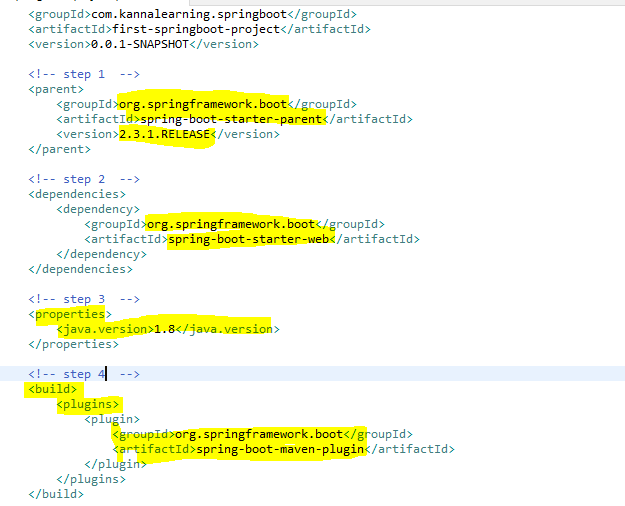
**Spring Boot**

Maven helps with dependency management very well. In general if we are including dependencies in pom file we include release version but if we have spring boot starter parent, it will take care of all the versions of dependencies. This is the spring boot magic. Now if we want to develop a web application in general we have to add lot of dependencies in the pom file but if we use spring boot starter web it will bring in all the jars needed for web application.

In order to create a Spring boot application there are five basic steps

1. Add a Spring boot starter parent in pom file
2. Add a Spring boot starter web in pom file
3. Specify java version in pom file
4. Add Spring boot plugin in pom file
5. Create spring boot application launcher.



Step 5



@SpringBootApplication is the one that launches the application. When a class has this annotation, then spring boot will component scan automatically all the components with in this package. But if we want a component scan from other packages also then @ComponentScan has to be used.

**Spring-boot-starter-parent:**

It defines versions of the dependencies, java version, and few plugins. But if we want to update the version of any of these dependencies, then we can use properties tag in pom file.

**Spring-boot-starter-web:**

What version of spring-boot-starter-web is used?

This comes from spring-boot-starter-parent.

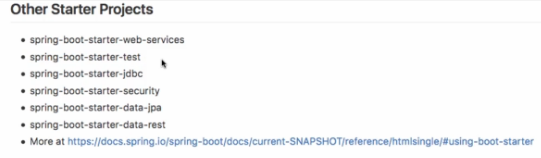
What dependencies does spring-boot-starter-web bring in?

It brings in embedded tomcat, Jackson bind, spring mvc, logging, hibernate-validator etc.

What auto-configuration does it do?

One of the auto-configuration is DispatcherServlet. It also takes auto configures of error handling. We can check this in the logs by creating an entry in application.properties file under resources.

logging.level.org.springframework: DEBUG



**Spring Boot VS Spring**

**Spring**

* Spring is just a dependency injection framework. Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.
* First half of the 2000 decade! EJBs
* EJBs were NOT easy to develop.
* Write a lot of xml and plumbing code to get EJBs running
* Impossible to Unit Test
* Alternative - Writing simple JDBC Code involved a lot of plumbing
* Spring framework started with aim of making Java EE development simpler.
* Goals
* Make applications testable. i.e. easier to write unit tests
* Reduce plumbing code of JDBC and JMS
* Simple architecture. Minus EJB.
* Integrates well with other popular frameworks.

**Applications with Spring Framework**

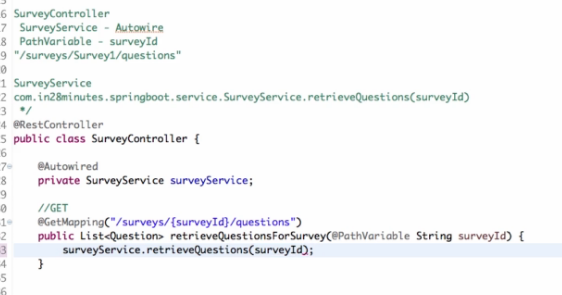
* Over the next few years, a number of applications were developed with Spring Framework
* Testable but
* Lot of configuration (XML and Java)
* Developing Spring Based application need configuration of a lot of beans!
* Integration with other frameworks need configuration as well!
* In the last few years, focus is moving from monolith applications to microservices. We need to be able to start project quickly. Minimum or Zero start up time
* Framework Setup
* Deployment - Configurability
* Logging, Transaction Management
* Monitoring
* Web Server Configuration

**Spring Boot**

* Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.
* We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss.
* Example Problem Statements
* You want to add Hibernate to your project. You dont worry about configuring a data source and a session factory. I will do if for you!
* Goals
  + Provide quick start for projects with Spring.
  + Be opinionated but provide options.
  + Provide a range of non-functional features that are common to large classes of projects (e.g. embedded servers, security, metrics, health checks, externalized configuration).

**What Spring Boot is NOT?**

* It’s not an app or a web server
* Does not implement any specific framework - for example, JPA or JMS
* Does not generate code



Here two important annotations are introduced.

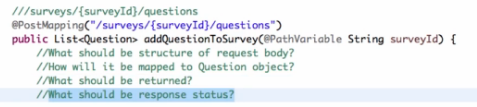
@GetMapping is used in place of @RequestMapping(“/”,RequestMethod.GET). Here we have given the url mapping beside the GetMapping.

@PathVariable is used to pass variables from the rest url path to the rest method.

**Message Converters:**

When using the above rest service, the response is automatically getting converted to JSON. This is being done by Jackson data binding. This is both object to json and json to object.

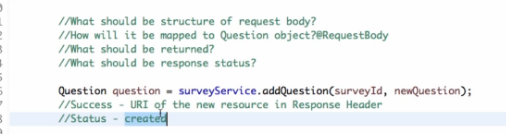
**@PostMapping**



///////////////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////////////



///////////////////////////////////////////////////////////////////////////////////////////////////////////////

