Spring Data JPA in Spring Boot

Various DAO Techniques

Various DAO Techniques

- Version 1: Use EntityManager but leverage native Hibernate API

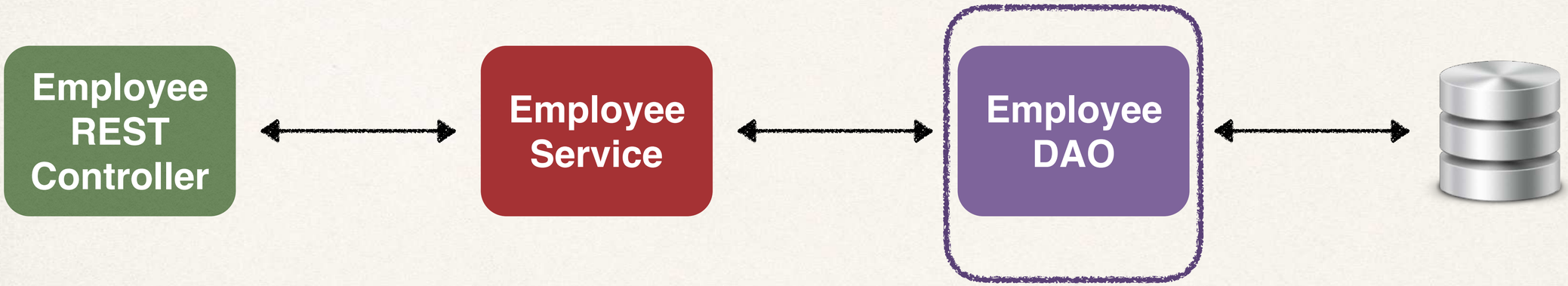
Various DAO Techniques

- Version 1: Use EntityManager but leverage native Hibernate API
- Version 2: Use EntityManager and standard JPA API

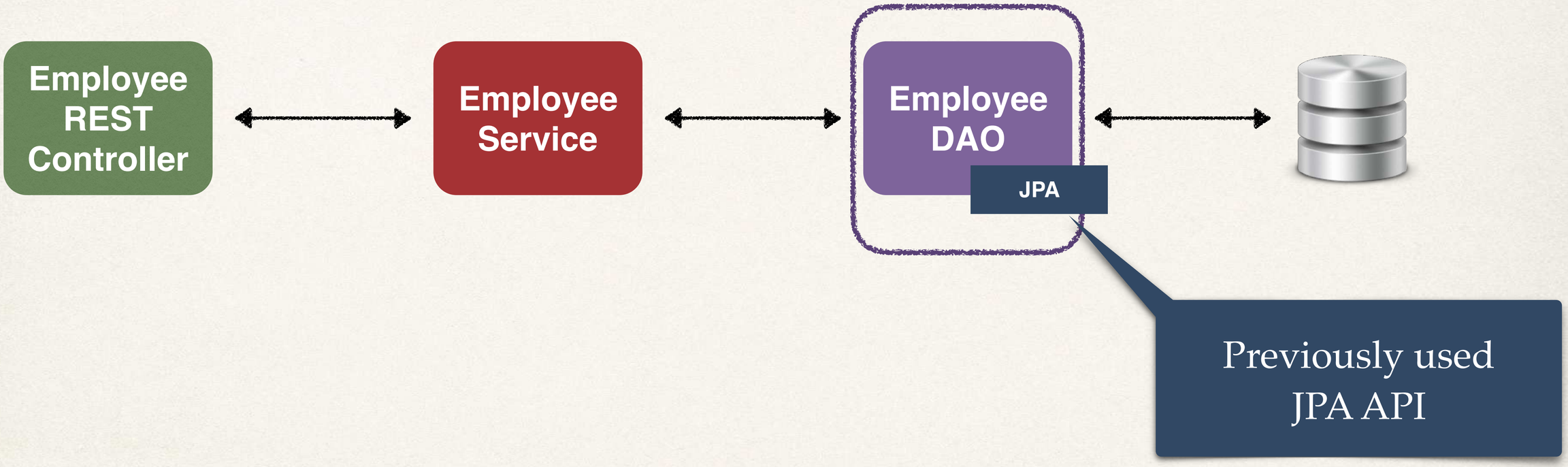
Various DAO Techniques

- Version 1: Use EntityManager but leverage native Hibernate API
- Version 2: Use EntityManager and standard JPA API
- Version 3: Spring Data JPA

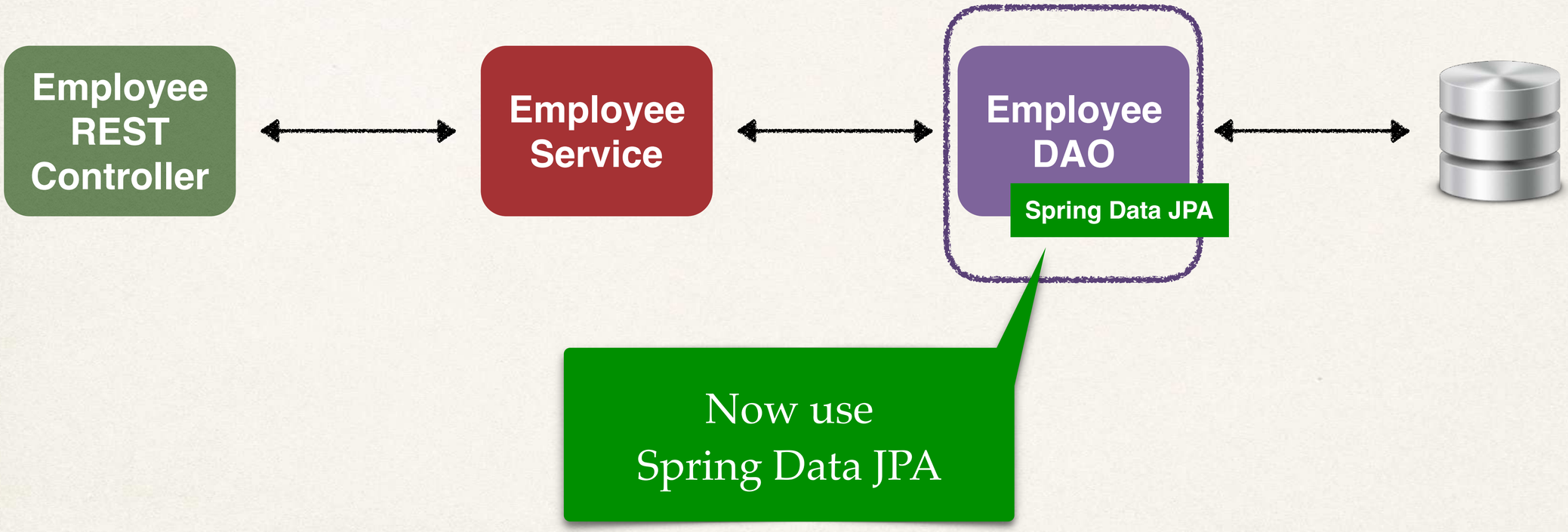
Application Architecture



Application Architecture



Application Architecture



The Problem

The Problem

- We saw how to create a DAO for **`Employee`**

The Problem

- We saw how to create a DAO for **`Employee`**

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

The Problem

- We saw how to create a DAO for **`Employee`**

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

The Problem

- We saw how to create a DAO for **`Employee`**

- What if we need to create a DAO for another entity?

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

The Problem

- We saw how to create a DAO for **`Employee`**

- What if we need to create a DAO for another entity?
  - **`Customer`**, **`Student`**, **`Product`**, **`Book`** ...

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

The Problem

- We saw how to create a DAO for **`Employee`**
- What if we need to create a DAO for another entity?
  - **`Customer`**, **`Student`**, **`Product`**, **`Book`** ...
- Do we have to repeat all of the same code again???

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

Creating DAO

Creating DAO

- You may have noticed a pattern with creating DAOs

Creating DAO

- You may have noticed a pattern with creating DAOs

```
@Override
public Employee findById(int theId) {

  // get data
  Employee theData = entityManager.find(Employee.class, theId);

  // return data
  return theData;
}
```

Creating DAO

- You may have noticed a pattern with creating DAOs

```
@Override
public Employee findById(int theId) {

  // get data
  Employee theData = entityManager.find(Employee.class, theId);

  // return data
  return theData;
}
```

Most of the code is the same

Creating DAO

- You may have noticed a pattern with creating DAOs

```
@Override
public Employee findById(int theId) {

  // get data
  Employee theData = entityManager.find(Employee.class, theId);

  // return data
  return theData;
}
```

Most of the code is the same

Only difference is the entity type and primary key

Creating DAO

- You may have noticed a pattern with creating DAOs

```
@Override
public Employee findById(int theId) {

  // get data
  Employee theData = entityManager.find(Employee.class, theId);

  // return data
  return theData;
}
```

Most of the code is the same

Only difference is the entity type and primary key

Creating DAO

- You may have noticed a pattern with creating DAOs

```
@Override
public Employee findById(int theId) {

  // get data
  Employee theData = entityManager.find(Employee.class, theId);

  // return data
  return theData;
}
```

Most of the code is the same

Only difference is the entity type and primary key

Entity type

Creating DAO

- You may have noticed a pattern with creating DAOs

```
@Override
public Employee findById(int theId) {

  // get data
  Employee theData = entityManager.find(Employee.class, theId);

  // return data
  return theData;
}
```

Most of the code is the same

Only difference is the entity type and primary key

Entity type

Primary key

My Wish

My Wish

- I wish we could tell Spring:

My Wish

- I wish we could tell Spring:

Create a DAO for me

My Wish

- I wish we could tell Spring:

Create a DAO for me

Plug in my entity type and primary key

My Wish

- I wish we could tell Spring:

Create a DAO for me

Plug in my entity type and primary key

Give me all of the basic CRUD features for free

My Wish Diagram

My Wish Diagram

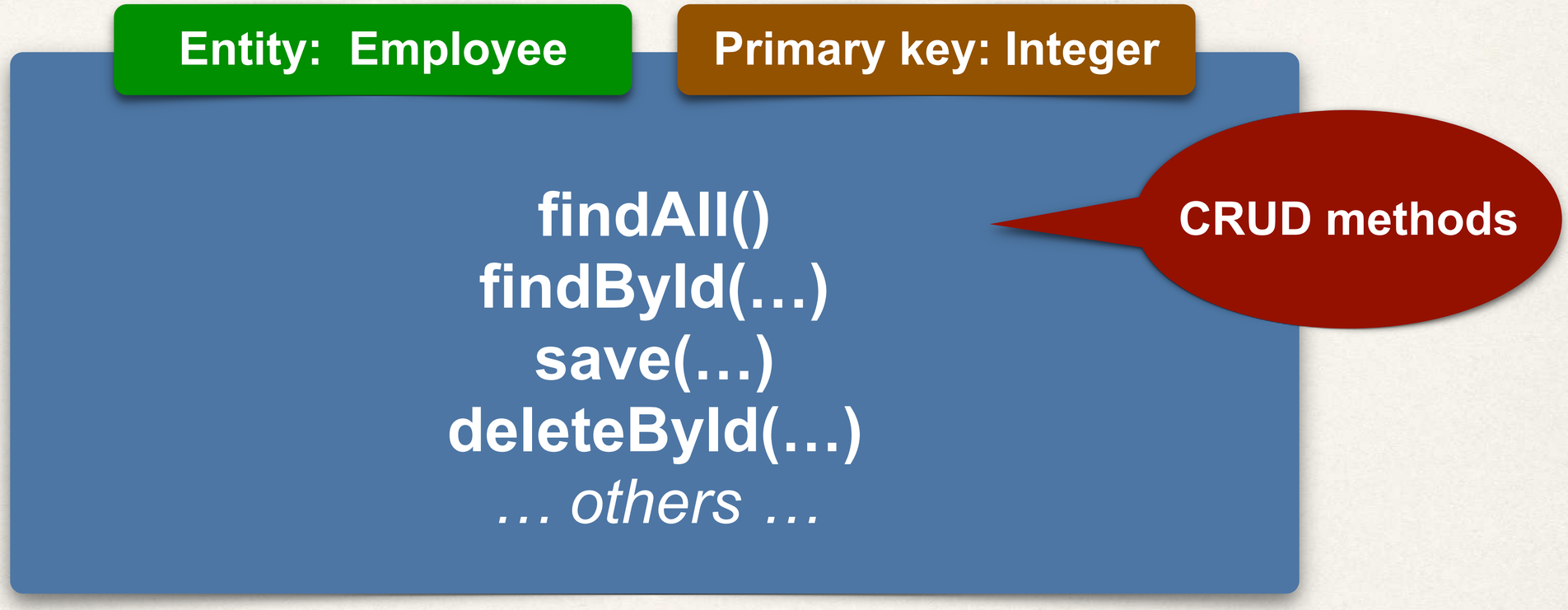
My Wish Diagram

My Wish Diagram

My Wish Diagram

My Wish Diagram

My Wish Diagram



My Wish Diagram

My Wish Diagram

My Wish Diagram

Spring Data JPA - Solution

Spring Data JPA - Solution

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

Spring Data JPA - Solution

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

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

Spring Data JPA - Solution

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

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

- Create a DAO and just plug in your entity type and primary key

Spring Data JPA - Solution

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

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

- Create a DAO and just plug in your entity type and primary key

Spring Data JPA - Solution

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

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

- Create a DAO and just plug in your entity type and primary key

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

Spring Data JPA - Solution

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

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

- Create a DAO and just plug in your entity type and primary key

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

Spring Data JPA - Solution

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

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

- Create a DAO and just plug in your entity type and primary key

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

Spring Data JPA - Solution

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

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

- Create a DAO and just plug in your entity type and primary key

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

More than 70% reduction in code … depending on use case

JpaRepository

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**
- Exposes methods (some by inheritance from parents)

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**
- Exposes methods (some by inheritance from parents)

findAll()

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**
- Exposes methods (some by inheritance from parents)

findAll()
findById(…)

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**
- Exposes methods (some by inheritance from parents)

findAll()
findById(…)
save(…)

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**
- Exposes methods (some by inheritance from parents)

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

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**
- Exposes methods (some by inheritance from parents)

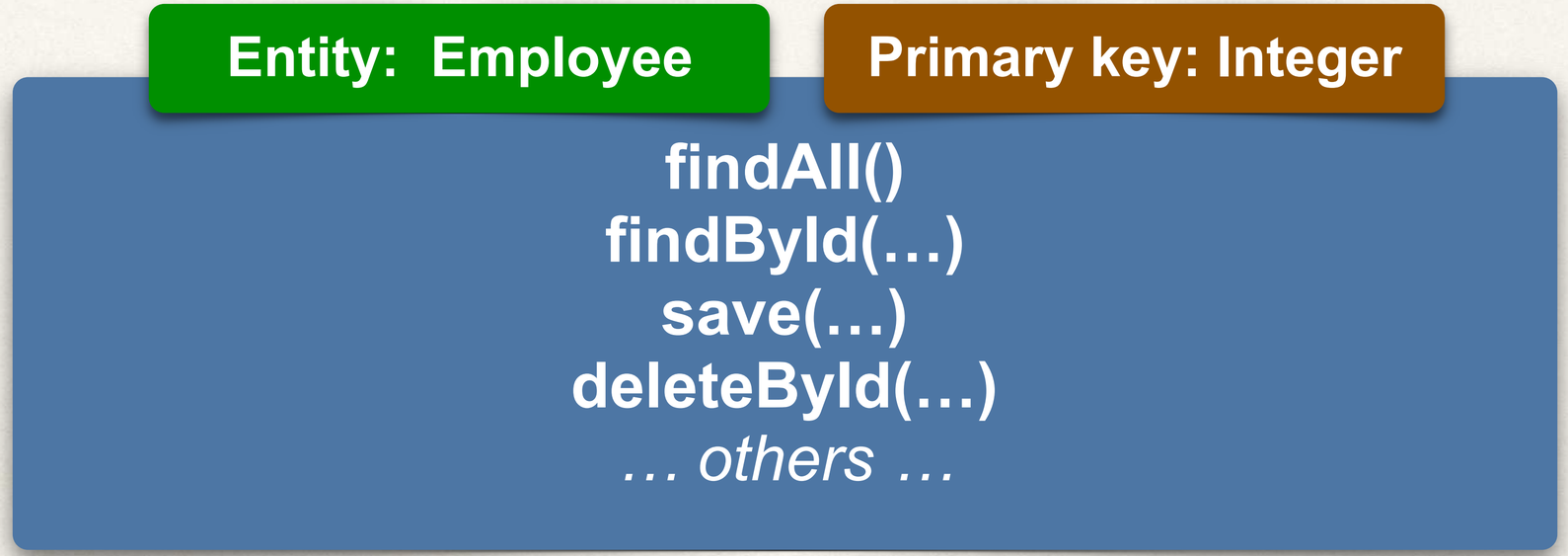
findAll()
findById(…)
save(…)
deleteById(…)
… others …

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**
- Exposes methods (some by inheritance from parents)

JpaRepository

- Spring Data JPA provides the interface: **`JpaRepository`**
- Exposes methods (some by inheritance from parents)



Development Process

Development Process

1. Extend **`JpaRepository`** interface

Development Process

1. Extend **`JpaRepository`** interface

2. Use your Repository in your app

Development Process

Step-By-Step

1. Extend **`JpaRepository`** interface

2. Use your Repository in your app

No need for implementation class

Step 1: Extend JpaRepository interface

Step 1: Extend JpaRepository interface

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

}
```

Step 1: Extend JpaRepository interface

Entity type

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

}
```

Step 1: Extend JpaRepository interface

Entity type

Primary key

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

}
```

Step 1: Extend JpaRepository interface

Entity type

Primary key

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

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

}
```

Step 1: Extend JpaRepository interface

Entity type

Primary key

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

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

}
```

Entity: Employee

Primary key: Integer

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

Step 1: Extend JpaRepository interface

Entity type

Primary key

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

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

}
```

Get these methods for free

Entity: Employee

Primary key: Integer

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

Step 1: Extend JpaRepository interface

Entity type

Primary key

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

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

}
```

No need for implementation class

Get these methods for free

Entity: Employee

Primary key: Integer

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

JpaRepository Docs

- Full list of methods available … see JavaDoc for **`JpaRepository`**

JpaRepository Docs

- Full list of methods available … see JavaDoc for **`JpaRepository`**

www.luv2code.com/jpa-repository-javadoc

Step 2: Use Repository in your app

Step 2: Use Repository in your app

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

  private EmployeeRepository employeeRepository;
```

Step 2: Use Repository in your app

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

  private EmployeeRepository employeeRepository;

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

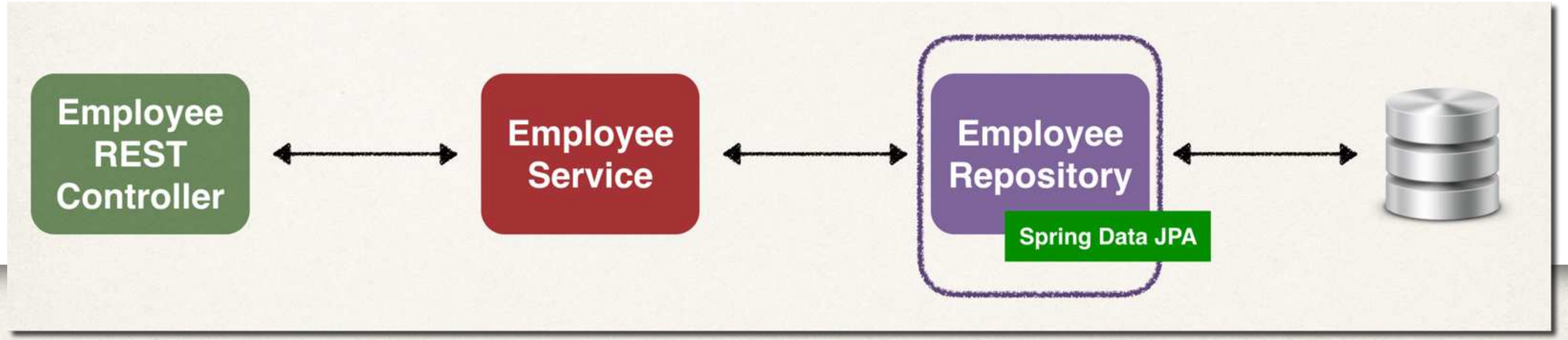
Step 2: Use Repository in your app

```
@Service
public class EmployeeServiceImpl implements EmployeeService {

  private EmployeeRepository employeeRepository;

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

Our repository



Step 2: Use Repository in your app

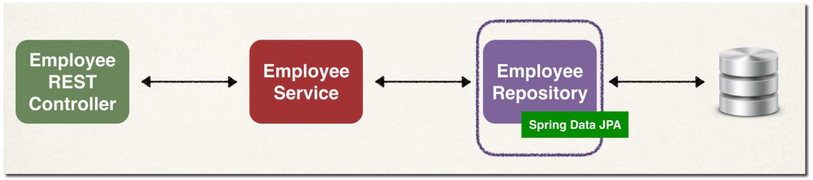
```
@Service
public class EmployeeServiceImpl implements EmployeeService {

  private EmployeeRepository employeeRepository;

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

  @Override
  public List<Employee> findAll() {
```

Our repository



Step 2: Use Repository in your app

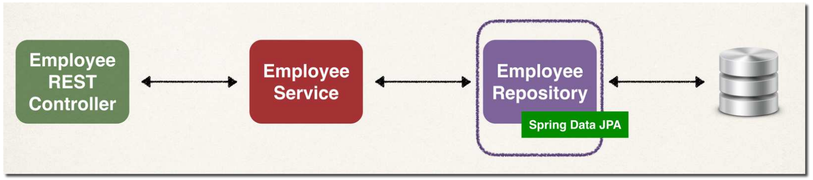
```
@Service
public class EmployeeServiceImpl implements EmployeeService {

  private EmployeeRepository employeeRepository;

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

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

Our repository



Step 2: Use Repository in your app

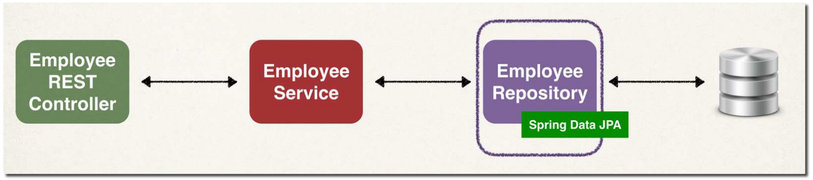
```
@Service
public class EmployeeServiceImpl implements EmployeeService {

  private EmployeeRepository employeeRepository;

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

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

Our repository



Magic method that is available via repository

Minimized Boilerplate Code

Minimized Boilerplate Code

Before Spring Data JPA

Minimized Boilerplate Code

Before Spring Data JPA

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

Minimized Boilerplate Code

Before Spring Data JPA

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

Minimized Boilerplate Code

Before Spring Data JPA

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

2 Files
30+ lines of code

Minimized Boilerplate Code

Before Spring Data JPA

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

2 Files
30+ lines of code

After Spring Data JPA

Minimized Boilerplate Code

Before Spring Data JPA

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

2 Files
30+ lines of code

After Spring Data JPA

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

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

}
```

Minimized Boilerplate Code

Before Spring Data JPA

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

2 Files
30+ lines of code

After Spring Data JPA

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

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

}
```

1 File
3 lines of code!

Minimized Boilerplate Code

Before Spring Data JPA

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Employee findById(int theId);

    public void save(Employee theEmployee);

    public void deleteById(int theId);

}
```

```
@Repository
public class EmployeeDAOJpaImpl implements EmployeeDAO {

    private EntityManager entityManager;

    @Autowired
    public EmployeeDAOJpaImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Employee> findAll() {

        // create a query
        TypedQuery<Employee> theQuery =
                entityManager.createQuery("from Employee", Employee.class);

        // execute query and get result list
        List<Employee> employees = theQuery.getResultList();

        // return the results
        return employees;
    }

    @Override
    public Employee findById(int theId) {

        // get employee
        Employee theEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

2 Files
30+ lines of code

After Spring Data JPA

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

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

}
```

1 File
3 lines of code!

No need for implementation class

Minimized Boilerplate Code

Before Spring Data JPA

```
public interface EmployeeDAO {

    public List<Employee> findAll();

    public Emplo      yId(int theId);

    public void       ee theEmployee

    public void d     theId)

}
```

```
@Repository
public class Employee

    private EntityManag

    @Autowired
    public EmployeeDAOJpa      tyManager) {
        entityManager = th
    }

    @Override
    public List<Empl

        // create
        TypedQue                    loyee.class);

        //
        L                  = theQ

    }

    @Ov
    publ          dById(int theId) {

        // get
        Employe       eEmployee =
                entityManager.find(Employee.class, theId);

        // return employee
        return theEmployee;
    }
```

2 Files
30+ lines of code

After Spring Data JPA

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

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

}
```

1 File
3 lines of code!

No need for
implementation class

Advanced Features

Advanced Features

- Advanced features available for

Advanced Features

- Advanced features available for
  - Extending and adding custom queries with JPQL

Advanced Features

- Advanced features available for
  - Extending and adding custom queries with JPQL
  - Query Domain Specific Language (Query DSL)

Advanced Features

- Advanced features available for
  - Extending and adding custom queries with JPQL
  - Query Domain Specific Language (Query DSL)
  - Defining custom methods (low-level coding)

Advanced Features

- Advanced features available for
  - Extending and adding custom queries with JPQL
  - Query Domain Specific Language (Query DSL)
  - Defining custom methods (low-level coding)

www.luv2code.com/spring-data-jpa-defining-custom-queries