Q: What is Spring Boot, and how does it differ from the Spring Framework?

A: Spring Boot is an opinionated framework built on top of the Spring Framework that simplifies the development of Java applications. It aims to provide a convention-over-configuration approach, making it easier to set up and configure Spring applications. Unlike the Spring Framework, which requires manual configuration of various components, Spring Boot offers auto-configuration and embeds an application server, reducing the need for explicit configuration.

Q: How do you create a new Spring Boot application using Spring Initializr?

A: To create a new Spring Boot application using Spring Initializr, you can follow these steps:

Visit the Spring Initializr website (start.spring.io).
Choose the desired project options such as language (Java or Kotlin), Spring Boot version, and project metadata.
Select the necessary dependencies, such as Spring Web for RESTful APIs or Spring Data JPA for database access.
Click "Generate" to download a zip file containing the project skeleton.
Extract the downloaded zip file and import it into your favorite IDE as a Maven or Gradle project.
Start building your Spring Boot application.

Q: What is the purpose of the @SpringBootApplication annotation?

The *@SpringBootApplication* annotation is a convenience annotation that combines three commonly used annotations in a Spring Boot application: *@Configuration*, *@EnableAutoConfiguration*, and *@ComponentScan*. It indicates that the annotated class is the primary Spring Boot application class and enables the auto-configuration and component scanning features provided by Spring Boot.

Q: How does Spring Boot simplify the configuration of database connections?

Spring Boot simplifies the configuration of database connections through its auto-configuration feature. By including a specific database driver dependency on the classpath and providing the necessary configuration properties, Spring Boot can automatically configure a DataSource bean and manage the database connection. It uses sensible default settings but also allows customization through application properties or YAML files.

Q: What is the significance of the application.properties or application.yml file in a Spring Boot application?

The *application.properties* or *application.yml* file is used for externalizing configuration in a Spring Boot application. It allows developers to specify various settings, such as database connection details, server port, logging configuration, and more. These files provide a centralized place to configure the application, and Spring Boot automatically loads and applies the properties defined in these files during application startup.

Q: What is the purpose of the @RestController annotation in Spring Boot? A: The *@RestController* annotation is used to mark a class as a specialized version of the *@Controller* annotation. It combines the *@Controller* and *@ResponseBody* annotations, indicating that the annotated class is responsible for handling incoming HTTP requests and producing the response directly as the HTTP response body. It is commonly used in building RESTful APIs in Spring Boot.

Q: Explain the role of the @Autowired annotation in Spring Boot. A: The ***@Autowired*** annotation is used for automatic dependency injection in Spring Boot. When applied to a field, constructor, or setter method, Spring Boot will automatically resolve and inject the appropriate bean dependency at runtime. It eliminates the need for manual bean wiring and promotes loose coupling between components.

Q: How does Spring Boot support the creation of RESTful APIs? A: Spring Boot provides several features to simplify the creation of RESTful APIs, including:

The ***@RestController*** annotation for building REST controllers.
Automatic serialization and deserialization of request/response bodies using Jackson or other JSON libraries.
Support for request mapping annotations (***@RequestMapping***, ***@GetMapping***, ***@PostMapping***, etc.) to define the API endpoints and HTTP methods.
Built-in exception handling mechanisms to return proper HTTP responses for various scenarios.
Integration with Spring Data and JPA for easy database access and persistence.

Q: What is Spring Boot Actuator, and what are its main features? A: Spring Boot Actuator is a module in Spring Boot that provides several production-ready features to monitor and manage your application. Its main features include:

Health checks: Provides information about the health of the application.
Metrics: Collects and exposes various metrics about the application, such as memory usage, CPU usage, request counts, etc.
Auditing: Tracks application events and provides an audit trail.
Environment details: Exposes information about the application's environment, configurations, and properties.
Request tracing: Provides tracing information for HTTP requests.
Log file management: Supports viewing and managing application log files.
Endpoint customization and security: Allows customization and securing of Actuator endpoints.

Q: How can you implement authentication and authorization in a Spring Boot application?

Authentication and authorization can be implemented in a Spring Boot application using various techniques:

Spring Security: It provides a comprehensive security framework that supports authentication, authorization, and session management.
JSON Web Tokens (JWT): You can integrate JWT-based authentication and authorization by using libraries such as ***jjwt*** to generate and validate tokens.

Q: How does Spring Boot handle error and exception handling?

Spring Boot provides error and exception handling mechanisms to handle various types of errors and exceptions gracefully. It includes:

Global exception handling: You can define a global exception handler using the ***@ControllerAdvice*** annotation to handle exceptions across multiple controllers.

Custom error pages: Spring Boot can render custom error pages for specific HTTP error codes or exceptions.

@ExceptionHandler: You can annotate specific methods within a controller with ***@ExceptionHandler*** to handle exceptions thrown by those methods.

Default error handling: Spring Boot provides default error handling that returns standardized JSON or HTML error responses.