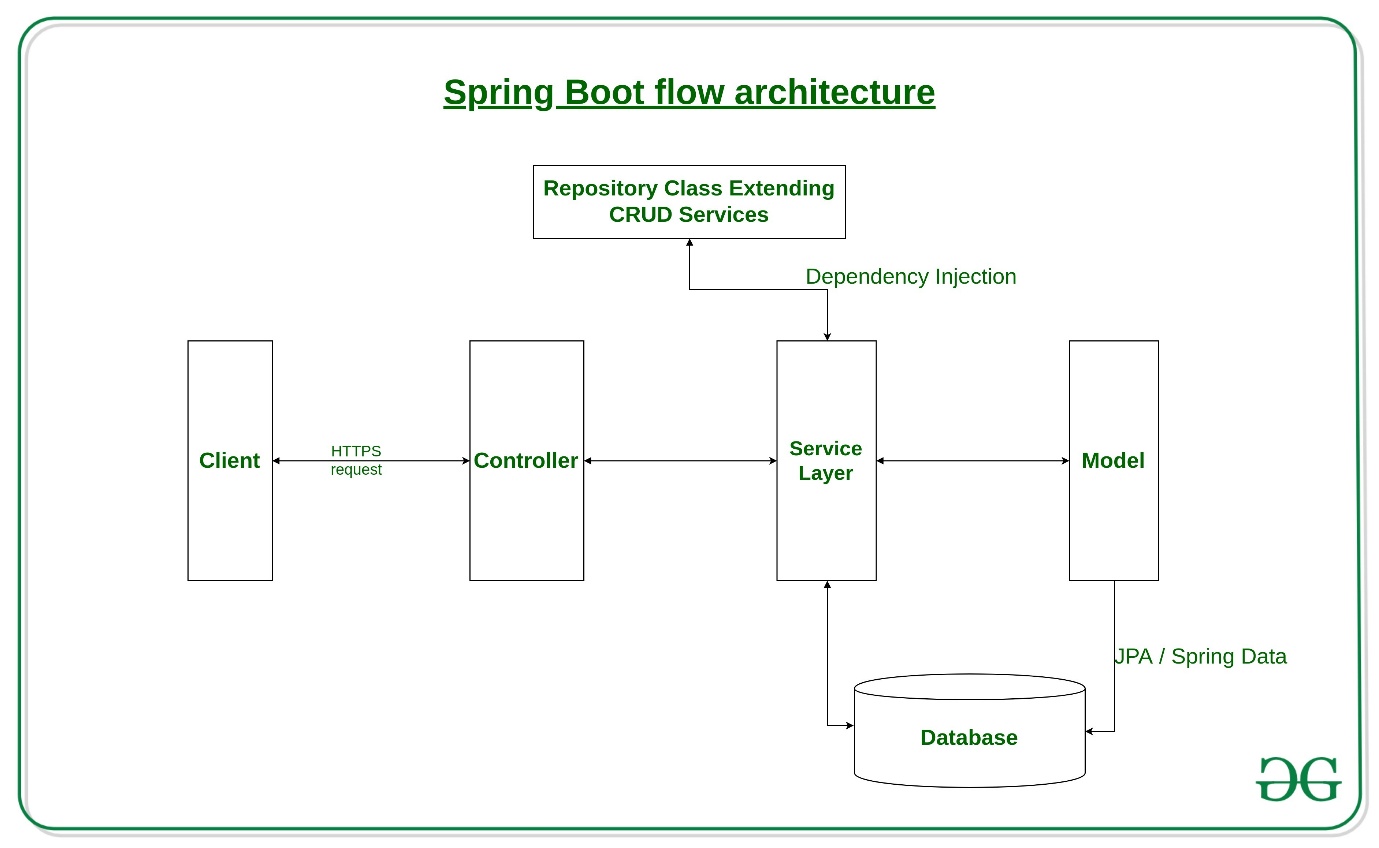
**Spring Boot Architecture**

**Layers in Spring Boot**

* **Client Layer:**
  + This represents the external system or user that interacts with the application by sending HTTPS requests.
* **Controller Layer (Presentation Layer):**
  + Handles incoming HTTP requests from the client.
  + Processes the request and sends a response.
  + Delegates business logic processing to the Service Layer.
* **Service Layer (Business Logic Layer)**:
  + Contains business logic and service classes.
  + Communicates with the Repository Layer to fetch or update data.
  + Uses Dependency Injection to get required repository services.
* **Repository Layer (Data Access Layer)**:
  + Handles CRUD (Create, Read, Update, Delete) operations on the database.
  + Extends Spring Data JPA or other persistence mechanisms.
* **Model Layer (Entity Layer):**
  + Represents database entities and domain models.
  + Maps to tables in the database using JPA/Spring Data.
* **Database Layer:**
  + The actual database that stores application data.
  + Spring Boot interacts with it through JPA/Spring Data.

**Spring Boot Flow Architecture**



**Request Flow in Spring Boot**

* A Client makes an HTTPS request (GET/POST/PUT/DELETE).
* The request is handled by the Controller, which is mapped to the corresponding route.
* If business logic is required, the Controller calls the Service Layer.
* The Service Layer processes the logic and interacts with the Repository Layer to retrieve or modify data in the Database.
* The data is mapped using JPA with the corresponding Model/Entity class.
* The response is sent back to the client. If using Spring MVC with JSP, a JSP page may be returned as the response if no errors occur.
* It allows for avoiding heavy configuration of XML which is present in the spring
* It provides easy maintenance and creation of REST endpoints
* It includes embedded Tomcat-server
* Deployment is very easy, war and jar files can be easily deployed in the Tomcat server
* **@SpringBootApplication**
* It is used to mark the main class of a Spring Boot application.
* It encapsulates **@SpringBootConfiguration**, **@EnableAutoConfiguration**, and **@ComponentScan**annotations with their default attributes.

**@SpringBootConfiguration**

It can be used as an alternative to Spring’s standard @Configuration annotation so that configuration can be found automatically.

if an application uses @SpringBootApplication, it is already using @SpringBootConfiguration.

**@EnableAutoConfiguration**

This annotation auto-configures the beans that are present in the classpath.

Spring boot auto-configures [Tomcat](https://www.geeksforgeeks.org/embedding-tomcat-server-in-maven-project/), and [Spring MVC](https://www.geeksforgeeks.org/difference-between-spring-mvc-and-spring-boot/)

**@ComponentScan**

@ComponentScan tells Spring in which packages you have annotated classes that should be managed by Spring.

if you have a class annotated with [@Controller](https://www.geeksforgeeks.org/spring-controller-annotation-with-example/)which is in a package that is not scanned by Spring, you will not be able to use it as a Spring controller.

@ComponentScan enables Spring to scan for things like configurations, controllers, services, and other components that are defined.

@ComponentScan annotation is used with @Configuration annotation to specify the package for Spring to scan for components.

**@Controller**

It is used to create Controller classes and simultaneously it handles the HTTP requests.

**@RestController**

This annotation is used to handle REST APIs such as GET, PUT, POST, DELETE etc. and also used to create RESTful web services using Spring MVC.

It encapsulates **@Controller**annotation and **@ResponseBody**annotation with their default attributes.

**@RequestMapping**

It is used to map the HTTP requests with the handler methods inside the controller class.

For handling specific HTTP requests we can use

* @GetMapping
* @PutMapping
* @PostMapping
* @PatchMapping
* @DeleteMapping
* **@RequestParam**

This annotation is basically used to obtain a parameter from URI.

In other words, we can say that **@RequestParam**annotation is used to read the form data and binds the web request parameter to a specific controller method.

**@PathVariable**

**This annotation is used to extract the data from the URI path. It binds the URL template path variable with method variable.**

**@RequestBody**

**This annotation is used to convert HTTP requests from incoming JSON format to domain objects directly from request body.**

**@ResponseBody**

**is annotation is used to convert the domain object into HTTP request in the form of JSON or any other text. Here, the return type of the method binds with the HTTP response body.**

**@ModelAttribute**

**This annotation refers to model object in Spring MVC. It can be used on methods or method arguments as well.**

**@ModelAttribute("author")  
public Author author(){  
 //insert code here  
}**

**1. Presentation Layer**

**The Presentation Layer is the topmost layer of the Spring Boot architecture. It primarily consists of REST controllers that handle HTTP requests (GET, POST, PUT, DELETE). It performs authentication, request validation, and JSON serialization/deserialization (conversion of JSON to Java objects and vice versa). After processing the request, it forwards the request to the business layer.**

**2. Business Layer**

**The Business Layer is responsible for implementing the application's core logic. It consists of service classes that:**

* **Process and validate data.**
* **Handle authentication and authorization (integrating Spring Security if needed).**
* **Apply transaction management using @Transactional.**
* **Interact with the Persistence Layer to store or retrieve data.**

**3. Persistence Layer**

**The Persistence Layer manages database transactions and storage logic. It consists of repository classes using Spring Data JPA, Hibernate, or R2DBC for data access. It is responsible for:**

* **Mapping Java objects to database records using ORM frameworks.**
* **Managing**[**CRUD**](https://www.geeksforgeeks.org/spring-boot-crud-operations/)**(Create, Read, Update, Delete) operations.**
* **Supporting relational and NoSQL databases.**

**4. Database Layer**

**The Database Layer contains the actual database where the application data is stored. It can support:**

* **Relational Databases (MySQL, PostgreSQL, Oracle, SQL Server).**
* **NoSQL Databases (MongoDB, Cassandra, DynamoDB, Firebase).**
* **Cloud-based databases for scalability.**

**Spring Boot's Actuator dependency is used to monitor and manage the Spring web application.**

* **We can use it to monitor and manage the application with the help of HTTP endpoints or with JMX.**

| **EndpointID** | **Description** |
| --- | --- |
| **beans** | **Displays a complete list of all the Spring beans in your application.** |
| **caches** | **Exposes available caches.** |
| **conditions** | **Shows conditions evaluated on configuration and auto-configuration classes.** |
| **health** | **Shows application health information.** |
| **httptrace** | **Displays HTTP trace information (last 100 requests).** |
| **loggers** | **Shows and modifies the configuration of loggers in the application.** |
| **mappings** | **Displays a collated list of all @RequestMapping paths.** |
| **sessions** | **Retrieves and deletes user sessions (requires Spring Session).** |
| **threaddump** | **Performs a thread dump.** |

**Important Methods of HTTP**

**The main methods of HTTP we build web services for are:**

1. **GET: Reads existing data.**
2. **PUT: Updates existing data.**
3. **POST: Creates new data.**
4. **DELETE: Deletes the data.**

**HTTP Standard Status Codes**

**The status codes defined in HTTP are the following:**

* **200: Success**
* **201: Created**
* **401: Unauthorized**
* **404: Resource Not Found**
* **500: Server Error**

**HTTP PUT request is used to replace and update the entire resource or document, while the PATCH request only updates the specific parts of that document.**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **SPRING MVC** | **SPRING BOOT** |
| **1.** | **Spring MVC is a Model View, and Controller based web framework widely used to develop web applications.** | **Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs.** |
| **2.** | **If we are using Spring MVC, we need to build the configuration manually.** | **If we are using Spring Boot, there is no need to build the configuration manually.** |
| **3.** | **In the Spring MVC, a deployment descriptor is required.** | **In the Spring Boot, there is no need for a deployment descriptor.** |
| **4.** | **Spring MVC specifies each dependency separately.** | **It wraps the dependencies together in a single unit.** |
| **5.** | **Spring MVC framework consists of four components : Model, View, Controller, and Front Controller.** | **There are four main layers in Spring Boot: Presentation Layer, Data Access Layer, Service Layer, and Integration Layer.** |
| **6.** | **It takes more time in development.** | **It reduces development time and increases productivity.** |
| **7.** | **Spring MVC do not provide powerful batch processing.** | **Powerful batch processing is provided by Spring Boot.** |
| **8.** | **Ready to use feature are provided by it for building web applications.** | **Default configurations are provided by it for building a Spring powered framework.** |

**Exception Handling**

**An exception is an event, which occurs during the execution of a program, that interrupts the normal flow of the program's**

**instructions.**

What is “handle or declare” rule?

Ans:- a) whenever any method raises **checked** exception/s , method has to either **handle [try….catch]** or **declare [throws]** that checked exception/s.

b) whenever u invoke a method which has **declared [using throws] checked exception/s** , caller method has to either **handle [try….catch] or declare [throws]** that checked exception/s.

**throws keyword is used to "declare" exception.**

**what do you mean by "declaring" exception ?**

**"Declaring" exception means propagating it to the caller and then it's caller's resposibility to "handle or declare".**

| **Feature** | **throw** | **throws** |
| --- | --- | --- |
| **Action** | **Actually throws an exception** | **Declares method may throw exception(s)** |
| **Placement** | **Inside method or block** | **In method signature** |
| **Followed by** | **An exception object** | **Exception class names** |
| **Purpose** | **For handling actual exception occurrence** | **For informing caller about exceptions** |

**exception can be raised by**

**a) JVM**

**or**

**b) Application [ Developer ]**

**When exception is raised , it happens in two steps :**

**a) instantiation of that particular exception class**

**b) throwing that exception (that instance) to the caller.**

**Exception**

**[ unchecked exceptions ] [ checked exceptions ]**

**RuntimeException IOException**

**[ due to programmer's logical [ these can be raised in**

**mistake ] a logically correct program ]**

**[ they can be avoided using simple**

**"if....else" , so java does not enforce [ java enforces programmer to handle these ] programmer to handle**

**these]**

**NullPointerException FileNotFoundException**

**NumberFormatException EOFFileException**

**ArrayIndexOutOfBoundsException SQLException**

**ClassCastExceptionvcv**

**finally block gets executed irrespective of whether exception is raised or not.**

**it can be used to release resources such as file,socket,database connection etc. since you can not rely upon "finalized" method for**

**the same task.**

**finally block can follow after catch (try..catch...finally ) or even after try (try...finally).**

**finally block will not get executed if**

**a) System.exit(0) is called inside try ,catch or finally block.**

**b) exception gets raised in finally block itself.**