Final Project Report

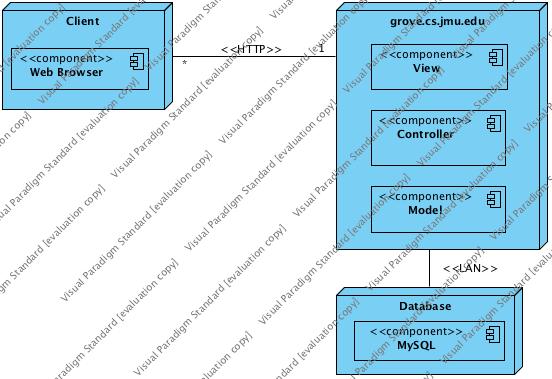
Team27

Section 2

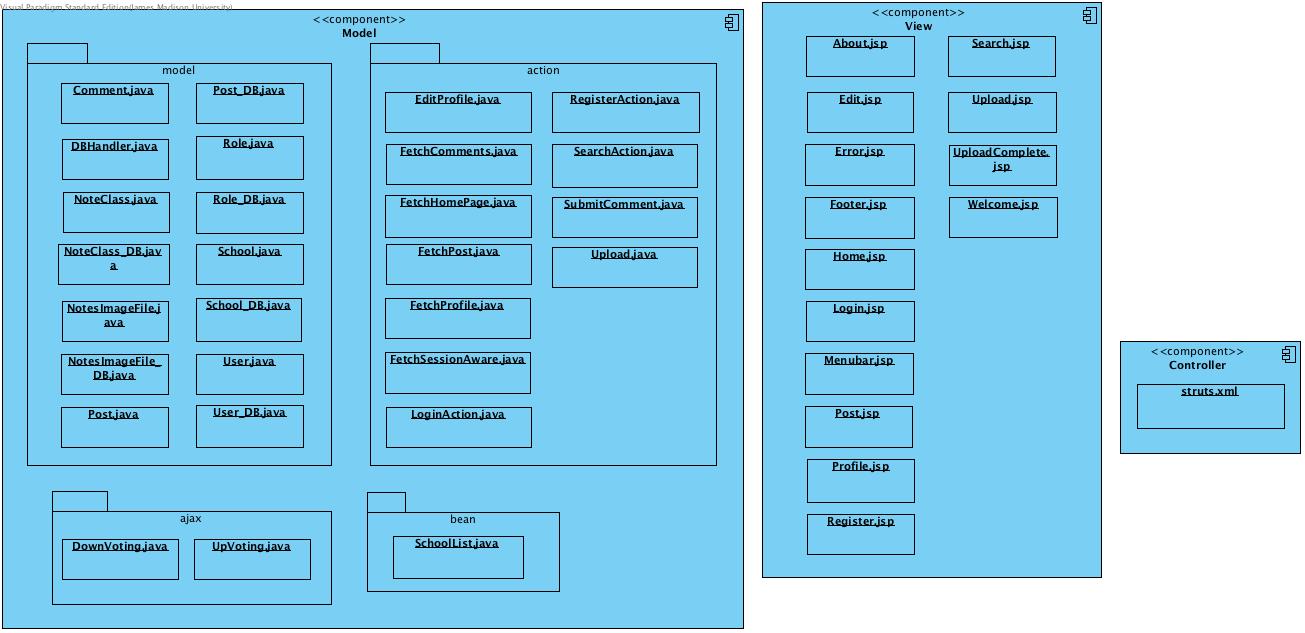
Christopher Recinos, Miles Greatwood, Jacob Wilson, Kendrick Smith

**Technical Overview**:

During the design phase, we'd planned on having a MVC architecture for the application, with struts2 acting as the controller, a number of different jsp files providing the view, and action and data files acting as the model. Our overall architecture stayed as MVC, but the number of components inside the view and model changed during development. For the view, the number of jsp files doubled from what we'd originally intended it to have as we just kept wanting to add more and more to the application. In addition, the view also contains an application-wide .css file we used to handle a majority of the styling, and a couple of .js files to handle our AJAX transactions related to our comment system, where we wanted to update the page with the upvote or downvote on the comment without needing to reload the page. For our model, we originally planned on having 4-5 data classes with a DB class for each data class, as well as a DBHandler to create the connections to begin with. We ended up using 7 data classes, with 6 other DB classes and the DBHandler. The increase in the number of classes was because, primarily, of how over database was continuously evolving throughout the development stage. In addition to the data classes, we have 11 action classes, with some of these action classes dealing with user interaction upon clicking something on a page and the rest dealing with dynamically grabbing data upon the loading of the page based upon what session variables were set as, or what parameters were passed to the page. This final result required a lot more of a time commitment than previously anticipated, but it allowed us to put a lot more content into the application than we expected to, in addition to just showing a greater separation of duties amongst the various parts of the application.

**

*Figure 1. Updated Deployment Diagram*



*Figure 2. Updated Component Diagram*

**Final Database Design:**

**School Table:**

* *id:* INT, PRIMARY KEY, AUTO\_INCREMENT
  + Generated primary key for school
* *school:* VARCHAR(255), NOT NULL
  + A String representing the name of the school

**Role Table:**

* *id:* INT, PRIMARY KEY, AUTO\_INCREMENT
  + Generated primary key for role
* *role\_name:* VARCHAR(255), NOT NULL
  + A String representing the role name (ADMIN or USER)

**Class Table:**

* *id:* INT, PRIMARY KEY, AUTO\_INCREMENT
  + Generated primary key for class
* *class\_num:* INT, NOT NULL
  + Integer value representing the number of the class (i.e. 347 for Web-Based Information Systems)
* *class\_name:* VARCHAR(255), NOT NULL
  + String value representing the name of the class, as given by the registrar.
* *class\_desc:* VARCHAR(255)
  + An optional description of the class, such as the final goals of the class or tools used

**User Table:**

* *id:* INT, PRIMARY KEY, AUTO\_INCREMENT
  + Generated primary key for each user
* *password:* VARCHAR(255), NOT NULL
  + String representing the hashed value of the user's password
* *email:* VARCHAR(255), NOT NULL
  + String representing the user registered to the user
* *name:* VARCHAR(255), NOT NULL
  + String representing the name of the user
* *username:* VARCHAR(255), NOT NULL
  + String representing the username shown for the user
* *role\_id:* INT, NOT NULL, FOREIGN KEY (role\_id) REFERENCES role(id)
  + Foreign key representing whether the user is a USER or an ADMIN
* *is\_professor:* TINYINT
  + Boolean value representing whether or not a user is a professor or a student. If null, user is treated as a student
* *school\_id:* INT, NOT NULL, FOREIGN KEY (school\_id) REFERENCES school(id)
  + Foreign key representing the school the user attends

**Posts Table:**

* *int:* INT, PRIMARY KEY, AUTO\_INCREMENT
  + Generated primary key for the posts table
* *author\_id:* INT, NOT NULL, FOREIGN KEY(author\_id) REFERENCES user(id)
  + Foreign key to the user table representing the user who created the post
* *class\_id:* INT, NOT NULL, FOREIGN KEY(class\_id) REFERENCES class(id)
  + Foreign key to the class table representing the class the post is associated with
* *text:* TEXT, NOT NULL
  + Any note text entered by poster when creating the post
* *rating:* FLOAT
  + Average rating for the post, meant to show whether or not other users think the post is relevant or not
* *endorse:* TINYINT
  + Boolean representing whether or not the professor associated with the posting approves the notes or not, similar to Piazza (not currently implemented)
* *notes\_desc:* TEXT
  + Optional description for the post, outside of the post text itself
* *title:* TEXT
  + Optional title for the post

**Comments Table:**

* *id:* INT, PRIMARY KEY, AUTO\_INCREMENT
  + Generated primary key for the comments table
* *post\_id:* INT, NOT NULL, FOREIGN KEY(post\_id) REFERENCES posts(id)
  + Foreign key to the post table representing the parent post the comment is associated with
* *comment:* TEXT, NOT NULL
  + Text for the comment
* *author\_id:* INT, NOT NULL, FOREIGN KEY(author\_id) REFERENCES user(id)
  + Foreign key to the user table representing the user who created the comment
* *score:* INT
  + Numerical value for the commenting representing upvotes – downvotes

**Image Table:**

* *id:* INT, PRIMARY KEY, AUTO\_INCREMENT
  + Generated primary key for the image table
* *file\_location:* VARCHAR(255), NOT NULL
  + String representing the location of the file
* *post\_id:* INT, NOT NULL, FOREIGN KEY(post\_id) REFERENCES posts(id)
  + Foreign key to the posts table representing the post the image is associated with
* *content\_type:* VARCHAR(30), NOT NULL
  + String representing the file type of the image being stored (i.e. jpg, png, etc.)

**Retrospective: TODO**

**Process Improvement:**

While the application development didn't go as smoothly as we intended, but it was definitely a great learning experience. While there was a lot that we feel went reasonably well, there were a couple things that, looking back, definitely were more of a hindrance than a help. A large portion of our wasted cost coding-wise is a result of how we went about actually implementing the application, since we started implementing back to front, instead of front to back. Because we were writing database classes without any knowledge of what information we'd need to retrieve from the database, a lot of the early methods in the database classes were simply guesses as to what we might need, which led to a lot of wasted code when we didn't actually use the method. If we could start the application now, we'd definitely start with the jsps and then add only what we needed. Another thing that could've made the process a lot more streamlined and painless involves where in the jsp creation we started. If we had started with the login.jsp and worked backwards, we would have been able to connect things as we went along, rather than having a bunch of individual jsps that we then had to go back and connect later, usually meaning we were doing a lot of extra debugging in the process. In addition, because we used the struts2 plugin, we had several limitations placed upon us, which wouldn't have been the case if we'd been using the struts2 jar instead. Firstly, using the plugin meant that we had to go through the installation and setup of the plugin before being able to do any work on the application. While that entire time is only a few minutes, it's still time that could've been used to something more productive, especially since those few minutes add up each time a new project is cloned to a new lab computer. The second issue with the plugin is that the CLASSPATH for the struts2 library wasn't setting up correctly on some of the newer versions of NetBeans. Chris in particular could not get the library working and because of that, he was forced to work in the lab. Using the jar file instead would've meant that it would've been included in the cloning of the repository, which would've brought struts2 with it, eliminating the issue where NetBeans could not find the struts2 library. Lastly, we would've had more freedom with our styling. Because we were using the struts2 plugin, we couldn't use bootstrap in our project. Therefore, a lot of our form styling isn't as clean as it could be, or had to be achieved in a hacky sort of way (such as moving a button by adjusting margin for the container). A final thing that we should've used and would implement in future projects is the better use of a project backlog. While we always knew what needed to be done overall, there was never any tracking of who was working on what, nor what needed to be done for each jsp. Essentially we had no real planning going into the development stage and it really cost us a lot of wasted time throughout the process.