Seating Chart Generator for Evening with Industry

June 10th, 2020

Team CSM SWE: Joel Bettis, Connor Pashak, Carly Arndt

Advisor: Dr. Wendy Fisher

Client: Kelly Knechtel, Annette Pilkington

Table of Contents

Introduction

The client for the project is one of Mines's own organizations, the Society of Women Engineers (SWE). They are the largest student organization on campus and the largest collegiate section of SWE in the country. This section of SWE is focused on providing meaningful professional development, networking, K-14 outreach, mentoring, leadership, and social enrichment opportunities to over 700 members.

The Society of Women Engineers (SWE) at Mines hosts an annual Evening with Industry event the night before career day. Between 300-400 students, recruiters, faculty, and VIPs attend this networking event. The evening begins with an hour of networking followed by a sit down dinner, with a formal program including a keynote speaker from industry. There are assigned seats during the dinner, and the goal is to give students the opportunity to have longer, more meaningful conversations with companies who are seeking students from specific majors. Our Team would like to help the organizers of this event accomplish this goal making the process easier for them.

The primary goal of this project is to streamline the generation of prioritized seating assignments that have flexible inputs and outputs so that students may have more meaningful networking connections with recruiters and faculty. Taking in data from client-provided excel sheets, column headers must be dynamically read in order to generate a seating chart taking into account each person's major interests. The program will then allow manual editing until the user decides to export the data using our flexible export options. Using our program, the Society of Women Engineers will be able to quickly and efficiently generate seating charts for their Evening with Industry Event, giving students and recruiters important networking opportunities with the women at Mines.

Functional Requirements

- Be able to accept flexible data formats
    - Able to handle .csv, .xlsx, and .xls files
    - Robust error handling
    - Accept an arbitrary number of files
- Allow user to specify table preferences
    - Number of tables
        - Can be manually or automatically set
    - Number of seats per table
- Allow user to pre-assign people to a table
    - Search list of people from imported files
    - Select table number and assign
- Automatically generate seating charts
    - Evenly distribute faculty and recruiters
    - Give priority to higher-level students
    - Clump degree interests
- Ability to change seating after initial generation
    - Insert
    - Delete
    - Swap
        - With other people
        - With empty seats
- Ability to dynamically export data
    - Dietary restrictions, name and major, etc
    - Exporting name tags & place tags

Non-functional Requirements

- Intuitive for the user to use
- Easy to download and run
- Create a simple, modular GUI with Java
- Simple user's guide and documentation
- Scalable
- Cross-platform compatibility

System Architecture

Import page

The import page is where the user can import files containing the people, and set the table information. This page is split into four panels, as shown in our UML diagram in *Figure 1*. For the GUI display of our Import Page, see the appendix.

Each of these panels gives the user control to create a custom seating chart, including choosing the number of tables and people per table, and choosing specific guests to place at each table. These options were implemented so that there would be minimal manual assigning after generation of the seating chart. When the user is done inputting information, they can press a button to generate the seating chart and move on to the seating chart page.

Import File

This panel is used to select the file to import. The user selects what type of information the file holds, which can be students, recruiters, faculty, or an existing seating chart. Then, the user selects the file from a popup file menu, and presses the “Import File” button.

Display File

The Display File panel is used to show the files that have been imported. It displays the name of the file, as well as the type of information contained in the file.

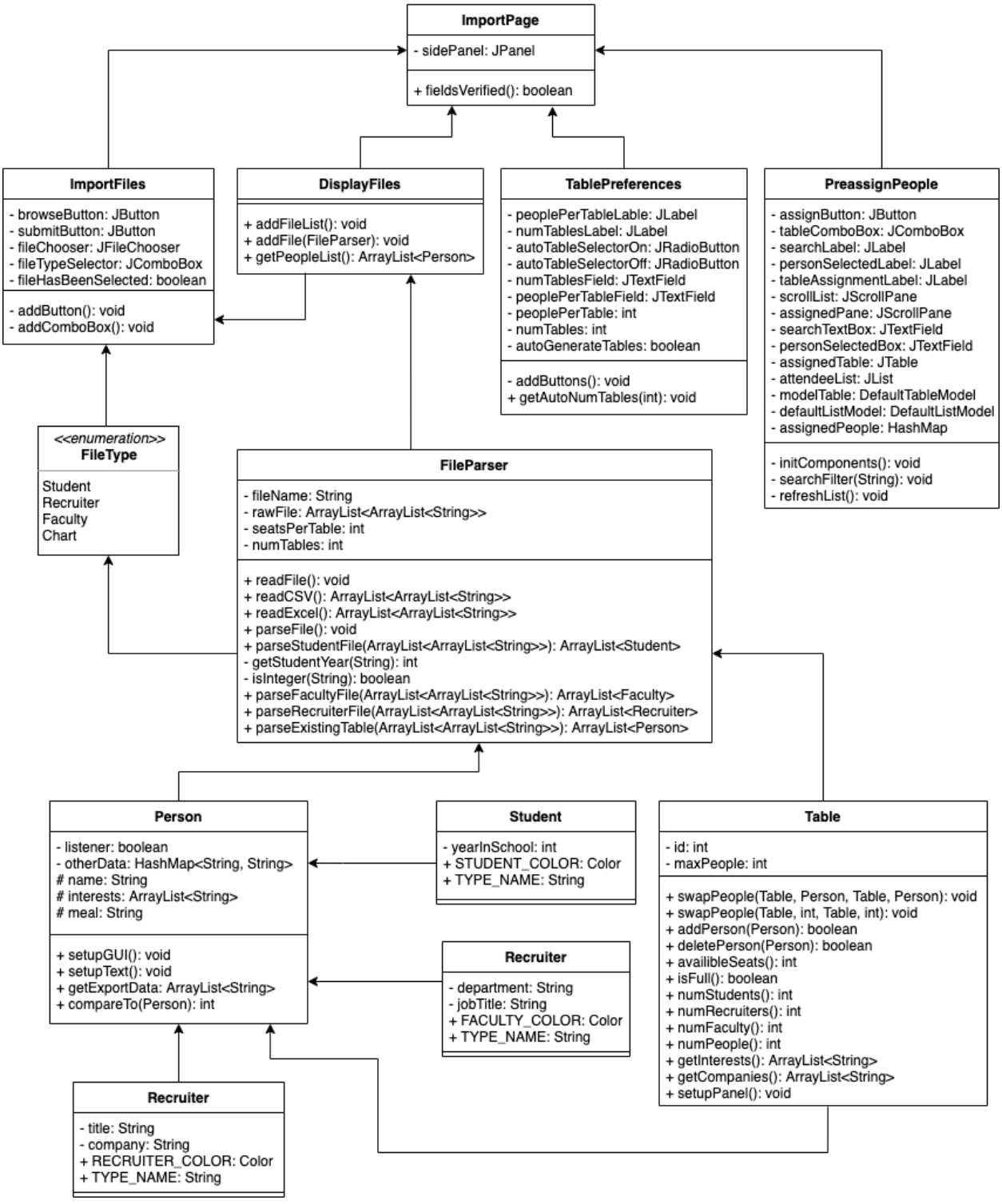
Table Preferences

In this panel, the user can set the table preferences. They can set how many seats are at each table, and how many tables there are. The user can either manually set the number of tables, or select the auto option. If the auto option is selected, the program will use the minimum number of tables needed.

Preassign People

In the preassign panel, the user can assign specific people to specific tables before the seating chart is generated. The user first selects a person that has been imported in one of the files from a searchable list, then selects the table that they should be assigned to.

Figure 1: Import Page UML



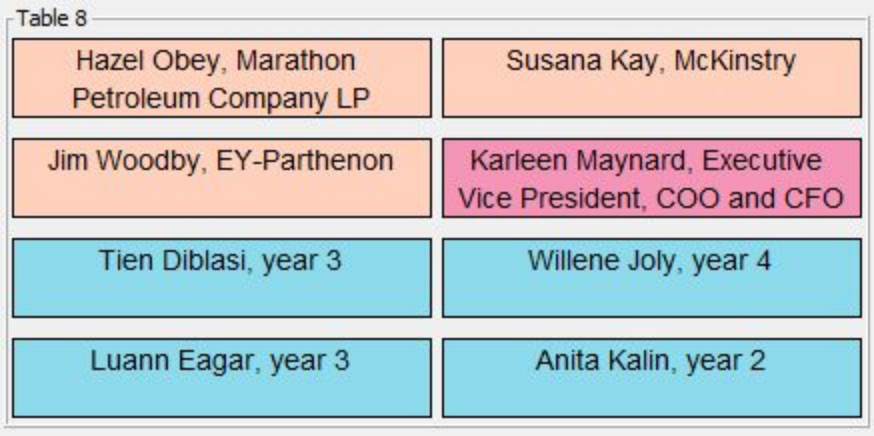
Seating Chart Page

Our team has created an algorithm that properly weights the major interests for students, recruiters, and faculty, which will be described in depth in the technical design section. Next, the program must allow for manual editing by the user after seating chart generation. To accomplish this, we created a simple GUI so the user has clear visuals of the seating chart and can perform the edits. The GUI shows up to nine tables on the page at a time, and color codes students, recruiters, and faculty so the distribution can be seen easily. A sample table is shown in *Figure 2*. In each seat the name of the person is displayed, followed by person-type specific data.

- Recruiter: name, company
- Student: name, year in school
- Faculty: name, job title

Additionally, by hovering the mouse over any person's text box, their major preferences are shown in a hovering text window.

Figure 2: Table display



This page also has the edit panel, shown in *Figure 3*, where people can be swapped, inserted, or deleted into the seating chart. To swap or delete people, the user selects the appropriate radio button, clicks on a person to select them, and then presses the swap/delete button.

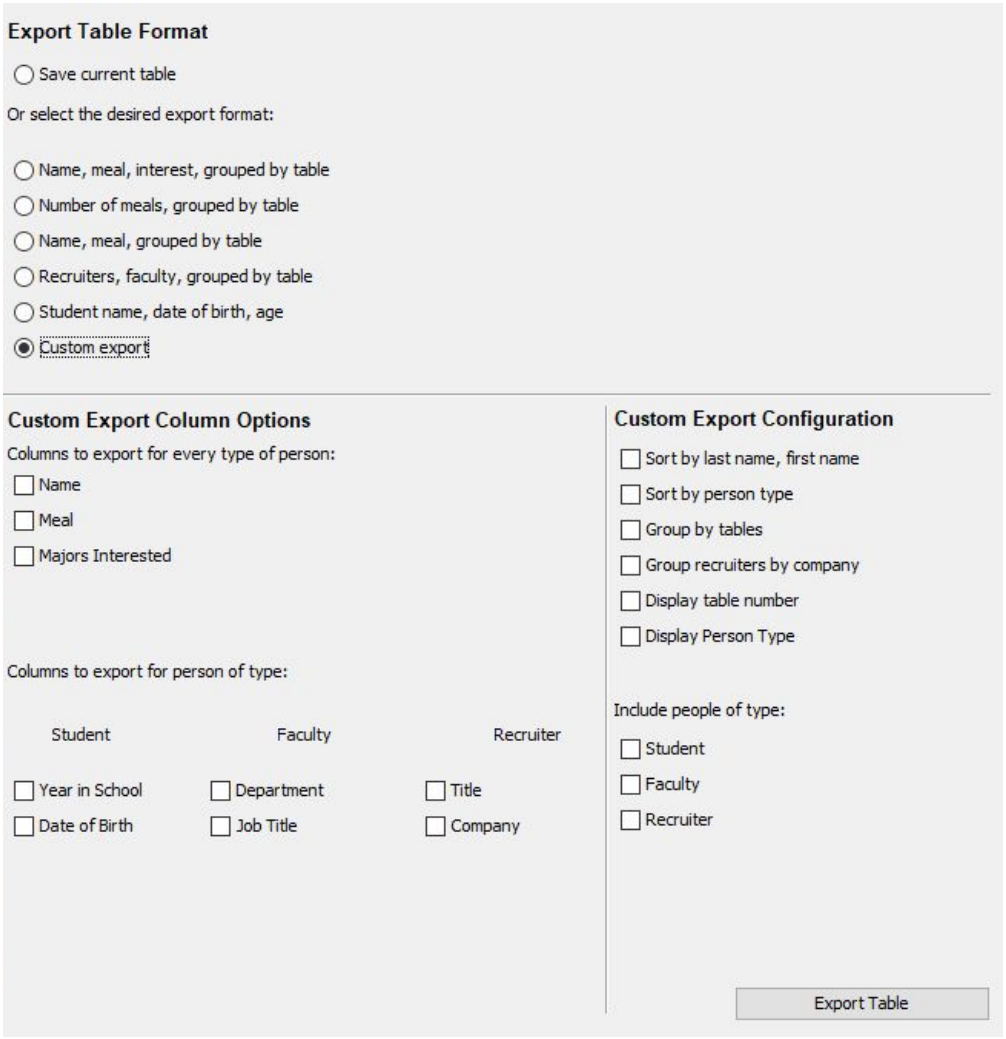
Figure 3: Edit panel

Export Dialog

Being able to export the seating chart in multiple formats was one of the main functional requirements. Export options are split into two types: preset exports and custom exports. Configuration for the preset export options was created based on predictions of the most common desired export formats. For the custom export configuration, the user is able to change the export formats to include all other columns.

Accessible from a “File” menu bar on the seating chart page, the export dialog appears and gives the user a multitude of preset exports as well as a custom export option. When the user presses “Export Table” a save file menu appears on the system where the user types in the name of the file, chooses file type, and chooses the directory to save to. The export formats can be seen in *Figure 4*.

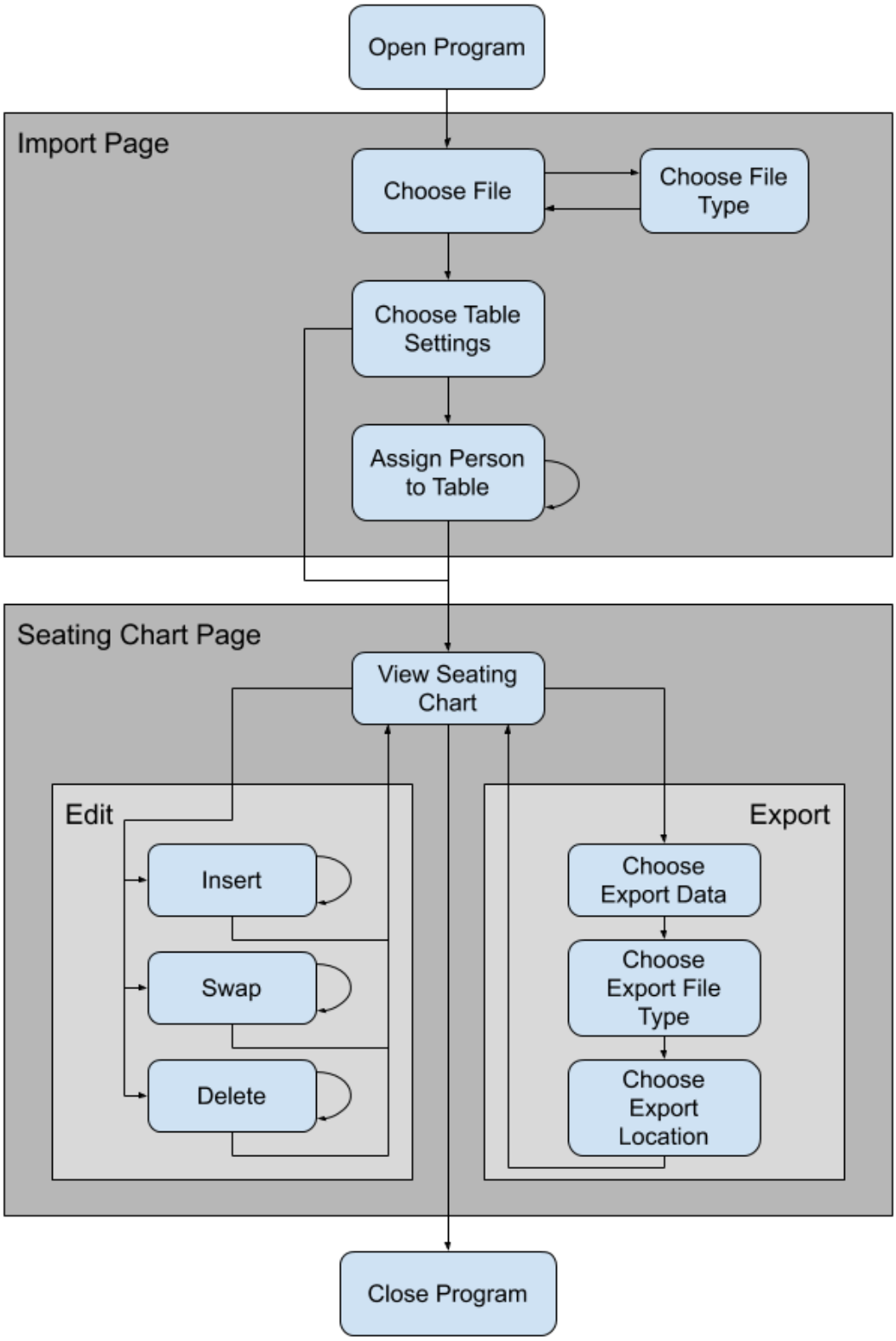
Figure 4: Export dialog



Program Run

Tying all of these components together, the process to run the program is clear and user-friendly. A graphic use case is shown in *Figure 5* to walk the user through the entire process as a whole.

Figure 5: Use case



Technical Design

Flexible file imports

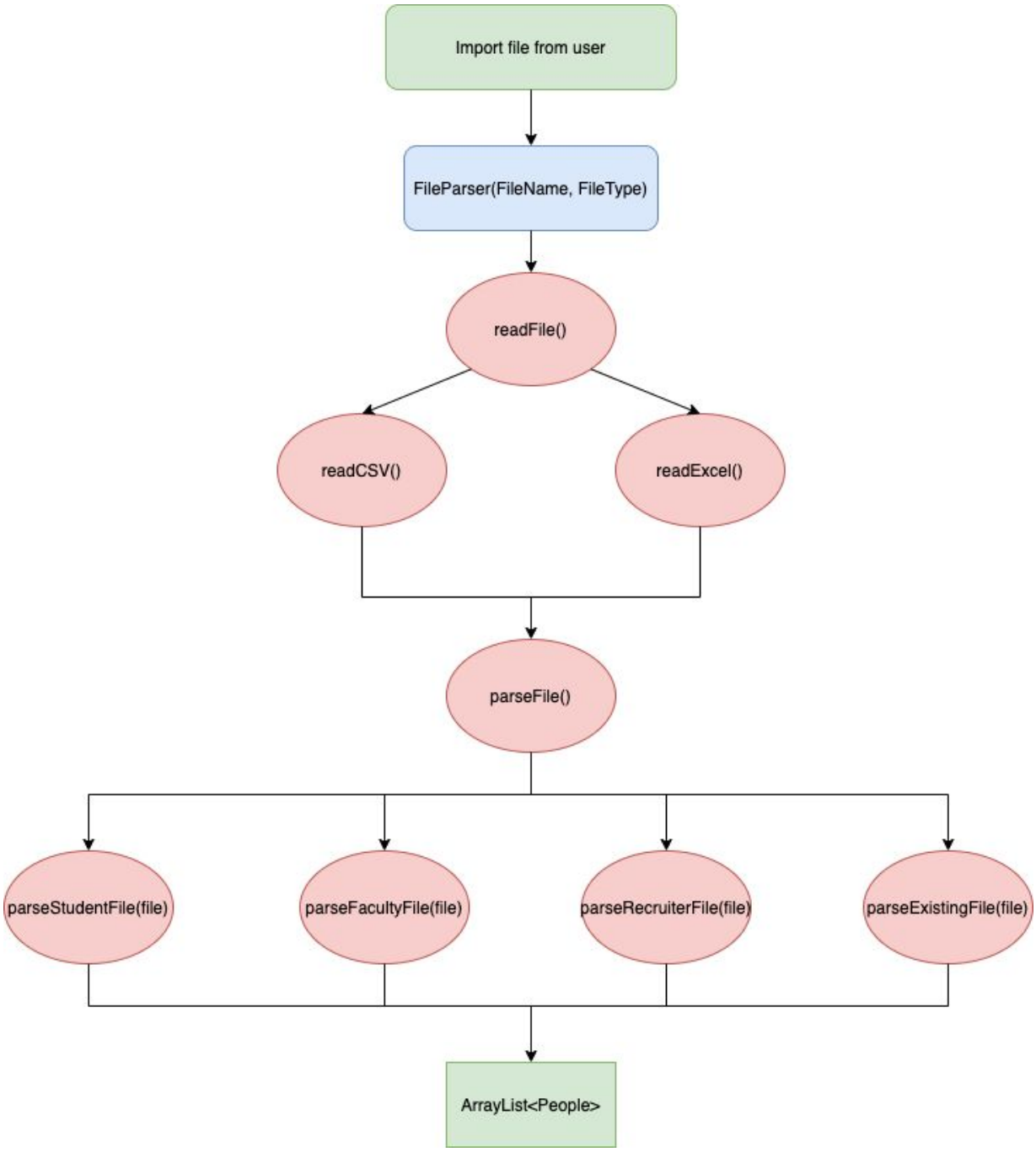
An important aspect of this project for it to be successful is the operation of file importing. The previous seating chart program only allowed .csv files after significant data cleanup from the raw files that the client has with each person's information. As per the client's requests, our program must support Excel files and take the raw survey data from the students, recruiters, and faculty into the program. To accomplish this, the team's program both supports more file formats (.xls and .xlsx) and allows for dynamic file parsing in case the file format changes or new data is collected.

A FileParser class was created that takes in a file as well as file type, and then reads and parses it based on those two fields. Two functions, readCsv, and readExcel put the file data into a two dimensional list of strings which are then parsed by separate functions. Reading csv files is straightforward with an InputStreamReader and BufferedReader. In order to be able to read and write excel files, we used the Apache POI API library, since this functionality is not built into Java. Then, a new "workbook" is created through Apache that is used to do functions on the excel file. Then, a for each loop is used to iterate through each row in the sheet, getting the value of each cell as a String and adding it to the two-dimensional list.

After the files have been read and transformed into a workable format as a two dimensional list of strings, they are passed to one of four parsing files that are called based on the file type. The valid file types that were created are Student, Recruiter, Faculty, and Chart. The "Chart" type is an existing seating chart that allows charts to be saved and worked on later. Each parsing file looks dynamically for the column names it is expecting. For example, for each recruiter row it looks for company name, major interests, names for all attendees, meal preferences for all attendees, and job titles for all attendees. These columns indexes are then stored, and used later to read the information from the columns. Additionally, all other columns which the parser is not explicitly looking for are put into a HashMap stored in each person. When the file parsers successfully finish reading the rows, they return a list of type Person, a superclass to Student, Recruiter, Faculty shown in *Figure 6*.

When all necessary files have been imported and the generate button is pressed, the list of FileParsers is then handed to the SeatingChart class to run its algorithm.

Figure 6: FileParser flow



Seating Chart Algorithm

The main part of our project is our seating chart algorithm. As described in the functional requirements, the program must evenly disperse recruiters and faculty, and then assign students where their major preferences match. But, before this is done, the program seats the people who were pre-assigned to tables on the import page. Once this happens, it moves on to assigning recruiters.

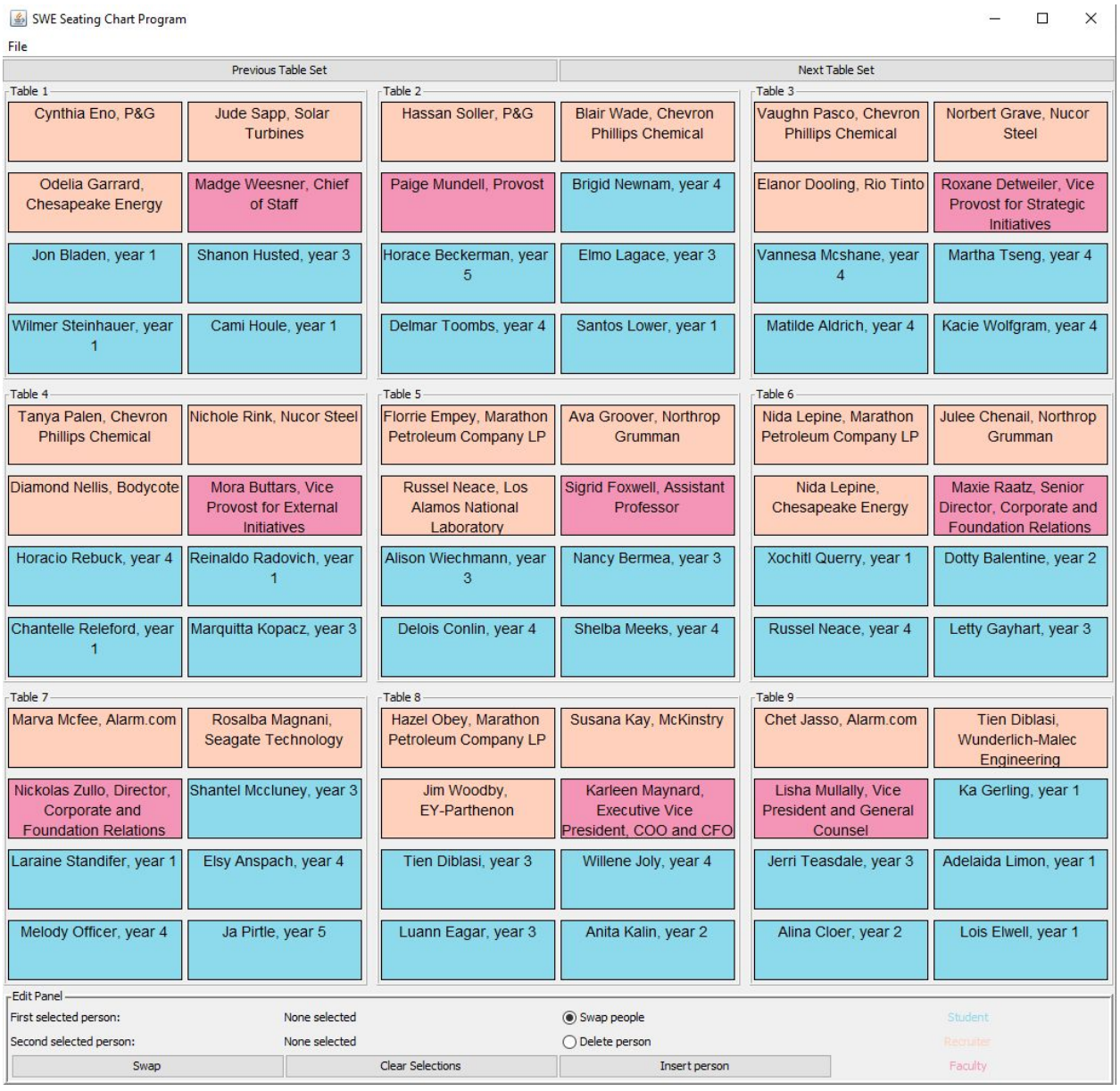
To assign the recruiters, they are first grouped by their major interests. Then, the program goes through each of these interests. If there are enough recruiters with the same interest, the algorithm tries to make tables that only contain recruiters with that interest, leaving room at the table for students and faculty. When these tables are created, the program keeps track of the recruiters' other major interests, and tries to match them, so that these tables have as few different interests as possible. It also makes sure that there are not any multiple recruiters from the same company at the same table. Then, if there are any remaining recruiters, the algorithm either waits to assign them if they have another interest that has not already been assigned, or assigns them to a table that has enough room.

Then, to assign faculty, the program calculates the average number of faculty per table. Going through the faculty members, it first assigns those who have a major interest that matches any of the recruiters' major interests who are already at the table, unless the table has already reached the average number of faculty that was calculated. After all of the faculty with a major interest are assigned, the rest are evenly dispersed until each table has the desired number of faculty per table.

Finally, the program assigns students to seats. The students are first split into four groups: seniors and graduate students, juniors, sophomores, and freshmen. Graduate students and seniors are given the highest priority, followed by juniors, etc. Next, the program goes through these groups, going through the highest priority group first, and the lowest priority group last. For each one of these groups then, the program iterates through the students several times, and tries to assign students to their first major interest. If there are students who were not able to be seated with their first interest, the program goes through the group again and tries to assign them to tables that have their secondary interest. This process repeats until either all the students in the group have been assigned to their highest possible interest preference, or they have run out of interest. When this happens, the program moves on to the next group. After it has gone through all the groups, it then assigns all the students who were not able to be assigned to one of their interests by evenly distributing them to available tables.

After all three assignment algorithms are run, the seating chart is complete. The resulting chart has people grouped by their major interests, company attendees dispersed, and students weighted by grade level. The GUI is then generated, complete with the edit panel and file menu for exporting, as shown in *Figure 7*.

Figure 7: Seating chart displayed



Software Quality Plan

In order to ensure our product is successful and isn't shipped with bugs and is a pain to use, the team implemented a software quality plan to guarantee a working, useful product. This plan consists of unit testing, user interface testing, integration testing, weekly client input, and documentation.

The first type of testing done was unit testing, which the team utilized JUnit for in Java in order to test the backend functionality. These tests ensured that any of our implemented functionality, such as our parsing functions and our seating chart algorithm worked as expected. Because of the unit tests, the team was able to debug easily and find many "deep" bugs that occur rarely, but are important to fix.

Next, the team completed user interface testing on all of the GUI elements, including the import page, seating chart page, and the export dialog. While no automatic testing of the GUI is necessary because it is a simple Swing program, manual walk throughs of the system were completed for each piece of the GUI.

Another important form of testing done, integration testing, was done to ensure that the program as a whole functions properly after each individual component was tested. For example, after testing the file parsing functions and seating chart algorithm separately, we integrated the two tests later on, feeding the list of people from the file parser into the seating chart algorithm and viewed the results. For the GUI, integration testing comes in the form of being able to transition between the three panels smoothly and ensure no bugs exist when using the program as a whole.

With the above testing done, the program will be robust and bug-free. However, this testing is useless if the client does not approve of our specific design decisions in trying to meet their requirements. Therefore, every week the team conducted user acceptance testing with the client so that the client knows our project direction and can ask for any modifications to the user interface or any of the algorithmic design. Fixing issues that the client has while they are noticed allowed for smoother development as the team did not have to go back through work finished early and modify things, which tends to create unforeseen bugs.

Finally, because this is a team project with many working parts, documentation was important for two reasons. First, any team member looking through and trying to use another member's code can be done much more efficiently. A comment at the top of each function describing its inputs and outputs makes using other team members' code much easier. Second,

because of the possibility of future programmers expanding on our project, documentation in this scenario is invaluable for quick understanding of how the code functions. By implementing all of the above steps, we are left with a professional product that is bug free, designed with the client's preferences in mind, and is easy to be expanded upon in the future.

Project Results

The primary goal of this project is to streamline the generation of prioritized seating assignments that have flexible inputs and outputs so that students may have more meaningful networking connections with recruiters and faculty. Students, faculty, and recruiters are assigned based on common major interests using a seating chart algorithm that our team has created.

All of the primary and secondary goals the client requested were met during the development period. Additionally, the stretch goal for flexible file import/export formats with .xls and .xlsx files was also completed. However, two stretch goals asked for by the client were not met, which was to include a waitlist as well as the ability to use the program for events other than Evening with Industry such as Girls Lead the Way.

While working through the project using Java along with Swing as our development tools, we discovered that Swing is not ideal for making GUIs. Considerable time was spent creating and designing the layouts for each JPanel that was created. However, tools to automatically help with GUI generation were found to save time and frustration, such as the Java Swing tools in the Netbeans IDE. We also learned that reading and writing Excel files was much easier then we expected, due to the Apache POI library.

For future development, we would like to see the next team implement waitlist functionality. The events that SWE hosts have high attendance, so automatic waitlist generation could be a necessary feature for the program. We also hope that the next team will be able to refine the UI, and make the flow of the program better. An additional feature that could be added is the ability to remove files that have been already imported, in case the user accidentally imports the wrong file.

Lessons Learned

For our product, we learned many valuable technical and non-technical skills that are essential for software development. Our team was new to designing a full-stack product for a client, and navigated many trials and errors through the development process.

Our team decided on using Java for our program, as we all had experience with the language from previous classes. This decision was made because we had less than 5 weeks for development, and needed to create a new, fully functional product. If we had more time, it would have made more sense to use a different framework, such as Electron JS. Java Swing, which is Java's library for graphical user interfaces, had a few drawbacks that impacted our project. The first is that Swing does not make beautiful applications, and the complexity increases dramatically as more components are added. If we had used something like Electron JS, the result would have been a cleaner, more user friendly application using HTML, CSS, and JavaScript. This would come at the cost of making a robust backend, since we would have had to focus our attention on the frontend. In the end, our team decided that creating a functional backend was more important than having an elegant user interface.

With our decision to use Java, we also learned the complexity of designing a good user experience. While creating the design, we encountered many issues with making consistent layouts in Java. Swing has lots of layout options, but many of them did not work for our needs so many of the button and panel locations are hard coded. Creating a static size for our application is a risk, as our program could be too big for some screen sizes. The other option, creating a dynamic size for our program, also has some risks as the changing button and text field sizes make it more difficult to understand what is happening in our program. Aside from the visual component, it was also challenging to anticipate how the user would want to use the program. To us as developers, our design made sense as we are the ones who created it, but it may not be clear to a user who has no context on how the system was built.

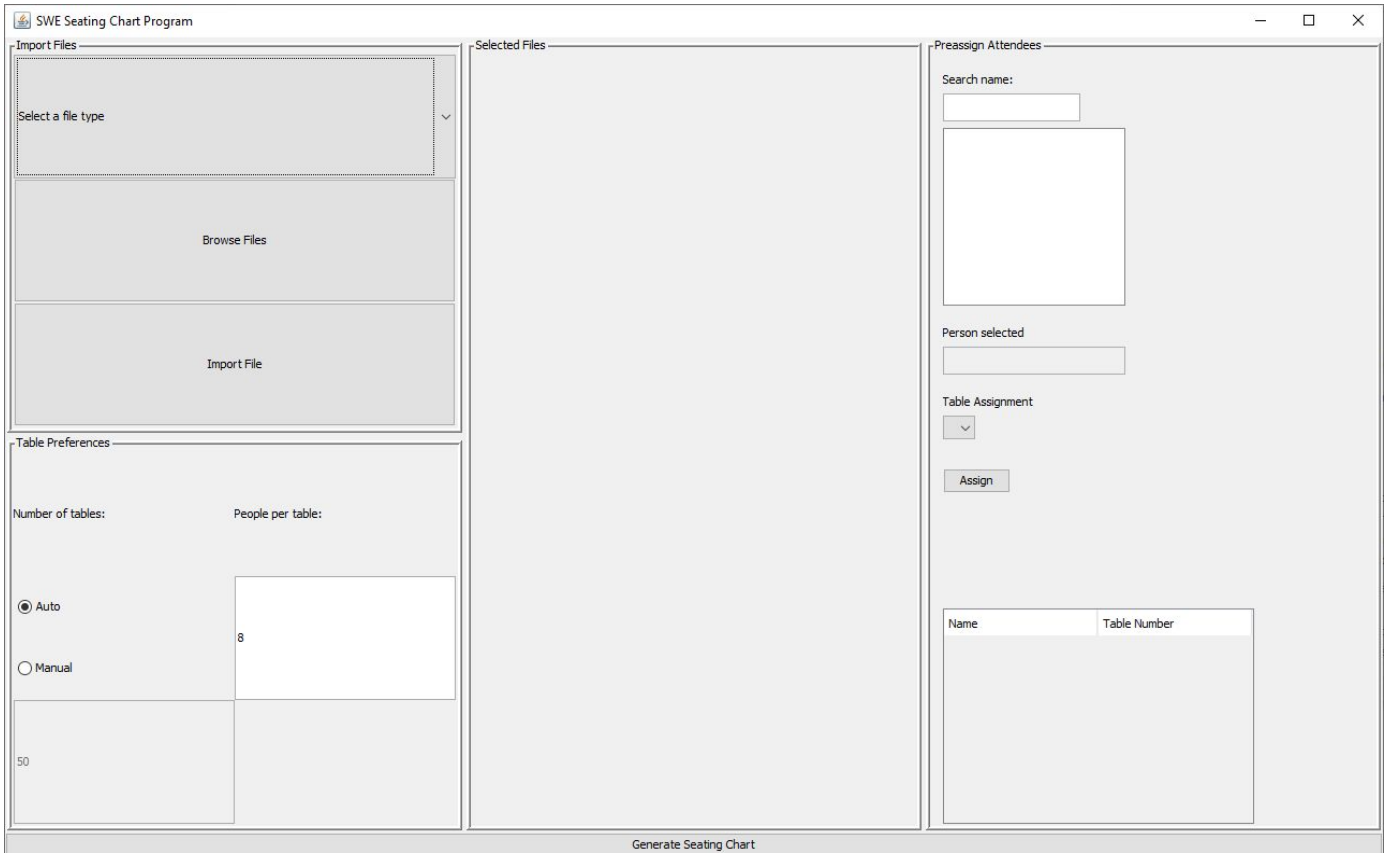
Another challenge our team faced with our seating chart generator was project scope. At the beginning of our project, we assumed that our project scope was relatively small compared to other teams projects. We had high expectations that we would meet all of our goals and most of our stretch goals. In reality, our project was more complex than we thought. Because we were starting from scratch, we had to learn to create a functioning backend with a usable frontend. Connecting these two components was more complex than anticipated; each button, checkbox, and text field needed to be linked to the right functions. We were able to accomplish the main project goals, but had to leave some of our stretch goals out because of our project scope and time constraints.

Possibly the most important lesson we learned was how important communication is throughout the development process. Being able to communicate well to other teammates, our client, our advisor, and to other field session students was crucial to creating a working product. On our team, we worked individually to make each component, so relaying information by using comments and video conferencing regarding what each component was doing was the key to making our project successful. In the future, it would be beneficial to practice paired programming and periodically doing code reviews. The components each of us have created are designed differently, though the output is consistent, it could be improved with uniformity.

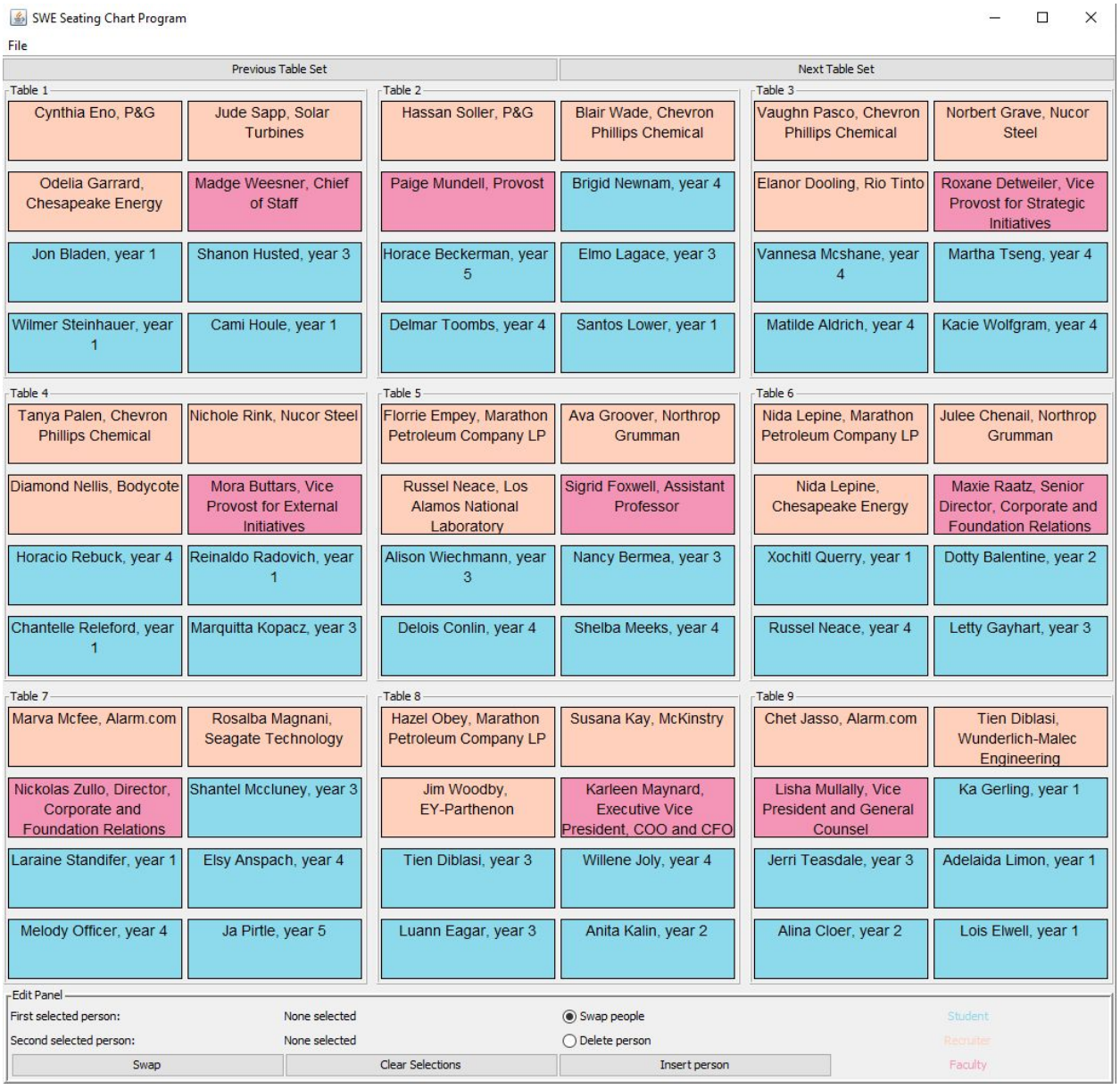
Even with the hurdles of current events preventing us from meeting in person, we successfully completed our project, and are proud of the finished product.

Appendix

I. Import Page



II. Seating Chart Page



III. Export Dialog