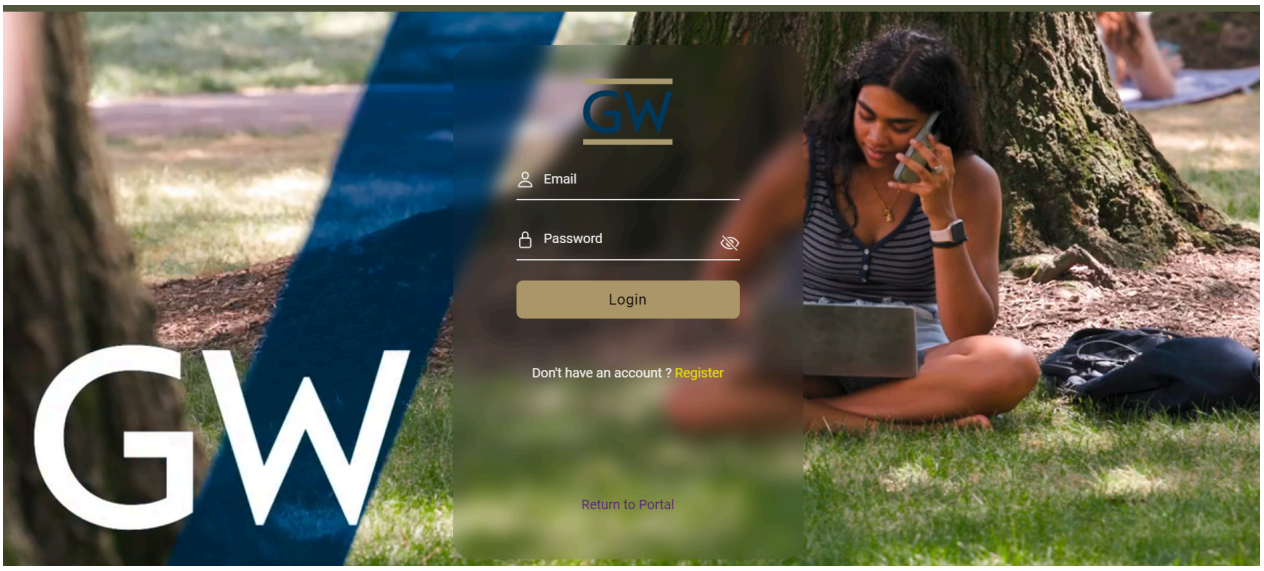
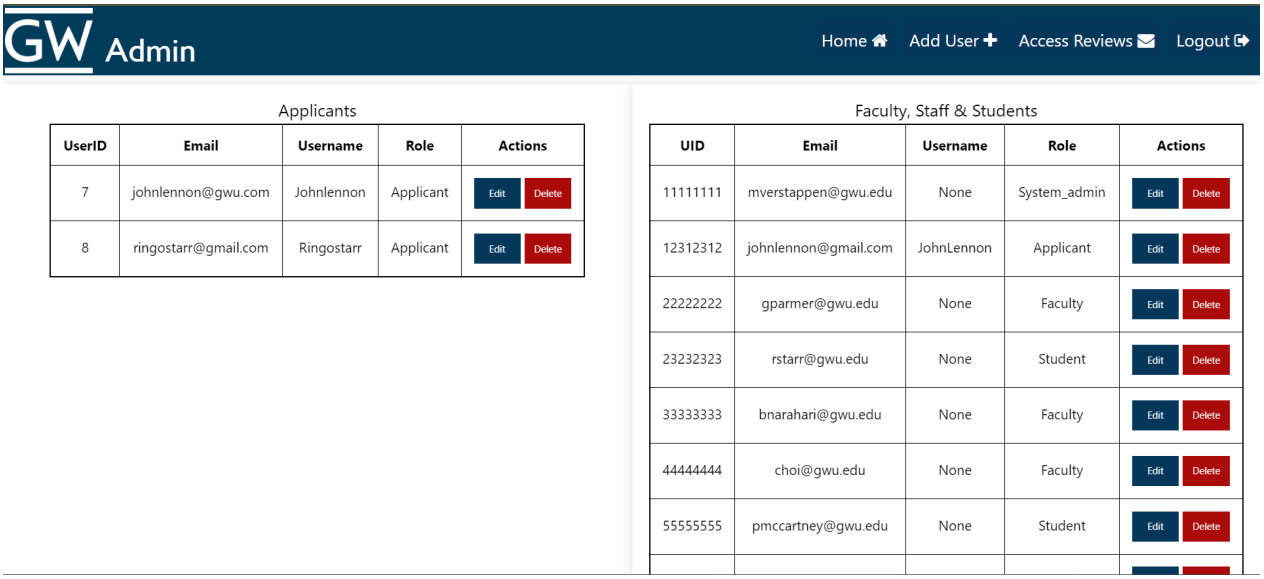
University Enrollment Management System Documentation

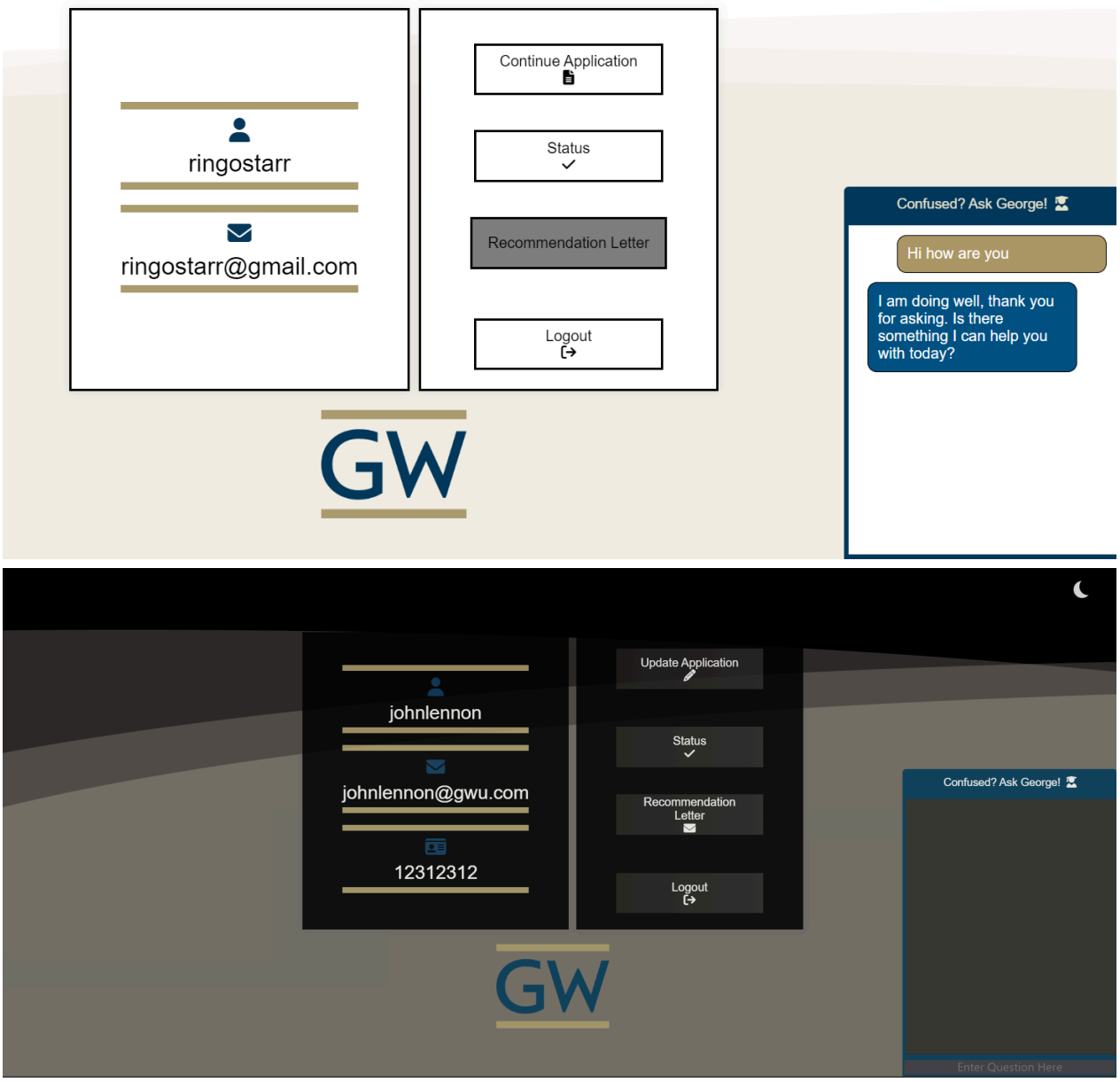
Python Flask, Amazon AWS, MySQL, CSS and HTML

Aaron Guo

Video Demo: University Enrollment Management System DEMO (SQL, AWS, Python Fla…



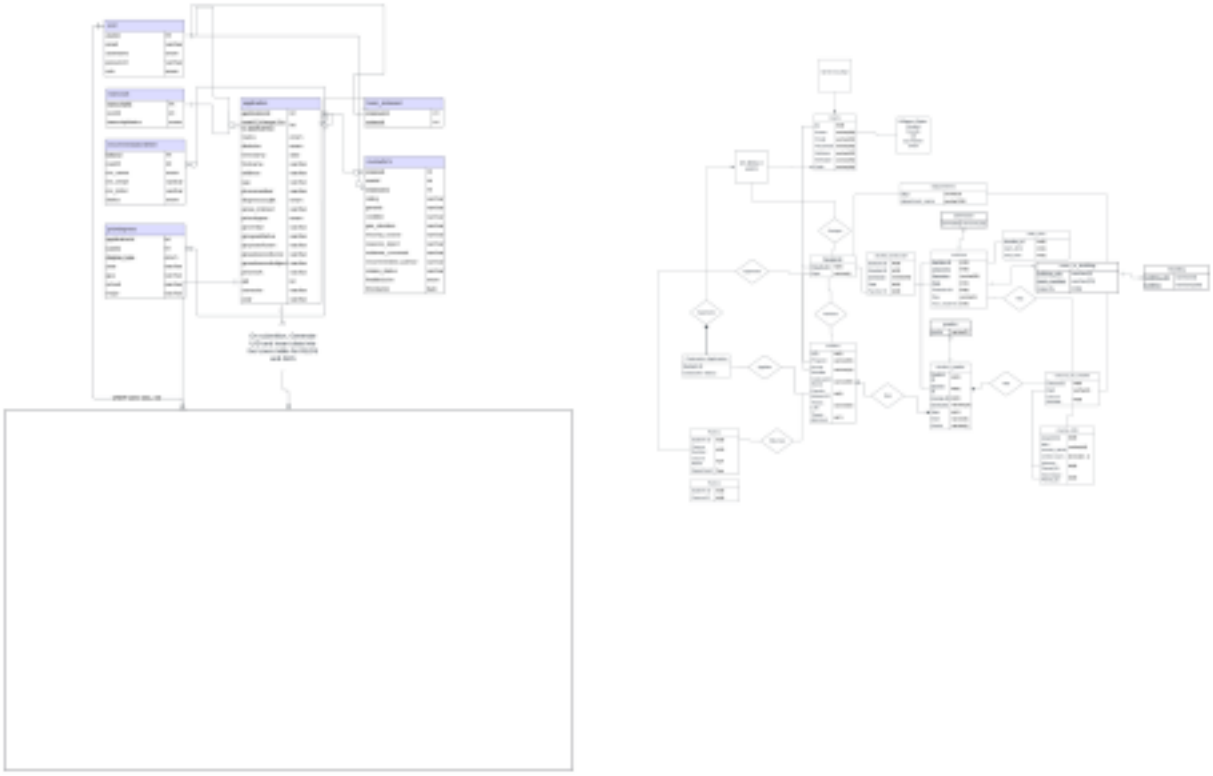




Faris Jiwad , Nathan Dixon, Gustavo Londono, Ahmed M.

Final Project Report

ER Diagram/Database Schema



Our database meets the requirement for 3NF, and we'll break it down step by step.

In order for our database to meet the requirements for 1NF, the database cannot have repeating information or multiple pieces of information in 1 field. In the entire database, the only time this gets slightly close is when a course has two prerequisite possibilities, but thanks to naming conventions (such as the fact that a course can have a primary prerequisite and a secondary prerequisite, if a primary does not exist, a secondary will not exist), the two fields are distinct enough that they don't get considered under repeating information. Therefore, the tables do meet 1NF.

In order for the tables to meet 2NF, no value in a table should be dependent on only part of a key that uniquely identifies a row. For most of the tables, there was only 1 value in the key,

and in the case of sections, the course_id, section_id, semester, and year cannot uniquely identify any one of its attributes without having the whole key. Therefore, all of our tables do meet 2NF.

In order for the tables to meet 3NF, no value in a table should be dependent on only part of a key. This directly ties non-prime to non-prime relationships. There are no instances in REGS, ADS, or APPS where there are non-prime to non-prime relationships, such as first name and last name derives key user information in the users table, as that information is stored separately in a personal information table independent of the user id. Therefore, our table does indeed meet 3NF constraints.

However, our table does not meet BCNF constraints due to a recursive relationship in the department table for REGS that existed in Phase 1. The documentation for this is listed as shown: It is not possible for some of our tables to reach BCNF as there are symmetric relationships, thus making it not possible for the schema to meet BCNF. An example of this is the departments table that contains (**Dept**, dept_name). Dept is the abbreviation for the name, dept_name is the name of the department. dept_name functionally determines Dept. It is not possible to store these two columns in the database while representing their relationship without them being in the same table, thus the decision to keep the data and only maintain 3NF was made. Maintaining only 3NF not BCNF also simplified some of our SQL queries, making them easier to understand at a glance.

Visual Overview of Different Phases

Recording 2024-05-01 092516.mp4 ADS/REGS Workflow

2024-05-01 13-55-29.mkv APPS Workflow

Design Justification

APPS had the least amount of connection that was required, as that was directly related to students actually entering the university and applying and being accepted. There was some integration that had to be accomplished for roles such as faculty, graduate secretary, and others, but for the most part, the core integration was independent of the rest of the system, so ADS and REGS had the most compromises that had to be made.

Upon initial completion of Phase 1, ADS had an initial schema where each student was assigned a transcript, and that transcript contained all of the courses that they had taken, as well as the grades submitted for it. What that means is that there was a student with information including their student ID, a table called transcripts which linked a student to a particular transcript, which contained a transcript ID, and the transcript ID that contained a bunch of courses.

In Phase 1 for REGS, this was formatted so that each student was linked to a particular section, called student_grades, and that student_grades table contained a course_id, the section, semester, and year of a particular course, as well as the grade that a student received on that particular course. This allowed for joins on many tables, including course_list_master, course_info, sections, section_instructors, and other tables, as well as showed the same basic information that ADS needed.

Therefore, to support the integration of the two systems, the tables were migrated to the REGS thought process and all of the queries were adjusted from the ADS side in order to support this change. Other than that, the ADS tables, APPS tables, and REGS tables all lived in isolation and cooperation with each other, which made integration much more seamless in terms of

frontend and backend development.

Special Features

- **AI Chatbot System**: Based on Google Gemini, a Python library and an API key later, “Ask George” allows both students and applicants to the university to ask basic questions, such as how can a user register for a course, why isn’t a student able to register for a course (could be for a variety of reasons such as lack of Form 1 submission or lack of prerequisites) or how to move the process of application along. Additional features such as future courses a student can take depending on courses that they’ve already taken is also an additional query. -Aaron
- **Email Recommendations**: Upon request by a student/applicant, an email will be sent to an end user where they can submit a submission for the student’s recommendation letter that will be included in the application of a student in the system. - Aaron
- **Darkmode:** On the Applicant Side, Light/Dark Mode exists with a single click on the top right corner. - Aaron

- **PDF Download of Transcripts and Review Forms:** For Students and the CAC, they are both able to download files directly to their computer with a single click. Students can download their transcript and CAC can download Review Forms. - Aaron

- **Course Catalog:** When students need to add courses to their Form 1, they can choose to either enter the course details manually, or choose from a list of all courses in the catalog. When selecting a course, checks to see if the prerequisites are met or if the course has already been taken are run. If both checks pass, the course is added and the Form 1 table is updated. - Faris
- **Gmail Recommendation Letters:** Applicants can send up to 3 recommendation letters and they will be emailed to the email they choose, that recommender can fill out the letter with the link. - Aaron

Work Breakdown

Aaron Guo:

In Phase 1 my part was APPS. My primary role in terms of basic functionality was to process new applicants' data and upon acceptance, enrollment, and matriculation. This includes inserting it in a way to ADS+REGS so they could process the new student. Besides the basic functionality I did a majority of the CSS and Javascript (*Flash Disappear, Animated Collapsible Chatbot, Fixed ADS JS for Apply for Graduation, PDF Download, Login Page Password, Darkmode Saved Locally)* related to the CSS *(applicants, students, faculty, admin, staff etc. as well as all child templates for those roles)* for the group, I also implemented new features such as gmail based recommendation letters, darkmode, downloading documents as PDF and discussed extensively possible ideas for “bells and whistles" for their end. I also implemented most of the integration for the functionality of the Graduate Secretary and Admin pages (Adding Users, Deleting Users, and Editing Roles, Nate did the Students Functionality for GS). Besides new implementations, I came up with many ideas that could take our project to the next level such as

a course catalog that would add classes to the form1 once you click for a more intuitive and user-friendly advising process (While Nate Implemented it). Another idea I discussed with Faris was the AI chatbot for ADS+REGS that he modified for students. I also added many of the initial tasks for all of our roles onto the trello board to make sure we were on track.

Nathan Dixon:

I worked on ADS in Phase 1, which consisted of student, faculty, GS, admin, and alumni portals. The student portal was the most in-depth and allowed students to view their current courses, update their information, add courses to their Form 1, and submit a thesis if they were in a doctorate program. Once the Form 1 and thesis were approved through their faculty advisor's portal, they could then apply for graduation. They were then approved for graduation by the grad secretary. If they logged in after this, they would then be taken to the alumni portal where they can view their transcript and what program they were in. My workflow had the most overlap with REGS, so we had to figure out how to merge our tables together. Since there was much more information stored in REGS' course tables, (such as sections, times, and buildings) I decided to update my queries to reference the information stored in these tables. Instead of storing all course information in one table and having a table that linked users to transcripts, we stored all courses being taken by users in one table and used their user ID as the identifier. This ensured that courses with the same name or course ID would not cause issues. There were some other minor requirements that I had to figure out while integrating into REGS, but this was by far the largest hurdle in terms of getting the stored course information. Integrating with APPS was a much smoother process, since we just had to ensure that a user had all the information already stored and ready to transfer to the student table when they were accepted and promoted from

applicant status. We also used most of the APPS styling since it was the cleanest of the 3, which made it much easier to just implement the required functionality. Overall, Aaron, Faris, and I did a good job of tracking our progress and deciding on design choices. Our 3 workflows worked together very well and we had consistent styling and functionality across all of our portals.

Faris Jiwad: I was the REGS person on the team, so my job included ensuring that the workflow of registering for courses, submitting grades as faculty and GS, and all of the checks that were required was implemented into the new system. Firstly, I had to ensure that the merge of databases between REGS and ADS was smooth, due to the fact that many parts of our system were different, so I had to figure out a way to merge the two without adding too much work overhead. Additional functionality I was a part of was checking to ensure that a student had submitted a form1 before registering for a course to ensure that the client was pleased with the groundwork for the system. I also created the Registrar route, which was solely in charge of creating new courses and updating grades for all of the students at the University. In order to add a new course, a check has to occur that determines if a course with the same department and course number exists. If it does exist, then the course cannot be created, otherwise the course creation can proceed. Adding courses included checking if a department and course name combination already existed in our system to ensure that there wasn't any redundancy in our system. Finally, I also implemented some of Aaron's chatbot functionality into the student route by adding some sample queries, such as suggested courses that a student could take in the future considering what courses they had already satisfied at their time at GW, as well as support for why students weren't able to register for courses and where they could find information for certain things. Aaron's leadership and organization made it extremely easy to integrate all the functionality of REGS into the system, as separating queries and HTML renders made it very

easy to bring multiple pieces from our many projects together.