|  |  |
| --- | --- |
| Cameron Blanchard | 100886476 |
| Matthew Maynes |  |
| Kevin Cox |  |

1.

2. SCE (Software Engineering)

3. Mr. Abaza

4. Cameron Blanchard – web client views, course selection, scheduling, database design

Matthew Maynes – Java client, database design

Kevin Cox – Back-end database functions, prerequisites

5.

database/ - Contains the course data, prerequisites, and program patterns to be loaded into the database

server/ - Contains index.html, install.php, and some style sheets required for views

server/api - Contains API endpoints

server/api/lib - Contains db.php for configuration of database username and password

server/js - Contains Javascript required for Student Views 1 and 2

6.

7.

Year Standing

The code does not implement year standing prerequisites.

Concurrent Courses

The structure of the prerequisites in the database is as follows: Each course may have one or more “course groups” as prerequisites. Each course group has a flag called “concurrent”, which indicates whether or not the courses within that group may be taken concurrently as a prerequisite.

Program Transfers

When a student navigates to Student View 1 (index.html), they should select “off pattern” if they have transferred from another program. They are presented with a list of courses from the program they have selected; they also have the option to manually add course codes to the list of courses that they have completed. A transferring student should manually add any courses that he has taken from another program, or as an elective. As long as these prerequisites exist correctly in the database, the course suggestion algorithm will take them into consideration. For the SYSC 2006 example, if the student entered ECOR 1606 manually into the list of completed courses, and ECOR 1606 correctly exists as a prerequisite in the database, the prerequisites query will allow them to take SYSC 2006.

8.

Scheduling Algorithm

The general behaviour of the scheduling algorithm is as follows:

The algorithm will attempt to place the student in the 5 courses which appear earliest in their pattern which they have not already completed. If one of these courses is an elective, it will show a dropdown for the particular elective options (CSE, note a, note b, etc), and schedule 1 less course to leave room for the elective. If these courses cannot all be scheduled without conflicts in the same semester, the algorithm will step through a list of alternative courses for which prerequisites have been satisfied, and attempt to create a conflic-free timetable. Many conflict free timetables may be generated for the same course input; the maximum number of timetables to generate is configurable.

When registering in lab or tutorial sections, the algorithm will place the student in the first lab section found which is not full and does not create a conflict in the timetable under consideration. If there are multiple lab sections which satisfy this requirement, they will show up on a different version of the schedule.

9.

The server process both identifies which courses may be taken and performs the generation of conflict-free timetables.

The major factor in the decision to place these functions on the server side was the ease of creating multiple clients once the server process had been implemented. The scheduler outputs timetables in an XML format which can be displayed using XSL and Javascript for the web client, and parsed fairly easily in Java. If these functions were moved to the client side, complex scheduling algorithms would need to be written in both Java and Javascript, and the behaviour of these algorithms might not match exactly. It was also more straightforward to access the database from the server side: the amount of data needed to implement the scheduling algorithm is significant. Rather than serializing data to an intermediate format to be parse in both clients, the server can deal directly with the database.

There are disadvantages to implementing the scheduling and course identification algorithms on the server side. The most significant drawback is the amount of processing time required to generate a conflict-free timetable. The algorithm is recursive and could take a considerable amount of resources from the server. Thus, this application may not scale well with too many concurrent users. To combat this issue, the server process can be configured to generate a maximum number of schedules. Using this parameter, we have found that timetables are generated within a very reasonable amount of time.