***Group Divider***

**What is this tool?**

*Preference points*

This tool gives the possibility to form groups of students based on their preference for each other. Every student has a number of points to give to the other students, where he can choose a number between 1 and 9 for every individual fellow student. The higher the number, the more he/she wants to be in a group with him/her.

*Example*

There could be for example a total of 9 students who are wished to be distributed over 3 groups of 3 students. This program maximizes on above mentioned preference points, which are only valid if students actually end up in the same group. The output is in the form of ‘the first group consists of this, this and this student, the second group, and so on’.

*Optimization*

It would be possible to calculate every possible solution and see if it is the best one, but this gets exponentially complex with an increasing number of students. This is where ‘optimization’ comes in. Optimization makes it possible to put a problem like this in a mathematical model and then calculate the best solution, without having to check every possible solution. This saves computation time for bigger problems.

*Additional constraints*

In this tool, apart from preferences, there are other things that can be taken into account: actiontypes and programming experience. The teacher can take tests on students to find out if they are introvert, extravert, intuitive, sensing, thinking, feeling, judging or perceiving and then put this information in the program to for example have at least one extravert person in every group. The same can be done for programming experience: every student gets a grade from 1 to 9 to describe their programming experience. This information can be added to the program to enforce that for example every group has at least one student with a programming experience level of 7 or every group has not more than one student with a level of 3 or less. And finally there can be constraints added like; in whatever case, put these two students in one group; or the opposite: in whatever case don’t put them together.

**How I came up with this project**

It was around Christmas 2021 that a teacher asked everyone in our class; who do you want to be in a group with for the next education period? And she said: “send me an email with your most and second most favorite person you want to be in a group with”. So I sent an email mentioning 5 people and saying “you can ignore number 3, 4 and 5 in my top 5, I can imagine that it is a difficult puzzle to form groups with all those preferences from everyone”. Then she said “yes indeed, it would be nice if there would be a tool for that” and so then I started thinking; I could make this tool, since I was doing a math course ‘optimization’ and this problem was just exactly like one and so in combination with my programming skills, I started building this tool.

**Process**

Doing this project started with the following steps:

* putting the preference problem into a mathematical model:
  + hardest part is to choose the right decision variables
  + I chose a group of 4 students which had to be divided into 2 groups of 2. This is the smallest non-trivial problem, which reduces the length of formulas to write down, while the principle for smaller or bigger groups remains the same.
* once I managed to do the above, it was relatively easy to add the additional constraints to the mathematical model: actiontype, programming experience, having specific students (not) in one group in whatever case
* do research which optimization libraries there are in python and finding one that can handle this mathematical model. In the end PuLP was the one to go for.
* writing it all down in code

**Aftermath**

*Calculating computation time*

I tried to write a formula which could predict computation time for different input values, but I partly managed to do this. It got messy quite quickly so if I would start this again, I would approach this in a more structured way.

*Putting it into a web application*

Once it all worked, giving input was still a matter of changing values inside a bunch of code and output was printed as plain text. To change this, I set up a FLASK app with a SQLAlchemy database, which so far I couldn’t complete yet to have it fully functional, but to get an impression of how far I got, see the image on the next page.

**Explanation of code**

There are comments added to the code, so this makes things more clear. Two things that should be pointed out:

* The preference matrix is randomly generated. Normally this input would be given by the teacher, but in this program, I wrote a few lines of code to create one.
* On line 188, the variable ‘desired\_number\_of\_additional\_constraints’ can be found. This variable can have values 0 till 5. Based on this number different amounts of additional constraints are added. All constraints are independent from each other, so the code could also be adapted, to only have the second constraint for example. To get an idea how the additional constraints do their work:
  1. give the variable a value 0.
  2. run the program
  3. increase the variable with 1
  4. run the program
  5. repeat step c and d until the variable has value 5.

See also the file ‘explanation\_restrictions.txt’.

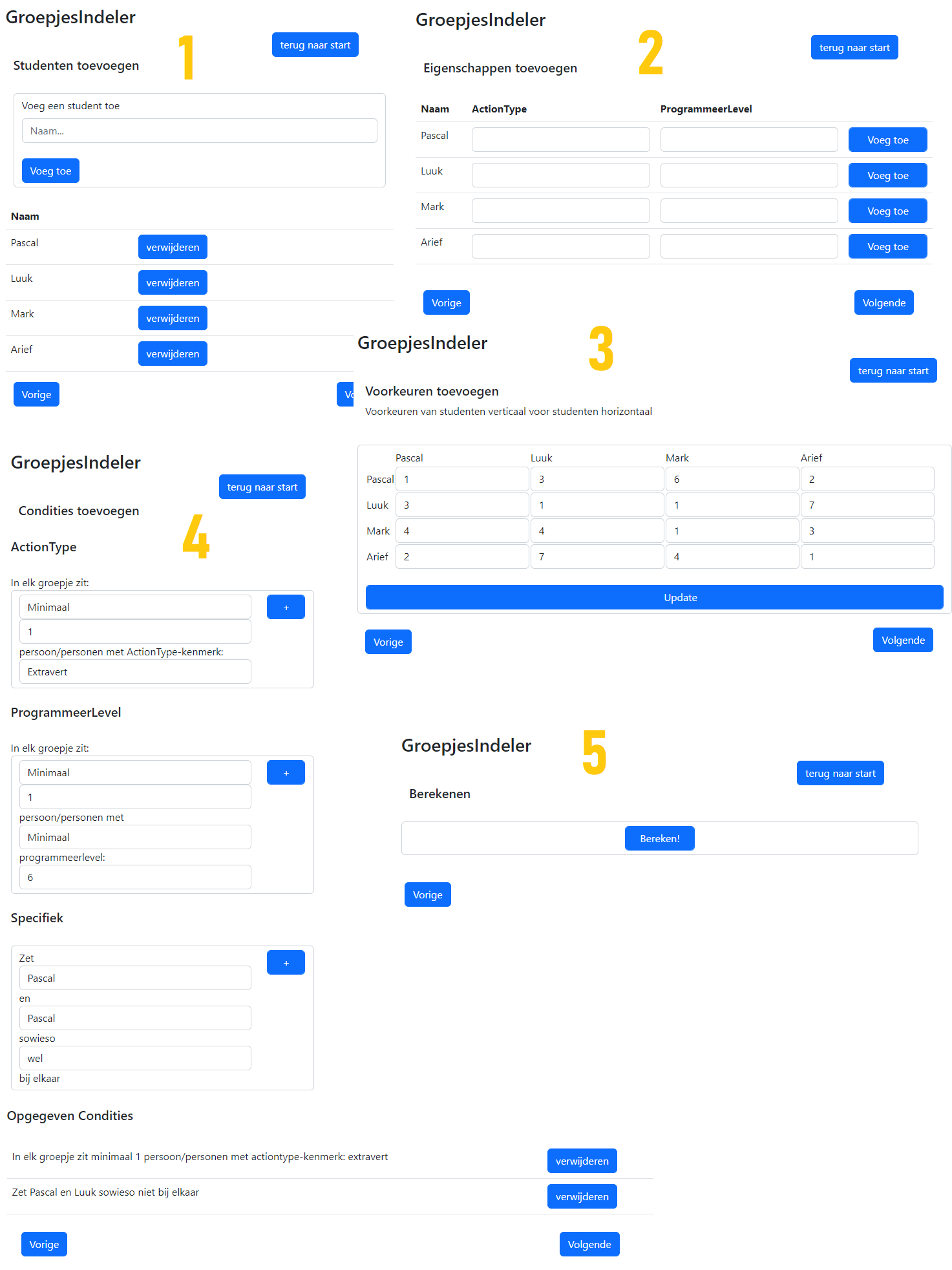


Image 1. Web application using Flask and SQLAlchemy, see paragraph ‘Aftermath’