

```
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

```

```

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

```