Spring Certified #9

A question lead guide to prepare Spring certification

Testing

What is the purpose of the @MockBean annotation in a Spring Boot web slice test? (select 3)

- ➔ **It adds mock objects to the Spring application context.**
- ➔ **Can be used as a class-level annotation or on fields.**
- ➔ **It does the same as @Mock , it is just from another library.**
- ➔ **It is used to define a new Spring Bean in the test context.**

It adds mock objects to the Spring application context.
Can be used as a class-level annotation or on fields.
It is used to define a new Spring Bean in the test context.

@interface **MockBean**

Annotation that can be used to add mocks to a Spring `ApplicationContext`. **Can be used as a class level annotation or on fields** in either `@Configuration` classes, or test classes that are `@RunWith` the `SpringRunner`.

Mocks can be registered by type or by `bean name`. When registered by type, any existing single bean of a matching type (including subclasses) in the context will be replaced by the mock. When registered by name, an existing bean can be specifically targeted for replacement by a mock. In either case, if no existing bean is defined a new one will be added. Dependencies that are known to the application context but are not beans (such as those `registered directly`) will not be found and a mocked bean will be added to the context alongside the existing dependency.

When `@MockBean` is used on a field, as well as being registered in the application context, the mock will also be injected into the field.
https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/test/mock/mockito/MockBean.html

Testing

What are the different web environment options that can be used with the @SpringBootTest annotation in Spring Boot? (select 2)

- ➔ **DATABASE**
- ➔ **DEBUG**
- ➔ **RANDOM_PORT**
- ➔ **MOCK**

RANDOM_PORT & MOCK

Using `@SpringBootTest(webEnvironment = WebEnvironment.MOCK)` loads a web application context and provides a mock web environment. It doesn't load a real http server, just mocks the entire web server behavior.

`WebEnvironment.MOCK` gives you some advantages like ease of use or isolation of other factors but it might not be a good integration test practice.

Integration tests should be as similar as possible to the production environment. Considering this, using `@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)` would be a better choice. This approach is closer to test the real application. You can see whether the whole system is going to work as expected.

An enumeration of web environment modes:

1) `DEFINED_PORT`

2) `MOCK`

3) `NONE`

4) `RANDOM_PORT`
https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/test/context/SpringBootTest.WebEnvironment.html

Spring Actuator

Which of the following are the health statuses provided by SpringBoot Actuator? (select 4)

- ➔ **UP**
- ➔ **RUNNING**
- ➔ **DOWN**
- ➔ **OUT_OF_SERVICE**
- ➔ **UNKNOWN**
- ➔ **NOT_DEFINED**

UNKNOWN, OUT_OF_SERVICE, DOWN, UP

By default, Spring Boot defines four different values as the health *Status*:

- *UP* — The component or subsystem is working as expected
- *DOWN* — The component is not working
- *OUT_OF_SERVICE* — The component is out of service temporarily
- *UNKNOWN* — The component state is unknown

https://www.baeldung.com/spring-boot-health-indicators#5-health-status

https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/actuate/health/Status.html

https://bit.ly/2v7222

https://spring-book.mystrikingly.com