CS3200: Database Design

Final Report

Justin Littman and Christopher Wong
Blackboard Group Name: WongLittman
April 19, 2017

README

Located in project root directory under README.md

Technical Specifications

More details can be found in the project root directory under README.md

User features:

- Sign-up / Sign-in(using custom login system)
    - Built to support OAUTH but disabled for the purposes of demo/debugging
    - Placeholder values of ‘admin’ and ‘password’ can be used to submit the login form
- Add/remove classes and assignment category breakdowns
- Add/remove assignments and grades one at a time
- Update assignment weights
- View assignments and grades that have been added to the database
- View JSON dump to the screen for testing and debugging
- Single button to clear the data from the database for testing

Application features:

- View total grade for class
- View mean score for each assignment category

UI features:

- Fully responsive and mobile-friendly (device agnostic)
- Clean modern design language (fast!)
- Utilized Bootstrap grids to display different UI views on different browser sizes
- Used Jinja to pass inputs to the front end on page refresh in order to dynamically provide feedback to the user
- Also utilized Flask message flashing to display status messages to the screen

Reach features:

- View “what if” scenarios based on grade projections
- Peer comparison
- Auto-import from Bottlenose (CCIS grading system)
    - View the Jupyter Notebooks folder for Scraping bottlenose.ipynb to see how we scraped the session cookie and retrieved the assignment page. This logic was developed but not implemented in the final version.
- Deep analytics

Database type: MongoDB

We used MongoDB for our database due to it's rich data model that allows for a dynamic schema. It's ease of use interfacing with Python (via PyMongo) allowed easy access to and from the database. MongoDB is flexible and scales easily especially across distributed platforms. Within our Python Flask backend we built a series of convenience functions that combined several PyMongo commands such as db_connect() which established a connection to the database and returned a cursor object. This allowed us to quickly instantiate connections while reducing code duplication. We also wrote additional functions that allowed us to retrieve different collections in one line of code. Each time a page was refreshed or reloaded, the back-end will serve up the updated database on the fly.

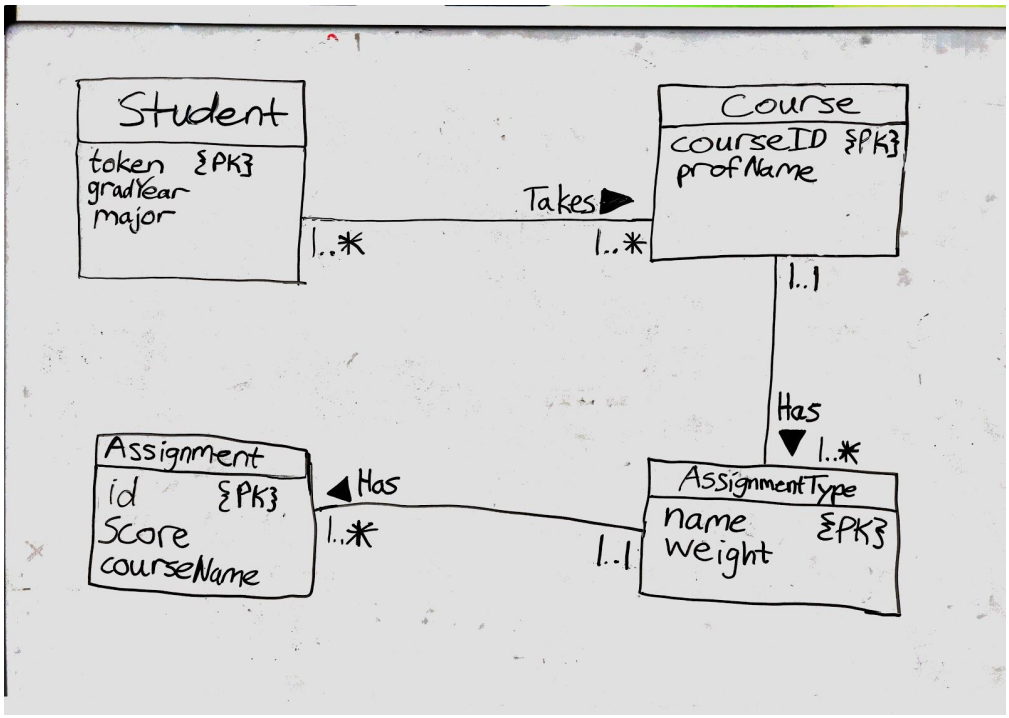
Hosting:

Our development environment will be done locally on macOS Sierra with LAN based testing conducted on a RaspberryPi. Remote hosting is provided via a single DigitalOcean droplet running Ubuntu 16.04 LTS and an accompanying MongoDB 3.4.3 database. This all sat behind a UFW firewall for security purposes.

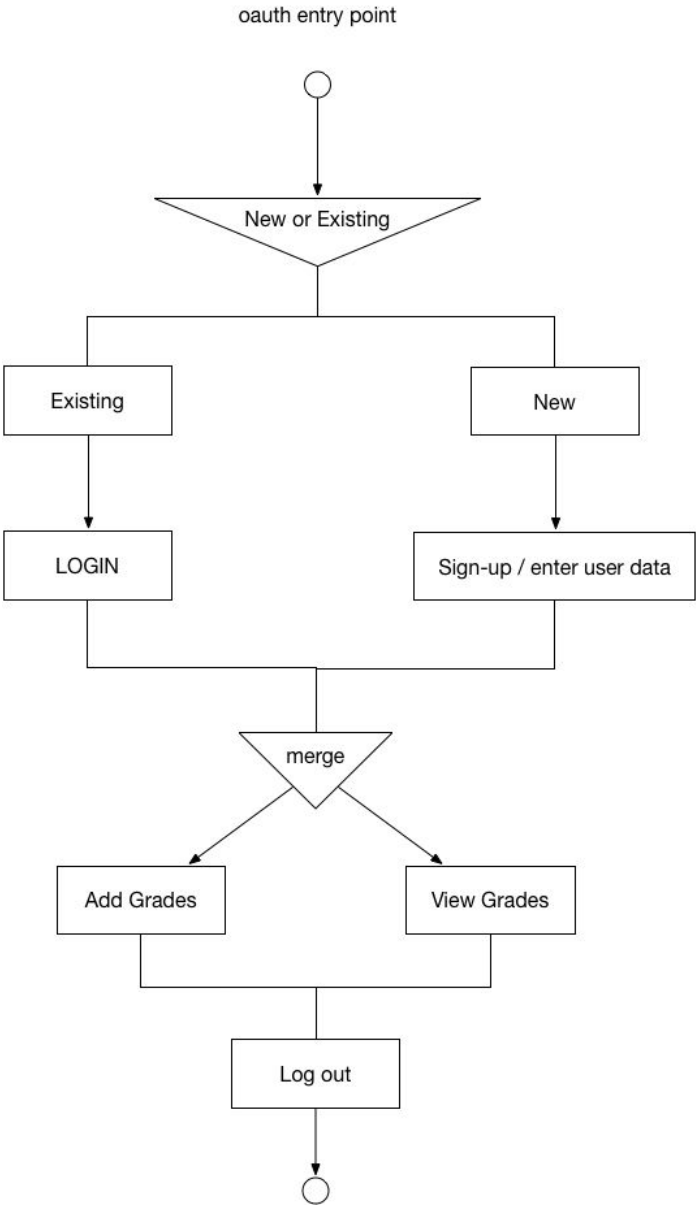
Languages/Tools/Frameworks:

- Python 3 (installed via Anaconda, for UI and data processing)
    - Numpy
    - Pandas
    - Flask/Jinja
- Twitter Bootstrap (responsive front-end framework)
- HTML / CSS / JS
- PyCharm (allows 1-click environment setup, server start-up, auto-formatting)

UML Diagram:


Student
token {PK}
gradYear
major
Takes
1..*
1..*
Course
courseID {PK}
profName
1..1
Has
1..*
Assignment
id {PK}
Score
courseName
Has
1..*
1..1
AssignmentType
name {PK}
weight

Activity Diagram:



Lessons Learned:

This focus of this project was to build an application that would be modern, flexible, and scalable. While we were able to get comfortable with MySQL during the semester, this project offered the opportunity to gain further experience using a NoSQL database. Due to the web-based nature of our application and future web-scraping needs, we opted to utilize MongoDB as it's flexible schema and JSON format nicely integrated with our proposed tech stack. Using the theory we learned while working with MySQL it was easy to adapt to the syntax of MongoDB and quickly get up and running. The application was written mainly in Python and the Flask web microframework. Throughout this project, we became comfortable developing within Python. We are confident that we conceptually understand how to develop an application that connects and interacts with a database. Coming into this project our group members had disparate areas of expertise: Christopher was proficient developing Flask based web-applications while Justin was knowledgeable in javascript and using numpy for data analysis. The combination of these skillsets made a great working environment to solve problems and quickly build a functional prototype. Having two partners with different skillsets made it easy to learn new tools and frameworks as we progressed through the development of the project.

Note: The peer-comparison ratio does not currently display an actual value. It contains a placeholder.

Future Work:

Gradr has the potential to develop into a popular tool for students to use in order to gain deep insights into their academics. Students can add information about themselves along with the courses they are taking in order to see their grades in courses.

While we are able to tell a user what their course grade is given assignment weights and assignment scores, we envision that this is just scratching the surface on the potential of this application. We look forward to gaining more information on students to provide further insights and functionality to users.

Students should be able to use Gradr throughout the semester rather than at just the end. If a student has provided scores for only some assignments, we want to provide the functionality to allow the user to see their predicted course grade based on how they have fared so far. We also want to provide the user with "What-If Scenarios", where Gradr can calculate the course grade based on given scores for the scenario.

On a higher level, we want users to be able to compare themselves to other students who were in similar situations. Without providing the names of similar students, we can identify similar students based on major, graduation year, and course load for a given semester. This can help students during course selection, as they can see how similar students fared based on the courses they are enrolled in.

Ideally, we would eventually like to identify a difficulty level for each course. This will most likely be derived from the number of hours a student dedicates to that course each week, and the average final grade in the course. Once we retrieve this information, we want to be able to calculate an adjusted GPA that more accurately represents the success of each student. As we know, not all majors are equal in

difficulty, and GPA is not always an accurate representation because higher GPA's may be easier to attain in certain majors. By calculating an adjusted GPA, students will have a better opportunity to see a more accurate representation of their efforts.