Spring 2015

Electronic Health Record Simulator

Samuel M. Brown
University of Akron Main Campus, smb193@zips.uakron.edu

University of Akron

Electronic Health Record Simulator

For use in an academic setting

Sam Brown
4-16-2015

Samuel Brown

Introduction

This project involves the simulation of electronic health records (EHRs) for use in a classroom setting, specifically with application to aspiring dietetics students. While similar EHR simulators exist, few if any allow the use of "dummy" patients; that is, in order to use these EHRs in a classroom setting, it is required that real people (with real social security numbers) be used, which isn't an option due to HIPAA. Many health professions programs, therefore, do not teach with respect to EHRs, which is only of detriment to the students. By being exposed to EHRs prior to seeing the ones installed in most every hospital, students are better prepared to enter the workforce.

The primary purpose of this project, then, was to satisfy a fellow undergraduate's design specifications for such an application. It was required of me to implement a number of primary features and make the interface appear as desired. It was then up to me to determine the appropriate technologies required to satisfy the request of a "client". I was to have an application where an instructor could fill in the chart of a fictional patient which students could then see and give responses such as diagnoses and intervention plans. Instructors should then be able to see these responses. All of this should be organized by class.

Design/Planning

When it came to design, many of the functional requirements had already been established for me. First, instructors needed to be able to create fictional patients. This is to be similar to an instructor filling in an electronic chart. Upon patient creation, the data is stored in a bank which allows any instructor to view any patient. Instructors then needed to be able to post these patients to individual classes. When posted, students that are members of particular classes could then view the patient's chart, and fill out information such as height and weight conversions and nutritional recommendations. The instructor would then be able to view student responses for each patient.

From this description, I could refine my requirements: I would need a login system; different users would be allowed to access different parts of the application; I would need several database tables for patient information; and I would need some sort of

administration user. This administration user would be able to delete old classes and change the basic student account into an instructor account.

This problem seemed best able to be addressed by a web application. By using ASP.NET, which includes many software facilities for things such as form authentication, SQL database access, and website planning, I was able to accomplish more work. Since ASP.NET is a Microsoft technology, it also made sense to use Visual Studio for development and Azure for hosting, as it only took one click to publish the project every time I wanted to test and debug my code.

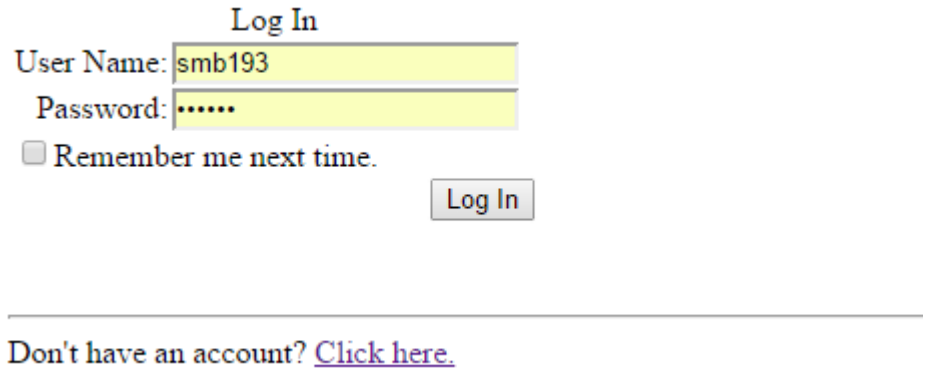
The technologies I used included ASP.NET, HTML, C#, SCSS, and SQL. ASP.NET, HTML, and C# all went hand-in-hand, allowing for the creation of active server pages which could easily connect to the SQL database storing all user, class, and patient information. SCSS (or Sass, as it is also known) allowed for more programmatic CSS styling, since it allowed for variables and nested declarations that could then be compiled into actual CSS. SQL (a language with which I am admittedly inexperienced) was used for the database queries, with most of the actual database design done using Microsoft Azure's online database management portal.

Implementation

There were a total of 17 pages in the website, organized into three directories. The home directory (which all users could access) were login, register, refresh, index, ViewClass, apply, and response. The Admin subdirectory contained the pages for managing Users and Classes. The Teach subdirectory contained the pages AddClass, AddPatient, Classes, EditPatient, Patients, ViewClass, ViewResponses, and response.

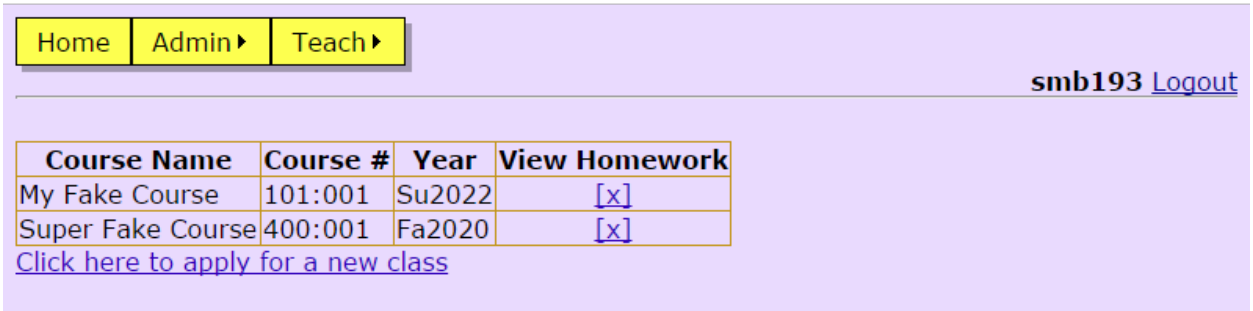
I first implemented the user authentication and authorization. User authentication (and authorization) is handled through a login form [1]. When the user logs in, it authenticates the session and also updates the database to update the last date on which the user was seen—this is useful for database management on a long term scale, allowing for the deletion of accounts that are not used within a certain timeframe. I also included a date of creation for the account, for use in a similar purpose. This table also helped when

dealing with authorization; I could assign each user a role number (compared against a second table for a role name) which would prevent student users from accessing instructor or administrator pages. On authentication, a session is started which stores the user's unique identifier, allowing other tabular data to be accessed in other pages.



1-A simple login page, with a link for account creation.

One issue I ran into is that the user's unique ID was lost if the user idled on one page too long. For example, the student could be spending time looking through references while entering patient information. To prevent this, I placed an iframe which contained an empty page, refresh.aspx, in the footer of my master page (which all pages inherit from) [2]. This page refreshes every several minutes, which keeps the session active as long as the browser is left open.

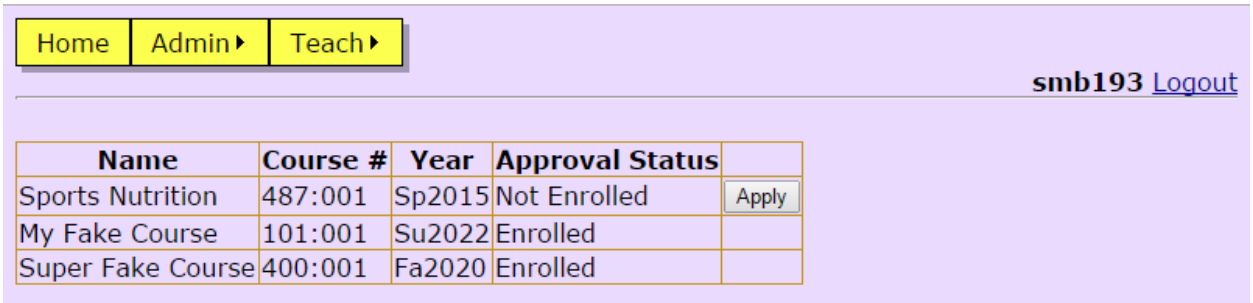


2-All pages contains an invisible page which refreshes automatically.

After implementing the login system, I had to handle user administration before I could do anything else. While administrators technically have privilege to create classes, and there's nothing wrong with that, it is not wise for every instructor to have administrator privilege.

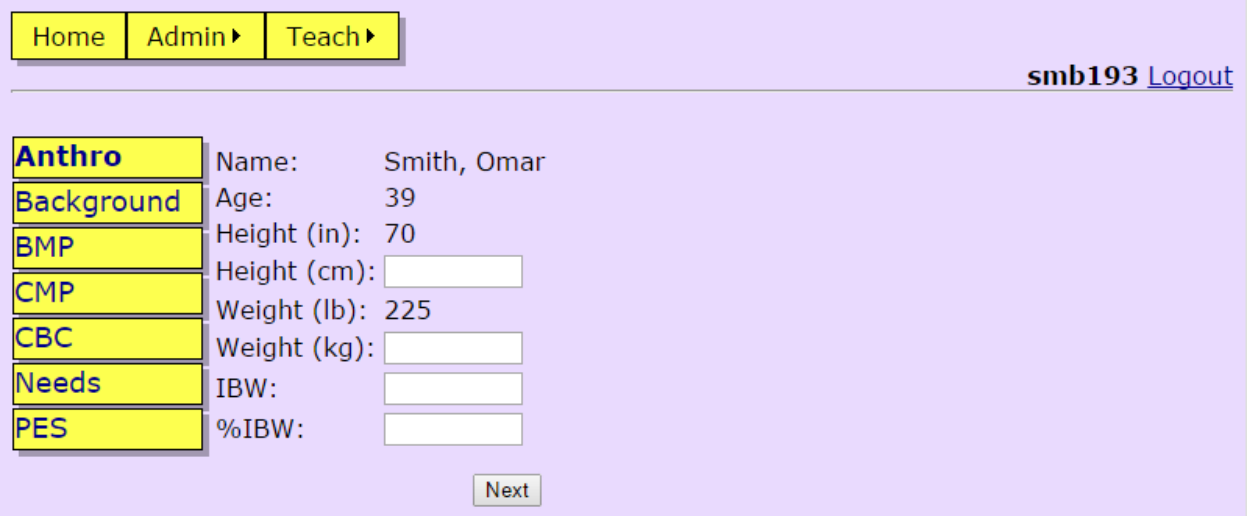
3-User administration page, class administration is similar. Users have three roles: Admin, Prof, and Student(default).

The next major step was figuring out how to "enroll" students in a course. Having instructors create courses was easy enough (it was roughly the same implementation as creating a patient, which is just a SQL `INSERT INTO` query), but it was not as simple for just anyone to be able to enroll in said courses. Since there are no limits on who can create an account, it made sense for me to put a restriction here. Students can apply to a class from the landing page, listed under their current courses. From there, they can see a list of all classes, as well as buttons to apply for any class they have not requested to be enrolled in [4]. Instructors can then view their class and approve any student pending approval. In the scope of the University of Akron, instructors would only approve someone whose username was identical to a UANet ID on the class list.



4-Students can apply for courses they have not yet applied to. "Pending" is displayed when Enrolled but not approved.

Students then needed to be able to enter in their responses. The .NET Wizard web control seemed natural for this: I was able to spread information to multiple 'tabs', much like an EHR would do, while collecting various information that could then be submitted to a separate database table that instructors could access later[5].



5- Step 1 of the response wizard. Submit button is located on the final page.

Since many pages share similar functionality, it is redundant to discuss every page of the website. For instance, several pages access the same table of my database, organized in a slightly different manner. As such, I have omitted the pages for adding patients to a class, approving students, administrating classes, creating a patient, and viewing individual student responses.

Closing/Reflection

This project taught me much about the process of creating real software. I already knew that working alone on large projects was a lot less glamorous than the media would lead you to believe, but firsthand experience with such a situation truly cemented that fact. It is difficult to only have oneself to rely on when a problem arises, and not having a second pair of eyes to help debug leads to many an hour of head scratching. Given another opportunity, I would have definitely tried to find another student to help create this website, so that it would be more fully featured.

I also was able to learn a couple of new technologies, namely SQL and SCSS/Sass. I already knew a bit about SQL and how databases work, but I had never done anything like designing my own schema, which was a lot more interesting than simple queries. I now have confidence in my ability to not only write SQL queries but also to write schemas.

SCSS was different in that I already knew how to write CSS, but did not enjoy writing it. SCSS changed that by making CSS more fun: I could now declare variables for things like

padding and color. This made it easy to fiddle with elements without having to scroll through a CSS document to find the specific element I wanted to change. The best part was, since SCSS compiled into CSS automatically every time I saved the file in Visual Studio, I didn't have to do anything differently in my implementation.

Lastly, this project taught me about the importance of communication between developer and client. Without the constant communication I had with my "client", I would not have been able to stay as on track with the intended design as I did. I was also able to get important input on styling—which saved me hours of attempting to make something that looks pleasing to me, but not to others.

The experience has left me excited to enter the workforce and face new challenges. I haven't had the privilege of being able to work internships like many of my classmates, so it is important to me that I have projects that I can be proud of to show prospective employers.

Appendix

Website: http://smb193-hrp.azurewebsites.net/

Github: https://github.com/smb193/EHR-Sim