Harvard Extension School
December 2015

CSCIE 18: Final Project / Fall 2015

Team Name: xTeam

Team Members: Frank Krazer
Gwong Long
Otto Luna
Dimitri Olchanyi
Boris Rugel

I. Introduction

The website developed by the xTeam allows users to easily and efficiently search the catalog of courses offered by the Harvard University Faculty of Arts & Sciences (FAS). This report presents an overview of how the website was developed, including information on the website's browse and search features, its structure and design, as well as the various XML technologies employed by its developers. In addition to a fully-featured HTML platform, the site also includes content in Mobile and PDF formats. The Bootstrap framework was used to create a common layout and design for each page and was also employed to achieve a mobile-ready responsive design.

The application was built using eXistdb, an open-source, native XML database.
To deploy the application:

- Clone the Git repository located at: https://github.com/cscie18fall2015/xteam-finalproject
- Take the `cscie18-xteam-finalproject-0.1.xar` file located in the Build directory and upload to eXistdb using the Package Manager.
- Go to http://localhost:8080/exist/apps/cscie18-xteam-finalproject/index.html to access the website's landing page.
- If the home page (or any other page) takes a long time to load (more than 30 seconds), make a small modification to the.xconf file located in the Data directory (for example, add a line of empty space) and save it. After this modification, indexing will be reapplied to the collection of XML source files and all pages should load significantly faster.

***Special note regarding XML source files**: The source data had some inconsistent formatting of special characters in the description and notes elements in the form of*

"double" escaping. The source data was corrected by processing it outside of Exist using R Studio. Attempts to do this correction on the fly in XQuery or XSL were unsuccessful in Exist. When processed by Oxygen it can be corrected via XSL using disable-output-escaping="yes". It would probably be possible to make these corrections on the fly in Exist using a custom Java function.

Example:
Course Title: Africa Rising? New African Economies/Cultures and Their Global Implications

This course has 2 separate xml files containing what should be identical descriptions. They differ only in the special characters. The special characters in the description for 209A are not encoded or escaped at all, while 209B they are "double" escaped.

…the "transformative... vs. …the "transformative…

At first the "&quot" looks correct, however the source xml file has """. When the HTML is transformed, only the initial "&" is decoded leaving it as "&quot" in the displayed text.

Example XML Files: 108680-11071-2015-fall.xml, 108694-11143-2015-spring.xml

II. HTML Site

The HTML site was tested and validated against HTML5 using https://validator.w3.org. HTML was validated via file upload.

Special note: the site's landing page (index.html) does not validate against HTML5 because of eXistdb's parser. This parser uses XML Schema, not DTD, to validate the legitimacy of the entered code. XML Schema does not recognize Doctype statements and for this reason there is no such statement at the top of index.html.

A. Browse and Search

The site's landing page presents the three following search options:

Explore courses by faculty departments: clicking on this options takes users to a "Departments" page (http://localhost:8080/exist/apps/cscie18-xteam-finalproject/departments) that presents them a hyperlinked list of all academic departments in alphabetical order. If the department and course group string are the same, then only the department is listed (e.g. "Computer Science"). If a department has multiple course groups, then the course groups are listed as children (nested list) of the departments. An example of this is the Romance Languages and Literatures Department, which has course groups for the various Romance languages

(French, Italian, Spanish, etc.). Clicking on a department or course group takes users to a page listing all courses available in that department or course group.

Advanced Search: clicking on this option takes users to a page (http://localhost:8080/exist/apps/cscie18-xteam-finalproject/form) that allows them to search by multiple fields, including:

- Term: users are provided a dropdown menu. Choices include: “Spring,” “Fall” and “Any.”
- Department: users are presented a search box. An integrated Autocomplete widget (see “Extraordinary Distinction” section” below) suggests Department names as users begin to type their search term(s).
- Day of Week -- check boxes are presented. Single or multiple days can be selected. There is also a choice for “Any” day.
- Time: allows users to search by class start time in HH:MM format.
- Staff: users are presented a search box. An integrated Autocomplete widget (see “Extraordinary Distinction” section” below) suggests staff member names as users begin to type their search term(s).
- General search -- a search box is presented that allows users to search for keywords contained in the course description and/or course title. This enables users to quickly find courses on “leadership,” for example, or any other topic.

Explore faculty staff by departments: clicking on this option takes users to a page (http://localhost:8080/exist/apps/cscie18-xteam-finalproject/staff) that allows them to search for courses by staff name. A list of all departments is presented. Clicking on a department name opens a drop-down menu with a list of staff members belonging to that department. Clicking on a name takes users to a page listing courses taught by the chosen staff member.

B. Structure and Design

The HTML site includes the following pages (in addition to the three search pages described above):

- **Home/Landing Page**: presents the three search options mentioned above.

  Corresponding files: main_page.xq, main_page.xsl and main_page-fo.xsl.

- **Course Listings Page**: includes a hyperlinked list of all courses resulting from the query parameters entered via the search pages. Basic information for each course is presented in tabular format with responsive

design implemented for mobile view. Fields displayed include: title, term, day of week, and class start and end time. Results are “paged” in groups of 25 or less. The paging feature is controlled via Bootstrap. Clicking on a course takes users to the Course Details Page. A link presents the option of displaying course listings information in PDF format. A link is also included that takes users back to the home/search page.

Corresponding files: course_groups.xq, course_groups.xsl and course_groups-fo.xsl.

- **Course Details Page**: includes detailed information for the selected course. Fields displayed include: Title, Short title, Description, Instructor(s), class start time and end time, etc. Only fields deemed of value for the user were included. Less useful fields, such as “Seniority sort,” were excluded for the sake of clarity.

  Data is broken up into 4 tables:
  - ➢ Table displaying most relevant course information, including Title, Description and Department.
  - ➢ Table displaying meeting information. This is a collection, so the table rows are added by the XSL template iteration.
  - ➢ Table displaying staff information. This is a collection, so the table rows are added by the XSL template iteration.
  - ➢ Table displaying other details, like the course ID. This table structure is static.

  A link presents the option of displaying course information in PDF format. A link is also included that takes users back to the home/search page.

  Corresponding files: course_details.xq, course_details.xsl and course_details-fo.xsl.

All of the HTML pages discussed above have a common layout and design. This was achieved via the development of a **common.xsl file** that contains all repetitive <head> and <body> elements, as well as references to CSS and JavaScript files necessary for the Bootstrap framework and JQuery widgets, and references to local JavaScript scripts and CSS documents. This file also provides a common responsive menu located on the top-right of the screen. This menu allows users to select one of the three search modes to initiate a new query: search by Department, Advanced Search, and Staff search. In addition, this file provides basic site navigation through breadcrumbs and a "back to top" button at the bottom of the page. Finally, the “Harvard University Faculty of Arts & Sciences” banner at the top of each page hyperlinks to the homepage of the FAS

website (fas.harvard.edu). Other xsl files reference the common.xsl file via the xsl:include element.

III. Mobile Site

For the site's mobile platform, we utilized the Bootstrap framework, which provides an easy-to-implement responsive web design that adjusts the site's content dynamically to fit varying screen sizes. The main approach was to use attributes, both "hidden" and "visible," on crucial design elements to achieve favorable presentation on screens under 480px and to rely on Bootstrap's built in responsive functionality on bigger screens. Because of issues with the W3C Mobile OK Checker, we checked the usability of the mobile site simply by resizing the browser window. We believe our site is compatible with screen widths greater than 350px and less than 1200px, making it suitable for viewing on mobile devices from all major manufacturers, including: Amazon, Apple, Blackberry, LG, Nokia,Samsung and others.

In addition to Bootstrap, we followed some basic web design principles to achieve a mobile-friendly application, including:

- Navigation is simple and intuitive. Menus are minimalistic. Breadcrumbs are provided.
- It is easy for users to return to the home page or start a new search.
- Search results are relevant and accurate.
- Enhanced site search by adding an auto-complete feature.
- Implemented an "Advanced Search" feature that enables users to easily create complex queries and thus achieve more relevant search results.
- Advanced search form provides placeholder text and help text to assist the user.
- Search oriented design - all relevant site content is presented as hyperlinks that allow further content exploration.
- Content was organized vertically, making it easier for users to scroll through the website on mobile devices.
- Large, fixed-width elements have been avoided.
- Calls-to-action that open new windows are avoided.

IV. PDF Content

Every page of the HTML site that displays search results has a PDF equivalent. For instance, after a user searches for courses in a particular department and the results are displayed in the Course Listings HTML Page, a button is made available to view the results in PDF format ("View as PDF" button). For long PDF documents containing large amounts of course data, a hyperlinked Table of Contents is provided for ease of navigation.

In addition, the “Departments” page features a “View as PDF” button with a drop-down menu that allows users to select a particular department. This generates a PDF showing all courses in the particular department chosen. The PDF includes two sections: a Table of Contents, and a section containing course details for each course.

V. XML Technology Implementation

Various XML technologies were employed to solve the problems of the project. xQuery was used to query the XML data, while XSL was used to transform the XML data into HTML and PDF formats. All data and presentational elements were properly separated and kept consistent and clear; all of our xQuery XML results contain no formatting data, while XSL handles all formatting.

xQuery

The primary xQuery is course_groups.xq. This xQuery gets the query parameters via the get-parameter function. This allows parameters to be gathered from a preceding page’s form or from the URL. This also allows for the page to be tested independently of the main query page. A start record parameter is passed to index to the next set of 25 results.

A single query string is built up from the parameters. If a parameter is left blank it is left out of the query thereby returning all matching results for the remaining parameters. The query is executed using the util:eval function. This allows for flexibility and efficiency in the query without many sets of cascading query commands. However, it does make debugging a bit more difficult.

Queries for terms in the course title, description or staff person are case insensitive and use the contain function to find general matches. Time data in the source XML (class start and end time) are specific down to seconds. Since all course times end in “:00”, this was concatenated so the user only has to input time in HH:MM format. If time is not entered in the correct format, the query returns no results.

Data from the xQuery was formatted in a hierarchy of department, course group and course. In the case where the course group and the department had the same name the course group is omitted and the courses become children of the department. This architecture was established to allow for the easy structure of the xsl. The structure of the elements and names were kept as close as possible to the source data for continuity.

Some XQuery functions were included to keep things DRY (Don’t Repeat Yourself). Additional functions could have been implemented but we were unable to do so because of time constraints. The conversion of the input parameters into

the XQuery syntax, for example, could have been done via a couple of functions to ensure formatting.

XSL

The xsl (course_groups.xsl) processes the result of the xQuery discussed above. HTML pages validate with no errors using https://validator.w3.org as HTML5. The course_groups.css file contains additional stylings for this page. Mobile compatibility is achieved via Bootstrap and responsive tables. XSL:import was used to gain the common Bootstrap layout for the page.

XML File Structure:

```
FINAL PROJECT
│
├──controller.xql
│
├──data
│   │   collection.xconf (configures and indexes the collection of data files)
│   │
│   └──courses
│       └──ALL COURSES XML FILES
├──xsl
│       common.xsl
│       course_details-fo.xsl
│       course_details.xsl
│       course_groups-fo.xsl
│       course_groups.xsl
│       main_page-fo.xsl
│       main_page.xsl
│
└──xquery
    │       course_details-pdf.xq
    │       course_details.xq
    │       course_groups-pdf.xq
    │       course_groups.xq
    │       main_page-pdf.xq
    │       main_page.xq
    │       autocomplete_list.xqm (for jQuery Autocomplete)
    |       courses_by_dept.xqm
    │
    └──xquery
    │         course_groups.css  jQuery.css
    │         jQuery.css
```

```
│
│   README.md
│   build.xml
│    expath-pkg.xml
│    repo.xml
```

VI. Extraordinary Distinction

A. **jQuery Autocomplete widget**: the site includes the jQuery Autocomplete widget, a feature that suggests search terms as a user types into a search field (see autocomplete_list.xqm). This significantly improves a user's experience by leveraging searching and filtering. The suggested search terms are taken from standalone xquery module functions, which are based on a values from the data files. If new data files are added, the widget automatically incorporates this new information. To aid users of our site, we added the Autocomplete widget to two fields on the Advanced Search page: the Department field and the Staff name field. This enables users to quickly find a department and/or professor's name without having to type the entire name. This is especially handy when the user cannot remember the full name of a department or staff member. A name/label combination was used. As a user types a department name, for example, the corresponding department code is used to query the course data. For more information on the jQuery Autocomplete widget see: jqueryui.com/autocomplete.

B. **Optimization of collection.xconf file:** early in the development of the project, it was noted that the certain pages, particularly the "Search by staff" page (localhost:8080/exist/apps/cscie18-xteam-finalproject/staff), were taking a long time to load (~30 seconds). It was determined that this slow response time was due to the fact that the collection.xconf file was indexing all attributes used in department and course-related code attributes, but not staff persons' id attributes. The xconf file was therefore manipulated to enable it to also index by the staff/person id attribute. To accomplish this, the following code was added:

```
<create qname="@id" type="xs:string"/>
<ngram qname="@id"/>
<ngram qname="/courses/course/staff/person/@id"/>
```

After the optimization of the collection.xconf file, the loading time of the homepage was drastically reduced from about 30 seconds to ~10 seconds.

For more information on the role of the collection.xconf file in eXistdb databases, see: http://exist-db.org/exist/apps/doc/indexing.xml.

C. Implementation of various frameworks (Bootstrap & jQuery Autocomplete)

While we employed Bootstrap to design the site's layout, we also employed jQuery's Autocomplete widget to enhance the site's functionality and the user experience. We believe that including both of these javascript libraries in our project has greatly improved the quality of the application.

VII. Lessons Learned

A. Initial Planning and Communication

Before writing a single line of code, all team members met several times over Skype to develop a working plan. Topics discussed during these initial meetings included: the role(s) each team member would play, site structure and design, development of mobile site, and possible extraordinary distinction features. In addition, we discussed all the necessary XML files needed for the project and defined their overall structure. Lastly, each file was assigned to an individual team member. These early meetings were also an opportunity to clarify the requirements of the project. For example, there was some initial misunderstanding over whether or not XForms were needed for the website. In the end, our initial discussions proved invaluable in helping the team establish a project workflow, goals and a sustainable timeline.

B. Importance of Working Iteratively

We made sure to get a simple, functioning version of the website up and running before tackling more sophisticated solutions. For example, we waited to incorporate the jQuery Autocomplete widget until after having first developed a working version of the application.

C. Separation of concerns

Throughout the project, we applied the computer science principle known as Separation of concerns, which advocates that each section of a computer program should address a specific concern or problem. In accordance with this principle, we created a separate xquery module (autocomplete_list.xqm) which focuses on providing a data set for the autocomplete fields. This module was then imported into the main_page.xq file. This separation will allow us (or any other developer) to change the implementation in the future to use different sources (if desired), or to use a different sort, without needing to modify the main_page.xq and/or main_page.xsl in any way.

D. Time Constraints

Due to time limitations, we were unable to implement an even more streamlined and efficient XML structure. For instance, with additional time, it would have been possible for the course_groups-fo.xsl and the course_details-fo.xsl to have fed off the same xQuery as the HTML pages. Minor changes to the XSL and xQuery

would have allowed for this. However, since different team members were working on the output (HTML & PDF), changing the base XML often would have taken too much time.

VIII. Assignment Q&A

Are your structures flexible and data-driven (instead of hard-coded)?
Yes, collections like meeting and staff are iterated by the XSL templates.

How easy would this be to turn into a University Catalog if you received data about courses from other schools within Harvard (Law, Business, Extension, Education, Design, Medical, Public Health, Government, etc.)?
Selecting the school would filter by the school_id attribute of the course element. In the case of this assignment we have only school_id="fas". The user would get a control to select the school on the main page. An additional XML resource would be helpful as a lookup table to translate between the school code (e.g. fas) and the full familiar name. An example of this would be https://coursecatalog.harvard.edu/icb/icb.do. The variable that holds the school id would need to be passed to each page or maintained as a global variable for all pages to utilize. Subsequent pages can then consume the id to process XQueries or XSLT and translate it via the lookup resource for display if needed.

Are choices between XQuery and XSLT consistent and clear?
Yes, since the course element in the source data is well formatted for the XSL to process easily it is simply passed along to the XSL.

What if new departments were created - would this require a code change?
No, this would not require a change in our code. Our code allows any new department data to be easily displayed.

How easy would it be to accommodate additional data fields about courses (e.g. if requirements fulfillment were provided as part of the course feed)?
The XSL would need to include table cells to accommodate each additional data field.

How easy would it be to add a new output format (for example, ODF, Excel, eBook)?
This would simply require a new XSL, which would perform a transformation into one of these formats (ODF, Excel, or eBook) in the same way xsl.fo transforms XML into PDF format.