Spring Data REST in Spring Boot

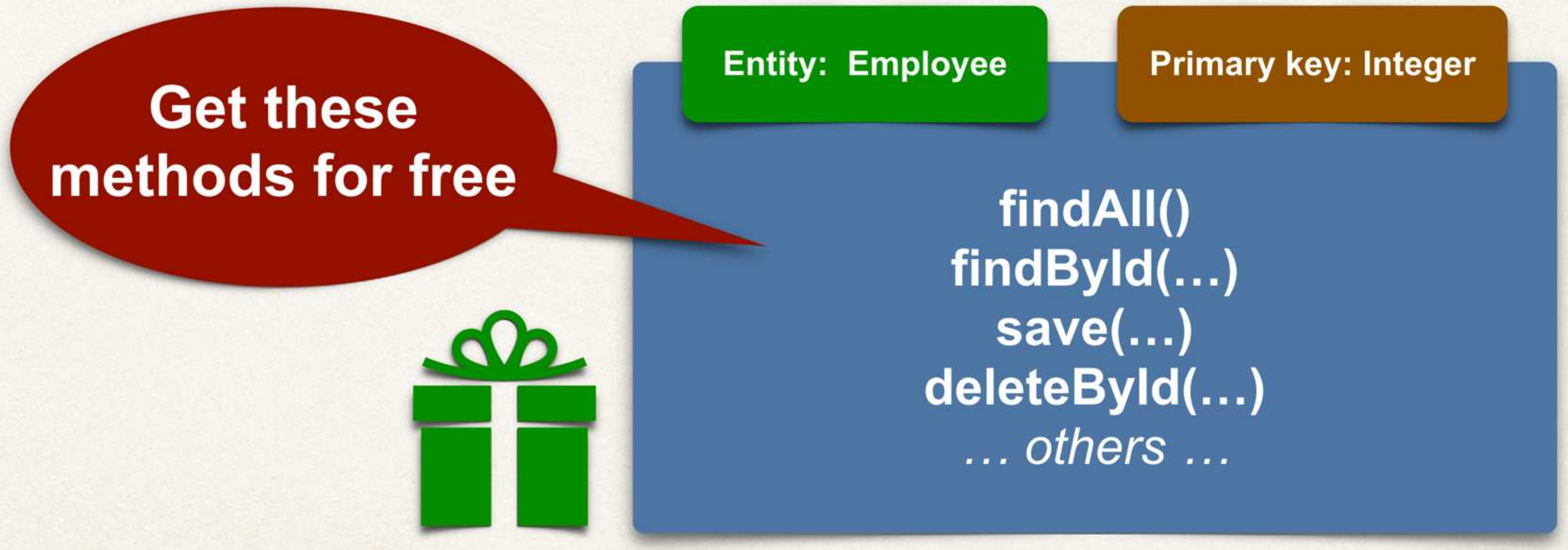
Spring Data JPA

Spring Data JPA

- Earlier, we saw the magic of Spring Data JPA

Spring Data JPA

- Earlier, we saw the magic of Spring Data JPA



Get these methods for free

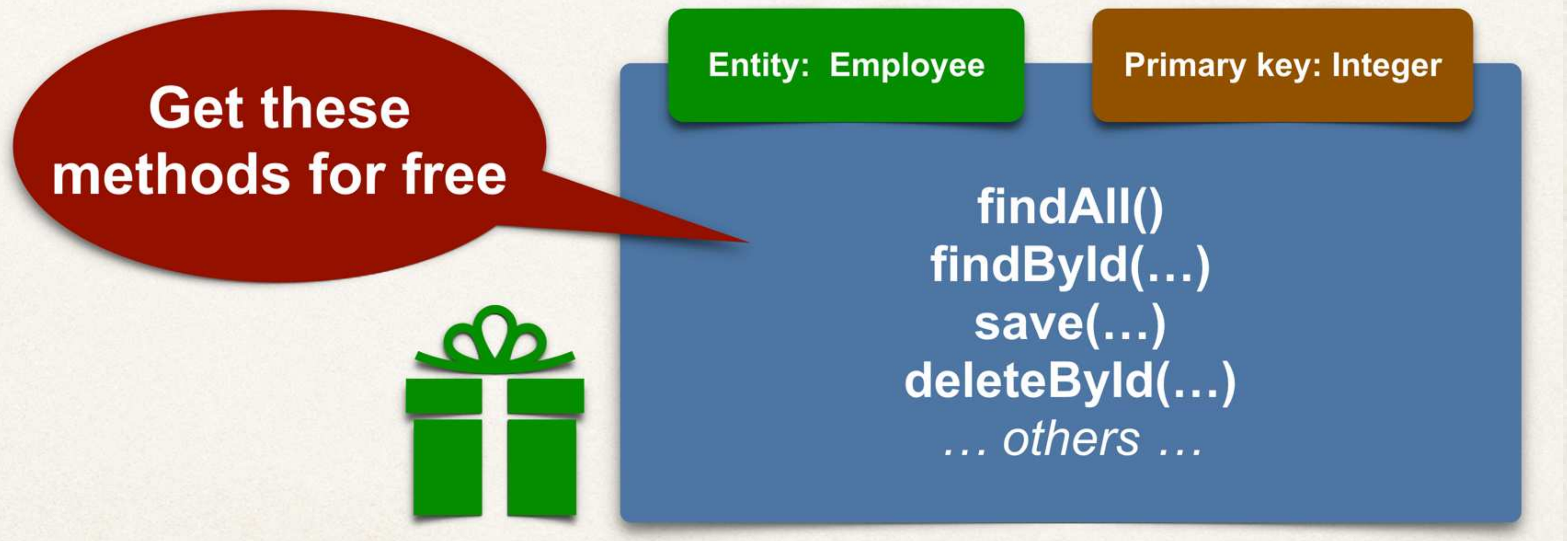
Entity: Employee

Primary key: Integer

findAll()
findById(…)
save(…)
deleteById(…)
… others …

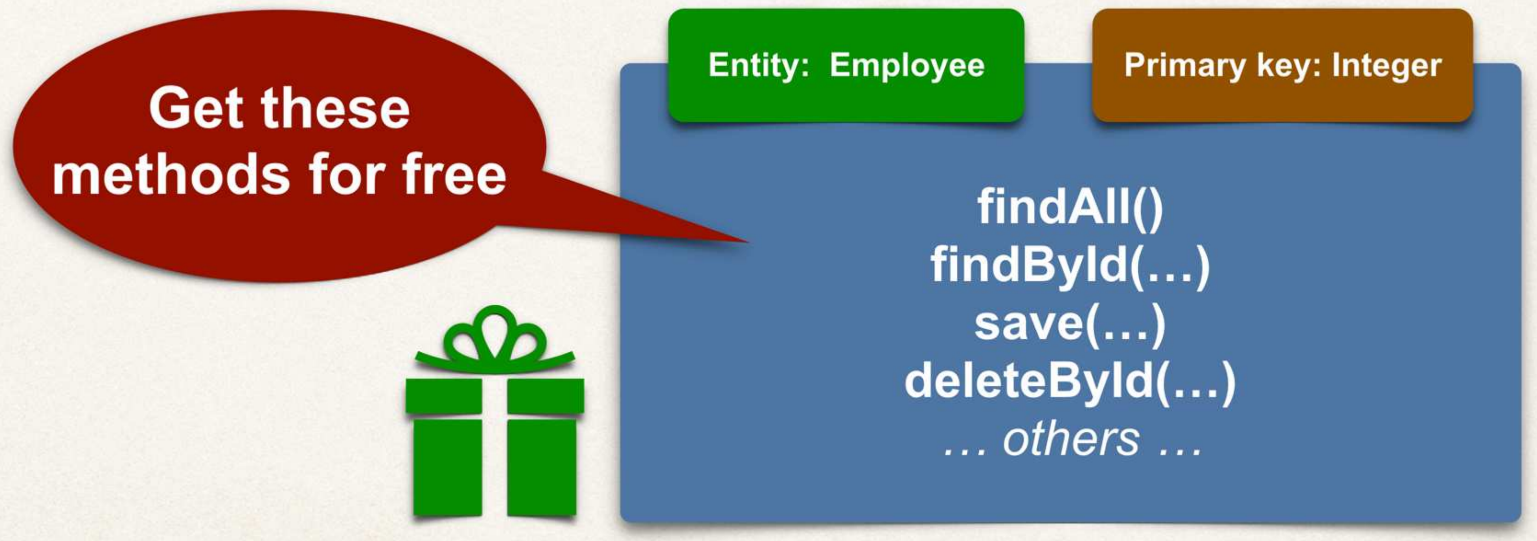
Spring Data JPA

- Earlier, we saw the magic of Spring Data JPA



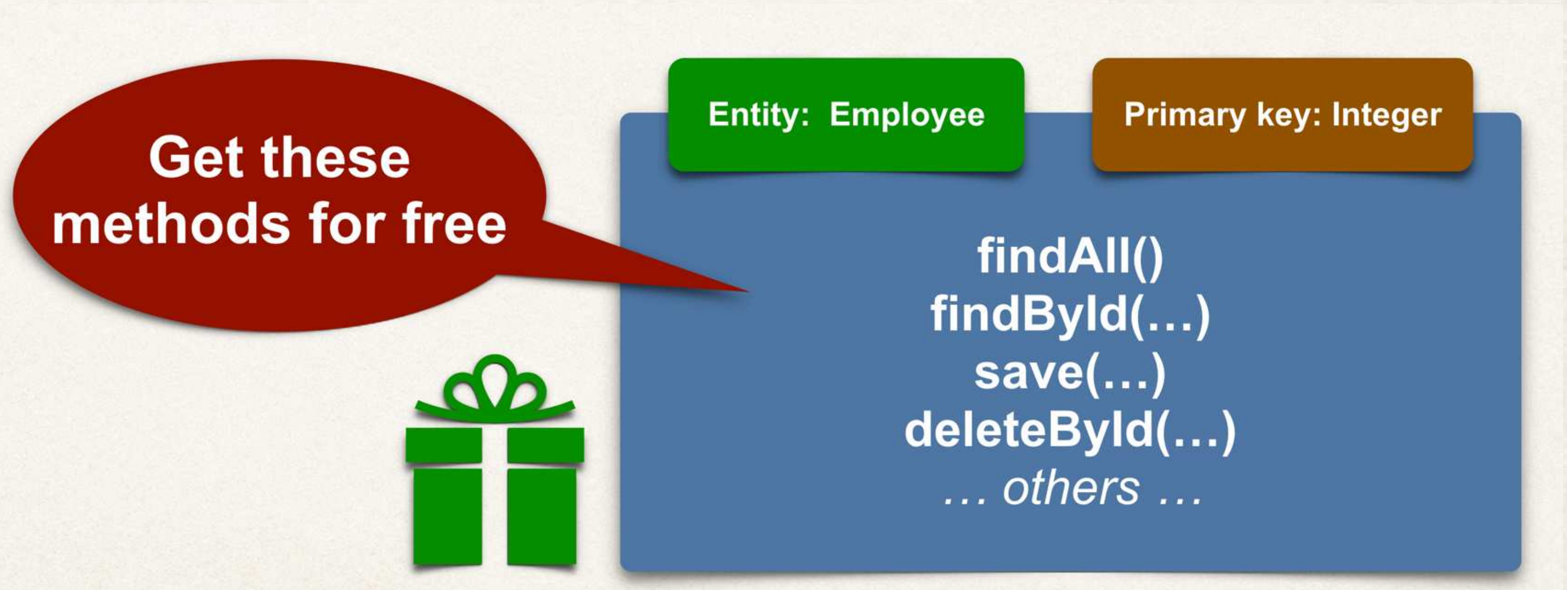
Spring Data JPA

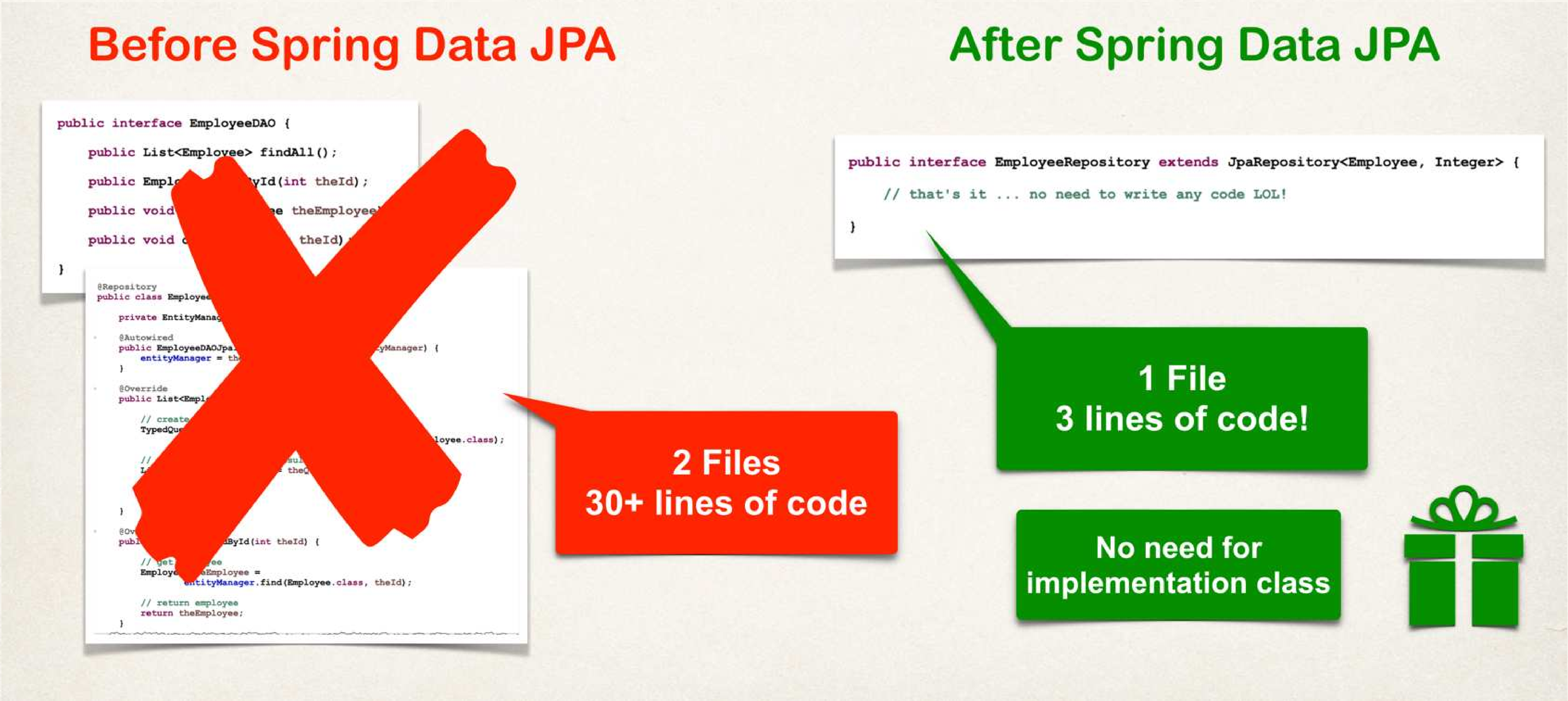
- Earlier, we saw the magic of Spring Data JPA
- This helped to eliminate boilerplate code



Spring Data JPA

- Earlier, we saw the magic of Spring Data JPA
- This helped to eliminate boilerplate code





```java
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

    // that's it ... no need to write any code LOL!

}
```

Hmmm …

Can this apply to REST APIs?

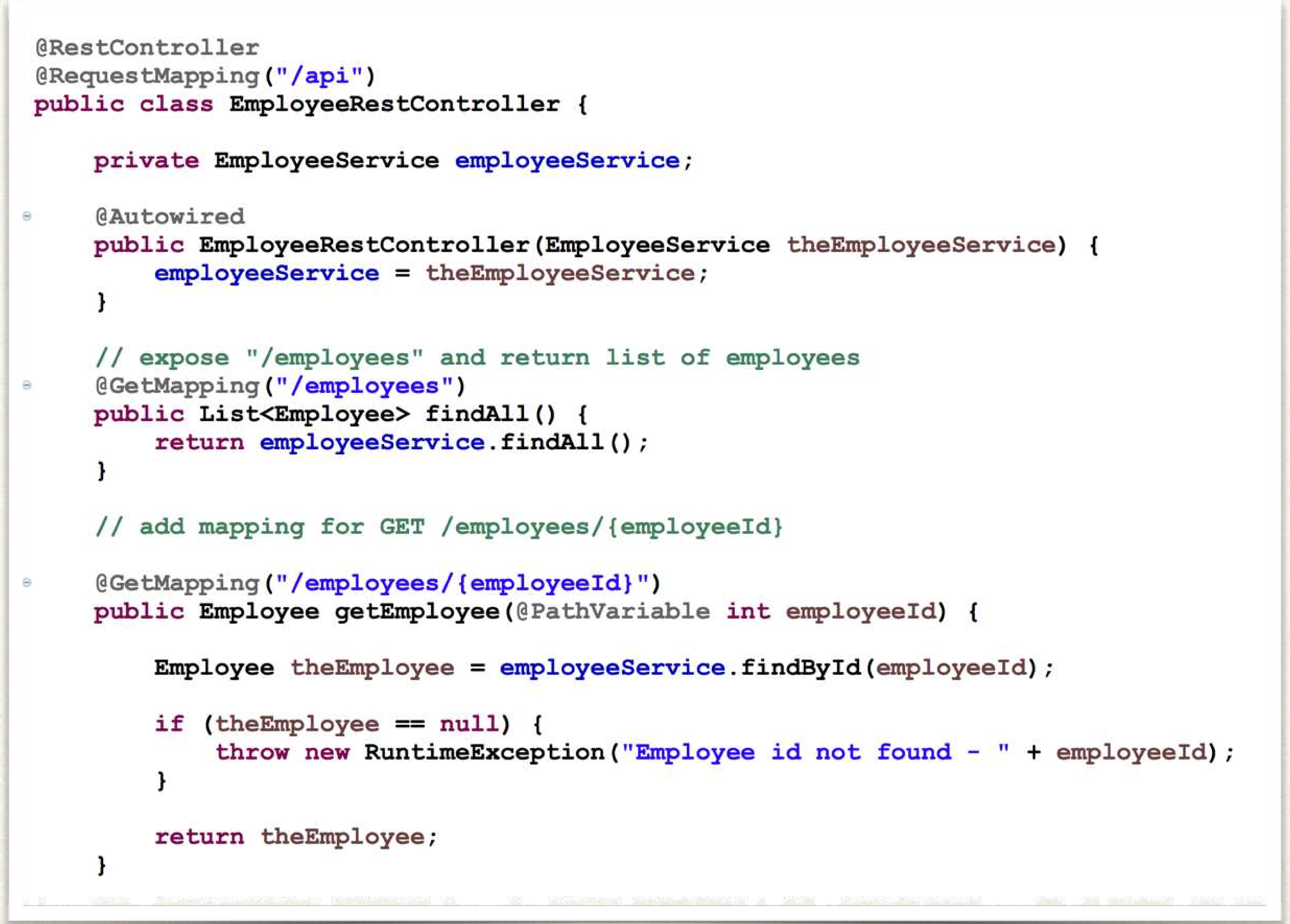
The Problem

The Problem

- We saw how to create a REST API for `Employee`

The Problem

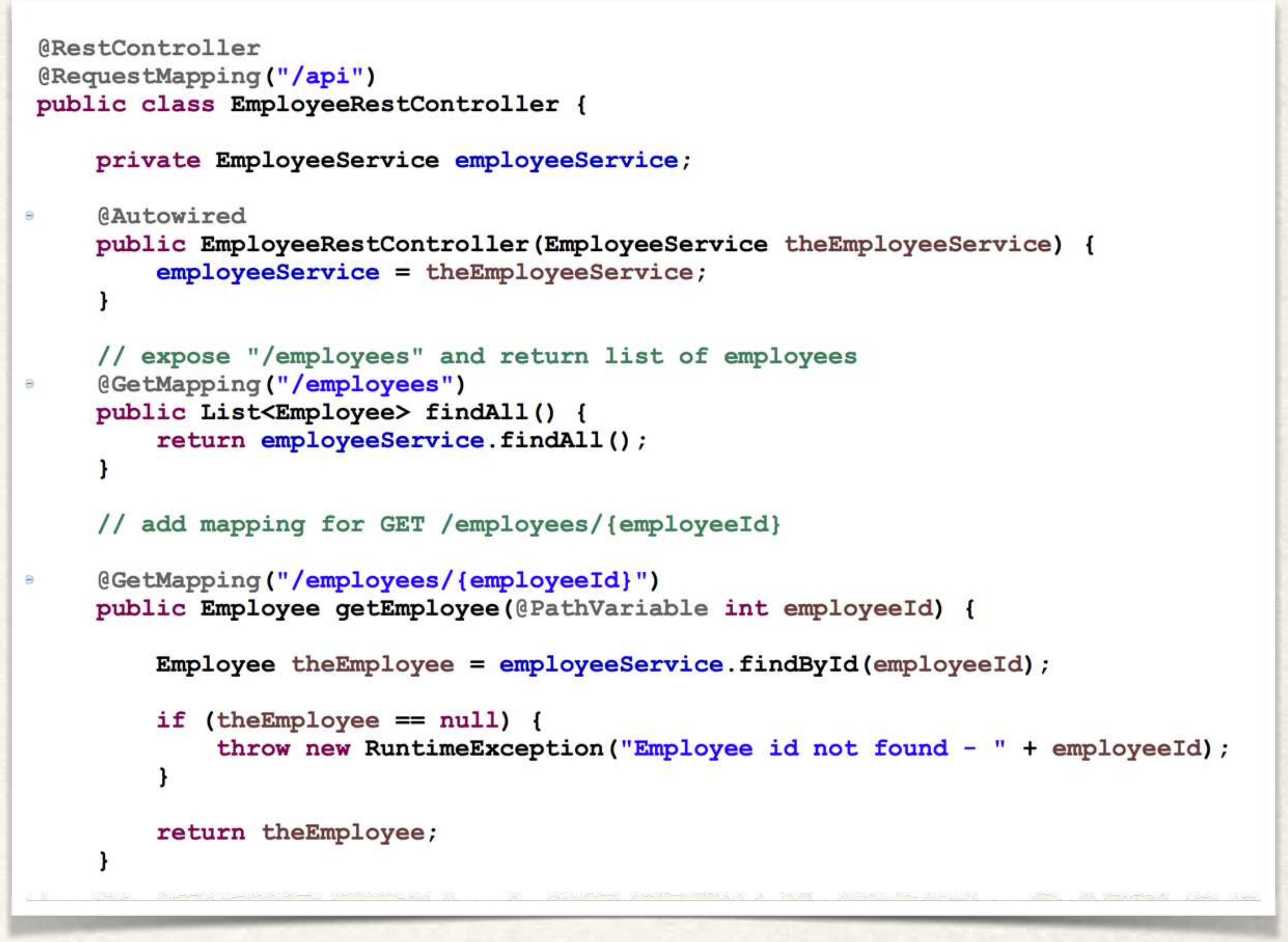
- We saw how to create a REST API for **`Employee`**



```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
        return employeeService.findAll();
    }

    // add mapping for GET /employees/{employeeId}

    @GetMapping("/employees/{employeeId}")
    public Employee getEmployee(@PathVariable int employeeId) {

        Employee theEmployee = employeeService.findById(employeeId);

        if (theEmployee == null) {
            throw new RuntimeException("Employee id not found - " + employeeId);
        }

        return theEmployee;
    }
```

The Problem

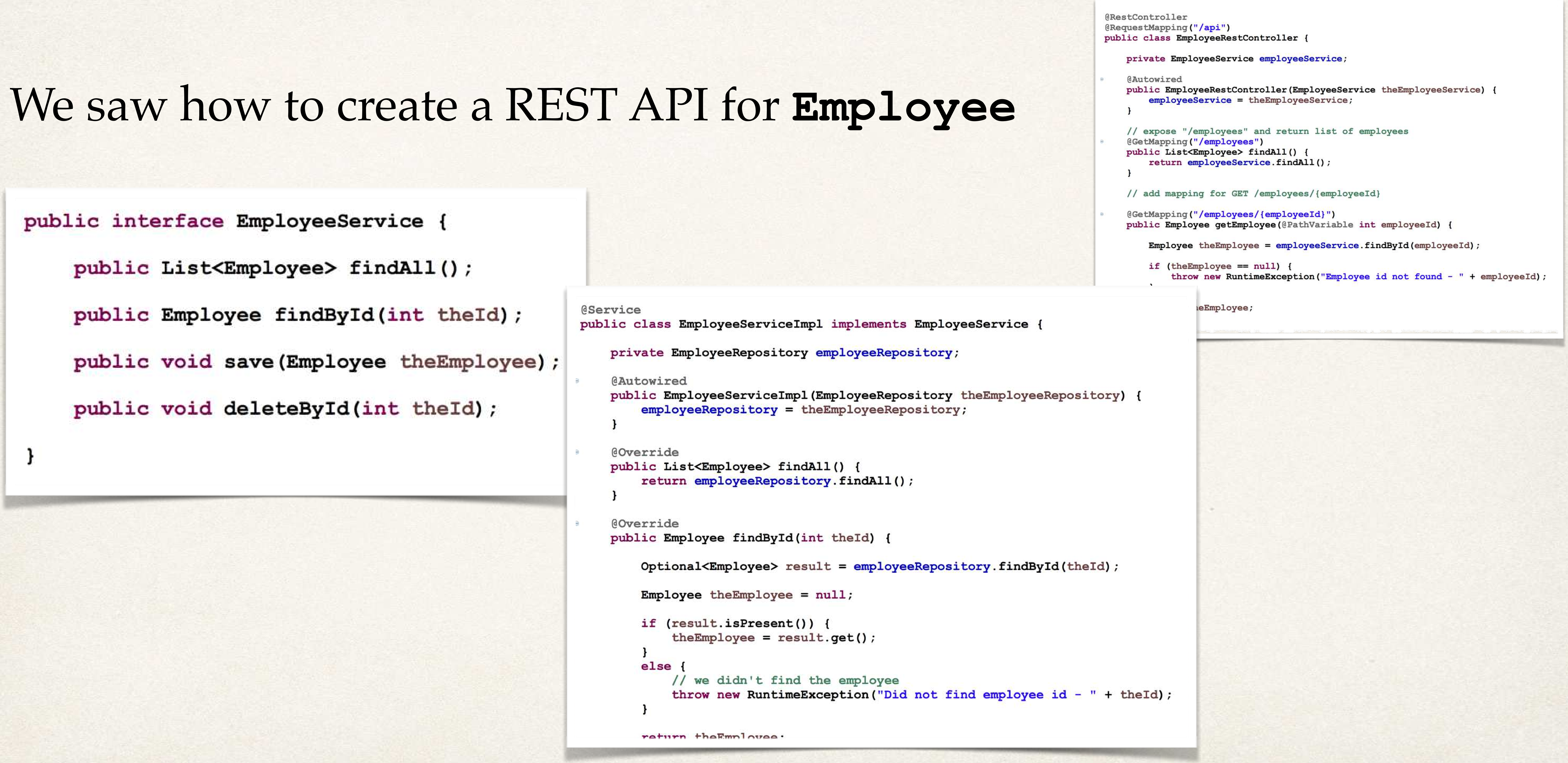
- We saw how to create a REST API for **`Employee`**



```java
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
        return employeeService.findAll();
    }

    // add mapping for GET /employees/{employeeId}

    @GetMapping("/employees/{employeeId}")
    public Employee getEmployee(@PathVariable int employeeId) {

        Employee theEmployee = employeeService.findById(employeeId);

        if (theEmployee == null) {
            throw new RuntimeException("Employee id not found - " + employeeId);
        }

        return theEmployee;
    }
```

The Problem

- We saw how to create a REST API for **`Employee`**



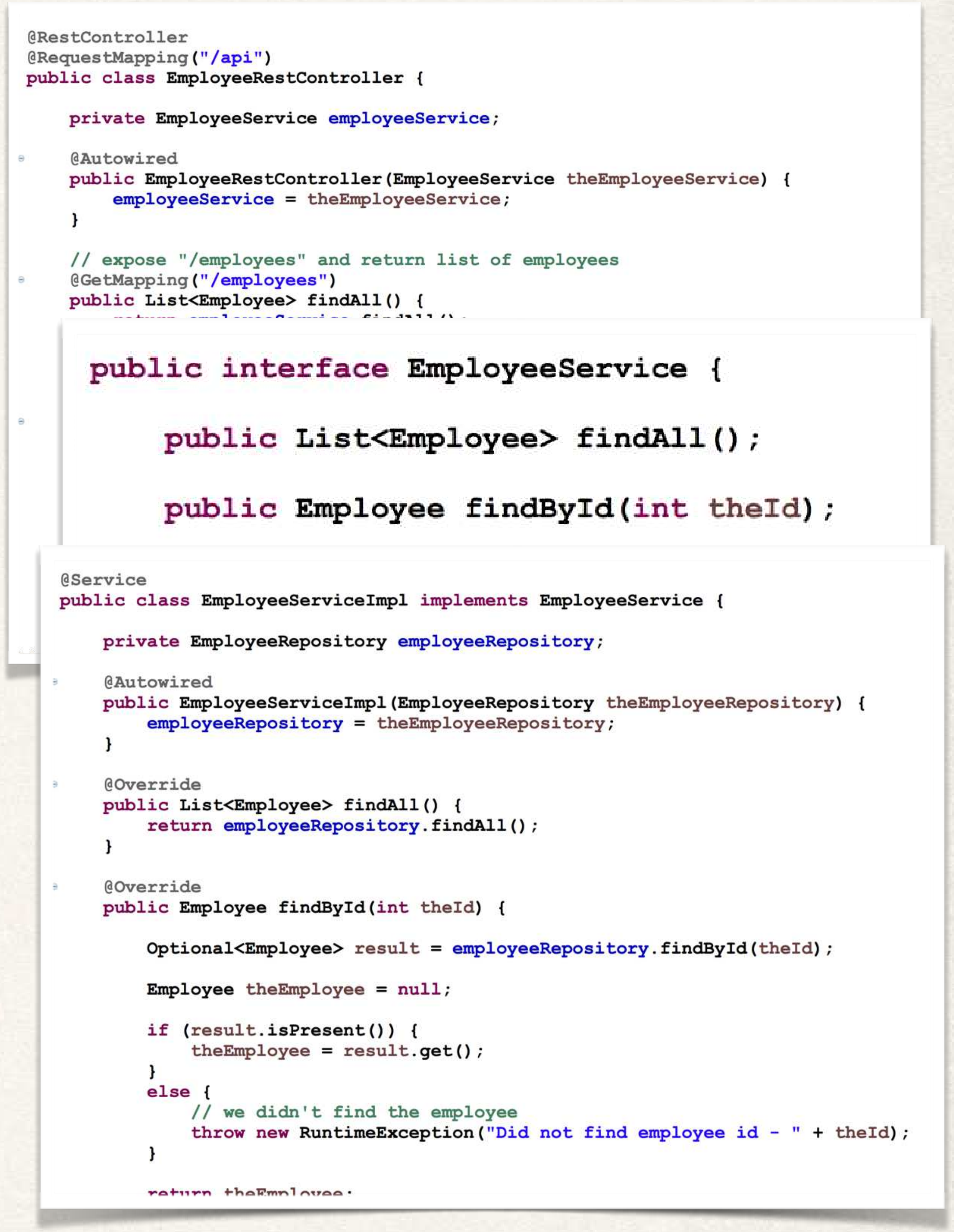
```java
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```java
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

    @Override
    public Employee findById(int theId) {

        Optional<Employee> result = employeeRepository.findById(theId);

        Employee theEmployee = null;

        if (result.isPresent()) {
            theEmployee = result.get();
        }
        else {
            // we didn't find the employee
            throw new RuntimeException("Did not find employee id - " + theId);
        }

        return theEmployee;
```

```java
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
        return employeeService.findAll();
    }

    // add mapping for GET /employees/{employeeId}

    @GetMapping("/employees/{employeeId}")
    public Employee getEmployee(@PathVariable int employeeId) {

        Employee theEmployee = employeeService.findById(employeeId);

        if (theEmployee == null) {
            throw new RuntimeException("Employee id not found - " + employeeId);

                  eEmployee;
```

The Problem

- We saw how to create a REST API for **`Employee`**



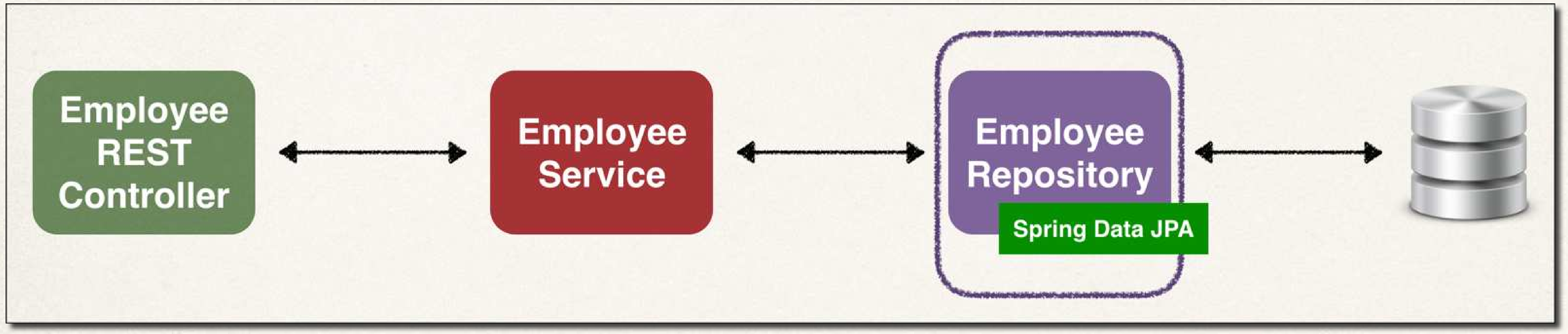
```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
        return employeeService.findAll();
```

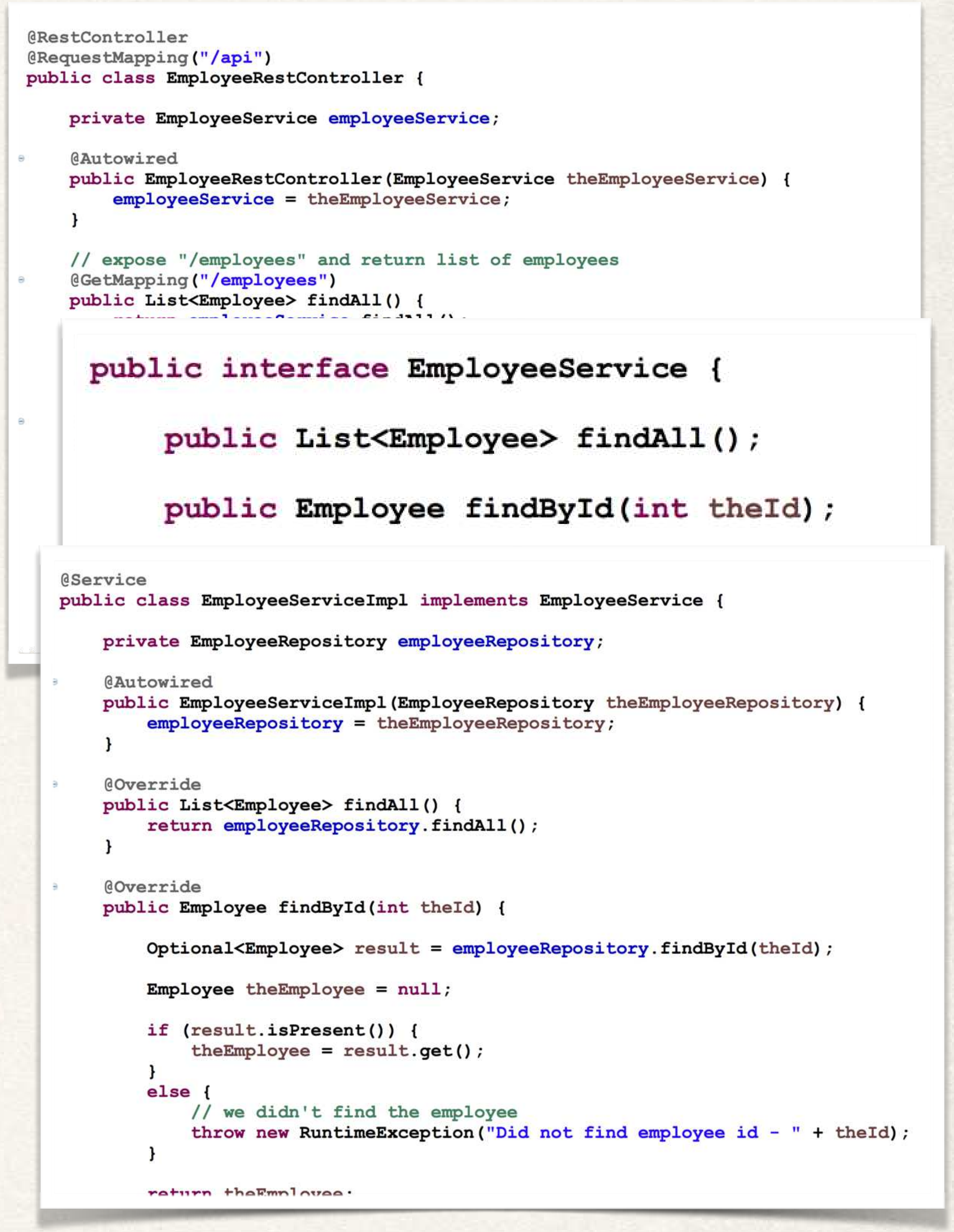
```
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);
```

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

    @Override
    public Employee findById(int theId) {

        Optional<Employee> result = employeeRepository.findById(theId);

        Employee theEmployee = null;

        if (result.isPresent()) {
            theEmployee = result.get();
        }
        else {
            // we didn't find the employee
            throw new RuntimeException("Did not find employee id - " + theId);
        }

        return theEmployee;
```

The Problem

- We saw how to create a REST API for **`Employee`**





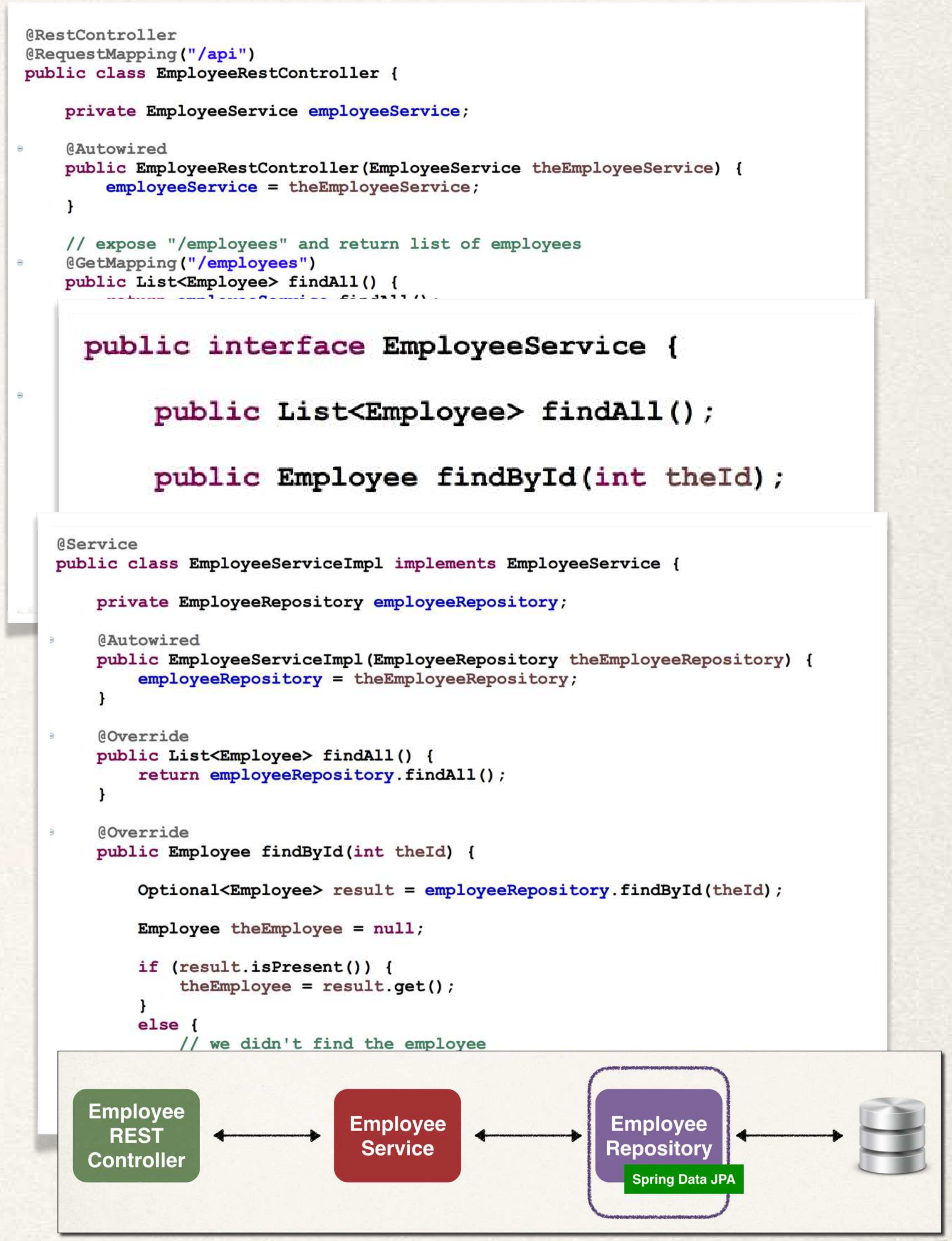
```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
        return employeeService.findAll();
```

```
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);
```

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

    @Override
    public Employee findById(int theId) {

        Optional<Employee> result = employeeRepository.findById(theId);

        Employee theEmployee = null;

        if (result.isPresent()) {
            theEmployee = result.get();
        }
        else {
            // we didn't find the employee
            throw new RuntimeException("Did not find employee id - " + theId);
        }

        return theEmployee;
```

The Problem

- We saw how to create a REST API for **`Employee`**



```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
```

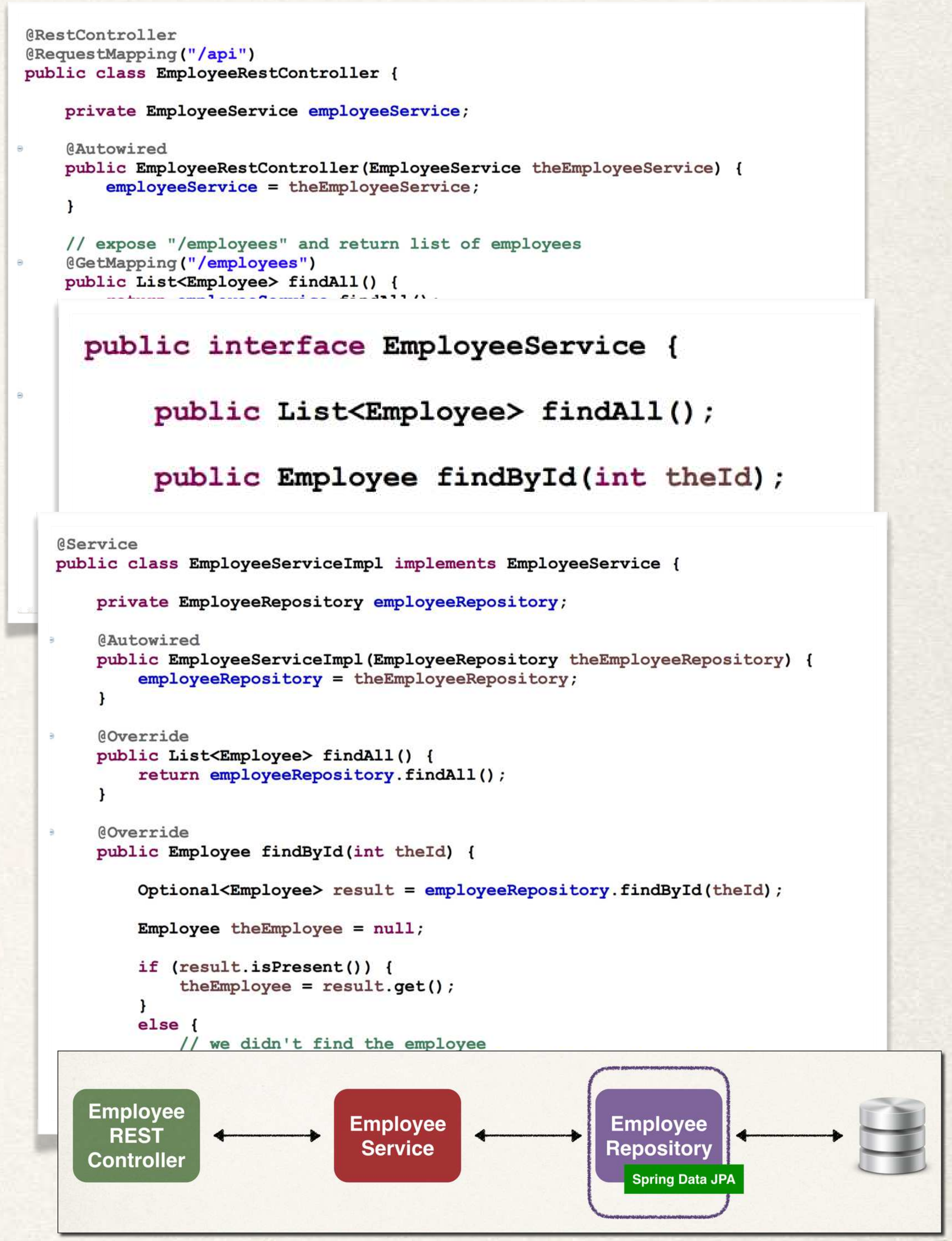
```
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);
```

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

    @Override
    public Employee findById(int theId) {

        Optional<Employee> result = employeeRepository.findById(theId);

        Employee theEmployee = null;

        if (result.isPresent()) {
            theEmployee = result.get();
        }
        else {
            // we didn't find the employee
```

The Problem

- We saw how to create a REST API for **`Employee`**

- Need to create REST API for another entity?



```java
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
```

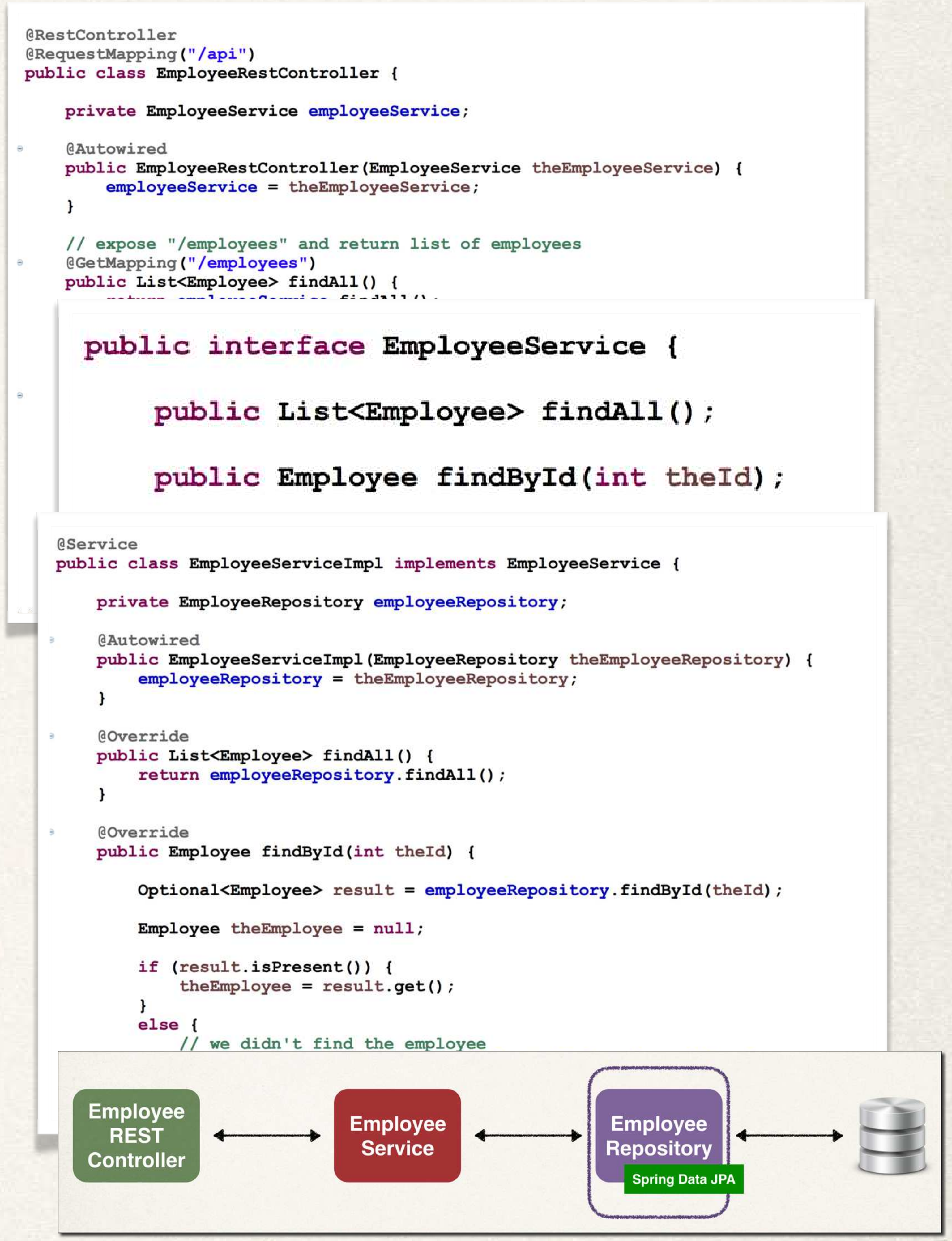
```java
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);
```

```java
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

    @Override
    public Employee findById(int theId) {

        Optional<Employee> result = employeeRepository.findById(theId);

        Employee theEmployee = null;

        if (result.isPresent()) {
            theEmployee = result.get();
        }
        else {
            // we didn't find the employee
```

The Problem

- We saw how to create a REST API for **`Employee`**

- Need to create REST API for another entity?
  - **`Customer`**, **`Student`**, **`Product`**, **`Book`** …



```java
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
```

```java
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);
```

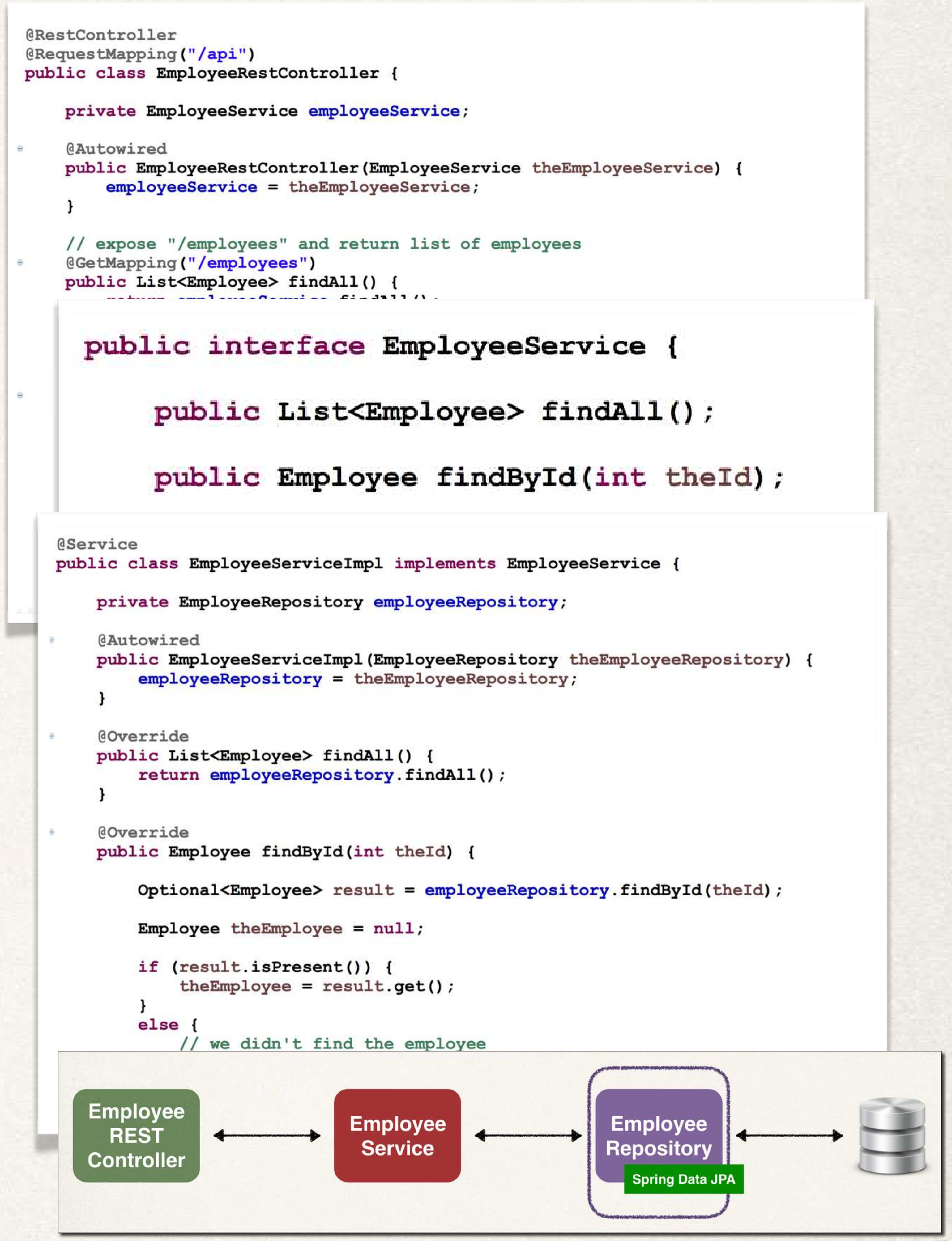
```java
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

    @Override
    public Employee findById(int theId) {

        Optional<Employee> result = employeeRepository.findById(theId);

        Employee theEmployee = null;

        if (result.isPresent()) {
            theEmployee = result.get();
        }
        else {
            // we didn't find the employee
```

The Problem

- We saw how to create a REST API for **`Employee`**

- Need to create REST API for another entity?
  - **`Customer`**, **`Student`**, **`Product`**, **`Book`** …

- Do we have to repeat all of the same code again???



```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
```

```
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);
```

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

    @Override
    public Employee findById(int theId) {

        Optional<Employee> result = employeeRepository.findById(theId);

        Employee theEmployee = null;

        if (result.isPresent()) {
            theEmployee = result.get();
        }
        else {
            // we didn't find the employee
```

My Wish

My Wish

- I wish we could tell Spring:

My Wish

- I wish we could tell Spring:

Create a REST API for me

My Wish

- I wish we could tell Spring:

Create a REST API for me

Use my existing JpaRepository (entity, primary key)

My Wish

- I wish we could tell Spring:

Create a REST API for me

Use my existing JpaRepository (entity, primary key)

Give me all of the basic REST API CRUD features for free

Spring Data REST - Solution

Spring Data REST - Solution

- **Spring Data REST** is the solution!!!!

Spring Data REST - Solution

- **Spring Data REST** is the solution!!!!

https://spring.io/projects/spring-data-rest

Spring Data REST - Solution

- **Spring Data REST** is the solution!!!!

https://spring.io/projects/spring-data-rest

- Leverages your existing **`JpaRepository`**

Spring Data REST - Solution

- **Spring Data REST** is the solution!!!!

https://spring.io/projects/spring-data-rest

- Leverages your existing **`JpaRepository`**

- Spring will give you a REST CRUD implementation for FREE …. like MAGIC!!

Spring Data REST - Solution

- **Spring Data REST** is the solution!!!!

https://spring.io/projects/spring-data-rest

- Leverages your existing **`JpaRepository`**

- Spring will give you a REST CRUD implementation for FREE …. like MAGIC!!

Spring Data REST - Solution

- **Spring Data REST** is the solution!!!!

https://spring.io/projects/spring-data-rest

- Leverages your existing **`JpaRepository`**

- Spring will give you a REST CRUD implementation for FREE …. like MAGIC!!
  - Helps to minimize boiler-plate REST code!!!

Spring Data REST - Solution

- **Spring Data REST** is the solution!!!!

https://spring.io/projects/spring-data-rest

- Leverages your existing **`JpaRepository`**

- Spring will give you a REST CRUD implementation for FREE …. like MAGIC!!
  - Helps to minimize boiler-plate REST code!!!
  - No new coding required!!!

REST API

REST API

- Spring Data REST will expose these endpoints for free!

REST API

- Spring Data REST will expose these endpoints for free!

HTTP Method		CRUD Action
POST	/employees	Create a new employee
GET	/employees	Read a list of employees
GET	/employees/{employeeId}	Read a single employee
PUT	/employees/{employeeId}	Update an existing employee
DELETE	/employees/{employeeId}	Delete an existing employee

REST API

- Spring Data REST will expose these endpoints for free!

HTTP Method		CRUD Action
`POST`	`/employees`	`Create a new employee`
`GET`	`/employees`	`Read a list of employees`
`GET`	`/employees/{employeeId}`	`Read a single employee`
`PUT`	`/employees/{employeeId}`	`Update an existing employee`
`DELETE`	`/employees/{employeeId}`	`Delete an existing employee`

Get these REST endpoints for free

Spring Data REST - How Does It Work?

Spring Data REST - How Does It Work?

- Spring Data REST will scan your project for **`JpaRepository`**

Spring Data REST - How Does It Work?

- Spring Data REST will scan your project for **`JpaRepository`**
- Expose REST APIs for each entity type for your **`JpaRepository`**

Spring Data REST - How Does It Work?

- Spring Data REST will scan your project for **`JpaRepository`**
- Expose REST APIs for each entity type for your **`JpaRepository`**

```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

REST Endpoints

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form

REST Endpoints

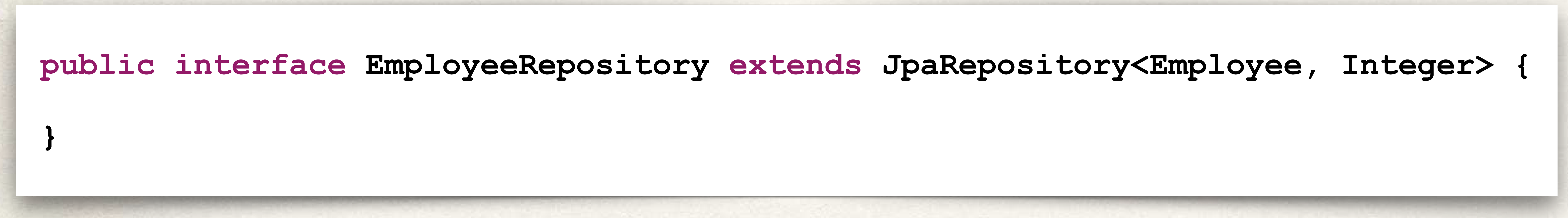
- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type
- <u>Simple</u> pluralized form
  - First character of Entity type is lowercase
  - Then just adds an "s" to the entity

REST Endpoints

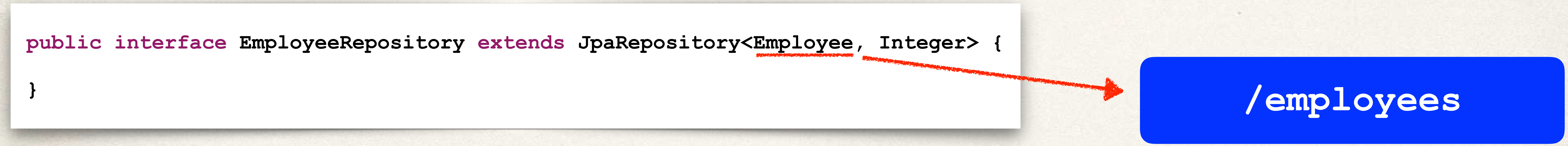
- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase
  - Then just adds an "s" to the entity



```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase
  - Then just adds an "s" to the entity



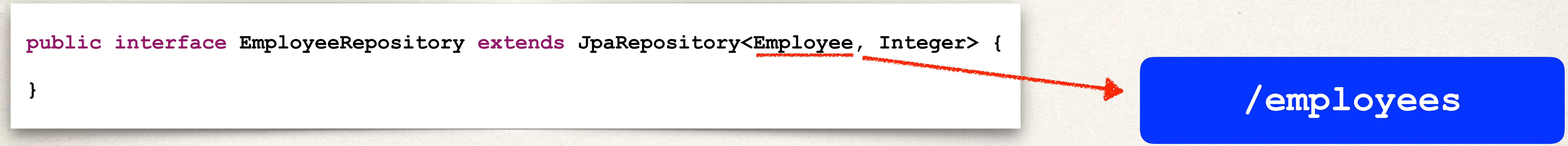
```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

/employees

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase
  - Then just adds an "s" to the entity

More on plural forms in later video



```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

Development Process

Step-By-Step

Development Process

Step-By-Step

1. Add Spring Data REST to your Maven POM file

Development Process

Step-By-Step

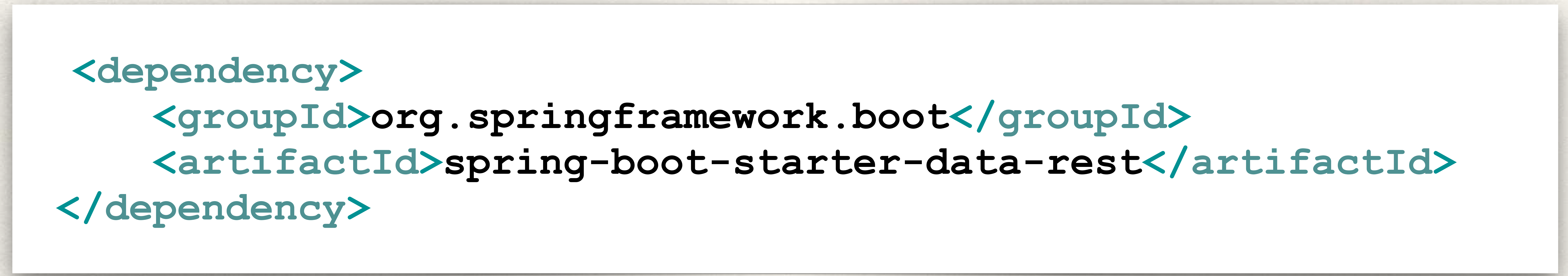
1. Add Spring Data REST to your Maven POM file

That's it!!!

Absolutely NO CODING required

Step 1: Add Spring Data REST to POM file

Step 1: Add Spring Data REST to POM file



```
 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

Step 1: Add Spring Data REST to POM file

```
 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

That's it!!!

Absolutely NO CODING required

Step 1: Add Spring Data REST to POM file

```
 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

That's it!!!

Absolutely NO CODING required

Spring Data REST will scan for JpaRepository

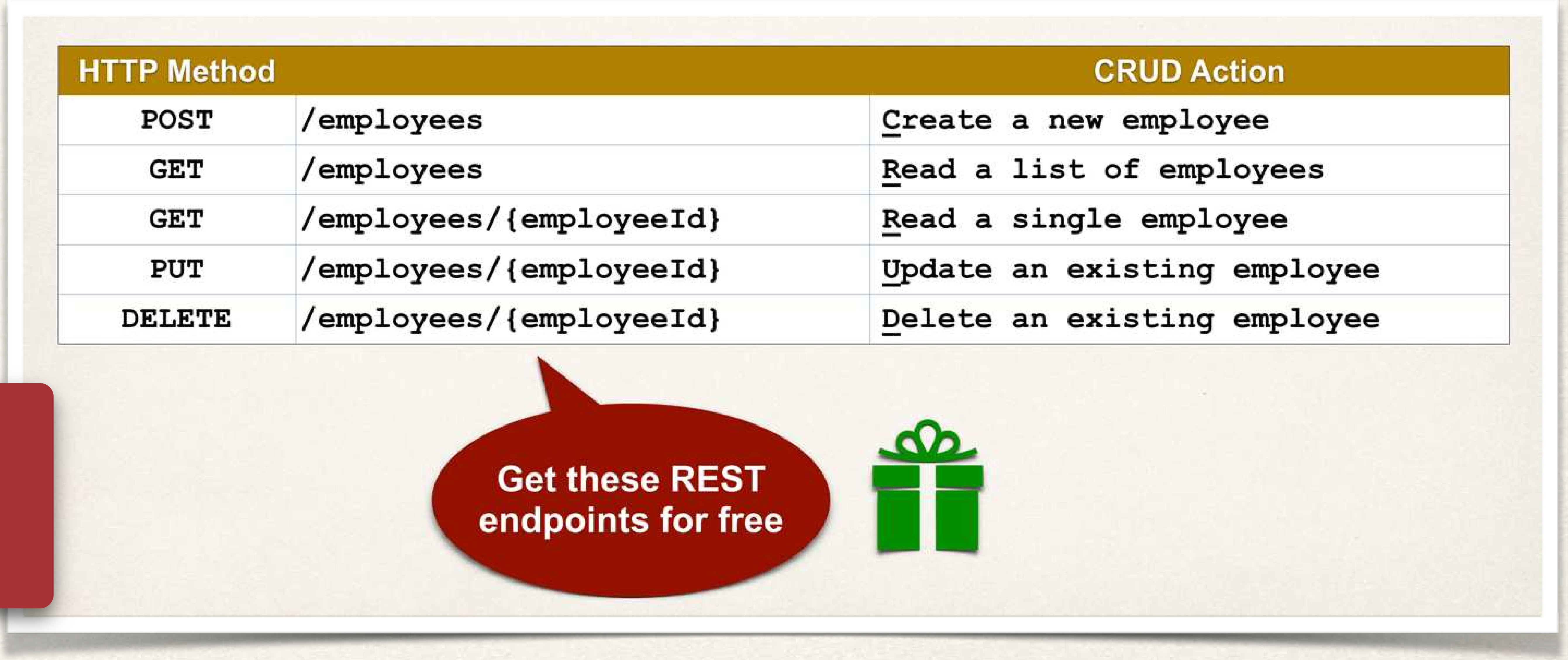
Step 1: Add Spring Data REST to POM file

```
 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

That's it!!!

Absolutely NO CODING required

Spring Data REST will scan for JpaRepository



HTTP Method		CRUD Action
POST	/employees	Create a new employee
GET	/employees	Read a list of employees
GET	/employees/{employeeId}	Read a single employee
PUT	/employees/{employeeId}	Update an existing employee
DELETE	/employees/{employeeId}	Delete an existing employee

Get these REST endpoints for free

In A Nutshell

In A Nutshell

For Spring Data REST, you only need 3 items

In A Nutshell

For Spring Data REST, you only need 3 items

1. Your entity: **`Employee`**

In A Nutshell

For Spring Data REST, you only need 3 items

1. Your entity: `Employee`
2. JpaRepository: `EmployeeRepository extends JpaRepository`

In A Nutshell

For Spring Data REST, you only need 3 items

1. Your entity: **`Employee`**
2. JpaRepository: **`EmployeeRepository extends JpaRepository`**
3. Maven POM dependency for: **`spring-boot-starter-data-rest`**

In A Nutshell

For Spring Data REST, you only need 3 items

We already have these two

1. Your entity: **`Employee`**
2. JpaRepository: **`EmployeeRepository extends JpaRepository`**
3. Maven POM dependency for: **`spring-boot-starter-data-rest`**

In A Nutshell

For Spring Data REST, you only need 3 items

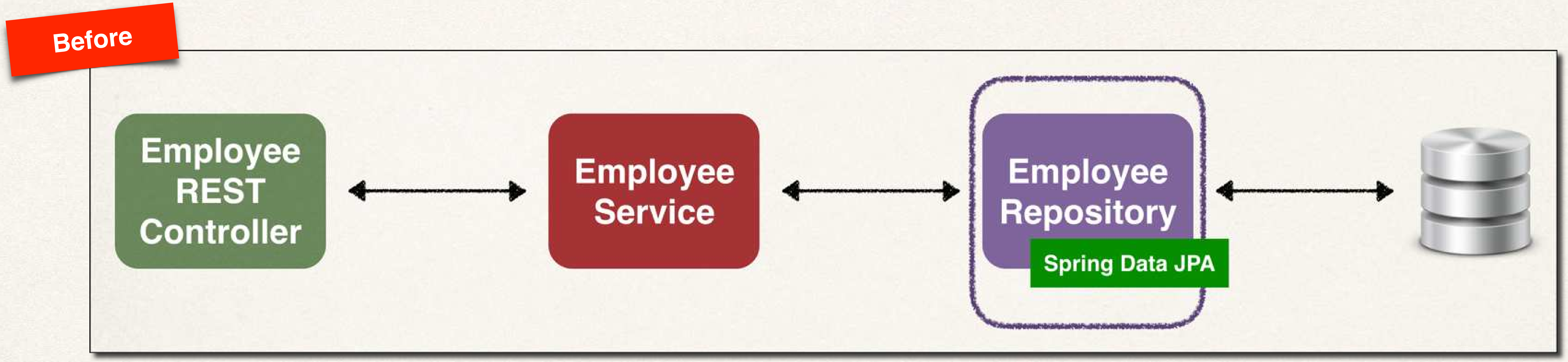
We already have these two

1. Your entity: **`Employee`**
2. JpaRepository: **`EmployeeRepository extends JpaRepository`**
3. Maven POM dependency for: **`spring-boot-starter-data-rest`**

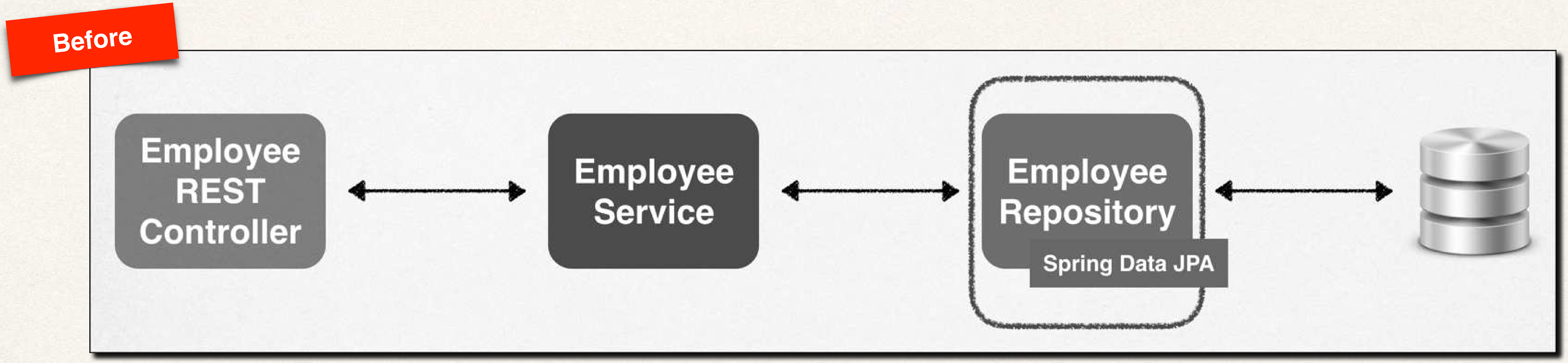
Only item that is new

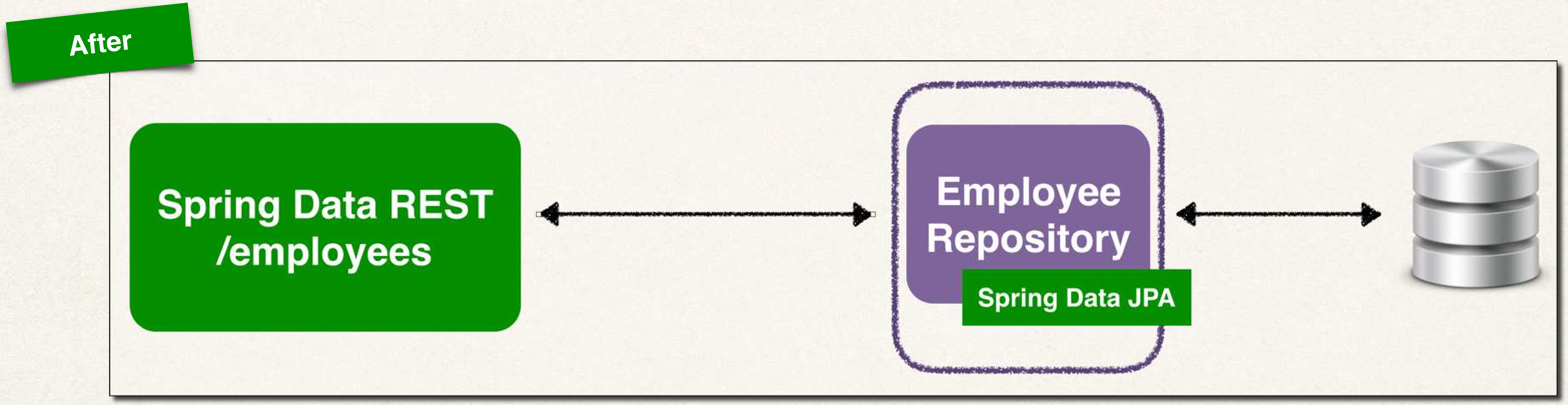
Application Architecture

Application Architecture



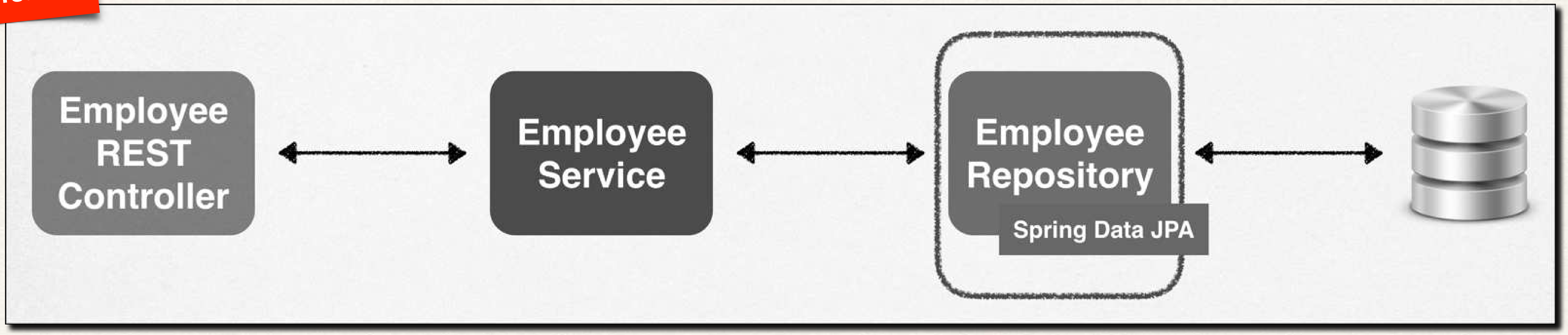
Application Architecture



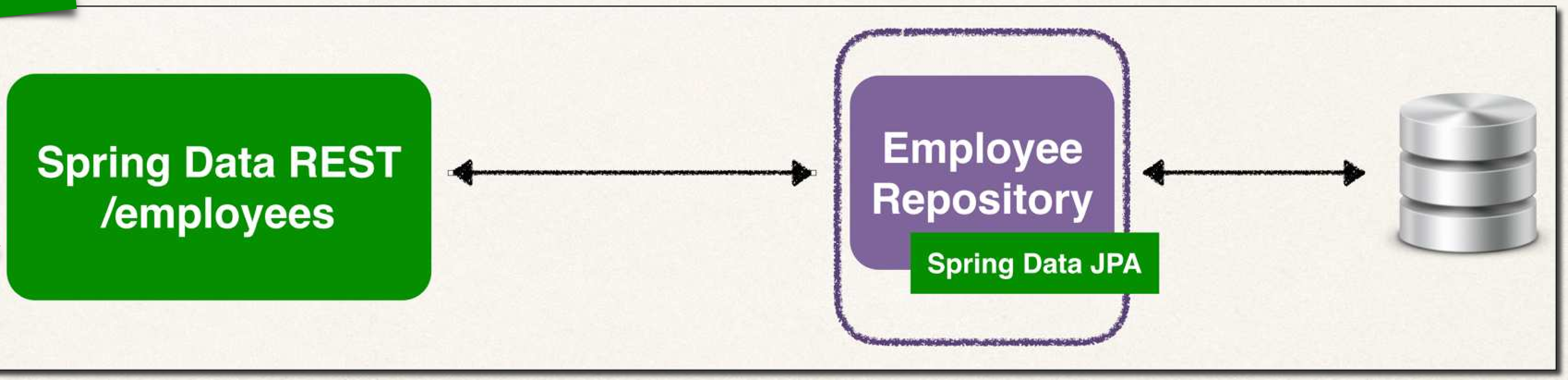


Application Architecture

Before

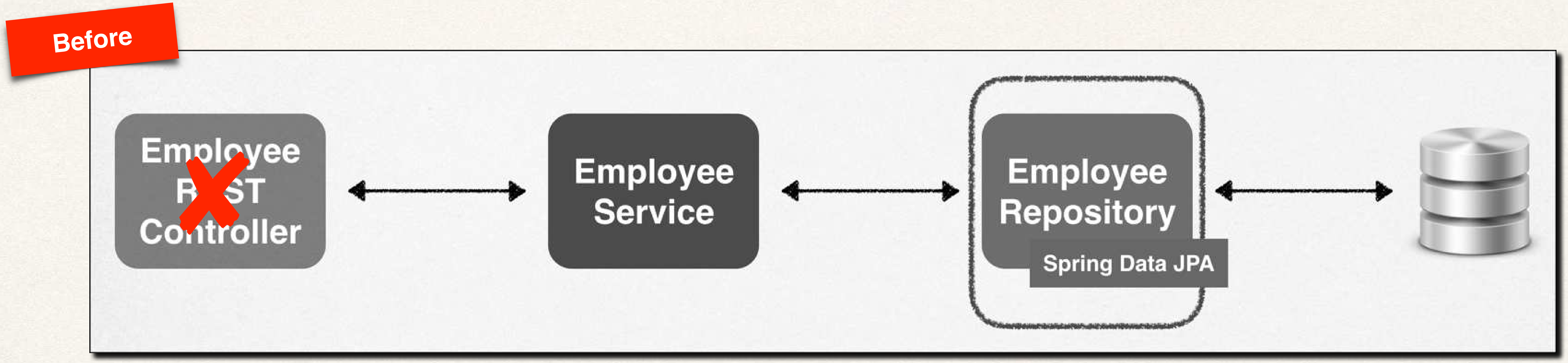


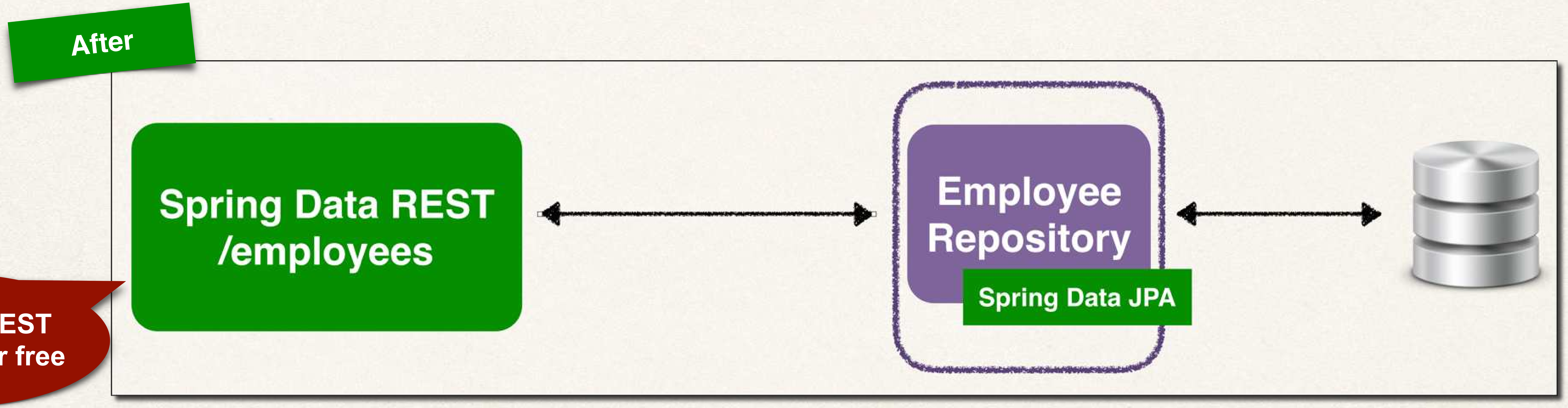
After



Get these REST endpoints for free

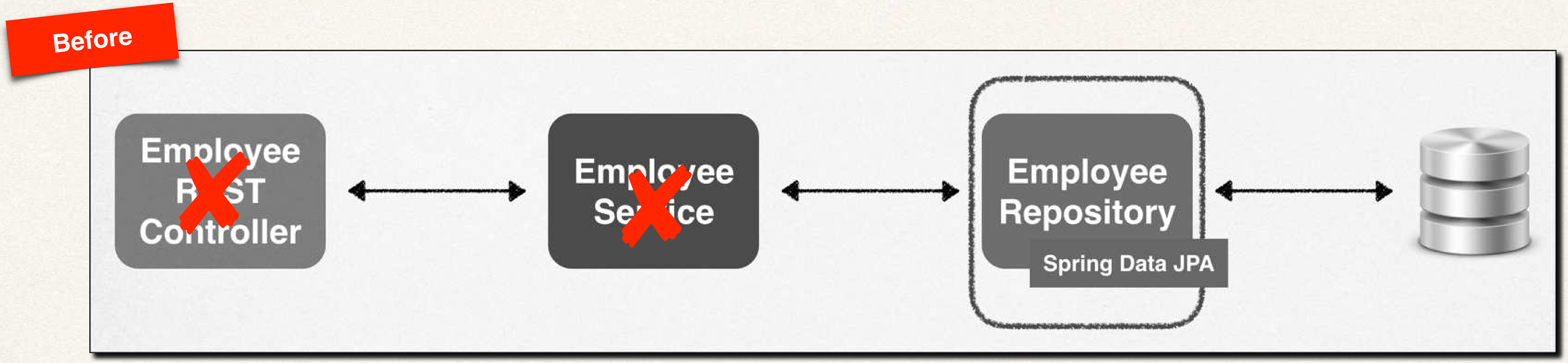
Application Architecture

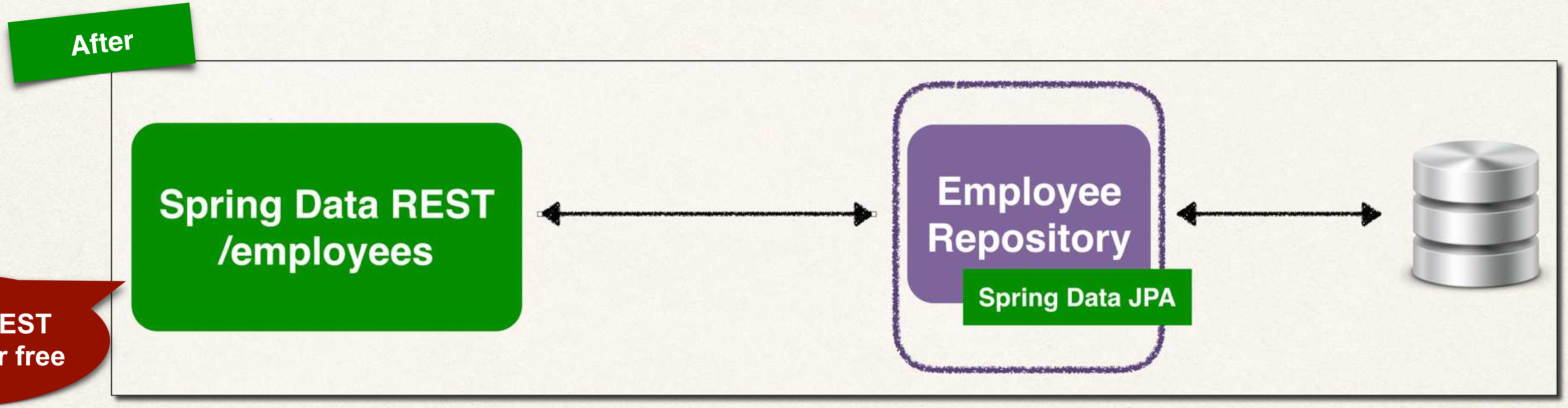




Get these REST endpoints for free

Application Architecture





Get these REST endpoints for free

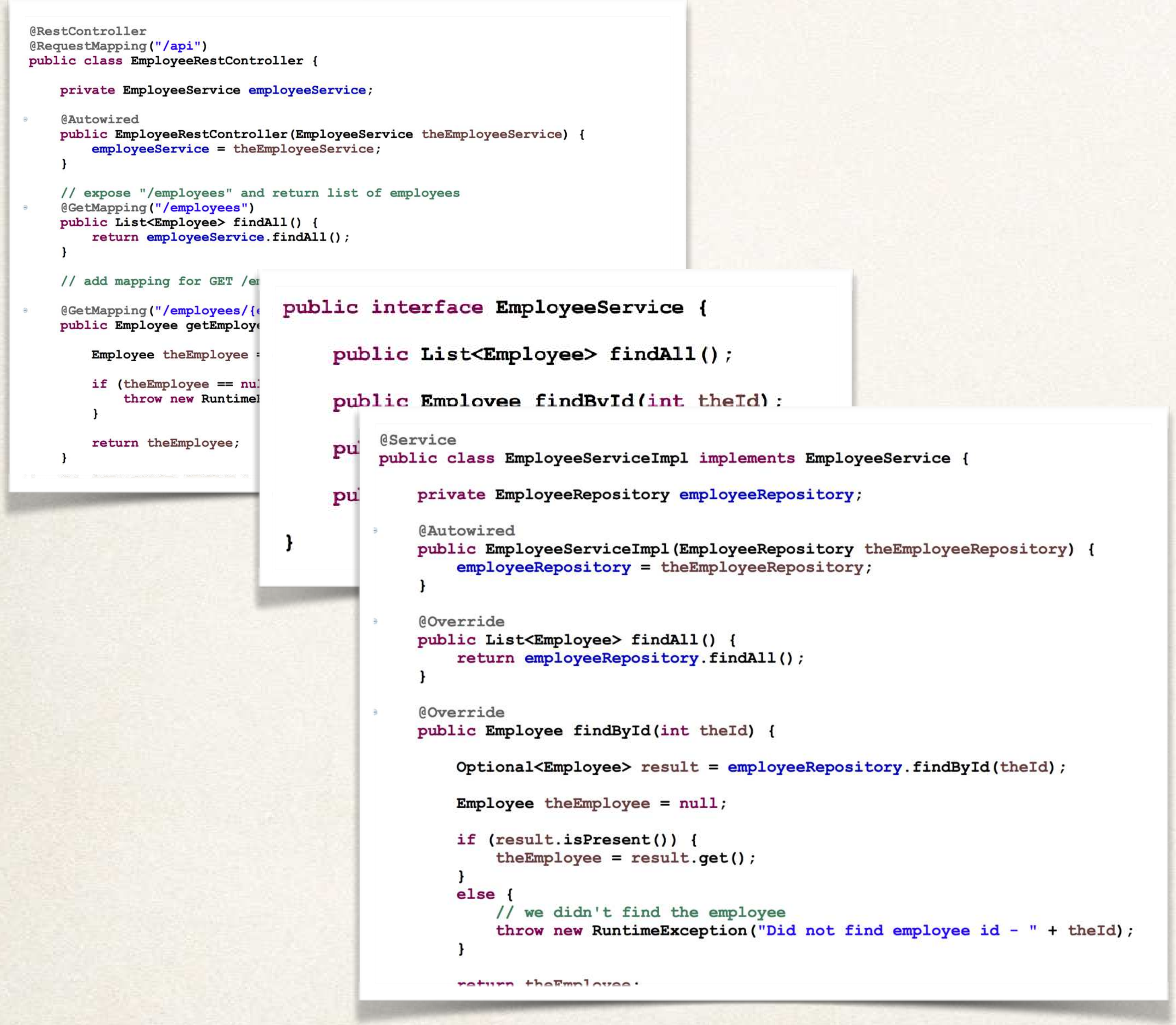
Minimized Boilerplate Code

Minimized Boilerplate Code

Before Spring Data REST

Minimized Boilerplate Code

Before Spring Data REST



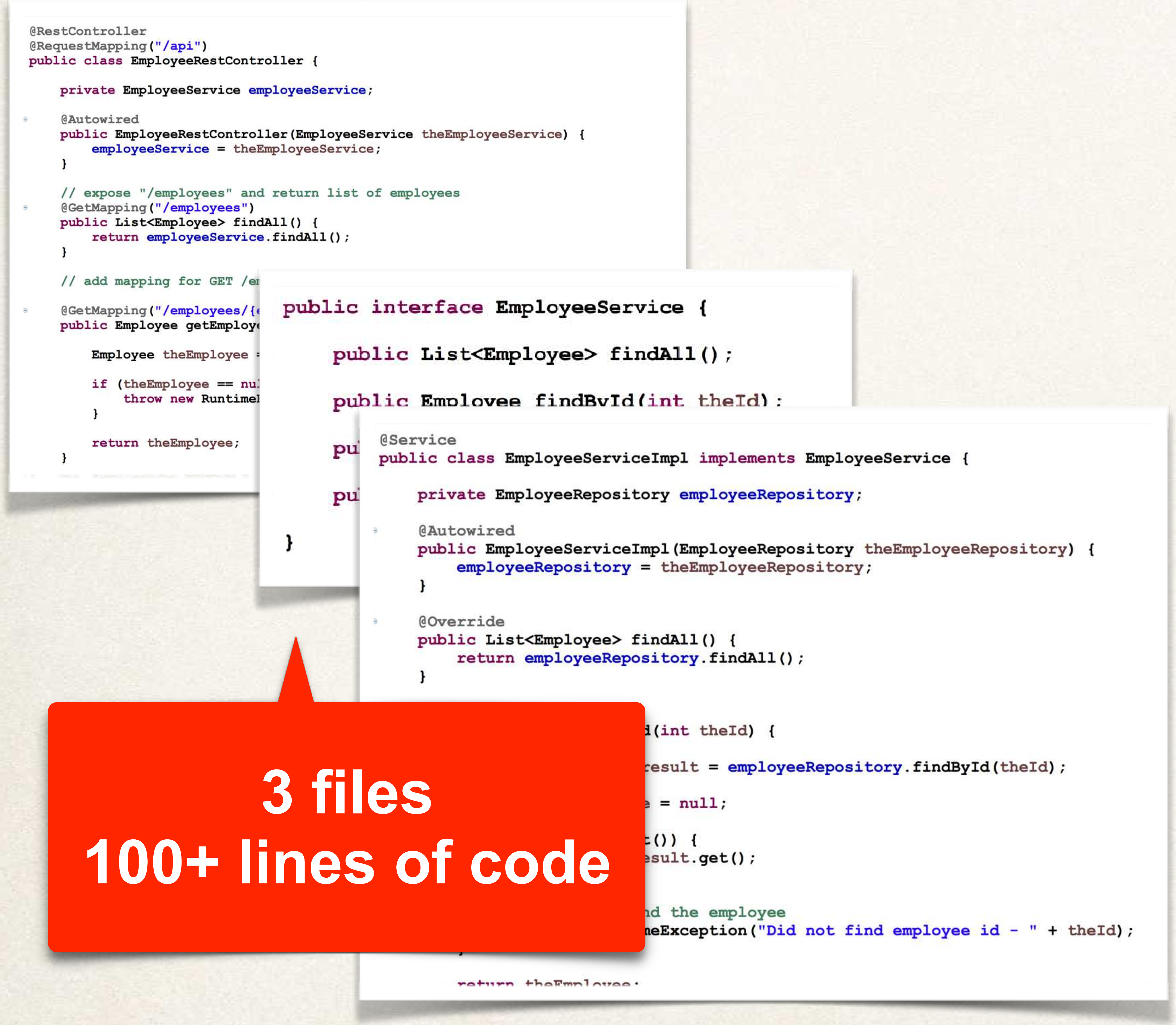
```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
        return employeeService.findAll();
    }

    // add mapping for GET /er

    @GetMapping("/employees/{
    public Employee getEmploy

        Employee theEmployee =

        if (theEmployee == nul
            throw new Runtime
        }

        return theEmployee;
    }
```

```
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);

    pu

    pu

}
```

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

    @Override
    public Employee findById(int theId) {

        Optional<Employee> result = employeeRepository.findById(theId);

        Employee theEmployee = null;

        if (result.isPresent()) {
            theEmployee = result.get();
        }
        else {
            // we didn't find the employee
            throw new RuntimeException("Did not find employee id - " + theId);
        }

        return theEmployee;
```

Minimized Boilerplate Code

Before Spring Data REST



```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
        return employeeService.findAll();
    }

    // add mapping for GET /en

    @GetMapping("/employees/{e
    public Employee getEmploye

        Employee theEmployee =

        if (theEmployee == nul
            throw new RuntimeE
        }

        return theEmployee;
    }
```

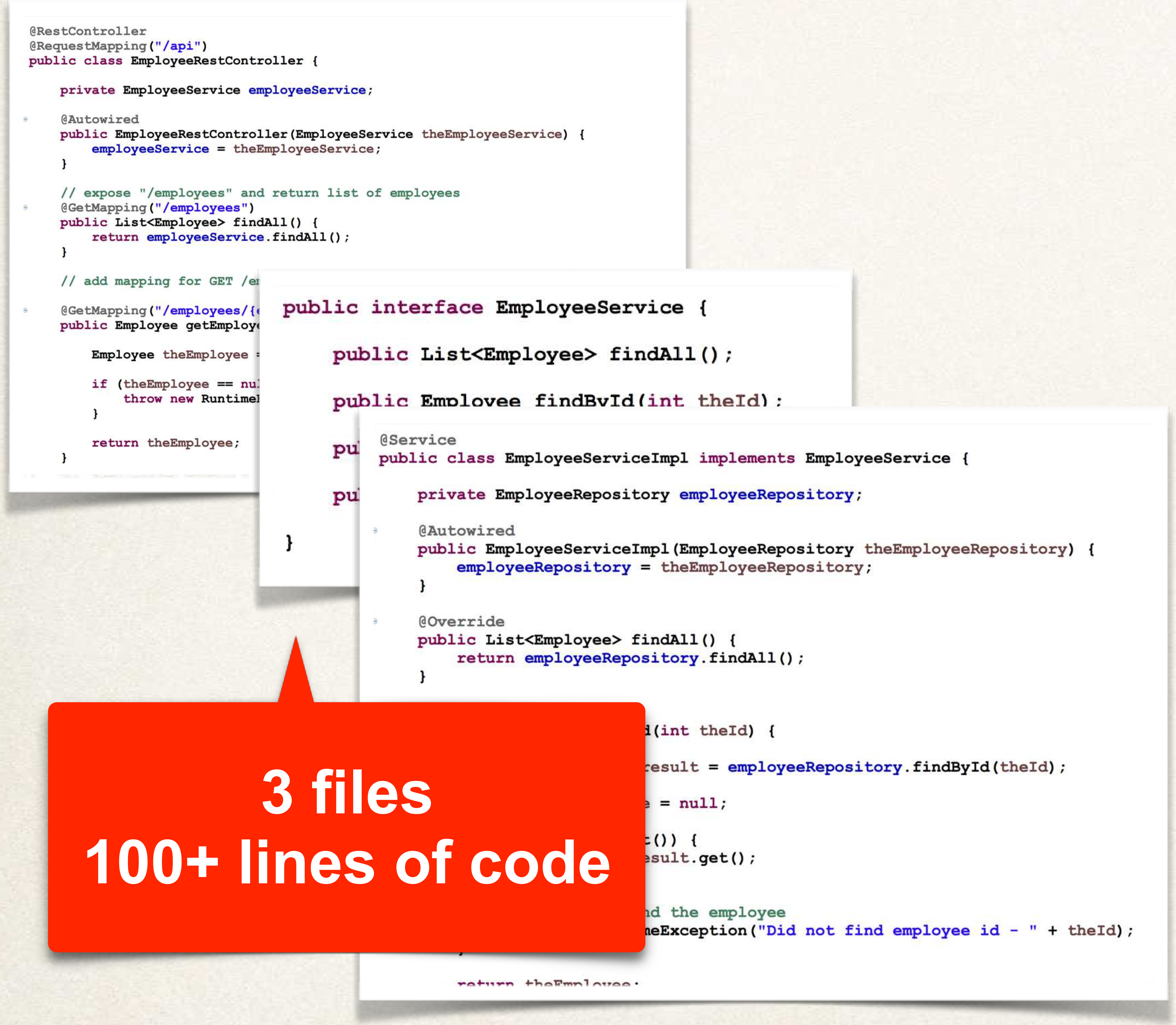
```
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);

    pu

    pu

}
```

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

                        d(int theId) {

                        result = employeeRepository.findById(theId);

                        = null;

                        ()) {
                        esult.get();

                        d the employee
                        meException("Did not find employee id - " + theId);

        return theEmployee;
```

3 files
100+ lines of code

Minimized Boilerplate Code

Before Spring Data REST



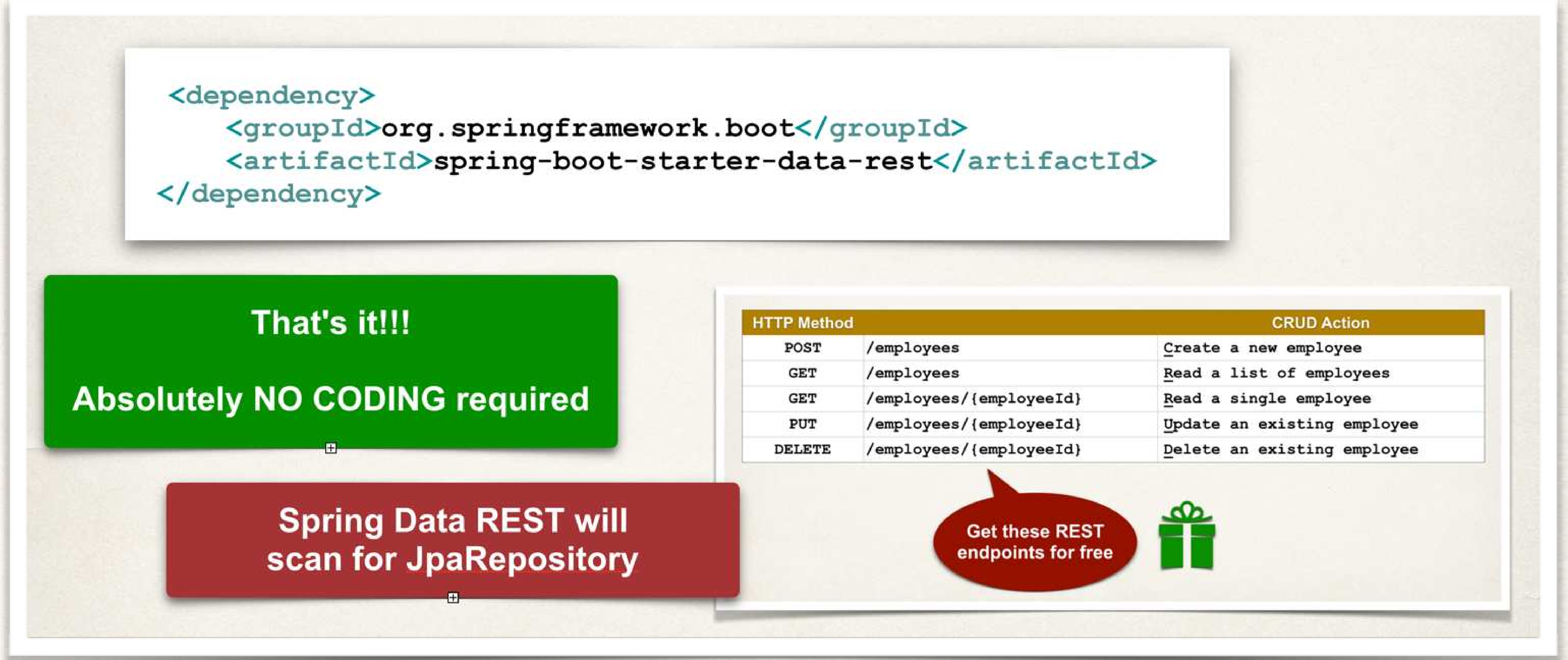
```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeService theEmployeeService) {
        employeeService = theEmployeeService;
    }

    // expose "/employees" and return list of employees
    @GetMapping("/employees")
    public List<Employee> findAll() {
        return employeeService.findAll();
    }

    // add mapping for GET /em

    @GetMapping("/employees/{e
    public Employee getEmploye

        Employee theEmployee =

        if (theEmployee == nul
            throw new RuntimeE
        }

        return theEmployee;
    }
```

```
public interface EmployeeService {

    public List<Employee> findAll();

    public Employee findById(int theId);

    pu

    pu

}
```

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

    private EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeServiceImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

                  d(int theId) {

                  esult = employeeRepository.findById(theId);

                  e = null;

                  ()) {
                  esult.get();

                  d the employee
                  meException("Did not find employee id - " + theId);

        return theEmployee;
```

3 files
100+ lines of code

After Spring Data REST



```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

That's it!!!

Absolutely NO CODING required

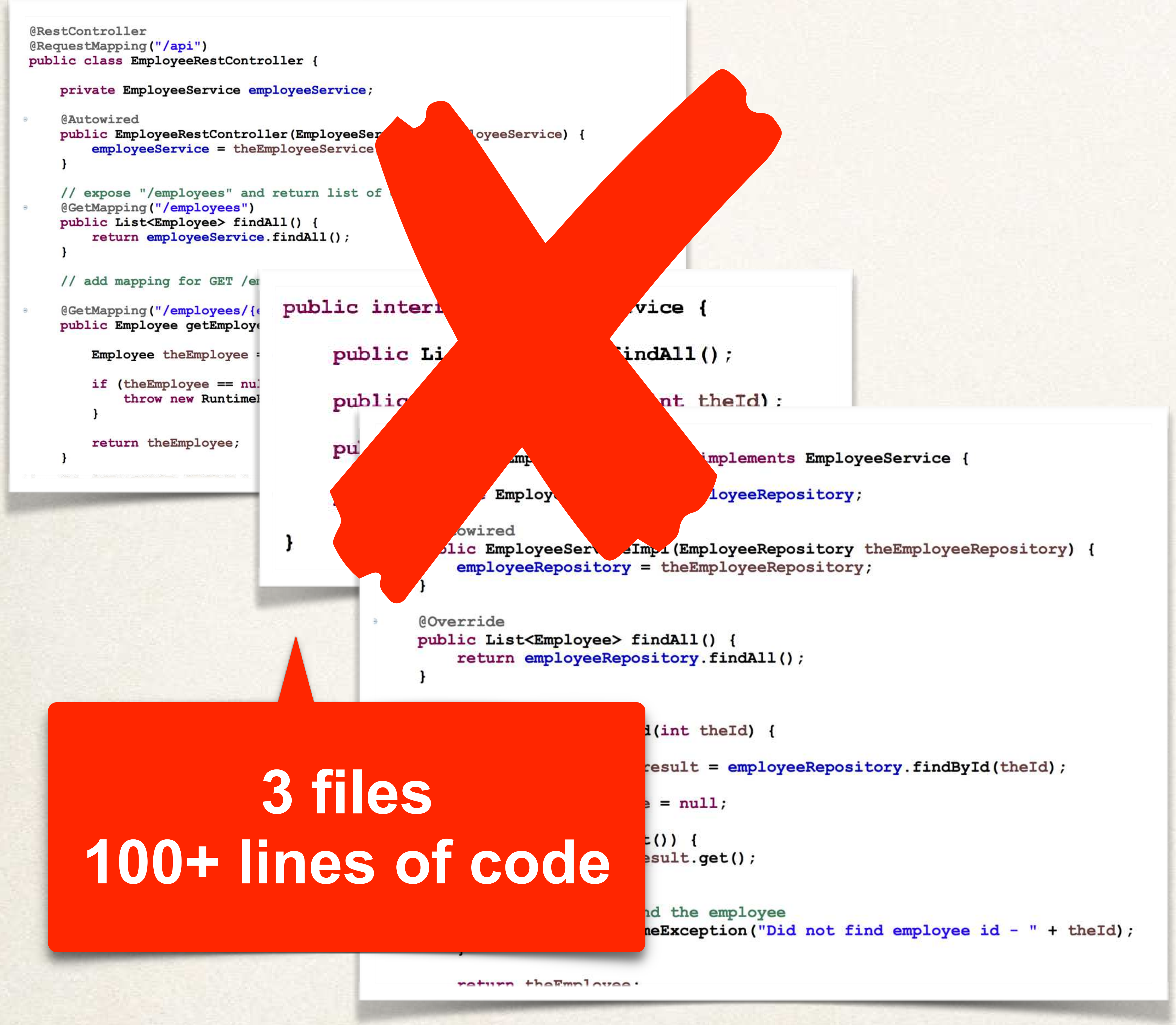
Spring Data REST will scan for JpaRepository

HTTP Method		CRUD Action
POST	/employees	Create a new employee
GET	/employees	Read a list of employees
GET	/employees/{employeeId}	Read a single employee
PUT	/employees/{employeeId}	Update an existing employee
DELETE	/employees/{employeeId}	Delete an existing employee

Get these REST endpoints for free

Minimized Boilerplate Code

Before Spring Data REST



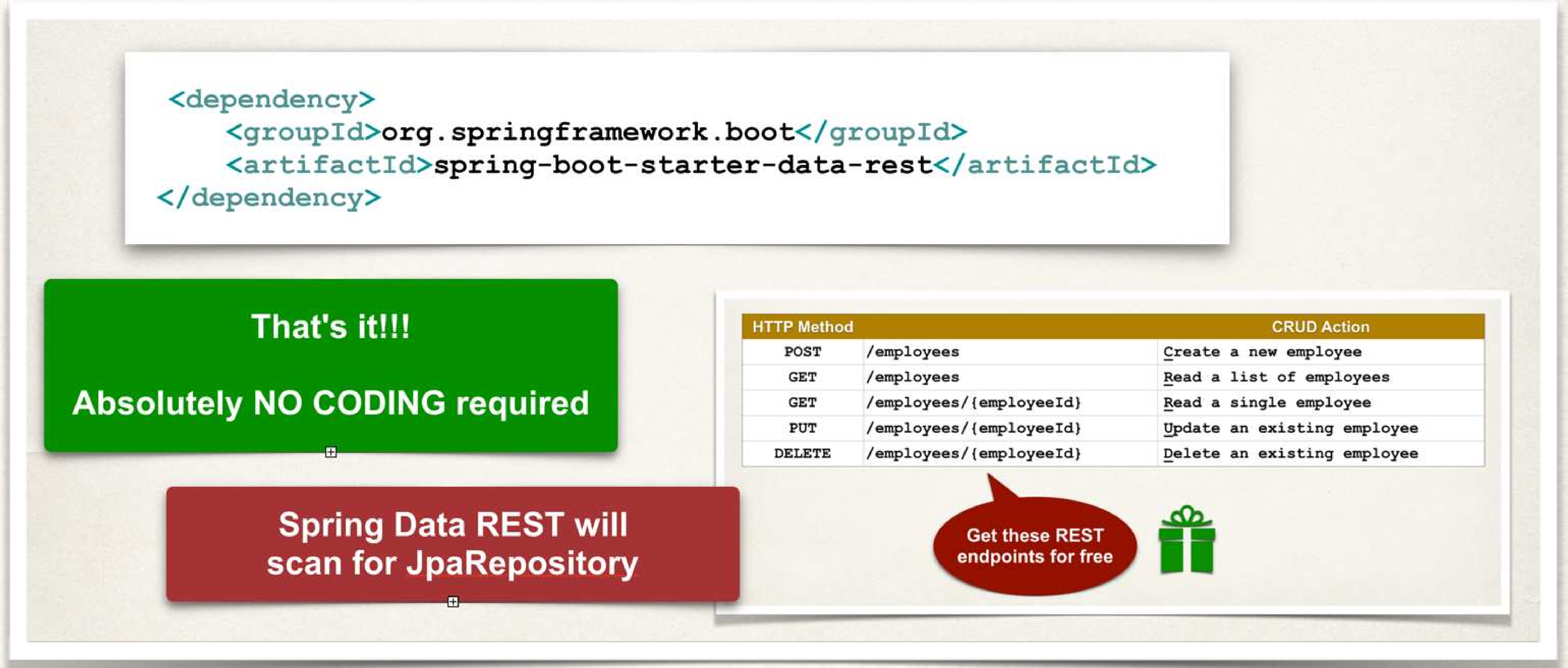
```
@RestController
@RequestMapping("/api")
public class EmployeeRestController {

    private EmployeeService employeeService;

    @Autowired
    public EmployeeRestController(EmployeeSer          loyeeService) {
        employeeService = theEmployeeService
    }

    // expose "/employees" and return list of
    @GetMapping("/employees")
    public List<Employee> findAll() {
        return employeeService.findAll();
    }

    // add mapping for GET /er

    @GetMapping("/employees/{e
    public Employee getEmploye

        Employee theEmployee =

        if (theEmployee == nu
            throw new Runtime

        }

        return theEmployee;
    }
```

```
public interf              vice {

    public Li              indAll();

    public                 nt theId);

    pu

}
```

```
        implements EmployeeService {

    Employ             loyeeRepository;

    owired
    lic EmployeeSer    eImpl(EmployeeRepository theEmployeeRepository) {
        employeeRepository = theEmployeeRepository;
    }

    @Override
    public List<Employee> findAll() {
        return employeeRepository.findAll();
    }

                  d(int theId) {

                  result = employeeRepository.findById(theId);

                  e = null;

                  ()) {
                  esult.get();

                  nd the employee
                  meException("Did not find employee id - " + theId);

        return theEmployee;
```

3 files
100+ lines of code

After Spring Data REST



```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

That's it!!!

Absolutely NO CODING required

Spring Data REST will scan for JpaRepository

HTTP Method		CRUD Action
POST	/employees	Create a new employee
GET	/employees	Read a list of employees
GET	/employees/{employeeId}	Read a single employee
PUT	/employees/{employeeId}	Update an existing employee
DELETE	/employees/{employeeId}	Delete an existing employee

Get these REST endpoints for free

HATEOAS

HATEOAS

- Spring Data REST endpoints are HATEOAS compliant

HATEOAS

- Spring Data REST endpoints are HATEOAS compliant
  - **HATEOAS**: Hypermedia as the Engine of Application State

HATEOAS

- Spring Data REST endpoints are HATEOAS compliant
  - **HATEOAS**: Hypermedia as the Engine of Application State
- Hypermedia-driven sites provide information to access REST interfaces

HATEOAS

- Spring Data REST endpoints are HATEOAS compliant
  - **HATEOAS**: Hypermedia as the Engine of Application State

- Hypermedia-driven sites provide information to access REST interfaces
  - Think of it as meta-data for REST data

HATEOAS

- Spring Data REST endpoints are HATEOAS compliant
    - **HATEOAS**: Hypermedia as the Engine of Application State

- Hypermedia-driven sites provide information to access REST interfaces
    - Think of it as meta-data for REST data

https://spring.io/understanding/HATEOAS

HATEOAS

HATEOAS

- Spring Data REST response using HATEOAS

HATEOAS

Get single employee

- Spring Data REST response using HATEOAS
- For example REST response from: **`GET /employees/3`**

HATEOAS

Get single employee

- Spring Data REST response using HATEOAS
- For example REST response from: **`GET /employees/3`**

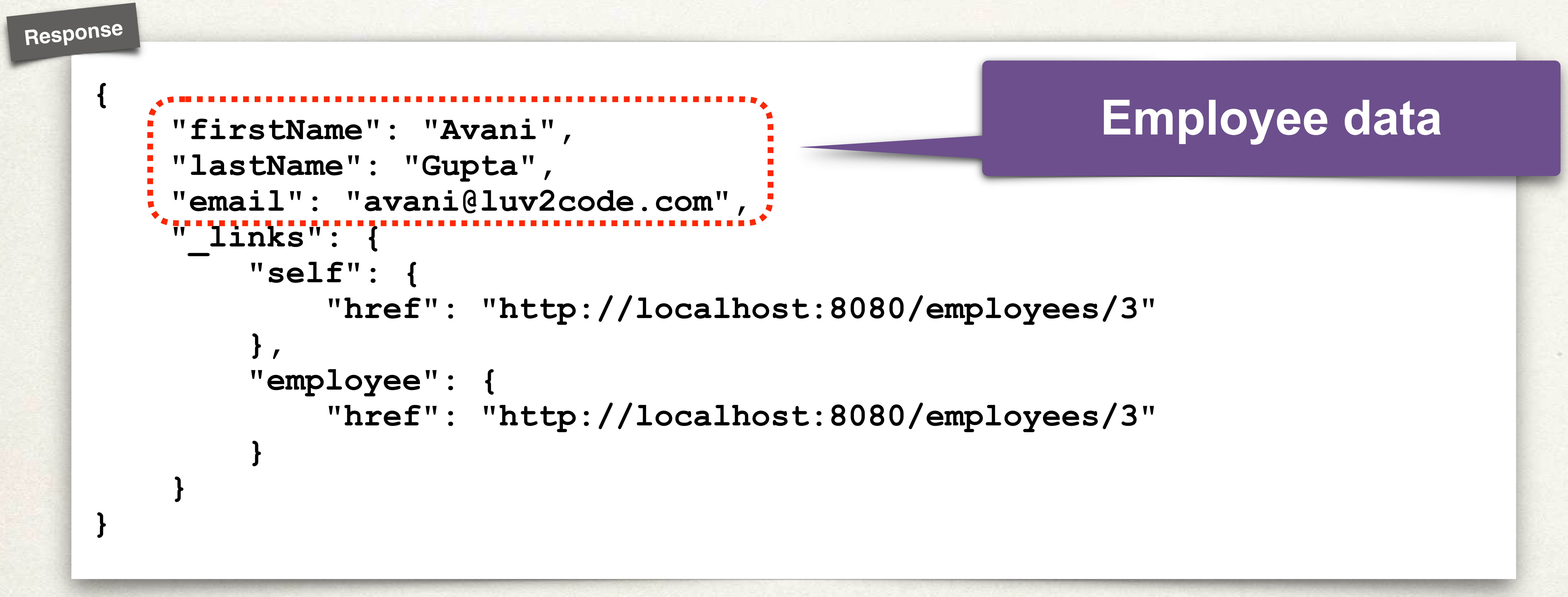
Response

```
{
    "firstName": "Avani",
    "lastName": "Gupta",
    "email": "avani@luv2code.com",
    "_links": {
        "self": {
            "href": "http://localhost:8080/employees/3"
        },
        "employee": {
            "href": "http://localhost:8080/employees/3"
        }
    }
}
```

HATEOAS

Get single employee

- Spring Data REST response using HATEOAS
- For example REST response from: **`GET /employees/3`**



Response

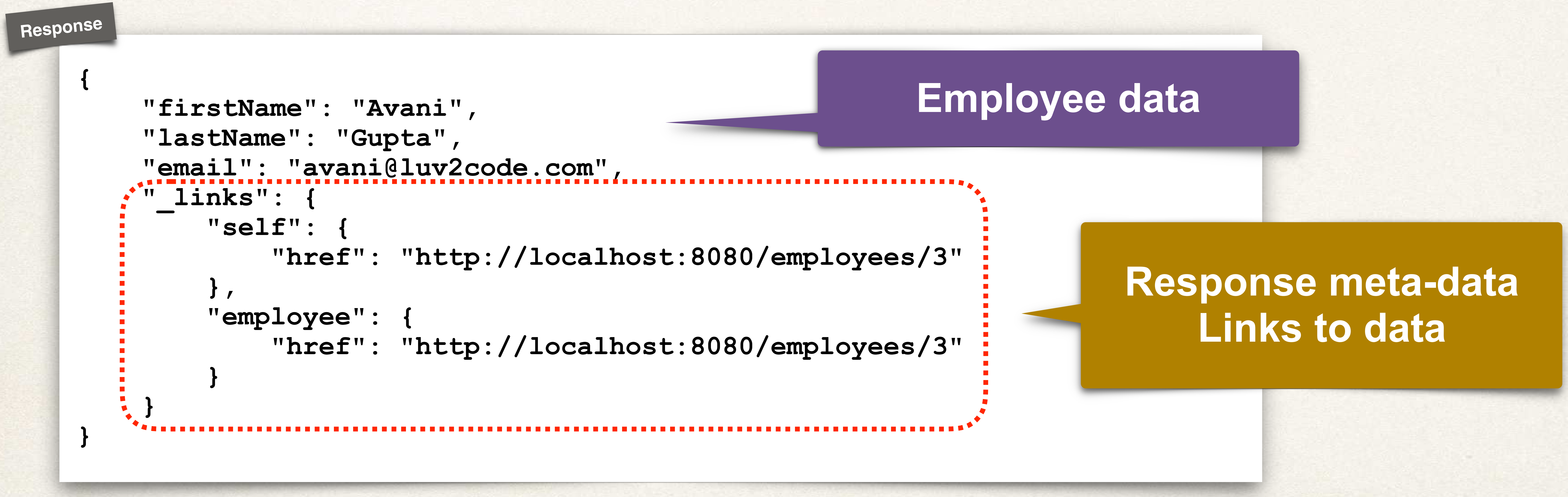
```
{
    "firstName": "Avani",
    "lastName": "Gupta",
    "email": "avani@luv2code.com",
    "_links": {
        "self": {
            "href": "http://localhost:8080/employees/3"
        },
        "employee": {
            "href": "http://localhost:8080/employees/3"
        }
    }
}
```

Employee data

HATEOAS

Get single employee

- Spring Data REST response using HATEOAS
- For example REST response from: **`GET /employees/3`**



Response

```
{
    "firstName": "Avani",
    "lastName": "Gupta",
    "email": "avani@luv2code.com",
    "_links": {
        "self": {
            "href": "http://localhost:8080/employees/3"
        },
        "employee": {
            "href": "http://localhost:8080/employees/3"
        }
    }
}
```

Employee data

Response meta-data
Links to data

HATEOAS

HATEOAS

- For a collection, meta-data includes page size, total elements, pages etc

HATEOAS

- For a collection, meta-data includes page size, total elements, pages etc
- For example REST response from: **`GET /employees`**

Get list of employees

HATEOAS

- For a collection, meta-data includes page size, total elements, pages etc
- For example REST response from: **`GET /employees`**

Get list of employees

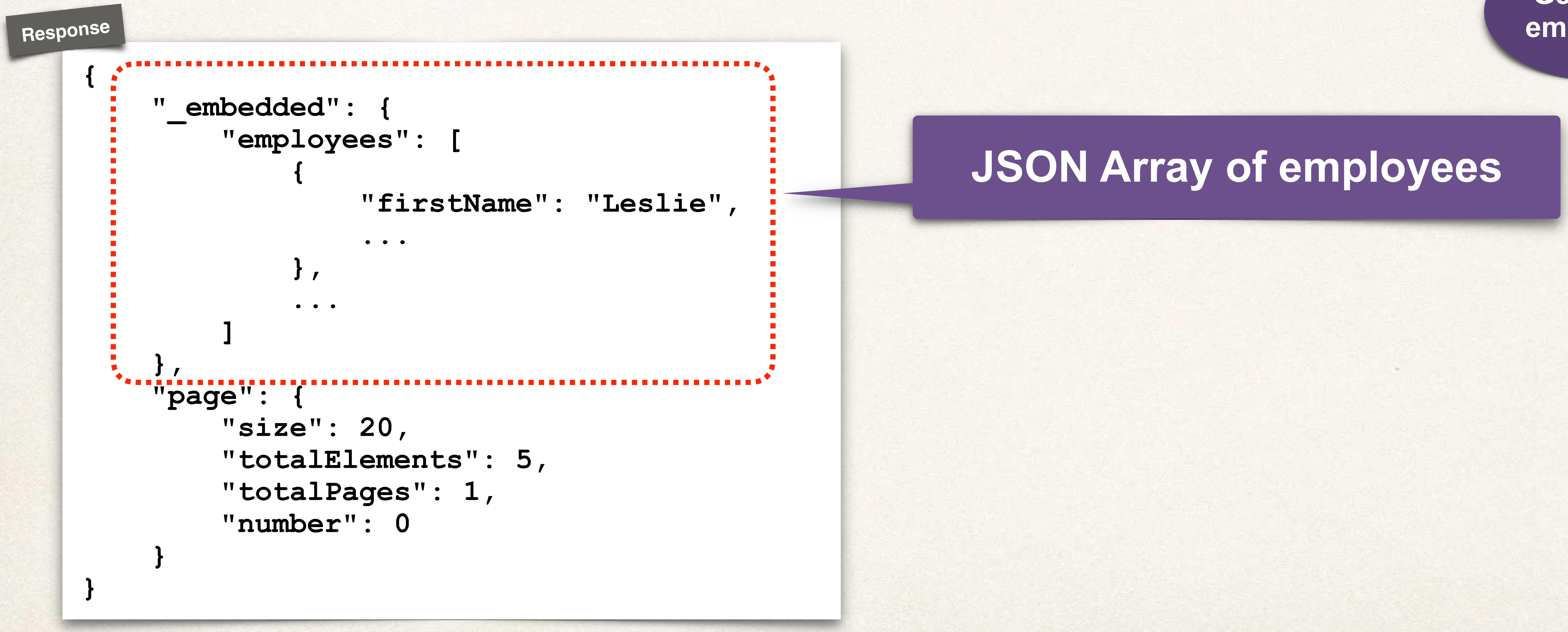
Response

```
{
    "_embedded": {
        "employees": [
            {
                "firstName": "Leslie",
                ...
            },
            ...
        ]
    },
    "page": {
        "size": 20,
        "totalElements": 5,
        "totalPages": 1,
        "number": 0
    }
}
```

HATEOAS

- For a collection, meta-data includes page size, total elements, pages etc
- For example REST response from: **`GET /employees`**

Get list of employees



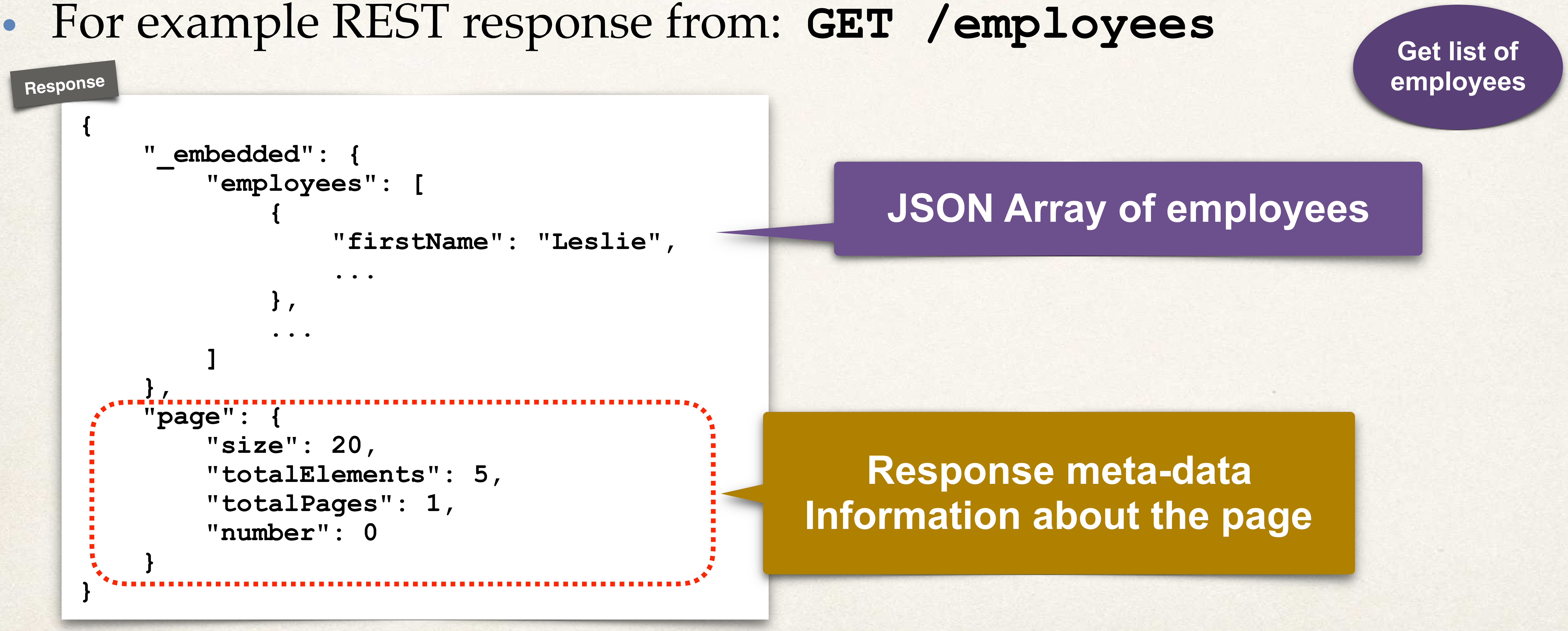
Response

```
{
    "_embedded": {
        "employees": [
            {
                "firstName": "Leslie",
                ...
            },
            ...
        ]
    },
    "page": {
        "size": 20,
        "totalElements": 5,
        "totalPages": 1,
        "number": 0
    }
}
```

JSON Array of employees

HATEOAS

- For a collection, meta-data includes page size, total elements, pages etc
- For example REST response from: **`GET /employees`**



Response

```
{
    "_embedded": {
        "employees": [
            {
                "firstName": "Leslie",
                ...
            },
            ...
        ]
    },
    "page": {
        "size": 20,
        "totalElements": 5,
        "totalPages": 1,
        "number": 0
    }
}
```

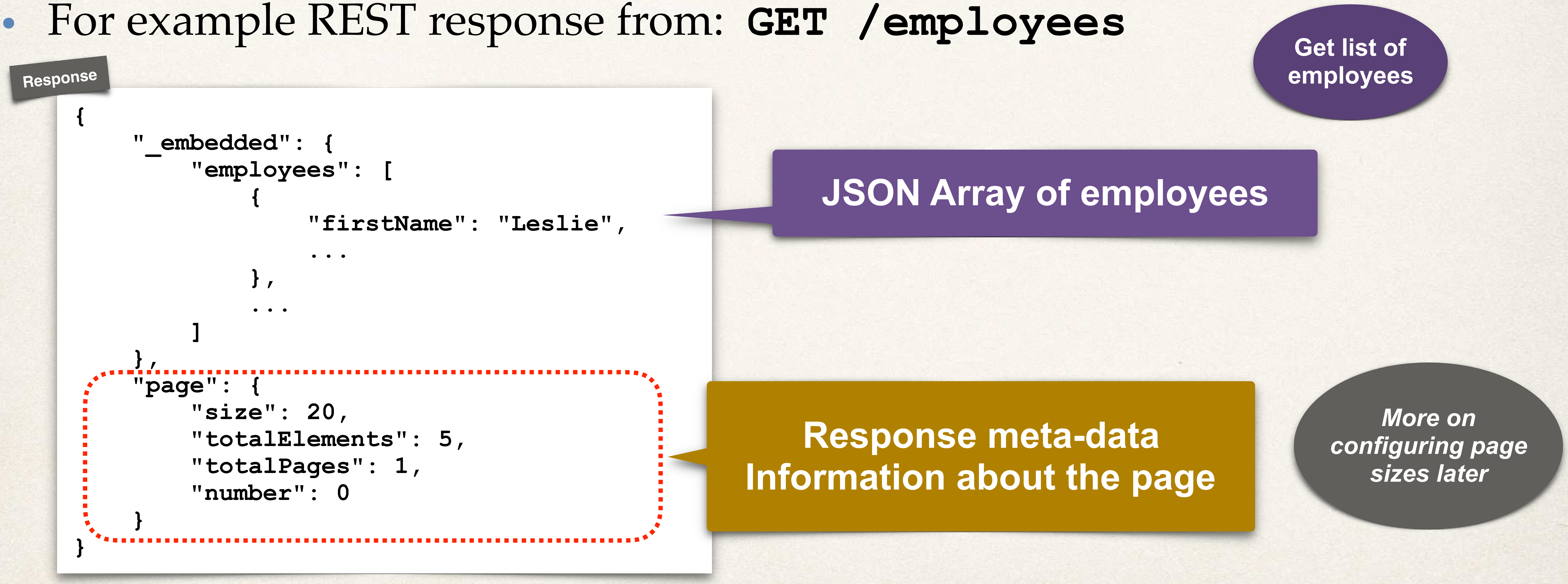
Get list of employees

JSON Array of employees

Response meta-data
Information about the page

HATEOAS

- For a collection, meta-data includes page size, total elements, pages etc
- For example REST response from: **`GET /employees`**



Get list of employees

Response

```
{
    "_embedded": {
        "employees": [
            {
                "firstName": "Leslie",
                ...
            },
            ...
        ]
    },
    "page": {
        "size": 20,
        "totalElements": 5,
        "totalPages": 1,
        "number": 0
    }
}
```

JSON Array of employees

Response meta-data
Information about the page

More on configuring page sizes later

HATEOAS

HATEOAS

- For details on HATEOAS, see

https://spring.io/understanding/HATEOAS

HATEOAS

- For details on HATEOAS, see

https://spring.io/understanding/HATEOAS

- HATEOAS uses Hypertext Application Language (HAL) data format

HATEOAS

- For details on HATEOAS, see

https://spring.io/understanding/HATEOAS

- HATEOAS uses Hypertext Application Language (HAL) data format

- For details on HAL, see

https://en.wikipedia.org/wiki/Hypertext_Application_Language

Advanced Features

Advanced Features

- Spring Data REST advanced features

Advanced Features

- Spring Data REST advanced features
  - Pagination, sorting and searching

Advanced Features

- Spring Data REST advanced features
  - Pagination, sorting and searching
  - Extending and adding custom queries with JPQL

Advanced Features

- Spring Data REST advanced features
  - Pagination, sorting and searching
  - Extending and adding custom queries with JPQL
  - Query Domain Specific Language (Query DSL)

Advanced Features

- Spring Data REST advanced features
  - Pagination, sorting and searching
  - Extending and adding custom queries with JPQL
  - Query Domain Specific Language (Query DSL)

https://spring.io/projects/spring-data-rest

Spring Data REST
Configuration, Pagination and Sorting

REST Endpoints

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type
- <u>Simple</u> pluralized form

REST Endpoints

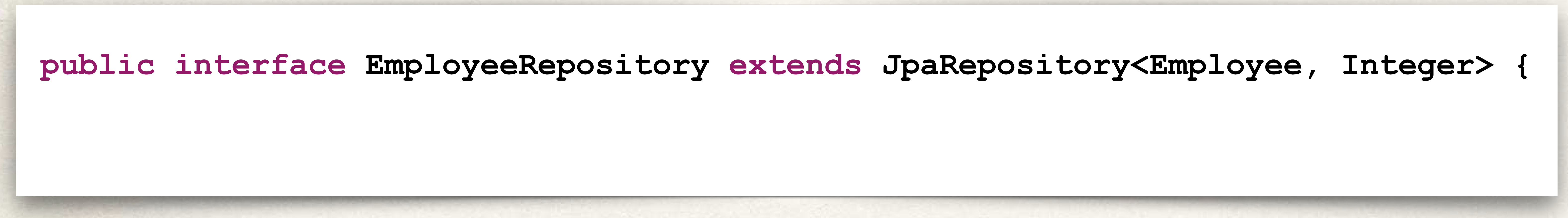
- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase
  - Then just adds an "s" to the entity

REST Endpoints

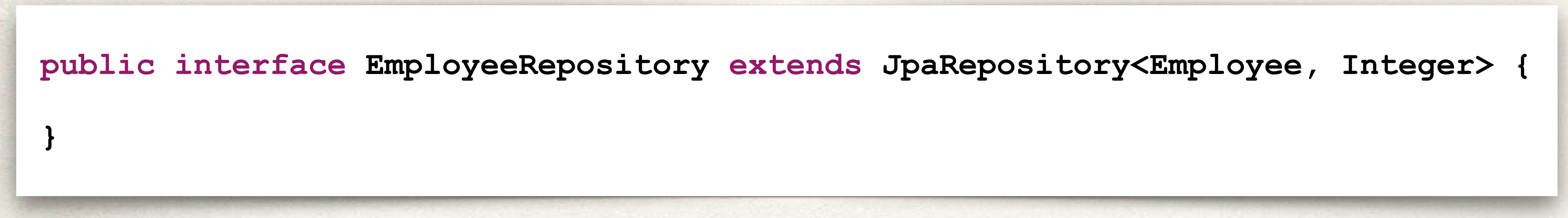
- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase
  - Then just adds an "s" to the entity



```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {
```

REST Endpoints

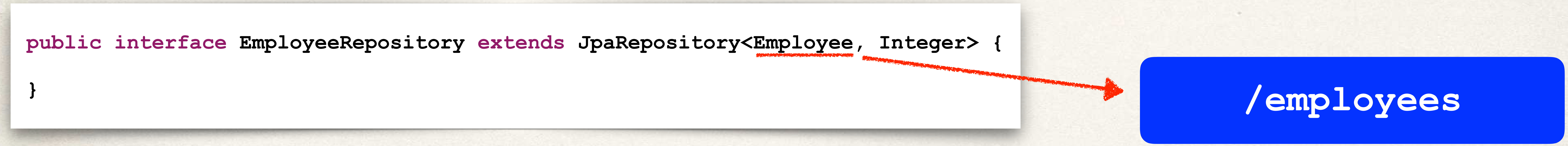
- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase
  - Then just adds an "s" to the entity



```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase
  - Then just adds an "s" to the entity



```
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

Pluralized Form

Pluralized Form

- Spring Data REST pluralized form is VERY simple

Pluralized Form

- Spring Data REST pluralized form is VERY simple
  - Just adds an "s" to the entity

Pluralized Form

- Spring Data REST pluralized form is VERY simple
  - Just adds an "s" to the entity
- The English language is VERY complex!

Pluralized Form

- Spring Data REST pluralized form is VERY simple
  - Just adds an "s" to the entity

- The English language is VERY complex!
  - Spring Data REST does NOT handle

Pluralized Form

- Spring Data REST pluralized form is VERY simple
  - Just adds an "s" to the entity

- The English language is VERY complex!
  - Spring Data REST does NOT handle

Singular	Plural
Goose	Geese
Person	People
Syllabus	Syllabi
…	…

Pluralized Form

- Spring Data REST pluralized form is VERY simple
  - Just adds an "s" to the entity

- The English language is VERY complex!
  - Spring Data REST does NOT handle

Singular	Plural
Goose	Geese
Person	People
Syllabus	Syllabi
…	…

Problem

Problem

- Spring Data REST does not handle complex pluralized forms

Problem

- Spring Data REST does not handle complex pluralized forms
  - In this case, you need to specify plural name

Problem

- Spring Data REST does not handle complex pluralized forms
  - In this case, you need to specify plural name
- What if we want to expose a different resource name?

Problem

- Spring Data REST does not handle complex pluralized forms
  - In this case, you need to specify plural name
- What if we want to expose a different resource name?
  - Instead of **`/employees`** … use **`/members`**

Solution

Solution

- Specify plural name / path with an annotation

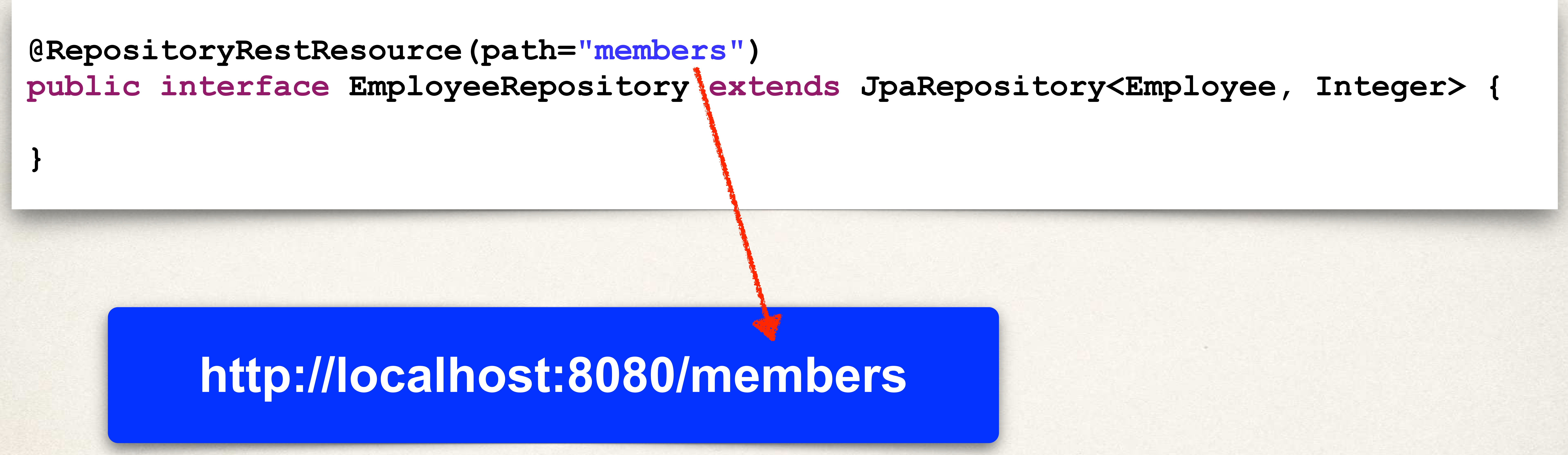
Solution

- Specify plural name / path with an annotation

```
@RepositoryRestResource(path="members")
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

Solution

- Specify plural name / path with an annotation



```
@RepositoryRestResource(path="members")
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}
```

http://localhost:8080/members

Pagination

Pagination

- By default, Spring Data REST will return the first 20 elements
  - Page size = 20

Pagination

- By default, Spring Data REST will return the first 20 elements
  - Page size = 20

- You can navigate to the different pages of data using query param

Pagination

- By default, Spring Data REST will return the first 20 elements
  - Page size = 20

- You can navigate to the different pages of data using query param

```
http://localhost:8080/employees?page=0

http://localhost:8080/employees?page=1

...
```

Pagination

- By default, Spring Data REST will return the first 20 elements
  - Page size = 20

- You can navigate to the different pages of data using query param

```
http://localhost:8080/employees?page=0

http://localhost:8080/employees?page=1

...
```

Pages are zero-based

Spring Data REST Configuration

- Following properties available: application.properties

Name	Description
`spring.data.rest.base-path`	Base path used to expose repository resources
`spring.data.rest.default-page-size`	Default size of pages
`spring.data.rest.max-page-size`	Maximum size of pages
…	…

Spring Data REST Configuration

- Following properties available: application.properties

Name	Description
`spring.data.rest.base-path`	Base path used to expose repository resources
`spring.data.rest.default-page-size`	Default size of pages
`spring.data.rest.max-page-size`	Maximum size of pages
...	…

More properties available

www.luv2code.com/spring-boot-props

Spring Data REST Configuration

- Following properties available: application.properties

Name	Description
`spring.data.rest.base-path`	Base path used to expose repository resources
`spring.data.rest.default-page-size`	Default size of pages
`spring.data.rest.max-page-size`	Maximum size of pages
…	…

More properties available

www.luv2code.com/spring-boot-props

`spring.data.rest.*`

Sample Configuration

Sample Configuration

File: application.properties

```
spring.data.rest.base-path=/magic-api

spring.data.rest.default-page-size=50
```

Sample Configuration

http://localhost:8080/magic-api/employees

File: application.properties

```
spring.data.rest.base-path=/magic-api

spring.data.rest.default-page-size=50
```

Sample Configuration

http://localhost:8080/magic-api/employees

File: application.properties

```
spring.data.rest.base-path=/magic-api

spring.data.rest.default-page-size=50
```

Returns 50 elements per page

Sorting

Sorting

- You can sort by the property names of your entity

Sorting

- You can sort by the property names of your entity
  - In our Employee example, we have: **`firstName`**, **`lastName`** and **`email`**

Sorting

- You can sort by the property names of your entity
  - In our Employee example, we have: **`firstName`**, **`lastName`** and **`email`**

- Sort by last name (ascending is default)

```
http://localhost:8080/employees?sort=lastName
```

Sorting

- You can sort by the property names of your entity
    - In our Employee example, we have: **`firstName`**, **`lastName`** and **`email`**

- Sort by last name (ascending is default)

```
http://localhost:8080/employees?sort=lastName
```

- Sort by first name, descending

```
http://localhost:8080/employees?sort=firstName,desc
```

Sorting

- You can sort by the property names of your entity
  - In our Employee example, we have: **`firstName`**, **`lastName`** and **`email`**

- Sort by last name (ascending is default)

```
http://localhost:8080/employees?sort=lastName
```

- Sort by first name, descending

```
http://localhost:8080/employees?sort=firstName,desc
```

- Sort by last name, then first name, ascending

```
http://localhost:8080/employees?sort=lastName,firstName,asc
```