Spring with spring boot

Rajeev Gupta

rgupta.mtech@gmail.com
Java Trainer & consultant
@rajeev_gupta76

Rajeev Gupta

FreeLancer Corporate Java JEE/ Spring Trainer

New Delhi Area, India

freelance

Institution of Electronics and Telecommunication...

See contact info

See connections (500+)

Add profile section More...

1. Expert trainer for Java 8, GOF Design patterns, OOAD, JEE 7,Spring 5, Hibernate 5 ,Spring boot, microservice, netflix oss, Spring cloud, angularjs, Spring MVC, Spring Data, Spring Security, EJB 3, JPA 2, JMS 2, Struts 1/2, Web service

2. Helping technology organizations by training their fresh and senior engineers in key technologies and processes.

3. Taught graduate and post graduate academic courses to students of professional degrees.

I am open for advance java training /consultancy/ content development/ guest lectures/online training for corporate / institutions on freelance basis

Contact :

==========

raupta.mtech@gmail.com

Clients:

==========

Gemalto, Noida
Cyient Ltd, Noida
Fidelity Investment Ltd
Blackrock, Gurgaon
Mahindra comviva
Steria
Bank Of America
incedo gurgaon
MakeMyTrip
Capgemini India
HCL Technologies
CenturyLink
Deloitte consulting
Nucleus Software
Ericsson Gurgaon
Avaya gurgaon
Kronos Noida
NEC Technologies
A.T. Kearney
ust global
TCS
North Shore Technologies Noida
IBM
Sapient
Accenture
Incedo
Genpact
Indian Air force
Indian railways
Vidya Knowledge Park

Spring with Spring boot

- Spring framework, Configuration, 3 tier architecture
- What, Why spring boot? Intro to Microservice
- Spring boot CRUD application, with JPA
- Handling Exceptions
- Handling with status code: Using ResponseEntity & http status code
- JSR 303 validation
- hateoas

Spring boot step by step

- Swagger Documentation
- Implementation Filtering for Rest service response
- Versioning RESTful Services
- Enable cacheing
- Enable ScheduledProcess
- pagable rest response
- Spring mvc jsp
- Spring MVC with thymeleaf
- Case study: Blog and Comment : one to many Spring boot hibernate
- Bank application

- Spring framework, Configuration, 3 tier architecture

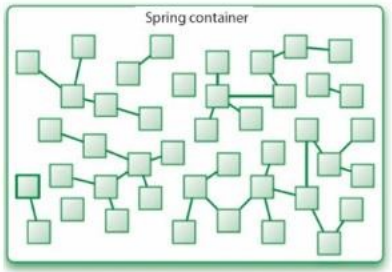
Spring framework

- Introduction Spring framework, where it fits, design patterns
- Dependency Injection, Configuration-XML, Java
- Aspect Oriented Programming, how it helps, configuration, examples

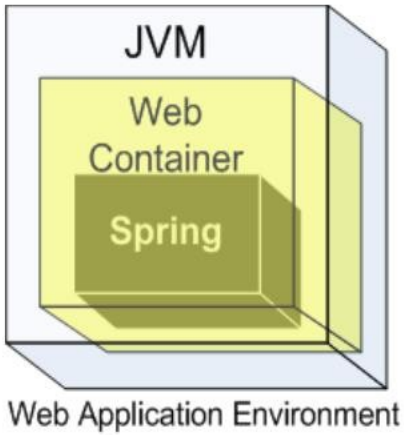
What is spring framework?

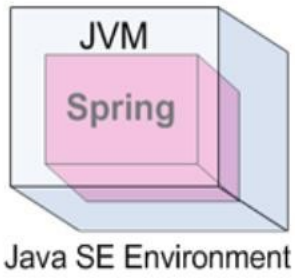
Spring framework is a contrainer that manage life cycle of bean.

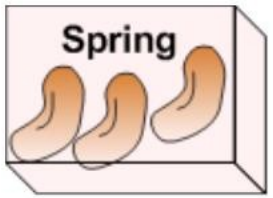
It Does 2 primary Jobs:
1. bean wiring
2. bean weaving

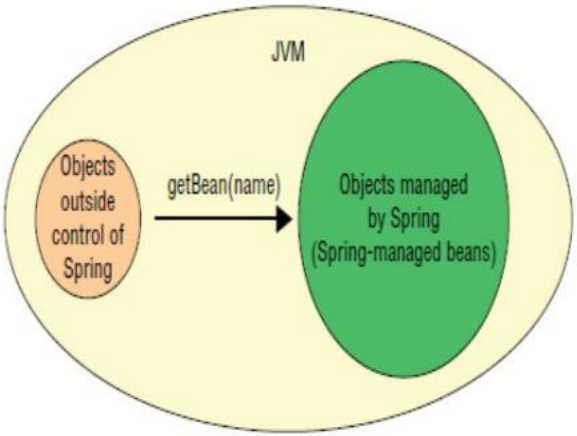


A bean is an object that is instantiated, assembled, and managed by a Spring IoC container.





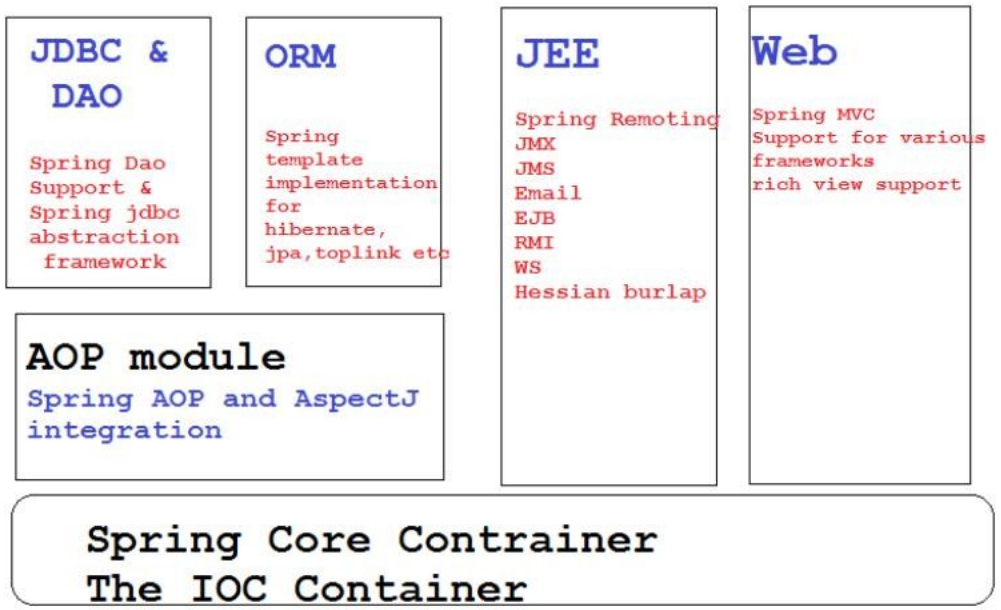




Why Spring framework? Where it fits?

Spring Framework is focused on simplifying enterprise Java development through

- dependency injection
- aspect-oriented programming
- boiler-plate code reduction using template design pattern



Spring framework design patterns

Design Patterns used in Spring Framework

Proxy Design Pattern

Singleton Design Pattern

Factory design pattern

Template Design Pattern

Model View Controller Pattern

Front Controller Pattern

View Helper Pattern

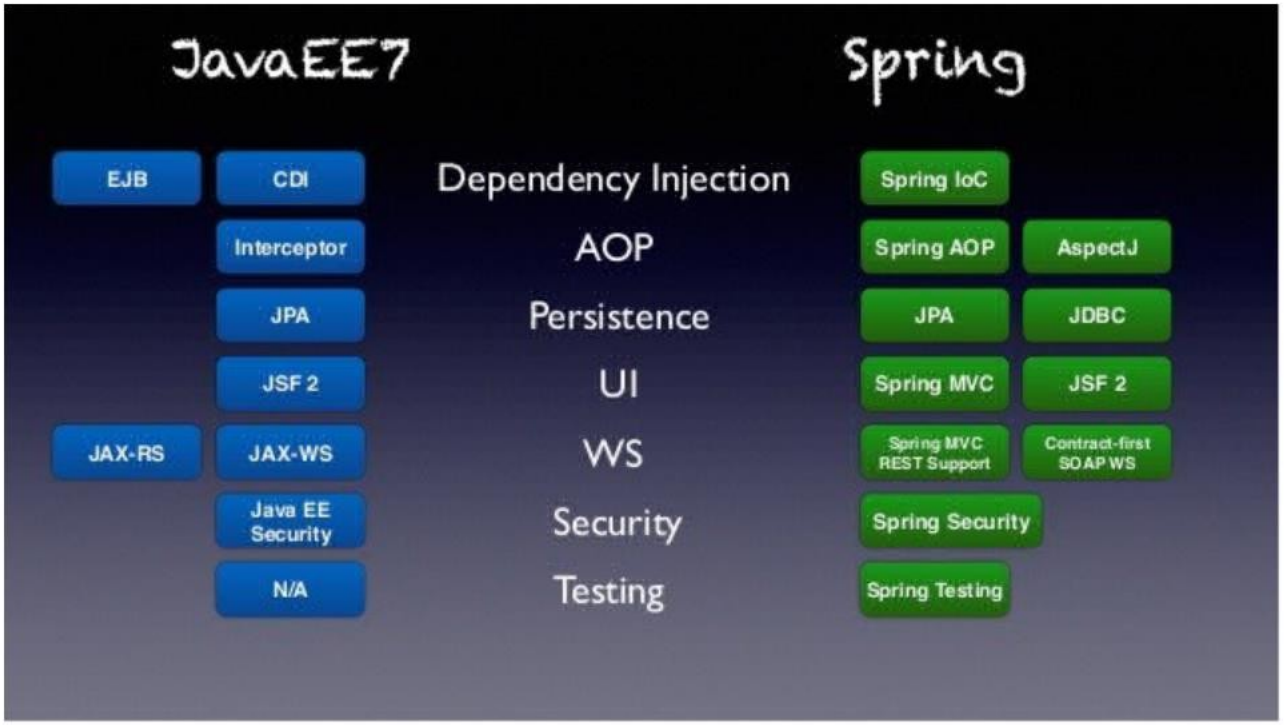
Dependency injection or inversion of control

Service Locator Pattern

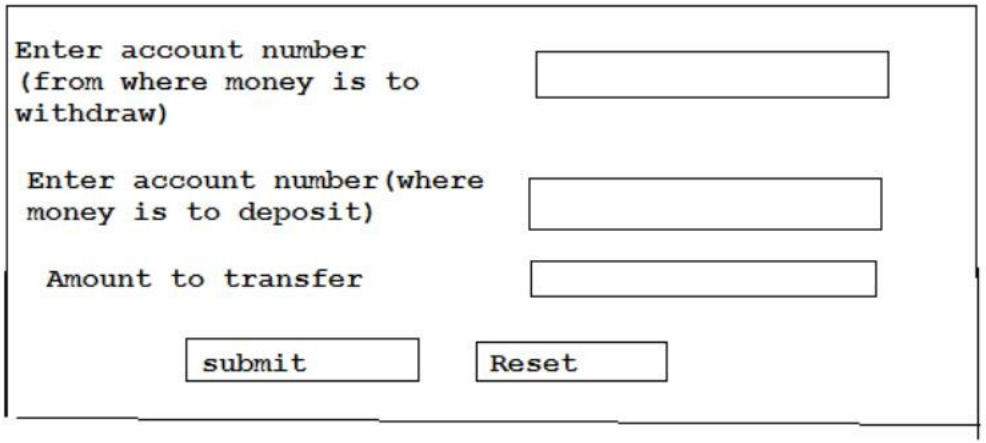
Observer-Observable

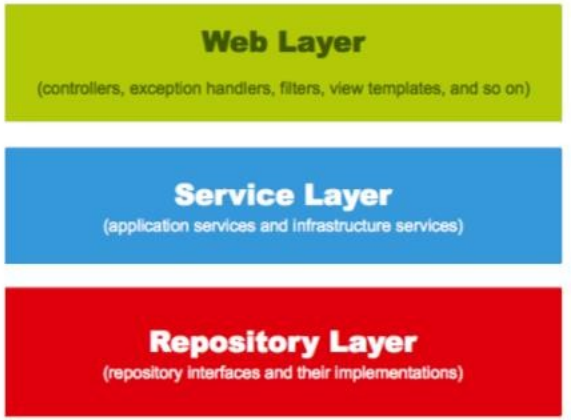
Context Object Pattern

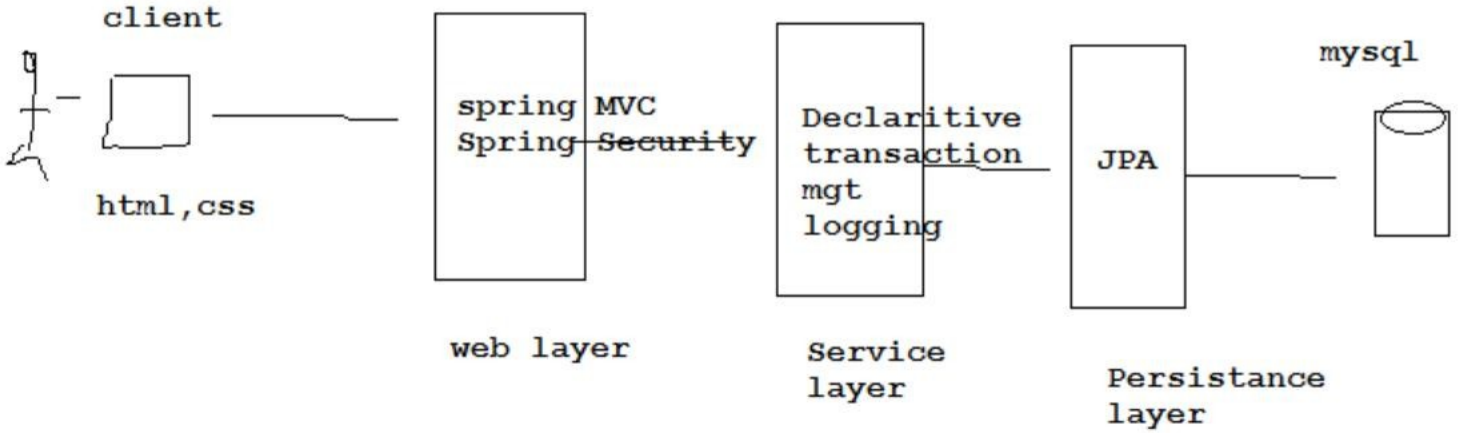
Java EE vs Spring framework



Spring, Hibernate 3 Tier architecture

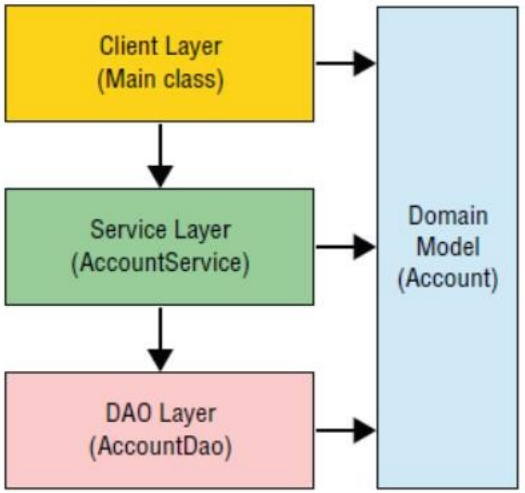






Application example: Fund transfer Bank Application

```
public class Account {
    private int id;
    private String name;
    private double balance;
```




```
public interface AccountDao {
    public void update(Account account);
    public Account find(int id);
}

public class AccountDaoImp implements AccountDao {

    private Map<Integer, Account> accouts = new HashMap<Integer, Account>();

    public void update(Account account) {..

public interface AccountService {
    public void transfer(int from, int to, int amout);
    public void deposit(int id, double amount);
    public Account getAccount(int id);
}
```

Spring framework

- Introduction Spring framework, where it fits, Spring boot design patterns
- Dependency Injection, Configuration-XML, Java
- Aspect Oriented Programming, how it helps, configuration, examples
- Spring Hibernate

Dependency Injection xml, java configuration

XML Configuration

```xml
<bean id="accountService" class="com.sample.bank.model.service.AccountServiceImp" autowire="byType"/>
<bean id="accountDao1" class="com.sample.bank.model.persistance.AccountDaoImp"/>
<bean id="accountDao2" class="com.sample.bank.model.persistance.AccountDaoImp" autowire-candidate="false"/>
```

Java Configuration

```java
@Configuration
@ComponentScan(basePackages={"com.sample.bank.*"})
@Scope(value="prototype")
public class AppConfig {

    @Bean(autowire=Autowire.BY_TYPE)
    @Scope(value="prototype")
    public AccountService accountService(){
        AccountServiceImp accountService=new AccountServiceImp();
        //accountService.setAccountDao(accountDao());
        return accountService;
    }


    @Bean
    public AccountDao accountDao(){
        AccountDao accountDao=new AccountDaoImp();
        return accountDao;
    }
}
```

```java
AnnotationConfigApplicationContext ctx=new AnnotationConfigApplicationContext(AppConfig.class);

AccountService s=ctx.getBean("accountService", AccountService.class);
s.transfer(1, 2, 100);
```

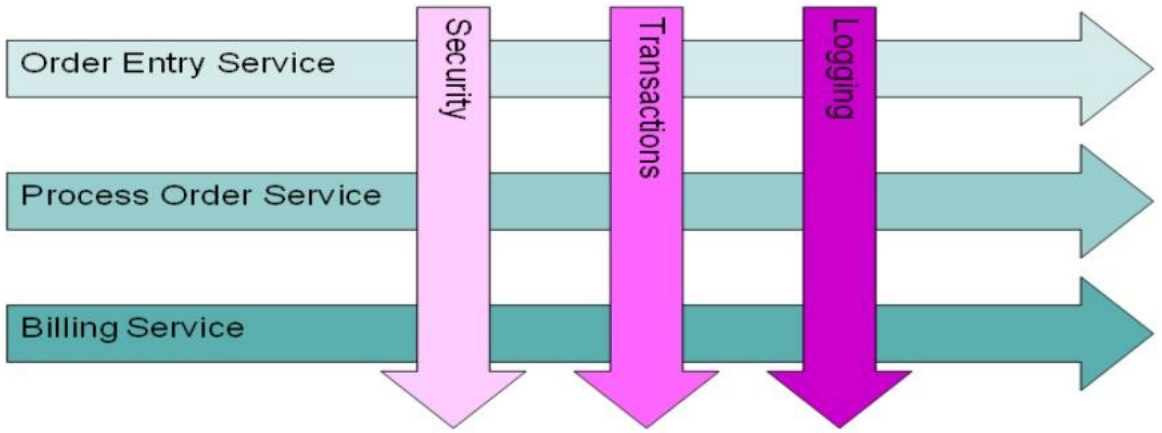
Spring framework

- Introduction Spring framework, where it fits, Spring boot design
- patterns
- Dependency Injection, Configuration-XML,
- Java

  Aspect Oriented Programming, how it helps, configuration,
  examples

Introduction to AOP

- What is AOP?
  - AOP is a style of programming, mainly good in separating cross cutting concerns
- How AOP works?
  - Achieved usages Proxy design Pattern to separate CCC's form actual code
  - Cross Cutting Concern ?
  - Extra code mixed with the actual code is called CCC's
  - Extra code mixed with code lead to maintenance issues
    - Logging
    - validations
    - Auditing
    - Security



Normal Java Class

```
class Account {
public void withdraw() {
// Withdraw Logic
// Authentication
// Logging
// Transaction
}

public void deposit() {
// Deposit Logic
// Authentication
// Logging
// Transaction
}
}
```

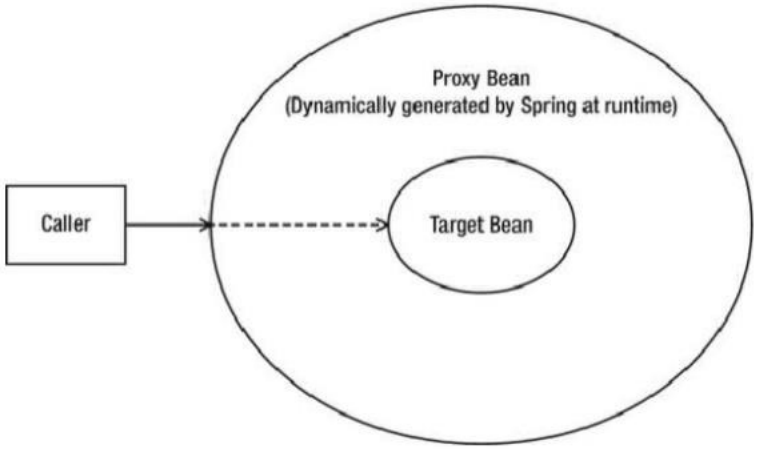
Spring AOP

```
class Account {
public void withdraw() {
// Withdraw Logic
}

public void deposit() {
// deposit Logic
}
}
```

Authentication

Logging

Transaction



AOP - Definitions

- **Advice** defines what needs to be applied and when.
- **Jointpoint** is where the advice is applied.
- **Pointcut** is the combination of different joinpoints where the advice needs to be applied.
- **Aspect** is applying the Advice at the pointcuts.

What, Why spring boot? Intro to Microservice

Spring vs Spring Boot

What is Spring Boot?

Spring Boot = Spring Framework + Embedded HTTP Servers (Tomcat, Jetty) - XML <bean> Configuration or @Configuration

How Spring boot works?

- Spring use autoconfiguration of bean from spring.factories file

- META-INF/spring.factories

1. Enable
2. Disable

Based on @Conditional and @Configuration

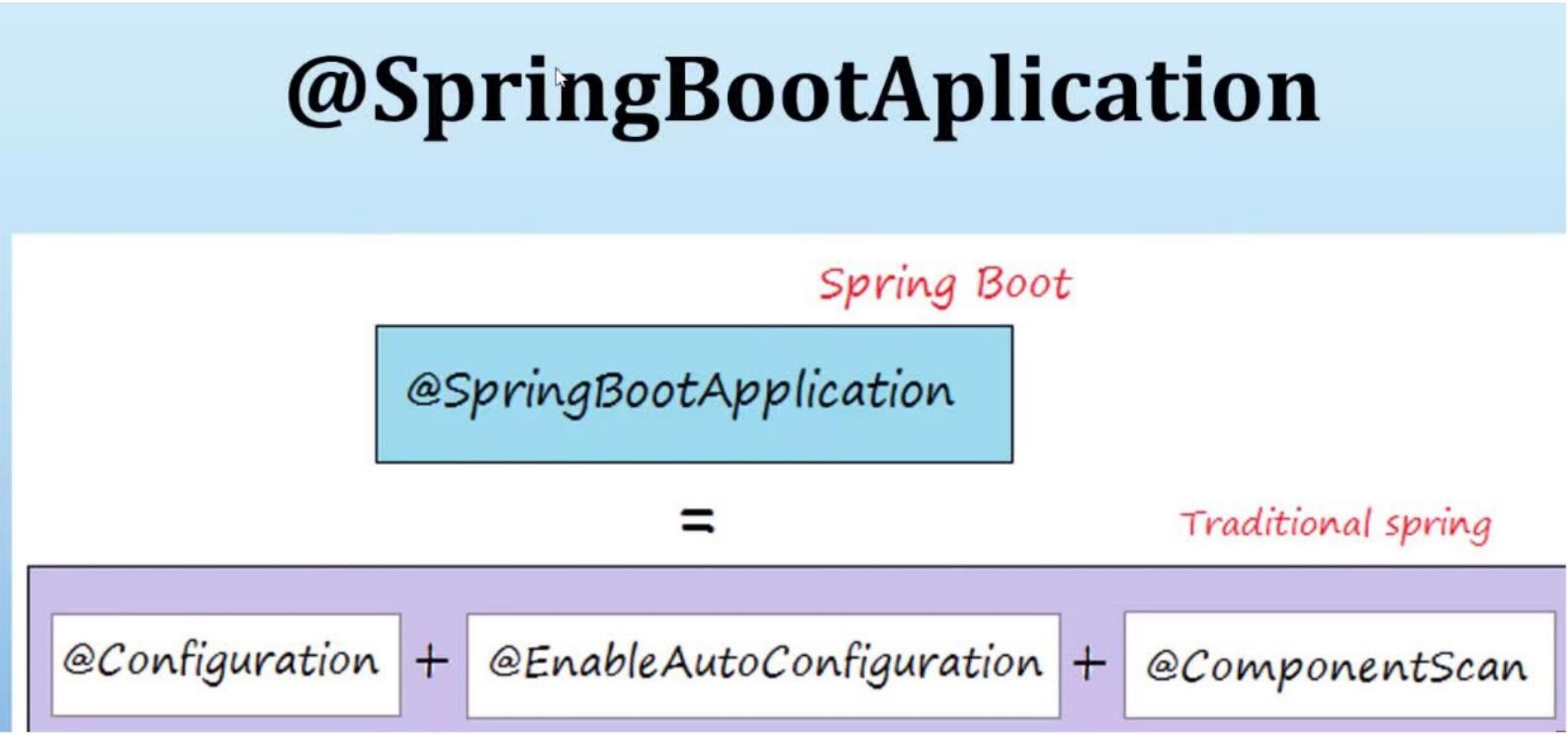
Spring bean auto configuration

Out-of-the-box conditions

Condition	Description
OnBeanCondition	Checks if a bean is in the Spring factory
OnClassCondition	Checks if a class is on the classpath
OnExpressionCondition	Evalutates a SPeL expression
OnJavaCondition	Checks the version of Java
OnJndiCondition	Checks if a JNDI branch exists
OnPropertyCondition	Checks if a property exists
OnResourceCondition	Checks if a resource exists
OnWebApplicationCondition	Checks if a WebApplicationContext exists

Understanding

@SpringBootApplication annotation



Why we need main method in SpringBoot

- Main method is not required for the typical deployment scenario of building a war and placing it in webapps folder of t
- However, if you want to be able to launch the application from within an IDE (e.g. with Eclipse's Run As -> Java Application) while developing or build an executable jar or a war that can run standalone with Spring Boot's embedded tomcat by just java -jar myapp.war command

SpringApplication.run(...);

Following is the high level flow of how spring boot works.

From the run method, the main application context is kicked off which in turn searches for the classes annotated with @configuration, initializes all the declared beans in those configuration classes, and stores those beans in JVM, specifically in a space inside JVM which is known as IOC container. After creation of all the beans, automatically configures the dispatcher servlet and registers the default handler mappings, messageconverts and all other basic things.

run(...) Internal Flow

- Create application context
- Check Application Type
- Register the annotated class beans with the context
- Creates an instance of TomcatEmbeddedServletContainer and adds the context

Using automatic configuration

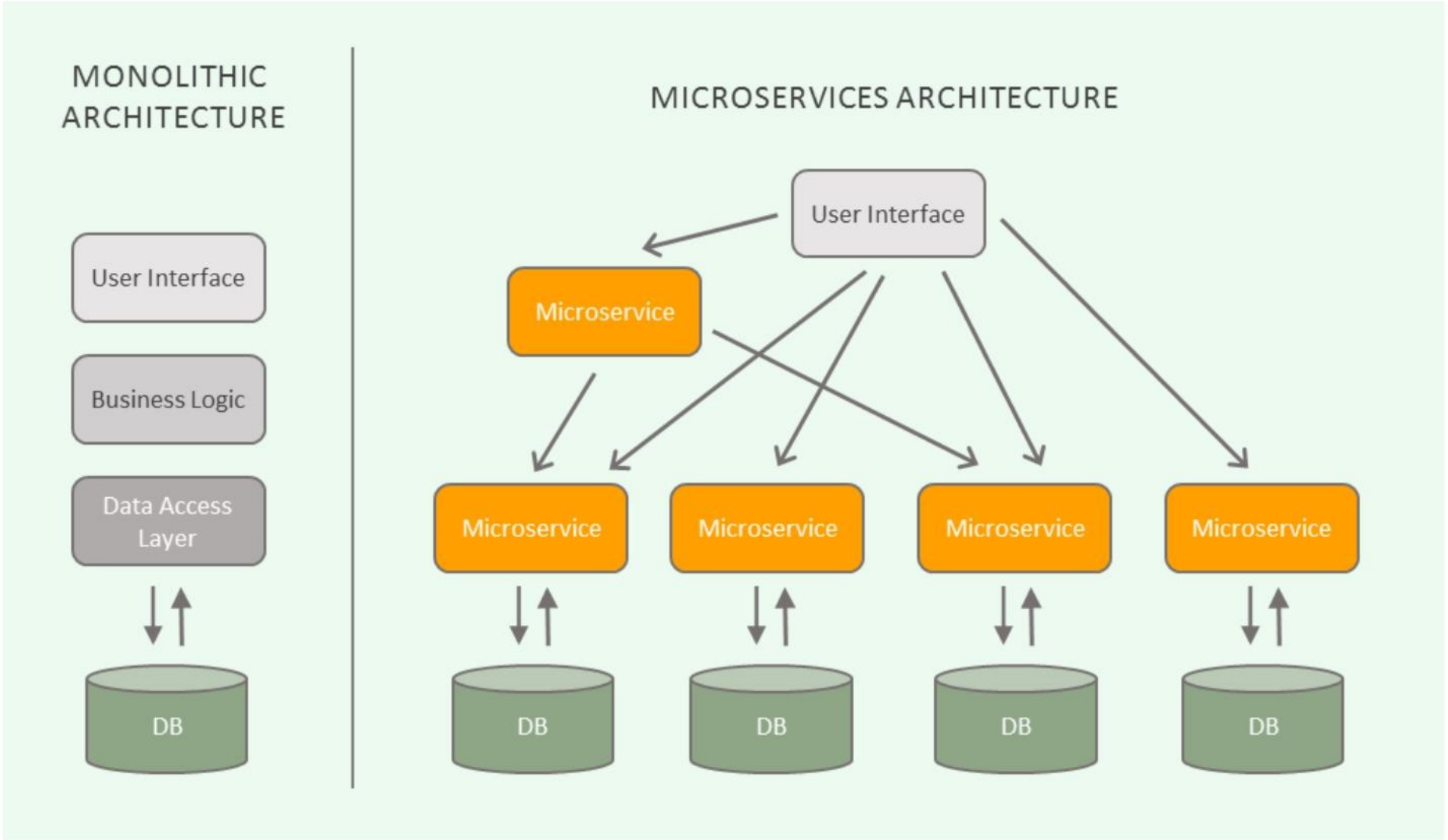
In a nutshell, Spring Boot auto-configuration is a runtime (more accurately, application startup-time) process that considers several factors to decide what Spring configuration should and should not be applied. To illustrate, here are a few examples of the kinds of things that Spring Boot auto-configuration might consider:

- Is Spring's `JdbcTemplate` available on the classpath? If so and if there is a `DataSource` bean, then auto-configure a `JdbcTemplate` bean.
- Is Thymeleaf on the classpath? If so, then configure a Thymeleaf template resolver, view resolver, and template engine.
- Is Spring Security on the classpath? If so, then configure a very basic web security setup.

There are nearly 200 such decisions that Spring Boot makes with regard to auto-configuration every time an application starts up, covering such areas as security, integration, persistence, and web development. All of this auto-configuration serves to keep you from having to explicitly write configuration unless absolutely necessary.

```
$ java -jar readinglist-0.0.1-SNAPSHOT.jar --server.port=8000
```

Spring Boot & microservice



Why spring boot?

- What is Spring Boot?
  - Spring Boot is a Framework from “The Spring Team” to ease the bootstrapping and development of new Spring Applications.
  - It provides defaults for code and annotation configuration to quick start new Spring projects within no time.
  - It follows “Opinionated Defaults Configuration” Approach to avoid lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.

- What is NOT Spring Boot?
  - Spring Boot Framework is not implemented from the scratch by The Spring Team, rather than implemented on top of existing Spring Framework (Spring IO Platform).
  - It is not used for solving any new problems

Why spring boot?

- Why Spring Boot?
  - To ease the Java-based applications Development,

    Unit Test and Integration Test Process.
  - To reduce Development, Unit Test and Integration Test time by providing some defaults.
  - To increase Productivity.

Why spring boot?

- It is very easy to develop Spring Based applications with Java or Groovy.
- It reduces lots of development time and increases productivity.
- It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
- It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
- It follows “Opinionated Defaults Configuration” Approach to reduce Developer effort
- It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.
- It provides CLI (Command Line Interface) tool to develop and test Spring Boot(Java or Groovy) Applications from command prompt very easily and quickly.
- It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle
- It provides lots of plugins to work with embedded and in-memory Databases very easily.

Why Spring Boot?-Too much configuration

```java
@Configuration
@ComponentScan(basePackages={"com.bookapp"})
@EnableAspectJAutoProxy
@PropertySource(value="db.properties")
@EnableTransactionManagement
public class ModelConfig {

    @Autowired
    private Environment environment;

    @Bean(name="dataSource")
    public DataSource getDataSource(){
        DriverManagerDataSource ds=new DriverManagerDataSource();
        ds.setDriverClassName(environment.getProperty("driver"));
        ds.setUrl(environment.getProperty("url"));
        ds.setUsername(environment.getProperty("username"));
        ds.setPassword(environment.getProperty("password"));
        return ds;
    }

    @Bean
    public LocalSessionFactoryBean getSessionFactory(){
        LocalSessionFactoryBean sf=new LocalSessionFactoryBean();
        sf.setDataSource(getDataSource());
        sf.setPackagesToScan("com.bookapp.model.persistance");
        sf.setHibernateProperties(getHibernateProperties());
        return  sf;
    }


    public Properties getHibernateProperties() {
        Properties properties=new Properties();
        properties.setProperty("hibernate.hbm2ddl.auto", "validate");
        properties.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
        properties.setProperty("hibernate.show_sql", "true");
        properties.setProperty("hibernate.formate_sql", "true");
        return properties;
    }

    @Bean
    public PersistenceExceptionTranslationPostProcessor
    getPersistenceExceptionTranslationPostProcessor(){
        PersistenceExceptionTranslationPostProcessor ps=
                new PersistenceExceptionTranslationPostProcessor();
        return ps;
    }

    @Bean(name="transactionManager")
    //@Autowired
    public HibernateTransactionManager getHibernateTransactionManager(SessionFactory factory){
        HibernateTransactionManager tm=new HibernateTransactionManager();
        tm.setSessionFactory(factory);
        return tm;
    }
}
```

Hello World spring boot

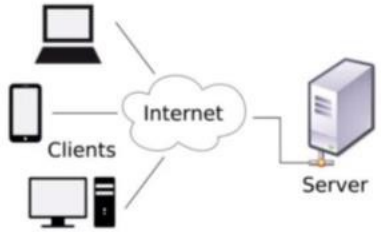
```java
@RestController
@SpringBootApplication
public class RestApplication {

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

    public String hello(){
        return "hello to spring boot";
    }

}
```

```
server.servlet-path=/bookapp
server.port=8080
```

Some fundamentals

What is a Client?

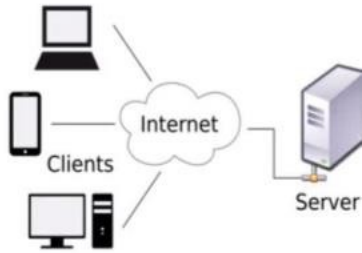
A client is
- Computer
- Mobile
- Host



That sends that **Request** to server for any service, data using HTTP Protocol(using URL) and receive a Response.

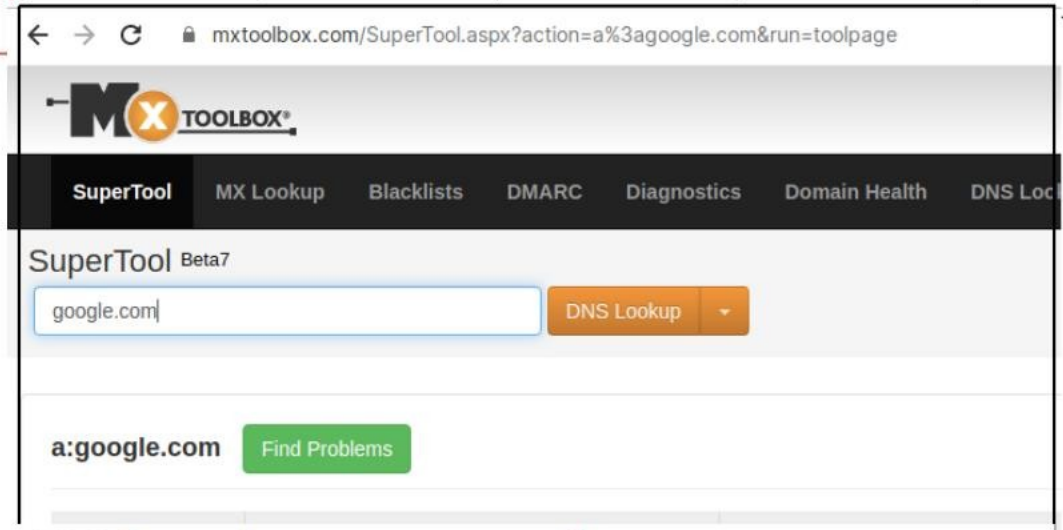
What is a Server?

A Server is Remote Machine That receive request from client and Serve them with appropriate information as **Response** using HTTP/s.



What Happens when we type Google.com

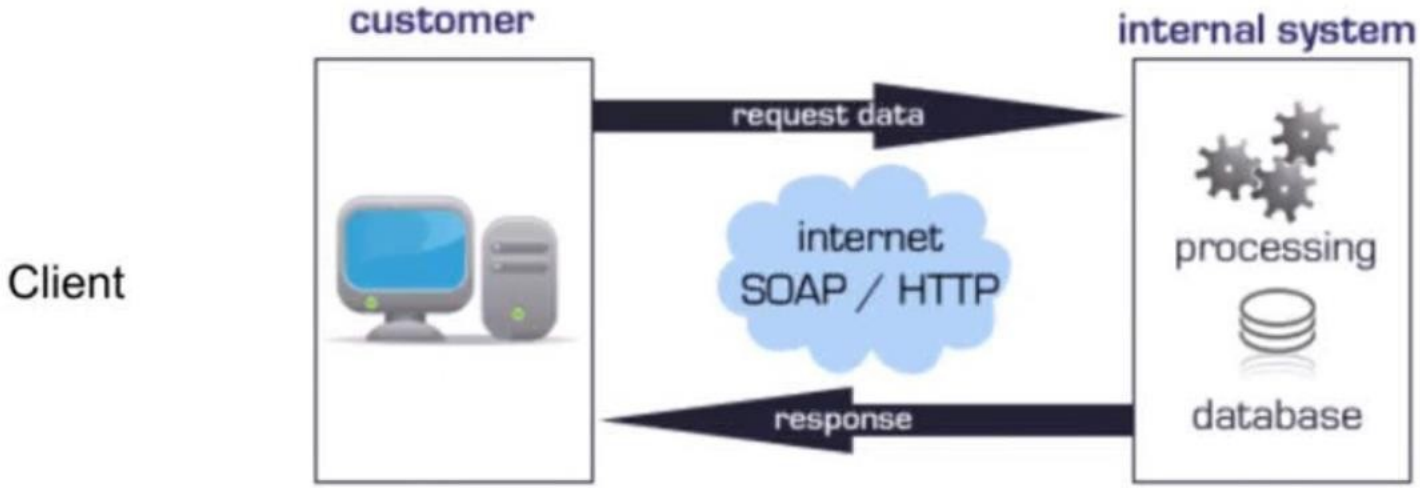
- You enter a URL e.g google.com, Client is browser, request goes to the DNS.
- DNS lookup happens and IP address searched and if found sends back to Browser.
- Now Browser then sends over an HTTP/HTTPS request to the web server's IP.
- The server responds with Resource Page.(index.htm).
- Now Browser will Render the HTML page.



What happens when you type a URL in the browser and press enter?

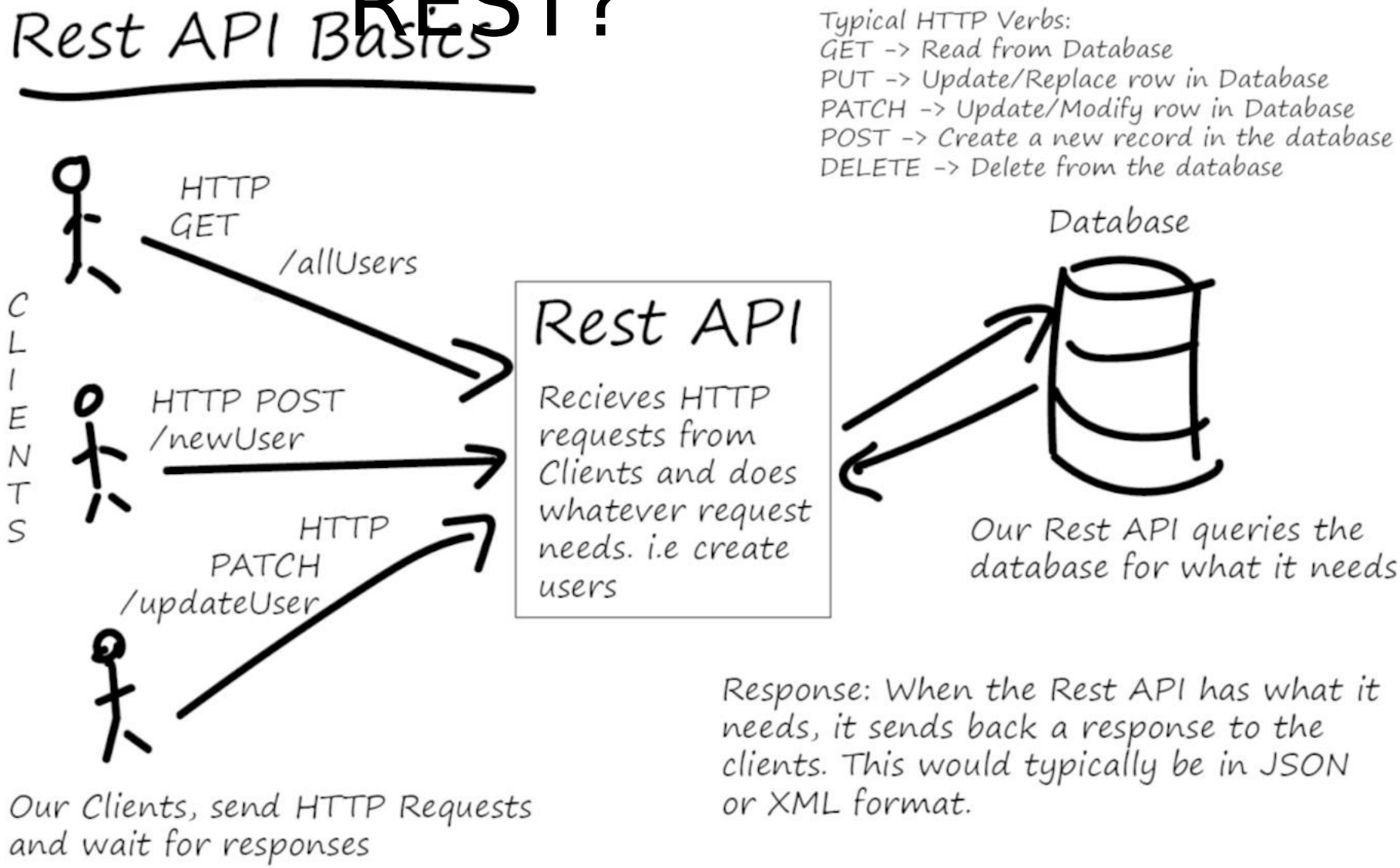
What is Web Service?

They provide a common platform that allows multiple applications built on various programming languages to have the ability to communicate with each other



Service Oriented Architecture

What is REST?


Rest API Basics
Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database
HTTP
GET
/allUsers
CLIENTS
HTTP POST
/newUser
HTTP
PATCH
/updateUser
Rest API
Recieves HTTP requests from Clients and does whatever request needs. i.e create users
Database
Our Rest API queries the database for what it needs
Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.
Our Clients, send HTTP Requests and wait for responses

REST characteristics

- Resources: Application state and functionality are abstracted into resources.
  - *URI:* Every resource is **uniquely addressable** using URIs.
  - *Uniform Interface:* All resources share a uniform

interface for the transfer of state between client and resource, consisting of

    - Methods: Use only HTTP methods such as **GET**, **PUT**, **POST**, **DELETE**, **HEAD**
    - Representation
- Protocol (The constraints and the principles)
  - *Client-Server*
  - *Stateless*
  - *Cacheable*
  - *Layered*

What is Authentication?

Authentication is a process of presenting your credentials like username, password or another secret key to the system and the system to validate your credentials or you.

	Authentication	Authorization
Meaning	"Are you allowed to *access* X app?"	"What are you allowed to *modify* in X app?"
Methods	Password, 2FA, MFA, X509 Certificates, Biometric authenticators, WebAuthN	Access control for URI, Access control lists etc.

In the API terms;
Authentication is used to protect the content over web mean only a valid user with valid credentials can access that API endpoint. These credentials tell the system about who you are. Which enables the system to ensures and confirms a user's identity.

Here system can be anything, it can be a computer, phone, bank or any physical office premises.

Authentication Types

Basic authentication - String is encoded with Base64.
curl --header "Authorization: Basic am9objpzZWNyZXQ=" my-website.com

Digest Authentication - Authentication is performed by transmitting the password in an ENCRYPTED form.(With Some Salt etc)

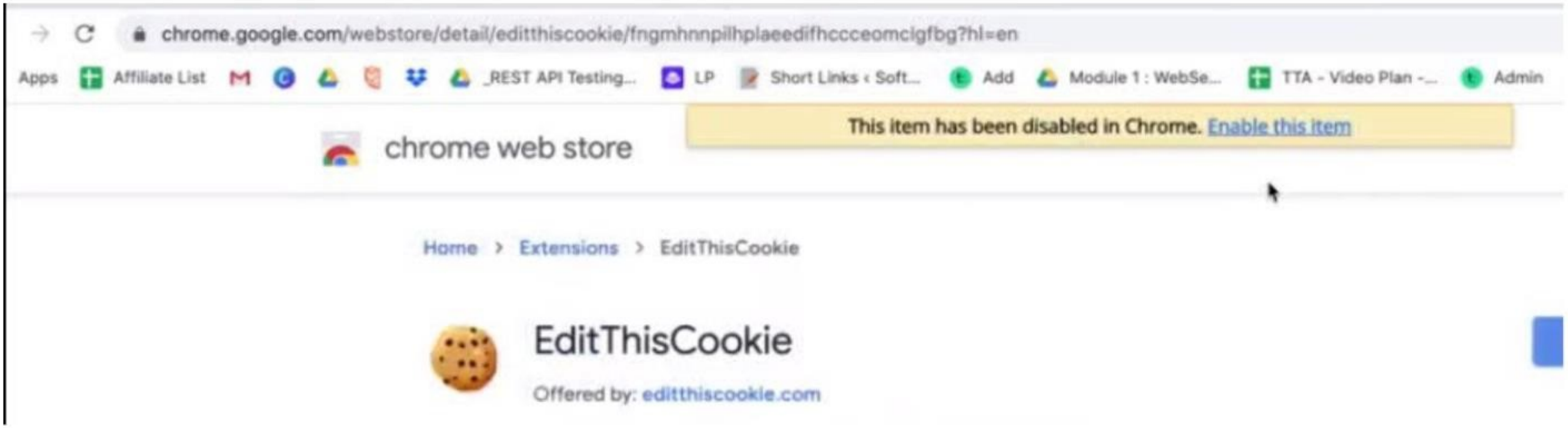
OAuth- Authentication protocol that allows you to approve one application interacting with another on your behalf without giving away your password.

Cookies

Cookies are usually small text files, given ID tags that are stored on your computer's browser directory or program data subfolders.

GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: theme=light; sessionToken=abc123

Record the user's browsing activity.
Which pages were visited in the past.



JSON Basics

JSON: JavaScript Object Notation.

When exchanging data between a browser and a server, the data can only be text.

JSON Data Types.

- JSON Strings : { "name":"John" }
- JSON Numbers : { "age":30 }
- JSON Objects : {
"employee":{ "name":"John", "age":30, "city":"New York" }
}
- JSON Arrays
  {"employees":["John", "Anna", "Peter"] }
- JSON Booleans
  { "sale":true }
- JSON null
  { "middlename":null }

HyperText Transfer Protocol(HTTP)

- HTTP (Hypertext Transfer Protocol) is perhaps the most popular application protocol used in the Internet (or The WEB).
- An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a pull protocol, the client pulls information from the server (instead of server pushes information down to the client).
- HTTP is a stateless protocol. In other words, the current request does not know what has been done in the previous requests.
- HTTP permits negotiating of data type and representation, so as to allow systems to be built independently of the data being transferred.

HTTP methods & status code

Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options

- 1xx: Informational Messages
- 2xx: Successful
- 3xx: Redirection
- 4xx: Client Error
- 5xx: Server Error

HTTP Status Codes

For great REST services the correct usage of the correct HTTP status code in a response is vital.

1xx – Informational

This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line

100 – Continue
101 – Switching Protocols
102 – Processing

2xx – Successful

This class of status code indicates that the client's request was successfully received, understood, and accepted.

200 – OK
201 – Created
202 – Accepted
203 – Non-Authoritative Information
204 – No Content
205 – Reset Content
206 – Partial Content
207 – Multi-Status

3xx – Redirection

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request.

300 – Multiple Choices
301 – Moved Permanently
302 – Found
303 – See Other
304 – Not Modified
305 – Use Proxy
307 – Temporary Redirect

4xx – Client Error

The 4xx class of status code is intended for cases in which the client seems to have erred.

400 – Bad Request
401 – Unauthorised
402 – Payment Required
403 – Forbidden
404 – Not Found
405 – Method Not Allowed
406 – Not Acceptable
407 – Proxy Authentication Required
408 – Request Timeout
409 – Conflict
410 – Gone
411 – Length Required
412 – Precondition Failed
413 – Request Entity Too Large
414 – Request URI Too Long
415 – Unsupported Media Type
416 – Requested Range Not Satisfiable
417 – Expectation Failed
422 – Unprocessable Entity
423 – Locked
424 – Failed Dependency
425 – Unordered Collection
426 – Upgrade Required

5xx – Server Error

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request.

500 – Internal Server Error
501 – Not Implemented
502 – Bad Gateway
503 – Service Unavailable
504 – Gateway Timeout
505 – HTTP Version Not Supported
506 – Variant Also Negotiates
507 – Insufficient Storage
510 – Not Extended

Examples of using HTTP Status Codes in REST

201 – When doing a POST to create a new resource it is best to return 201 and not 200.
204 – When deleting a resources it is best to return 204, which indicates it succeeded but there is no body to return.
301 – If a 301 is returned the client should update any cached URI's to point to the new URI.
302 – This is often used for temporary redirect's, however 303 and 307 are better choices.
409 – This provides a great way to deal with conflicts caused by multiple updates.
501 – This implies that the feature will be implemented in the future.

Special Cases

306 – This status code is no longer used. It used to be for switch proxy.
418 – This status code from RFC 2324. However RFC 2324 was submitted as an April Fools' Joke. The message is *I am a teapot.*

Key	Description
Black	HTTP version 1.0
Blue	HTTP version 1.1
Aqua	Extension RFC 2295
Green	Extension RFC 2518
Yellow	Extension RFC 2774
Orange	Extension RFC 2817
Purple	Extension RFC 3648
Red	Extension RFC 4918

Difference between GET and POST Http methods:

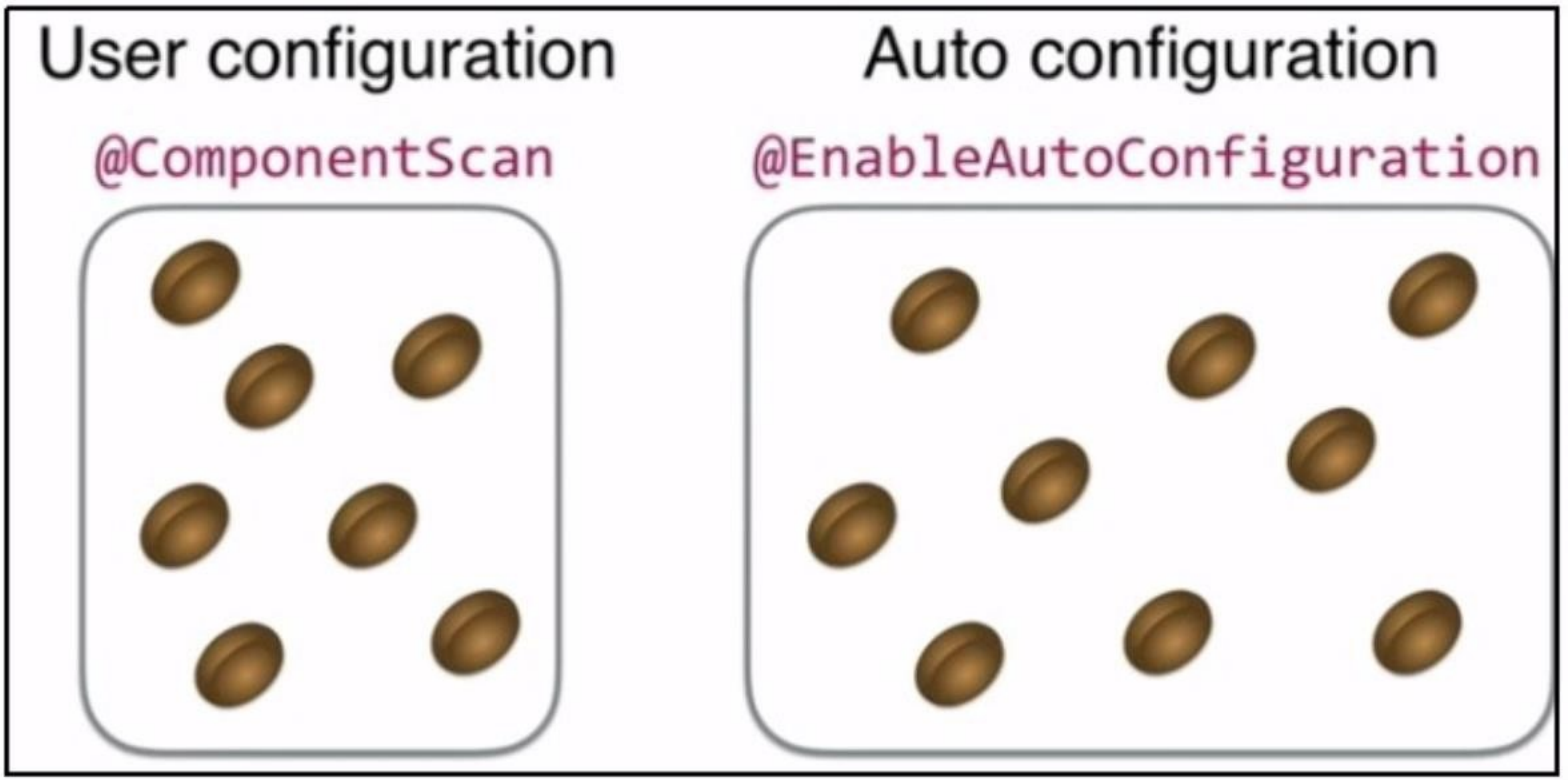
	GET	POST
1	It is used for getting information from the server side	It is used to post the information to the server.
2	It performs read-only operation	It performs update/write operation.
3	Whatever the request can be sent that is append with URL	Whatever request send to the server that is not exposed in the URL
4	Only text or character type of data can be sent	Any type of data can be send
5	Limited amount of data can be send (up to 2KB)	No length limit of data that we can send
6	It provides less security because the user's request is directly exposed in URL bar. Hence, sensitive information is not recommended to be send.	It provides high security
7	Bookmarking is possible which is the advantage	Bookmarking is not possible
8	Caching is also possible which is also another advantage of get method	Caching is not possible

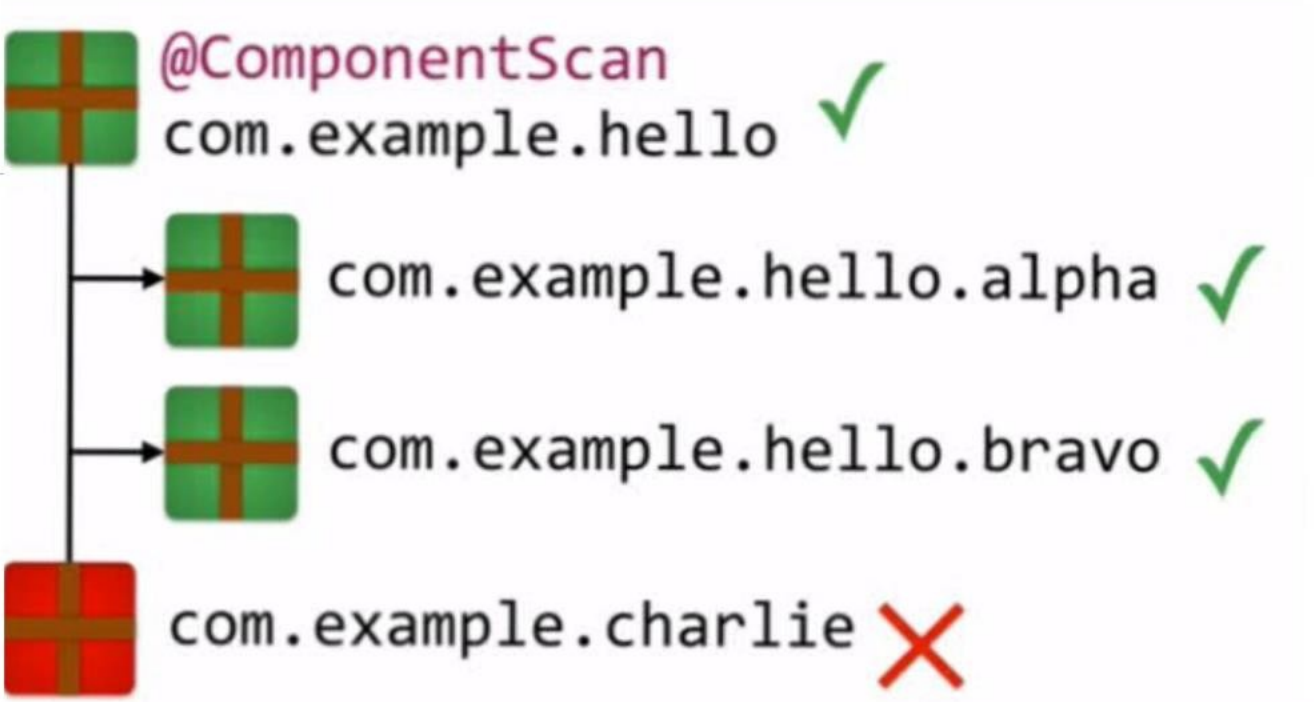
Generally – **not** necessarily – **POST** APIs are used to create a new resource on server. So when you invoke the same **POST** request N times, you will have N new resources on the server. So, **POST is not idempotent**.

PUT	POST
Replacing existing resource or Creating if resource is not exist *http://www.example.com/customer/{id}* *http://www.example.com/customer/123/orders/456* Identifier is chosen by the client	Creating new resources(and subordinate resources) *http://www.example.com/ customer/* *http://www.example.com/customer/123/orders* Identifier is returned by server
Idempotent i.e. if you PUT a resource twice, it has no effect. Ex: Do it as many times as you want, the result will be same. x=1;	POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Ex: x++;
Works as specific	Works as abstractive
If you create or update a resource using PUT and then make that same call again, the resource is still there and still has the same state as it did with the first call.	Making two identical POST requests will most-likely result in two resources containing the same information.

spring boot hello world, CRUD application

@SpringBootApplication





@Repository @Configuration

@Component

@Controller @Service

@RestController

Spring boot hello world

```java
@RestController
@RequestMapping(path = "api")
public class Hello {

    private Logger logger = LoggerFactory.getLogger(Hello.class);

    @GetMapping(path = "hello/{name}/{address}")
    public String helloWithName(@PathVariable(name = "name") String name,
            @PathVariable(name = "address") String address) {
        return "hello spring rest:" + name + ": " + address;
    }

    @GetMapping(path = "hello")
    public String helloWithRequestParam(
            @RequestParam(name = "name", required = false, defaultValue = "amit") String name,
            @RequestParam(name = "age", required = false, defaultValue = "30") int age) {

        logger.info("spring boot hello world.....");
        return "hello spring rest:" + name + ": " + age;
    }

}
```

Creating DAO, DTO

```java
@Entity
@Table(name="book_table")
public class Book {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String title;
    private String author;
    private double price;
    private String pubName;
    @Temporal(TemporalType.DATE)
    private Date pubDate;
```

```java
@Repository
public interface BookRepo extends CrudRepository<Book, Integer> {
    public List<Book>findByTitle(String title);
}
```

Hibernate/JPA integration

- Step 1: configure hibernate in boot

```
spring.datasource.url=jdbc:mysql://localhost:3306/boot_demo?useSSL=false
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver


spring.jpa.hibernate.ddl-auto=update
logging.level.org.springframework.web: DEBUG
logging.level.org.hibernate: ERROR
spring.jpa.show-sql=true
```

- Step 2: Annotate POJO

```
@Entity
@Table(name="book_table")
public class Book {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    @Column(nullable=false, unique=true, name="isbn")
    private String isbn;
```

- Step 3: Just inject EntityManager

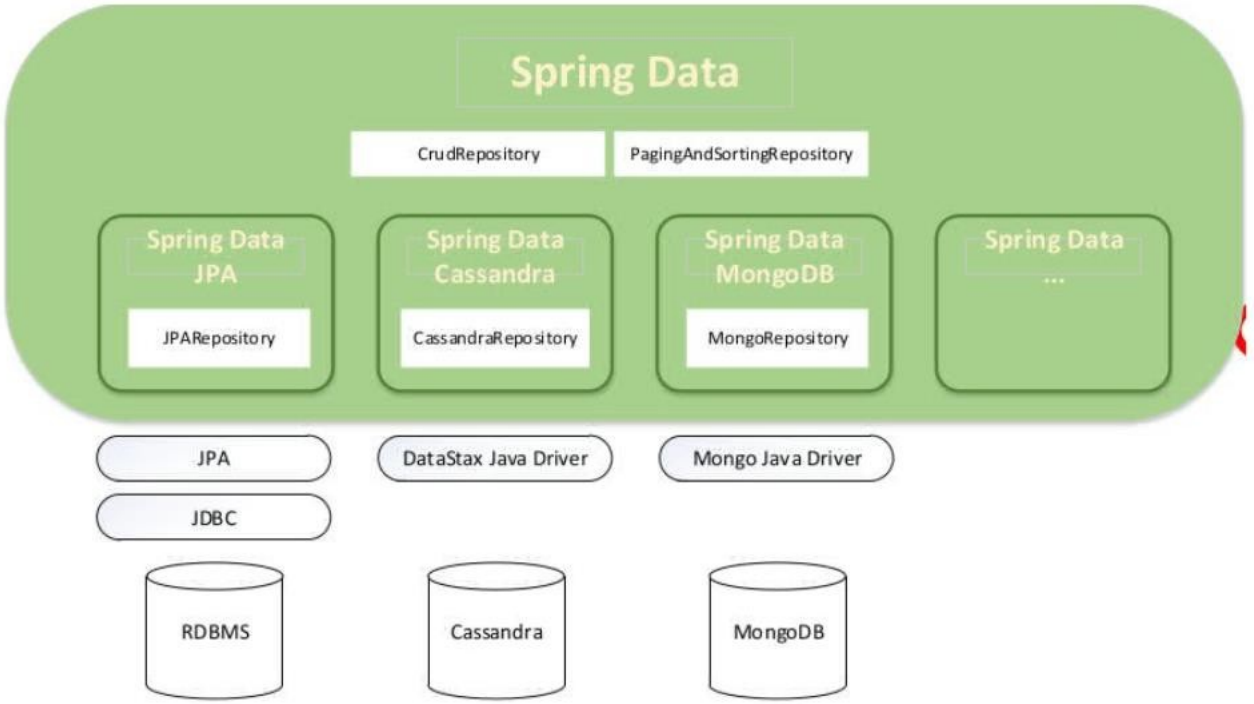
```
@Repository
public class BookDaoImplHib implements BookDao{
    @Autowired
    private EntityManager em;
    @Override
    public List<Book> getAllBooks() {
        return em.createQuery("from Book", Book.class).getResultList();
    }
```

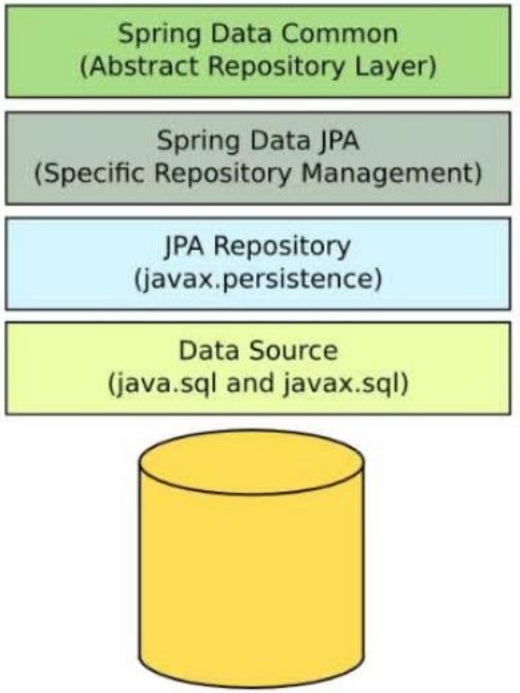
Spring boot bootstrapping

```
@SpringBootApplication
@ComponentScan({"com.bookapp"})
@EntityScan("com.bookapp.model.dao")
@EnableJpaRepositories("com.bookapp.model.dao")
public class BookappspringbootApplication {

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

}
```

- https://stackoverflow.com/questions/40384056/consider-defining-a-bean-of-type-package-in-your-configuration-spring-boot

Spring Data






```
public interface CrudRepository<T, ID extends Serializable>
                 extends Repository<T, ID> {
        <S extends T> S save(S entity);
        T findOne(ID primaryKey);
        Iterable<T> findAll();
        Long count();
        void delete(T entity);
        boolean exists(ID primaryKey);
        // … more functionality omitted.
}
```

Query Lookup Strategy

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with prepended %)
ngWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with appended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false

Query Lookup Strategy

- @NamedQuery

```
@Entity
@NamedQuery(name = "User.findByEmailAddress",
  query = "select u from User u where u.emailAddress = ?1")
public class User {

}
public interface UserRepository extends JpaRepository<User, Long> {

  List<User> findByLastname(String lastname);

  User findByEmailAddress(String emailAddress);
}
```

@Query

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query("select u from User u where u.emailAddress = ?1")
  User findByEmailAddress(String emailAddress);
}
```

Spring boot implement service layer

Step 4: Service layer

```java
public interface BookService {

    public List<Book> getAllBooks();
    public Book getBookById(int id);
    public Book addBook(Book book);
    public List<Book> getBookByTitle(String title);
    public Book updateBook(int bookId, Book book);
    public Book deleteBook(int bookId);

}
```

```java
@ResponseStatus(code=HttpStatus.NOT_FOUND)
public class BookNotFoundException extends RuntimeException{
    private static final long serialVersionUID = 4351688402749113855L;

}
```

Service layer implementation

```java
@Service
@Transactional
public class BookServiceImpl implements BookService{
    @Autowired
    private BookRepo bookRepo;

    @Override
    public List<Book> getAllBooks() {
        return (List<Book>) bookRepo.findAll();
    }

    @Override
    public Book getBookById(int id) {
        return bookRepo.findById(id).orElseThrow(BookNotFoundException::new);
    }

    @Override
    public Book addBook(Book book) {
        return bookRepo.save(book);
    }

    @Override
    public  List<Book> getBookByTitle(String title) {
        return bookRepo.findByTitle(title);
    }

    @Override
    public Book updateBook(int bookId, Book book) {
```

Service layer implementation

```java
@Override
public Book addBook(Book book) {
    return bookRepo.save(book);
}

@Override
public  List<Book> getBookByTitle(String title) {
    return bookRepo.findByTitle(title);
}

@Override
public Book updateBook(int bookId, Book book) {
    Book bookToUpdated=getBookById(bookId);
    bookToUpdated.setPrice(book.getPrice());

    return bookRepo.save(bookToUpdated);
}

@Override
public Book deleteBook(int bookId) {
    Book bookToBeDeleted=getBookById(bookId);
    bookRepo.delete(bookToBeDeleted);
    return bookToBeDeleted;
}
```

Rest Controller

Step 5: Rest controller

```java
@RestController
@RequestMapping(path="api")
public class BookRestController {

    @Autowired
    private BookService bookService;

    @GetMapping(path="book", produces=MediaType.APPLICATION_JSON_VALUE)
    public List<Book>getAllBooks(){
        return bookService.getAllBooks();
    }

    @GetMapping(path="book/{id}", produces=MediaType.APPLICATION_JSON_VALUE)
    public Book getBookById(@PathVariable(name="id")int id){
        return bookService.getBookById(id);
    }


    @PostMapping(path="book", produces=MediaType.APPLICATION_JSON_VALUE
            ,consumes=MediaType.APPLICATION_JSON_VALUE)
    public Book addBook(@RequestBody Book book){
        return bookService.addBook(book);
    }


    @PutMapping(path="book/{id}", produces=MediaType.APPLICATION_JSON_VALUE
            ,consumes=MediaType.APPLICATION_JSON_VALUE)
    public Book updateBook(@PathVariable(name="id")int id,  @RequestBody Book book){
        return bookService.updateBook(id, book);
    }
```

Rest controller

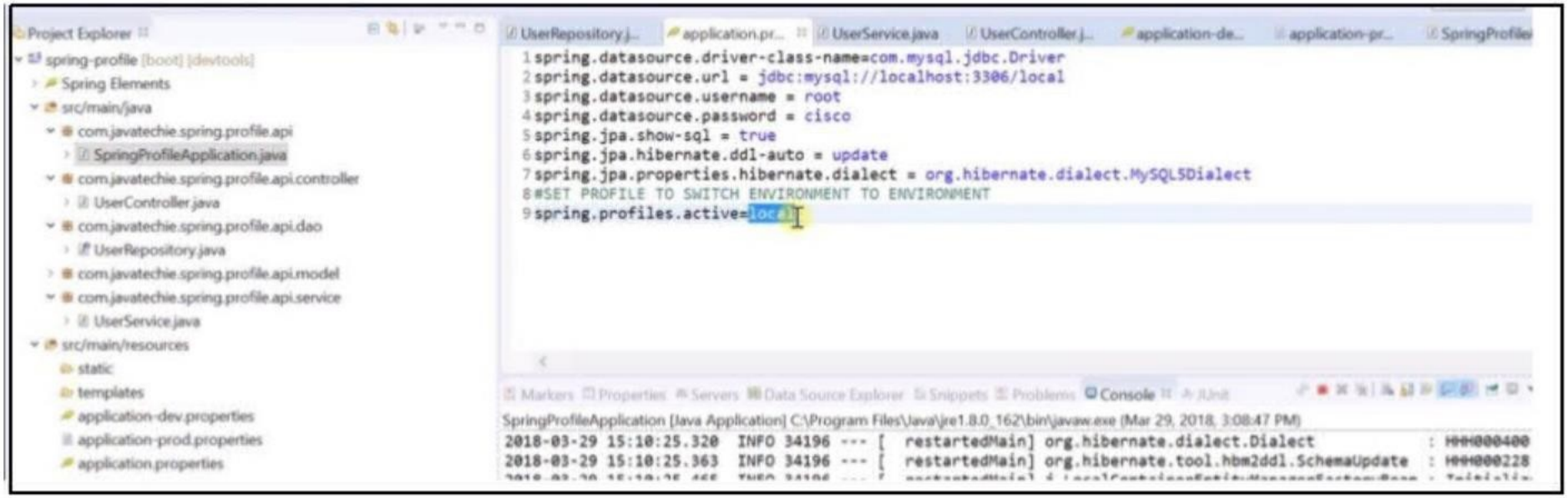
```
@PutMapping(path="book/{id}", produces=MediaType.APPLICATION_JSON_VALUE
        ,consumes=MediaType.APPLICATION_JSON_VALUE)
public Book updateBook(@PathVariable(name="id")int id,  @RequestBody Book book){
    return bookService.updateBook(id, book);
}


@DeleteMapping(path="book/{id}", produces=MediaType.APPLICATION_JSON_VALUE)
public Book deleteBook(@PathVariable(name="id")int id){
    return bookService.deleteBook(id);
}
```

Step 6: Spring boot config

```
server.servlet.context-path=/bookapp
server.port=8080

spring.datasource.url=jdbc:mysql://localhost:3306/hcl_jdbc?useSSL=false
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
logging.level.org.springframework.web: DEBUG
logging.level.org.hibernate: ERROR
spring.jpa.show-sql=true
```

```
@SpringBootApplication
public class BookappApplication implements CommandLineRunner{

    @Autowired
    private BookService bookService;

    public static void main(String[] args) {
        SpringApplication.run(BookappApplication.class, args);
    }
    @Override
    public void run(String... args) throws Exception {
        bookService.addBook(new Book("head first java", "katthy", 560, "abc", new Date()));
        bookService.addBook(new Book("head first design pattern", "katthy", 760, "abc", new Date()));
    }

}
```

Spring boot profile



Spring REST exception handling

Step 7: Handling BookNotFoundException

```java
@ResponseStatus(value=HttpStatus.NOT_FOUND)
public class BookNotFoundException extends RuntimeException {
    private static final long serialVersionUID = -2906350775886465681L;

}
```

```java
@ControllerAdvice
@RestController
public class ExceptionHandlerController {

    @ExceptionHandler(BookNotFoundException.class)
    public ResponseEntity<Object> handleBookNotFoundEx(BookNotFoundException ex, WebRequest req){
        ErrorDetails details=new ErrorDetails(new Date(), "book not found", req.getDescription(false));
        return new ResponseEntity<Object>(details, HttpStatus.NOT_FOUND);
    }
}
```

```java
@ExceptionHandler(Exception.class)
public final ResponseEntity<Object> handleAllExceptions(Exception ex,
        WebRequest request) {
    ErrorDetails errorDetails = new ErrorDetails(new Date(),
            ex.getMessage(), request.getDescription(false));
    return new ResponseEntity<Object>(errorDetails,HttpStatus.INTERNAL_SERVER_ERROR);
}
```

```java
public class ErrorDetails {
    private Date timestamp;
     private String message;
     private String details;
```

REST Handling HttpStatus code

Step 8: Handling HttpStatus code

- @RestController
- // @RestController=@Controller + @ResponseBody

HTTP Status Codes

Code	Description	Code	Description
200	OK	400	Bad Request
201	Created	401	Unauthorized
202	Accepted	403	Forbidden
301	Moved Permanently	404	Not Found
303	See Other	410	Gone
304	Not Modified	500	Internal Server Error
307	Temporary Redirect	503	Service Unavailable

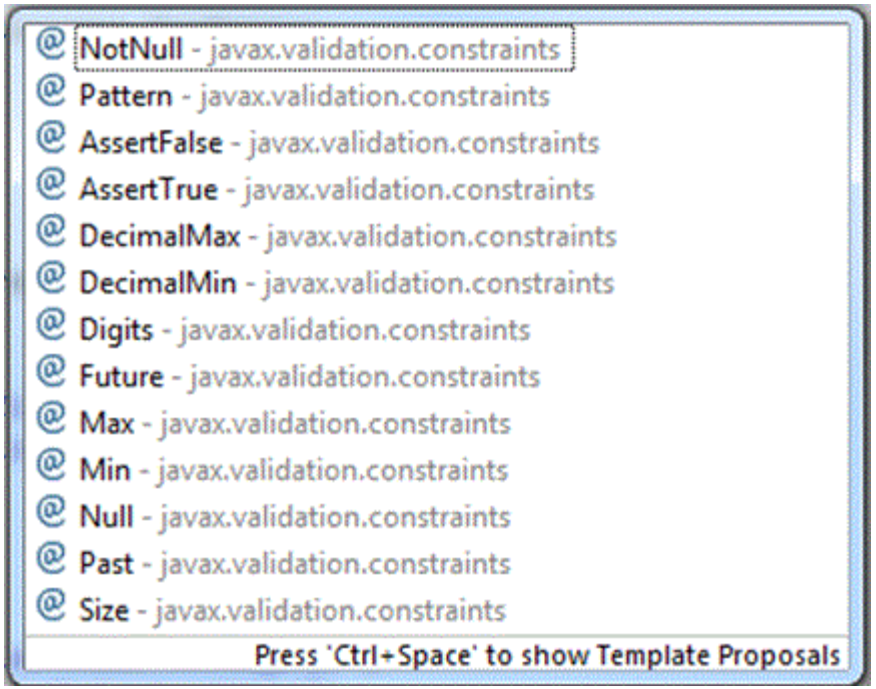
REST Using Response Entity and Status code

Step 8: Using ResponseEntity

```
@RestController
@RequestMapping(path="api")
public class BookRestController {

    @Autowired
    private BookService bookService;

    @GetMapping(path="book", produces=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<List<Book> > getAllBooks(){
        return new ResponseEntity<List<Book>>(bookService.getAllBooks(), HttpStatus.OK);
    }

    @GetMapping(path="book/{id}", produces=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Book> getBookById(@PathVariable(name="id")int id){
        return new ResponseEntity<Book>(bookService.getBookById(id), HttpStatus.OK);
    }


    @PostMapping(path="book", produces=MediaType.APPLICATION_JSON_VALUE
            ,consumes=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Book> addBook(@RequestBody Book book){
        return new ResponseEntity<Book>(bookService.addBook(book), HttpStatus.CREATED);
    }


    @PutMapping(path="book/{id}", produces=MediaType.APPLICATION_JSON_VALUE
            ,consumes=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Book> updateBook(@PathVariable(name="id")int id,  @RequestBody Book book){
        return new ResponseEntity<Book>(bookService.updateBook(id, book), HttpStatus.CREATED);
    }
```

```
    @DeleteMapping(path="book/{id}", produces=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Void> deleteBook(@PathVariable(name="id")int id){
         bookService.deleteBook(id);
         return new ResponseEntity<Void>(HttpStatus.NO_CONTENT);
    }
```

JSR 303validation



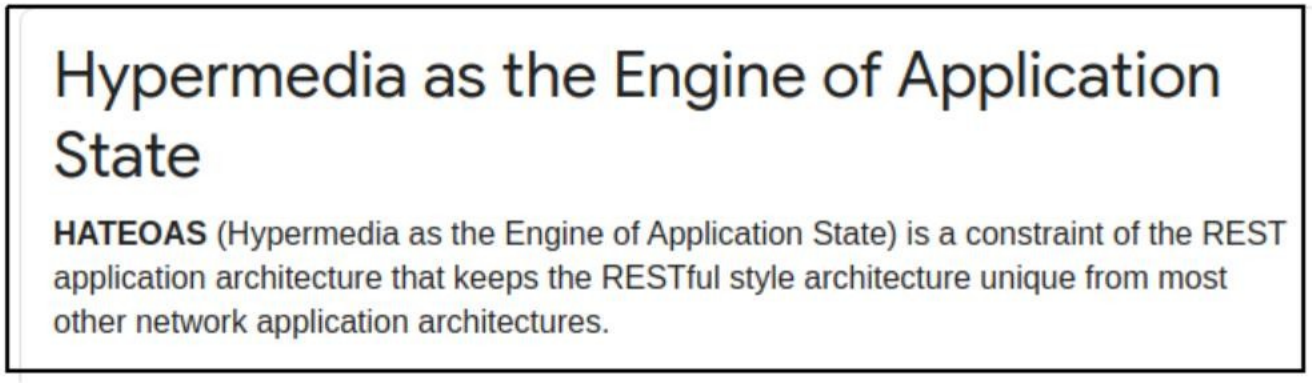
Step 9: JSR 303 validation

```java
@Entity
@Table(name="book_table")
public class Book {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    @Size(min=2, message="title should have atleast 4 characters")
    private String title;
```

```java
@PostMapping(path="book", produces=MediaType.APPLICATION_JSON_VALUE
        ,consumes=MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Book> addBook(@Valid @RequestBody Book book){
    return new ResponseEntity<Book>(bookService.addBook(book), HttpStatus.CREATED);
}
```

```java
@ControllerAdvice
@RestController
public class ExceptionHandlerController extends ResponseEntityExceptionHandler{

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
        HttpHeaders headers, HttpStatus status, WebRequest request) {
      ErrorDetails errorDetails = new ErrorDetails(new Date(), "Validation Failed",
          ex.getBindingResult().toString());
      return new ResponseEntity<Object>(errorDetails, HttpStatus.BAD_REQUEST);
    }
```

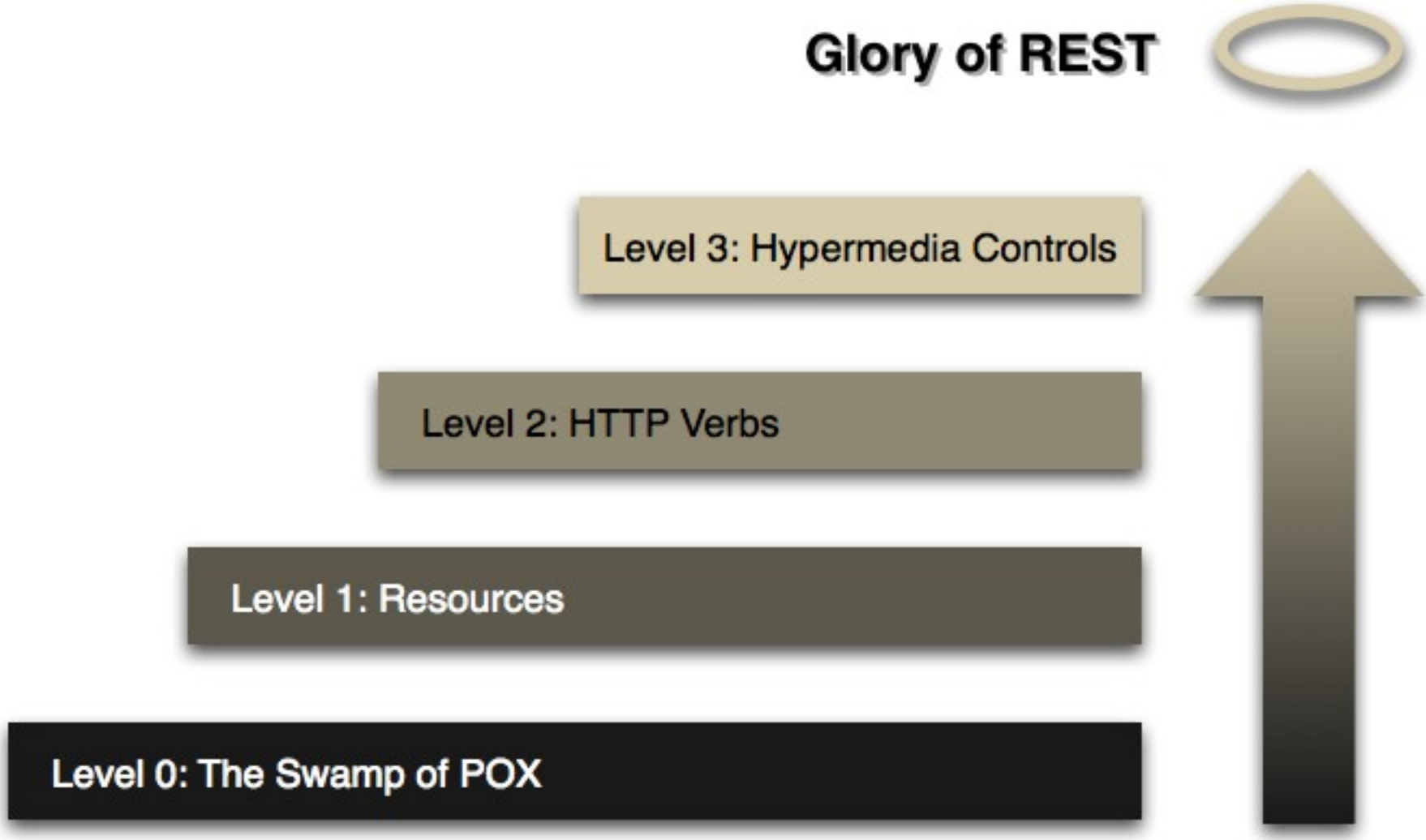
HATEOAS


Hypermedia as the Engine of Application State
HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture that keeps the RESTful style architecture unique from most other network application architectures.

Step 10: hateoas

Hypermedia as the Engine of Application State (**HATEOAS**) is a component of the REST application architecture that distinguishes it from other network application architectures. With **HATEOAS**, a client interacts with a network application whose application servers provide information dynamically through hypermedia.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.acme.account+json
Content-Length: ...

{
    "account": {
        "account_number": 12345,
        "balance": {
            "currency": "usd",
            "value": 100.00
        },
        "links": {
            "deposit": "/accounts/12345/deposit",
            "withdraw": "/accounts/12345/withdraw",
            "transfer": "/accounts/12345/transfer",
            "close": "/accounts/12345/close"
        }
    }
}
```


Glory of REST
Level 3: Hypermedia Controls
Level 2: HTTP Verbs
Level 1: Resources
Level 0: The Swamp of POX

Hateoas

- HATEOAS- Hypertext as the engine of Application state
- Use link in the application state (Response data)
- **Advantage:**
  - the client can have a single entry point to the application and att the further action can be taken
  - based on the links in the response data
  - Reduce dependency between client and service, so that the server can make changes to its URI without breaking the client side code
- **Implementation:**
- RepresentationModel
- WebMvcLinkBuilder
- Link

Spring REST :HATEOAS Steps

- Step 1: Put Hateoas dependencies
- Step 2: Make POJO extends a class RepresentationModel

```
@Entity
@Table(name = "book_table")
public class Book extends RepresentationModel<Book>{
```

-

```
@GetMapping(path = "/book2/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
public EntityModel<Book> getBookById2(@PathVariable(name = "id") int id) {
    Link link=WebMvcLinkBuilder
            .linkTo(WebMvcLinkBuilder.methodOn(BookRestController.class).getBookById2(id))
            .withSelfRel();

    Book book = bookService.getBookById(id);
    book.add(link);

    return EntityModel.of(book);
}
```

Spring REST :HATEOAS Steps

```
// get all the books
@GetMapping(path = "book", produces = MediaType.APPLICATION_JSON_VALUE)
public CollectionModel<Book> getAllBooks() {
    List<Book> books = bookService.getAllBooks();
    for(Book book: books) {
        Link link=WebMvcLinkBuilder
                .linkTo(WebMvcLinkBuilder.methodOn(BookRestController.class).getBookById(book.getId()))
                .withSelfRel();
        book.add(link);
    }
    return CollectionModel.of(books);
}
```

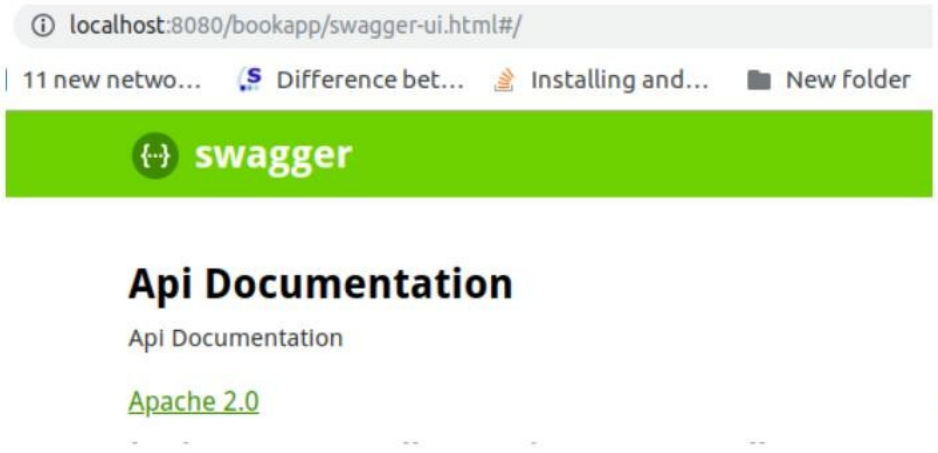
Swagger documentation

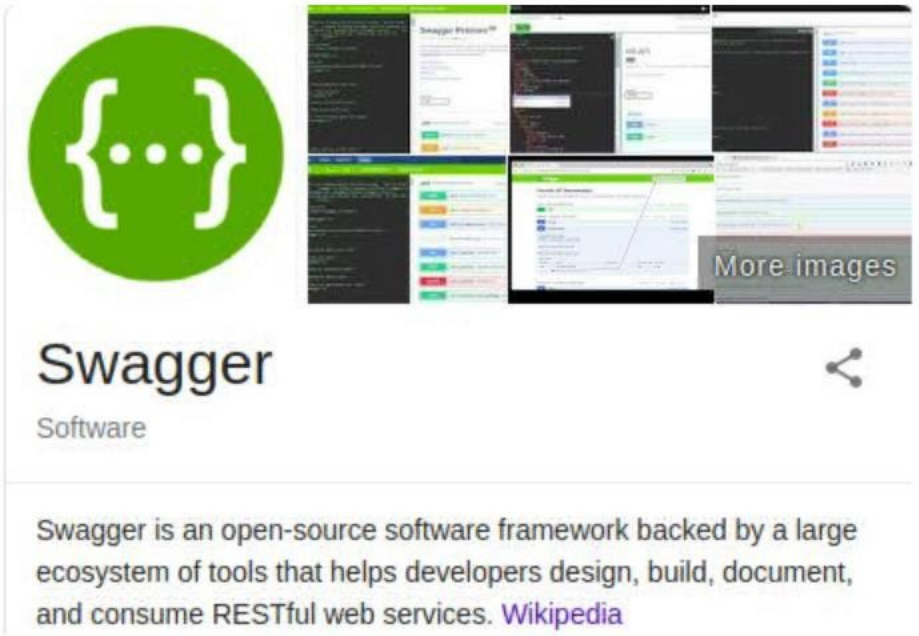
Step 11: swagger documentation

- What & Why Swagger : -
- Swagger Eco system : -
  - Swagger UI
  - Swagger Editor
  - Swagger Codegen
- Swagger Specification :-
  - Resource Listing - Can be JSON or YAML, Case sensitive fields
  - API Description with Details

Step 11: swagger documentation

```java
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2);
    }
}
```





http://localhost:8080/swagger-ui.html

```xml
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.4.0</version>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.4.0</version>
</dependency>
```

Swagger Doc Customization

```java
@Configuration
@EnableSwagger2
public class SwaggerConfig {

    public static final Contact DEFAULT_CONTACT = new Contact("rajeev gupta",
            "http://abc.com", "rgupta.mtech@gmail.com");

    public static final ApiInfo DEFAULT_API_INFO = new ApiInfo(
            "Awesome API Title", "Awesome API Description", "1.0", "urn:tos",
            DEFAULT_CONTACT, "Apache 2.0",
            "http://www.apache.org/licenses/LICENSE-2.0");

    private static final Set<String> DEFAULT_PRODUCES_AND_CONSUMES = new HashSet<String>(
            Arrays.asList("application/json", "application/xml"));

    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(DEFAULT_API_INFO)
                .produces(DEFAULT_PRODUCES_AND_CONSUMES)
                .consumes(DEFAULT_PRODUCES_AND_CONSUMES);
    }
}
```

Customization of swagger doc

```
@SwaggerDefinition(
        info = @Info(
                description = "Awesome Resources",
                version = "V12.0.12",
                title = "Awesome Resource API",
                contact = @Contact(
                   name = "rajeev gupta",
                   email = "rgupta.mtech@gmail.com",
                   url = "http://abc.com"
                ),
                license = @License(
                   name = "Apache 2.0",
                   url = "http://www.apache.org/licenses/LICENSE-2.0"
                )
        ),
        consumes = {"application/json", "application/xml"},
        produces = {"application/json", "application/xml"},
        schemes = {SwaggerDefinition.Scheme.HTTP, SwaggerDefinition.Scheme.HTTPS},
        externalDocs = @ExternalDocs(value = "Read This For Sure", url = "http://abc.com")
)
public interface UserApiDocumentationConfig {
}
```

```
@ApiModel(description="information about book")
public class Book {
    private int id;
    private String isbn;
    @ApiModelProperty(notes="should be min 4 char and max 30 char")
    @Size(min=5, max=30, message="title must be between 5 to 30 char")
    private String title;
```

Implementing Filtering for RESTful Service

Step 12: Implementing Filtering for RESTful Service

- What if dont want to expose some property as rest end point?

```
//@JsonIgnoreProperties(value={"field1","field2"})
public class SomeBean {
    private String field1;

    @JsonIgnore
    private String field2;
    private String field3;
```

```
@RestController
public class FilteringController {
    @RequestMapping("/filterdemo")
    public SomeBean getBean(){
        return new SomeBean("value 1", "value 2", "value 3");
    }

       @RequestMapping("/filterdemo-list")
    public List<SomeBean> getBeanList(){
        List<SomeBean> beans=Arrays.asList(new SomeBean("value 1", "value 2", "value 3"),
                new SomeBean("value 1", "value 2", "value 3"));
        return beans ;
    }
}
```

Implementing Enable cacheing

Step 13: Enable cacheing

Ehcache is an open source, standards-based **cache** that boosts performance, offloads your database, and simplifies scalability. It's the most widely-used Java-based **cache** because it's robust, proven, full-featured, and integrates with other popular libraries and frameworks.

EHCACHE

30.3 Declarative annotation-based caching

For caching declaration, the abstraction provides a set of Java annotations:

- `@Cacheable` triggers cache population
- `@CacheEvict` triggers cache eviction
- `@CachePut` updates the cache without interfering with the method execution
- `@Caching` regroups multiple cache operations to be applied on a method
- `@CacheConfig` shares some common cache-related settings at class-level

Step 13: Enable cacheing

```
import com.bookapp.model.service.BookService;
@EnableCaching
@SpringBootApplication
public class Bookapp2Application implements CommandLineRunner {

    @Autowired
    private BookService bookService;

    @Bean
    public CacheManager cacheManager(){
        ConcurrentMapCacheManager  cacheManager=new ConcurrentMapCacheManager("books");
        return cacheManager;
    }
```

```
@Service
@Transactional
public class BookServiceImpl implements BookService {

    @Autowired
    private BookDao bookDao;

    @Cacheable(value="books",key="#id" )
    @Override
    public Book getBookById(int id) {
        System.out.println("called.....");
        return bookDao.findById(id).orElseThrow(BookNotFoundException::new);
    }
```

```java
@Service
@Transactional
public class BookServiceImpl2 implements BookService {

    private BookRepository dao;

    @Cacheable(value="books",key="#id" )
    @Override
    public Optional<Book> findById(int id) {
        return dao.findById(id);
    }

    @CacheEvict(value="books", key="#id")
    @Override
    public void deleteBook(int id) {
        dao.deleteById(id);
    }

    @CachePut(value="books", key="#result.id")
    @Override
    public Book addBook(Book book) {
        return dao.save(book);
    }


@CachePut(value="books", key="#result.id")
@Override
public Book updateBook(int id, Book book) {
    Book bookToBeUpdate = dao.findById(id).orElseThrow(
            BookNotFoundException::new);
    bookToBeUpdate.setPrice(book.getPrice());

    return dao.save(bookToBeUpdate);
}

@CacheEvict(value="books", allEntries=true)
public void evictCache() {}
```

```java
@Configuration
@EnableScheduling
@ConditionalOnProperty(name = "scheduling.enabled", matchIfMissing = true)
class SchedulingConfiguration {
```

Implementing Enable Scheduled process

Step 14: Enable Scheduled process

- https://stackoverflow.com/questions/16407750/how-often-you-do-this-in-a-cron-0

```
@Component
public class ScheduledJob {
    private Logger logger = LoggerFactory.getLogger(this.getClass());
    @Autowired
    private BookService service;

    @Scheduled(cron = "0,30 * * * * *")
    public void cronJob() {
        logger.info("> cronJob");
        // Add scheduled logic here
        Collection<Book> books = service.getAllBooks();
        logger.info("There are {} books in the data store.", books.size());
        logger.info("< cronJob");
    }
    @Scheduled(initialDelay = 5000, fixedRate = 15000)
    public void fixedRateJob() {
        logger.info("> fixedRateJob");
        // Add scheduled logic here

        Collection<Book> greetings = service.getAllBooks();
        logger.info("There are {} books in the data store.", greetings.size());

        logger.info("< fixedRateJob");
    }
}
```

Enable Scheduled process

It will work once every hour, exactly at x:00.

Keep in mind the format of crontab is:

```
+----------------- minute (0 - 59)
|  +-------------- hour (0 - 23)
|  |  +---------- day of month (1 - 31)
|  |  |  +------- month (1 - 12)
|  |  |  |  +---- day of week (0 - 6) (Sunday=0 or 7)
|  |  |  |  |
*  *  *  *  *  command to be executed
```

so 0 in the first position means every minute 0, any hour, and day.

Versioning RESTful Services

Step 15: Versioning RESTful Services - Basic Approach with URIs

-
  - Versioning?

    -Media type versioning (aka "content negotiation" or " accept header")- Github

    -Custome headers versioning- Microsoft

    -URI versioning -Twitter

    -Parameter versioning -Amazon

-
  - Factors effect decision

    -url pollution

    - misuse of http headers

    - caching

    - can we execute teh request on browser?

    - api doc

    - No perfect sol

Spring REST versioning example

```java
public class Name {
    private String firstName;
    private String lastName;
```

```java
public class PersonV1 {
    private String name;
```

```java
public class PersonV2 {
    private Name name;
```

```java
@RestController
public class PersonVersioningController {
  @GetMapping("v1/person")
  public PersonV1 personV1() {
    return new PersonV1("rajeev gupta");
  }
  @GetMapping("v2/person")
  public PersonV2 personV2() {
    return new PersonV2(new Name("rajeev", "gupta"));
  }
  @GetMapping(value = "/person/param", params = "version=1")
  public PersonV1 paramV1() {
    return new PersonV1("rajeev gupta");
  }
  @GetMapping(value = "/person/param", params = "version=2")
  public PersonV2 paramV2() {
    return new PersonV2(new Name("rajeev", "gupta"));
  }
  @GetMapping(value = "/person/header", headers = "X-API-VERSION=1")
  public PersonV1 headerV1() {
    return new PersonV1("rajeev gupta");
  }
  @GetMapping(value = "/person/header", headers = "X-API-VERSION=2")
  public PersonV2 headerV2() {
    return new PersonV2(new Name("rajeev", "gupta"));
  }
  @GetMapping(value = "/person/produces", produces = "application/vnd.company.app-v1+json")
  public PersonV1 producesV1() {
    return new PersonV1("rajeev gupta");
  }
  @GetMapping(value = "/person/produces", produces = "application/vnd.company.app-v2+json")
```

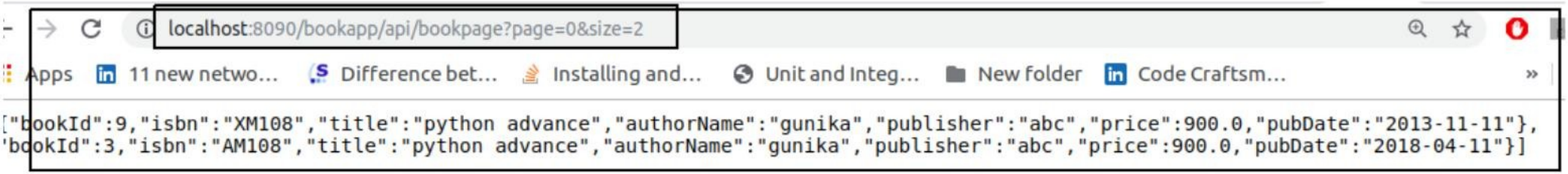
Pagable rest response

Step 16: pagable rest response

```
public List<Book> getAllBooksPagable(Pageable pageable);
```

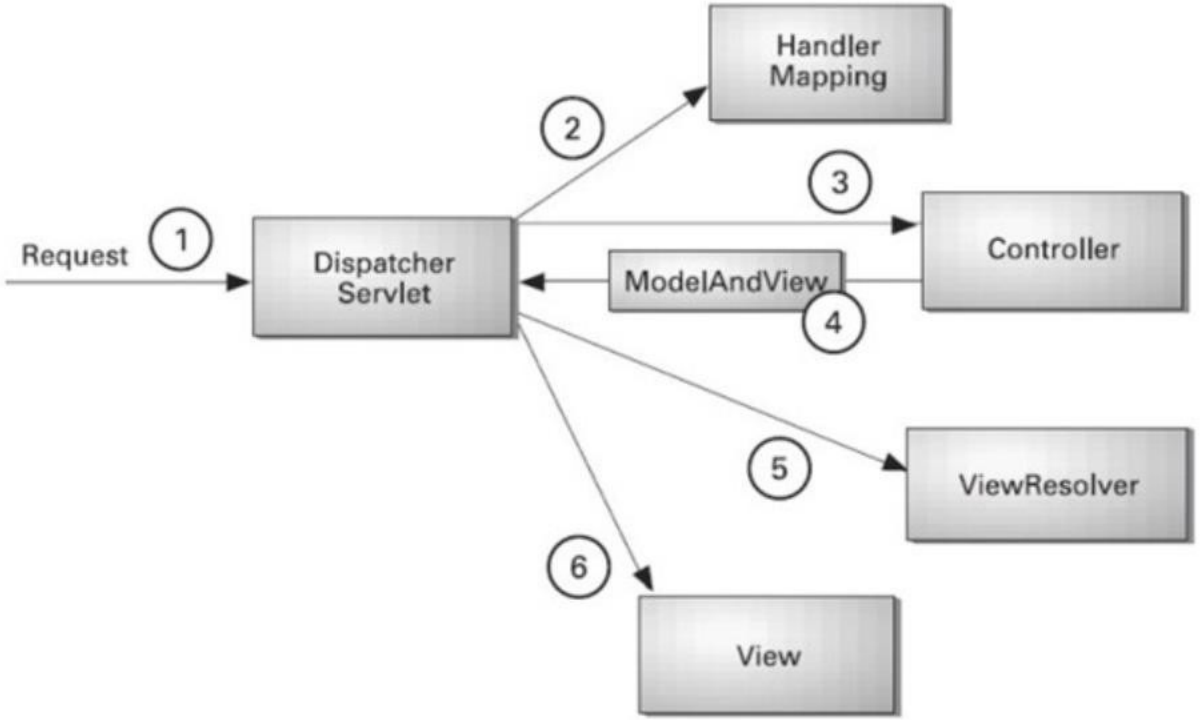
```
@Override
public List<Book> getAllBooksPagable(Pageable pageable) {
    return dao.findAll(pageable).getContent();
}
```

```
@GetMapping(path = "/bookpage", produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<List<Book>> getAllBooksPagination(Pageable pagable) {
    return new ResponseEntity<List<Book>>(
            bookService.getAllBooksPagable(pagable), HttpStatus.OK);
}
```



Spring MVC with JSP

Spring MVC


Handler
Mapping
2
3
Request
1
Dispatcher
Servlet
ModelAndView
Controller
4
5
ViewResolver
6
View

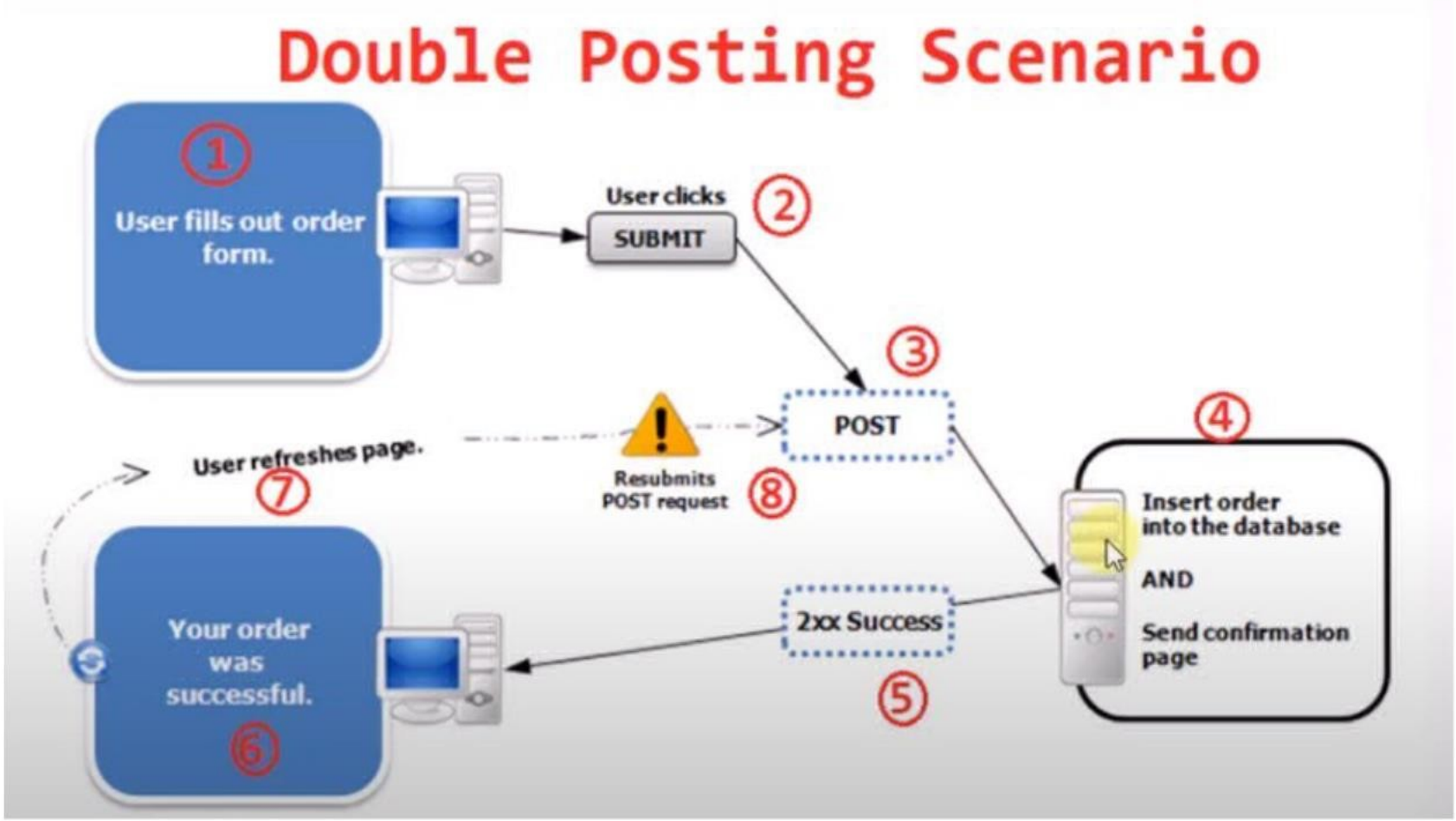
Step 18: spring boot jsp

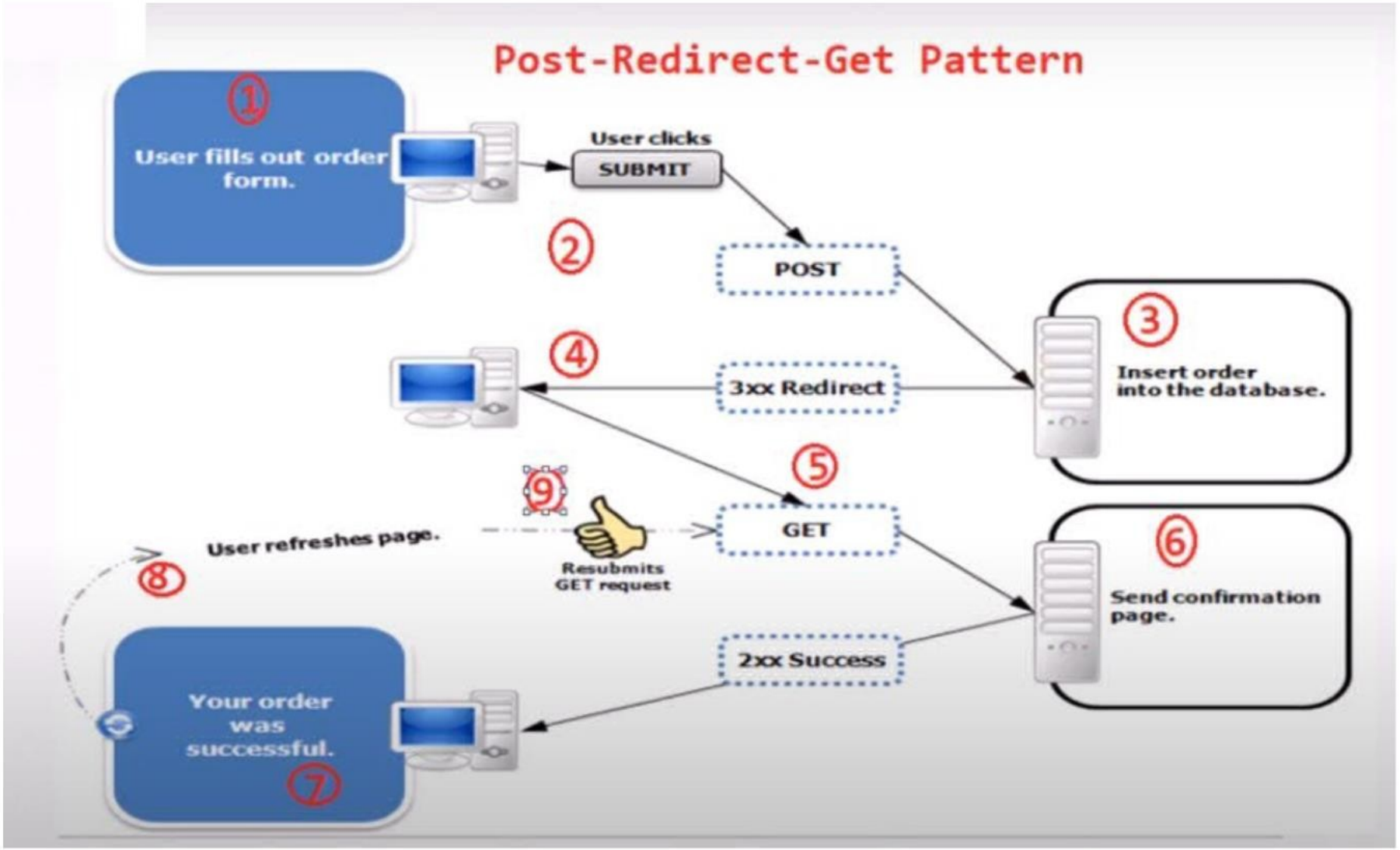
```
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```


```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>

<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
</dependency>
```

```
@Controller
public class BookController {

    private BookService bookService;

    @Autowired
    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @GetMapping(path="allbooks")
    public ModelAndView allbooks(ModelAndView mv){
        mv.addObject("books",bookService.getAllBooks());
        mv.setViewName("books");
        return mv;
    }
    @GetMapping(path="addbook")
    public String addBookGet(Model model){
        model.addAttribute("book", new Book());
        return "addbook";
    }
    @PostMapping(path="addbook")
    public String addBookPost(Book book){
        bookService.addBook(book);
        return "redirect:/allbooks";
    }
}
```

```
<table border="1">

        <thead>
            <tr>
                <th>id</th>
                <th>isbn</th>
                <th>title</th>
                <th>author</th>
                <th>price</th>
                <th>pub email</th>
                <th>publisher</th>
                <th>pub date</th>
            </tr>
        </thead>
        <tbody>
            <c:forEach items="${books}" var="book">
               <tr>
                <td>${book.id }</td>
                <td>${book.isbn }</td>
                <td>${book.title }</td>
                <td>${book.author }</td>
                <td>${book.price }</td>
                <td>${book.pubEmail }</td>
                <td>${book.publisher }</td>
                <td><fmt:formatDate value="${book.pubDate }" pattern="dd/MM/yyyy"/></td>
               </tr>
            </c:forEach>
        </tbody>
    </table>
    <a href="addbook">addbook</a>
```


Double Posting Scenario
1
User fills out order form.
User clicks
SUBMIT
2
3
POST
4
Insert order into the database
AND
Send confirmation page
2xx Success
5
Resubmits POST request
8
User refreshes page.
7
Your order was successful.
6


Post-Redirect-Get Pattern
1
User fills out order form.
User clicks
SUBMIT
2
POST
3
Insert order into the database.
4
3xx Redirect
5
GET
6
Send confirmation page.
2xx Success
7
Your order was successful.
8
User refreshes page.
9
Resubmits GET request

```java
@RequestMapping(value = "/createUser", method = RequestMethod.POST)
public String createUserAcc(@ModelAttribute("userModel") User user, RedirectAttributes attribues) {
    logger.info(" user form submitted :: " + user);

    // logic to insert record to DB

    //model.addAttribute("msg", "Account Created Successfully");
    attribues.addFlashAttribute("msg", "Account Created Successfully");

    return "redirect:/userAccCreationSuccess";
}

/**
 * This method is used to display sucess msg post registration
 *
 * @return
 */
@RequestMapping(value = "/userAccCreationSuccess", method = RequestMethod.GET)
public String userAccCreationSuccess(Model model) {
    logger.info(" userAccCreationSuccess()  method called ");
    model.addAttribute("userModel", new User());
    return "createUserAcc";
}
```

Spring MVC with Thymeleaf

Step 19: spring boot Thymeleaf

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
</head>
<body>
    <table>
        <tr>
            <th>bookId</th>
            <th >isbn</th>
            <th >title</th>
            <th>authorName</th>
            <th>publisher</th>
            <th>price</th>
            <th>pubDate</th>
        </tr>
        <tr th:each="book : ${books}">
          <td th:text="${book.bookId}">Id</td>
          <td th:text="${book.isbn}">isbn</td>
          <td th:text="${book.title}">title</td>
          <td th:text="${book.authorName}">authorname</td>
          <td th:text="${book.publisher}">publisher</td>
          <td th:text="${book.price}">price</td>
           <td th:text="${book.pubDate}">pubDate</td>
        </tr>
    </table>
    <a href="#" th:href="@{/addbook}">Add book</a>
</body>
</body>
</html>
```

```
<form action="#" th:action="@{/addbook}" th:object="${book}" method="POST">
  <table>

    <tr><td>ISBN</td>
        <td>
          <input type="text" th:field="*{isbn}" />
          <label th:if="${#fields.hasErrors('isbn')}"  th:class="'error'">Enter isbn</label>
        </td>
    </tr>
    <tr><td>Gender</td>
       <td>
           <input type="radio" th:field="*{gender}" value="Male"/><label>Male</label>
           <input type="radio" th:field="*{gender}" value="Female"/><label>Female</label>
           <label th:if="${#fields.hasErrors('gender')}"  th:class="'error'">Select Gender</label>
       </td>
    </tr>
    <tr><td>Married?</td>
       <td>
           <input type="checkbox" th:field="*{married}" />
       </td>
    </tr>
    <tr><td>Profile</td>
       <td>
           <select th:field="*{profile}">
               <option th:each="profile : ${allProfiles}"
               th:value="${profile}" th:text="${profile}">Profile</option>
           </select>
       </td>
    </tr>

       <tr><td colspan="2">
             <input type="submit" th:value="Submit"/>
             <input type="reset" th:value="Reset"/>
           </td>
       </tr>
   </table>
</form>
```

Blog and Comment application spring boot

Step 21: Blog and Comment application spring boot

```java
@Entity
@Table(name="blog_table")
public class Blog {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;
    private String content;


    @OneToMany(mappedBy="blog", cascade=CascadeType.ALL)
    @JsonIgnore
    List<Comment>comments=new ArrayList<Comment>();

    public void addComment(Comment comment){
        comments.add(comment);
        comment.setBlog(this);
    }

    public void removeComment(Comment comment){
        comments.remove(comment);
        comment.setBlog(null);
    }
```

```java
@Entity
@Table(name="comment_table")
public class Comment {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;
    private String comment;
    private LocalDateTime createdAt;

    @JoinColumn(name="bid_fk")
    @ManyToOne(fetch=FetchType.LAZY, optional=false)

    @JsonIgnore
    private Blog blog;
```

Blog and Comment application: Repo layer

```java
public interface BlogRepo extends JpaRepository<Blog, Long>{
}
```

```java
public interface CommentRepo extends JpaRepository<Comment, Long>{
    List<Comment> findByBlogId(Long blogId);
    Optional<Comment> findByIdAndBlogId(Long id, Long postId);
}
```

```java
Blog blog=new Blog("spring5", "amit", "spring 5 is lastest form spring.io");
Blog blog2=new Blog("java8", "raj", "java 8 is morden java");

Comment comment1=new Comment(blog, "good blog on spring 5");
Comment comment2=new Comment(blog, "spring 5 rock");
Comment comment3=new Comment(blog, "i need basic into to spring first");

Comment comment4=new Comment(blog2, "good blog on spring 8");
Comment comment5=new Comment(blog2, "need more details");
Comment comment6=new Comment(blog2, "i need basic of collection");

blogRepo.save(blog);
commentRepo.save(comment1);
commentRepo.save(comment2);
commentRepo.save(comment3);

blogRepo.save(blog2);
commentRepo.save(comment4);
commentRepo.save(comment5);
commentRepo.save(comment6);
```

Blog and comment application:BlogController

```java
@RestController
@RequestMapping(path = "api")
public class BlogController {

    @Autowired
    private BlogRepo blogRepo;

    @GetMapping(path = "blog")
    public ResponseEntity<List<Blog>> getAllBlogs() {
        return ResponseEntity.ok().body(blogRepo.findAll());
    }

    @PostMapping(path = "blog")
    public ResponseEntity<Blog> postBlogs(@RequestBody Blog blog) {
        blogRepo.save(blog);
        return ResponseEntity.status(HttpStatus.CREATED).body(blog);
    }

    // update blog
    @PutMapping(path = "blog/{id}")
    public ResponseEntity<Blog> upddateBlog(@PathVariable(name = "id") Long id,
            @RequestBody Blog blogReq) {

        return blogRepo.findById(id).map(blog-> {
            blog.setContent(blogReq.getContent());
            blog.setTitle(blogReq.getTitle());
            return ResponseEntity.ok().body(blogRepo.save(blog));
        }).orElseThrow(() -> new ResourceNotFoundException("blog with id " + id
                + " not found"));
    }
```

Blog and comment application: BlogController

```
    //delete blog
    @DeleteMapping(path = "blog/{id}")
    public ResponseEntity<?>deleteBlog(@PathVariable(name = "id") Long id){
        return blogRepo.findById(id).map(blog->{
            blogRepo.delete(blog);
            return ResponseEntity.noContent().build();
        }).orElseThrow(() -> new ResourceNotFoundException("blog with id " + id
                + " not found"));
    }
    //get blog by id
    @GetMapping(path = "blog/{id}")
    public ResponseEntity<Blog>getByIdBlog(@PathVariable(name = "id") Long id){
        return blogRepo.findById(id).map(blog->{
            return ResponseEntity.ok().body(blog);
        }).orElseThrow(() -> new ResourceNotFoundException("blog with id " + id
                + " not found"));
    }
}
```

Blog and comment application:

CommentController

```java
@RestController
@RequestMapping(path = "api")
public class CommentController {

    @Autowired
    private BlogRepo blogRepo;

    @Autowired
    private CommentRepo commentRepo;

    // get all comments for given blog
    @GetMapping(path = "/blog/{blogId}/comment")
    public ResponseEntity<List<Comment>> getAllCommentByPostId(
            @PathVariable(name = "blogId") Long blogId) {
        List<Comment> comments = commentRepo.findByBlogId(blogId);
        return ResponseEntity.ok().body(comments);

    }
```

```java
    @PostMapping("/blog/{blogId}/comment")
    public ResponseEntity<Comment> createComment(
            @PathVariable(value = "blogId") Long blogId,
            @RequestBody Comment comment) {

        Blog blog = blogRepo.findById(blogId).orElseThrow(
                () -> new ResourceNotFoundException("PostId " + blogId
                        + " not found"));

        blog.addComment(comment);
        blogRepo.save(blog);
        commentRepo.save(comment);

        return ResponseEntity.ok().body(comment);

    }
```

Blog and comment application:CommentController

```java
@PutMapping("/blog/{blogId}/comment/{commentId}")
public ResponseEntity<Comment> updateComment(@PathVariable(value = "blogId") Long blogId,
        @PathVariable(value = "commentId") Long commentId,
        @RequestBody Comment commentRequest) {

    if (!blogRepo.existsById(blogId)) {
        throw new ResourceNotFoundException("blogId " + blogId
                + " not found");
    }

    Comment comment = commentRepo.findById(commentId).orElseThrow(
            () -> new ResourceNotFoundException("CommentId " + commentId
                    + "not found"));

    comment.setComment(commentRequest.getComment());
    commentRepo.save(comment);
    return ResponseEntity.ok().body(comment);
}

@DeleteMapping("/blog/{blogId}/comment/{commentId}")
public ResponseEntity<?> deleteComment(@PathVariable (value = "blogId") Long blogId,
                           @PathVariable (value = "commentId") Long commentId) {
    return commentRepo.findByIdAndBlogId(commentId, blogId).map(comment -> {
        commentRepo.delete(comment);
        return ResponseEntity.noContent().build();
    }).orElseThrow(() -> new ResourceNotFoundException
            ("Comment not found with id " + commentId + " and blogId " + blogId));
}
```

Blog and comment application:ExceptionHandling

```java
@ResponseStatus(HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    private static final long serialVersionUID = 1L;
    public ResourceNotFoundException(String message) {
        super(message);
    }
}
```

```java
public class ErrorDetails {
    private String message;
    private LocalDateTime timeStamp;
    private String detail;
    private String contactTo;
```

```java
@ControllerAdvice
@RestController
public class ExceptionHandlerRestController {
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ErrorDetails> handleBookNotFoundEx(

            ResourceNotFoundException ex, WebRequest request) {
        ErrorDetails details = new ErrorDetails("Resource not found",
                LocalDateTime.now(), request.getDescription(false),
                "https://www.rgupta.com/support");

        return new ResponseEntity<ErrorDetails>(details, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorDetails> handleOtherEx(

            Exception ex, WebRequest request) {
        ErrorDetails details = new ErrorDetails("some server side error",
                LocalDateTime.now(), request.getDescription(false),
                "https://www.rgupta.com/support");

        return new ResponseEntity<ErrorDetails>(details, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```