```java
1  package nl.rug.search.opr.controller;
2
3  import java.util.ArrayList;
4  import java.util.Collection;
5  import java.util.HashMap;
6  import java.util.List;
7  import java.util.Map;
8  import javax.ejb.EJB;
9  import javax.faces.bean.ManagedBean;
10 import javax.faces.bean.SessionScoped;
11 import javax.faces.context.FacesContext;
12 import javax.faces.event.ActionEvent;
13 import javax.servlet.http.HttpServletRequest;
14 import nl.rug.search.opr.AbstractFormBean;
15 import nl.rug.search.opr.component.ConsequenceWrapper;
16 import nl.rug.search.opr.component.ForceWrapper;
17 import nl.rug.search.opr.component.TextBlockWrapper;
18 import nl.rug.search.opr.entities.pattern.Consequence;
19 import nl.rug.search.opr.entities.pattern.Force;
20 import nl.rug.search.opr.entities.pattern.Pattern;
21 import nl.rug.search.opr.pattern.PatternLocal;
22 import nl.rug.search.opr.entities.pattern.PatternVersion;
23 import nl.rug.search.opr.entities.pattern.TextBlock;
24 import nl.rug.search.opr.entities.template.Component;
25 import nl.rug.search.opr.entities.pattern.File;
26
27 /**
28  *
29  * @author Martin Verspai <martin@verspai.de>
30  * @date 07.12.2009
31  */
32 @ManagedBean(name="editVersionCtrl")
33 @SessionScoped
34 public class EditVersionController extends AbstractFormBean {
35
36     @EJB
37     private PatternLocal pl;
38
39     private static final String PATTERN_ID = "patternId";
40     private static final String VERSION_ID = "versionId";
41
42     private static final String successMsg  = "Pattern has been edited!";
43     private static final String failMsg     = "Pattern has not been edited!";
44
45     private Pattern pattern;
46     private PatternVersion patternVersion;
47     private Map<String, TextBlockWrapper> blocks;
48     private Collection<ForceWrapper> forces;
49     private Collection<ConsequenceWrapper> consequences;
50
51     public EditVersionController() {
52         init();
53     }
54
55     @Override
56     public String getFormId() {
57         return "editDescription";
58     }
59
60     @Override
61     public String successMessage() {
62         return EditVersionController.successMsg;
63     }
64
65     @Override

66     public String failMessage() {
67         return EditVersionController.failMsg;
68     }
69
70     public void reset(ActionEvent e) {
71         reset();
72     }
73
74     @Override
75     public void reset() {
76         initData();
77     }
78
79     private void init() {
80         this.pattern        = null;
81         this.patternVersion = null;
82         this.blocks         = new HashMap<String, TextBlockWrapper>();
83         this.forces         = new ArrayList<ForceWrapper>();
84         this.consequences   = new ArrayList<ConsequenceWrapper>();
85     }
86
87
88     @Override
89     public void execute() {
90         List<TextBlock> tmpBlocks           = new ArrayList<TextBlock>();
91         List<Force> patternForces           = new ArrayList<Force>();
92         List<Consequence> patternConsequences = new ArrayList<Consequence>();
93
94         for (TextBlockWrapper tbw : blocks.values()) {
95             TextBlock tb = tbw.getTextBlock();
96             tb.setId(null);
97             tmpBlocks.add(tb);
98         }
99
100        for(ForceWrapper fw : this.forces) {
101            Force f = fw.getForce();
102            f.setId(null);
103            patternForces.add(f);
104        }
105
106        for(ConsequenceWrapper cw : this.consequences) {
107            Consequence c = cw.getConsequence();
108            c.setId(null);
109            patternConsequences.add(c);
110        }
111
112        PatternVersion newVersion = new PatternVersion();
113            newVersion.setAuthor(this.patternVersion.getAuthor());
114            newVersion.setBlocks(tmpBlocks);
115            newVersion.setConsequences(patternConsequences);
116            newVersion.setForces(patternForces);
117            newVersion.setLicense(this.patternVersion.getLicense());
118            newVersion.setSource(this.patternVersion.getSource());
119            newVersion.setTemplate(this.patternVersion.getTemplate());
120            newVersion.setFiles((List<File>)this.patternVersion.getFiles());
121
122        this.pattern.setCurrentVersion(newVersion);
123
124        pl.editVersion(this.pattern);
125
126        Pattern p = pl.getById(this.pattern.getId());
127
128        this.pattern = p;
129        this.patternVersion = p.getCurrentVersion();
130
131        initData();
132    }
133
```

```java
134    public Pattern getPattern() {
135        HttpServletRequest request = (HttpServletRequest) FacesContext.getCurrentInstance().getExternalContext().getRequest();
136
137        String patternIdStr = request.getParameter(PATTERN_ID);
138        String versionIdStr = request.getParameter(VERSION_ID);
139
140        if( patternIdStr != null && !patternIdStr.equals("") ) {
141            Pattern tmpPattern;
142            PatternVersion tmpVersion;
143
144            try {
145                Long tmpId = Long.parseLong(patternIdStr);
146
147                if( (tmpPattern = pl.getById(tmpId)) != null) {
148                    tmpVersion = tmpPattern.getCurrentVersion();
149
150                    if( versionIdStr != null && !versionIdStr.equals("") ) {
151                        long versionId = Integer.parseInt(versionIdStr);
152
153                        for(PatternVersion pv : tmpPattern.getVersions()) {
154                            if( pv.getId().equals(versionId) ) {
155                                tmpVersion = pv;
156                            }
157                        }
158                    }
159
160                    if(this.pattern == null || !this.pattern.equals(tmpPattern)) {
161                        this.pattern = tmpPattern;
162                        this.patternVersion = tmpVersion;
163
164                        initData();
165                    }
166
167                    if(this.patternVersion == null || !this.patternVersion.equals(tmpVersion)) {
168                        this.patternVersion = tmpVersion;
169
170                        initData();
171                    }
172                }
173            } catch (NumberFormatException nfe) { /* TODO: Do something? */ }
174        }
175
176        return this.pattern;
177    }
178
179    private void initData() {
180        List<File> uploads = new ArrayList<File>();
181        uploads.addAll(this.patternVersion.getFiles());
182
183        this.blocks        = new HashMap<String, TextBlockWrapper>();
184        this.forces        = new ArrayList<ForceWrapper>();
185        this.consequences  = new ArrayList<ConsequenceWrapper>();
186
187        for (Component c : this.patternVersion.getTemplate().getTextComponents()) {
188            if (!blocks.containsKey(c.getIdentifier())) {
189                TextBlock block = new TextBlock();
190                block.setComponent(c);
191                block.setText("");
192                blocks.put(c.getIdentifier(), new TextBlockWrapper(block));
193            }
194        }
195
196        for (TextBlock tb : this.patternVersion.getBlocks()) {
197            blocks.put(tb.getComponent().getIdentifier(), new TextBlockWrapper(tb));
198        }
199
200        for (Force f : this.patternVersion.getForces() ) {
201            this.forces.add(new ForceWrapper(f));
202        }
203
204        for (Consequence c : this.patternVersion.getConsequences() ) {
205            this.consequences.add(new ConsequenceWrapper(c));
206        }
207    }
208
209    public PatternVersion getVersion() {
210        return this.patternVersion;
211    }
212
213    public Map<String, TextBlockWrapper> getBlocks() {
214        return this.blocks;
215    }
216
217    public Collection<ForceWrapper> getForces() {
218        return forces;
219    }
220
221    public void setForces(Collection<ForceWrapper> forces) {
222        this.forces = forces;
223    }
224
225    public void addForce(ActionEvent e) {
226        ForceWrapper fw = new ForceWrapper(new Force());
227        fw.setEditMode(true);
228        forces.add(fw);
229    }
230
231    public void removeForce(ActionEvent e) {
232        ForceWrapper f = (ForceWrapper) e.getComponent().getAttributes().get("force");
233        forces.remove(f);
234    }
235
236    public Collection<ConsequenceWrapper> getConsequences() {
237        return this.consequences;
238    }
239
240    public void setConsequences(Collection<ConsequenceWrapper> consequences) {
241        this.consequences = consequences;
242    }
243
244    public void addConsequence(ActionEvent e) {
245        ConsequenceWrapper cw = new ConsequenceWrapper(new Consequence());
246        cw.setEditMode(true);
247        this.consequences.add(cw);
248    }
249
250    public void removeConsequence(ActionEvent e) {
251        ConsequenceWrapper c = (ConsequenceWrapper) e.getComponent().getAttributes().get("consequence");
252        this.consequences.remove(c);
253    }
254
255    public void removeFile(ActionEvent e) {
256        File f = (File) e.getComponent().getAttributes().get("file");
257        if (f != null) {
258            patternVersion.getFiles().remove(f);
259        }
260    }
261
262 }
263
264
```

```java
1  package nl.rug.search.opr;
2
3  import org.apache.commons.logging.Log;
4  import org.apache.commons.logging.LogFactory;
5
6  import javax.servlet.http.HttpSessionEvent;
7  import javax.servlet.http.HttpSessionListener;
8  import java.io.File;
9
10 /**
11  *
12  * @author Martin Verspai <martin@verspai.de>
13  */
14 public class FileJanitor implements HttpSessionListener {
15
16     public static final Log logger = LogFactory.getLog(FileJanitor.class);
17     public static final String FILE_UPLOAD_DIRECTORY = "upload";
18
19     @Override
20     public void sessionDestroyed(HttpSessionEvent event) {
21         logger.info("Cleaning up files for destroyed session.");
22         String sessionId = event.getSession().getId();
23
24         String applicationPath = event.getSession().getServletContext().getRealPath(
25             event.getSession().getServletContext().getServletContextName());
26
27         File userDirectory = new File(applicationPath + FILE_UPLOAD_DIRECTORY + sessionId);
28
29         if (userDirectory.isDirectory()) {
30             try {
31                 userDirectory.delete();
32             }
33             catch (SecurityException e) {
34                 logger.error("Error deleting file upload directory: ", e);
35             }
36         }
37     }
38
39     @Override
40     public void sessionCreated(HttpSessionEvent event) {
41         // Nothing to do here
42     }
43
44 }
45
46
```

```java
14
15 /**
16  *
17  * @author cm
18  */
19 public class FileServlet extends HttpServlet {
20
21     /**
22      * Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
23      * @param request servlet request
24      * @param response servlet response
25      * @throws ServletException if a servlet-specific error occurs
26      * @throws IOException if an I/O error occurs
27      */
28     private static final Logger logger = Logger.getLogger(FileServlet.class.getName());
29     public static final String FILE_DIRECTORY = "opr.FILE_DIRECTORY";
30
31     @EJB
32     private FileLocal fb;
33
34     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
35             throws ServletException, IOException {
36
37         OutputStream os = null;
38         try {
39         String requestedFile = request.getPathInfo().substring(1);
40         String filename = requestedFile;
41
42         int size = -1;
43         boolean scaleCubic = false;
44         if (requestedFile.matches("(xy)?[1-9][0-9]*px-.*")) {
45
46             if (requestedFile.startsWith("xy")) {
47                 scaleCubic = true;
48                 requestedFile = requestedFile.substring(2);
49             }
50
51             String[] parts = requestedFile.split("px-");
52             try {
53                 size = Integer.parseInt(parts[0]);
54             } catch (NumberFormatException ex) {
55                 size = -1;
56             }
57
58             if(parts.length == 2) {
59                 filename = parts[1];
60             }
61         }
62
63         long id;
64         try {
65             id = Long.parseLong(filename);
66         } catch (NumberFormatException e) {
67             response.sendError(404);
68             return;
69         }
70
71         File file = fb.getById(id);
72
73
74         if (file == null) {
75             response.sendError(404);
76             return;
77         }
78
```

```java
79          response.setContentType(file.getMime());
80          response.addHeader("Content-Disposition", "filename="+ file.getName() +";");
81          os = response.getOutputStream();
82
83
84          if (size > 0 && file != null) {
85
86              byte[] bytes;
87              if( (bytes = fb.getThumbnail(file, size, scaleCubic)) != null) {
88                  os.write(bytes);
89              }
90
91              return;
92          }
93
94
95          os.write(file.getContent());
96
97          return;
98
99          } finally {
100             try {
101                 os.close();
102             } catch(Exception ex) {
103                 logger.log(Level.SEVERE, null, ex);
104             }
105         }
106
107     }
108
109     // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
110     /**
111      * Handles the HTTP <code>GET</code> method.
112      * @param request servlet request
113      * @param response servlet response
114      * @throws ServletException if a servlet-specific error occurs
115      * @throws IOException if an I/O error occurs
116      */
117     @Override
118     protected void doGet(HttpServletRequest request, HttpServletResponse response)
119             throws ServletException, IOException {
120         processRequest(request, response);
121     }
122
123     /**
124      * Handles the HTTP <code>POST</code> method.
125      * @param request servlet request
126      * @param response servlet response
127      * @throws ServletException if a servlet-specific error occurs
128      * @throws IOException if an I/O error occurs
129      */
130     @Override
131     protected void doPost(HttpServletRequest request, HttpServletResponse response)
132             throws ServletException, IOException {
133         processRequest(request, response);
134     }
135
136     /**
137      * Returns a short description of the servlet.
138      * @return a String containing servlet description
139      */
140     @Override
141     public String getServletInfo() {
142         return "Short description";
143     }// </editor-fold>
144 }
145
146
```

```java
23 /**
24  *
25  * @author Jan Nikolai Trzeszkowski <info@jn-t.de>
26  */
27 @ManagedBean(name = "formattingStep")
28 @ViewScoped
29 public class Formatting implements WizardStep {
30
31     @ManagedProperty(value = "#{addWizardController}")
32     protected AddWizardController wizard;
33     private Consequence editConsequence = null;
34     private Force editForce = null;
35     private UploadHelper upload = new UploadHelper();
36     private TextBlock editTextBlock = null;
37     private Map<String, TextBlock> blocks = new HashMap<String, TextBlock>();
38     private String reload;
39
40     public String getReload() {
41         return reload;
42     }
43
44     public void setReload(String reload) {
45         this.reload = reload;
46     }
47
48     public Map<String, TextBlock> getBlocks() {
49         return blocks;
50     }
51
52     public TextBlock getEditTextBlock() {
53         return editTextBlock;
54     }
55
56     public Consequence getEditConsequence() {
57         return editConsequence;
58     }
59
60     public Force getEditForce() {
61         return editForce;
62     }
63
64     public AddWizardController getWizard() {
65         return wizard;
66     }
67
68     public void setWizard(AddWizardController wizard) {
69         this.wizard = wizard;
70     }
71
72     @PostConstruct
73     private void load() {
74         PatternVersion v = wizard.getPatternVersion();
75         Template template = v.getTemplate();
76
77         blocks.clear();
78
79         for (Component c : template.getTextComponents()) {
80             if (!blocks.containsKey(c.getIdentifier())) {
81                 TextBlock block = new TextBlock();
82                 block.setComponent(c);
83                 block.setText("");
84                 blocks.put(c.getIdentifier(), block);
85             }
86         }
87
```

```java
88          for (TextBlock tb : v.getBlocks()) {
89              blocks.put(tb.getComponent().getIdentifier(), tb);
90          }
91
92          if (template.getTextComponents().size() > 0) {
93              editTextBlock = blocks.get(template.getTextComponents().get(0).getIdentifier());
94          }
95
96          v.setBlocks(blocks.values());
97      }
98
99      public UploadHelper getUpload() {
100         return this.upload;
101     }
102
103     public void removeFile(ActionEvent e) {
104         File f = (File) e.getComponent().getAttributes().get("file");
105         if (f != null) {
106             wizard.getPatternVersion().getFiles().remove(f);
107         }
108     }
109
110     public void changeEditTextBlock(ActionEvent e) {
111         TextBlock textBlock = (TextBlock) e.getComponent().getAttributes().get("textBlock");
112         if (textBlock == editTextBlock) {
113             editTextBlock = null;
114         } else {
115             editTextBlock = textBlock;
116         }
117     }
```

```java
1
2  package nl.rug.search.opr;
3
4  import java.io.IOException;
5  import javax.ejb.EJB;
6  import javax.servlet.ServletException;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;
10 import nl.rug.search.opr.file.FileLocal;
11 import org.slf4j.Logger;
12 import org.slf4j.LoggerFactory;
13
14 /**
15  *
16  * @author Christian Manteuffel <cm@notagain.de>
17  */
18 public class InitializationServlet extends HttpServlet {
19
20     private static final long serialVersionUID = 1l;
21
22     private Logger logger = LoggerFactory.getLogger(InitializationServlet.class);
23
24     @EJB
25     private FileLocal fileBean;
26
27     @Override
28     public void init() {
29         /*logger.info("Start Initialization");
30         logger.info("Init Search");
31         try {
32             searchBean.createSearchQuery("test");
33         } catch (QueryParseException ex) {
34             logger.info("Search Bean initialization error");
35         }
36         */
37         logger.info("Init FileBean Timer");
38         try {
39             fileBean.init();
40         } catch (Exception ex) {
41             logger.info("FileBean Timer initialization error");
42         }
43     }
44
45     /**
46      * Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
47      * @param request servlet request
48      * @param response servlet response
49      * @throws ServletException if a servlet-specific error occurs
50      * @throws IOException if an I/O error occurs
51      */
52     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
53     throws ServletException, IOException {
54     }
55
56     /**
57      * Handles the HTTP <code>GET</code> method.
58      * @param request servlet request
59      * @param response servlet response
60      * @throws ServletException if a servlet-specific error occurs
61      * @throws IOException if an I/O error occurs
62      */
63     @Override
64     protected void doGet(HttpServletRequest request, HttpServletResponse response)
65     throws ServletException, IOException {
```

Left column:

```
66        processRequest(request, response);
67    }
68
69    /**
70     * Handles the HTTP <code>POST</code> method.
71     * @param request servlet request
72     * @param response servlet response
73     * @throws ServletException if a servlet-specific error occurs
74     * @throws IOException if an I/O error occurs
75     */
76    @Override
77    protected void doPost(HttpServletRequest request, HttpServletResponse response)
78    throws ServletException, IOException {
79        processRequest(request, response);
80    }
81
82    /**
83     * Returns a short description of the servlet.
84     * @return a String containing servlet description
85     */
86    @Override
87    public String getServletInfo() {
88        return "Short description";
89    }
90
91 }
92
93
```

Right column:

```
1 package nl.rug.search.opr.backingbean;
2
3 import java.io.FileInputStream;
4 import java.io.FileNotFoundException;
5 import java.io.IOException;
6 import java.util.Map;
7 import javax.annotation.PostConstruct;
8 import javax.ejb.EJB;
9 import javax.faces.context.FacesContext;
10 import nl.rug.search.opr.entities.pattern.File;
11 import javax.faces.bean.ManagedBean;
12 import javax.faces.bean.ViewScoped;
13 import javax.faces.component.UIComponent;
14 import nl.rug.search.opr.file.FileException;
15 import nl.rug.search.opr.entities.pattern.License;
16 import nl.rug.search.opr.entities.pattern.PatternVersion;
17 import nl.rug.search.opr.file.FileLocal;
18 import org.icefaces.component.fileentry.FileEntry;
19 import org.icefaces.component.fileentry.FileEntryEvent;
20 import org.icefaces.component.fileentry.FileEntryResults;
21 import org.slf4j.Logger;
22 import org.slf4j.LoggerFactory;
23
24 /**
25  *
26  * @author Martin Verspai <martin@verspai.de>
27  * @version 2.0
28  * @date 26.10.2009
29  */
30 @ManagedBean(name = "uploadHelper")
31 @ViewScoped
32 public class UploadHelper {
33
34
35    protected PatternVersion version;
36    private License license;
37    private FileEntryResults.FileInfo fileInfo;
38    private String name;
39    private UIComponent uploadComponent;
40    @EJB
41    private FileLocal fileBean;
42    private Logger logger = LoggerFactory.getLogger(UploadHelper.class);
43
44
45    public UploadHelper() {
46    }
47
48    @PostConstruct
49    public void init() {
50        FacesContext ctx = FacesContext.getCurrentInstance();
51        String beanName = ctx.getExternalContext().getRequestParameterMap().get("patternVersion");
52
53        version = findBean(beanName, PatternVersion.class);
54    }
55
56    public static <T> T findBean(String beanName, Class<T> beanClass) {
57        FacesContext context = FacesContext.getCurrentInstance();
58        return beanClass.cast(context.getApplication().evaluateExpressionGet(context, "#{" + beanName + "}", beanClass));
59    }
60
61    public PatternVersion getVersion() {
62        return version;
63    }
64
65    public void setVersion(PatternVersion version) {
```

```java
66          this.version = version;
67      }
68
69      public void listener(FileEntryEvent event) {
70          FileEntry fileEntry = (FileEntry) event.getSource();
71          FileEntryResults results = fileEntry.getResults();
72
73          for (FileEntryResults.FileInfo fi : results.getFiles()) {
74              FacesContext context = FacesContext.getCurrentInstance();
75              if (fi.isSaved()) {
76                  fileInfo = fi;
77                  name = fi.getFileName();
78                  saveFile(fileInfo, name);
79              }
80          }
81
82      }
83
84      private void saveFile(FileEntryResults.FileInfo fileInfo, String name) {
85
86
87          try {
88
89              File file = null;
90              if (fileInfo.getStatus().isSuccess()) {
91
92                  String fileName = fileInfo.getFileName();
93                  file = fileBean.add(license, fileName, new FileInputStream(fileInfo.getFile()));
94              }
95              version.getFiles().add(file);
96
97          } catch (FileNotFoundException ex) {
98              System.out.println("File not found");
99          } catch (IOException ex) {
100             System.out.println("IO exception");
101         } catch (FileException ex) {
102         } finally {
103             java.io.File physicalFile = fileInfo.getFile();
104             if (physicalFile.exists()) {
105                 physicalFile.delete();
106             }
107             fileInfo = null;
108             name = "";
109         }
110
111     }
112
113     public String getName() {
114         return name;
115     }
116
117     public void setName(String name) {
118         this.name = name;
119     }
120
121     public License getLicense() {
122         return license;
123     }
124
125     public void setLicense(License license) {
126         this.license = license;
127     }
128
129     public UIComponent getUploadComponent() {
130         return uploadComponent;
131     }
132
133     public void setUploadComponent(UIComponent uploadComponent) {
```

```java
134         this.uploadComponent = uploadComponent;
135     }
136
137     public String getSupportedFileTypes() {
138         String out = "";
139         for (String fileType : fileBean.getSupportedMimeTypes()) {
140
141             String[] split = fileType.split("/");
142
143             if (split.length == 2) {
144                 out = out.concat("." + split[1] + ", ");
145             }
146         }
147         out = out.substring(0, out.length() - 2);
148
149         return out;
150     }
151
152     public int getMaximumFileSize() {
153         return fileBean.getMaximumFileSizeMb();
154     }
155 }
156
157
```