Exam Generator

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

James Domagalski

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

October, 2010

Exam Generator

By James Domagalski

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

_____	_____
Dr. Kenny Hunt Examination Committee Chairperson	Date
_____	_____
Dr. Kasi Periyasamy Examination Committee Member	Date
_____	_____
Dr. David Riley Examination Committee Member	Date

Abstract

Domagalski, James, J., “Exam Generator”, Master of Software Engineering, December 2010, (Dr. Kenny Hunt, Dr. Kasi Periyasamy).

The UW-La Crosse Exam Generator is an online web application that was created to aid faculty members of educational institutions with creating and maintaining exams. With the Exam Generator faculty members can create and maintain a repository of questions. Questions within the system can be used by faculty members to manually or automatically generate exams. The Exam Generator also provides students of these education institutions the opportunity to prepare for a course by taking online practice exams using exams and questions that have been shared with the student.

This manuscript describes the development of the UW-La Crosse Exam Generator as well as the challenges and issues that arose during its development.

Acknowledgements

I would like to express my sincere thanks to my project advisors Dr. Kenny Hunt and Dr. Kasi Periyasamy for their guidance. I would also like to express my thanks to the Computer Science Department and the University of Wisconsin-La Crosse for providing the knowledge and tools for completing the project. Finally, I wish to thank my friends and family for their patience and encouragement throughout the project's tenure.

Table of Contents

List of Tables/Figures

Glossary

Application Framework

A software framework used by developers to implement the standard structure of an application for a specific development environment (such as an operating system or a web application)

GUI

Graphical User Interface (GUI) is the part of the application that the user sees and interacts with

Hibernate

An object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database

IEEE

The Institute of Electrical and Electronics Engineers is a professional organization that establishes some standards, publishes a variety of journals, and sponsors conferences.

iText

A free and open source Java library written by iText Software Corp that allows developers to easily create and manipulate PDF documents.

PDF

Portable Document Format (PDF) is a file format created by Adobe Systems in 1993 for document exchange. PDF is used for representing two-dimensional

documents in a manner independent of the application software, hardware, and operating system.

Seam Framework

A web application framework developed by JBoss, a division of Red Hat, that combines the features of Enterprise JavaBeans (EJB3) and JavaServer Faces (JSF) together.

RichFaces

An open source Ajax enabled component library for JavaServer Faces host by JBoss.org that allows easy integration of Ajax capabilities into enterprise application development.

UML

Unified Modeling Language (UML) is a collection of metalanguages that is used for specifying, visualizing, constructing, and documenting the artifacts of software. The types of UML diagrams included in this document include Use Case Diagrams, Class Diagrams, and Entity Relationship(ER) Diagrams.

1. Background Information

Each year teachers and faculty members at education institutions across the world are searching for ways to properly evaluate their students' understanding of course content. A proven method for doing this is to test a student's knowledge through the use of multiple choice quizzes or exams. The use of multiple choice exams as a way of evaluating knowledge comes with the risk of a student cheating. This certainly becomes the case when there is a high volume of students taking the same exam in one area. This scenario creates the possibility of a student looking at a neighboring student's exam sheet to copy answers.

One popular method for discouraging cheating on an examination is to create multiple versions and distribute them so students sitting near each other complete different exam forms. Multiple exam versions makes it difficult for students to copy answers but puts strain on the faculty members since multiple versions of the same exam must be created and graded.

The Exam Generator is an application that has been created to aid faculty members in creating exams as well as to aid students in preparing for exams. The Exam Generator provides the tools for a faculty member to create and categorize questions. Questions created with the system can then be included in exams. Exams can be randomized and printed, along with their answer sheets. This functionality allows faculty members to focus on the content of the exam instead of the layout. Exams can then be shared with other faculty members or with students. Students have the ability to log into the application and take a shared exam.

2. Life cycle model

Many development models exist for developing software, each containing their own advantages and disadvantages. The development model chosen for the Exam Generator was an iterative development model. A synopsis of this model is described below.

The iterative development model defines a process of developing an application in a cyclical fashion. Such an engineering cycle starts with the developer gathering a subset of the user requirements from the customer. The requirement subset is then designed, coded, and tested. After testing is complete for an iteration, the developer starts the next iteration. This process continues until the project has been fully developed. Following this approach allows the developer to spot potential design and coding weaknesses earlier.

If the application being developed is GUI intensive, the iterative development model may be extended to include a demo. Within the demo, a prototype of the application is shown to the customer. Providing a demo allows the customer to see progress of the application and allows the customer request changes in the GUI design or functionality in earlier stages of development.

One concern for the iterative development model is that the developer does not have a complete understanding of all requirements before starting the design. Major design decisions made in an earlier iteration may not be valid in future iterations, causing modification to design and code. To offset this concern, the developer modified the iterative development model to include gathering all requirements for the application within the first stage of the development process. Gathering all of the requirements before beginning the development iterations allowed the developer to gain oversight on all that would be required in the application. This encompassing view would help the developer make appropriate design decisions and lower any risk of design and code rewrites.

The diagram in Figure 1 illustrates the iterative development model used for developing the Exam Generator.

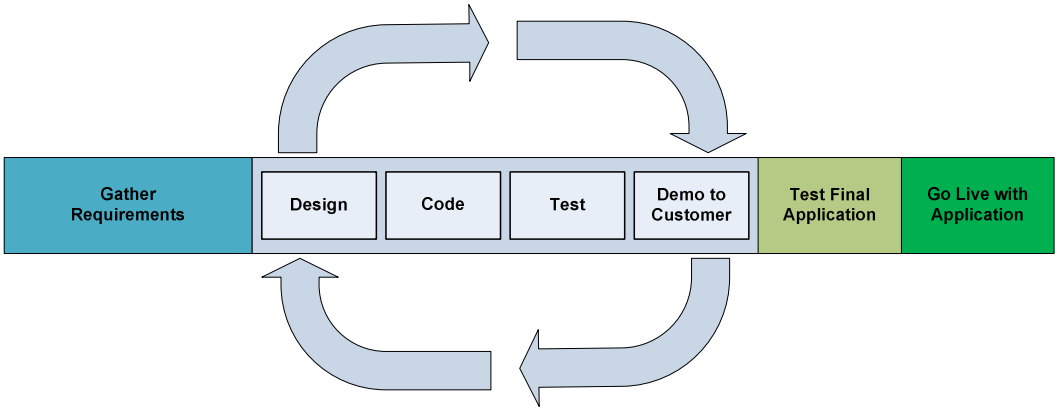

Figure 1: Exam Generator Development Model

3. Requirements Phase

The gathering requirements phase consisted of the customer and the developer discussing the functionality of the Exam Generator. The customer for the Exam Generator was an UW-L Faculty Member named Dr. Kenny Hunt. Discussions of the requirements were performed through email and face-to-face meetings between the customer and the developer. This occurred throughout the Spring of 2008. A total of thirteen face-to-face meetings were held which lasted thirty minutes to an hour each meeting. A total twenty emails that were exchanged.

3.1. Requirements Document

A segment of the requirements document can be seen within Figure 2. Each functional requirement is defined with the following fields.

- an index (a unique code for the functional requirement)
- a descriptive name
- a purpose (a short description of the requirement)
- input parameters
- output parameters
- actions (a set of tasks or activities that must be performed in order to satisfy this requirement)
- exceptions (set of conditions that indicate a situation in which the function will stop)
- remarks that explain more about the functionality
- cross-references to other functional requirements that are related to the current function

3.3.1 Create User

Index: A-Create User
Name: Create User
Purpose: Allows an Administrator to create a user account.
Input parameters:

- User Name
- Account Type
- First Name
- Last Name
- Email Address
- Password
- Confirm Password

Actions:

- Ensure user is logged in and has account type of Administrator
- Ensure all fields contain valid data.
- Ensure user name is not already used within the application.
- Ensure password meets minimal requirements
- Ensure password and confirm password match.
- Have system generate a unique User ID for the user.
- Add the user to the application.

Output parameters:

- None

Exceptions:

- User is not logged in or does not have account type of Administrator.
- One or more fields do not contain valid data.
- User name is already used within the application.
- Password does not meet minimal requirements.
- Password and confirm password do not match.

Remarks: None
Cross-references: None

Figure 2: Section of Exam Generator Requirements Document

The completed requirements document contains a total of thirty-nine requirements that follow the same format as shown in Figure 2. Thirty-four requirements define the functionality of the application, while the remaining four define the nonfunctional requirements. Three Use Case diagrams were included in the requirements document to augment the requirements.

Nonfunctional requirements within the requirements document did not follow any specific format. The four nonfunctional requirements discussed in the requirements document were performance, safety, security, and software quality. Some bullet points from each section are listed below.

Performance

- The application must perform well at all times.
- Response time must be must remain under a minute for all transactions within the application.

Safety

- Data stored within the application will be backed up on a weekly basis in order to prevent data loss.

Security

- Users that do not have privileges to access content should not be allowed to view that content.
- Application must be coded in order to prevent any type of attack (SQL injection, XSS attack, etc) by any user.

Software Quality Attributes

- Application should be designed in a way that promotes easy maintainability.
- The design must be easy to understand and must allow for easy modification.

3.2. User Groups

The requirements for the Exam Generator rely upon three types of user groups. The three user groups, short descriptions of their requirements, and Use Case diagrams for each are described below.

Faculty Members

Faculty members have the ability to manage categories, references, questions, and exams within the system. The following is a listing of some of their functionality within the Exam Generator.

Faculty members can ...

- log in and out of the system.
- create/edit/delete categories.
- assign zero or more categories to multiple references, questions, or exams.
- create/edit/delete references.
- can add an image or text references to a question
- create/edit/delete multiple choice and true false questions.
- create/edit/delete exams using questions stored in the application.
- automatically generate an exam using user-accessible questions.
- randomize the questions of an exam.
- generate the exam and the exam's answer sheet as a PDF.

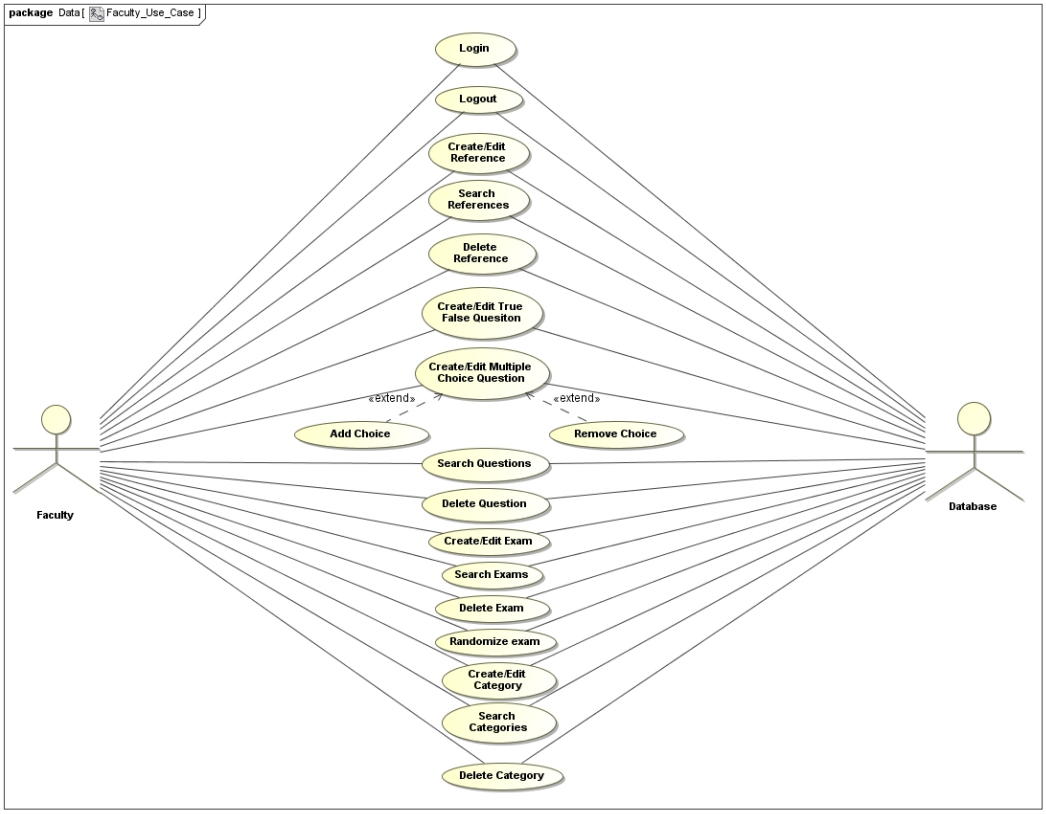

Figure 3: Faculty Use Case Diagram

Students

Functionality for the student users is limited to searching for exams, randomly generating exams, and taking an exam online as a practice exam. The following is a listing of some of the student's functionality.

Students can...

- log in and out of the system.
- search for exams shared with the student.
- generate exams using student-accessible questions.
- can take online practice exams. Students receive a summary after completing online exams; such a summary displays the exam score and identifies which questions have been answered correctly and incorrectly.

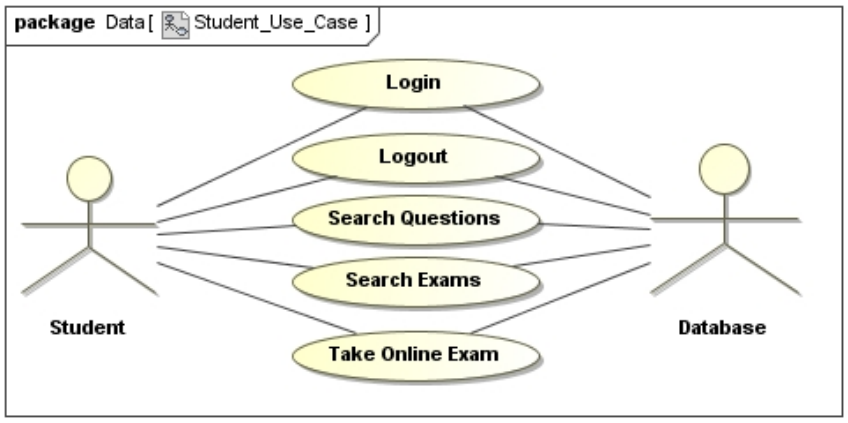

Figure 4: Student Use Case Diagram

Administrator

Administrators manage all user accounts. The following is a listing of an administrator's capabilities.

Administers can...

- log in and out of the system.
- create/edit/delete/search system users.

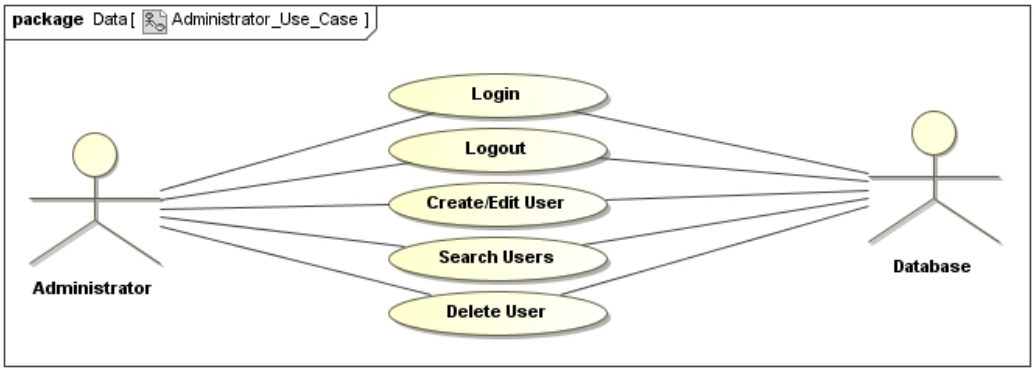

Figure 5: Administrator Use Case Diagram

4. Development Phase

4.1. Development Tools

Research on tools and packages used for building web applications was performed before starting the design. Familiarity with the Java programming language led the developer to build the application in Java. Doing so allowed the developer to focus on the application itself and less on learning a new programming language. Ease of future maintenance is an additional benefit to Java.

As requested by the customer, the Exam Generator was designed as a web-based application capable of being hosted on a Tomcat web server. At the time of this paper many application frameworks existed for developing web applications in Java, all of which had strengths and weaknesses. One of these web applications was the Seam Framework, which was developed by the JBoss team. Seam Framework has features that aided the developer while creating the Exam Generator. Below are some of the features that led the developer to choose Seam for this project.

- The Seam Framework contained libraries that were configured to work well with each other. The inclusion of the iText PDF Generation library was useful for producing exams in PDF format. The inclusion of JPA and Hibernate was as a database. The RichFaces library provided many of the GUI components used by the application's frontend.
- The Seam Framework includes many custom classes that reduced redundant code that is normally required when developing a web based application. Some include:
    - The EntityHome class used for CRUD operations on entities found in the database using a persistence manager.

    - Classes providing compression and resizing of JPEG and GIF images, used for storing and displaying image references within the GUI as well as generated PDF's.
    - Classes providing data encryption used for encrypting user's passwords stored within the database.
- The Seam Framework applications include tools for authentication.
- The JBoss team developed a set of plugins for Eclipse IDE that allow easy creation and editing of applications using Seam Framework. Most helpful is the Seam Generator which generates an Eclipse project containing all of the files and configurations required for a basic Seam Framework web application.
- The Seam Framework provides support for deploying the application to any Java Application Web Server.
- The Seam Framework includes many examples demonstrating how to properly use the framework.
- The Seam Framework's home web page includes a managed forum where developers could post questions about the framework and receive support from the creators and power users of the framework.

4.2. Design Document

At the start of each development iteration the developer created and updated a design document for the Exam Generator. The design document contains object-oriented design for what was implemented in previous iterations as well as what was to be implemented within the subsequent iteration. The design document includes the entity and session designs (described as class diagrams and class definitions) and a Database Design (described as an Entity Relationship Diagram). These diagrams and key features in each design will be discussed in the following sections of this paper.

In addition to the entity and session diagrams each design was described using a collection of class definitions. Class definitions describe the structural and behavioral properties for a class. Within the final development iteration of the Exam Generator the design document contained a total of forty-six of these class definitions.

Figure 5 contains the class definition for the Authenticator class. The class definition contains a class box and a listing of methods. The box contains the class' name, attributes, purpose, and any additional notes that would be helpful to the developer. Following the class box is a listing of methods (minus the generic setter and getter methods for class attributes) contained within the class. To ensure that the figure fits on one page only the authenticate method is shown from the Authenticator class.

Class name: Authenticator

Attributes:

private	User	currentUser	/** Current User logged into the system.**/
private	Identity	identity	/** Seam class that stores a login status **/
private	Credentials	credentials	/** Seam class that provides the login credentials when authenticating. **/
private	EntityManager	entityManager	/** Class that provides access to the db.
private	boolean	conversationStarted	/** Boolean flag representing if a conversation has been started. **/

Purpose:
Aids in the Authentication of users in and out of the system.

Notes:
- currentUser, identity, and credentials are all injected into this class from the Session class.
- conversationStarted is used to keep information about the logged in users conversation with the application.
- If the conversation times out, the conversationStarted boolean will be switched. This is used so if the main page is refreshed manually by the user, a new conversation is not started.

** **Getter method only provided for conversationStarted boolean.** **

Methods:

3.4.2.1 authenticate

Name: authenticate
Synopsis: void <- authenticate()
Purpose: Allows a user to log into the system.
Visibility: public
Input parameters: None
Output parameter: None
Local variables: None
Pseudocode:

```
// Retrieve the credentials of the login from credentials object.
// Using entityManager, ensure there exists a user stored within the database that matches the
//     credentials provided within the credentials object.
// If authentication successful, raise event that notifies the controller that successful login has
//     occurred and mark the conversationStarted boolean to true. Otherwise throw error.
```

Exceptions:
- Username and/or password does not match an entry within the database

Remarks:
- None

Figure 6: Authenticator Class Definition Example

4.3. Core Design

The design for the Exam Generator can be divided into essentially three parts: the Entity Layer Design, Database Design, and Session Layer Design. Sections 4.3.1 through 4.3.3 describe what is incorporated within the each design as well as each of the design's key features.

4.3.1. Entity Layer Design

Entity layer classes are Java classes that supply the structure of all the objects in the Exam Generator. Figure 7 shows the entity class diagram for Exam Generator. Described in this section are some of the decisions made by the developer while creating the structure of the entity layer classes.

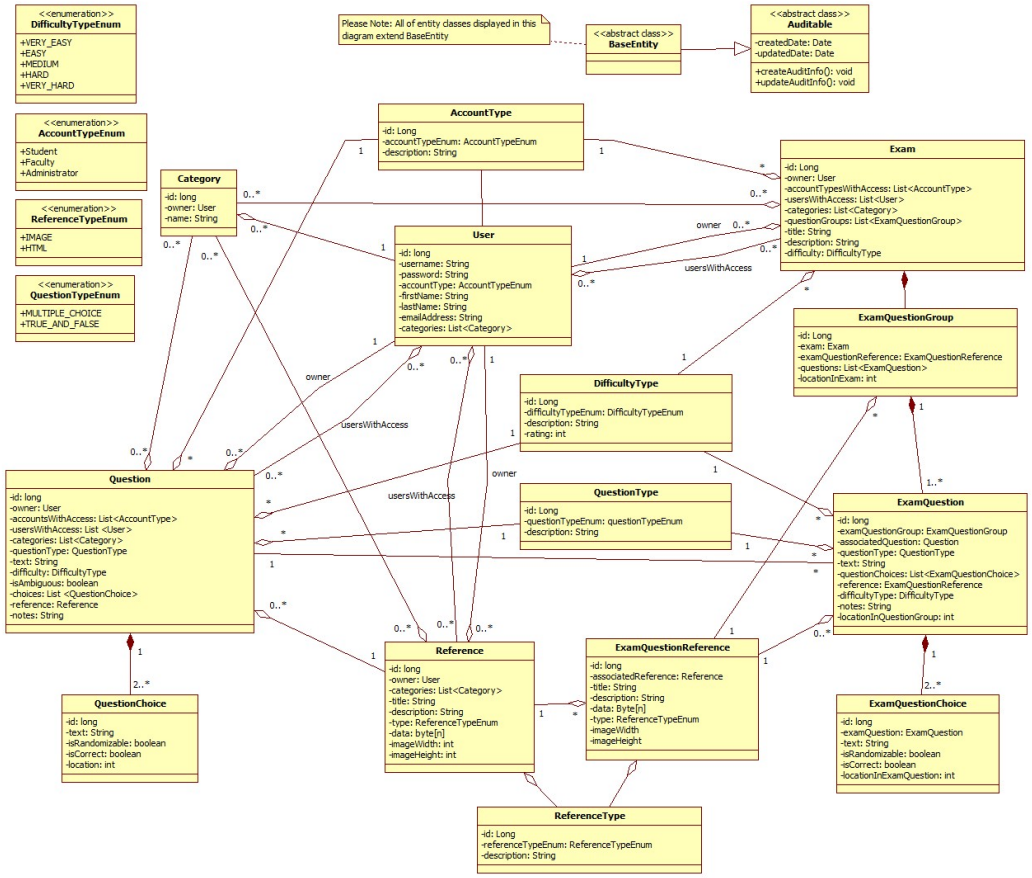

Figure 7: Final Entity Class Diagram

Within Figure 7, it can be seen that classes Question, Choice, and Reference are not directly associated with an Exam. Question, QuestionChoice, and Reference associate with classes ExamQuestion, ExamChoice, and ExamQuestionReference. ExamQuestion, ExamQuestionChoice, and ExamQuestionReference are then directly associated with an Exam. Creating the class association in this manner allowed an Exam object to contain copies of Question, Choice, and Reference objects. Since the Exam object only maintains a copy of the Question, Choice, and Reference objects, the owner can make changes to them without the changes being reflected on the Exam. If the owner of the Exam wants an ExamQuestion to reflect the new changes, the user must manually update the exam.

One of the requirements stated by the customer is that when an exam is viewed by a user, questions can be grouped together based on their reference. The ExamQuestionGroup class was created to fulfill this requirement. With this design an Exam can contain a list of ExamQuestionGroups. Each ExamQuestionGroup contains both ExamReference that all the ExamQuestions refer to as well as a list of ExamQuestions that belong to that group. If an ExamQuestion did not refer to an ExamReference, then that ExamQuestion would be the only ExamQuestion found in that group. This solution made the complexity of grouping questions relatively simple. In addition, the solution worked well with Seam's use of RichFaces to display content onto the webpage, allowing no need to create a custom view for displaying the data to the user.

While developing the application, the developer discovered that saving creation and update dates is beneficial for application maintenance, as well as providing more advanced search queries within future enhancements of the application. The creation of two abstract classes named Auditable and BaseEntity provided these fields for all entities. The Auditable class contains two date fields - one for when the entity is created and one for when it is updated. BaseEntity was created as a parent to all entity classes. BaseEntity extends Auditable, providing all entity classes that extend BaseEntity date fields. This approach also provides a means of adding fields to all entities in future iterations.

4.3.2. Database Design

The database design is for data within the Exam Generator. Figure 8 shows the database design for the Exam Generator using an entity relationship (ER) diagram.

If the reader compares Figure 8 with the class diagram found in Figure 7, it can be seen that each entity class (minus the enumeration and abstract classes) contain their own table within the database. Tables shown within Figure 8 that do not have an underscore inside of their name can be directly mapped to an entity layer

class from Figure 7. The word "mapped" denotes that each value within an entity class can be found as a column within a database table. For example, the Exam class within Figure 7 can be mapped to the exam table within figure 8.

Tables shown within figure 8 that have an underscore within their name represent one-to-many or many-to-many relationships between entities. For example, within the entity layer class diagram the Question class contains a list of Category classes. The question category table represents this association within the database. The question_category table contains a row for each question category association that exists. If a Category object is added to the categories list inside of a Question object, then a new row will be added to the question_category table with the category id and question id that has the relationship. Similarly, if the category is removed from the list, then that row within the question_category table is then removed.

Designing the database so that it resembles the Entity class structure has two main benefits. The first benefit was to have the Entity Layer and Database layer contain a relational mapping between each other. The classes found in Figure 7 each have a Hibernate or JPA annotation that maps them to the appropriate database tables. This relational mapping is then used by the entity manager within the session layer of the Exam Generator. More of how entity manager is used is described in the Session Layer Design section of this paper.

The second benefit for designing the database so that it resembles the Entity class structure is that it keeps the database normalized. Normalization within a database means removing redundant data from within the database and making sure that the dependencies within the data make sense. Normalization allows the database to be better managed, while providing efficient performance for searching, sorting, and creating indexes. This design keeps the database normalized by removing duplicate columns within the same table, grouping related data into its own separate tables, and adding a primary key to every table within the database.

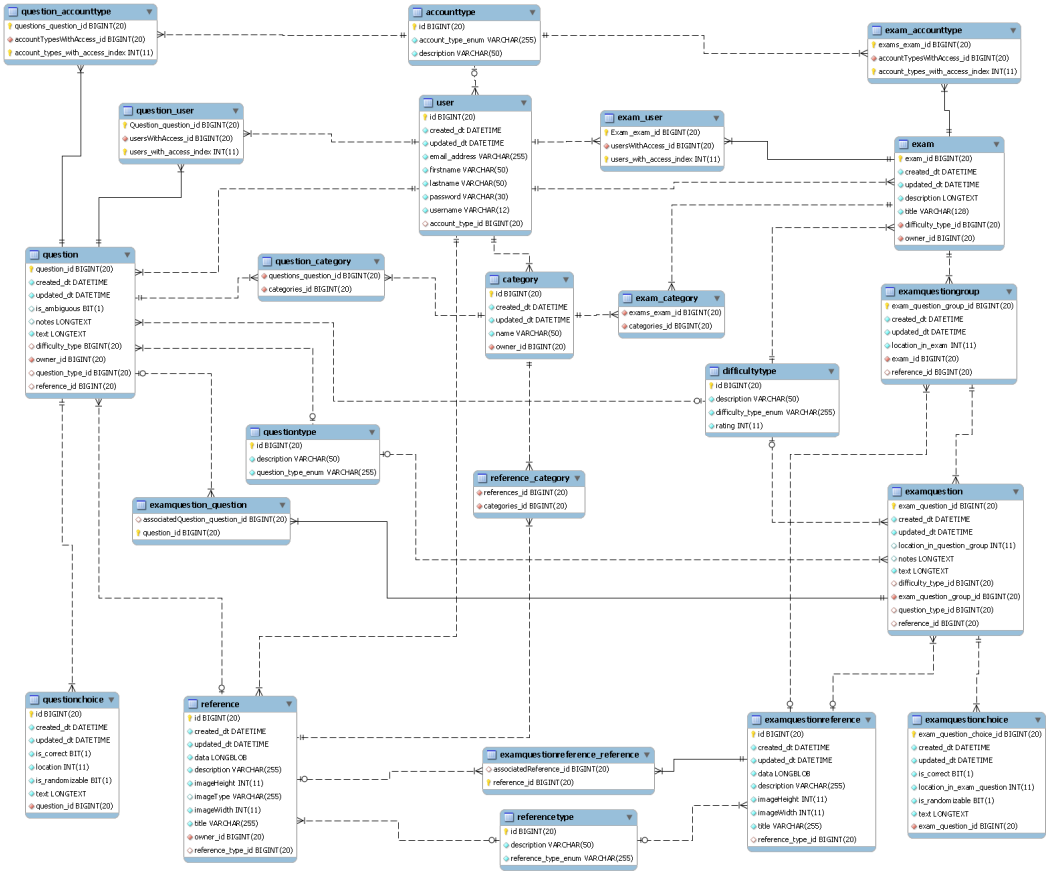

Figure 8: Final ER Diagram

4.3.3. Session Layer Design

Session classes provide functionality to the application. These classes manage both the basic functionality that a user finds in typical web applications, as well as the functionality that is unique to the Exam Generator. Figure 9 shows the session layer class diagram for the Exam Generator. This section will discuss the following features of the Exam Generator and how they are implemented using the classes presented in Figure 9.

- Database Interaction

- Page Navigation
- Randomizing an Exam
- Randomly Creating an Exam

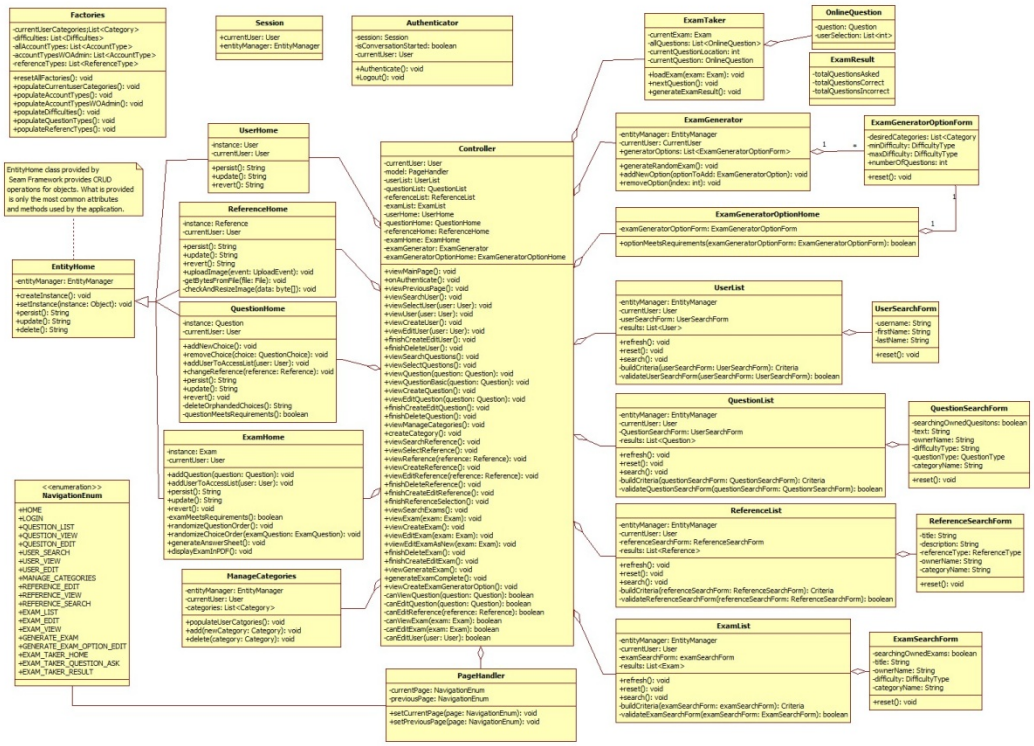

Figure 9: Final Session Class Diagram

Database Interaction

The Exam Generator interacts with the database through the use of a persistence manager. A persistence manager is an API that is used for performing core database operations. As mentioned in section 4.2.3, the persistence manager uses Hibernate and JPA annotations placed within the entity classes to understand what classes are entity classes and which database tables the entity classes are mapped to. By using the persistence manager, the notion of a database does not exist within the application. The application only contains methods for saving and retrieving objects from the entity manager class within the system.

The major benefit to using the entity manager for accessing the database was that all SQL code was handled by the entity manager. The developer was able to spend less time debugging SQL and more time writing Java code. An added benefit was the additional focus on security. The entity manager took care of qualifying the data that was inserted inside of SQL statements that was generated, removing any chances of accidental or malicious SQL injection.

The classes UserList, QuestionList, ReferenceList, and ExamList within Figure 9 are the classes that provide the functionality of searching the database for objects. Each of these classes contains their own reference to the entity manager. The classes that provide searching functionality can be seen in more detail within Figure 10.

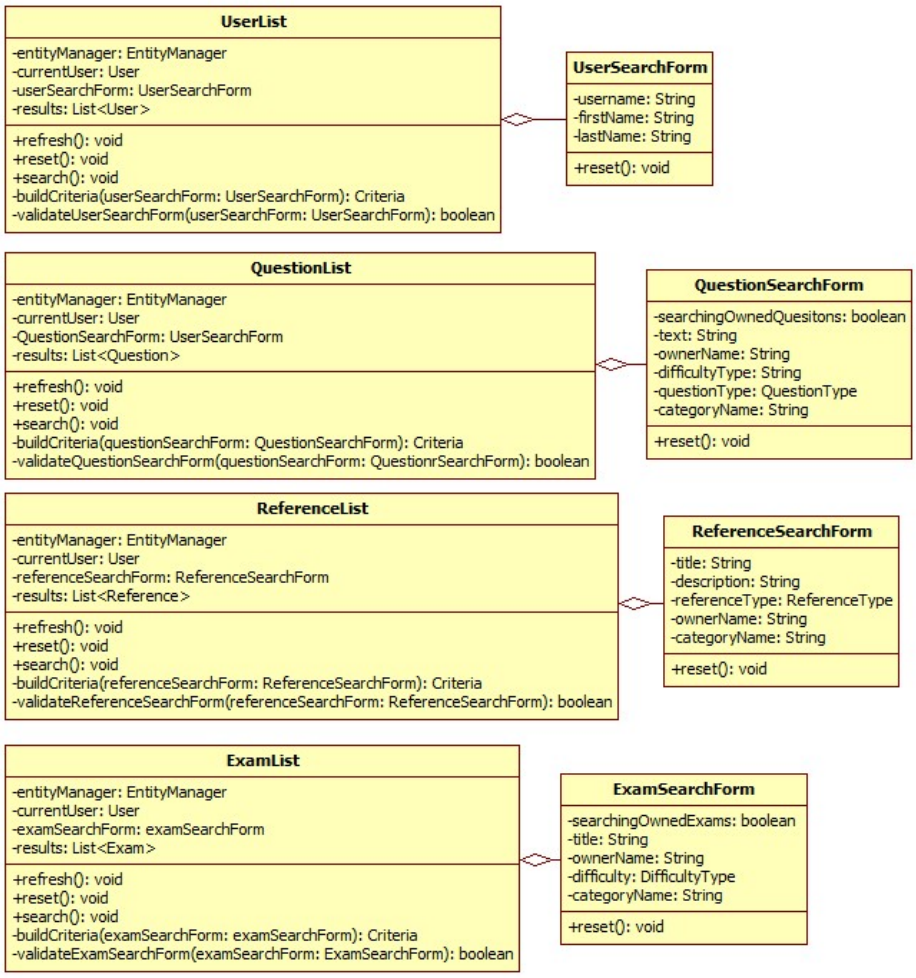

Figure 10 : Portion of Session Layer Class Diagram - Classes That Provide Search Functionality

When a user wants to search for an entity, the user provides input parameters to the search. These search values are then stored within a form object. (All of the classes that end in __Form are form classes.) When the *submit* button on the search page is pressed, the search criteria is then directed to the entity manager reference. The entity manager then generates the SQL, performs the query, and returns any results back to the user.

For viewing and managing entities, the Seam Framework provides a class named the EntityHome class to aid the developer. The EntityHome class is a

generic class that comes packaged with the Spring Framework. EntityHome class manages entity instances by caching them and providing Create, Read, Update, and Delete (CRUD) operations on them with the persistence manager [1]. The classes UserHome, ReferenceHome, QuestionHome, and ExamHome all extend the EntityHome class. By using this, the developer did not need to worry about caching any of the data manually. The classes that provide CRUD operations on entities can be seen in more detail within Figure 11.

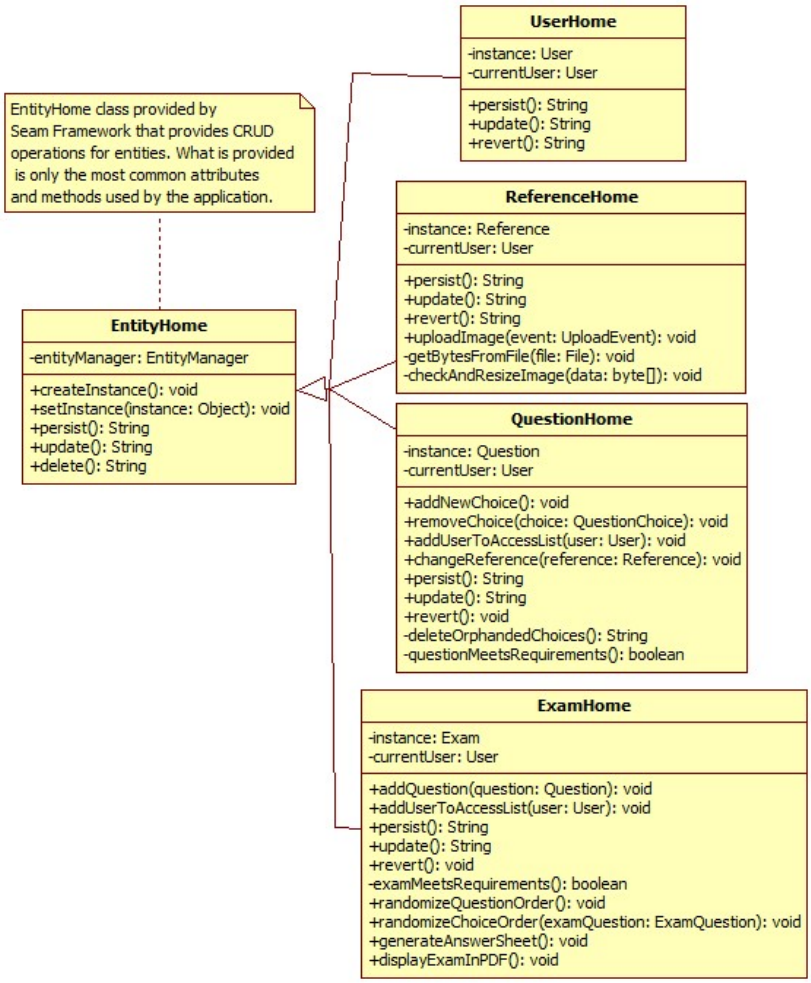

Figure 11: Portion of Session Layer Class Diagram - Classes That Provide CRUD Operations on Entities

Page Navigation

The developer designed the Exam Generator so that one class would handle all page navigation. This class is named the Controller class.

The Controller class contains a method for each user-accessible page. The Controller class also contains a PageHandler class and the NavagationEnum class. The PageHandler class maintains the page as it is displayed. The NavagationEnum class contains enumerations that correspond to pages that can be displayed to the user. When the user wants to navigate to a different page, the corresponding method within the Controller class is called. The method first validates user authorization for the page. If authorization is confirmed, the Controller class changes the currentPage value within the PageHandler class to the appropriate page enumeration. The browser is then refreshed and the new content is displayed to the user. The classes that provide page navigation can be seen in more detail within Figure 12.

Figure 12: Portion of Session Layer Class Diagram - Classes That Controls Page Navigation

This approach for page navigation provides the developer more control in prohibiting invalid users from accessing restricted pages which helps the Exam Generator be less acceptable to security attacks . The disadvantage is that if more pages are added to the Exam Generator, the Controller class continues to grow.

Randomizing Examination Questions

One of the requirements of the Exam Generator is that a faculty member must have the ability to randomize the order of examination questions, allowing a faculty member to conveniently create multiple versions of the same exam.

The Entity Class Diagram in Figure 7 points out the following structure:

- Exam contains a list of ExamQuestionGroup
- ExamQuestionGroup contains a list of ExamQuestion
- ExamQuestion contains a list of ExamQuestionChoice

To provide randomization for an Exam, three methods were created. Those methods were randomizeExamQuestion (which randomly reorders the list of ExamQuestionChoice), randomizeExamQuestionGroup (which randomly reorders the list of ExamQuestion), and randomizeExam (which reorders the list of ExamQuestionGroup). The randomizeExam method is called to randomize an Exam. This method in turn calls the shuffle method from java.util.Collections. After the list order shuffled randomizeExamQuestionGroup is applied to each ExamQuestionGroup within the ExamQuestionGroup list. Similarly, the randomizeQuestionGroup method first shuffles the list of ExamQuestion and then calls randomizeQuestion on each question within the list. When randomizing an ExamQuestion, only the choices marked as *randomizable* are shuffled. To ensure nonrandomizable choices are not shuffled, the choices that are flagged as *randomizable* are copied into a new list. The new list is then randomized using the same previously-mentioned shuffle method. The shuffled choices are then copied back to the original list, replacing the randomizable choices.

Randomly Creating an Exam

Randomly creating exams follows an approach that resembles searching for other entities within the Exam Generator. An exam is randomized by first first searching for questions and then displaying the result as an Exam.

Searching for questions in the generated exam is accomplished by creating a list of search queries. Each search query finds a subset of questions that the user would like to see in the exam.. The ExamGeneratorOptionForm holds the search criteria for one search query. The list of ExamGeneratorOptionForm is stored as a value named generatorOptions of the ExamGenerator class. These classes and values can be viewed in more detail within Figure 13.

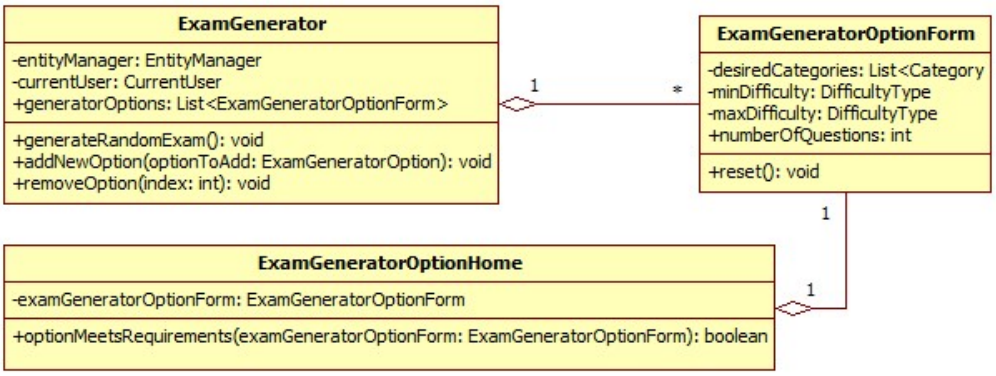

Figure 13: Portion of Session Layer Class Diagram - Classes That Generate Exams

When the list of search criteria is created, the generateRandomExam method within the ExamGenerator class is called. When called, the method iterates over each ExamGeneratorOptionForm within the generatorOptions list, using the entity manager to create the search query, run the search query, and add the results of the search query to one big list of questions that serves as the combined result list. To simplify things, each search query returns a limited number of results. The search query also contains a shuffle method that shuffles the results into a random order.

When all of the search queries are completed, the system calls the createInstance method to create a new Exam object. Each item of the result list is then processed by addQuestion. The addQuestionMethod creates the question as an ExamQuestion and adds the new ExamQuestion to the appropriate ExamQuestionGroup, creating a new group if one does not already exist. When all questions have been added to the exam, the Controller class is then notified that a new Exam has been created. This notification causes the current page to change. The user can then edit and/or save the new exam.

4.4. User Interface

The Exam Generator user interface was developed along with the core functionality of the application. Using xhtml pages that contained RichFaces components, the developer was able to effectively display pages. Each prototype demonstrated contained the user interface for the functionality designed for that iteration.

Figure 14 depicts the user interface of the faculty search questions page. This figure gives includes the key parts of the page:

- a menu bar that contains links to different pages available to the user
- a link to the user's profile
- a link to log off from the system
- a page title that identifies the page being displayed

The buttons and links within the page state the actions available to the user. This layout allows the user to know exactly what page he or she is currently on while providing both a simple way to navigating from one portion of the application to another. Providing this type of layout helps the application to be easy to navigate and user friendly.

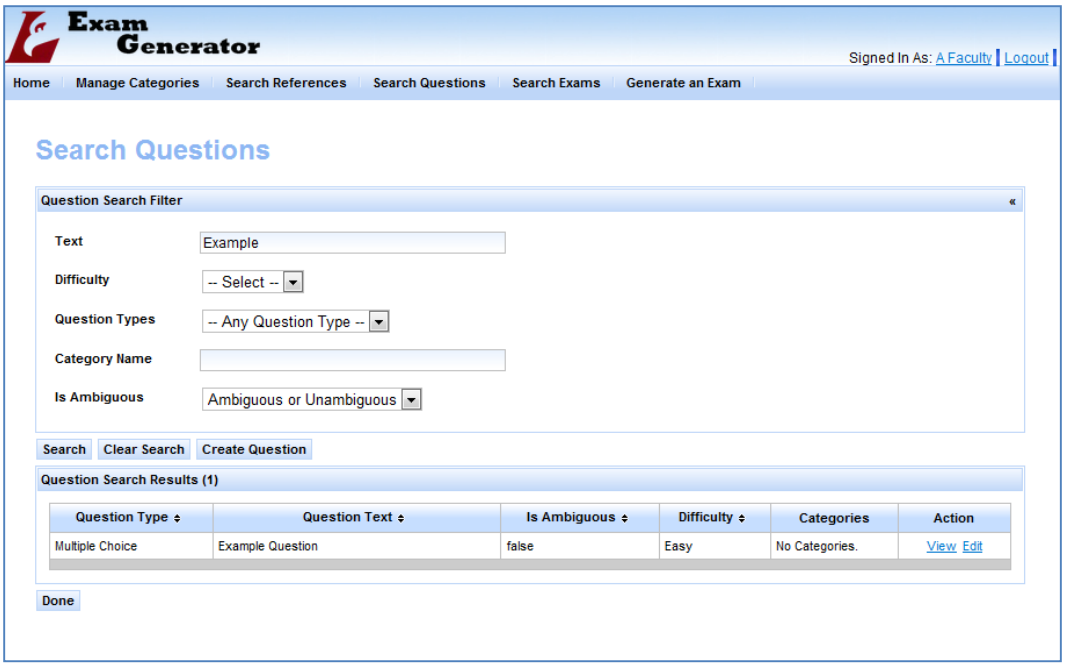

Figure 14: Search Questions Page Print Screen

5. Testing

A dynamic test plan was followed for testing the Exam Generator. The first test plan was created after completing the coding section within the first development iteration. The test plan was then expanded upon future development iterations, adding tests for new functionality. Following this approach allowed the developer to ensure that new functionality was properly tested and that functionality added in previous iterations had not been altered. If an issue was spotted while following the test plan, testing would be stopped and the issue would be fixed. When the issue had been resolved, testing would be restarted.

Figure 15 displays a segment of the test

Requirements	Purpose	Steps To Test	Expected Result
Login – Correct Credentials	Ensure that a valid user can log into the system. Ensure that proper homepage is displayed.	1. Click the login link on top right of screen 2. Enter in valid username and password 3. Click the login button.	Proper home page should be displayed, based on users Account Type. Top right of screen should show a link to the user's profile
Login – Incorrect Username	Ensure user is unable to log in if username provided is incorrect	1. Click on login link on top right of screen 2. Enter a invalid username and any string for password 3. Click the login button.	Warning message should be displayed stating the username and or password is incorrect.
Login – Incorrect Password	Ensure user is unable to log in if password provided is incorrect	1. Click on login link on top right of screen 2. Enter a valid username and an incorrect password 3. Click the login button.	Warning message should be displayed stating the username and or password is incorrect.
Logout	Ensuring that when the logout button is pressed, the faculty member is logged appropriately.	1. Ensure that faculty member is currently logged in. 2. Click on logout link in top right of screen	Application home page is displayed after logout.

Figure 15: Basic Functionality Test Plan Segment

After the developer finished walking through the test plan, the prototype was uploaded to a web server on the Internet. At this time the customer was able to access the prototype and perform his own testing. The customer's testing consisted of walking through the new functionality and ensuring that the quality and flow was as desired.

6. Continuing Work

One limitation of the Exam Generator is the inability to dynamically update questions on exams. The application was designed so that if changes are made to a question and the question is currently associated with an Exam, the exam would not reflect the changes. If the user desires to have the updated question in the exam, the user must remove the old question and then add the question with the new content. A future use case for the Exam Generator would be to notify the owner of an exam if a question within it has been updated. The owner of the exam could then proceed to view the changes made to the question. If desired, the changes to the question can then be reflected on the exam, removing the need for the owner to remove the old question and add the new one.

Another limitation of the Exam Generator is that faculty members currently do not receive any notification that a student has completed an a practice exam. A future feature may include configuring the Exam Generator to use an email server to send out an email to users. Email could report student scores and question analysis. Faculty could use this to gain a better understanding of how well the students understand the material of the class.

One more limitation of the Exam Generator is the lack of encryption between the client's web browser and the server. The lack of encryption allows for someone to intercept another user's messages that are being sent between his or her web browser and the Exam Generator application server. Once intercepted, this person can view the data that should otherwise be denied. Future work for the Exam Generator is to have the web application use SSL encryption for all data sent between the two pages. This would ensure that any messages intercepted between a user's web browser and the Exam Generator application server would not be easily understood.

7. Conclusion

The Exam Generator is a tool for creating, maintaining, and administering exams. The Exam Generator provides the means of creating easily searchable questions. These questions can then be used to manually create exams; alternately, the system can automatically generate an exam. Exams and associated answer sheets can be printed in a format that is clean and easy to understand. If so desired, questions and exams can be shared with students so that they can use the questions to take online practice exams.

In conclusion, the Exam Generator promises to be a fast, simple, and effective tool that many faculty members can use to create and maintain their exams.

8. Bibliography

1. Dan Allen, *Seam In Action*, Manning Publications Co., 2009
2. IEEE, *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications*, IEEE Computer Society, 1998

9. Appendices

9.1. Page Flow Diagrams

The following figures describe the page flow of the application.

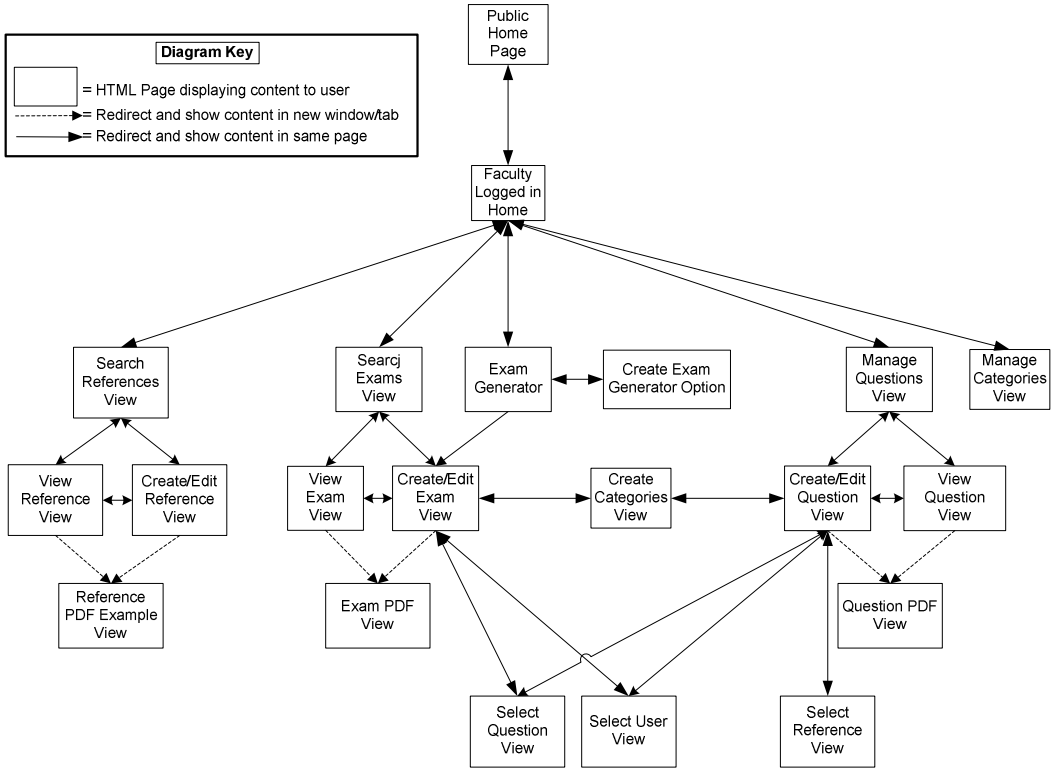

Figure 16. Faculty application navigation diagram.

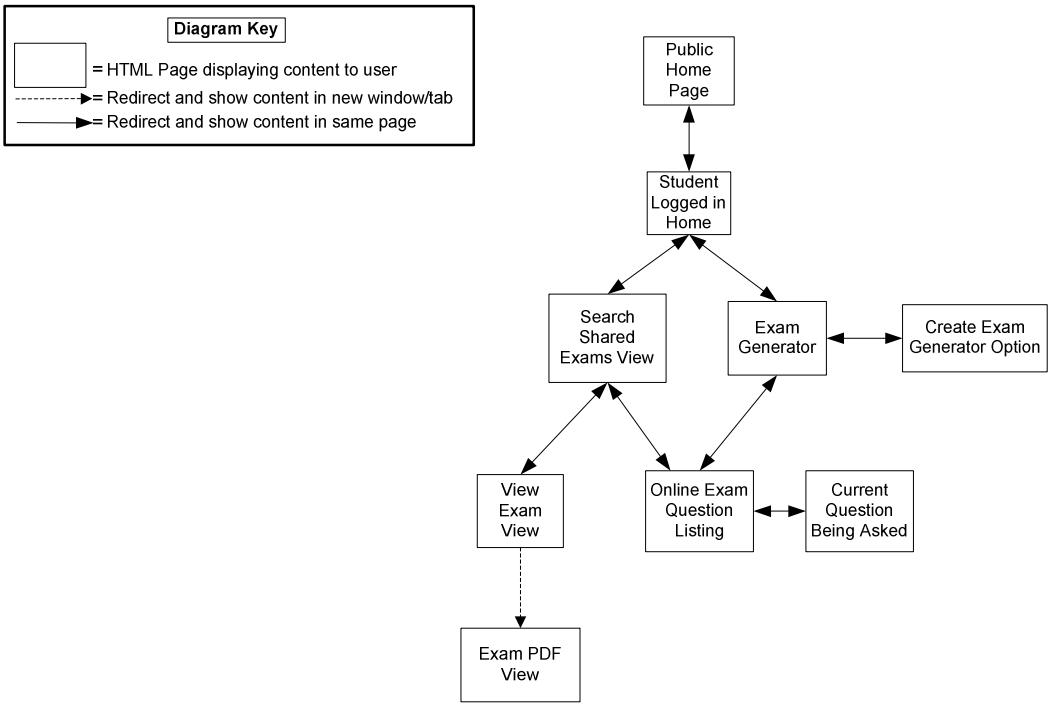

Figure 17. Student application navigation diagram.

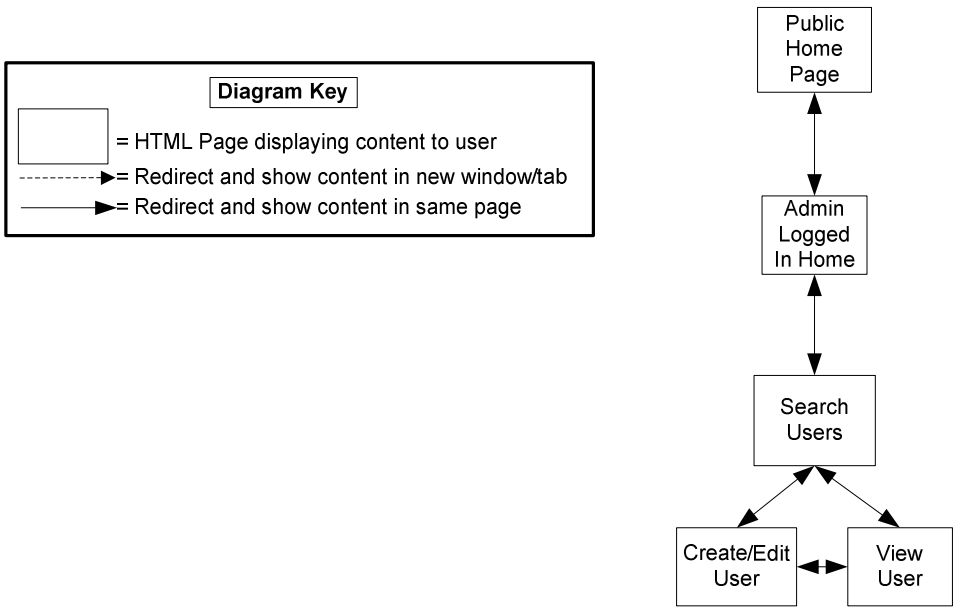

Figure 18: Administrator application navigation diagram.

9.2. Application Screen Shots

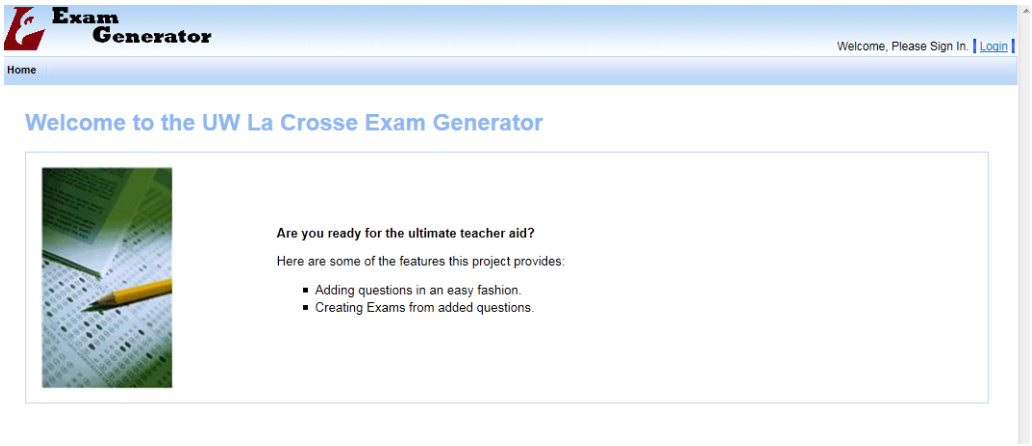

Figure 19: Application Home

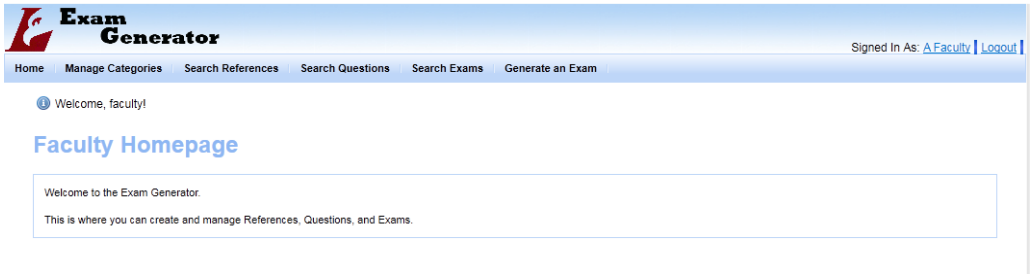

Figure 20: Faculty Home

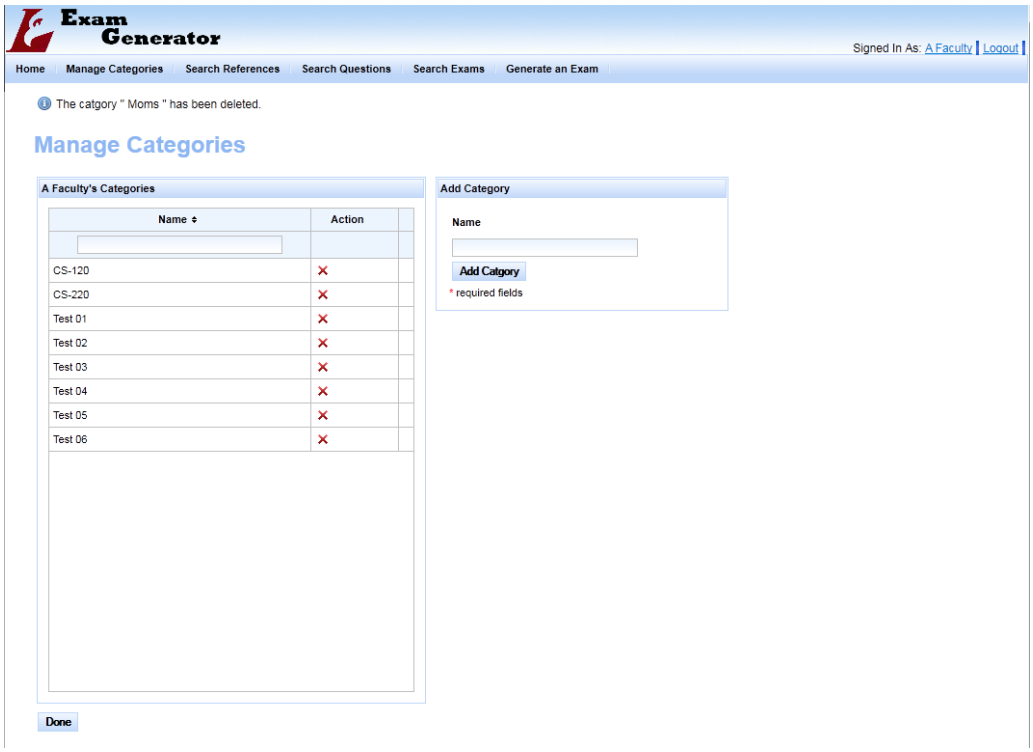

Figure 21: Faculty Manage Categories

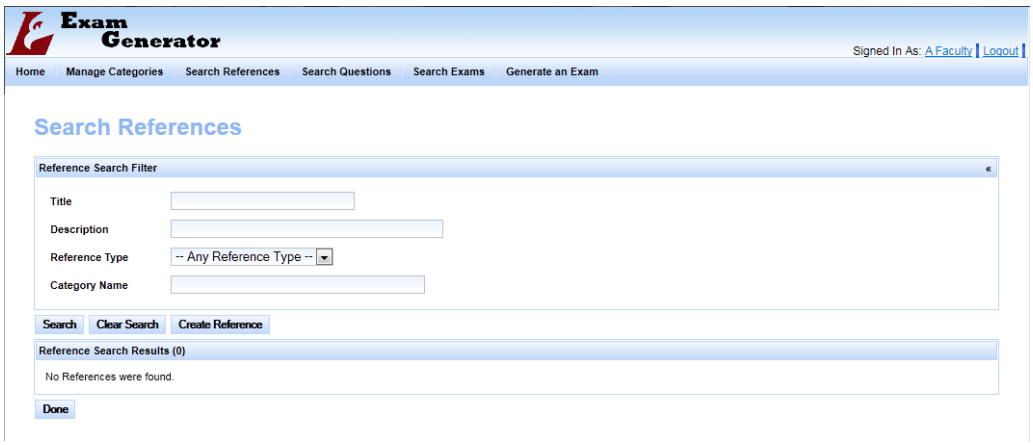

Figure 22: Faculty Search References

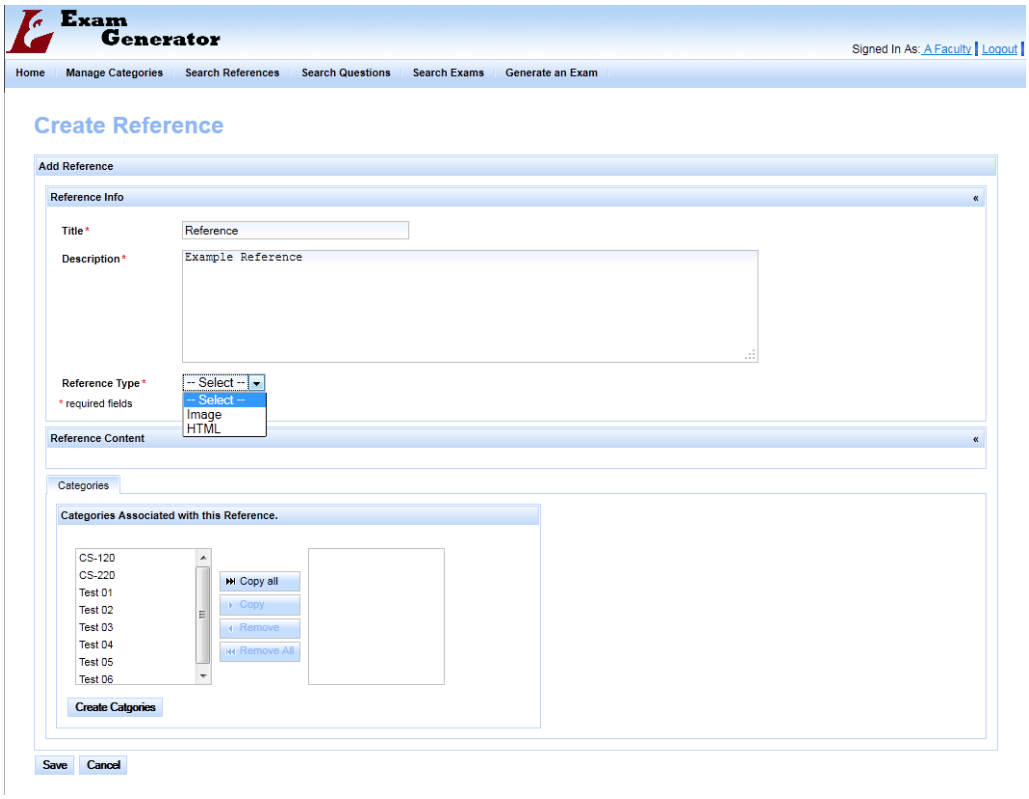

Figure 23: Faculty Create Reference

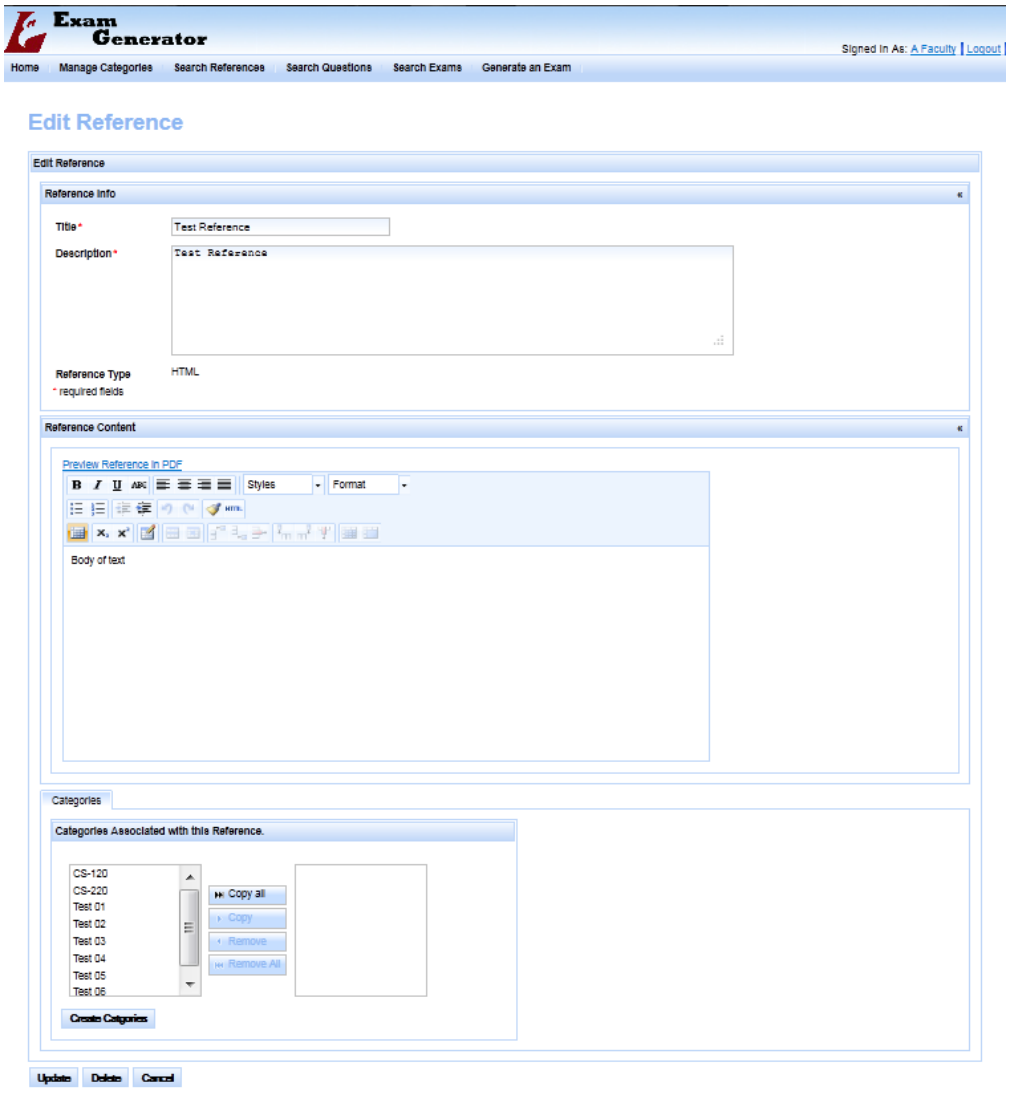

Figure 24: Faculty Edit Reference

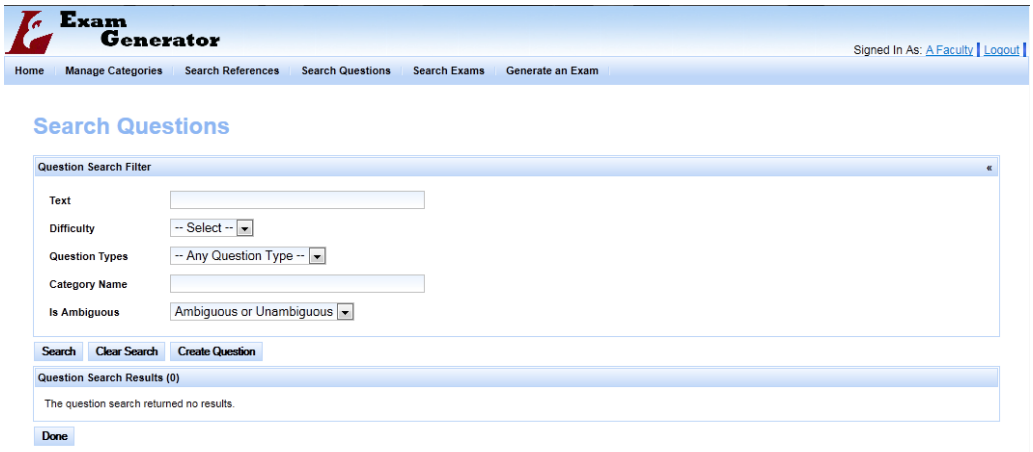

Figure 25: Faculty Search Questions

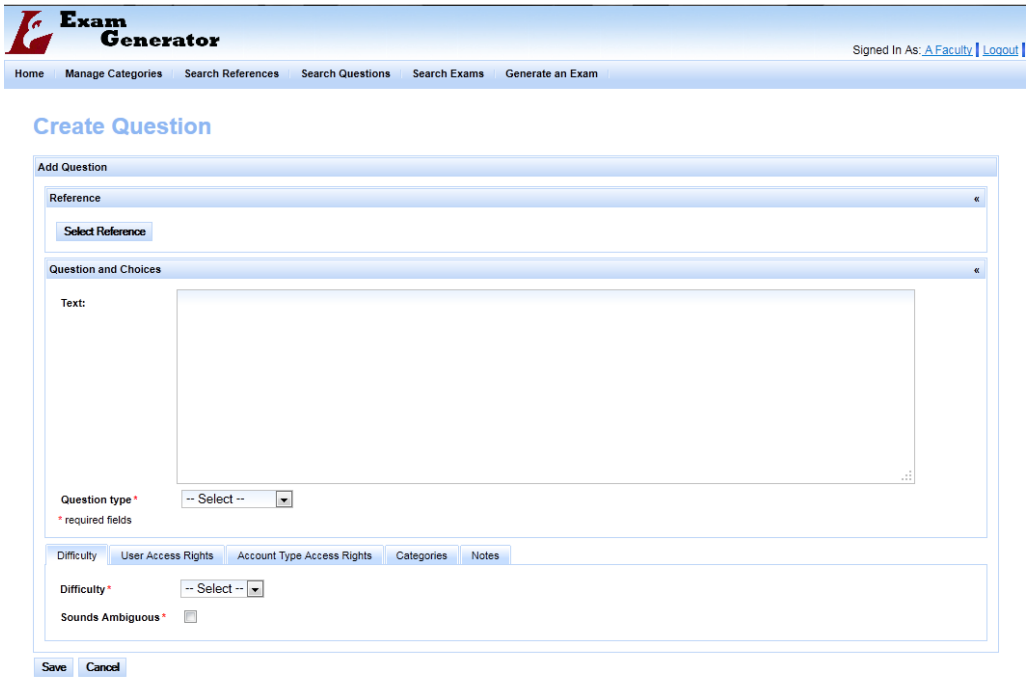

Figure 26: Faculty Create Question

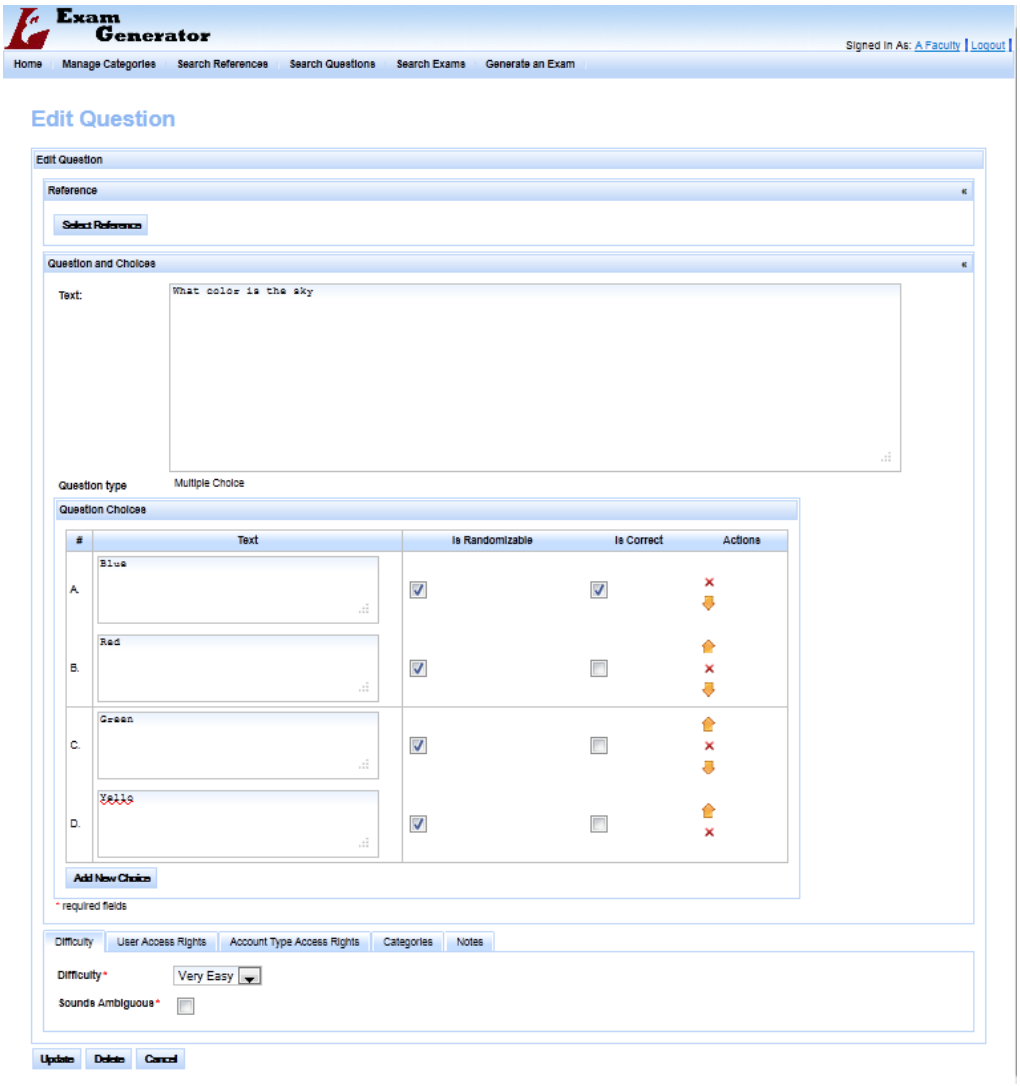

Figure 27: Faculty Edit Question

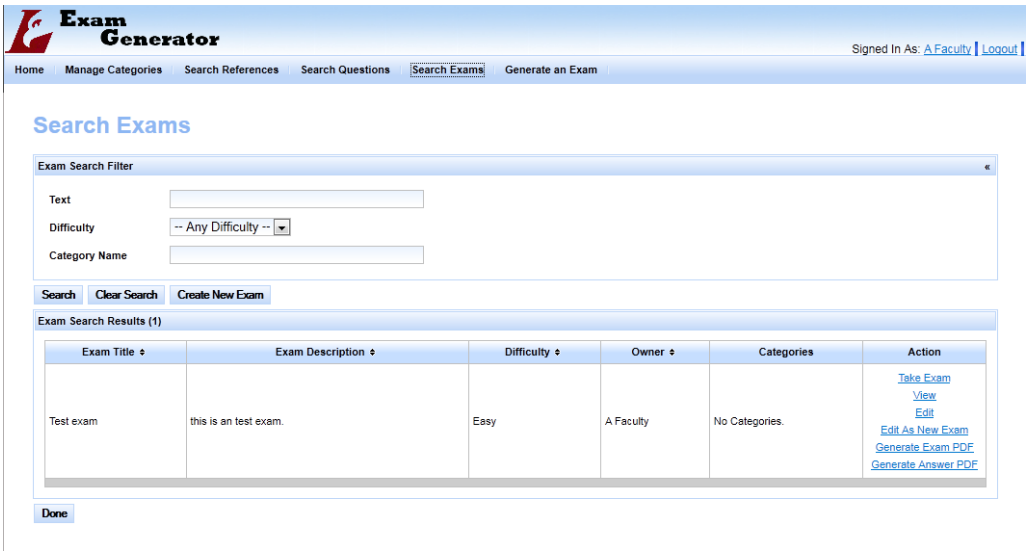

Figure 28: Faculty Search Exams

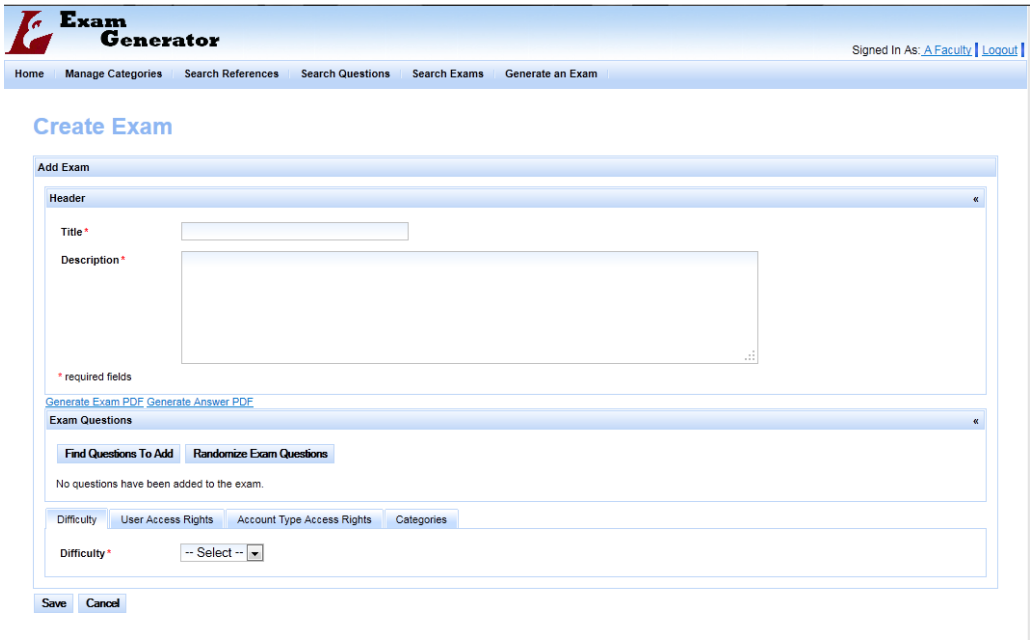

Figure 29: Faculty Created Exam

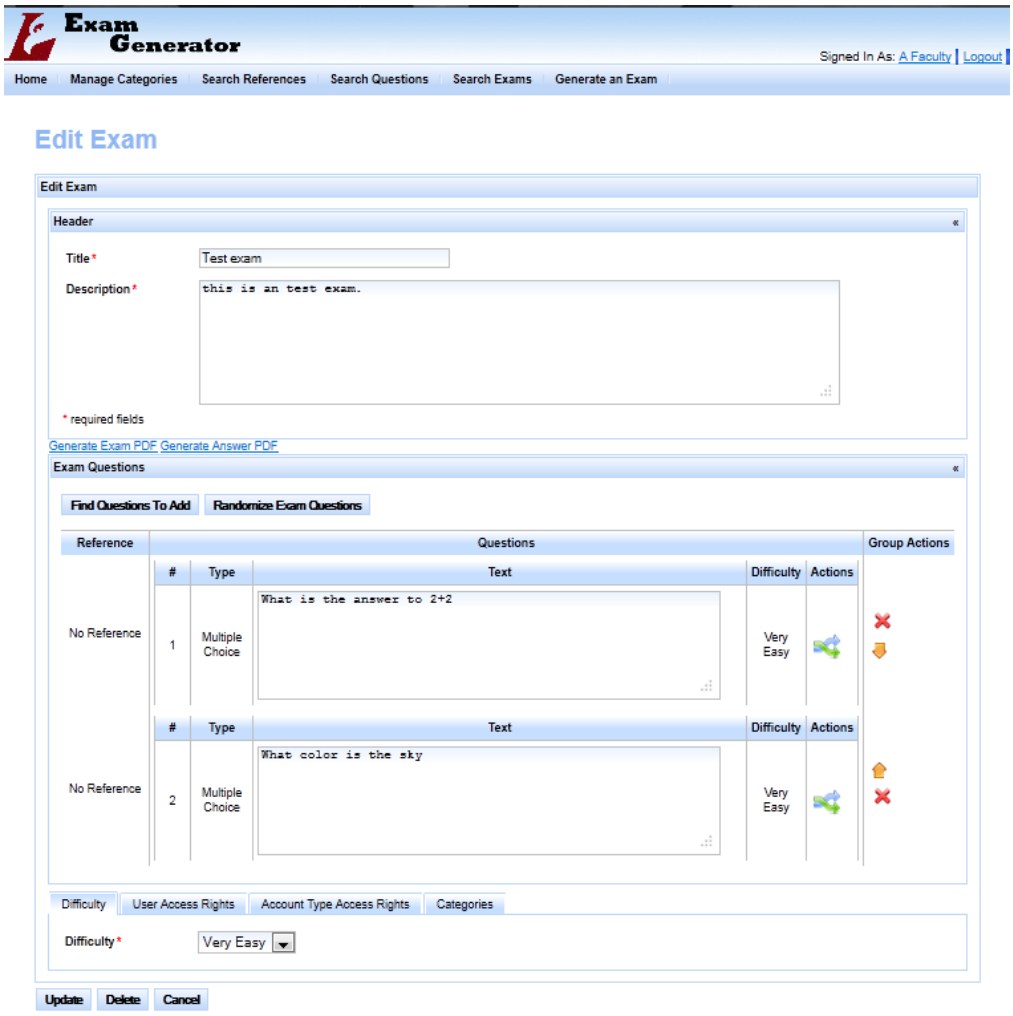

Figure 30: Faculty Edit Exam

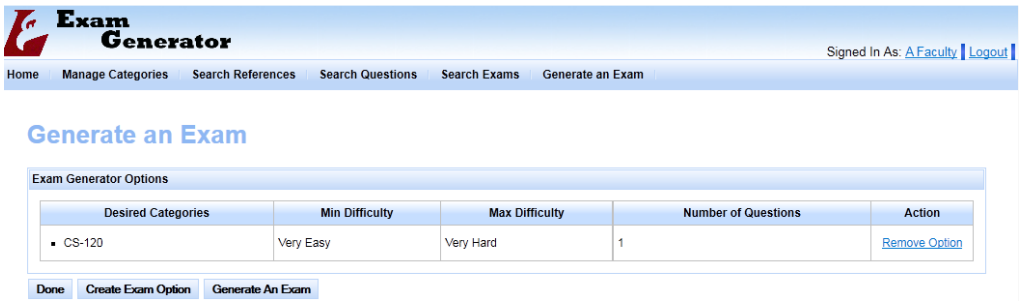

Figure 31: Faculty Generate Exam

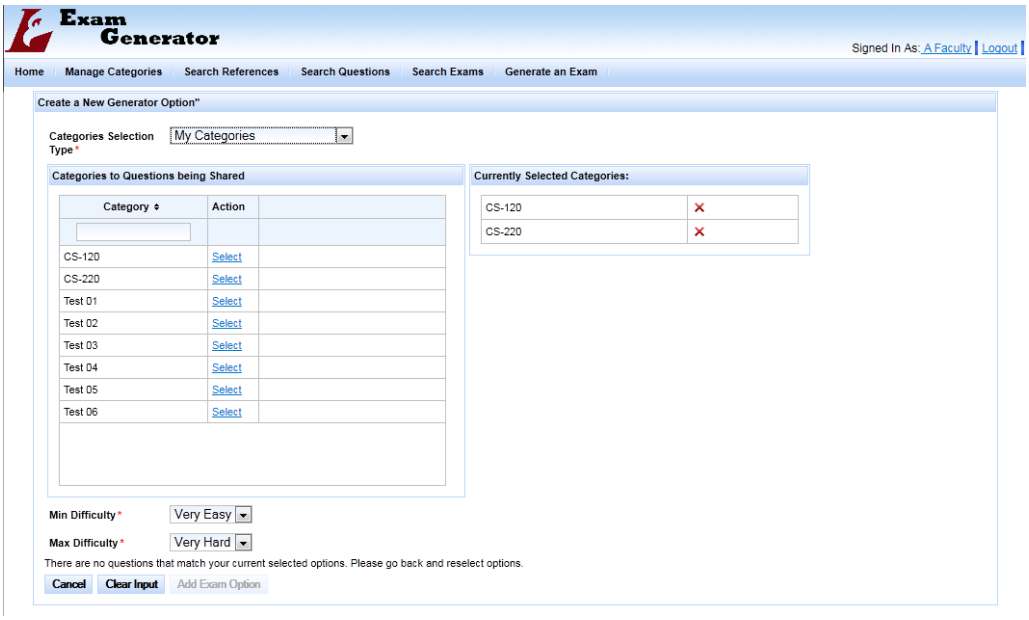

Figure 32: Faculty Create Exam Generator Option

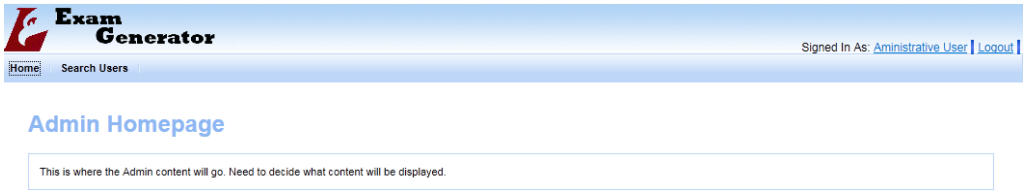

Figure 33: Admin Home

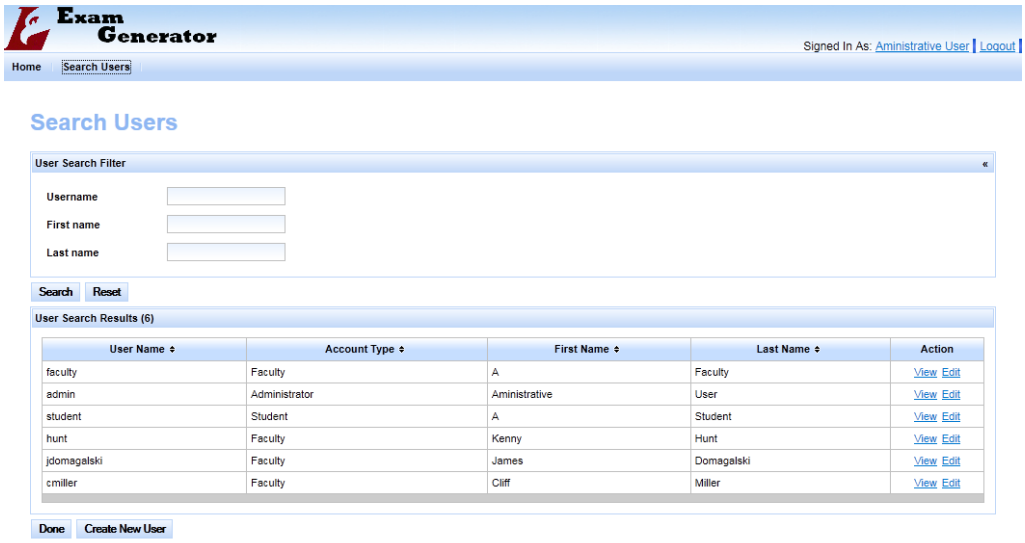

Figure 34: Admin Search Users

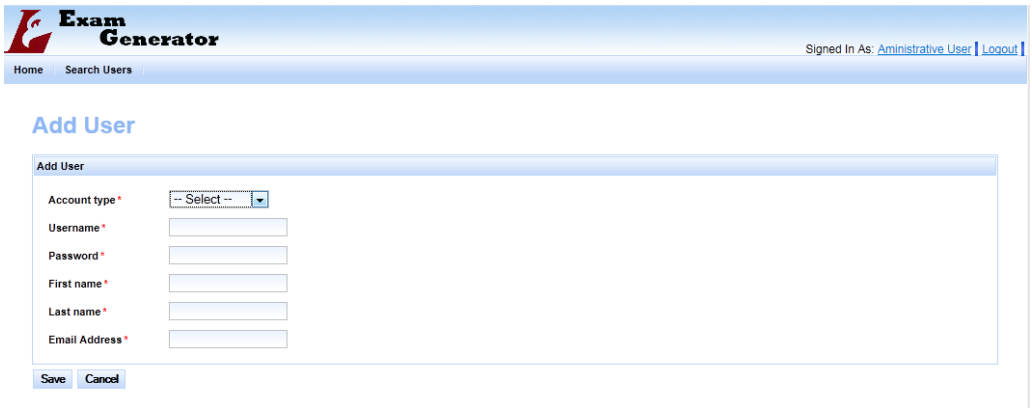

Figure 35: Admin Add User

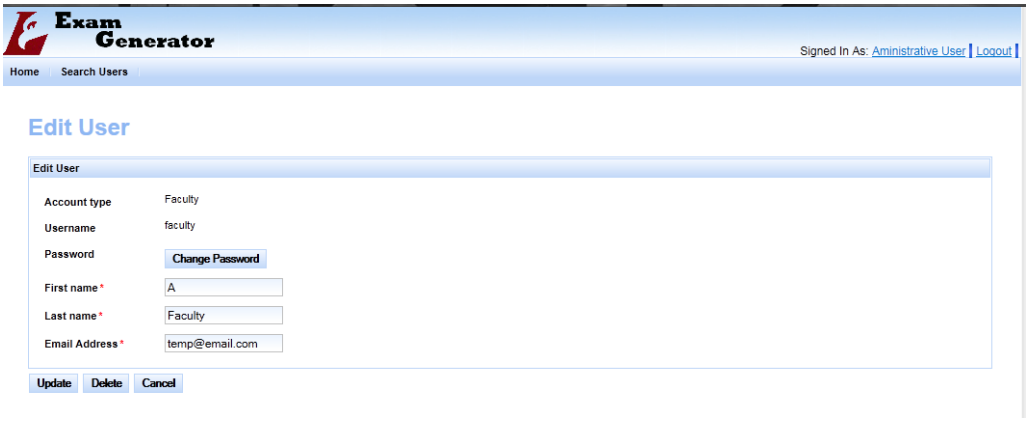

Figure 36: Admin Edit User