

TeamName

Michael Theisen, Jasharn Thiara, Patrick Tibbals, Trevor Tomlin

<https://teamname-tcss360.github.io/>

teamname.tcss360@gmail.com

Table of Contents

Introduction	Page 2
Requirements Addressed	Page 2
How to run the Application	Page 5
Contributions.....	Page 12
Tests	Page 13
Source Catalog	Page 15
Emergency Contacts.....	Page 16

Introduction

The application that we wished to create was one that we wanted to be scalable. The intention was to adhere to the primary scenario written out in Bob Keener's original client interview, but to also make the program as basic as possible so that we could enact user stories independent of each other in a way that made it so none of them interfered. In the process of trying to enact the three primary user stories, we found that alongside Iteration 1 and 2, we were inevitably fulfilling more than just our three primary user stories in the process. This ended up being fine because the process of importing and exporting allowed us to better understand the methods of implementing the process behind the scenes in the Graphical User Interface. We were then able to use this implementation as a springboard to fulfill our primary objectives, user stories 1, 2, and 3, and further include ten additional user stories.

The process took place over the entire quarter and involved three total Deliverables, including this one, and two Iterations in order to ensure we were continuously working toward a goal while maintaining a functional product. It was important that at all points of the projects, we would be able to use GitHub as a fallback to reload our code in the event something became broken beyond repair. Over the course of the quarter, we met every week, once a week, for 2-3 hours each meeting. During these meetings we made sure that a particular task was met each time so that the meetings had purpose. We followed the Agile method guidelines as best as possible and consistently had a Facilitator and Scribe on constant rotation to ensure that we were always on task and always had a thorough record of every meeting's events.

Requirements Addressed

The first and foremost priority for requirements was to ensure proper Javadoc comments in order to be able to explain the code to others and ourselves for future use. Each class has correct Javadoc documentation for author comments as well as method comments and inline comments when necessary. We also ensured that there were no instances of "my" in any of the classes such as "myPanel" or "myFrame" and used camelCase as the primary naming convention.

It was important to ensure that our primary three user stories were fulfilled, but we wanted to do so while still fulfilling some Heuristics from Riel's Object-Oriented Heuristics. The first heuristic we intended to follow was Heuristic 3.1, "Distribute system intelligence horizontally as uniformly as possible, that is, the top-level classes in a design should share the work uniformly." The design allows the program to have separate packages that contain the graphical interface classes separate from the packages that contain the classes responsible for managing the file system.

We also followed Heuristic 3.5, "In applications that consist of an object-oriented model interacting with a user interface, the model should never be dependent on the interface. The interface should be dependent on the model." The interface the user interacts with is responsible for calling the file system in order to generate the user's screen. This will keep the storage system separate from our interface unless the users are using functionality that will set or get information from the file system.

The first required user story was US01: As a user, I want to be able to store documents. Originally, we had the idea to make this process a "drag and drop" feature, but we decided what was most important was the ability to simply add a file to the system and store documents. The function we created can import these new documents and store them in the application.

The second primary user store was US02: As a user, I want to be able to use a keyword search. In order to do this, we decided that the best available method of implementation was to add a search bar with a clickable button. When typing in anything, clicking "Search" will scour the entire application looking for any instance where the typed information is present and then will display it.

The third and final primary user story was US03: As a user, I want to be able to delete old info. This implementation was essentially to create a dropdown tab that populates when right clicking on a file. After right clicking, the word "Delete" is displayed. After clicking on this button, the file is then removed from the application.

It was not intentional that we fulfilled additional user stories by default. We found that in the process of trying to create a functional application, some of the additional user stories would be necessary and desirable to have in place prior to fulfilling some of the primary user stories. Additional user stories that we ended up fulfilling is US04: As a user, I want to be able to edit documents. This function ended up already being built into the document settings. When clicking on a file to edit in the application, depending on the word document program being used in the background, the ability to save the file after editing it is tied to the original file path and therefore means that any edits are saved to the file in the application. The next user story we implemented by default is US06: As a user, I want to be able to access all drives. In order to import and store documents, the built-in system already has the ability to search and access all drives in the computer.

User story 8 was fulfilled by accident. US08: As a user, I want to be able to upload my data to the cloud. In the process of attempting to run our code from our desktop, we inadvertently had a few significant errors that affected file paths for export. Some of the developers were saving the GitHub files into Window's build in OneDrive. While not on purpose, this demonstrated that, after logging onto a different computer and loading a personal Microsoft Windows OneDrive, the settings were indeed saved to the cloud. It just so happens that the cloud storage was automatically put on Windows OneDrive and not any self-made servers.

The first user story we ended up implementing was US09: As a user I want to sign in to a profile. In order for us to be able to realistically use the program and save user settings, we needed to be able to have separate users log in and log out. Iteration 2 was useful in this regard in that it forced us to be able to enact this user story before beginning in earnest on our first user stories for the final application. The next automatically enacted user story was US10: As a user, I want to be able to control who has access to the info, and US11: As a user, I want to be able to share documents. The process of having individual users that have the ability to only have certain permissions created a situation where by default, all files are automatically shareable. The next implemented user story was US07: As a user, I want to have the ability to export data. If the user wished to share files outside the application, this previously enacted user story allowed us to export and share files regardless because of its implementation during Iteration 2.

The next user story implemented was US12: As a user, I want to be able to sort the documents. This was one of the last functions we added around the same time as the search function. In the top tab next to the Search bar, there is a Sort by: A-Z and Z-A function that allows the user to alphabetize the currently visible files from A to Z and in reverse from Z to A.

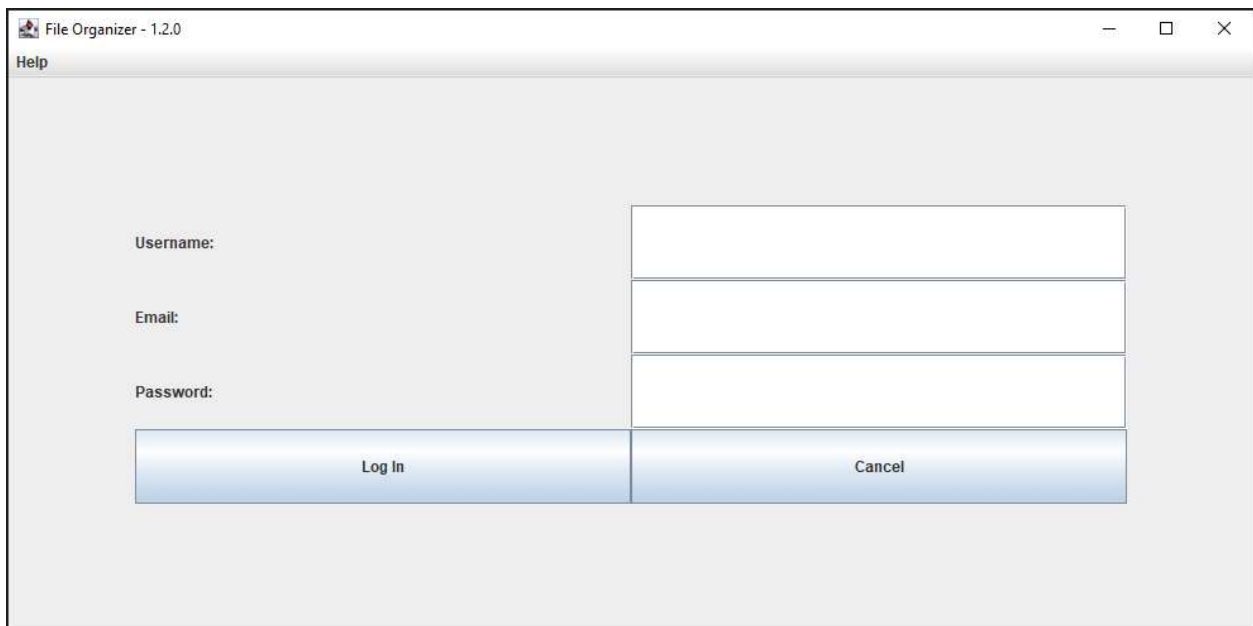
We wanted the ability to navigate through the application easily but realized that, since the files are visible on the left side panel, back and forward buttons were unnecessary for US13: As a user, I want to have navigation buttons to move through folders efficiently as well as including the ability to sign out in the file menu. US14: As a user, I want to use the app offline. This application is automatically usable offline and did not get created with the intent to use it online. In the future, we may create a syncing option that allows the user to synchronize with a server and then can use the application offline by clicking an “use application offline only” button. We did not enact the final 5 user stories as those are only tangentially necessary and may be part of future updates. These user stories are: US15: As a user, I want to have a service reminder, US16: As a DIYer, I want to be able to edit the code, US17: As a developer I want to have a user suggestion feature, US18: As a developer I want to be able to use a bug reporting feature, US19: As a developer I want access to copies of the current code.

How to run the Application

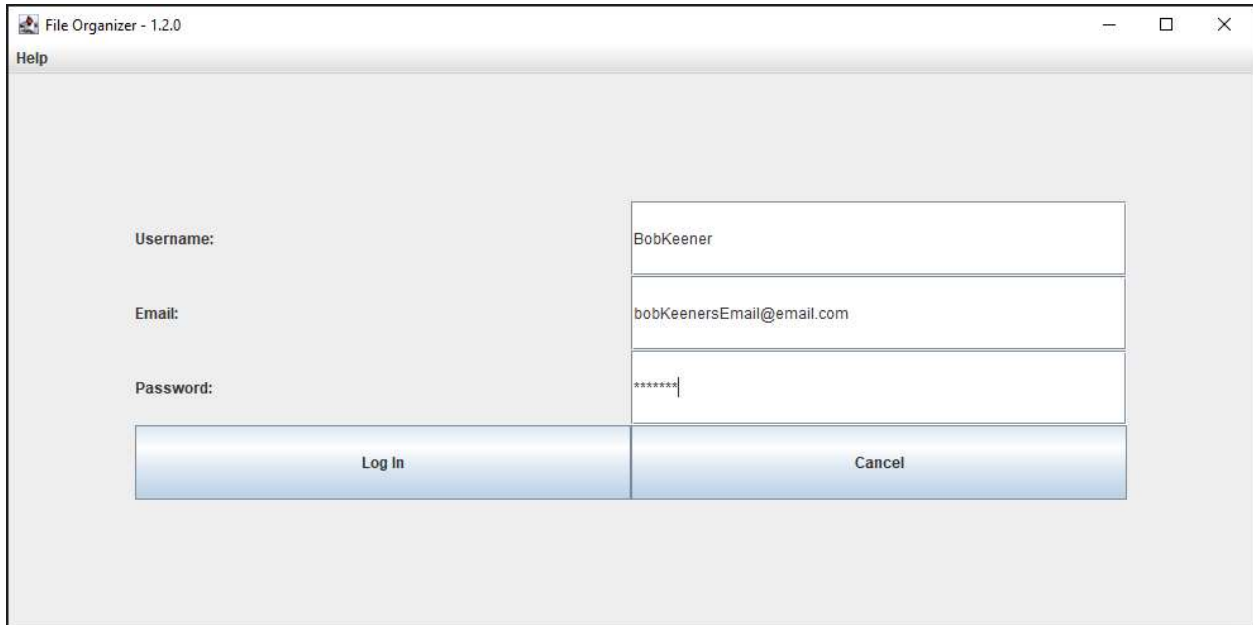
After starting the application by clicking run, the user is first met by the opening screen:



From here, there are a few options, including the ability to create a new profile, import a different user's profile data from outside the application, or export current user data to the desktop. After click on "Create New Profile", we are met with this screen where we can enter new information for our primary user.

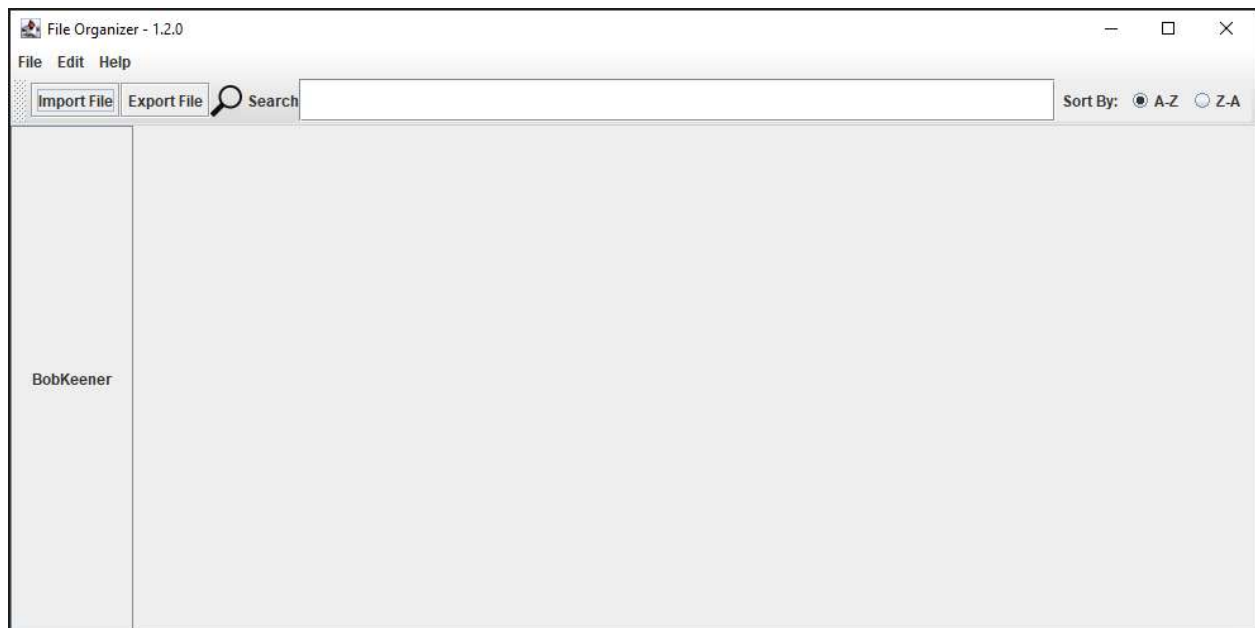
The screenshot shows the same window titled "File Organizer - 1.2.0". The main area now contains a login form. On the left side of the form, the labels "Username:", "Email:", and "Password:" are aligned vertically. To the right of each label is a corresponding text input field. These three input fields are stacked vertically. Below the input fields, there are two buttons: "Log In" on the left and "Cancel" on the right, both with a blue gradient.

After typing in user information, the user will be able to access the profile again later after signing out. Our example shows Bob Keener creating a new profile “BobKeener” with one word and no spaces.

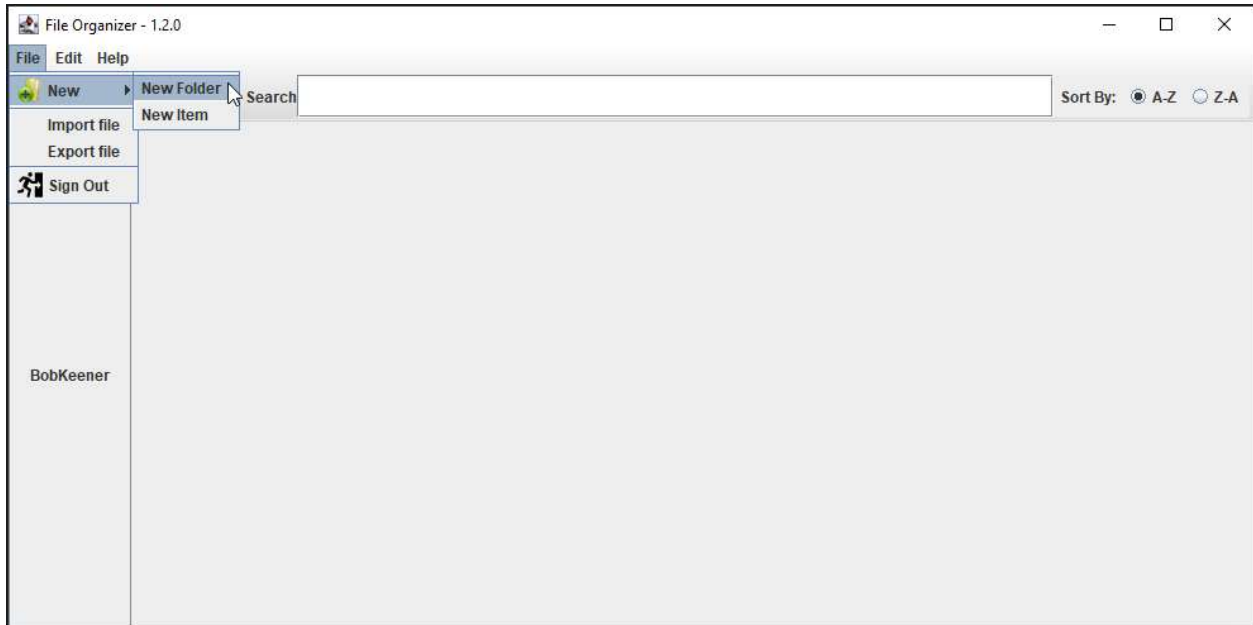


The screenshot shows the 'File Organizer - 1.2.0' application window. It has a title bar with standard window controls. Below the title bar is a menu bar with 'Help'. The main area contains a login form with three input fields: 'Username:' with the text 'BobKeener', 'Email:' with the text 'bobKeenersEmail@email.com', and 'Password:' with masked characters '*****'. Below these fields are two buttons: 'Log In' and 'Cancel'.

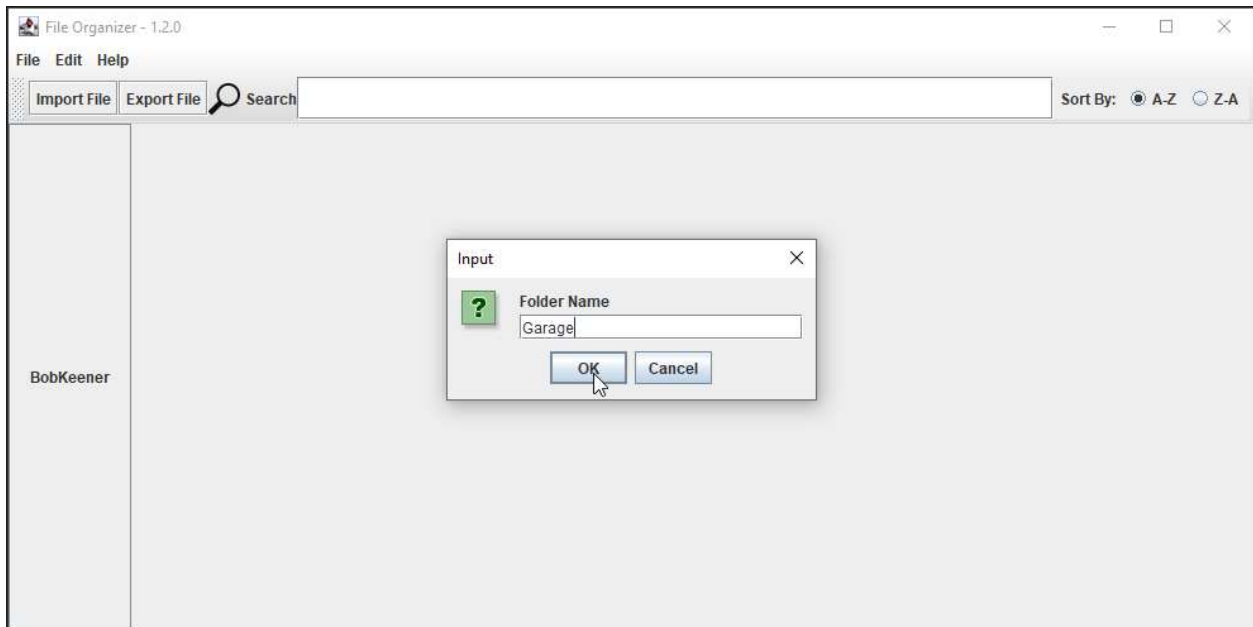
After clicking on “Log In”, the user will be met with the primary user folder where all the other folders are visible as long as he has privileges to access them. In the beginning, no folders will be available until the user creates one.



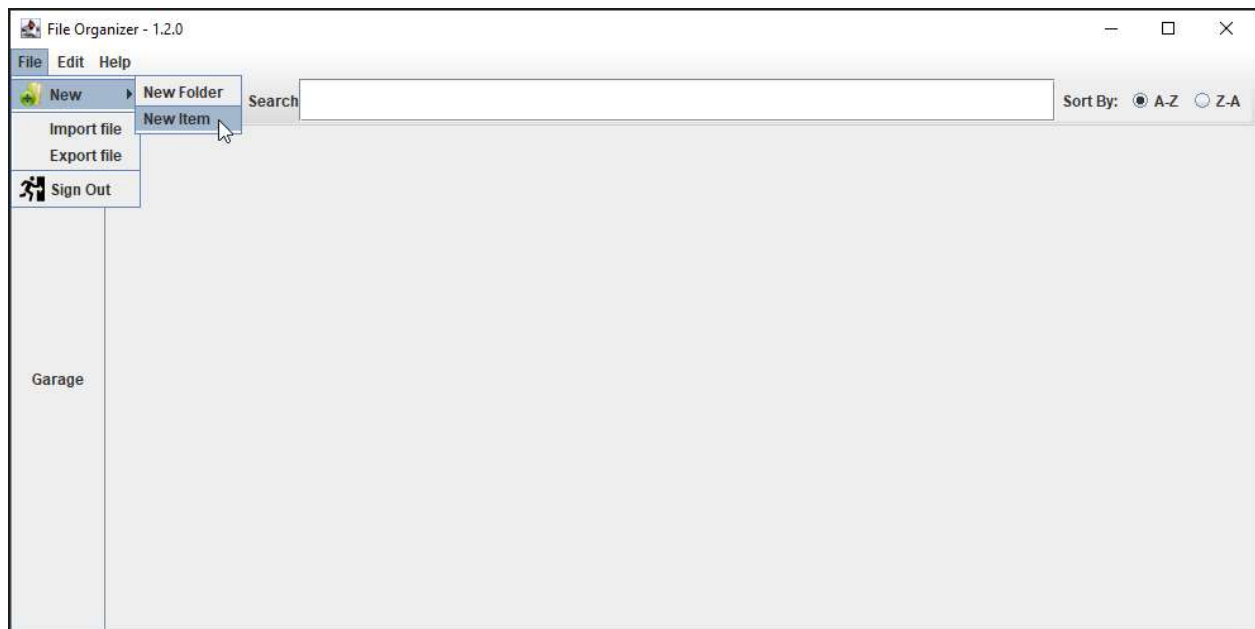
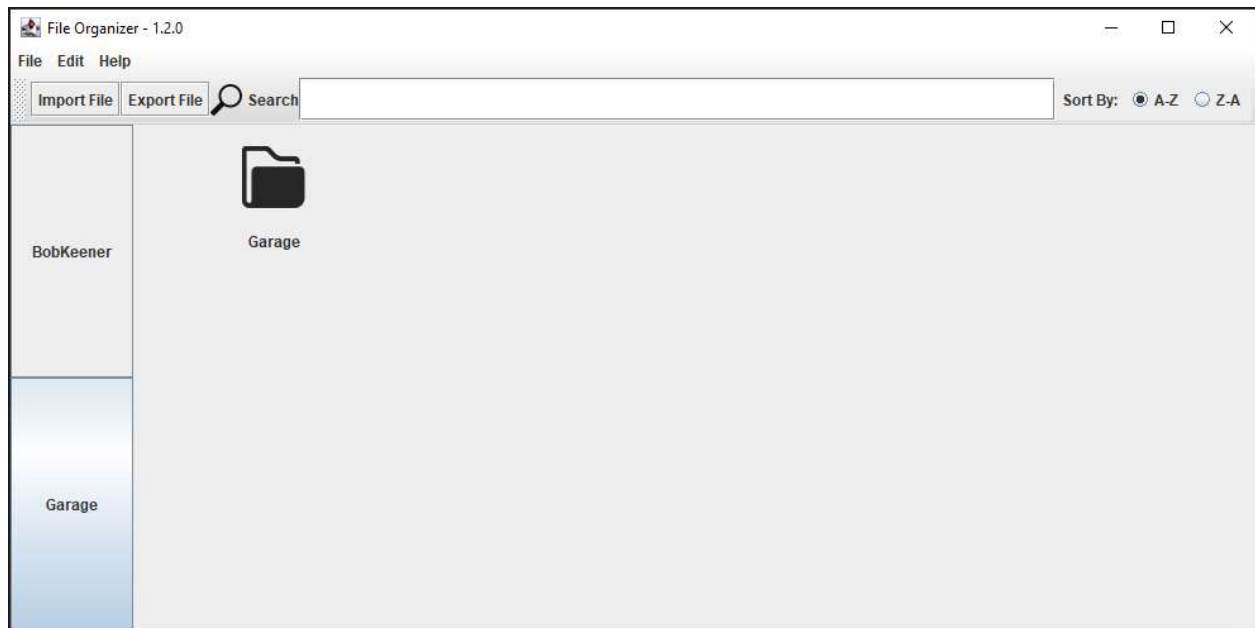
For example, for the user to create a new file to function as one of the objects around the house, they would click “File” => “New” => “New Folder”. They will then be prompted to name this new room’s folder.



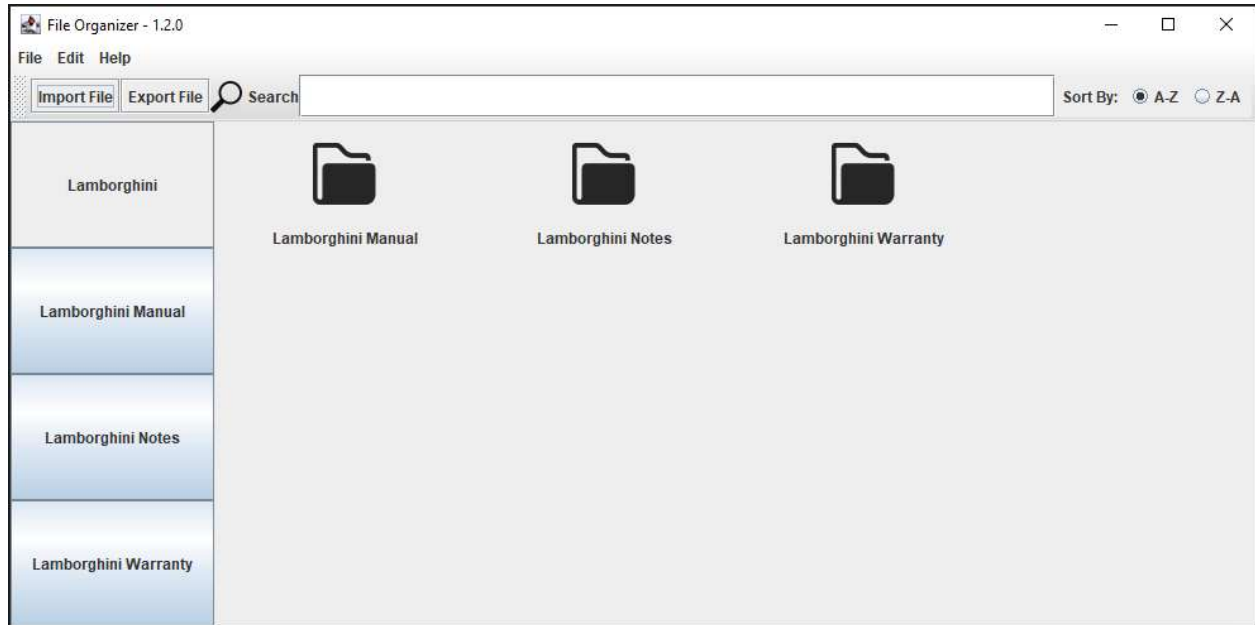
In this example, the user “BobKeener” is creating a new folder for their “Garage”.



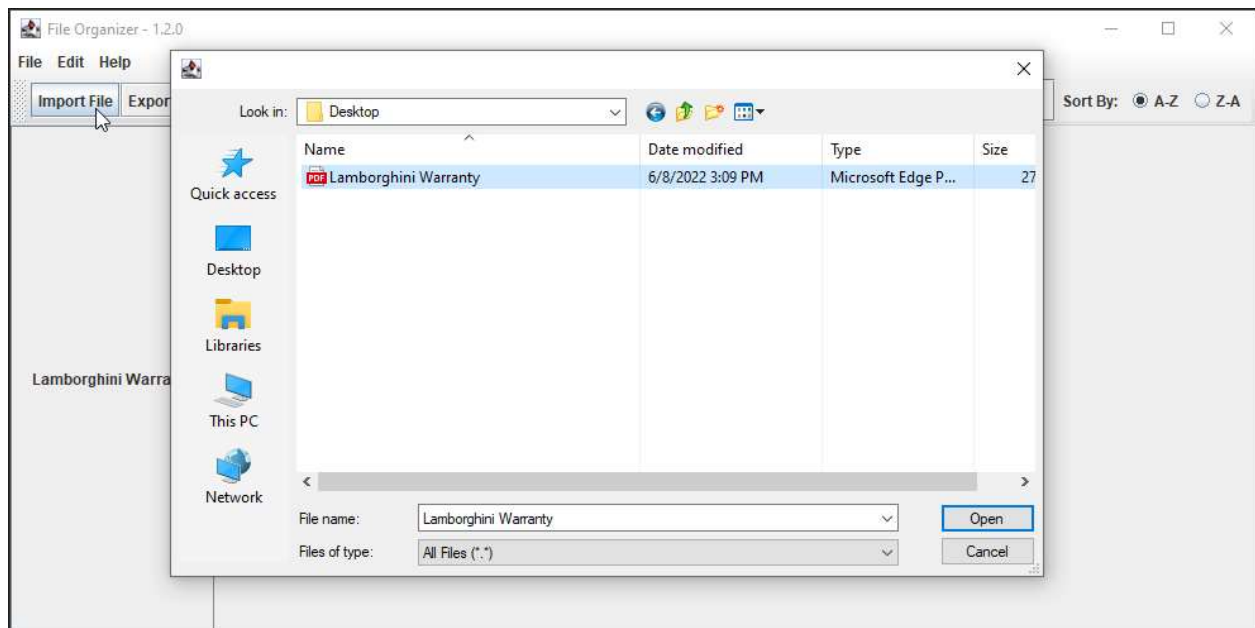
After the “Garage” folder is populated, the user can then click the folder to go into the room and add files within the room’s folders.

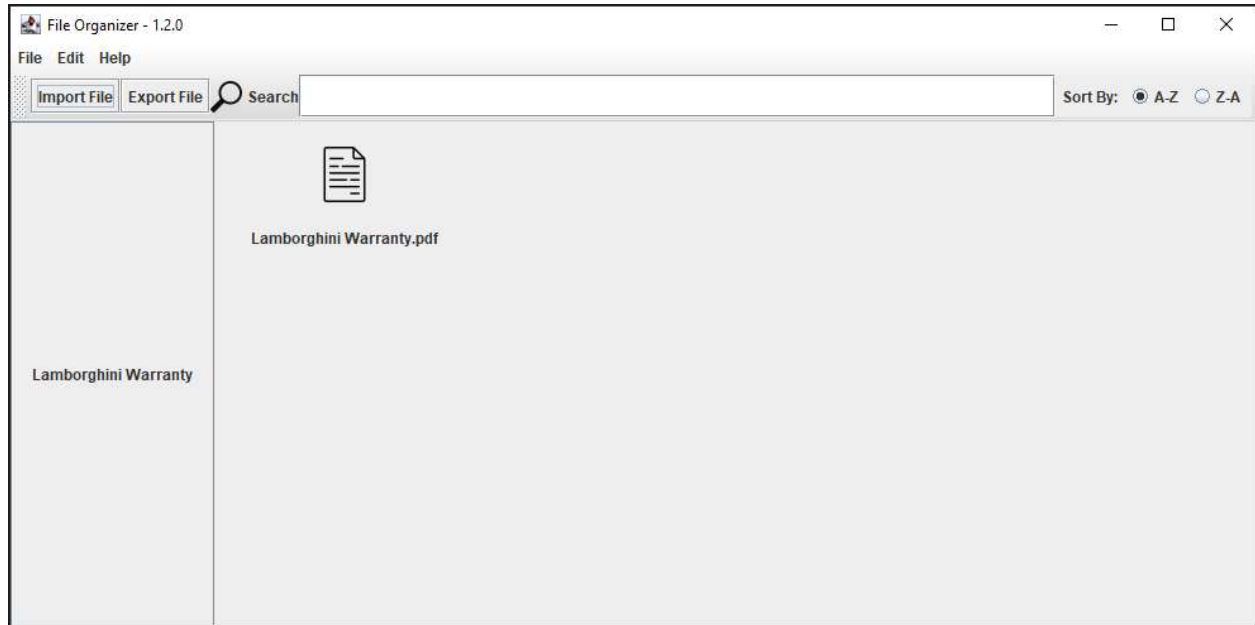


After typing in the name of the object that is being added to the “Garage”, in our case a “Lamborghini”, the folders for the object’s “Manual”, “Notes”, and “Warranty” will then be populated.

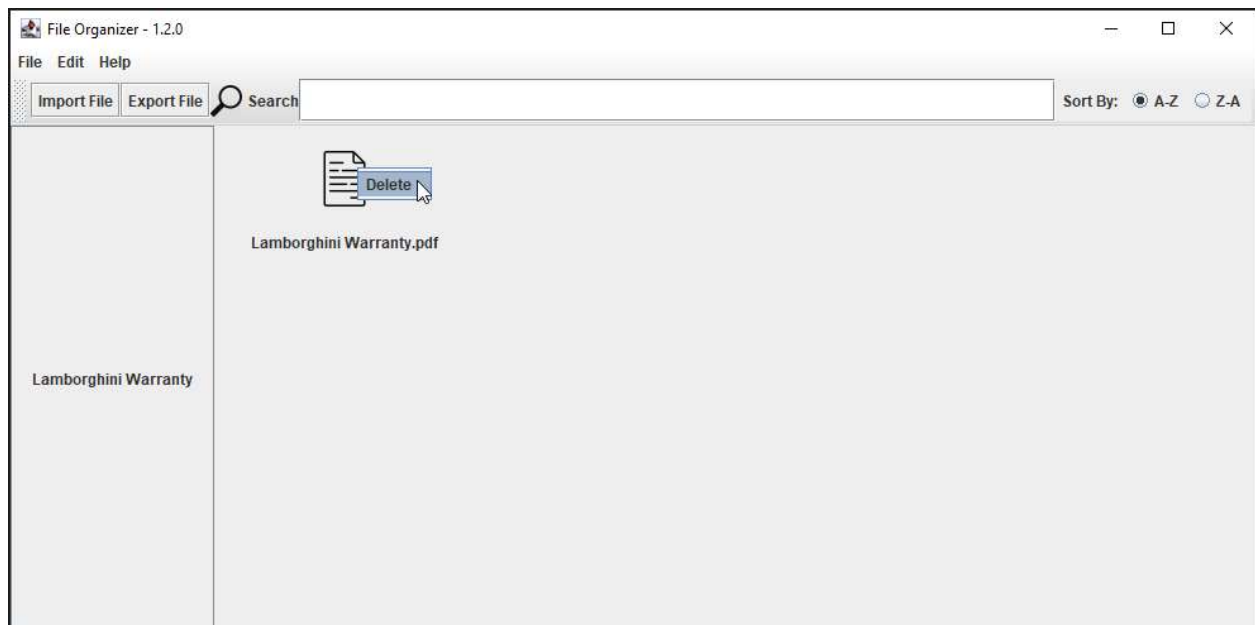


After clicking into the “Lamborghini Warranty” folder, the user can then click on “Import File” and proceed to add the necessary documents to the folder. In this case, we are pulling the PDF for the “Lamborghini Warranty” from the Desktop and importing it to the “Lamborghini Warranty” folder.

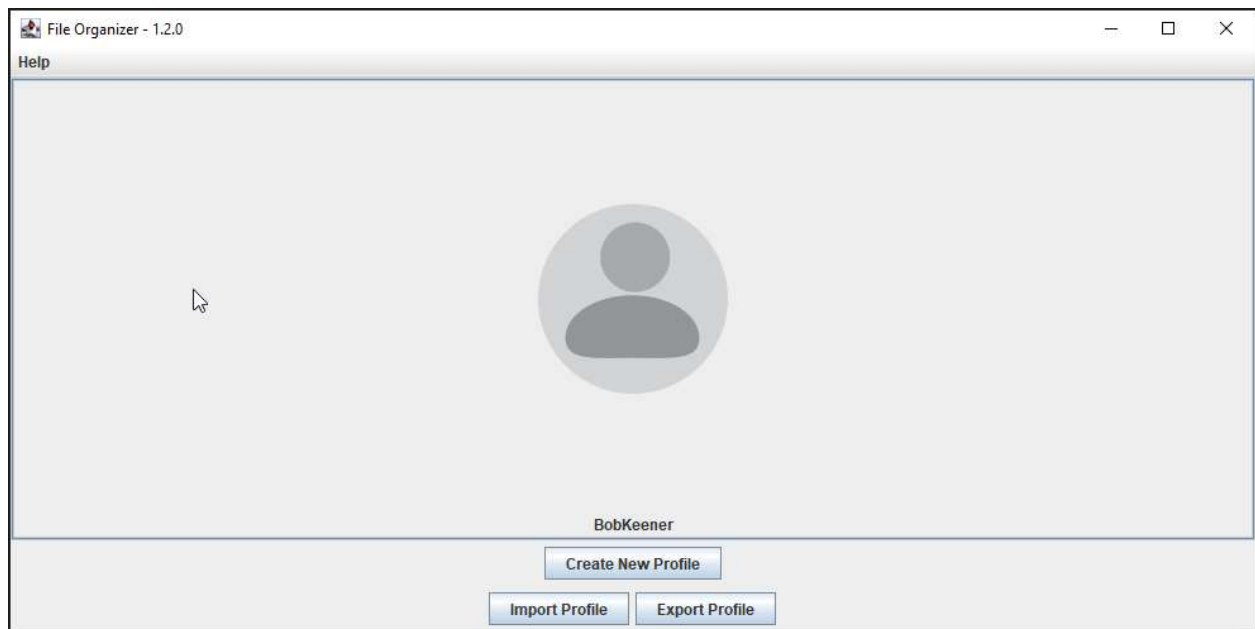
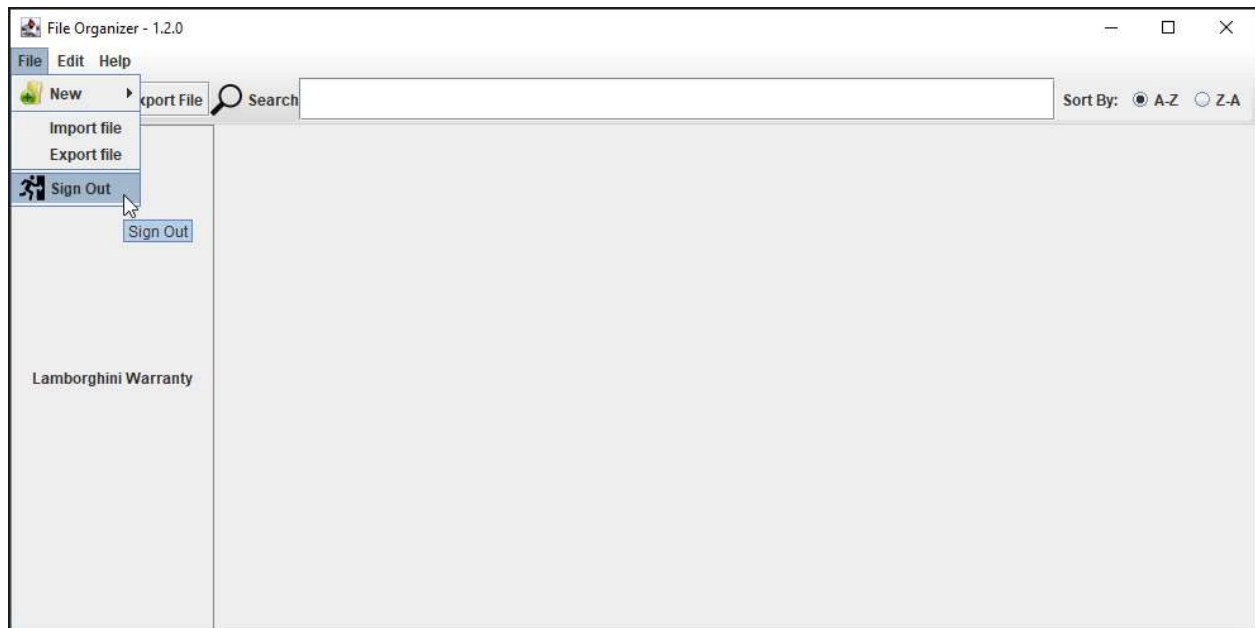




The user can then begin to add as many folders as they want for rooms in their house. They can continue to add new file objects and import any files they need and export those files to the Desktop if they wanted to as well. If they click into the “Search” box, and type in a few letters of what they are searching for and click “Search”, they can then find all the files that contain those letters. Furthermore, they will be able to sort these files by clicking in the circle next to “A-Z” or “Z-A” in the “Sort By:” section just to the right of the “Search” box. The final implementation is the ability to have the user right click on a file and select “Delete” to remove it from the application.



The user can click on “File” => “Sign Out” to be sent back to the main screen.



When the user decides to go back into the application and reload their profile, all they have to do is click on the image of their profile.

Contributions

Michael began by adding user friendly image improvements to the Graphical User Interface. After some time into the project, Michael was tasked with adding an additional feature whereby any attempt to sign-in to the wrong user would prompt the user to create a new profile instead and have it added to the main Log-In screen. This newly added “Create New Profile” feature could then be signed out of and signed back in at will. Since Michael’s skills with coding are not as experienced as the rest of the group, he functioned as the Scrum Master and ensured that all meetings were held weekly in a timely manner as well as being the Keeper of the Papers. Michael also assisted with the typing in both Deliverables 1 and 2 as well as typing this document Deliverable 3’s sections for the Introduction, Requirements Addressed, How to run the Application and Contributions.

Jasharn began with creating a user class for the project. Much of the inspiration came from the TCSS305 class for the vehicle rental program. Essentially, this process was replicated here for our program. These user objects would contain the fields to represent name, email, password, and privileges. Another class Jasharn created was the registration class. This class essentially read from our user TXT file, created users, and then filled the fields appropriately. This class also has methods to check login results and return true or false depending on if the user is registered or not. Jasharn also created a versionControl class to simply return the version of the program.

Working alongside Patrick, Jasharn also helped create the fileTools class that interacted with the files to return an array of desired files to support functionalities on the main view of the program. Jasharn also created registrationTests, fileToolsTest, and versionControlTest. One of the challenges in doing this was learning how to test functions that were represented in the GUI. Once these methods that created the functionality were created in the FileTools class, it was much easier to pass them data and ensure they worked correctly. Jasharn assisted in typing Deliverable 1 and 2 and writing the test section of this deliverable.

Patrick started off by creating the basis for the GUI. He also assisted with the typing in both Deliverables 1 and 2. Implementing the button functions required communicating with other teammates’ codes. As such, many portions of the code were looked over and edited as needed. The main contribution to this final deliverable was building the packaging and reformatting the code for the JAR file version of the code to function correctly. This took much more effort than initially understood and caused a significant reworking of the code.

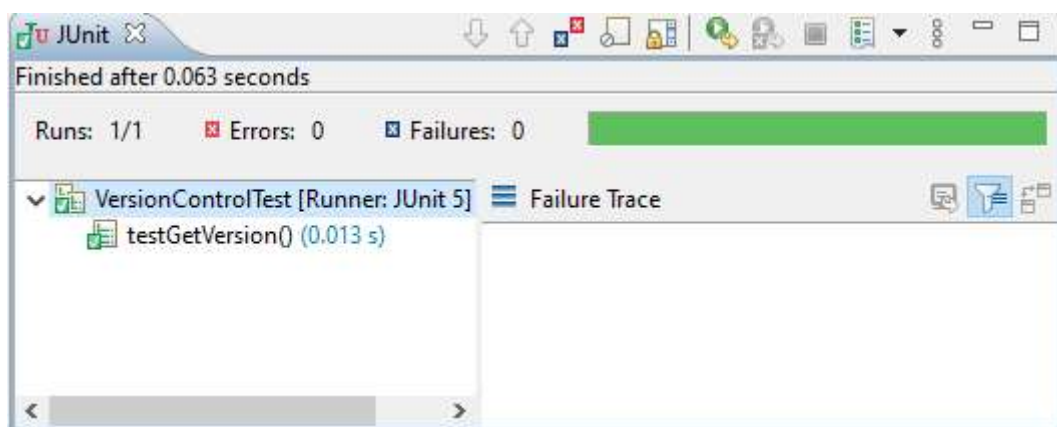
Trevor began by ensuring that the entire team was well versed in GitHub and was responsible for managing the repository. He assisted with the typing in both Deliverables 1 and 2 as well as creating the login screen panels, and the ImporterExporter class for reading and writing to a CSV file. He added some methods to the FileView class like incorporating the importing of files and adding a right click to delete popup. Trevor also created and managed the website.

Tests

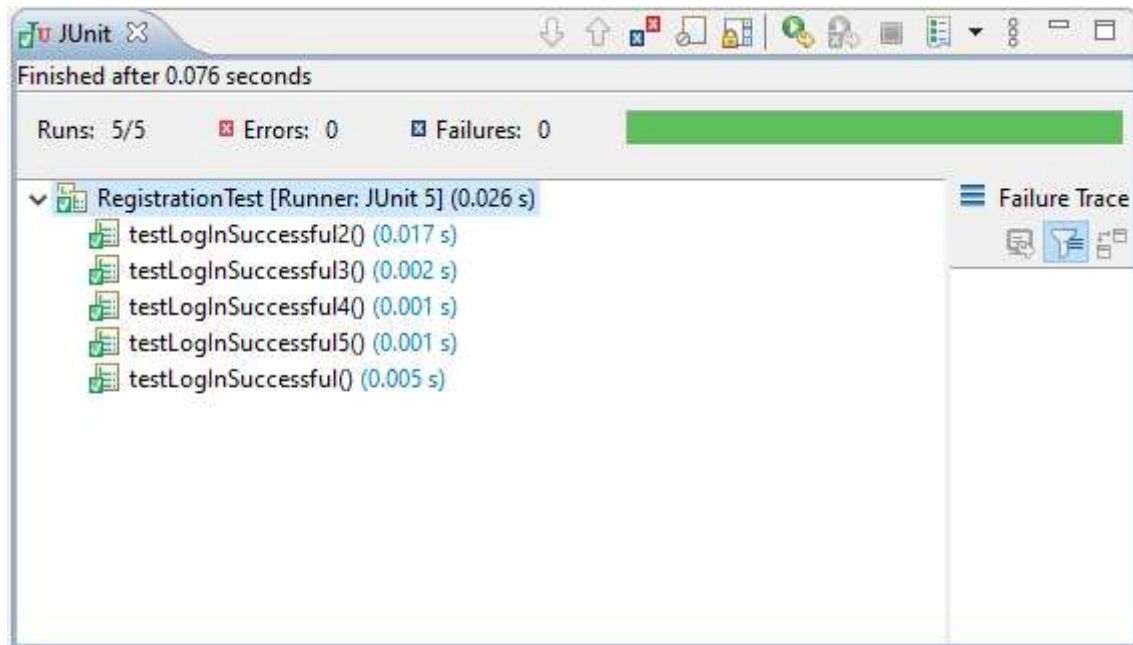
For testing, we decided to utilize JUnit for all of the tests. The first testing class that was created was the `versionControlTest.class`. This was very straightforward and only required a simple comparison that the getter within this class returned the correct version. The registration class was much more difficult to test. This required creating instances of the registration class so that testing was possible to begin with as well as providing the class constructor with different file paths, so that testing would not mess with our current file hub. This testing for this class was done to test the `loginSuccessful` method that returns a boolean result with different cases. One of the cases was being all the correct fields that would return the boolean true. The second case is for correct fields but an incorrect password which would return the boolean false. The third case is for all correct fields but an incorrect email which would also return the boolean false. The final case is for having an incorrect username which should also return the boolean false. This class is also tested to ensure that the `addToList` method works within the registration class. Simply adding using the `add` method, creating a new user, and then trying to login with this user did the trick.

Finally, the `FileTools` testing was the most difficult to code. This is because many of the methods worked within our file hub. The idea was to create a new file array and pass it to the method. Using this, we could create a sorted version and then compare that the two arrays were the same after the `SortFiles` method call. This same trick worked as well for the `reverseSortFiles` but, in this case, the second array created would be a list that is sorted in reverse order and then call the method, make the comparison, and ensure they are the same.

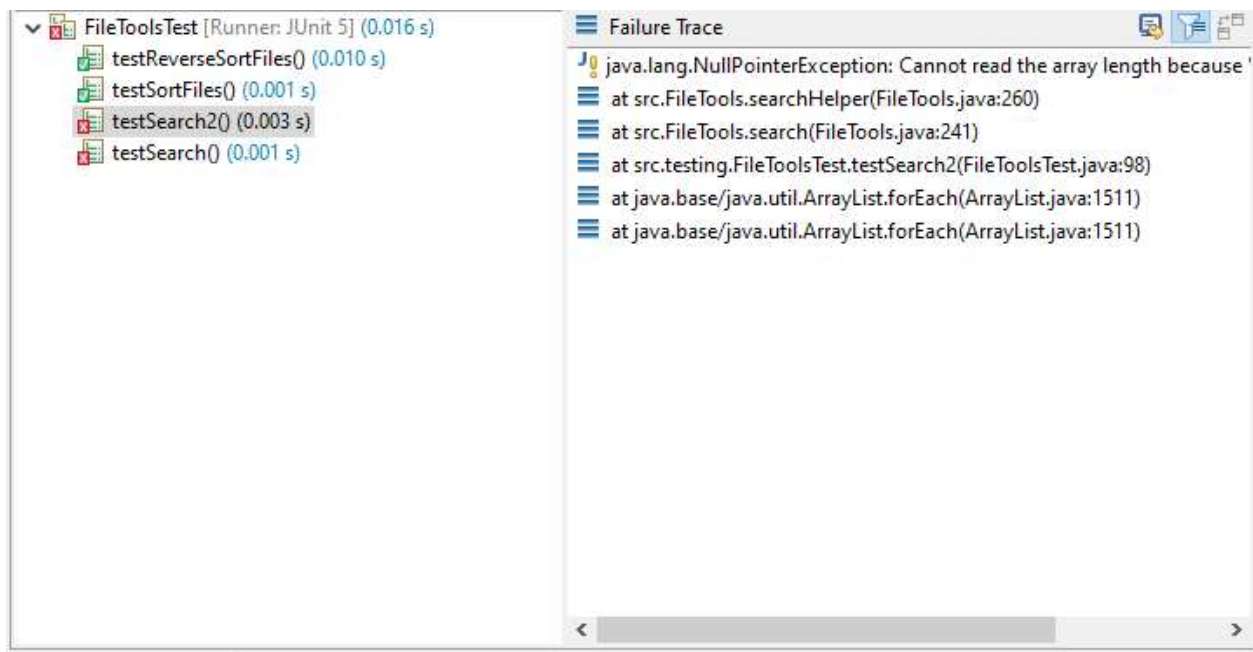
Testing can sometimes be difficult to incorporate after the fact code is written. If we were to go back and do this again, we would either write tests first or, during the coding of the project, keep in mind to structure the project in a way that is testable. It was, however, a good learning experience after the fact and very interesting to attempt on a larger scale project than previous classes. Below is a representation of the JUnit tests:



Test for VersionControl class and GetVersion method

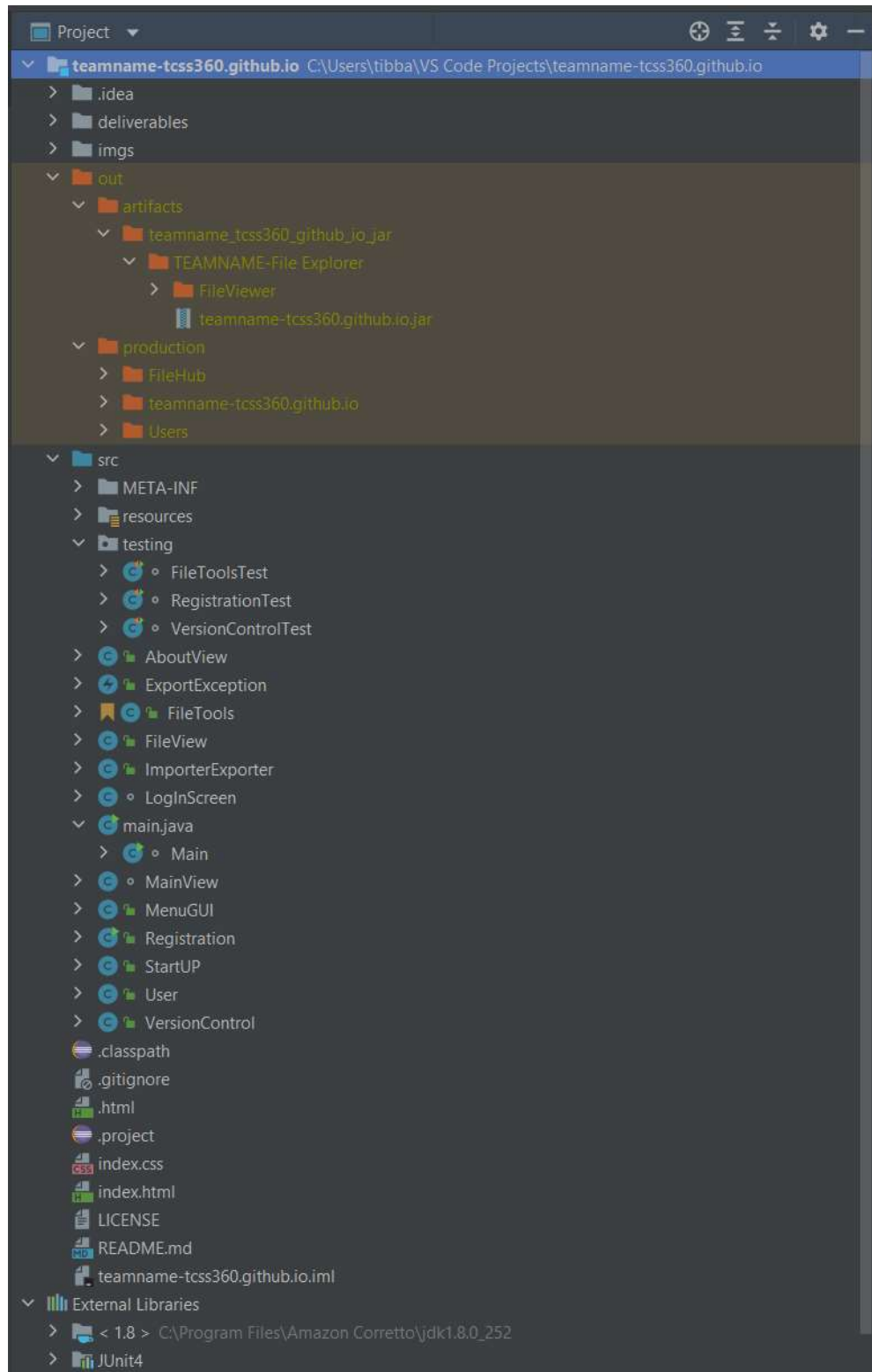


Test for Registration class and LogIn method



Test for FileTools class and ReverseSort, SortFiles, Search2, and Search methods

Source Catalog



Emergency Contacts

1. Trevor Tomlin : ttomlin@uw.edu
2. Patrick Tibbals : ptibbals@uw.edu
3. Jasharn Thiara : jthiara@uw.edu
4. Michael Theisen : mtheis@uw.edu