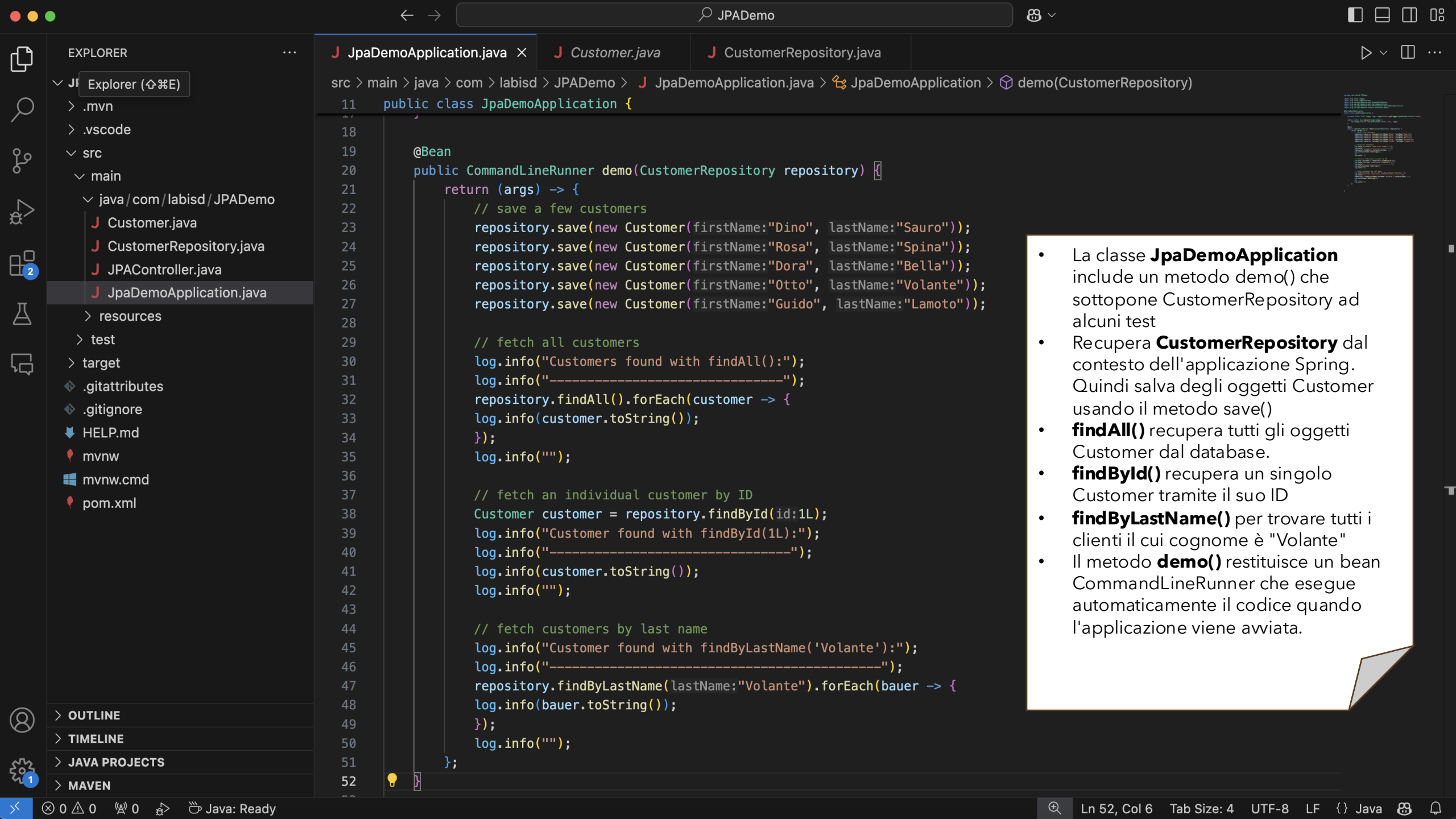
Interface for generic CRUD operations on a repository for a specific type.

Author:
Oliver Gierke, Eberhard Wolff, Jens Schauder

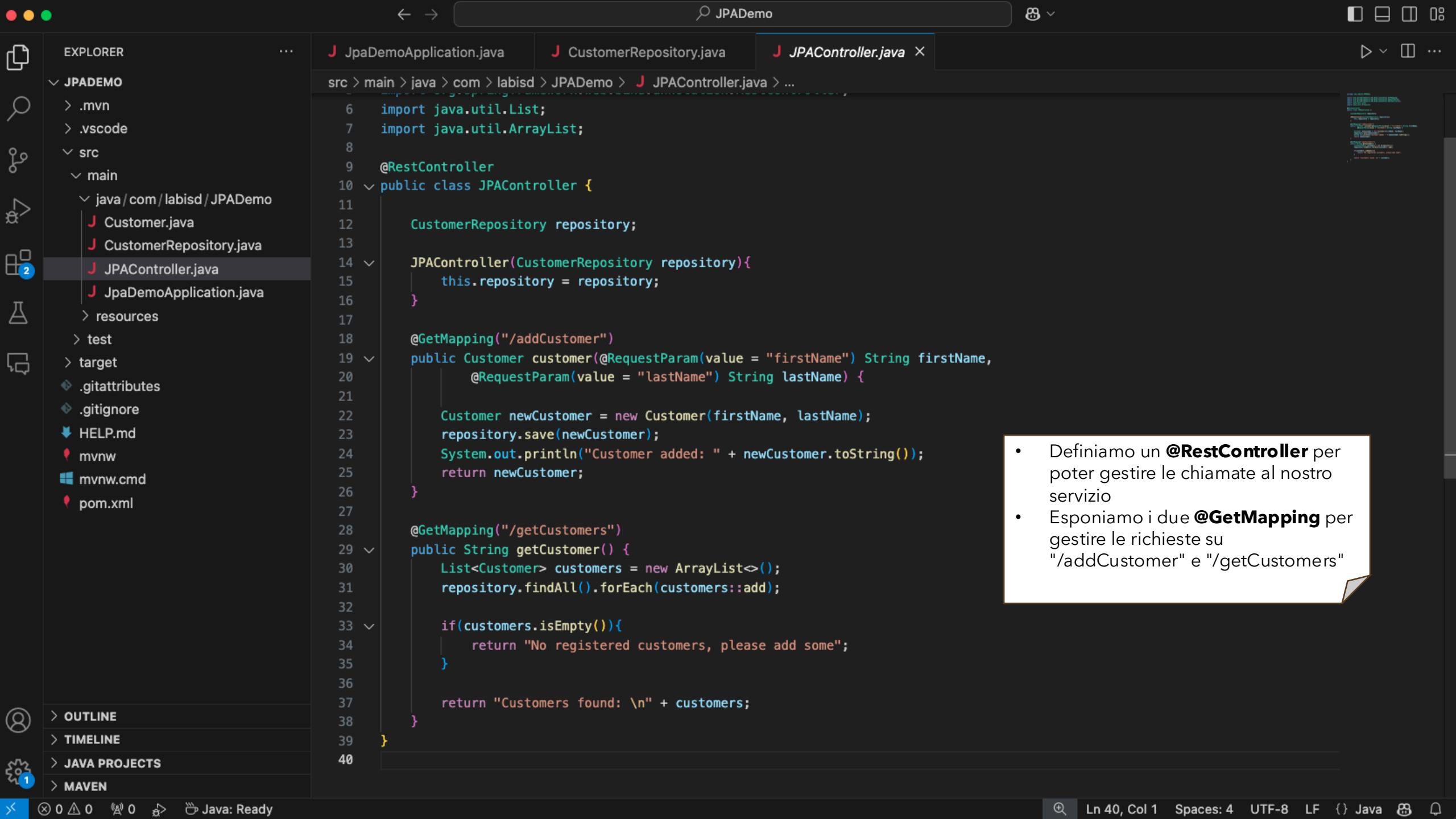
Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
long	`count()`	Returns the number of entities available.
void	`delete(T entity)`	Deletes a given entity.
void	`deleteAll()`	Deletes all entities managed by the repository.
void	`deleteAll(Iterable <? extends T> entities)`	Deletes the given entities.
void	`deleteAllById(Iterable <? extends ID> ids)`	Deletes all instances of the type T with the given IDs.
void	`deleteById(ID id)`	Deletes the entity with the given id.
boolean	`existsById(ID id)`	Returns whether an entity with the given id exists.
`Iterable <T>`	`findAll()`	Returns all instances of the type.
`Iterable <T>`	`findAllById(Iterable <ID> ids)`	Returns all instances of the type T with the given IDs.
`Optional <T>`	`findById(ID id)`	Retrieves an entity by its id.
`<S extends T> S`	`save(S entity)`	Saves a given entity.
`<S extends T> Iterable <S>`	`saveAll(Iterable <S> entities)`	Saves all given entities.



- La classe **JpaDemoApplication** include un metodo demo() che sottopone CustomerRepository ad alcuni test
- Recupera **CustomerRepository** dal contesto dell'applicazione Spring. Quindi salva degli oggetti Customer usando il metodo save()
- **findAll()** recupera tutti gli oggetti Customer dal database.
- **findById()** recupera un singolo Customer tramite il suo ID
- **findByLastName()** per trovare tutti i clienti il cui cognome è "Volante"
- Il metodo **demo()** restituisce un bean CommandLineRunner che esegue automaticamente il codice quando l'applicazione viene avviata.


JPADemo
EXPLORER
JPADEMO
.mvn
.vscode
src
main
java / com / labisd / JPADemo
Customer.java
CustomerRepository.java
JPAController.java
JpaDemoApplication.java
resources
test
target
.gitattributes
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
JpaDemoApplication.java
CustomerRepository.java
JPAController.java
src > main > java > com > labisd > JPADemo > JPAController.java > ...
6 import java.util.List;
7 import java.util.ArrayList;
8
9 @RestController
10 public class JPAController {
11
12 CustomerRepository repository;
13
14 JPAController(CustomerRepository repository){
15 this.repository = repository;
16 }
17
18 @GetMapping("/addCustomer")
19 public Customer customer(@RequestParam(value = "firstName") String firstName,
20 @RequestParam(value = "lastName") String lastName) {
21
22 Customer newCustomer = new Customer(firstName, lastName);
23 repository.save(newCustomer);
24 System.out.println("Customer added: " + newCustomer.toString());
25 return newCustomer;
26 }
27
28 @GetMapping("/getCustomers")
29 public String getCustomer() {
30 List<Customer> customers = new ArrayList<>();
31 repository.findAll().forEach(customers::add);
32
33 if(customers.isEmpty()){
34 return "No registered customers, please add some";
35 }
36
37 return "Customers found: \n" + customers;
38 }
39 }
40
• Definiamo un @RestController per poter gestire le chiamate al nostro servizio
• Esponiamo i due @GetMapping per gestire le richieste su "/addCustomer" e "/getCustomers"
Java: Ready
Ln 40, Col 1
Spaces: 4
UTF-8
LF
Java

Interroghiamo il nostro servizio

- Possiamo quindi interrogare il nostro servizio attraverso il nostro browser
  - http://localhost:8080/getCustomers
  - http://localhost:8080/addCustomer?firstName=Ale&lastName=Midolo

REST API per accedere con JPA

REST API per accedere con JPA

- Creiamo un'applicazione Spring che consente di creare e recuperare oggetti ***Person*** archiviati in un database utilizzando Spring Data REST
- Spring Data REST prende le funzionalità di Spring HATEOAS e Spring Data JPA e le combina automaticamente
  - Con HATEOAS, un client interagisce con un'applicazione di rete i cui server applicativi forniscono informazioni in modo dinamico tramite ipermedia (e.g. URL)

Inizializziamo il Progetto

- Navigate all'url https://start.spring.io/ per inizializzare il progetto
- Scegliere Maven e java come linguaggio
- Selezionare le dipendenze Rest repositories, Spring Data JPA e H2 Database
- Generare il progetto
- Scaricare il file zip ed estrarlo all'interno del proprio IDE

JPARESTDemo

EXPLORER

JPARESTDEMO
- .mvn
- src
  - main
    - java / com / labisd / JPARESTDemo
      - JparestDemoApplication.java
      - Person.java
    - resources
  - test
- target
- .gitattributes
- .gitignore
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml

OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN

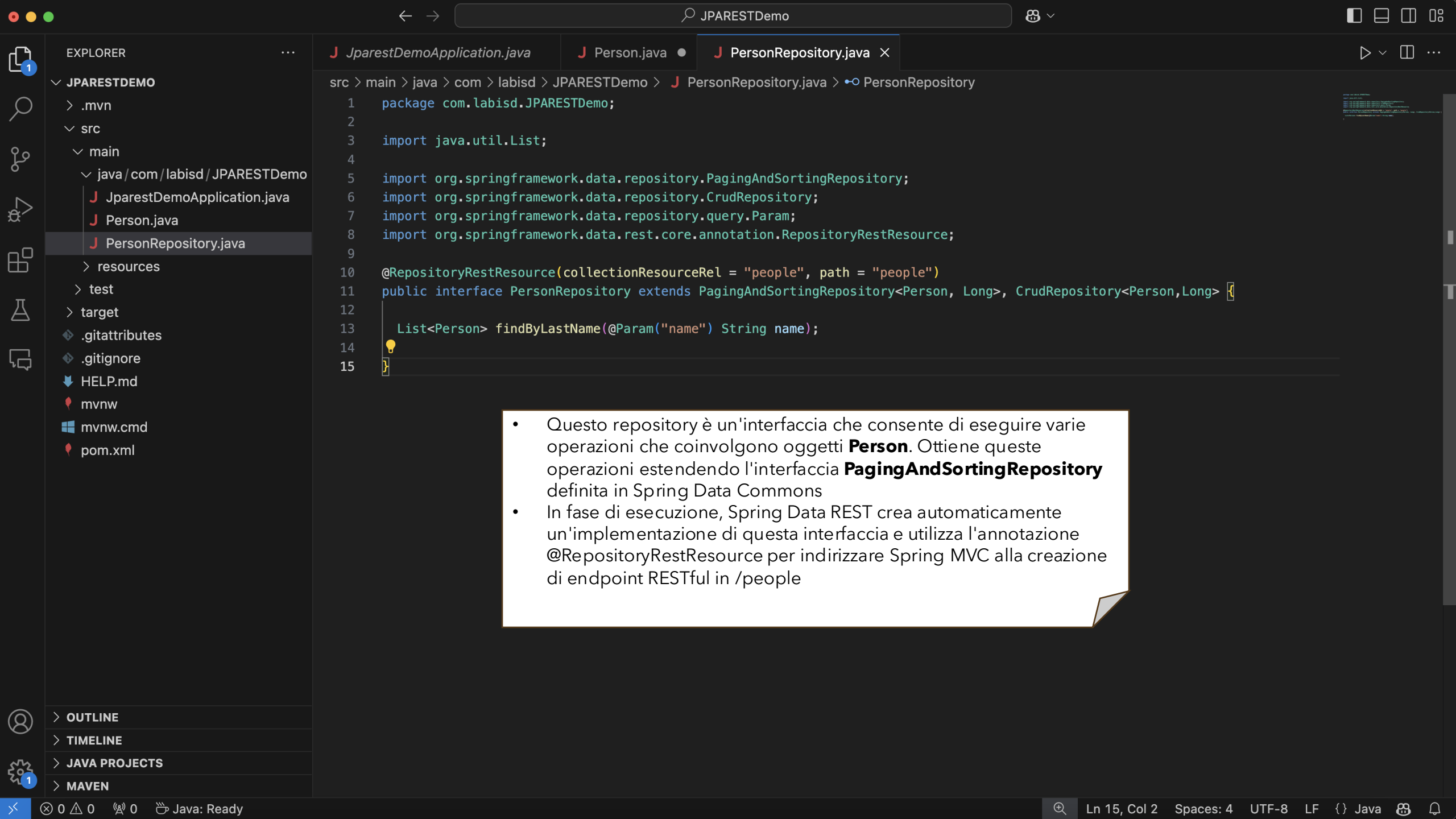
JparestDemoApplication.java | Person.java

src > main > java > com > labisd > JPARESTDemo > Person.java > ...

```java
package com.labisd.JPARESTDemo;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Person {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long id;

  private String firstName;
  private String lastName;

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }
}
```

- Definiamo la classe **Person** che rappresenta la nostra entità all'interno del Database
- L'**ID** verrà generato automaticamente
- I campi firstName e lastName sono correlati con i rispettivi getter e setter

0 0 0 Java: Ready | Ln 34, Col 1 | Spaces: 4 | UTF-8 | LF | Java



- Questo repository è un'interfaccia che consente di eseguire varie operazioni che coinvolgono oggetti **Person**. Ottiene queste operazioni estendendo l'interfaccia **PagingAndSortingRepository** definita in Spring Data Commons
- In fase di esecuzione, Spring Data REST crea automaticamente un'implementazione di questa interfaccia e utilizza l'annotazione @RepositoryRestResource per indirizzare Spring MVC alla creazione di endpoint RESTful in /people

Interroghiamo il servizio

- Invocando solo http://localhost:8080, il servizio mostra ciò che espone
  - C è un link /people che mette a disposizione tre diverse opzioni: ?page, ?size e ?sort
  - Queste opzioni sono ereditate dall'interfaccia **PagingAndSortingRepository**

```
Alessandros-MacBook-Air:~ midolo$ curl http://localhost:8080
{
  "_links" : {
    "people" : {
      "href" : "http://localhost:8080/people{?page,size,sort*}",
      "templated" : true
    },
    "profile" : {
      "href" : "http://localhost:8080/profile"
    }
  }
Alessandros-MacBook-Air:~ midolo$
```

Attualmente non ci sono "**Person**" all'interno del nostro database, quindi la chiamata a /people restituisce la lista vuota

```
Alessandros-MacBook-Air:~ midolo$ curl http://localhost:8080/people
{
  "_embedded" : {
    "people" : [ ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/people?page=0&size=20"
    },
    "profile" : {
      "href" : "http://localhost:8080/profile/people"
    },
    "search" : {
      "href" : "http://localhost:8080/people/search"
    }
  },
  "page" : {
    "size" : 20,
    "totalElements" : 0,
    "totalPages" : 0,
    "number" : 0
  }
```

Creiamo un nuovo record di tipo **Person** usando il comando curl (ci permette di effettuare chiamate POST):

curl -i -H "Content-Type:application/json" -d '{"firstName": "Frodo", "lastName": "Baggins"}' http://localhost:8080/people

- **-i** : Assicura che si possa vedere il messaggio di risposta, incluse le intestazioni. Viene mostrato l'URI della Persona appena creata
- -**H** "Content-Type:application/json": Imposta il tipo di contenuto in modo che l'applicazione sappia che il payload contiene un oggetto JSON
- **-d**: indica i dati che si vogliono inviare
- Se usate windows, bisogna usare le doppie virgolette e non le singole
  - *-d "{\"firstName\": \"Frodo\", \"lastName\": \"Baggins\"}"*

```
Alessandros-MacBook-Air:~ midolo$ curl -i -H "Content-Type:application/json" -d '{"firstName": "Frodo", "lastName": "Baggins"}' http://localhost:8080/people
HTTP/1.1 201
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Location: http://localhost:8080/people/1
Content-Type: application/hal+json
Transfer-Encoding: chunked
Date: Wed, 06 Nov 2024 14:37:51 GMT

{
  "firstName" : "Frodo",
  "lastName" : "Baggins",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/people/1"
    },
    "person" : {
      "href" : "http://localhost:8080/people/1"
    }
  }
```

- L'oggetto *people* contiene una lista che include la persona *Frodo* appena creata
- Al suo interno la persona creata ha un campo "self" che indica proprio l'url per accedere direttamente alla risorsa

```
}Alessandros-MacBook-Air:~ midolo$ curl http://localhost:8080/people/1
{
  "firstName" : "Frodo",
  "lastName" : "Baggins",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/people/1"
    },
    "person" : {
      "href" : "http://localhost:8080/people/1"
    }
  }
```

```
Alessandros-MacBook-Air:~ midolo$ curl http://localhost:8080/people
{
  "_embedded" : {
    "people" : [ {
      "firstName" : "Frodo",
      "lastName" : "Baggins",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/people/1"
        },
        "person" : {
          "href" : "http://localhost:8080/people/1"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/people?page=0&size=20"
    },
    "profile" : {
      "href" : "http://localhost:8080/profile/people"
    },
    "search" : {
      "href" : "http://localhost:8080/people/search"
    }
  },
  "page" : {
    "size" : 20,
    "totalElements" : 1,
    "totalPages" : 1,
    "number" : 0
  }
```

Con la "search" è possibile farsi restituire tutti gli url per effettuare delle "query", inclusi anche i parrametri HTTP da usare

```
[Alessandros-MacBook-Air:~ midolo$ curl http://localhost:8080/people/search
 {
   "_links" : {
     "findByLastName" : {
       "href" : "http://localhost:8080/people/search/findByLastName{?name}",
       "templated" : true
     },
     "self" : {
       "href" : "http://localhost:8080/people/search"
     }
   }
```

```
}Alessandros-MacBook-Air:~ midolo$ curl http://localhost:8080/people/search/findByLastName?=name=Baggins
{
  "_embedded" : {
    "people" : [ ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/people/search/findByLastName?name=Baggins"
    }
  }
```

- Si possono usare i comandi REST ***PUT***, ***PATCH*** e ***DELETE*** per sostituire, aggiornare ed eliminare record esistenti

```
Alessandros-MacBook-Air:~ midolo$ curl -X PUT -H "Content-Type:application/json" -d '{"firstName": "Bilbo", "lastName": "Baggins"}' http://localhost:8080/people/1
{
  "firstName" : "Bilbo",
  "lastName" : "Baggins",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/people/1"
    },
    "person" : {
      "href" : "http://localhost:8080/people/1"
    }
  }
```

```
}Alessandros-MacBook-Air:~ midolo$ curl -X PATCH -H "Content-Type:application/json" -d '{"firstName": "Bilbo Jr."}' http://localhost:8080/people/1
{
  "firstName" : "Bilbo Jr.",
  "lastName" : "Baggins",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/people/1"
    },
    "person" : {
      "href" : "http://localhost:8080/people/1"
    }
  }
```

```
}Alessandros-MacBook-Air:~ midolo$ curl -X DELETE http://localhost:8080/people/1
{
  "firstName" : "Bilbo Jr.",
  "lastName" : "Baggins",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/people/1"
    },
    "person" : {
      "href" : "http://localhost:8080/people/1"
    }
  }
```

Consume a RESTful Web Service

Consume a RESTful Web Service

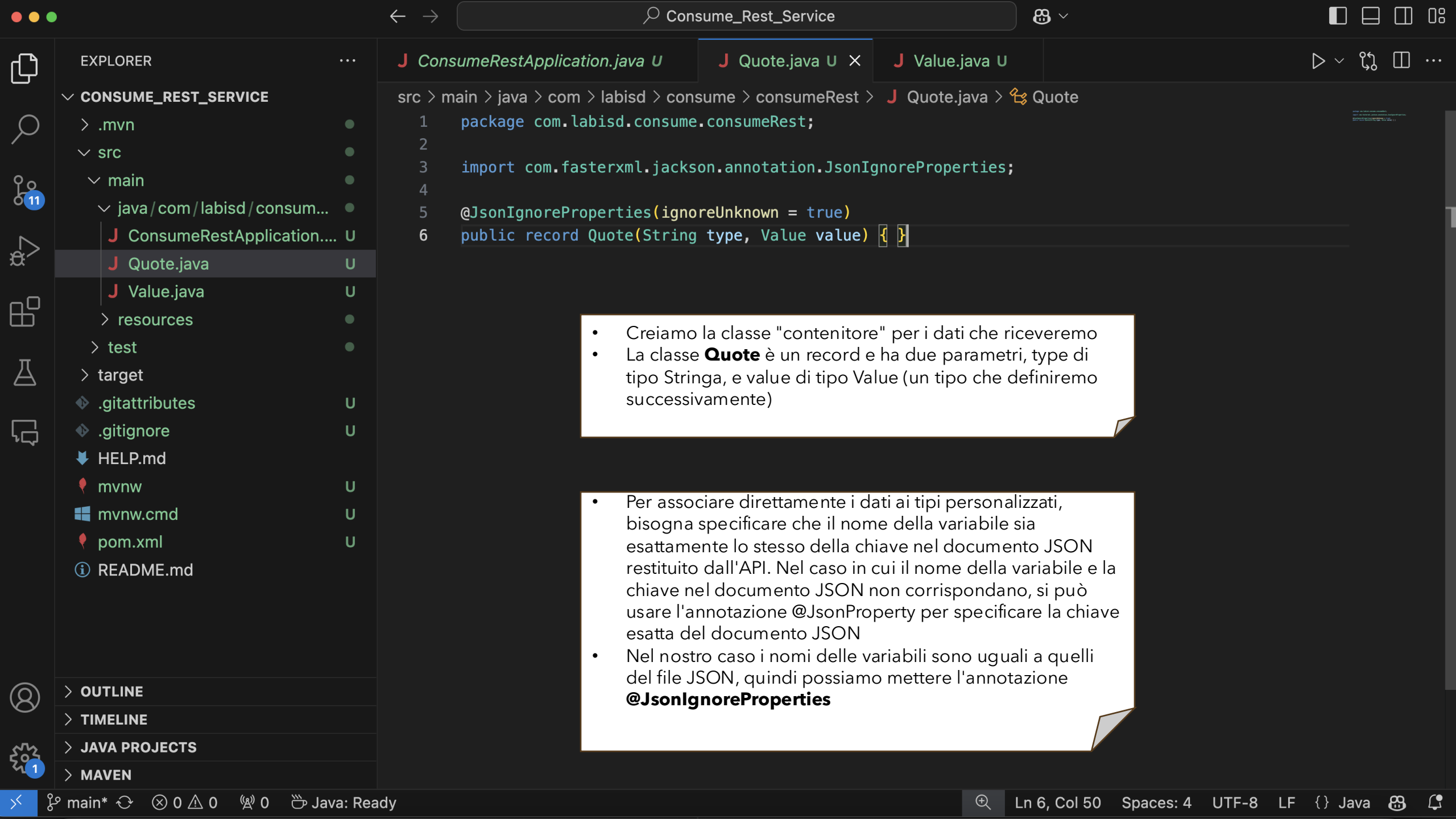
- Creiamo un'applicazione Spring che consente di "consumare" un servizio web RESTful
- L'applicazione userà il *RestTemplate* di Spring per ottenere una citazione random su SpringBoot, attraverso l'indirizzo *http://localhost:8080/api/random*

Inizializziamo il Progetto

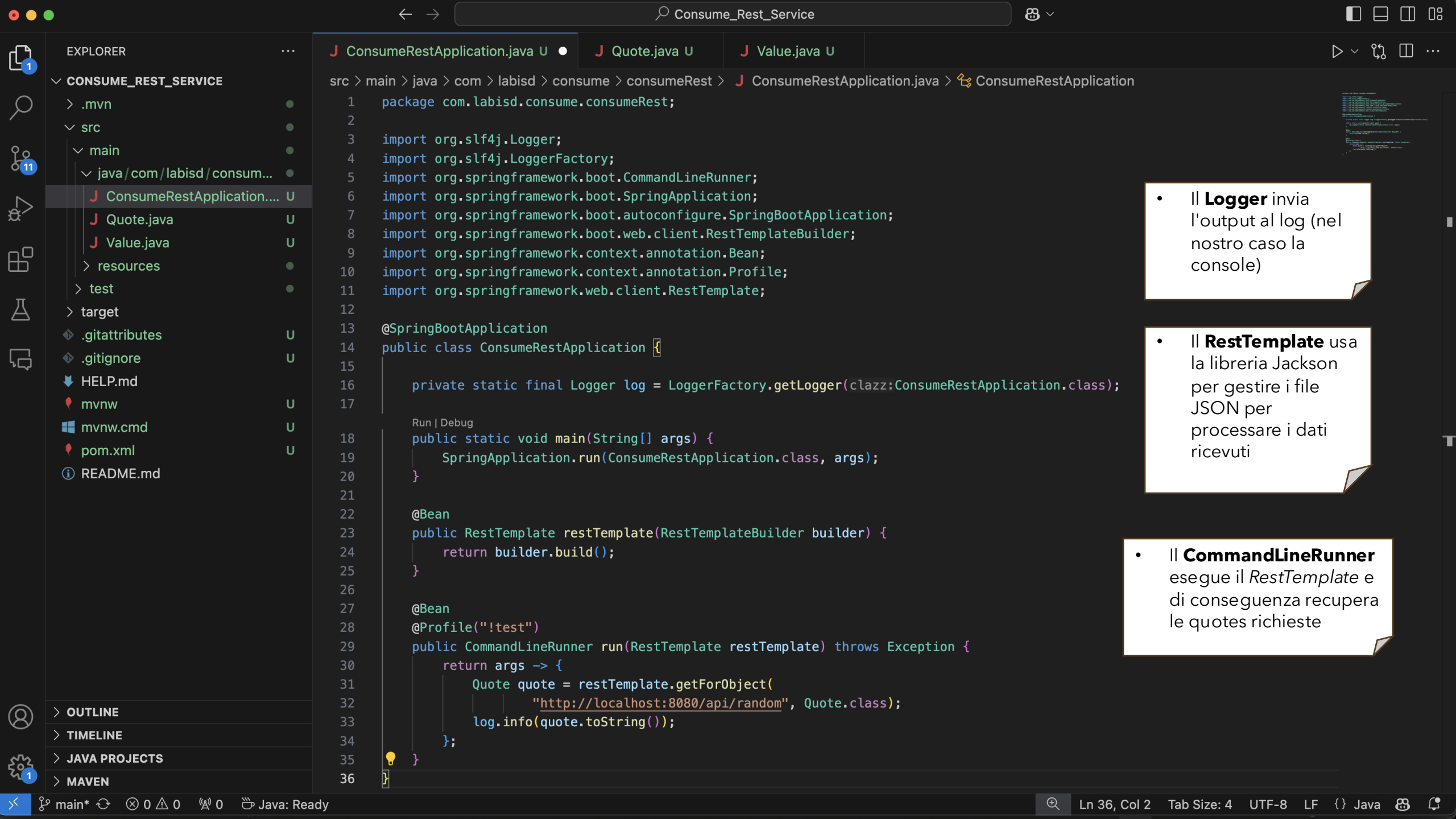
- Navigate all'url https://start.spring.io/ per inizializzare il progetto
- Scegliere Maven e java come linguaggio
- Selezionare la dipendenza Spring Web
- Generare il progetto
- Scaricare il file zip ed estrarlo all'interno del proprio IDE

Spring Boot Quoters

- Abbiamo bisogno di un **servizio REST** che esponga effettivamente delle risorse
- Usiamo il servizio messo a disposizione da Spring Boot, chiamato **quoters** (https://github.com/spring-guides/quoters)
- Facciamo quindi il clone del repository e lo eseguiamo nella nostra macchina
- Possiamo accedere ai risultati attraverso l'url http://localhost:8080/api/random (Questo indirizzo recupera casualmente una citazione su Spring Boot e la restituisce come documento JSON)
- A questo punto torniamo all'**applicazione** definita prima e la modifichiamo tale che possa usufruire in modo automatico e programmato delle risorse messe a disposizione da questo servizio


Consume_Rest_Service
EXPLORER
CONSUME_REST_SERVICE
.mvn
src
main
java/com/labisd/consum...
ConsumeRestApplication....
Quote.java
Value.java
resources
test
target
.gitattributes
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
README.md
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
ConsumeRestApplication.java U
Quote.java U
Value.java U
src > main > java > com > labisd > consume > consumeRest > Quote.java > Quote
1 package com.labisd.consume.consumeRest;
2
3 import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
4
5 @JsonIgnoreProperties(ignoreUnknown = true)
6 public record Quote(String type, Value value) { }
• Creiamo la classe "contenitore" per i dati che riceveremo
• La classe Quote è un record e ha due parametri, type di tipo Stringa, e value di tipo Value (un tipo che definiremo successivamente)
• Per associare direttamente i dati ai tipi personalizzati, bisogna specificare che il nome della variabile sia esattamente lo stesso della chiave nel documento JSON restituito dall'API. Nel caso in cui il nome della variabile e la chiave nel documento JSON non corrispondano, si può usare l'annotazione @JsonProperty per specificare la chiave esatta del documento JSON
• Nel nostro caso i nomi delle variabili sono uguali a quelli del file JSON, quindi possiamo mettere l'annotazione @JsonIgnoreProperties
main* 0 0 0 Java: Ready
Ln 6, Col 50 Spaces: 4 UTF-8 LF Java

Consume_Rest_Service
EXPLORER
CONSUME_REST_SERVICE
.mvn
src
main
java/com/labisd/consum...
ConsumeRestApplication.... U
Quote.java U
Value.java U
resources
test
target
.gitattributes U
.gitignore U
HELP.md
mvnw U
mvnw.cmd U
pom.xml U
README.md
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
ConsumeRestApplication.java U
Quote.java U
Value.java U
src > main > java > com > labisd > consume > consumeRest > Value.java > Value
1 package com.labisd.consume.consumeRest;
2
3 import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
4
5 @JsonIgnoreProperties(ignoreUnknown = true)
6 public record Value(Long id, String quote) { }
• La classe **Value** è un record e ha due parametri, **id** di tipo Long, e quote di tipo String
• Questi sono i campi che troviamo all'interno del file JSON restituito dal servizio REST
main* 0 0 0 Java: Ready
Ln 6, Col 47 Spaces: 4 UTF-8 LF Java



- Il **Logger** invia l'output al log (nel nostro caso la console)

- Il **RestTemplate** usa la libreria Jackson per gestire i file JSON per processare i dati ricevuti

- Il **CommandLineRunner** esegue il *RestTemplate* e di conseguenza recupera le quotes richieste

application.properties

application.properties src/main/resources file results

EXPLORER

CONSUME_REST_SERVICE

- .mvn
- src
  - main
    - java / com / labisd / consum...
      - ConsumeRestApplication.... U
      - Quote.java U
      - Value.java U
    - resources
  - test
- target
- .gitattributes U
- .gitignore U
- HELP.md
- mvnw U
- mvnw.cmd U
- pom.xml U
- README.md

OUTLINE

TIMELINE

JAVA PROJECTS

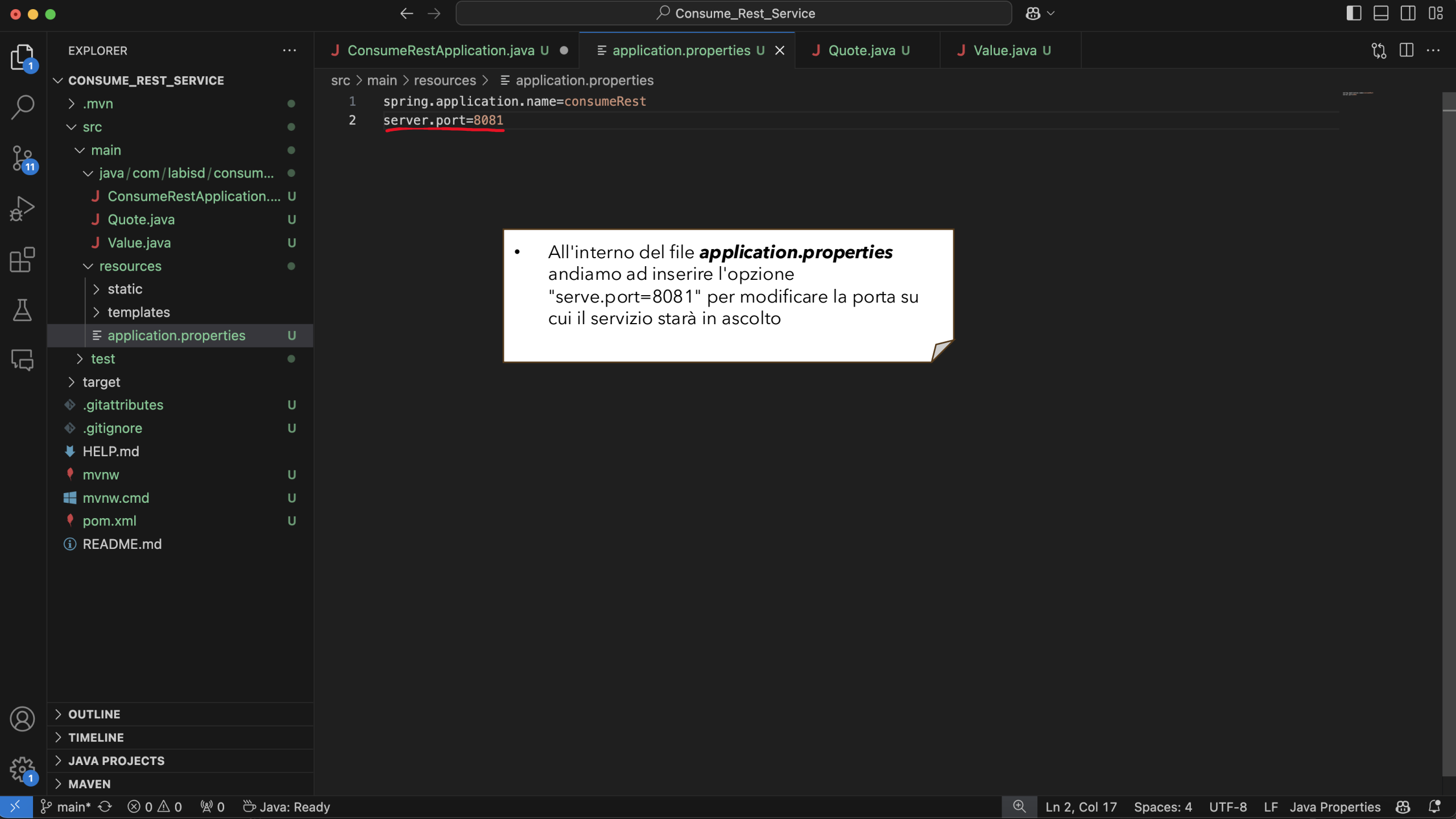
MAVEN

src > main > java > com > labisd > consume > consumeRest > ConsumeRestApplication.java > ConsumeRestApplication > log

```java
package com.labisd.consume.consumeRest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Profile;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class ConsumeRestApplication {

    private static final Logger log = LoggerFactory.getLogger(clazz:ConsumeRestApplication.class);

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(ConsumeRestApplication.class, args);
    }

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }

    @Bean
    @Profile("!test")
    public CommandLineRunner run(RestTemplate restTemplate) throws Exception {
        return args -> {
            Quote quote = restTemplate.getForObject(
                    "http://localhost:8080/api/random", Quote.class);
            log.info(quote.toString());
        };
    }
}
```

- Attenzione! L'applicazione quotes di spring boot che abbiamo in background sta già usando la **porta 8080**
- Dobbiamo quindi modificare le **impostazioni** di default della nostra app per cambiare porta ed eseguirla senza problemi
- Andiamo a modificare il file ***application.properties***

main* 0 0 0 Java: Ready Ln 16, Col 93 Tab Size: 4 UTF-8 LF Java


Consume_Rest_Service
EXPLORER
CONSUME_REST_SERVICE
.mvn
src
main
java / com / labisd / consum...
ConsumeRestApplication.... U
Quote.java U
Value.java U
resources
static
templates
application.properties U
test
target
.gitattributes U
.gitignore U
HELP.md
mvnw U
mvnw.cmd U
pom.xml U
README.md
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
ConsumeRestApplication.java U
application.properties U
Quote.java U
Value.java U
src > main > resources > application.properties
1 spring.application.name=consumeRest
2 server.port=8081
• All'interno del file application.properties andiamo ad inserire l'opzione "serve.port=8081" per modificare la porta su cui il servizio starà in ascolto
main*
0 0 0
Java: Ready
Ln 2, Col 17
Spaces: 4
UTF-8
LF
Java Properties

Università di Catania

Uni ct MATEMATICA E INFORMATICA

The End 

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

alessandro.midolo@unict.it

https://www.dmi.unict.it/amidolo/

A.A. 2024/2025