CMSC 433 – Scripting Languages

Project 1: Worksheet for Computer Science Majors

Members:

Jerson Guansing (jguansi1@umbc.edu)

Joseph O'Malley (om4@umbc.edu)

Justin Sternberg (justins3@umbc.edu)

Project Description:

- The main purpose of this project is to apply what we have learned in class by creating a website that does the following:
    - Create a worksheet for Computer Science majors that allow students to enter courses they have taken.
    - Simulate some of PeopleSoft's functionality in terms of displaying the student's academic progress.
    - Dynamically create course recommendations based on previously taken course.
- The project uses a "Revolving Door" layout with only one file (index.php) acting as the user interface, but uses several helper files for specific functions (5 PHP files and 2 JavaScript files).
- The GL server is hosting the website and the MySQL database (studentdb-maria).
- The project used the UMBC's online course catalog as the source to populate the database with course information. Also, The UMBC directory is accessed whenever a new student uses the website.
- The coding requirement for this project calls for HTML/CSS for display output, and PHP/MySQL for server-side codes. However, JavaScript is extensively used for client-side validation and visuals.

Project Web Address:

- The project's URL is http://userpages.umbc.edu/~jguansi1/project1/
- It runs on the GL server.

Video Documentation:

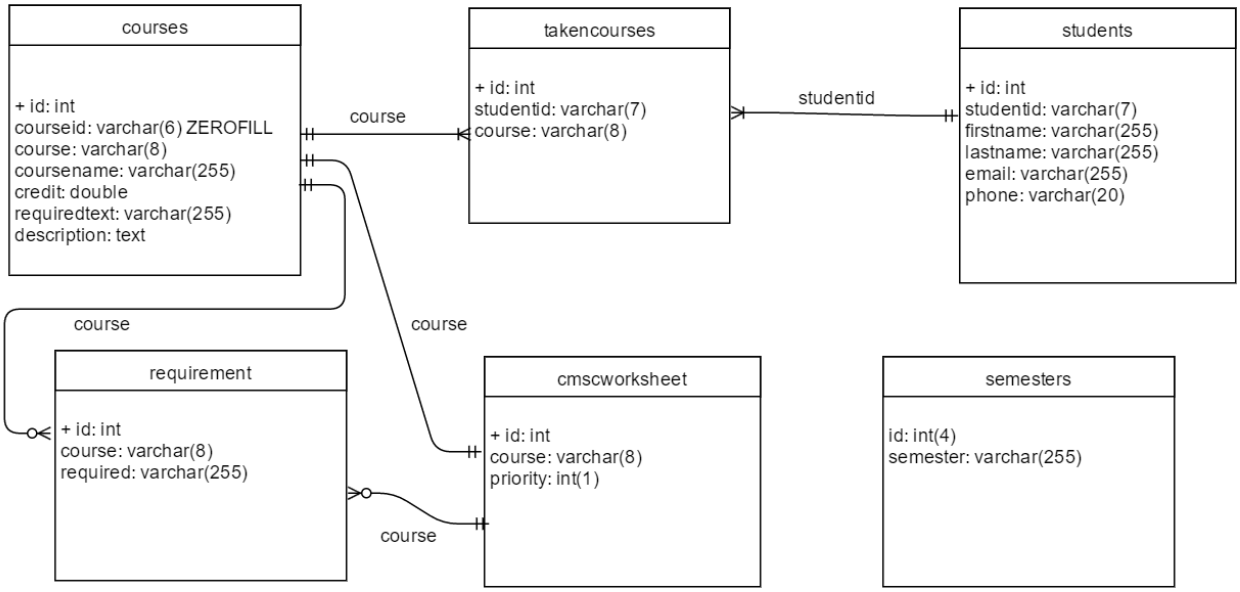
- The video can be found here https://youtu.be/umi5gfZy-dQ.
- The video documentation explains the project's usability and functions. It goes over the login process, validation, adding or deleting a course, course recommendation, and the student's academic progress.

Gathering Data:

- To build up the database for the project, all the information about courses are needed. The program, CoursesUMBC.java, was used to scrape data about all the courses offered at UMBC.
    - CoursesUMBC.java in the data folder is a simple program that scrapes course information (course ID, name, description, credit, etc.) from UMBC's online course catalog. Its only purpose is to generate a tab delimited file that will be imported into the courses table of the database.
- The CMSC Worksheet and the requirement for each course in that list has been converted into database table cmscworksheet and requirement.

Database Setup:

- The database used is GL's studentdb-maria MySQL server. This is the database's entity-relationship diagram.



Database Setup → Import Settings:

- Refer to this guide when importing data files used for this project into to the database. All the data files are located in the data folder.
    - Format: CSV
    - Number of rows to skip: 1 (skip the column headers)
    - Columns separated with: Tab
    - Columns enclosed and escaped with: Back tick (`)

Database Setup → courses:

- This table contains all the courses offered at UMBC. Create this table using the SQL command below:

```
CREATE TABLE courses
(
id int NOT NULL AUTO_INCREMENT UNIQUE,
courseid int(6) ZEROFILL NOT NULL,
course varchar(8) NOT NULL UNIQUE,
coursename varchar(255) NOT NULL,
credit double NOT NULL,
requiredtext varchar(255) NOT NULL,
description text NOT NULL,
PRIMARY KEY (id)
)
```

- The file, courses.csv, located in the data folder can then imported into the courses table. It contains over 2600 records. Refer to the import settings.
- For column names option of the import setting, just omit the id column.

Database Setup → takencourses:

- This table holds all the taken courses associated to a student ID (foreign key). Create this table using this SQL command:

```
CREATE TABLE takencourses
(
id int NOT NULL AUTO_INCREMENT UNIQUE,
studentid varchar(7) NOT NULL,
course varchar(8) NOT NULL,
PRIMARY KEY (id)
)
```

Database Setup → students:

- This table holds all the student information of users who have used the website. When a new student accesses the project, their information is either pulled from the UMBC directory or manually entered, and then inserted into this table. Create this table using this SQL command:

```
CREATE TABLE students
(
id int NOT NULL AUTO_INCREMENT UNIQUE,
studentid varchar(7) NOT NULL UNIQUE,
firstname varchar(255) NOT NULL,
lastname varchar(255) NOT NULL,
email varchar(255) NOT NULL,
phone varchar(20) NOT NULL,
PRIMARY KEY (id)
)
```

Database Setup → requirement:

- This table holds the requirements (prerequisites) for a specific course. This was designed to make course validation much easier by making the prerequisite component database-driven instead of static checks, and in turn making it more scalable (at the moment only the prerequisites for courses in the CMSC worksheet is entered and not all courses offered at UMBC, but gradually adding prerequisite is just a simple INSERT command). Create this table using this SQL command:

```
CREATE TABLE requirement
(
id int NOT NULL AUTO_INCREMENT UNIQUE,
course varchar(8) NOT NULL,
required varchar(255) NOT NULL,
PRIMARY KEY (id)
)
```

- The file, requirement.csv, located in the data folder can then imported into the requirement table. Refer to the import settings.
- For column names option of the import setting, just omit the id column.

Database Setup → cmscworksheet:

- This table holds all the courses listed in the CMSC worksheet. The courses are categorized by priority (1 = required CMSC courses, 2 = required Math courses, etc.), and is necessary for providing course recommendations to the user. Create this table using this SQL command:

```
CREATE TABLE cmscworksheet
(
id int NOT NULL AUTO_INCREMENT UNIQUE,
course varchar(8) NOT NULL UNIQUE,
priority int(1) NOT NULL,
PRIMARY KEY (id)
)
```

- The file, cmscworksheet.csv, located in the data folder can then imported into the cmscworksheet table. Refer to the import settings.
- For column names option of the import setting, just omit the id column.

Database Setup → semesters:

- This table holds the current and previous semesters at UMBC. This is used along with courseID from the courses table to pull section information of a course for a given semester. Create this table using this SQL command:

```
CREATE TABLE cmscworksheet
(
id int NOT NULL UNIQUE,
semester varchar(255) NOT NULL,
PRIMARY KEY (id)
)
```

- The file, semesters.csv, located in the data folder can then imported into the semesters table. Refer to the import settings.
- For column names option of the import setting, just omit the id column.

Project Files:

- All the project files are located in the www folder. The hyperlinks within the files are relative to allow deployment into other folders/subfolders.
- PHP: courses.php, dbConnection.php, index.php, projectFunctions.php, sections.php, and students.php.
- JavaScript: getCourses.js, and login.js
- CSS: styles.css
- Other files: icon.png, loadingBar.gif, retrievers.jpg, and search1.png

courses.php:

- This helper file outputs a JSON content.
- The user needs to be logged in, and requires access to the database. If not logged in, it will output an empty array.
- This is a helper file that expects an HttpRequest parameter course which is then used to query the courses table matching the given parameter. The result is limited to 12 records for efficiency purposes.
- This helper file is called through an XMLHttpRequest to dynamically generate a dropdown list of courses based on the search input.

dbConnection.php:

- The main purpose of this helper file is to create a connection to the database needed for this project. It connects to GL's studentdb-maria MySQL server.
- All the connection details are delegated in this file which is then called by other PHP files. This makes database access more streamlined, and future database access changes easier to handle.

getCourses.js > getCourses():

- This JavaScript function is called when the "input" eventListener for the search bar is triggered.
- This function serves two purpose:
    - Validating the text input of the search bar.
        - As the user types in a course(s), the function validates the input and remove characters that does not meet the criteria.
            - Valid inputs: letters, digit and comma
            - Courses need to match this format:
                - [A-Z]{3,4}[0-9]{2,3}[A-Z]?
                - Example: CMSC201, CMSC201H, MATH99, ART326, etc.
            - Commas are only accepted after a valid course entry.
    - Dynamically creating a dropdown list of courses matching the search keyword.
        - This functionality requires a valid single course input. Courses separated with comma will not work.
        - Whenever the function is called, it clears the innerHTML of searchResults.
        - The function will start an XMLHttpRequest to a helper file (courses.php) and passing it the course parameter.
        - The responseText (JSON output of courses.php) is then parsed, and converted into an array of courses matching the keyword.
        - It will then iterate through each array element, and adding the course details into the innerHTML of the div searchResults.

getCourses.js > display():

- This JavaScript function is called when the “click” eventListener for each course div in the recommended section is clicked.
- This function serves as a toggle to either display or hide the course information.

getCourses.js > getSections():

- This JavaScript function is called when the “click” eventListener for each “Sections” button of a course in the recommended list is clicked.
- The function retrieves the courseID (zero filled value; not CMSCXXX) and the selected semester.
- The function will start an XMLHttpRequest to a helper file (sections.php), and passing it the courseID and semester parameters.
- The responseText (JSON output of sections.php) is then parsed, and converted into an array of sections.
- It will then iterate through each array element, and adding the section details into the innerHTML of the div scheduleDiv for the corresponding course.

getCourses.js > displayLoading():

- This JavaScript function is called when the user clicks any button with the attribute class=”AddIt” (this covers both “AddIt” and “Delete” buttons) and adds a course using the search bar / dropdown list.
- The main purpose of this function is to provide an immediate feedback to the user when the events listed above are trigged, and to prevent the user from smashing (repeatedly clicking) the buttons.
- This function serves as a toggle to either display or hide the loading div with the highest z-index (on top of all other layers). This essentially prevents the user from triggering any other event when a current request/process is being executed.

index.php → Login:

- The student needs to enter their UMBC ID. There is a “keyup” eventListener added to the UMBC ID textbox, and once the input matches the required format AAXXXXX (2 characters and 5 digits), then it will do a student search using the getStudentInfo() function.

- **index.php → Login → NEW student logging in (UMBC ID)**
    - If the student is not in the database and enters their UMBC ID, the JavaScript function, getStudentInfo(), will start an XMLHttpRequest to a helper file (students.php).

- The webpage (students.php) will first check if the student is in the database. If the ID is not found, it will search the UMBC Directory with the given student ID, and retrieve the student information by scraping the Directory webpage (since we do not have access to the UMBC's student database). The scraped data (JSON format) is returned and parsed into an array.
- The JavaScript function, getStudentInfo(), will then fill in the first name, last name and email, but the user needs to enter the phone number.

- **index.php → Login → EXISTING students (already in the database)**
  - Entering their UMBC ID will call the JavaScript function, getStudentInfo(), that will start an XMLHttpRequest to a helper file (students.php).
  - The webpage (students.php) will check if the student ID is in the database, and return the SQL result containing the student information in JSON format.
  - The JavaScript function, getStudentInfo(), will parse the data into an array, and then fill in the first name, last name, email and phone number.
  - The user will then have the option to update their student information by editing the form textboxes before submitting. This will update their student information in the database to reflect the changes made.

- **index.php → Login → FORM VALIDATION**
  - The login page has a three-tiered process.
    - First, the pattern attribute of the textboxes will validate if the value matches the regular expression before submission.
    - Second, JavaScript is employed to validate the textboxes as the user types in the data. As the user enters the data, it will remove characters that does not match the condition given. There is "keyup" eventListener added to each textbox that will call a corresponding JavaScript function for validation.
    - Upon submission, the data is verified again by the PHP function, studentInfo(), if it matches the regular expression for each data field. If any of the required data fails the validation, it will display the login page again so the user can make the necessary corrections.

index.php → Main:

- This is the main hub of the website. Since the project uses a "Revolving Door" layout, index.php handles the majority of the tasks such as database queries and user interface. It displays the student information, add or delete courses, and display the academic progress of the student as well as recommended courses to take.

- **index.php → Main → DEFAULT – either logged in or not**
  - Whenever index.php is accessed, it will call the PHP functions studentInfo() and sessionInfo(). The first function, studentInfo(), will retrieve the data from HttpRequest (GET or POST) which is used for login and other functions of the page. For each HttpRequest parameter that is retrieved, it will do a regular

expression validation to see if it matches the format. The other function, sessionInfo(), will retrieve student information stored in the session.

- When the sessionInfo() function sets validProcess to true, then it means a user is logged in.
  - This means that the session variables studentID, firstname, lastname, email and phone are set.
  - If the session variable validation is true, then it will set validProcess = true. This will allow other processes (adding, deleting, recommended courses, etc.) to execute.
  - Lastly, it will display the main layout of the page with the student information at the top, recommended courses on the left side, and the academic progress of the student on the right side.
- When the sessionInfo() function sets validProcess to false, then it means that the user is not logged in.
  - validProcess is set to false every time the page is loaded, but is changed to true when pr = 2 is passed (this is the login function) OR when the session variables mentioned above are set.
  - By default, it will display the login page if validProcess is set to false.

- **index.php → Main → LOGIN (pr = 2)**
  - This is called when the HttpRequest parameter pr (for process) is passed with the value of 2.
  - It requires access to the database table students.
  - The login process sequence will execute given that the other HttpRequest parameters (studentID, firstname, etc.) are all valid (not equals to an empty string after being retrieved by the studentInfo() PHP function). First, it will check if the studentID is already in the students table of the database. Then, there are two possible cases:
    - It will either insert a new student record in the database if the studentID does not exist. The student information is saved in the students table. Afterwards, it will set the session variables to hold the student's information.
    - Update the student information with the passed values if the studentID exists in the database. The new student information is saved in the students table of the database. Afterwards, it will set the session variables to hold the student's information.

- **index.php → Main → ADDING Courses (pr = 1)**
  - This function will only execute if validProcess = true. It also requires access to the following database tables: courses, requirement and takencourses.
  - All the courses offered by UMBC are stored in the courses table. It contains information such as course name, description and credit.
  - The requirement table contains the prerequisite information for certain courses.
  - The takencourses table holds the list of all courses taken for each student.

- There are three possible ways of adding courses to the database.
  - Adding a course by typing the course(s) in the search bar, and submitting.
    - There is a “keyup” eventListener added to the search bar that calls the getCourses() JavaScript function.
    - The function validates the input as the user types in the course(s), and removes characters that does not match the format.
    - The pattern attribute of the search bar also is used to ensure that the input value matches the regular expression.
    - If the input is valid, then the course HttpRequest parameter is split into an array (using the explode() function). This handles single or multiple course entries using a comma as the delimiter.
  - Adding a course by clicking it from the dropdown list of the search result.
    - Like I mentioned before, the search bar has a “keyup” eventListener that calls a JavaScript function.
    - Given that the value of search bar is just one course (course with no commas OR just course prefixes like CMSC or MATH4), then it will start an XMLHttpRequest to a helper file (courses.php).
    - The webpage (courses.php) will search the database for courses matching the keyword provided. An example will be to return the list of 400 level CMSC courses with course parameter = “CMSC4”. However, the search result is limited to a maximum of 12 records for efficiency purposes. The data (JSON format) is then returned.
    - The JavaScript function, getCourses(), then parses the returned data, and converts it into an array.
    - Lastly, it generates a dropdown list of courses by changing the innerHTML of the div right under the search bar.
    - Each of the course is a link passing pr = 1 and the course to add.
    - Similar to the method above, the course is then added to an array of courses to be added.
  - Adding a course from the list of recommended courses.
    - Each courses in the list of recommended courses has a link setting pr = 1 and course = the specific course to add.
    - It goes through the same process where the course is appended into an array of courses that need to be added.
- With the array of courses to be added generated, the webpage then does a loop to iterate through each course in the array.
- It checks if the course to be added is even in the courses table (list of all courses offered at UMBC). If not in the database, then it will mark that course as not in the database, and continue to the next one.
- If the course is in the database, it checks if the course is already in the list of taken courses. If it is in takencourses, then it marks it as so, and continue to the next one.

- If the course passes those two validation, then it checks for the requirements (prerequisites) for the course by calling the compareRequirements() function. If the function returns false, then it marks it as so and displays the text requirement for the given course, and then continue to the next one.
- If compareRequirements() function returns true, then the course is associated with the studentID, and gets added to the takencourses table.
- The process is repeated until it iterates through all the elements in the array of courses to be added.

- **index.php → Main → Deleting Courses (pr = 3)**
  - This function will only execute if validProcess = true. Also, it requires access to tables requirement and takencourses in the database.
  - The table requirement contains the prerequisite information for certain courses.
  - The takencourses will hold the list of all courses taken for each student.
  - This function requires that pr = 3 and course match the valid format through a regular expression check.
  - If the course is valid, then it gets added to the queue of courses to be deleted.
  - It then uses a while loop to iterate through each course until the queue is empty.
  - For each iteration, it will issue a DELETE SQL command to remove the course from the takencourses table. Also, it unsets (remove) it from the array list of takencourses that was generated when the page was loaded.
  - Next, it will check each course left in the list using the compareRequirements() function if they still meet the prerequisites / requirements.
  - If it failed the validation, the course gets added to the queue of courses to be deleted, and the loop continues.

- **index.php → Main → Logout (pr = 4)**
  - The main premise of this function is to logout the current user or invalidate the current session. There are two ways this is accomplished:
    - Clicking the Logout link at the top navigation bar.
      - This requires that HttpRequest parameter pr = 4.
      - This will unset the current session, and then destroy it.
      - Right after, it will start a new session and display the login page.
    - The last session activity is more than 30 minute ago.
      - The webpage (index.php) keeps track of the user's last activity by using the session variable LAST_ACTIVITY.
      - Each time there's an event / activity, the webpage checks if the last event was more than 30 minutes ago.
      - If so, it follows the same procedure where it unsets and destroys the current session.
      - This forces the user to re-login or allow another student to sign in.
  - As mentioned above, once the session is destroyed, a new session is started. This means that validProcess will be set to false, and the login page will be displayed.

- **index.php → Main → Recommended Courses**
    - This function will only execute if validProcess = true. Also, it requires access to tables cmscworksheet, requirement, and takencourses in the database.
    - The table cmscworksheet holds the list of courses divided into categories / priorities that a Computer Science major needs (required, elective, etc.). The course and their corresponding priority is queried and saved in an array called cmscworksheet.
    - Along with displaying the student's academic progress, this process is always called after every other processes finish (delete or add courses) when validProcess = true.
    - A for loop is used to iterate through each course in the cmscworksheet array to check if its already taken by comparing it with the takencourses list.
    - If it's not in the takencourses array, then the compareRequirements function is called to verify if the prerequisites have been met for the particular course.
    - If the function returns true, then does a few more validation:
        - CMSC345 or CMSC447. If the student has taken either course, then it will not recommend the other.
        - For courses that fall under CMSC electives, it will not recommend the courses in the list if the student has fulfilled both CMSC electives (2 of 2) and technical electives (3 of 3).
        - Similarly, courses that fall under Technical electives will not be recommended if the technical electives requirement has been fulfilled (3 of 3).
        - For Science courses, it will not recommend anymore courses if the 12 credit requirement has been fulfilled. Also If the Science sequence is not yet completed, then it will save a slot for a course that will complete the sequence, and skip recommending the other Science courses.
    - All the recommendations go into the array called recommended.
    - The data is displayed on the sidebar section of the webpage. A for loop is used to iterate through each course.
    - The functionality to allow students view sections of a course for a particular semester is added. This requires access to the table semesters in the database.
    - Iterating through each recommended course, the following procedures are done:
        - A variable called lastCategory is used to keep track of changes in course priority. This will display a div separator to break the recommended courses into categories (Required CMSC courses, Science courses, Math courses, electives, etc.).
        - Each course is encapsulated in a div with an id matching the course. This has a "click" eventListener that will either show or hide details about the course. This is done to condense the list, but still allow the users to expand it to view more details if need be.

        - The details about the course is displayed (course name, credits, description and requirements).
        - There are two buttons for each recommended course. Add it and Sections:
            - The first button is called Add it.
                - Add it is a simple hyperlinked button with course = the current course, and pr = 1. Refer to the Adding Courses section for more details.
            - The second button is called Sections.
                - This has a "click" eventListener added to it that will call the JavaScript function getSections(). The function requires the UMBC ID of the course (this is not CMSCXXX, but a 6 digit value that is zero filled), and the semester selected.
                - The function starts an XMLHttpRequest to a helper file (sections.php with parameters courseID and semester).
                - The webpage (sections.php) will validate the HttpRequest parameters, and then call the PHP function getCourseSchedule() by passing it the courseID and semester.
                - This function will search the UMBC course catalog, and then scrape page for all the section for the given course.
                - This includes section number, date/time, professor teaching it, and section status.
                - The data is then returned (JSON format), and parsed into an array.
                - The JavaScript function getSections() will then iterate through each section, and append it to the innerHTML of the selected course's section div.
    - The process is repeated until all recommended courses is listed by category / priority.

- **index.php → Main → Academic Progress**
    - This is the visualization of the student's academic progress. It breaks down the courses taken into different categories.
    - This function will only execute if validProcess = true. Also, it requires access to tables cmscworksheet and takencourses in the database.
    - A for loop is used to iterate through each category, and display the following information.
        - The category (Required CMSC course, Technical elective, etc.).
        - A progress bar displaying what percentage is completed for the given category.
        - The PHP function displayCourses() is called to display a table of courses taken that falls into the given category.

login.js > getStudentInfo():

- This JavaScript function is called when the “input” eventListener for the studentID textbox is triggered.
- This function serves two purpose:
    - Validating the text input of the studentID.
        - As the user types in a student ID, the function validates the input and remove characters that does not meet the criteria.
            - Valid inputs: letters and digit.
            - Student ID need to match this format:
                - [A-Z]{2}[0-9]{5}
    - Retrieve student information from the database or the UMBC directory.
        - This functionality requires a valid student ID.
        - The function will start an XMLHttpRequest to a helper file (students.php) and passing it the studentID parameter.
        - The responseText (JSON output of courses.php) is then parsed, and converted into an array matching the studentID.
        - It will then iterate through each array element, and setting its value to the corresponding textboxes (firstname, email, etc.).

login.js > formatName(), formatEmail(), formatPhone():

- These JavaScript functions are called when the “input” eventListener for their corresponding textboxes are triggered.
- As the user enters a value, the function validates the input and remove characters that does not meet the criteria.

projectFunctions.php > studentInfo():

- This PHP function is called whenever index.php is requested.
- The main purpose of this function is to retrieve the HttpRequest parameters such as studentID, firstname, lastname, email, course, etc.
- The function also validates each value if it matches the require format using a regular expression.
- All the parameter values are stored in an array, and then returned.

projectFunctions.php > compareRequirements():

- This PHP function is used for the following processes: adding a course, deleting a course, and generating the recommended courses list.
- The function expects three parameters: the course that needs to be compared, takencourses, and the student ID. This function requires access to the database.

- The function will check if the course has prerequisites listed in the requirement table of the database. If it does not, it will set allowed to true.
- If it does have prerequisites (required column in the requirement table), then it will do the following processes:
    - It will iterate through each record that has a matching course. Each row/record of prerequisites is considered an AND statement.
    - The required value is then split/exploded using a comma as a delimiter to cover OR statements. This is stored into an array.
    - The function will iterate through this OR statement array, and checking if the courses are in the takencourses list. If any of the courses are in takencourses, then the whole OR block is considered true, and it will continue to the next row/record of the database query.
        - If the course in the OR statement array is not in the takencourses list, then there three possible scenarios:
            - There is an AND statement within the OR statement block (Example: CMSC201,MATH152_AND_MATH221,CMSC203).
                - The function will split it using "_AND_" as delimiter.
                - It will check each course from this split against the takencourses list.
                - If the courses are not all in the takencourses list, then it will set allowed to false and break out of the _AND_ loop.
            - There is a wildcard (Example: CMSC4%).
                - The function will query the takencourses table if there are entries matching the course for the given student.
                - If there is a match, then the whole OR block is considered true and it will continue to the next row/record of the database query.
                - If there are no matches, then it will set allowed to false.
            - It is not in the takencourses list.
                - It will set allowed to false.
            -
    - The function will keep looping until all the row/record has been compared to the takencourses list.
- The Boolean value of allowed is returned.

projectFunctions.php > sessionInfo():

- This PHP function is how the project knows if the user is logged in or not.
- The project keeps track of when the last activity happened for the logged in user, and invalidates session that has been inactive for more than 30 minutes.
- The function also, retrieves session variables that hold the student information of the user currently logged in.
- The value is stored in an array, and then returned.

projectFunctions.php > displayCourses():

- This PHP function is called by index.php when it displays the academic progress of the student currently logged in.
- The function expects a parameter: an array list of courses for a given category/priority.
- It displays a table listing the course, course name, and a delete button.

projectFunctions.php > curl():

- This PHP function gets the source code of the URL that is passed to the function.
- It will work with both http and https requests.
- The data containing the source code is then returned.

projectFunctions.php > scrapeStudentInfo():

- This PHP function is called by students.php when the studentID is not in the database, and it needs to search the UMBC directory.
- This function expects one parameter: studentID.
- It calls the PHP function curl, and passing it the UMBC Directory URL with the studentID as the search parameter.
- The function then scrapes the returned data for the following student information: firstname, lastname, and email address.
- The scraped data are stored in an array, and then returned.

projectFunctions.php > exists():

- This PHP function expects two parameters: course and the array list to compare it to.
- The function is for convenience only. It is the same as the built-in PHP function array_key_exists(). The only difference is that it checks whether the course and its honors version is in the array list.

projectFunctions.php > getCourseSchedule():

- This PHP function is called by sections.php to search the UMBC catalog for section details for a specific course.
- The function expects two parameters: courseID and semester.
- It calls the PHP function curl, and passing it the UMBC course catalog URL with the parameters courseID and semester. The returned data from the curl() function call will then be processed.
- There is a while loop to iterate through all the sections of the given course.
- For each iteration, it will scrape the section, date/time, professor teaching it, and the course status.
- The scraped data are stored in a 2D array, and then returned.

sections.php:

- This helper file outputs a JSON content.
- The user needs to be logged in, and requires access to the database. If not logged in, it will output an empty array.
- This is a helper file that expects courseID and semester as HttpRequest parameters which is then passed to the getCourseSchedule() function call.
- It then iterates through each section element from the returned array.
- This helper file is called through an XMLHttpRequest to dynamically generate all the sections of a course for a particular semester.

students.php:

- This helper file outputs a JSON content.
- It requires access to the students table in the database. Also, it expects studentID as an HttpRequest parameter.
- It will search the students table in the database with the given studentID. If a record is found, then it outputs the student's information.
- If the student is not in the database, then it calls the function scrapeStudentInfo() passing it the parameter studentID.
- The returned array from the function call is then displayed.
- This helper file is called through an XMLHttpRequest to fill in the values of textboxes of the login page.

Sources:

- CMSC Worksheet
  http://catalog.umbc.edu/preview_program.php?catoid=3&poid=318&returnto=104
- Project Description
  http://userpages.umbc.edu/~slupoli/notes/ScriptingLanguages/projects/CMSC%20Advising%20Project%20-%20Part%201.txt
- UMBC Course Catalog
  https://highpoint-prd.ps.umbc.edu/app/catalog/listCatalogCareers
- UMBC Directory
- http://www.umbc.edu/search/directory/