The Course Swapper
A Project Proposal by Kristine Cho, Alon Lapid and Sam Ness

**Description**

We propose the implementation of an OLTP database system called "The Course Swapper" that will allow students from the Claremont Colleges to swap sections in an organized manner. During the initial registration period, students are unable to enroll in their first choice of courses for a variety of reasons, such as the section being full by their registration time. They then enroll in other sections in order to avoid part-time status. This problem cascades, causing many students to enroll in sections that they are not interested in or in an inconvenient section. According to our survey that we created to assess user opinions regarding this proposed tool (see Appendix), 36.4% of respondents said that they have frequently enrolled in courses that they did not want to take, and 45.5% of respondents said that they have frequently wanted to swap out of a section. There is a clear demand for the ability to swap courses with other students.

Currently, students use social media to ask other students to trade sections, but this is unorganized and risky, as a third party may enroll in either course before the transaction is complete once the parties drop their respective sections. This database system will allow students who are unhappy with their section enrollment to trade their sections with other students, thus increasing the satisfaction of students at the Claremont Colleges.

**Functionalities**

The Course Swapper will include various functionalities that will allow students to trade sections conveniently and privately. 54.5% of survey respondents said that they would prefer anonymization during course swapping. The core functionalities will thus allow students to swap sections while maintaining user privacy and to perform basic search operations. The core functionalities thus include:

- Search for courses being offered by course ID
- Post an anonymous swap request
- Conduct these swaps (i.e., change the student enrollment of the students involved) without revealing the information of each student

In the survey, students ranked the instructor (90.9%), the requirement fulfillment (81.8%), and the days that the section is taught (72.7%) as the top three features that they look at when looking for courses. For these reasons, additional functionalities of the Course Swapper will allow students to search for swap requests using more search criteria. Students might have ideas about what course areas, departments, or times they want to look for without knowing which exact course they want. Additional functionalities would thus broaden the search functionalities to include:

- Search for courses being offered by
    - department
    - time and day
    - campus
    - course area

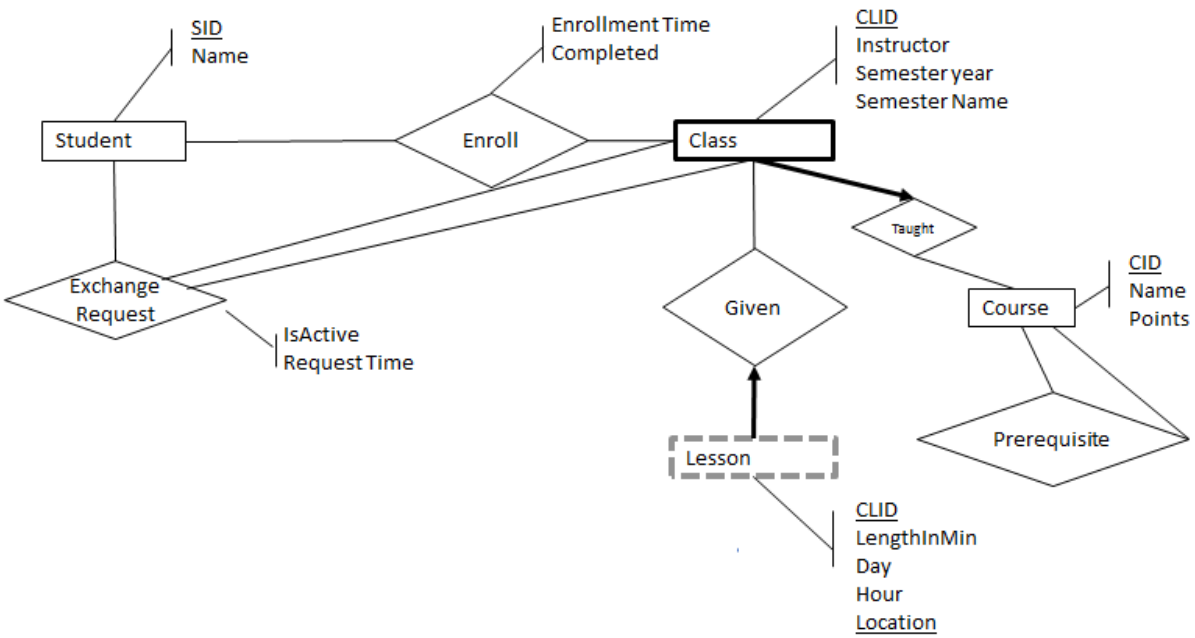
The majority of survey respondents (54.5%) expressed that they would prefer for this feature to be built into the Registrar system, while 45.5% of respondents said that they would like the option to use a chat feature. We thus envision this system being extended in the future by adding the following functionalities:

- Integrate the system into the college's Registrar system
- Add an optional chat feature
- Integrate ASPC reviews when looking for courses

**Resources**
We will use a relational schema from the Registrar system, including schemas for course information and student enrollment information. We plan to use the SQL Server hosted using Microsoft's Azure, Python, and pyodbc.

**Database Modeling Design**
The core key decision was to have key entities:
Course - Describes the topic that is taught and has ID (like CS133) and a name.
Section- the instance of a course that is given in a specific semester by a specific instructor and a set of lessons assigned to it.
Lesson - The time and location of a given Section. Section can have 1 or more lessons. It is a weak entity dominated by the Section- entity. It is needed to validate that a Section- swap does not create a schedule conflict. Another decision is privacy and fairness of the process. The user should be able to see only her enrolment and no other users' enrollment.
She requests a section to be swapped with another section and the request is logged into the system. The registrar is the one who finds the matches between the students. This prevents the situation where a student with a rich network of friends would get a priority over a student who has less social connections.
Main scenario: Student logs in and watches his current enrollments, previous course credits and searches for the sections offered in this semester. He submits an exchange request providing a section he is enrolled to and a section he would like to join. The system checks the student has the prerequisite and that the new section will not cause a schedule conflict. If validation is passed, the request is logged and marked as active. The user can also view other exchange requests made by other students (their identity is hidden ) and adjust his previous request to match better what is actually available by cancelling and resubmitting a new exchange request. The registrar logs in to the system and views active requests, he selects to find matches (this could be done also in the background) and a list of swap matches is presented. He executes a swap request and the section are swapped in a transaction with the relevant exchange requests marked as completed.

Below is the project ER diagram :




**Business Logic**

Show Enrollments – show the current section the logged in student is enrolled in (no information about other users is exposed)
Show Credits – show the list of courses the logged in student already completed
Show Section- – show the list of section- that are offered in the semester
Show Lessons – show the list of lessons for a given section
Show prerequisites – Show courses and their prerequisites
Show Exchange Requests – the login user can see the status of their own exchange request. The registrar can see all exchange requests.
See the full set of commands in the appendix.

Request Exchange – request exchange of a section with another section. These rules are validated:

- The sections are offered in the semester
- The student has the prerequisite for the course of the section he would like to enroll to
- There are no schedule conflict to the student schedule by adding the requested section
- There is no other pending exchange request for the logged in student
- The student is enrolled to the section she wants to give up
- The student is not enrolled and did not complete the section she wants to enroll to

Cancel Request - cancel a pending exchange request.
Find Matches – The registrar can view all the pairs of exchange requests that are matching which means :

- 2 different students requested to swap a section

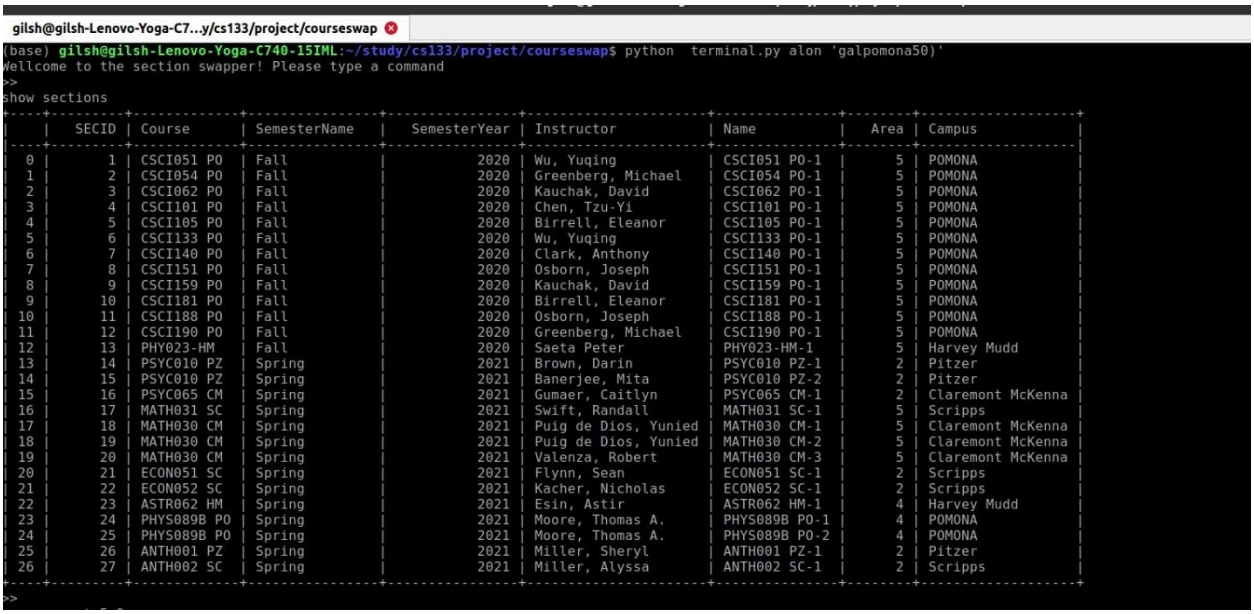
- The section that one student wants to give up and the section that the other user would like to enroll and via versa
- The course pre condition rule are not violated
- Time conflict will not be created because of the swap

Execute matches – The registrar executes one or more matching pair of requests in a single transaction and the sections are swapped between the students. If there is more then one way to execute swap on a qualifying match, the swaps will be chosen randomly.

Note: We decided not to allow a student to have more than a single pending request at any given time. First this is reasonable from fairness perspective and second it creates complication in case that executing one swap request, invalidates another pending swap request.

**Interface Design**

After struggling with html UI using PHP without success, we decided to change direction and implement the user interface as textual terminal written in python in which the user type a textual command to the server and get a textual information back. This enables us to meet the project schedule and also to enable a very flexible and rich search capabilities on top of the data. The terminal has 2 basic commands:
1)show command that enables the user to view a specific entity like section, courses, lessons and so on
2) exe command that enables the user to execute a task that has side effect, namely update the database. For example, submitting or canceling exchange request and executing the sections swaps are performed using the exe command.



gilsh@gilsh-Lenovo-Yoga-C7...y/cs133/project/courseswap

```
(base) gilsh@gilsh-Lenovo-Yoga-C740-15IML:~/study/cs133/project/courseswap$ python  terminal.py alon 'galpomona50)'
Wellcome to the section swapper! Please type a command
>>
show sections
```

| | SECID | Course | SemesterName | SemesterYear | Instructor | Name | Area | Campus |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | CSCI051 PO | Fall | 2020 | Wu, Yuqing | CSCI051 PO-1 | 5 | POMONA |
| 1 | 2 | CSCI054 PO | Fall | 2020 | Greenberg, Michael | CSCI054 PO-1 | 5 | POMONA |
| 2 | 3 | CSCI062 PO | Fall | 2020 | Kauchak, David | CSCI062 PO-1 | 5 | POMONA |
| 3 | 4 | CSCI101 PO | Fall | 2020 | Chen, Tzu-Yi | CSCI101 PO-1 | 5 | POMONA |
| 4 | 5 | CSCI105 PO | Fall | 2020 | Birrell, Eleanor | CSCI105 PO-1 | 5 | POMONA |
| 5 | 6 | CSCI133 PO | Fall | 2020 | Wu, Yuqing | CSCI133 PO-1 | 5 | POMONA |
| 6 | 7 | CSCI140 PO | Fall | 2020 | Clark, Anthony | CSCI140 PO-1 | 5 | POMONA |
| 7 | 8 | CSCI151 PO | Fall | 2020 | Osborn, Joseph | CSCI151 PO-1 | 5 | POMONA |
| 8 | 9 | CSCI159 PO | Fall | 2020 | Kauchak, David | CSCI159 PO-1 | 5 | POMONA |
| 9 | 10 | CSCI181 PO | Fall | 2020 | Birrell, Eleanor | CSCI181 PO-1 | 5 | POMONA |
| 10 | 11 | CSCI188 PO | Fall | 2020 | Osborn, Joseph | CSCI188 PO-1 | 5 | POMONA |
| 11 | 12 | CSCI190 PO | Fall | 2020 | Greenberg, Michael | CSCI190 PO-1 | 5 | POMONA |
| 12 | 13 | PHY023-HM | Fall | 2020 | Saeta Peter | PHY023-HM-1 | 5 | Harvey Mudd |
| 13 | 14 | PSYC010 PZ | Spring | 2021 | Brown, Darin | PSYC010 PZ-1 | 2 | Pitzer |
| 14 | 15 | PSYC010 PZ | Spring | 2021 | Banerjee, Mita | PSYC010 PZ-2 | 2 | Pitzer |
| 15 | 16 | PSYC065 CM | Spring | 2021 | Gumaer, Caitlyn | PSYC065 CM-1 | 2 | Claremont McKenna |
| 16 | 17 | MATH031 SC | Spring | 2021 | Swift, Randall | MATH031 SC-1 | 5 | Scripps |
| 17 | 18 | MATH030 CM | Spring | 2021 | Puig de Dios, Yunied | MATH030 CM-1 | 5 | Claremont McKenna |
| 18 | 19 | MATH030 CM | Spring | 2021 | Puig de Dios, Yunied | MATH030 CM-2 | 5 | Claremont McKenna |
| 19 | 20 | MATH030 CM | Spring | 2021 | Valenza, Robert | MATH030 CM-3 | 5 | Claremont McKenna |
| 20 | 21 | ECON051 SC | Spring | 2021 | Flynn, Sean | ECON051 SC-1 | 2 | Scripps |
| 21 | 22 | ECON052 SC | Spring | 2021 | Kacher, Nicholas | ECON052 SC-1 | 2 | Scripps |
| 22 | 23 | ASTR062 HM | Spring | 2021 | Esin, Astir | ASTR062 HM-1 | 4 | Harvey Mudd |
| 23 | 24 | PHYS089B PO | Spring | 2021 | Moore, Thomas A. | PHYS089B PO-1 | 4 | POMONA |
| 24 | 25 | PHYS089B PO | Spring | 2021 | Moore, Thomas A. | PHYS089B PO-2 | 4 | POMONA |
| 25 | 26 | ANTH001 PZ | Spring | 2021 | Miller, Sheryl | ANTH001 PZ-1 | 2 | Pitzer |
| 26 | 27 | ANTH002 SC | Spring | 2021 | Miller, Alyssa | ANTH002 SC-1 | 2 | Scripps |

```
>>
```

Figure 1: Terminal UI example

Detailed documentation of how to use the terminal is on the appendix.

**Challenges**
We predict that we may encounter several challenges. This is a tumultuous time in the world therefore it's of the utmost importance to prioritize our health both physically and mentally. If something comes up for someone outside of school, it's important we communicate and support one another. We are also all busy with our workloads so it will be necessary to be on top of deadline tasks and plan ahead. Another challenge is our inexperience with SQL servers, we will have to prepare and plan accordingly for the adjustment and learning period of becoming acquainted with these tools. It turned out that most of the challenges were on the UI side where we had hard time to integrate it with the remote database. The T-SQL programing was time consuming but was well understood.

**Business logic refinements:**
Overall, there were no changes of the business logic that is described above, but we finally landed on the right vocabulary for this project which includes: Course, Section, Lesson are the main entities where the concept of a Class was dropped. Also we decided that all the business logic will be implemented on the SQL Server side, and the UI will call either a table function or a Store Procedure with no access to the raw database tables.

**Implementation details:**

Front end:

The terminal front end is written in Python, using the pyodbc package. It is a simple wrapper that takes the user input and calls the SQL Server table functions or store procedures and presents the result. Most of the work here was to validate that the user input is valid and to display error messages if needed. For each show command, an additional optional filter parameter was introduced to enable flexible data selection. It was translated to a where clause that was sent directly to the server as part of the select query from the table functions. Formatting the data to be nicely displayed to the user was done using the tabulate package.

Server-side code:

The server code was written in T-SQL (the programing language for SQL Server) and all the raw SQL tables were abstracted a way for the user (using the terminal application). For getting data (show commands) sql table functions were called. Those functions implemented 2 important functionalities:

1) The requested data was augmented with additional attributes that were useful for the user. For example fn_section() returns also the course information for each section by joining the section table with the course table.

2) Enforcing access control to the data ensuring each student can see only their own private data. The database login name was assumed to be the same as the name of the student in the student

table so that the student id is retrieved on each call and the access validation is performed. The database function fn_canaccess (@sid int) returns 1 if the current login has access to the information for the given sid and 0 otherwise. It was used in many places in the code to filter out data that the current user (student) should not be capable of viewing. However, if it were called by the registrar (database owner), it would always return 1 and allow access to all data.

For executing commands (exe command from the terminal) SQL store procedures were written. Also, here access control was checked and also all the business logic for the project. To do the actual swap, SQL Server T-SQL cursor was used to loop over all matching records. The actual swap to sections was done in a transaction to ensure atomicity.

**Lesson learned:**

1) html UI is not trivial, at least not with PHP, and we need to get familiar with it or with other technology to present a modern UI to the users. Terminal solution will not be acceptable solution in all cases.

2) Commercial database like SQL Server has a lot of nonstandard functionality like managing logins and users, access control and server-side cursors. Mastering standard SQL is juts the starting point for a successful project.

3) Python and SQL Sever work very well together and have a lot of sample and help online. On the other hand, the integration of PHP and SQL Server is more challenging (at least to us)

4) Putting a lot of check constraints on the data is very useful and prevent a lot of user mistakes.

5) It is very important for the team to speak the same vocabulary from the beginning. At the start of the project, each team member called the same thing in a different name (section, class, course) and that created confusion and reduced productivity.

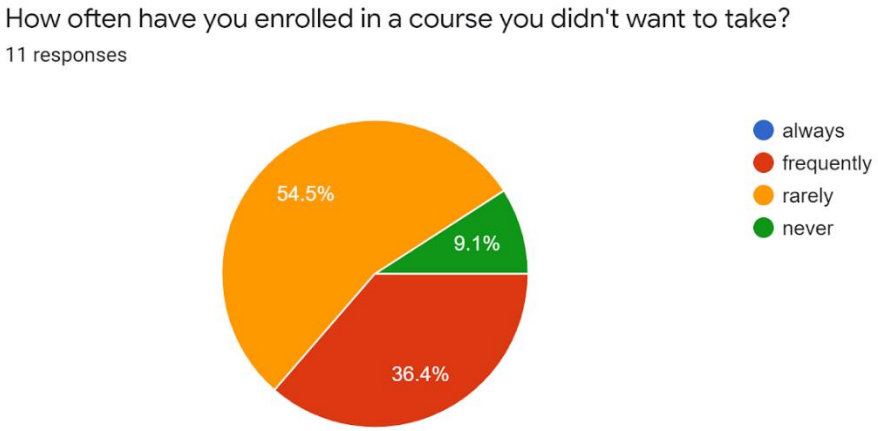
**Appendix:**
**Terminal commands summary**

| command | Description |
|---|---|
| show lessons [filter] | shows the list of lessons |
| show sections [filter] | shows the list of sections |
| show courses [filter] | shows the list of courses |
| show enrollments [filter] | shows list of enrolments for the student or all enrollments if run by the registrar |
| show requests [filter] | show requests submitted by the student or all requests if run by the registrar |

| | |
|---|---|
| show credits [filter] | show courses completed by the user |
| show prereqs [filter] | show the prerequisite for each course |
| show matches [filter] | show the matches for the requests |
| exe request | execute exchange request |
| exe cancel | cancel the requests for the student |
| exe swap | execute the swap of requests (registrar only) |
| exit | exit form the terminal |
| help | show help on available commands |
| help show | show help on the show command |
| help exe | show help on the exe command |

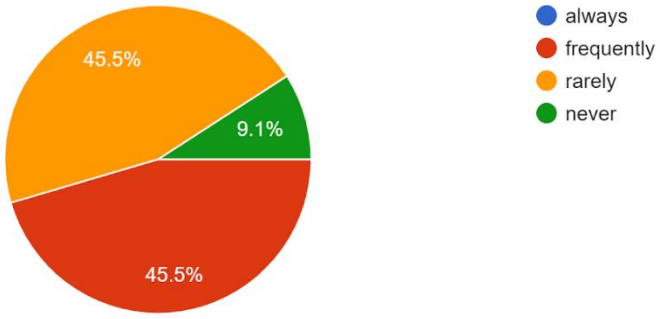
**Survey Results and Summary**

We conducted a survey in order to understand what functionalities and features our intended audience for the course swapper would prefer. We asked a total of six questions (questions and results below) and got 11 responses from students amongst the Claremont Colleges. From the results, we are able to learn a few things to help direct our project. First off we know that our project will help people with the common issue of wanting to switch out of a course they're enlisted in. The majority of our participants would want the feature to be anonymous but are largely undecided on whether they would want direct contact/ conversation with the person they would be swapping their course with. Along with this, it would be helpful for our tool to be built directly into the registrar page. From these results we are able to start going about constricting the course swapper with a better understanding for the roadmap of our project.



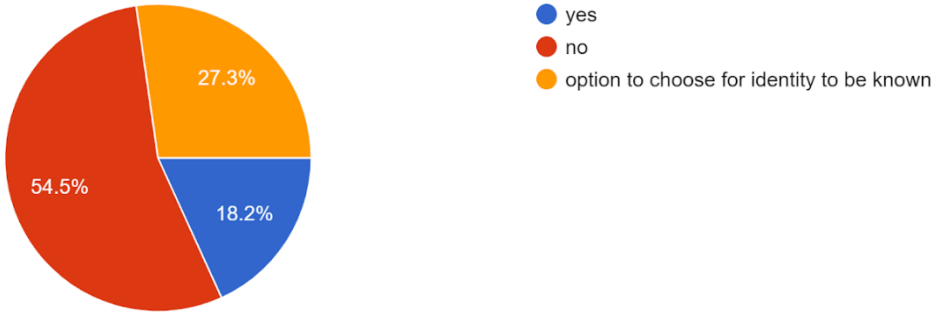
How often have you enrolled in a course and wanted to switch out of it?

11 responses



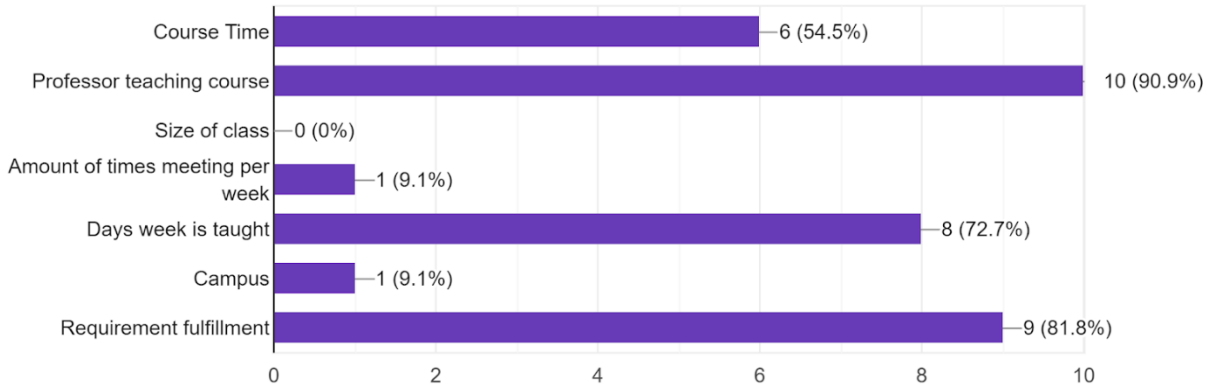

If you were to swap courses with someone would you want to know their identity/ they know your identity or to stay anonymous?

11 responses



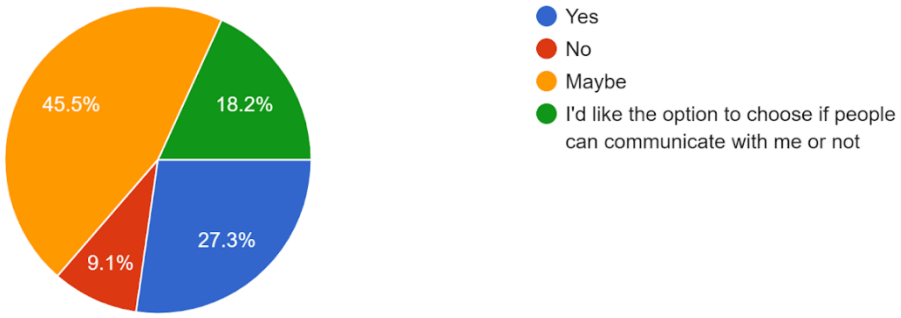
When looking for courses what features do you look for? (Choose top 3)

11 responses



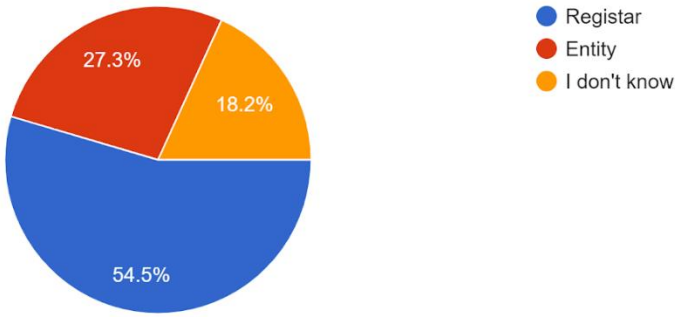

Would you like to be able to talk to/ message the person you are thinking about swapping a course with?

11 responses




Would you want this feature built in to the registar site or as its own entity?

11 responses



**Links:**

Github: https://github.com/alonlapid/courseswap

Demo video:
https://onedrive.live.com/?authkey=%21ADgjJ1Jnsg1i6do&cid=62BC2D38A4D1D467&id=62BC2D38A4D1D467%2116574&parId=root&o=OneUp