A web app for preference-based matching

Petros Kitazos 2526547k

December 15, 2023

1 Status report

1.1 Proposal

1.1.1 Motivation

This project, as the title suggests, revolves around the process of building a web app to manage instances of student-project allocations. Student-project allocation involves fairly allocating the projects available to all the students in a particular group, while taking into account their preferences over said projects as well the project and supervisor capacity constraints. This is a process undertaken in educational institutions all around the world, all with unique variations and constraints applied to this problem. That is why it's critical that the web app is flexible enough to allow for any variations and constraints to be respected, while keeping the allocation process simple and straightforward.

1.1.2 Aims

The completed web app should be able to host the entire project allocation process; from project collection to publishing student-project allocations. The platform should allow for all critical parts of the allocation, for all three types of users:

- **students** can view all projects and submit a ranked preference list.
- **supervisors** can submit the projects they wish to supervise and view the students allocated to each of their projects
- **admins** can run different algorithms on the students' submitted preference lists and select an optimal matching to publish to all users

1.2 Progress

- rebuilt existing web-app using modern web framework (NextJS, Typescript, tRPC, TailwindCSS)
- performed requirements gathering and prioritisation
- modified database schema to capture new requirements
- created new sitemap and lo-fi wireframes for new pages
- built basic UI skeleton for all new pages

- implemented role-based access control
- modified matchingproblems package to be called programatically on a dedicated server
- implemented all required matching algorithms (generous, greedy, minimum-cost, greedy-generous)
- created population seeding script to add testing data
- implemented communication between web-app and matching-server
- worked extensively on admin-panel: admins can run matching algorithms, view summarised and detailed versions of a given matching and select and publish a given matching to all users
- started working on stage-based access control i.e limiting what actions users can perform based on the stage of the allocation instance

1.3 Problems and risks

1.3.1 Problems

There have been a couple problems that kept re-appearing throughout the course of the semester.

- The first problem to persist over several weeks was to do with seeding the database. Before learning about seeding scripts, I was attempting to populate the database via a custom dev page I built into the web app that allowed me to run several database mutations sequentially. Because of the volume of data I was trying to insert at once I was facing time-out issues that were non-deterministic. I rewrote the population script several times, before reaching the final version which is a node script I run separately to the web-app dev server. This has solved the non-determinism problem.
- The second problem I've had that has been slowing down progress is that I'm having to make changes to the database schema quite often. As I slowly work my way through the required features I identified, I'm discovering more restrictions or relationships that I need to be represented in my database. Changes to the schema however are expensive because my internal data-fetching API is very tightly coupled with my database ORM, so any time I update the schema I risk breaking some of my API endpoints. I've identified two possible solutions to this problem:
    1. creating a data-access layer between the database layer and my internal API. This way when I make any changes to my database schema I only need to worry about updating the database layer to reflect these changes. I won't need to manually test all my API endpoints to ensure that they still work. The API endpoints will depend on a fixed interface defined in my data-access layer.
    2. creating a test-suite to be ran after any database schema changes to ensure that the functionality of all my API endpoints remains intact.

1.3.2 Risks

There are two areas that may potentially cause problems in the next semester.

- The first one is to do with using the university single sign-on for user authentication, as I've not worked with OIDC before. I've been supplied with a document that explains how to set up a particular auth flow which I will study more thoroughly next semester in order to get authentication up and running.

- The second problem is to do with creating a test-suite for the web-apps internal API. I haven't written unit or property tests for an application written in Typescript before, so it would be difficult to determine how much time it's going to take away from the development of new features. I'm aware of a few testing frameworks that I plan to study in the second semester in order to help with the creation of my test-suite.

1.4 Plan

- **Week 1** stage-based access control
- **Week 2** Student/Supervisor user flows
- **Week 3** creation of test-suite
- **Week 4** instance forking
- **Week 5** focus on UX/UI; ensure interface is accessible and design is consistent
- **Week 6** create and run evaluation surveys
- **Week 7** analyse findings and evaluate application
- **Week 8** begin dissertation writing
- **Week 9** work on dissertation
- **Week 10** finalise dissertation and work on project presentation
- **Week 11** Dissertation submission deadline and presentations.