**Project X-Zam**



By: Chris Simon, Shah Newaz and Marco Campos

Computer Science 131: Software Engineering

Dr. Doan Nguyen

**Table of Contents**

**Problem Statement**

**Goal of Project**

**Proposed Solution**

**Functional and Nonfunctional Requirements**

**Methodology Usage**

**System Diagram**

**Sequence Diagram and Interface Specifications**

**Requirement Specifications and Use Case Diagram**

**Traceability Matrix (Use Cases Vs Requirements)**

**User Interface Design & Implementation**

**Detailed Screenshots of UI**

**Activity Diagram**

**Class Diagram**

**Data Flow Diagram**

**Entity Relationship Diagram**

**History of Work and Current Status**

**Testing and Evaluation**

**Future Steps and Possible Extensions**

**Lessons Learned**

**Contributions of CSC 131**

**Problem Statement**

For the past few years, an increasing number of students have had the goal of acquiring a position in the medical field. Requiring copious amounts of memorization of numerous medical terminologies and key words, studying for such a career can be a daunting process. Conventional methods of note taking and reading can become mundane to students who study over long periods of time. As a result, students who study in this repetitive manner will eventually retain less information over time.

**Goal of Project**

With the goal of improving memorization skills through interactive, enjoyable learning, our team will be developing an application to aid in student studying, particularly for those students who are perusing a medical degree. Students who utilize the app will be able to choose from various categories of which they are currently reviewing or studying for. With the plan of developing a practice test generator for each category, students will choose the amount of questions that they desire being tested on. As our primary goal is to improve retention, test questions will be asked at random, and users will be able to select questions that they no longer want to be tested on anymore. When a run of the practice test has been completed, users will have the option of retaking the same exam with the same set of questions, or only going over the questions that they had trouble with. Users will have the ability to exit out of a test at any given time.

**Proposed Solution**

As none of our team members are medical students, surveying, and information research using viable online resources and textbooks, will be done in order to collect the data necessary for our application. Our information will be stored and retrieved from a database, and the application will make requests to pull data from it. Students also have the ability of reporting questions that they believe to be incorrect, so that an administrator can update the information accordingly.

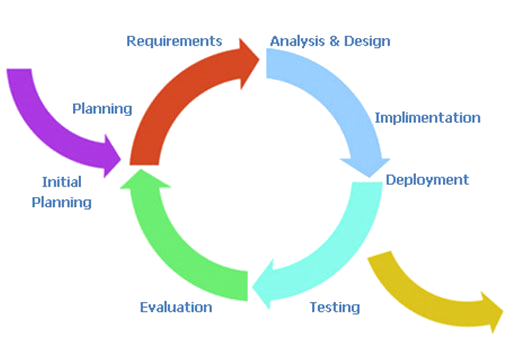
**Completed Functional Requirements**

|  |  |
| --- | --- |
| Requirement | Description |
| Termination | The user should have the ability to terminate the application at any time. |
| Choose from subcategories | The user should be able to choose the topic that they desire to be tested on. Once the topic is chosen, questions generated for that test must only come from that specified topic. |
| Choose number of questions | After selecting a subcategory, the user should be able to set the number of questions to be included on the practice test. |
| Scores List | At end of each test, the application will provide the number of correct answers over the number of total selected questions for that test. |
| Random Question Generator | The application will generate a random set of questions that will eventually be retrieved from the database when a read request is made. |
| Review Previous Incorrect Questions or Whole Test | After taking a test, the user should have the option of retaking the same test, with the option of retaking only the incorrect questions again or the whole test again. |
| Read Question | After generating a list of questions to select for the test, the application will obtain the selected questions from the database through the API, so that they can be displayed to the customer one at a time. |
| Create Question | The Admin will create new questions for the database through the API. |
| Create Question | The Admin will create new questions for the database through the API. |
| Update Question | The Admin will update the database with advised questions, provided from the users. |
| Delete Question | The Admin will delete existing questions from the database through the API. |
| Request to revise Question | The user will be able to submit a request for revisal of a question if he/she believes that it yields false information. |
| Apply updates to Application | The admin will modify the application in order to fix bugs and to add features. |

**Nonfunctional Requirements**

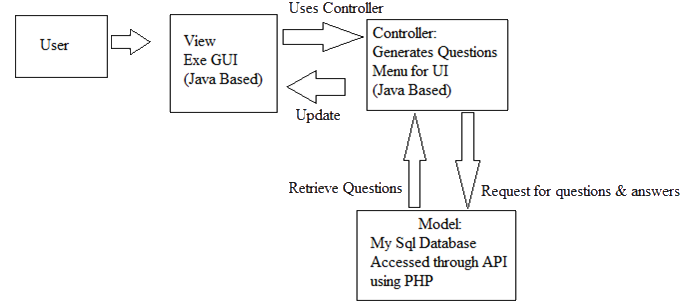
|  |  |
| --- | --- |
| Requirement | **Description** |
| Graphical User Interface (GUI) | The application will be visually pleasing, using textboxes, pictures, and buttons when applicable. The application must be fluid and responsive. |
| Navigation | The user will be able to navigate to whatever they need in a timely manner that does not disrupt workflow or slow down the user. |
| Interaction | Test taking will be constructed in such a way that allows for the user to review certain questions or content within a specific subcategory whenever they desire, while in the selected subcategory. |

**Methodology Usage**

****

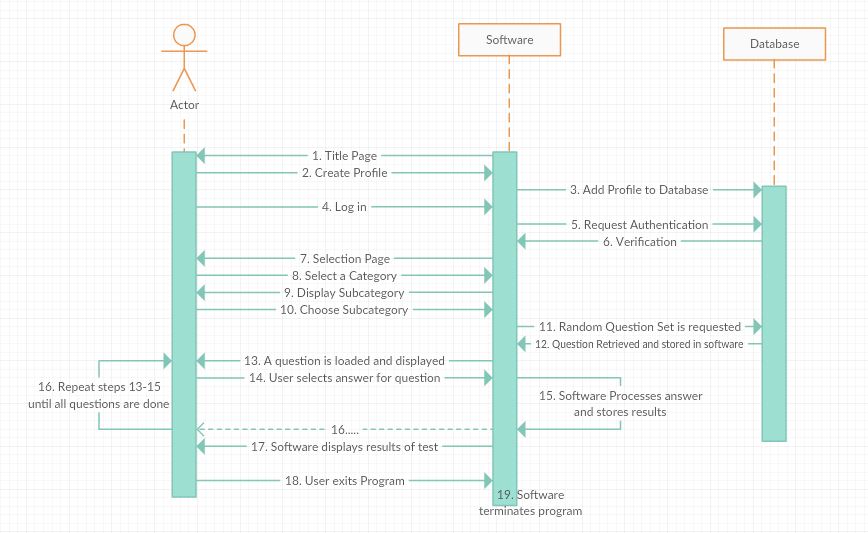
* Our project will resemble the incremental modeling process.
* Initially we discussed the effectiveness of the agile process
  + Decided against it because it’s our first time creating a project through repetitive documentation
* Decided that incremental model might be more beneficial for a steady and controlled development process

**System Diagram**

****

The system diagram represents the flow of events within the application, according to the user’s perspective. A user, when interacting with the application will first be shown the view, which contains the interface for the user to interact with. Users interact with the application using the GUI which is bound to the controller, which delegates actions between the model and the view. After the controller sends the user action to the model, the model sends the response back to the view through the controller. In this case, the model is separate from the actual application, as it is a database.

**Sequence Diagram**

****

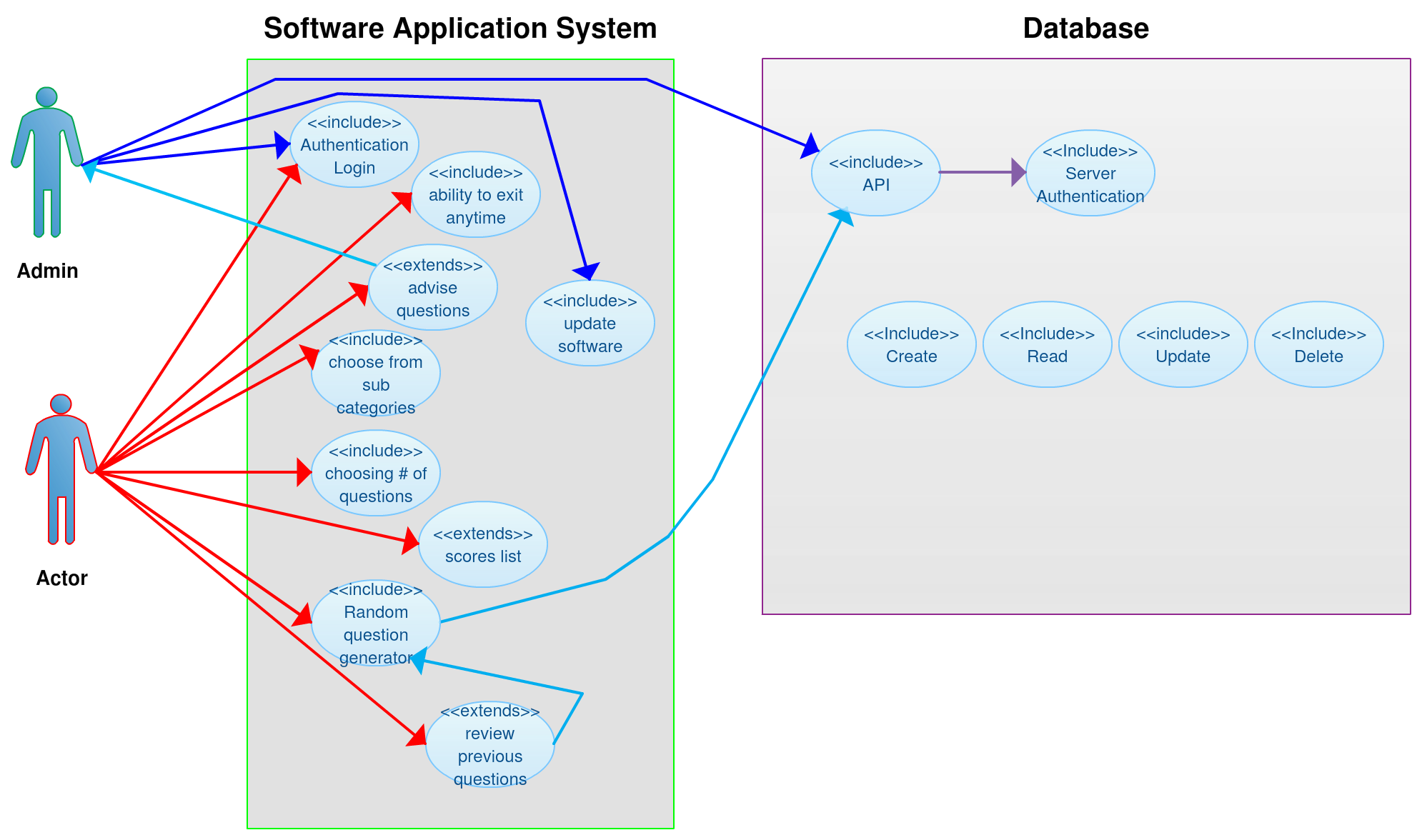
The sequence diagram displays the interactions between the user, software and the database. As you can see that the user interactions are displayed on the left hand side of the diagram, while the interactions with the software and the database interactions are displayed on the right. Here is a step by step sequence diagram which shows how the interactions will work.

This diagram will illustrate how the user will interact with the program and how the program will interact with the database. Not only does it show the user will interact with the program, but also it shows the steps of how one scenario of many will occur during log in.

**Use Cases**

|  |  |  |
| --- | --- | --- |
| **Use Case #** | **User Case Description** | **Use Case Details** |
| **1** | User chooses one of the Systems | Use Case Name: System  Actors: User  Priority: High  Status: Initial Development  Pre-Condition: Authenticated  Post-Condition: Current system is selected. Access to Subcategories.  Extension Points: N/A  “Used” Use Cases: Subcategory |
| **2** | User chooses a Subcategory:   * Medical Terminology * Multiple Choice * Definitions * Test | Use Case Name: Subcategory  Actors: User  Priority: High  Status: Initial Development  Pre-Condition: User must have selected a system.  Post-Condition: The subcategory is selected. Selected subcategory options are displayed.  Extension Points: N/A  “Used” Use Cases: Subcategory |
| **3** | Subcategory options :  User will take a practice test, deciding how many questions they desire being tested on.  When user answers an answer correctly, they earn a point.  When user answers an answer incorrectly no point will be given and answer will be optionally displayed.  The user may quit the test at any time.  The user has the option to move to the previous menu at any time. | Use Case Name: Subcategory options  Actors: User  Priority: High  Status: Initial Development  Pre-Condition: User must have selected a subcategory  Post-Condition: User is prompted for the number of questions.  Extension Points: Taking the test.  “Used” Use Cases: N/A |
| **4** | Test:  User takes the test | Use Case Name: Taking Test  Actors: User  Priority: High  Status: Initial Development  Pre-Condition: User must have selected to take a test. User has selected the # of questions on the test.  Post-Condition: User finishes the test.  Extension Points: termination of application  “Used” Use Cases: N/A |
| **5** | Incorrect answers:  User may review all the incorrect answers from each subcategory | Use Case Name: Reviewing  Actors: User  Priority: Medium  Status: Initial Development  Pre-Condition: User must have taken a test. User must have incorrect answers.  Post-Condition: User will have gone over all the incorrect answers.  Extension Points: N/A  “Used” Use Cases: Taking the test |
| **6** | Requesting new questions: User suggests new questions to the Admin | Use Case Name: Advise questions  Actors: User  Priority: low  Status: Initial Development  Pre-Condition: User must be Authenticated  Post-Condition: Admin will have new questions to input to the database  Extension Points: N/A  “Used” Use Cases: N/A |

**Use Case Diagram**

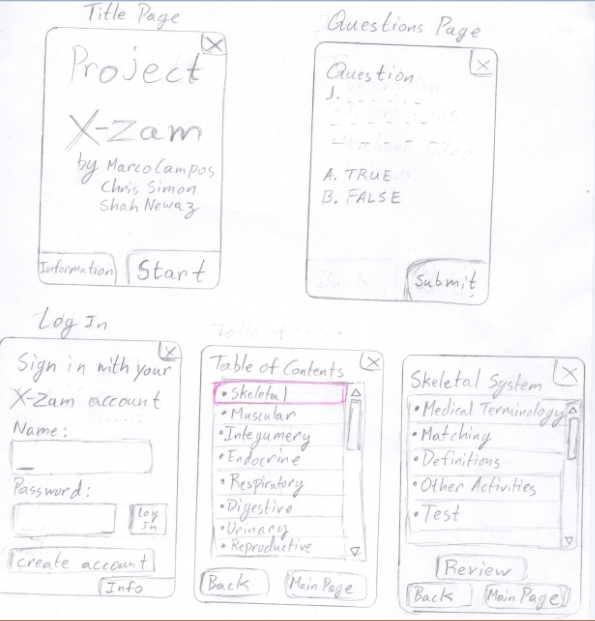
****

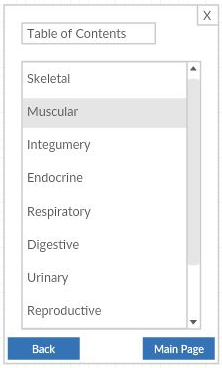
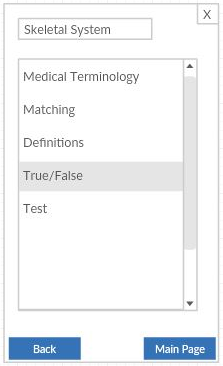
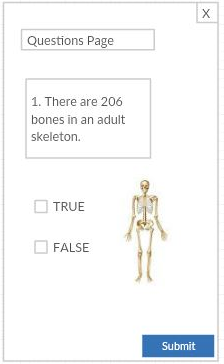
The use case diagram demonstrates the whole processes of the actor and the admin interacting with the Software Application System and Database respectively. After authenticating, the admin has access to the database through the API, which allows him/her to create, update, and delete content in the database. The user, after navigating through the Application System and selecting a subcategory, will be presented with a set of questions retrieved from the database through the API. For users, extends features consist of making requests to revise questions, reviewing the previous question answered incorrectly after taking an exam, and viewing their score after an exam.

**Traceability Matrix**

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Use Case Actor | Use Case Admin | Use Case Database |
| Log in | Access to software | Access to Software and database | Authentication API |
| Navigation | Ability to move to next pages | Ability to move to next page | N/A |
| Subcategories | Choosing category | The database organizes in a way that the questions will only be accessed by the category the actor chooses | The database organizes in a way that the questions will only be accessed by the category the actor chooses |
| Choose the Amount of Questions | Actor choose the amount of questions that will be asked | N/A | N/A |
| Correct Answer Display | After every wrong answer the correct answer will display | N/A | The database will include the answer only if the actor gets the answer incorrect |
| Random Question Generator | N/A | N/A | Generates the questions that will be asked |
| Score List | display the correct answers | N/A | Store the number of correct answer |
| Ability to Terminate | Actor can quit at any time during test | N/A | N/A |
| Incorrect Answers Review | Actor will have the ability to review the incorrect answers | N/A | The database will recognize the incorrect answers and store them for later use. |
| Update Questions | N/A | Admin will have the ability to update new questions and delete questions from database | The database will be updated for ambiguous questions, or needed questions |
| Read | The questions will be displayed to the actor | N/A | Generated Questions to be read by the actor |
| Create | N/A | Admin creates | The ability to add additional questions |
| Update Software | N/A | Admin will update in the database | N/A |
| Advise Questions | Actor will advise questions to the admin | If advised questions are valid, questions will be added to database | N/A |

**User Interface Design & Implementation**

****

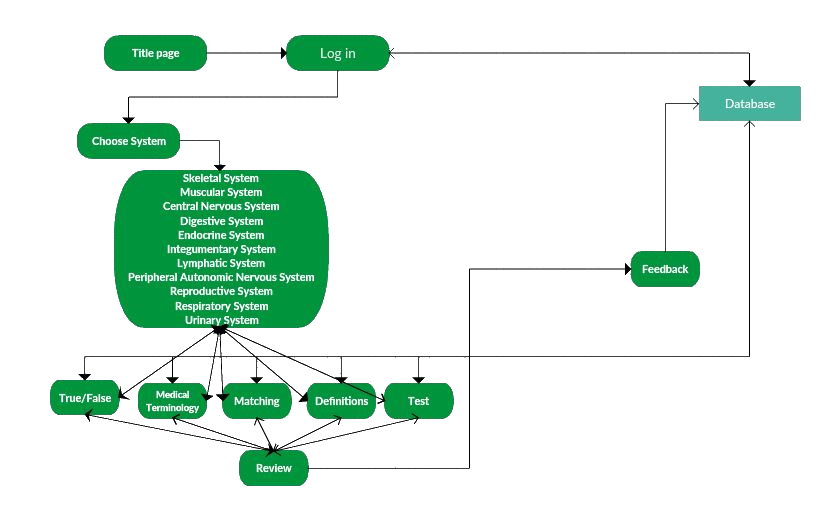
**** **** 

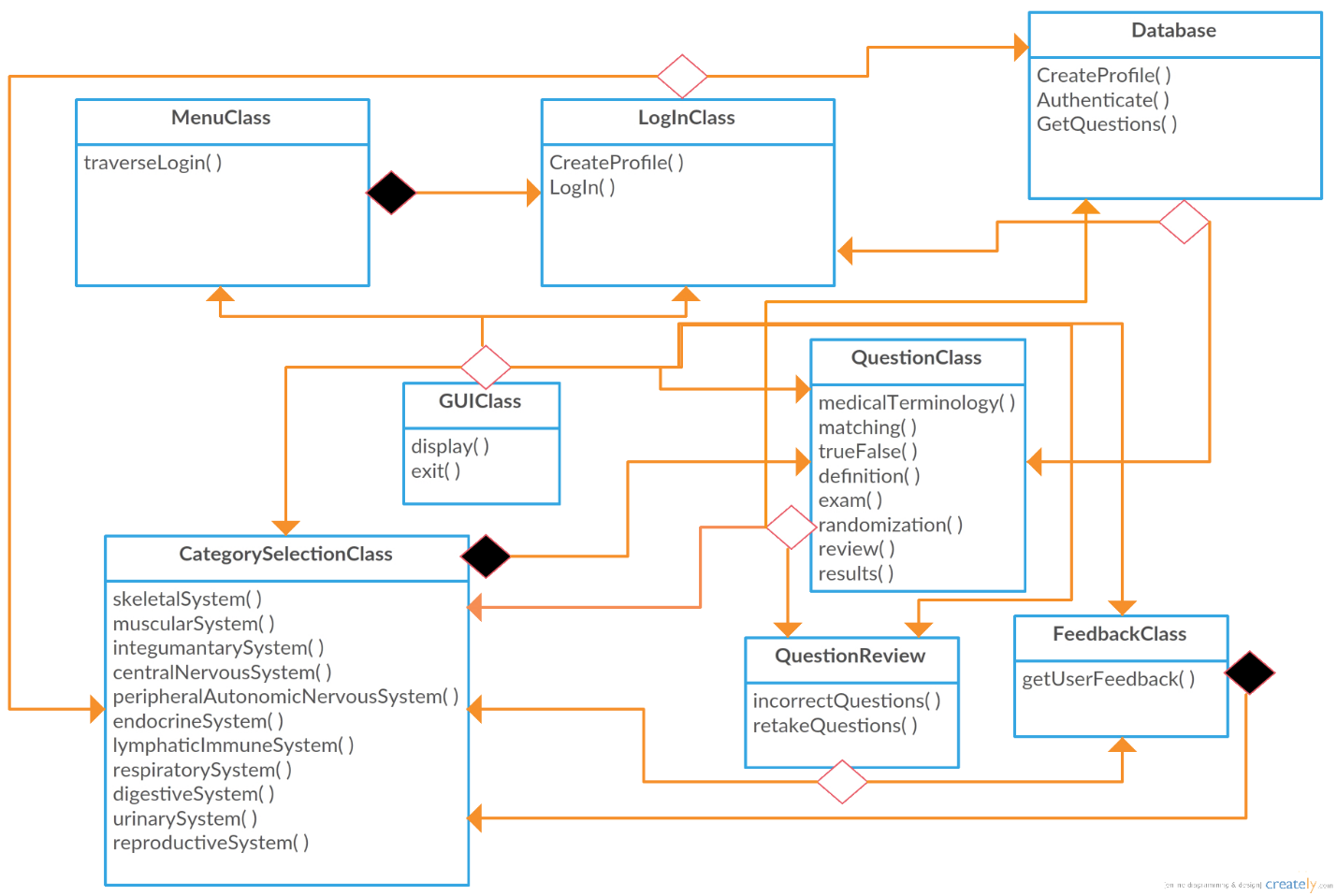
The user shall be able to The user then chooses what Finally this displays how

choose a variety of systems. types of questions he or she a certain question

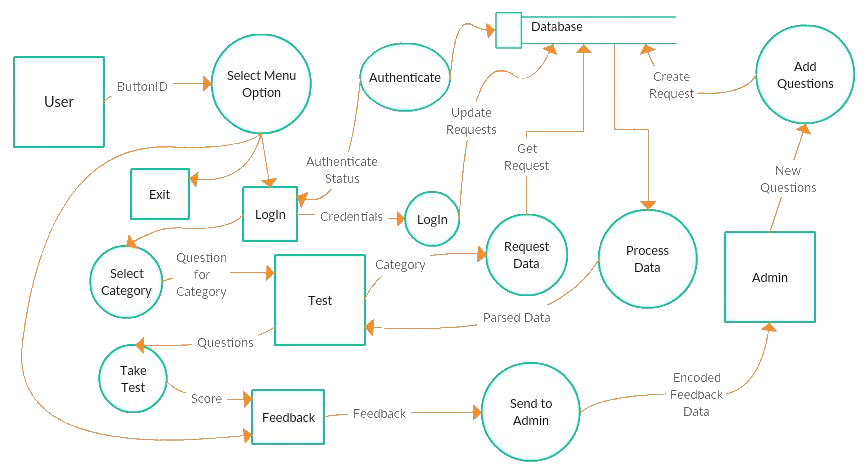
desires to be asked. shall be asked.

**Activity Diagram**



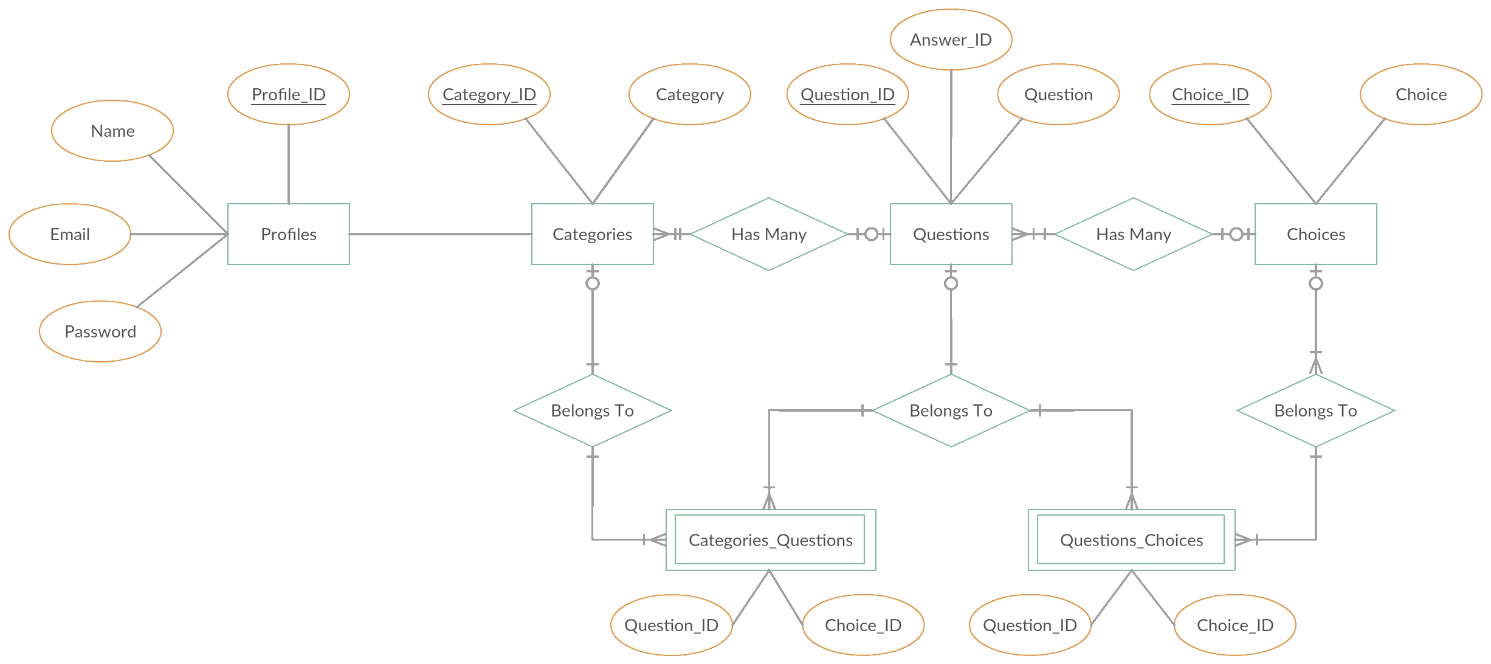
**Class Diagram**

**Data Flow Diagram**



The data flow diagram illustrates how data objects flow within the program and how state is transformed. In this diagram, squares represent external entities that receive and distribute data, while circles represent the methods that transform this data. The arrows represent where data shall flow and the parallel lines represent where data is stored. In our application, the database is what stores all the user information.

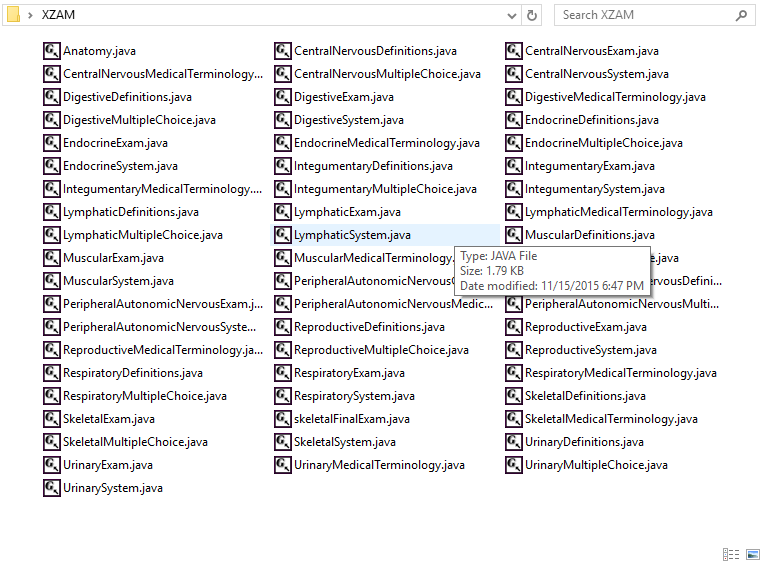
**Entity Relationship Diagram**

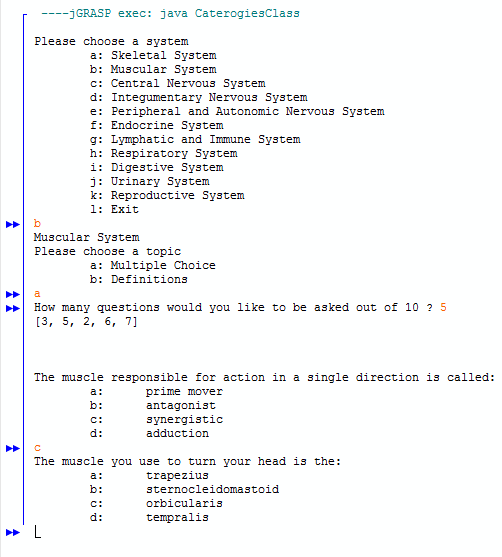


The entity relationship diagram, describes the structure of the data stored in the database. In our application, users are stored as Profiles, containing an email, a name and a password. The different Categories have many Questions. Each Question has many Choices. In order to link the Questions to specific Categories, a join table, Categories\_Questions is created in order to map each question to a specific category. This allows for the same question to exist in multiple categories.

**History of Work**

**Original Prototype (Before this class):**

****

****

Before this class, we used 58 separate java class files to contain each of the different categories.

**History: Deciding on the Problem Statement**

With the goal of wanting to create an application that benefits students in their future education, our problem statement was straight forward. Creating an application which would help students retain information in a fun manner was our statement through the whole project.

**History: Requirements**

We then worked on our software requirements, where we organized what shall be required of the project. Using a priority list, we described what each requirement shall do. After collecting functional and nonfunctional requirements, we then started creating our diagrams.

**History: Diagrams**

The class diagram acted like a blue print for our requirements, explaining how the implementation of our project shall be structured. The sequence diagram guided us on how the user would interact with the program.

**Current Status of Work**

Currently, we are on the implementation phase of the first iteration. Currently, we are in the process of translating the work from our class and data flow diagrams into code. Coding has been implemented for Category and subcategory selection, along with the question class.

As we near the end of the semester, we shall continue to work on this project where we will implement a log in in order for the user to save his or her score and where the user’s strengths and weaknesses are. For now we are using a flat file to store our questions and answers, but as the amount of questions and answers, we plan to learn more on how to implement them into a database. Below is a list of features that we plan to add into our next iteration cycle.

* Questions class
  + The questions shall randomly generated from the database.
    - If a certain topic has over 50 questions, then the user shall input how many questions they desire to be asked.
    - If a question has been asked, then that question will not be asked again.
  + A point will be given for every correct answer.
  + For each incorrect answer to a question, that question will be available on the next exam.

**Testing and Evaluation**

As we near the end of the design phase in our first iteration, plans for testing are being made. As described in our SRS, we shall be implementing unit tests to check the validity of our work, to see if it meets our documented requirements and use cases. Unit Tests will be generated for all classes in our class diagram, including tests generated for server side code for the database. Database queries shall be tested manually by checking to see if the data returned is in fact the data that is expected from the database. We shall then evaluate our program as a whole, by beta testing it, to make sure that our nonfunctional requirements are also met. If some of the tests fail or some of the functional and nonfunctional requirements are not met, redevelopment and retesting and evaluation will take place. When everything successfully passed, we will move onto the next iteration of our project.

**Future Steps and Possible Extensions**

Below is a list of features that we are considering as possible extensions to what is planned for our application. If these features are added, they will be included into a new iteration and our diagrams and documentation will be updated accordingly.

* Finishing up functional requirements, followed by nonfunctional requirements. Mostly database integration:
  + Authentication via API
  + Review previous incorrect questions or whole test
  + Read question
  + Create question
  + Update question
  + Delete question
  + Request to revise question
* GitHub integration
  + Easier to manage code compared to Google drive.
  + Follows iterative design model
* BlueJ
  + Switch to BlueJ IDE for use of automatic UML Generator

**Lessons Learned**

* Refactoring
  + Helped us to reduce the amount of coding needed to create the same functionality, especially when writing separate class files for each category at first.
* Class Diagram Usage
  + Helped us to breakdown our work into the classes contained in our application
* Pair Programming
  + We found pair programing beneficial, enhancing the overall quality of our work
* Entity Relationship Diagram
  + Helps structure how data is stored

**Contributions of this Class**

* Because of this class, we were able to learn how to document our work using the software engineering process.
* Had we not taken this course, we would have started out coding in the beginning, which would have made it difficult for us to make any changes to our software over time, creating the possibility of losing valuable time and money.
* Most industries use these processes which are safer and more cost effective than our previous methods of approach, and so it’s good to be familiarized with these.
* For future projects, we can look back to this experience, and apply similar techniques in order to make projects that are easier to document and to create.