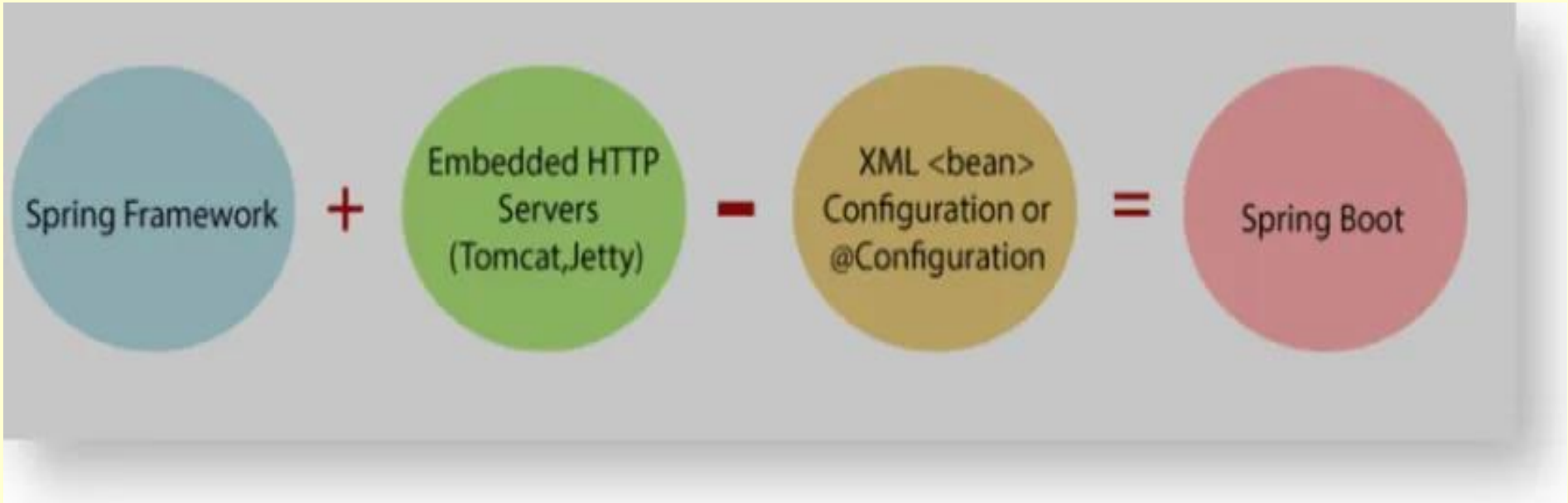
Spring Boot

Introduction

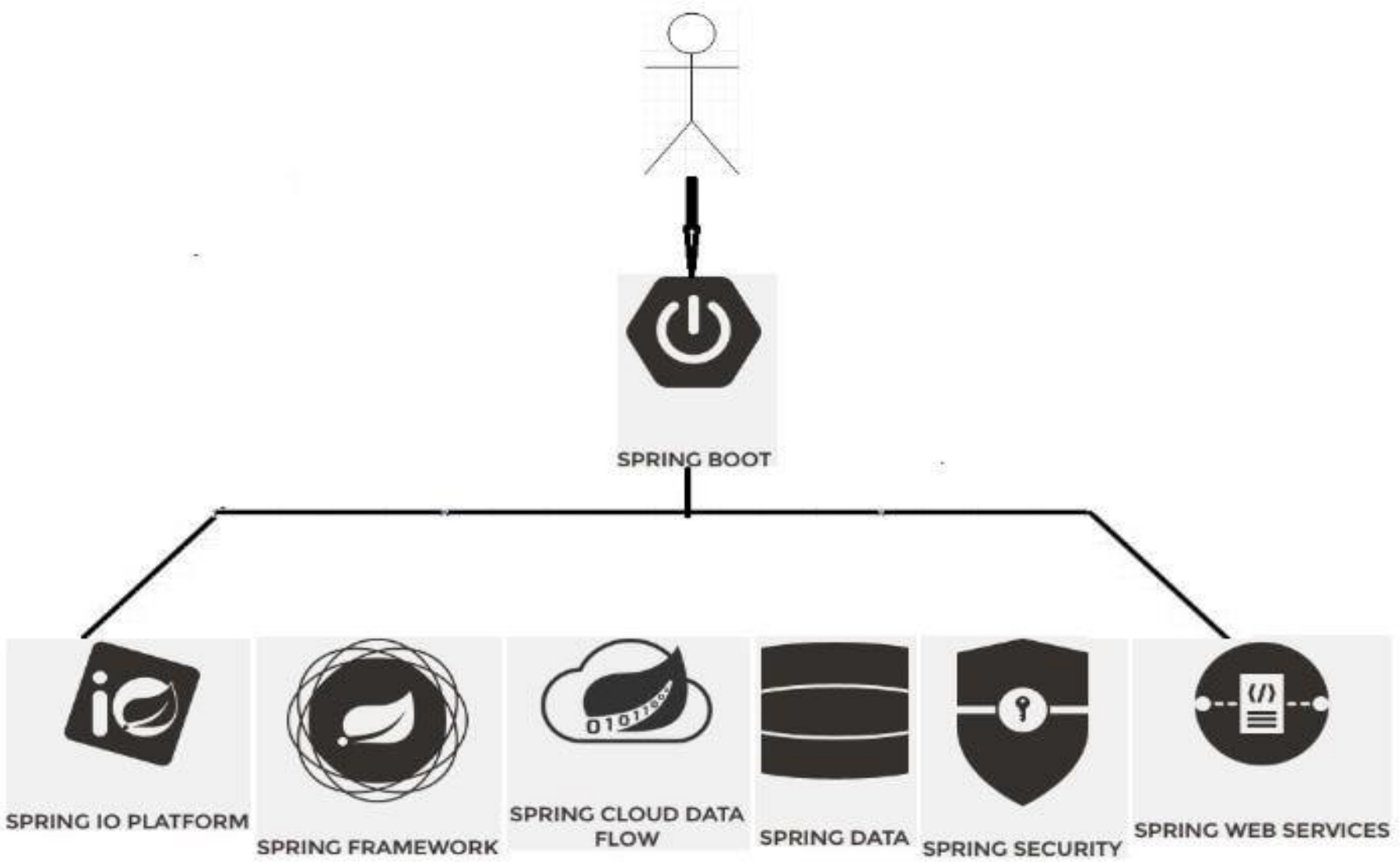
- Spring Boot is a framework from "**The Spring Team**" to ease the **bootstrapping** and **development** of new Spring Applications.
- Spring Boot uses **OPINIONATED APPROACH**.
- Spring Boot provides **defaults** for **code** and **annotation configuration** to quick start new Spring.
- Spring Boot improves –
  - ✓ Development,
  - ✓ Unit Testing and
  - ✓ Integration Test process.
- Spring Boot is **NOT IMPLEMENTED** from the start but on top of existing Spring framework.
- Spring Boot is not used for solving any new problems but same as that of Spring framework.

contd..

- Spring Boot is the combination of Spring Framework and Embedded Servers.



contd..


SPRING BOOT
SPRING IO PLATFORM
SPRING FRAMEWORK
SPRING CLOUD DATA FLOW
SPRING DATA
SPRING SECURITY
SPRING WEB SERVICES

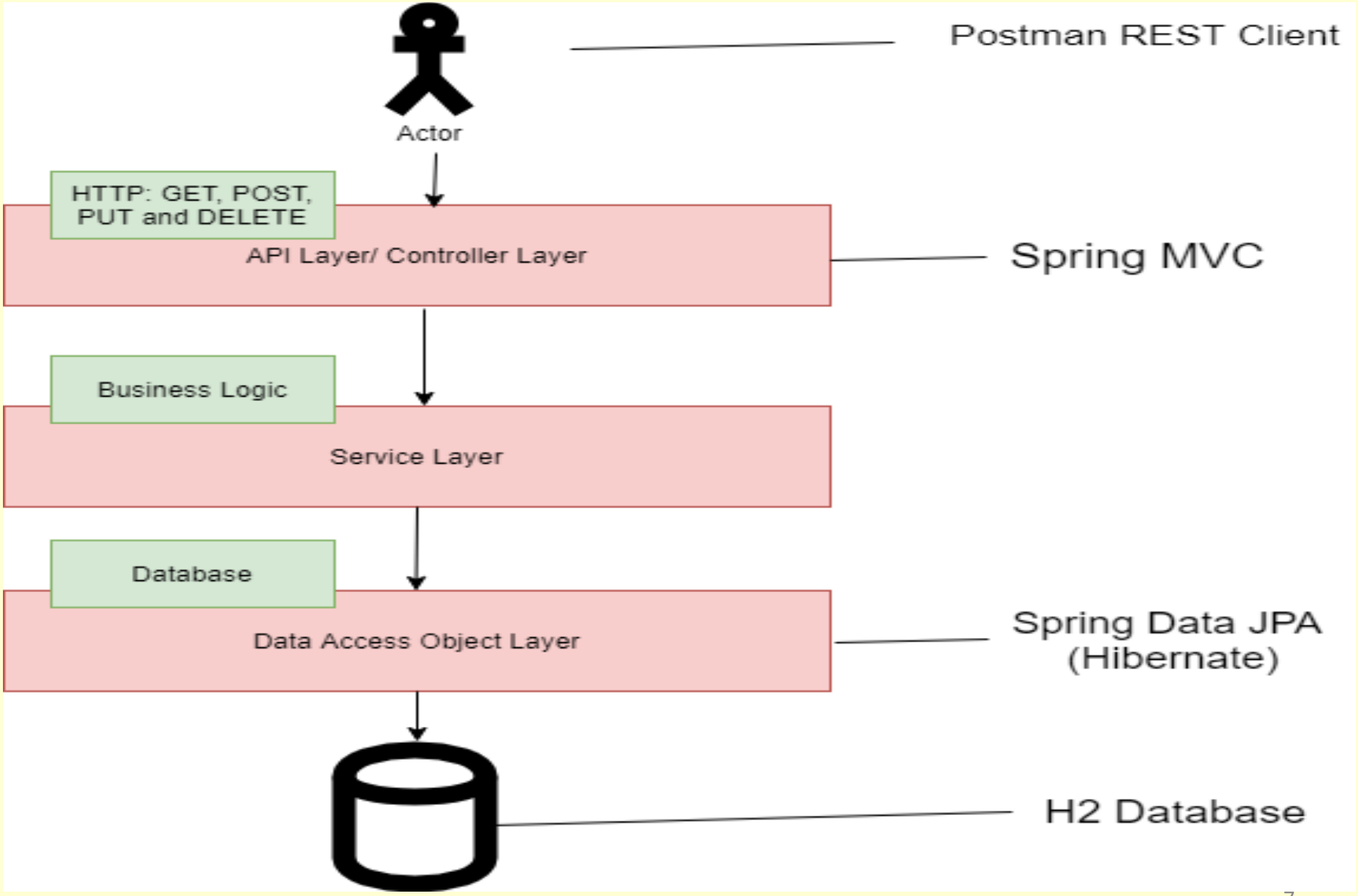
Spring Boot features

- Web Development
- Spring Application
- Application events and listeners
- Admin features
- Externalized Configuration
- Properties Files
- YAML Support
- Type-safe Configuration
- Logging
- Security

Opinionated Software

- **Opinionated** software means that there is **one-way** to do things and the framework guides the developer into their way of doing things. The **one-way** is called as **right-way™**
- **Opinionated** approach makes it easy for development as number of decisions user has to make is reduced and the ability of the software designers focus on making the software work.
- To start **Opinionated** application i.e. to create Spring Boot applications, the **Spring Team** (the **Pivotal Team**) has provided following three approaches.
  - ✓ Spring Boot CLI Tool
  - ✓ Spring STS IDE
  - ✓ Spring Initializr Website – **http://start.spring.io**

High level architecture of Spring Boot

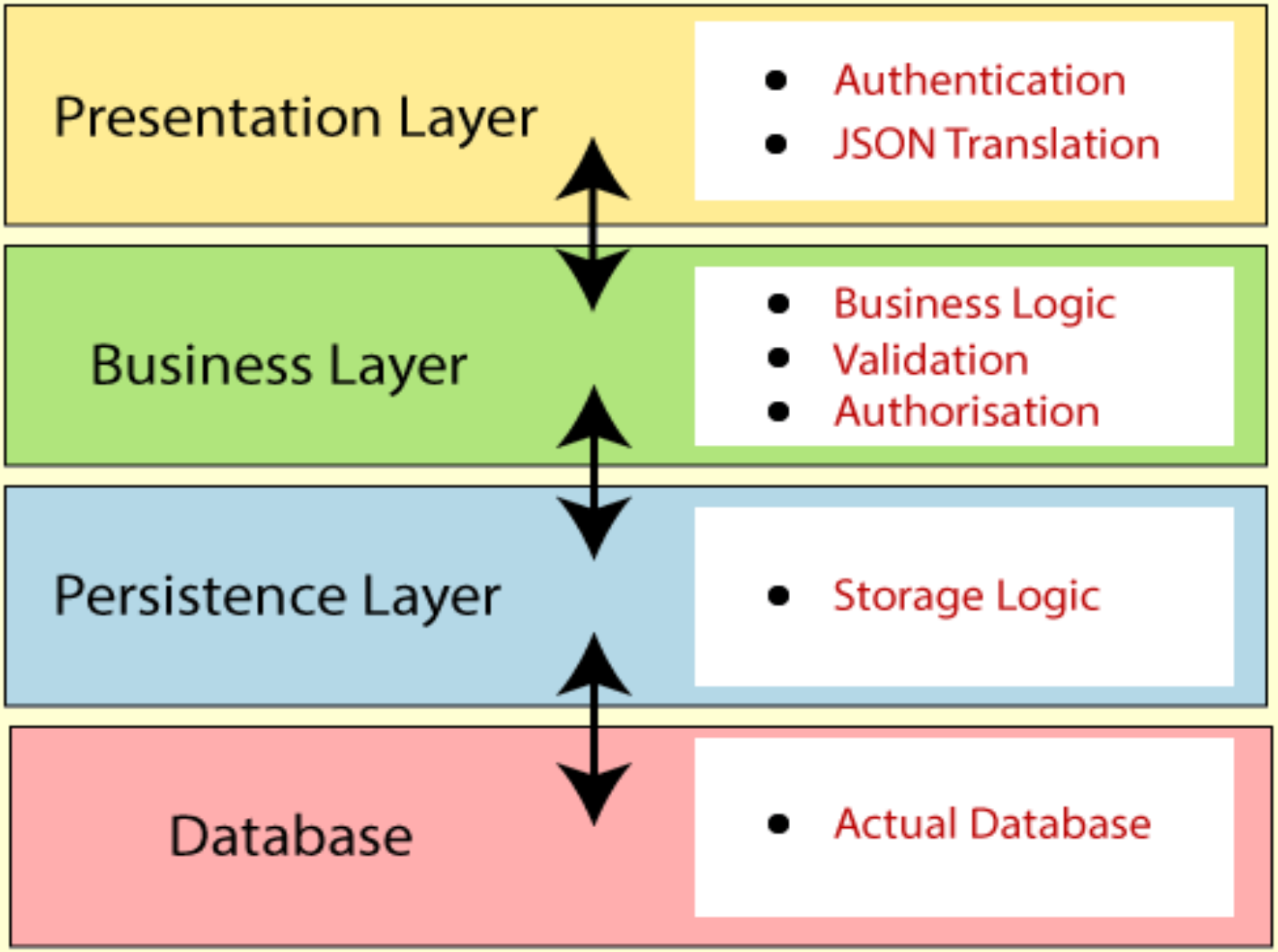


contd..

- Spring Boot provides –
  - ✓ easy way to develop **stand–alone**, **production–grade** Spring based applications. Avoids **boilerplate code, xml configs** and provides easy integration with **Spring–Jdbc, Spring–ORM, Spring Security** etc.
  - ✓ embedded **HTTP servers** like **Tomcat, Jetty** etc. to develop and test web applications very easily.
  - ✓ **plugins** to use build tools like **maven, gradle**, **CLI tool** to develop and test Spring Boot applications from **command prompt.**
  - ✓ **opinionated** view of **Spring platform** and **third party libraries** i.e. **out of the box** approach.
  - ✓ a range of **non–functional** features that are common to large classes of projects e.g. **embedded servers, security, metrics, health checks, externalized configuration** and **like**.

Spring Boot layers

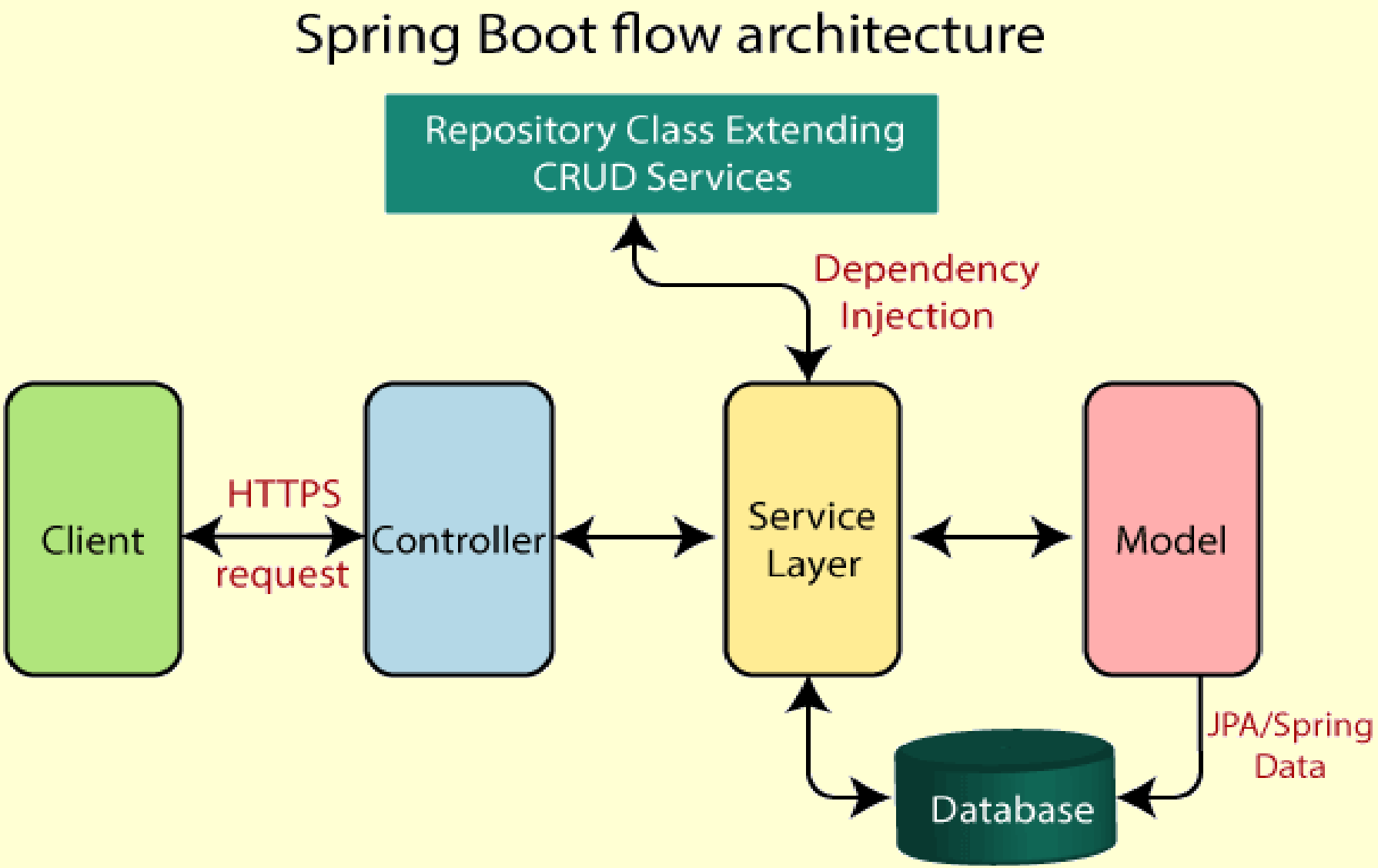
- The **four** layers in Spring Boot are –



contd..

- **Presentation Layer:** The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of **views** i.e., frontend part.
- **Business Layer:** The business layer handles all the **business logic**. It consists of service classes and uses services provided by data access layers. It also performs **authorization** and **validation**.
- **Persistence Layer:** The persistence layer contains all the **storage logic** and translates business objects from and to database rows.
- **Database Layer:** In the database layer, **CRUD** (create, retrieve, update, delete) operations are performed.

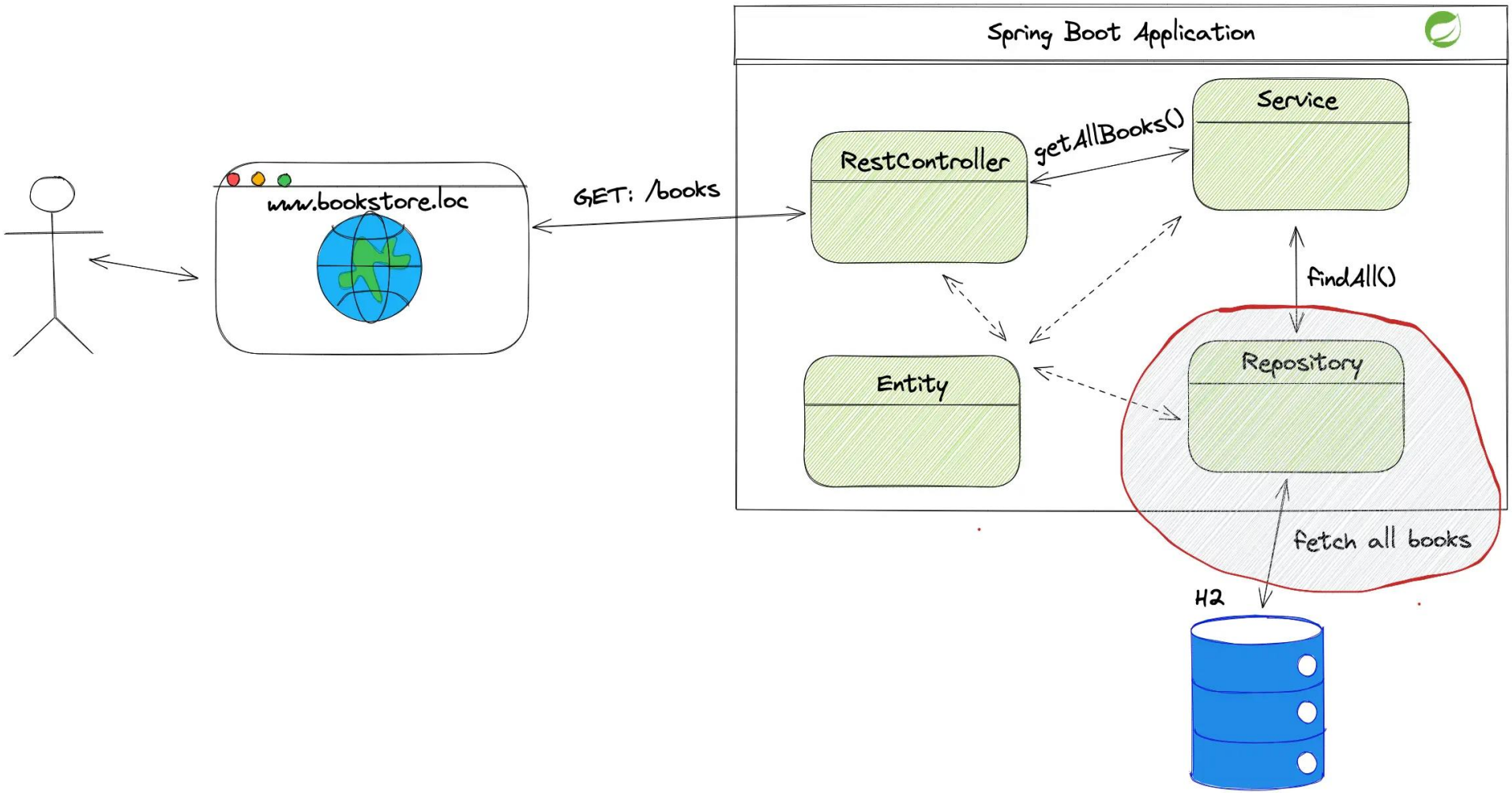
Spring Boot flow architecture



contd..

- Description –
  - ✓ The architecture of Spring Boot is the same as the architecture of Spring MVC, but no need for **DAO** and **DAOImpl** classes.
  - ✓ The components are three classes – **validator, view and utility**.
  - ✓ Spring Boot uses all Spring modules– **Spring Jdbc, Spring MVC, Spring ORM,** creates data access layer to perform **CRUD** operations.
  - ✓ The client makes the **HTTP requests** (**PUT** or **GET**).
  - ✓ The **request** goes to the **controller**, and the **controller maps** that **request** and **handles** it. After that, it calls the **service logic**.
  - ✓ In the **service layer**, all the **business logic** performs **operations** on the data that is mapped to JPA with model classes.
  - ✓ A JSP page is returned to the user if no error occurred.

contd..


Spring Boot Application
Service
RestController
getAllBooks()
www.bookstore.loc
GET: /books
findAll()
Entity
Repository
fetch all books
H2

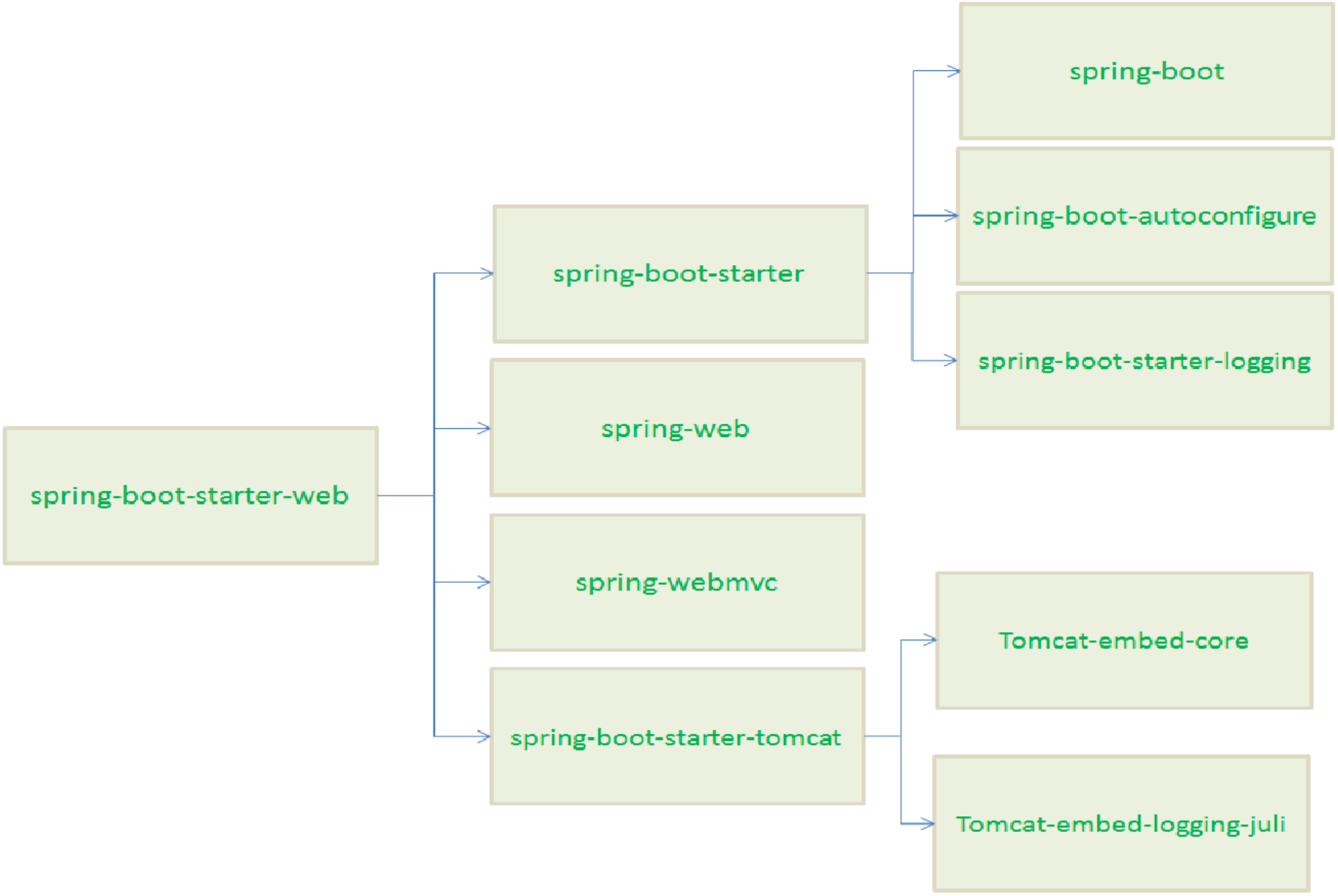
Components of Spring Boot

- Key components of Spring Boot framework –
  - ✓ Spring Boot Starter
  - ✓ AutoConfigurator
  - ✓ CLI
  - ✓ Actuator
  - ✓ Initilizr
  - ✓ IDEs

Spring Boot Starter

- **Spring Boot starter** simplifies project build dependencies by **combining** a group of **common** or **related** dependencies into **single** dependency**.** Offer **convenient dependency descriptors** that user can include in the application.
- **Spring Boot starter** contain a lot of dependencies that user needs to get a project up and running quickly and with a **consistent, supported set of managed transitive dependencies**.
- Developer gets a one–stop shop for all the Spring and related technologies that are needed without having to hunt through sample code and copy–paste loads of dependency descriptors e.g. **spring–boot–starter–data–jpa** includes Spring and JPA.
- **spring–boot–starter–web.jar** automatically downloads all the required jars and adds them to the project classpath.
- **spring–boot–starter–logging.jar** file loads all its' dependency jars like **jcl–over–slf4j, jul–to–slf4j, log4j–over–slf4j, logback– classic** to the project classpath.

contd..


spring-boot-starter-web
spring-boot-starter
spring-web
spring-webmvc
spring-boot-starter-tomcat
spring-boot
spring-boot-autoconfigure
spring-boot-starter-logging
Tomcat-embed-core
Tomcat-embed-logging-juli

contd..

- "**main() method**" – is the final part of application, a standard that follows the Java convention for an application entry point.

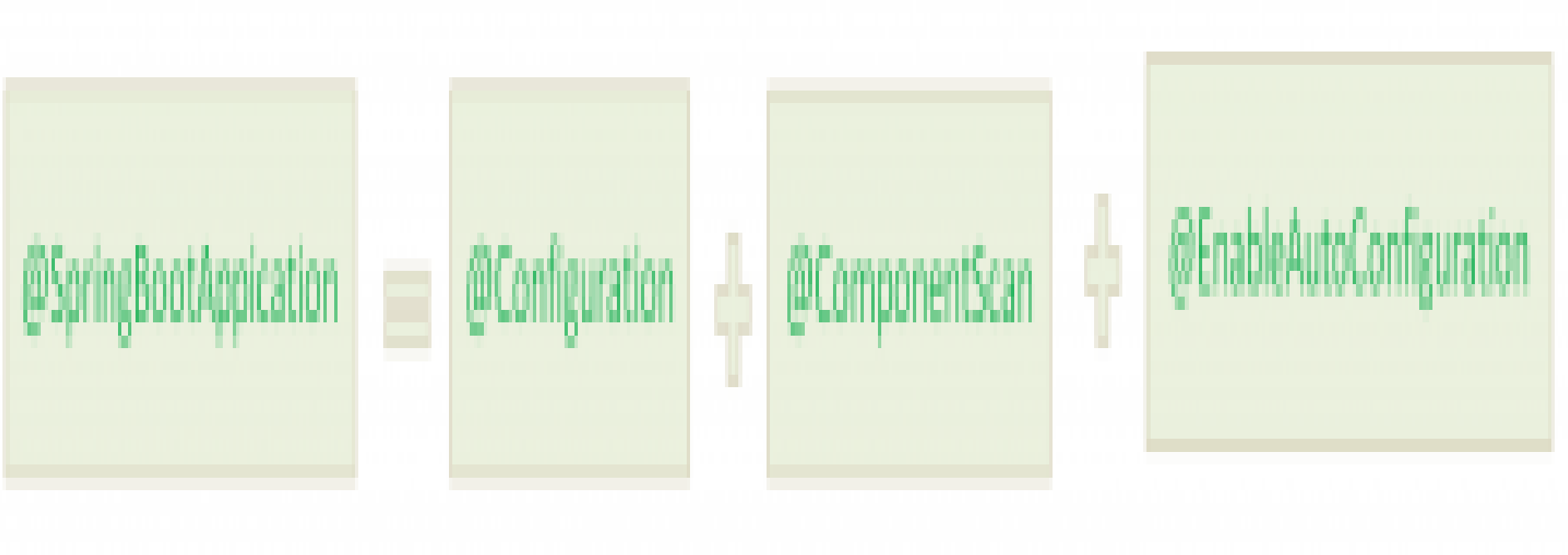
  1. The main method delegates to Spring Boot's **SpringApplication** class by calling **run()**.
  2. **SpringApplication** bootstraps the application by starting Spring which in turn, starts the auto-configured Tomcat web server.

- The class of main() method is passed as an argument to the run() method to tell **SpringApplication** which is the primary Spring component.
- The args array is also passed through to expose any command-line arguments.

Auto-Configurator

- **Auto-Configurator** is an important key component of SB framework providing solution to the problem of writing lot of configurations. Aims to **automatically configure** application based on the jar dependencies.
- Auto-Configurator **reduces** the **Spring configuration** as there is **no need to define single XML config** and **almost no** or **min Annotation** config. AutoConfigurator component takes care of providing these information.
- The **spring-boot-starter-web.jar** in the project build file, will resolve views, view resolvers etc. **automatically**.
- The **@SpringBootApplication** is combo of @Configuration, @ComponentScan and @EnableAutoConfiguration. AutoConfigurator will **automatically add** all required annotations to Java class ByteCode.
- If HSQLDB is on the classpath, and is not manually configured any database connection beans, then Spring Boot auto-configures an in-memory database.

contd..



@SpringBootApplication = @Configuration + @ComponentScan
+ @EnableAutoConfiguration.

Spring Boot annotations

1. **@EnableAutoConfiguration** – This annotation tells Spring Boot to "guess" i.e. how user wants to configure Spring, based on the jar dependencies that have been added. E.g. if spring–boot– starter–web added Tomcat and Spring MVC, the auto-configuration assumes that user is developing a web application and sets up Spring accordingly.
2. **@SpringBootApplication –** It is combination of three annotations **@EnableAutoConfiguration, @ComponentScan,** and **@Configuration**.
3. **@RequestMapping – maps** the web requests to specific handler. It has many optional elements like **consumes, header, method, name, params, path, produces**, and **value**. It can be used as class or method level.

contd..

4. **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**
5. **@PostMapping:** It maps the **HTTP POST** requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
6. **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**

contd..

7. **@DeleteMapping –** Maps the **HTTP DELETE** request on the specific handler method. Creates a web service that endpoint **deletes** a resource. Used instead of **@RequestMapping(method = RequestMethod.DELETE)**
8. **@PatchMapping –** Maps the **HTTP PATCH** requests on the specific handler method. Used instead of **@RequestMapping(method = RequestMethod.PATCH)**
9. **@RequestBody –** Binds HTTP request with an object in a method parameter. Internally, uses **HTTP message–converters** to convert the body of the request. When a method is annotated with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.

contd..

10. **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.
11. **@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.
12. **@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a **query parameter**. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.

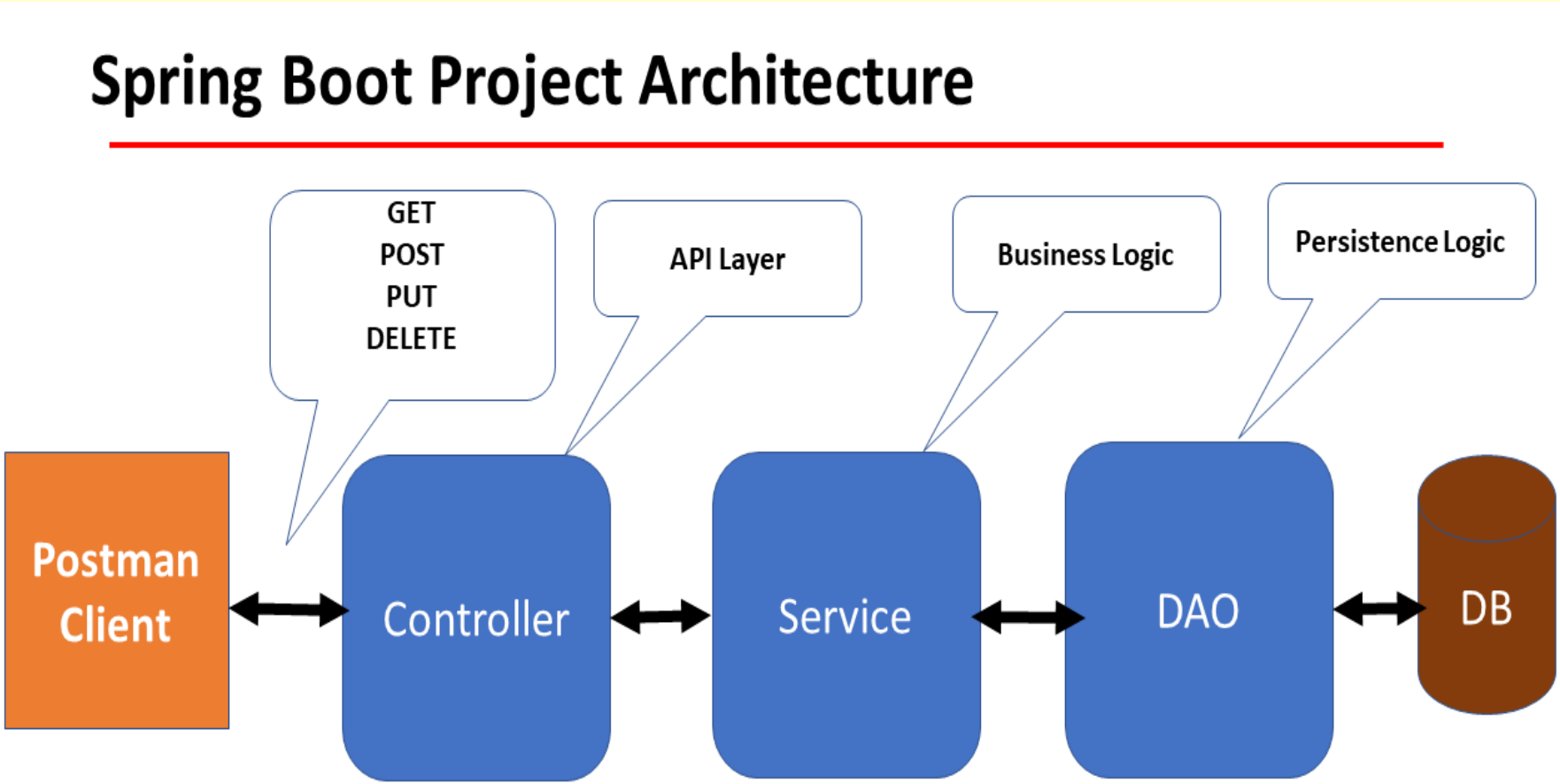
contd..

13. **@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.** For each detail in the header, we should specify separate annotations. We can use it multiple time in a method
14. **@RestController:** It can be considered as a combination of **@Controller** and **@ResponseBody** annotations. The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.
15. **@RequestAttribute:** It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

Dependency Management

- Every release of Spring Boot provides a **curated dependencies** list that it supports.
- The **curated list** contains all the spring modules that can be used with Spring Boot as well as refined list of third party libraries.
- The list is available as a standard Bill of Materials (spring–boot dependencies) that can be used with both Maven and Gradle.
- Developer **dont need to provide version** for any of these dependencies in the build configuration, as Spring Boot manages them. These are upgraded as well in a consistent way.
- The **version** can be specified and override the Spring Boot's recommendations if need be.

Architecture of SB project



Spring Initializr

- **Spring Initializr** is a **web–based tool** provided by the **Pivotal Web Service**.
- **Spring Initializr**, easily generate the structure of the **Spring Boot Project**. It offers extensible API for creating JVM-based projects.
- It also provides various options for the project that are expressed in a metadata model.
- The metadata model allows us to configure the list of dependencies supported by JVM and platform versions, etc.
- It serves its metadata in a well-known that provides necessary assistance to third-party clients.

Spring Initializr Modules

- Spring Initializr has the following module :

- ✓ **initializr–actuator:** It provides additional information and statistics on project generation. It is an optional module.

- ✓ **initializr–bom:** In this module, **BOM** stands for **Bill Of Materials**. In Spring Boot, BOM is a special kind of **POM** that is used to control the **versions** of a project's **dependencies**.

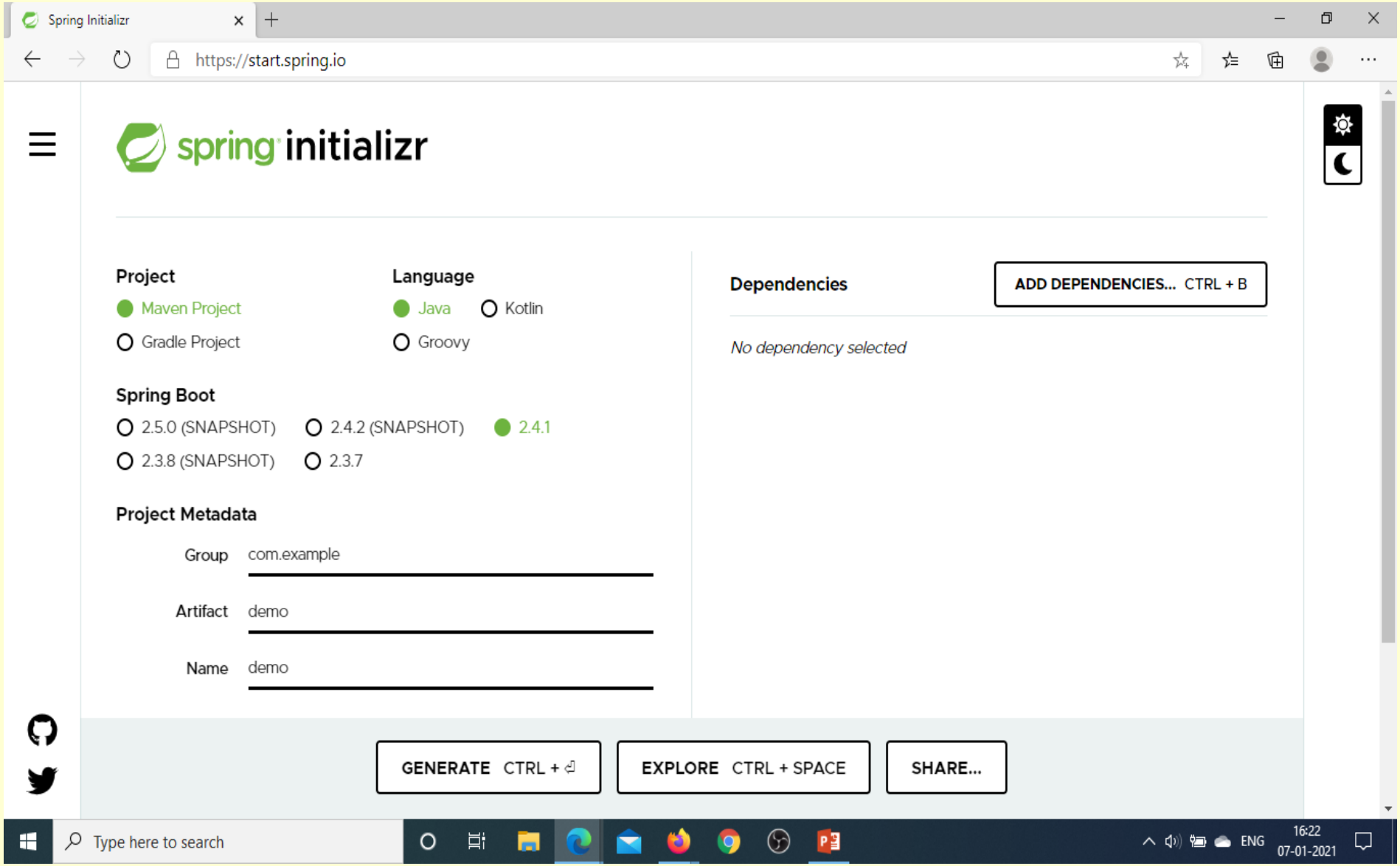
- ✓ It provides a central place to define and update those versions. It provides flexibility to add a dependency in our module without worrying about the versions.

- ✓ Outside the software world, the **BOM** is a list of parts, items, assemblies, and other materials required to create products. It explains **what, how,** and **where** to collect required materials.

contd..

- ✓ **initializr-docs:** It provides documentation.
- ✓ **initializr-generator:** It is a core project generation library.
- ✓ **initializr-generator-spring:**
- ✓ **initializr-generator-test:** It provides a test infrastructure for project generation.
- ✓ **initializr-metadata:** It provides metadata infrastructure for various aspects of the projects.
- ✓ **initializr-service-example:** It provides custom instances.
- ✓ **initializr-version-resolver:** It is an optional module to extract version numbers from an arbitrary POM.
- ✓ **initializr-web:** It provides web endpoints for third party clients.

contd..


Spring Initializr
https://start.spring.io
spring initializr
Project
Maven Project
Gradle Project
Language
Java
Kotlin
Groovy
Spring Boot
2.5.0 (SNAPSHOT)
2.4.2 (SNAPSHOT)
2.4.1
2.3.8 (SNAPSHOT)
2.3.7
Project Metadata
Group
com.example
Artifact
demo
Name
demo
Dependencies
ADD DEPENDENCIES... CTRL + B
No dependency selected
GENERATE CTRL + ⏎
EXPLORE CTRL + SPACE
SHARE...
Type here to search
ENG
16:22
07-01-2021

Getting Started

- **SpringApplication** SpringApplication class provides a convenient way to bootstrap a Spring application that is started from a main() method. It can also be delegated to the static SpringApplication.run method –

```
public static void main(String[] args)
{ SpringApplication.run(MySpringConfiguration.class, args);
 }
```

- **Startup Failure** if the application fails to start, registered FailureAnalyzers provide a dedicated error message and a concrete action to fix the problem. E.g. if you start a web application on port 8080 and that port is already in use, should see something similar to the following message:
- Spring Boot provides numerous FailureAnalyzer implementations, and can be added upon.

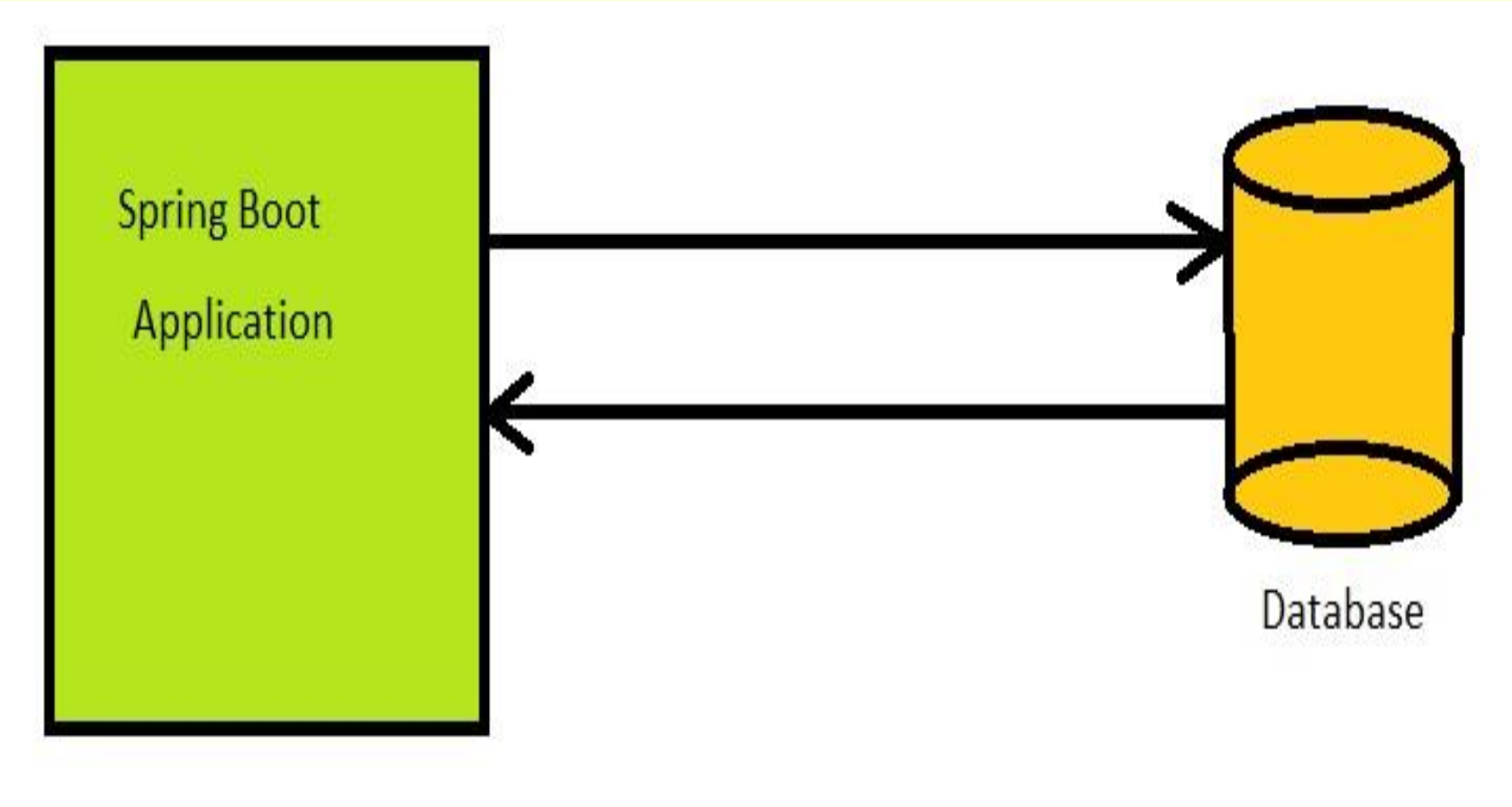
contd..

- **Web Environment** SpringApplication attempts to create the right type of ApplicationContext for user. The algorithm used to determine a WebApplicationType –

1. If Spring MVC is present, an AnnotationConfigServletWebServerApplicationContext is used
2. If Spring MVC is not present and Spring WebFlux is present, an AnnotationConfigReactiveWebServerApplicationContext is used
3. Otherwise, AnnotationConfigApplicationContext is used

Note – This means that if you are using Spring MVC and the new WebClient from Spring WebFlux in the same application, Spring MVC will be used by default. Can override that easily by calling setWebApplicationType(WebApplication Type).

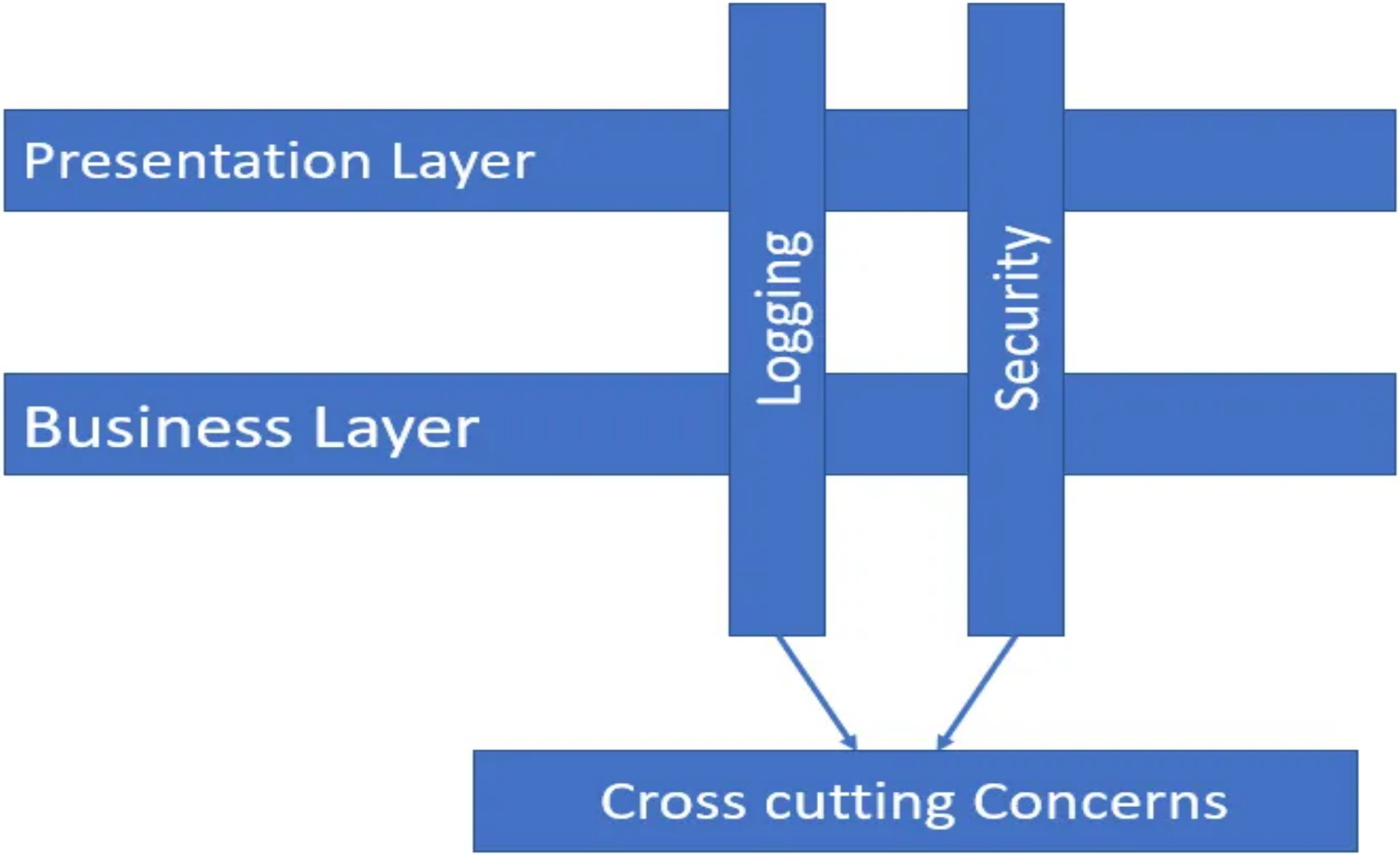
Spring Boot with JDBC



Spring Boot with AOP

- **Aspect Oriented Programming** (AOP) provides modularity; AOP breaks the program logic into distinct parts called **concerns**.
- **Aspect** is a modularization of **concern** that cuts across multiple classes. E.g. unified logging. The unit of modularity in AOP is **Aspect.**
- Most applications have concerns that **'cut'** across different abstraction layers. E.g. logging. A **cross-cutting concern** is a concern that can affect the whole application and should be centralized in one location in code e.g. transaction management, authentication, logging, security etc.
- Spring boot application mainly divided in to three layers:
  - ✓ Web Layer for exposing the services using RESTFul web services.
  - ✓ Business layer to handle business logic.
  - ✓ Data Layer for data persistence logic.
- Each layer is having different responsibility and there are some common **aspects** which gets apply to all layers

Diagrammatic Representation

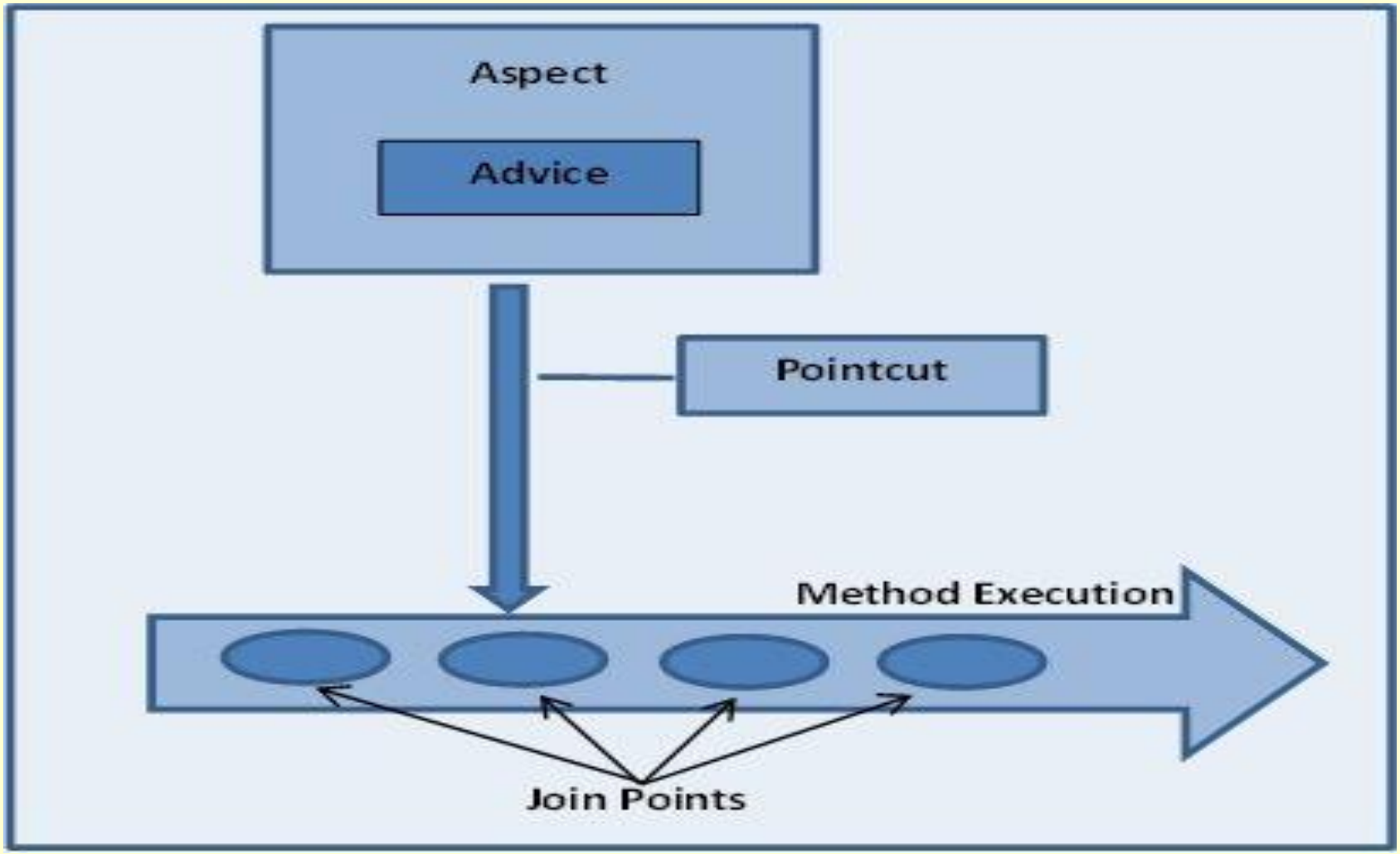


contd..

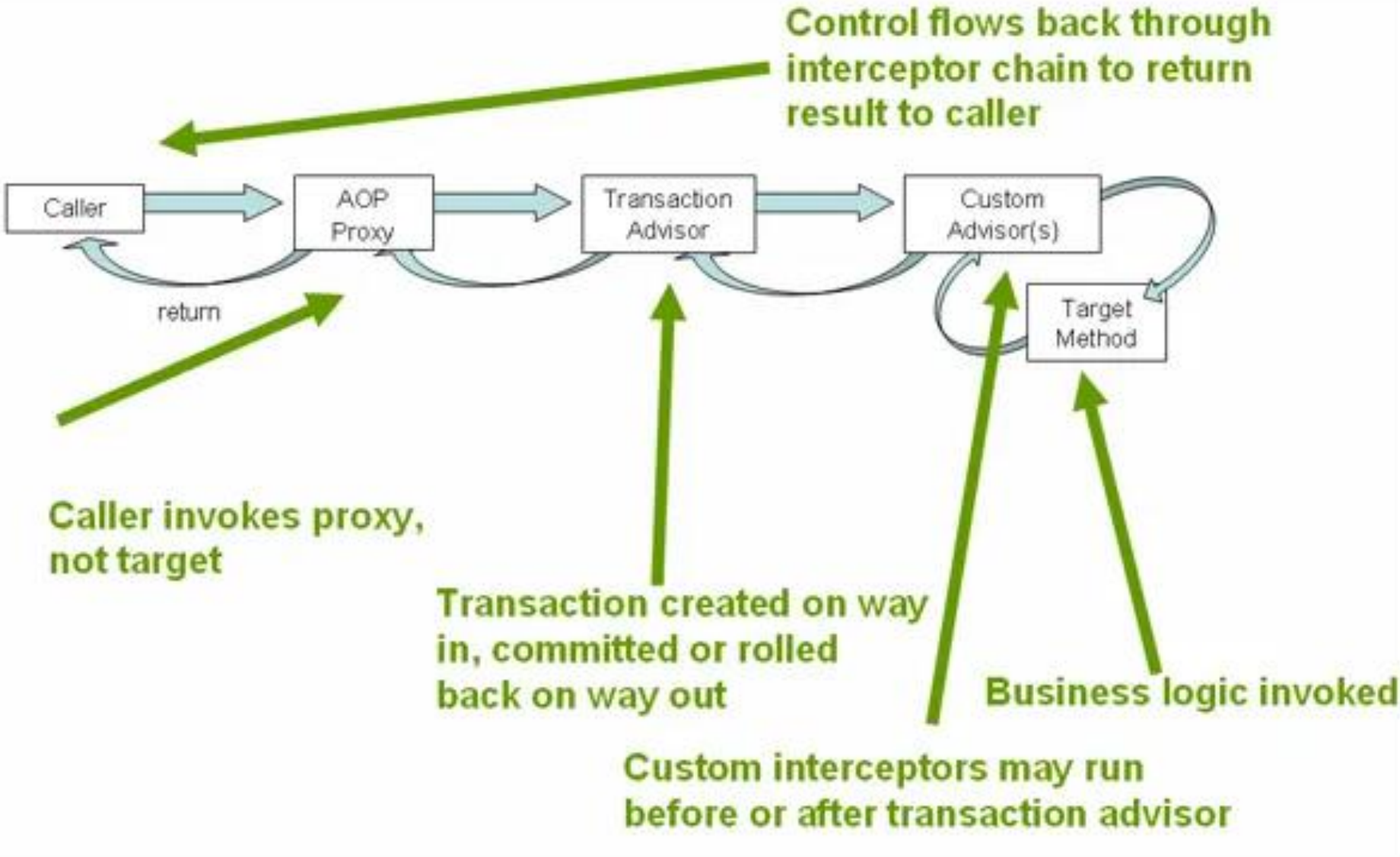
- **Scenario –** in usual way of application development, log statements all over the service layer, but logging is actually one concern and as such should be separated from the **business logic** into a **different entity**.

1. Aspect–Oriented Programming framework provides – An **aspect** i.e. modularization of **concern** that cuts across multiple classes. **:** *A package of functionality providing the cross-cutting requirements.*
2. A **Joinpoint** is a point during the execution of a program, such as execution of a method or the handling of an exception, exception handling, field access etc. **JoinPoint** represents a method execution.
3. An **advice** is an action taken by an aspect at a particular Joinpoint. Different types of advice include **around, before** and **after advice.**

contd..



contd..


Control flows back through interceptor chain to return result to caller
Caller
AOP Proxy
Transaction Advisor
Custom Advisor(s)
Target Method
return
Caller invokes proxy, not target
Transaction created on way in, committed or rolled back on way out
Business logic invoked
Custom interceptors may run before or after transaction advisor

contd..

- **Advice** – Advice represents an action taken by an aspect at a particular join point. **Advice** is *implementation of functionality that will be applied.*
- There are different types of advices:
  - ✓ **Before Advice**: it executes before a join point.
  - ✓ **After Returning Advice**: it executes after a joint point completes normally.
  - ✓ **After Throwing Advice**: it executes if method exits by throwing an exception.
  - ✓ **After (finally) Advice**: it executes after a join point regardless of join point exit whether normally or exceptional return.
  - ✓ **Around Advice**: It executes before and after a join point.

contd..

4. AOP provides the pluggable way to dynamically add the additional concern **before, after or around** the actual logic. The *Advice* is **modeled** as an interceptor, maintaining a chain of interceptors around the *Joinpoint*.
5. A **pointcut** is a place or several places in the code where the crosscutting concern, the advice, is to be applied. The advice is applied to certain **pointcuts** in the code. *A rule for matching the parts of the object model that the functionality will* be applied to. Helps match an *Advice* to be applied by an *Aspect* at a particular *JoinPoint*.
6. The **advice** together with a **pointcut** is called an **aspect**, hence the name Aspect-Oriented Programming. Spring uses its own frame work based upon dynamic proxies and/or CGLIB byte code generation, but can integrate with others like AspectJ.

contd..

6. **AOP Proxy** – It is used to implement aspect contracts, created by AOP framework. It will be a JDK dynamic proxy or CGLIB proxy in spring framework.
7. **Introduction** – It means introduction of additional method and fields for a type. It allows you to introduce new interface to any advised object.
8. **Target Object** – It is the object i.e. being advised by one or more aspects. It is also known as proxied object in spring because Spring AOP is implemented using runtime proxies.
9. **Weaving** – It is the process of linking aspect with other application types or objects to create an advised object. Weaving can be done at compile time, load time or runtime. Spring AOP performs weaving at runtime.

contd..

- **Joinpoint:** A joinpoint is a *candidate* point in the **Program Execution** of the application where an aspect can be plugged in. e.g. a method being called, an exception being thrown, or even a field being modified. These are the points where your aspect's code can be inserted into the normal flow of the application to add new behavior.
- **Advice:** This is an object which includes API invocations to the system wide concerns representing the action to perform at a joinpoint specified by a point.
- **Pointcut:** A pointcut defines at what joinpoints, the associated Advice should be applied. Advice can be applied at any joinpoint supported by the AOP framework. Theres no need to apply all of your aspects at all of the possible joinpoints.

contd..

- Analogy

- ✓ *Joinpoints are options on the menu and Pointcuts are items user select.*
- ✓ A Joinpoint is an opportunity within code for user to apply an aspect...just an opportunity. Once user take that opportunity and select one or more Joinpoints and apply an aspect to them, user has got a Pointcut.

contd..

- Annotations and wild card expressions –

1. To access method to only specific class, give package name
   @Before("execution(public String Circle.getName())") – this aspect is run only for Circle class.
2. To apply aspect for all the methods which starts with get and irrespective of return type, use wildcard.
   @Before("execution(public * get*())")
   @Before("execution(* get*())")

contd..

3. To pass argument
   a) @Before("execution(* get*())") –> Zero arguments
   b) @Before("execution(* get*(**))") –> only one argument
   c) @Before("execution(* get*(..))") –> two dots means, can have zero or more parameter.
   d) @Before("execution(* org.yash.watertechsol.model.*.get*())") –> it applies to all classes available in Model

- @After annotated methods run exactly after the all methods matching with pointcut expression.

1. @EnableAspectJAutoProxy(proxyTargetClass=true)

- It enables support for handling components marked with AspectJ's @Aspect annotation. It is used with @Configuration annotation. We can control the type of proxy by using the **proxyTargetClass** attribute. Its default value is **false**.

interface JointPoint

- This interface represents a generic runtime joinpoint (in the AOP terminology).
- A runtime joinpoint is an *event* that occurs on a static joinpoint (i.e. a location in a the program).
- For instance, an invocation is the runtime joinpoint on a method (static joinpoint). The static part of a given joinpoint can be generically retrieved using the getStaticPart() method.
- In the context of an interception framework, a runtime joinpoint is then the reification of an access to an accessible object (a method, a constructor, a field), i.e. the static part of the joinpoint. It is passed to the interceptors that are installed on the static joinpoint.

contd..

- AOP implementations are provided by:
  - ✓ AspectJ
  - ✓ Spring AOP
  - ✓ JBoss AOP
- Spring AOP – Spring AOP can be used by 3 ways given below. But the widely used approach is Spring AspectJ Annotation Style. The 3 ways to use spring AOP are given below:
  - ✓ By Spring1.2 Old style (dtd based) (also supported in Spring3)
  - ✓ By AspectJ annotation-style
  - ✓ By Spring XML configuration-style(schema based)

Spring Boot CLI

- Spring Boot CLI(Command Line Interface) is a Spring Boot software to run and test Spring Boot applications from command prompt.
- On executing Spring Boot applications using CLI, it internally uses Spring Boot Starter and Spring Boot AutoConfigurate components to resolve all dependencies and execute the application.
- Simple Spring Boot CLI Commands can be used to execute Spring Web Applications.
- Spring Boot CLI has introduced a new "spring" command to execute Groovy Scripts from command line.

Spring Boot Actuator

- Spring Boot Actuator components gives many features, but two major features are
  - ✓ Providing management **EndPoints** to Spring Boot applications.
  - ✓ Spring Boot applications metrics.
- When Spring Boot web application is run using CLI, Spring Boot Actuator automatically provides **hostname** as **localhost** and **default port number** as **8080**.
- This application can be accessed using "http://localhost:8080/" end point.
- HTTP Request methods like GET and POST are used to represent Management EndPoints using Spring Boot actuator.

Groovy

- Groovy is an object oriented language which is based on Java platform. Groovy 1.0 was released in January 2, 2007 with Groovy 2.4 as the current major release.
- Features of Groovy

1. Support for both static and dynamic typing.
2. Support for operator overloading.
3. Native syntax for lists and associative arrays and native support for regular expressions.
4. Native support for various markup languages such as XML and HTML.
5. Groovy is simple for Java developers since the syntax for Java and Groovy are very similar.
6. Can be used with existing Java libraries. Groovy extends the java.lang.Object.

contd..

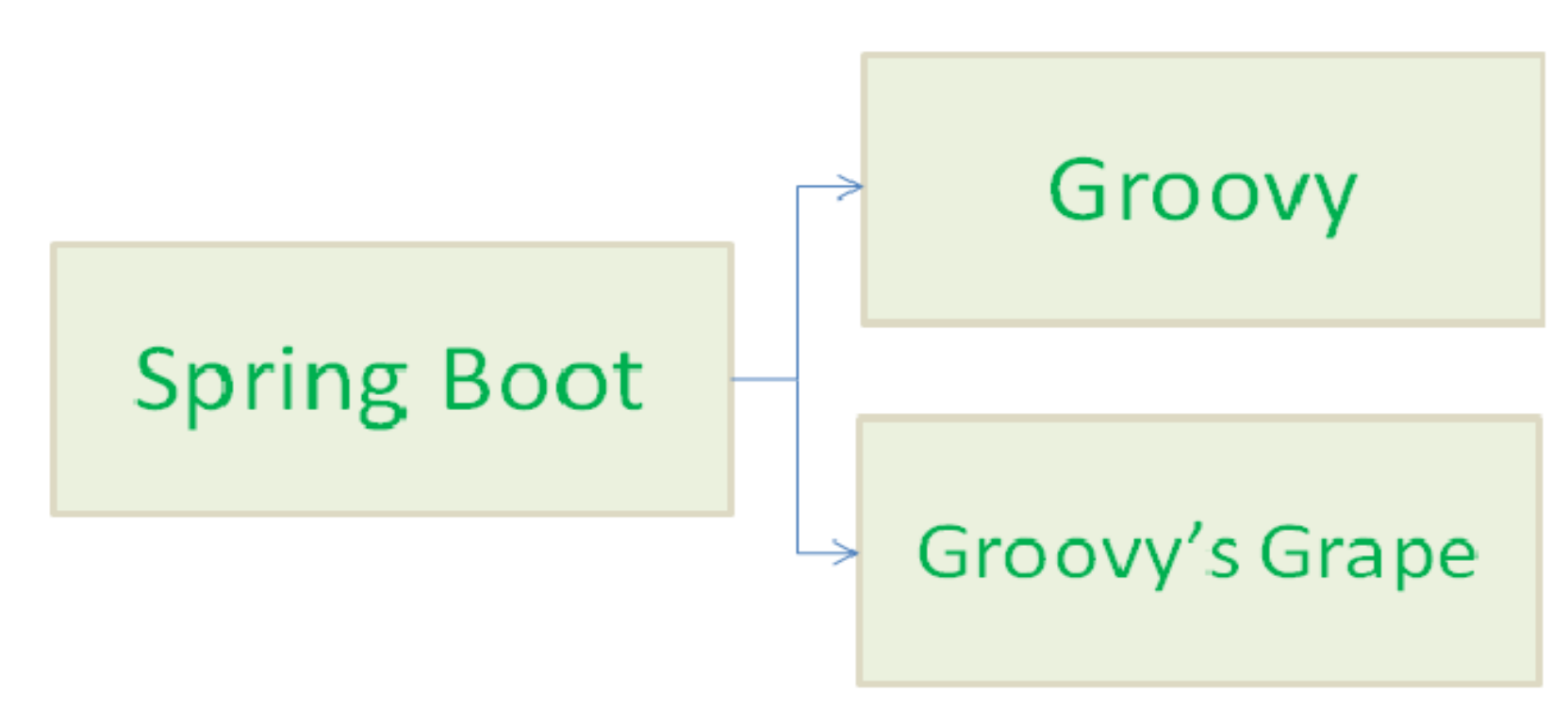
- In Groovy Programming language, there is no need to add some imports and no need to add some dependencies to Groovy project.
- When compiled Groovy scripts using Groovy Compiler(groovyc), it automatically adds all default import statements then compile it.
- In the same way, Groovy Programming language contains a JAR Dependency Resolver to resolve and add all required jar files to Groovy Project classpath.
- Spring Boot Framework internally uses Groovy to add some defaults like Default import statements, Application main() method etc. To run Groovy Scripts from CLI Command prompt, it uses this main() method to run the Spring Boot Application.

contd..

- **Grape**

- ✓ Grape is an Embedded Dependency Resolution engine. Its a JAR Dependency Manager embedded into Groovy.
- ✓ Grape lets user quickly add maven repository dependencies to the project classpath to reduce build file definitions.
- ✓ Spring Boot Framework internally depends on these two major components: Groovy and Grape.

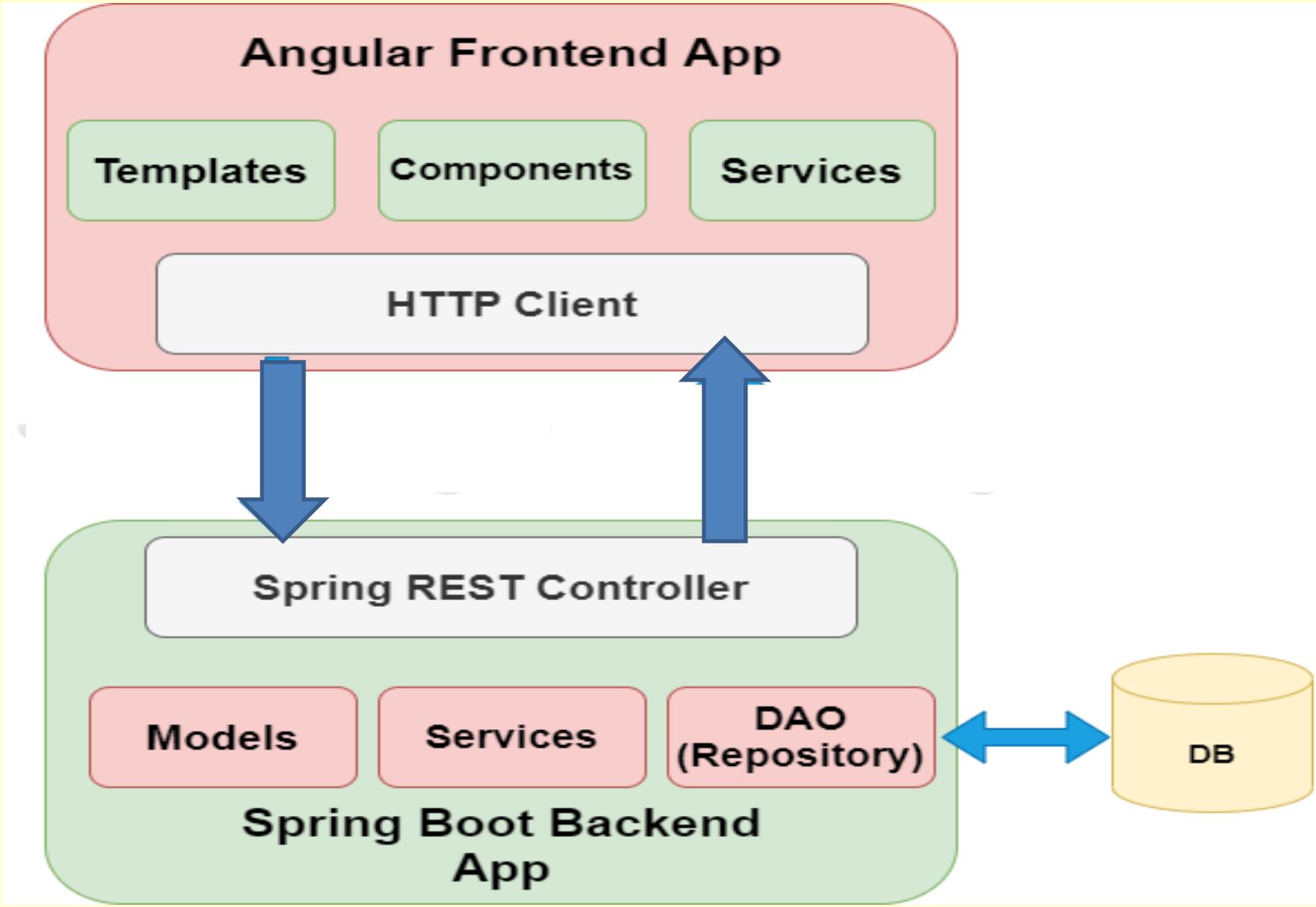
contd..



CrossOrigin

- Cross-Origin Resource Sharing (CORS) is **an HTTP-header based mechanism.**
- **It provides for a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading of resources**. However for security reasons, browsers restrict cross-origin HTTP requests initiated from scripts.
- Cross-origin resource sharing (CORS) is **a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served**.
- A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos.
- In **most cases work by default in browsers**.

contd..


Angular Frontend App
Templates
Components
Services
HTTP Client
Spring REST Controller
Models
Services
DAO
(Repository)
DB
Spring Boot Backend
App

contd..

- Create two projects:

1. **Springboot2-jpa-crud** : This project is used to develop CRUD RESTFul APIs. E.g. **Employee Management System** using Spring Boot 2, JPA and MySQL as a database.
2. **Angular-Springboot-client**: This project is used to develop single page application using Angular 8 as front-end technology. This Angular 8 application consumes CRUD Restful APIs developed and exposed by a **springboot2-jpa-crud-example** project.

Thank You