CISS430: Database Systems
Project

The goal is to develop a web application.

You should use a cloud-based python application hosting web site. Right now we are using `pythonanywhere.com`. See https://en.wikipedia.org/wiki/PythonAnywhere. The free account is probably sufficient. It's possible to upgrade the account if you want to keep working on the project after the course and would like to have more storage, CPU time, etc.

This will involve building a web application with database storage in the backend. There will be a minimalistic list of requirements so you can be free to do what you want. However this is a project for a *database* course. So obviously there will be some strict requirements for the database side of the project. You will present your work during our CS Presentations (usually on Thursday of finals week).

The system must be sufficiently complex. It's hard to define complexity. There are technical ways of defining software complexity but some of them don't work all the time. It's better to measure complexity at least in terms of features and in terms of software object relation/interaction patterns. But once I see your project proposal I would have a clearer idea of whether it's complex enough or not. For sure playing a game of tic-tac-toe is not enough. The following are some vague guidelines:

- The database must contain at least 10 tables. The table relations must be sufficiently complex and must exhibit one-to-many and many-to-many relations. The database must be well-designed. (In reality, it's very likely that your database will have more than 10.)

- You must write your own SQL. In other words, you are not allowed to use software libraries that perform SQL code generation, such as an ORM library. However, you may write your *own* code to organize the generation of SQL statements. Because of the requirements on relationship between tables (see above), besides queries on individual tables, there must be queries involving joins.

- The backend code must be at least 1000 lines. (And I mean the python code, not your HTML.) Lines-of-code does not really measure software complexity. But this is better than nothing. (In reality I fully expect your project to go way beyond 1000 lines.)

- User inputs must be checked and sanitized to prevent SQL injection attack. You can for instance restrict an input to a set of ASCII characters. For instance your team might impose the condition that a user name can contain only letters and digits.

- Your web site must be reasonably responsive. In particular SQL statements should be reasonably efficient. For instance if you are using two columns from a table, there no reason to fetch five columns from that table; if you are displaying 10 rows from a table, there's no reason to fetch 100 rows from that table.

- You must include an ER diagram of your database design.

One component that is required is user management.

- Users of your web application must go through a user registration process. This must involve email confirmation. Python code for sending email through a gmail account will be provided.

- You must impose some password security measures: The length must be sufficiently long (you decide on the length) but not unreasonably long and there must be enough different types of characters in the password. There's no need to check the password by going through a brute force attacks on it – but you should at least spend one hour reading up on this topic on your own. There's no need to force users to change their passwords at regular time intervals.

- You must impose some basic security measures on password storage and authentication. The password provided by your user must be salted and hashed before it is stored. The salt must be sufficiently long and computed using a cryptographically secure pseudo-random number generator (python's `os.urandom()` is enough for this project). Salt must not be reused. Hashing must be done with a standard cryptographic hash function. For instance SHA256 is enough. (But you should know that there are stronger cryptographic hash functions.) The salting and hashing must be done a reasonable number times – you decided. Since you are using a free account, don't overdo the number of rounds. You need not use a slow hash function.

- User password reset should be done with email authentication with a random token that is sufficiently long and have an expiration of, say, about 10-20 minutes.

- There are of course many other security issues you need to be aware of. But as this is a class project, there must be a reasonable limit to how secure your system can be. The above are some basic requirements. If you want to add more of your own security measure, you are welcome to do so.

- Technically speaking when sensitive data is sent between the browser client and the server, you should use HTTPS. But I leave that to you. Just be aware of that. (Spend an hour reading up on HTTPS when you have time.)

If you want to use a tool/library, you should first ask me. For instance it's perfectly fine if you want to learn to use a library to help generate HTML (these are called templating

engines). You can use javascript or AJAX libraries as long as you are not violating any of the above rules.