# Pagination with Spring and JSF 2

## Introduction

In a web application, Pagination, or Page Navigation, is a technique often used to display a large set of data. For a large hundred or thousand or even millions of records, it would cause large performance problem to return all of that data in a single page. Pagination is used to present data to user one page at a time, typically 25 to 100 records per page. Moreover, keyword search, advance search option and column sorting are features often needed together with Pagination.

This article will focus on an implementation of Pagination with Spring Framework that will provide full features of pagination, search option and column sorting.

## Technologies

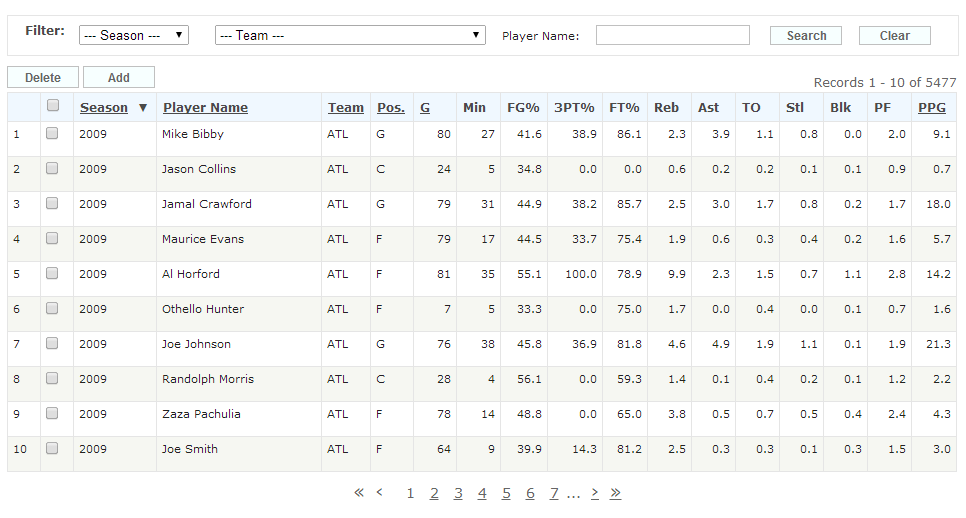
The following technologies are needed:

* Pagination Framework for Spring
* Spring Framework 3.2.3.RELEASE
* Hibernate as JPA provider
* JSF 2.1
* JDK 1.6

## Approach

This article will use a top-down approach, starting with user interface (View), working the way down to Controller, then to Service and Repository (Model / DAO) layer.

Suppose our Project Manager has gathered user requirements for a Basketball (NBA) Statistic System and an application prototype produced:



Top of the page will have a few search options for users to search through statistics. On the middle left there are 2 action buttons for users to delete or add statistics. On the middle right side total number of basketball players and the corresponding page number will be shown. The basketball players are displayed to user one page at a time, 10 players per page. Columns can be sorted by Season, Player Name, etc. And finally, pagination component is rendered for user to change page. The left most part is for go to the first page; follow by link to go to previous page; follow by numeric items to go to a specific page; follow by next page and last page link.

In order to implement pagination, we will need to create the following files:

|  |  |  |
| --- | --- | --- |
| **Object** | **Extends** | **Description** |
| BoSeasonStat | BoPaginationResultRow | Business Object for storing search options |
| BoSeasonStatSearchParam | BoPaginationParam | Business Object for storing one row of data |
| SeasonStatManagedBean | PaginationControllerAbstract | Controller to interpret user input and set data into model |
| SeasonStatService | PaginationServiceAbstract | Business Logic is implemented and raw data from JPA is mapped to Business Object |
| SeasonStatDao | PaginationDaoDbEntityManagerAbstract | Database query |
| seasonstat.xhtml | http://pagination/pagination-jsf | User Interface presentation |
| PgNbaPlayer  PgNbaSeason  PgNbaTeam | Nil | Database mapping object |

## User Interface (JSF)

To create a JSF page for Basketball Statistics System, we will use Pagination Framework for Spring. Pagination Framework contains a predefined JSF Tag Library for generating our pagination page. We will also need JSF core library.

|  |
| --- |
| <html xmlns="http://www.w3.org/1999/xhtml"  xmlns:ui="http://java.sun.com/jsf/facelets"  xmlns:f="http://java.sun.com/jsf/core"  xmlns:h="http://java.sun.com/jsf/html"  xmlns:pg="http://pagination/pagination-jsf"  > |

Next, we will need html form tag for search option submission. We will use varies JSF form tag to create our form.

|  |
| --- |
| <h:form id="pgform">  .  .  .  </h:form> |

Next, use Pagination Framework tag library to create our pagination search and data content. Pagination tag library has tag-file name “pagination” (<pg:pagination>) to help generate the pagination content. pg:pagination has 6 attributes: pparam, paginationResult, searchContent, controlButton, columnsContent and contentAfterLastRow. For pparam and paginationResult, I will explain them in more details in Business Object section below. Now let’s first look at how we use <pg:pagination> tag:

|  |
| --- |
| <pg:pagination pparam="#{seasonStatManagedBean.pparam}" paginationResult="#{seasonStatManagedBean.map['paginationResult']}">  <ui:define name="searchContent">  <table class="searchtable">  <tr>  <td class="caption" style="width:50px;">Filter:</td>  <td style="width:120px;">  <h:selectOneMenu id="season" value="#{pparam.season}" >  <f:selectItem itemLabel="--- Season ---" itemValue="-" />  <f:selectItem itemLabel="2009" itemValue="2009"/>  <f:selectItem itemLabel="2008" itemValue="2008"/>  <f:selectItem itemLabel="2007" itemValue="2007"/>  <f:selectItem itemLabel="2006" itemValue="2006"/>  <f:selectItem itemLabel="2005" itemValue="2005"/>  <f:selectItem itemLabel="2004" itemValue="2004"/>  <f:selectItem itemLabel="2003" itemValue="2003"/>  <f:selectItem itemLabel="2002" itemValue="2002"/>  <f:selectItem itemLabel="2001" itemValue="2001"/>  <f:selectItem itemLabel="2000" itemValue="2000"/>  </h:selectOneMenu>  </td>  <td style="width:200px;">  <h:selectOneMenu id="teamName" value="#{pparam.teamName}" >  <f:selectItem itemLabel="--- Team ---" itemValue="-" />  <f:selectItems value="#{seasonStatManagedBean.map['teamList']}" var="tt" itemLabel="#{tt.teamAlias} - #{tt.location} #{tt.teamName}" itemValue="#{tt.teamAlias}" />  </h:selectOneMenu>  </td>  <td style="width:80px;">Player Name:</td>  <td style="width:160px;"><h:inputText value="#{pparam.playerName}" style="width:150px;"/></td>  <td style="width:75px;"><span class="button"><h:commandButton value="Search" action="#{seasonStatManagedBean.clickSearchButton}" style="width:70px;"><f:param name="buttonAction" value="searchButton" /></h:commandButton></span></td>  <td style="width:75px;"><span class="button"><h:commandButton value="Clear" action="#{seasonStatManagedBean.clickClearButton}" style="width:70px;"><f:param name="buttonAction" value="clearButton" /></h:commandButton></span></td>  </tr>  </table>  </ui:define>  <ui:define name="controlButton">  <div style="padding-top:10px;">  <span class="button"><h:commandButton value="Delete" action="#{seasonStatManagedBean.clickDeleteButton}" style="width:70px;"><f:param name="buttonAction" value="deleteButton" /></h:commandButton></span>  <span class="button"><h:commandButton value="Add" action="#{seasonStatManagedBean.clickAddButton}" style="width:70px;"><f:param name="buttonAction" value="addButton" /></h:commandButton></span>  </div>  </ui:define>  <ui:define name="columnsContent">  <td class="cell"><span><h:outputText value="#{item.season}"/></span></td>  <td class="cell"><span style="white-space:nowrap;"><h:outputText value="#{item.playerName}"/></span></td>  <td class="cell"><span style="white-space:nowrap;"><h:outputText value="#{item.teamAlias}" title="#{item.teamName}"/></span></td>  <td class="cell"><span><h:outputText value="#{item.position}"/></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.gamePlay}"/></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.minutePerGame}"><f:convertNumber pattern="##0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.fgPercent}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.tpPercent}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.ftPercent}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.reboundPerGame}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.assistPerGame}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.turnoverPerGame}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.stealPerGame}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.blockPerGame}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.pfPerGame}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  <td class="cell" style="text-align:right;"><span><h:outputText value="#{item.pointPerGame}"><f:convertNumber pattern="##0.0"/></h:outputText></span></td>  </ui:define>  </pg:pagination> |

searchContent, controlButton, columnsContent and contentAfterLastRow are all jsp fragments.

searchContent contains the search options;

controlButton contains 2 buttons for delete and add action.

columnsContent defines how we present the search result, that is each row of data (each player), to our users. Here an object instance named “bo” is binded to this jsp fragment and can be used to load data from our BoSeasonStat object (see below).

contentAfterLastRow should defines an area just below the last row of data in current page. We didn’t need it in this example.

## Business Object (BO)

### pparam

Page Parameter (pparam) is a business-object representing the search options. We declare this object in com.github.paginationspring.example.spring3.bo.BoSeasonStatSearchParam. This object extends com.github.paginationspring.bo.BoPaginationParam (POJO) and added 5 fields from our search options:

|  |
| --- |
| public class BoSeasonStatSearchParam extends BoPaginationParam {  private String season;  private String playerName;  private String teamName; |

pparam will keep value of search option in this pojo and use it in our pagination framework.

### paginationResult

Pagination Result is a business-object representing results of our search. It contains one page of data (as seen by user on his current page), and it also contains some settings like record-per-page, default sorting order, etc. Each row of data (in our case, each player) is represented by declaring object com.github.paginationspring.example.spring3.bo.BoSeasonStat. This object extends com.github.paginationspring.bo.BoPaginationResultRow (POJO) and added fields for display our search results:

|  |
| --- |
| public class BoSeasonStat extends BoPaginationResultRow<Integer> {  private int nbaSeasonId;  private String playerName;  private String height;  private String weight;  private String teamAlias;  private String teamName;  private String position;  private String season;  private int gamePlay;  private double minutePerGame;  private double pointPerGame;  private double reboundPerGame;  private double assistPerGame;  private double stealPerGame;  private double blockPerGame;  private double turnoverPerGame;  private double pfPerGame;  private double fgPercent;  private double ftPercent;  private double tpPercent;  @Override  public Integer getPk() {  return messageId;  } |

Notice the type inference of <Integer>, this Integer represents the object type of the unique key of this row of data (i.e. nbaSeasonId).

## ManagedBean

ManagedBean will define settings of our pagination page, map requests URLs into a handler method and pass business logic to service layer.

|  |
| --- |
| @ManagedBean  @RequestScoped  public class SeasonStatManagedBean extends PaginationControllerAbstract<BoSeasonStatSearchParam> { |

In our constructor, we will define some settings for our pagination results.

|  |
| --- |
| public SeasonStatManagedBean() {  setOptionDisplayCheckbox(true);  setOptionDisplaySerialNo(true);  setOptionWidth(950);  setDefaultRecordPerPage(10);  setDefaultSortName("Season");  setDefaultSortAscDesc("d");  setPageLink("/xhtml/org/pagination/example/jsf2/view/seasonstat.faces");  } |

|  |  |
| --- | --- |
| setOptionDisplayCheckbox | whether or not the left side checkbox should be displayed |
| setOptionDisplaySerialNo | whether or not the left side serial no. should be displayed |
| setOptionWidth | the width of our search results, in pixels |
| setDefaultRecordPerPage | default record per page |
| setDefaultSortName | default sorting column |
| setDefaultSortAscDesc | default sorting order, value “a” is ascending and “d” is decending. |
| setPageLink | url of your pagination page |
| setRewriteUrl | to give you nicer, SEO (Search Engine Optimization) friendly, urls. For example, by setting setRewriteUrl to true, URL mailclient/inbox.do?sortAscDesc=a&sortName=From becomes mailclient/sort/From/a/inbox.do |

Next, we will need to map request URLs into a handler method.

|  |
| --- |
| @PostConstruct  public void getRun() {  try {  log.debug("seasonStatService="+seasonStatService);  super.setPaginationService(seasonStatService);  // setup pparam and buttonAction  Map<String,String> rpmap = FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap();  Map<String,String[]> rpvmap = FacesContext.getCurrentInstance().getExternalContext().getRequestParameterValuesMap();    BeanUtils.populate(pparam, rpvmap);  if ( "-".equals(rpmap.get("season")) ) pparam.setSeason (null);  if ( "-".equals(rpmap.get("teamName")) ) pparam.setTeamName (null);  String buttonAction=rpmap.get("buttonAction");  log.debug("buttonAction="+buttonAction);  if ( buttonAction!=null ) {  map = new HashMap<String, Object>();  } else {  map = assignModel(pparam, null);  }  map.put("teamList", seasonStatService.retrieveTeamAliass());  } catch (Exception e) {  log.error("",e);  }  } |

Inside this method, we will call a parent method assignModel from our Pagination Framework, set the returned objects into our model arguments and result the url. The assignModel method will call a service bean, which we will be injecting into our controller. The service bean implements our business logic in our pagination, which we will explain in the next section.

At last, inject our service bean:

|  |
| --- |
| @ManagedProperty(value="#{seasonStatService}")  private SeasonStatService seasonStatService; |

## Service

Service bean serve 2 purposes: 1) map database raw data into our business object and 2) define column properties for result data.

First, we define our service bean with @Service annotation and extend a service base PaginationServiceAbstract as shown here:

|  |
| --- |
| @Service  public class SeasonStatService extends PaginationServiceAbstract<BoSeasonStatSearchParam, BoSeasonStat, PgNbaSeason> { |

Type inference of <BoSeasonStatSearchParam, BoSeasonStat, PgNbaSeason> represent our page parameter, business object and data entity object respectively. Data Entiry PgNbaSeason is generated with Hibernate / JBoss Tools from database structure.

To map database raw data into our business object, override assignDataToBo method:

|  |
| --- |
| @Override  protected BoSeasonStat assignDataToBo(PgNbaSeason entity) throws Exception {  BoSeasonStat bo = new BoSeasonStat();  bo.setNbaSeasonId(entity.getNbaSeasonId());  bo.setPlayerName (entity.getPgNbaPlayer().getFirstName()+" "+entity.getPgNbaPlayer().getLastName());  bo.setHeight (entity.getPgNbaPlayer().getHeightFeet()+"-"+entity.getPgNbaPlayer().getHeightInches());  bo.setWeight (String.valueOf(entity.getPgNbaPlayer().getWeight()));  bo.setTeamName (entity.getPgNbaTeam().getLocation()+" "+entity.getPgNbaTeam().getTeamName());  bo.setTeamAlias (entity.getPgNbaTeam().getTeamAlias());  bo.setPosition (entity.getPgNbaPlayer().getPosition());  bo.setSeason (entity.getSeason ());  bo.setGamePlay (entity.getGamePlay ());  if ( entity.getGamePlay() != 0 ) {  bo.setMinutePerGame ((double) entity.getMinute ()/(double) entity.getGamePlay());  bo.setPointPerGame ((double) entity.getPoint ()/(double) entity.getGamePlay());  bo.setReboundPerGame ((double) entity.getRebound ()/(double) entity.getGamePlay());  bo.setAssistPerGame ((double) entity.getAssist ()/(double) entity.getGamePlay());  bo.setStealPerGame ((double) entity.getSteal ()/(double) entity.getGamePlay());  bo.setBlockPerGame ((double) entity.getBlock ()/(double) entity.getGamePlay());  bo.setTurnoverPerGame ((double) entity.getTurnover ()/(double) entity.getGamePlay());  bo.setPfPerGame ((double) entity.getPf ()/(double) entity.getGamePlay());  }  if ( entity.getFgAttempt() != 0 ) {  bo.setFgPercent (((double) entity.getFgMade() / (double) entity.getFgAttempt())\*(double) 100);  }  if ( entity.getTpAttempt() != 0 ) {  bo.setTpPercent(((double) entity.getTpMade() / (double) entity.getTpAttempt())\*(double) 100);  }  if ( entity.getFtAttempt() != 0 ) {  bo.setFtPercent(((double) entity.getFtMade() / (double) entity.getFtAttempt())\*(double) 100);  }  return bo;  } |

To define column properties, override assignColumnsDefinition method:

|  |
| --- |
| @Override  public void assignColumnsDefinition(List<BoPaginationColumn> columns)  throws Exception {  log.debug("setting columns def.");  BoPaginationColumn col = null;    col = new BoPaginationColumn();  col.setColumnName("Season");  col.setOrderColumns("a.season, c.teamAlias, b.lastName, b.firstName");  col.setOrderDirections("desc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Player Name");  col.setOrderColumns("b.lastName, b.firstName, a.season");  col.setOrderDirections("asc, asc, asc");  col.setWidth(150);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Team");  col.setOrderColumns("c.teamAlias, a.season, b.lastName, b.firstName");  col.setOrderDirections("asc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Pos.");  col.setOrderColumns("b.position, a.season, c.teamAlias, b.lastName, b.firstName");  col.setOrderDirections("asc, asc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("G");  col.setOrderColumns("a.gamePlay, a.season, c.teamAlias, b.lastName, b.firstName");  col.setOrderDirections("asc, asc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Min");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("FG%");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("3PT%");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("FT%");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Reb");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Ast");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("TO");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Stl");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Blk");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("PF");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("PPG");  col.setOrderColumns("(a.point/a.gamePlay), a.season, c.teamAlias, b.lastName, b.firstName");  col.setOrderDirections("desc, asc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  } |

We create a new BoPaginationColumn bean for each column. BoPaginationColumn has 4 variables: columnName, orderColumns, orderDirections and width. The columnName is the name of column to be displayed to user. The orderColumns is a query string to be added to our JPA query “order by” clause. The orderDirections has value “asc” or “desc” for ascending or descending shorting of the column. For sorting across multiple fields, use comma to separate between required fields (as shown in “From” column above). The width variable defines the column width in pixels.

At last, inject our DAO bean with @Autowired annotation and call a parent method setPaginationDao:

|  |
| --- |
| @Autowired  public void setPaginationDao(SeasonStatDao seasonStatDao) {  super.setPaginationDao(seasonStatDao);  this.seasonStatDao = seasonStatDao;  } |

## Data Access Layer (DAO)

DAO bean load the raw data from database using JPA entity manager. First, we define our DAO with @Repository annotation and extends PaginationDaoDbEntityManagerAbstract<PgNbaSeason, BoSeasonStatSearchParam> from Pagination Framework:

|  |
| --- |
| @Repository  public class SeasonStatDao extends PaginationDaoDbEntityManagerAbstract<PgNbaSeason, BoSeasonStatSearchParam> { |

PgNbaSeason is the database entity bean and BoSeasonStatSearchParam is the incoming page parameters.

First, we will setup query to load data from database:

|  |
| --- |
| private static final String EJBQL = "select distinct a, b, c " +  "from PgNbaSeason a " +  "join a.pgNbaPlayer b " +  "join a.pgNbaTeam c " +  "where a.gamePlay > 0 ";    private static final String[] RESTRICTIONS = {  "a.season = :season",  "lower(concat(concat(b.firstName,' '),b.lastName)) like :playerNameCustomized",  "c.teamAlias=:teamName"  };  public SeasonStatDao() {  this.ejbql = EJBQL;  this.restrictions = Arrays.asList(RESTRICTIONS);  } |

Let’s look at our query. We select distinct PgNbaSeason (alias a) in order to pass this object to assignDataToBo method in Service bean. In order for the query to be able to sort by “Player Name” field (see assignColumnsDefinition in Service bean), we need to select pgNbaPlayer (alias b) in our query. Moreover, we will define pgNbaTeam (alias c) and in where clause we restrict our query for gamePlay larger than 0.

Restrictions define our search options. There are 2 kinds of restrictions binding: 1) binding directly to variable in pparam (BoSeasonStatSearchParam) and 2) binding via customization.

The first restriction a.season=:season means that PgNbaSeason.season are binding directly to variable “season” in pparam. Other restrictions are defined in binding in customizeRestrictions method:

|  |
| --- |
| @Override  public void customizeRestrictions(Map<String, Object> queryParameters,  BoSeasonStatSearchParam pparam) throws Exception {  if ( StringUtils.isEmpty(pparam.getPlayerName()) ) {  queryParameters.put("playerNameCustomized", null);  } else {  queryParameters.put("playerNameCustomized", "%"+StringUtils.trim(pparam.getPlayerName().toLowerCase())+"%");  }  } |

In this customizeRestrictions method, playerNameCustomized is customized and then put into a map queryParameters. Our Pagination Framework will use this map to bind variables into our JPA query.

Finally, we need to inject entityManager via @PersistenceContext annotation:

|  |
| --- |
| @PersistenceContext  public void setEntityManager(EntityManager entityManager) {  super.setEntityManager(entityManager);  } |

In this example, we used JPA Entity Manager (PaginationDaoDbEntityManagerAbstract) for persistence management. There are 2 more options available in our Pagination Framework, one is PaginationDaoDbHibernateTemplateAbstract which uses spring hibernate template and the other is PaginationDaoLuceneAbstract which uses lucene index as datasource.

## Spring Framework Setup

To setup spring framework, we setup spring and JSF in web.xml as we normally would:

|  |
| --- |
| <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- JSF mapping -->  <servlet>  <servlet-name>Faces Servlet</servlet-name>  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <!-- Faces Servlet Mapping -->  <servlet-mapping>  <servlet-name>Faces Servlet</servlet-name>  <url-pattern>\*.faces</url-pattern>  </servlet-mapping> |

In applicationContext.xml file, define the following for our annotated components:

|  |
| --- |
| <context:annotation-config />  <context:component-scan base-package="com.github.paginationspring.example.jsf2" /> |

You should also define EntityManagerFactory for our JPA.

In faces-config.xml, define the following for view handler:

|  |
| --- |
| <application>  <variable-resolver>org.springframework.web.jsf.DelegatingVariableResolver</variable-resolver>  <el-resolver>org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-resolver>  </application>  <component>  <component-type>pagination-url-component</component-type>  <component-class>com.github.paginationspring.web.PaginationUrlLinkJsfComponent</component-class>  </component>  <component>  <component-type>pagination-sorting-url-component</component-type>  <component-class>com.github.paginationspring.web.PaginationSortingUrlLinkJsfComponent</component-class>  </component> |

## Running our example

Download this zip file and extract the content into your computer. The content included full source code used in this example. Open command prompt and type run.bat (for window OS) or ./run.sh (for linux OS). Choose your application server and wait around 30 seconds for the server to startup. After server started, go to url <http://localhost:8080/pagination-example-jsf2>