|  |
| --- |
| Mercer University |
| Project 3 |
| SSE 656 - Object Oriented Project Methods |
|  |
| **Tanya Do, Grey Newell, and John Robison** |
| **12/5/2014** |

|  |
| --- |
|  |

Contents

[Introduction 7](#_Toc405588365)

[Problem Statement 8](#_Toc405588366)

[Initial Meeting With the Customer 8](#_Toc405588367)

[Commonality and Variability Analysis 9](#_Toc405588368)

[Features and Requirements 10](#_Toc405588369)

[Use Cases and Use Case Diagrams 12](#_Toc405588370)

[Problem Submission Use Case 13](#_Toc405588371)

[Problem Judging Use Case 13](#_Toc405588372)

[Creating a Competition Use Case 14](#_Toc405588373)

[Use Case Diagram 14](#_Toc405588374)

[Technologies Used 15](#_Toc405588375)

[Architectural Analysis 18](#_Toc405588376)

[The Three Q's of Architecture 18](#_Toc405588377)

[Risk Analysis and Reduction 20](#_Toc405588378)

[Additional Discussion with the Customer 21](#_Toc405588379)

[Test Cases Used 23](#_Toc405588380)

[Automated Unit Tests 24](#_Toc405588381)

[Competition Creation Test 24](#_Toc405588382)

[Team Creation Test 24](#_Toc405588383)

[Problem Creation Test 24](#_Toc405588384)

[Competition Population Test 25](#_Toc405588385)

[Language Creation Test 25](#_Toc405588386)

[User Acceptance Tests 25](#_Toc405588387)

[Judge User Acceptance Test 25](#_Toc405588388)

[Team User Acceptance Test 26](#_Toc405588389)

[Development of the System 26](#_Toc405588390)

[Development of the Admin Portal Feature 27](#_Toc405588391)

[User Accounts 27](#_Toc405588392)

[Problem Creation 31](#_Toc405588393)

[Language Creation 33](#_Toc405588394)

[Contest Creation 36](#_Toc405588395)

[Development of the Problem Submission Use Case 38](#_Toc405588396)

[Problem Submission 38](#_Toc405588397)

[Go to the Team page 39](#_Toc405588398)

[Choose the Programming Language 41](#_Toc405588399)

[Choose the Problem 43](#_Toc405588400)

[Upload solution file 46](#_Toc405588401)

[Press Submit 47](#_Toc405588402)

[Development of the Problem Judging Use Case 48](#_Toc405588403)

[Setup Work 48](#_Toc405588404)

[Index View 49](#_Toc405588405)

[Competition View 49](#_Toc405588406)

[Submission Judging View 50](#_Toc405588407)

[Concepts at Work 51](#_Toc405588408)

[Test Results and Quality Analysis 52](#_Toc405588409)

[Conclusion 52](#_Toc405588410)

[Challenges 53](#_Toc405588411)

[Goals for the Next Iteration 54](#_Toc405588412)

[Takeaways 54](#_Toc405588413)

[AccountController.cs 57](#_Toc405588414)

[AdminController.cs 66](#_Toc405588415)

[CompetitionsController.cs 68](#_Toc405588416)

[Appendix: Source Code 72](#_Toc405588417)

[HomeController.cs 72](#_Toc405588418)

[JudgeController.cs 73](#_Toc405588419)

[LanguagesController.cs 75](#_Toc405588420)

[ProblemsController.cs 78](#_Toc405588421)

[TeamController.cs 81](#_Toc405588422)

[UsersController.cs 84](#_Toc405588423)

[AccountModels.cs 87](#_Toc405588424)

[CompetitionModel.cs 89](#_Toc405588425)

[LanguageModel.cs 91](#_Toc405588426)

[ProblemModel.cs 92](#_Toc405588427)

[SubmissionModel.cs 93](#_Toc405588428)

[TeamModel.cs 94](#_Toc405588429)

[Account/\_ChangePasswordPartial.cshtml 95](#_Toc405588430)

[Account/\_ExternalLoginsListPartial.cshtml 96](#_Toc405588431)

[Account/\_RemoveExternalLoginsPartial.cshtml 97](#_Toc405588432)

[Accounts/\_SetPasswordPartial.cshtml 98](#_Toc405588433)

[Account/ExternalLoginConfirmation.cshtml 99](#_Toc405588434)

[Account/ExternalLoginFailure.cshtml 100](#_Toc405588435)

[Account/Login 101](#_Toc405588436)

[Account/Manage.cshtml 102](#_Toc405588437)

[Account/Register.cshtml 103](#_Toc405588438)

[Admin/Competitions.cshtml 104](#_Toc405588439)

[Admin/Index.cshtml 105](#_Toc405588440)

[Admin/Languages.cshtml 106](#_Toc405588441)

[Admin/Problems.cshtml 107](#_Toc405588442)

[Admin/Users.cshtml 108](#_Toc405588443)

[Competitions/Create.cshtml 109](#_Toc405588444)

[Competitions/Delete.cshtml 111](#_Toc405588445)

[Competitions/Details.cshtml 112](#_Toc405588446)

[Competitions/Edit.cshtml 113](#_Toc405588447)

[Competitions/Index.cshtml 114](#_Toc405588448)

[Home/Chart.cshtml 115](#_Toc405588449)

[Home/Index.cshtml 116](#_Toc405588450)

[Home/Team.cshtml 117](#_Toc405588451)

[Judge/Claim.cshtml 118](#_Toc405588452)

[Judge/Competition.cshtml 119](#_Toc405588453)

[Judge/Index.cshtml 120](#_Toc405588454)

[Languages/Create.cshtml 121](#_Toc405588455)

[Languages/Delete.cshtml 122](#_Toc405588456)

[Languages/Details.cshtml 123](#_Toc405588457)

[Languages/Edit.cshtml 124](#_Toc405588458)

[Languages/Index.cshtml 125](#_Toc405588459)

[Problems/Create.cshtml 126](#_Toc405588460)

[Problems/Delete.cshtml 127](#_Toc405588461)

[Problems/Details.cshtml 128](#_Toc405588462)

**Table of Figures**

[Figure 1: Use Case Diagram 16](#_Toc405588463)

[Figure 2: MVC Interaction Diagram 17](#_Toc405588464)

[Figure 3: A Visual Representation of Loose Coupling 18](#_Toc405588465)

[Figure 4: A Visual Representation of Strong Coupling 18](#_Toc405588466)

[Figure 5: User Creation Page 29](#_Toc405588467)

[Figure 6: User Account Role Based Access Matrix 30](#_Toc405588468)

[Figure 7: User Edit Screen 31](#_Toc405588469)

[Figure 8: User Delete Confirmation 31](#_Toc405588470)

[Figure 9: Problem Create View 32](#_Toc405588471)

[Figure 10: Problem Read View 33](#_Toc405588472)

[Figure 11: Problem Update View 33](#_Toc405588473)

[Figure 12: Product Delete View 34](#_Toc405588474)

[Figure 13: Language Create View 35](#_Toc405588475)

[Figure 14: Language Read View 35](#_Toc405588476)

[Figure 15: Language Update View 36](#_Toc405588477)

[Figure 16: Language Delete View 36](#_Toc405588478)

[Figure 17: Contest Create View 37](#_Toc405588479)

[Figure 18: Contest Read View 38](#_Toc405588480)

[Figure 19: Contest Update View 38](#_Toc405588481)

[Figure 20: Contest Delete View 39](#_Toc405588482)

[Figure 1 Mockup of Team page 40](#_Toc405588483)

[Figure 2 Code snippet of HTML form 41](#_Toc405588484)

[Figure 3 Mockup of Language combo box 42](#_Toc405588485)

[Figure 4 Code snippet of Language list 43](#_Toc405588486)

[Figure 5 Code snippet of Language drop down list 43](#_Toc405588487)

[Figure 6 Mockup of Problem combo box 44](#_Toc405588488)

[Figure 7 Code snippet of database query 45](#_Toc405588489)

[Figure 8 Code snippet of Problems list 45](#_Toc405588490)

[Figure 9 Code snippet of HTML drop down list of ProblemList 46](#_Toc405588491)

[Figure 10 Mockup of File Explorer 47](#_Toc405588492)

[Figure 11 Code snippet of the file input 47](#_Toc405588493)

[Figure 12 Mockup of the submit button 48](#_Toc405588494)

[Figure 13 Code snippet of submit input 48](#_Toc405588495)

[Figure 14 Code snippet of the POST method 49](#_Toc405588496)

[Figure 21: Judge Index View 50](#_Toc405588497)

[Figure 22: Judge Competition View 51](#_Toc405588498)

[Figure 23: Submission Judging View 52](#_Toc405588499)

**Table of Tables**

# Introduction

Our development team is Trident Developments. Our team members are Tanya Do, Alexander Newell, and John Robision.



In this paper, our development team will provide a problem statement about out the web-based application we created, covering the purpose of the application. We will provide the transcription of interviews we had with our customer, Mercer University’s Computer Science Department, where they provide in detail the requirements of the application and the context in which the application will be used, follow up concerns, and discussion of the progress of development.

Additionally, to maintain transparency between our team and the customer, we discuss our decisions and development process in detail. We have included our commonality and variability analysis. We’ve extracted information from the interview and compiled a features and requirements for the judging application. Domain analysis of the application is covered, as well as use case scenarios.

We choose to use the MVC (Model View Controller) design pattern for this application. And this paper, we discuss what MVC is, and why it was the best match for the application. We will also talk detail our architectural analysis and the three questions that determine architectural significance. We will then delve into coding development, dividing the discussion into parts of the application that were use-case-driven and parts that were feature-driven. And we’ll review our testing methods and the results.

# Problem Statement

Our development team was approached by the Mercer University Computer Science Department (MU CS Dept.), to create a tool to judge their annual programming competition. The MU CS Dept. is trying to expose and encourage more students to the field of computer science and programming. As a result of this initiative, the MU CS Dept. organizes and hosts an annual programming competition aim at college students.

We are building a web-based application that will facilitate the operations of a programming competition. A programming competition will have multiple teams submitting solution to problems, and a team of judges pulling the problems, and grading them.

# Initial Meeting With the Customer

Our development team received an email expressing interest in our services, from Mercer's CS Department. To clarify the requirements and purpose of the project, we scheduled a meeting to discuss the program judging application.



**Trident Developments:** Hello! We're so glad you considered our development team for your project. Could you tell us more about the programming competition you are organizing?

**Mercer CS Department**: Absolutely. We want to encourage more students to study Computer Science. And we thought it would be best to start before they start college. We primarily market our programming competition towards other college students. And we are thinking about expanding the number of participants for the competition. And so what we are using now would not support a larger number of teams participating.

**Tri Dev:** Okay, great. Could you please walk us through how the programming competition works?

**MU CS:** Yes. For the competition we have a set of programming problems that we hand out to the teams. The teams may solve the problem in the language of their choice: C#, python, java, C++, etc. So when they code up the solution to the programming problem, they have to log into the system with their team login. Go to the submission page, choose which language the solution is in, and then upload their solution file.

**Tri Dev:** Meanwhile, what does the Judge do?

**MU CS:** Well after the teams submits their solutions, the Judge is able to see the upload and the accompanying information about it. And then they run the solution on their own computers and compare the output with the correct output. If the solution is correct, the team’s score is posted on to the score board. If not, the Judge will give feedback on the type of error the solution produced. And post to the scoreboard that the number of times the team has attempted the problem so far.

**Tri Dev:** And who sets up the competition, such as start time and team logins?

**MU CS:** We would like an Administer role that does that. The admin should be able to set up the parameters of a competition. Such as how many programming problems there are, what the contest start date and time is, when the contest ends, and how many teams logins are needed. Additionally, they should be able to act as a Judge or a Participant.

**Tri Dev:** Okay, great. Thank you. We’ll get to work.

# Commonality and Variability Analysis

Based on this meeting with the customer, our team began analyzing other similar systems to see how they had approached the problem of providing a Programming Team Judge client to schools and competitions. The analyzed systems are: The judging client used at the Association Computing Machinery (ACM) International Collegiate Programming Competition (ICPC) hosted by Baylor University, the judging client used by the Consortium for Computing Sciences in Colleges (CCSC) Southeast Competition, and the open sourced judge client created by Chip Bell (<https://github.com/chipbell4/Judge>). Each of these systems has strengths and weaknesses that our team used as references in the design of our system.

There were features from each of the examined systems that would be useful in the design of our Judging Client. Other than the common scoreboard that each client had, some of these features were: the ability for teams to ask questions to the judges given by the CCSC client, the neatness of the web interfaces given by Chip Bell's client, and the layout of ACM's scoreboard. Our team really liked the idea of allowing teams to interact with the judges in a way that allowed common problems to be easily clarified. An intuitive and neat interface allows users to interact with the system more efficiently and reduce any unnecessary stress or strain on the users. Our team felt that the advantages of the ACM scoreboard was that it would highlight all correct submissions with special colors (such as gold for the first correct submission) as well as provide the time of the correct submission which would allow users to follow the trends of the competition and enhance strategies. However, since the team/judge interaction was not within the scope of the project given to us by our client they have requested that we wait until the second revision of this application to implement this feature. The other liked features will still be implemented because they do fall within the scope of the project.

While each system had features that were admired, they also had features that our design team did not want to implement in our Judge Client. These features include: The unintuitive and slow design of the CCSC client, and the lack of user interaction with Chip Bell and ACM's clients. While an intuitively designed user interface enhances a user's experience, a slow and unintuitive design will inhibit the user. Our team felt that the CCSC system was confusing to use and would often cause major slowdowns at various stages of the competition. Even though our client would like to wait to add the user/judge interactions, we still felt that the lack of this interaction was a downfall of both Chip Bell and ACM's systems.

# Features and Requirements

After meeting with the customer and performing a commonality and variability analysis on the system, our team derived a list of the features that would define the system as well as the requirements that would compose each of these features. These features are listed below:

* User Accounts
  + Teams
  + Judges
  + Admins
* Creating a Contest
  + Creating Problems
  + Adding Teams
  + Setting the Usable Languages
  + Setting the Start and End Times
* Viewing the Scoreboard
* Submitting a Solution
  + Uploading Source Code
  + Selecting the Used Language
* Judging a Submission
  + Claiming a Submission
  + Downloading the Judging Packet
  + Choosing the Appropriate Result
    - Correct
    - Wrong Output
    - Compile Time Error
    - Runtime Error
    - Presentation Error
* Reviewing Judged Submissions

The User Accounts will be responsible for allowing users to have different experiences with the system based on what their role in the competition is. These accounts will limit each user's operations and responsibilities within the system so that they are not able to overstep their limits. These responsibilities are as follows:

* Team Accounts will be the most common user to the system. They will be responsible for allowing participants to be able to submit solutions to problems and review their judged solutions. The submissions will consist of the user's source code which will be uploaded to the server as well as the programming language that was used in the submission.
* Judge Accounts will be responsible for allowing Team Accounts to have correct (or incorrect) submissions. They will be able to claim a submission and then judge it based on the output from the submitted source code. To allow a Judge the ability to judge submissions, the system will provide the Judge with a judging packet that consists of: the submitted source code, the judging input, and the judging output. Using these files, the judge will be able to run the source code with the judge input and compare this output with the provided judge output. The possible reviews that a Judge can give a submission are: Correct, Wrong Output, Compile Time Error, Runtime Error, and Presentation Error.
* Admin Accounts will be responsible for setting up and managing competitions. They will be able to add users (such as Teams and Judges), problems, and usable languages. The Admin will also be able to create a competition using this information as well as set the start and end times of these competitions.
* All users will be able to view the scoreboard that will show the ranking of the teams based on their number of correct submissions and penalty points accrued.

# Use Cases and Use Case Diagrams

To ensure that our team has a more complete understanding of how the system should work, use cases were derived for the major interactions that users can have with the system. These interactions include:

* Submitting problems
* Judging problems
* Creating contests
* Creating problems
* Creating users

Our design team has determined that there will be three types of users for this system: Admins, Teams, and Judges. A use case of the main path was created for each of these types of users.

An Admin will be the user that sets up the competitions as well as creating user accounts for the Teams and Judges. The Admins will be able to specify all of the aspects of a competition such as: the Problems used in the competition, the Users participating in the competition, and the Languages that the Users will be able to submit their source code in. The Admin will also decide the start and end times of each competition.

Teams will be the main users of the system. They will be the users who are solving and submitting answers to the problems posted by the Admins. They will be able to decide which problem they want to solve and in what language they want from the lists provided by the Admin. After they have submitted their solutions, they will be able to review the judging of their solution and resubmit if necessary.

The final user group is the Judges. The Judges will be responsible for reviewing the Teams' submissions. To review these submissions, they will be proved the submitted source code as well as judging inputs and outputs. The judge will run the source code with the provided inputs and then compare the produced output to the provided output. Based on this comparison, the Judge will select the most appropriate response (i.e. "Correct Solution" or "Wrong Output").

The following subsections contain these use cases as well as the use case diagram that shows all possible interactions that a user can have.

## Problem Submission Use Case

1. The user navigates to the web URL associated with the judge application
2. The user logs in to the judge application with a team's credentials
3. The user views the scoreboard and selects a problem to solve
4. The user clicks the "Team" link
5. The user selects the solved problem from the drop down, selects the language used, and uploads the source file
6. The user waits for the problem to be judged
7. Once the problem is judged, the user returns to the Team page and views the judged problem to see the judge's comments

## Problem Judging Use Case

1. The user navigates to the web URL associated with the judge application
2. The user logs in to the judge application with a judge's credentials
3. The user clicks the "Judge" link
4. The user selects which competition he or she would like to judge submissions for.
5. The user claims a submitted problem
6. The user downloads the zip file that contains the submitted source code, the input file, and the output file
7. The user runs the source code with the input file as the input to the system
8. The user compares the given output to the generated output
9. Based on this comparison, the user selects a judging option for the claimed problem

## Creating a Competition Use Case

1. The user navigates to the web URL associated with the judge application
2. The user logs in to the judge application with an Admin's credentials
3. The user clicks the "Admin" link
4. The user clicks the "Users" link
5. The user clicks the "New User" button
6. The user enters information for the new team and clicks the "Create" button
7. The user clicks the "Problems" link
8. The user clicks the "New Problem" link
9. The user enters a Problem name and provides the judge input and output and clicks the "Create" button
10. The user clicks the "Languages" button
11. The user clicks the "New Language" button
12. The user enters the language of their choosing and clicks the "Create" button
13. The user clicks the "Contests" link
14. The user clicks the "New Contest" button
15. The user selects the added team and problem, and selects start and end times
16. The user click the "Create" button

## Use Case Diagram



Figure 1: Use Case Diagram

# Technologies Used

This project employed the use of the ASP.NET MVC (Model-View-Controller) framework. The framework streamlines adoption of the MVC design pattern which encourages designs to be loosely coupled through a separation of concerns, and also enforces MVC related standards in our code. This separation is achieved by using the three main components: Models, Views, and Controllers. Models define the schema by which system data is stored and interacted with. Views are responsible for constructing the façade the user interacts with and displaying data from the appropriate Models in a human-readable fashion. Controllers are the “under-the-hood” components which respond to user inputs by updating the models and views accordingly. Below in Figure 2 is a diagram that visualizes the interaction between these components.



Figure : MVC Interaction Diagram

The ASP.NET framework builds on the MVC design pattern by relying on a "convention over configuration" approach which reduces the amount of code required to implement the design pattern for a project. This approach enforces certain facets of the design pattern, such that items are placed in the correct directories and named according to the MVC Schema; i.e. Controllers must be in the Controller folder and its name will end with "Controller.” Not only does this convention based approach reduce code requirements, it also aids in overall readability of the system.

Due to the loose coupling offered by the MVC design pattern, our team chose ASP.NET MVC as the framework for this system because it allowed for code reuse and distributed development in parallel. Since the models, views, and controllers are separated, they can easily be reused within another application, individually or as a whole. An example of this type of code reuse would be if the customer requested a mobile application. Model and Controller classes could be reused as-is for the backend, and all developers have to do is define new views for the mobile platform. This separation also allows components to be designed and built separately. One developer can build the store user interface in the views while another developer designs the underlying business logic in the models concurrently.

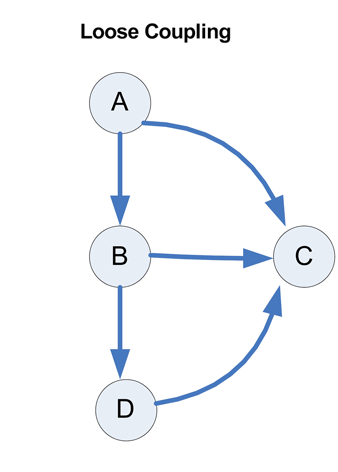


Figure : A Visual Representation of Loose Coupling

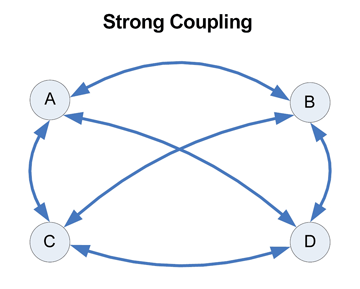


Figure : A Visual Representation of Strong Coupling

In addition to the aforementioned benefits of MVC's separation of concerns listed above, our team decided to develop with the ASP.NET MVC Framework for its use of .NET languages (in particular C# and Visual Basic) which we are already familiar with, and due to the success of our previous project employing this framework. In addition, the Razor view engine was easy to work with and teach to our newest developer, who actually became proficient with the engine quickly enough to implement most of our Views. C#, our .NET language of choice, is extremely powerful and easy to develop with due to its number of available libraries and the use of Microsoft Visual Studio’s Intellisense utility. Intellisense can auto-fill code for the developer such as variable and method names, whether the variable is a member of a project or one of its associated dependencies. The Razor view engine is a combination of HTML and either C# or Visual Basic, which makes it very natural for .NET developers to write and learn. The use of C# in the markup allows for very powerful dynamic web pages. Razor also allows the use of layouts which enables developers to have a single Razor file act as a template for all other views. These layouts reduce duplicate code by encapsulating common view elements into the layout. The final advantage of Razor that will aid in development is the ability to use Intellisense to quickly write the markup and code contained in the Razor files.

# Architectural Analysis

As a means of ensuring that the project is completed to the best of our abilities and within the time and budget constraints provided by our customer, our group performed an architectural analysis on the system to determine the possible bottlenecks and dependencies of the system. To establish the most architecturally significant portions of the system, our team looked at what are known as the "Three Q's of Architecture". Once the architectural significances were determined, our team used this knowledge to determine where the major risk areas were and devise a system that would reduce this risk as much as possible. These analyses are contained in the following subsections.

## The Three Q's of Architecture

A common method of determining the architectural significance of features and components in software development is to ask the Three Q's of Architecture for each major feature or component, which are as follows:

* Is it the essence of the system?
* What does the feature mean?
* How do you implement the feature?

Features are deemed more architecturally significant based on the answers that the developers have for each feature. The qualifications for significance are if the feature makes up the essence of the system, if the meaning of the feature is unclear to the developers, or if the developers are unsure of how to implement the feature. Features that meet more of the criteria are more architecturally significant than features that only meet one criterion or meet none of these criteria. Each of these three criteria is as important as the others, so these questions hold no additional weighs when determining architectural significance.

The features that have been determined to be the essence of the system are: user accounts, the admin portal, the ability to submit solutions, the ability to judge solutions, and a running scoreboard that shows the ranking of the teams. Without these features, the system would not meet the expectations of the customer and would not be considered a complete product. Users of the system will have to be able to log in to the system to interact with it. To allow these interactions to be specific, there will need to be three classes of users: Admins, Teams, and Judges. The admin portal will be used by Users in the role of Admin and is responsible for managing and creating all aspects of a competition: the problems for teams to solve, the users who can participate in the competitions, the usable languages, as well as the start and end times of the competitions. Having the ability to create and manage competitions is only useful if users are able to actively participate in the competitions. Because of this, it is important to establish a means for users in the Team role to submit their solution's source code to be judged. However, being able to submit problems is useless without having a mechanism that will assess the submission. To allow this to happen, a system will have to be implemented that allows a type of user who is a Judge to review a Team's submitted source code. To allow users to see their relative performance in a competition, the results of a Judge's review will be compiled into a scoreboard that is accessible by all user types.

Due to the nature of this system, there were no aspects of the project that our design team had uncertainties on the meaning of. This reduces the amount of risk inherent in the development of this system.

The features that our team is currently unsure of how to implement are the submission and judging of solutions. The fact that our team is unsure of how to implement these features makes them significant because they have the potential to cause blockages in development as well delays and budget problems. The component of the submission feature that our team is unsure of how to implement the ability to upload their source code and store it in a database in a way that can be later accessed to be judged. There will need to be a light weight solution for this problem that our team will decide on. After this problem is figured out, our team will then have to figure out how to use this submitted source code, the uploaded judging input, and the uploaded judging output to create a judging packet that will be a .zip file. This file, as well as the files that are contained within, will need to be created programmatically. Our team has no prior experience with handling these types of requirements, so extensive research will need to be conducted into these fields.

After performing this analysis using the Three Q's of Architecture, our team has determined that the most architecturally significant features of the system are: the user accounts, the admin portal, the ability to submit and judge problems, and an updatable scoreboard. The feature that was not considered architecturally significant was the user/ judge interaction. This feature was not considered significant for the following reasons:

* It is not a vital component for the system
* The feature had a clear and obvious meaning
* It has negligible time and resource requirements to complete
* It was deemed unimportant for this iteration by our client

## Risk Analysis and Reduction

After determining the architecturally significant components of the system, our team had to decide a planned order in which to develop these features in such a way that will reduce the risks inherent in development. Even though it did not contain the most qualifications from the Three Q's Analysis, the User Accounts will be developed first since all of the other features will build off of this. Having these accounts set up first will reduce any risks associated with integration of the other features. Also, since our team has prior experience with setting up a user account system, this will be a task that should cause little to no difficulties. This feature is set with the highest priority because there will need to be distinct user roles that will ensure that users can only access sections of the system that they are supposed to. An example of this is that a Team should not be able to access the Judging pages because they would then be able to artificially inflate their rankings by wrongfully judging their own work.

Following the development of the User Accounts, the Admin Portal will be developed because it is used to set up all aspects of the contests which the other features rely on. The development of this feature should not involve any delays or blocks because it is a type of feature that our team has dealt with before. Due to the other features' dependencies on this component, it was selected to be of a higher priority than the following features. Without this component, it will be a difficult process to create Users, Problems, Languages, and Competitions that the other features will need to be usable as well as to be tested for correctness.

After these two features have been developed, both the Solution Submission and Judging features will have an equal importance. However, our team has decided to deal with the Solution Submission first because it will be the more used feature of the two. The largest constriction of this feature is that our team will need to research how to store the source code in our database that will be able to later be accessed by the Judging feature. This task is of great risk to the development of this system since it is such a foreign task for our team to complete. After overcoming this hurdle, our team will develop the user interface that will allow a Team user to upload this file as well as specify which language was used. The task of designing and developing the user interface will be a trivial task compared to the task of storing the submitted source code, so it will be the last task completed of this feature.

After finishing this feature, our team will develop the system for Judging Submissions. The completion of this feature relies on the research that will be done for the Solution Submission feature. It will also require research into creating files as well as putting those files into a .zip folder programmatically. This is required because these files that comprise the .zip file will be used as the Judging packet when a Judge user claims a submission. This task is of great risk to the development of this system since it is such a foreign task for our team to complete. As with the Solution Submission feature, the next step in completing this feature is to design and develop the user interface that will allow a Judge user to claim a submission, download this judge packet, and then give a review of the submission based on the results of the judging packet. This will also be a trivial task compared to the task of creating the judging packet, so it will be handed last in the completion of this feature.

The final architecturally significant feature that our team will work on is the Scoreboard. The scoreboard will be an image that displays the current rankings of all Teams based on the number of problems that they submit correctly as well as the number of penalty points accrued. This task, while architecturally significant, poses little to no risk to the development of the system. Due to this lack of risk or dependency, it will be developed last. The scoreboard will need to be viewable from all types of users (Admins, Teams, and Judges) and will also need to update on its own. This scoreboard image will be programmatically generated whenever the home page of the service is accessed, which is a trivial task using built in C# libraries.

Using a set schedule such as this will greatly reduce risk in both the design and implementation stages of development. The ordering of the completion of these tasks is as follows: the creation of User Accounts, the development of an Admin Portal, the development of the Solution Submission process, the development of the Submission Judging process, and finally the implementation of the scoreboard feature. For the reasons listed above, our team believes that this ordering of tasks will mitigate many of the risks that are inherent in the development of this system and will help to ensure that the project remains on time and on budget. This risk mitigation plan will also allow our team to carefully plan for any bottlenecks or complications in the development process and be able to appropriately and efficiently deal with these concerns.

# Additional Discussion with the Customer

At Trident Developments, we pride ourselves on communication skills and transparency with our customers. After intensive analysis and planning, we decided to give Mercer’s Computer Science Department a status report on their judging application. So gave them call and a set up an in-person meeting.



**Trident Developments:** Hi! We’re glad you could make it. We wanted to sit down and give you an update on the development of your Judging application.

**Mercer University Computer Science Department:** Great! We’re glad to hear that.

**Tri Dev:**  During our commonality and variability analysis, we compared the features and requirements of your Judging application with existing ones.

The three we took a look at were the ones used by the ACM International Collegiate Programming Competition, Consortium for Computing Sciences in Colleges Southeast Competition, and the open source one created by Chip Bell.

**MU CS:** Okay, good. Those are the other judging applications that we took a look at, before coming to you guys.

**Tri Dev:** We found that the features that hindered the other Judging applications was implementation of an unintuitive design. And slow reaction time to user input.

**MU CS:** Right. What we saw with the CCSCSC’s Judging application that we really liked, was the feature that allowed participants to ask the judges questions about the problems they were solving.

**Tri Dev:**  We also see that as a positive, however, as we’ve discussed earlier, it would be best to wait for the next iteration of the application to implement that feature.

**MU CS:** We definitely understand. We did just want to iterate that it is a feature that we want in the future.

**Tri Dev:**  We also decided to use the Model-View-Controller (MVC) design pattern. ASP.NET has a framework that supports this pattern. This framework encourages the design of the system to be loosely coupled, via separation of responsibility.

The decoupling of the different parts of the system will allow for our team to work on multiple parts of the system simultaneously. This increases productivity and decreases development time, resulting in you receiving your application much quicker!

**MU CS:** That sounds great!

**Tri Dev:** Additionally, it will be much easier to make changes if you wanted to update the system. And therefore cheaper for you too.

If you wanted us to just go and change the way the application looked, we could easily and quickly do that with little or no disruption to you. The way MVC is set up, we the looks of the application should have no bearing on how the system works.

**MU CS:** Great. We definitely like the maintainability aspect of your plan.

**Tri Dev:** And in front of you are the use-cases and use-case diagrams we’ve compiled for you.

**MU CS:** Could you explain to us what they are?

**Tri Dev:** Yes. Definitely.

So in the individual use-case scenarios, we start off by picking a user and then picking an initial action they would do in the system and following it to completion. We repeat this until all possible actions have been covered.

**MU CS:** Great so what is this diagram then?

**Tri Dev:** This diagram is a use-case diagram. And it the same thing as the use-case scenarios. Except, this puts all of the scenarios into a graphical format. The graphical format is so you can see all of the use-cases at once. And also, how the different roles relate or do not relate to each other.

**MU CS:** Okay. The project plan looks great so far.

**Tri Dev:** Definitely feel free to let us know if you have question or anything.

**MU CS:** Thank you for the update. Please continue to keep us informed.

# Test Cases Used

In this section will be described the test scenarios used in development of our application. The first section covers the automated unit tests our team used for Functionality Driven Development. The team determined automated unit testing to be most important when development is navigated by functionality; that is to say, each unit of functionality introduced should be tested. Building on the necessity of our unit tests, the feature-driven portion of development encompassed user authentication, CRUD data manipulation, and the basic building blocks of our other systems. If even one method malfunctions, every other use case generates invalid input as well. For our use case driven development, however, the team elected to employ the use of acceptance testing. An automated unit test doesn’t quite satisfy a use case on its own; What if the internal objects and their methods function perfectly, but don’t produce or display a correct and functional interface for the user to interact with? For the use case driven portions of development user acceptance testing was employed. Developers manually compiled the project, ran the IISServer, and navigated to the web page our app was hosted at to test these functionalities *in the way the user will*- a use case! Used in conjunction, automated unit testing and user acceptance testing can help enforce good convention in any development undertaking. While it is difficult to measure testing coverage of unintended usage scenarios, we estimate the coverage of our tests to range between 80 and 100% of intended use scenarios.

## Automated Unit Tests

### Competition Creation Test

This test ensures that competition objects may be created programmatically within our application.

1. A start date was defined as December 1, 2014 5:30PM.
2. An end date was defined as December 1, 2014 8:00PM.
3. Name was defined as “Test Competition.”
4. A Competition was created with the above values.
5. The competition’s start date, end date, and name were compared to the above values.

### Team Creation Test

This test ensures that team objects may be created programmatically within our application.

1. Name was defined as “teamA”
2. A Team was created with the above value.
3. The team’s name was compared to the above value.

### Problem Creation Test

This test ensures that problem objects may be created programmatically within our application.

1. Name was defined as “Test”
2. A problem was created with the above values.
3. The problem’s name was compared to the above value.

### Competition Population Test

This test ensures that an existing competition object may be populated with teams and problems.

1. A competition was created.
2. A list of teams was created containing one team, named “teamA.”
3. A list of problems was created containing one problem, named “Test.”
4. Teams and Problems were added to the competition from step 1.
5. The competition’s teams and problems were compared to the original.

### Language Creation Test

This test ensures that programming language objects may be created programmatically within our application.

1. A name was defined as “Python.”
2. A file extension was defined as “.py”
3. A language object was created using the above values.
4. The language object’s name and extension were compared to the original.

## User Acceptance Tests

### Judge User Acceptance Test

This test ensures user acceptance for the role of Judge. Since there aren’t yet any end users, our development team served as the user. This test ensures both that the underlying functionality works correctly and also that the user interface is adequate and intuitive. The Problem Judging use case will be totally satisfied if this test is passed.

1. The user navigates to the Programming Competition Judge’s homepage.
2. The user logs in with a valid username and password associated with a user account possessing Judge Permissions.
3. The user navigates to the Judging page via the menu navigation aide.
4. The user selects a competition to judge submissions of from a list generated by the page of all competitions, sorted chronologically by end date.
5. The user is displayed a list of problems within the selected competition and the relevant problem submissions. The user may select a problem submission and claim it by clicking a link located within the associated table row.
6. The user is prompted to download the Judging packet, comprised of Sample Input, Expected Output, and Submitted Source Code in a .zip archive.
7. The user locally compiles the provided source code and runs it with the provided input.
8. The user compares the given output to the generated output.
9. The user is prompted to input a judging outcome and a score for the submission.
10. Once the evaluation is submitted, the user is returned to the list of problem submissions, where the previous submission has been updated with his evaluation and user id.

### Team User Acceptance Test

This test is meant to ensure that teams can add their problem submissions to competitions.

1. The user navigates to the Programming Competition Judge’s homepage.
2. The user logs in with a valid username and password associated with a user account for his team.
3. The user is shown the current scoreboard for the active competition his team is participating in and is prompted to select a problem to solve.
4. The user clicks the "Team" link attached to the associated problem.
5. The user selects the solved problem from the drop down, selects the language used, and uploads the source file.
6. The user waits until the problem has been appropriately evaluated by the judges.
7. Once the problem is judged, the user returns to the Team page and views the judged problem to see the judge's comments, score, and judging outcome.

# Development of the System

Along with the Test Driven Development methodology that our team implemented, we also used Feature Driven Development. While Test Driven Development deals with ensuring that the behavior of a system is as expected, Feature Driven Development deals with deciding logical sections to code a solution to the given problem. Feature Driven Development is the process of working on a specific piece of functionality one at a time. Feature Driven Development is most useful in systems that have many disconnected features or features that don't interact much. One advantage of this approach is that it allows developers to have working features very quickly which can be shown to customers as proof of work completed. Another advantage is that this approach ensures that all features are accounted for and are fully fleshed out. Feature Driven Development was used in the design of the Admin Portal, discussed in the following subsection.

Another development methodology we employed in bringing this project to successful completion is Use Case Driven Development. This process means selecting your features and components to develop according to a certain scenario from a use case. When a Use Case Driven development cycle is over, all features necessary to perform every scenario within a certain use case should be implemented (as much as necessary) and tested. Tackling development with this strategy is a more user-centric and less granular approach as compared to feature driven development, and is considered a “big-picture” approach by the authors of *HeadFirst OOA&D*. Developers often utilize this approach when the system has many scenarios and processes that comprise a single, cohesive user experience. The Problem Submission and Problem Judging portions of our system were developed in this manner.

## Development of the Admin Portal Feature

For the development of the Admin Portal, our design team decided to follow a Feature Driven Development methodology. This was because there was no main path to follow. An Admin would have many different tasks that they could be performing at any time that would be used in the creation and management of a competition. Since this Admin Portal was disconnected from the rest of the system, this Feature Driven Development approach was ideal. Once our team decided that we were going to take a Feature Driven Development approach to the Admin Portal, we began modularizing the portal further so that we could design the system in an iterative fashion. The smaller sections of the Admin Portal that we derived were: creating a Competition, creating Problems, registering Users, and creating Languages.

### User Accounts

The first component designed was the User Accounts portion. This was developed first because all of the other features, including the Admin Portal, relied on having a User Account. This component was also placed as the first task in the schedule derived in the Risk Analysis and Reduction due to these dependencies, which if left unresolved could cause unnecessary risks. The base of these User Accounts was provided in the ASP.NET MVC Internet Application template, so the development was not started from scratch which ensured that this functionality would work as intended. Using this template, our team began adding in the necessary additional components of the User Account, such as the User Roles. As stated before, the roles that a User can have are Admin, Team, or Judge. To accomplish this, we set up a database table that would keep track of each user's available roles, using a proven computer security method called Role Based Access Control. This would ensure that users can only access systems that they are supposed to, for example: a Team user cannot access the Admin Portal.

After setting up the Role Based Access Matrix, our team began setting up the interface that would allow an Admin to create and manage Users. Due to the nature of the system, it was not desirable to allow users to create their own accounts as this would open the system to anyone who could navigate to the web site and would remove the need or the want for teams to register to compete. Giving the Admins full control over the User Profiles also allowed for the easy management of users in a contest so that users could very easily and quickly be added, deleted, or changed. Our team based the interface on standard website registration pages, but added the ability to define the created user's role in this same interface. The interface can be seen in Figure 5 below.

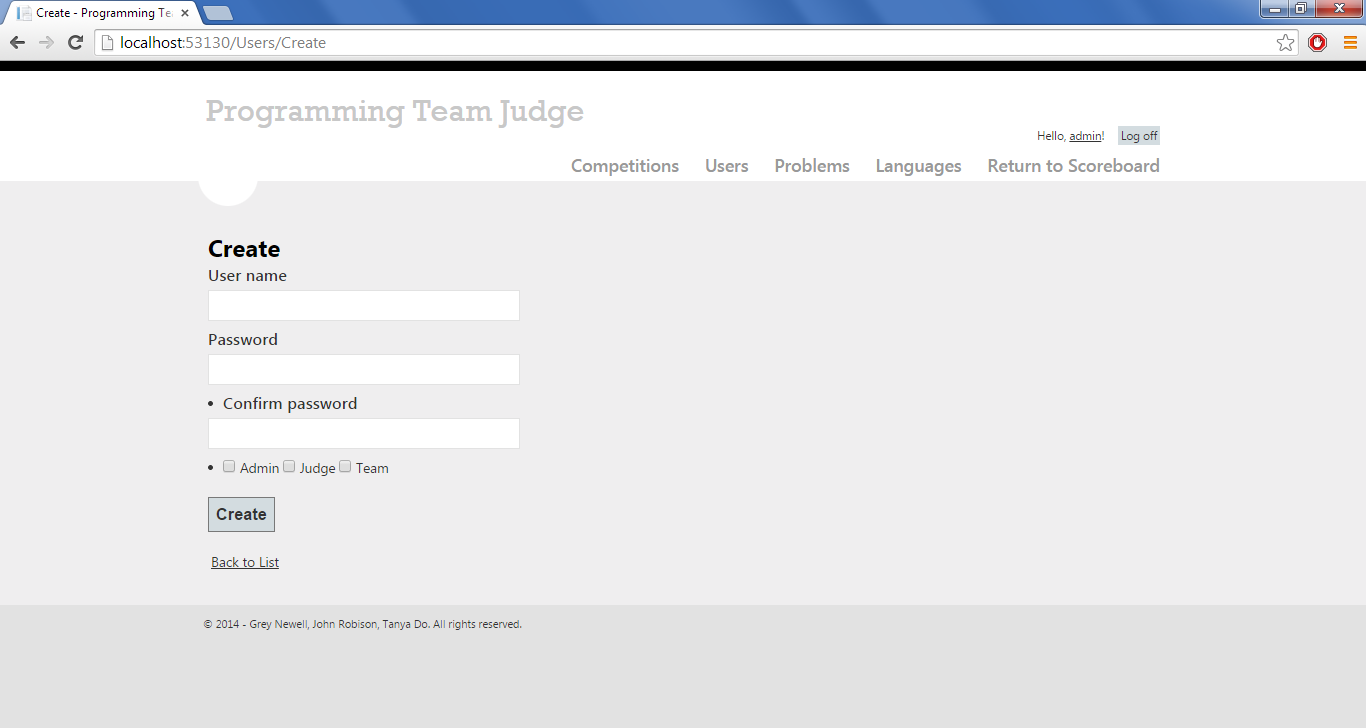


Figure : User Creation Page

After developing the system for an Admin to create User Profiles, our team began developing the system for an Admin to manage these User Profiles. There needed to be an intuitive and simple way for an Admin to quickly and easily view all of the User Profiles as well as their roles in the system. To do this, our team created a table that would display the Role Based Access Matrix in a visual manor such that they could view either a single user or a single role at the same time. Next to each User Account in this table, our team provided buttons that will allow an Admin to edit, delete, or view the details of a User Account. This visualization can be seen in Figure 6 below.

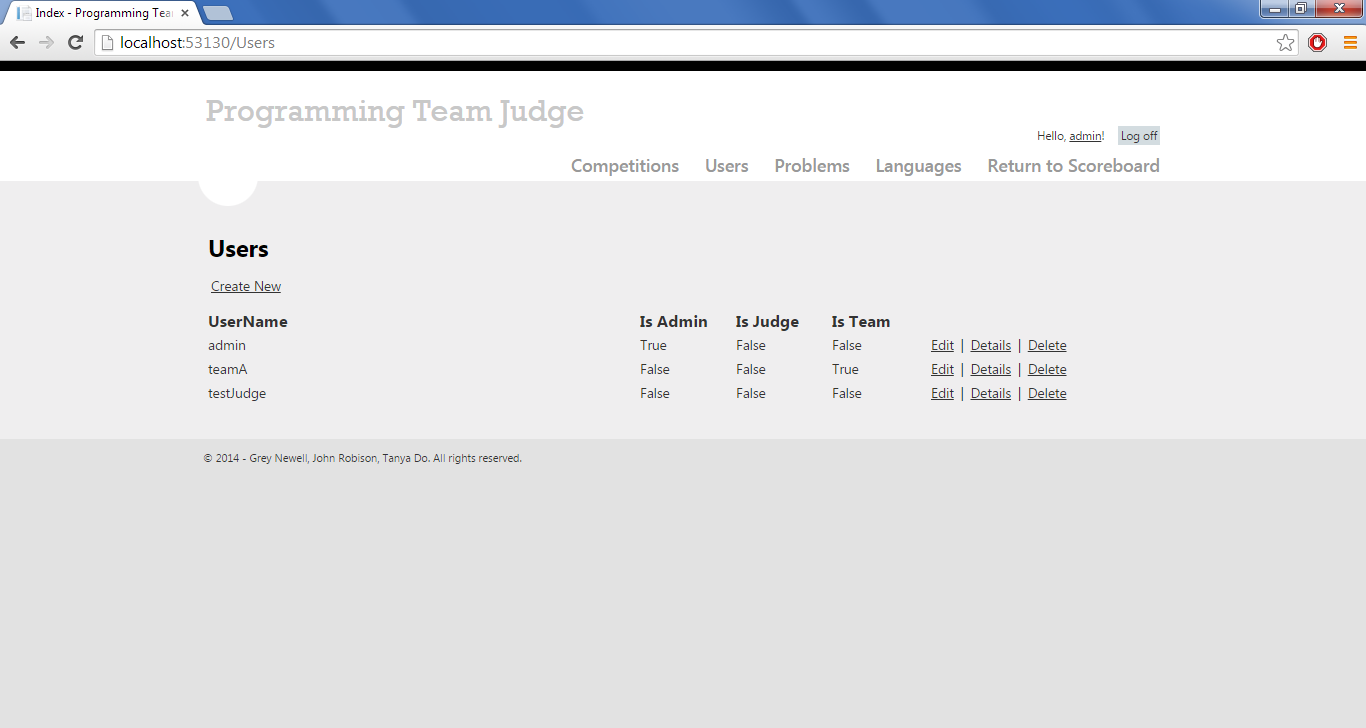


Figure : User Account Role Based Access Matrix

The options that an Admin now has to manage the User Accounts are Edit and Delete. The Details option simply displays the User Account's information, such as their available Roles. The Edit option will allow an Admin to change a user's name as well as their Roles. This page can be seen in Figure 7 below. The Delete option will first take the Admin to a screen that will ask if they are sure they want to delete the user. If the Admin is sure, they will then be able to delete the user which will return them to the User listing. This screen can be seen in Figure 8 below.

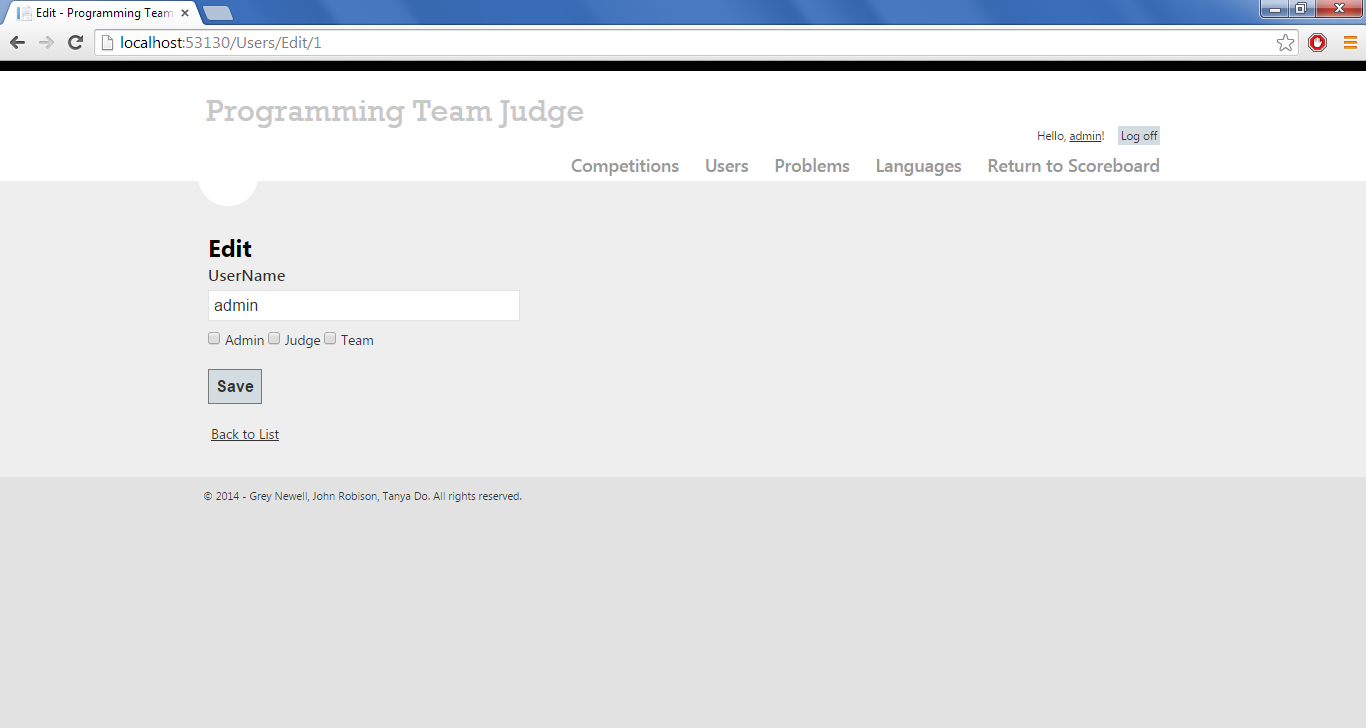


Figure : User Edit Screen

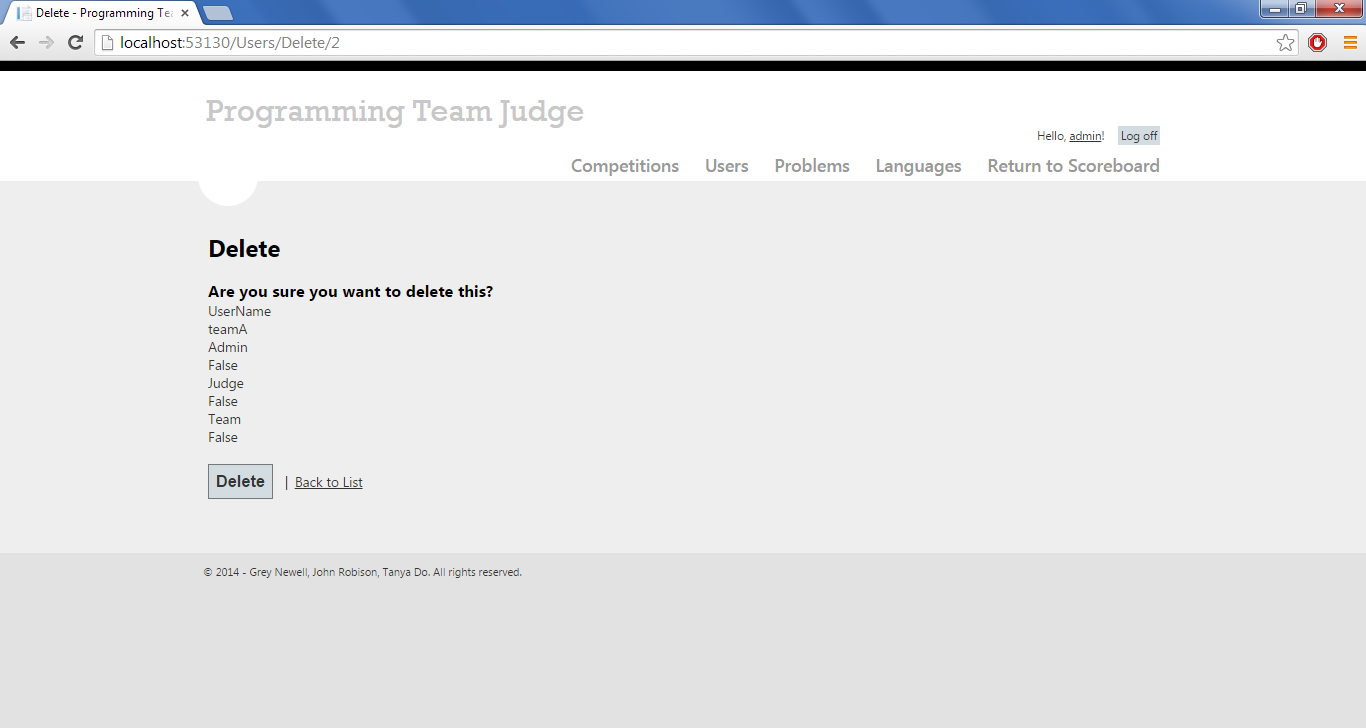


Figure : User Delete Confirmation

### Problem Creation

To prepare for the creation of a competition, all of the components that comprise a competition must be created. One of these components is the Problem. Problem objects will store the problem name, the judging input, judging output, as well as a list of submissions and the competition (or competitions) that it is linked to. As with the User Accounts, our development team wanted to ensure that Problems were easy to create and manage for Admins. We established a CRUD (Create Read Update Delete) interface for the Problems as well which included a list view similar to the one provided for the User Accounts. These Create, Read, Update, and Delete interfaces can be seen in Figure 9, Figure 10, Figure 11, and Figure 12 respectively.

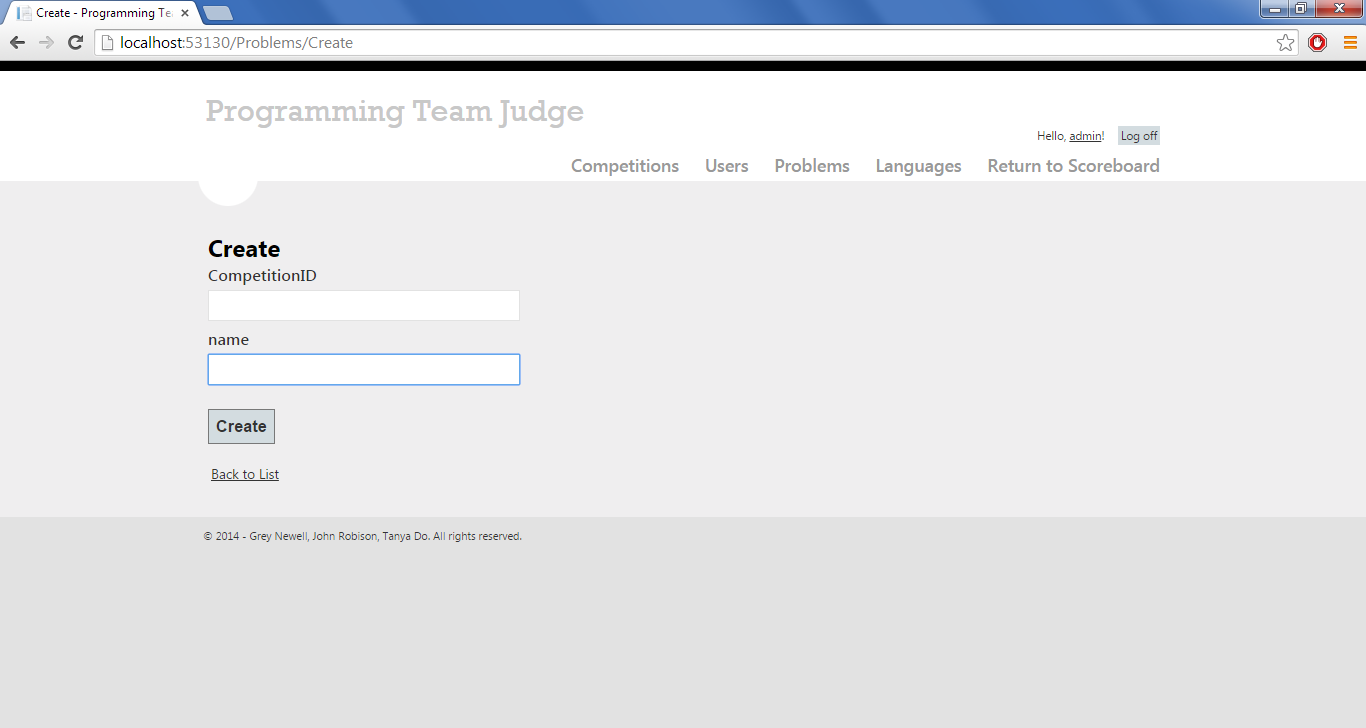


Figure : Problem Create View

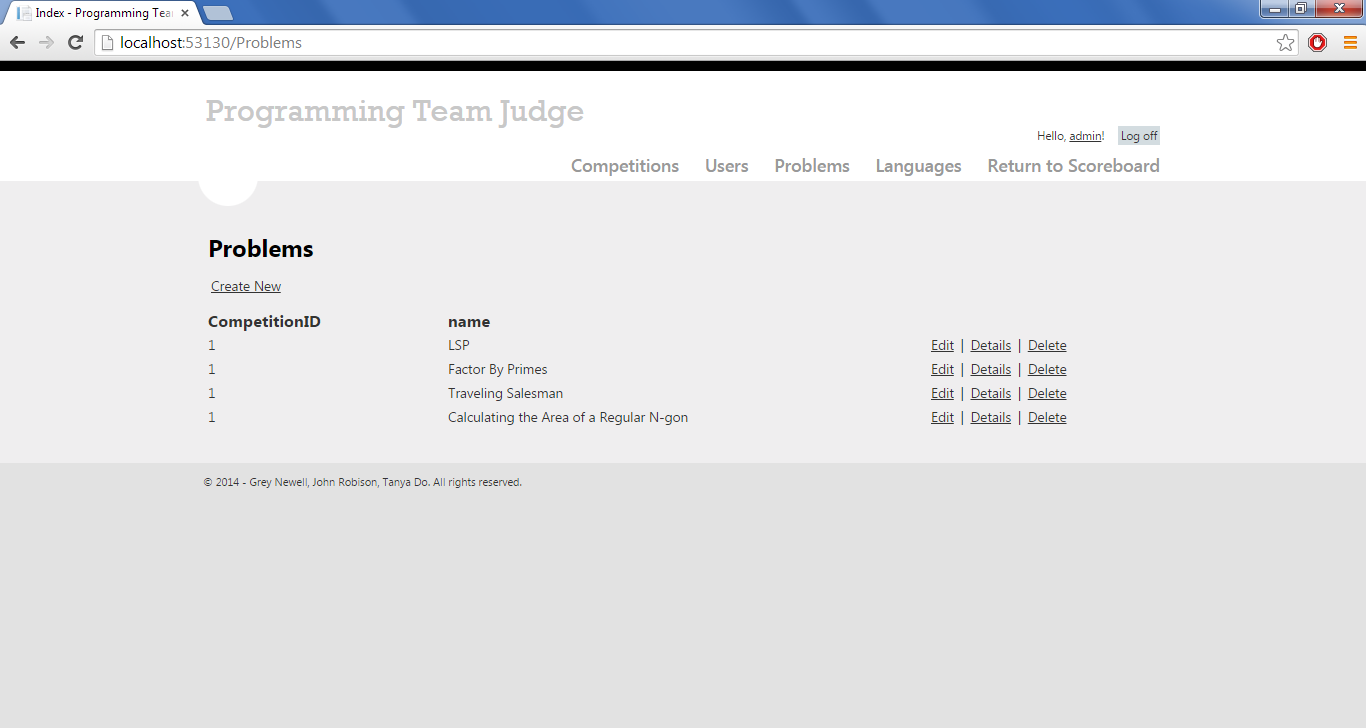


Figure : Problem Read View

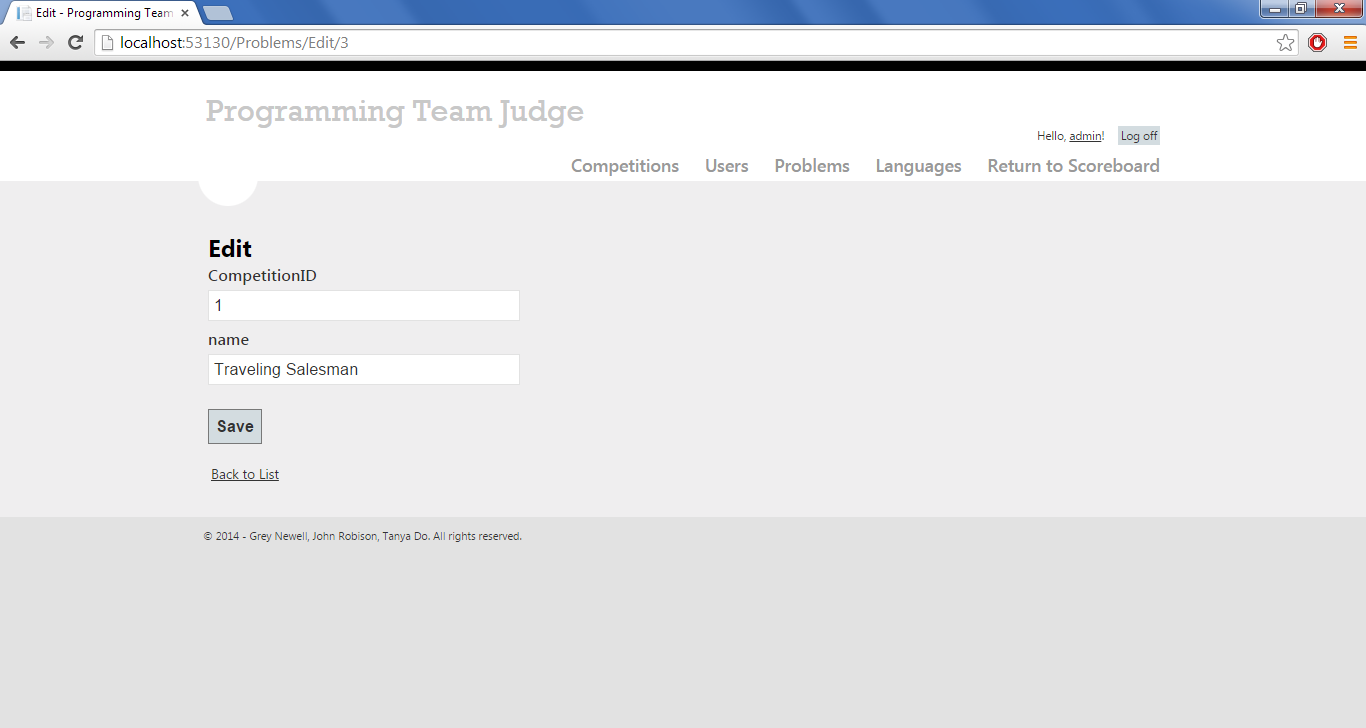


Figure : Problem Update View

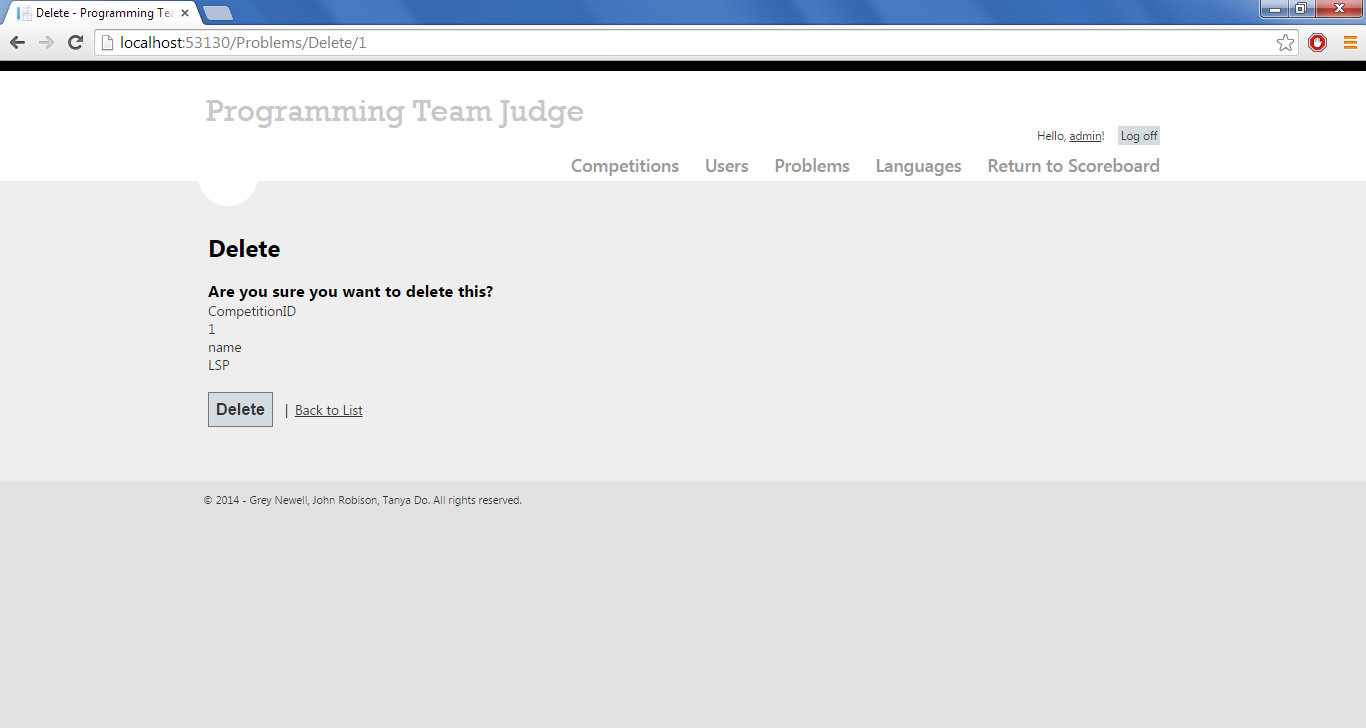


Figure : Product Delete View

### Language Creation

The final component that would comprise the Contest was the Languages that Teams were allowed to use. The use of this language filter will ensure that Teams only submit source code that the Judges will be able to use and will allow Judges to easily review submitted source code. As before, a CRUD interface was created for the Languages. The Create, Read, Update, and Delete interfaces are shown in figures Figure 13, Figure 14, Figure 15, and Figure 16 respectively.

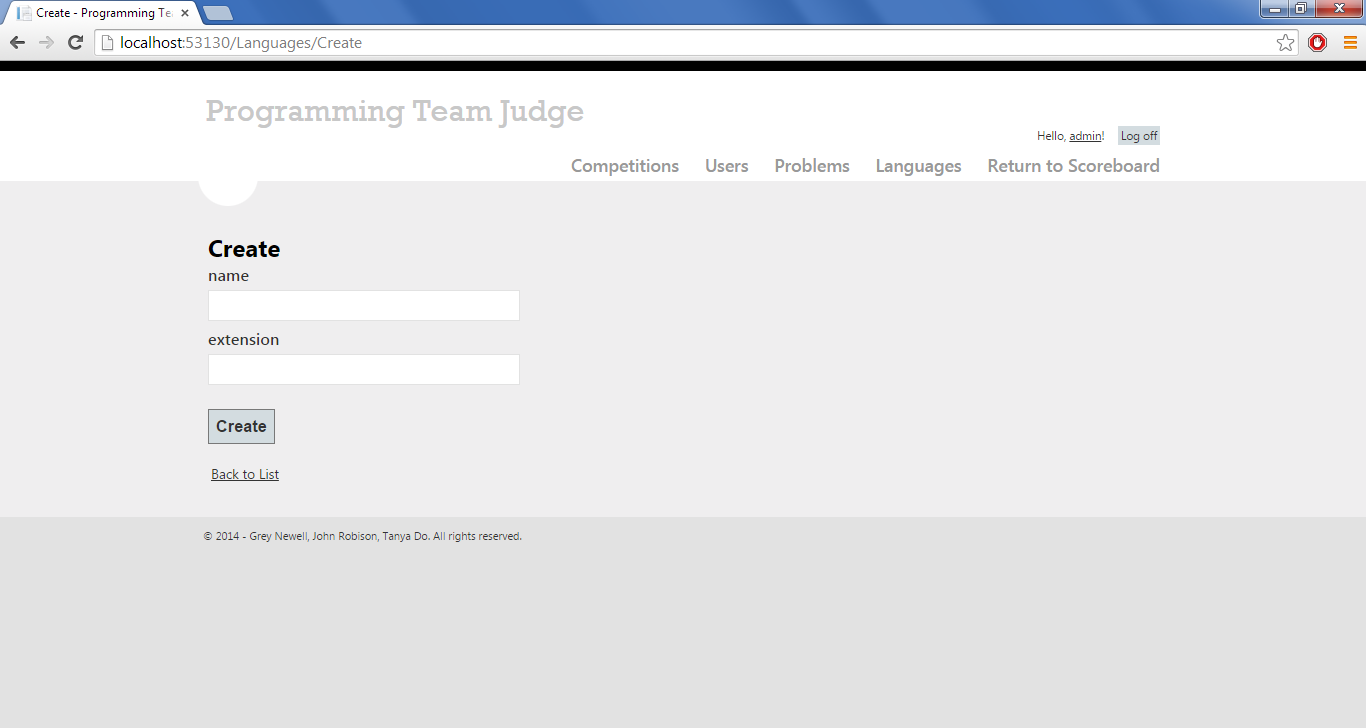


Figure : Language Create View

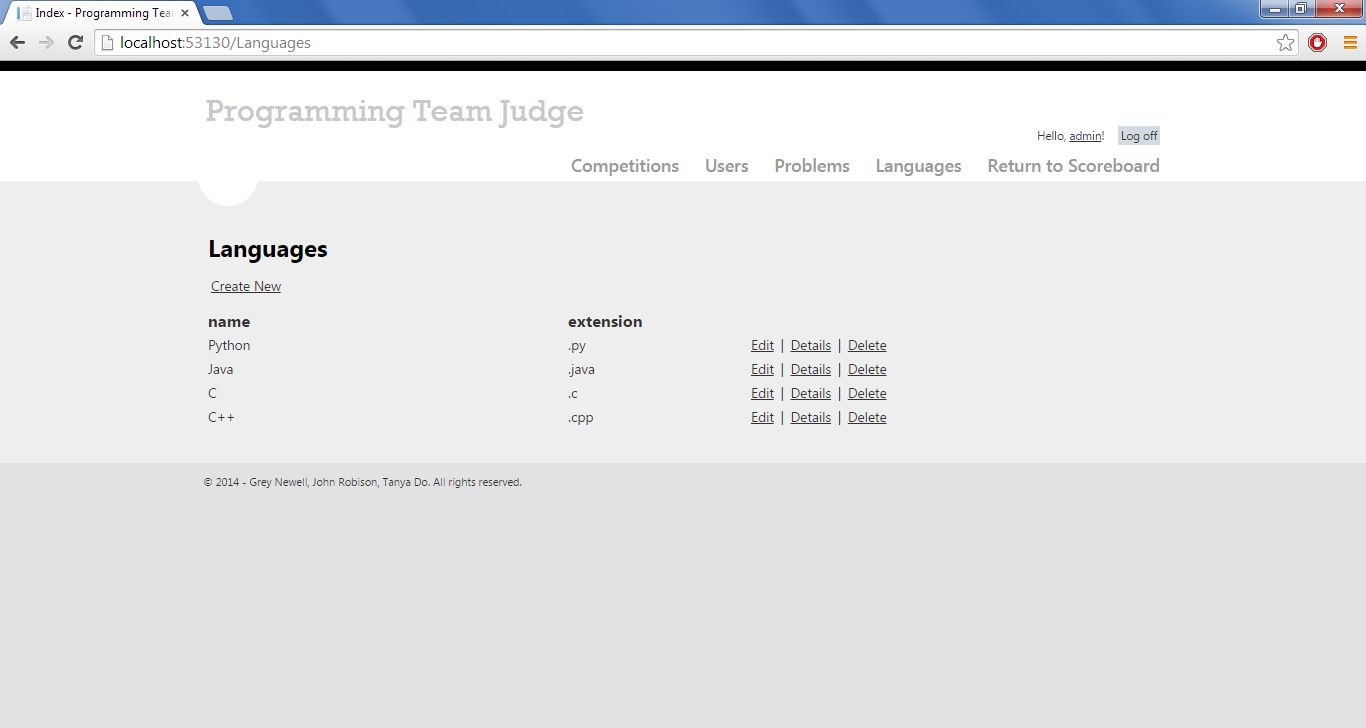


Figure : Language Read View

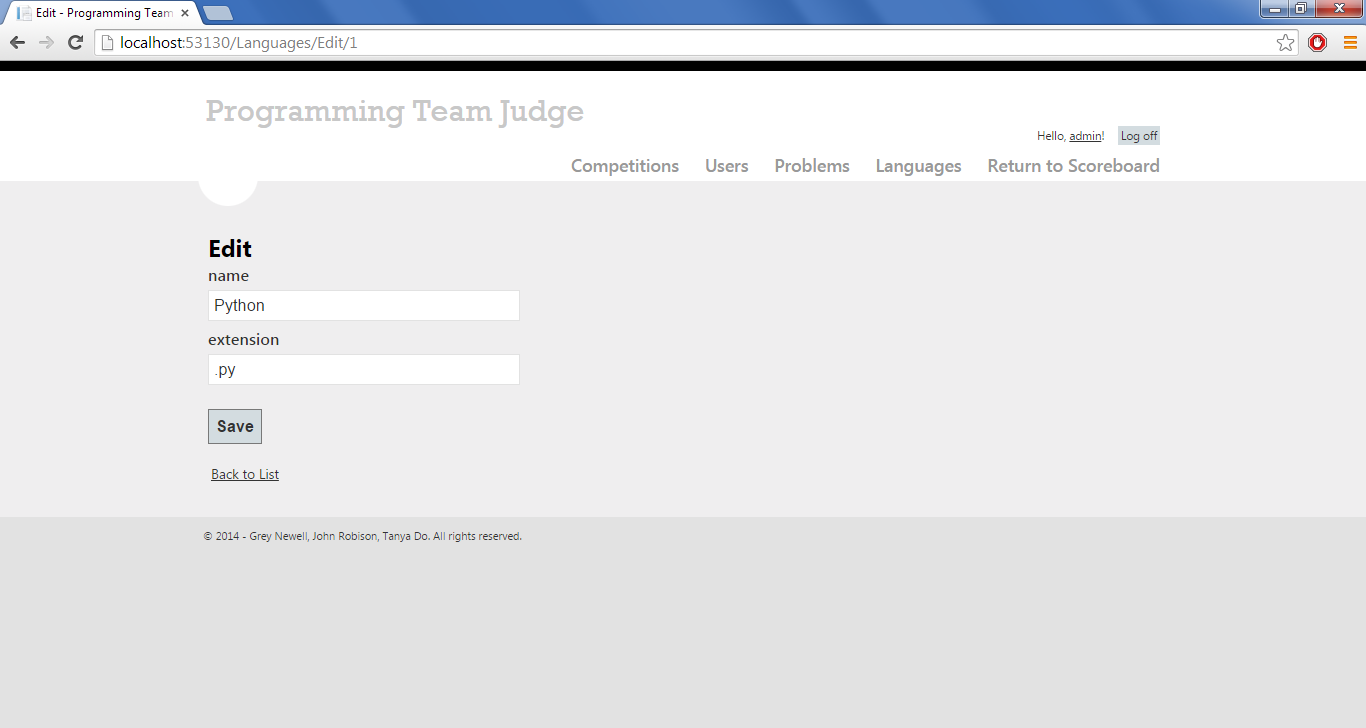


Figure : Language Update View

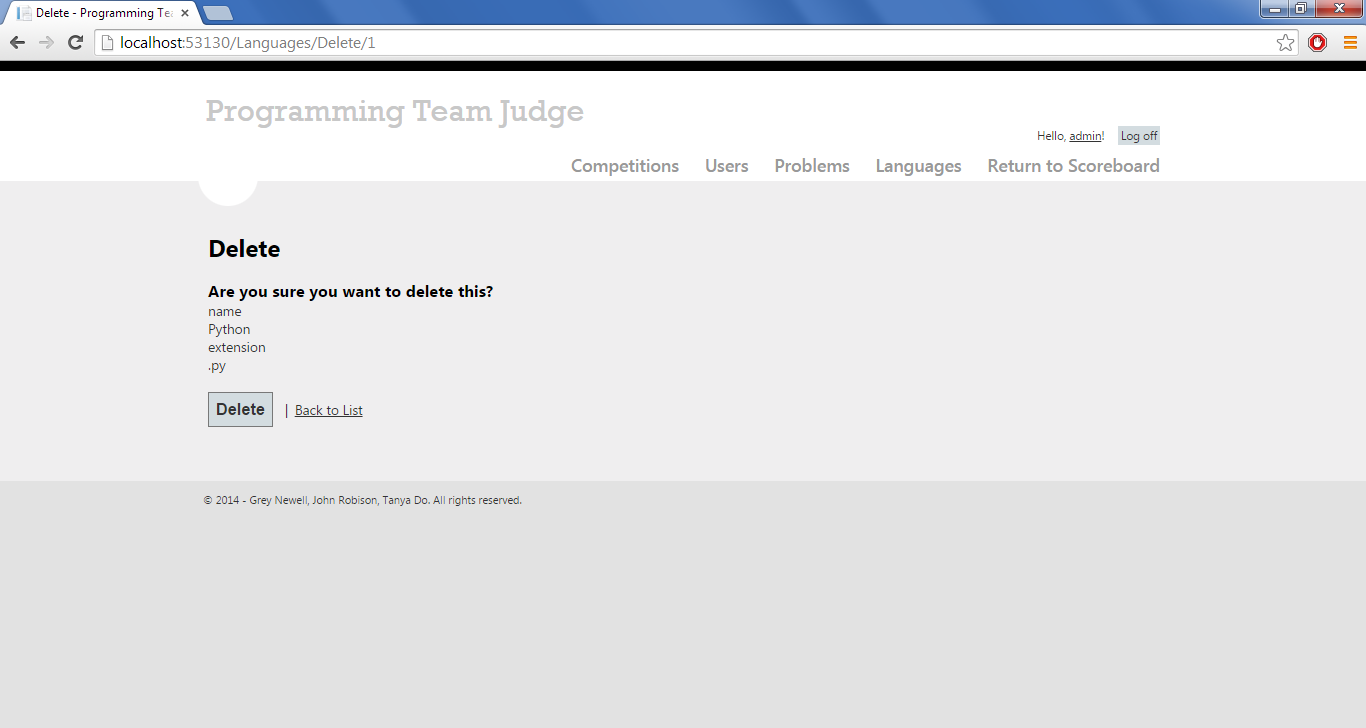


Figure : Language Delete View

### Contest Creation

After developing each component of a contest, the Contest manager can be created. The Competitions will be composed of Users and Problems as well as a list of Languages that Users can code in. The management of Competitions will include setting the start and end times of the competition, managing the users who can participate (be it as a Team or as a Judge), and setting the list of Problems and Languages that the Teams will have to solve and use.

As before, our team designed a CRUD interface for the competitions. The Create, Read, Update, and Delete interfaces can be seen in Figure 17, Figure 18, Figure 19, Figure 20.

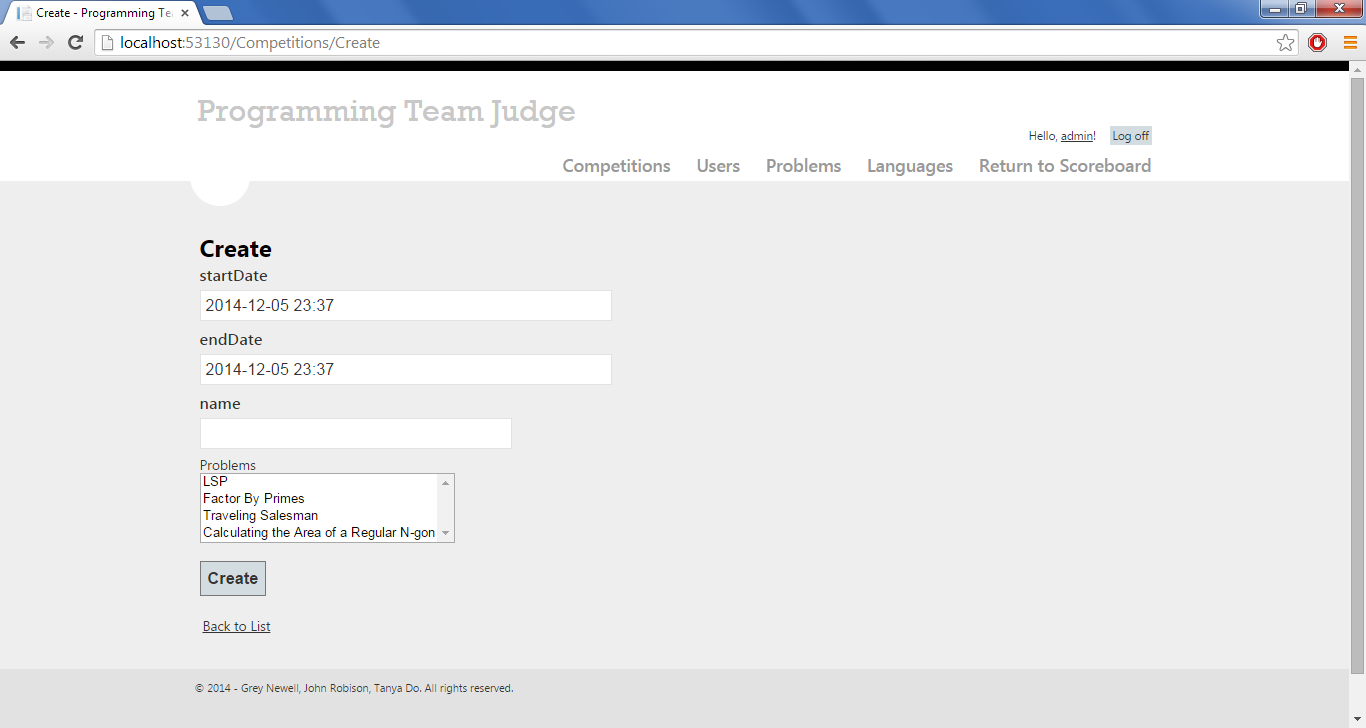


Figure : Contest Create View

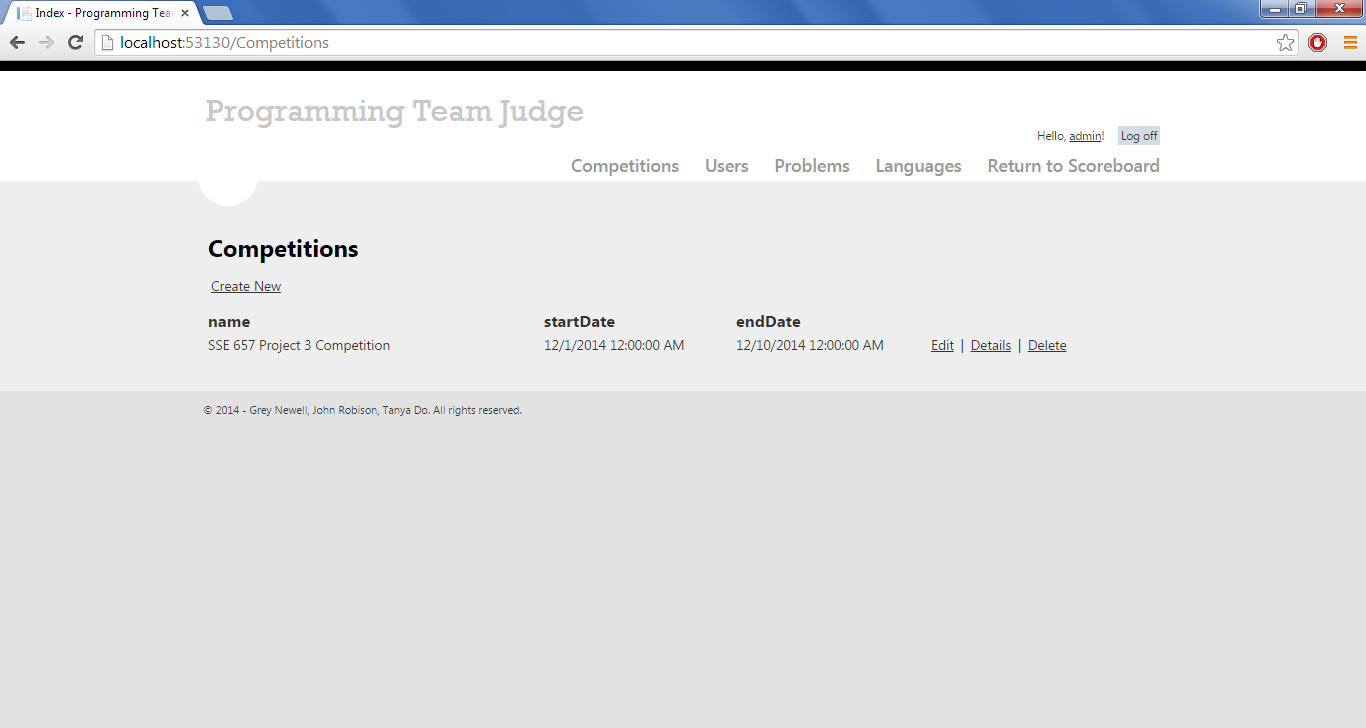


Figure : Contest Read View

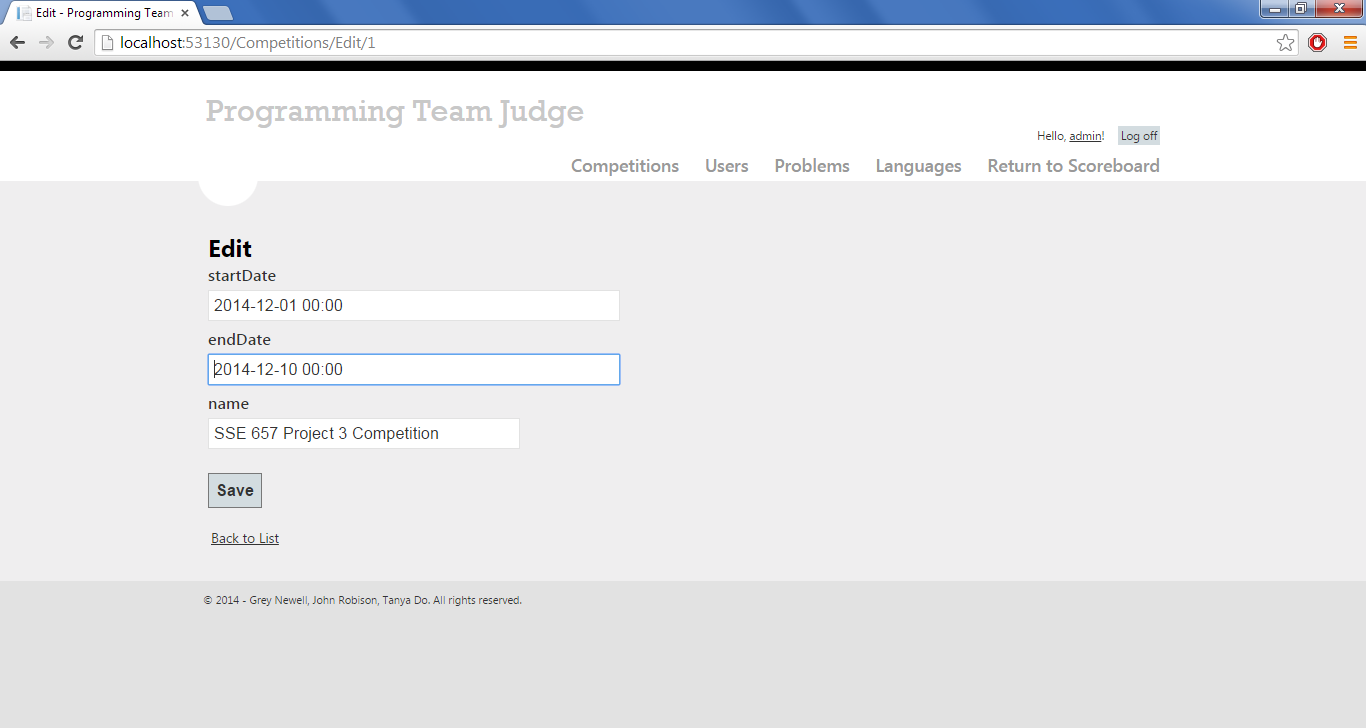


Figure : Contest Update View

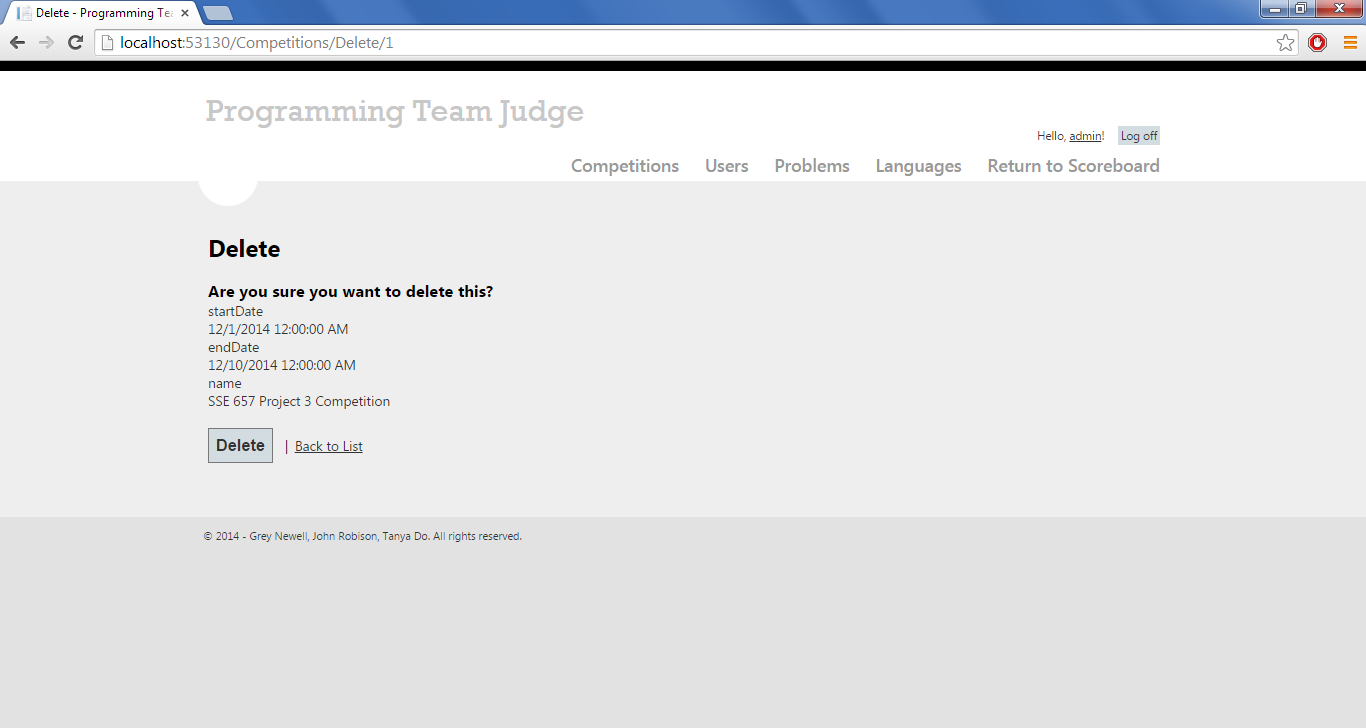


Figure : Contest Delete View

## Development of the Problem Submission Use Case

## Problem Submission

The use-case development driven is suitable for the “Problem Submission” scenario. This scenario places more importance on the experience of the user.

For the “Problem Submission” scenario, we will be simulating the path of the contest participant, Team B. We will be completing the scenario of Team B, submitting a solution to the problem. Where we start with the Team B visiting the application, and ending with Team B pressing the “submit” button.

Working as Team B, the following is the steps in which they will go through to complete a “Problem Submission” scenario. From the Judging Application’s home screen:

1. Go to the Team page
2. Choose from a drop down list, the programming language the solution is coded in
3. Choose from a drop down list, the problem to submit a solution for
4. Choose the file from the local computer, that the team wants to upload
5. Press “submit”

## Go to the Team page

In order for Team B to go to a Team page, a team page must be created. Because we are using an MVC framework, a View and a Controller must be created for this page.

The view should have a combo box for Team B to select their chosen programming language. It should have another combo box for Team B to select the problem that they are submitting a problem for. And it should have an input box in which Team B can select a file from their local machine to upload. And lastly, the page should have a submit button which sends all the previous information to application.

We created a mockup of how the Team page should look, in order to have a point of reference when programming how the View should look.

The view should look something like the mock up below:

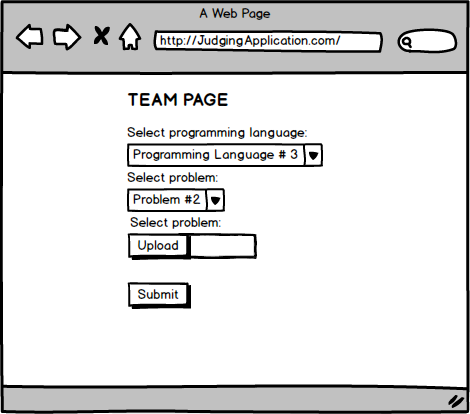


Figure Mockup of Team page

We started off a blank view and added an HTML form to get Team’s submission information and send it to the application. Instead of using traditional HTML syntax, we used the HTML helpers that ASP.NET MVC’s razor provide. The HTML form helper was sent the method that the form sends information to, the name of the controller the (previous) method resides in, and what type of Form Method this form is.

Below is the code thus far:

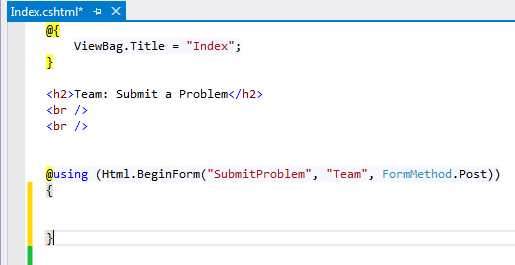


Figure Code snippet of HTML form

## Choose the Programming Language

Team B will see a combo box in which they can choose which language they will submit the solution in. Much like the mock up below:

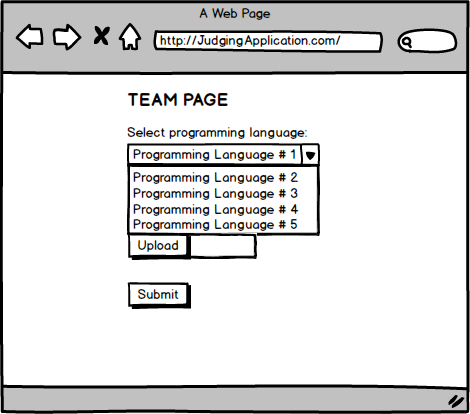


Figure Mockup of Language combo box

But in order to create a drop down list, we need the items that go into the list. Which we will define in the TeamController. So switching to the controller, we see that in order to have a drop down list we had to create a list of programming language options. We define a list of type “select list item” and added the programming languages allowed in the programming competition. When done, we pass the list of languages into a ViewBag and specified the key to be called “ProgLangs.”

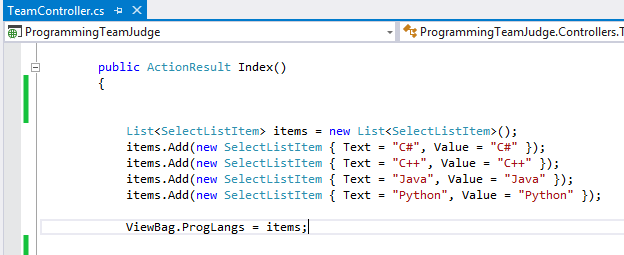


Figure Code snippet of Language list

We will now switch back to the View where we will continue to use HTML helpers to create the first drop down list that contains the available programming languages. The code below shows how we pass the list called “ProgLangs” into the drop down list HTML helper. This is possible because earlier we placed the list into the ViewBag.

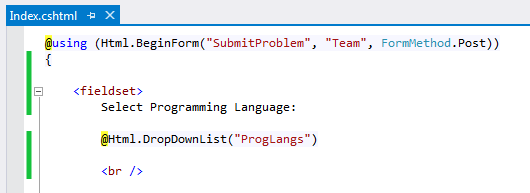


Figure Code snippet of Language drop down list

## Choose the Problem

Next, Team B will select the problem in which they are submitting a solution for. This selection will also occur via a combo box of choices.

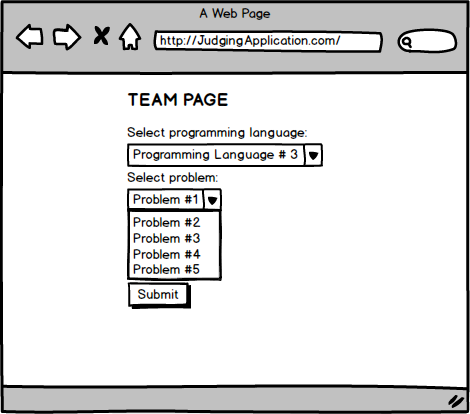


Figure Mockup of Problem combo box

For the View of the “Select problem” combo box, the html/razor code is similar to the code for the “Select programming language” combo box. However, the list of problems is not static, and will change with each competition. As such, we will be connecting the database that hosts the competition’s problem set.

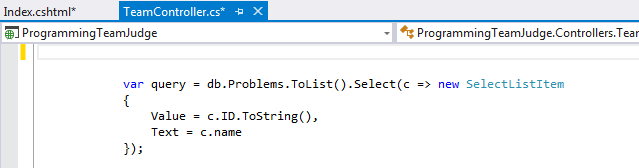
First we create a query of new “select list items” from the database of problems. And we assign the Value and Text of each “select list item.”

Figure Code snippet of database query

We then create a List of type “select list items” and then add each item in the query to the List, “problemsList.” Then, we add the “problemsList” to the ViewBag. And we label the key “ProblemsList” so we can access it in the View.

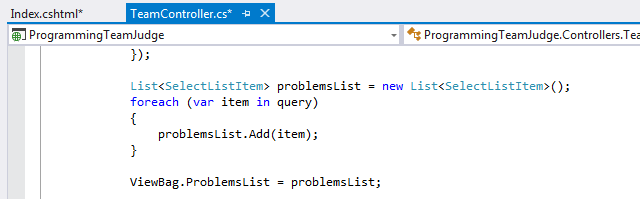


Figure Code snippet of Problems list

In the View, we create the drop down list the exact same way.

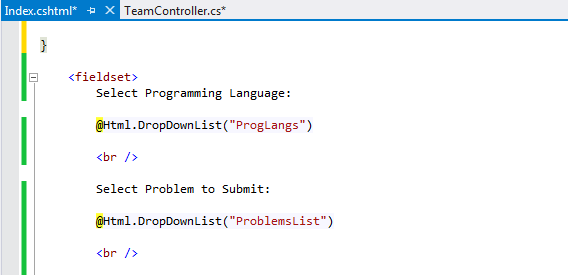


Figure Code snippet of HTML drop down list of ProblemList

## Upload solution file

Team B will now need to select the file they choose to submit. The team will be able to click the upload button, which will open local machine’s file explorer. Where Team B can now choose the file to upload.

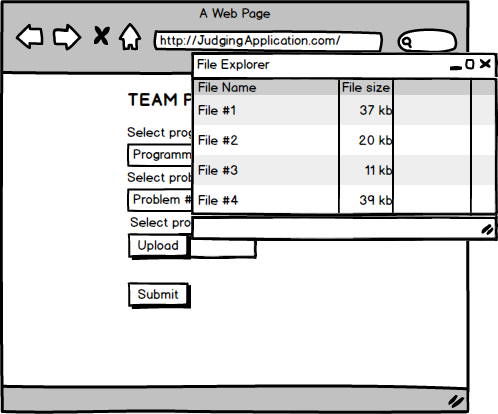


Figure Mockup of File Explorer

In the View, we create the upload box with the input HTML tag. We specify the type, “file”, and then name it. We do not to visit the TeamController this time.

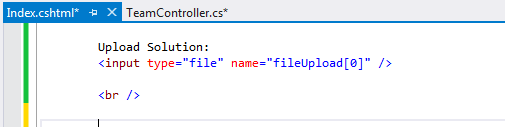


Figure Code snippet of the file input

## Press Submit

When Team B is finished selecting the appropriate values for each field, they will now want to send it to the Judges. This is done by clicking the “submit” button.

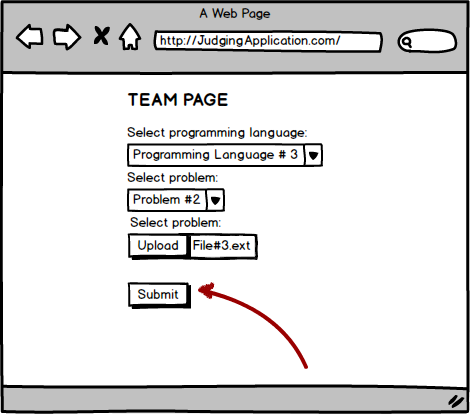


Figure Mockup of the submit button

We create the “submit” button with submit functionality, with the use of the input HTML tag again. However, this time, we will specify the type as “submit.”

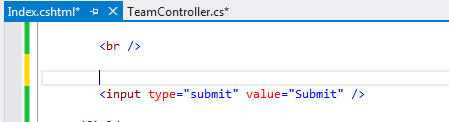


Figure Code snippet of submit input

Upon hitting on the “submit” button, all of the information in the fields will be sent to the “SubmitProblem” method in the TeamController page. The fields are passed in as parameters and sent to the Submission Model, which then sent to the database.

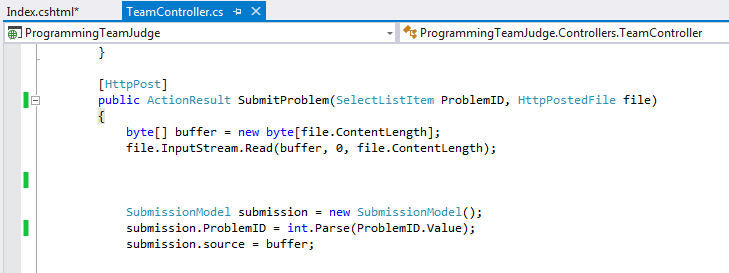


Figure Code snippet of the POST method

## Development of the Problem Judging Use Case

The Problem Judging use case involves all functionality which has to do with viewing a competition’s problems and submissions, claiming a submission for judging, delivering the Judge’s Packet, and receiving the scoring of the submission. Development proceeded in a largely view-driven manner after a significant period of setup work to instantiate the necessary infrastructure and data to facilitate the use case. Subsequently development proceeded in order through the views a user would see and interact with in the process of judging a problem submission. Throughout this section there will be references to the (HYPERLINK)Problem Judging Use Case, as this was the notional itinerary I used when progressing through the four primary phases of developing this use case.

### Setup Work

The first requisite task was development of the Model, Controller, and View for Competitions. These components stored the relevant information concerning a competition, provided programmatic access to it, and established a template by which user-readable output could be generated from the database, respectively. In addition, I chose to implement CRUD functionality (Create, Read, Update, and Delete) within the controller and views. This allowed me to get or set values of any database record during development, which became important later. Once this was accomplished, I wrote a database initializer within the CompetitionModel to fill the database with properly formatted Competitions, Teams, Problems, and a few sample submissions. Next, a JudgeController was defined to handle requests relevant only to judging problem submissions (Use Case Step #1). In order to make requests to the Judge controller, the User must be in the role of “Judge” (Use Case Step #2).This authentication barrier conceals Judging functions from unauthorized users, preventing teams or admins from maliciously altering competition standings. This controller is accessible via a RESTful API, as any HttpRequest URIs beginning with the root URI followed by the string “/Judge” are directed to it. Requests by default are directed to the Index() view, including when Judge is clicked from the navigation bar (Use Case Step #3).

### Index View

The first view served from the JudgeController is the Index view. This view is responsible for querying the database for all Competition records and programmatically generating a view of these records, ordered chronologically by end date. This view also enables the Judge user to select a competition to view the problems and submissions contained within (Use Case Step #4). Since the Judge cannot edit or create competitions, defining a custom view **encapsulates** the unnecessary and potentially dangerous functionalities away from the user. Below in Figure 35 can be seen the final Judge Index view, populated with an example competition.

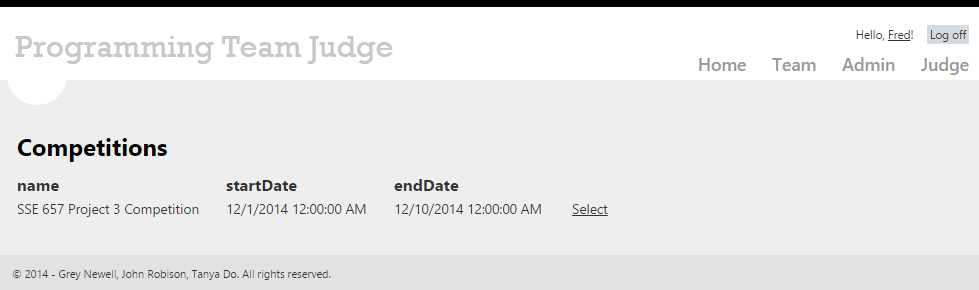


Figure : Judge Index View

### Competition View

Once the user selects a competition from the Index View, the next view which needed developed was the Competition View. This view is responsible for displaying all problems within an individual competition, and every current submission for every problem. Unscored submissions must display appropriate values for the columns ‘Judging Outcome,’ ‘Score,’ and ‘Judge,’ like Unknown, 0, and blank, respectively. This view also needed to provide an interactive element by which judges could claim submissions for evaluation (Use Case Step #5). Controller and View methods involved in creating this view were designed with OOAD principles in mind. In particular, **loose coupling** is enforced because the controller is responsible for fetching the relevant data and passing it to the view; if something changes about the structure of data storage, nothing needs to change about the view at all. Likewise, if the view needs to change, the controller is unharmed. Secondly, the Model-View-Controller pattern applied to this view allowed the team as designers to **encapsulate what varies**. Any competition’s problems and submissions may be retrieved and displayed by *exactly the same code* by simply passing the competition’s ID value to the controller-view, thanks to our object oriented paradigm. Below in Figure 6 can be seen the finalized view.

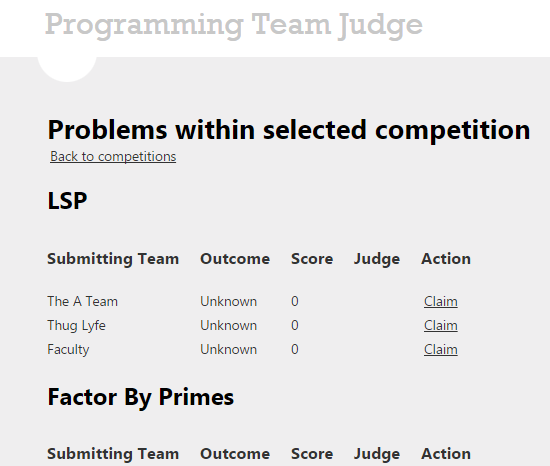


Figure : Judge Competition View

### Submission Judging View

The final view served by the Judge controller, and concurrently the final view in the Problem Judging use case, is the Submission Judging View. Portions of the competition view were reused to display the information stored within a Submission database record- and example of smooth **code reuse** enabled by the MVC design pattern. In addition to this, I added an html form to accept the Judge user’s evaluation and update the submission record accordingly. Before the judge can properly evaluate a submission, he must download a .zip archive containing the team’s source code, sample input, and expected output. To facilitate this, another controller method was written, designed to pull the appropriate byte arrays from problem (input and output) and submission (source). These byte arrays were then programmatically added to a ZipFile object, which is dynamically generated when the judge clicks download. Dynamic Generation allows the Judge to receive only the most up-to-date copy of a team’s source and prevents storing of the .zip archive server-side. The resulting object was written to the Http response’s outputstream, and the header updated appropriately to notify the browser to download rather than display the file. One simple click allows the Judge to download the necessary files (Use Case Step #6). Next the judge downloads, compiles, and runs the source code using the provided input (Use Case Step #7). Next the Judge must compare either visually or with the benefit of a diff tool the output of the submitted program with the provided output (Use Case Step #8). Finally, he may select a judging outcome and a score to assign the submission (Use Case Step #9). Next he submits the form and is returned to the Competition View to claim and judge another problem. CRUD functions were reused efficiently. Below in Figure 7 is the finalized Submission Judging View.

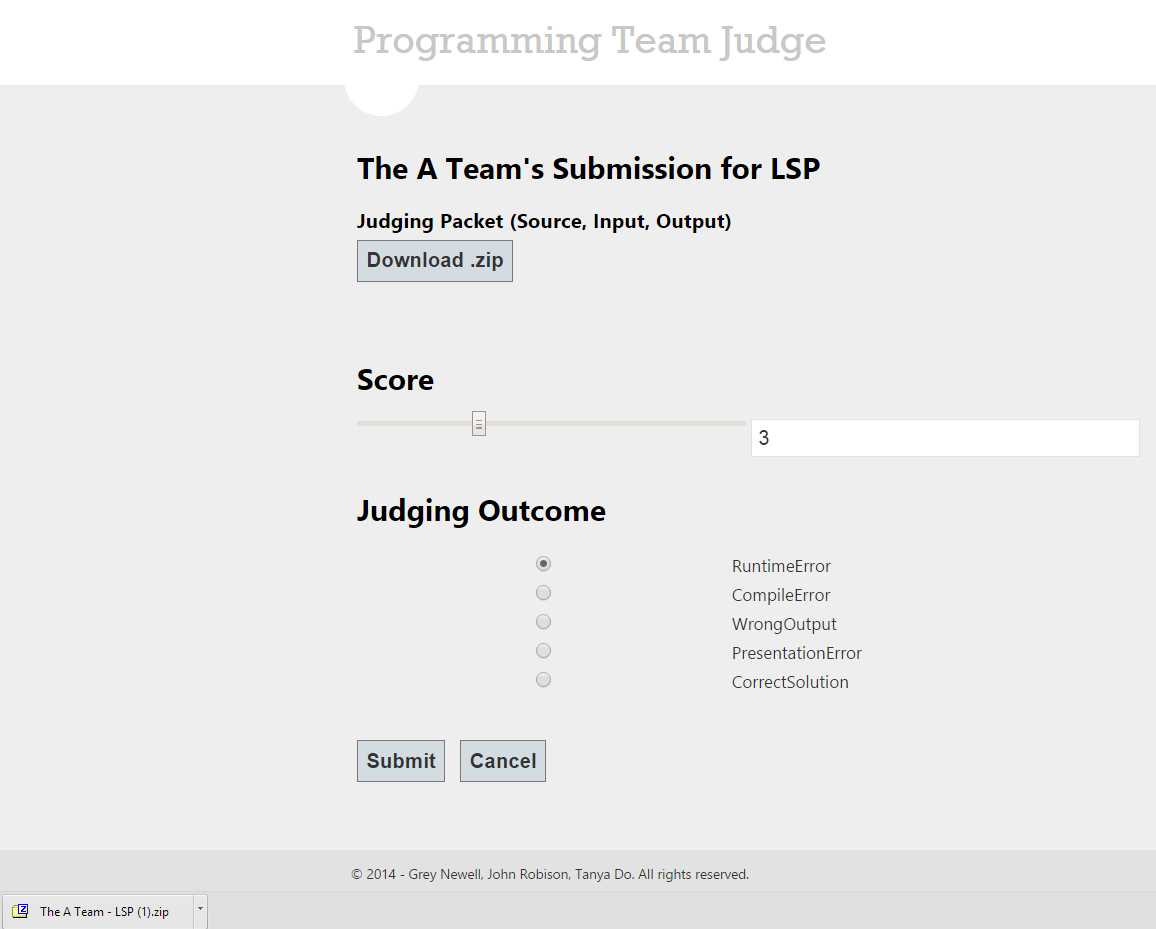


Figure : Submission Judging View

### Concepts at Work

**Use Case Driven development** is the primary concept at work within this development activity. As described before, the idea is to begin with the big picture in mind. Once a use case has been targeted by the developer he or she systematically progresses through it, implementing only the functionality needed to successfully perform the use case- although it’s well worth additional time investment to insure that implementation is done so in a robust yet flexible manner. One tool nearly all OOAD professionals employ is automated testing, a facet of **Test Driven Development**. By writing and religiously running a suite of both automated and manual tests, the developer can ensure that every corner of requisite functionality is implemented. Furthermore, any past functionality which has been broken by new changes is revealed! Lastly, tests (particularly user acceptance tests) put the developer in the client’s shoes, allowing him to focus on how his product will actually be used and what it needs to do rather than technical details which remain (and shall remain) unimportant to the customer. **Flexibility in design** was demonstrated within the development work simply due to the amount of code I was able to reuse in varying situations, thanks both to the application of the **existing Model View Controller design pattern** and to **variability analysis**. Although this problem should not be considered large in a classification sense, it’s one of the largest OOAD challenges I’ve ever tackled, and the authors of *Headfirst!...* Gave me the tools I needed to carry the project through to favorable completion.

# Test Results and Quality Analysis

# Conclusion

In the course of conducting this project, the team displayed a knowledge of object oriented analysis and design by thoroughly executing an iteration of the OOAD lifecycle. Beginning with client interviews, the team conducted Commonality vs. Variability analysis of the anticipated product compared to other solutions to the same or a similar problem. Next, concrete lists of features and requirements of the solution were derived from the client interviews and CvsV analysis. Using the features and requirements garnered from the previous steps, the team developed and diagrammed use cases which described how users would interact with the system. The use case diagrams suggested how the internal workings of the system might be broken into components and subsystems, and therefore subsequently arranged. This implied structure was refined and codified during our Architectural Analysis, where the team applied The Three Q’s of Architecture to determine how each problem should be addressed. At this point, the team found it necessary to consult the client for verification and additional information, which included sharing with the client all analysis and design work products. The client approved of our design, and we moved forward by defining Test Cases which the system must satisfy. After the test cases were written and approved, the team began distributed parallel development of our web service using different development methodologies. Once development was considered feature-complete a final testing phase was conducted to ensure all unit and integration tests passed, all use cases were satisfied, and user acceptance testing was valid. After a brief quality review, the system was complete!

The way our project was conducted demonstrates the application of a mature understanding of the software development lifecycle as it applies to an object oriented system. In the end, we faced several project Challenges, established Goals for the Next Iteration, and left the project with several takeaways that improved the team as both object oriented designers and developers.

### Challenges

One of the biggest challenges the team faced during development resulted from the **distributed team development environment**. While working on the section of the project we were individually responsible for, often it became necessary to interact with someone else’s portion of the system- which may or may not exist yet. Furthermore, when designing our own components we made decisions which heavily impacted the way other components should be built or the way that ours would be interacted with. To overcome this challenge, the team employed the techniques of Scaffolding, Programming by Contract, and Communication.

* **Scaffolding** is a technique developers can employ thanks to encapsulation. In a situation where the functionality being written or tested relies on others which have not yet been written or tested, developers can construct Scaffolding, or temporary boilerplate pieces of code which return the appropriate values the constructed system would. This allows developers to focus on their own area of expertise, as well as increasing separation between Unit and Integration tests. A unit test generally employs scaffolding to isolate only the functionality developed within that Unit, while an integration test will utilize the component itself. Scaffolding was almost totally replaced as development proceeded.
* **Programming by Contract is the practice of defining a standard for the way you expect users to interact with your code. Even if Tanya, for example, didn’t know how the Problem Judging subsystem would specifically use her Problem Submission subsystem, she had the ability to define a Contract by which all interaction took place. This meant that to use her code, you had to follow the standards for input/output, organization of code, and separation of responsibility in the way she had already defined it. This technique is useful because it allows a developer to assume objects, methods, and data within her system are populated in the way that she would expect them to be. For example, if Tanya programmed by the contract that no null variables or objects would be passed, she is allowed to assume everywhere in her system that nothing is null, saving valuable time checking input and exceptions.**
* **Communication** is perhaps the simplest and yet most important tool for developers to overcome the challenges associated with the distributed development environment. Developers utilized dynamic communication methods (like texts and phone calls) to resolve simple factual questions that might arise, such as “How are problem submissions stored in the database?” In conjunction with this, static communication methods (like a github repository) were established to communicate information such as standards, code, and responsibilities which would be hard to express otherwise. Whenever information was needed about a specific facet of the project, developers could access this information using either tool.

### Goals for the Next Iteration

The scope of the project covered in this report has been completed, but the product itself has further goals established by the developers and the client.

* **Communication between Teams and Judges:** The client was overjoyed to receive our completed system, but almost immediately turned around and requested this feature. They felt that while in their current use case (localized programming competitions) it was very simple to communicate between Judges and Teams, what if they decided to host an online competition? A facilitated communication method within our web app opens up many possibilities for distributed competitions, and also improves usability of the app as a whole.
* **Content Filtering:** In light of the new distributed use case the client would like implemented, some type of content filtering must be implemented in order to prevent inappropriate team names, malicious code submissions, and fraudulent entries. While in the current use case it is up to the host of the competition and his Admins to protect against these types of threats, in a distributed environment our framework will need to handle some of these functionalities for the Admin, and provide tools to assist in the others.
* **Automated build/run of submissions:** Currently, judges manually download and compile solution candidates. The judging process would be quicker and better if our application built the submissions automatically and ran them with the sample input given, allowing judges to spend all of their time examining the source code and output.

When the next design iteration begins, the team will embark on a similar process as the one this report describes. After the appropriate design and analysis has conducted, development will proceed, and hopefully the customer will be just as pleased with the next iteration as this one.

### Takeaways

As this project was meant to demonstrate a holistic understanding of the concepts presented within the text as well as how to apply them, the team came up with several Takeaways. These are bits of knowledge we gained from executing the project which either were not shared in the text, were at first counterintuitive, or are revelations personal in nature.

* **Existing design patterns are more useful the more skill and practice you have in applying them.** The original members of the team had already worked extensively with the Model-View-Controller pattern and its ASP.NET implementation in the previous project. The newest member of the team shared only a passing familiarity with the concepts of MVC, but was very quickly “brought up to speed” by the team. Several important ideas such as what to encapsulate where make perfect sense in theory, but specific functionality sometimes doesn’t intuitively fit into one of the three MVC components. It was only through experience that the whole team became skilled in working with our chosen design pattern.
* **Both distributed and centralized version control systems are useful; which is optimal depends on domain and preference.** Initially the team began development using Git, a distributed version control system. This allowed the team to develop quickly, utilizing feature branching to very quickly launch into their own portions of the project. Soon, development slowed to a crawl as integration errors began to occur, important overlapping functionalities began to malfunction, and system-specific errors began to occur. When the team switched to a centralized version control system, the one clean working copy was preserved all the rest of the way through development. Lesson learned? Each VCS has its area of application, and a skilled designer will be able to tell beforehand which one is likely the better choice.
* **Design is intrinsically both a predictive and reactive activity.** In the world of software development methodology, a war rages on between two schools of thought. One school of thought, Lean and Agile Methodologies, states that design should sport a purely iterative and reactive approach by identifying and conforming to customer requirement. Waterfall Methodologies on the other hand state that it is the designer’s job to anticipate these requirements and plan for them beforehand, preventing the need to adapt. Conducting this project the team has learned that in the real world, it takes a little bit of both strategies. Although a thorough design phase was conducted before the first line of code was committed, at several points in development small design changes were necessary- and not just due to customer requirements! When a particular piece of functionality doesn’t or won’t work in the way the designer had imagined, every intertwined piece must change accordingly. Thanks to our commitment to build a loosely coupled application based firmly in OOA&D principles, overall impact to the system and development time remained relatively small, and the final product delivered was more than satisfactory.

In conclusion, we hope this project has demonstrated our understanding of the material and how to use it in our development projects. It’s been a fantastic semester, and good luck to the rest of the class with the remainder of finals!

# AccountController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Transactions;

using System.Web;

using System.Web.Mvc;

using System.Web.Security;

using DotNetOpenAuth.AspNet;

using Microsoft.Web.WebPages.OAuth;

using WebMatrix.WebData;

using ProgrammingTeamJudge.Filters;

using ProgrammingTeamJudge.Models;

namespace ProgrammingTeamJudge.Controllers

{

[Authorize]

[InitializeSimpleMembership]

public class AccountController : Controller

{

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Login(LoginModel model, string returnUrl)

{

if (ModelState.IsValid && WebSecurity.Login(model.UserName, model.Password, persistCookie: model.RememberMe))

{

return RedirectToLocal(returnUrl);

}

// If we got this far, something failed, redisplay form

ModelState.AddModelError("", "The user name or password provided is incorrect.");

return View(model);

}

//

// POST: /Account/LogOff

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

WebSecurity.Logout();

return RedirectToAction("Index", "Home");

}

//

// GET: /Account/Register

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Register(RegisterModel model)

{

if (ModelState.IsValid)

{

// Attempt to register the user

try

{

WebSecurity.CreateUserAndAccount(model.UserName, model.Password);

if (!Roles.RoleExists("Admin"))

Roles.CreateRole("Admin");

if (!Roles.RoleExists("Judge"))

Roles.CreateRole("Judge");

if (!Roles.RoleExists("Team"))

Roles.CreateRole("Team");

if (model.isAdmin)

Roles.AddUserToRole(model.UserName, "Admin");

if (model.isJudge)

Roles.AddUserToRole(model.UserName, "Judge");

if (model.isTeam)

Roles.AddUserToRole(model.UserName, "Team");

WebSecurity.Login(model.UserName, model.Password);

return RedirectToAction("Index", "Home");

}

catch (MembershipCreateUserException e)

{

ModelState.AddModelError("", ErrorCodeToString(e.StatusCode));

}

}

// If we got this far, something failed, redisplay form

return View(model);

}

//

// POST: /Account/Disassociate

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Disassociate(string provider, string providerUserId)

{

string ownerAccount = OAuthWebSecurity.GetUserName(provider, providerUserId);

ManageMessageId? message = null;

// Only disassociate the account if the currently logged in user is the owner

if (ownerAccount == User.Identity.Name)

{

// Use a transaction to prevent the user from deleting their last login credential

using (var scope = new TransactionScope(TransactionScopeOption.Required, new TransactionOptions { IsolationLevel = IsolationLevel.Serializable }))

{

bool hasLocalAccount = OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

if (hasLocalAccount || OAuthWebSecurity.GetAccountsFromUserName(User.Identity.Name).Count > 1)

{

OAuthWebSecurity.DeleteAccount(provider, providerUserId);

scope.Complete();

message = ManageMessageId.RemoveLoginSuccess;

}

}

}

return RedirectToAction("Manage", new { Message = message });

}

//

// GET: /Account/Manage

public ActionResult Manage(ManageMessageId? message)

{

ViewBag.StatusMessage =

message == ManageMessageId.ChangePasswordSuccess ? "Your password has been changed."

: message == ManageMessageId.SetPasswordSuccess ? "Your password has been set."

: message == ManageMessageId.RemoveLoginSuccess ? "The external login was removed."

: "";

ViewBag.HasLocalPassword = OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

ViewBag.ReturnUrl = Url.Action("Manage");

return View();

}

//

// POST: /Account/Manage

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Manage(LocalPasswordModel model)

{

bool hasLocalAccount = OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

ViewBag.HasLocalPassword = hasLocalAccount;

ViewBag.ReturnUrl = Url.Action("Manage");

if (hasLocalAccount)

{

if (ModelState.IsValid)

{

// ChangePassword will throw an exception rather than return false in certain failure scenarios.

bool changePasswordSucceeded;

try

{

changePasswordSucceeded = WebSecurity.ChangePassword(User.Identity.Name, model.OldPassword, model.NewPassword);

}

catch (Exception)

{

changePasswordSucceeded = false;

}

if (changePasswordSucceeded)

{

return RedirectToAction("Manage", new { Message = ManageMessageId.ChangePasswordSuccess });

}

else

{

ModelState.AddModelError("", "The current password is incorrect or the new password is invalid.");

}

}

}

else

{

// User does not have a local password so remove any validation errors caused by a missing

// OldPassword field

ModelState state = ModelState["OldPassword"];

if (state != null)

{

state.Errors.Clear();

}

if (ModelState.IsValid)

{

try

{

WebSecurity.CreateAccount(User.Identity.Name, model.NewPassword);

return RedirectToAction("Manage", new { Message = ManageMessageId.SetPasswordSuccess });

}

catch (Exception)

{

ModelState.AddModelError("", String.Format("Unable to create local account. An account with the name \"{0}\" may already exist.", User.Identity.Name));

}

}

}

// If we got this far, something failed, redisplay form

return View(model);

}

//

// POST: /Account/ExternalLogin

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider, string returnUrl)

{

return new ExternalLoginResult(provider, Url.Action("ExternalLoginCallback", new { ReturnUrl = returnUrl }));

}

//

// GET: /Account/ExternalLoginCallback

[AllowAnonymous]

public ActionResult ExternalLoginCallback(string returnUrl)

{

AuthenticationResult result = OAuthWebSecurity.VerifyAuthentication(Url.Action("ExternalLoginCallback", new { ReturnUrl = returnUrl }));

if (!result.IsSuccessful)

{

return RedirectToAction("ExternalLoginFailure");

}

if (OAuthWebSecurity.Login(result.Provider, result.ProviderUserId, createPersistentCookie: false))

{

return RedirectToLocal(returnUrl);

}

if (User.Identity.IsAuthenticated)

{

// If the current user is logged in add the new account

OAuthWebSecurity.CreateOrUpdateAccount(result.Provider, result.ProviderUserId, User.Identity.Name);

return RedirectToLocal(returnUrl);

}

else

{

// User is new, ask for their desired membership name

string loginData = OAuthWebSecurity.SerializeProviderUserId(result.Provider, result.ProviderUserId);

ViewBag.ProviderDisplayName = OAuthWebSecurity.GetOAuthClientData(result.Provider).DisplayName;

ViewBag.ReturnUrl = returnUrl;

return View("ExternalLoginConfirmation", new RegisterExternalLoginModel { UserName = result.UserName, ExternalLoginData = loginData });

}

}

//

// POST: /Account/ExternalLoginConfirmation

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLoginConfirmation(RegisterExternalLoginModel model, string returnUrl)

{

string provider = null;

string providerUserId = null;

if (User.Identity.IsAuthenticated || !OAuthWebSecurity.TryDeserializeProviderUserId(model.ExternalLoginData, out provider, out providerUserId))

{

return RedirectToAction("Manage");

}

if (ModelState.IsValid)

{

// Insert a new user into the database

using (UsersContext db = new UsersContext())

{

UserProfile user = db.UserProfiles.FirstOrDefault(u => u.UserName.ToLower() == model.UserName.ToLower());

// Check if user already exists

if (user == null)

{

// Insert name into the profile table

db.UserProfiles.Add(new UserProfile { UserName = model.UserName });

db.SaveChanges();

OAuthWebSecurity.CreateOrUpdateAccount(provider, providerUserId, model.UserName);

OAuthWebSecurity.Login(provider, providerUserId, createPersistentCookie: false);

return RedirectToLocal(returnUrl);

}

else

{

ModelState.AddModelError("UserName", "User name already exists. Please enter a different user name.");

}

}

}

ViewBag.ProviderDisplayName = OAuthWebSecurity.GetOAuthClientData(provider).DisplayName;

ViewBag.ReturnUrl = returnUrl;

return View(model);

}

//

// GET: /Account/ExternalLoginFailure

[AllowAnonymous]

public ActionResult ExternalLoginFailure()

{

return View();

}

[AllowAnonymous]

[ChildActionOnly]

public ActionResult ExternalLoginsList(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return PartialView("\_ExternalLoginsListPartial", OAuthWebSecurity.RegisteredClientData);

}

[ChildActionOnly]

public ActionResult RemoveExternalLogins()

{

ICollection<OAuthAccount> accounts = OAuthWebSecurity.GetAccountsFromUserName(User.Identity.Name);

List<ExternalLogin> externalLogins = new List<ExternalLogin>();

foreach (OAuthAccount account in accounts)

{

AuthenticationClientData clientData = OAuthWebSecurity.GetOAuthClientData(account.Provider);

externalLogins.Add(new ExternalLogin

{

Provider = account.Provider,

ProviderDisplayName = clientData.DisplayName,

ProviderUserId = account.ProviderUserId,

});

}

ViewBag.ShowRemoveButton = externalLogins.Count > 1 || OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

return PartialView("\_RemoveExternalLoginsPartial", externalLogins);

}

#region Helpers

private ActionResult RedirectToLocal(string returnUrl)

{

if (Url.IsLocalUrl(returnUrl))

{

return Redirect(returnUrl);

}

else

{

return RedirectToAction("Index", "Home");

}

}

public enum ManageMessageId

{

ChangePasswordSuccess,

SetPasswordSuccess,

RemoveLoginSuccess,

}

internal class ExternalLoginResult : ActionResult

{

public ExternalLoginResult(string provider, string returnUrl)

{

Provider = provider;

ReturnUrl = returnUrl;

}

public string Provider { get; private set; }

public string ReturnUrl { get; private set; }

public override void ExecuteResult(ControllerContext context)

{

OAuthWebSecurity.RequestAuthentication(Provider, ReturnUrl);

}

}

private static string ErrorCodeToString(MembershipCreateStatus createStatus)

{

// See http://go.microsoft.com/fwlink/?LinkID=177550 for

// a full list of status codes.

switch (createStatus)

{

case MembershipCreateStatus.DuplicateUserName:

return "User name already exists. Please enter a different user name.";

case MembershipCreateStatus.DuplicateEmail:

return "A user name for that e-mail address already exists. Please enter a different e-mail address.";

case MembershipCreateStatus.InvalidPassword:

return "The password provided is invalid. Please enter a valid password value.";

case MembershipCreateStatus.InvalidEmail:

return "The e-mail address provided is invalid. Please check the value and try again.";

case MembershipCreateStatus.InvalidAnswer:

return "The password retrieval answer provided is invalid. Please check the value and try again.";

case MembershipCreateStatus.InvalidQuestion:

return "The password retrieval question provided is invalid. Please check the value and try again.";

case MembershipCreateStatus.InvalidUserName:

return "The user name provided is invalid. Please check the value and try again.";

case MembershipCreateStatus.ProviderError:

return "The authentication provider returned an error. Please verify your entry and try again. If the problem persists, please contact your system administrator.";

case MembershipCreateStatus.UserRejected:

return "The user creation request has been canceled. Please verify your entry and try again. If the problem persists, please contact your system administrator.";

default:

return "An unknown error occurred. Please verify your entry and try again. If the problem persists, please contact your system administrator.";

}

}

#endregion

}

}

## AdminController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ProgrammingTeamJudge.Controllers

{

public class AdminController : Controller

{

//

// GET: /Admin/

public ActionResult Index()

{

ViewBag.Message = "Admins log in here to add competitions, languages, problems, and users.";

//if (User.IsInRole("Admin"))

{

return View();

}

//else return View("PermissionsError");

}

//

// GET: /Admin/Users

public ActionResult Users()

{

ViewBag.Message = "Add or Manage Users";

//if (User.IsInRole("Admin"))

{

var model = new List<Models.UserProfile>();

return View(model);

}

//else return View("PermissionsError");

}

//

// GET: /Admin/Problems

public ActionResult Problems()

{

ViewBag.Message = "Add or Manage Problems";

//if (User.IsInRole("Admin"))

{

var model = new List<Models.ProblemModel>();

return View(model);

}

//else return View("PermissionsError");

}

//

// GET: /Admin/Contests

public ActionResult Competitions()

{

ViewBag.Message = "Add or Manage Competitions";

if (User.IsInRole("Admin"))

{

var model = new List<Models.CompetitionModel>();

return View(model);

}

else return View("PermissionsError");

}

//

// GET: /Admin/Languages

public ActionResult Languages()

{

ViewBag.Message = "Add or Manage Languages";

//if (User.IsInRole("Admin"))

{

var model = new List<Models.LanguageModel>();

return View(model);

}

//else return View("PermissionsError");

}

}

}

## CompetitionsController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using ProgrammingTeamJudge.Models;

namespace ProgrammingTeamJudge.Controllers

{

public class CompetitionsController : Controller

{

private CompetitionDBContext db = new CompetitionDBContext();

//

// GET: /Competitions/

public ActionResult Index()

{

if (User.IsInRole("Admin"))

{

return View(db.Competitions.ToList());

}

else return View("PermissionsError");

}

//

// GET: /Competitions/Details/5

public ActionResult Details(int id = 0)

{

if (User.IsInRole("Admin"))

{

CompetitionModel competitionmodel = db.Competitions.Find(id);

if (competitionmodel == null)

{

return HttpNotFound();

}

var problems = db.Problems.Include("Submissions").Where(p => p.CompetitionID == id);

ViewBag.problems = problems;

return View(competitionmodel);

}

else return View("PermissionsError");

}

//

// GET: /Competitions/Create

public ActionResult Create()

{

if (User.IsInRole("Admin"))

{

var query = db.Problems.ToList().Select(c => new SelectListItem

{

Value = c.ID.ToString(),

Text = c.name

});

List<SelectListItem> problemsList = new List<SelectListItem>();

foreach (var item in query)

{

problemsList.Add(item);

}

ViewBag.problemsList = problemsList;

return View();

}

else return View("PermissionsError");

}

//

// POST: /Competitions/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(CompetitionModel competitionmodel, ICollection<ProblemModel> problems)

{

if (User.IsInRole("Admin"))

{

if (ModelState.IsValid)

{

competitionmodel.problems = problems;

db.Competitions.Add(competitionmodel);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(competitionmodel);

}

else return View("PermissionsError");

}

//

// GET: /Competitions/Edit/5

public ActionResult Edit(int id = 0)

{

if (User.IsInRole("Admin"))

{

CompetitionModel competitionmodel = db.Competitions.Find(id);

if (competitionmodel == null)

{

return HttpNotFound();

}

return View(competitionmodel);

}

else return View("PermissionsError");

}

//

// POST: /Competitions/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(CompetitionModel competitionmodel)

{

if (User.IsInRole("Admin"))

{

if (ModelState.IsValid)

{

db.Entry(competitionmodel).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(competitionmodel);

}

else return View("PermissionsError");

}

//

// GET: /Competitions/Delete/5

public ActionResult Delete(int id = 0)

{

if (User.IsInRole("Admin"))

{

CompetitionModel competitionmodel = db.Competitions.Find(id);

if (competitionmodel == null)

{

return HttpNotFound();

}

return View(competitionmodel);

}

else return View("PermissionsError");

}

//

// POST: /Competitions/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

if (User.IsInRole("Admin"))

{

CompetitionModel competitionmodel = db.Competitions.Find(id);

db.Competitions.Remove(competitionmodel);

db.SaveChanges();

return RedirectToAction("Index");

}

else return View("PermissionsError");

}

protected override void Dispose(bool disposing)

{

db.Dispose();

base.Dispose(disposing);

}

}

}

# Appendix: Source Code

## HomeController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using ProgrammingTeamJudge.Filters;

namespace ProgrammingTeamJudge.Controllers

{

public class HomeController : Controller

{

[InitializeSimpleMembership]

public ActionResult Index()

{

ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";

return View();

}

public ActionResult Team()

{

ViewBag.Message = "Teams login here to submit problem solutions.";

return View();

}

public ActionResult Chart()

{

return View();

}

}

}

## JudgeController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using ProgrammingTeamJudge.Models;

using Ionic.Zip;

namespace ProgrammingTeamJudge.Controllers

{

public class JudgeController : Controller

{

private CompetitionDBContext db = new CompetitionDBContext();

//

// GET: /Judge/

public ActionResult Index()

{

ViewBag.Message = "Judges login here to view previously submitted solutions and judge problem solutions.";

if (User.IsInRole("Judge"))

{

return View(db.Competitions.ToList());

}

else return View("PermissionsError");

}

public ActionResult Competition(int id=0)

{

return View(db.Problems.Include("Submissions")

.Where(p => p.CompetitionID == id));

}

public ActionResult Claim(int submission = 0, int problem = 0)

{

ViewBag.Submission = db.Submissions.First(s => s.ID==submission);

ViewBag.Problem = db.Problems.First(p => p.ID == problem);

return View();

}

public void Download(int submission = 0, int problem = 0)

{

var Problem = db.Problems.First(p => p.ID == problem);

var Submission = db.Submissions.First(s => s.ID == submission);

ZipFile zip = new ZipFile();

zip.Name = "Judging Packet.zip";

zip.AddEntry("Source.cs", Submission.source);

zip.AddEntry("Input.txt", Problem.input);

zip.AddEntry("Output.txt", Problem.output);

Response.AddHeader("Content-Disposition", "attachment; filename=\"" + Submission.team + " - " + Problem.name + ".zip\"");

zip.Save(Response.OutputStream);

}

public ActionResult SubmitJudgement(string Judge, int Score, int SubmissionID, ProgrammingTeamJudge.Models.JudgingOutcome Outcome)

{

var Submission = db.Submissions.First(s => s.ID == SubmissionID);

Submission.judge = Judge;

Submission.score = Score;

Submission.outcome = Outcome;

db.SaveChanges();

return View(Submission);

}

}

}

## LanguagesController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using ProgrammingTeamJudge.Models;

namespace ProgrammingTeamJudge.Controllers

{

public class LanguagesController : Controller

{

private CompetitionDBContext db = new CompetitionDBContext();

//

// GET: /Language/

public ActionResult Index()

{

if (User.IsInRole("Admin"))

{

return View(db.LanguageModels.ToList());

}

else return View("PermissionsError");

}

//

// GET: /Language/Details/5

public ActionResult Details(int id = 0)

{

if (User.IsInRole("Admin"))

{

LanguageModel languagemodel = db.LanguageModels.Find(id);

if (languagemodel == null)

{

return HttpNotFound();

}

return View(languagemodel);

}

else return View("PermissionsError");

}

//

// GET: /Language/Create

public ActionResult Create()

{

if (User.IsInRole("Admin"))

{

return View();

}

else return View("PermissionsError");

}

//

// POST: /Language/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(LanguageModel languagemodel)

{

if (User.IsInRole("Admin"))

{

if (ModelState.IsValid)

{

db.LanguageModels.Add(languagemodel);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(languagemodel);

}

else return View("PermissionsError");

}

//

// GET: /Language/Edit/5

public ActionResult Edit(int id = 0)

{

if (User.IsInRole("Admin"))

{

LanguageModel languagemodel = db.LanguageModels.Find(id);

if (languagemodel == null)

{

return HttpNotFound();

}

return View(languagemodel);

}

else return View("PermissionsError");

}

//

// POST: /Language/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(LanguageModel languagemodel)

{

if (User.IsInRole("Admin"))

{

if (ModelState.IsValid)

{

db.Entry(languagemodel).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(languagemodel);

}

else return View("PermissionsError");

}

//

// GET: /Language/Delete/5

public ActionResult Delete(int id = 0)

{

if (User.IsInRole("Admin"))

{

LanguageModel languagemodel = db.LanguageModels.Find(id);

if (languagemodel == null)

{

return HttpNotFound();

}

return View(languagemodel);

}

else return View("PermissionsError");

}

//

// POST: /Language/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

if (User.IsInRole("Admin"))

{

LanguageModel languagemodel = db.LanguageModels.Find(id);

db.LanguageModels.Remove(languagemodel);

db.SaveChanges();

return RedirectToAction("Index");

}

else return View("PermissionsError");

}

protected override void Dispose(bool disposing)

{

db.Dispose();

base.Dispose(disposing);

}

}

}

## ProblemsController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using ProgrammingTeamJudge.Models;

namespace ProgrammingTeamJudge.Controllers

{

public class ProblemsController : Controller

{

private CompetitionDBContext db = new CompetitionDBContext();

//

// GET: /Problems/

public ActionResult Index()

{

if (User.IsInRole("Admin"))

{

return View(db.Problems.ToList());

}

else return View("PermissionsError");

}

//

// GET: /Problems/Details/5

public ActionResult Details(int id = 0)

{

if (User.IsInRole("Admin"))

{

ProblemModel problemmodel = db.Problems.Find(id);

if (problemmodel == null)

{

return HttpNotFound();

}

return View(problemmodel);

}

else return View("PermissionsError");

}

//

// GET: /Problems/Create

public ActionResult Create()

{

if (User.IsInRole("Admin"))

{

return View();

}

else return View("PermissionsError");

}

//

// POST: /Problems/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(ProblemModel problemmodel)

{

if (User.IsInRole("Admin"))

{

if (ModelState.IsValid)

{

db.Problems.Add(problemmodel);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(problemmodel);

}

else return View("PermissionsError");

}

//

// GET: /Problems/Edit/5

public ActionResult Edit(int id = 0)

{

if (User.IsInRole("Admin"))

{

ProblemModel problemmodel = db.Problems.Find(id);

if (problemmodel == null)

{

return HttpNotFound();

}

return View(problemmodel);

}

else return View("PermissionsError");

}

//

// POST: /Problems/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(ProblemModel problemmodel)

{

if (User.IsInRole("Admin"))

{

if (ModelState.IsValid)

{

db.Entry(problemmodel).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(problemmodel);

}

else return View("PermissionsError");

}

//

// GET: /Problems/Delete/5

public ActionResult Delete(int id = 0)

{

if (User.IsInRole("Admin"))

{

ProblemModel problemmodel = db.Problems.Find(id);

if (problemmodel == null)

{

return HttpNotFound();

}

return View(problemmodel);

}

else return View("PermissionsError");

}

//

// POST: /Problems/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

if (User.IsInRole("Admin"))

{

ProblemModel problemmodel = db.Problems.Find(id);

db.Problems.Remove(problemmodel);

db.SaveChanges();

return RedirectToAction("Index");

}

else return View("PermissionsError");

}

protected override void Dispose(bool disposing)

{

db.Dispose();

base.Dispose(disposing);

}

}

}

## TeamController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using ProgrammingTeamJudge.Models;

using System.Data;

namespace ProgrammingTeamJudge.Controllers

{

public class TeamController : Controller

{

//

// GET: /Team/

private CompetitionDBContext db = new CompetitionDBContext();

public ActionResult Index()

{

List<SelectListItem> items = new List<SelectListItem>();

items.Add(new SelectListItem { Text = "C#", Value = "0" });

items.Add(new SelectListItem { Text = "C++", Value = "1" });

items.Add(new SelectListItem { Text = "Java", Value = "2", Selected = true });

items.Add(new SelectListItem { Text = "Python", Value = "3" });

ViewBag.ProgLangs = items;

//ViewBag.Problems = (db.Problems.ToList());

//var content = from p in db.Problems

// select new { p.name, p.ID };

//ViewBag.ProblemsList = content

// .Select(p => new SelectListItem

// {

// Text = p.name

// }).ToList();

var query = db.Problems.ToList().Select(c => new SelectListItem

{

Value = c.ID.ToString(),

Text = c.name

});

List<SelectListItem> problemsList = new List<SelectListItem>();

foreach (var item in query)

{

problemsList.Add(item);

}

ViewBag.ProblemsList = problemsList;

return View();

}

[HttpPost]

public ActionResult SubmitProblem()

{

return null;

// return View();

}

//

// GET: /Team/Details/5

public ActionResult Details(int id)

{

return View();

}

//

// GET: /Team/Create

public ActionResult Create()

{

return View();

}

//

// POST: /Team/Create

[HttpPost]

public ActionResult Create(FormCollection collection)

{

try

{

// TODO: Add insert logic here

return RedirectToAction("Index");

}

catch

{

return View();

}

}

//

// GET: /Team/Edit/5

public ActionResult Edit(int id)

{

return View();

}

//

// POST: /Team/Edit/5

[HttpPost]

public ActionResult Edit(int id, FormCollection collection)

{

try

{

// TODO: Add update logic here

return RedirectToAction("Index");

}

catch

{

return View();

}

}

//

// GET: /Team/Delete/5

public ActionResult Delete(int id)

{

return View();

}

//

// POST: /Team/Delete/5

[HttpPost]

public ActionResult Delete(int id, FormCollection collection)

{

try

{

// TODO: Add delete logic here

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

## UsersController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using ProgrammingTeamJudge.Models;

using ProgrammingTeamJudge.Filters;

namespace ProgrammingTeamJudge.Controllers

{

[InitializeSimpleMembership]

public class UsersController : Controller

{

private UsersContext db = new UsersContext();

//

// GET: /Users/

public ActionResult Index()

{

if (User.IsInRole("Admin"))

{

return View(db.UserProfiles.ToList());

}

else return View("PermissionsError");

}

//

// GET: /Users/Details/5

public ActionResult Details(int id = 0)

{

if (User.IsInRole("Admin"))

{

UserProfile userprofile = db.UserProfiles.Find(id);

if (userprofile == null)

{

return HttpNotFound();

}

return View(userprofile);

}

else return View("PermissionsError");

}

//

// GET: /Users/Create

public ActionResult Create()

{

if (User.IsInRole("Admin"))

{

return View();

}

else return View("PermissionsError");

}

//

// POST: /Users/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(UserProfile userprofile)

{

if (User.IsInRole("Admin"))

{

if (ModelState.IsValid)

{

db.UserProfiles.Add(userprofile);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(userprofile);

}

else return View("PermissionsError");

}

//

// GET: /Users/Edit/5

public ActionResult Edit(int id = 0)

{

if (User.IsInRole("Admin"))

{

UserProfile userprofile = db.UserProfiles.Find(id);

if (userprofile == null)

{

return HttpNotFound();

}

return View(userprofile);

}

else return View("PermissionsError");

}

//

// POST: /Users/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(UserProfile userprofile)

{

if (User.IsInRole("Admin"))

{

if (ModelState.IsValid)

{

db.Entry(userprofile).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

return View(userprofile);

}

else return View("PermissionsError");

}

//

// GET: /Users/Delete/5

public ActionResult Delete(int id = 0)

{

if (User.IsInRole("Admin"))

{

UserProfile userprofile = db.UserProfiles.Find(id);

if (userprofile == null)

{

return HttpNotFound();

}

return View(userprofile);

}

else return View("PermissionsError");

}

//

// POST: /Users/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

if (User.IsInRole("Admin"))

{

UserProfile userprofile = db.UserProfiles.Find(id);

db.UserProfiles.Remove(userprofile);

db.SaveChanges();

return RedirectToAction("Index");

}

else return View("PermissionsError");

}

protected override void Dispose(bool disposing)

{

db.Dispose();

base.Dispose(disposing);

}

}

}

## AccountModels.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity;

using System.Globalization;

using System.Web.Security;

namespace ProgrammingTeamJudge.Models

{

public class UsersContext : DbContext

{

public UsersContext()

: base("DefaultConnection")

{

}

public DbSet<UserProfile> UserProfiles { get; set; }

}

[Table("UserProfile")]

public class UserProfile

{

[Key]

[DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]

public int UserId { get; set; }

public string UserName { get; set; }

}

public class RegisterExternalLoginModel

{

[Required]

[Display(Name = "User name")]

public string UserName { get; set; }

public string ExternalLoginData { get; set; }

}

public class LocalPasswordModel

{

[Required]

[DataType(DataType.Password)]

[Display(Name = "Current password")]

public string OldPassword { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "New password")]

public string NewPassword { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm new password")]

[Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

public class LoginModel

{

[Required]

[Display(Name = "User name")]

public string UserName { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[Display(Name = "Remember me?")]

public bool RememberMe { get; set; }

}

public class RegisterModel

{

[Required]

[Display(Name = "User name")]

public string UserName { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

public bool isAdmin { get; set; }

public bool isJudge { get; set; }

public bool isTeam { get; set; }

}

public class ExternalLogin

{

public string Provider { get; set; }

public string ProviderDisplayName { get; set; }

public string ProviderUserId { get; set; }

}

}

## CompetitionModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

using System.Text;

namespace ProgrammingTeamJudge.Models

{

public class CompetitionModel

{

public int ID { get; set; }

public DateTime startDate { get; set; }

public DateTime endDate { get; set; }

public string name { get; set; }

public ICollection<TeamModel> teams { get; set; }

public ICollection<ProblemModel> problems { get; set; }

public CompetitionModel()

{

}

public CompetitionModel(DateTime startDate, DateTime endDate, string name)

{

// TODO: Complete member initialization

this.startDate = startDate;

this.endDate = endDate;

this.name = name;

}

public CompetitionModel(DateTime startDate, DateTime endDate, string name, List<TeamModel> teams, List<ProblemModel> problems)

{

this.startDate = startDate;

this.endDate = endDate;

this.name = name;

this.teams = teams;

this.problems = problems;

}

}

public class CompetitionDBContext : DbContext

{

public CompetitionDBContext() : base("CompetitionDBContext")

{

}

public DbSet<CompetitionModel> Competitions { get; set; }

public DbSet<TeamModel> Teams { get; set; }

public DbSet<ProblemModel> Problems { get; set; }

public DbSet<SubmissionModel> Submissions { get; set; }

public DbSet<LanguageModel> LanguageModels { get; set; }

}

public class CompetitionInitializer : System.Data.Entity.DropCreateDatabaseAlways<CompetitionDBContext>

{

protected override void Seed(CompetitionDBContext context)

{

var teams = new List<TeamModel>

{

new TeamModel{name="The A Team"},

new TeamModel{name="Thug Lyfe"},

new TeamModel{name="Faculty"},

};

teams.ForEach(s => context.Teams.Add(s));

context.SaveChanges();

var submissions = new List<SubmissionModel>

{

new SubmissionModel{ProblemID=1, team="The A Team", source=Encoding.ASCII.GetBytes("This is source code.")},

new SubmissionModel{ProblemID=1, team="Thug Lyfe", source=Encoding.ASCII.GetBytes("This is source code.")},

new SubmissionModel{ProblemID=1, team="Faculty", source=Encoding.ASCII.GetBytes("This is source code.")},

};

submissions.ForEach(s => context.Submissions.Add(s));

context.SaveChanges();

var problems = new List<ProblemModel>

{

new ProblemModel{name="LSP", CompetitionID=1, submissions = submissions, input=Encoding.ASCII.GetBytes("This is input."), output=Encoding.ASCII.GetBytes("This is output.")},

new ProblemModel{name="Factor By Primes", CompetitionID=1, input=Encoding.ASCII.GetBytes("This is input."), output=Encoding.ASCII.GetBytes("This is output.")},

new ProblemModel{name="Traveling Salesman", CompetitionID=1, input=Encoding.ASCII.GetBytes("This is input."), output=Encoding.ASCII.GetBytes("This is output.")},

new ProblemModel{name="Calculating the Area of a Regular N-gon", CompetitionID=1, input=Encoding.ASCII.GetBytes("This is input."), output=Encoding.ASCII.GetBytes("This is output.")},

};

problems.ForEach(s => context.Problems.Add(s));

context.SaveChanges();

var competitions = new List<CompetitionModel>

{

new CompetitionModel{startDate=DateTime.Parse("2014-12-01"), endDate=DateTime.Parse("2014-12-10"), name="SSE 657 Project 3 Competition", teams=teams, problems=problems},

};

competitions.ForEach(s => context.Competitions.Add(s));

context.SaveChanges();

}

}

}

## LanguageModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

namespace ProgrammingTeamJudge.Models

{

public class LanguageModel

{

[Key]

public int ID { get; set; }

public string name { get; set; }

public string extension { get; set; }

public LanguageModel()

{

}

public LanguageModel(string name, string extension)

{

// TODO: Complete member initialization

this.name = name;

this.extension = extension;

}

}

}

## ProblemModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

namespace ProgrammingTeamJudge.Models

{

public class LanguageModel

{

[Key]

public int ID { get; set; }

public string name { get; set; }

public string extension { get; set; }

public LanguageModel()

{

}

public LanguageModel(string name, string extension)

{

// TODO: Complete member initialization

this.name = name;

this.extension = extension;

}

}

}

## SubmissionModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

namespace ProgrammingTeamJudge.Models

{

public enum JudgingOutcome

{

Unknown, RuntimeError, CompileError, WrongOutput, PresentationError, CorrectSolution

}

public class SubmissionModel

{

[Key]

public int ID { get; set; }

public int ProblemID { get; set; }

public byte[] source { get; set; }

public string team { get; set; }

public string judge { get; set; }

public JudgingOutcome outcome { get; set; }

public int score { get; set; }

public SubmissionModel()

{

}

}

}

## TeamModel.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

namespace ProgrammingTeamJudge.Models

{

public class TeamModel

{

[Key]

public int ID { get; set; }

public string name { get; set; }

public TeamModel()

{

}

public TeamModel(string name)

{

// TODO: Complete member initialization

this.name = name;

}

}

}

## Account/\_ChangePasswordPartial.cshtml

@model ProgrammingTeamJudge.Models.LocalPasswordModel

<h3>Change password</h3>

@using (Html.BeginForm("Manage", "Account")) {

@Html.AntiForgeryToken()

@Html.ValidationSummary()

<fieldset>

<legend>Change Password Form</legend>

<ol>

<li>

@Html.LabelFor(m => m.OldPassword)

@Html.PasswordFor(m => m.OldPassword)

</li>

<li>

@Html.LabelFor(m => m.NewPassword)

@Html.PasswordFor(m => m.NewPassword)

</li>

<li>

@Html.LabelFor(m => m.ConfirmPassword)

@Html.PasswordFor(m => m.ConfirmPassword)

</li>

</ol>

<input type="submit" value="Change password" />

</fieldset>

}

## Account/\_ExternalLoginsListPartial.cshtml

@model ICollection<AuthenticationClientData>

@if (Model.Count == 0)

{

<div class="message-info">

<p>There are no external authentication services configured. See <a href="http://go.microsoft.com/fwlink/?LinkId=252166">this article</a>

for details on setting up this ASP.NET application to support logging in via external services.</p>

</div>

}

else

{

using (Html.BeginForm("ExternalLogin", "Account", new { ReturnUrl = ViewBag.ReturnUrl }))

{

@Html.AntiForgeryToken()

<fieldset id="socialLoginList">

<legend>Log in using another service</legend>

<p>

@foreach (AuthenticationClientData p in Model)

{

<button type="submit" name="provider" value="@p.AuthenticationClient.ProviderName" title="Log in using your @p.DisplayName account">@p.DisplayName</button>

}

</p>

</fieldset>

}

}

## Account/\_RemoveExternalLoginsPartial.cshtml

@model ICollection<AuthenticationClientData>

@if (Model.Count == 0)

{

<div class="message-info">

<p>There are no external authentication services configured. See <a href="http://go.microsoft.com/fwlink/?LinkId=252166">this article</a>

for details on setting up this ASP.NET application to support logging in via external services.</p>

</div>

}

else

{

using (Html.BeginForm("ExternalLogin", "Account", new { ReturnUrl = ViewBag.ReturnUrl }))

{

@Html.AntiForgeryToken()

<fieldset id="socialLoginList">

<legend>Log in using another service</legend>

<p>

@foreach (AuthenticationClientData p in Model)

{

<button type="submit" name="provider" value="@p.AuthenticationClient.ProviderName" title="Log in using your @p.DisplayName account">@p.DisplayName</button>

}

</p>

</fieldset>

}

}

## Accounts/\_SetPasswordPartial.cshtml

@model ProgrammingTeamJudge.Models.LocalPasswordModel

<p>

You do not have a local password for this site. Add a local

password so you can log in without an external login.

</p>

@using (Html.BeginForm("Manage", "Account")) {

@Html.AntiForgeryToken()

@Html.ValidationSummary()

<fieldset>

<legend>Set Password Form</legend>

<ol>

<li>

@Html.LabelFor(m => m.NewPassword)

@Html.PasswordFor(m => m.NewPassword)

</li>

<li>

@Html.LabelFor(m => m.ConfirmPassword)

@Html.PasswordFor(m => m.ConfirmPassword)

</li>

</ol>

<input type="submit" value="Set password" />

</fieldset>

}

## Account/ExternalLoginConfirmation.cshtml

@model ProgrammingTeamJudge.Models.RegisterExternalLoginModel

@{

ViewBag.Title = "Register";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

<h2>Associate your @ViewBag.ProviderDisplayName account.</h2>

</hgroup>

@using (Html.BeginForm("ExternalLoginConfirmation", "Account", new { ReturnUrl = ViewBag.ReturnUrl })) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>Association Form</legend>

<p>

You've successfully authenticated with <strong>@ViewBag.ProviderDisplayName</strong>.

Please enter a user name for this site below and click the Confirm button to finish

logging in.

</p>

<ol>

<li class="name">

@Html.LabelFor(m => m.UserName)

@Html.TextBoxFor(m => m.UserName)

@Html.ValidationMessageFor(m => m.UserName)

</li>

</ol>

@Html.HiddenFor(m => m.ExternalLoginData)

<input type="submit" value="Register" />

</fieldset>

}

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Account/ExternalLoginFailure.cshtml

@{

ViewBag.Title = "Login Failure";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

<h2>Unsuccessful login with service.</h2>

</hgroup>

## Account/Login

@model ProgrammingTeamJudge.Models.LoginModel

@{

ViewBag.Title = "Log in";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

</hgroup>

<section id="loginForm">

<h2>Use a local account to log in.</h2>

@using (Html.BeginForm(new { ReturnUrl = ViewBag.ReturnUrl })) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>Log in Form</legend>

<ol>

<li>

@Html.LabelFor(m => m.UserName)

@Html.TextBoxFor(m => m.UserName)

@Html.ValidationMessageFor(m => m.UserName)

</li>

<li>

@Html.LabelFor(m => m.Password)

@Html.PasswordFor(m => m.Password)

@Html.ValidationMessageFor(m => m.Password)

</li>

<li>

@Html.CheckBoxFor(m => m.RememberMe)

@Html.LabelFor(m => m.RememberMe, new { @class = "checkbox" })

</li>

</ol>

<input type="submit" value="Log in" />

</fieldset>

<p>

@Html.ActionLink("Register", "Register") if you don't have an account.

</p>

}

</section>

<section class="social" id="socialLoginForm">

<h2>Use another service to log in.</h2>

@Html.Action("ExternalLoginsList", new { ReturnUrl = ViewBag.ReturnUrl })

</section>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Account/Manage.cshtml

@model ProgrammingTeamJudge.Models.LocalPasswordModel

@{

ViewBag.Title = "Manage Account";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

</hgroup>

<p class="message-success">@ViewBag.StatusMessage</p>

<p>You're logged in as <strong>@User.Identity.Name</strong>.</p>

@if (ViewBag.HasLocalPassword)

{

@Html.Partial("\_ChangePasswordPartial")

}

else

{

@Html.Partial("\_SetPasswordPartial")

}

<section id="externalLogins">

@Html.Action("RemoveExternalLogins")

<h3>Add an external login</h3>

@Html.Action("ExternalLoginsList", new { ReturnUrl = ViewBag.ReturnUrl })

</section>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Account/Register.cshtml

@model ProgrammingTeamJudge.Models.RegisterModel

@{

ViewBag.Title = "Register";

}

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

<h2>Create a new account.</h2>

</hgroup>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

@Html.ValidationSummary()

<fieldset>

<legend>Registration Form</legend>

<ol>

<li>

@Html.LabelFor(m => m.UserName)

@Html.TextBoxFor(m => m.UserName)

</li>

<li>

@Html.LabelFor(m => m.Password)

@Html.PasswordFor(m => m.Password)

</li>

<li>

@Html.LabelFor(m => m.ConfirmPassword)

@Html.PasswordFor(m => m.ConfirmPassword)

</li>

<li>

@Html.CheckBoxFor(m => m.isAdmin) Admin

@Html.CheckBoxFor(m => m.isJudge) Judge

@Html.CheckBoxFor(m => m.isTeam) Team

</li>

</ol>

<input type="submit" value="Register" />

</fieldset>

}

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Admin/Competitions.cshtml

@model IEnumerable<ProgrammingTeamJudge.Models.CompetitionModel>

@{

ViewBag.Title = "Competitions";

Layout = "\_AdminLayout.cshtml";

}

<h2>Competitions</h2>

<table width="100%">

<col style="width:50%" />

<col style="width:25%" />

<col style="width:25%" />

<col style="width:10%" />

<tr>

<th>

Name

</th>

<th>

Start Time

</th>

<th>

End Time

</th>

</tr>

@foreach (var competition in Model)

{

<tr>

<td>

@competition.name

</td>

<td>

@competition.startDate

</td>

<td>

@competition.endDate

</td>

</tr>

}

</table>

## Admin/Index.cshtml

@{

ViewBag.Title = "Admin";

Layout = "\_AdminLayout.cshtml";

}

<h2>Admin</h2>

## Admin/Languages.cshtml

@model IEnumerable<ProgrammingTeamJudge.Models.LanguageModel>

@{

ViewBag.Title = "Languages";

Layout = "\_AdminLayout.cshtml";

}

<h2>Lanugages</h2>

<table width="50%">

<col style="width:75%" />

<col style="width:25%" />

<tr>

<th>

Name

</th>

<th>

Extension

</th>

</tr>

@foreach (var language in Model)

{

<tr>

<td>

@language.name

</td>

<td>

@language.extension

</td>

</tr>

}

</table>

## Admin/Problems.cshtml

@model IEnumerable<ProgrammingTeamJudge.Models.ProblemModel>

@{

ViewBag.Title = "Problems";

Layout = "\_AdminLayout.cshtml";

}

<h2>Problems</h2>

<table width="50%">

<tr>

<th>

Name

</th>

</tr>

@foreach (var problem in Model)

{

<tr>

<td>

@problem.name

</td>

</tr>

}

</table>

## Admin/Users.cshtml

@model IEnumerable<ProgrammingTeamJudge.Models.UserProfile>

@{

ViewBag.Title = "Users";

Layout = "\_AdminLayout.cshtml";

}

<h2>Users</h2>

<table width="100%">

<col style="width:40%" />

<col style="width:20%" />

<col style="width:20%" />

<col style="width:20%" />

<tr>

<th>

Name

</th>

<th>

Is Admin

</th>

<th>

Is Judge

</th>

<th>

Is Team

</th>

</tr>

@foreach (var user in Model)

{

<tr>

<td>

@user.UserName

</td>

<td>

@false

</td>

<td>

@false

</td>

<td>

@false

</td>

</tr>

}

</table>

## Competitions/Create.cshtml

@model ProgrammingTeamJudge.Models.CompetitionModel

@{

ViewBag.Title = "Create";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Create</h2>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>CompetitionModel</legend>

<div class="editor-label">

@Html.LabelFor(model => model.startDate)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.startDate)

@Html.ValidationMessageFor(model => model.startDate)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.endDate)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.endDate)

@Html.ValidationMessageFor(model => model.endDate)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.name)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.name)

@Html.ValidationMessageFor(model => model.name)

</div>

<div class="editor-label">

Problems

</div>

<div class="editor-field">

@Html.ListBox("problemsList")

</div>

<p>

<input type="submit" value="Create" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Competitions/Delete.cshtml

@model ProgrammingTeamJudge.Models.CompetitionModel

@{

ViewBag.Title = "Delete";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>

<fieldset>

<legend>CompetitionModel</legend>

<div class="display-label">

@Html.DisplayNameFor(model => model.startDate)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.startDate)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.endDate)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.endDate)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.name)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.name)

</div>

</fieldset>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

<p>

<input type="submit" value="Delete" /> |

@Html.ActionLink("Back to List", "Index")

</p>

}

## Competitions/Details.cshtml

@model ProgrammingTeamJudge.Models.CompetitionModel

@{

ViewBag.Title = "Details";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Details</h2>

<fieldset>

<legend>CompetitionModel</legend>

<div class="display-label">

@Html.DisplayNameFor(model => model.startDate)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.startDate)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.endDate)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.endDate)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.name)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.name)

</div>

@foreach(var prob in ViewBag.problems)

{

<li>

@prob.name

</li>

}

</fieldset>

<p>

@Html.ActionLink("Edit", "Edit", new { id=Model.ID }) |

@Html.ActionLink("Back to List", "Index")

</p>

## Competitions/Edit.cshtml

@model ProgrammingTeamJudge.Models.CompetitionModel

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Edit</h2>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>CompetitionModel</legend>

@Html.HiddenFor(model => model.ID)

<div class="editor-label">

@Html.LabelFor(model => model.startDate)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.startDate)

@Html.ValidationMessageFor(model => model.startDate)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.endDate)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.endDate)

@Html.ValidationMessageFor(model => model.endDate)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.name)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.name)

@Html.ValidationMessageFor(model => model.name)

</div>

<p>

<input type="submit" value="Save" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Competitions/Index.cshtml

@model IEnumerable<ProgrammingTeamJudge.Models.CompetitionModel>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Competitions</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table width="100%">

<col style="width:35%" />

<col style="width:20%" />

<col style="width:20%" />

<col style="width:25%" />

<tr>

<th>

@Html.DisplayNameFor(model => model.name)

</th>

<th>

@Html.DisplayNameFor(model => model.startDate)

</th>

<th>

@Html.DisplayNameFor(model => model.endDate)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.name)

</td>

<td>

@Html.DisplayFor(modelItem => item.startDate)

</td>

<td>

@Html.DisplayFor(modelItem => item.endDate)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.ID }) |

@Html.ActionLink("Details", "Details", new { id=item.ID }) |

@Html.ActionLink("Delete", "Delete", new { id=item.ID })

</td>

</tr>

}

</table>

## Home/Chart.cshtml

@{

Chart CurrentCompetition = new Chart(800, 600, ChartTheme.Blue);

CurrentCompetition.AddTitle("Competition Scoring");

CurrentCompetition.Write("png");

}

## Home/Index.cshtml

@{

ViewBag.Title = "Home Page";

}

<h1>Current Competition:</h1>

<img src="/Home/Chart" />

<h2>Past Competitions:</h2>

<ol class="square">

<li class="one">

<a href="http://www.google.com">Competition name</a>

</li>

<li class="two">

<a href="http://www.google.com">Competition name</a>

</li>

<li class="three">

<a href="http://www.google.com">Competition name</a>

</li>

</ol>

## Home/Team.cshtml

@{

ViewBag.Title = "Team";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Team</h2>

## Judge/Claim.cshtml

@{

ViewBag.Title = "Claim";

}

@using ProgrammingTeamJudge.Models;

@model ProgrammingTeamJudge.Models.SubmissionModel

<script type="text/javascript">

function updateTextInput(val) {

document.getElementById('textInput').value = val;

}

</script>

<h2>@ViewBag.Submission.team's Submission for @ViewBag.Problem.name</h2>

<h3>Judging Packet (Source, Input, Output) </h3> <input type="button" value="Download .zip" onclick="location.href='@Url.Action("Download", "Judge", new { submission = ViewBag.Submission.ID, problem = ViewBag.Problem.ID})'" />

<br /><br /><br />

<form action="/Judge/SubmitJudgement" method="post">

<h2>Score</h2><input type="range" name="Score" min="0" max="10" onchange="updateTextInput(this.value);" /> <input type="text" disabled="disabled" width="5" id="textInput" value=""> <br />

<h2>Judging Outcome</h2><br />

<input type="radio" name="Outcome" value="RuntimeError">RuntimeError

<br>

<input type="radio" name="Outcome" value="CompileError">CompileError

<br>

<input type="radio" name="Outcome" value="WrongOutput">WrongOutput

<br>

<input type="radio" name="Outcome" value="PresentationError">PresentationError

<br>

<input type="radio" name="Outcome" value="CorrectSolution">CorrectSolution

@\*@Html.DropDownListFor(model => model.outcome, new SelectList(Enum.GetValues(typeof(ProgrammingTeamJudge.Models.JudgingOutcome))))\*@

<input type="hidden" name="Judge" value="@User.Identity.Name" />

<input type="hidden" name="SubmissionID" value=@ViewBag.Submission.ID />

<br /><br />

<p>

<input type="submit" value="Submit")'"/>

<input type="button" value="Cancel" onclick="location.href='@Url.Action("Competition", "Judge", new { id = ViewBag.Problem.CompetitionID})'"/>

</p>

</form>

## Judge/Competition.cshtml

@model IEnumerable<ProgrammingTeamJudge.Models.ProblemModel>

@using ProgrammingTeamJudge.Models;

@{

ViewBag.Title = "Competition";

}

<h1>Problems within selected competition</h1>

@Html.ActionLink("Back to competitions", "Index")

@foreach (var item in Model)

{

<h2>@Html.DisplayFor(modelItem => item.name)</h2>

<table cellpadding="20">

@if (item.submissions != null)

{

<tr>

<th>Submitting Team </th>

<th>Outcome </th>

<th>Score </th>

<th>Judge </th>

<th>Action </th>

</tr>

foreach (var submission in item.submissions)

{

<tr>

<td>

@Html.DisplayFor(modelItem2 => submission.team)

</td>

<td>

@Html.DisplayFor(modelItem2 => submission.outcome)

</td>

<td>

@Html.DisplayFor(modelItem2 => submission.score)

</td>

<td>

@Html.DisplayFor(modelItem2 => submission.judge)

</td>

<td>

@if (JudgingOutcome.Unknown.Equals(submission.outcome))

{

@Html.ActionLink("Claim", "Claim", new { submission = submission.ID, problem = item.ID});

}

</td>

</tr>

}

}

</table>

}

## Judge/Index.cshtml

@model IEnumerable<ProgrammingTeamJudge.Models.CompetitionModel>

@{

ViewBag.Title = "Judge";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Competitions</h2>

<table>

<tr>

<th>

@Html.DisplayNameFor(model => model.name)

</th>

<th>

@Html.DisplayNameFor(model => model.startDate)

</th>

<th>

@Html.DisplayNameFor(model => model.endDate)

</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.name)

</td>

<td>

@Html.DisplayFor(modelItem => item.startDate.Date)

</td>

<td>

@Html.DisplayFor(modelItem => item.endDate.Date)

</td>

<td>

@Html.ActionLink("Select", "Competition", new {id = item.ID})

</td>

</tr>

}

</table>

## Languages/Create.cshtml

@model ProgrammingTeamJudge.Models.LanguageModel

@{

ViewBag.Title = "Create";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Create</h2>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>LanguageModel</legend>

<div class="editor-label">

@Html.LabelFor(model => model.name)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.name)

@Html.ValidationMessageFor(model => model.name)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.extension)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.extension)

@Html.ValidationMessageFor(model => model.extension)

</div>

<p>

<input type="submit" value="Create" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Languages/Delete.cshtml

@model ProgrammingTeamJudge.Models.LanguageModel

@{

ViewBag.Title = "Create";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Create</h2>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>LanguageModel</legend>

<div class="editor-label">

@Html.LabelFor(model => model.name)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.name)

@Html.ValidationMessageFor(model => model.name)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.extension)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.extension)

@Html.ValidationMessageFor(model => model.extension)

</div>

<p>

<input type="submit" value="Create" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Languages/Details.cshtml

@model ProgrammingTeamJudge.Models.LanguageModel

@{

ViewBag.Title = "Details";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Details</h2>

<fieldset>

<legend>LanguageModel</legend>

<div class="display-label">

@Html.DisplayNameFor(model => model.name)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.name)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.extension)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.extension)

</div>

</fieldset>

<p>

@Html.ActionLink("Edit", "Edit", new { id=Model.ID }) |

@Html.ActionLink("Back to List", "Index")

</p>

## Languages/Edit.cshtml

@model ProgrammingTeamJudge.Models.LanguageModel

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Edit</h2>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>LanguageModel</legend>

@Html.HiddenFor(model => model.ID)

<div class="editor-label">

@Html.LabelFor(model => model.name)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.name)

@Html.ValidationMessageFor(model => model.name)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.extension)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.extension)

@Html.ValidationMessageFor(model => model.extension)

</div>

<p>

<input type="submit" value="Save" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Languages/Index.cshtml

@model IEnumerable<ProgrammingTeamJudge.Models.LanguageModel>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Languages</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table width="75%">

<col style="width:50%" />

<col style="width:25%" />

<col style="width:25%" />

<tr>

<th>

@Html.DisplayNameFor(model => model.name)

</th>

<th>

@Html.DisplayNameFor(model => model.extension)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.name)

</td>

<td>

@Html.DisplayFor(modelItem => item.extension)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.ID }) |

@Html.ActionLink("Details", "Details", new { id=item.ID }) |

@Html.ActionLink("Delete", "Delete", new { id=item.ID })

</td>

</tr>

}

</table>

## Problems/Create.cshtml

@model ProgrammingTeamJudge.Models.ProblemModel

@{

ViewBag.Title = "Create";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Create</h2>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

@Html.ValidationSummary(true)

<fieldset>

<legend>ProblemModel</legend>

<div class="editor-label">

@Html.LabelFor(model => model.CompetitionID)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.CompetitionID)

@Html.ValidationMessageFor(model => model.CompetitionID)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.name)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.name)

@Html.ValidationMessageFor(model => model.name)

</div>

<p>

<input type="submit" value="Create" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## Problems/Delete.cshtml

@model ProgrammingTeamJudge.Models.ProblemModel

@{

ViewBag.Title = "Delete";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>

<fieldset>

<legend>ProblemModel</legend>

<div class="display-label">

@Html.DisplayNameFor(model => model.CompetitionID)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.CompetitionID)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.name)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.name)

</div>

</fieldset>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

<p>

<input type="submit" value="Delete" /> |

@Html.ActionLink("Back to List", "Index")

</p>

}

## Problems/Details.cshtml

@model ProgrammingTeamJudge.Models.ProblemModel

@{

ViewBag.Title = "Details";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<h2>Details</h2>

<fieldset>

<legend>ProblemModel</legend>

<div class="display-label">

@Html.DisplayNameFor(model => model.CompetitionID)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.CompetitionID)

</div>

<div class="display-label">

@Html.DisplayNameFor(model => model.name)

</div>

<div class="display-field">

@Html.DisplayFor(model => model.name)

</div>

</fieldset>

<p>

@Html.ActionLink("Edit", "Edit", new { id=Model.ID }) |

@Html.ActionLink("Back to List", "Index")

</p>