**TiGERS**



Tiffany Goldmine Environmental Management Report System

**CSSE7024 Sample Code**

Wei Yeap Cheng (Nicholas),  
Jane Harrison,   
Xing-Shu Liu,   
Justin Mancinelli,   
Chih-Hsiang Tang (Sean) and Dell Topel



Tiffany Gold Mine Environmental Management Reporting System – Sample Code

[1 Sample Model 3](#_Toc274083813)

[2 Sample Dao 6](#_Toc274083814)

[2.1 Interface 6](#_Toc274083815)

[2.2 Implementation 6](#_Toc274083816)

[3 Sample Manager 8](#_Toc274083817)

[3.1 Interface 8](#_Toc274083818)

[3.2 Implementation 8](#_Toc274083819)

[4 Sample Form Controller 9](#_Toc274083820)

[5 Sample Form View 10](#_Toc274083821)

[6 Search Sample Controller 12](#_Toc274083822)

[7 Search Sample View 14](#_Toc274083823)

[8 Dispatcher Servlet 16](#_Toc274083824)

# Sample Model

The sample model is used for representing sample objects; it also defines the attributes, constraints and relationships of the sample table.

|  |
| --- |
| /\*\*  \* This class represents the "sample" object in TiGERS  \*  \* @author CSSE3004GC  \*/  @Entity  @Table(name="sample")  public class Sample extends BaseObject {  private Long id;  private Date date\_taken;  private BigDecimal ph;  private BigDecimal ec;  private BigDecimal temperature;  private BigDecimal collar\_depth;  private BigDecimal arsenic;  private BigDecimal grease;  private BigDecimal fluoride;  private BigDecimal chromium;  private User laboratory;  private Sampler sampler;    @Id @GeneratedValue(strategy=GenerationType.AUTO)  public Long getId() {  return id;  }  public void setId(Long id) {  this.id = id;  }    @Column(nullable=false)  public Date getDate\_taken() {  return date\_taken;  }  public void setDate\_taken(Date dateTaken) {  this.date\_taken = dateTaken;  }    @Column(precision=5, scale=2)  public BigDecimal getPh() {  return ph;  }  public void setPh(BigDecimal ph) {  this.ph = ph;  }  @Column(precision=5, scale=0)  public BigDecimal getEc() {  return ec;  }  public void setEc(BigDecimal ec) {  this.ec = ec;  }  @Column(precision=5, scale=1)  public BigDecimal getTemperature() {  return temperature;  }  public void setTemperature(BigDecimal temperature) {  this.temperature = temperature;  }  @Column(precision=5, scale=0)  public BigDecimal getCollar\_depth() {  return collar\_depth;  }  public void setCollar\_depth(BigDecimal collarDepth) {  collar\_depth = collarDepth;  }  @Column(precision=5, scale=3)  public BigDecimal getArsenic() {  return arsenic;  }  public void setArsenic(BigDecimal arsenic) {  this.arsenic = arsenic;  }  @Column(precision=3, scale=0)  public BigDecimal getGrease() {  return grease;  }  public void setGrease(BigDecimal grease) {  this.grease = grease;  }  @Column(precision=5, scale=3)  public BigDecimal getFluoride() {  return fluoride;  }  public void setFluoride(BigDecimal fluoride) {  this.fluoride = fluoride;  }  @Column(precision=5, scale=3)  public BigDecimal getChromium() {  return chromium;  }  public void setChromium(BigDecimal chromium) {  this.chromium = chromium;  }  @ManyToOne  @JoinColumn(name = "laboratory")  public User getLaboratory() {  return laboratory;  }  public void setLaboratory(User laboratory) {  this.laboratory = laboratory;  }  @ManyToOne  @JoinColumn(name = "sampler")  public Sampler getSampler() {  return sampler;  }  public void setSampler(Sampler sampler) {  this.sampler = sampler;  }    // converts a value to string, taking into consideration special formatting rules  // eg, null -> "NaN"; -0.05 -> "<0.05", 2.5 -> "2.5"  public static String getValueDisplayString(BigDecimal value) {  if(value == null) return "NaN";  if(value.compareTo(new BigDecimal(0)) < 0)  return "<" + value.abs().toString();  return value.toString();  }  // reverse of the above  public static BigDecimal getDisplayStringValue(String value) {  // not part of the spec, but just in case...  if(value == null) return null;    value = value.trim();  if(value.toLowerCase().equals("nan")) return null;  if(value.charAt(0) == '<')  return new BigDecimal(value.substring(1).trim()).negate();  return new BigDecimal(value);  }  // for easy use on JSP  @Transient public String getPhString() {return getValueDisplayString(ph); }  @Transient public String getEcString() {return getValueDisplayString(ec); }  @Transient public String getTemperatureString()  {return getValueDisplayString(temperature); }  @Transient public String getCollar\_depthString()  {return getValueDisplayString(collar\_depth); }  @Transient public String getArsenicString()  {return getValueDisplayString(arsenic); }  @Transient public String getGreaseString()  {return getValueDisplayString(grease); }  @Transient public String getFluorideString()  {return getValueDisplayString(fluoride); }  @Transient public String getChromiumString()  {return getValueDisplayString(chromium); }    @Override  public boolean equals(final Object other) {  if (!(other instanceof Sample))  return false;  Sample castOther = (Sample) other;  return new EqualsBuilder().append(date\_taken, castOther.date\_taken)  .append(ph, castOther.ph).append(ec,  castOther.ec).append(sampler,  castOther.sampler).isEquals();  }  @Override  public int hashCode() {  return new HashCodeBuilder(591165033, -1663524385).append(date\_taken)  .append(ph).append(ec).append(sampler).toHashCode();  }  @Override  public String toString() {  return new ToStringBuilder(this).append("id", id).append("date\_taken",  date\_taken).append("ph", ph).append("ec", ec)  .append("tag", sampler).toString();  }  } |

# Sample Dao

Sample Dao is a database access object used for handling communications between the back end and the database. It contains two parts, the interface and the implementation. The former one is used to define an interface for accessing sample the table for web developers and the latter one is the implementation of the interface. The reason for having an interface is we can have different implementations with the same interface, so, if the implementation is changed, other parts of the system that use the DAO (Database Access Object) do not need to change.

## Interface

|  |
| --- |
| public interface SampleDao extends GenericDao<Sample, Long>{  /\*\*  \* Gets Course information based on coursecode  \* @param coursecode the coursecode  \* @return populated course object  \*/  public List<Sample> findByDateRange(Date start, Date end);  public List<Sample> findSamplesByTagAndDateRange(String tag, Date start, Date end);  public List<Sample> findSamplesByTag(String tag);  public List<Sample> findSamplerId();  public List<Sample> findSamplesByLab(User lab);  public List<Sample> findSamplesByLabAndDateRange(User lab, Date from,  Date to);  public List<Sample> findSamplesByLabAndSamperIdAndDateRange(User lab, String sampelrId,  Date from, Date to);  } |

## Implementation

|  |
| --- |
| public class SampleDaoHibernate extends GenericDaoHibernate<Sample, Long>  implements SampleDao {  /\*\*  \* Constructor to create a Generics-based version using Sample as the entity  \*/  public SampleDaoHibernate() {  super(Sample.class);  }  public List<Sample> findByDateRange(Date start, Date end) {  if (start.compareTo(end) > 0) {  return null;  }  Object[] dates = { start, end };  List samples = getHibernateTemplate().find(  "from Sample where date\_taken between ? and ?", dates);  if (samples.isEmpty()) {  return null;  } else {  return samples;  }  }  public List<Sample> findSamplerId() {  List samples = getHibernateTemplate().find("from Sample");  if (samples.isEmpty()) {  return null;  } else {  return samples;  }  }  public List<Sample> findSamplesByTagAndDateRange(String tag, Date from,  Date to) {  Object[] tagAndDateRange = { tag, from, to };  return getHibernateTemplate().find(  "from Sample where sampler.tag=? and "  + "date\_taken between ? and ?", tagAndDateRange);  }  public List<Sample> findSamplesByTag(String tag) {  Object[] tagParam = { tag };  return getHibernateTemplate().find(  "from Sample where sampler.tag=? "  + "order by date\_taken desc", tagParam);  }  public List<Sample> findSamplesByLab(User lab) {  return getHibernateTemplate().find(  "from Sample where laboratory=?", lab);  }  public List<Sample> findSamplesByLabAndDateRange(User lab, Date from,  Date to) {  Object[] labAndDateRange = { lab, from, to };  return getHibernateTemplate().find(  "from Sample where laboratory=? and "  + "date\_taken between ? and ?", labAndDateRange);  }    public List<Sample> findSamplesByLabAndSamperIdAndDateRange(User lab, String samplerId,  Date from, Date to) {  Object[] labAndDateRange = { lab, samplerId, from, to };  return getHibernateTemplate().find(  "from Sample where laboratory=? and sampler.tag=? and "  + "date\_taken between ? and ?", labAndDateRange);  }  } |

# Sample Manager

Sample Manager is used to handle all the functions involved in sample objects. The functions are not necessarily a pure database query (which is different from the DAO), e.g., it can perform some processes before or after accessing the database. The manager also contains an interface and an implementation for the same reason as the DAO.

## Interface

|  |
| --- |
| @WebService  public interface SampleManager extends GenericManager<Sample, Long> {  List<Sample> findSampleByDateRange(Date start, Date end);  List<Sample> findSamplerId();  List<Sample> findSamplesByTagAndDateRange(String tag, Date from, Date to);  List<Sample> findSamplesByTag(String tag);  List<Sample> findSamplesByLab(User lab);  List<Sample> findSamplesByLabAndDateRange(User lab, Date from, Date to);  List<Sample> findSamplesByLabAndSamplerAndDateRange(User lab, String samplerId,  Date from, Date to);  List<String> getMySamplerIdList(User lab);  } |

## Implementation

|  |
| --- |
| @WebService(serviceName = "SampleService", endpointInterface = "com.tiffany.service.SampleManager")  public class SampleManagerImpl extends GenericManagerImpl<Sample, Long>  implements SampleManager{  SampleDao sampleDao;    public SampleManagerImpl(SampleDao sampleDao) {  super(sampleDao);  this.sampleDao = sampleDao;  }    public List<Sample> findSampleByDateRange(Date start, Date end) {  List<Sample> samples = sampleDao.findByDateRange(start, end);  return samples;  }    public List<Sample> findSamplerId() {  List<Sample> samples = sampleDao.findSamplerId();  return samples;  }    public List<Sample> findSamplesByTagAndDateRange(String tag, Date from, Date to) {  return sampleDao.findSamplesByTagAndDateRange(tag, from, to);  }    public List<Sample> findSamplesByTag(String tag) {  return sampleDao.findSamplesByTag(tag);  }  public List<Sample> findSamplesByLab(User lab) {  return sampleDao.findSamplesByLab(lab);  }  public List<Sample> findSamplesByLabAndDateRange(User lab, Date from,  Date to) {  return sampleDao.findSamplesByLabAndDateRange(lab, from, to);  }    public List<Sample> findSamplesByLabAndSamplerAndDateRange(User lab, String  samplerId, Date from, Date to) {  return sampleDao.findSamplesByLabAndSamperIdAndDateRange(lab,  samplerId, from, to);  }    public List<String> getMySamplerIdList(User lab) {  List<Sample> sampleList = findSamplesByLab(lab);  Set<String> samplerIdList = new HashSet<String>();  for (Sample sample : sampleList) {  samplerIdList.add(sample.getSampler().getTag());  }  List<String> ids = new ArrayList<String>();  ids.addAll(samplerIdList);  Collections.sort(ids);  return ids;  }  } |

# Sample Form Controller

The controller controls the data and the views that a user requests. For different people, we will have different data and, for different requests, we will have a different view.

|  |
| --- |
| public class SampleFormController extends BaseFormController {  private SampleManager sampleManager = null;  private SamplerManager samplerManager = null;  private UserManager userManager = null;  public void setSampleManager(SampleManager sampleManager) {  this.sampleManager = sampleManager;  }  public void setSamplerManager(SamplerManager samplerManager) {  this.samplerManager = samplerManager;  }  public void setUserManager(UserManager userManager) {  this.userManager = userManager;  }  public SampleFormController() {  setCommandClass(Sample.class);  setCommandName("sample");  }  protected Object formBackingObject(HttpServletRequest request)  throws Exception {  String id = request.getParameter("id");  if (!StringUtils.isBlank(id)) {  return sampleManager.get(new Long(id));  }    return new Sample();  }    protected Map referenceData(HttpServletRequest request, Object command,  Errors errors) throws Exception {    Map referenceData = new HashMap();  referenceData.put("samplerList", samplerManager.getAll());  referenceData.put("laboratoryList", userManager.getLaboratories());    return referenceData;  }  public ModelAndView onSubmit(HttpServletRequest request,  HttpServletResponse response, Object command,  BindException errors)  throws Exception {  log.debug("entering 'onSubmit' method...");  Sample sample = (Sample) command;  boolean isNew = (sample.getId() == null);  String success = getSuccessView();  Locale locale = request.getLocale();    if (request.getParameter("delete") != null) {  sampleManager.remove(sample.getId());  saveMessage(request, getText("sample.deleted", locale));  } else {  sample = sampleManager.save(sample);  String key = (isNew) ? "sample.added" : "sample.updated";  saveMessage(request, getText(key, sample.getId().toString(), locale));  if (!isNew) {  success = "redirect:samples.html?id=" + sample.getId();  } else {  success = "redirect:samples.html?id=" + sample.getId();  }  }  return new ModelAndView(success);  }  } |

# Sample Form View

This is the view for displaying fields for users to insert or update sample data:

|  |
| --- |
| <%@ include file=*"/common/taglibs.jsp"*%>  <head>  <title><fmt:message key=*"sampleList.title"*/></title>  <meta name=*"heading"* content=*"*<fmt:message key=*'sampleList.heading'*/>*"*/>  </head>  <form:form commandName=*"sample"* method=*"post"* action=*"sampleform.html"* id=*"sampleForm"*>  <form:errors path=*"\*"* cssClass=*"error"* element=*"div"*/>  <form:hidden path=*"id"*/>  <ul>  <li>  <appfuse:label styleClass=*"desc"* key=*"sample.date\_taken"*/>  <form:errors path=*"date\_taken"* cssClass=*"fieldError"*/>  <form:input path=*"date\_taken"* id=*"date\_taken"* disabled=*"true"*  cssClass=*"text medium"*/>  <!-- <button id="dateButton" type="button" class="button"> ... </button> -->  </li>  <li>  <appfuse:label styleClass=*"desc"* key=*"sample.ph"*/>  <form:errors path=*"ph"* cssClass=*"fieldError"*/>  <form:input path=*"ph"* id=*"ph"* cssClass=*"text medium"*/>  </li>    <li>  <appfuse:label styleClass=*"desc"* key=*"sample.ec"*/>  <form:errors path=*"ec"* cssClass=*"fieldError"*/>  <form:input path=*"ec"* id=*"ec"* cssClass=*"text medium"*/>  </li>    <li>  <appfuse:label styleClass=*"desc"* key=*"sample.temperature"*/>  <form:errors path=*"temperature"* cssClass=*"fieldError"*/>  <form:input path=*"temperature"* id=*"temperature"* cssClass=*"text medium"*/>  </li>    <li>  <appfuse:label styleClass=*"desc"* key=*"sample.collar\_depth"*/>  <form:errors path=*"collar\_depth"* cssClass=*"fieldError"*/>  <form:input path=*"collar\_depth"* id=*"collar\_depth"* cssClass=*"text medium"*/>  </li>    <li>  <appfuse:label styleClass=*"desc"* key=*"sample.arsenic"*/>  <form:errors path=*"arsenic"* cssClass=*"fieldError"*/>  <form:input path=*"arsenic"* id=*"arsenic"* cssClass=*"text medium"*/>  </li>    <li>  <appfuse:label styleClass=*"desc"* key=*"sample.grease"*/>  <form:errors path=*"grease"* cssClass=*"fieldError"*/>  <form:input path=*"grease"* id=*"grease"* cssClass=*"text medium"*/>  </li>    <li>  <appfuse:label styleClass=*"desc"* key=*"sample.fluoride"*/>  <form:errors path=*"fluoride"* cssClass=*"fieldError"*/>  <form:input path=*"fluoride"* id=*"fluoride"* cssClass=*"text medium"*/>  </li>    <li>  <appfuse:label styleClass=*"desc"* key=*"sample.chromium"*/>  <form:errors path=*"chromium"* cssClass=*"fieldError"*/>  <form:input path=*"chromium"* id=*"chromium"* cssClass=*"text medium"*/>  </li>    <li>  <appfuse:label styleClass=*"desc"* key=*"sample.username"*/>  <form:errors path=*"laboratory"* cssClass=*"fieldError"*/>  <form:select path=*"sampler.laboratory"* id=*"laboratory"* disabled=*"true"*  cssClass=*"text medium"*>  <form:option value=*""* label=*"Select"* />  <form:options items=*"*${laboratoryList}*"* itemValue=*"id"* itemLabel=*"companyName"*/>  </form:select>  </li>    <li>  <appfuse:label styleClass=*"desc"* key=*"sample.tag"*/>  <form:errors path=*"sampler"* cssClass=*"fieldError"*/>  <form:select path=*"sampler.id"* id=*"sampler"* disabled=*"true"* cssClass=*"text medium"*>  <form:option value=*""* label=*"Select"* />  <form:options items=*"*${samplerList}*"* itemValue=*"id"* itemLabel=*"tag"*/>  </form:select>  </li>    <li class=*"buttonBar bottom"*>  <input type=*"submit"* class=*"button"* name=*"save"* value=*"*<fmt:message  key=*"button.save"*/>*"*/>  <c:if test=*"*${**not empty** sample.id}*"*>  <input type=*"submit"* class=*"button"* name=*"delete"* onclick="return  confirmDelete('sample')"  value=*"*<fmt:message key=*"button.delete"*/>*"* />  </c:if>  <input type=*"submit"* class=*"button"* name=*"cancel"* value=*"*<fmt:message  key=*"button.cancel"*/>*"*/>  </li>  </ul>  </form:form>  <script type=*"text/javascript"*>  Form.focusFirstElement($('sampleForm'));    Calendar.setup(  {  inputField : "date\_taken", // id of the input field  //dateFormat : "%d/%m/%Y %H:%M", // the date format  trigger : "dateButton", // id of the button  showTime: 12,  onTimeChange : updateFields  }  );  **function** updateFields(cal) {  **var** hour=cal.getHours();  **var** min=cal.getMinutes();  date=document.getElementById("date\_taken").value.split(" ")[0];  document.getElementById("date\_taken").value=date+" "+hour+":"+min;  };  </script> |

# Search Sample Controller

This controller is used to handle viewing samples requests. The controller allows users to specify a date range and a sampler ID for searching samples.

|  |
| --- |
| **public** **class** SampleSearchController **extends** BaseFormController {  **private** SampleManager sampleManager = **null**;  **private** SamplerManager samplerManager = **null**;  **private** UserManager userManager = **null**;    **public** SampleSearchController() {  setCommandClass(SampleSearch.**class**);  setCommandName("search");  }    **public** **void** setSampleManager(SampleManager sampleManager) {  **this**.sampleManager = sampleManager;  }  **public** **void** setSamplerManager(SamplerManager samplerManager) {  **this**.samplerManager = samplerManager;  }  **public** **void** setUserManager(UserManager userManager) {  **this**.userManager = userManager;  }    //================= formBackingObject =======================================  **protected** Object formBackingObject(HttpServletRequest request) **throws** Exception {  log.debug("entering \"formBackingObject\" method ...");  SampleSearch search = **new** SampleSearch();  **return** search;  }    //======================= referenceData ====================================  **protected** Map referenceData(HttpServletRequest request, Object command,  Errors errors) **throws** Exception {  log.debug("entering \"referenceData\" method in SampleSearchController...");  Map<String, Object> referenceData = **new** HashMap();    //========= sampler list ==========  List<String> samplerIdList = **new** ArrayList<String>();  User remoteUser = userManager.getUserByUsername(request.getRemoteUser());  List<String> samplerList = sampleManager.getMySamplerIdList(remoteUser);  **if** (samplerList.size() != 0) {  samplerIdList.add("ALL");  samplerIdList.addAll(samplerList);  } **else** {  saveError(request, "No sampler available");  }  List<Sample> sampleList = sampleManager.findSamplesByLab(remoteUser);  //======================================  referenceData.put("samplerIdList", samplerIdList);  referenceData.put("sampleList", sampleList);  **return** referenceData;  }    //====================== onSubmit ========================  **public** ModelAndView onSubmit(HttpServletRequest request,  HttpServletResponse response,  Object command, BindException errors) **throws** Exception {  log.debug("entering \"onSubmit\" method");    String success = getSuccessView();    User remoteUser = userManager.getUserByUsername(request.getRemoteUser());  SampleSearch search = (SampleSearch)command;  String tag = search.getSamplerId();  String from = search.getFrom().trim();  String to = search.getTo().trim();  // Initialize search date range  SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyy-MM-dd");  Date fromDate = dateFormat.parse("1990-01-01");  Date toDate = **new** Date();    **if** (!from.equals("")) {  fromDate = dateFormat.parse(from);  }  **if** (!to.equals("")) {  toDate = dateFormat.parse(to);  }  //=============================================================  List<Sample> sampleList = **new** ArrayList();  **if** (tag.equals("ALL")) {  sampleList = sampleManager.findSamplesByLabAndDateRange(remoteUser,  fromDate, toDate);  } **else** {  sampleList = sampleManager.findSamplesByLabAndSamplerAndDateRange(  remoteUser, tag, fromDate, toDate);  }  //=============================================================  //========= sampler list ==========  List<String> samplerIdList = **new** ArrayList<String>();  List<String> samplerList = sampleManager.getMySamplerIdList(remoteUser);  **if** (samplerList.size() != 0) {  samplerIdList.add("ALL");  samplerIdList.addAll(samplerList);  } **else** {  saveError(request, "No sampler available");  }  //=============== Add Objects =======================  ModelAndView mv = **new** ModelAndView("laboratory/samplesearch",  getCommandName(), command);  mv.addObject("sampleList", sampleList);  mv.addObject("samplerIdList", samplerIdList);  mv.addObject("samplerId", tag);  mv.addObject("firstTime", "nup");  **return** mv;  }  } |

# 

# Search Sample View

This view is used to display sample details and contains some fields for search purposes.

|  |
| --- |
| <%@ include file=*"/common/taglibs.jsp"*%>  <head>  <title><fmt:message key=*"sampleList.title"*/></title>  <meta name=*"heading"* content=*"*<fmt:message key=*'sampleList.heading'*/>*"*/>  <meta name=*"menu"* content=*"SampleMenu"*/>  <script type=*"text/javascript"* src=*"*<c:url value=*'/scripts/calendar/jscal2.js'*/>*"*></script>  <script type=*"text/javascript"* src=*"*<c:url value=*'/scripts/calendar/lang/en.js'*/>*"*></script>  <link rel=*"stylesheet"* type=*"text/css"* href=*"/styles/jscal2/jscal2.css"* />  <link rel=*"stylesheet"* type=*"text/css"* href=*"/styles/jscal2/border-radius.css"* />  <link rel=*"stylesheet"* type=*"text/css"* href=*"/styles/jscal2/gold/gold.css"* />  </head>  <!-- ============================ Search Form =========================== -->  <form:form commandName=*"search"* method=*"post"* action=*"/laboratory/samplesearch.html"* id=*"sampleSearch"*>  <form:errors path=*"\*"* cssClass=*"error"* element=*"div"*/>  <ul>  <li>  <appfuse:label styleClass=*"desc"* key=*"sampleSearch.samplerId"*/>  <form:errors path=*"samplerId"* cssClass=*"fieldError"*/>  <form:select path=*"samplerId"* id=*"samplerId"* cssClass=*"text medium"*>  <c:forEach items=*"*${samplerIdList}*"* var=*"Id"*>  <option value=*"*<c:out value="${Id}"/>*"*  <c:choose>  <c:when test="${Id == samplerId}">selected</c:when>  </c:choose>  ><c:out value=*"*${Id}*"*/></option>  </c:forEach>  </form:select>  </li>  <li>  <appfuse:label styleClass=*"desc"* key=*"sampleSearch.from"*/>  <form:errors path=*"from"* cssClass=*"fieldError"*/>  <form:input path=*"from"* id=*"from"* cssClass=*"text medium"*/>  <input type=*"button"* id=*"fromB"* class=*"button"* value=*"..."*/>  </li>  <li>  <appfuse:label styleClass=*"desc"* key=*"sampleSearch.to"*/>  <form:errors path=*"to"* cssClass=*"fieldError"*/>  <form:input path=*"to"* id=*"to"* cssClass=*"text medium"*/>  <input type=*"button"* id=*"toB"* class=*"button"* value=*"..."*/>  </li>  <li class=*"buttonBar bottom"*>  <input type=*"submit"* class=*"button"* name=*"search"* value=*"*<fmt:message key=*"button.search"*/>*"*/>  </li>  </ul>  </form:form>  <!-- ===================== Search Results ================================= -->  <display:table name=*"sampleList"* cellspacing=*"0"* cellpadding=*"0"* requestURI=*""*  id=*"sampleList"* pagesize=*"25"* class=*"table sampleList"* export=*"true"*>  <display:column property=*"id"* escapeXml=*"true"* sortable=*"true"*  url=*"sampleform.html"* paramId=*"id"* paramProperty=*"id"* titleKey=*"sample.id"*/>  <display:column property=*"date\_taken"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.date\_taken"*/>  <display:column property=*"ph"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.ph"*/>  <display:column property=*"ec"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.ec"*/>  <display:column property=*"temperature"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.temperature"*/>  <display:column property=*"collar\_depth"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.collar\_depth"*/>  <display:column property=*"arsenic"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.arsenic"*/>  <display:column property=*"grease"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.grease"*/>  <display:column property=*"fluoride"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.fluoride"*/>  <display:column property=*"chromium"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.chromium"*/>  <display:column property=*"laboratory.companyName"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.laboratory"*/>  <display:column property=*"sampler.tag"* escapeXml=*"true"* sortable=*"true"* titleKey=*"sample.sampler"*/>  <display:setProperty name=*"paging.banner.item\_name"* value=*"sample"*/>  <display:setProperty name=*"paging.banner.items\_name"* value=*"samples"*/>  <display:setProperty name=*"export.excel.filename"* value=*"Sample List.xls"*/>  <display:setProperty name=*"export.csv.filename"* value=*"Sample List.csv"*/>  <display:setProperty name=*"export.pdf.filename"* value=*"Sample List.pdf"*/>  </display:table>  <!-- ========================================================================== -->  <script type=*"text/javascript"*>  highlightTableRows("sampleList");  Form.focusFirstElement($('sampleSearch'));  //Calendars  **var** cal1=Calendar.setup(  {  inputField : "from", // id of the input field  //dateFormat : "%d/%m/%Y %H:%M", // the date format  trigger : "fromB", // id of the button  }  );  **function** updateFields1(cal) {  **var** hour=cal.getHours();  var min=cal.getMinutes();  date=document.getElementById("from").value.split(" ")[0];  document.getElementById("from").value=date+" "+hour+":"+min;  };  **var** cal2=Calendar.setup(  {  inputField : "to", // id of the input field  //dateFormat : "%d/%m/%Y %H:%M", // the date format  trigger : "toB", // id of the button  }  );  **function** updateFields1(cal) {  **var** hour=cal.getHours();  **var** min=cal.getMinutes();  date=document.getElementById("from").value.split(" ")[0];  document.getElementById("from").value=date+" "+hour+":"+min;  };  </script> |

# Dispatcher Servlet

All the user requests are handled first by the dispatcher servlet. The dispatcher servlet will send the user requests to the corresponding controllers.

|  |
| --- |
| <bean id=*"sampleSearchController"* class=*"com.tiffany.webapp.controller.SampleSearchController"*>  <property name=*"validator"* ref=*"sampleSearchValidator"* />  <property name=*"successView"* value=*"/laboratory/samplesearch"* />  <property name=*"sampleManager"* ref=*"sampleManager"* />  <property name=*"samplerManager"* ref=*"samplerManager"* />  <property name=*"userManager"* ref=*"userManager"* />  </bean>  <bean id=*"sampleFormController"* class=*"com.tiffany.webapp.controller.SampleFormController"*>  <property name=*"validator"* ref=*"beanValidator"* />  <property name=*"successView"* value=*"redirect:samples.html"* />  <property name=*"sampleManager"* ref=*"sampleManager"* />  <property name=*"samplerManager"* ref=*"samplerManager"* />  <property name=*"userManager"* ref=*"userManager"* />  </bean> |