Delft University of Technology

Faculty Electrical Engineering, Mathematics and Computer Science

---

Managing XML Database with eXist
Web Data Management (IN4331)

Bouke Nederstigt 4008812
Abhishek Sen 4319850

Sunday 1st June, 2014

Contents

List of Figures

1 Introduction

For this project we chose to investigate the use of eXist as an XML database. This report gives an overview of the different experiments that we ran and their results. Some code and query excerpts will be shown in the report but the entire source code will be available at the following GitHub repository:

https://github.com/bouke-nederstigt/webdatamanagement

The repository also contains a README file that contains source code installation instructions for each of the applications discussed below.

2 XPATH and XQUERY Experiment Results

This sections lists the XPATH and XQUERY commands execute to complete the exercises from Chapter 5 of the Web Data Management textbook. The output generated from these commands is shown in the corresponding Github directories. The commands were tested using the *movies.xml* XML database provided in the textbook.

2.1 XPATH Query Output

Title

/movies//title

All movie titles

/movies//title/text()

Title of the movies published after 2000

/movies/movie[year¿2000]/title/text()

Summary of "Spider-man"

/movies/movie[title="Spider-Man"]/summary/text()

Who is the director of Heat?

concat(/movies/movie[title="Heat"]/director/firstname/text()

Title of the movies featuring Kirsten Dunst

/movies/movie[actor/firstname="Kirsten" and actor/lastname="Dunst"]/title/text()

Which movies have a summary?

/movies/movie[summary]/title/text()

Which movies do not have a summary?

/movies/movie[not(summary)]/title/text()

Titles of movies published more than 5 years ago

/movies/movie[year¡2009]/title/text()

What was the role of Clint Eastwoord in Unforgiven?

/movies/movie[title="Unforgiven"]/actor[firstname="Clint" and lastname="Eastwood"]/role/text()

What is the last movie of the document?

/movies/movie[last()]

Title of the film that immediatly precedes Marie Antoinette in the document?

/movies/movie[title="Marie Antoinette"]/preceding-sibling::*[1]/title/text()

Get the movies whose title contains a "V"
/movies/movie[contains(title, "V")]/title/text()

Get the movies whose cast consist of exactly three actors
/movies/movie[count(actor) = 3]

2.2 XQUERY Output

List the movies published after 2002, including their title and year

```
<results> {
let $ms := doc("movies/movies_alone.xml"), $as := doc("movies/artists_alone.xml")
for $a in $ms//movie where $a/year > 2002
return
<result> {$a/title} {$a/year} </result>
} </results>
```

Create a flat list of all title-role pairs, enclosed in a "result element

```
<results>{
let $ms := doc("movies/movies_alone.xml"),
$as := doc("movies/artists_alone.xml")
for $t in $ms//movie
return
    for $a in $t//actor
    return
        <result>
            {$t/title}
            <role>{string($a/@role)}</role>
        </result>
}
</results>
```

Give the title of movies where the director is also one of the actors

```
let $ms := doc("movies/movies_alone.xml"),
$as := doc("movies/artists_alone.xml")
for $t in $ms//movie
where $t/director/@id = $t/actor/@id
return
```

```
$t/title/text()
```

Show the movies, grouped by genre

```
<results>{
let $ms := doc("movies/movies_alone.xml"),
$as := doc("movies/artists_alone.xml")
for $genre in distinct-values($ms//movie/genre)
let $t := $ms//movie[genre=$genre]
return
    <result> <genre> {$genre} </genre> {$t} </result>
}</results>
```

For each distinct actor's id, show the titles of the movies where this actor plays a role

```
<results>{
let $ms := doc("movies/movies_alone.xml"),
$as := doc("movies/artists_alone.xml")
for $actorId in distinct-values($ms//movie/actor/@id)
let $t := $ms//movie[actor/@id = $actorId]/title
return
    <actor> {$actorId}, {$t} </actor>
}</results>
```

Variant only showing actors playing in at least two movies

```
<results>{
let $ms := doc("movies/movies_alone.xml"),
$as := doc("movies/artists_alone.xml")
for $actorId in distinct-values($ms//movie/actor/@id)
let $t := $ms//movie[actor/@id = $actorId]/title
return
    if(count($t) > 1) then
        <actor> {$actorId}, {$t} </actor>
    else
        ()
}</results>
```

Give the title of each movie, along with the name if its director

```
<results>{
let $ms := doc("movies/movies_alone.xml"),
$as := doc("movies/artists_alone.xml")
for $movie in $ms//movie, $director in $as//artist[@id = $movie/director/@id]
return
    <result>
        {$movie/title}
        <director>
            {$director/first_name}
            {$director/last_name}
        </director>
    </result>
}</results>
```

Give the title of each movie, and a nested element giving the list of actors with their role

```
<results>{
let $ms := doc("movies/movies_alone.xml"),
$as := doc("movies/artists_alone.xml")
for $movie in $ms//movie
return
    <result>
        {$movie/title}
        <actors> {
        for $a in $movie/actor, $actor in $as//artist[@id = $a/@id]
     return
       <actor>
           {$actor/first_name}
           {$actor/last_name}
           <role>{string($a/@role)}</role>
       </actor>
     } </actors>
    </result>
}</results>
```

For each movie that has at least two actors, list the title and firs two actors, and an empty

```
<results>{
let $ms := doc("movies/movies_alone.xml"), $as := doc("movies/artists_alone.xml")
for $movie in $ms//movie
let $count := count($movie/actor)
return
    if($count > 1) then
        <result>
            {$movie/title} {
             for $a in subsequence($movie/actor, 1, 2), $actor in $as//artist[@id =
         return
                <actor>
                {$actor/first_name/text()} {$actor/last_name/text()} as {string($a/
                </actor>
         }
         {if($count > 2) then
             <et-al/>
         else
             ()
         }
        </result>
    else
        ()
}</results>
```

List the titles and years of all movies directed by Clint Eastwood after 1990, in alphabetical

```
<results>{
let $ms := doc("movies/movies_alone.xml"),
$as := doc("movies/artists_alone.xml")
for $actor in $as//artist[first_name="Clint" and last_name="Eastwood"]
    for $movie in $ms//movie[director/@id = $actor//@id]
    where $movie/year > 1990
    order by $movie/title
    return
        <result>
            {$movie/title}
            {$movie/year}
        </result>
}</results>
```

3 Movie Application

For this part of the project we were asked to implement a simple movie lookup application from an XML database. Figure 1 below shows a screenshot of the movie application running on the eXist environment on the local machine. The application was written in the eXist IDE environment and all the code for this application can be found on GitHub link.

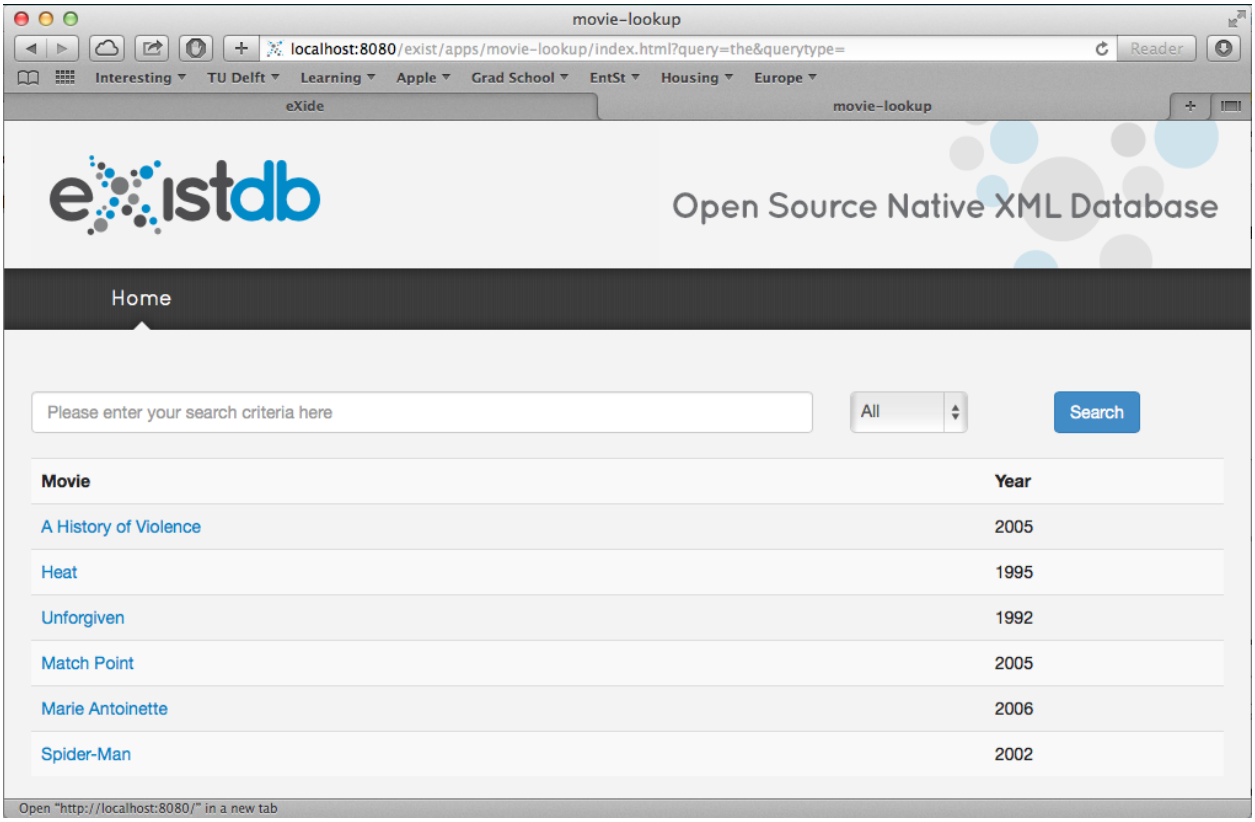

Figure 1: Movie Application

3.1 Search Criteria

The search bar allows users to search movies with the following options:

- Title
- Keywords
- Year

- Director
- Actor
- Genre

3.2 Selecting Movie From Returned Results

If matching movies are found, a list is returned showing the movie title and release year. More details about the movie can be found by clicking on the movie title as shown in Figure 2.

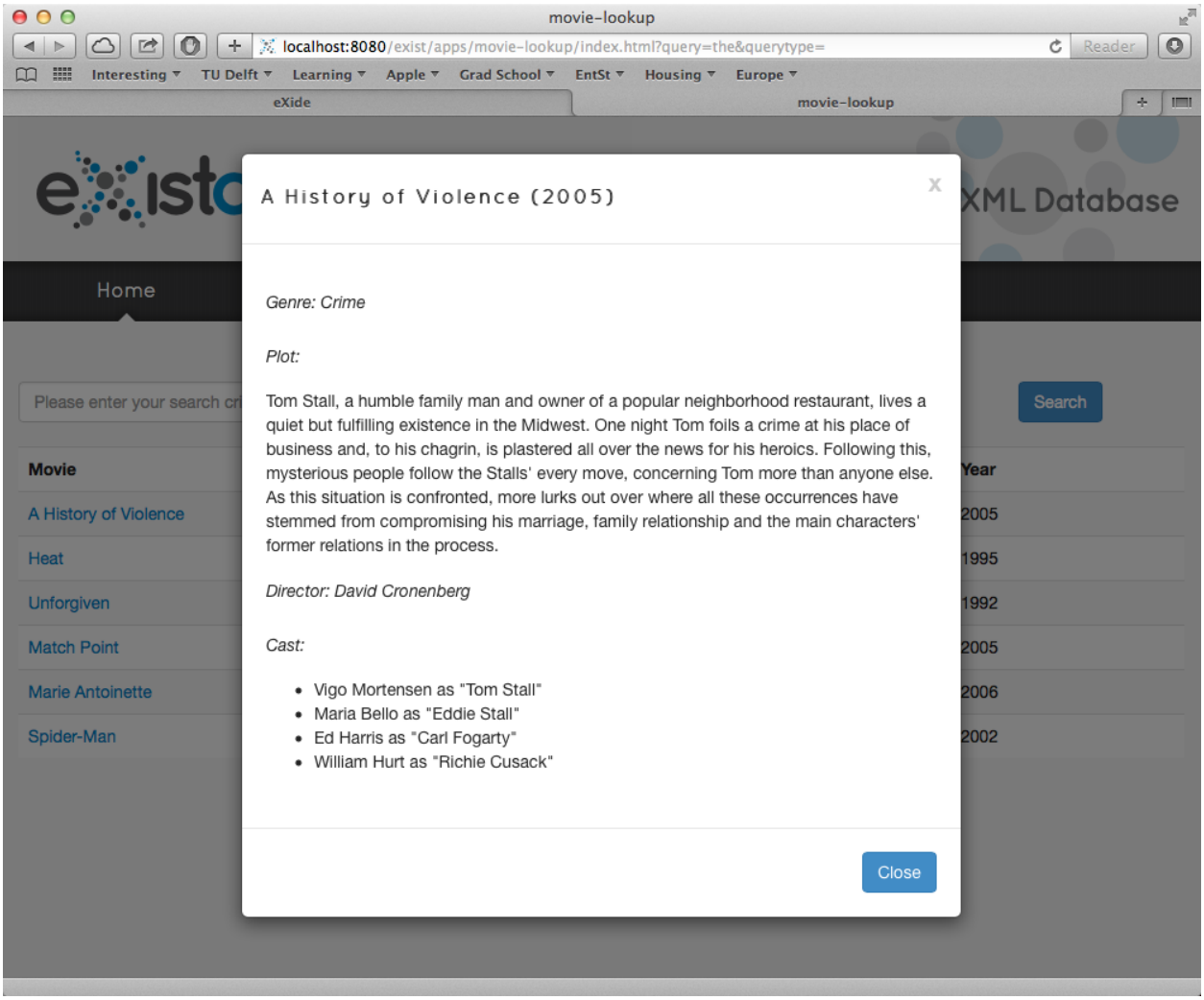

Figure 2: Additional Movie Information

4 Shakespeare Application

The Shakespeare application allows users to browse through a Shakespearean play and analyze its content, read some specific parts and maybe find related information. The application was written in the eXist IDE environment and all the code can be found on GitHub link from section 1. Figure 3 shows a screenshot of the homepage for the application with the available plays listed in alphabetical order.

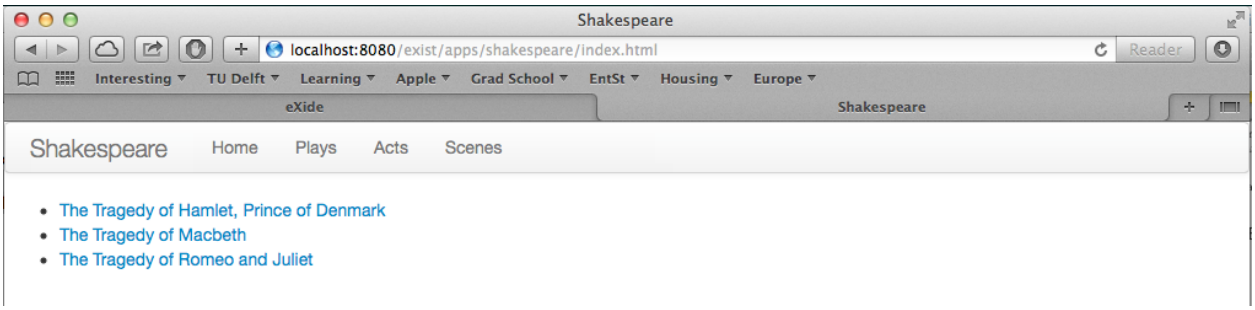

Figure 3: Shakespeare Application

4.1 Architecture

The architecture for the Shakespeare is detailed below:

1. Navigation panel - Allows user to browse between plays, acts, scenes and characters
2. Contents - When a play is selected, the contents pages shows the acts, scenes and characters involved in the play
   (a) Act - Shows the entire act of the play
   (b) Scene - Shows the entire scene from the act of the selected play with characters involved in that scene shown at the top of the page.
   (c) Character - Shows every act/scene where the character speaks
      i. Selecting a specific scene/act where the character has lines opens up a drop-down panel that shows all of the character's lines from that act/scene
3. Scene/Act - Character links present in the layout show all the lines of that character from all scenes/acts

Figure 4 below shows the table of contents for Macbeth. The acts, scenes and characters are listed.

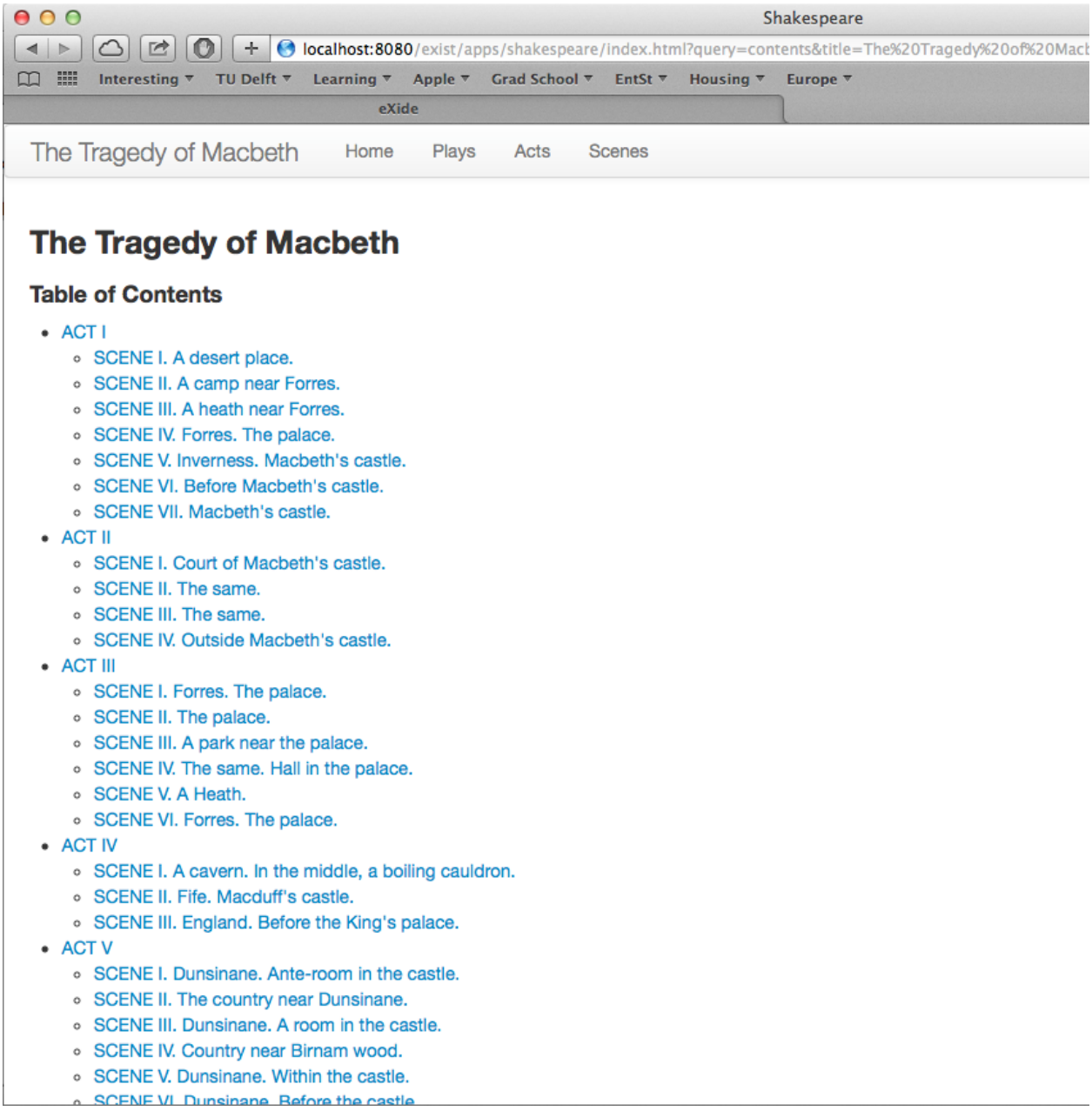

Figure 4: Macbeth Contents Page

4.2 Character Lines Display

Figure 5 shows Macbeth's lines from Act V, Scene VIII in a drop-down panel box. The navigation panel at the top of the page is always accessible to the user and they can always jump between scenes, acts and/or characters once a play is selected.

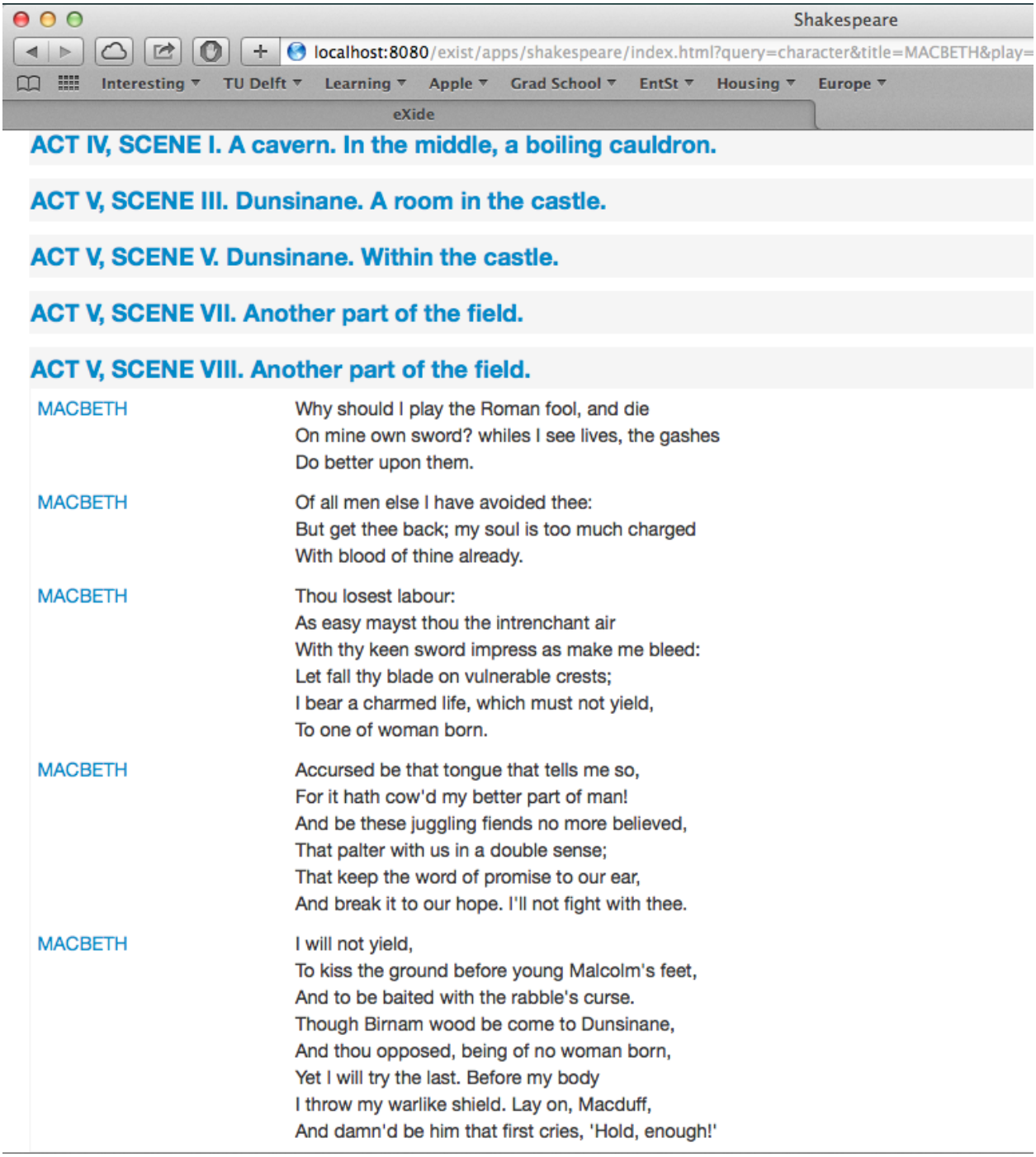

Figure 5: Macbeth's lines from Act V, Scene VIII

5 MusicXML Online Application

The purpose of this application is to offer a platform where users can convert their music XML files to sheet music and sound and extract useful information. This way they can listen to the music and also search through its contents. This section describes the application's architecture and the results of the experiment. In conclusion, some future extensions to the application are also described.

5.1 Architecture

The application is built as a web application using Java (JDK 1.7). When XML files are uploaded, the musicxml2ly command is automatically executed on the command line to convert the XML file to a lilypond (.ly) file. Once completed, the command line version of Lilypond is invoked to convert the .ly file to a pdf and a MIDI-audio file. The backend of the application consists of three classes described below:

- Exist - Database wrapper which handles connecting to the database, uploading XML files to the database and executing queries. Most of this code was taken from the textbook and/or the eXist website and then slightly modify to suit the specific needs of the application.
- XMLFileUploadServlet - This servlet deals with the heavy lifting around uploading files. After an uploaded file is saved on disk the servlet also uploads the content of the XML file to the eXist database (using Exist.java). It also immediately converts the .xml file to an audio file and a pdf.
- MusicXMLOnline - Being the backbone of the application, the MusicXMLOnline class offers functions to retrieve all documents and documentTitles, to extract the lyrics from a document and to search the lyrics and titles of the songs in the database.

To deliver all user content, .jsp files are used. The first jsp file is used for displaying the available files and search results, the second one to upload the files, and the third file handles the case where users can listen to the music and read the lyrics. To easily create a powerful user experience that works on mobile phones as well as traditional browsers, the bootstrap framework is used.

5.2 Results

After working hard on the application, the basic functionality that was planned was also completed. It is possible for users to upload and convert XML files, search through these files, view the sheet music, and also listen to the music while reading the lyrics.

As can be seen from the Figures 6 and 9 the functionality for displaying an overview of the songs and the sheet music works exactly as expected.

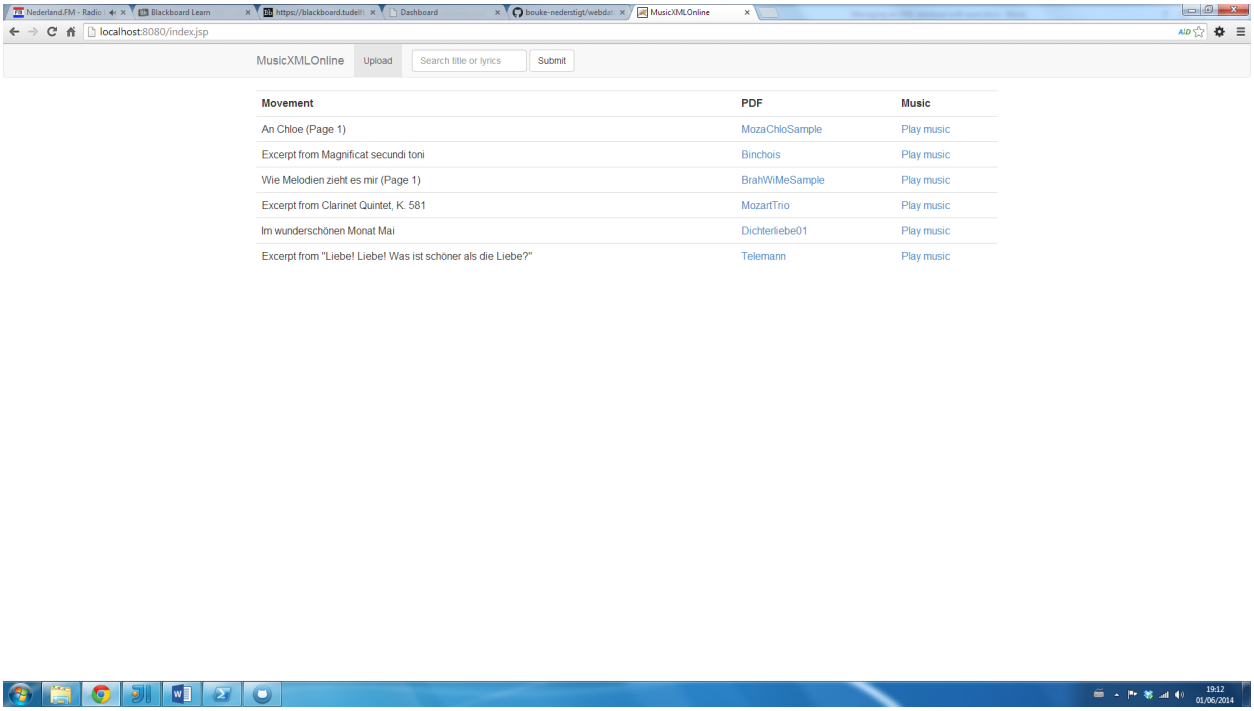

Figure 6: Music Online Contents Page

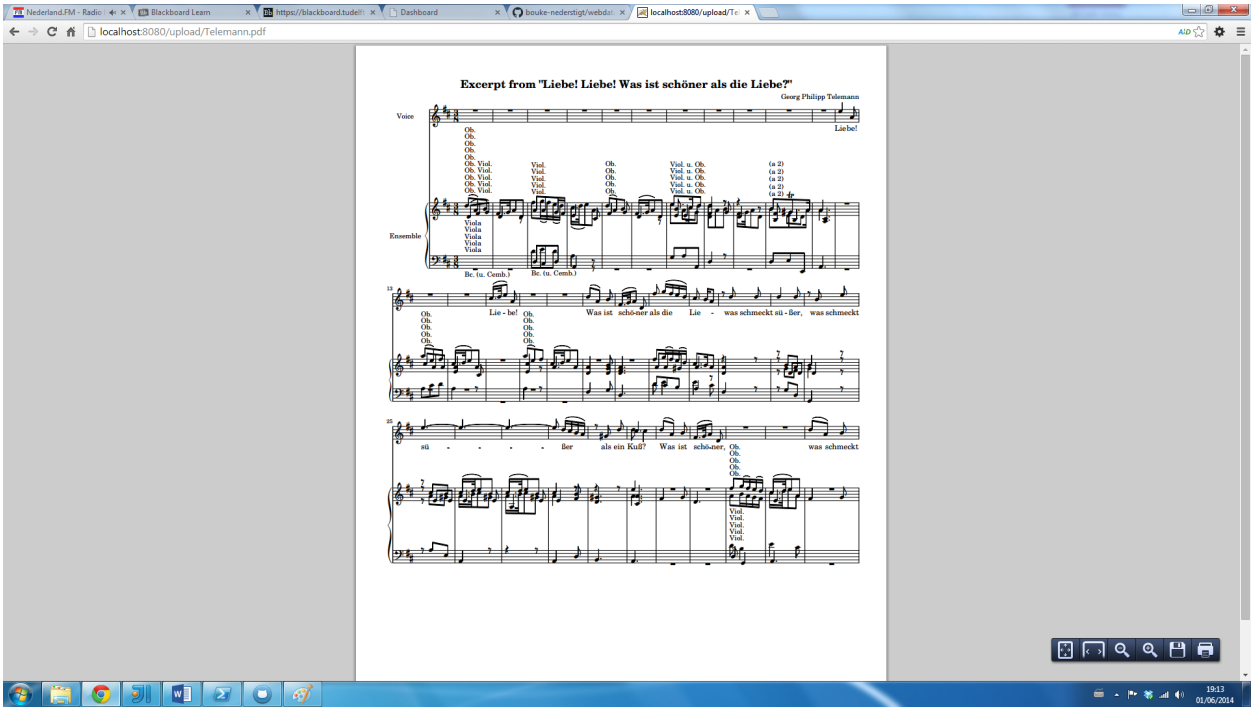

Figure 7: Additional Movie Information

The functionality for displaying the lyrics and searching through the songs is however slightly more limited. There are two main drawbacks that are described below:

1. The search functionality is quite rudimentary. When a user searches for a string and all the words in the string are found in either the lyrics, or the title the song is returned. A better search function would however also keep in mind thinks like capitalization, or offer the possibility to combine words from the title and lyrics. It could also be possible to include e.g. author information. The output results are shown in Figure 8.

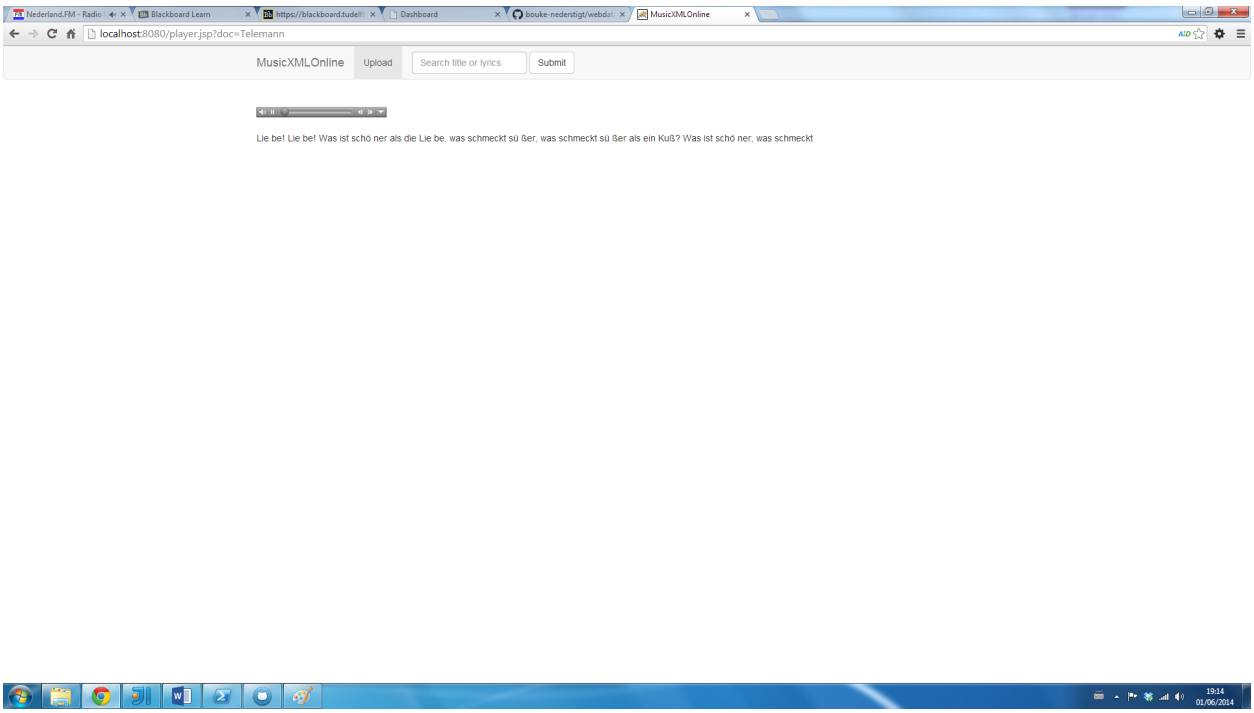

Figure 8: Song Lyrics Search

2. All of the lyrics are retrieved from the database, but the words are not always put together correctly. Because the words are sometimes split over different notes they are not always retrieved in one piece - Figure 9 shows this problem.

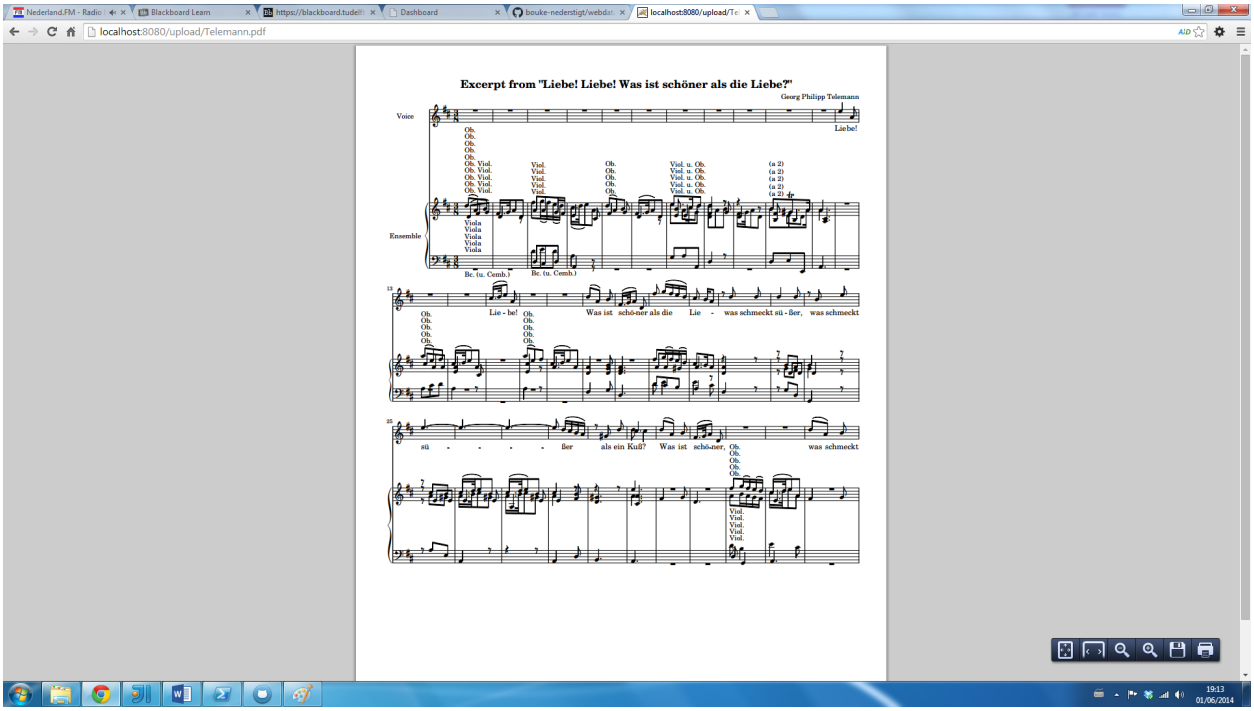

Figure 9: Additional Movie Information

6 Future Improvements to MusicXML Online Application

It was nice to conduct an experiment where all the tools for converting music XML files into pdf and audio were put together. A lot of future extensions could be added to make this web application more robust and they are listed as follows:

- Highlighting lyrics when a song is playing
- Extraction of extra information like melody or author information
- Creating a more elaborate and robust search functionality by correctly retrieving the song lyrics
- Improved file upload procedure
- Currently the application only accepts XML files but MXL files seem to be a more common format. The implementation of this could be done within a short timeframe because the mxl files are basically just zipped XML files. Unfortunately it was not possible to achieve this within the deadline of the project.
- Some steps would need to be taken before the application could be used in a production environment. User input should be validated (right now files are uploaded straight to the server) and error handling could also be improved.