Annex: Description of my tasks:

As a Senior Java developer, I develop, test and deploy applications.

1. Development

a. Configurations

I start with the project definition and configurations. For that, I use some config files:
*- **A POM file** where I define the project name, version and packaging type. After that, I add a list of all dependencies I will need for my project. Usually, dependencies used are spring boot starter, spring core, spring boot starter web, actuator, json, log4j2, Lombok, apache commons, springdoc openapi, spring boot test, Junit, Mockito, cucumber. Many other dependencies can be added if needed. After the list of dependencies, we define the configuration for the various phases of the build lifecycle.*
*- **A YAML or PROPERTIES file** where I define values for all config variables like profile, environment variables, servers parameters, local variables, retry, etc... Note that you can define multiple yml/properties files for different purposes/environments eg: application.yml, application-test.yml, application-dev.yml, application-uat.yml and application-prod.yml. In that case, you use spring.profiles.active: or spring.profiles.include: to set which yml file you want to use.*
*- **A CONFIG class** where we have these annotations: @Configuration and @PropertySource("classpath:application.yml"). In the main class, the annotation @SpringBootApplication finds the ConfigProperties class, even though we didn't annotate this class with @Component. We can use @ConfigurationPropertiesScan annotation points to diff package.*

b. Development

Now we create our differentes packages and classes:

*- **Entities:** entities classes represente database objects. Annotations that can be used here are: @Entity, @Data or @Getters and @Setters, @Id, @GeneratedValue(strategy=….), @Tabe(name="....."), @Column(name="…", length=.., nullable=…, unique=….), @Transient, @OneToOne, @ManyToOne, @ManyToMany, @JoinColumn(name = "....")*

*- **DTOs**: These are objects manipulated in methods and can be mapped to entities. DTOs are not persisted in the database.*

*- **Repositories**: they are repositories of the Entities we created if we use Spring Data. The annotation @repository can be used on this class but it optional. It indicate that the class provides the mechanism for storage, retrieval, search, update and delete operation on objects. This class usually extends CrudRepository, PagingAndSortingRepository or JpaRepository.*

*- **Mappers**: these interfaces are used to define a mapping between Entities, DTOs and XML objects. We use mapstruct for that. Annotation used at class level is @Mapper. Other annotations example:*
@Mapping(target = "id", ignore = true)
@Mapping(target = "creationDate", expression = "java(new java.util.Date())")
@Mapping(target = "name", source = "firstName")

*- **Config classes**: we need to write some config classes like Listeners, Messages Queues configs, etc.. Annotation used here can be* @EventListener, @KafkaListener(id = "….", topics = "myTopic"), @KafkaHandler, @RabbitListener(queues = "myQueue"), @RabbitHandler

- **Interfaces**: interfaces are used for **loose coupling between classes, polymorphism** and to achieve multiple inheritances. It can also facilitate testing by mocking or stubbing an interface and isolate the classes that depend on it.

- Services: *these classes contains methods for the business functionalities that manipulate DTOs and other input to achieve a goal defined. This class is like a factory. The annotations used here are @Service, @Autowired,* @Transactional …

- Controllers: *these classes are responsible for handling incoming HTTP requests and returning an appropriate response. Annotation used here are: @Controller or @RestController, @RequestMapping(value = "..." , @GetMapping, @PostMapping, @DeleteMapping, @PutMapping, @RequestParam, @PathVariable, @ResponseBody*
Note that we can use Swagger and OpenAPI specifications for the definition, generation and documentation of components/DTOs and endpoints.

2. Testing

To be sure I deliver quality code, I need multiple stage of testing.

a. ***Unit tests****: This means testing the methods of each class individually. I use Junit for that. Annotation example:* @Test, @Before, @After, …
b. ***Integration and functional tests****: It verifies that different components work together seamlessly, interact correctly as expected and output are as expected. We use testing frameworks like spring boot test, Mockito, Selenium, TestNG, and Cucumber.*
   - Mockito Annotation example: @RunWith, @Mock, @InjectMocks, @Spy, @Captor
   Best pratice to use the Mockito verify method at the end of the test.
   - Cucumber Annotation example: @Given, @And, @When, @Then
   - TestNG Annotation example: @BeforeTest, @BeforeClass, @BeforeMethod
c. ***End-to-end testing****: this step is to replicates a user behavior with the software in a complete application environment. It verifies that various user flows work as expected.I use POSTMAN for this testing.*

3. Database

I write SQL scripts to create or update a database. Scripts are in files with named with sql extension. These files are versioned incrementally. These files are created for each environment (local, dev, test, prod).
I am confortable with databases like MySQL, Oracle, PostgreSQL, IBM Db2, *MongoDB, Couchbase, Amazon RDS…*
I also use tools like Dbeaver.

4. CI/CD DevOps

I have 10+ years with CI/CD pipelines using tools like Gitlab, Jenkins, Bamboo, Dockers, Kubernetes, Istio, Argo CD, Helm, AWS (codebuild, CodePipeline, CodeDeploy, cloudwatch, ec2, S3) and (DevOps, Pipelines),

5. Code review

Code review is part of my daily tasks. I review merge requests from colleagues according to clean code, SOLID principles, security, speed and performance.

6. Documentation

I update documentations on Confluence when it needed. I have 10+ years experiences with creating and updating documentations on Confluence.