**Advanced Web Development – Midterm (100 points)**

*Mary J. Snyder*

**Due date on blackboard. No late submissions will be graded.**

Instructions:

1. This is a take-home midterm. Please submit typed answers and computer generated diagrams through the blackboard, if you choose to include hand-drawn diagrams, then they should be neat, legible, and properly referenced in text.
2. If you have hand drawn diagrams then scan and email to me or give hardcopies to the department secretary by due date.
3. All work is to be done independently: you may not discuss any of the material with your classmates, friends, or colleagues.
4. You may use external references including web pages, research literature, and textbooks. However, you must indicate which resources you have consulted. If your answers make use of specific facts that you find in an external resource, please reference the resource. Don’t just copy from an external resource, use your own words. Any use of external resources without proper attributionwill be examined for signs of possible plagiarism. I take this very seriously.

Question 1 – Web application design (70%)

The aim of this question is to **design** **(not implement)** a web application (using Rails) to manage online testing and certifications (for an example see [www.brainbench.com](http://www.brainbench.com) – individual certification). The website will offer tests in various categories (e.g., information technology, aptitude, finance etc.) and under each category offer specific certification (e.g., Java certification under IT, problem solving under aptitude, etc.). The functionality provided by the website will include the following:

1. Users will be able to search for (using various criteria), register, login, and take tests.
2. Each test area (say Java) will lead to a certification (say Java certification) if the user passes the test.
3. The application **will store the results of all the tests that a user has taken and also store all the certifications that a user has received.**
4. Each **test result and certification can be marked as “public”** by the user so the world can see it.
5. Unregistered users (say employers) will be able to access the results of tests taken by a user and certifications received by entering a **PIN number provided by the user**. They can also provide the **user lastname and email to access all the results that the user has marked as “public**.”
6. The application will store a question bank for all the tests that it provides. There will be two question banks: **real and practice**. Each user should be able to take practice tests for any certification.
7. The application should maintain relevant statistics about usage: **how many tests have been taken, when, in what categories, etc.? How much time have users spent on each test?**
8. The questions, certifications, categories, users, and other pertinent data can be modified by administrators.

Answer the following based on the description above. Be as comprehensive as possible with the functionalities in mind. While designing, keep good design principles in mind with respect to the database, application structure (MVC etc.), and scalability. With each answer include a brief description whenever needed:

1. Provide all the models (in the Rails sense) that you will need to implement the above application. Include all attributes (table columns) that each model needs to have to implement the above functionality. (10 points)
   1. **Users**
      1. ***userID (integer)*, userName (string), userFirstName (string), userLastName (string), userEmail (string), userPwd (string), userPin (integer: null when no pin active/set), userAdmin (boolean: true if admin, false if not admin)**
   2. **Categories**
      1. ***categoryID* (integer), categoryName (string),**
   3. **Certifications**
      1. ***certificationID* (integer), certificationName (string), *categoryID* (integer)**
   4. **Tests**
      1. ***testID (integer)*, testName (string), *certificationID* (integer)**
   5. **Questions**
      1. ***questionID (integer)*, questionText (string), questionAnswer (string), questionPractice (boolean: true question is a practice question, false question is a real question), *testID* (integer)**
   6. **Results**
      1. ***userID (integer)*, *testID (integer)*, resultPractice (boolean: true result is for a practice test, false result is for a real test), timeSpent (integer: seconds elapsed), resultDate (date), resultGrade (boolean: true if pass, false if fail), resultPublic (boolean: true result is public, false result is not public)**
2. Provide the relationships between all the models (using the has\_many, has\_one, belongs\_to, etc.). (10 points)
   1. **Categories has\_many Certifications**
   2. **Certifications belongs\_to Categories**
   3. **Tests belongs\_to Certifications**
   4. **Questions belongs\_to Tests**
   5. **Tests has\_many Questions**
   6. **Certifications has\_one Tests**
   7. **Results belongs\_to Users**
   8. **Results belongs\_to Tests**
   9. **Users has\_many Results**
   10. **Tests has\_many Results**
3. Provide a list of all the controllers that you might need for such an application. Please think about REST (each resource has one controller) principles here (10 points)
   1. **Main Controller**
   2. **User Controller**
   3. **Test Controller**
   4. **Results Controller**
   5. **Admin Controller**
4. Provide the main functions/methods/actions for each controller that will be needed to implement the above functionality. (10 points)
   1. **Main Controller**
      1. **Index**
      2. **Register**
      3. **Login**
      4. **Logout**
      5. **User Pin Search**
      6. **Public Search**
   2. **User Controller**
      1. **Index**
      2. **Search**
      3. **Manage (add, edit, remove) pin**
      4. **Manage info (name, email, password)**
   3. **Test Controller**
      1. **Index**
      2. **Search**
      3. **Results (display results for test\_id)**
      4. **Manage**
   4. **Results Controller**
      1. **Index**
      2. **User (display results by user\_id)**
      3. **Test (display results by test\_id)**
      4. **Category (display results by category\_id)**
      5. **Certification (display results by certification\_id)**
      6. **Admin results**
      7. **Manage public results**
   5. **Admin**
      1. **Index**
      2. **View stats**
      3. **Manage (edit/add/remove) question**
      4. **Manage test**
      5. **Manage certification**
      6. **Manage category**
      7. **Manage user**
5. Show the workflow through the system for the following scenarios (see example below): (30 points)
   1. User logs in
      1. **User enters username, password, and clicks the “login” button in the *index* view of the *main* controller**
      2. **The “login” button triggers the method *login* in *main* controller, passing the *user\_name* and *user\_pwd***
      3. **If the password does not match the userPwd stored for the userName=*user\_name*, the user is redirected back to the *index* view of the *main* controller**
      4. **If the password does match the userPwd stored for userName=*user­\_name* and the user is not an admin, the user is redirected to the *index* view for the *user* controller, passing the *user\_id* (userID)**
   2. User searches for tests to take
      1. **From the *index* view of the *user* controller, the user clicks the “search” button in the *index* view of the *user* controller**
      2. **The “search” button redirects to the *search* view in *test* controller, passing the *user\_id* (userID)**
      3. **The *search* view in *test* controller provides search options for the user and when selection is complete, the user clicks the “submit” button**
      4. **The “submit” button redirects to the *results* view of the *test* controller passing the *user\_id* (userID) and search criteria (*test\_name, test\_category, etc.)*, which displays the results of the search**
   3. User lists all the tests he/she took with a listing of certifications obtained.
      1. **From the *index* view of the *user* controller, the user clicks the “view complete” button in the *index* view of the *user* controller**
      2. **The “view complete” button redirects to the *user* view in the *results* controller, passing the *user\_id* (userID) and public\_only=false**
      3. **The *user* view in the *results* controller searches for all records with userID=*user\_id* (not just public ones) and posts the results for the user**
      4. **The *user* view in the *results* controller also searches for all certification by using the testID for any non-practice results to obtain the certificationID, which is used to post the certificationName for the user**
   4. Employer searches for all the public tests and certifications for a user “Smith” with email address “smith@gmail.com.”
      1. **From the *index* view of the *main* controller, the employer clicks the “public search” button in the *index* view of the *main* controller**
      2. **The “public search” button redirects to the *public* view in the *main* controller**
      3. **The employer enters the user last name “Smith” as well as the email address “smith@gmail.com” and clicks the “search” button in the *public* view of the *main* controller**
      4. **The “search” button redirects to the method *public* view in the *results* controller, passing the *user\_last\_name* and *user\_email***
      5. **The *public* view in the *results* controller queries for a userID using the *user\_last\_name* and *user\_email***
      6. **If there are no users with userLastName=*user\_last\_name* and userEmail=*user\_email*, the user is redirected back to *index* view of the *main* controller**
      7. **If there is a user with the userLastName=*user\_last\_name* and userEmail=*user\_email*, the user is redirected to the *user* view of the *results* controller, passing user\_id (userID) and public\_only=true**
      8. **The *user* view of the *results* controller searches for all records with userID=*user\_id* where resultPublic=true and posts the results for the user**
      9. **The *user* view of the *results* controller also searches for all certifications by using the testID for resultPublic=true, non-practice results to obtain the certificationID, which is used to post the certificationName for the user**
   5. Administrator wants to know how much total time the users have spent testing on the system.
      1. **From the *index* view of the *main* controller, the admin enters username, password, and clicks the “login” button in the *index* view of the *main* controller**
      2. **The “login” button triggers the method *login* in the *main* controller, passing the *user\_name* and *user\_pwd***
      3. **If the password does not match the userPwd stored for the userName=*user\_name*, the user is redirected back to the *index* view of the *main* controller**
      4. **If the password does match the userPwd stored for userName=*user­\_name*, but userAdmin is false, the user is redirected to the *index* view for the *user* controller, passing the *user\_id***
      5. **If the password does match the userPwd stored for userName=user\_name, and userAdmin is true, the user is redirected to the *index* view for the *admin* controller, passing the *user\_id*.**
      6. **From the *index* view of the *admin* controller, the admin clicks the “Results Stats” button in the *index* view of the *admin* controller**
      7. **The “Results Stats” button redirects to the *admin* view of the *results* controller**
      8. **The admin selects the option “Test Time” and clicks the “submit” button from the *admin* view of the *results* controller**
      9. **The “Total Time” option for the “submit” button triggers the method *total\_test\_time* in the *results* controller**
      10. **The method *total\_test\_time* calculates the total test time from all the test results and posts the answer (total time)**

Example using the Depot application: user adds item to cart

* User clicks button “add to cart” in the *index* view of *store* controller
* Button triggers method *add\_to\_cart* in *store* controller, passes the *product\_id*
* *add\_to\_cart* uses the *product* model to find the item to be added (by *product\_id*)
* the *add\_to\_cart* view shows the cart with the added product

The above example is in prose, you can use also an activity diagram to show the above if you wish. However, explicitly mention all the models, views, and controller methods being called, use swim lanes if necessary.

Question 2: Agile development (30%)

Your company got the contract to implement the web application you designed in Question 1. Write a short white paper aimed at convincing your managers and their bosses that your organization should adopt agile development methods. Describe what those methods are, how they can help, and what should be done to implement them. Compare them to other methods of development and how they may or may not work for the above work in Question 1. As your management is very cost-sensitive, they may be worried that agile development would increase software development costs: how would you convince them that this is not a problem? Use the papers discussed in class and any others if needed with references. Your management has a short attention span, your job is to explain agile development and convince your management as best as you can in *two* pages (double-spaced). *Please write in your own words, cite all outside sources.*

**Agile Development for Online Test & Certification Management Application**

**Agile development improves upon other methodologies such as waterfall or code-and-fix to save time and money while still limiting risk. In the more traditional waterfall methodology, the development process is carried out sequentially starting with requirements definition with the customer, designing the application or product, development including coding and unit testing, and final testing and/or certification. With this methodology, the customer may not see the resulting product until near or at completion. This creates an environment of high risk, leaving little to no time for any changes to be made if the interpretation of the requirements does not match the expectation of the customer. Even minor changes can cause delays or redesign, which takes time and costs money.**

**With agile development, development is broken down into a series of smaller pieces, based on feature development, called “sprints.” Each sprint is like a smaller version of Waterfall in that requirements are defined with the customer, the design for this particular piece of the product is completed, coding and unit testing is performed, and final testing or any certification is done. Since the result of each sprint is a complete feature, a prototype of the final product can be created in a short amount of time (length of the sprint). Having a prototype provides a way for the customer and/or target audience to give feedback during the development process, before the entire project is complete. This allows the development team to make changes as early as possible, impacting less of the overall production and costing less money overall to update/fix, reducing overall risk.**

**In other methodologies such as waterfall, customer feedback is given once the development has been completed. The customer may realize at the end one feature is not as important as they had originally thought or that they forgot a feature that is important for user interaction. Unfortunately, at that point in the process, it is almost too late to make any changes to the original requirements as the costs in time and money would be substantial. Since the work is broken down by features in agile development, the developers and customer work together to prioritize and complete the features that are most important or have the most impact early on. The customer also has the flexibility during the process to change their minds and add, delete, or change features from the original vision. Completing a few features at a time also allows the customer to have a proof of concept before the entire product is complete, which garners confidence and trust in the development team and process as a whole.**

**Agile development also encourages collaboration and communication throughout the process, not only among the development team, but with the customer as well. Each sprint has one team member designated as the leader, called a SCRUM master. Since the SCRUM master changes with each sprint, no one person is being sacrificed from the development to lead, their talents can be using in other/future sprints. The SCRUM master holds weekly or sometime daily meetings on progress, so any issues can be taken care of as soon as possible. In addition, frequent meetings allow for better communication with the customer on the status of the product, which can be a major cause of contention in other methodologies.**

**Another advantage of the agile methodology for web development is the ability to work on development in parallel. Once the models for the interface between the front end and back end are decided upon, the work in the controllers and views can be done simultaneously. This not only saves development time, but also allows the test team to learn the APIs and begin work on automated testing during each sprint. By the end of the last sprint, a fully functioning automated test suite can be complete to accompany the software without spending extra time. Automated tests will help ensure a more stable final product and the savings of developer and tester time translates into overall savings for the project.**

**In conclusion, the agile development process has many advantages over other methodologies, especially for web development. It allows for rapid prototyping and proof-of-concept for early feedback from the customer and/or user, it streamlines the development process, allowing for parallel development of front-end and back-end production as well as automated tests, all while reducing risk and cost.**