BookOrder

CSIT 455 - Relational and Object Databases

Tyler Roesler, Matt Siragusa, Steven Treacy

|  |  |  |
| --- | --- | --- |
| Project Components | Member Name | % of Workload on component |
| Backend logic/processing, Frontend design | Tyler Roesler | 80%, 15% |
| Backend logic/processing,  Frontend design | Matt Siragusa | 10%, 45% |
| Backend logic/processing,  Frontend design, Query | Steven Treacy | 10%, 40% |
|  |  |  |

**I. Introduction**

When tasked with this project opportunity it became very clear that we could use our knowledge and resources to tackle an actual real world problem. Shortly after the project was presented to us, our group became aware of a problem that departments on campus have with their professors telling their students which books to order for each class. There currently is no good system in place for keeping track of which books are recommended by professors and therefore makes it difficult for the department heads and the book store. Before we wrote any code or designed a schema, we spoke with the secretary of the Computer Science department as well as a representative from the book store. We knew it was important to design a system that met their requirements, as well as the requirements for the project.

After we spoke to those individuals in both the computer science department and the bookstore, we were eager to get started. We set out to make a web application and database to streamline the book ordering process. If utilized properly, any department on campus could have a system in place to make that process.

To properly create this application, we decided that we needed a few key pieces of functionality in order to meet everyone’s needs. First and foremost, we would need a system to create and account and log into the system. A part of that system is giving the person logging in the ability to tell the application whether or not they are an administrator. That way, if you are an administrator, you have more functionality such as adding classes, assigning classes to professors, as well as editing and deleting classes from the system. Also, if the user is an admin, they can easily toggle to the ‘teacher’ view of the site to see the classes that they teach. The other type of account is a basic ‘teacher’ account. When logging in with this type of account, the teacher will see all of the classes assigned to them. From there they can add, edit and delete books and media to any class.

Lastly, there is a profile page. From this page, the teacher can view their account information, the number of classes they teach, as well as the number of classes they teach that don’t yet have books assigned to them. All of this functionality is developed using the Ruby on Rails web development platform. We will expand more on implementation details in the next section.

**II. Methodology**

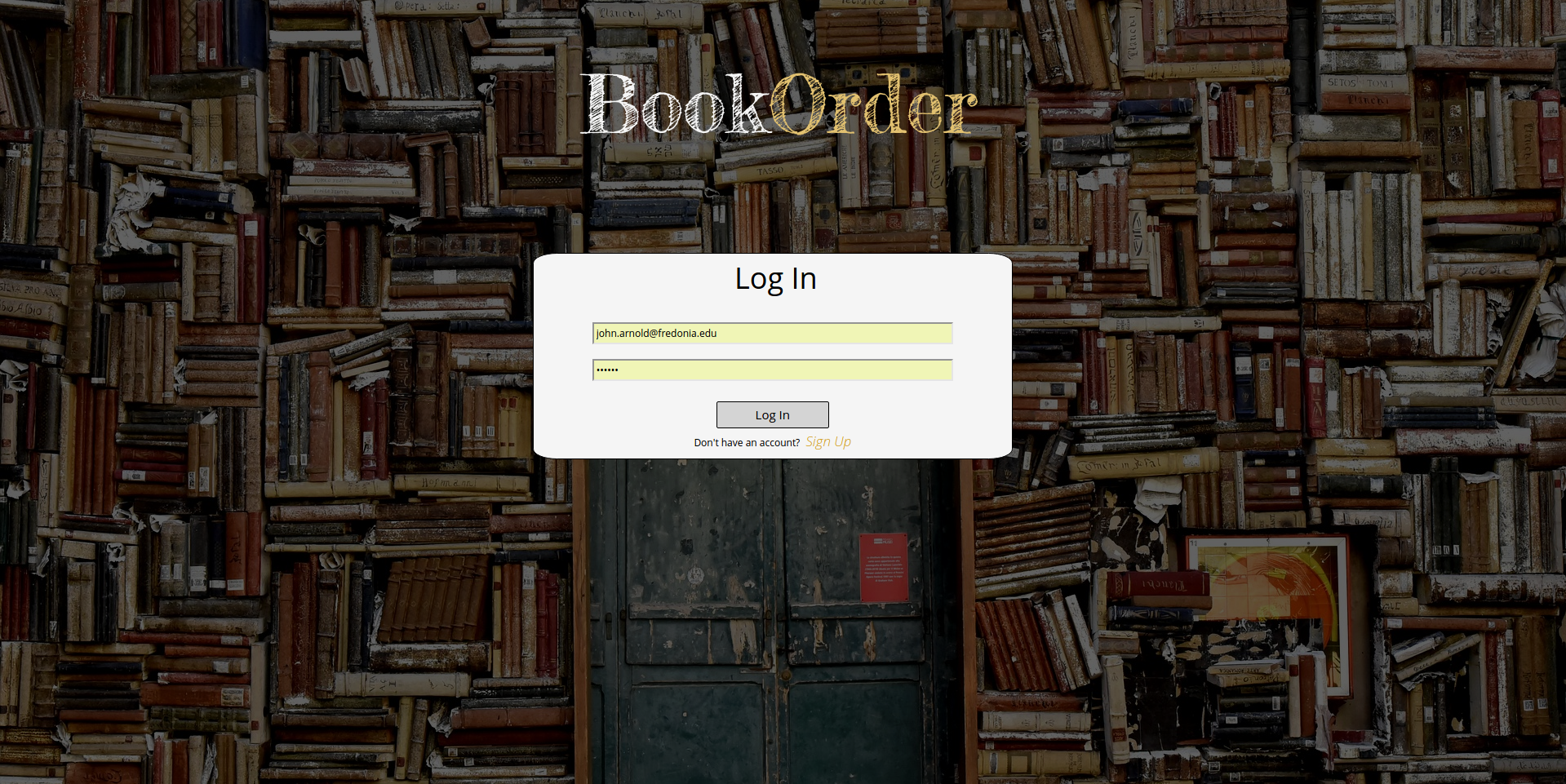
As previously stated, this project was chosen to be developed in Ruby on Rails. This decision was made on the basis that Rails provides a robust programming interface that, once the initial learning curve has been traversed, allows for quick development of a working, full-stack web application. The Rails version that was used was 5.0.1, the most current version of the framework, and we kept most of the initial configurations (ie. using sqlite3 as the database and puma as the server). Another large influence was the out-of-the-package integration with SASS, a compiled extension of CSS3. SASS offers much more concise and intuitive syntax, as well as extended use for functions and mixins. When being served to the client, all SASS files are compiled into a single compressed CSS file to save space and loading times. In addition, we used the JQuery library for animations and AJAX with the sever.

For anybody familiar with Rails, there are many conventions that are recommended to follow which was much of the basis for the design process. Rails follow the MVC architecture (Model-View-Controller) of web application design. Because Rails is, essentially, an example of an OODBMS traditional relations are defined as models for Rails to encapsulate and extend features with an object oriented design. As a result, we defined each teachers, courses, books and media, as well as a relational join table books\_for\_classes. Rather than writing vanilla SQL to create the relations and their respective cardinal relationships, Rails instead uses keywords such “belongs\_to” and “has\_many\_through” which automatically create the relationship internally, allowing for more concise syntax when referencing the object from a controller or view. This is also the way that constraints are placed on the system. A key example of this is the model for books in the Book.rb file.

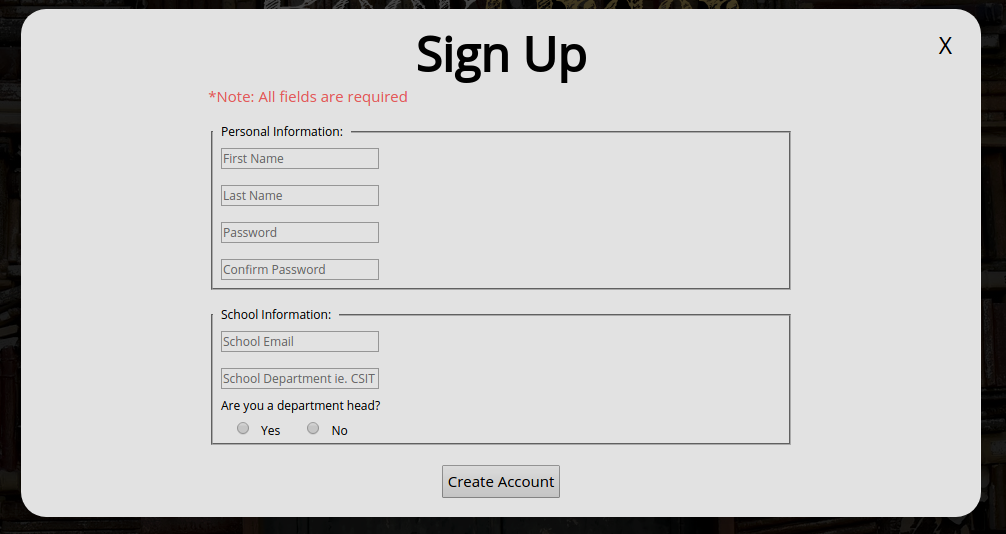
When working with a server and dynamic user content, server-side preprocessing is essentially. Rails handles this in a very unique way. The brunt of all the processing is done in the controller logics present in the /app/controller directory. There exists one such controller for each entity. Within these controllers are methods defined both by convention and the user. Each method can have an associated view (a web page with the same name as the method) or the method can be called in another fashion (ie. with an AJAX call or another method). By Rails and web application design standards, each controller has (or should have) an index, new, edit and destroy method to satisfy the CRUD criteria which is general convention for applications.

While this was generally the idea behind the design and implementation of this application, another inspiration was concurrency and fluidity in the application. To achieve this, we used a much more complex system of calling in which much of the information can be viewed from one page, rather than jumping around pages often. The primary example of this would be the pop-up forms that are present throughout much of the application. Each of these forms is located in what is called a Rails partial which is simply HTML and Rails preprocessing snippets to be loaded into a container through the use of AJAX. This allowed us to significantly reduce the amount of overall “pages” in the application and allowed fluidity in the user experience. Other contributing factors are the search feature implemented by Steven and the very actionable user interface. Overall, I would say we did a good job in providing a real-world application which is both functional and pleasant to use.

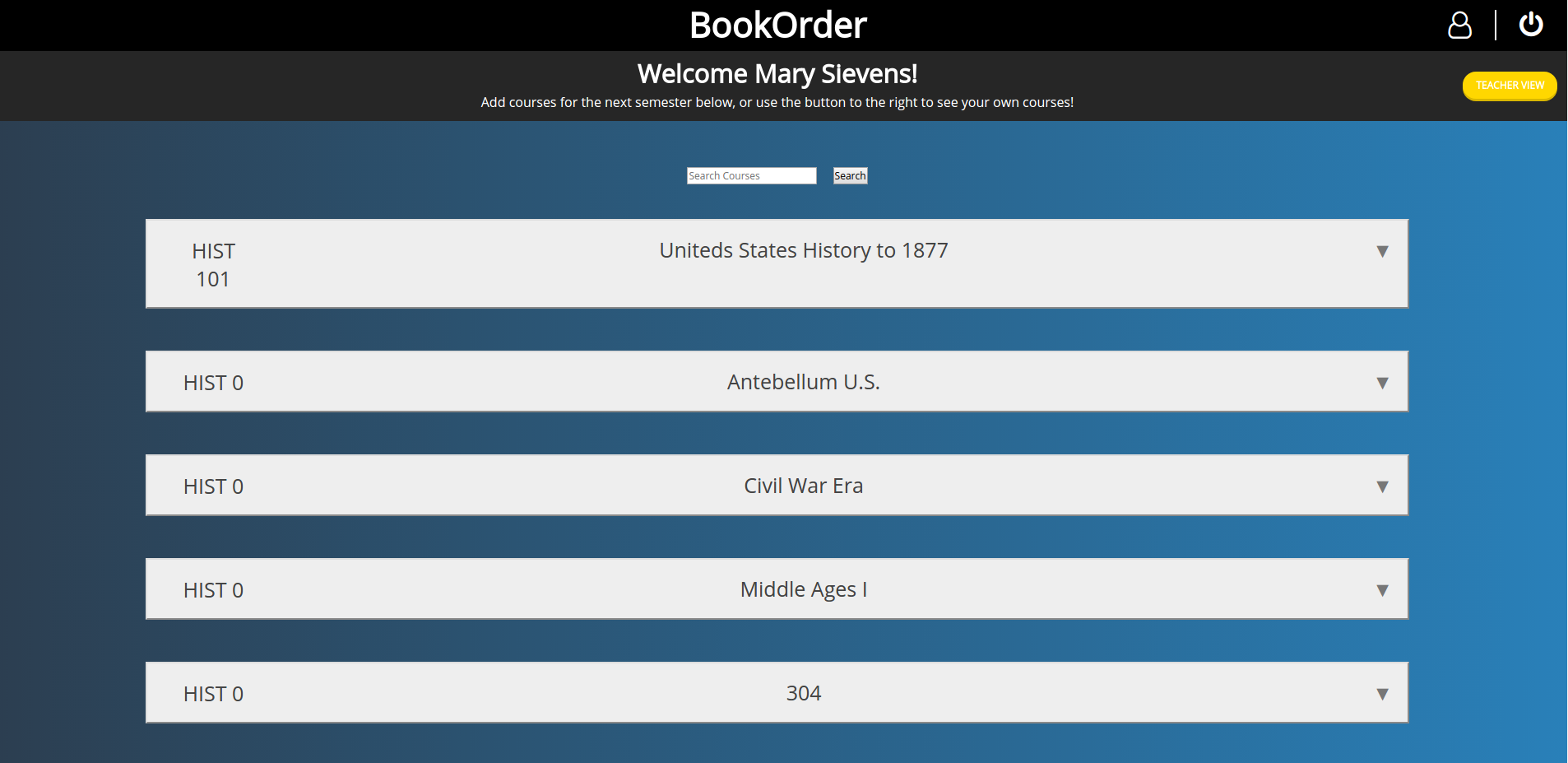
Thankfully, installing Ruby on Rails has become much less of a hassle, especially for those running Windows. There is a simple and easy to follow guide here: http://blog.teamtreehouse.com/installing-rails-5-windows . Once you have Rails set-up on your computer, the application is as easy to run as running ‘bundle install’ then ‘rails s’.

**III. Exhibition** 

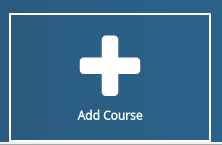
* This is the main page of the Book Ordering application. From here the user can sign in if they an account, or create one if they don’t.



* This is the Signup page, the user needs to put in all their correct credentials, and information. The email textbox will check to more sure an email has both the “@” symbol and ends with “.com”. The Email will serve as the user's username, the department will be used to group up all the professors and admins to ensure that Admins from another department can accidentally pull information from outside their department. Finally, Admin status is derived from the last question. If you answer “yes” you are given admin status, if not you are simply a professor.

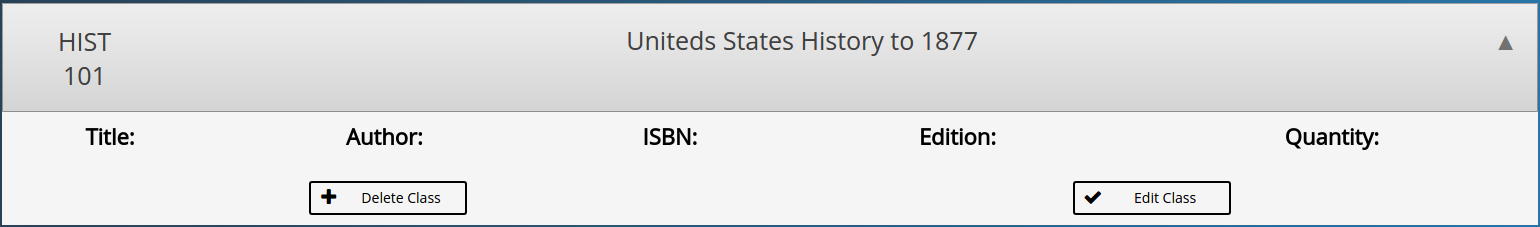


* This is the main admin page. This will be the administrator's main method of interacting with other courses, and offers the ability to search courses within their own department, as well as create new courses, edit courses, delete courses, search courses, and switch over to “teacher view” and, from all the pages you can always log out.

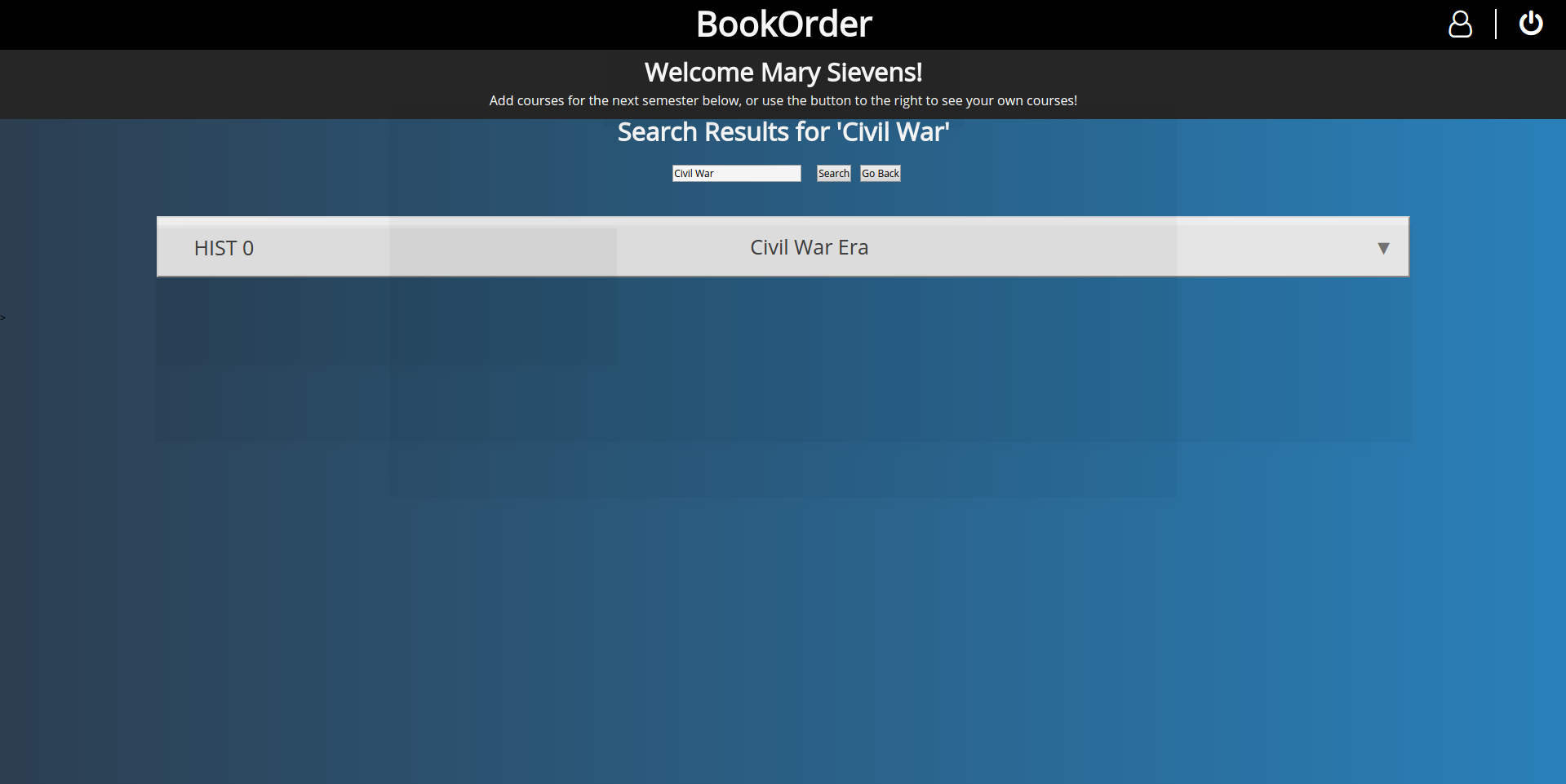




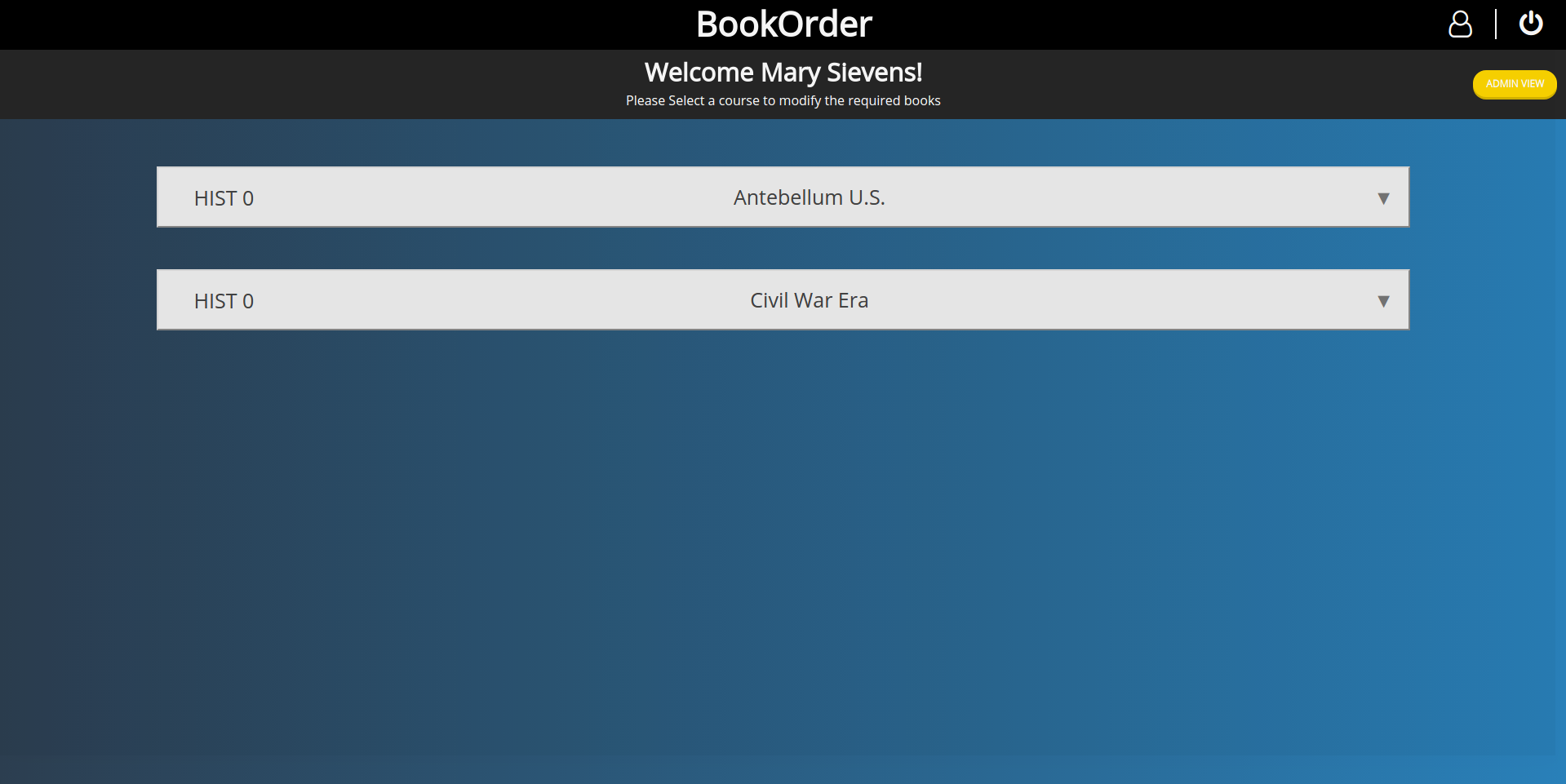
* Arguably, the most important aspect of the admin is to create courses, and assign them to professors within their department. The admins can assign course number, name, section and the professor that will teach that course.



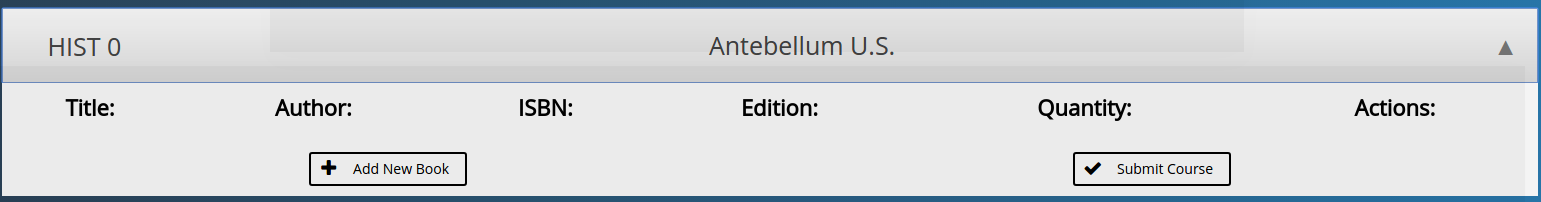
* Once the course is created the admin is free to edit it at any time. This includes course number, name of course, section, and professor. This was done to allow the admin to quickly make any changes if something were to arirse, or if there was an error made in creating the course. Furthermore, the admin is allowed to delete the course if, for example, the course wasn’t being offered.

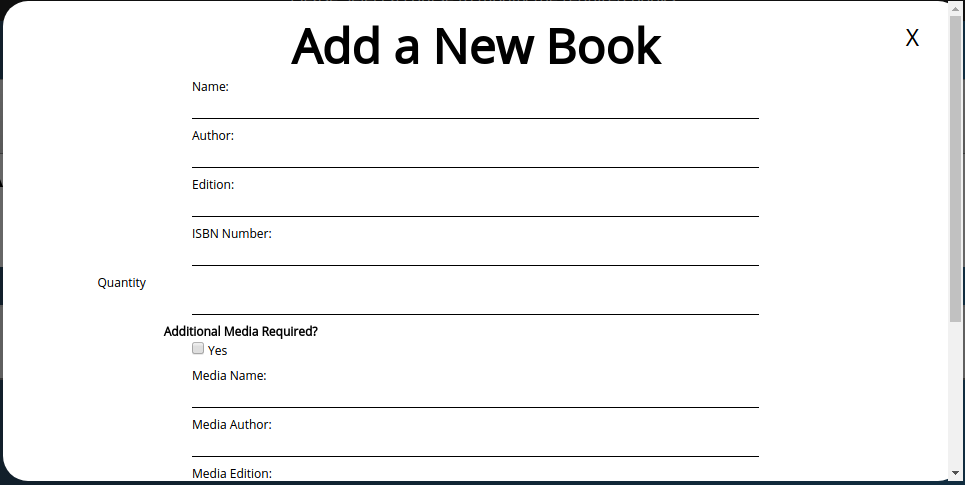


* This is the admin page after querying the database for a course name. As you can see it will only get the courses that the admin is looking for. Just like before they are free to edit and delete the course as they see fit. To ensure a decent search the character length of the search must be 3 characters or more, otherwise it will resend you to the main page.

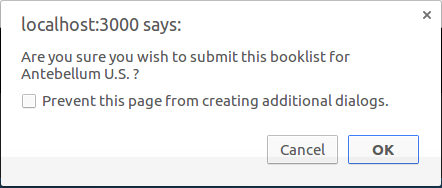


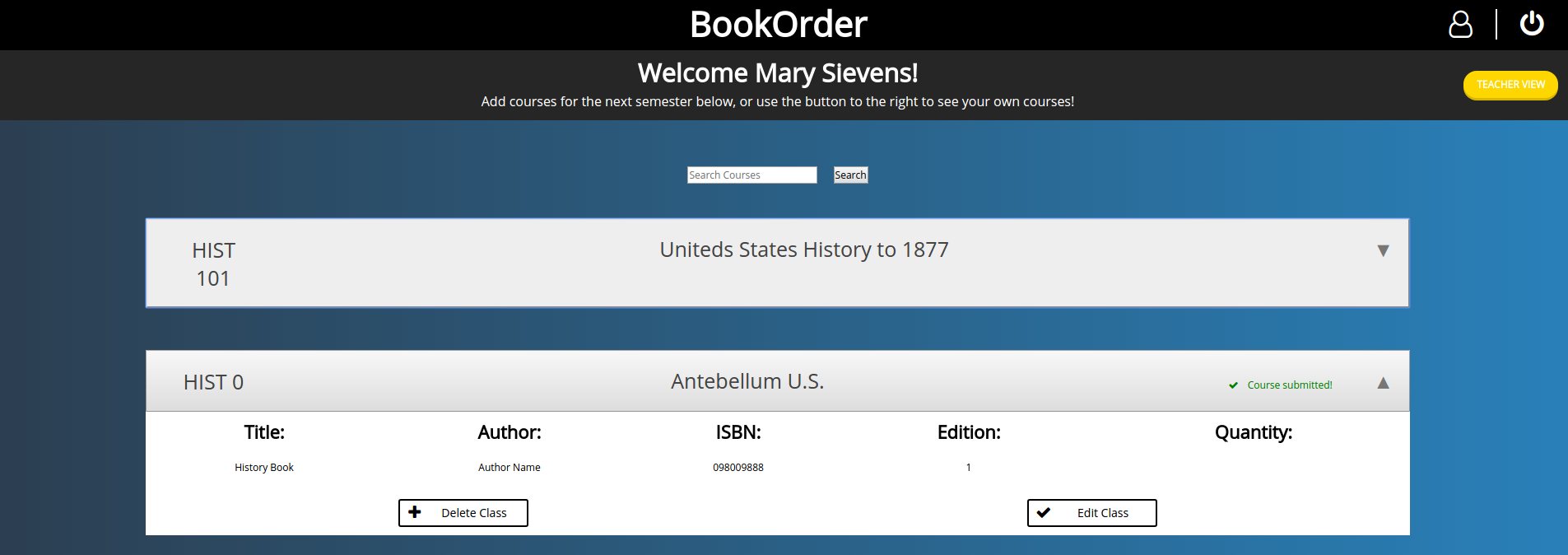
* This is the teacher page. As you can see we are still the same person “Mary Sievens” who as an admin, but to ensure they are no mistakes being made we have a teacher page for admins which will only display the courses they are teaching.



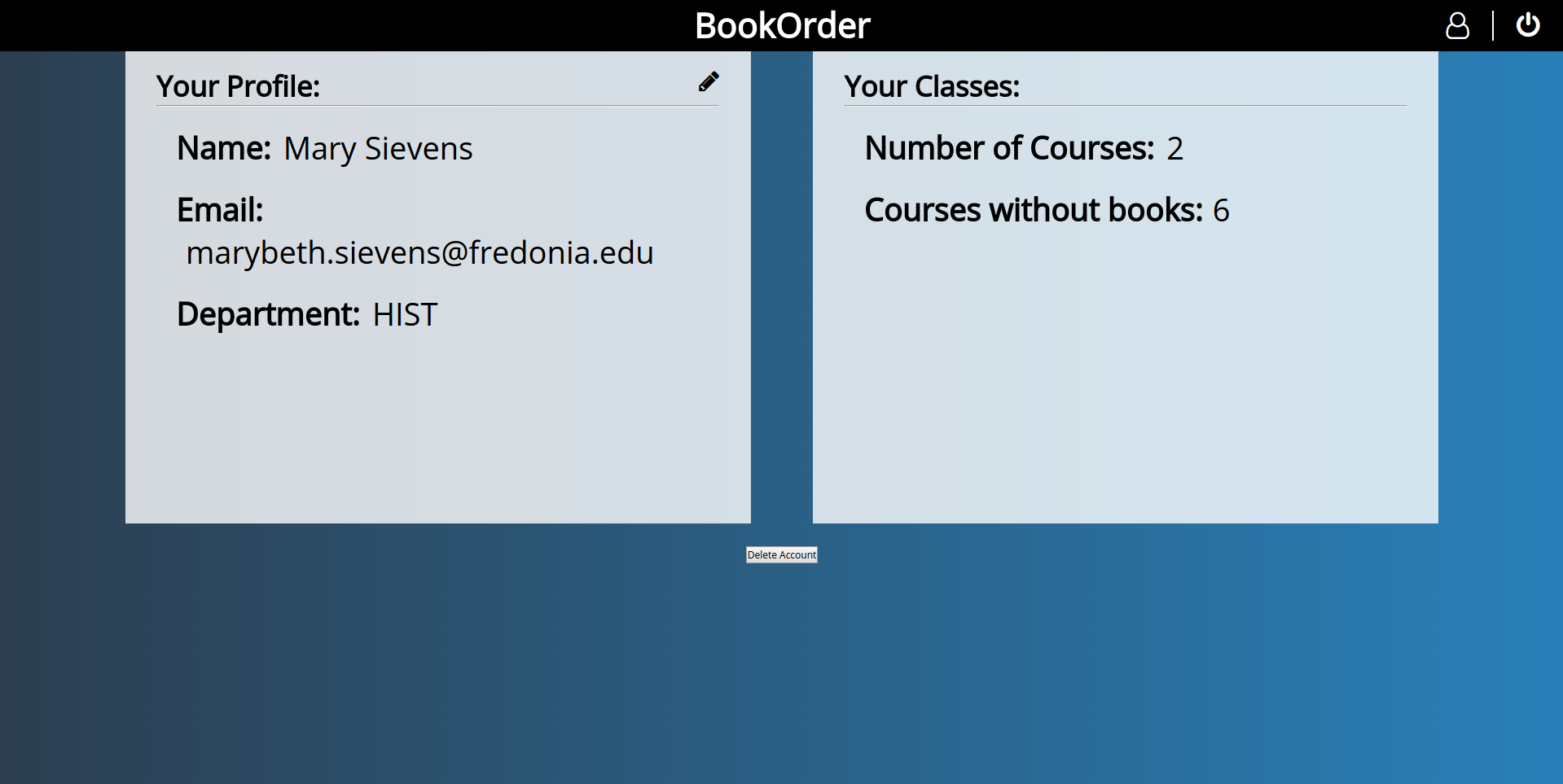


* When the teacher wants to add a new book, or submit the book list for that course, all they need to do is click on the drop down menu with the name of the course they wish to edit. Click on “Add New Book” and then they will be prompted to put in the books information.





* After adding the book the drop down menu will reflect that change. The teacher can add as many books as they need, and when they are done all they have to do is click on the “Submit Course” button. Before the book is sent the application will have a warning asking if they really want to submit this booklist for the current course. After that the change will be confirmed, and the admin page will be updated with a green check mark and text that says “Course Submitted”



* This page is used to rely the current user their profile information, such as name, email, and department. As well as information regarding classes that the current professor is teaching, and the current amount of courses without books.

**IV. Conclusion**

At the conclusion of this project, we have a very functional application that is very complete and useful. All of the goals and milestones that we had originally set out for our project has been met. We even had enough time to add smaller features that we hadn’t originally thought of. We were able to use a lot of the ideas and concepts that we learned in class, as well as some knowledge from outside the scope of the class. Things such as properly addressing relationships between tables, use of keys, normalization and constraints all factored heavily in the implementation of this project.