Web-based information systems 1

WebFilters

AJAX APIs

File uploads

Michiel Noback (NOMI)
Institute for Life Sciences and Technology
Hanze University of Applied Sciences

Topics

- Using WebFilters to intercept requests
- Serving data instead of pages: AJAX APIs
- Handling files uploads that are part of a submitted form

Web filters

Why Filters?

- Suppose you want to intercept requests for a (group of) servlets, to log what is requested and where it came from
- For instance, to do data mining later on to pinpoint the most used resources
- Or more relevant: intercept requests to asses authentication status of the user

A basic example

- Here is our old friend, the PhraseServlet

```
@WebServlet(name = "PhraseServlet", urlPatterns = "/give.phrase")
public class PhraseServlet extends HttpServlet {
  /*irrelevant code omitted*/
  protected void doGet(HttpServletRequest request,
                       HttpServletResponse response)
          throws ServletException, IOException {
    String phraseType = request.getParameter("phrase_category");
    WebContext ctx = new WebContext(/*omitted*/);
    final String phrase = PhraseFactory.getPhrase(phraseType);
    ctx.setVariable("phrase", phrase);
    templateEngine.process("phrase_of_the_day", ctx,
                           response.getWriter());
    }
}
```

Enter @WebFilter

- This is the way to keep track of requests for **`/give.phrase`**

```
@WebFilter(urlPatterns = "/give.phrase")
public class PhraseServletFilter implements javax.servlet.Filter{
  /*irrelevant code omitted*/
  @Override
  public void doFilter(ServletRequest request, ServletResponse response,
                FilterChain chain) throws IOException, ServletException {
    System.out.println("Intercepted a request for URL " +
    ((HttpServletRequest)request).getRequestURL().toString());
    System.out.println("Query: " +
    ((HttpServletRequest)request).getQueryString());
    System.out.println("Remote address: " + request.getRemoteAddr());
    try {
      chain.doFilter(request, response);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

The result

- Here is the output after a few requests

```
Intercepted a request for URL http://localhost:8080/give.phrase
Query: null
Remote address: 0:0:0:0:0:0:0:1
Intercepted a request for URL http://localhost:8080/give.phrase
Query: phrase_category=bullshit
Remote address: 0:0:0:0:0:0:0:1
```

More relevant: check for authentication

- Suppose you have an application with a few public pages and a few pages for which authentication is required.
- This is a typical use case for WebFilter
- See next slide (only relevant code displayed)

Authentication Filter

```
//use "/*" for catch-all filters
@WebFilter(urlPatterns = {"/user.dashboard", "/list.secrets"})
public class AuthenticationFilter implements Filter {
  @Override
  public void doFilter(ServletRequest request,
                       ServletResponse response,
                       FilterChain chain) throws IOException, ServletException {
    if (request instanceof HttpServletRequest) {
      HttpServletRequest req = ((HttpServletRequest)request);
      System.out.println("[AuthenticationFilter] Intercepted URL: " +
                         req.getRequestURL().toString());
      final HttpSession session = req.getSession();
      if (session.getAttribute("user") == null) {
        System.out.println("[AuthenticationFilter] no authenticated user:
                           redirecting to /login");
        ((HttpServletResponse)response).sendRedirect("/login");
      } else {
        System.out.println("[AuthenticationFilter] authenticated status checked;
                           user= " + session.getAttribute("user"));
        try {
          chain.doFilter(request, response);
        } catch (Exception e) {
          e.printStackTrace();
        }
      }}}}
```

AuthenticationFilter in action

- Here is the output after a few requests

```
[AuthenticationFilter] Intercepted URL: http://localhost:8080/list.secrets
[AuthenticationFilter] no authenticated user: redirecting to /login

#logged in as Henk

[AuthenticationFilter] Intercepted URL: http://localhost:8080/list.secrets
[AuthenticationFilter] authenticated status checked; user= User{name='Henk',
email='henk@example.com', password='null', role=USER}
[AuthenticationFilter] Intercepted URL: http://localhost:8080/user.dashboard
[AuthenticationFilter] authenticated status checked; user= User{name='Henk',
email='henk@example.com', password='null', role=USER}
```

@WebServlet annotations

- Here are all possible annotation parameters

```
@WebFilter {
    description="";                 // Descriptive text about the servlet.
    WebInitParam[] initParams={}, // Used to pass filter config parameters.
    filterName="",                  // Name of the filter.
    [] servletNames={},             // Array of Servlet names to which this
                                    // Filter applies.
    [] urlPatterns={},              // Used for multiple URL and other
                                    // attributes set (Required).
    DispatcherType[] dispatcherTypes={DispatcherType.REQUEST},  // Look at
                                    // table below.
    boolean asyncSupported=false} // Specifies asynchronous processing
                                    // or not.
/*
Possible dispatcher types:
REQUEST: Only when the request comes directly from the client. (DEFAULT)
ASYNC:   Only when the asynchronous request comes from the client
FORWARD: Only when the request has been forwarded to a component
INCLUDE: Only when the request is being processed by a component that
         has been included
ERROR:   Only when the request is being processed with the error page
         mechanism
*/
```

A simple web service

Why web services

Modern web applications are often composed of a "main" backend serving the view and "helper" backends serving only data for (updating) parts of the view

A very simple example

- Here you will only see a very simple example, using a dedicated servlet and the GSON (Google JSON) library to convert Java objects to JSON (JavaScript Object Notation).
- In the module "Webbased Information Systems 2" you will get to know industry-strength techniques for building Service Oriented Architectures using REST

JSON

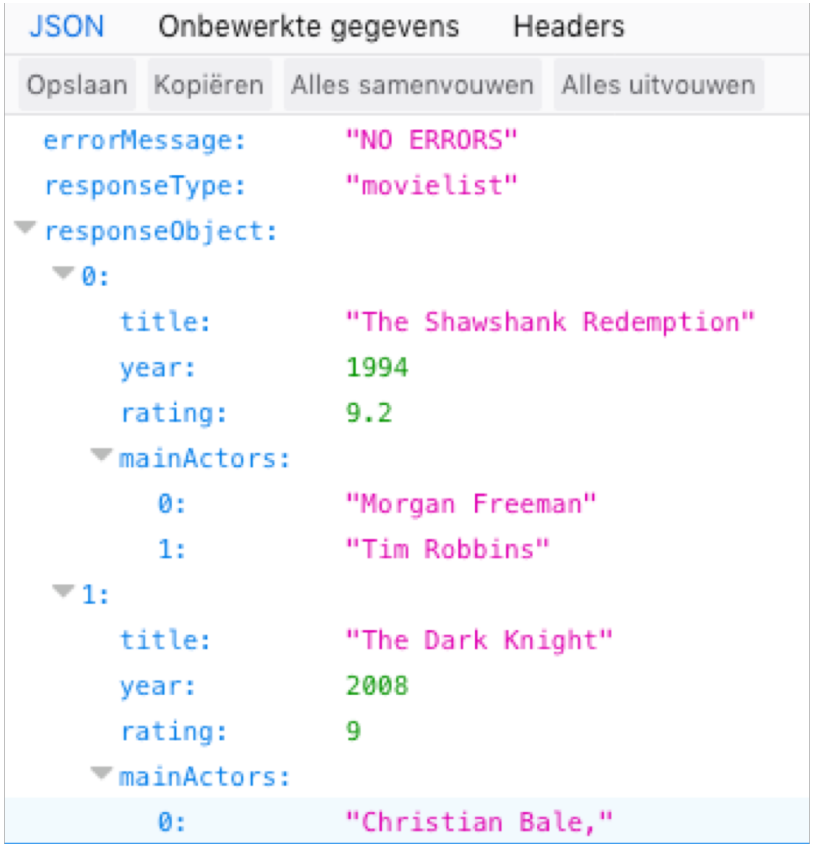
- JSON – JavaScript Object Notation is the best format to exchange data from one system to another
- It is very much like a complex map literal:

```
{"errorMessage": "NO ERRORS",
  "responseType": "movielist",
  "responseObject": [
    {"title": "The Shawshank Redemption",
     "year": 1994,
     "rating": 9.2,
     "mainActors": ["Morgan Freeman", "Tim Robbins"]
    },
     "title": "The Dark Knight",
     "year": 2008,
     "rating": 9.0, "mainActors": ["Christian Bale,", "Heath Ledger"]
    }
  ]
}
```

JSON in the browser

- Most browsers have a nice default representation of JSON, as long as you specify the correct content-type:

```
response.setContentType("application/json;charset=UTF-8");
```

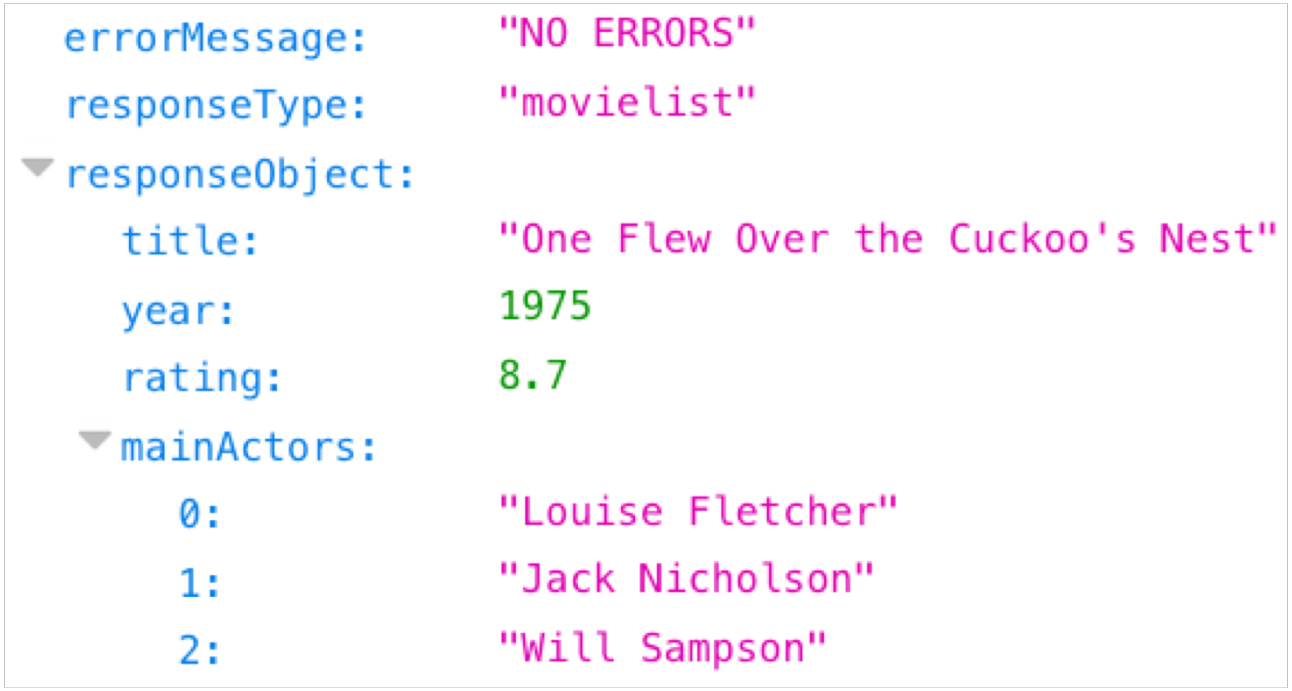


A JSON serving servlet

```
protected void doGet(...) throws ... {
    String requestType = request.getParameter("request_type");
    response.setContentType("application/json;charset=UTF-8");
    final PrintWriter writer = response.getWriter();
    JsonResponse jsonResponse = new JsonResponse();
    if (requestType.equals("random")) {
        List<Movie> movies = Movie.getAllMovies();
        Movie selected = movies.get((int)(Math.random()*movies.size()));
        jsonResponse.responseObject = selected;
    }
    Gson gson = new Gson();
    writer.write(gson.toJson(jsonResponse));
    writer.flush();
}
//convenience inner class
private static class JsonResponse {
    String errorMessage = "NO ERRORS";
    String responseType = "movie";
    Object responseObject;
}
```

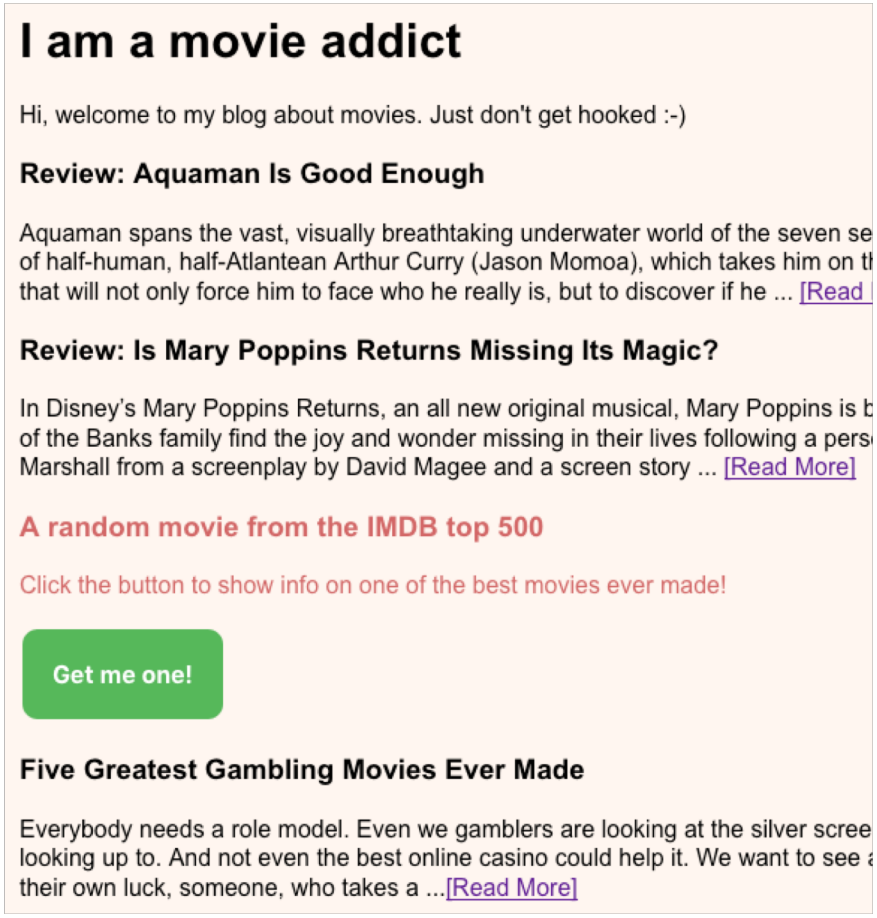
A JSON serving servlet

```
http://localhost:8080/movie_service?request_type=random
```



Consuming the data: AJAX

- Suppose you have a site that is about movies, and you want to show the information of a random movie, updating it every 5 seconds
- Here is the view



Consuming the data: AJAX

- The simple html:

```
<div>
    <h3><span>A random movie from the IMDB top 500</span></h3>
    <div id="random_movie">
        <span>Click the button to for the best movies ever made!</span>
        <br/>
    </div>
    <button id="get_random_movie">Get me one!</button>
</div>
```

- The Javascript of the front-end is goin to do the job for us:

Consuming the data: Javascript

- First, we need jQuery:

```
<script
        src="https://code.jquery.com/jquery-3.3.1.min.js"
        integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
        crossorigin="anonymous">
</script>
```

Processing the AJAX response

```
function processMovie(result) {
  var movie = result.responseObject;
  var movieHtml = "<h3>" + movie.title + " (" + movie.year + ")</h3>";
  movieHtml += "IMDB rating: " + movie.rating + "<br />Main actors <br />";
  $.each(movie.mainActors, function(i, actor){
    movieHtml += "  " + (i+1) + ". " + actor + "<br />";
  });
  movieHtml += "<br />"
  $("#random_movie").html(movieHtml);
  $("#get_random_movie").text("Get another one!");
}

function getRandomMovie() {
  var movie_url = "movie_service?request_type=random";
  jQuery.getJSON({
    url:movie_url,
    success: function(result) {
      processMovie(result);
    }
  });
};

$("#get_random_movie").click(getRandomMovie);
```

Note on HTML from JS

- Don't do it like this!

File Upload

File Upload is an essential feature

- Many websites provide some form of upload functionality:
  - Photos & Video
  - Data
  - Configuration
- Uploading is boilerplate code
- …But what will you do with the file?
  - Into database
  - Store locally
  - Process directly

An upload html form

- Suppose we want to offer a scv data upload
- This is the html for uploading csv data

```
<form th:action="@{'data_upload'}"
      method="post"
      enctype="multipart/form-data">
    <input type="file" name="scv_data" accept="text/csv">
    <input type="submit" value="Upload" />
</form>
```

An upload servlet

- Here is the class annotation of this servlet (explained on next slide)

```
@MultipartConfig(location="/tmp",
        fileSizeThreshold = 1024 * 1024,
        maxFileSize = 1024 * 1024 * 5,
        maxRequestSize = 1024 * 1024 * 5 * 5)
@WebServlet(name = "FileUploadServlet", urlPatterns =
"/data_upload")
public class FileUploadServlet extends HttpServlet {...
```

An upload servlet

- `location`: An ***absolute*** path to a directory on the file system. This location is used to store files temporarily while the parts are processed or when the size of the file exceeds the specified fileSizeThreshold setting. The default location is "".
- `fileSizeThreshold`: The file size in bytes after which the file will be temporarily stored on disk. The default size is 0 bytes.
- `maxFileSize`: The maximum size allowed for uploaded files, in bytes. If the size is greater, the container will throw an exception (IllegalStateException). The default size is unlimited.
- `maxRequestSize`: The maximum size allowed for a multipart/form-data request, in bytes. The web container will throw an exception if the overall size of all uploaded files exceeds this threshold. The default size is unlimited.

Servlet annotation or web.xml

- Instead of hardcoding the constraints, you can also put it in web.xml. This tutorial uses web.xml.

```
<servlet-mapping>...</servlet-mapping>
<servlet>
    <servlet-name>FileUploadServlet</servlet-name>
    <servlet-class>....FileUploadServlet</servlet-class>
    <init-param>
        <param-name>upload_dir</param-name>
        <param-value>/Users/michiel/tmp/upload</param-value>
    </init-param>
    <multipart-config>
        <location>/tmp</location>
        <!-- all in bytes -->
        <max-file-size>20848820</max-file-size>
        <max-request-size>418018841</max-request-size>
        <file-size-threshold>1048576</file-size-threshold>
    </multipart-config>
</servlet>
```

Specify your final upload location

- You do this in your servlets `init()` of course, and read from web.xml **`init-param`**.

```
public class FileUploadServlet extends HttpServlet {
    private String uploadDir;

    @Override
    public void init() throws ServletException {
        this.uploadDir = getInitParameter("upload_dir");
    }
...
```

First a little check

```
//inside doPost()
File fileSaveDir = new File(this.uploadDir);
    if (! fileSaveDir.exists()) {
        throw new IllegalStateException("Upload dir does not exist: " +
                                        this.uploadDir);
    }
...
}
```

HTTP Status 500 – Internal Server Error

Type Exception Report

Message Upload dir does not exist: /Users/michiel/tmp/uploads

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

```
java.lang.IllegalStateException: Upload dir does not exist: /Users/michiel/tmp/uploads
        nl.bioinf.wis_on_thymeleaf.servlets.FileUploadServlet.doPost(FileUploadServlet.java:53)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:660)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
```

Note The full stack trace of the root cause is available in the server logs.

Do the actual upload

```
//inside doPost()

File fileSaveDir = new File(this.uploadDir);
...

//Do this only if you are sure there won't be any file name conflicts!
//An existing one will simply be overwritten
String fileName;
for (Part part : request.getParts()) {
    fileName = part.getSubmittedFileName();
    part.write(this.uploadDir + File.separator + fileName);
}

//go back to the upload form
WebContext ctx = new WebContext(...);
ctx.setVariable("message", "upload successfull, wanna do another on?");
templateEngine.process("upload_form", ctx, response.getWriter());
```

Create a safe file name

- If you want to be sure files won't get overwritten, you can use random generated file names
- Of course, you'll have to keep track of which file belongs to whom with what original name

```
//the safe way, with a generated file name becomes something like
//my_app_upload14260971264207930189.csv
File generatedFile = File.createTempFile("my_app_upload", ".csv");
for (Part part : request.getParts()) {
    part.write(this.uploadDir +
               File.separator +
               generatedFile.getName());
}
```

Review of topics

- Web filters
- Serving and consuming AJAX data
- File Upload