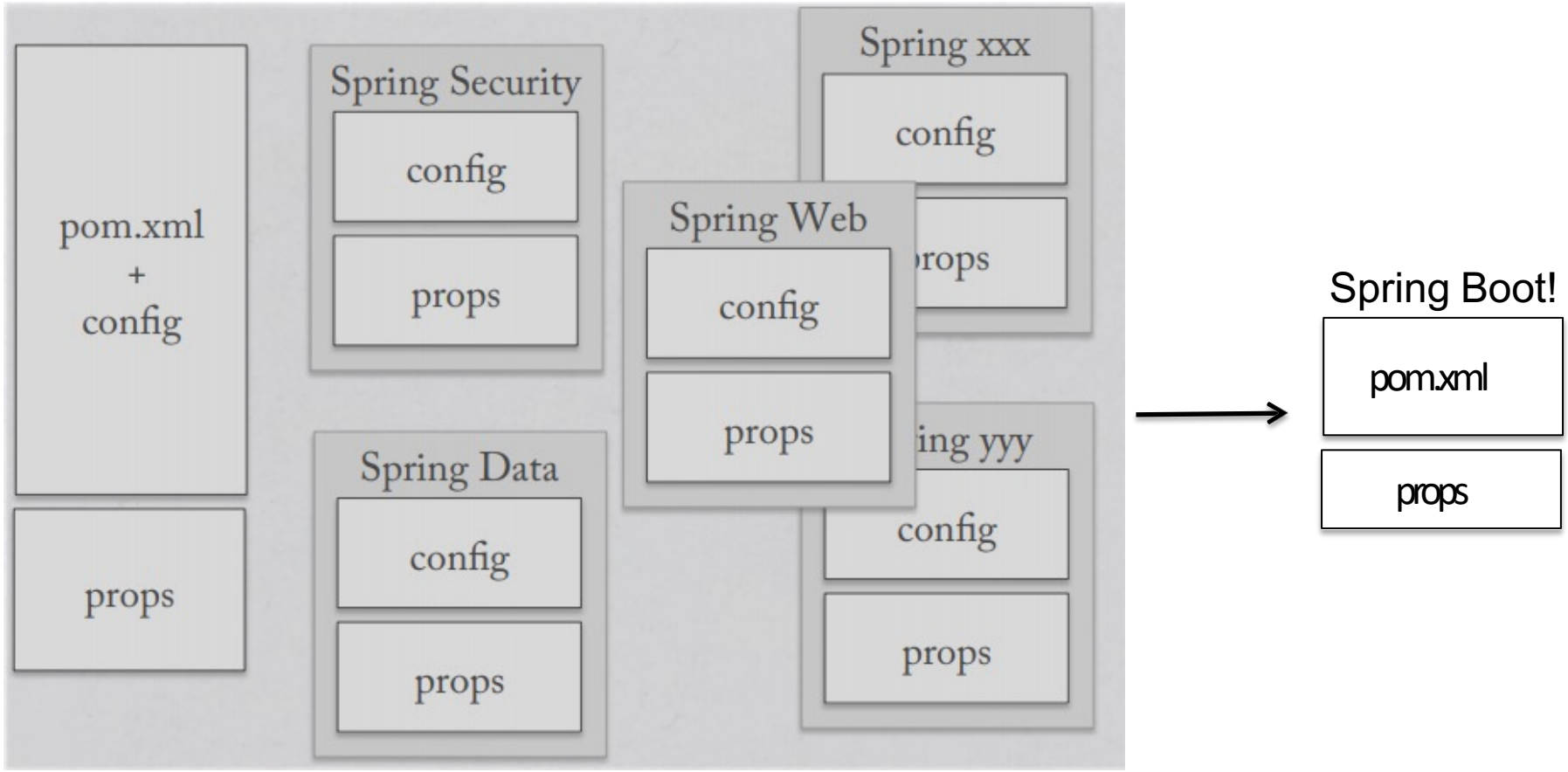
Introduction to Spring Boot & REST

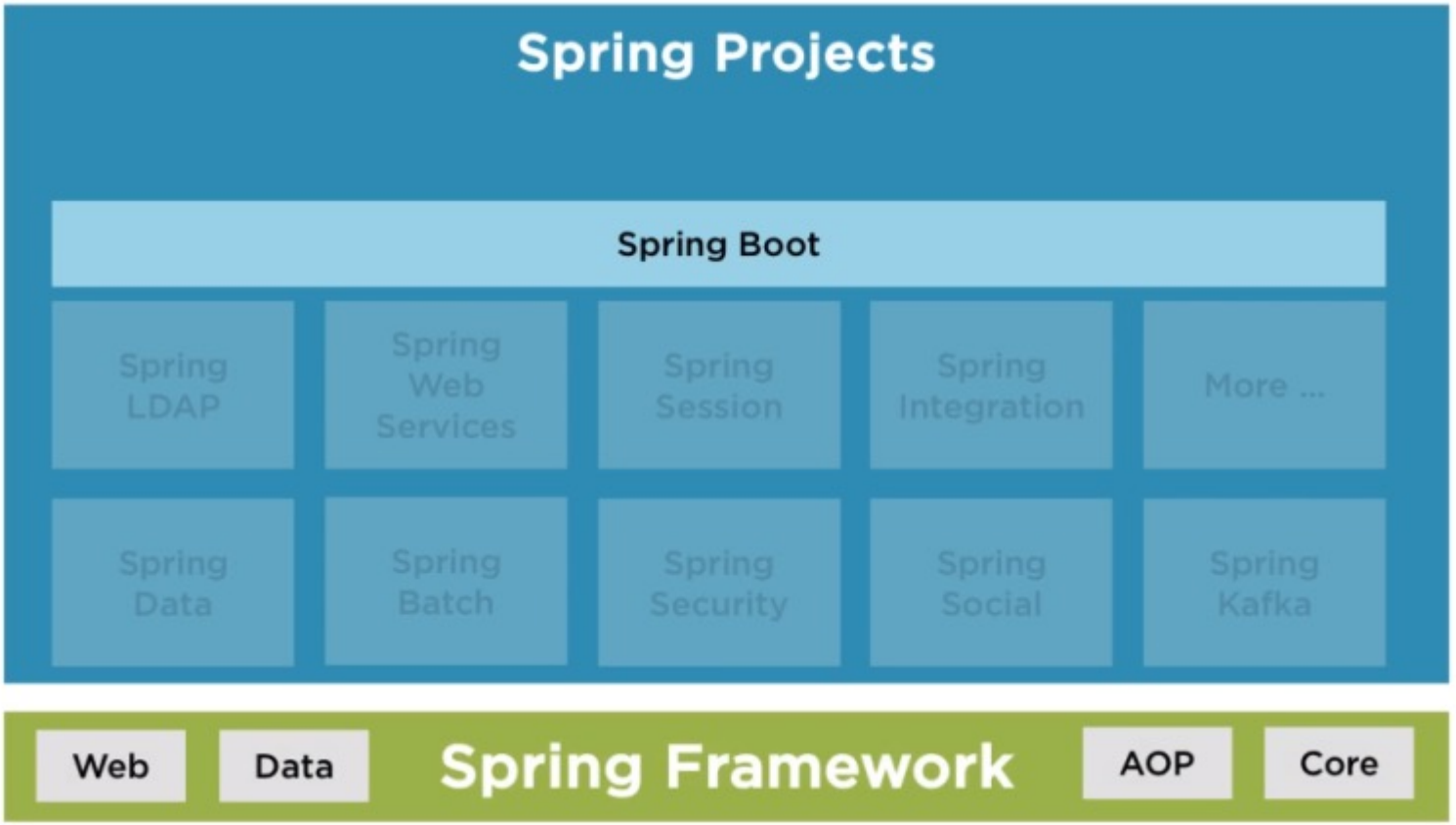
Home of All the Laws of Nature

Spring Boot Emerging



What is Spring Boot?

- Spring Boot is a project built on the top of the Spring framework. It provides a simpler and faster way to set up, configure, and run both simple and web-based applications.

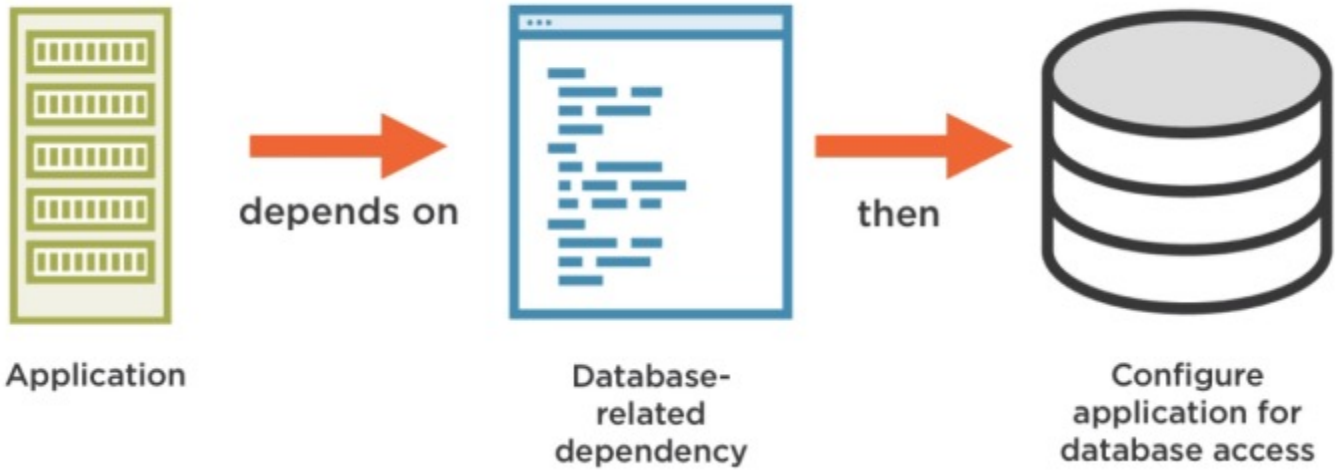


Features

- **Auto-configuration:** It sets up your application based on the surrounding environment, as well as hints what the developers provide.
- **Standalone:** Literally, it's completely standalone. Hence, you don't need to deploy your application to a web server or any special environment. Your only task is to click on the button or give out the run command, and it will start.
- **Opinionated:** This means that the framework chooses how to things for itself. In the most cases, the developers use the same most popular libraries. All that the Spring Boot does is that it loads and configures them in the most standard way. Hence, the developers don't need to spend a lot of time to configure up the same thing over and over again.

Auto Configuration
@EnableAutoConfiguration

- Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added.
- Guess and configure beans that you are likely to need.

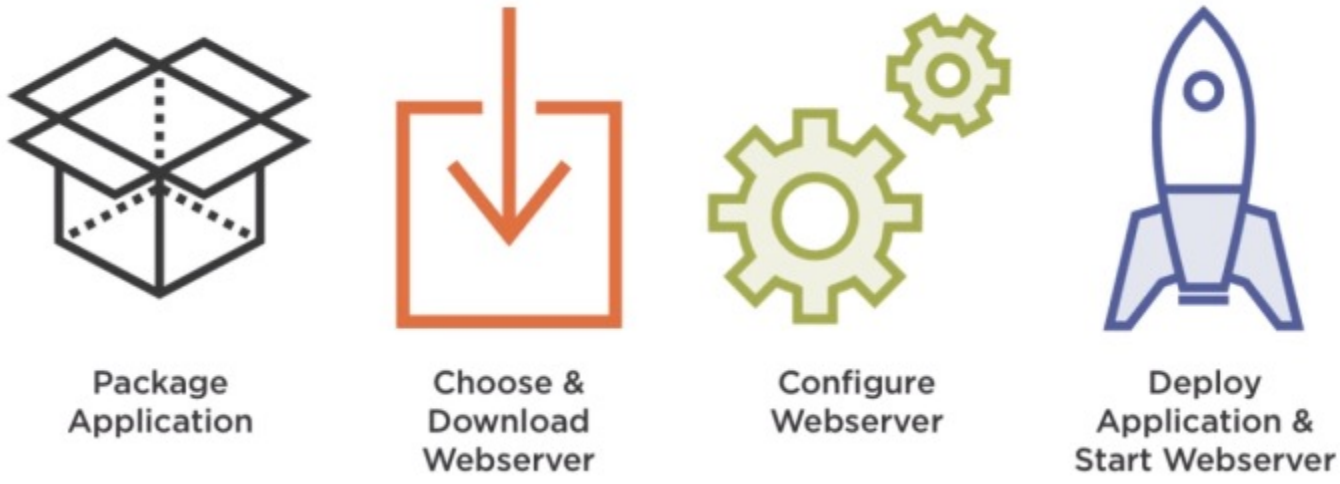


Features

- **Auto-configuration:** It sets up your application based on the surrounding environment, as well as hints what the developers provide.
- **Standalone:** Literally, it's completely standalone. Hence, you don't need to deploy your application to a web server or any special environment. Your only task is to click on the button or give out the run command, and it will start.
- **Opinionated:** This means that the framework chooses how to things for itself. In the most cases, the developers use the same most popular libraries. All that the Spring Boot does is that it loads and configures them in the most standard way. Hence, the developers don't need to spend a lot of time to configure up the same thing over and over again.

The Process of Starting a Java-Based Web Application

- **Package** your application.
- Choose **which type of web servers** you want to use and download it.
- **Configure** that specific web server.
- **Organize the deployment process** and start your web server.



Spring Boot Way

- Install JDK
- Package your application
- Run it with some simple command like
  java -jar my-application.jar
- *java -jar demo-0.0.1-SNAPSHOT.jar --server.port=8083*

Embedded Server

- With Embedded server, you can run a web application as a normal Java application.
  - Embedded server implies that our deployable unit contains the binaries for the server (example, tomcat.jar).

- spring-boot-starter-web: includes a dependency on starter tomcat.

Features

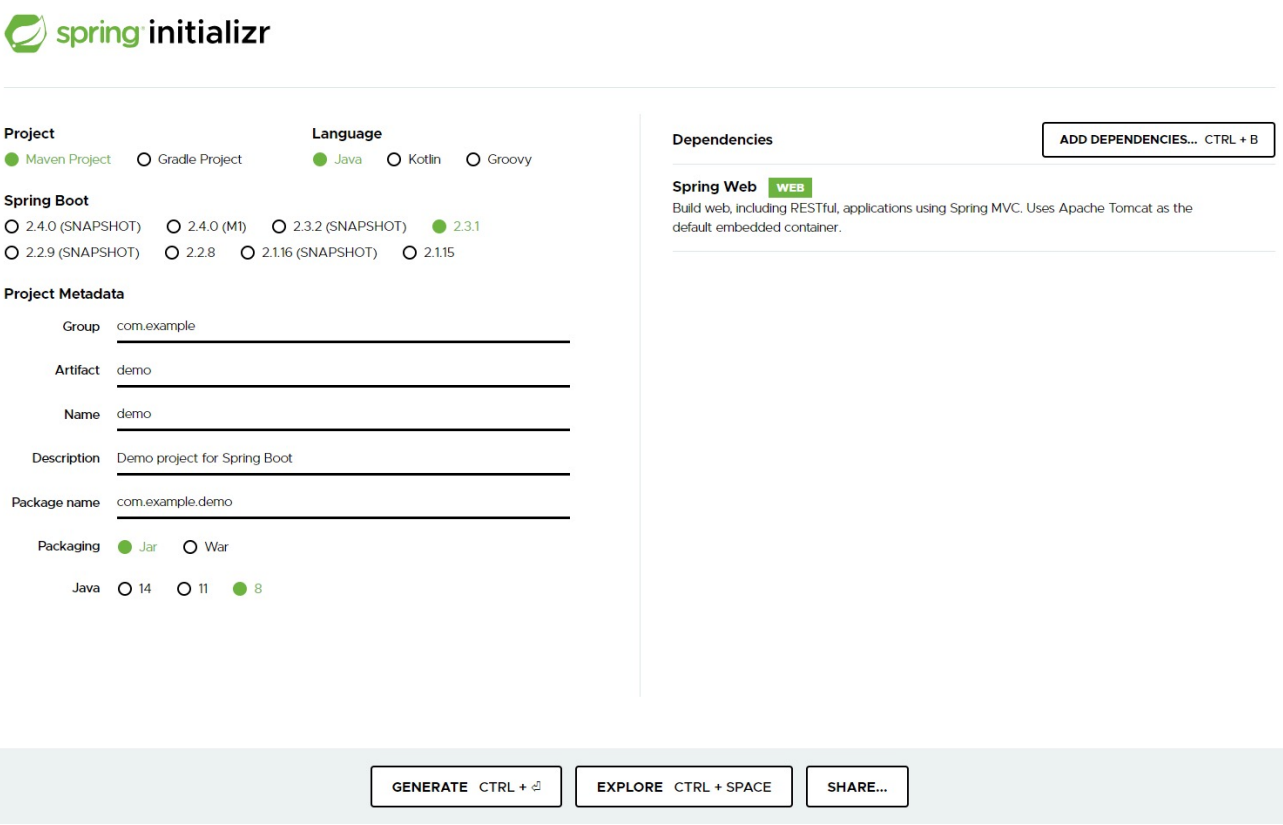
- **Auto-configuration:** It sets up your application based on the surrounding environment, as well as hints what the developers provide.
- **Standalone:** Literally, it's completely standalone. Hence, you don't need to deploy your application to a web server or any special environment. Your only task is to click on the button or give out the run command, and it will start.
- Opinionated: This means that the framework chooses how to configure things for itself. In the most cases, the developers use the same most popular libraries. All that the Spring Boot does is that it loads and configures them in the most standard way. Hence, the developers don't need to spend a lot of time to configure up the same thing over and over again.

pom.xml

- Spring Boot provides a number of "Starters" that let you add jars to your classpath.
- spring-boot-starter-parent
  - inherits dependency management from spring-boot-dependencies
  - Override property e.g. java.version
  - default configuration for plugins e.g.maven-jar-plugin
- Other starters
  - spring-boot-starter-web
  - spring-boot-starter-thymeleaf
  - spring-boot-starter-test

Spring MVC With Spring Boot

1. Create a Spring Starter Project → http://start.spring.io
2. Specify Project Name, artifact Id, etc
3. Select dependency
4. Generate



Main Application Class

```
@SpringBootApplication
public class SpringBootWebJspApplication {

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

`@SpringBootApplication` annotation is equivalent to using

- `@Configuration` : enable Java-based configuration,
- `@EnableAutoConfiguration` and
- `@ComponentScan`: enable component scanning

with their default attributes

Controller & JSP

```
@Controller
public class WelcomeController {
  @GetMapping("/")
  public String index() {
    return "index";
  }
}
```

- index.jsp

```
<body>
  Welcome...
</body>
```

Spring Boot JSPs

- Spring boot has limitations in JSP support
- It works if you use war packaging. A jar will not work because of a hard coded file pattern in Tomcat.
- Add dependency to be able to compile JSP – MUST

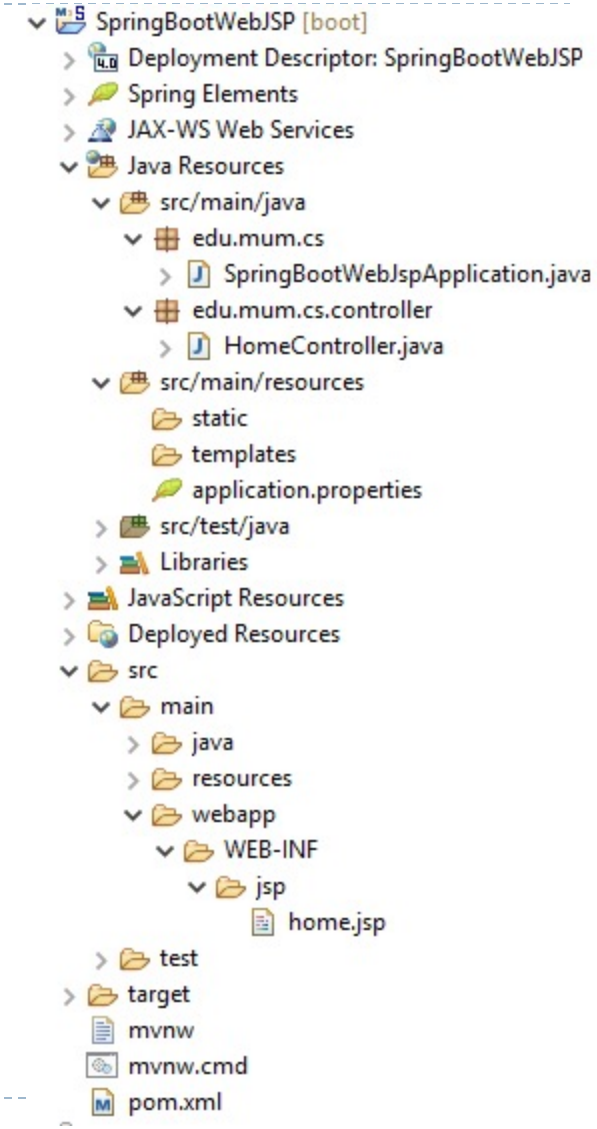
```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope>
</dependency>
```

- Config view resolver in application.properties

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

How to Run

- Run As -> Maven Install – It generates jar/war file under target folder
- Open cmd, java –jar SpringBootWebJSP-0.0.1-SNAPSHOT.war
- MANIFEST.MF
  - Main-Class: org.springframework.boot.loader.WarLauncher
  - Start-Class: edu.mum.cs.SpringBootWebJspApplication



Developer Tools

- Automatic restart
  - Code changes require restart of application
  - Two classloaders in Spring boot
- LiveReload
  - html, javascript, templates, images, stylesheets, …

Main Point

- Spring Boot simplifies dependency management AND application configuration by adhering to a set of rules that apply to all applications.

- Adhering to the fundamental Laws of Nature allows life to be lived simply and easily.

REST Web Services

- REST = **RE**presentational **S**tate **T**ransfer
- REST is an architectural style consisting of a coordinated set of architectural constraints
- First described in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine
- RESTful is typically used to refer to web services implementing a REST architecture
- Alternative to other distributed-computing specifications such as SOAP
- Simple HTTP client/server mechanism to exchange data
- Everything – the UNIVERSE is available through a URI
- Utilizes HTTP: GET/POST/PUT/DELETE operations

- REST architectural constraints

Uniform Interface:

- It is a key constraint that differentiate between a REST API and Non-REST API. 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).
  There are four guidelines principle of Uniform Interface are:
- **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: Usually user get a user id when user request for a list of users and then use that id to delete or modify that particular user.
- **Self-descriptive Messages:** Each message includes enough information to describe how to process the message so that server can easily analyses the request.
- **Hypermedia as the Engine of Application State (HATEOAS):** It need to include links for each response so that client can discover other resources easily.

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 whether as a part of query params, headers or URI.
- Statelessness enables greater availability since the server does not have to maintain, update or communicate that session state.
- There is a drawback when the client need to send too much data to the server so it reduces the scope of network optimization and requires more bandwidth.

Cacheable:

- Every response should include whether the response is cacheable or not and for how much duration responses can be cached at the client side.
- Client will return the data from its cache for any subsequent request and there would be no need to send the request again to the server.
- A well-managed caching partially or completely eliminates some client–server interactions, further improving availability and performance. But sometime there are chances that user may receive stale data.

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, and server is someone who holds the resources and are not concerned with the user interface or user state.
- They can evolve independently. Client doesn't need to know anything about business logic and server doesn't need to know anything about frontend UI.

Layered system:

- An application architecture needs to be composed of multiple layers.
- Each layer doesn't know any thing about any layer other than that of immediate layer and there can be lot of intermediate servers between client and the end server.
- Intermediary servers may improve system availability by enabling load-balancing and by providing shared caches.

- **(Optional) Code on demand:** It is an optional feature. According to this, servers can also provide executable code to the client. The examples of code on demand may include the compiled components such as Java applets and client-side scripts such as JavaScript.