Duke Conversations

CS 316

Noah Burrell, Anne Driscoll, Kimberly Eddleman, Summer Smith, Sarp Uner

Application

The Problem

Duke Conversations is a program run through the provost's office that holds dinners twice a week with faculty members as a way to bring the Duke community together. The goal of the program is to create small group interaction between faculty members and students and to build an atmosphere of academic engagement.

Currently the program accepts dinner applications through google forms, where individuals on the planning committee accept or reject dinner applicants by reading all the interest statements through a google sheet and manually respond to each request. Our product will make this process more fair and streamlined by providing the netID of students rather than just their name, as well as the ability to look up how many dinners a student has applied to, how many he has been selected for, and how many of those he has attended.

The App

The application we built will help the organizing team keep track of attendees. By keeping count of how many times each person has applied, if they've missed previous dinners, and their thoughts on the program, we'll be able to ensure that the selection process is standardized, and the team won't have to worry about equity across dinners they aren't overseeing. Effectively the app is intended to make the selection process easier for administrators, as well as build a system that gives administrators the necessary information to select fairly.

To do that, we created a login process for administrators and applicants, a way for people to apply, and a way for planners to select applicants based on their past data.

Database Design

The project overall aims to collect data on dinner applicants in a meaningful way to aid in organizing better Duke Conversations dinners. To do that we have the following organization of data:

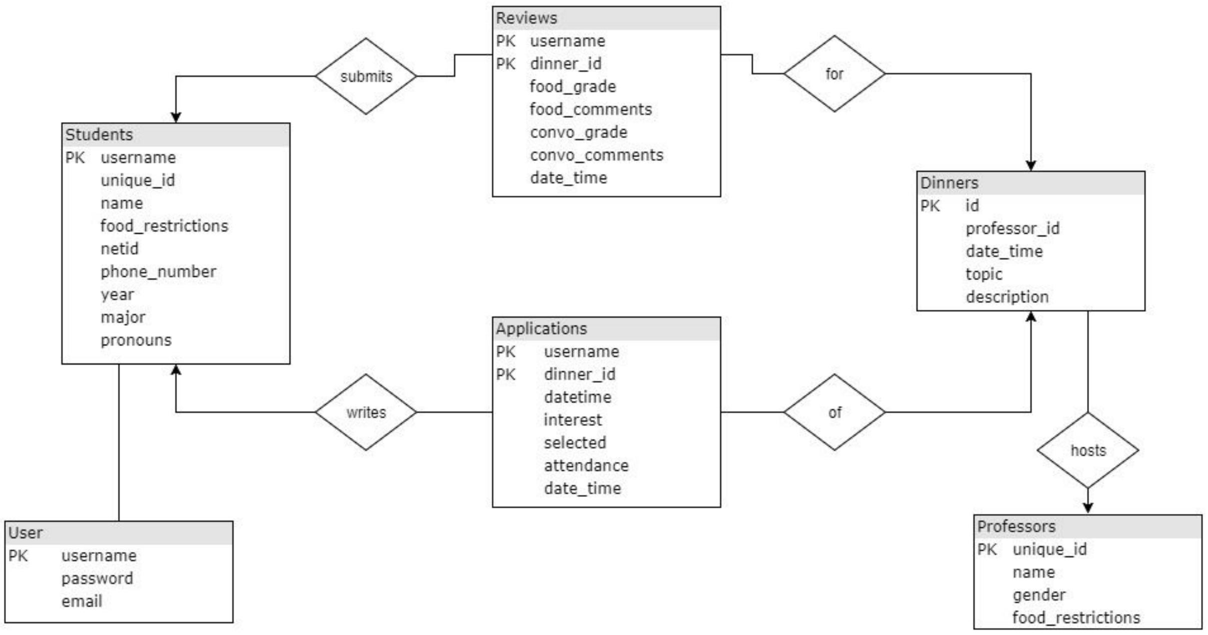
User authentication data collected is minimal, only a username and password.

Connected to user authentication information by username, we have general user information, including Unique ID, NetID, name, food restrictions, phone number, year, major and pronouns.

This student data then connects to data on the two form types students can submit on the site. This includes dinner applications, which student applied to which dinner, and their interest in the dinner. The other form type is reviews, which include which student reviewed which dinner, and any comments they had on it.

Admin managed data is focused on the planning aspects of the dinner. Administrators can add professor information, including Unique ID, name, gender, and food restrictions, as well as the dinners that students can apply to. Dinners are linked to professors through their Unique ID, and are linked to applications through a generated dinner ID.

The ER diagram for the database is provided below



Data

We have compiled the applications from a few of the dinners last semester, totaling 600 applications. This sample set includes variables for all the past questions, with NA's for the applications that were not asked that question originally. As questions have been phased in over time, there are many individuals that have answered different questions at different times. For occasions where an individual has given different answers at different times, the most recent answer was chosen.

We are began by using a simple test database while we build the platform, which contained several items in each table, and was generated to resemble the types of inputs that we should see in reality.

Later in development, we reformatted the data from the old applications to conform to our schema, and stored it in a way that can be uploaded in bulk to our application, which has become our production dataset.

Platform Choice

Overall Implementation

We chose to implement the web app in python using Django. We chose Django over Flask because it provides many of the features we needed built in. Django allows us out of the box to create user logins, and provides preliminary admin data editing capabilities.

The front end is primarily done using an adapted version of bootstrap themes to create a responsive user interface.

Database Implementation

As we are using Django, we have to use the built in system for data storage. Though ultimately queries are enacted in SQLite, the schema is created through a Django specific set of model definitions. With the created model definitions, Django generates SQL code to create a database based on the model implementation in Python.

Due to the differences in Django implementation, details of the model are adapted to the abilities of the model definition system. Django has several limitations that aren't present in SQL. For example, each 'model' in Django must have a singular primary key. As such, we have implemented a unique together method, along with an index, so that the tables that require keys over multiple variables can still enforce their uniqueness.

Django has more fundamental limitations in terms of constraints on input data. In this application there are many constraints on when data is appropriate to input. With primary keys we can enforce that only the correct form of references can exist in certain tables, eg. only students can sign up for dinners.

Other things, like students only being able to review dinners that they have applied for, been selected for, and attended, Django is unable to implement. Given the hierarchical nature of our data, and the inability for Django to handle it, we found that enforcing these relationships outside of the database level was effective. Nearly all data is input by student users, and can be managed by regulating the options available to them.

On each page, only the relevant options are displayed to the student. For example, on the review page, students are only given options to review dinners that they are confirmed as having attended.

App Usage

User Side

Users are able to create an account from the home page. Creating an account will prompt them to create a profile, which includes information such as name, unique ID, netID, and several other fields that are then added to the database. If they already have an account, the user can log in as a student, admin, or reset their password. Once the student has logged in, they have the option to Sign up for a dinner, Review a dinner (if they have attended a dinner in the past that has not yet been reviewed), edit their profile, or logout. Sign-up for a dinner, Review Dinner, and edit profile are forms that update the database and are used to determine what shows up on the other pages (e.g. the sign-up and attended tables are taken into account on the review dinner page). Additionally, an email system has been implemented that allows students to confirm that they have signed up and later know if they have been selected.

Admin Side

After logging in, an admin has access to applications, dinners, professors, reviews, students, selection, and attendance. The pages for applications, reviews, and students are read-only, since administrators have no need to be able to change this values; the are editable by the user accounts.

The professor and dinner pages allow administrators to add or delete a professor from the database, and to create (or delete) a dinner event for any professor existing in the database.

The selection page allows administrators to locate a dinner by searching for a professor’s name or the date of the dinner, or filtering by a combination of professor name and dinner date, review relevant information from applications, including year, major, interest in the event, and some statistics about their participation in the program, like number of applications, percent of dinners that they applied to for which they were selected, and the number of dinners they were selected for that they actually attended, and select students for the relevant dinner. The selection process automatically notifies students via email of their status as selected or not selected for a dinner.

The attendance page is designed to serve as an attendance sheet for those selected for a dinner (administrators can again locate a dinner by searching for a professor’s name or the date of the dinner, or filtering by a combination of professor name and dinner date). The page displays only the students selected for a dinner, along with relevant information like name, preferred pronouns, phone number, and food restrictions, and allows the administrators to mark attendance for each student who participates in the dinner.

Each admin page also has export functionality, so that administrators can select and download csv files of the information on each page, so that the Conversations program can continue to do data analysis on the information collected by our application.

Differences From the Proposal

We were able to implement everything we had originally hoped to in our original project proposal. The only difference between the proposal and our final project, is that we did not implement an algorithm that automatically selects students to attend dinners based on how many dinners students have applied for and how many they have been selected to attend. We decided not to implement this feature because it is not a fair method to select students. An algorithm would not be able to take into account the description of why a student is interested in a particular dinner, something that enables Duke Conversations to choose those students who would help make the dinner conversation the most stimulating.

Potential Future Improvements

Some additional admin functionality, and finesse in handling data would improve the app. Additionally, there is some neatening of code, and cleaning of the interface that would improve the potential longevity of the project.

Current open 'stretch goal' issues, that will be pursued after the semester ends, include:

- Reworking the current html structure to take advantage of the Django template inheritance structure. This would make the codebase easier for future users to implement and adjust.
- Create a custom user signup form, in order to allow the user management parts of the site to conform to the styles of the rest of the app.
- Allow sorting by calculated columns on the admin site. This will help administrators better use the calculated fields (such as % of applications selected, % of selected attended) to select students appropriate for each dinner.
- Considering the potential to restrict dinner signups to those who have reviewed all the dinners they have already attended.
- Customize the admin selection page to be able to filter by dinner in a more user friendly way.