@SpringBootApplication(scanBasePackages= {"com.ibm.bank"})

@EnableJpaRepositories("com.ibm.bank")

@EntityScan("com.ibm.bank.model.persistence.account")

@RestController

@ComponentScan("com.ibm.bank")

@Autowired

@RequestMapping(value = "/getAllAccount", method = RequestMethod.***GET***)

@ResponseBody

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**What will happen if we interchange @service and @repository annotation in the spring MVC**

According to [documentaion](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/" \l "beans-stereotype-annotations) @Repository,@Service,@Controller are all synonyms. They all just specializations of @Component annotation. So, generally, they can be used one istead of other. But ... you should not do this.

First reason: any of this annotations *make clear the role* of your component in the application. Shows - is this component belongs to controller, service, or data layer.

Second reason: some of this annotations processed differently by different Spring modules. For example, Spring Data JPA will process @Repository, and will try to replace with implementation any interface marked by this annotaion. Spring also will apply automatic exception translation to such classes. Another example, Spring Web MVC process @Controller, and use classes marked with it in URL mappings.

Actually, in future versions, some modules of Spring could process @Service in particular way. Not as simple @Component. That's why documentation advice:

It is also possible that @Repository, @Service, and @Controller may carry additional semantics in future releases of the Spring Framework. Thus, if you are choosing between using @Component or @Service for your service layer, @Service is clearly the better choice.

Client -> Rest endpoint -> service -> DAO -> database

With @Component, @Repository, @Service and @Controller annotations in place and after enabling automatic component scanning, spring will automatically import the beans into the container so you don’t have to define them explicitly to autowire them.

**@Component**

The @Component annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context. To use this annotation, apply it over class as below:

**@Repository**

Although above use of @Component is good enough but you can use more suitable annotation that provides additional benefits specifically for DAOs i.e. @Repository annotation. The @Repository annotation is a specialization of the @Component annotation with similar use and functionality. In addition to importing the DAOs into the DI container, it also makes the unchecked exceptions (thrown from DAO methods) eligible for translation into Spring DataAccessException.

**@Service**

The @Service annotation is also a specialization of the component annotation. It doesn’t currently provide any additional behavior over the @Component annotation, but it’s a good idea to use @Service over @Component in service-layer classes because it specifies intent better. Additionally, tool support and additional behavior might rely on it in the future.

**@Controller**

@Controller annotation marks a class as a Spring Web MVC controller. It too is a @Component specialization, so beans marked with it are automatically imported into the DI container. When you add the @Controller annotation to a class, you can use another annotation i.e. @RequestMapping; to map URLs to instance methods of a class.

In your scenario, it has no impact on application flow whether you use @Service or @Repository, application will work as they are elligible for autowiring. But standard practice is to use @Repository for dao classes.

[What is difference between CrudRepository and JpaRepository interfaces in Spring Data JPA?](https://stackoverflow.com/questions/14014086/what-is-difference-between-crudrepository-and-jparepository-interfaces-in-spring)

[JpaRepository](http://static.springsource.org/spring-data/data-jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html) extends [PagingAndSortingRepository](http://static.springsource.org/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html) which in turn extends [CrudRepository](http://static.springsource.org/spring-data/data-commons/docs/current/api/org/springframework/data/repository/CrudRepository.html).

Their main functions are:

* [CrudRepository](http://static.springsource.org/spring-data/data-commons/docs/current/api/org/springframework/data/repository/CrudRepository.html) mainly provides CRUD functions.
* [PagingAndSortingRepository](http://static.springsource.org/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html) provides methods to do pagination and sorting records.
* [JpaRepository](http://static.springsource.org/spring-data/data-jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html) provides some JPA-related methods such as flushing the persistence context and deleting records in a batch.

Because of the inheritance mentioned above, JpaRepository will have all the functions of CrudRepository and PagingAndSortingRepository. So if you don't need the repository to have the functions provided by JpaRepository and PagingAndSortingRepository , use CrudRepository.

There are three important repository interfaces that you should know while using Spring Data JPA **:**

1. **CrudRepository** interface provides methods for CRUD operations, so it allows you to create, read, update and delete records without having to define your own methods.
2. **PagingAndSortingRepository** provides additional methods to retrieve entities using pagination and sorting.
3. **JpaRepository** has additional JPA specific methods that support for example [Query By Example](https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#query-by-example) , deleting in batches, manual flushing changes to database and querying methods return List instead of Iterable

GPA

Generally the best idea is to use **CrudRepository** or **PagingAndSortingRepository** depending on whether you need sorting and paging or not.

The **JpaRepository** should be avoided if possible, because it ties you repositories to the JPA persistence technology, and in most cases you probably wouldn’t even use the extra methods provided by it.

The Hibernate cache--- mechanism allows you to cache entities, entity collections and query results while

Spring cache offers application-level memoization.

Hibernate cache-- is to bypass the database and reduce load on the Database while the Spring cache can bypass the data access layer as well and it’s not related to caching results from the DB.

[share](https://stackoverflow.com/a/59899597)[improve this answer](https://stackoverflow.com/posts/59899597/edit)

[**Spring: @Component versus @Bean**](https://stackoverflow.com/questions/10604298/spring-component-versus-bean)

1. @Component **auto detects** and configures the beans using classpath scanning whereas @Bean **explicitly declares** a single bean, rather than letting Spring do it automatically.
2. @Component **does not decouple** the declaration of the bean from the class definition where as @Bean **decouples** the declaration of the bean from the class definition.
3. @Component is a **class level annotation** where as @Bean is a **method level annotation** and name of the method serves as the bean name.
4. @Component **need not to be used with the @Configuration** annotation where as @Bean annotation has to be **used within the class which is annotated with @Configuration**.
5. We **cannot create a bean** of a class using @Component, if the class is outside spring container whereas we **can create a bean** of a class using @Bean even if the class is present **outside the spring container**.
6. @Component has **different specializations** like @Controller, @Repository and @Service whereas @Bean has **no specializations**.

**@Component** Preferable for component scanning and automatic wiring.

When should you use ***@Bean***?

Sometimes automatic configuration is not an option. **When?** Let's imagine that you want to wire components from 3rd-party libraries (you don't have the source code so you can't annotate its classes with @Component), so automatic configuration is not possible.

The **@Bean** annotation **returns an object** that spring should register as bean in application context. The **body of the method** bears the logic responsible for creating the instance.

Spring bean life cycle <https://dzone.com/articles/spring-bean-lifecycle>