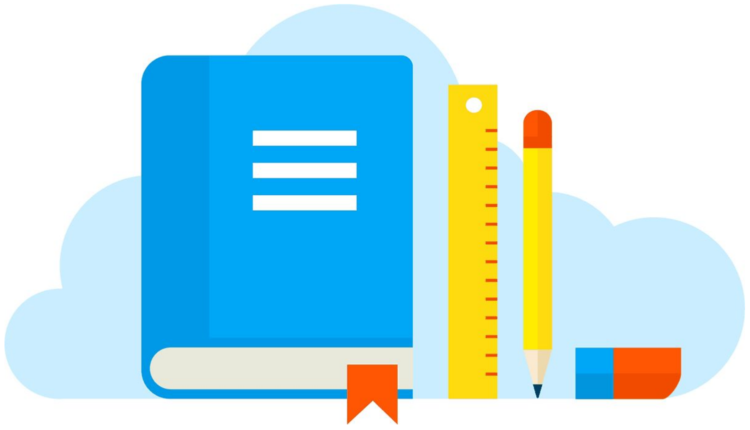


Homeroom Helper

A reading tool for teachers, by teachers.

Kaitlyn Bosch

Senior Project, Spring 2019

Introduction

Homeroom Helper is a teaching assistant tool that was inspired by teachers, for teachers. The tool contains four modules, each with the intent to provide guidance on improving student reading. The Classroom Manager tracks basic student data, specifically name, the student's reading level from the beginning of the school year, the student's current reading level, and the student's goal reading level to achieve by the end of the year. The Student Conferences module allows the teacher to take specific notes on each student, as well as view information related to making intimate conferences more productive. The Help a Reader module provides actions the teacher can take based on what they see in the classroom, as well as in individual student's reading abilities to help them focus and improve. Finally, the Analytics module provides overall class information, as well as reading progression for individual students.

The reading levels, conference information, and reading flags in this program were determined based on classroom documentation provided by 4th grade teachers at Stonybrook Elementary School in Kinnelon, New Jersey. They are sourced from various references from the Teacher's College of Columbia University.

In essence, the reading levels are an A-Z scale that defines the comfort level and abilities of students in each letter band. A readers are in the lowest band, and are learning to read simple books, with lots of pictures and basic vocabulary. Z readers are reading at a high school level, with complex plots and vocabulary, in addition to historical, political, or cultural references. While this program is intended for elementary-level teachers, the program supports any reading level from A-Z.

Student conferences are very important to a student's success, and many teachers have difficulty not only tracking what they have discussed with each student, but also with handling specific, yet common issues. The student conferences module handles both of these issues, by allowing the teacher to write notes for each student, but also to learn about each category of conference in the same convenient window.

Help a Reader has two main goals: to educate or remind the teacher of what each specific reading level entails, and to give them recommendations on how to increase a reader's level. This module contains all the provided data on what each reading level consists of, without being overwhelming to look at. It also provides an analysis tool to gauge how well or poorly students react to reading assignments.

Finally, the analytics panel provides some simple, yet powerful charts to assist the teacher in her overall class analysis. It will help the teacher hone in on what level her class is generally at, how many students are at each level, and how each student is doing individually. With this data, each teacher can more easily make decisions on how to best help their class succeed and get ahead.

Installation Instructions

Step 1: Move .ZIP folder to desired location

Step 2: Unzip files

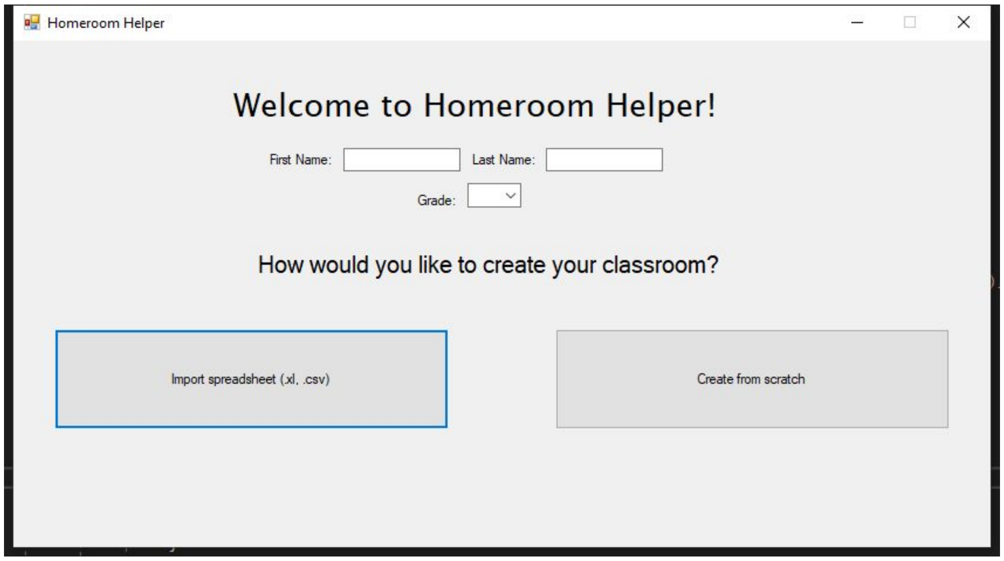
Step 3: Run the .exe file!

If desired, make a desktop shortcut by right-clicking on the .exe file, and clicking “Create Shortcut”. Then move this to your desktop and you can start the program easily!

User Manual

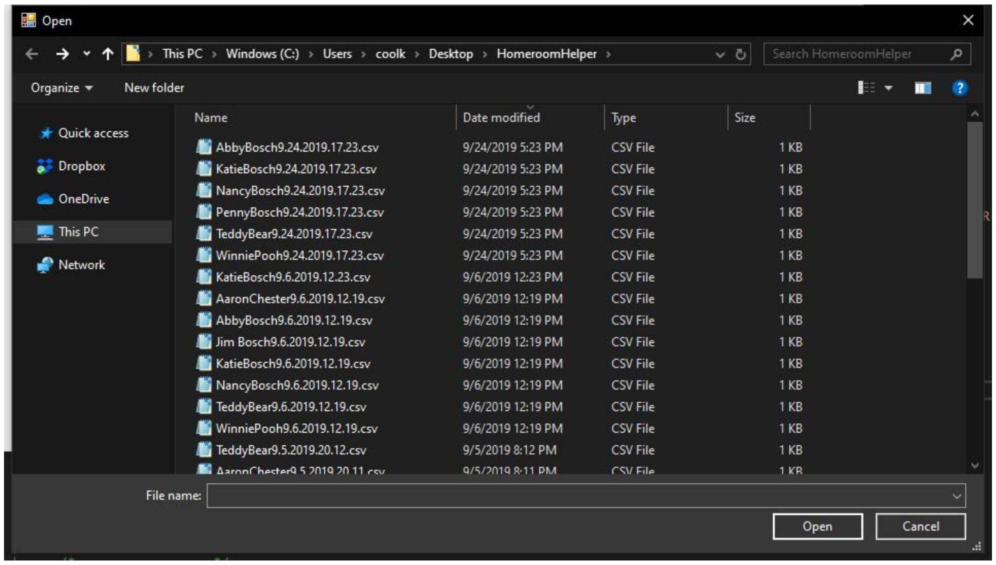
First Run

The first run of the program will start with this screen:


Homeroom Helper
Welcome to Homeroom Helper!
First Name:
Last Name:
Grade:
How would you like to create your classroom?
Import spreadsheet (.xl, .csv)
Create from scratch

Here, you will enter your information (first name, last name, and grade level). Then, you will select which method you would like to use to create your classroom. Importing requires having a student file in a particular format, namely one that was exported using this program by another teacher. All of the data from a student file will be loaded into the program, and then added upon.

When you click on the “Import spreadsheet” button, the HomeroomHelper directory will appear, prompting you to select the file(s) you would like to import:

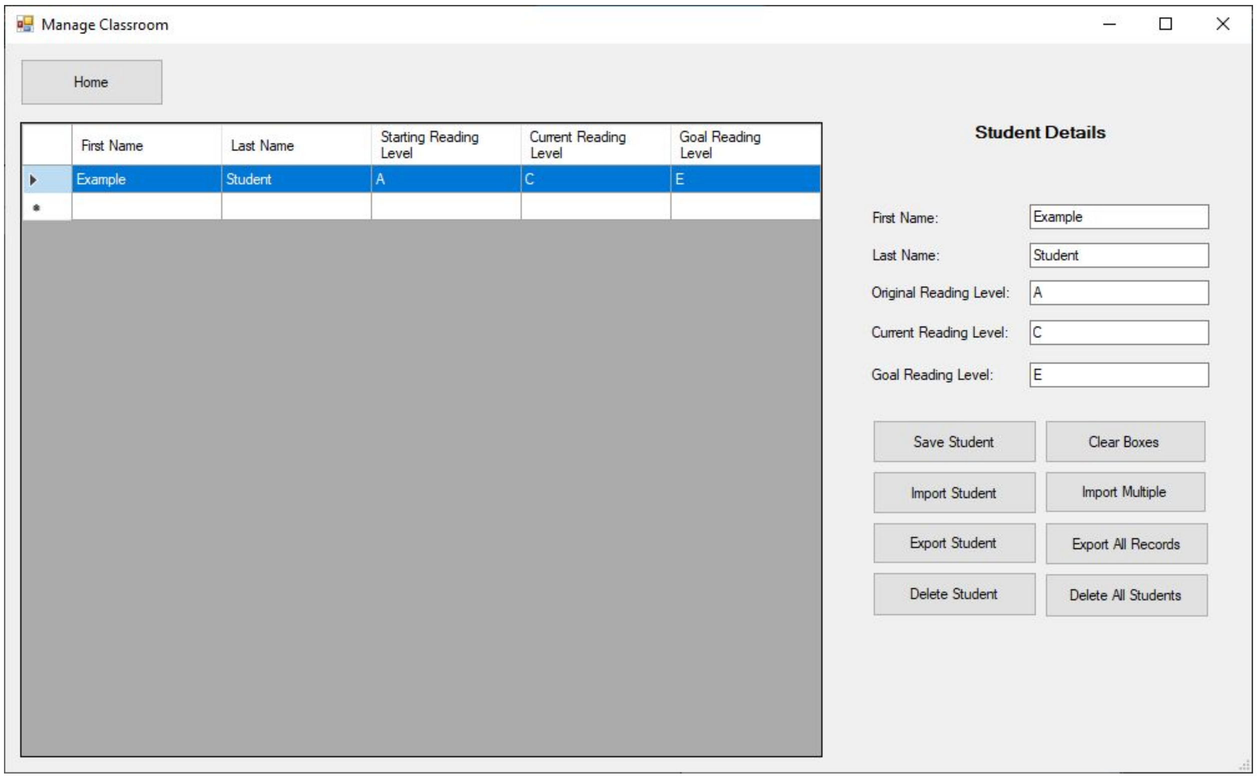


Name	Date modified	Type	Size
AbbyBosch9.24.2019.17.23.csv	9/24/2019 5:23 PM	CSV File	1 KB
KatieBosch9.24.2019.17.23.csv	9/24/2019 5:23 PM	CSV File	1 KB
NancyBosch9.24.2019.17.23.csv	9/24/2019 5:23 PM	CSV File	1 KB
PennyBosch9.24.2019.17.23.csv	9/24/2019 5:23 PM	CSV File	1 KB
TeddyBear9.24.2019.17.23.csv	9/24/2019 5:23 PM	CSV File	1 KB
WinniePooh9.24.2019.17.23.csv	9/24/2019 5:23 PM	CSV File	1 KB
KatieBosch9.6.2019.12.23.csv	9/6/2019 12:23 PM	CSV File	1 KB
AaronChester9.6.2019.12.19.csv	9/6/2019 12:19 PM	CSV File	1 KB
AbbyBosch9.6.2019.12.19.csv	9/6/2019 12:19 PM	CSV File	1 KB
Jim Bosch9.6.2019.12.19.csv	9/6/2019 12:19 PM	CSV File	1 KB
KatieBosch9.6.2019.12.19.csv	9/6/2019 12:19 PM	CSV File	1 KB
NancyBosch9.6.2019.12.19.csv	9/6/2019 12:19 PM	CSV File	1 KB
TeddyBear9.6.2019.12.19.csv	9/6/2019 12:19 PM	CSV File	1 KB
WinniePooh9.6.2019.12.19.csv	9/6/2019 12:19 PM	CSV File	1 KB
TeddyBear9.5.2019.20.12.csv	9/5/2019 8:12 PM	CSV File	1 KB
AaronChester9.5.2019.20.11.csv	9/5/2019 8:11 PM	CSV File	1 KB

It is recommended to keep all student files in this folder, however you can navigate to wherever you wish to find the file. Once you have selected all the files to be imported, click “Open” and the program will load in each file programmatically.

Manage Classroom

The other method of creating your classroom is to do so from scratch. This will take you to this screen:



First Name	Last Name	Starting Reading Level	Current Reading Level	Goal Reading Level
Example	Student	A	C	E

First Name:	Example
Last Name:	Student
Original Reading Level:	A
Current Reading Level:	C
Goal Reading Level:	E

This is the main view for your class. All students in the database will be displayed here, and you can click on the column headers to sort by that category. You can also access this module from the main menu by clicking on the “Manage Classroom” button.

The textboxes on the right of the window are the key to interacting with the database. To clear them so you can type information in, click on the “Clear Boxes” button.

To add a student, clear the textboxes, then type in the student's information. Click "Save Student" and the student will be added to the database and the current view.

To import one student, click on the "Import Student" button. This will bring up the HomeroomHelper directory where you can select a file to import. If you would like to import more than one student, select the "Import Multiple" button and you can select more than one file at a time.

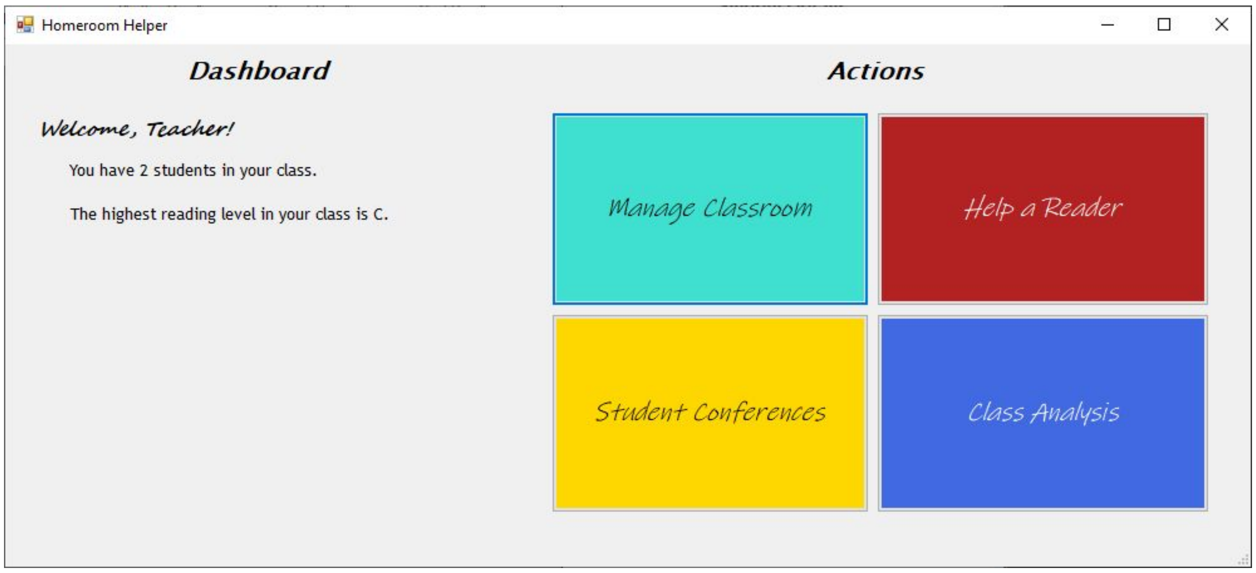
To export a student to a file, select the student in the view that you would like to export, and then click the "Export Student" button. It will automatically generate the student's importable file in the HomeroomHelper directory. "Export All Records" exports all students, each to their own .csv file, in the HomeroomHelper directory.

To delete a student, select them in the view and then click the "Delete Student" button. This will remove all data related to that student from the database, so export the student first if you would like to maintain their information. If there are no more students left in the database after a delete, the program will generate an "Example Student" automatically.

Deleting all students will clear the database of all students, conference information, and analytical information. It is only recommended to do this once all students have been exported, and you want to start fresh (for example, at the beginning of the school year).

Main Menu

After the initial startup of the program, this is the menu you will see upon executing the program each time:



You have 2 students in your class.

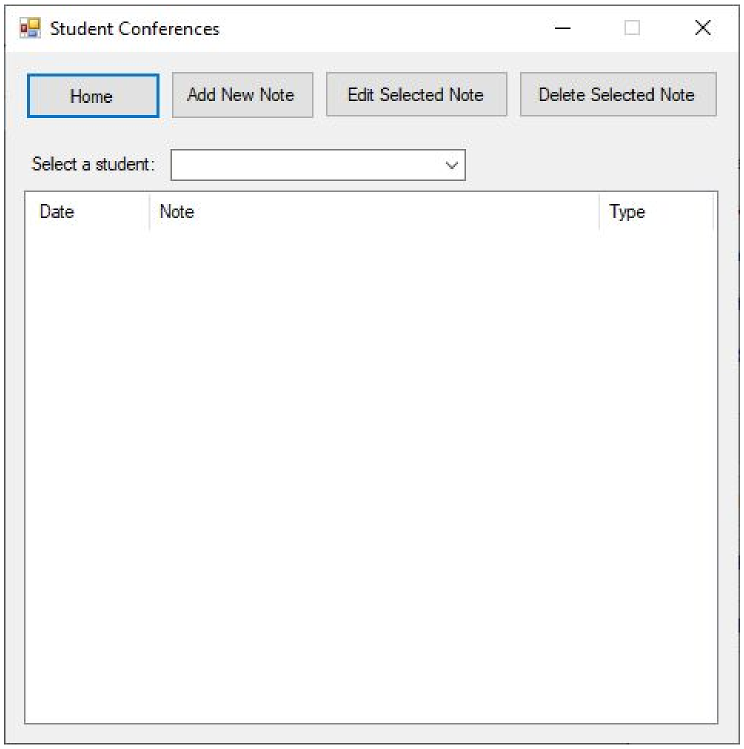
The highest reading level in your class is C.

The dashboard panel displays the teacher's name, the number of students in the class, and the highest reading level. The actions panel displays the four main modules - "Manage Classroom", "Student Conferences", "Help a Reader" and "Class Analysis". Clicking on any of these buttons will open a new window for that module.

Clicking on "Manage Classroom" will bring you to the module discussed above.

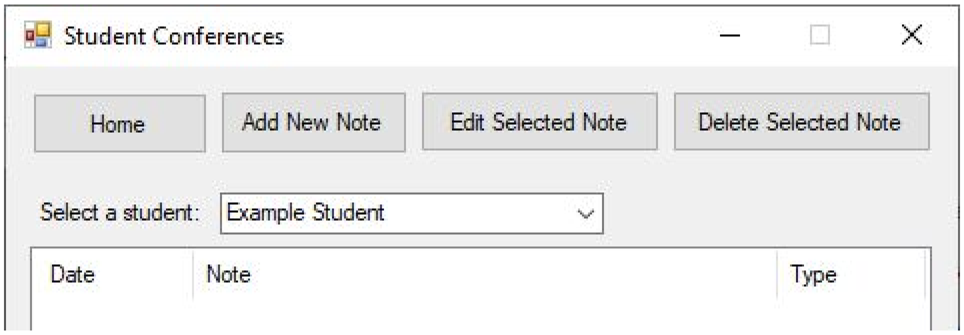
Student Conferences

Clicking “Student Conferences” from the main menu will display this window:

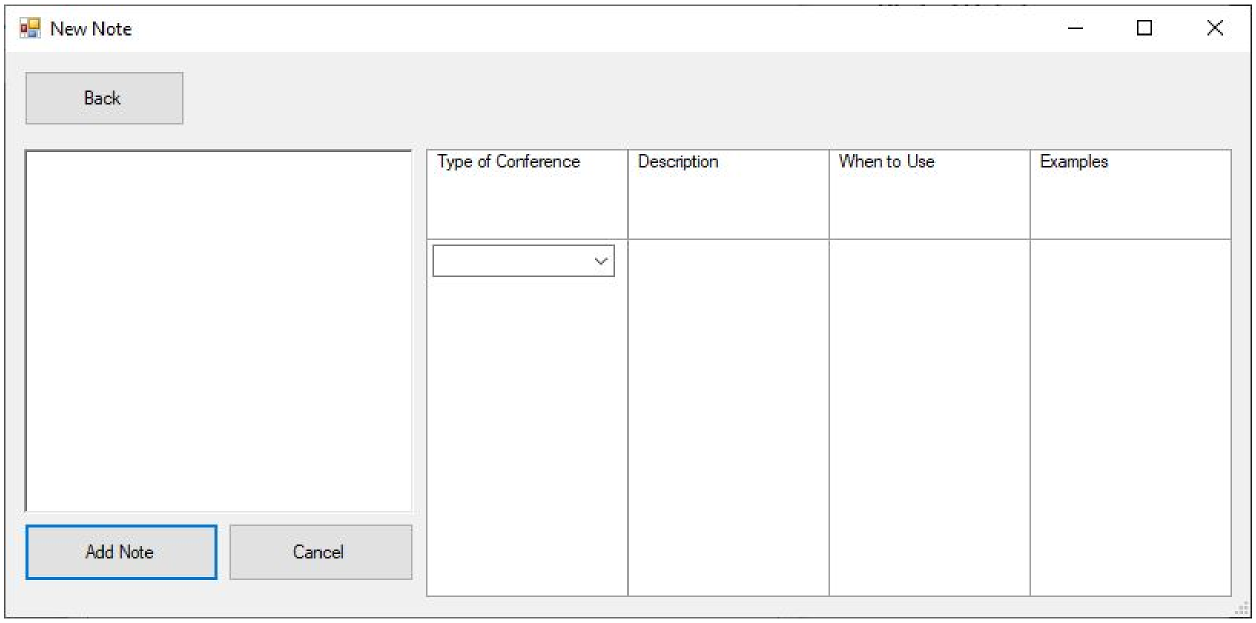


The “Home” button brings you back to the main menu.

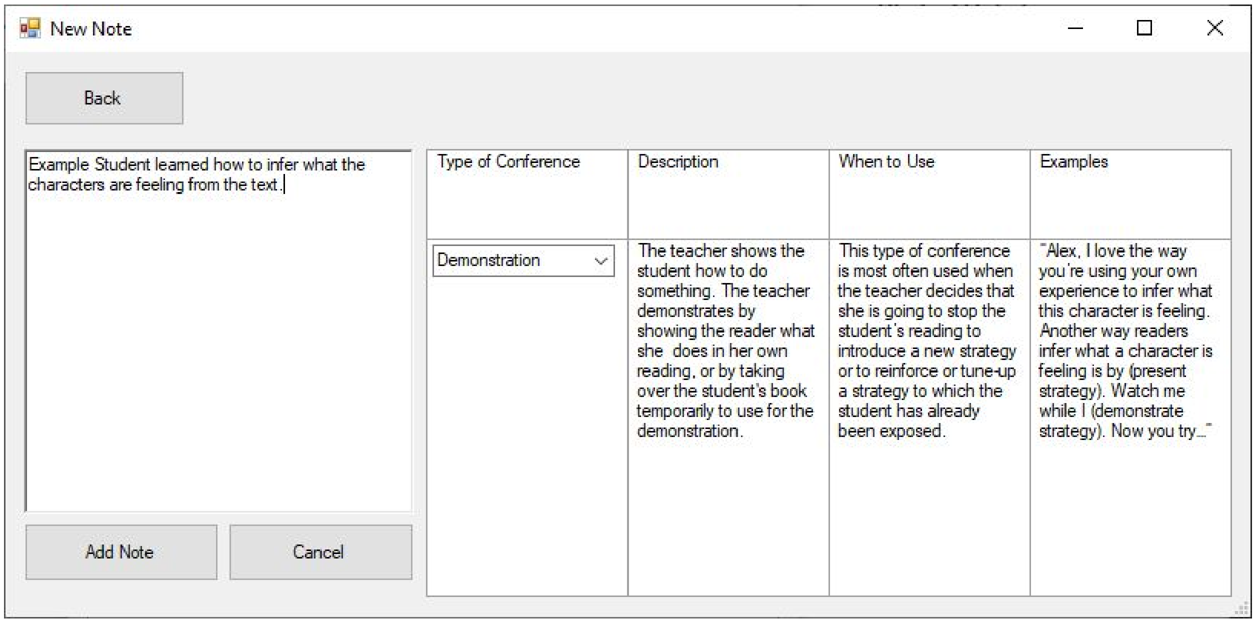
To add, edit, or delete notes, first select a student from the dropdown menu.



Then, clicking “Add New Note” will bring up this menu:



You can enter the note details in the box on the left. On the right, select the type of conference from the dropdown to get details on the selected type. An example note shown below:



Example Student learned how to infer what the characters are feeling from the text.

Type of Conference	Description	When to Use	Examples
Demonstration	The teacher shows the student how to do something. The teacher demonstrates by showing the reader what she does in her own reading, or by taking over the student's book temporarily to use for the demonstration.	This type of conference is most often used when the teacher decides that she is going to stop the student's reading to introduce a new strategy or to reinforce or tune-up a strategy to which the student has already been exposed.	"Alex, I love the way you're using your own experience to infer what this character is feeling. Another way readers infer what a character is feeling is by (present strategy). Watch me while I (demonstrate strategy). Now you try..."

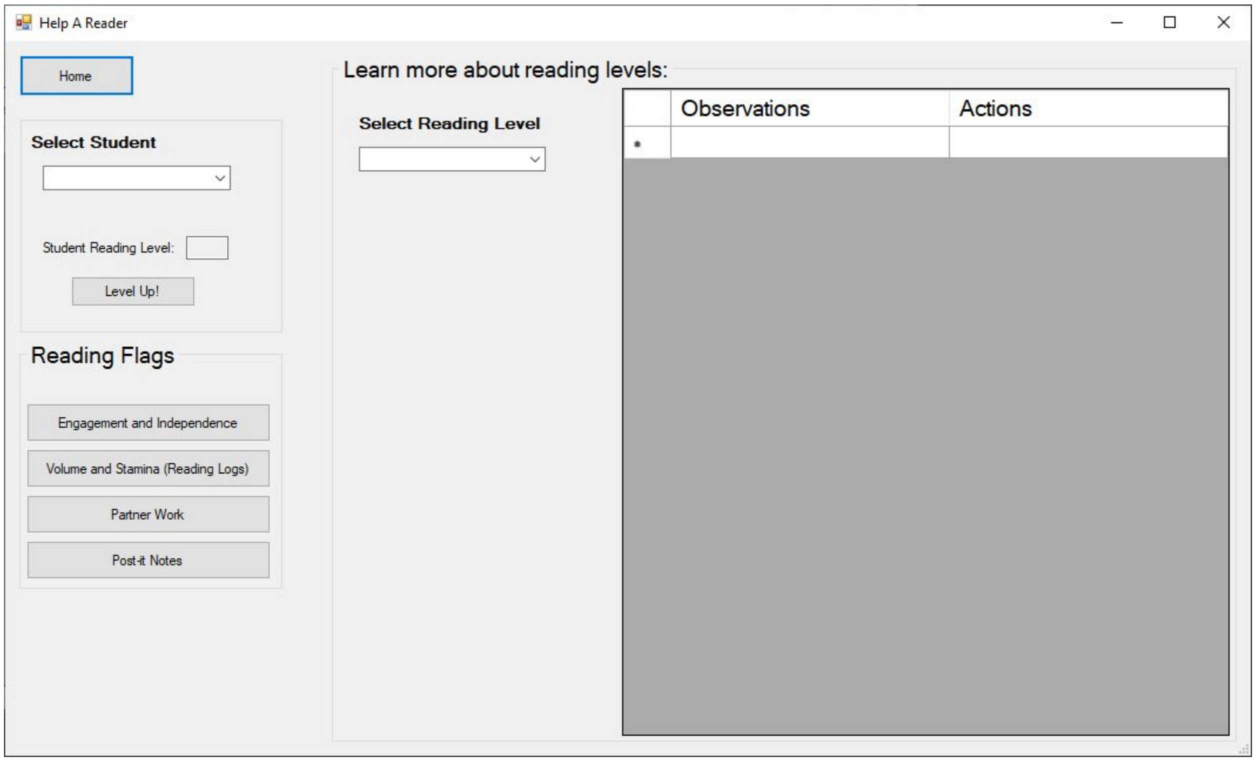
When satisfied, click on “Add Note”. This will save the note’s contents and the type of conference held.

To edit a note, select it in the list and click “Edit Selected Note”. This will bring up the note editing pane again, so you may edit the text or type of conference.

To delete a note, select it in the list and click “Delete Selected Note”. This will remove it from the view and the database. Note: to keep notes simple and accessible, deleting a note does not bring up a prompt before the delete. Use with caution.

Help a Reader

The next module, Help a Reader, looks like this:



Reading Levels

There are several connected purposes for this module, so they are contained in the same space. First, you can view a student's current reading level by selecting them from the dropdown under "Select Student". To increase their level by 1, click the "Level Up!" button.

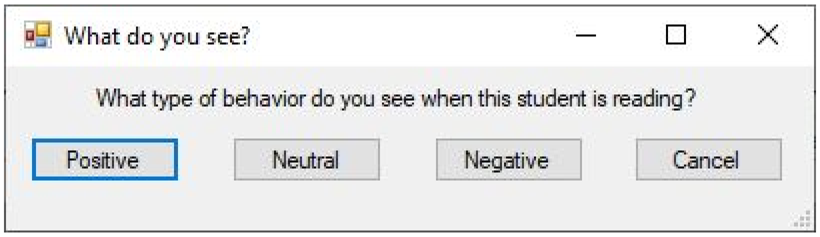
For the most ease when changing reading levels, the level information is contained in the panel on the right hand side. Simply select the level you would like to view from the dropdown and it will display. Note: some reading levels are grouped, which means the content may not change if two levels are part of the same group. For example, levels C and D are in the same group, with C being the lower end and D the higher end of the student's abilities.

Flags

Related to the reading levels are reading flags. A reading flag is a method of helping students improve their reading skills based on the teacher's observations. There are four categories of observation areas: Engagement and Independence, Volume and Stamina, Partner Work and Post-it Notes.

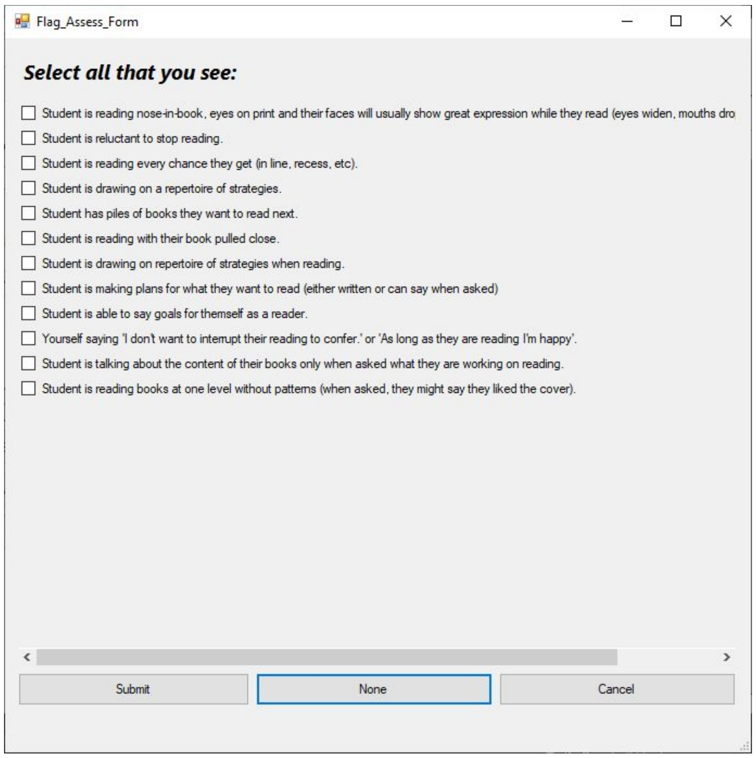
Engagement and Independence relates to how the student responds to reading in the classroom, whether for D.E.A.R. time or classroom assignments. Volume and Stamina relates to the student's reading logs, namely how much they are reading at home and for how long. For example, a student may read many different kinds of media but put them down very quickly after starting. Partner Work relates to how the student handles researching with a partner, or how they bounce literary ideas off of each other. Post-it Notes quite literally describes how the students are using post-its to help with their reading.

Based on these categories, the teacher can press the button in the lower-left box that best relates to what they are observing. Clicking on any button brings up an initial evaluation:



What type of behavior do you see when this student is reading?

Depending on your choice, the next menu will populate with checkboxes to evaluate the student's behavior:



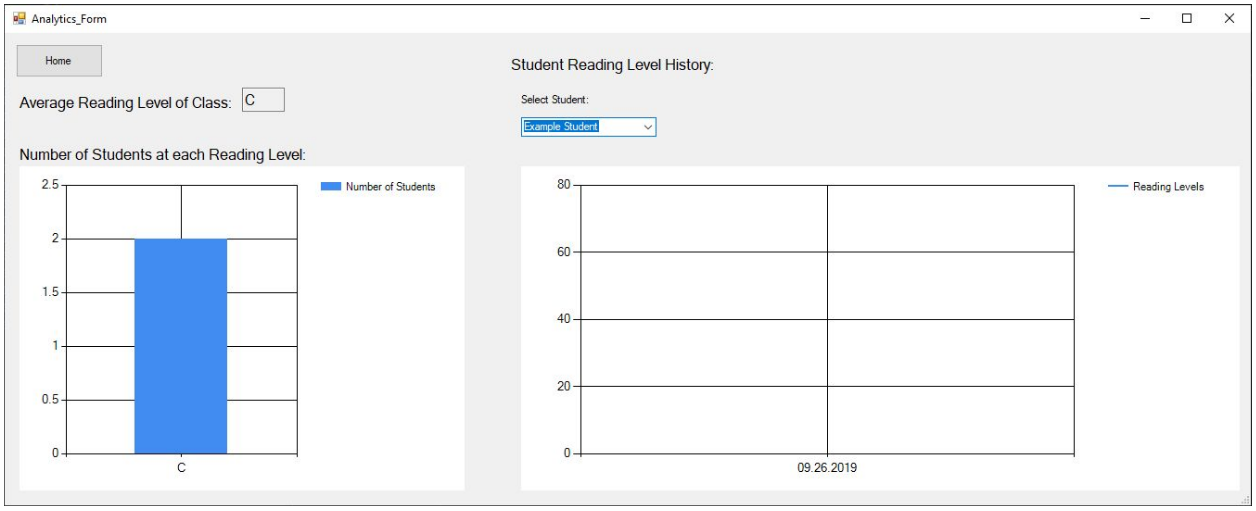
Select all that you see:

- ☐ Student is reading nose-in-book, eyes on print and their faces will usually show great expression while they read (eyes widen, mouths dro
- ☐ Student is reluctant to stop reading.
- ☐ Student is reading every chance they get (in line, recess, etc).
- ☐ Student is drawing on a repertoire of strategies.
- ☐ Student has piles of books they want to read next.
- ☐ Student is reading with their book pulled close.
- ☐ Student is drawing on repertoire of strategies when reading.
- ☐ Student is making plans for what they want to read (either written or can say when asked)
- ☐ Student is able to say goals for themself as a reader.
- ☐ Yourself saying 'I don't want to interrupt their reading to confer.' or 'As long as they are reading I'm happy'.
- ☐ Student is talking about the content of their books only when asked what they are working on reading.
- ☐ Student is reading books at one level without patterns (when asked, they might say they liked the cover).

Select all the checkboxes that you observe in the student, and click "Submit". The program will then assign a flag to the situation, and generate a pop-up with helpful tips on how to help the reader. The more flags you can handle, the better your readers will be.

Class Analytics

While the class analytics module is simple, this information is key to helping teachers assess their class as a whole. The module has several key pieces of data:



On the top left is the average reading level of the class. This is to ensure that your class in general is on par with the reading level expected of their grade.

The left chart is a bar graph that shows the number of students at each level. This can be used to identify the strengths and weaknesses of the classroom, and can be followed up through the Classroom Management panel to get specific details. The chart on the right is a line graph. First, select a student from the dropdown, and then the chart will be populated with their historical reading data. Note: because this is using the Example Student, the line chart only has one point currently.

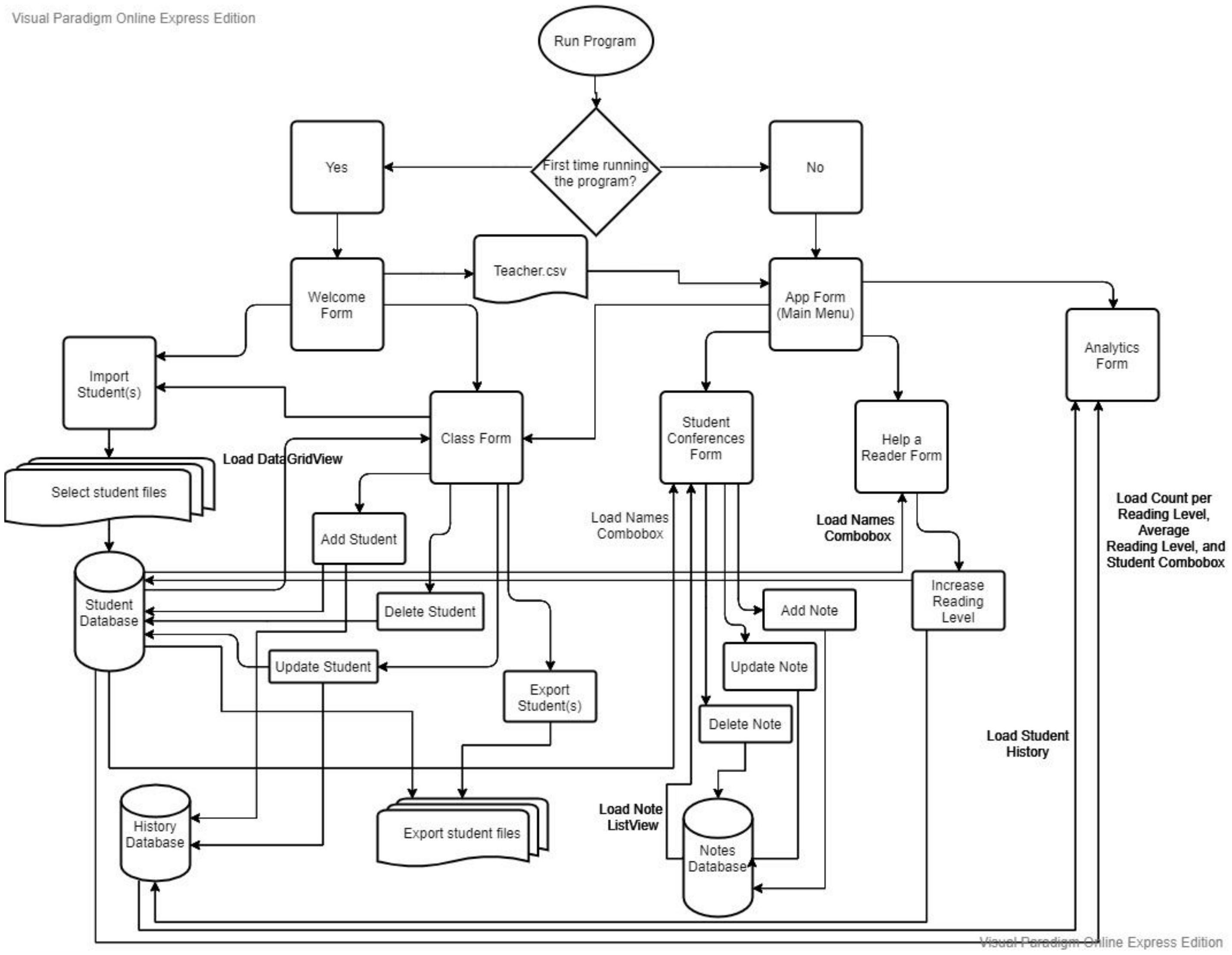
Design

The basic design for this program was to have a user-friendly experience, that was robust enough to be useful but simple and straightforward to understand. The first run of the program sets up a lot of information, and welcomes the user to the program by demonstrating a little of what it can do. The user is met with two options - start your classroom from scratch or import some student files. Both options will lead the user to the main menu, where the main interactions will take place.

Each module is designed to be simple, yet deliver a lot of information. The average teacher does not have a lot of time to be digging around for the information they want, so related information is displayed together. Each module also has a specific purpose, that is separate from the other modules. This prevents the screen from getting too cluttered with information that it is impossible to understand.

Overall, the design choices made were to keep the program focused on filling in a gap in the educational software market. This program is stored locally, both to ensure the security of the students, and to avoid server maintenance. This allows the application to be used with or without any future updates. Upon completion of this project, the application will be gifted to the teachers of Stonybrook School in Kinnelon, New Jersey for their personal use in the classroom.

The following diagram is a data flow diagram for the entire program:



This diagram shows the movement of data throughout the program. When the program is executed, if the user has not run the program before, then the Welcome Form will show. The Welcome Form will prompt the user, assuming he or she is a teacher, to enter their Name and Grade level. This is saved to a file and loaded into the main menu as a part of the dashboard. Implementing a profile system was a wish list item, but this is a simple way for the user to have some customization and feel like the program has some personality despite this feature not being fully completed.

From the Welcome Form, the user has the option of two buttons: importing their class data from csv files, or typing in the information from scratch. If the user chooses to import, the user is met with the HomeroomHelper desktop directory, and is asked to select which student they would like to import. Once they have selected these files, the import function parses each file line by line to get the student data, any notes that were included, and the historical data. It was important to keep all this information in one file, but because notes and history entries can both have n lines (as opposed to 1 like the student data), I included a tag to identify what kind of data the line represents. The function parses out the tag and stores the data in the appropriate table.

If the user chooses to create their class from scratch, they are brought directly to the Classroom Manager, which is the heart of this program. It controls the insertion, update, and deletion of student data by communicating with the Database Interface class, using it to send the appropriate queries. The user can add a student to the database via the textboxes on the form. I chose to use textboxes because there were only 6 fields to fill with student data, it made the information easy to verify and clean, and it resembled an internet sign-up form where each field is separate. When the user is interacting with the form, any student they have highlighted will be loaded into the textboxes so the user can edit them if desired, but they also make it really clear which student has been selected. The user can only interact with the database by entering data and clicking buttons; all of the commands have been obfuscated by functions for security and normalization purposes.

On the topic of the student data, one of the ongoing struggles of this project was how to define a student's id. At first, it seemed like the ideal solution would be to assign a student an integer starting at 1, then 2, then 3, etc. Most classrooms use this numbering system to identify a student quickly and easily out of 25 or 30 kids. However, using this numbering scheme proved to be quite challenging, and unnecessary. Keeping track of the count of students, needing to renumber the entire class if a student was deleted, and then having to change the student number on related notes and history entries for that student, it no longer made sense to keep that structure when a static, unique id was simpler and a database design standard. In the end, it will still work with the classroom's system because it is usually created based on the student's last name - if the teacher sorts by last name, the list will reflect what the numbering scheme would have been anyway.

Continuing with the database design choices, I chose to make the Database Interface static so it could be called from anywhere in the program, execute a specific function, and then return. The class itself does not have any reason to be instantiated more than once, as there are no member variables or properties to set. It simply encapsulates the database functionality in a simple format. Seeing how simple and elegant this solution was, I also chose to implement the IO class in the same way; it could be called from anywhere in the program without a declaration or instantiation, serve its purpose, and return. Figuring this out was one of the more difficult parts of gaining momentum on this project, as it took me quite a while to settle on this design.

Whether the user has run the program for the first time or not, eventually they will reach the main menu. This menu has two main sections: the dashboard and the actions. The dashboard shows some basic information about the class. The actions contains four buttons that will redirect the user to various forms: Classroom Manager, Student Conferences, Help a Reader, and Analytics. The purpose of formatting it like this was to keep the flow simple; you can get to almost every form from the main menu. The buttons are colorful and attractive, and the names of the modules are clear and concise.

Since we have already discussed the classroom manager, the next form is the Student Conferences form. The idea behind conferences is simple yet powerful - the teacher will sit down with the student and have a discussion, or intervene while the student is doing independent or group work. During this discussion, the teacher will take note of how the student is doing with their reading, and how they responded to the feedback received. As a bonus, this screen also includes reading conference information to help a teacher tackle even the most challenged reader. The conferences are tracked as “Notes”, both in the form and in the database. The database tracks when the conference took place, what the note is, and what type of category was chosen as the type. The view I chose to display the note information was a list view. It offered the most compact, yet descriptive way to display all the notes for a student. If I had to change one thing about the view, I would make it so it could sort by category (like the data grid view). I chose not to use a data grid view because I did not want to clutter the screen too much, and the list view looks very clean. But, sorting would add more functionality to the list that might have been useful in hindsight.

To display the information about conferences, I created a struct called Conference Types. Based on how the struct is defined, it will return the associated data for that type. So, paired with a combobox in the view, the user could select what kind of conference they wanted to learn about and it was easy to retrieve that data. I decided to “hardcode” this data in a struct instead of a file for a few reasons (this is also true for other “data” classes in the program). First, the information that I was using to complete this project was given to me from teachers in a paper format. I had to type pages of data to make it accessible, and it was simply not available in a digital format. In addition, I clarified with the teachers that this information has not changed in over 40 years, and was not likely to going forward. In most cases, I would shy away from using hard coded data in an app like this, but in this case it made more sense to integrate it into the heart of the program.

The same setup was used to create the Reading Levels struct for the Help a Reader form. I needed to be able to define what the level was, and what information was associated with it. A struct provided the perfect way to encapsulate this data easily. Depending on what reading level is chosen, the struct will return a list of the related information in a string array. Then in the form, it can easily be deconstructed to appear in the view.

While using structs worked well for conferences and reading levels, it did not work out well for the flag types. The information for flags are defined by two characteristics, color and category. Because there were crosses of each color and category, there were 20 possible entry results. In addition, the way that I would be passing information within the form would not support using just a struct, as I needed to be able to use a default constructor. So I used a class instead, but kept some of the struct-like features.

The Help a Reader form, mentioned above, has two primary functions: one, to educate the user about different reading levels to help them move their students along, and two, to quickly analyze a reading situation to help the teacher react in the best way using flags (not the coding kind). Displaying the reading levels was pretty simple; the user could select a reading level via a combobox, and then the associated information would load from the Reading Levels struct and display on screen. The flag analysis algorithm was much harder, and even the solution I ended up with is not entirely waterproof.

Flags are like reading emergencies. They can be good or bad, but generally include the teacher noticing some form of behavior and deciding how to act on it. Initially, I was going to make the flag analysis cycle through many times to specifically verify what the issue was and how to help the reader. The algorithm involved every category and color being evaluated, and recursively decided which solution to show once it was satisfied. While impressive, this algorithm was not practical in real-life use. When I proposed this idea to the teachers, they liked the idea of including the flag evaluation, but did not want to spend so much time working through the algorithm until it was satisfied. A quick and dirty solution was actually preferred, because it meant that the teacher did not have to spend more than a few seconds clicking on buttons before the algorithm would return the best fitting advice it could find.

So, on the form, I broke down the flags into their respective categories: Engagement and Independence, Volume and Stamina, Partner Work and Post-its. Each one has a button, and based on the situation the user would click on the most appropriate button. This would trigger the form to show a subform that asks in general, what kind of response the student has to reading - positive, neutral, or negative. The form records the user's response and returns, allowing the function to generate another form that will contain the observations of three categories: for positive, the top three flags are represented (blue, green, yellow), neutral the middle three (green, yellow, and orange) and negative the lower three (yellow, orange, red). A list of checkboxes are displayed, containing the text from the observation list. Each box is connected to a CheckedChanged event of their color. The user is asked to check off the boxes that they see for this student, and once they submit the algorithm does a simple max calculation to see which color got the most points. It then generates a message that contains suggestions on how to handle the situation in the classroom.

The Analytics form was designed as a proof-of-concept view that teachers could use to see the statistics of their class. To keep it simple, I included the average reading level of the class, how many students are at each reading level, and the progression of each student on a line graph. This module was one of the last ones that I had to implement, and I had decided to stick with Windows forms at the beginning of the project. Overall, the chart implementation for Windows forms is pretty unimpressive. I cannot use characters on the Y axis of a graph, which makes no sense considering that characters have an integer representation as well. While the charts were easy to use, they contained such rigid functionality that I found myself quite bored with a module that used to excite me at just the thought. Creating a better front-end is on my wish list for this project, and I would love to see what other frameworks have to offer charts. Not only would the entire project look better, but the data would be represented correctly, and there would be more motivation to add more data.

Finally, the Log class. It was very important to me to be able to log when hidden exceptions occurred, especially if I am assuming that a group of teachers are going to use this program and potentially break it; being able to have them access the logs and figure out where I went wrong was a requirement for me. I structured this class so I could invoke a Log object without assigning it, and it would record the exception, function title, and custom message. This way, it was simple to create a log and record the necessary details. I also decided to create a new log file each day. That is, if the program is run on a new day, the program will generate a log file for the current date. The logs are sleek, straightforward, and because I capture the exception I can extract more data to the log if I find it necessary.

File Structure

Above, we discussed the modules and some examples of design challenges and how they were overcome. Now, to briefly describe the file structure that is generated; essentially, the program creates a desktop folder called HomeroomHelper. It contains two subfolders, Settings and Logs. The Logs folder contains any logs generated by the program so they can be evaluated and help improve the program. The Settings folder contains two files: startup.txt and teacher.csv. The startup file is used at the beginning of the program to determine whether to run the welcome form or not. Teacher.csv contains the teacher data collected by the welcome form. Finally, the sqlite database is generated in the same folder the program is executed from.

Class List

Program - The main entry point for the application. This class' primary responsibility is controlling the initialization of the entire program.

Analytics_Form - This class is to support the functionality of the Analytics Form.

App_Form - The class supporting the main menu form.

Class_Form - The class supporting the classroom manager form.

Conference_Form - The class supporting the Student Conferences form.

Conference_Types - This structure is used to contain the different Student Conference types and information so they can be called in the program.

Database_Interface - This class is how the program connects to the database.

Flag_Assess_Form - The class that supports the Flag Assess form.

Flag_Initial_Form - The supporting class for the initial flag form.

Help_Reader_Form - The supporting class for the Help a Reader form.

History_Entry - A container to help history entries be passed around the program.

IO - The class responsible for importing and exporting student and teacher files.

Log - A logging class used to track any errors and exceptions that appear in the program.

New_Note_Form - The form for adding and editing notes, and getting conference information.

Note - A class to encapsulate all Note data.

Reading_Levels - A struct containing the reading level information.

Student - A class to encapsulate all the student data together.

Teacher - A class to keep teacher data together.

Welcome_Form - The supporting code for the Welcome form.

Program

Test Plan

The following are the tests done on the completed program. Unit tests were done along the way to ensure that each function worked as expected and unused code was not left in, while the tests below determine if the program as a whole reacts as expected.

The order that these tests were done were as follows: Main Menu, Classroom Manager, Student Conferences, Analytics, Welcome Form, and Help a Reader.

Welcome Form

Test	How it was evaluated	Result
Collect and verify teacher data	1. Run the program, enter the data and see if it generates the teacher.csv file correctly after creating the classroom 2. If the user does not enter any teacher data and tries to continue, the	Teacher.csv was loaded correctly in HomeroomHelper\Settings .

	program stops them from progressing until they enter all necessary data	
Import student data to start classroom	1. Dialog menu popped up and worked as expected. 2. Files imported correctly. 3. User was redirected to the Classroom Manager module 4. Cancelling the import returns you to the Welcome Form. 5. When done importing, the program opens a new Classroom Manager Form.	1-3 and 5-6 worked as expected. 4 - the program did not go back to the welcome form on the first try. The code was edited and now works as expected on the second try.

	6. When the Classroom Manager is closed, the Main Menu is opened instead of going back to the Welcome Form.	
Create a classroom from scratch	1. Navigate to the Classroom Manager Form successfully. 2. When the home button is clicked or the form is closed, the program redirects the user to the Main Menu.	1 and 2 work as expected.

Main Menu

Test	**How it was evaluated**	**Result**
Dashboard	1. Teacher name loaded in and displayed correctly in welcome statement. 2. Number of students in database queried correctly. 3. Average reading level of students queried correctly. 4. Teacher data can be edited.	1-3 worked as expected. 2 and 3 were verified in the classroom manager and using a SQLite Browser. 4 is not currently a feature due to lack of time, but it would be a great add-on as a next step.
Actions	1. Each button opened the associated form	Initially, the function used to open forms was not reliable. Creating a new form object and using the ShowDialog function has been more reliable thus far. Using ASP.NET or another front-end would be the logical next step.

Test	How it was evaluated	Result
Display student data from the database	1. Is the student data read only? 2. Can the data be easily viewed and understood?	1 is true, but even if a user was somehow able to change the property, the information would still refresh to be correct because it is being loaded from the database. If the data is edited using the proper channels, then the updated data appears. 2 is also true, as the display object used (DataGridView) provides automatic scroll bars and highlighted rows.
Adding a student to the database	1. Can the user easily enter data into the program?	1 - The textboxes have multiple purposes, but the user can clear them using the clear button.Then they

	2. Is the data normalized and clean? 3. Does the data get saved for future use? 4. Does the database check for duplicates?	type in their data and click "Save Student". 2 - The data is in a relational database and is normalized. The names get trimmed for spaces by the program, and the characters get trimmed and capitalized before they are entered into the database. This is evident by how they appear in the view. 3 - Yes, a local SQLite database is used to preserve data, and it appears in the program again even after it has been closed and executed again. 4 - Yes, the program will execute a pop-up that

		verifies the action you would like to take with this student.
Editing a student and updating the database	1. Can the student be selected by the user? 2. Is the new data replacing the old data?	1 - Yes, clicking on a row will select a student and display their information in the textboxes for editing. 2 - Yes, when the student is edited no new rows are added/deleted. It is a true UPDATE statement. However, the student's name is not updated.
Importing a student from a file	1. Can one or more students be imported? 2. Do the proper dialog/settings pop up? 3. Is the data parsed correctly?	1 - Yes. They currently exist as separate functions but you can import just one student or multiple. In the future, this feature would be streamlined. 2 - Yes, the dialog box appears and works as expected.

	4. Does the database check if this student already exists?	3 - At first, there was a bug where the characters were being stored as their integer values, which meant that nothing would import. After that, there was an issue importing unlimited notes and history entries, but this issue was solved by starting the line with a tag to identify what kind of data it was. After these issues were solved, importing worked as intended. 4- Yes! The database will ask the user what action they would like to pursue.
Exporting a student to a file	1. Can one or more students be exported?	1 - Yes. The user can select one student to be exported, or export the whole class as individual

	2. Is the user informed of where the export files are? 3. Are the files in a consistent format? 4. Can the files be re-imported back into the database?	files. This feature could be upgraded to be more flexible in the future. 2 - Yes, after exporting a pop-up appears with the path. 3 - Yes. Student data, note data, and history data are stored according to their field order in the database. 4 - Yes, exported files can immediately be re-imported.
Deleting one student	1. Does the program warn the user? 2. Does the program remove the student data from the database? 3. Does the program handle the edge	1 - Yes, a pop-up appears to verify that the user actually wants to delete the student. 2 - Yes, the student is completely removed from the database. There is no undo, unless the student has been previously

	case of all students being deleted?	exported. The view also refreshes to indicate the change. 3 - Yes. If the one student you are trying to delete is the last one in the database, the database will essentially start from scratch (explained in more detail below).
Deleting all students	1. Does the program warn the user? 2. Does the program remove all students?	1 - Yes. The program displays a pop-up to verify the user's action. 2 - The program removes all students that currently exist in the database, and then adds one "Example Student". The reason for this is that the database does not persist if it does not have at least one record, and other functions

		will fail if they detect that there are no students. It also is a nice way to show the user how the display is supposed to function.

Student Conferences

Test	How it was evaluated	Result
Viewing Notes	1. Can the notes be easily viewed? 2. Are the notes organized?	1 - The notes are in a List View, so when a student is selected the user can see all of the notes they have taken on this student. If the note extends beyond the window, the user can hover over it to view it as a tooltip. This feature could be improved, but works as-is for now. 2 - The notes are organized by date and tagged with a category. In the future, it would be better if they could be sorted by any column, similar to the DataGridView control.

Add New Note	1. Is the new note form easy to understand? 2. Does the program provide more information on what student conferences are? 3. Is the note saved in a retrievable format?	1 - Yes, it is easy to understand where to type notes and where to select a conference category. 2 - The drop-down menu contains types of student conferences, and upon selection will provide details for each kind to assist the user. 3 - Yes, all notes are saved in a database and can be viewed on the main conference window.
Edit Selected Note	1. Is the note to-be-edited retrieved in the window? 2. Is the note properly updated?	1 - The note data was, but the combobox did not populate. 2 - Yes, the note is updated in the database and this change is reflected in the view.

Delete Selected Note	1. Is the user warned before deleting a note? 2. Is the note removed from the database?	1 - Due to the nature of student conferences, there was not a warning implemented for notes. They should be simple to add and delete, so the user does not get frustrated if they want to delete several notes at once. 2 - Yes, the note is entirely removed from the database. This is reflected in the view.

Help a Reader

Test	How it was evaluated	Result
Student View	1. Are all students in the selection? 2. Is the student reading level displayed correctly? 3. Does the “Level Up” button work as expected?	1 - Yes, the students’ first and last names are loaded from the database into the combobox. 2 - Yes, the student’s id is found and their reading level is displayed in the box accordingly. 3 - Yes. The button will increase the student’s level by 1, or will do nothing if they are already at the maximum level.
Reading Flags	1. Can the user select a reading category? 2. Is the initial assessment clear? 3. Are the checkboxes easy to understand?	1 - Yes, the reading categories are broken into four groups: Engagement and Independence, Volume and Stamina, Partner Work and Post-it Notes.

	4. Does the algorithm produce a reasonable answer and corresponding information?	Depending on which part the user is evaluating, the user may select the appropriate category. 2, 3 - Yes, the initial flag assessment and the checkboxes are generally easy to understand. Some checkboxes may seem duplicated - this is due to using the same terminology for two or more flag colors. 4 - Yes. The algorithm takes the greatest number of checkboxes associated with a flag color, and shows helpful suggestions for the teacher based on the flag color they see.
Reading Level Information	1. Are all the reading levels represented?	1 - All reading levels A-Z are represented in the

	2. Is the information easy to read and apply?	combobox, but not all levels have corresponding information based on the documentation received. Therefore, levels such as A, B, and E do not display data. 2 - Because of the breakdown by level, the information is easier to digest and is helpful to apply with the student level box right next to this information.

Analytics

Test	How it was evaluated	Result
Reading Level Average	1. Is the correct average reading level displayed?	1 - Yes, the correct reading level is displayed.
Reading Level Counts	1. Are the counts correct? 2. Is the information easy to read?	1 - Yes, the counts are correct. This was verified in the Classroom Manager and an SQLite Browser. 2 - The data is easy to understand, although the bar graph does not display the reading levels in character order, which can be confusing if you aren't paying attention.
Student Progression	1. Is the information correct? 2. Is the graph easy to read?	1 - This information is correct. Each time a student is updated, their history is updated as well.

		All of that historical data is transferred onto the graph to show their improvement over time. 2 - Unfortunately, due to a limitation with Windows Forms, the line graph will not display characters. For now, the graph will use numbers to represent the general improvement, but in the future they will be changed to characters.

Known Bugs

Line graphs for students do not display reading levels as characters (Y-axis). This is due to restrictions on the chart object. To fix, a new UI would need to be used.

Teacher data is not editable after the initial entry.

Number of students is not always accurate on the Dashboard.

Flag checkmarks are duplicated. This is because they appear in more than one flag category. Workaround: if applicable, select both checks.

If flag checkboxes are selected more than once, that color will have an unfair advantage in the calculations. Workaround: if a mistake is made, restart the flag process.

Summary

I learned a lot through the process of completing this project. I had not used any form of GIT very extensively, so it was good to learn that process and see for myself why it is useful, and how to use it most effectively. I admit that the way I used it in my project was not the best way to use it, but I think I have gained a better handle on how to use it in the future. At work, especially, I feel that I would be able to use GIT effectively with my peers. I also learned to just keep things simple, and to not overcomplicate things for the sake of overcomplication. If something seems that easy, sometimes that just means that it is; making it more complicated may not make it more useful.

The idea for this project came about through connections. Roughly half of my family are teachers, and I have always valued education as a means of ongoing learning and as a profession. So, in the spirit of wanting to create something useful for teachers, I asked teachers that I knew personally what kind of program they would want if it could do anything for them in the classroom. A day after this request was posted, several 4th grade teachers banded together and provided me with the specifications and related information specified above. I spent a lot of time going over their needs, and communicating with them to make sure the design was to their liking. This taught me the importance of communicating with the client, and verifying details ahead of time to ensure the final product is what they want.

While this program could not be distributed on a global scale due to the use of the Teacher's College resources without their formal consent, it does highlight the lack of genuinely useful tools in the education field. Through this project, I hope to have gained a better understanding of both my programming knowledge, but also what the education field is lacking from the technology field, and how we can help each other come up with great solutions to everyday problems.

Bibliography

"Count the Characters Individually in a String Using C#." StackOverflow, 31 May 2012,

stackoverflow.com/questions/10830228/count-the-characters-individually-in-a-s tring-using-c-sharp. Accessed 17 Sept. 2019.

"C# Programming Guide." Microsoft .NET, Microsoft, 1 May 2017,

docs.microsoft.com/en-us/dotnet/csharp/programming-guide/. Accessed 20 Sept. 2019.

"C# Windows Form Creating Checkboxes." Code Project, 7 Oct. 2013,

www.codeproject.com/Questions/650309/Csharp-Windows-Forms-Creating-Che ckboxes. Accessed 20 Sept. 2019.

"How Do I Create a Message Box with 'Yes', 'No' Choices and a DialogResult?"

StackOverflow, 14 June 2010, stackoverflow.com/questions/3036829/how-do-i-create-a-message-box-with-yes -no-choices-and-a-dialogresult. Accessed 25 Aug. 2019.

Sharma, Anoop Kumar. "Chart Control in Windows Forms Application." C# Corner, 8 Mar.

2015, www.c-sharpcorner.com/UploadFile/1e050f/chart-control-in-windows-form-appli cation/. Accessed 18 Sept. 2019.

"SQL as Understood by SQLite." SQLite, www.sqlite.org/lang.html. Accessed 20 Sept.

2019.

Github: https://github.com/ka18m/senior-project