**Spring BOOT:**

Boot is bootstrap! It makes easy to create stand along, production-grade Spring based Applications that you can just run!

Create spring-based applications!

Stand-alone – runs on its own

**Spring?**

Application framework

That lets you build enterprise java applications.

The underlying foundation of all the java application have many similarities, and spring handles all those things.

Programming and configuration Model

Let you build classes, where you can use annotation.

Infrastructure Support – connecting to the database!

**Problems with spring:**

Huge Framework

Multiple setup steps

Multiple configuration steps

Multiple build and deploy steps

**Can we abstract these steps? 🡪**

Spring Boot is opinionated.

Convention over configuration concept!

Stand alone, Production Ready

**Setup: JAVA 8 SDK**

**Java –version**

**Echo $JAVA\_HOME**

**Manage Dependencies:**

Get the jars and add it to the class path – not doing this

We will be using MAVEN

LETS YOU DECLARE ALL THE DEPENDENCIES IN A SINGLE FILE

POM.XML – LIST OF ALL YOUR DEPENDECIES. MAVEN IS A DEPENDECY MANAGEMENT TOOL!

Create the starter project – maven does that for us!

1. Create a Maven project
2. Group id: io.javabrains.springbootquickstart
3. Articraft id: course-api

Step 1: Add the parent section in pom.xml –our project is the child project of the parent. Inheriting

Step 2: Maven serves as the dependency management tool! Add dependencies.

Step 3: This imports all the jars

Go to src/main/java and create a java class

@SpringBootApplication – tells springs boot that it is the starting point

SpringApplication.*run*(CourseApiApp.**class**, args);

Run is a static process

**Starting Spring Boot**

Sets us default configuration

Starts Spring application context

Performs class path scan

Starts Tomcat Server

Adding a controller:

A java class

Marked with annotations

Has info about, what URL access trigger it

What method to run when accessed?

@RestController: annotation from Spring MVC, anytime you are building a rest controller

You can have methods, where you can map to the URL

@RequestMapping: when a request is made, send it to the given URL 🡪 maps only to the GET method by default

Bill of Materials:

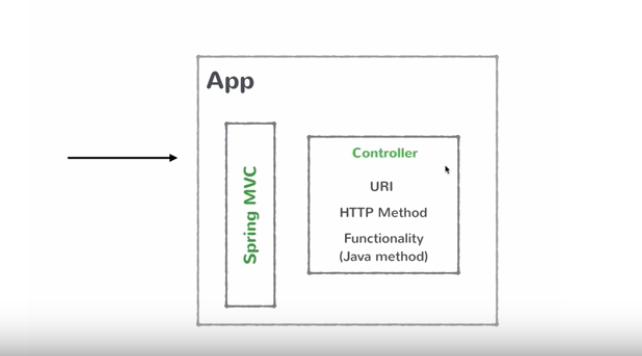
Embedded Tom Cat Server:

Convenience

Servlet container config is now application config

Useful for micro services architecture

**Spring MVC: The View Tier**



**Resources of your RestAPI:**

Topic, Course, Lesson

A topic can have multiple courses and a course can have multiple lessons

**GET /topics 🡪 get all topics**

**GET /topics/id 🡪 gets the topic**

**POST /topics 🡪 create new topic**

**PUT /topics/id 🡪 updates the topic**

**DELETE /topics/id 🡪 deletes the topic**

**@Service - marks as Spring business service**

**Business Service are singleton!**

**@Autowired: dependency injection**

**@PathVariable : variable in the path**

***Method = RewustMethod.POST***

***@RequestBody 🡪 request the whole body and do the assigned work***

**Booting Spring Boot**

Starting a Spring Boot App

Spring Initializr : available at http://start.spring.io

Spring BOOT CLI, STS IDE

Configuration

**Customizing Spring Boot:**

application.properties

server.port = 1111, changes your server port! You can access now at localhost:1111

docs.spring.io has all the properties that you can tweak.

**Spring Data JPA: The Data Tier**

Java Persistence API: SPECIFICATION THAT LETS YOU DO ORM (Object Relational MAPPING)

Let you map your entity classes into table

JPA is a way to use ORM and give it to the framework

Spring Data JPA: separate project, which makes things even easier