NYU Tandon School of Engineering April 28, 2021
Computer Science and Engineering
CS6083, Spring 2021

# Project #2 (due around May 15)

In the second part of the project, you need to create a web-based user interface for the database designed in the first project. In particular, patients should be able to register, login, and edit their availability and other information. They should be able to accept, decline, and cancel any appointments that are offered to them. Providers should be able to sign up, and to add appointments to the database. (You may assume that providers cannot cancel appointments once they are uploaded, but only patients can.) Also, the system should automatically and periodically assign available appointments to suitable patients, meaning patients that are eligible, live within a certain distance to the provider, and are available in terms of time slot. You should assume that in this prototype system, patients will only see appointment offers the next time they visit the system, though of course a real system would notify them via email or text message.

Note that you have more freedom in this second project to design your own system. You still have to follow the basic guidelines, but you can choose the actual look and feel of the site, and offer other features that you find useful. In general, design an overall nice and functional system. If you are doing the project as a group of two, note that both students have to attend the demo and know ALL details of the design. So work together with your partner, not separately. Also, slightly more will be expected if you are working in a team. Start by revising your design from the first project as needed. In general, part of the credit for this project will be given for revising and improving the design you did in the first project.

The followings are a few suggestions for the interface you might want to build for this project. After users log in, the may come to an initial page that would show any appointment offers they might have, or any appointment that they have already accepted, maybe together with information such as the distance to the provider. After the user clicks on an offer or an accepted appointment, they can decline or cancel. They should also be able to follow a link to another page where they can edit their schedule and other information. Providers might first land on a page that shows some summary information such as how many appointments have been accepted, cancelled, etc. Then they should be able to input additional appointments, or list all appointments by certain fields and tags -- e.g., list all accepted upcoming appointments sorted by appointment or acceptance time and date, or list all cancelled appointments by time and date of the cancellation. You can decide what is the best interface for such as system.

You should spend some time thinking about how to best assign appointment offers to patients. Given sets of available appointments and eligible patients, you should come up with an algorithm that tries to maximize the number of patients that can get an appointment offer. In fact, you should probably maximize the number of patients of the highest priority group that get an appointment offer, and only assign any unmatched appointments to lower priority patients. This is non-trivial – for example, by assigning an appointment x to user A, you might make it impossible to give an appointment offer to a user B whose only possible match in terms of time and location is appointment x. However, if you were to offer x to B, there might be other suitable appointments for A that would otherwise be unmatched. Formalize this as an optimization problem and design a suitable algorithm for getting a good matching!

Users should be able to perform all operations via a standard web browser. This should be implemented by writing a program that is called by a web server, connects to your database, then calls appropriate stored procedures that you have defined in the database (or maybe send queries), and finally returns the results as a web page. You can implement the interface in different ways. You may use programming languages and frameworks such as PHP, Java, Ruby on Rails, Python, NodeJS, or Golang, to connect to your backend database. Contact the TAs for technical questions. The main restriction is that the backend should be a relational DBMS with the schema you designed in the first part, with suitable improvements as needed.

Your interface must take appropriate measures to guard against SQL injection and cross-site scripting attacks. To prevent SQL injection you may use stored procedures and prepared statements (if your programming language supports them). If your language does not support prepared statements, your code should check and sanitize inputs from users before concatenating them into query strings. To guard against cross-site scripting, outputs to be returned to user's browsers should be checked or sanitized to avoid scripts. Some languages provide functions, such as htmlspecialchars in PHP, to help with this. You should also define appropriate transactions to make sure that multiple users can use the site at the same time. Make sure to address these requirements (protection against SQL injections etc., and concurrency) in your final document.

Every group is expected to demo their project to one of the TAs at the end of the semester. If you use your own installation, make sure you can access this during the demo. One popular choice is to use a local web server, database, and browser on your laptop (in this case, your project can just run locally on your laptop). Also, one thing to consider is how to keep state for a user session and how to assign URLs to content – it might be desirable if users could bookmark a page for a project or issue, or the result page of a search that was performed. Grading will be done on the entire project based on what features are supported, how attractive and convenient the system is for users, your project description and documentation (important!), and the appropriateness of your design in terms of overall architecture and use of the DBMS. Make sure to input some interesting data so you can give a good demo.

Describe and document your design. Log some sessions with your system. You should also be able to show and explain your source code during the demo. The documentation should consist of 15 to 20 pages of carefully written text describing and justifying your design and the decisions you made during the implementation, and describing how a user should use your system. Note that your documentation and other materials should cover both Projects 1 and 2, so you should modify and extend your materials from the first project appropriately. There will be opportunity to get extra credit by implementing cool extra features, but extra credit is limited to about 5-10% (and the TAs will decide what is cool). There may also be extra extra credit of up to 5% for doing an early demo before the deadline.