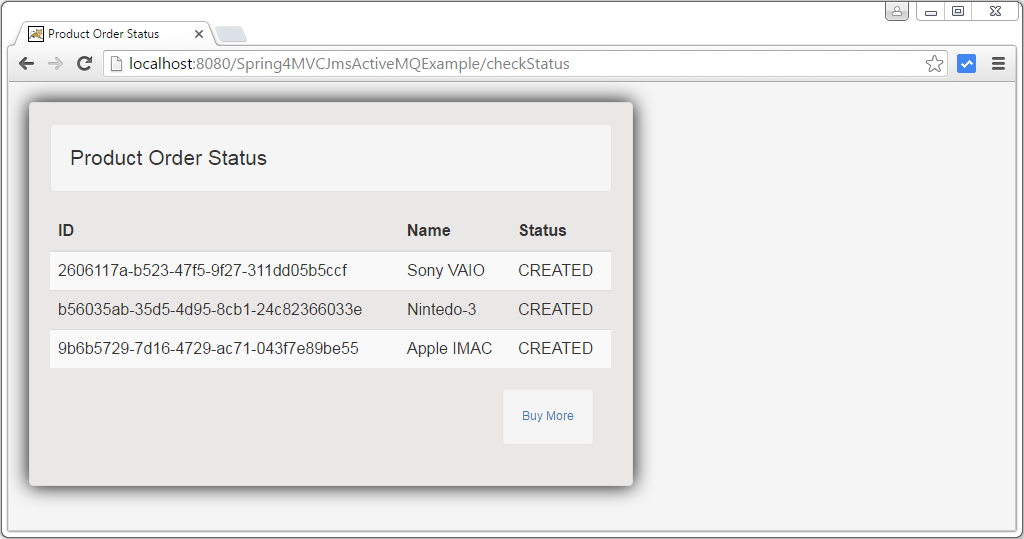
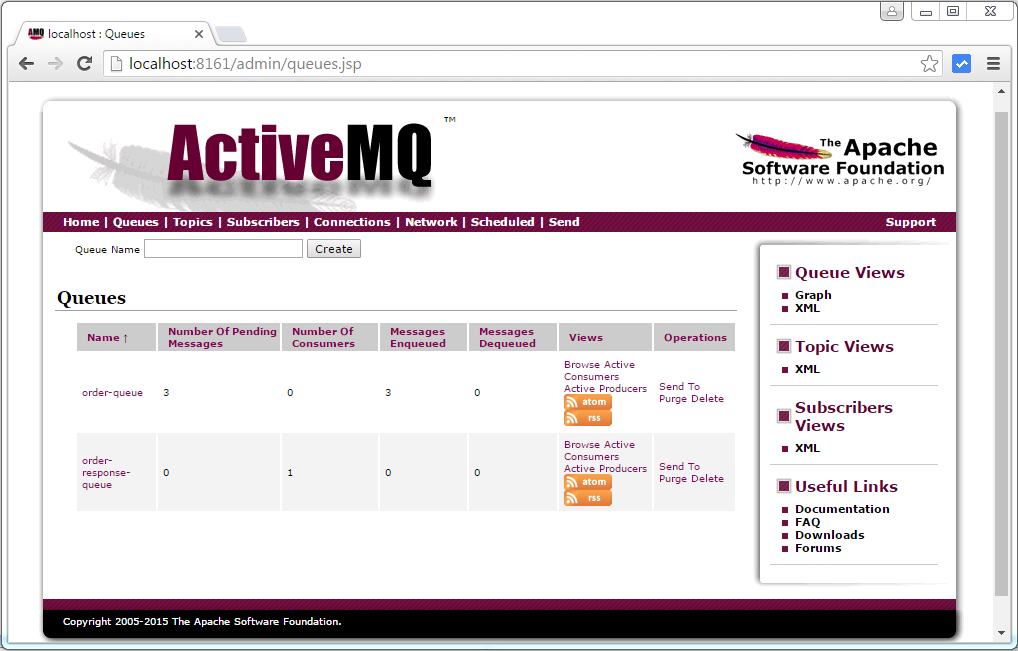
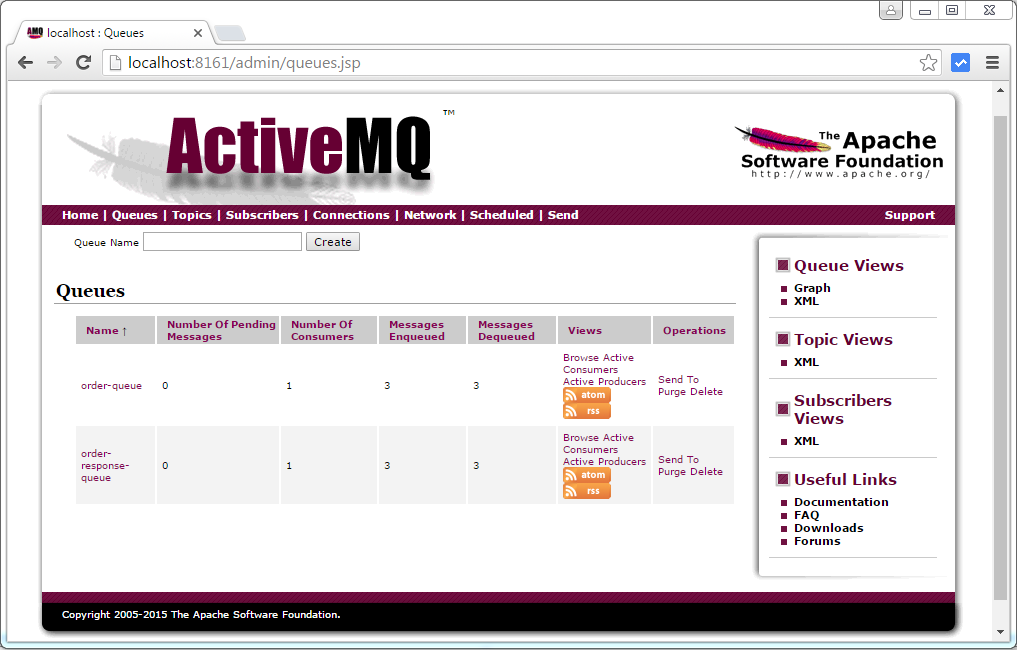
**Spring 4 MVC + JMS + ActiveMQ annotation based Example**

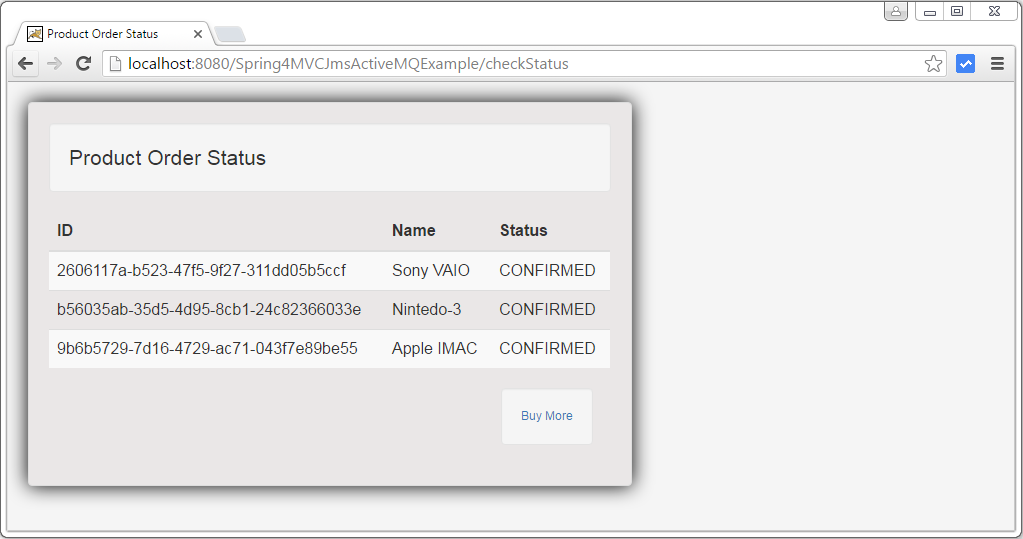
**Created on:** June 26, 2016 | **Last updated on:** March 12, 2017 [websystiqueadmin](http://websystique.com/author/blogadminuser/)

Spring provides first-class support for JMS, enabling applications integration using standard based messaging. In this post, we will build Spring 4 MVC application communicating with another Spring-based application, using JMS through Apache ActiveMQ. For a general introduction on JMS itself, please refer to post [Spring 4 + JMS+ActiveMQ Annoataion based Example](http://websystique.com/spring/spring-4-jms-activemq-example-with-annotations/). We will try to touch them in this post as well. Let’s get started.









**Other interesting posts you may like**

* [Spring Boot+AngularJS+Spring Data+Hibernate+MySQL CRUD App](http://websystique.com/spring-boot/spring-boot-angularjs-spring-data-jpa-crud-app-example/)
* [Spring Boot REST API Tutorial](http://websystique.com/spring-boot/spring-boot-rest-api-example/)
* [Spring Boot WAR deployment example](http://websystique.com/spring-boot/spring-boot-war-deployment-example/)
* [Spring Boot Introduction + Hello World Example](http://websystique.com/spring-boot/spring-boot-introduction-hello-world-example/)
* [Secure Spring REST API using OAuth2](http://websystique.com/spring-security/secure-spring-rest-api-using-oauth2/)
* [AngularJS+Spring Security using Basic Authentication](http://websystique.com/spring-security/angularjs-basic-authentication-using-spring-security/)
* [Secure Spring REST API using Basic Authentication](http://websystique.com/spring-security/secure-spring-rest-api-using-basic-authentication/)
* [Spring 4 Caching Annotations Tutorial](http://websystique.com/spring/spring-4-cacheable-cacheput-cacheevict-caching-cacheconfig-enablecaching-tutorial/)
* [Spring 4 Cache Tutorial with EhCache](http://websystique.com/spring/spring-4-cache-tutorial-with-ehcache/)
* [Spring 4 MVC+JPA2+Hibernate Many-to-many Example](http://websystique.com/springmvc/spring-4-mvc-jpa2-hibernate-many-to-many-example/)
* [Spring 4 Email Template Library Example](http://websystique.com/spring/spring-4-email-using-velocity-freemaker-template-library/)
* [Spring 4 Email With Attachment Tutorial](http://websystique.com/spring/spring-4-email-with-attachment-tutorial/)
* [Spring 4 Email Integration Tutorial](http://websystique.com/spring/spring-4-email-integration-tutorial/)
* [Spring MVC 4+JMS+ActiveMQ Integration Example](http://websystique.com/springmvc/spring-4-mvc-jms-activemq-annotation-based-example/)
* [Spring 4+JMS+ActiveMQ @JmsLister @EnableJms Example](http://websystique.com/spring/spring-4-jms-activemq-example-with-jmslistener-enablejms/)
* [Spring 4+JMS+ActiveMQ Integration Example](http://websystique.com/spring/spring-4-jms-activemq-example-with-annotations/)
* [Spring MVC 4+Spring Security 4 + Hibernate Integration Example](http://websystique.com/springmvc/spring-mvc-4-and-spring-security-4-integration-example/)
* [Spring MVC 4+AngularJS Example](http://websystique.com/springmvc/spring-mvc-4-angularjs-example/)
* [Spring MVC 4+AngularJS Server communication example : CRUD application using ngResource $resource service](http://websystique.com/springmvc/spring-4-mvc-angularjs-crud-application-using-ngresource/)
* [Spring MVC 4+AngularJS Routing with UI-Router Example](http://websystique.com/springmvc/spring-4-mvc-angularjs-routing-example-using-ui-router/)
* [Spring MVC 4+AngularJS Routing with ngRoute Example](http://websystique.com/springmvc/spring-4-mvc-angularjs-routing-example-using-ngroute/)
* [Spring MVC4 FileUpload-Download Hibernate+MySQL Example](http://websystique.com/springmvc/spring-mvc-4-fileupload-download-hibernate-example/)
* [Spring MVC 4 File Upload Example using Commons fileupload](http://websystique.com/springmvc/spring-mvc-4-file-upload-example-using-commons-fileupload/)
* [Spring MVC 4 File Upload Example using Servlet 3 MultiPartConfigElement](http://websystique.com/springmvc/spring-mvc-4-file-upload-example-using-multipartconfigelement/)

**Following technologies being used:**

* Spring 4.3.0.RELEASE
* Spring JMS 4.3.0.RELEASE
* ActiveMQ 5.13.3
* Maven 3
* JDK 1.7
* Tomcat 8.0.21
* Eclipse MARS.1 Release 4.5.1
* logback 1.1.7

**Application Overview**

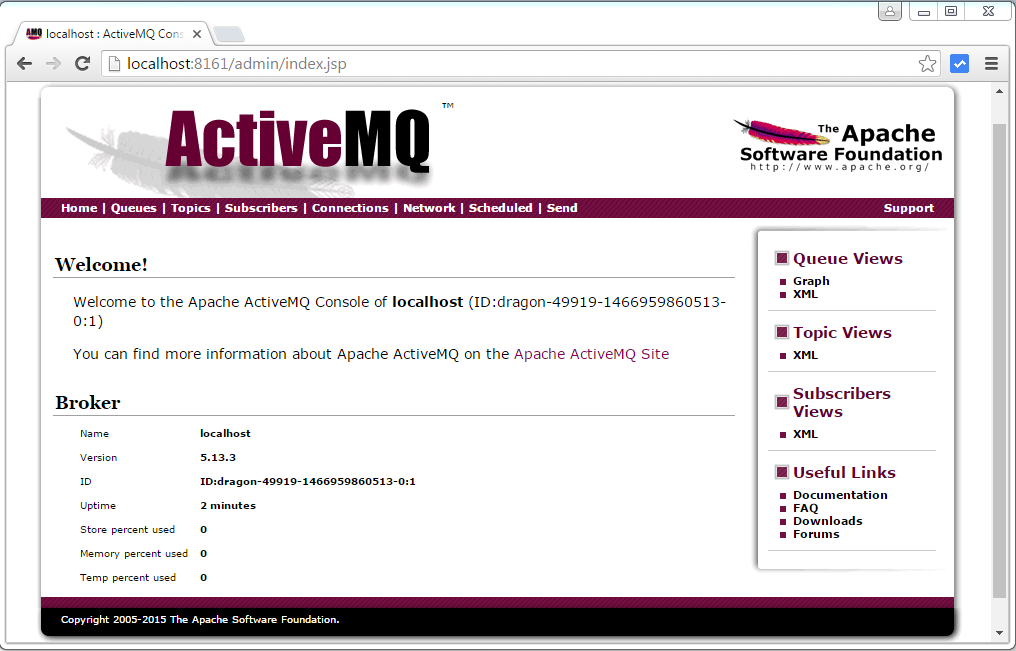
We have two Spring based java-applications. First one is a Spring MVC application [A Web-shop] where you can buy a product online. Once you have placed your order [means order status is 'created'], Web-shop application sends this order to an Inventory application [which is a Spring MVC based application, deployed as war] using JMS queue [named as 'order-queue'] via ActiveMQ Message broker, and configures a Listener on response queue [named as 'order-response-queue'] to get the order confirmation from Inventory application. Inventory application, which was listing on order queue ['order-queue'], gets the order, and process it. It then sends the confirmation on the response queue ['order-response-queue']. On receiving the order response, the Web-shop updates the order status in it’s repository.

**Start ActiveMQ Message Broker**

Download [ActiveMQ](http://activemq.apache.org/) Message Broker, unzip it, goto bin directory and start it using **$ ./activemq start**.

I installed

DEFAULT\_ACTIVEMQ\_HOME=C:\java\apache-activemq-5.9.0

You can quickly check the WebConsole [available at http://localhost:8161/admin/ with credentials admin/admin.  


## Integration with Spring Framework

1. To integrate ActiveMQ with your Spring application, add the below dependency to your pom.xml file.  
      
   pom.xml

|  |  |  |
| --- | --- | --- |
| 1 | <!-- Include ActiveMQ --> | |
| 2 | <dependency> |

|  |  |
| --- | --- |
| 3 | <groupId>org.apache.activemq</groupId> |
| 4 | <artifactId>activemq-core</artifactId> |

|  |  |  |
| --- | --- | --- |
| 5 | <version>${activemq.version}</version> | |
| 6 | </dependency> |

1. Dont forget to specify the proper version of ActiveMQ.
2. Specify the connectionFactory bean into the your application context file.  
      
   jmsContext.xml

|  |  |  |
| --- | --- | --- |
| 1 | <bean id="connectionFactory" class="org.apache.activemq.spring.ActiveMQConnectionFactory"> | |
| 2 | <property name="brokerURL" value="<tcp://localhost:61616>" /> |

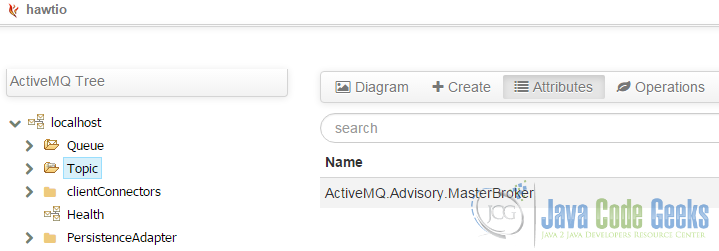
|  |  |
| --- | --- |
| 3 | </bean> |

1. We can now make use of this connectionFactory bean reference to specify jmsTemplate bean or the JMS listener into our application context file. Read more on creating [sender application](http://www.javacodegeeks.com/2015/10/jms-sender-application-with-activemq-and-maven.html) and [receiver application](http://www.jcombat.com/jms-tutorials/jms-receiver-application-with-activemq-and-maven) with ActiveMQ.

## 5. Monitoring a queue with ActiveMQ Hawtio admin console

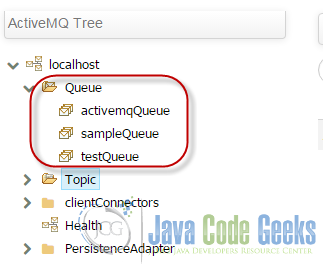
With ActiveMQ configured, we can monitor some important metrics through the ActiveMQ admin console.

1. We just need to go to the admin dashboard at URL http://localhost:8161/hawtio/, where we can see the left sidebar that lists out various categories like Queue, Topic, etc.
2. Username / password is admin/admin

[](http://examples.javacodegeeks.com/wp-content/uploads/2015/12/Hawtio-admin-dashboard2.png)

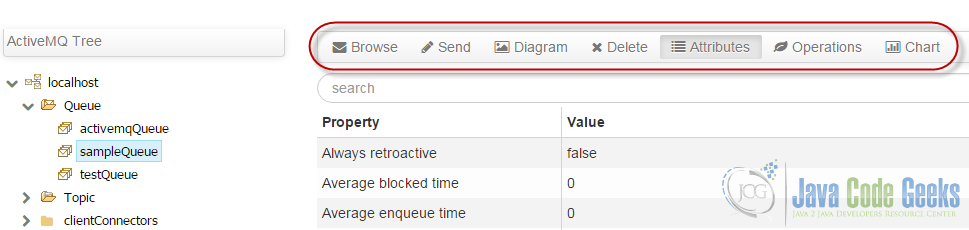
Hawtio admin dashboard

1. Expand the Queue link to check out all the available queues.

[](http://examples.javacodegeeks.com/wp-content/uploads/2015/12/Available-ActiveMQ-queues1.png)

Available ActiveMQ queues

1. Click on the queue that you want to monitor. All the details pertaining to that particular queue shows up on the main content area of the page. It also has several other links such as Browse, Send, Queue Diagram, Delete and so on, that further adds the provision to browse the current messages on the queue, drop/send any message to the queue or even completely delete the queue.

[](http://examples.javacodegeeks.com/wp-content/uploads/2015/12/Possible-operations-on-a-ActiveMQ-queue1.png)

Possible operations on a ActiveMQ queue

## 5. Conclusion

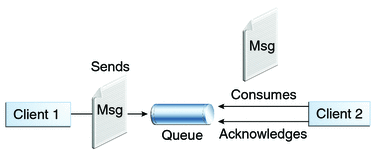
This is how we can configure ActiveMQ, integrate it with our Spring application and use the JMS API to efficiently communicate Messages asynchronously via configured ActiveMQ queues across different systems. We also explored on how we can monitor the ActiveMQ queues at real time with the help of ActiveMQ Hawtio admin dashboard, which indeed proves helpful in analysis and debugging purposes.

Now, let’s start with Web-shop application.

**Messaging Configuration**

1. ConnectionFactory : In order to connect to a Message broker (and eventually able to send receive messages), we need to configure a [ConnectionFactory](http://docs.oracle.com/javaee/6/api/javax/jms/ConnectionFactory.html). [ActiveMQConnectionFactory](http://activemq.apache.org/maven/apidocs/org/apache/activemq/ActiveMQConnectionFactory.html) is the ConnectionFactory implementation from Apache.

2. Destination : Each JMS message sent by an application is addressed with a [Destination](https://docs.oracle.com/cd/E19575-01/819-3669/bncel/index.html). Destinations in JMS are like postal mailboxes where the messages are placed until someone comes to pick them up. There are two types of destination in JMS: queue and topic.

* **Queues [point-to-point]**: Queue’s are based on point-to-point messaging model where messages are sent to a queue. Each message has exactly one sender and one receiver. Message is guaranteed to be delivered to only one receiver.  
    
  *JMS Queue destination illustration. Source:*[*Oracle*](http://docs.oracle.com/javaee/6/tutorial/doc/bnceh.html)
* **Topics [publish-subscribe]**:Topic’s are based on publish-subscribe model where messages are sent to a topic. N subscribers can be subscribed on a topic, and when a message arrives, each will get a copy of that message.  
    
  *JMS Topic destination illustration. Source:*[*Oracle*](http://docs.oracle.com/javaee/6/tutorial/doc/bnceh.html)

|  |
| --- |
| package com.websystique.springmvc.configuration;    import java.util.Arrays;    import org.apache.activemq.spring.ActiveMQConnectionFactory;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.jms.core.JmsTemplate;    @Configuration  public class MessagingConfiguration {        private static final String DEFAULT\_BROKER\_URL = "<tcp://localhost:61616>";        private static final String ORDER\_QUEUE = "order-queue";        @Bean      public ActiveMQConnectionFactory connectionFactory(){          ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory();          connectionFactory.setBrokerURL(DEFAULT\_BROKER\_URL);          connectionFactory.setTrustedPackages(Arrays.asList("com.websystique.springmvc"));          return connectionFactory;      }        @Bean      public JmsTemplate jmsTemplate(){          JmsTemplate template = new JmsTemplate();          template.setConnectionFactory(connectionFactory());          template.setDefaultDestinationName(ORDER\_QUEUE);          return template;      }    } |

Above configurations sets up a connection factory used to connect to Message broker itself.

3. JmsTemplate : Additionally, we have configured a [JmsTemplate](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jms/core/JmsTemplate.html) which provides an abstraction , hiding all the complexities of JMS communication. Without JmsTemplate, you will be forced to create connections/sessions/MessageProducers/MessageConsumers and catch all the nasty exception platform may throw. With JmsTemplate ,you get simple API’s to work with , and spring behind-the-scenes take care of all the JMS complexities. It takes care of creating a connection, obtaining a session, and finally sending [as well as synchronous receiving] of message. We will be using JmsTemplate for sending the message. Do note that JmsTemplate also provides possibilities for receiving message but that is synchronous[blocks the listening application], and usually not preferred when asynchronous communication is possible.

4. Message Listeners : We still need to configure the setup for Message listening on a destination. We could have used Standard javax.jms.MessageListener interface to do that, but spring provides a possibility to configure listeners using plain POJOs, without implementing an interface. To use that:

* 1. Annotate a POJO method with Spring’s very own [@JmsListener](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jms/annotation/JmsListener.html).
* 2. Configure a message-listener-container [ with [JmsListenerContainerFactory](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jms/config/JmsListenerContainerFactory.html)] : which listens on a destination [can be the one used with @JmsListener] and when any message arrives on this destination, it retrieved that message and passes to the bean annotated with @JmsListener for that destination.
* 3. Use [@EnableJms](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jms/annotation/EnableJms.html) which enables detection of JmsListener annotations on any Spring-managed bean in the container.

|  |
| --- |
| package com.websystique.springmvc.configuration;    import javax.jms.ConnectionFactory;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.jms.annotation.EnableJms;  import org.springframework.jms.config.DefaultJmsListenerContainerFactory;    @Configuration  @EnableJms  public class MessagingListnerConfiguration {        @Autowired      ConnectionFactory connectionFactory;        @Bean      public DefaultJmsListenerContainerFactory jmsListenerContainerFactory() {          DefaultJmsListenerContainerFactory factory = new DefaultJmsListenerContainerFactory();          factory.setConnectionFactory(connectionFactory);          factory.setConcurrency("1-1");          return factory;      }    } |

[DefaultJmsListenerContainerFactory](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jms/config/DefaultJmsListenerContainerFactory.html) is a JmsListenerContainerFactory implementation to build a regular DefaultMessageListenerContainer. You can configure several properties. At the very least, it needs a connection factory. Additionally, we have specified the concurrency [max number of concurrent users/consumers] using setConcurrency(“lowwe-upper”). You can also use setConcurrency(“upper”) which means lower will be 1.

Shown below is the @JmsListener bean.

|  |
| --- |
| package com.websystique.springmvc.messaging;    import javax.jms.JMSException;    import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.jms.annotation.JmsListener;  import org.springframework.messaging.Message;  import org.springframework.messaging.MessageHeaders;  import org.springframework.stereotype.Component;    import com.websystique.springmvc.model.InventoryResponse;  import com.websystique.springmvc.service.OrderService;    @Component  public class MessageReceiver {      static final Logger LOG = LoggerFactory.getLogger(MessageReceiver.class);        private static final String ORDER\_RESPONSE\_QUEUE = "order-response-queue";        @Autowired      OrderService orderService;          @JmsListener(destination = ORDER\_RESPONSE\_QUEUE)      public void receiveMessage(final Message<InventoryResponse> message) throws JMSException {          LOG.info("+++++++++++++++++++++++++++++++++++++++++++++++++++++");          MessageHeaders headers =  message.getHeaders();          LOG.info("Application : headers received : {}", headers);            InventoryResponse response = message.getPayload();          LOG.info("Application : response received : {}",response);            orderService.updateOrder(response);          LOG.info("+++++++++++++++++++++++++++++++++++++++++++++++++++++");      }  } |

And finally, a Service for sending the message using JmsTemplate.

|  |
| --- |
| package com.websystique.springmvc.messaging;    import javax.jms.JMSException;  import javax.jms.Message;  import javax.jms.ObjectMessage;  import javax.jms.Session;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.jms.core.JmsTemplate;  import org.springframework.jms.core.MessageCreator;  import org.springframework.stereotype.Component;    import com.websystique.springmvc.model.Order;    @Component  public class MessageSender {        @Autowired      JmsTemplate jmsTemplate;        public void sendMessage(final Order order) {            jmsTemplate.send(new MessageCreator(){                  @Override                  public Message createMessage(Session session) throws JMSException{                      ObjectMessage objectMessage = session.createObjectMessage(order);                      return objectMessage;                  }              });      }    } |

**Note : Messaging Configuration for Inventory Application is exactly same as this one, with queue’s being flipped: Order will be received on order-queue and response will be sent to order-response-queue.**

Rest of the code/setup shown here [For Web-shop Application] is plain spring MVC related stuff. you can find the complete code [for both applications] in the download section.

|  |
| --- |
| <project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"      xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/maven-v4_0_0.xsd>">      <modelVersion>4.0.0</modelVersion>      <groupId>com.websystique.springmvc</groupId>      <artifactId>Spring4MVCJmsActiveMQExample</artifactId>      <packaging>war</packaging>      <version>1.0.0</version>      <name>Spring4MVCJmsActiveMQExample Maven Webapp</name>        <properties>          <springframework.version>4.3.0.RELEASE</springframework.version>      </properties>        <dependencies>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-core</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-context</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-web</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-webmvc</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-jms</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.apache.activemq</groupId>              <artifactId>activemq-spring</artifactId>              <version>5.13.3</version>          </dependency>          <dependency>              <groupId>javax.validation</groupId>              <artifactId>validation-api</artifactId>              <version>1.1.0.Final</version>          </dependency>          <dependency>              <groupId>ch.qos.logback</groupId>              <artifactId>logback-classic</artifactId>              <version>1.1.7</version>          </dependency>          <!-- Servlet+JSP+JSTL -->          <dependency>              <groupId>javax.servlet</groupId>              <artifactId>javax.servlet-api</artifactId>              <version>3.1.0</version>          </dependency>          <dependency>              <groupId>javax.servlet.jsp</groupId>              <artifactId>javax.servlet.jsp-api</artifactId>              <version>2.3.1</version>          </dependency>          <dependency>              <groupId>javax.servlet</groupId>              <artifactId>jstl</artifactId>              <version>1.2</version>          </dependency>        </dependencies>        <build>          <pluginManagement>              <plugins>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-compiler-plugin</artifactId>                      <version>3.2</version>                      <configuration>                          <source>1.7</source>                          <target>1.7</target>                      </configuration>                  </plugin>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-war-plugin</artifactId>                      <version>2.4</version>                      <configuration>                          <warSourceDirectory>src/main/webapp</warSourceDirectory>                          <warName>Spring4MVCJmsActiveMQExample</warName>                          <failOnMissingWebXml>false</failOnMissingWebXml>                      </configuration>                  </plugin>              </plugins>          </pluginManagement>          <finalName>Spring4MVCJmsActiveMQExample</finalName>      </build>  </project> |
| package com.websystique.springmvc.controller;    import javax.validation.Valid;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.validation.BindingResult;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;    import com.websystique.springmvc.model.Order;  import com.websystique.springmvc.service.OrderService;    @Controller  public class AppController {        @Autowired      OrderService orderService;        @RequestMapping(value = { "/", "/home" }, method = RequestMethod.GET)      public String prepareProduct(ModelMap model) {          return "index";      }        @RequestMapping(value = { "/newOrder" }, method = RequestMethod.GET)      public String prepareOrder(ModelMap model) {          Order order = new Order();          model.addAttribute("order", order);          return "order";      }        @RequestMapping(value = { "/newOrder" }, method = RequestMethod.POST)      public String sendOrder(@Valid Order order, BindingResult result,              ModelMap model) {          if (result.hasErrors()) {              return "order";          }          orderService.sendOrder(order);          model.addAttribute("success", "Order for " + order.getProductName() + " registered.");          return "ordersuccess";      }        @RequestMapping(value = { "/checkStatus" }, method = RequestMethod.GET)      public String checkOrderStatus(ModelMap model) {          model.addAttribute("orders", orderService.getAllOrders());          return "orderStatus";      }  } |

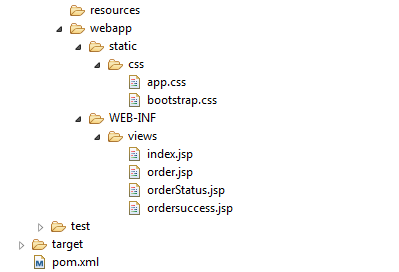
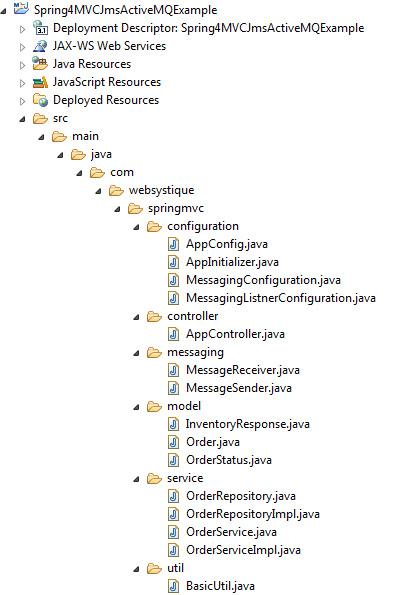
|  |
| --- |
| package com.websystique.springmvc.configuration;    import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.Import;  import org.springframework.web.servlet.config.annotation.EnableWebMvc;  import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;  import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;  import org.springframework.web.servlet.view.InternalResourceViewResolver;  import org.springframework.web.servlet.view.JstlView;    @Configuration  @ComponentScan(basePackages = "com.websystique.springmvc")  @Import({MessagingConfiguration.class,MessagingListnerConfiguration.class})  @EnableWebMvc  public class AppConfig extends WebMvcConfigurerAdapter{        @Override      public void configureViewResolvers(ViewResolverRegistry registry) {            InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();          viewResolver.setViewClass(JstlView.class);          viewResolver.setPrefix("/WEB-INF/views/");          viewResolver.setSuffix(".jsp");          registry.viewResolver(viewResolver);      }        /\*       \* Configure ResourceHandlers to serve static resources like CSS/ Javascript etc...       \*       \*/      @Override      public void addResourceHandlers(ResourceHandlerRegistry registry) {          registry.addResourceHandler("/static/\*\*").addResourceLocations("/static/");      }    } |
| package com.websystique.springmvc.service;    import java.util.Map;    import com.websystique.springmvc.model.InventoryResponse;  import com.websystique.springmvc.model.Order;    public interface OrderService {      public void sendOrder(Order order);        public void updateOrder(InventoryResponse response);        public Map<String, Order> getAllOrders();  } |

|  |
| --- |
| package com.websystique.springmvc.service;    import java.util.Map;    import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;    import com.websystique.springmvc.messaging.MessageSender;  import com.websystique.springmvc.model.InventoryResponse;  import com.websystique.springmvc.model.Order;  import com.websystique.springmvc.model.OrderStatus;  import com.websystique.springmvc.util.BasicUtil;    @Service("orderService")  public class OrderServiceImpl implements OrderService{        static final Logger LOG = LoggerFactory.getLogger(OrderServiceImpl.class);        @Autowired      MessageSender messageSender;        @Autowired      OrderRepository orderRepository;        @Override      public void sendOrder(Order order) {          LOG.info("+++++++++++++++++++++++++++++++++++++++++++++++++++++");          order.setOrderId(BasicUtil.getUniqueId());          order.setStatus(OrderStatus.CREATED);          orderRepository.putOrder(order);          LOG.info("Application : sending order request {}", order);          messageSender.sendMessage(order);          LOG.info("+++++++++++++++++++++++++++++++++++++++++++++++++++++");      }        @Override      public void updateOrder(InventoryResponse response) {            Order order = orderRepository.getOrder(response.getOrderId());          if(response.getReturnCode()==200){              order.setStatus(OrderStatus.CONFIRMED);          }else if(response.getReturnCode()==300){              order.setStatus(OrderStatus.FAILED);          }else{              order.setStatus(OrderStatus.PENDING);          }          orderRepository.putOrder(order);      }        public Map<String, Order> getAllOrders(){          return orderRepository.getAllOrders();      }    } |
| package com.websystique.springmvc.service;    import java.util.Map;    import com.websystique.springmvc.model.Order;    public interface OrderRepository {        public void putOrder(Order order);        public Order getOrder(String orderId);        public Map<String, Order> getAllOrders();  } |

|  |
| --- |
| package com.websystique.springmvc.service;    import java.util.Map;  import java.util.concurrent.ConcurrentHashMap;    import org.springframework.stereotype.Service;    import com.websystique.springmvc.model.Order;    @Service("orderRepository")  public class OrderRepositoryImpl implements OrderRepository{        private final Map<String, Order> orders = new ConcurrentHashMap<String, Order>();        @Override      public void putOrder(Order order) {          orders.put(order.getOrderId(), order);      }        @Override      public Order getOrder(String orderId) {          return orders.get(orderId);      }        public Map<String, Order> getAllOrders(){          return orders;      }  } |
| package com.websystique.springmvc.model;    import java.io.Serializable;    public class Order implements Serializable {        private String orderId;        private String productName;        private int quantity;        private OrderStatus status;        public String getOrderId() {          return orderId;      }        public void setOrderId(String orderId) {          this.orderId = orderId;      }        public String getProductName() {          return productName;      }        public void setProductName(String productName) {          this.productName = productName;      }        public int getQuantity() {          return quantity;      }        public void setQuantity(int quantity) {          this.quantity = quantity;      }        public OrderStatus getStatus() {          return status;      }        public void setStatus(OrderStatus status) {          this.status = status;      }          @Override      public int hashCode() {          final int prime = 31;          int result = 1;          result = prime \* result + ((orderId == null) ? 0 : orderId.hashCode());          return result;      }        @Override      public boolean equals(Object obj) {          if (this == obj)              return true;          if (obj == null)              return false;          if (getClass() != obj.getClass())              return false;          Order other = (Order) obj;          if (orderId == null) {              if (other.orderId != null)                  return false;          } else if (!orderId.equals(other.orderId))              return false;          return true;      }        @Override      public String toString() {          return "Order [orderId=" + orderId + ", productName=" + productName + ", quantity=" + quantity + ", status="                  + status + "]";      }      } |

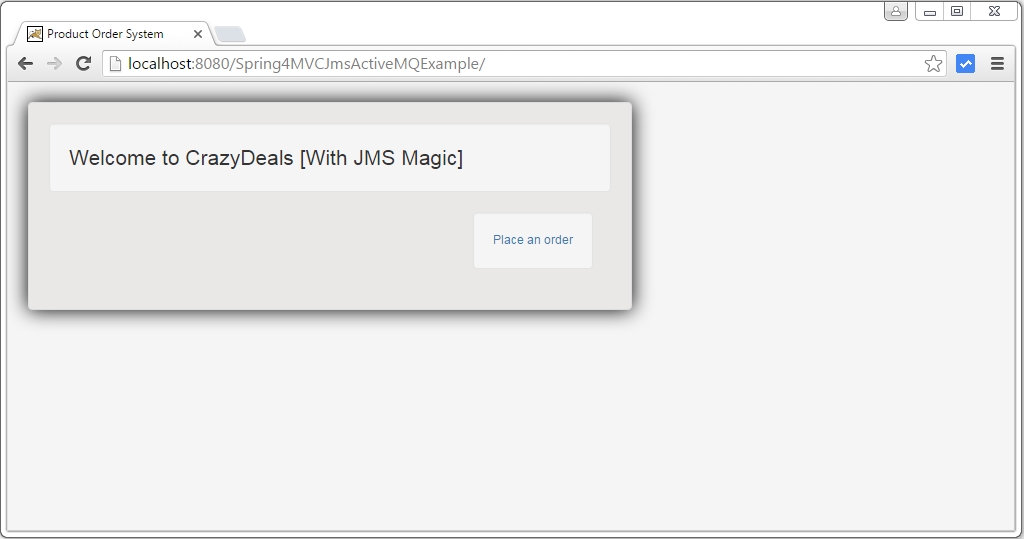
|  |
| --- |
| package com.websystique.springmvc.model;    public enum OrderStatus {        CREATED("Created"),      PENDING("Pending"),      CONFIRMED("Confirmed"),      FAILED("Failed");          private String status;        private OrderStatus(final String status){          this.status = status;      }        public String getStatus(){          return this.status;      }        @Override      public String toString(){          return this.status;      }          public String getName(){          return this.name();      }  } |
| package com.websystique.springmvc.model;    import java.io.Serializable;    public class InventoryResponse implements Serializable{        private String orderId;      private int returnCode;      private String comment;        public String getOrderId() {          return orderId;      }      public void setOrderId(String orderId) {          this.orderId = orderId;      }      public int getReturnCode() {          return returnCode;      }      public void setReturnCode(int returnCode) {          this.returnCode = returnCode;      }      public String getComment() {          return comment;      }      public void setComment(String comment) {          this.comment = comment;      }      @Override      public int hashCode() {          final int prime = 31;          int result = 1;          result = prime \* result + ((orderId == null) ? 0 : orderId.hashCode());          return result;      }      @Override      public boolean equals(Object obj) {          if (this == obj)              return true;          if (obj == null)              return false;          if (getClass() != obj.getClass())              return false;          InventoryResponse other = (InventoryResponse) obj;          if (orderId == null) {              if (other.orderId != null)                  return false;          } else if (!orderId.equals(other.orderId))              return false;          return true;      }      @Override      public String toString() {          return "InventoryResponse [orderId=" + orderId + ", returnCode=" + returnCode + ", comment=" + comment + "]";      }      } |

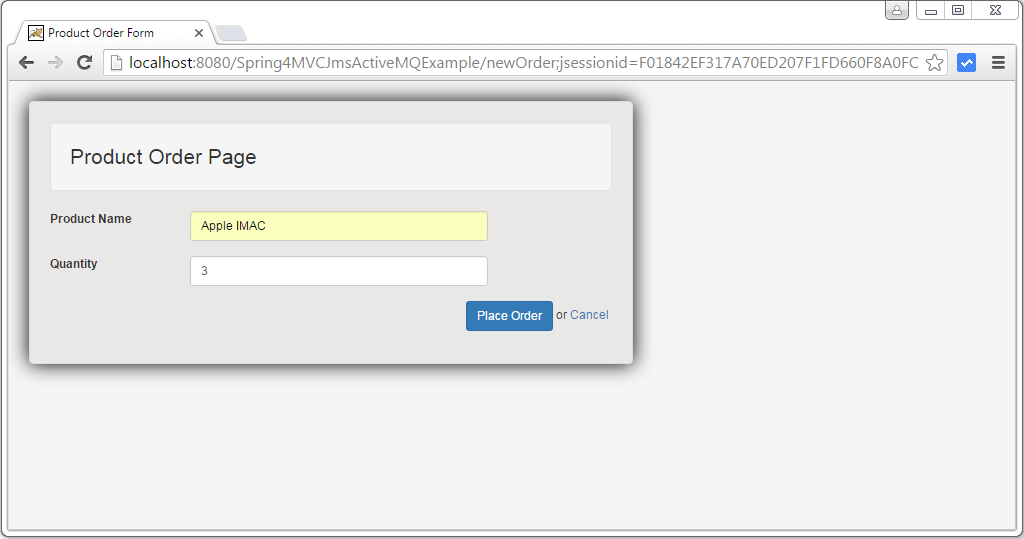
|  |
| --- |
| package com.websystique.springmvc.util;    import java.util.UUID;    public class BasicUtil {        public static String getUniqueId(){          return UUID.randomUUID().toString();      }  } |
| package com.websystique.springmvc.configuration;    import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;    public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {        @Override      protected Class<?>[] getRootConfigClasses() {          return new Class[] { AppConfig.class };      }        @Override      protected Class<?>[] getServletConfigClasses() {          return null;      }        @Override      protected String[] getServletMappings() {          return new String[] { "/" };      }    } |

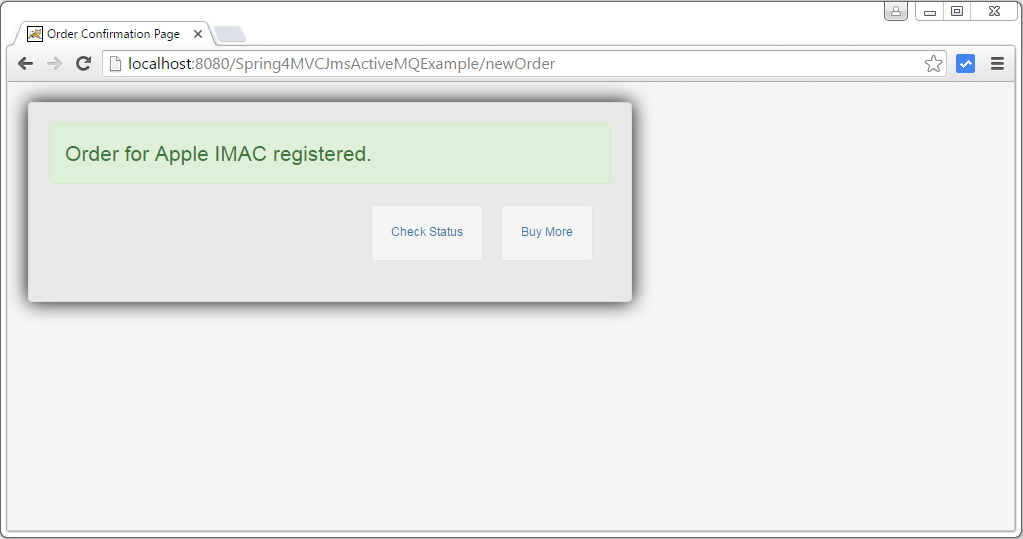
And Finally, the Project structure for Web-shop application is shown below.  
  
In order to limit the size of post, code for Inventory Application is not shown here, but available in download section.

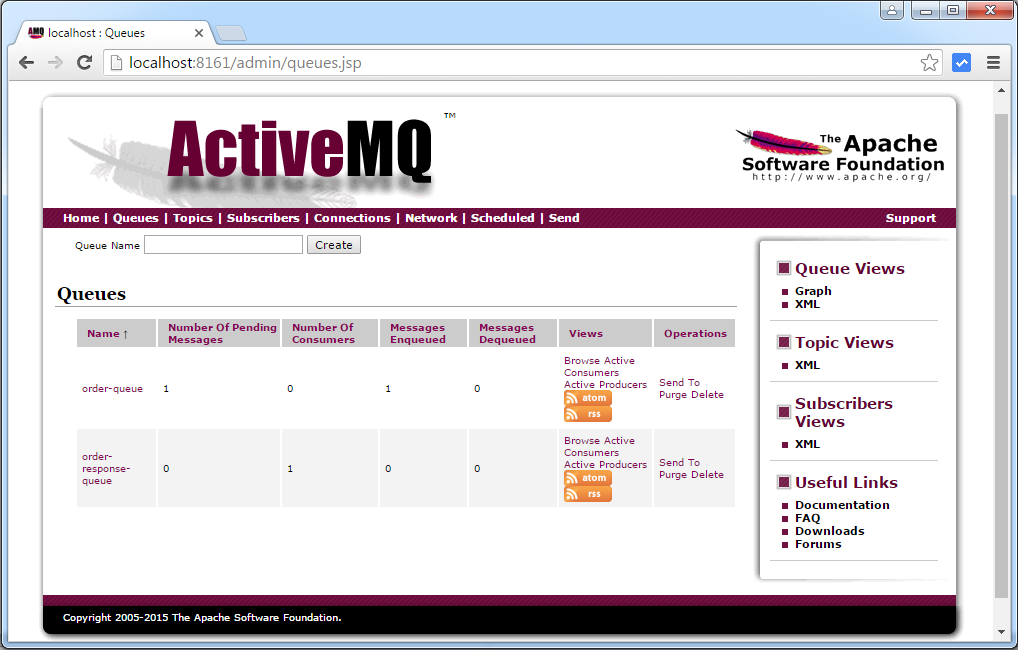
**Deploy & Run**

Build both the applications. Start tomcat and deploy only the Web-shop application for the time-being. Open the browser and visit to **http://localhost:8080/Spring4MVCJmsActiveMQExample/**.

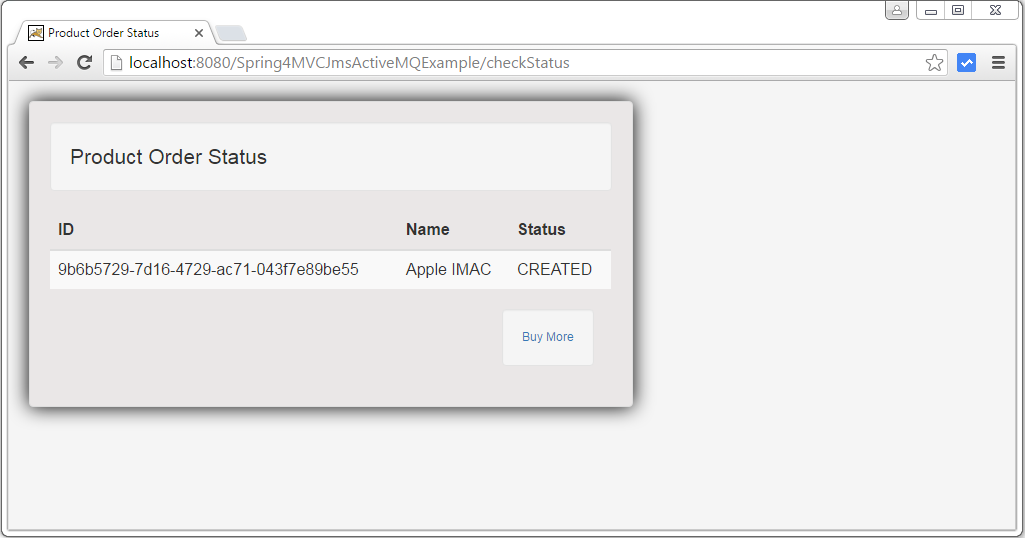


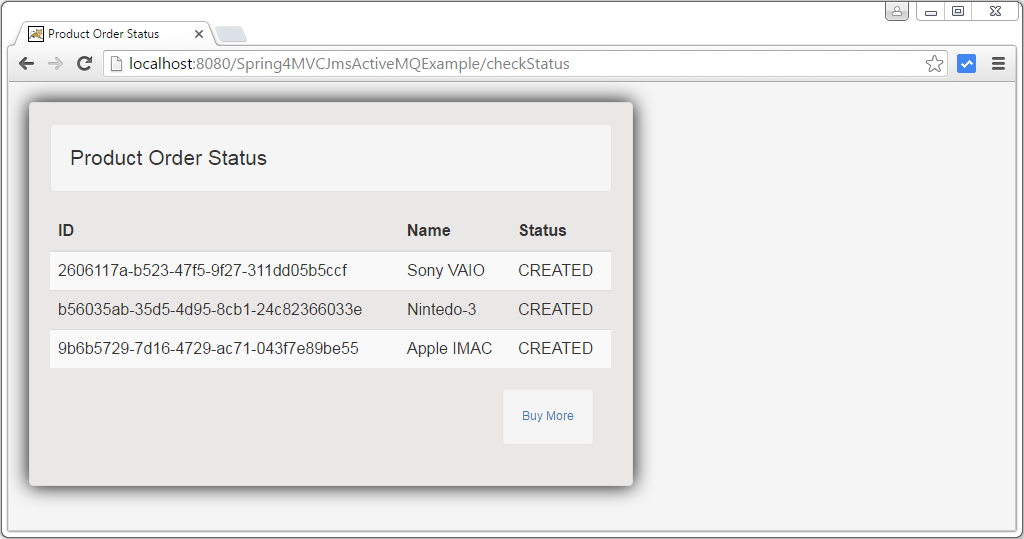


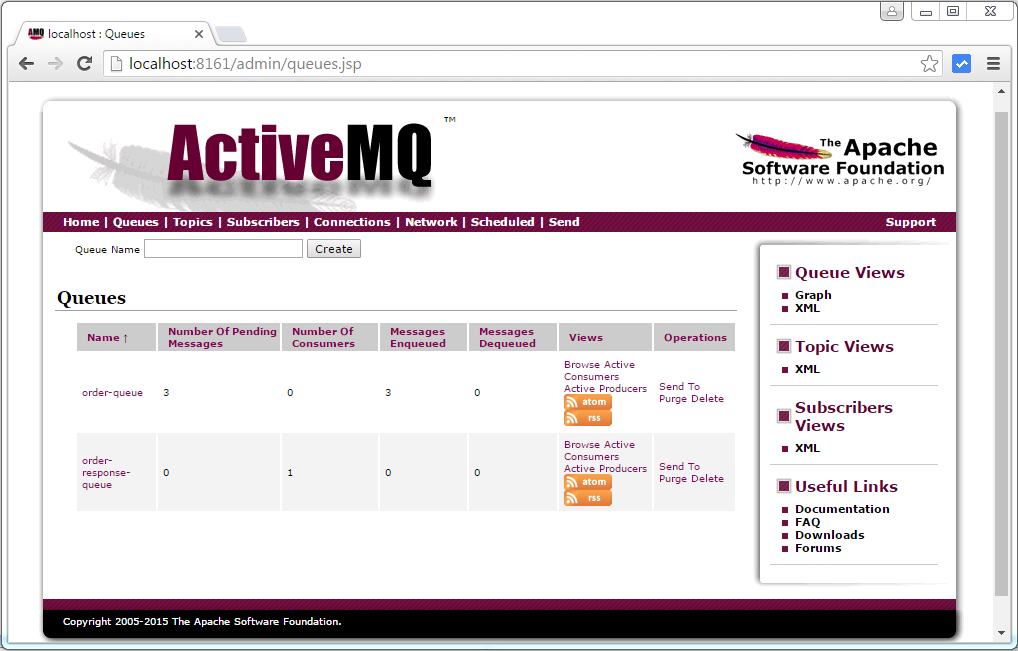


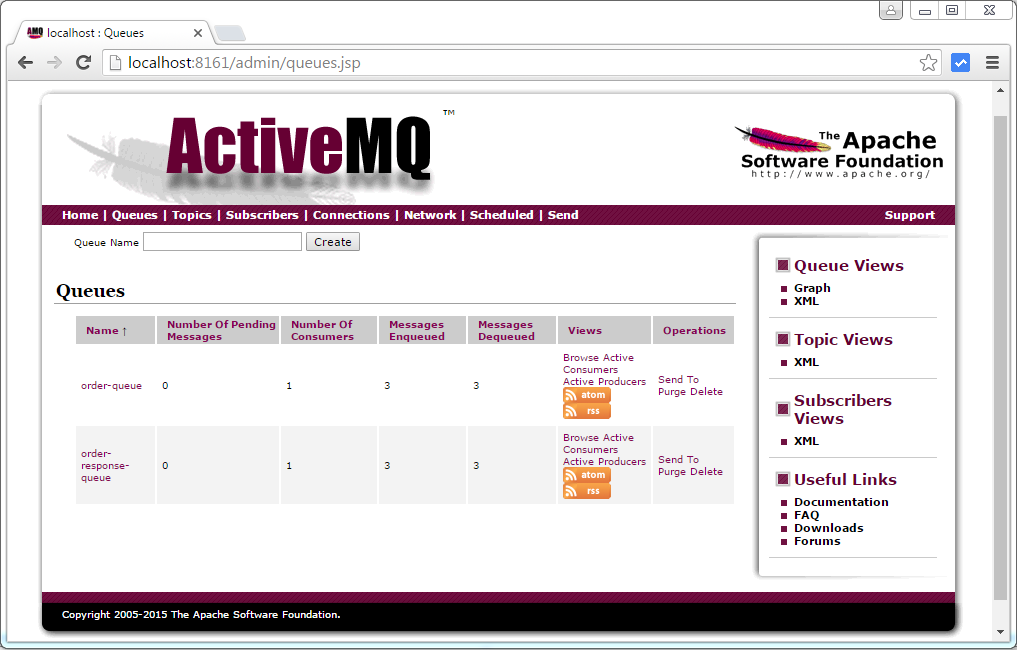
Now go to ActiveMQ webconsole and check the queues on http://localhost:8161/admin/queues.jsp. You will see both the queue’s and an enqueued message on order-queue. Also notice that web-shop application is rgistered as a consumer on order-response-queue [as it is listing on this queue].  


On our application side, you can check the order status. It is in created state [because it is not yet processed by Inventory application , we did not deploy it yet].

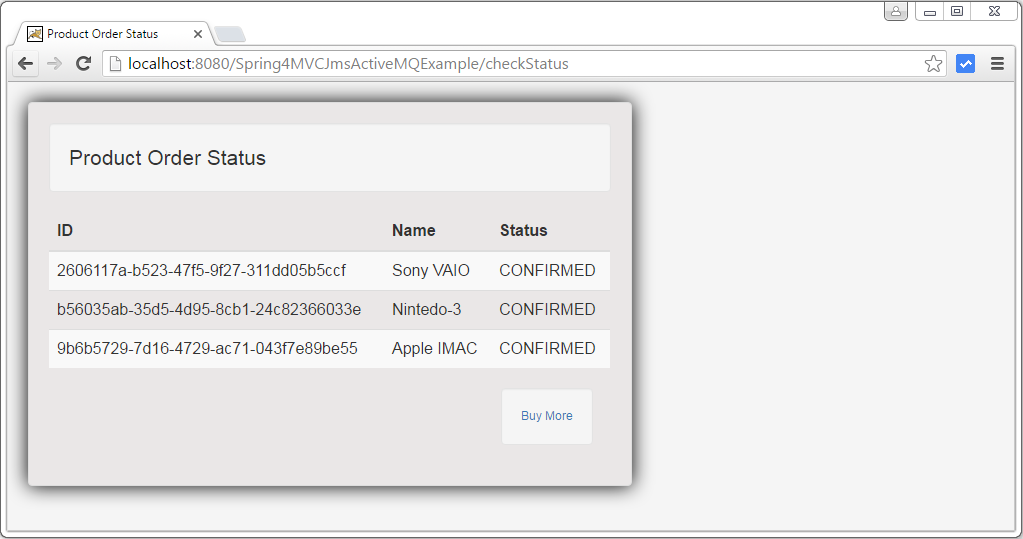
  
Now add few more order.



Check queue’s again.  


Now deploy the Inventory application in tomcat, and check the Active MQ console for queue’s again. All message should be dequeued [as Inventory applications was listening to order-queue, all message are dequede and processed, and response has been sent back to Web-shop application using order-response-queue.  


Check the Order status again on Web-shop application. All are confirmed.



That’s it. Download both the applications, and play with them locally to learn more. As we saw, SPRING JMS support is easy to use. Feel free to Comment, and suggest improvements.

***Download Source Code***

Web-shop Application :

[Download Now!](http://websystique.com/?smd_process_download=1&download_id=2567)

Inventory Application :

[Download Now!](http://websystique.com/?smd_process_download=1&download_id=2569)

**References**

* [Spring JMS Documentation](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/jms.html)
* [JMS reference Manual](http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html)
* [Spring 4 Reference Manual](http://docs.spring.io/spring/docs/4.3.0.RELEASE/spring-framework-reference/htmlsingle/)



[websystiqueadmin](http://websystique.com/author/blogadminuser/)

If you like tutorials on this site, why not take a step further and connect me on [Facebook](https://www.facebook.com/websystique) , [Google Plus](https://plus.google.com/+WebsystiqueDotCom/posts) & [Twitter](https://twitter.com/Websystique) as well? I would love to hear your thoughts on these articles, it will help me improve further our learning process.

If you appreciate the effort I have put in this learning site, help me improve the visibility of this site towards global audience by sharing and linking this site from within and beyond your network. You & your friends can always link my site from your site on [www.websystique.com](http://www.websystique.com), and share the learning.

After all, we are here to learn together, aren’t we?

**Related posts:**

1. [Spring 4+JMS+ActiveMQ Example with Annotations](http://websystique.com/spring/spring-4-jms-activemq-example-with-annotations/)
2. [Spring 4+JMS+ActiveMQ example with @JmsListener & @EnableJms](http://websystique.com/spring/spring-4-jms-activemq-example-with-jmslistener-enablejms/)
3. [Spring 4 Email Integration Tutorial](http://websystique.com/spring/spring-4-email-integration-tutorial/)
4. [Spring 4 MVC+Hibernate Many-to-many JSP Example with annotation](http://websystique.com/springmvc/springmvc-hibernate-many-to-many-example-annotation-using-join-table/)

[springmvc](http://websystique.com/category/springmvc/). [permalink](http://websystique.com/springmvc/spring-4-mvc-jms-activemq-annotation-based-example/).

**Post navigation**

[← Spring 4+JMS+ActiveMQ example with @JmsListener & @EnableJms](http://websystique.com/spring/spring-4-jms-activemq-example-with-jmslistener-enablejms/)

[Spring 4 Email Integration Tutorial →](http://websystique.com/spring/spring-4-email-integration-tutorial/)

**Recent Posts**

* [Spring Boot + AngularJS + Spring Data + JPA CRUD App Example](http://websystique.com/spring-boot/spring-boot-angularjs-spring-data-jpa-crud-app-example/)
* [Spring Boot Rest API Example](http://websystique.com/spring-boot/spring-boot-rest-api-example/)
* [Spring Boot WAR deployment example](http://websystique.com/spring-boot/spring-boot-war-deployment-example/)
* [Spring Boot Introduction + hello world example](http://websystique.com/spring-boot/spring-boot-introduction-hello-world-example/)
* [Secure Spring REST API using OAuth2](http://websystique.com/spring-security/secure-spring-rest-api-using-oauth2/)