WEB SERVER

SPRING BOOT

WEB SERVER

- The term **web server** can refer to hardware or software, or both of them working together.
- On the **hardware side**
  - A web server is a computer that stores web server software and a website's component files (for example, HTML documents, images, CSS stylesheets, and JavaScript files).
  - A web server connects to the Internet and supports physical data interchange with other devices connected to the web.
- On the **software side**
  - a web server includes several parts that control how web users access hosted files and resources
  - A HTTP server is software that understands URLs (web addresses) and HTTP (the protocol your browser uses to view webpages).
  - A HTTP server can be accessed through the domain names of the websites it stores, and it delivers the content of these hosted websites to the end user's device.
- A static web server, or stack, consists of a computer (hardware) with an HTTP server (software). We call it "static" because the server sends its hosted files as-is to your browser.
- A dynamic web server consists of a static web server plus extra software, most commonly an application server and a database.

WHAT IS SPRING BOOT

- Spring Boot is an open-source framework for building Java-based, production-grade, stand-alone, and microservices-based web applications
- It is part of the larger Spring Framework, which is a comprehensive framework for enterprise Java development
- "Convention over Configuration" (CoC)
  - is a software design paradigm that suggests that developers should follow a set of conventions and defaults in their projects
  - Minimises the need for explicit configuration.
  - The idea is to reduce the amount of boilerplate code and configuration that developers have to write by establishing sensible defaults based on common practices.
  - Interestingly, the Spring framework leans more towards “Configuration over convention”
- Embedded Web Server Support
  - Can run the Java application without the need for an external web server
- Auto-Configuration
  - Spring Boot attempts to automatically configure your application based on the dependencies you have added to your project
- Spring Boot Starters
  - These are pre-built templates that contain a set of dependencies for common tasks, such as data access, and messaging

DEPENDENCY

```java
class PaymentProcessor {
  public void process(double amount) {
      System.out.println("Processing Rs " + amount);
  }
}

class OrderService {
  private PaymentProcessor paymentProcessor;

  public OrderService() {
    // Direct dependency creation
      this.paymentProcessor = new PaymentProcessor();
  }

  public void placeOrder(double amount) {
      paymentProcessor.process(amount);
  }
}

public class Main {
  public static void main(String[] args) {
      OrderService orderService = new OrderService();
      orderService.placeOrder(100.0);
  }
}
```

```java
class PaymentProcessor {
  public void process(double amount) {
      System.out.println("Processing  Rs " + amount);
  }
}

class OrderService {
  private PaymentProcessor paymentProcessor;

  // Dependency Injection via constructor
  public OrderService(PaymentProcessor paymentProcessor) {
      this.paymentProcessor = paymentProcessor;
  }

  public void placeOrder(double amount) {
      paymentProcessor.process(amount);
  }
}

public class Main {
  public static void main(String[] args) {
      // Manually inject dependency
      PaymentProcessor paymentProcessor = new PaymentProcessor();
      OrderService orderService = new OrderService(paymentProcessor);

      orderService.placeOrder(100.0);
  }
}
```

- How an object gets another object/service on which it depends?
- Application wiring

APPLICATION WIRING

- Process of connecting different components of an application together to form a cohesive and functional system
- Many approaches
  - Manual wiring
  - Dependency Injection frameworks
  - Config files

DEPENDENCY INJECTION

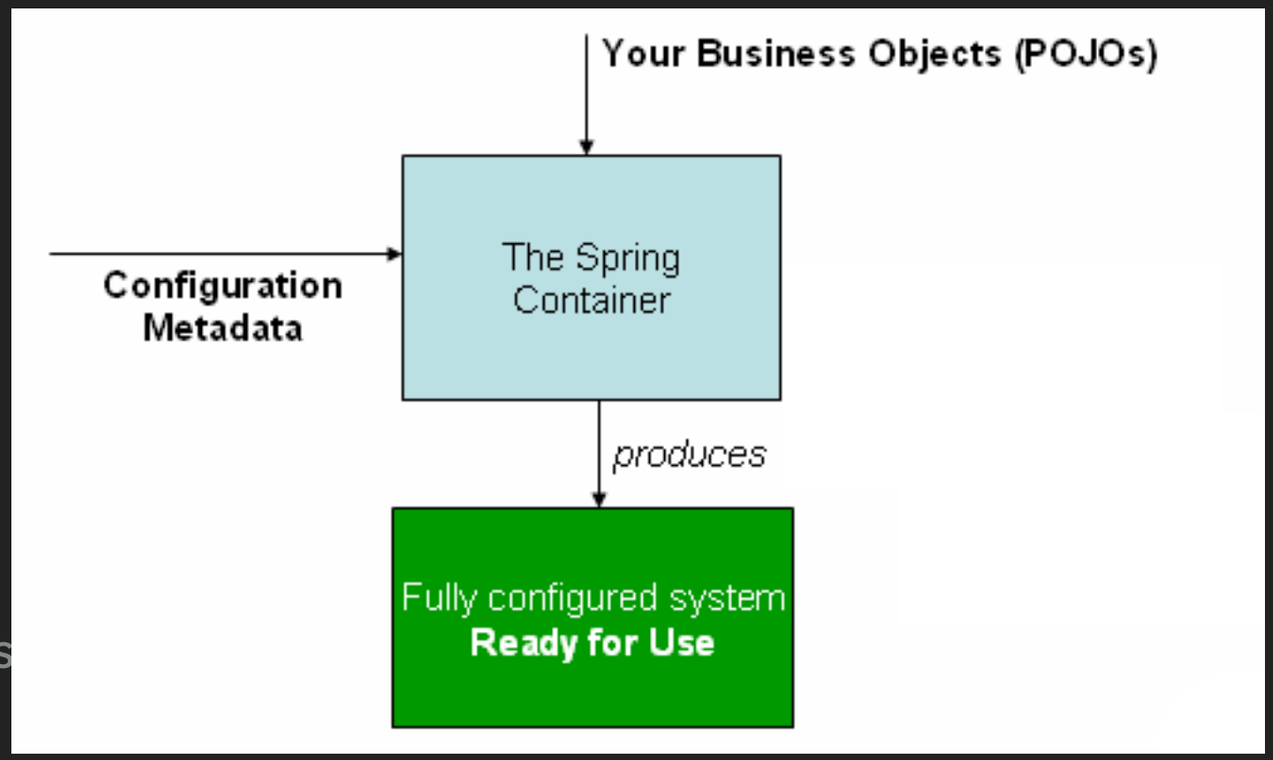
- A ***dependency*** is any object or service that a class or function requires to perform its job
- Instead of an object creating its dependencies directly (using *new* or instantiating them), dependencies are "***injected***" by an external component.
  - Constructor injection
  - Setter injection
- A container (an external entity) controls how dependencies are created and assigned
- Control of object creation and binding is inverted (IoC)

DEPENDENCY INJECTION

- Advantages
  - Code is cleaner
  - Object decoupling is more effective
  - Testing becomes easy
  - Mocking becomes easy
- Ref: https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html

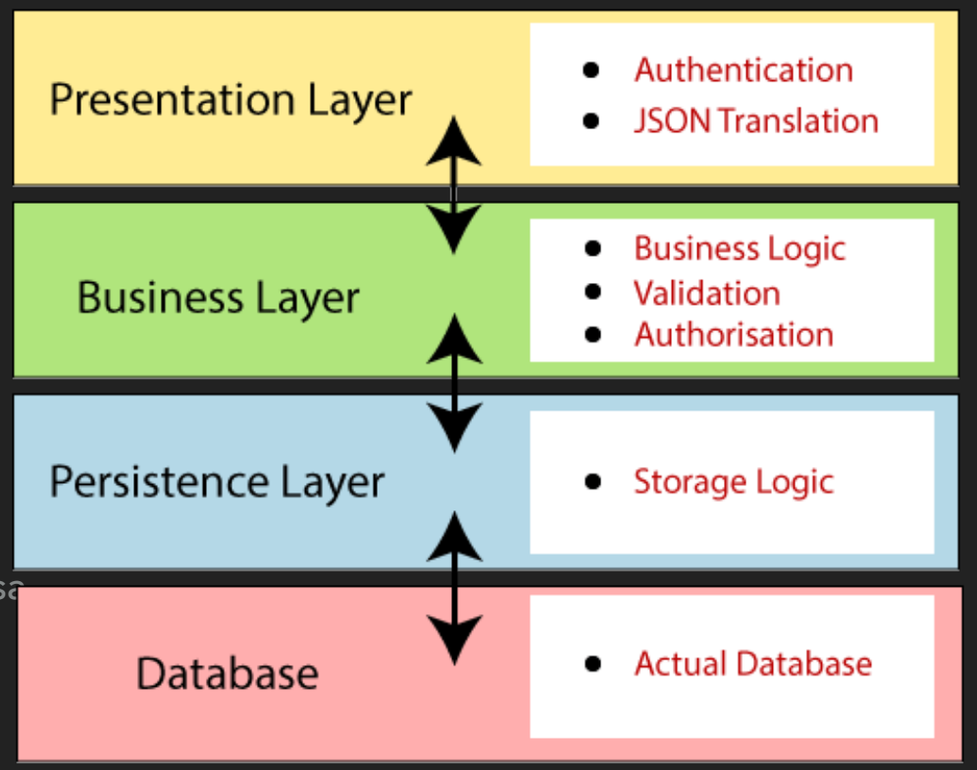
SPRING CONTAINER

- is responsible for instantiating, configuring, and assembling the beans
- Also know as the Spring IoC container
- The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata
- The configuration metadata is represented in XML, Java annotations, or Java code.
- It lets you express the objects that compose your application and the rich interdependencies between those objects.
- Ref: https://docs.spring.io/spring-framework/reference/core/beans/basics.html



SPRING BOOT ARCHITECTURE

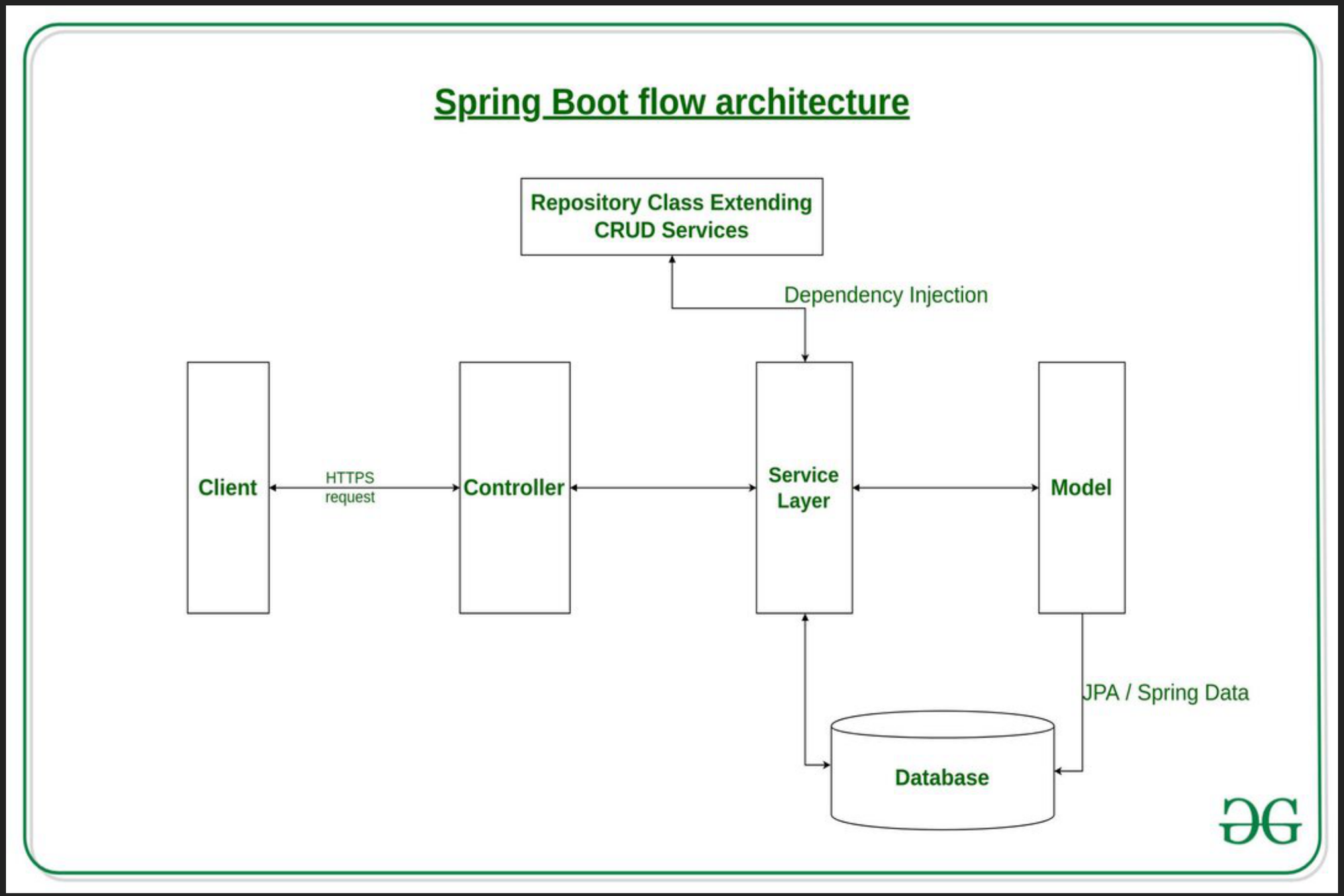
- Presentation Layer
  - Handles the HTTP requests, translates the JSON parameter to Java object and vice versa, authenticates the request and transfer it to the business layer.
  - Consists of Views (or the front end) of the application
- Business layer
  - The business layer contains all the business logic.
  - It consists of services classes.
  - It is responsible for validation and authorization.
- The persistence layer
  - contains the database storage logic.
  - It is responsible for converting business objects to the database rows and vice-versa
- The database layer
  - Layer where the data is actually stored
  - contains the databases such as MySql, MongoDB, etc.
  - This layer can contain multiple databases. It is responsible for performing the CRUD operations.



FLOW

- The Client makes an HTTP request(GET, PUT, POST, etc.)
- The HTTP request is forwarded to the Controller.
  - The controller maps that request and handles it
    - After that, it calls the service logic if required
- The business logic resides in the Service layer.
  - The request is processed using the data in the DB either directly or by using the JPA model classes in the Model layer.
- A view is returned as Response from the controller.
  - The View can be a JSON response or a JSP page (HTML, CSS, JS)

FLOW



src: https://www.geeksforgeeks.org/spring-boot-architecture/

HTTP & REST

WHAT IS REST?

- Representational State Transfer (REST) is an architectural style that defines a set of constraints to be used for creating web services.
- REST determines how the interaction between client and server is designed
- RESTful system
  - Server has resources
  - Clients request the resource(s)
  - A request is sent (from client to server) as a HTTP GET or POST or PUT or DELETE.
  - A response can be HTML, JSON, plain text, image, etc. Most popular format is JSON

ARCHITECTURAL CONSTRAINTS

- Client-Server
  - REST application should have a client-server architecture.
  - A Client is someone who is requesting resources and are not concerned with data storage, which remains internal to each server
  - A Server is someone who holds the resources and are not concerned with the user interface or user state.
- Uniform Interface
  - It suggests that there should be an uniform way of interacting with a given server irrespective of device or type of application (website, mobile app).
- Self-descriptive Messages (or Stateless): Each message includes enough information to describe how to process the message so that server can easily analyses the request.
- Resource-Based: Individual resources are identified in requests. For example: API/users.
  - Manipulation of Resources Through Representations: Client has representation of resource and it contains enough information to modify or delete the resource on the server, provided it has permission to do so.
  - Example
    - Client requests for User info (identified by a user ID) from the server
    - Server returns representation of the user (usually JSON object)
    - Client modifies the representation (JSON object) and sends it to server
    - Server updates the User record in the DB with the modifications from client

ARCHITECTURAL CONSTRAINTS

- Stateless
  - It means that the necessary state to handle the request is contained within the request itself and server would not store anything related to the session.
  - In REST, the client must include all information for the server to fulfill the request
    - Advantage - Servers can scale easily
    - Disadvantage - Client has to send more information over the network.
- Cacheable
  - Responses from the server can be explicitly marked as cacheable or non-cacheable.
  - This allows clients to cache responses, improving performance and reducing the load on the server.

HTTP & CRUD

- HTTP methods
  - GET: Read the representation of a resource
  - POST: Create a resource.
  - PUT: Update a resource
  - DELETE: Delete a resource
- CRUD operations
  - C (create) - POST
  - R (read) - GET
  - U (update) - PUT
  - D (delete) - DELETE

URI	HTTP Verb	Description
api/users	GET	Get all users
api/users	POST	Add a user
api/users/1	PUT	Update a user with id = 1
api/users/1	DELETE	Delete a user with id = 1
api/users/1	GET	Get a user with id = 1

Source: https://www.geeksforgeeks.org/rest-api-architectural-constraints/

HIBERNATE

WHAT IS HIBERNATE?

- Hibernate is a powerful and widely used open-source Object-Relational Mapping (ORM) framework for Java.
- Most application development is done in object oriented programming languages
- Data is stored in relational tables
- A translation layer is required between the objects and the tables, rows/columns
- ORMs form this translation layer
- Hibernate allows developers to map Java classes to database tables.
- Each Java class represents an entity, and the properties of the class map to columns in the corresponding database table.
- This mapping is typically done through the use of annotations or XML configuration files

FEATURES

- Hibernate Query Language (HQL)
  - Hibernate provides its own query language called HQL, which is a powerful and database-agnostic query language.
  - HQL queries are written in terms of Java objects and properties, rather than SQL tables and columns.
- Automatic Table Generation
  - Hibernate can automatically generate database tables based on the entity classes, eliminating the need for manual table creation.
  - It can also update the schema as the Java classes evolve.
- Caching
  - Hibernate includes caching mechanisms to improve performance. It supports both session based and global caching
- Transaction Management
  - Hibernate integrates with Java Transaction API (JTA) or Spring's transaction management
  - Ensures transactions are bound by the ACID properties
- Lazy Loading
  - Hibernate supports lazy loading, a technique where related objects are loaded from the database only when they are explicitly requested.
- Association Mapping
  - Hibernate supports various types of associations between entities, such as one-to-one, one-to-many, and many-to-many relationships