Project Group Members: Adrian Amora, Zhifei Zhang(Ethan),
Jeffrey Krystek, Katherine Milliken
Project Leaders: Dr. Ravi Patel, Mr. Jonathan Velez
Spring 2018 Capstone

Capstone Final Paper

For our Capstone, we created proper documentation for the oral pathology quiz game. The databases for the two quiz games had been merged, and we were initially responsible for merging the two different web applications under one roof, so to speak. The current quiz game application for oral pathology was to be the base and we were originally to include the pharmacy questions in it. The classes for the Graphic User Interface were written primarily in PHP, which is where we elucidated most of the interactions between project folders. In addition, the view of the game was implemented with HTML and CSS while the controller was written in Javascript. Within each class, we contributed the majority of the comments to guide future programmers during their revisions. We learned and attempted to utilize the Cake PHP framework for our front end, while MySQL informed the database interactions with the preliminary site.

We immediately ran into troubles when looking at the code base, as it had a number of problems. The primary problem was that much of the code base was not commented. Entire sections of code were not clear on what their functions were. When looking more into the code, the entire code base proved to be experimental at best. Some classes were not used at all, and no indication was given. A significant amount of time was used to try and find where the code was used in the main program in order to provide proper documentation for them before realizing that they were not utilized at all. In addition, even some of the commented code was jumbled and

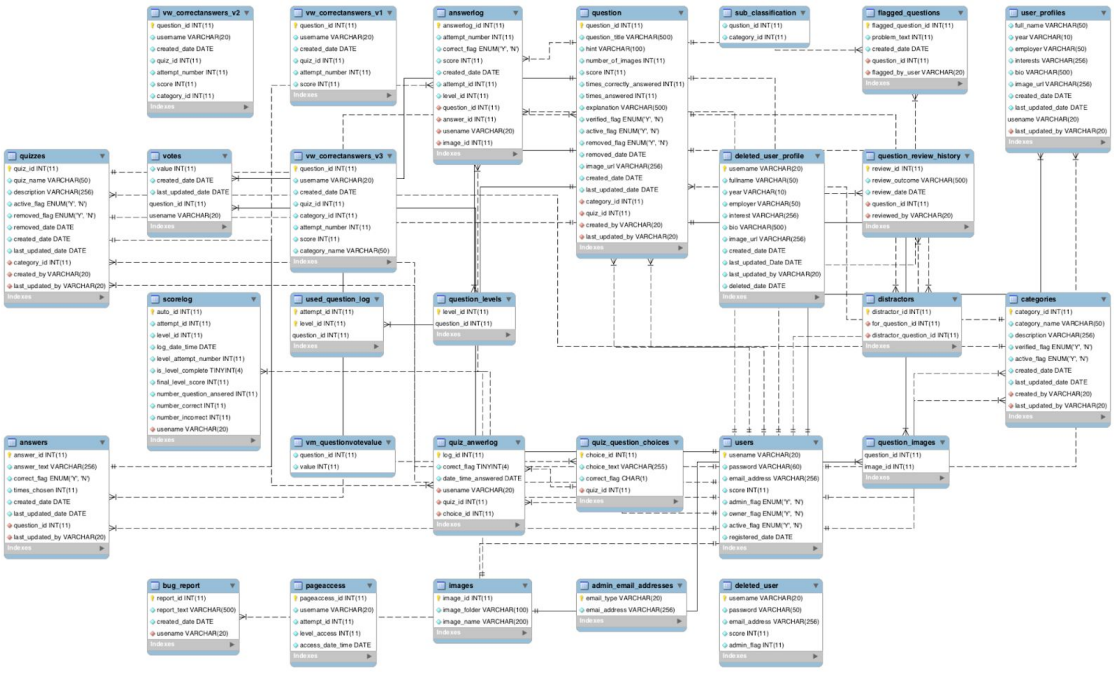
difficult to understand. The project appeared sloppily completed, such as misspelling some significant variable names. In addition, the scant documentation present was also messy and unhelpful in understanding the code base.

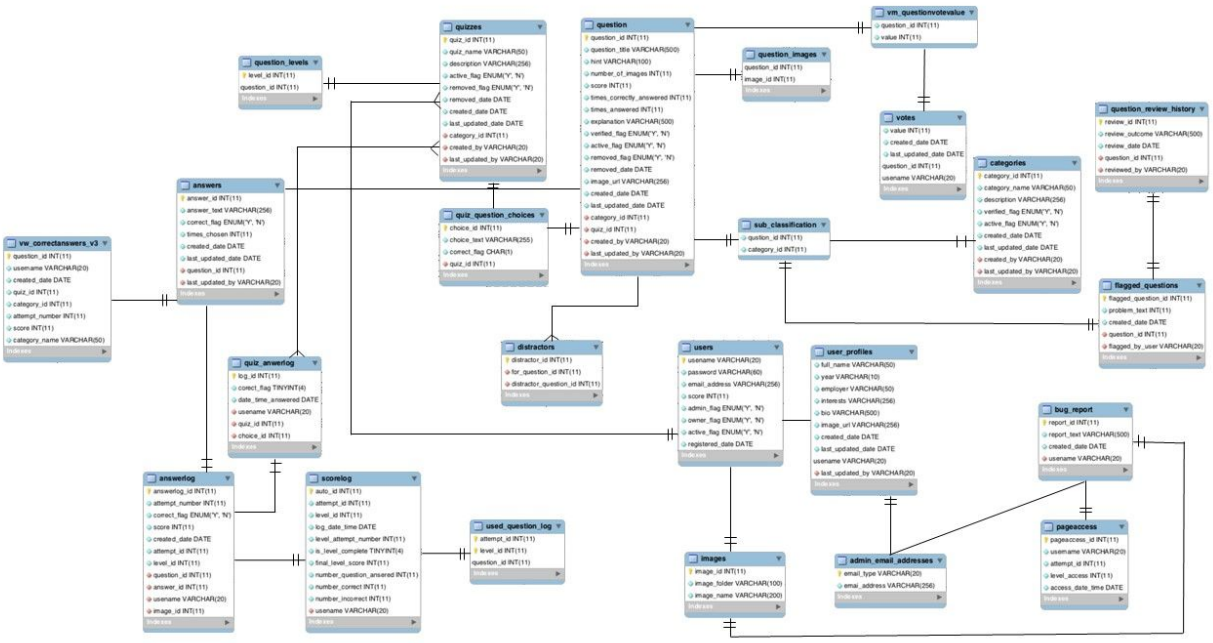
The primarily task for Adrian was documenting the existing codebase. This was done in a number of ways. First, he directly commented the code base as thoroughly as he could. He went through each individual file and commented the code so that the next group would be able to see the code base and understand it notably faster than we could. In addition to commenting directly on the code, Adrian created a outside document that described each class' function in layman's terms. This was done so non-coders could get idea of what each class did, as well as to provide another piece of documentation for coders if they are unable to understand the in-code comments.

A few tables were created in this document. One table showed which classes were inherited by which. Hopefully, this is helpful in showcasing which classes are important and essential for the functionality of the overall quiz game. A second table outlined the databases. This table includes what each column described, how each database table connected to each other, and which classes accessed them. This was done to help improve the preconstructed ER diagram, which had few usable elements. Adrian's outside file consolidates the simplest documentation in one place so that anyone can have easy access to the explanation of the code quickly without having to jump between all of the different code files.

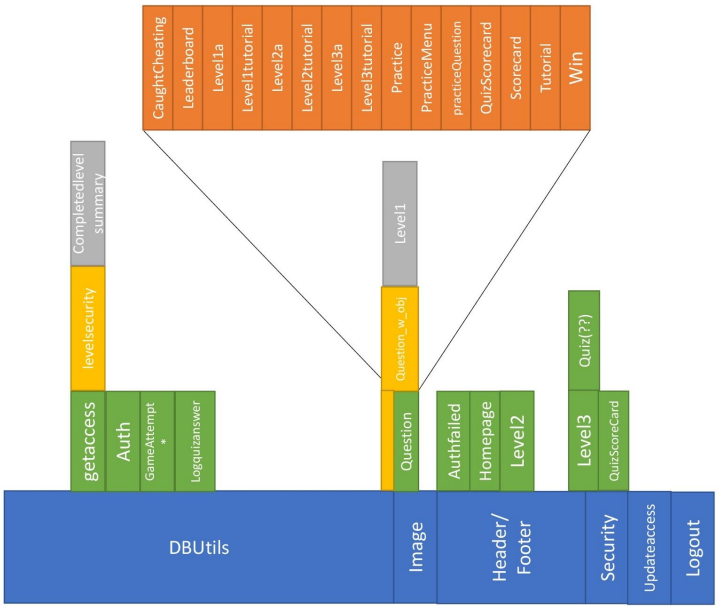
Finally, as Adrian was originally designated as the front-end guy, he included a small user interface/user experience review of the oral pathology quiz. This review described the basic functionality of each page of the game and includes any visual bugs and problems. Also, he included small recommendations on how he felt the game could be improved. As the application is a quiz game, he primarily compared it to Quizlet.

With respect to the Entity-Relationship Diagram discussed in the previous update, Katherine was responsible for revisions. In the original diagram, not all nodes had connectivity and some overlaid node paths, which confused the connections and made class interaction difficult to document. The objective was to simplify the interactions and to remove unnecessary nodes. The "crow's feet" notation was maintained for continuity between designs. The diagrams are depicted below for comparison.

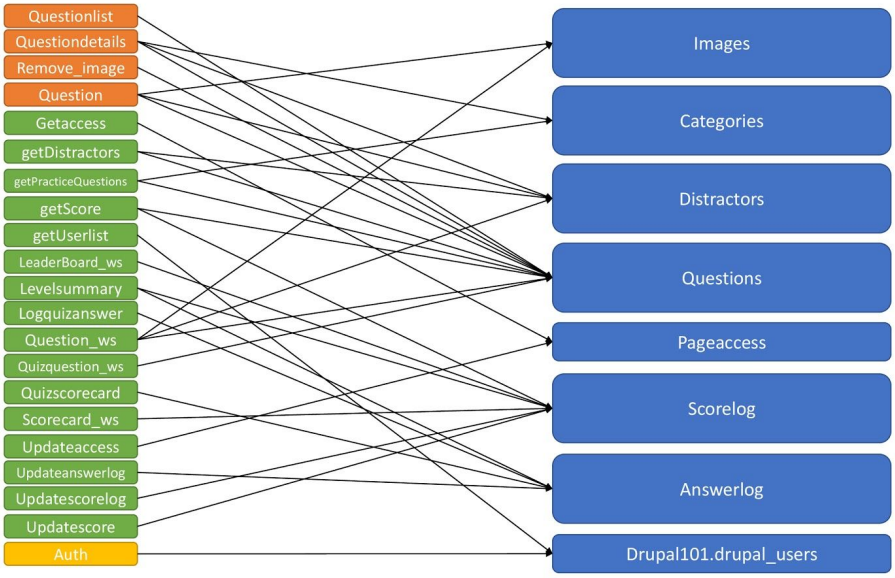




Katherine created simple reference diagrams of the class inheritance and database interactions. The class inheritance diagrams are designed with a “hierarchy of needs” perspective. For instance, if there are bugs with the DBUtils.php class, the entire application could crash. However for much smaller classes, their dysfunction would inhibit the application less voraciously. The more vital the class and the less it depends on other classes for functionality, the lower it is represented in the diagram. The most complex diagram is shown below. The classes in orange rely on all of the classes in blue and Question.php for function. The grey however only relies on the classes directly below. In regards to the class interactions with the databases, there are only eight SQL databases referenced by the PHP classes. The majority of PHP classes yielded from the Admin and Rest Folders. The objective of the visual aids was to provide understanding of the processes to non-programmers and as a quick reference guide for coders.



Below is the databse interaction schematic.



At the same time, Zhifei was in charge of creating the data flow diagram (DFD), which is equally important as the entity-relationship diagram and hadn't been created by previous groups who designed the web and overall program. However, there are multiple ways and standards of

drawing a DFD and most of them make sense in its own form, such as a logical DFD which focus more on business model vs. a physical DFD which is more like a designing blueprint for computer programmer, our group decided to 1) create the DFD, as assigned and 2) set a clear standard of drawing DFD for this Pharmacy Quiz Game so that later groups who join this program are able to easily understand the diagram and modify it under a consistent standard.

Based on all the information regarding DFD found online -- including Yourdon-Coad notation, Gane-Sarson notation -- a PDF file of how to draw the DFD was created, explaining the graphic notation and 19 basic rules. For instance, no process should have only inputs/outputs without an output/input, which served a general user story. A future developer would expect to have an adequate diagram as his/her blueprint and understand the overall structure of the program.

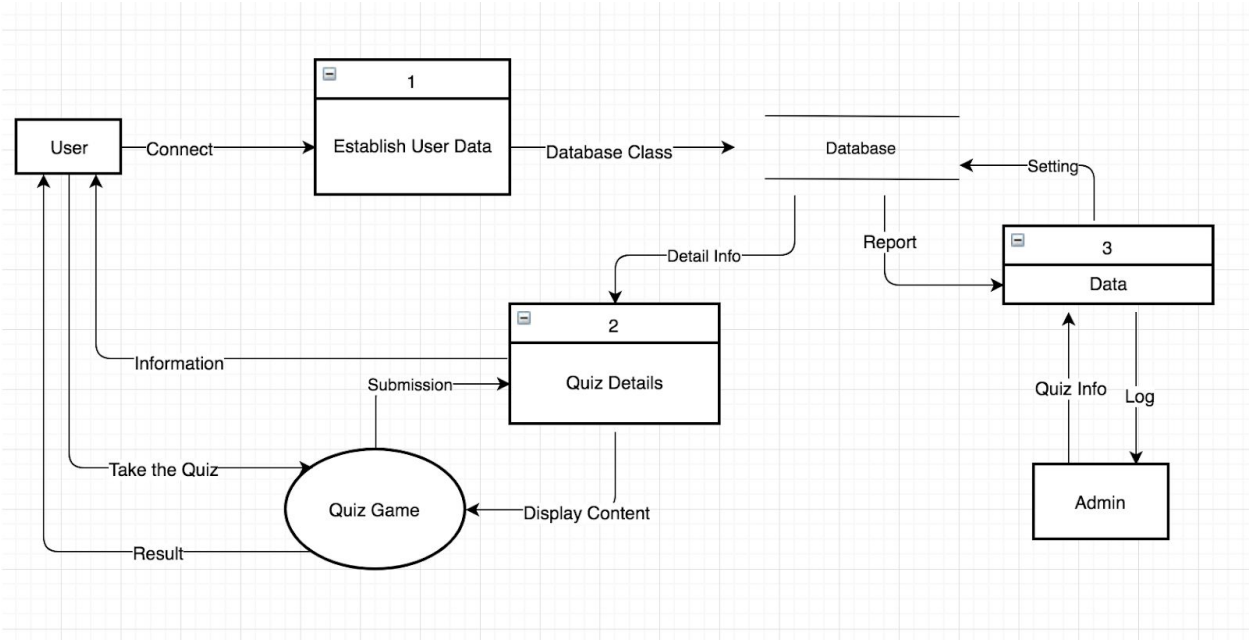


**diagram under construction*

Besides the development purpose, different levels of DFD explained the user stories and presented how different users -- student, programmer, database manager, and etc -- play their roles in this quiz game program, whether it's a client who wants a working prototype available through all steps of development, a professor who wants the quiz game to allow feedback on the quality of the questions, an admin who wants the software to be updated to the most current information, or a future programmer who wants to debug more efficiently and provide a better program. Since the codebase provided by the previous group is still under investigation and the prototype DFD was based on the architecture of the codebase, the above graph of DFD is still under construction and new modification will be done during the last sprint, including more specific data arcs, subsets of data storage, and process of exploring and managing quiz question.

Towards the middle of the semester we considered investigating web testing frameworks for automated testing. Jeff looked into using the Selenium IDE due its ease of use. It is a free, open source Mozilla Firefox software add-on. Unfortunately, after one of the recent updates to Mozilla Firefox Quantum the IDE is no longer compatible. It does not appear there will be future support. Capybara was offered as a second option, which is part of the Cucumber testing framework. After some research it appeared that Capybara seems best suited for Ruby on Rails applications. The workarounds to get it up and running were a bit too complex with the time constraints. If Jeff had more time, it would be best to focus on a framework that is tailor-made specifically for PHP.

In the meantime, manual testing was performed. Test cases were written using a template, which can be used to create automated tests for future developers. Defects were reported using another written template. These defects were described in such a way that automated tests could be written to cover the areas where the defects occur. At the very least, we hope that this could inspire of the types of tests that should be included in a fully functioning automated test suite.

Our project leads, Dr. Ravi Patel and Jonathan Velez were very understanding in our struggles. They were supportive in our pivot from our original plan to a primarily documentation route. Jonathan also informed us that our work this semester, which basically amounted to a code review, was a legitimate job in the industry. His primarily goal for us in the semester seemed to be teaching us tools that are commonly used in the industry. He provided us with tutorials for PHP and it's frameworks, the MVC (Model-View-Controller) design pattern, and the integrals of user interface and user experience. While we did not meet with Dr. Patel as often, he helped give us some direction for what our deliverables should be. As our final contribution to the project, we will be creating a short video describing our documentation for the shareholders of the project.

In conclusion, we learned much about the fundamentals for proper web development and the importance of documentation. Despite our high expectations for rejuivating and integrating the web applications, we faced considerable deficits due to the lack of clarity in the provided codebase. At times, we struggled with creating a testing environment for the existing website and instead provided the expectations for test cases and defect reports. We created data flow

diagrams that represented our user stories, The user stories have changed quite a bit from our initial proposal. Working as a team within the agile sprint framework taught us much about personal responsibility and responsibility for client expectations. We hope our clients are pleased and we've done much to "undo the evil of the past" for future capstone students.