**Spring boot :**

It is spring framework that has

1. Auto configuration based on Jar files added to classpath

@EnableAutoConfiguration

1. Auto Injection dependency resolution

We d not need to use @Autowire we can use construction

1. Embeded Tomcat
2. Actuator for managing End points
3. CLI command line interface to run application independent from maven

**Spring boot component** :

1. Boot Starter

It is just JAR files that boot uses to do auto-configuration

1. Boot Actuator

Spring boot uses Actuator to manage endpoints.

It is a features to help us monitor and manage application in production

We can choose to monitor your application by using HTTP endpoints or Auditing, health, and metrics gathering can also be automatically applied to your application.

1. Boot CLI (command line interface)
2. Boot initilzr

Spring Boot Initilizr is a tool to bootstrap Applications very easily.

Spring Boot Initilizr comes in the following forms:

1. Spring Boot Initilizr With Web Interface
2. Spring Boot Initilizr With IDEs/IDE Plugins
3. [Spring Boot Initilizr With Spring Boot CLI](https://www.journaldev.com/8609/spring-boot-initilizr-with-spring-boot-cli)
4. [Spring Boot Initilizr With ThirdParty Tools](https://www.journaldev.com/8650/spring-boot-initilizr-with-thirdparty-tools)

**Setup spring boot Security**

We need to create a SecurityConfiguration class that extends WebSecurityConfigurereAdapter

* 1. There are 2 annotation that we are used

@EnableWebSecurity

@EnabeGlobalMethodWebSecurity

* 1. We need 3 dependecy injection

AuthenticationManagerBuilder

TokenProvider

CorseFilter

UserService

WebSecurityConfigurereAdapter has 2 method to implemene

1. Protected void Configure(HttpSecurity h)
   1. Configure authentication and level of access per URL in details

AddFilterBefore

exceptionHandling

aauthenticationEntryPoint

h.antMatchers("/api/register").permitAll()

.antMatchers("/api/activate").permitAll(

1. Protected void Configure(WebSecurity w)

Matches the acceptable webURL

Web.ignoring().antMatchers("/app/\*\*/\*.{js,html}")

.antMatchers("/i18n/\*\*")

.antMatchers("/content/\*\*")

.antMatchers("/swagger-ui/index.html")

Unit Test Spring boot :

1. DAO layer
   1. memory DB such as H2 database
   2. @DataJpaTest that provide TestEntityManager
2. Service layer
   1. We mock persistent layer we do not need it
   2. We use @mockBean on Repository dependency injection

@Before

public void setUp() {

   Employee alex = new Employee("alex");

   Mockito.when(employeeRepository.findByName(alex.getName()))

     .thenReturn(alex);

}

@Test

public void whenValidName\_thenEmployeeShouldBeFound() {

    String name = "alex";

    Employee found = employeeService.getEmployeeByName(name);

     assertThat(found.getName())

      .isEqualTo(name);

 }

Controller Layer test

We mock Service layer using @MockBean

@RunWith(SpringRunner.class)

@WebMvcTest(EmployeeRestController.class)

public class EmployeeRestControllerTest {

    @Autowired

    private MockMvc mvc;

    @MockBean

    private EmployeeService service;

    // write test cases here

}