infogain

Spring Security

Objectives

www.infogain.com

- Introduction to Spring Boot
- Introduction to Spring Rest
- Introduction to Spring Security
- Use default Configuration
- Customize Default Spring Security Configuring
- In memory and Data Base Authentication

Prerequisite for Application Development

- **Prior Knowledge for this Session**
  - Spring MVC
  - Spring Rest
  - Spring Boot

What is Spring Boot?

- It is not a framework. it is a tool that allows you to create spring based application within no time.
- it helps you to build ,package and deploy the spring application with minimal or absolutely no configurations.
- It provides a set of Starter Pom's which one can use to add required dependencies and also facilitate auto configuration.

Spring Boot Working

How Does Spring Boot Work?

Java

Main method entry point

Spring Application

Spring context

Spring environment

Initializers

Embedded Server

Default is Tomcat

Auto configured

Spring Boot Working…

```
public static void main( … )


@SpringBootApplication



@Configuration
@EnableAutoConfiguration
@ComponentScan


SpringApplication.run( … );
```

- Starts Java and then the application

- A convenience annotation that wraps commonly used annotations with Spring Boot

- Spring configuration on startup
- Auto configures frameworks
- Scans project for Spring components

- Starts Spring, creates spring context, applies annotations and sets up container

SpringApplication

- The **SpringApplication** class provides a convenient way to bootstrap a Spring application that will be started from a main() method.
- It Create an appropriate ApplicationContext instance (depending on your classpath)

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

Spring Boot Working…

- **@SpringBootApplication: Adds all of the following:**
  - **@Configuration :** Tags the class which as a source of bean definitions for the application context.
  - **@EnableAutoConfiguration :** Tells Spring Boot to start adding beans based on class path settings, other beans, and various property settings.
  - **@EnableWebMvc** : For a Spring MVC app, but Spring Boot adds it automatically when it sees spring-webmvc on the class path. This flags the application as a web application and activates key behaviors such as setting up a DispatcherServlet.
  - **@ComponentScan :** Tells Spring to look for other components, configurations, and services in the specified package allowing it to find the controllers.

Starting Spring BOOT

- **SpringApplication.run(App.class,args);**
  - This is a magical line which takes two argument one is the class name annoted with @SpringBootAppliction and another is command line argument.

- **What @SpringBootAppliction does?**
  - Sets up the default configuration
  - Starts Spring application context
  - Performs class path scan
  - Starts the Tomcat server

Web Services

- **Spring Rest web Services**
  - Spring supports annotation based MVC framework for creating RESTful web services.
  - The key difference between a traditional Spring MVC controller and the RESTful web service controller is the way the HTTP response body is created.
  - In traditional MVC controller relies on the View technology, the RESTful web service controller simply returns the object and the object data is written directly to the HTTP response as JSON/XML.

Web Services(contd..)

www.infogain.com

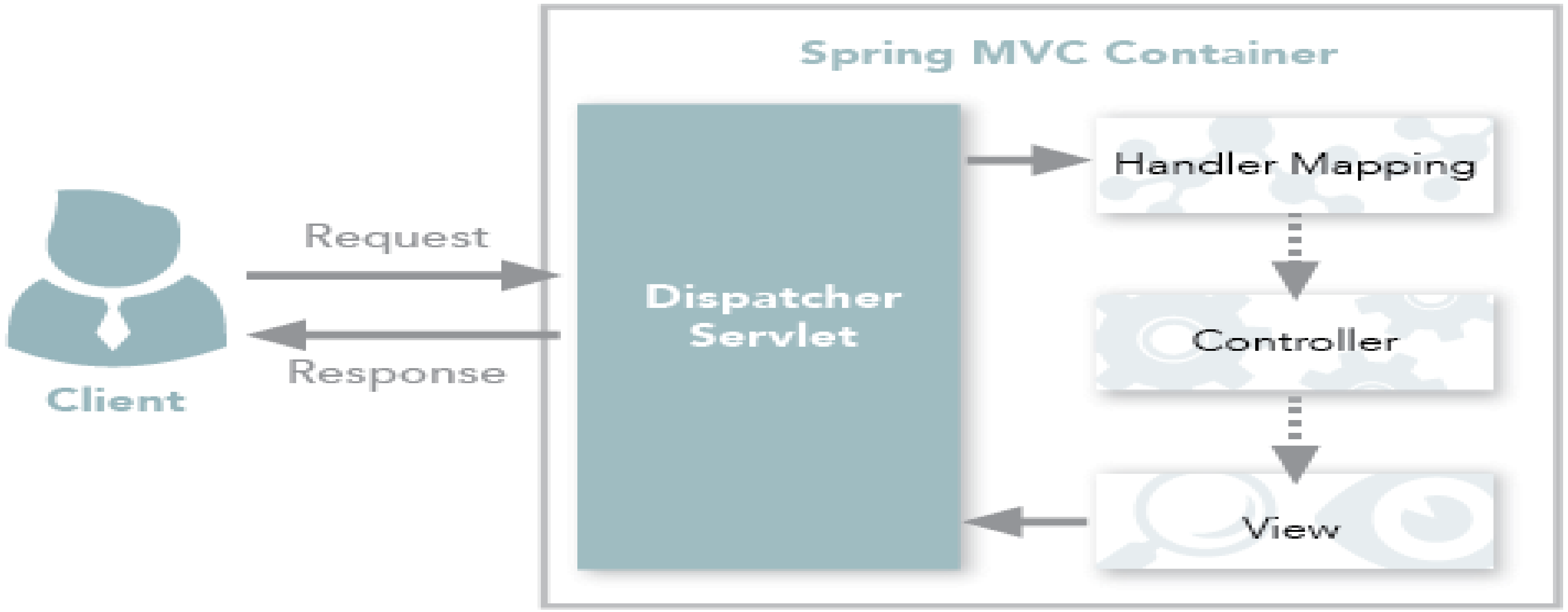

Figure 1: Spring MVC traditional workflow

Web Services(contd..)

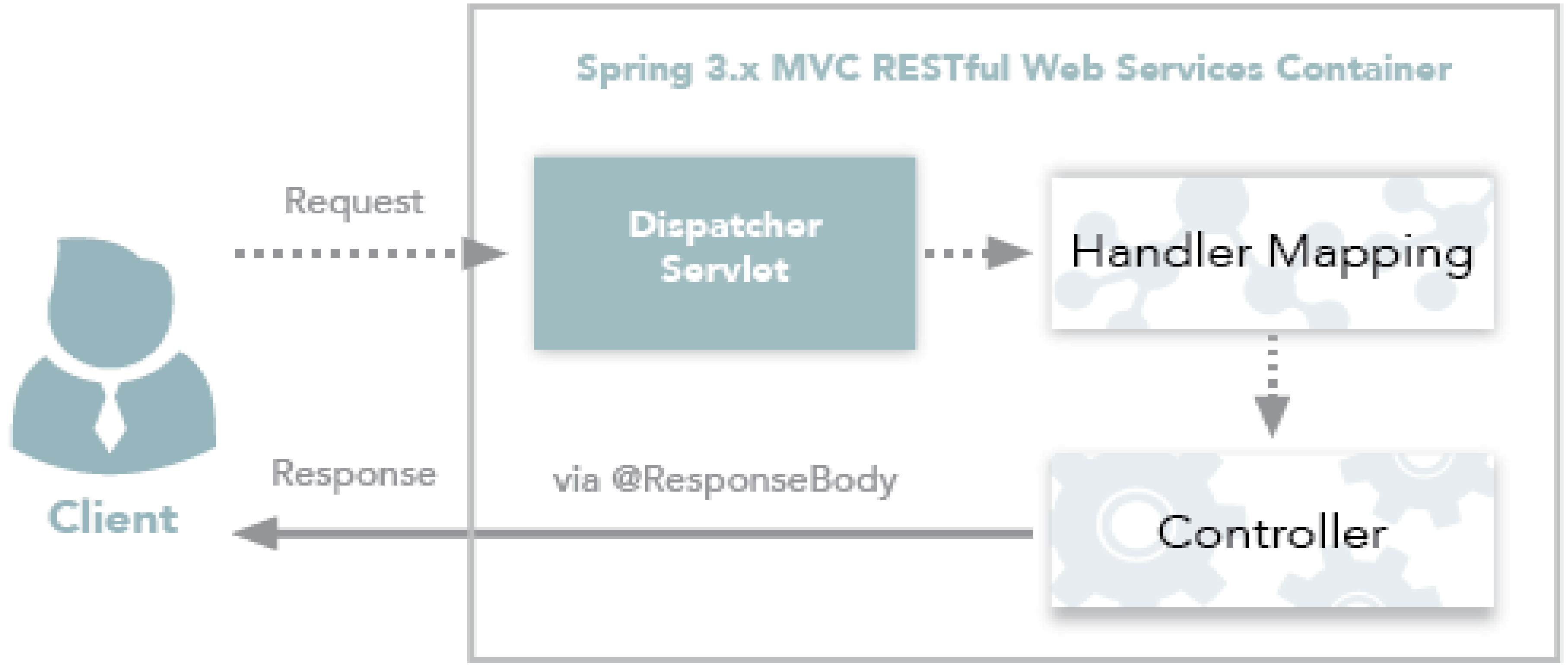

Figure 2: Spring MVC RESTful web services workflow

Web Services(contd..)

www.infogain.com

- **@RestController**
  - It Marks a class as Spring MVC Rest Controller. It is a specialized version of the controller annotation which includes the @Controller and @ResponseBody annotations and as a result, simplifies the controller implementation
  - **@RestController= Controller + ResponseBody**

Example

@RestController

public class MyController

{ ….. }

Web Services(contd..)

- **@RequestMapping**
  - Can be used at class level as well as method level
  - Annotate the method that should handle certain HTTP request.

```
@RestController
public class  MyController {
@RequestMapping(value="/empJson", method=RequestMethod.GET,produces="application/json")
public Employee getJSON()  {
    Employee emp = new Employee();
    return emp;
        }
        }
```

Spring Security

- **What is API Security**
  - It is a powerful framework that focuses on providing authentication and access control to secure Spring-based Java web application.
  - This framework targets two major areas of application they are authentication and authorization.
  - Authentication is the process of knowing and identifying the user that wants to access a resource.
  - Authorization is the process to allow authority to perform actions in the application.

- **Spring Project Modules**
  - In Spring Security 3.0, the Security module is divided into separate jar files. Based on their functionalities, so, the developer can integrate according to their requirement.

- **The following are some jar files that are included into Spring Security module.**
  - spring-security-core.jar
  - spring-security-web.jar
  - spring-security-config.jar
  - spring-security-ldap.jar

Include Spring Security in your Project

- **To include spring security in your project, include below dependency:**
- **spring-security-core.jar**
  - core jar file is required for every application that wants to use Spring Security. This jar file includes **core access-control** and **core authentication classes** and interfaces.
- **spring-security-web**
  - This jar is useful for Spring Security web authentication and **URL-based access control**. It includes filters and **web-security infrastructure.**
  - All the classes and interfaces are located into the **org.springframework.security.web** package.
- **Spring-security-config**
  - This jar file is required for Spring Security configuration using **XML and Java** both. It includes Java configuration code and security namespace parsing code. All the classes and interfaces are stored in **org.springframework.security.config** package.

HTTP Basic Authentication

- **HTTP Basic Authentication**
  - Basic authentification is a standard **HTTP** header with the user and **password** encoded in **base64**
  - The userName and password is encoded in the format **username:password.**
  - This is one of the technique to protect the REST resources because it does not require cookies. session identifiers or any login pages.
  - In case of basic authentication, the username and password is only encoded with Base64, but not encrypted or hashed in any way.

Configuring Spring Web Security

- **Environment Setup**
  - 1. JDK 8
  - 2. Spring Boot
  - 3. STS and Maven Dependencies

- **Maven Dependencies**
  - **spring-boot-starter**-parent: provides useful Maven defaults.
  - **spring-boot-starter-web:** includes all the dependencies required to create a web app. This will avoid lining up different spring common project versions.
  - **spring-boot-starter-tomcat:** enable an embedded Apache Tomcat 7 instance, by default.
  - **spring-boot-starter-security:** take care of all the required dependencies related to spring security.

Configuring Spring Web Security(Contd..)

- **Define Spring Security Configuration File**

WebSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
        Override   Required  Web Security Configuration Method


         }
}
```

- **The Configuration class is annotated with**
  - This configuration creates a Servlet Filter known as the **springSecurityFilterChain** which is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, etc) within your application.
  - **@EnableWebSecurity** to enable Spring web security support.
  - The **WebSecurityConfigurerAdapter** to override spring features with our custom requirements.

HTTP Security

- To enable HTTP Security in Spring, we need to extend the **WebSecurityConfigurerAdapter** and Override the default configuration in the **configure(HttpSecurity http)** method:

```
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
      .anyRequest().authenticated()
      .and().httpBasic();
}
```

- The above default configuration makes sure any request to the application is authenticated with HTTP basic authentication.

Spring Security Authentication Details Storage

- In memory
- database
- **Spring Security using In memory Authentication**
  - Store the user details inside Security Config file

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

     @Override
  public void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication().withUser("abc").password("abc").roles("USER");

  }
}
```

Authorization with Roles

www.infogain.com

- **Let’s now configure some simple authorization on each URL using roles:**

```
protected void configure(HttpSecurity http) throws Exception {
   http.authorizeRequests()
     .antMatchers("/", "/home").access("hasRole('USER')")
     .antMatchers("/admin/**").hasRole("ADMIN");
     }
```

Spring Security using Database Authentication

- **Spring Security using Database Authentication**
  - We need to store user and user roles inside tables
  - Than register a bean which type of **DriverManagerDataSource** inside Spring context
- **DriverManagerDataSource**
  - Used to contain the information about the database such as driver class name, connection URL, username and password
- **Create a datasource for the database connection**
  - driverClassName = oracle.jdbc.driver.OracleDriver
  - url = jdbc:oracle:thin:@localhost:1521:xe
  - username = system
  - password = password

Declare DriverManagerDataSource Bean

- **Java Based Configuration**

```
@Configuration
public class DataSouceConfig {
        @Bean(name = "dataSource")
        public DriverManagerDataSource dataSource() {
           DriverManagerDataSource driverManagerDataSource = new DriverManagerDataSource();
           driverManagerDataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
           driverManagerDataSource.setUrl("jdbc:oracle:thin:@127.0.0.1:1521:XE");
           driverManagerDataSource.setUsername("dbuser");
           driverManagerDataSource.setPassword("dbpassword");
           return driverManagerDataSource;
        }
}
```

Autowire DriverManagerDataSource Bean

www.infogain.com

- **Autowire DriverManagerDataSource Bean inside web security config File**

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
        @Autowired
         DataSource dataSource;
                        @Override
         public void configure(AuthenticationManagerBuilder auth) throws Exception {
           auth.jdbcAuthentication().dataSource(dataSource)
          .usersByUsernameQuery(
           "select username,password, enabled from logins where username=?")
          .authoritiesByUsernameQuery(
           "select l.username, r.role from logins l, roles r where l.login_id = r.login_id and l.username =?");
         }
}
```

infogain

Thank You

infogain

www.infogain.com

Infogain Corporation, HQ
485 Alberto Way Los Gatos,
CA 95032 USA
Phone: 408-355-6000
Fax: 408-355-7000

Pune
7th Floor, Bhalerao Towers, CTS No.1669 - 1670, Behind Hotel Pride,
Shivaji Nagar, Pune - 411005
Phone : +91-20-66236700

Infogain Irvine
41 Corporate Park,
Suite 390 Irvine, CA 2606 USA
Phone: 949-223-5100
Fax: 949-223-5110

Infogain Austin
Stratum Executive Center Building D
11044 Research Boulevard Suite 200
Austin, Texas 78759

Noida
A-16, Sector 60, Noida Gautam Budh agar,
201301 (U.P.) India
Phone: +91-120-2445144
Fax: +91-120-2580406

Dubai
P O Box 500588 Office No.105,
Building No. 4, Dubai Outsource Zone,
Dubai, United Arab Emirates
Tel: +971-4-458-7336