Criterion C: Development

Overview:
The development section of this internal assessment will go through the different libraries used to create the product, the database filing system, object oriented programming, memory management and the graphical user interface of the product.

Libraries:
There are 5 different large libraries used to create this software named Node JS, Electron JS, React JS, Less CSS, and Babel. Out of the 5, 3 libraries were used for the software architecture of the product, and 2 were used for post processing languages in order for me to better utilize JavaScript and CSS.

Node JS is an open source javascript library used for creating servers and the backend of most web applications that is built from the chrome V8 engine (Node.js). Node JS can be used through the terminal and run different commands such as creating projects and running them. This technology was required to use the electron js framework.

Electron JS is a cross platform application framework that allows javascript to be used to develop an application in either windows, macOS or linux (Electron). This technology combines the chromium and node js engine and was developed by Github. This framework is used in this internal assessment as the application structure for the project.

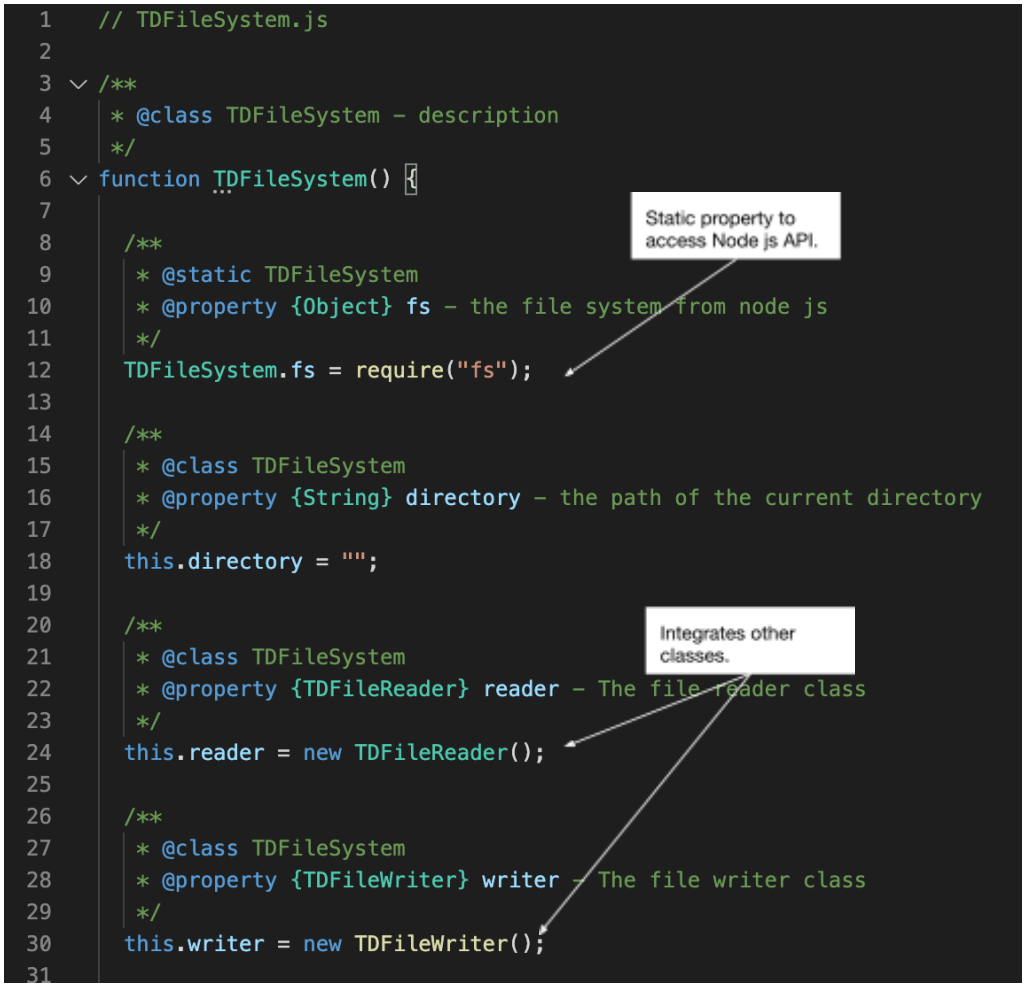
React JS is a front-end object oriented web framework developed at FaceBook used for simplifying building user interfaces in applications or websites (React). React JS possesses both client side and server side rendering capabilities but for my application I chose client side rendering. Client side rendering provided me with a greater control of manipulating the application user interface rather than having the server constantly rewriting the UI.

Less CSS is a post processing style sheet language that allows someone to develop the style of a web application efficiently and then compiles it to CSS through javascript (Getting started | Less.js). I chose Less CSS because it allowed me to more efficiently write my style sheets for my application.

Babel is an open source javascript compiler that allows typescript or JSX to be transpiled into javascript so it can run on the application (Babel). I used this library to simplify the process of creating the user interface in an xml format rather than a functional format.

Database JSON & ID System:

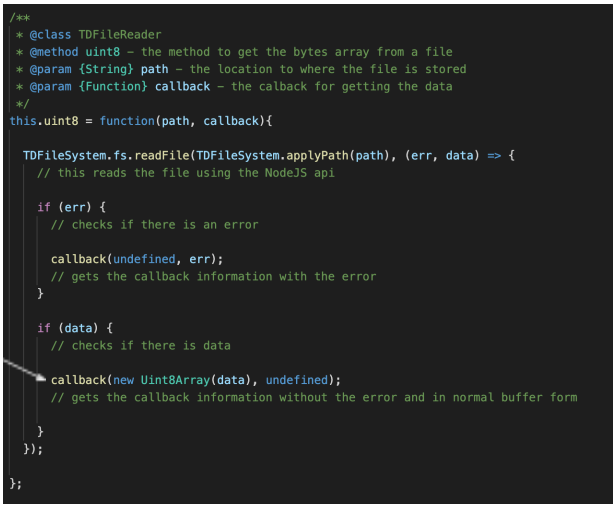
The database and JSON id system of the internal assessment is responsible for showing how the product writes and modifies data in the user's directory. This section also explains how pointers are used to identify the different school classes used throughout the entire system of the internal assessment.




```
1   // TDFileSystem.js
2
3   /**
4    * @class TDFileSystem - description
5    */
6   function TDFileSystem() {
7
8     /**
9      * @static TDFileSystem
10     * @property {Object} fs - the file system from node js
11     */
12    TDFileSystem.fs = require("fs");
13
14    /**
15     * @class TDFileSystem
16     * @property {String} directory - the path of the current directory
17     */
18    this.directory = "";
19
20    /**
21     * @class TDFileSystem
22     * @property {TDFileReader} reader - The file reader class
23     */
24    this.reader = new TDFileReader();
25
26    /**
27     * @class TDFileSystem
28     * @property {TDFileWriter} writer - The file writer class
29     */
30    this.writer = new TDFileWriter();
31
```

Screenshot 1: TDFileSystem.js file screenshot

The screenshot above displays the file system class and demonstrates how it's integrated with the other file based classes. The static property "fs" of the class TDFileSystem accesses the native Node JS file system api to use the premade functions offered by the library. The TDFileSystem class has two properties for reading and writing, that way if a programmer has a File system variable, they don't need to create another reading or writing one.



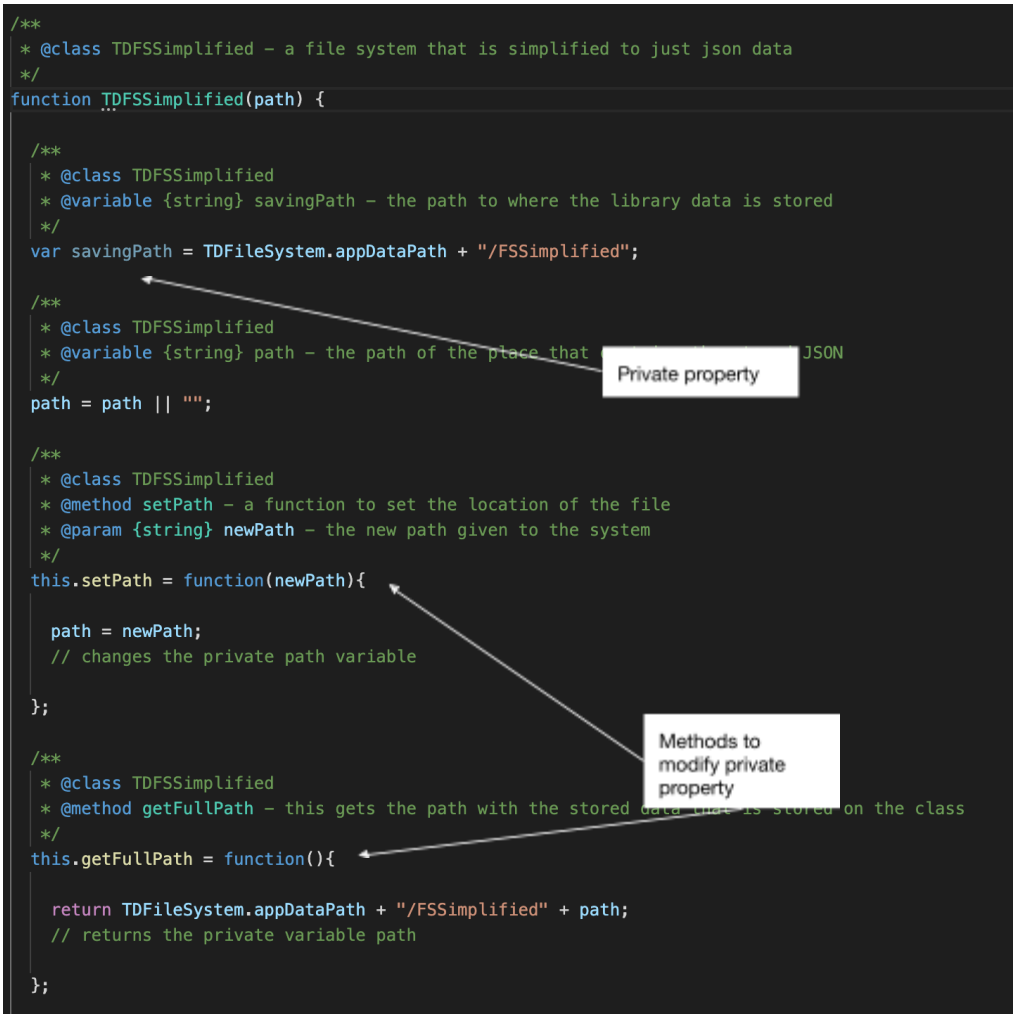
Screenshot 2: TDFileReader.js text method *Screenshot 3: TDFileReader.js uint8 method*

The two screenshots above demonstrates how the filereader class (TDFileReader) manages error handling through the node js file api and converts the buffer data from the callback into Unicode Transformation Format 8 bit text. This class also supports base64 file reading used primarily for image support and Unsigned Integer 8 bit data as shown in the screenshot 3.

```
new Uint8Array(Buffer.from(data.replace("data:;base64,", ""), "base64"))
```

Screenshot 4: TDFileWriter.js base64 write method with data

The focus of this screenshot can show how the system also removes the metadata of any base64 data when writing the files.



Screenshot 5: TDFSSimplified.js file

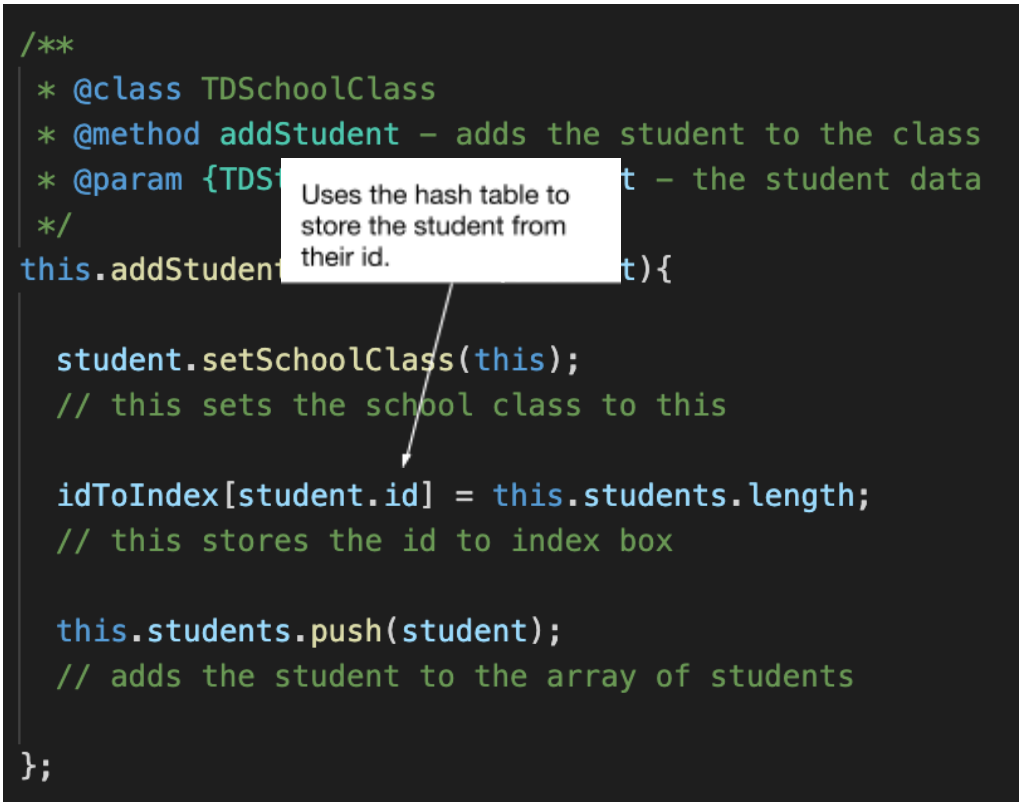
This area presents the FSSimplified class which handles reading and writing to the database in JavaScript Object Notation format. This is a simplified version of the file system class (TDFSSimplfied) that utilizes the file reading and writing classes (TDFileReader & TDFileWriter) to create and save JSON files to the database based on a fixed path. The class also uses native API to find the application support path that is stored in a private property named savingPath.

Screenshot 6: TDFSSimplified.js the load method

To retrieve data, it uses the file reader class (TDFileReader) to load the file based on the database path with a callback system to receive the data. It then converts the text from the database into an object that can be used with the code while also providing error data if it's present. The class also uses a queue to cache callbacks if the application currently hasn't received the file system path.

Screenshot 7: TDUserData.js static idToIndex

This static property displays a hash table (or dictionary) that contains all the ids of School Classes in the database. This hash table uses pointers to direct the application to the different school classes from the database.



Screenshot 8: TDUserData.js static idToIndex

In the method where the school class adds a student to the class. It uses the idToIndex hash table that contains the id and converts it to the index of the student class in the linked list named students. The id system is also stored in hexadecimal and converts the number into base 16.

```
/**
 * @static TDSchoolClass
 * @property {number} ids - the counter for all the ids that are stored for the class
 */
TDSchoolClass.ids = 0;
```

Screenshot 9: TDSchoolClass.js static ids

This section right here defines the next id that can be defined based on the highest number count. This was used so that the application doesn't accidentally use the same id twice and uses a new one instead.

Object Oriented Programming:

Object oriented programming is the process in which developers structure their software architecture using a class based approach ("Object-oriented programming - Learn web development | MDN"). This way of programming makes it easier for humans to code because it allows people to group the data that they need to mix together.

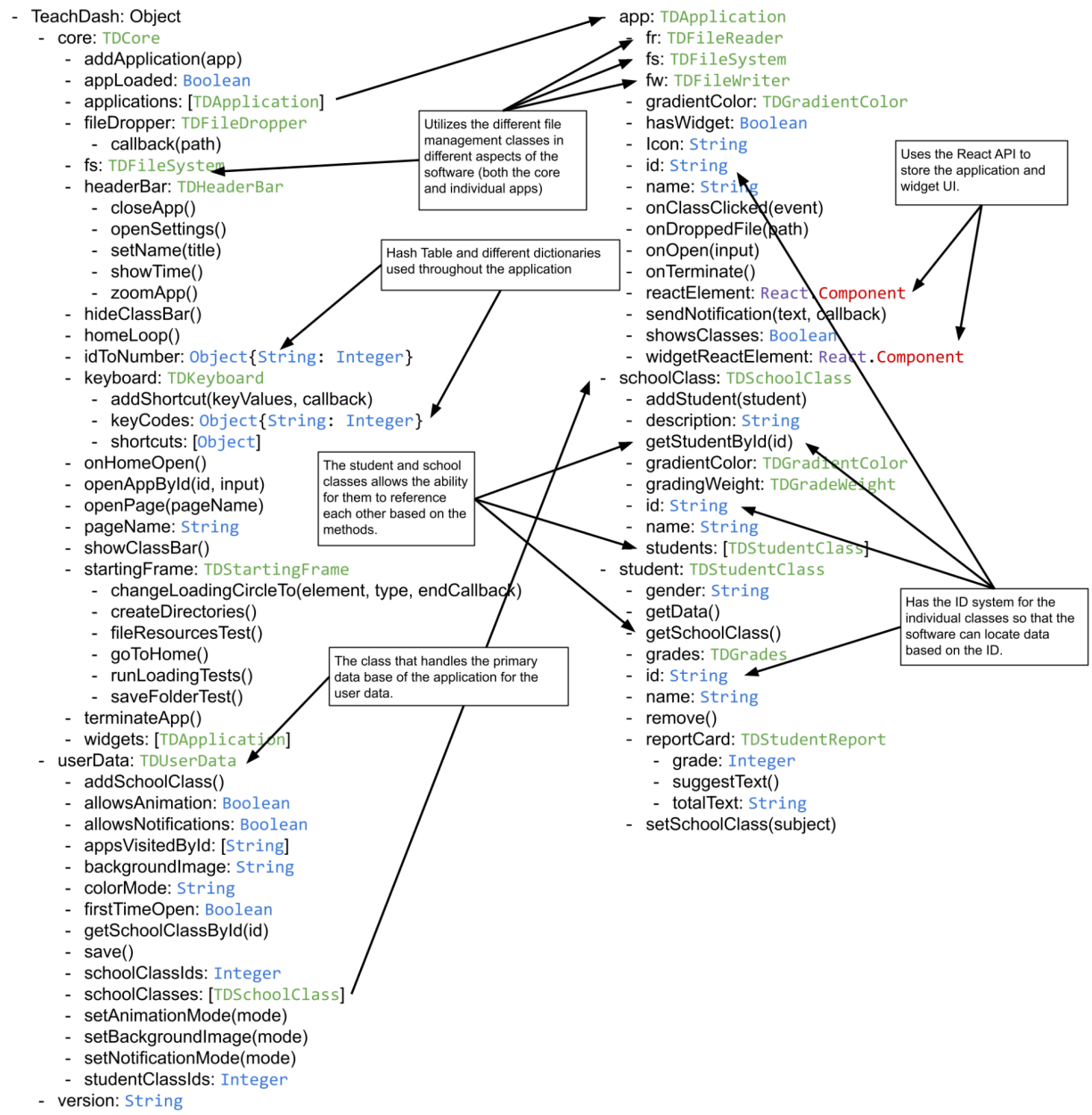
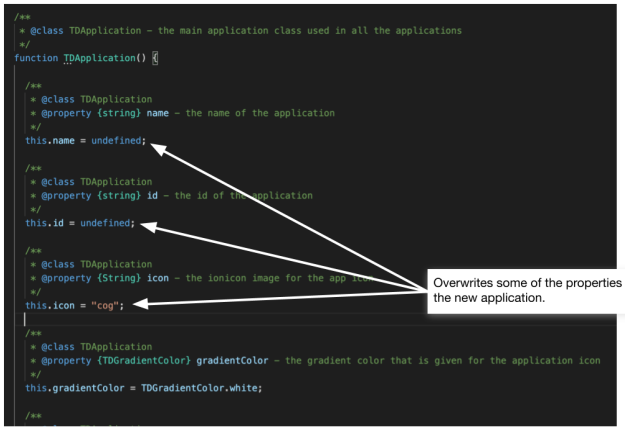


Figure 1: All the methods and variables in hierarchical structure

The image above displays all the ways the different classes are connected together with annotations to why they are connected with each other. From this diagram it can be observed how the software relies on other classes and relies on the fundamentals of object oriented programming.



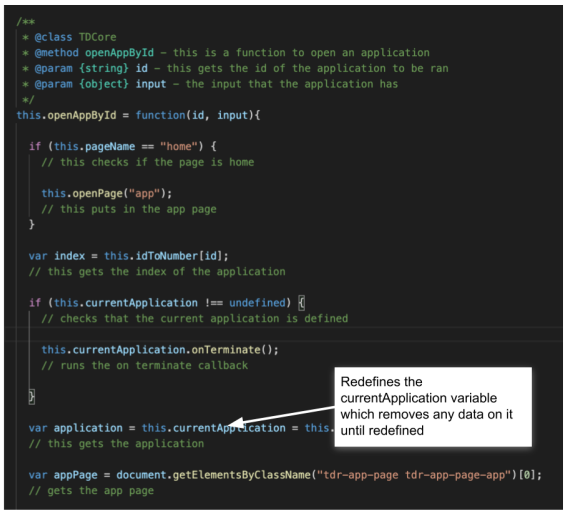
Screenshot 10: TDApplication.js class

Screenshot 11: ClassEditApp.js class

These two screenshots demonstrate how inheritance is used throughout the product where the TDApplications uses multiple different methods and properties in order for them to be reused. It has the default application properties such as id, icon and name that can be overwritten by new classes. Inheritance is a very prevalent aspect of object oriented programming in which it allows developers to reuse classes without rewriting them.

Modular Programming (Stacks):

Modular programming is a software development technique used to have all the different aspects of the program running independently from each other ("Modular Approach in Programming"). This allows the code to be able to easily add, remove or modify different sections without it altering the entire system.



Screenshot 12: TDCore.js addApplication method

Screenshot 13: TDCore.js openAppById

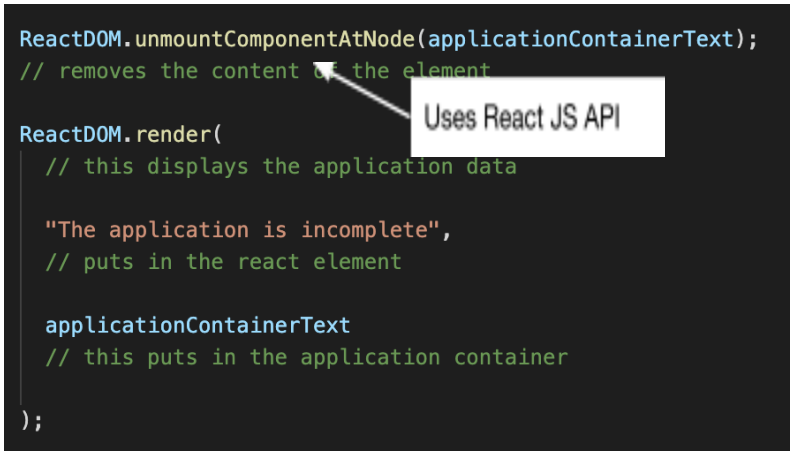
In the product, the sub-applications are added through this modular programming based approach with a linked list, and it's id stored in the hash table. The screenshot above shows the method for adding different sub-applications and where it is stored in the runtime of the application.

Memory Management:
Memory management is a very important aspect of software development because it is usually the biggest limitation to most users and needs to be properly managed in order for the application to run efficiently and smoothly.

In this portion of the application, the code presented displays how it allocates memory for the GUI by destroying and recreating unused UI using the React framework. Most of the sections of the applications are initially rendered with empty containers and all the content is rendered after the user input.

Screenshot 14: tdr-container.babel TDRContainer component

This shows all the main pages that are in the software but as previously mentioned, they are all empty to save on memory.



Screenshot 15: TDCore.js showing rendering invalid application

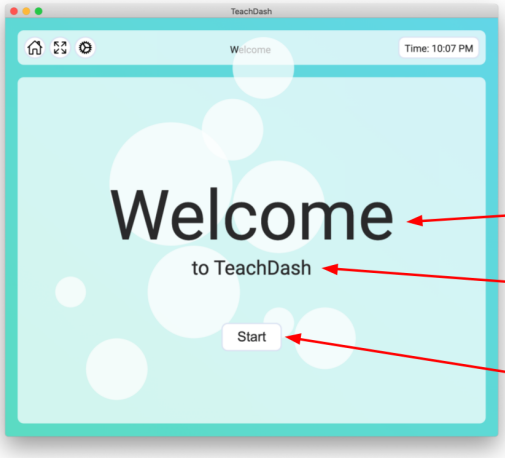
This example of code shows the software's error handling system where it creates an empty page if it notices that the application is incomplete while empties out the UI. When opening different frames of the product, it clears out all the unused pages by removing elements inside them.

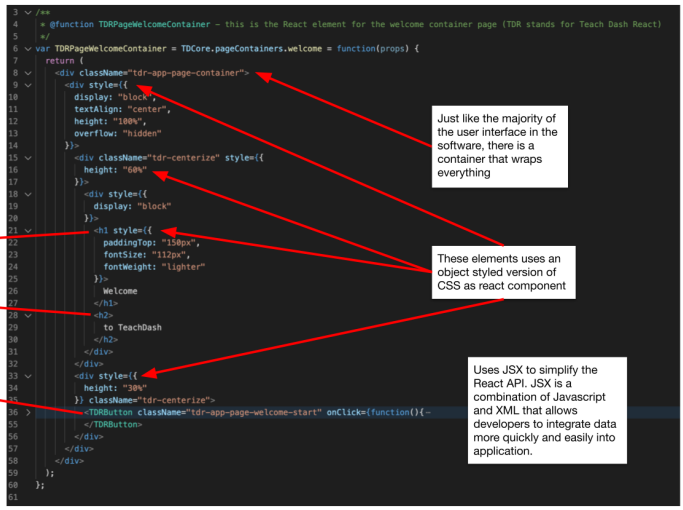
Screenshot 16: tdr-page-app.babel component *Screenshot 17: tdr-page-app-container.babel component*

This React Component shows the content of TDRPageApp when it gets filled up. This component is separated from the TDRPageApp component so that the software can efficiently add and destroy the element so that the user interface doesn't take a lot of space in the primary memory.

Graphical User Interface:

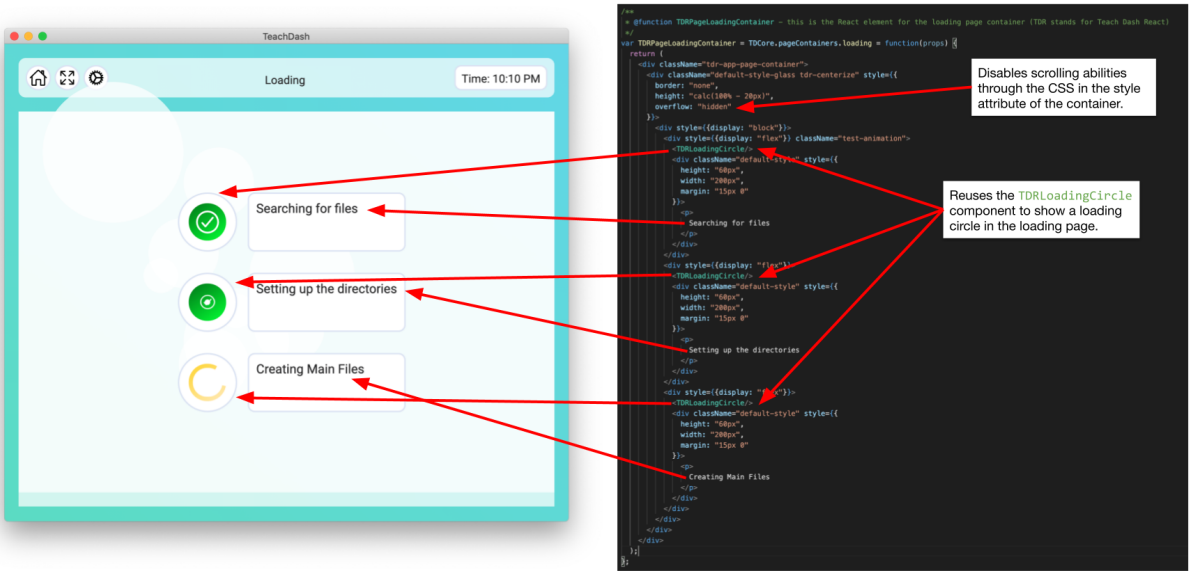
As previously mentioned, the GUI section of the entire product is created with React API and changes variably depending on the user input. The screenshots below will generally have arrows pointing directly to their respective rendered components such as buttons frames or text.





Screenshot 18: TeachDash welcome screen *Screenshot 19: welcome screen JSX*

The image and code provides an example of how the welcome screen was created using the react API and the direct correlation to the elements and the code. This demonstrates where babel was used on top of javascript so that it could simplify the React API and use an XML JavaScript format.

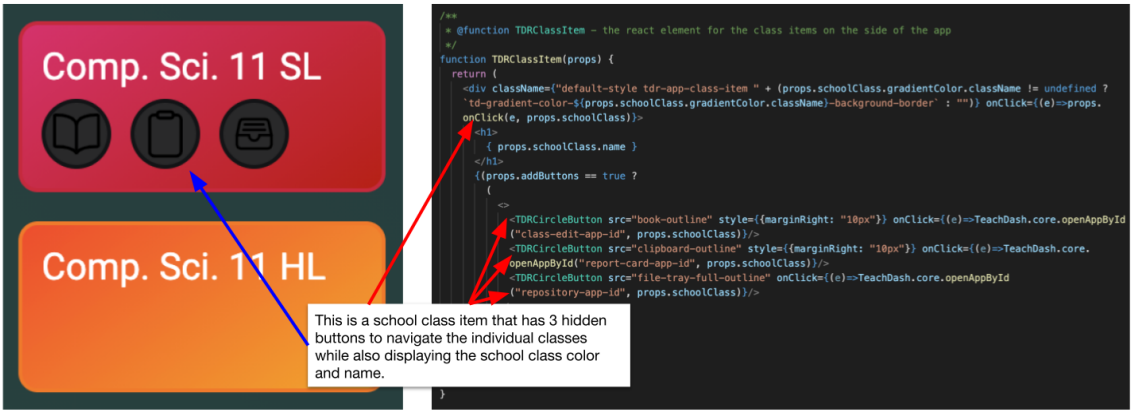


Screenshot 20: TeachDash loading screen *Screenshot 21: TDRPageLoadingContainer Component*

This section displays the loading screen with reference to the JSX in the application to create the user interface that the user is able to see. The JSX code also displays the styling of individual components in an object format rather than CSS which allowed me to have greater control on the style when needed.

Screenshot 22: TeachDash home layout *Screenshot 23: TDRPageHomeContainer.babel*

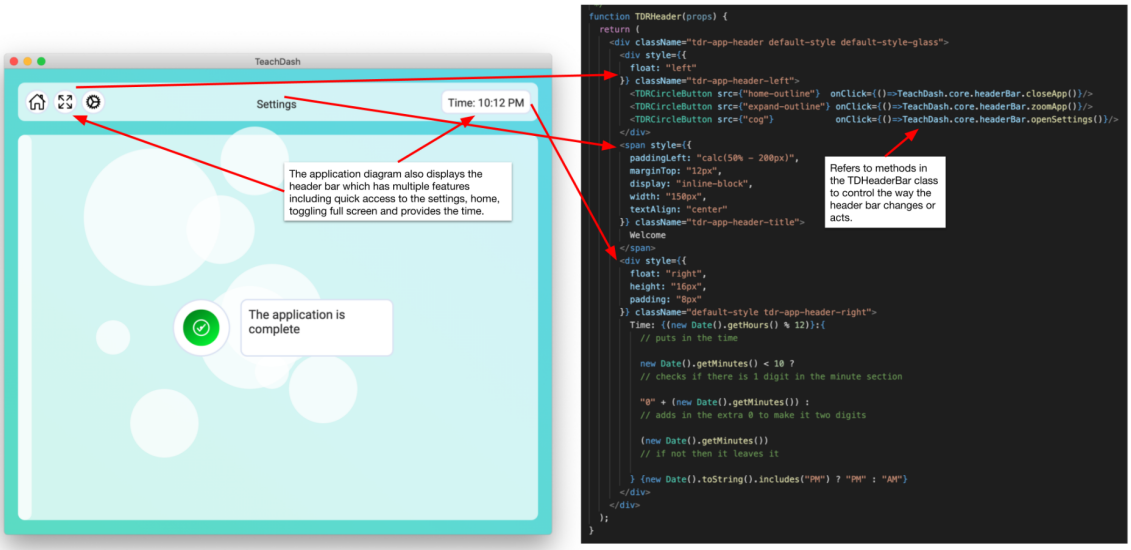
This presents the different sections of the home page with different React components that were created in order to create the software. The home page has multiple containers that can be filled up by other chunks of data. It also uses other components such as TDRAppLauncher and TDRClassItem to create the software.



Screenshot 24: TeachDash dark classes hovering

Screenshot 25: TDRClassItem Component

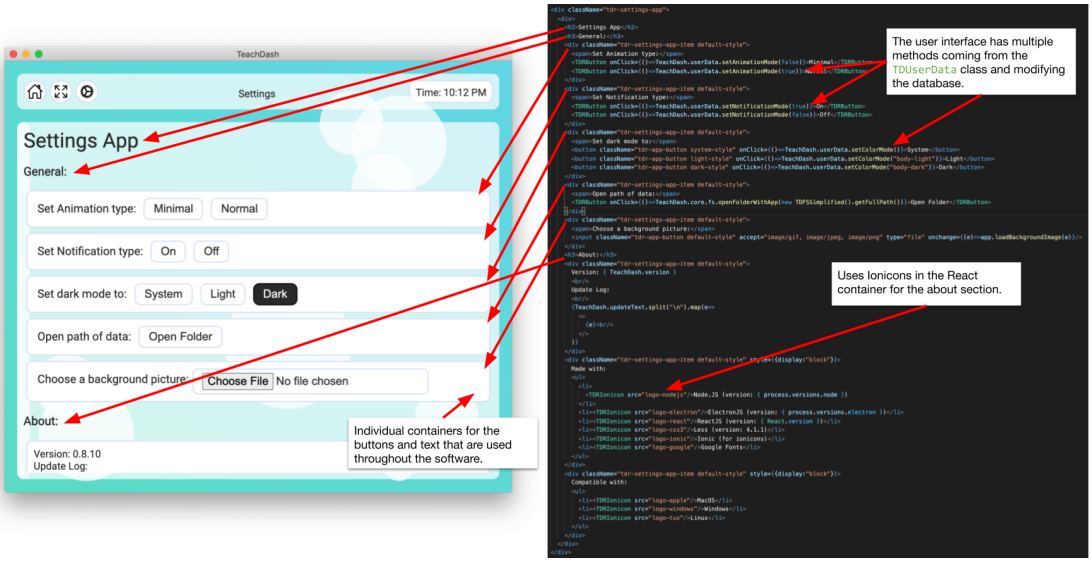
This screenshot shows the code for the classes and how it displays depending on whether the user is hovering on the buttons or not.



Screenshot 26: TeachDash settings app preload

Screenshot 27: TDRHeader Component

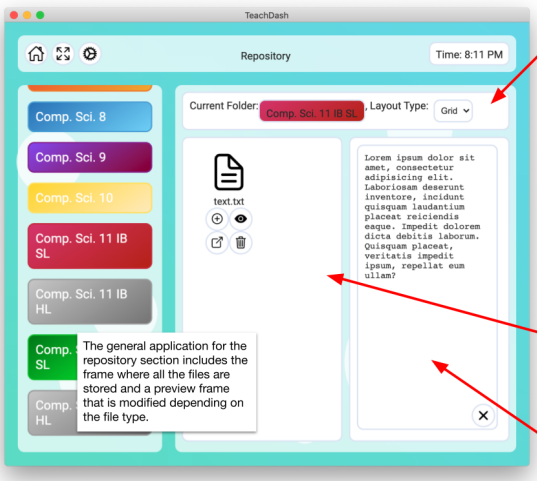
The diagram presents the code of a sub-application with a general explanation of the header bar. This primarily shows the different pieces of the header being used and its corresponding code from the JSX presented. There is both the primary and secondary container in the application page to modify the user interface easily.

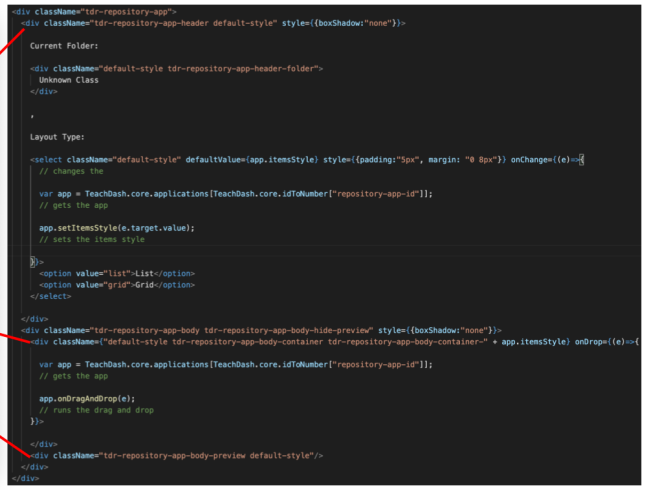


Screenshot 28: TeachDash Settings App

Screenshot 29: TDRSettingsAppContainer.babel

The diagram above compares the JSX code for the user interface of the settings pages of the software. The code also shows how the application modifies the userdata and the methods of that class to store the data.

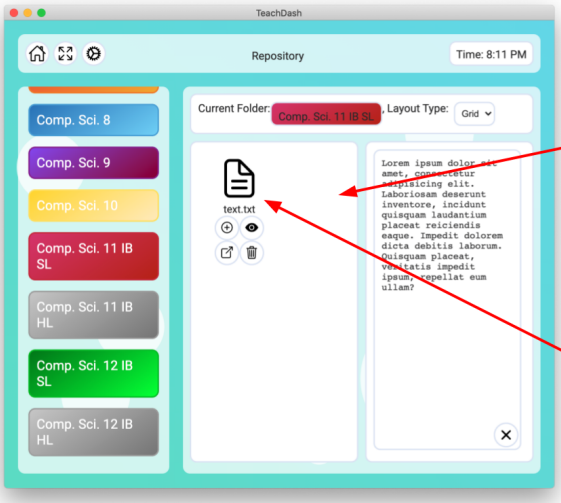


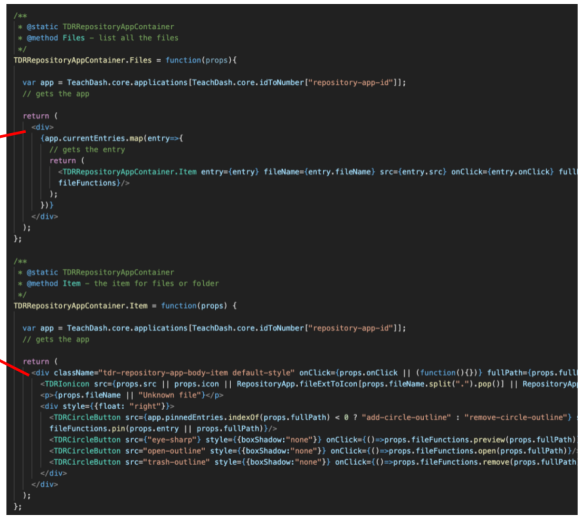


Screenshot 30: TeachDash Repository App

Screenshot 31: TDRRepositoryAppContainer.babel

This is the diagram of the repository sub-application that displays both the user interface components and code that manages the sub-application. Each button within the files in the body of the repository, is from the application class RepositoryApp that has all the methods for modifying the files. The main frame has a drag and drop file that will be stored in a folder that is controlled by the file and electron api. This process transfers the original file to the location in the database. In the preview section there are 3 file types which include images, text and an invalid one. For the text files, there is the onchange callback which writes the file based on it's path.

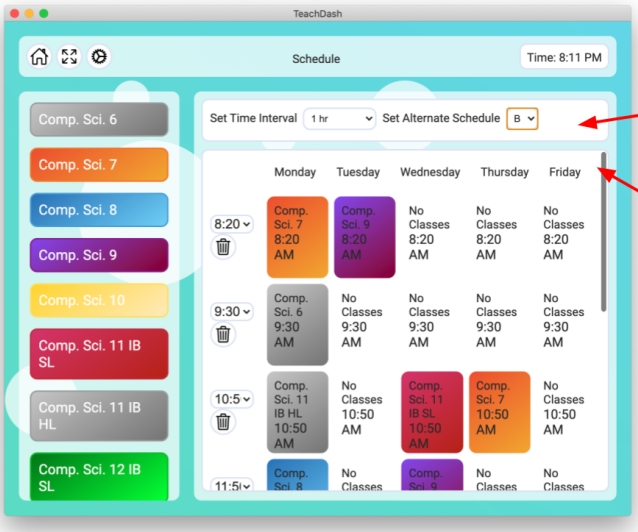


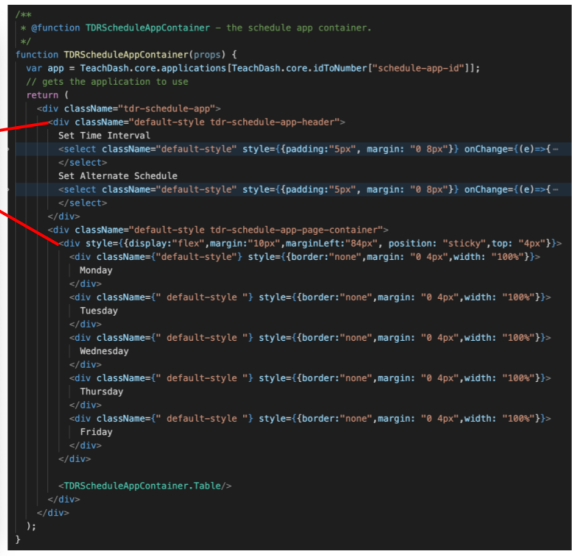


Screenshot 32: TeachDash Repository App

Screenshot 33: Repository Files Items Components

This primarily displays the individual file item of the repository application with the different buttons such as preview, delete and open. The JSX code uses the methods that are stored in the RepositoryApp class in order for the files to function properly.

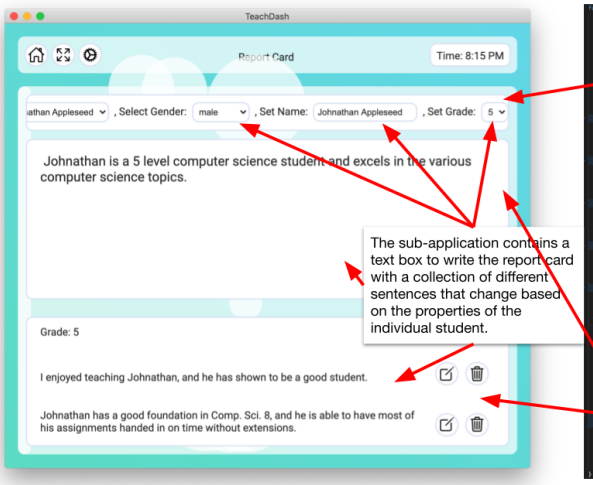


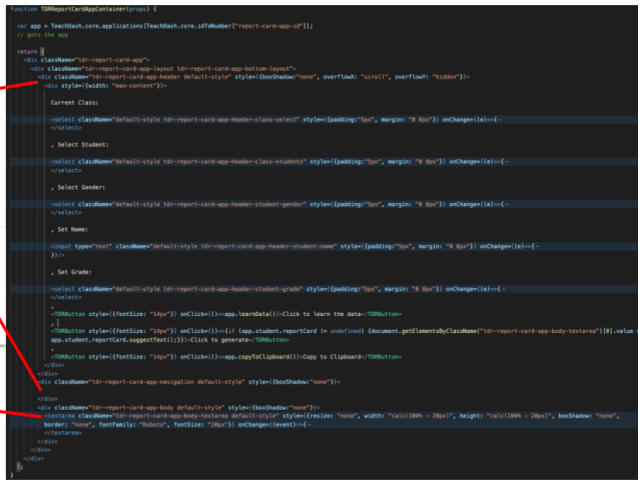


Screenshot 34: TeachDash Schedule App

Screenshot 35: TDRScheduleAppContainer.babel

This diagram displays the different React components of the sub-application for the scheduling system with the matrix data structure that stores the schedule information. The sub-application has to be re-rendered once changes are done by clearing the component and reading it. The app schedule system uses a matrix that stores the individual class ids to get the school class of the table item. This user interface component shows the individual table item that changes color and text based on the school class input it was provided with.

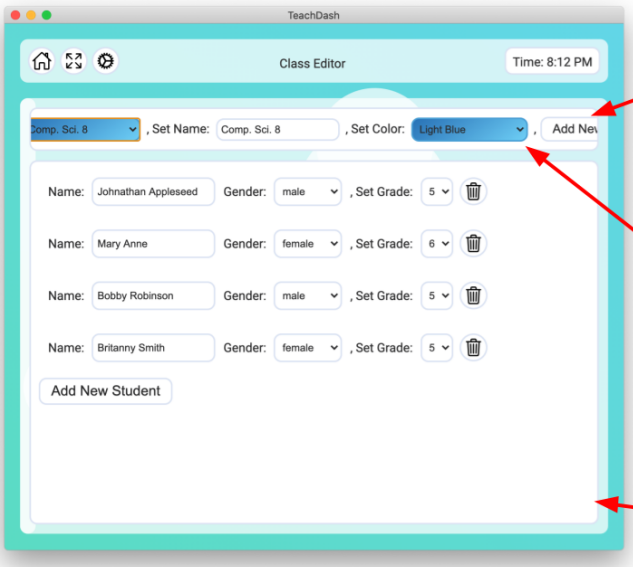


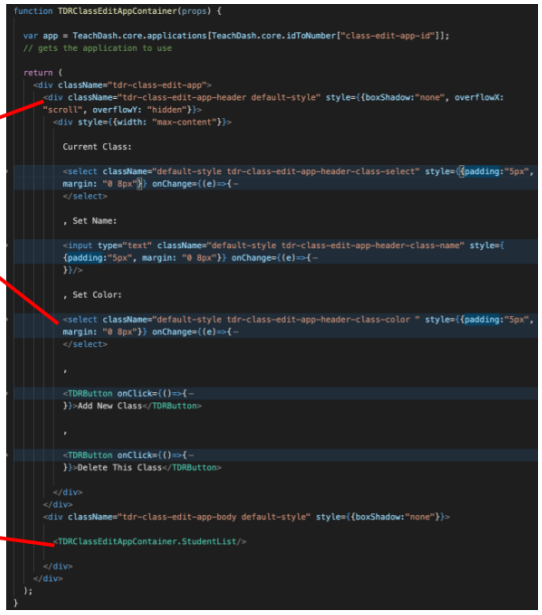


Screenshot 36: TeachDash Report Card App

Screenshot 37: TDRReportCardAppContainer.babel

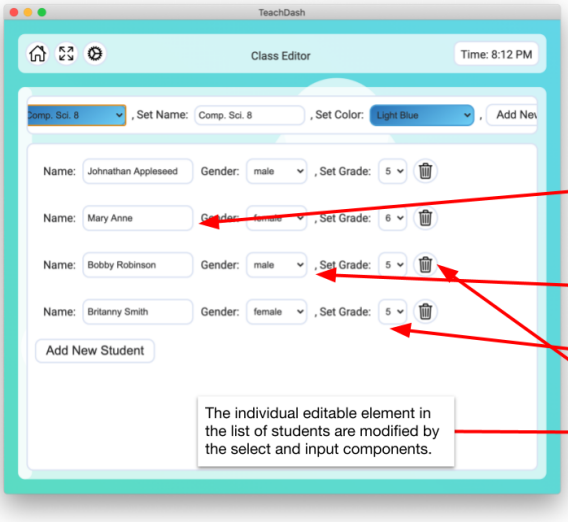
This portion of the diagram shows the Report Card sub-application with the UI and the machine learning code it uses to analyze user's sentence data from the text box. This like the previous screenshots demonstrates the comparison between the JSX and the application output.

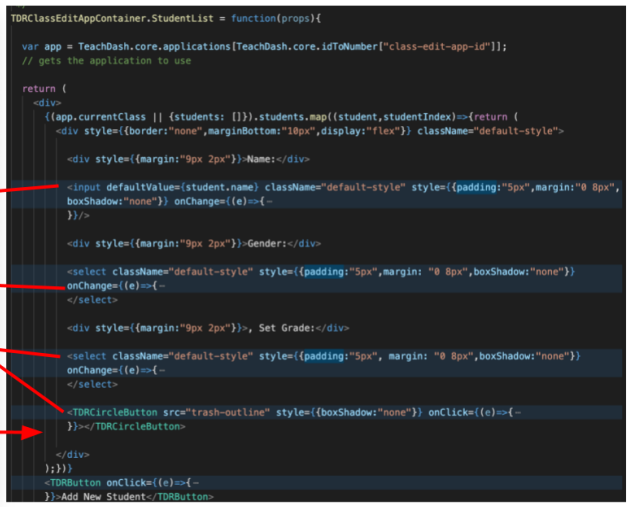




Screenshot 38: TeachDash Class Editor App *Screenshot 39: TDRClassEditAppContainer.babel*

The diagrams above show the class editor application and how it is able to modify the database. The buttons on the header of the sub-application allow the user to add and remove students from the array that is stored in the database.





Screenshot 40: TeachDash Class Editor App *Screenshot 41: Student list component*

These screenshots show the different input methods for the class editor on the different students such as the name, gender and name. Once the software detects an input, it will instantly save the data without the user needing to manually input.

Machine Learning:

In this product, the software uses the input from the text editor in the report card application to learn new sentences by identifying different words such as names, and pronouns to store it in the database in order for it to be reused by the user

```
TDStudentReport.learnFromSentence = function(sentence, student, schoolClass){

  schoolClass = student.getSchoolClass() || schoolClass;
  // gets the student class if it is defined

  sentence = sentence.split(schoolClass.name).join(schoolClass.name.split(" ").join("_")).split(" ");
  // this splits the sentences for all the words

  for (var wordCount in sentence) {
    // loops through all the words of the sentence

    var word = sentence[wordCount];
    // gets the current word

    var wordLeftOver = word.replace(word.replace(/[,.]+/g, ""), "");
    // gets the rest of the specialized word (example: "$OPRONOUN,.," returns ",.,.")

    var wordWithoutLeftOver = word.replace(wordLeftOver, "");
    // gets the word without the left over (example: "$OPRONOUN,.," returns "$OPRONOUN")

    for (var pronounId in TDStudentReport.pronounData) {
      // loops through all possible pronound ids


      if (TDStudentReport.pronounData[pronounId].includes(wordWithoutLeftOver.toLowerCase())) {
        // checks if the current word belongs to a certain id

        word = pronounId + wordLeftOver;
        // replaces it with the pronoun id

        break;
        // ends the loop

      }

    }

    if (wordWithoutLeftOver == student.name.split(" ")[0]) {
      // checks if the name is requested

      word = "$NAME" + wordLeftOver;
      // gets the first name of the student

    }

    if (wordWithoutLeftOver == schoolClass.name.split(" ").join("_")) {
      // checks if the name is requested

      word = "$CLASSNAME" + wordLeftOver;
      // gets the first name of the student

    }

    sentence[wordCount] = word;
    // replaces the previous

  }

  return sentence.join(" ");
  // returns the sentence

};
```

Screenshot 42: TDStudentReport.js static method for learning

This section is displaying the machine learning aspect of the sub-application where it learns from the user's sentences in the text box by identifying the class name, student's name and gender and then stores the sentence in the database so it can be reused. It separates the individual words and searches the word bank and the information it was given in the parameters to extract the names and then stores it depending on the grade. It also uses nested loops to compare the words with the pronouns. The code also uses Regex to remove any extra punctuation so that it doesn't interfere with the data.

Works Cited:

Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS., https://www.electronjs.org/. Accessed 20 February 2022.

Node.js, https://nodejs.org/. Accessed 20 February 2022.

React – A JavaScript library for building user interfaces, https://reactjs.org/. Accessed 13 March 2022.

Babel · The compiler for next generation JavaScript, https://babeljs.io/. Accessed 13 March 2022.

Getting started | Less.js, https://lesscss.org/. Accessed 13 March 2022.

"Modular Approach in Programming." *GeeksforGeeks*, 7 September 2018, https://www.geeksforgeeks.org/modular-approach-in-programming/. Accessed 22 February 2022.

"Object-oriented programming - Learn web development | MDN." *MDN Web Docs*, 3 February 2022, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_programming. Accessed 13 March 2022.