Spring 5, Hibernate 5, Security 5

spring

HIBERNATE

Rajeev Gupta

FreeLancer Corporate Java JEE/ Spring Trainer

New Delhi Area, India

Add profile section | More...

freelance

Institution of Electronics and Telecommunication...

See contact info

See connections (500+)

1. Expert trainer for Java 8, GOF Design patterns, OOAD, JEE 7,Spring 5, Hibernate 5 ,Spring boot, microservice, netflix oss, Spring cloud, angularjs, Spring MVC, Spring Data, Spring Security, EJB 3, JPA 2, JMS 2, Struts 1/2, Web service

2. Helping technology organizations by training their fresh and senior engineers in key technologies and processes.

3. Taught graduate and post graduate academic courses to students of professional degrees.

I am open for advance java training /consultancy/ content development/ guest lectures/online training for corporate / institutions on freelance basis

Contact :

==========

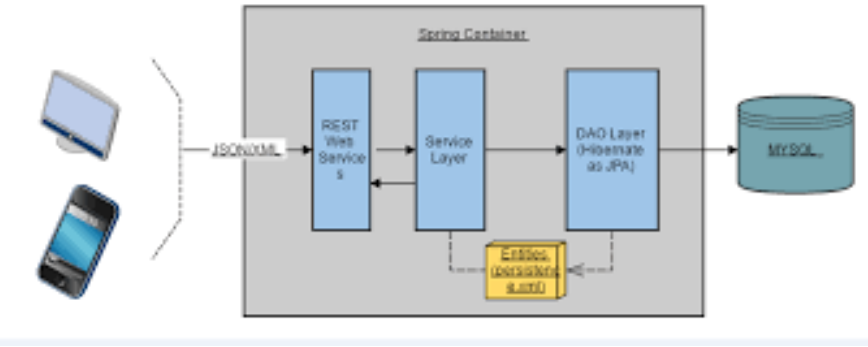
rgupta.mtech@gmail.com

Clients:

==========

Gemalto, Noida
Cyient Ltd, Noida
Fidelity Investment Ltd
Blackrock, Gurgaon
Mahindra comviva
Steria
Bank Of America
incedo gurgaon
MakeMyTrip
Capgemini India
HCL Technologies
CenturyLink
Deloitte consulting
Nucleus Software
Ericsson Gurgaon
Avaya gurgaon
Kronos Noida
NEC Technologies
A.T. Kearney
ust global
TCS
North Shore Technologies Noida
IBM
Sapient
Accenture
Incedo
Genpact
Indian Air force
Indian railways
Vidya Knowledge Park

- bookappspringhib
  - Deployment Descriptor: bookappspringhib
  - JAX-WS Web Services
  - Java Resources
    - src
      - com.app.model.controller
        - BookController.java
      - com.app.model.dao
        - Book.java
        - BookDao.java
        - BookDaoImpl.java
      - com.app.model.service
        - BookService.java
        - BookServiceImpl.java
    - Libraries
  - JavaScript Resources
  - build
  - WebContent
    - META-INF
    - WEB-INF
      - lib
      - views
        - fc-configuration.xml



book isbn	book title	book author	book price	
1338	c is c@@22	gjhgjh	777.0	delete update

Add new Book

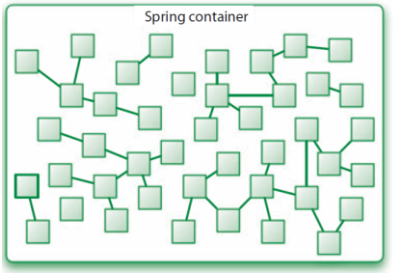
Quick look to Spring Framework
You alread know about it!

वसंत Spring

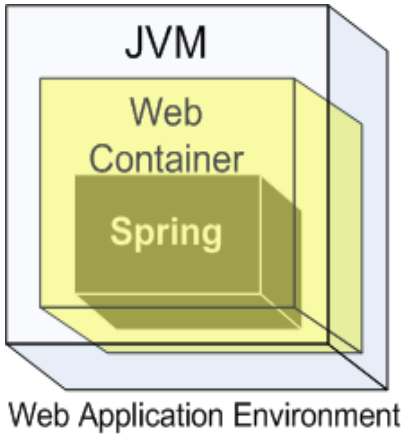
What is spring framework?

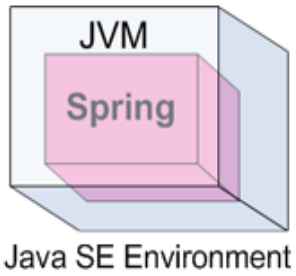
Spring framework is a contrainer that manage life cycle of bean.

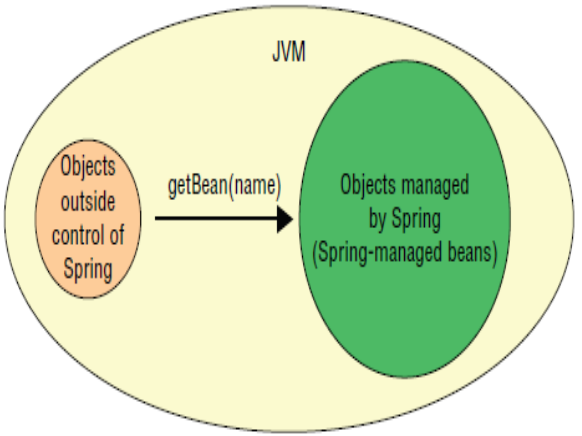
It Does 2 primary Jobs:
1. bean wiring
2. bean weaving



A bean is an object that is instantiated, assembled, and managed by a Spring IoC container.



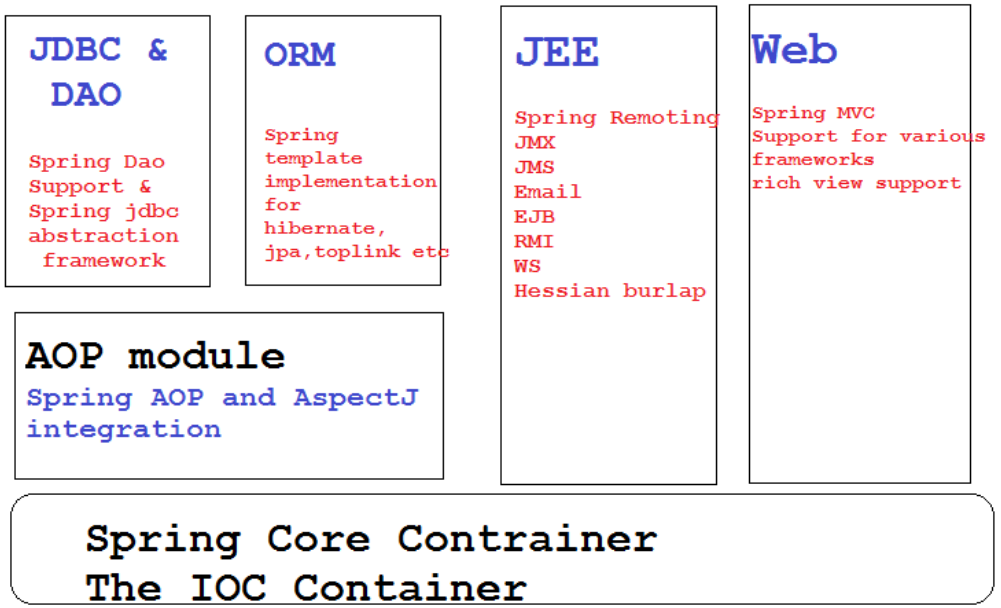




Why Spring framework? Where it fits?

Spring Framework is focused on simplifying enterprise Java development through

- dependency injection
- aspect-oriented programming
- boiler-plate code reduction using template design pattern



Spring framework design patterns

Design Patterns used in Spring Framework

Proxy Design Pattern

Singleton Design Pattern

Factory design pattern

Template Design Pattern

Model View Controller Pattern

Front Controller Pattern

View Helper Pattern

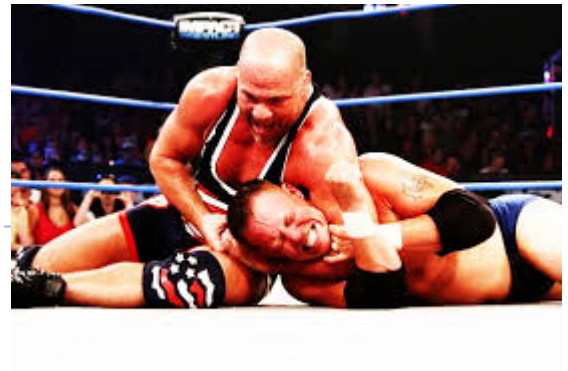
Dependency injection or inversion of control

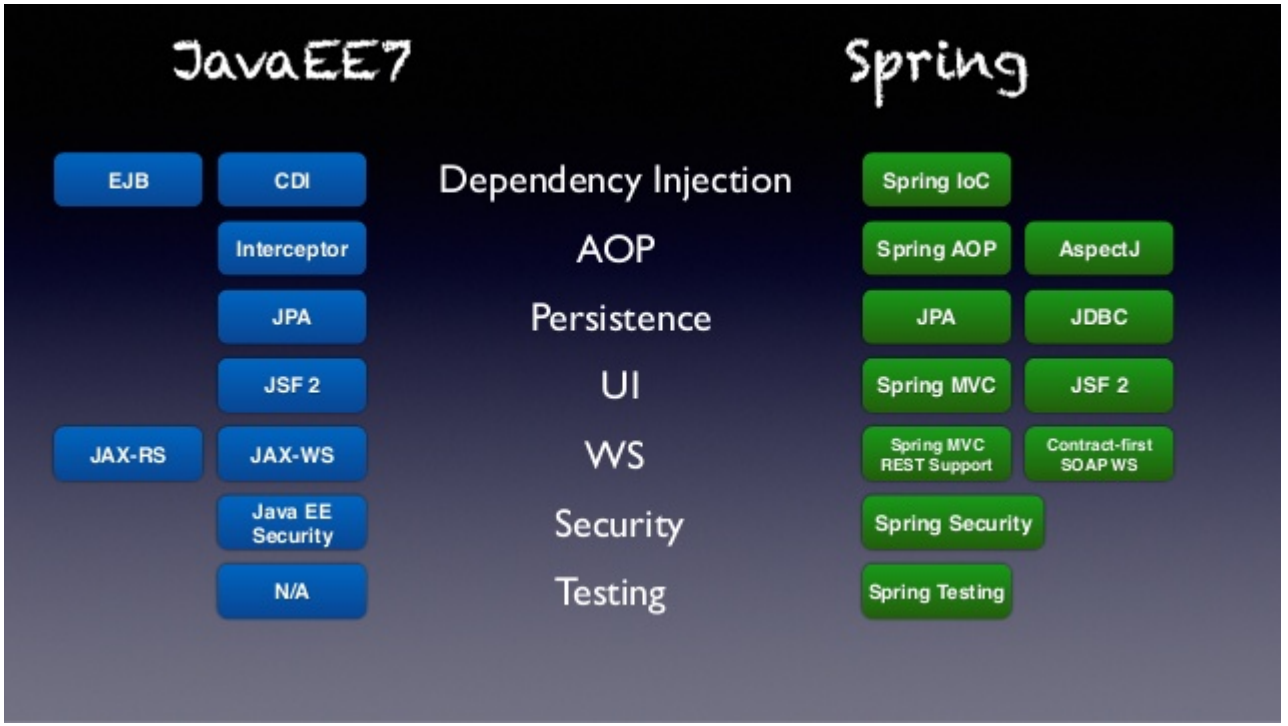
Service Locator Pattern

Observer-Observable

Context Object Pattern

Java EE vs Spring framework





Step 1: DAO, DTO

Dao Layer

```java
@Entity
public class Book {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    @Column(unique=true, nullable=false)
    private String isbn;
    private String title;
    private Double price;
    private String author;
    private String publisher;
    @Temporal(TemporalType.DATE)
    private Date pubDate;
```

```java
public interface BookDao {
    public List<Book> getAll();
    public Book add(Book book);
    public Book delete(int bookId);
    public Book update(Book book);
    public Book getBookById(int bookId);
    public Book getBookByIsbn(String isbn);

}
```

DaoImpl

```java
@Repository
public class BookDaoImpl implements BookDao {

    @Autowired
    private SessionFactory factory;

    private Session getSession(){
        return factory.getCurrentSession();
    }
    @Override
    public List<Book> getAll() {
        return getSession().createQuery("from Book").list();
    }

    @Override
    public Book add(Book book) {
         getSession().save(book);
         return book;
    }

    @Override
    public Book delete(int bookId) {
        Book book=getBookById(bookId);
        if(book!=null)
            getSession().delete(book);
        return book;
    }
```

DaoImpl

```java
@Override
public Book update(Book book) {
    getSession().merge(book);
    return book;
}

@Override
public Book getBookById(int bookId) {
    return (Book) getSession().get(Book.class, bookId);
}

@Override
public Book getBookByIsbn(String isbn) {
    return null;
}
```

Step 2: Service layer

Service Layer

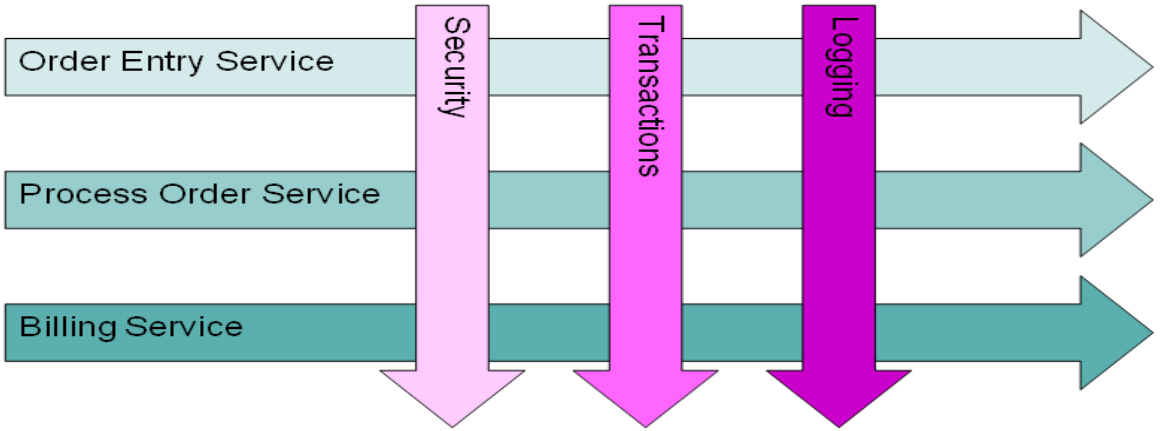
```java
public interface BookService {
    public List<Book> getAll();
    public Book add(Book book);
    public Book delete(int bookId);
    public Book update(Book book);
    public Book getBookById(int bookId);
    public Book getBookByIsbn(String isbn);
}
```

```java
import java.util.List;

@Service(value="bs")
@Transactional
public class BookServiceImpl implements BookService{

    @Autowired
    private BookDao dao;
    @Override
    public List<Book> getAll() {
        return dao.getAll();
    }

    @Override
    public Book add(Book book) {
        return dao.add(book);
    }

    @Override
```

Step 3: Applying AOP

Introduction to AOP

- What is AOP?
  - AOP is a style of programming, mainly good in separating cross cutting concerns
- How AOP works?
  - Achieved usages Proxy design Pattern to separate CCC's form actual code
  - Cross Cutting Concern ?
  - Extra code mixed with the actual code is called CCC's
  - Extra code mixed with code lead to maintenance issues
    - **Logging**
    - **validations**
    - **Auditing**
    - **Security**

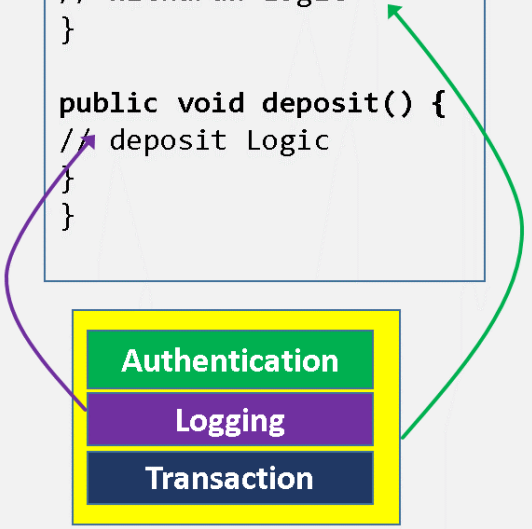


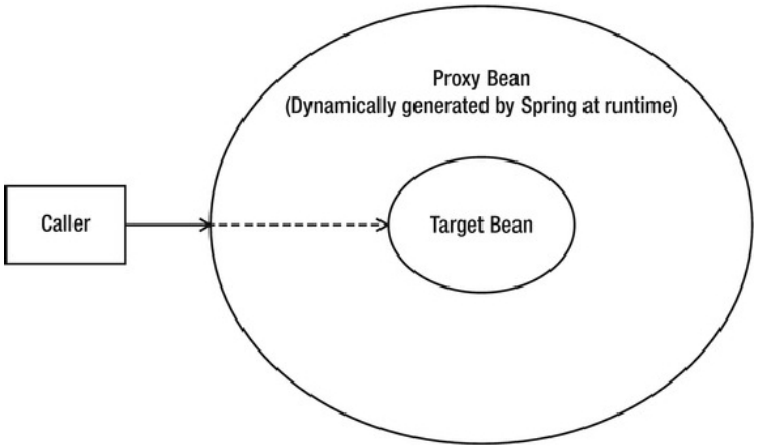
Normal Java Class

```
class Account {
public void withdraw() {
// Withdraw Logic
// Authentication
// Logging
// Transaction
}

public void deposit() {
// Deposit Logic
// Authentication
// Logging
// Transaction
}
}
```

Spring AOP

```
class Account {
public void withdraw() {
// Withdraw Logic
}

public void deposit() {
// deposit Logic
}
}
```





AOP - Definitions

- **Advice** defines what needs to be applied and when.
- **Jointpoint** is where the advice is applied.
- **Pointcut** is the combination of different joinpoints where the advice needs to be applied.
- **Aspect** is applying the Advice at the pointcuts.

Step 4: Spring Hibernate Configuration

Configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/t
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spr
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

    <context:annotation-config />
    <context:component-scan base-package="com.iris.bookapp"></context:component-scan>
    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="url" value="jdbc:mysql://localhost:3306/iris2" />
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="username" value="root" />
        <property name="password" value="root" />
    </bean>
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="packagesToScan">
            <list>
                <value>com.iris.bookapp.model.persistance</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
```

Data source configuration

Hibernate session factory configuration

```
driver=com.mysql.cj.jdbc.Driver
url=jdbc:mysql://localhost:3306/iris2?useSSL=false
username=root
password=root
```

Configuration

```
16          <property name="username" value="root" />
17          <property name="password" value="root" />
18      </bean>
19      <bean id="sessionFactory"
20          class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
21          <property name="dataSource" ref="dataSource" />
22          <property name="packagesToScan">
23              <list>
24                  <value>com.iris.bookapp.model.persistance</value>
25              </list>
26          </property>
27          <property name="hibernateProperties">
28              <props>
29                  <prop key="hibernate.hbm2ddl.auto">update</prop>
30                  <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
31                  <prop key="hibernate.show_sql">true</prop>
32                  <prop key="hibernate.format_sql">true</prop>
33              </props>
34          </property>
35      </bean>
36      <bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
37      <bean id="transactionManager"
38          class="org.springframework.orm.hibernate4.HibernateTransactionManager">
39          <property name="sessionFactory" ref="sessionFactory"></property>
40      </bean>
41
42      <tx:annotation-driven transaction-manager="transactionManager" />
43
44  </beans>
```

Hibernate factory configuration

Exception translation

Configuration of hibernate tx manager

asking spring for declerative tx

Testing

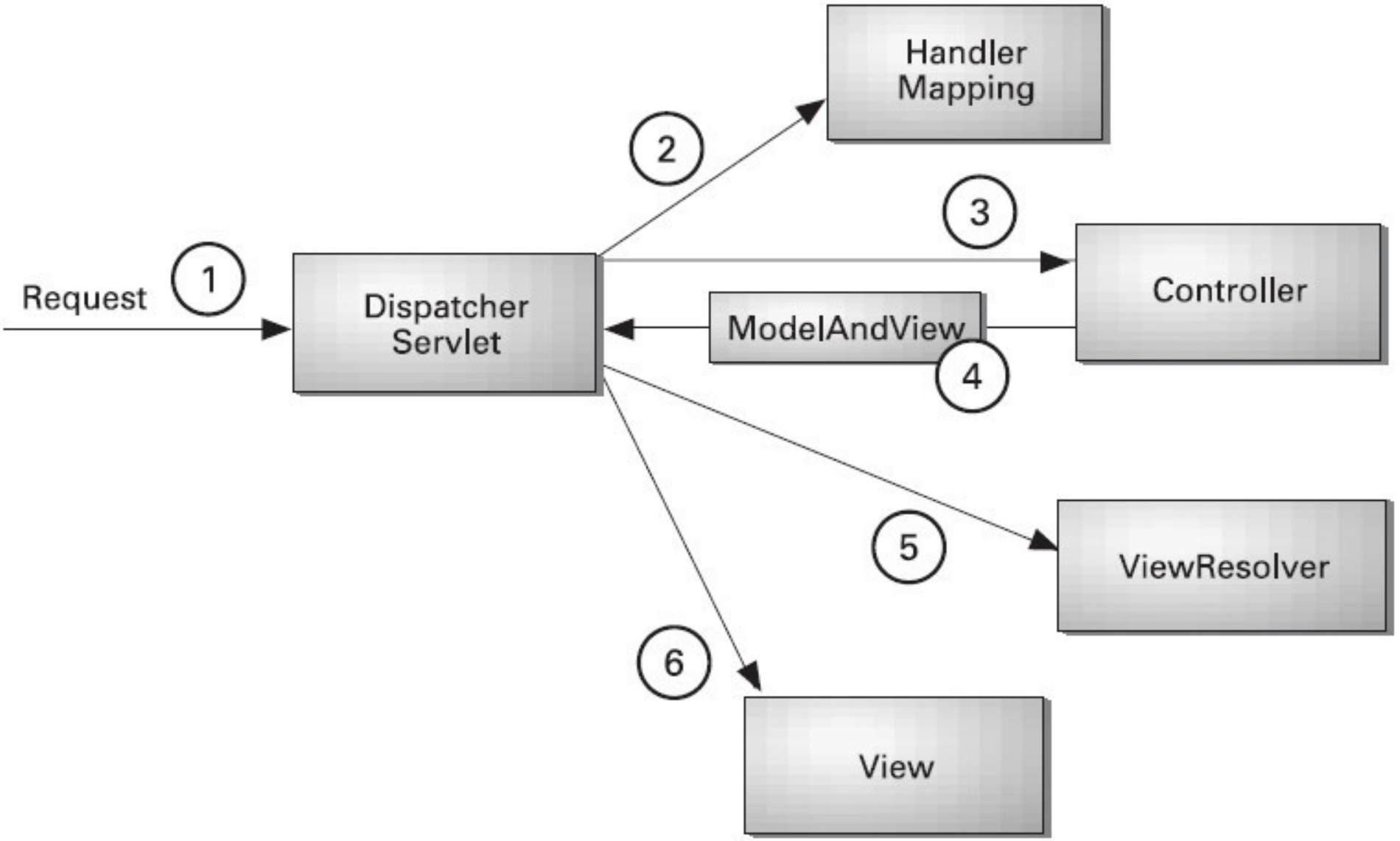
```
1  public class Tester {
2
3      public static void main(String[] args) {
4
5          ApplicationContext ctx=new ClassPathXmlApplicationContext("beans.xml") ;
6          BookService bs=ctx.getBean("bs", BookService.class);
7
8          /*List<Book> allBooks=bs.getAll();
9          for(Book temp: allBooks){
0              System.out.println(temp);
1          }*/
2          Book book=new Book("454", "a"    , "b", 99.0, new Date());
3          try{
4              bs.add(book);
5          }catch(DataAccessException ex){
6              System.out.println("handled...");
7          }
8      }
9
0  }
```

Step 5: Spring Hibernate Java Configuration

```java
@Configuration
@ComponentScan(basePackages={"com.bookapp"})
@EnableAspectJAutoProxy
@PropertySource(value="db.properties")
@EnableTransactionManagement
public class ModelConfig {

    @Autowired
    private Environment environment;

    @Bean(name="dataSource")
    public DataSource getDataSource(){
        DriverManagerDataSource ds=new DriverManagerDataSource();
        ds.setDriverClassName(environment.getProperty("driver"));
        ds.setUrl(environment.getProperty("url"));
        ds.setUsername(environment.getProperty("username"));
        ds.setPassword(environment.getProperty("password"));
        return ds;
    }

    @Bean
    public LocalSessionFactoryBean getSessionFactory(){
        LocalSessionFactoryBean sf=new LocalSessionFactoryBean();
        sf.setDataSource(getDataSource());
        sf.setPackagesToScan("com.bookapp.model.persistance");
        sf.setHibernateProperties(getHibernateProperties());
        return  sf;
    }
```

```
    public Properties getHibernateProperties() {
        Properties properties=new Properties();
        properties.setProperty("hibernate.hbm2ddl.auto", "validate");
        properties.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
        properties.setProperty("hibernate.show_sql", "true");
        properties.setProperty("hibernate.formate_sql", "true");
        return properties;
    }

    @Bean
    public PersistenceExceptionTranslationPostProcessor
    getPersistenceExceptionTranslationPostProcessor(){
        PersistenceExceptionTranslationPostProcessor ps=
                new PersistenceExceptionTranslationPostProcessor();
        return ps;
    }

    @Bean(name="transactionManager")
    //@Autowired
    public HibernateTransactionManager getHibernateTransactionManager(SessionFactory factory){
        HibernateTransactionManager tm=new HibernateTransactionManager();
        tm.setSessionFactory(factory);
        return tm;
    }
}
```

Step 6: Spring MVC basics


Handler
Mapping
2
3
Request
1
Dispatcher
Servlet
ModelAndView
Controller
4
5
ViewResolver
6
View

Hello world controller

```
create hello world controller


    @Controller
    @RequestMapping("/")
    public class HelloWorldController {

        @RequestMapping(method = RequestMethod.GET)
        public String sayHello(ModelMap model) {
            model.addAttribute("greeting", "Hello World from Spring 4 MVC");
            return "welcome";
        }

        @RequestMapping(value="/helloagain", method = RequestMethod.GET)
        public String sayHelloAgain(ModelMap model) {
            model.addAttribute("greeting", "Hello World Again, from Spring 4 MVC");
            return "welcome";
        }

    }
```

Web layer configuration

```
<mvc:annotation-driven />
<context:component-scan base-package="com" />

<!-- CONFIG INTERNAL RESO BEAN RESOLVER -->
<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
        <value>/WEB-INF/views/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

Front Controller Configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app 3 0.x
    version="3.0">
    <display-name>app</display-name>

    <servlet>
        <servlet-name>fc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/fc-configuration.xml</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>fc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

```
@PathVariable  vs @RequestParam
-----------------------------------
http://localhost:8080/app-01-spring/hello/delete/22
-----------------------------------
@Controller
@RequestMapping(value="/hello/*")
public class Hello2Controller {

	@RequestMapping(value="/delete/{sid}", method=RequestMethod.GET)
	public String sayHello(@PathVariable ("sid")int s){
		Foo foo=new Foo();
		System.out.println(s);
		return "hello";
	}

}
--------------------
foo?un=raj&pw=raj
------------------------
@Controller
@RequestMapping("/foo")
public class AnotherController {
	@RequestMapping(method=RequestMethod.GET)
	public void foo(@RequestParam("un")String un, @RequestParam("pw")String pw){
		System.out.println("un"+un);
		System.out.println("pw"+pw);
	}
}
```

ContextLoaderListner

```
=> Use ContextLoaderListener (aka ServletContextListner ) that can
        load some extra configuration files for you!



<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/otherContext.xml</param-value>
</context-param>

<listener>
        <listener-class>
                org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>



=> NOW WE CAN DEFINE OUR MODEL AND SERVICE LAYER RELATED BEANS IN otherContext.xml
```

Step 7: Spring MVC java configuration

```java
@Configuration // replacement of xml file, telling spring it is configuration file
@ComponentScan(basePackages={"com"})
@EnableWebMvc
public class AppConfig extends WebMvcConfigurerAdapter{

    @Bean
    public InternalResourceViewResolver getInternalResourceViewResolver() {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/pages/");
        resolver.setSuffix(".jsp");
        return resolver;
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
         // Don't forget the ending "/" for location or you will hit 404.
        registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");

    }
}
```

```java
public class WebInitilizer extends
        AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {

        return new Class[]{AppConfig.class};
    }

    @Override
    protected String[] getServletMappings() {

        return new String[]{"/"};
    }
}
```

```java
    }
    //Configuration for error handling
    @Override
    protected FrameworkServlet createDispatcherServlet(
            WebApplicationContext servletAppContext) {
        DispatcherServlet ds=new DispatcherServlet(servletAppContext);
        ds.setThrowExceptionIfNoHandlerFound(true);
        return ds;
    }
```

Step 8: Spring MVC CRUD

```java
@Controller
public class BookController {

    @Autowired
    private BookService bookService;

    @RequestMapping(value="showallbooks", method=RequestMethod.GET)
    public String showAllBooks(ModelMap map){
        map.addAttribute("books", bookService.getAllBooks());
        return "showallbooks";
    }

    @RequestMapping(value="addbook", method=RequestMethod.GET)
    public String showBookForm(ModelMap map){
        map.addAttribute("book", new Book());
        return "addbook";
    }

    @RequestMapping(value="updatebook", method=RequestMethod.GET)
    public String updateBook(ModelMap map, HttpServletRequest request){
        int id=Integer.parseInt(request.getParameter("id"));
        Book book=bookService.getBookById(id);
        map.addAttribute("book", book);
        return "addbook";
    }
```

```java
@RequestMapping(value="deletebook", method=RequestMethod.GET)
public String deleteBook(HttpServletRequest request){
    int id=Integer.parseInt(request.getParameter("id"));

    bookService.removeBook(id);
    return "redirect:/showallbooks";
}
//@valid must ==>BindingResult
@RequestMapping(value="addbook", method=RequestMethod.POST)
public String saveBook( @ModelAttribute(value="book") @Valid Book book, ModelMap map,
        BindingResult bindingResult  ){
    if(bindingResult.hasErrors()){
        return "addbook";
    }else{
        if(book.getId()==0)
            bookService.addBook(book);
        else
            bookService.updateBook(book);
        return "redirect:/showallbooks";
    }

}
```

Display All Books

```
<body>
    <div class="container">
        <h1>Hello World!</h1>
        <div class="row">
            <div class="col-sm-8">
                <table class="table table-striped">
                    <thead>
                        <tr>
                            <th>id</th>
                            <th>isbn</th>
                            <th>title</th>
                            <th>author</th>
                            <th>price</th>
                            <th>pubDate</th>
                            <th>publisher</th>
                        </tr>
                    </thead>
                    <tbody>
                        <c:forEach var="book" items="${books}">
                            <tr>
                                <td><c:out value="${book.id }" /></td>
                                <td><c:out value="${book.isbn }" /></td>
                                <td><c:out value="${book.title }" /></td>
                                <td><c:out value="${book.author }" /></td>
                                <td><c:out value="${book.price }" /></td>
                                <td><fmt:formatDate pattern="dd/MM/yyyy" value="${book.pubDate }" /></td>
                                <td><c:out value="${book.publisher }" /></td>
                                <td><a href="update?id=<c:out value="${book.id }"/>">update</a>
                                <td>
                                <td><a href="delete?id=<c:out value="${book.id }"/>">delete</a>
                                <td>
                            </tr>
                        </c:forEach>
                    </tbody>
                </table>

                <a href="addbook">add book</a>

            </div>
```

Add/ Edit a Book

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Add/Update Book</title>
<style>
.error {
color: #EF1313;
font-style: italic;
}
</style>
</head>
<body>
<form:form action="addbook" method="post" modelAttribute="book">
<form:hidden path="id"/>
    Enter book isbn:<form:input path="isbn"/><form:errors path="isbn"/><br/>
    Enter book title:<form:input path="title"/><form:errors path="title" /><br/>
    Enter book author:<form:input path="author"/><form:errors path="author"/><br/>
    Enter book publisher:<form:select path="publisher" items="${publishers }"/><form:errors path="publish
    Enter pub date:<form:input path="pubDate"/><form:errors path="pubDate" /><br/>
    Enter book price:<form:input path="price"/><br/>
    <input type ="submit"/>
</form:form>
</body>
</html>
```

```java
@Controller
public class HelloController {

    @RequestMapping("/welcome/{countryName}/{userName}")
    public ModelAndView helloWorld(@PathVariable Map<String,String> pathVars) {

        String name = pathVars.get("userName");
        String country = pathVars.get("countryName");

        ModelAndView model = new ModelAndView("HelloPage");
        model.addObject("msg","hello "+name+ " You are from "+country);

        return model;
    }
}
```

```java
  @RequestMapping("/welcome/countryName/{userName}")
  public ModelAndView helloWorld(@PathVariable("userName") String name) {

      ModelAndView model = new ModelAndView("HelloPage");
      model.addObject("msg","hello "+name);

      return model;
  }
```

```java
 @RequestMapping(value="/submitAdmissionForm.html", method = RequestMethod.POST)
 public ModelAndView submitAdmissionForm(@RequestParam("studentName") String name, @RequestParam("studentHobby") String

     ModelAndView model = new ModelAndView("AdmissionSuccess");
     model.addObject("msg","Details submitted by you:: Name: "+name+ ", Hobby: " + hobby);

     return model;
 }
```

Step 9: Spring MVC validations

JSR 303 Validation API

- ***@NotNull*** – validates that the annotated property value is not *null*
- ***@AssertTrue*** – validates that the annotated property value is *true*
- ***@Size*** – validates that the annotated property value has a size between the attributes *min* and *max*; can be applied to *String*, *Collection*, *Map*, and array properties
- ***@Min*** – *V*alidates that the annotated property has a value no smaller than the *value* attribute
- ***@Max*** – validates that the annotated property has a value no larger than the *value* attribute
- ***@Email*** – validates that the annotated property is a valid email address

  - ***@NotEmpty*** – validates that the property is not null or empty; can be applied to *String*, *Collection*, *Map* or *Array* values
  - ***@NotBlank*** – can be applied only to text values and validated that the property is not null or whitespace
  - ***@Positive*** and ***@PositiveOrZero*** – apply to numeric values and validate that they are strictly positive, or positive including 0
  - ***@Negative*** and ***@NegativeOrZero*** – apply to numeric values and validate that they are strictly negative, or negative including 0
  - ***@Past*** and ***@PastOrPresent*** – validate that a date value is in the past or the past including the present; can be applied to date types including those added in Java 8
  - ***@Future*** *and* ***@FutureOrPresent*** – validates that a date value is in the future, or in the future including the present

Entity with validation annotations

```java
@Entity
public class Book {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    @Column(unique=true, nullable=false)
    @NotBlank(message="isbn can not be blank")
    private String isbn;
    @NotBlank(message="title can not be blank")
    private String title;
    @Digits(integer=10, fraction=0, message="price can only be integer")
    private Double price;
    @NotBlank(message="author can not be blank")
    private String author;
    @NotBlank(message="publisher can not be blank")
    private String publisher;
    @NotNull
    @Past(message="date in dd/mm/yyyy formate must be past date")
    @DateTimeFormat(pattern = "dd/MM/yyyy")
    @Temporal(TemporalType.DATE)
    private Date pubDate;
```

```
some imp points:
-----------------
for converting string ==> desire data formate
-------------------------------------
@DateTimeFormat(pattern = "dd/MM/yyyy")

must use  <mvc:annotation-driven


auto populate some field:
-------------------------
Enter Book Type: <form:select path="pubName" items="${pubNammes}"/><br/>


@ModelAttribute(value="pubNammes")
        public List<String> getGender(){
                return ....;
}

Putting messages from external file
-----------------------------------
messages.properties
NotEmpty.book.isbn=isbn can not be blank

How spring come to know about it?
-------------------------------------
        <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
                <property name="basename" value="messages" />
        </bean>

```

Step 10: Spring MVC Exception handling

Exception handling

```
@ExceptionHandler(BookNotFoundException.class)
public ModelAndView handleBookNotFoundException(HttpServletRequest request, Exception ex){
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("exception", ex);
    modelAndView.addObject("url", request.getRequestURL());

    modelAndView.setViewName("error");
    return modelAndView;
}
```

```java
@ControllerAdvice
public class GlobalDefaultHandler {

    @ExceptionHandler(NoHandlerFoundException.class)
    public ModelAndView handlerNotFoundEx() {
        ModelAndView mv = new ModelAndView();
        mv.setViewName("404");
        mv.addObject("error", "resource/page not found");
        return mv;
    }

    @ExceptionHandler(DataAccessException.class)
    public String handleDataAccessException(HttpServletRequest request,
            Exception ex) {
        ModelAndView mv = new ModelAndView();
        mv.addObject("exception", ex);
        mv.addObject("url", request.getRequestURL());
        return "database_error";
    }
}
```

```xml
<servlet>
    <servlet-name>springDispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:web-config.xml</param-value>
    </init-param>
    <init-param>
        <param-name>throwExceptionIfNoHandlerFound</param-name>
        <param-value>true</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<!-- Map all requests to the DispatcherServlet for handling -->
<servlet-mapping>
    <servlet-name>springDispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

Step 11: Spring REST

Spring REST

HTTP Method	Operation Performed
GET	Get a resource (Read a resource)
POST	Create a resource
PUT	Update a resource
DELETE	Delete a resource

Spring Annotations for REST

Annotations	Usage
@Controller	mark the class as a MVC controller
@RequestMapping	Maps the request with path
@PathVariable	Map variable from the path
@RequestBody	unmarshalls the HTTP response body into a Java object injected in the method.
@ResponseBody	marshalls return value as HTTP Response
@Configuration	Spring Config as a class

Example showing Annotations

```
@Controller
@RequestMapping(value = "/ilo")
public class iLOController
{
    @RequestMapping(value = "/server/{id}", method = RequestMethod.GET)
    public @ResponseBody Book getServer(@PathVariable String id) {
        System.out.println("------Gettting Server ------"+id);
    }
        ……
        ……..
}
```

```java
@RestController//   @RestController=@Controller + @ResponseBody
public class BookResources {

    @Autowired
    private BookService service;

    @RequestMapping(value = "/api/book", method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Collection<Book>> getAllBooks() {
        Collection<Book> greetings = service.getAllBooks();
        return new ResponseEntity<Collection<Book>>(greetings, HttpStatus.OK);
    }

    @RequestMapping(value = "/api/book/{id}", method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Book> getAnBook(@PathVariable Integer id) {
        Book book = service.getBookById(id);
        if (book == null) {
            return new ResponseEntity<Book>(HttpStatus.NOT_FOUND);
        }

        return new ResponseEntity<Book>(book, HttpStatus.OK);
    }
```

```java
@RequestMapping(value = "/api/book", method = RequestMethod.POST,
        consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Book> createBook(@RequestBody Book book) {
    Book savedBook = service.addBook(book);
    return new ResponseEntity<Book>(savedBook, HttpStatus.CREATED);
}

@RequestMapping(value = "/api/book/{id}", method = RequestMethod.PUT,
        consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Book> updateBook(@PathVariable Integer id,
        @RequestBody Book book) {

    service.updateBook(book);

    return new ResponseEntity<Book>(HttpStatus.OK);
}

@RequestMapping(value = "/api/book/{id}", method = RequestMethod.DELETE)
public ResponseEntity<Book> deleteBook(@PathVariable("id") Integer id)
        throws Exception {

    service.removeBook(id);

    return new ResponseEntity<Book>(HttpStatus.NO_CONTENT);
}
```

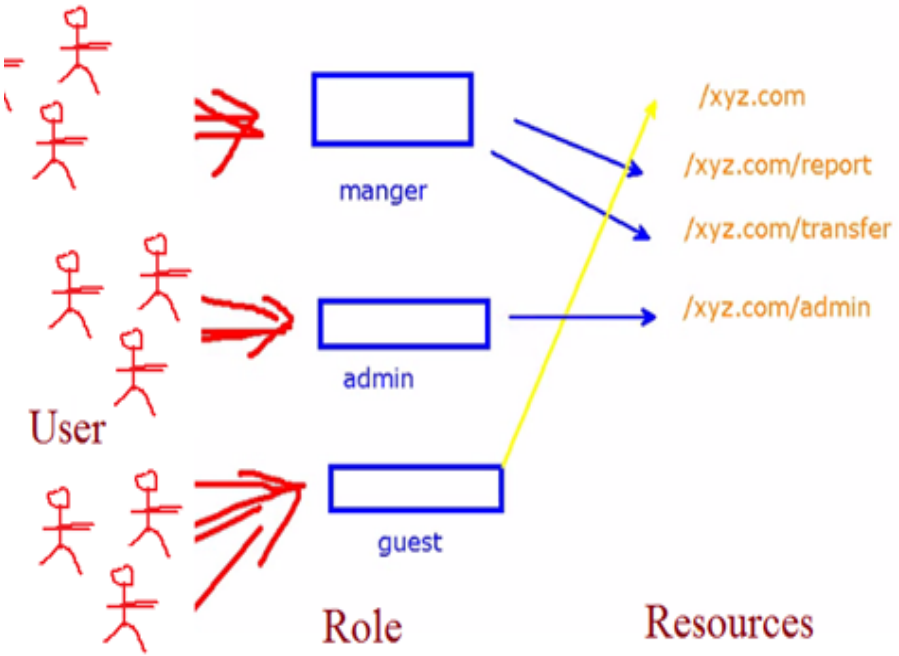
Step 12: Spring Integration test

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations="classpath:spring-context.xml")
@TransactionConfiguration(defaultRollback=true,transactionManager="transactionManager")
public class EmployeeHibernateDAOImplTest {

	@Autowired
	private EmployeeDAO employeeDAO;

	@Test
	public void testGetEmployeeById() {
		Employee emp = employeeDAO.getEmployeeById(1L);

		assertNotNull(emp);
	}

	@Test
	public void testCreateEmployee()
	{
		Employee emp = new Employee();
		emp.setName("Emp123");
		Long key = employeeDAO.createEmployee(emp);

		assertEquals(2L, key.longValue());
	}

}
```

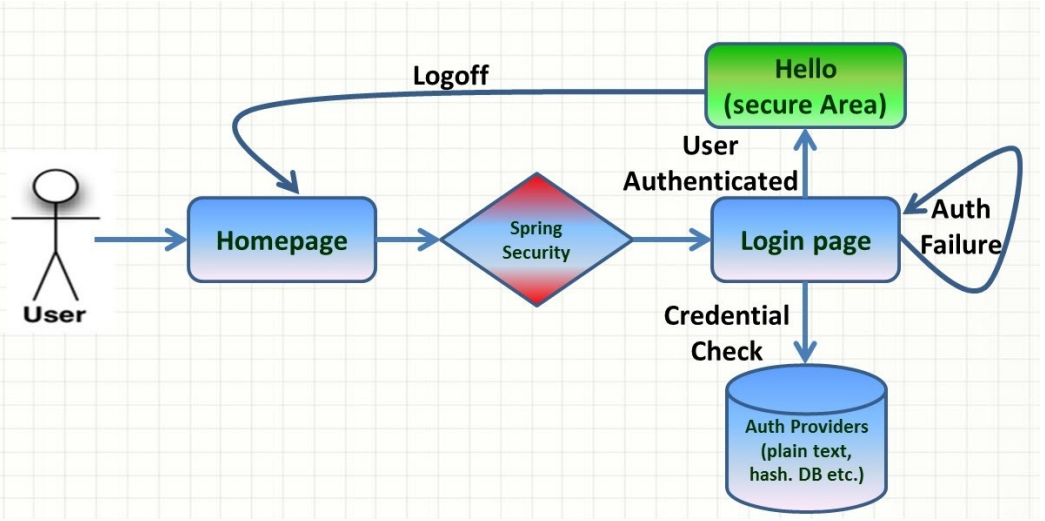
Step 13: Spring Security xml

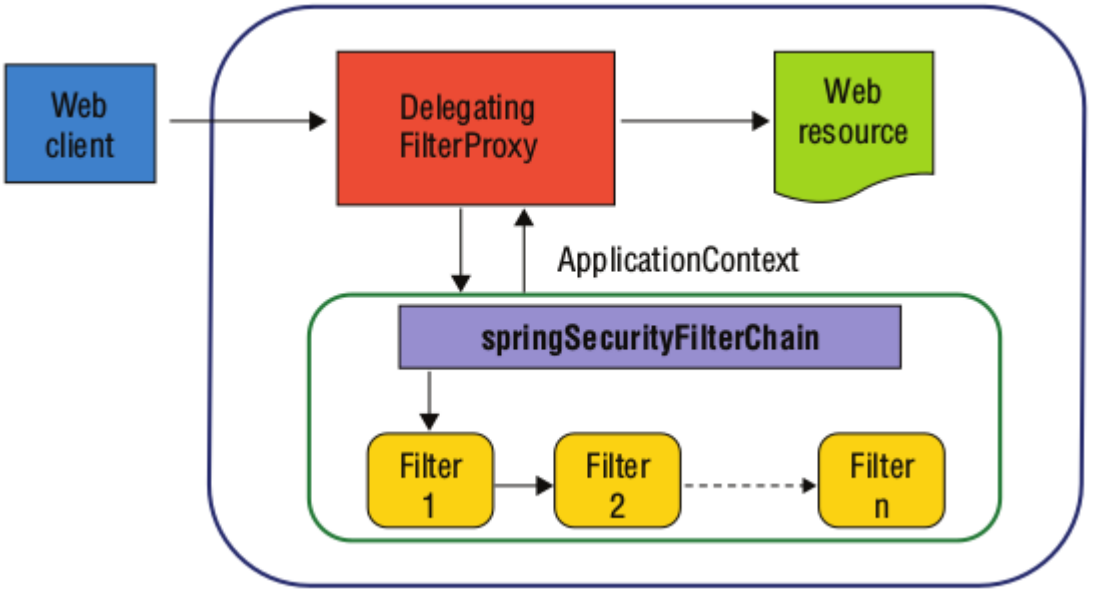
Role based Access Control RAC

- Role based Access Control (RAC)
  - As it is difficult to manage permission for each user, each user is assigned to a role and permission is set for the role
  - Authentication using Spring
    - Http Basic Authentication (uses in XML- pop up form)
    - Http form based Authentication(uses in XML- custom form)
    - Http form based Authentication(uses in DB)

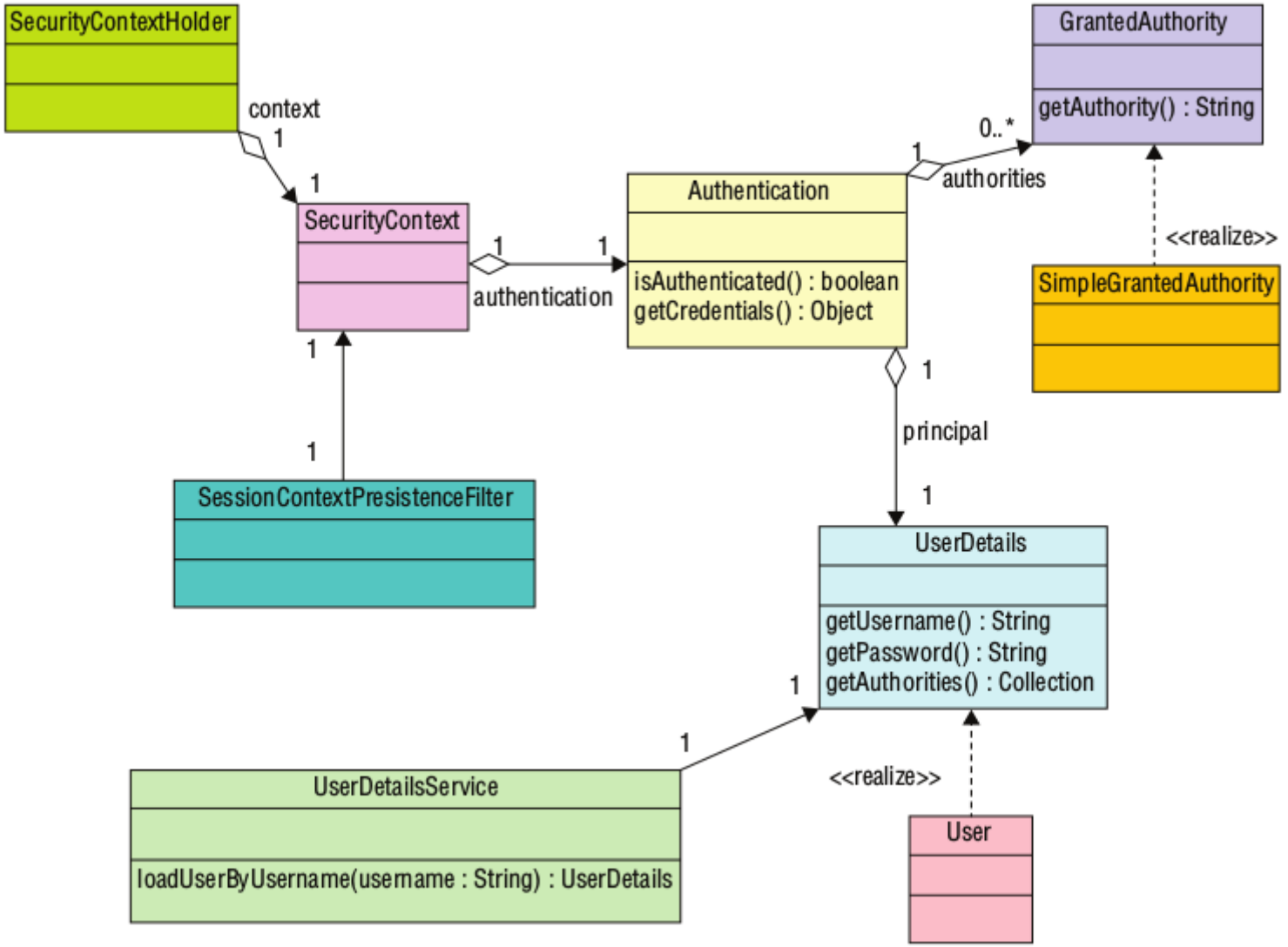


Spring security Why? How?

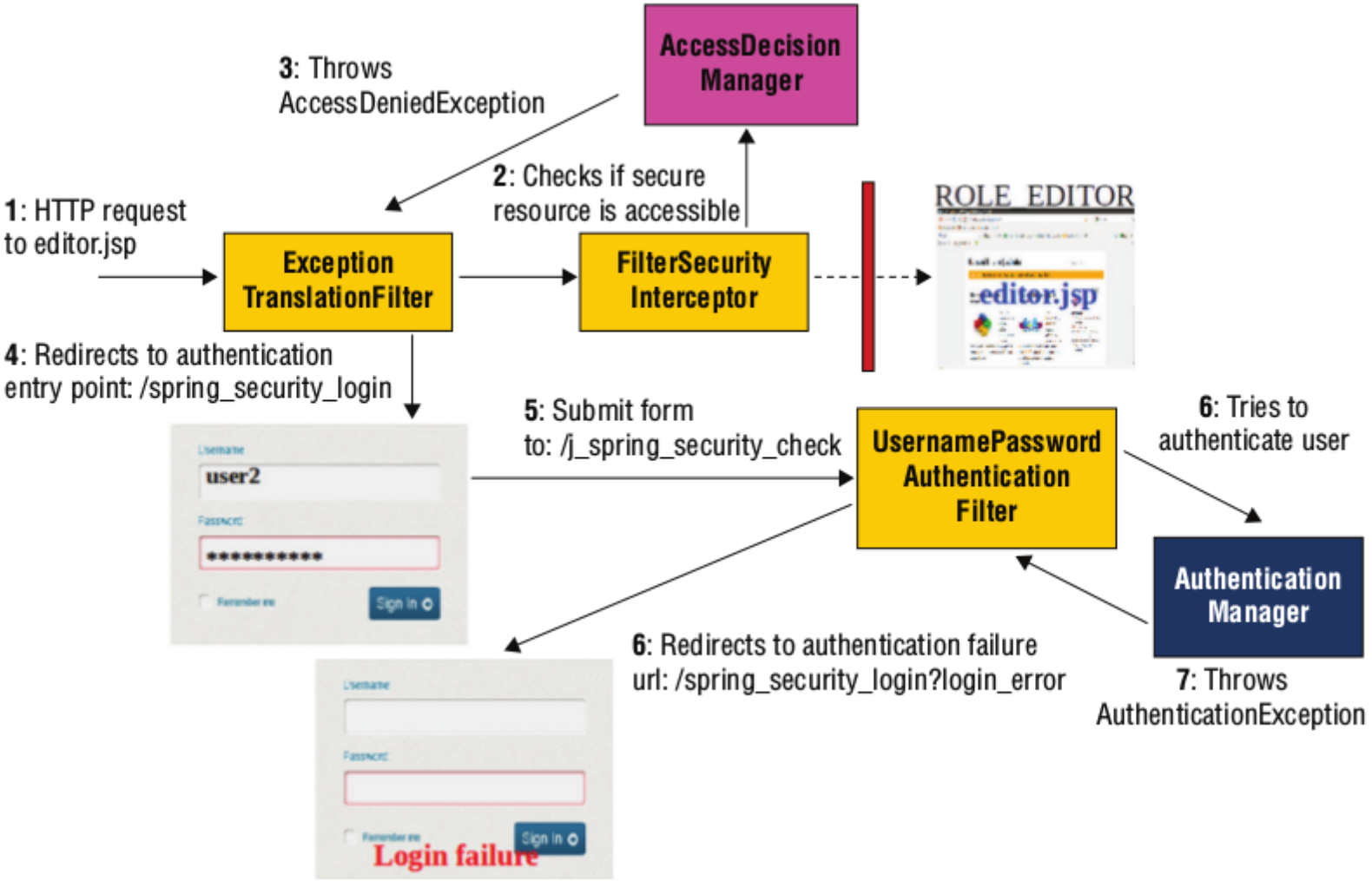




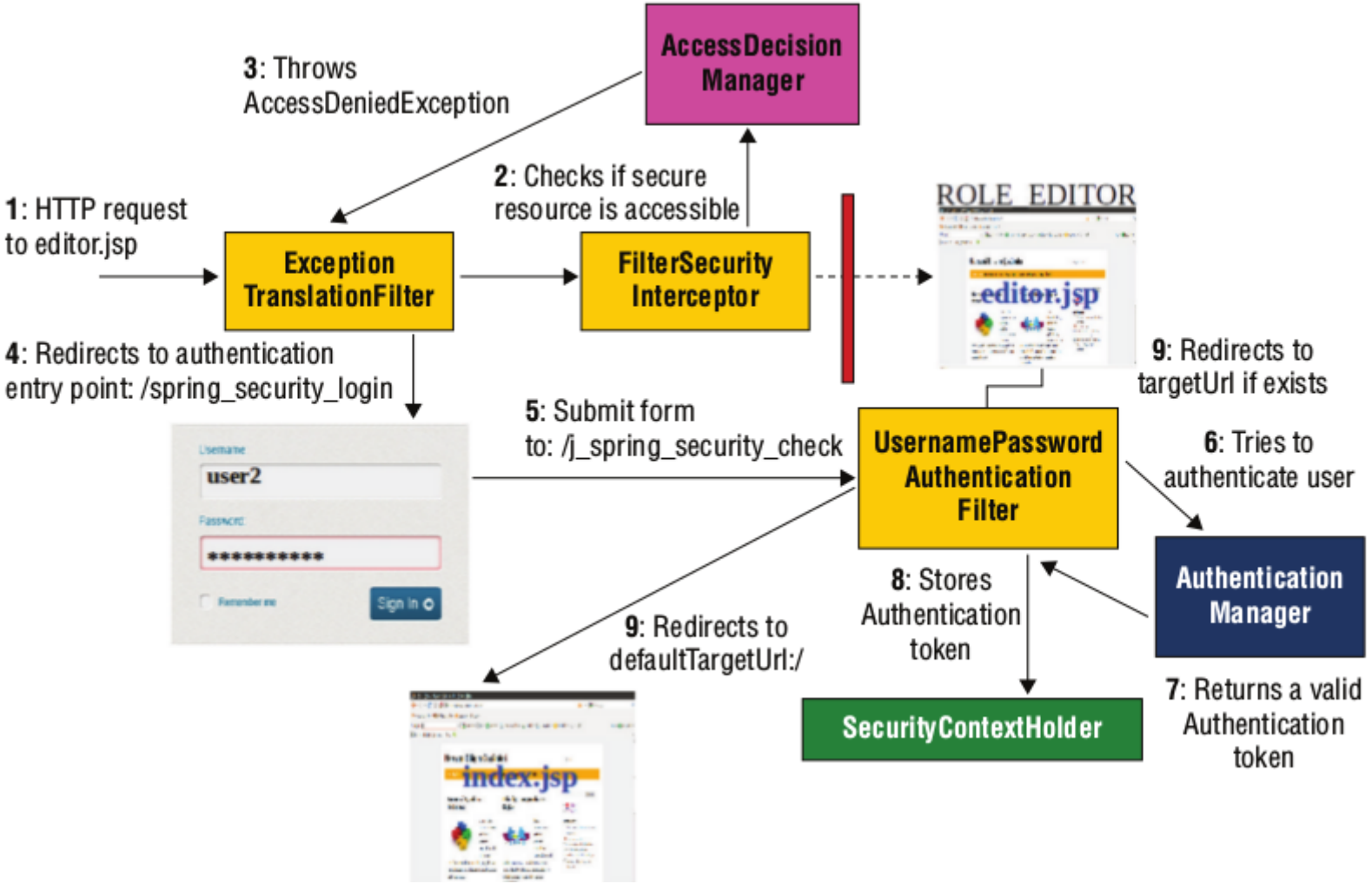
Building Blocks of Spring Security



Unsuccessful Login Flow



Successful Login Flow



Configuration spring security

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:model-config.xml,classpath:spring-security.xml</param-value>
</context-param>

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- Spring Security -->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

configure security configuration xml file
this contain information about authentication/auth

Configure security filter

Configuration spring security:authentication and authorization

```
<security:http use-expressions="true">
    <security:intercept-url pattern="/**" access="isAuthenticated()" />
    <security:form-login default-target-url="/home"/>
</security:http>
 <security:authentication-manager>
    <security:authentication-provider>
         <security:user-service>
            <security:user name="foo" password="{noop}foo" authorities="Admin, User"/>
            <security:user name="bar" password="{noop}bar" authorities="User"/>
        </security:user-service>
    </security:authentication-provider>
</security:authentication-manager>
```

Step 14: Spring Security java config

Spring Security java config Hello world

- Step 1: configure AbstractSecurityWebApplicationInitializer

```
//this class magically configure security filter
public class SecurityWebInitilizer extends
    AbstractSecurityWebApplicationInitializer{
}
```

- Step 2: Configure authentication

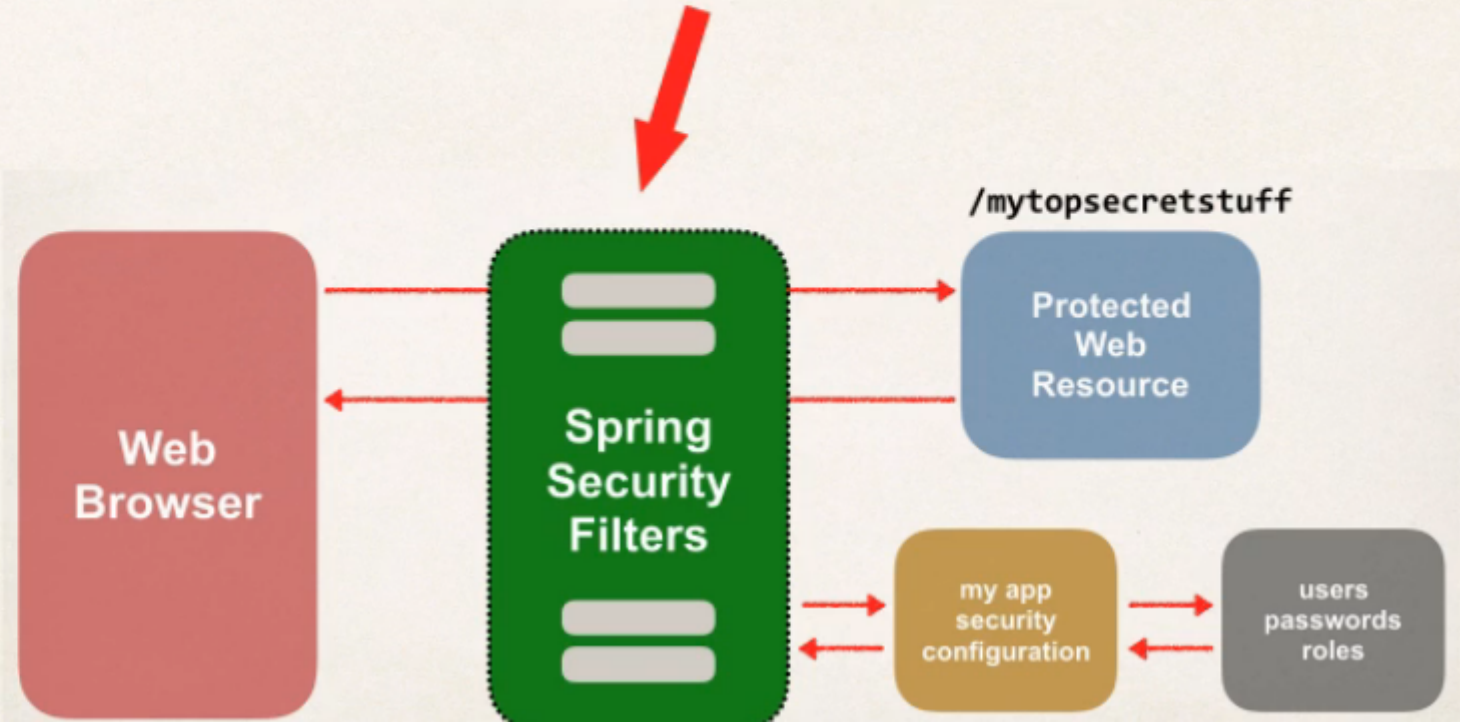
```
@Configuration
@EnableWebSecurity
@ComponentScan(basePackages={"com.bookapp.security.config"})
public class SecurityConfig extends WebSecurityConfigurerAdapter{
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        //add user for in memory auth
        UserBuilder users=User.withDefaultPasswordEncoder();
        auth.inMemoryAuthentication()
        .withUser(users.username("raj").password("raj").roles("admin"))
        .withUser(users.username("ekta").password("ekta").roles("mgr"))
        .withUser(users.username("gunika").password("gunika").roles("emp"));
    }
}
```

- Step 3: Don’t forget to configure security config in spring freamework

```
public class WebInitilizer extends AbstractAnnotationConfigDispatcherServletInitializer{

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[]{AppConfig.class, SecurityConfig.class};
    }
```

AbstractSecurityWebApplicationInitializer

Special class to register the **Spring Security Filters**



Spring security custom login page

- Step 1: Modify Spring config to refer custom login page

```
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
        .anyRequest().authenticated()
        .and()
        .formLogin()
            .loginPage("/showLoginPage")//url pattern of controller that display login page
            .loginProcessingUrl("/authTheUser")
            .permitAll();
    }
```

- Step 2: create an controller to show that login page

```
@Controller
public class LoginController {
    @GetMapping("/showLoginPage")
    public String showLoginPage(){
        return "login_page";
    }
}
```

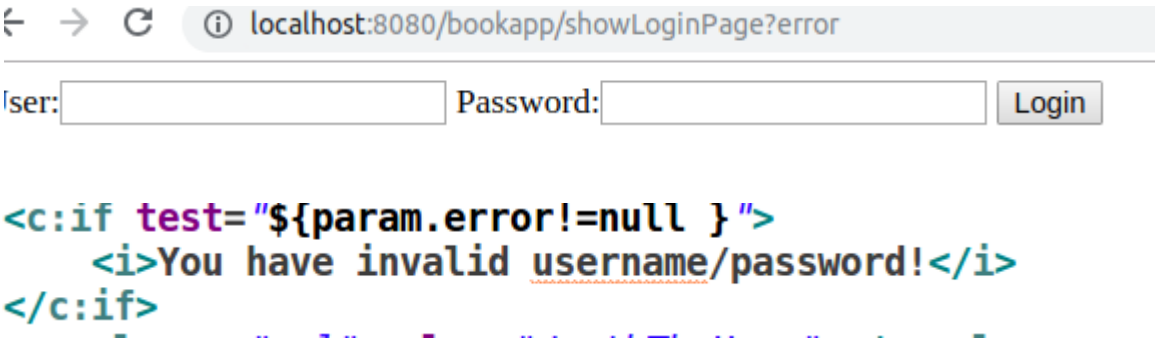
- Step 3 :create custom login page

```
<%-- <form:form action="${pageContext.request.contextPath }" method="Post"> --%>

<c:url var="url" value="/authTheUser"></c:url>
<form:form action="${url }" method="Post">
    <tr><td>User:</td><td><input type='text' name='username'></td></tr>
    <tr><td>Password:</td><td><input type='password' name='password'/></td></tr>
    <input name="submit" type="submit" value="Login"/>
</form:form>
```

Spring security custom login page

- Step 4: custom error messages

  When login failed, spring security put error parameter automatically




```
<c:if test="${param.error!=null }">
    <i>You have invalid username/password!</i>
</c:if>
```

You have invalid username/password!

User:

Password:

Login

Spring security custom login page

```
        .formLogin()
            .loginPage("/showLoginPage")
            .loginProcessingUrl("/authThe
            .permitAll()
        .and()
            .logout().permitAll();
}
```

- Step 4: custom logout
  - spring security automatically configure url /logout now we can create an logout controller, logout would be handled by spring security.
  - Spring security invalidate user httpsession and remove session cookies etc
  - Send back to your login page and append an logout parameter: ? logout

```
<c:url var="logout" value="/logout"></c:url>
<form:form action="${logout }" method ="post">
    <input type="submit" value="logout">
</form:form>


<a href="getallbooks">all books</a>
```

- Step 5: logout controller, no need to create any controller!, mention logout message on login page

```
<c:if test="${param.logout!=null }">
    <i>You logged out successfully!</i>
</c:if>
```

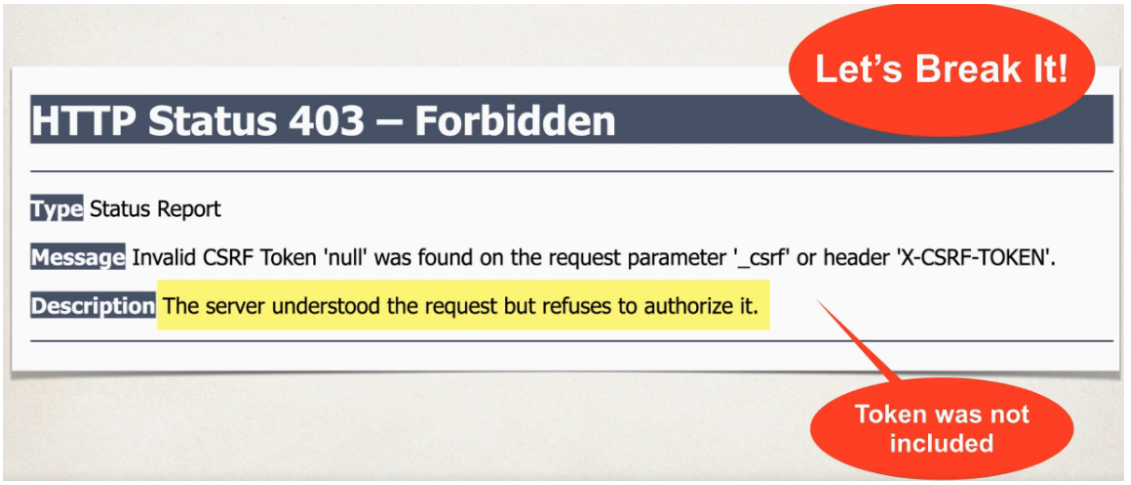
Spring security: csrf configuration

- We do not need to use csrf tags if we are using <form:form>

```
<form action="..." method="POST">

        <input type="hidden"
               name="${_csrf.parameterName}"
              value="${_csrf.token}" />

</form>
```

- <form:form> automagically adds CSRF token

Best Practice



Spring security custom login page

- Step 6: display user role and username on JSP after logged in

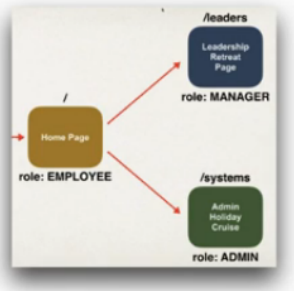
logout

User: raj User profile: [ROLE_ADMIN] all books

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

  User: <sec:authentication property="principal.username"/>
  User profile: <sec:authentication property="principal.authorities"/>
```

- Step 6 : restrict access as per role

```
@Override
protected void configure(HttpSecurity http) throws Exception {

  http.authorizeRequests()
    .antMatchers("/").hasRole("EMPLOYEE")
    .antMatchers("/leaders/**").hasRole("MANAGER")
    .antMatchers("/systems/**").hasRole("ADMIN")
    .and()
    .formLogin()
    ...
}
```



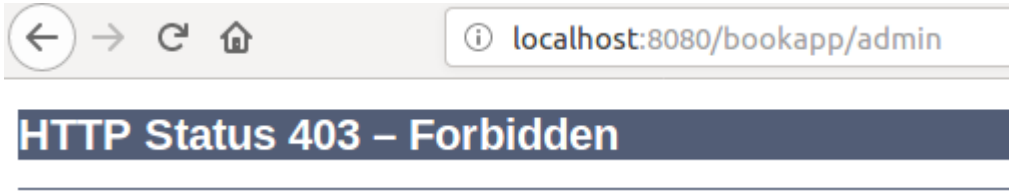

```
auth.inMemoryAuthentication()
  .withUser(users.username("john").password("test123").roles("EMPLOYEE"))
  .withUser(users.username("mary").password("test123").roles("EMPLOYEE", "MANAGER"))
  .withUser(users.username("susan").password("test123").roles("EMPLOYEE", "ADMIN"));
```

Spring security custom login page

- Step 6 : restrict access as per role

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
    .antMatchers("/").hasAnyRole("EMP","ADMIN","MGR")
    .antMatchers("/admin/**").hasRole("ADMIN")
    .antMatchers("/mgr/**").hasRole("MGR")
    /*.anyRequest().authenticated()*/
    .and()
    .formLogin()
        .loginPage("/showLoginPage")//url pattern of controller that display login page
        .loginProcessingUrl("/authTheUser")
        .permitAll()
    .and()
        .logout().permitAll();
}
```

```
<a href="admin">admin</a><br/>
<a href="mgr">admin</a><br/>
<a href="getallbooks">all books</a><br/>
```




```
@Controller
public class LoginController {
    @GetMapping("/")
    public String showPage(){
        return "home";
    }

    @GetMapping(value="/admin")
    public String showAdmin(){
        return "home_admin";
    }

    @GetMapping(value="/mgr")
    public String showMgr(){
        return "home_mgr";
    }
```

Spring security custom login page

- Step 7: create an custom access denied page

```
.and()
    .logout().permitAll()
.and()
    .exceptionHandling().accessDeniedPage("/access_denied");
```

```
@GetMapping(value="/access_denied")
public String showAccessDeniedPage(){
    return "access_denied";
}
```

```
access_denied
<c:url var="home" value="/"></c:url>
<a href="${home }">home</a>
</body>
```

Spring security custom login page

- Step 8: Display content based on role

```
.and()
    .logout().permitAll()
.and()
    .exceptionHandling().accessDeniedPage("/access_denied");
```

```
@GetMapping(value="/access_denied")
public String showAccessDeniedPage(){
    return "access_denied";
}
```

```
access_denied
<c:url var="home" value="/"></c:url>
<a href="${home }">home</a>
</body>
```

Spring security Display content based on roles

```
<sec:authorize access="hasAnyRole('MGR','ADMIN')">
<a href="mgr">MGR</a><br/>
</sec:authorize>

<sec:authorize access="hasRole('ADMIN')">
<a href="admin">admin</a><br/>
</sec:authorize>
```

Spring Security with hibernate

- Step 1: Create Entity, DAO, DAOImp ...

```java
@Entity
@Table
public class User {
    public static final BCryptPasswordEncoder encode=
            new BCryptPasswordEncoder();
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String username;
    private String password;
    private String email;
    private boolean status;

    private String []roles;
    public User(String username, String password, String email, boolean status,
            String[] roles) {
        super();
        this.username = username;
        setPassword(password);
        this.email = email;
        this.status = status;
        this.roles = roles;
    }

    public void setPassword(String password) {
        this.password = encode.encode(password);
    }
```

```java
public interface UserDao {
    public User findByUsername(String username);
    public void insert(User user);
}
```

```java
@Repository
public class UserDaoImpl implements UserDao {

    @Autowired
    private SessionFactory sessionFactory;

    private Session openSession() {
        return sessionFactory.getCurrentSession();
    }

    public User findByUsername(String username){
        Query query = openSession().createQuery("from User u where u.username = :username");
        query.setParameter("username", username);
        List<User>userList = query.list();
        if (userList.size() > 0)
            return userList.get(0);
        else
            return null;
    }

    public void insert(User user){
        openSession().save(user);
    }
}
```

Spring Security with hibernate

- Step 2: Create Service layer ...

```
public interface UserService {
    public User findByUsername(String username);
    public void insert(User user);
}
```

```
@Service
@Transactional
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDao userDao;

    public User findByUsername(String username){
        return userDao.findByUsername(username);

    }
    @Override
    public void insert(User user) {
        userDao.insert(user);
    }

}
```

```
@Service
@Transactional
public class DetailService implements UserDetailsService {

    @Autowired
    private UserService userService;

    @PostConstruct
    public void post(){
        User user1=new User("raj", "raj", "raj.mtech@gmail.com", true, new String[]{"ROLE_EMP", "ROLE_MGR","ROLE_ADM
        User user2=new User("eku", "eku", "eku.mtech@gmail.com", true, new String[]{"ROLE_MGR","ROLE_EMP"});
        User user3=new User("gun", "gun", "raj.mtech@gmail.com", true, new String[]{"ROLE_EMP"});

        System.out.println("inserting an default user");
        userService.insert(user1);
        userService.insert(user2);
        userService.insert(user3);
    }
    @Override
    public UserDetails loadUserByUsername(String username)throws UsernameNotFoundException {
        User user=userService.findByUsername(username);
        if(user==null)
            throw new UsernameNotFoundException("username is not found");
        return new org.springframework.security.core.userdetails.User(user.getUsername()
                ,user.getPassword(),AuthorityUtils.createAuthorityList(user.getRoles()));
    }
}
```

Spring Security with hibernate

- Step 3:Injecting UserDetailService to security config so that it use hibernate configuration

-

```
@Configuration
@EnableWebSecurity
@ComponentScan(basePackages={"com.bookapp.security.config"})
public class SecurityConfig extends WebSecurityConfigurerAdapter{
    @Autowired
    private UserDetailsService detailService;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
        .antMatchers("/").hasAnyRole("EMP","ADMIN","MGR")
        .antMatchers("/admin/**").hasRole("ADMIN")
        .antMatchers("/mgr/**").hasRole("MGR")
        /*.anyRequest().authenticated()*/
        .and()
        .formLogin()
            .loginPage("/showLoginPage")//url pattern of controller that display login page
            .loginProcessingUrl("/authTheUser")
            .permitAll()
        .and()
            .logout().permitAll()
        .and()
            .exceptionHandling().accessDeniedPage("/access_denied");
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(detailService).passwordEncoder(com.bookapp.model.entities.User.encode);

    }
}
```


Thank
you!!