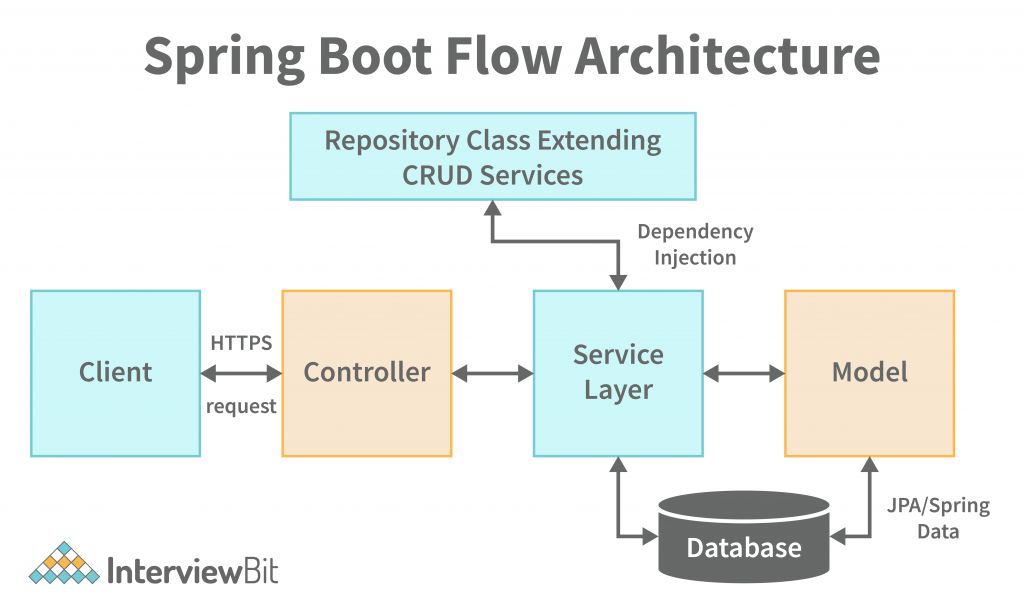
**SPRING BOOT**

Spring Boot is an open-source Java-based framework used to create stand-alone, production-grade Spring-based applications with minimal configuration. It simplifies the process of building and deploying applications by providing a convention-over-configuration approach, auto-configuration, and embedded servers.

* **Auto-Configuration:** Spring Boot automatically configures various components based on the dependencies present in the classpath. Developers can override the default configurations as needed.
* **Standalone Applications:** Spring Boot applications can be deployed as standalone JAR files, which include an embedded servlet container (e.g., Tomcat, Jetty, Undertow). This eliminates the need for external application servers.
* **Spring Boot Starters:** Starters are pre-configured dependencies that simplify the inclusion of commonly used libraries and frameworks (e.g., Spring MVC, Spring Data, Spring Security) in Spring Boot applications.
* **Actuator:** Spring Boot Actuator provides production-ready features such as health checks, metrics, monitoring, and endpoint exposure over HTTP or JMX.
* **Spring Boot CLI:** The Spring Boot Command-Line Interface allows developers to quickly create, test, and run Spring Boot applications using Groovy scripts.
* **Externalized Configuration:** Spring Boot allows configuration properties to be externalized, supporting various formats like properties files, YAML files, environment variables, and command-line arguments.
* **Spring Boot DevTools:** DevTools enhance developer productivity by providing features like automatic application restarts, live reloading, and remote debugging.

**ARCHITECTURE**

Spring Boot builds upon the Spring Framework, leveraging its core features such as dependency injection, aspect-oriented programming, and transaction management. However, it provides additional features to streamline application development and deployment.



**COMPONENTS**

* **Spring Boot Starter:** Starters are a set of convenient dependency descriptors that simplify dependency management. They include everything needed to configure a specific feature or functionality in the application.
* **Spring Boot Auto-Configuration:** Auto-configuration automatically configures Spring beans based on the dependencies present in the classpath. It reduces the need for manual configuration and boilerplate code.
* **Embedded Server:** Spring Boot includes embedded servlet containers (Tomcat, Jetty, or Undertow) which eliminate the need for deploying applications to external servers. Developers can package applications as executable JAR files.
* **Spring Boot Actuator:** Actuator provides production-ready features to monitor and manage Spring Boot applications. It exposes endpoints for health checks, metrics, environment information, and more.
* **Spring Boot CLI:** The Spring Boot Command-Line Interface allows developers to quickly create, test, and run Spring Boot applications using Groovy scripts. It provides commands for project initialization, dependency management, and application execution.

**SPRING LOGGING**

Logging is an essential aspect of any application, including those built with Spring Boot. Proper logging helps developers diagnose issues, monitor application behavior, and track down errors in production environments. Spring Boot provides robust logging support, allowing developers to configure and customize logging behavior according to their requirements.

**1. Logging Frameworks:** Spring Boot integrates with popular logging frameworks, including:

* **Logback:** The default logging framework in Spring Boot. It offers advanced features such as configuration through XML or Groovy files, appenders, log rotation, and logging levels customization.
* **Log4j2:** An alternative logging framework known for its performance and configuration capabilities. It supports asynchronous logging, plugins, and various appenders for output customization.
* **Java Util Logging (JUL):** The built-in logging framework in Java. While less commonly used in Spring Boot applications, it's still supported for those preferring its simplicity.

**2. Default Logging Configuration:** Spring Boot includes default logging configuration, which can be overridden or customized based on project requirements. By default, logs are written to the console with INFO level logging.

**3. Customizing Logging Configuration:** Spring Boot allows developers to customize logging configuration through properties files or YAML files.

**4. Logging Output:** Spring Boot supports various logging output formats:

* **Console:** Logs are printed to the console by default.
* **File:** Logs can be written to files with configurations for rotation, file size limits, and retention policies.
* **Remote:** Logs can be sent to remote servers or log aggregation services like Logstash, Splunk, or ELK stack (Elasticsearch, Logstash, Kibana).

**5. Loggers and Log Levels:** Spring Boot applications use loggers to generate log messages. Log levels determine the severity of log messages and include TRACE, DEBUG, INFO, WARN, ERROR, and FATAL. Developers can control the level of logging for different parts of the application to manage verbosity and focus on critical information.