Final Project Report

Joshua Silveous

CSET 4250: Comparative Programming Languages

Prof. Scott Brahaney

1 Table of Contents

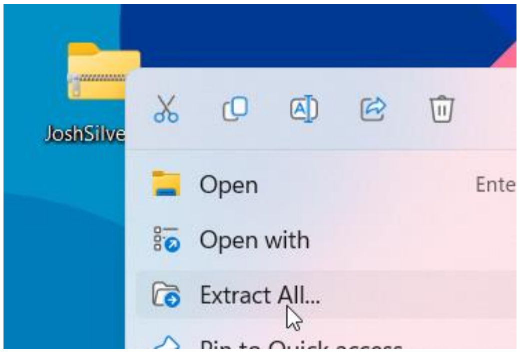
2 User Manual

a. Installation

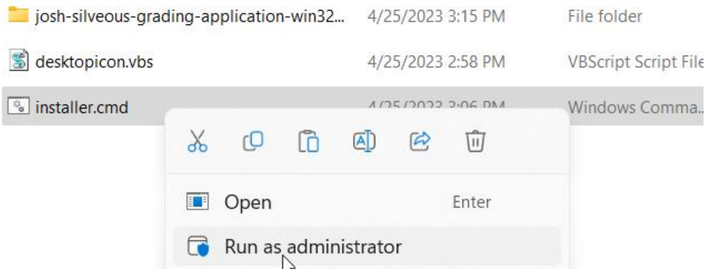
The file downloaded by the user is *JoshSilveousFinal.zip*. This zip file contains an installer, which will copy necessary files over and create a desktop shortcut for *all* users on the computer.

<u>Install Instructions</u>

1. Right-click on the downloaded *.ZIP* file, and choose *Extract All*. Choose *Extract* on the next menu.

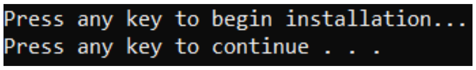


2. This will create a folder containing a few folders and files, including *installer.cmd*. Right-click this installer, and run it with an **Administrator account (required)**.



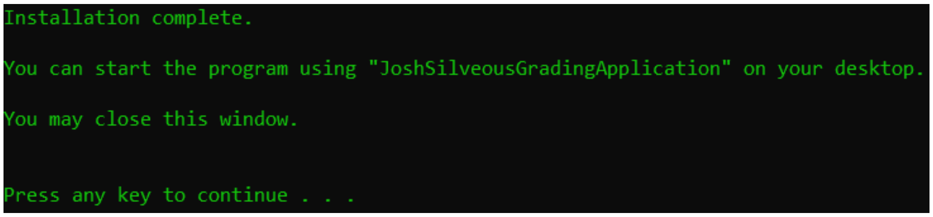
josh-silveous-grading-application-win32...	4/25/2023 3:15 PM	File folder
desktopicon.vbs	4/25/2023 2:58 PM	VBScript Script File
installer.cmd	4/25/2023 3:06 PM	Windows Comma...

3. The installer will validate privilege and files, and when ready, prompt the user to begin installation. Press any key to start the installation process.



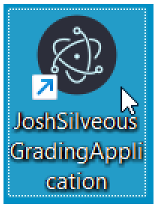
```
Press any key to begin installation...
Press any key to continue . . .
```

4. The installer will notify you after successful completion, and prompt you to close the installer.



```
Installation complete.

You can start the program using "JoshSilveousGradingApplication" on your desktop.

You may close this window.


Press any key to continue . . .
```

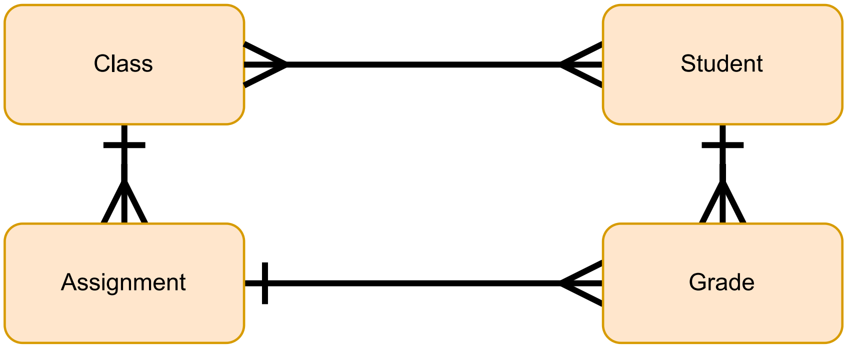
5. The desktop icon can be used to launch the program.



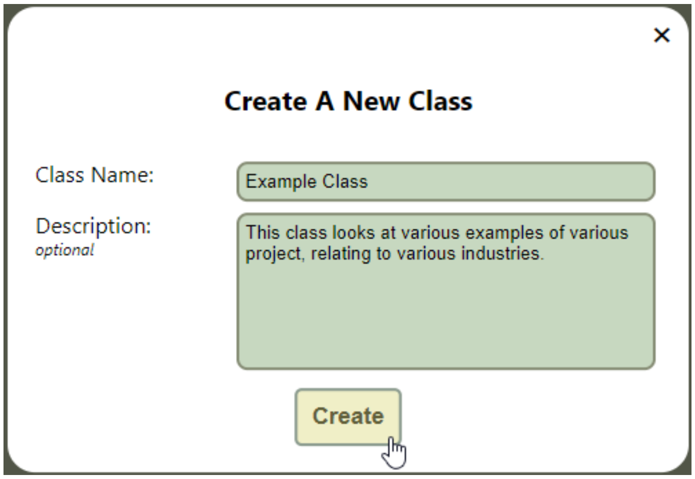
b. Usage

This application allows the user to create multiple classes, and multiple students.
Each class can have multiple students, and each student can be in multiple classes.
Each class can also have multiple assignments, and each student in a given class will have a grade entry associated for each assignment.

Illustration of the relationships between data



When you first launch the application, you will be greeted with a popup to create your first Class in the application. Fill this out as desired, and choose *Create* when ready.



Create A New Class

Class Name: Example Class

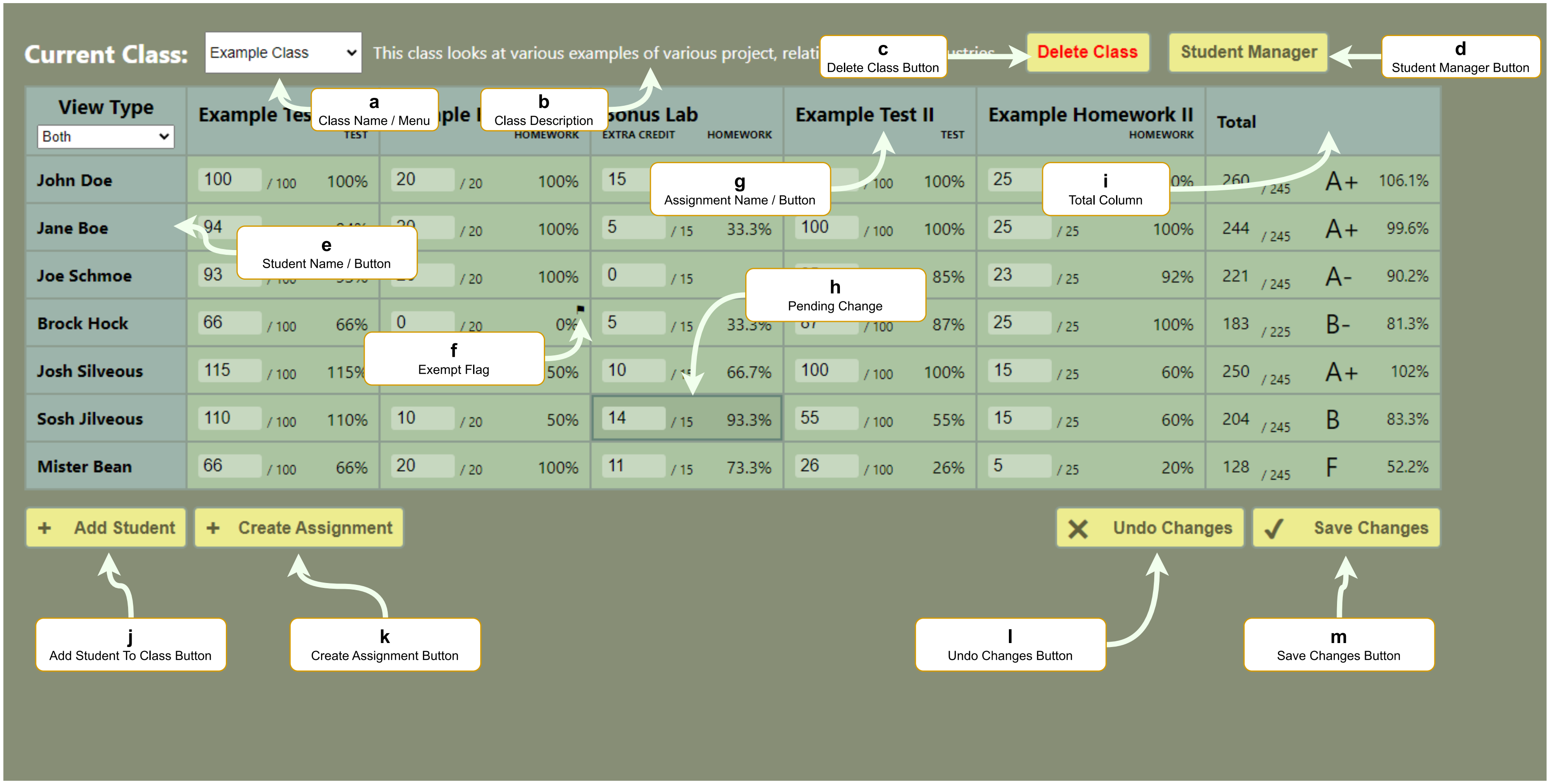
Description: This class looks at various examples of various project, relating to various industries.

Afterwards, the class's data will be displayed. Since there is no pre-existing data, you must add students (or assignments) to begin viewing the table.

i. Navigation Overview

a	Class Name / Menu	Displays the class name, also used to switch between multiple classes. There is also an option to create a new class.
b	Class Description	The description of the class, as set by the user.
c	Delete Class Button	Used to delete the current class (confirmation needed).
d	Student Manager Button	A menu used to manage all students in the database, including adding, deleting, and renaming.
e	Student Name / Button	Displays the student's name, clicking will reveal a popup to unenroll or edit the student.
f	Exempt Flag	Signals that this grade entry is exempt from the student's final grade- it can boost their grade but cannot harm it.
g	Assignment Name / Button	Displays the assignment's name and attributes, clicking will reveal a popup to edit or delete the assignment.
h	Pending Change	Signals that a change has been made to this grade entry, which has not yet been saved to the database.
i	Total Column	Shows the final grades for each student after all calculations (exempt and extra credit grades) are performed.
j	Add Student To Class Button	Used to enroll an existing student into the class (or create a new student).
k	Create Assignment Button	Used to create a new assignment for the class.
l	Undo Changes Button	Reverts all pending changes to their original state.
m	Save Changes Button	Saves all pending changes to the database.

Refer to the graphic on the next page.



Current Class: Example Class ⌄ This class looks at various examples of various project, relati ustries

Delete Class

Student Manager

View Type Both ⌄	Example Tes TEST	ple I HOMEWORK	onus Lab EXTRA CREDIT HOMEWORK	Example Test II TEST	Example Homework II HOMEWORK	Total
John Doe	100 / 100 100%	20 / 20 100%	15 [illegible]	[illegible] / 100 100%	25 [illegible] 0%	260 / 245 A+ 106.1%
Jane Boe	94 [illegible]	20 / 20 100%	5 / 15 33.3%	100 / 100 100%	25 / 25 100%	244 / 245 A+ 99.6%
Joe Schmoe	93 [illegible]	[illegible] / 20 100%	0 / 15 [illegible]	[illegible] 85%	23 / 25 92%	221 / 245 A- 90.2%
Brock Hock	66 / 100 66%	0 / 20 0%	5 / 15 33.3%	[illegible] / 100 87%	25 / 25 100%	183 / 225 B- 81.3%
Josh Silveous	115 / 100 115%	[illegible] 50%	10 / 15 66.7%	100 / 100 100%	15 / 25 60%	250 / 245 A+ 102%
Sosh Jilveous	110 / 100 110%	10 / 20 50%	14 / 15 93.3%	55 / 100 55%	15 / 25 60%	204 / 245 B 83.3%
Mister Bean	66 / 100 66%	20 / 20 100%	11 / 15 73.3%	26 / 100 26%	5 / 25 20%	128 / 245 F 52.2%

+ Add Student

+ Create Assignment

✕ Undo Changes

✓ Save Changes

ii. Creating a Student

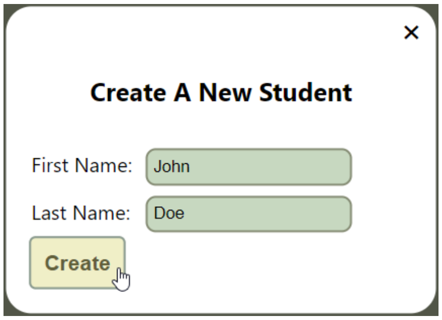
There are two different ways to create a student.

Creating student via *Student Manager*

In this application, Students aren't bound to their classes. A single student entry can be used in multiple classes.

If you'd like to create a student **without assigning them to a class**, you can do so in the **Student Manager** (accessible via the *Student Manager* button in the top-right).

Here, you can choose *Create New Student*, then fill out the student's information and choose *Create*.

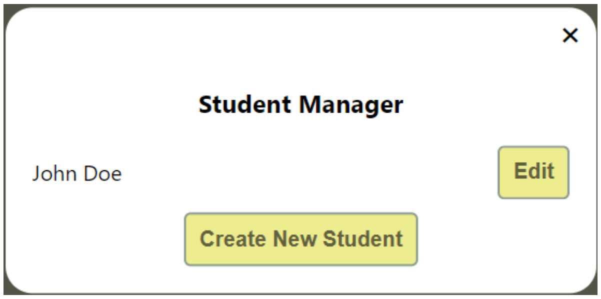


Create A New Student

First Name: John

Last Name: Doe

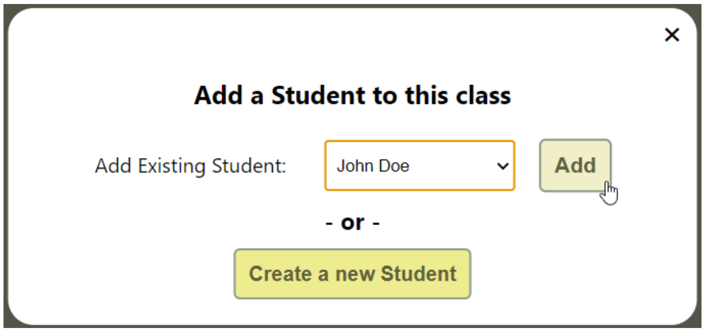
Now, the student will appear in the *Student Manager.*



Student Manager

John Doe

This student currently isn't enrolled in any classes. To enroll them in a class, you can choose the *Add Student* button on the bottom-left corner of the class's menu, and pick their name using the dropdown menu and choosing *Add.*



Add a Student to this class

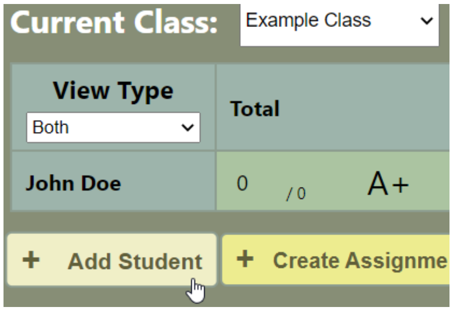
Add Existing Student: John Doe

The new student will now appear in the class's table.

Creating student *directly in a class*

To make things quicker, you can also create a student inside of a class, which will automatically enroll them in that class.

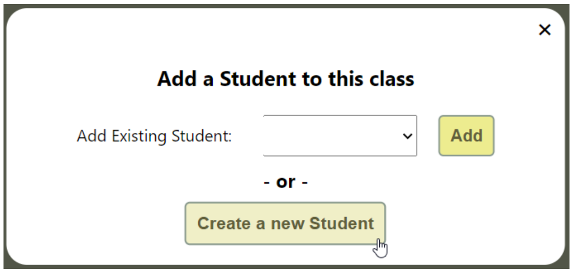
First, with the proper class selected, choose *Add Student* in the bottom-left.



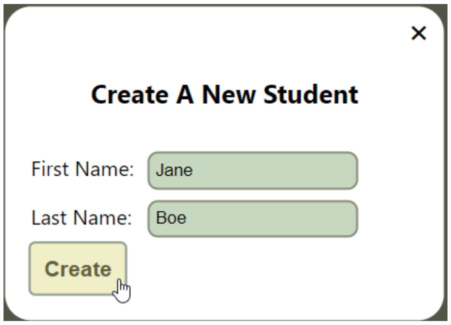
Current Class: Example Class

View Type Both	Total
John Doe	0 / 0 A+

Then, select *Create a new Student*.



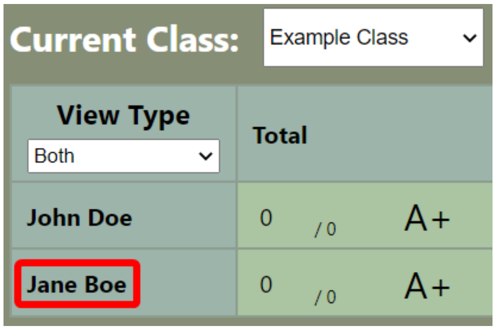
Here, you can fill out the student's name and choose *Create*.



First Name: Jane

Last Name: Boe

The student's record will be created, and they will automatically be enrolled in your current class.

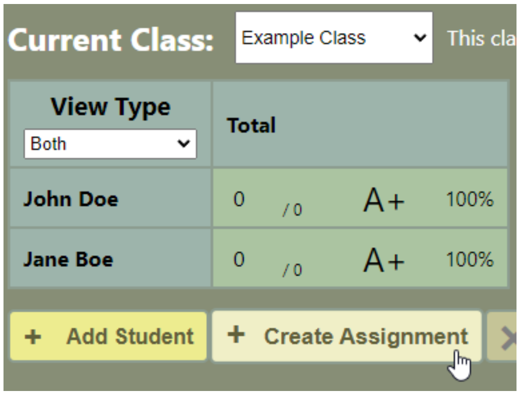


Current Class: Example Class

View Type Both	Total
John Doe	0 / 0 A+
Jane Boe	0 / 0 A+

iii. Creating an Assignment

You can create a new assignment by choosing the “Create Assignment” button at the bottom of the page.

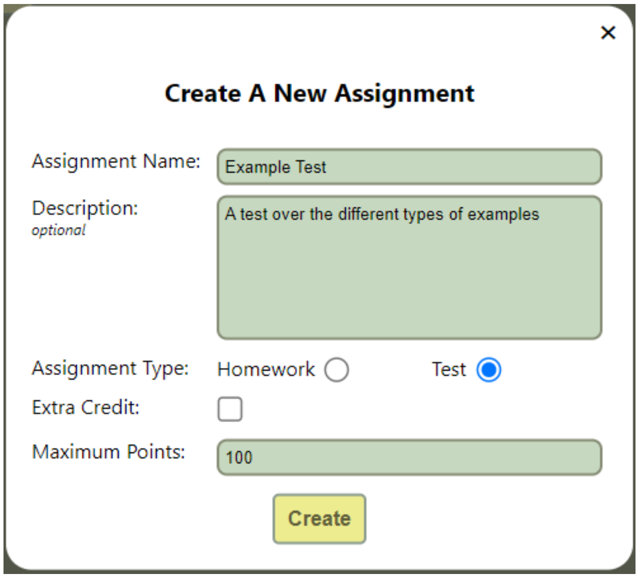


Current Class: Example Class

This cla

View Type Both	Total			
John Doe	0 / 0	A+	100%
Jane Boe	0 / 0	A+	100%

This will bring up the *Create A New Assignment* menu, which has many options you can fill out.



Create A New Assignment

Assignment Name: Example Test

Description: *optional* A test over the different types of examples

Assignment Type: Homework ○ Test ●

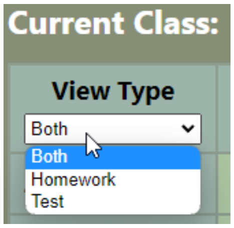
Extra Credit: □

Maximum Points: 100

Assignment Name: Required, the display name of the assignment.

Description: Optional, shows in the table when hovering over the assignment’s column.

Assignment Type: Required, allows you to organize based on type of assignment. The class’s table can be filtered to display only one type or the other using the *View Type* selector:

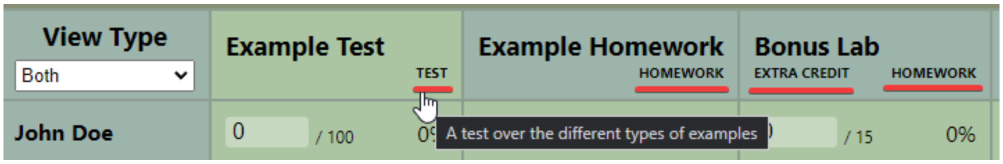


Extra Credit: Optional, points will be added to student’s total grade on top of other assignments, but won’t negatively impact them.

Maximum Points: Required, total points the assignment is worth. Individual grades can surpass this number, allowing you to give extra credit.

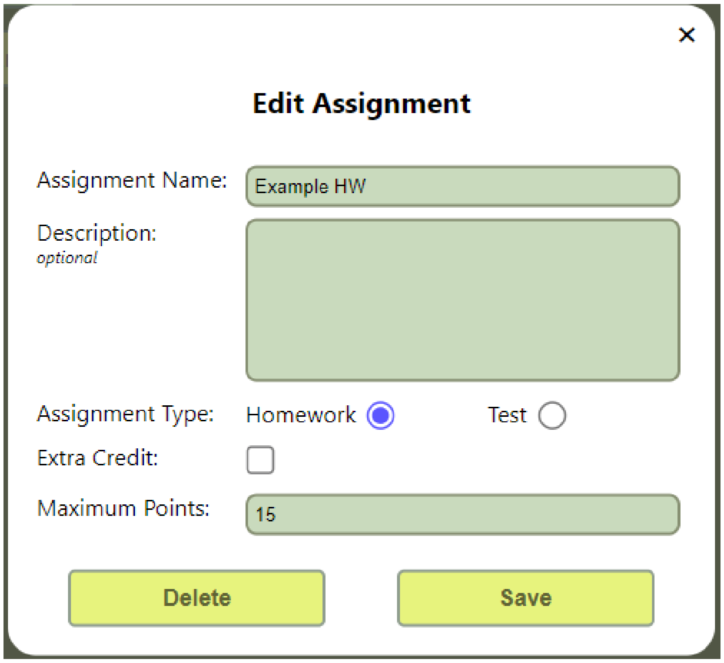
Once you've configured the assignment, choose "Create" to update the database.

Attributes for the assignment are displayed for each column, and you can hover over the assignment name to glance at the description:



View Type Both	Example Test TEST	Example Homework HOMEWORK	Bonus Lab EXTRA CREDIT HOMEWORK
John Doe	0 / 100 0%	A test over the different types of examples	/ 15 0%

You can also click the assignment's cell to edit it, or delete it.



Edit Assignment

Assignment Name: Example HW

Description: *optional*

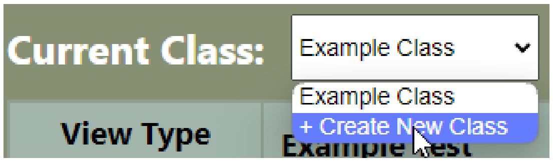
Assignment Type: Homework ● Test ○

Extra Credit: □

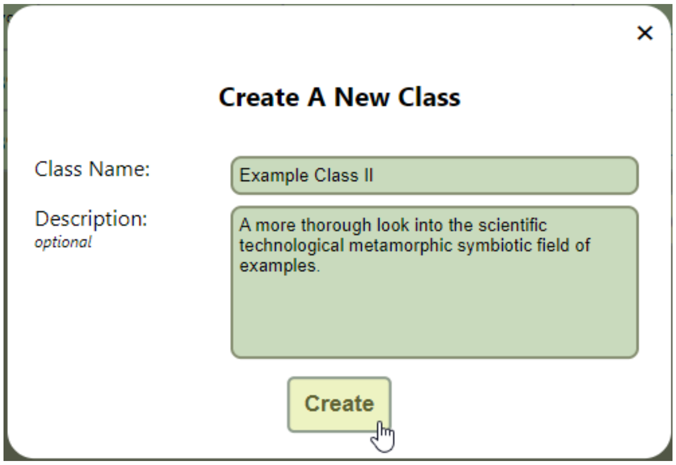
Maximum Points: 15

iv. Creating a Class

Creating a new class is very simple. Open the *Current Class* dropdown menu, and select "Create New Class".



The *Create A New Class* popup will appear, prompting you to fill out the class's information. After filling out the form, choose *Create*.


Create A New Class

Class Name: Example Class II

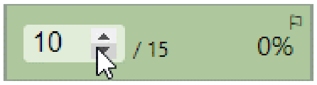
Description: *optional* A more thorough look into the scientific technological metamorphic symbiotic field of examples.

The new class will be added to the *Current Class* dropdown menu, and you will automatically switch to it. Here, you can go ahead and add student/assignments as you'd like.

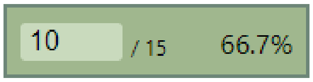
v. Grading System

Once your class is populated with students and assignments, entering grades is very simple.

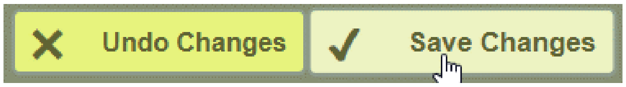
Each student has a *Grade Cell* for each assignment in the table. You can edit the student's current grade by selecting the current value, and just changing it. You can type the value, or use the arrows to increment.



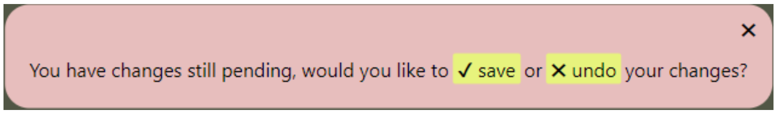
After updating the grade, the percentages in the table will update as well. You'll also notice that the cell changes color and a border appears..



This means that the cell has a *Pending Change*, and it hasn't yet been saved. You can save changes in bulk using the *Save Changes* button in the bottom-right, or you can undo all changes using the *Undo Changes* button.



Note: You cannot create assignments or add students while changes are pending. You will be prompted to Save or Undo changes.

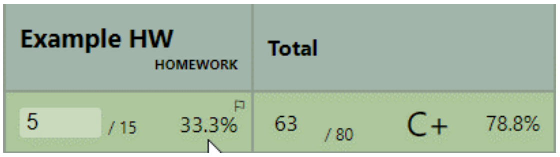


×

You have changes still pending, would you like to ✓ save or ✕ undo your changes?

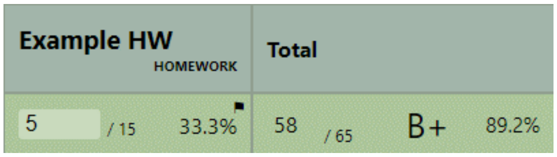
You'll also notice, when you hover over a cell, a little flag appears in the top-right corner. This is the *Exempt Flag*, and allows you to make a student's grade exempt from their final grade. Whatever grade they received will not affect their total.

Exempt disabled



Example HW HOMEWORK	Total
5 / 15 33.3%	63 / 80 C+ 78.8%

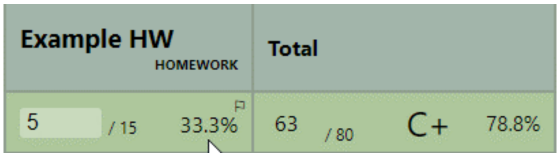
Exempt enabled



Example HW HOMEWORK	Total
5 / 15 33.3%	58 / 65 B+ 89.2%

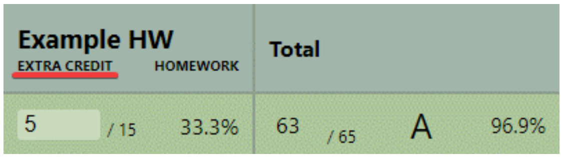
Assignments that are market as *Extra Credit* have a similar effect, but they apply to all students. Also, unlike an *exemption*, Extra Credit assignments can contribute positively to a user's final grade.

Regular Assignment



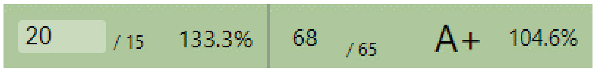
Example HW HOMEWORK	Total
5 / 15 33.3%	63 / 80 C+ 78.8%

Extra Credit Assignment



Example HW EXTRA CREDIT HOMEWORK	Total
5 / 15 33.3%	63 / 65 A 96.9%

This grading system **allows** percentages over 100%, as some teachers may like to reward exemplary effort put in by students.



20 / 15 133.3%	68 / 65 A+ 104.6%

3 Language / Technology Overview

a. TypeScript

TypeScript is a superset language that builds on top of JavaScript, a language primarily known for it's use as client-side code for web pages.

History

JavaScript is a high-level, interpreted programming language created in 1995 by Brendan Eich. Brendan's goal with JavaScript was to create a lightweight scripting language to make web pages more interactive by allowing client-side code to manipulate the Document Object Model (DOM). It was originally created for the NetScape web browser, but within a couple of years, Internet Explorer adopted JavaScript compatibility, and it has become a stable of web development since. Almost every computer (and phone, gaming console, etc) comes equipped with a JavaScript interpreter.

JavaScript has a ton of quirks that make it stand out as a programming language. For starters, it uses an asynchronous programming model, which allows code to continue running while waiting for a response from an external resource (such as a database query). Asynchronous languages are often harder to read, and can lead to unpredictable code if not structured correctly. Also, JavaScript isn't a typed language, meaning that a developer can change the type of data assigned to a variable at any point in the execution. Loosely typed languages are arguably easier to learn, which is part of the reason JavaScript is often the first language taught (alongside Python), however they can lead to many readability and writability issues. With it's original intent being small client-side programs, it's easy to understand why JavaScript is so loosely typed. However, with the scale of web applications growing massively in the early 2010's, a better system was needed.

In response to this problem, Microsoft created *TypeScript*, a superset of JavaScript which adds a strict typing system. TypeScript is written by the developer, and used during development/testing, but isn't used in the actual program that gets shipped to the users's browsers (since browsers only support JavaScript). Therefore, TypeScript's additions to your program are removed on compilation of your program. TypeScript recently surpassed JavaScript in usage for web applications, as most organizations now require it to be used instead. Both TypeScript and JavaScript continue to evolve over the years, working closely together to ensure that TypeScript is a clean, hassle-free option for developers.

Overview

We'll take a deeper dive into the specifics of TypeScript later on in this report, but here's some general remarks. TypeScript allows you to create your own *Interfaces*, allowing you to define and enforce an abstract data type for an object. This feature is very useful when developing, as most IDEs can use it for code auto-completion. Interfaces can also be nested, allowing you to simplify usage of very complex objects. Most mainstream JavaScript libraries come with TypeScript included, so it's very easy to take advantage of interfaces from other libraries. A typing system like this allows the IDE to notify you of many different potential errors BEFORE compilation, which results in a faster and cleaner workflow than using JavaScript alone.

TypeScript can infer types on its own when binded to a variable, such as a string, but you can also choose to explicitely type variables. It can also infer the return type of a function, but you can explicitely define that in the signature if you'd like. You can also define a function's signature before declaring the body, which can help you plan out a program before writing the logic.

Types can also be passed into functions, such as those from a library, to enforce types of variables that are abstract to you. For example, the *React* library has a function, `useState()`, which returns a variable (which you can use in your code), along with a function to change that variable. `useState()` is explained more thoroughly later in this report, but here's a sample of what it looks like.

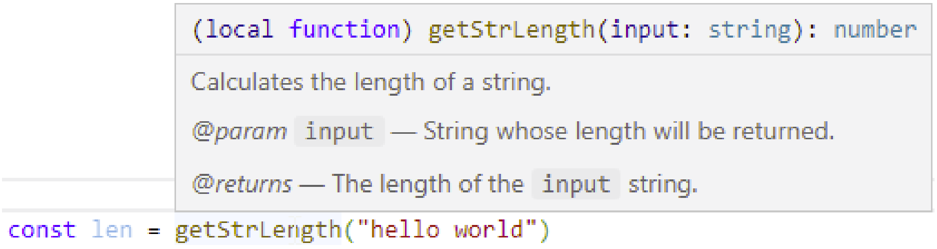
```
const [count, setCount] = React.useState<number>()
```

We are able to enforce the type of `number` onto this function, and the React developers have added functionality to reflect this in the variable and function returned. Trying to interpret `count` as a non-`number`, or trying to call `setCount()` with a non-`number` parameter, will throw an error in the IDE, as well as at compilation.

TypeScript also has a feature called *JSDocs*, which allows you to document your declarations (using comments) in a specific format. This documentation is often accessible by hovering over a function while using it, and is supported in most modern IDEs. You can apply JSDocs to variables, functions, type declarations, objects, and interfaces. Most mainstream libraries also include JSDoc documentation, and many organizations with complex TypeScript systems use it.

JSDoc Example (in VSCode)

```
/**
* Calculates the length of a string.
* @param input String whose length will be returned.
* @returns The length of the `input` string.
*/
function getStrLength(input: string): number {
    return input.length
}
```



```
(local function) getStrLength(input: string): number
```

Calculates the length of a string.

@param `input` — String whose length will be returned.

@returns — The length of the `input` string.

```
const len = getStrLength("hello world")
```

b. NodeJS

NodeJS is a open-source server-side JavaScript runtime environment which allows developers to create a web application's back end using JavaScript (with support for TypeScript).

History

In 2009, a developer named Ryan Dahl was working on a web project, when he became frustrated with the limitations of existing server-side technology. He wanted a solution that could handle simultaneous connections and real-time data streams well, and was disappointed with his options. He also wanted to be able to write server-side code using JavaScript, since it was already used for the client side. Thus, NodeJS was born.

NodeJS was very quickly adopted by the web development community, and in 2010, Dahl released *Node Package Manager (npm)*, a tool which made distributing libraries and frameworks much easier, as well as scripting features to automate repetitive tasks, such as compiling a project.

Today, NodeJS is reknown for it's stability and performance, and is used by many companies, such as Netflix, LinkedIn, and PayPal. It has become one of the most popular server-side technologies for building web applications that handle high data streams.

Overview

NodeJS operates as a server's back-end, allowing you to connect with databases and various APIs. It was created with an event-driven paradigm and a non-blocking I/O, contributing to it's design goal of handling multiple events and I/O operations efficiently. Because of this, it is able to handle multiple operations simultaneously, such as database queries, without blocking other operations.

NodeJS is designed to be very modular, making it easy to develop and maintain large-scale applications (supports abstraction). It has a massive community, which has contributed to the development of many useful modules and libraries, which are easily installed using the *Node Package Manager*.

Node is also open source, and supports many different styles of web development and server architecture. Since it's a back-end platform, it can handle HTTP requests to connect streams of content to the client. Also, Node can be capable of pre-rendering JavaScript on HTML pages, allowing for faster load times, and better *Search Engine Optimization (SEO)*.

c. SQLite

SQLite is essentially a lightweight SQL database, which writes all data immediately to a database file. It doesn't require a server process to run, and you can run traditional SQL queries on it using APIs or third-party software.

History

In 2000, Dr. Richard Hipp was working on a project for the United States Navy. At the time, SQL was a very popular language that was known by many developers. However, SQL providers at the time (such as MySQL) were server-based, and Richard needed something more portable.

Originally developed in TCL, and later rewrote in C, SQLite gained popularity due to it's reliability and lightweight nature. Over time, SQLite has been adopted by many different types of systems, such as mobile applications, desktop applications, and even web browsers.

SQLite is still seen as a go-to option by many application developers, as SQL is still extremely popular, and developers see no reason to learn a new database architecture when SQLite already meets their needs.

Overview

SQLite, like SQL, is a Relational Database Management System. It follows traditional SQL syntax, with little variance. An entire instance is stored in a single file, making it extremely easy to back up, share, and use. SQLite has many APIs, which allow it to be accessed by back-end languages. These APIs contain many functions, with support for prepared statements, enabling protection against *SQL Injection* attacks.

d. Electron

Electron is an extension for NodeJS, which allows you to create cross-platform desktop applications using HTML/CSS/JS, with access to operating system functionality, such as file access.

History

Electron was developed by Cheng Zhao (GitHub co-founder) in 2013. It was originally a closed-source project named *Atom Shell*, with to goal of creating a text-editor which ran on the desktop by bunding a Chromium instance with NodeJS.

In 2014, it was renamed to *Electron*, with a new open-source paradigm. Cheng realized that it could be used for much more complex applications than his original intent. Since 2014, Electron has gained massive popularity, probably due to the coinciding popularity of web development, as web technology evolved.

Electron is used for realtime communication tools, such as Slack, Discord, and Microsoft Teams, as well as Visual Studio Code and Trello.

Overview

Electron works by combining a Chromium instance (explained later) with a NodeJS runtime. Electron projects can be organized very similarly to web projects, as there is a "front-end" renderer process, a "back end" main process, and an Inter-Process Communication (IPC) bridge to communicate between the two.

This allows you to use the HTML/CSS/JS stack to create the UI for your desktop application, and use NodeJS to connect it to various APIs and OS functionalities. It comes equipped with a menu-bar system (which'll reflect whatever OS the application is running on), file browsing, and more.

Electron does have a major downside, however. Each installation of an electron instance comes bundled with an instance of Node. This is fine, as Node is very lightweight. Each installation also includes an instance of *Chromium*, a stripped-down version of the *Chrome* browser, created by google. Due to this, Electron applications have a boilerplate size of 85mb, and a minimum of 80mb of RAM usage per instance. These numbers are just for bare electron, anything you add to that will increase the usage. Large scale applications, such as Slack, can sometimes even reach multi-GB RAM usage. Though modern machines are perfectly capable of handling this, having multiple instances open can be a challenge, and these numbers are unreasonable for the complexity of the application. There is a common saying in the Electron community: *Developers love it, customers hate it.*

As a final note, in 2022, a competitor to Electron was release called *Tauri*. Tauri, like Electron, is designed to create cross-platform desktop application using web technologies. However, there are two distinctly intriguing differences with Tauri. Instead of Chromium, it uses a custom web renderer, which is extremely lightweight. Also, instead of using NodeJS as the “back-end”, it uses Rust. This results in lower RAM usage and application sizes that are often over 10x smaller than their Electron counterparts, and better performance overall. However, due to it’s newness, the technology is still quite experimental. So, I opted to use Electron instead for this project.

e. HTML/CSS

HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are two languages processed by the browser to create the UI of a web page. HTML handles the content, and CSS handles how the content is styled. The two languages are highly intertwined and often learned simultaneously, so I chose to combine them into one section.

History

HTML was first created in 1990 by Tim Berners-Lee, a CERN employee. Initially, HTML only allowed for the creation of basic text documents with hypertext links. In 1995, HTML 2.0 launched, adding support for table structures and images. In 1997, HTML 3.2 release wit hsupport for forms, frames, and cascading style sheets (CSS).

CSS was developed by another CERN employee, Håkon Wium Lie. CSS was designed to separate the presentation of a document from it’s content, allowing for more sophisticated and maintainable website designed.

Both languages have evolved drastically over the years. HTML’s last major update was in 2014, HTML5, which included support for multimedia, canvas drawings, improved semantics, and much more. CSS3 was also released in 2014, adding support for animations and 3D transformation. CSS continues to get minor updates frequenty, recently specifically focusing on improving approach options using psuedo-selectors.

HTML, CSS, and client-side JavaScript have a unique “release” system. Since this code is interpreted by the *browser*, the browser developers need to add compatibility for each update. Because of this, especially with CSS, there are sometimes cases where features are fully supported in some browsers, and not available in others. There also used to be an issue with *vendor prefixes*, requiring developers to use different CSS attributes for different browsers to achieve cross-browser compatibility. This would lead to redundant code, like this:

```
.anim {
    transition: all 4s ease;
    -webkit-transition: all 4s ease;
    -moz-transition: all 4s ease;
    -ms-transition: all 4s ease;
    -o-transition: all 4s ease;
}
```

However, in recent years, it seems competing browsers have mostly standardized their interpretation of CSS, so this is largely not an issue anymore.

Overview

HTML and CSS are essential tools in any developers toolkit these days, as many development jobs work with web UI at some point. HTML features a nested structure, using opening and closing tags to structure elements and content. While CSS can be done per-element in HTML’s markup, it is often used with the selector system, allowing you to select HTML elements in a variety of ways, including *class*, *id*, *type*, *type with specific attribute*, *psuedoselectors*, and more.

There is room to debate whether or not HTML is a programming language, since it cannot perform any actual calculation on it’s own. CSS, however, can perform calculations to generate display values, such as calculating the number of horizontal pixels on a page and cutting that number by 50%.

f. React

React is a JavaScript library designed to simplify creation of dynamic UIs with stateful management. It includes mechanisms that allow you to auto-update the UI when specific data changes.

History

React was created in 2011 by Jordan Walke, a software engineer at Facebook. It was originally designed to simplify declarative tasks in designing Facebook's increasingly complex UI, such as updating a DOM element when a value changed. It was used internally in facebook, and in 2013, React was released as an open-source project. Since then, it has been developed quite a bit, and has quickly become the most popular.

React's ripple effect on the web development community has been massive. Even though it is only a library, it has so many intricacies and specificities that it's developed a job bracket of it's own. Many companies hiring for front-end developers specifically require knowledge of React. It has also spawned many similar JavaScript frameworks with a similar goal- to simplify the relationship between JavaScript and the DOM. React has developed a large collection of libraries as well, such as React Router or Remix, which are designed to extend React's capabilities and further simplify certain concepts within it, such as state management and page navigation.

Overview

In a traditional React environment, all HTML content is rendered by the JavaScript engine, instead of being pre-defined in a HTML file. React is designed to be structured in a modular way, allowing separate components to be re-rendered while other components remain static. It uses a *Virtual DOM (Document Object Model)*. When a stateful component updates, the component is re-rendered in the *VDOM*, changes are compared to the current DOM, and differences are rendered on the page. React includes many methods designed to optimize code, to prevent unneccesary re-renders, and methods to share stateful values with other components. Here's an example of a simple React counter component:

```
function Counter() {
    const [count, setCount] = React.useState(0)

    function incrementCount() {
        setCount(count + 1)
    }

    return (
        <div>
            <h1>Counter: {count}</h1>
            <button onClick={incrementCount}>+</button>
        </div>
    );
}
```

Whenever the user clicks the button, the value of count is incremented by 1. Since it is incremented by using the setCount method, react re-renders the entire component, and the change is reflected on the user's page.

If you're familiar with JavaScript, this code may look confusing. The component *looks* like it is returning HTML elements, which isn't a native feature in JavaScript. While React is only a library, it is often paired with a JavaScript Syntax extension called *JSX*, which allows this feature. On compilation, the HTML-like code is converted to it's JavaScript equivalent. Here's what the previous example's JSX looks like after compilation:

```
return (
    React.createElement("div", null,
        React.createElement("h1", null, "Counter: ", count),
        React.createElement("button", { onClick: incrementCount }, "+")
    )
)
```

Many large websites use React, such as Facebook, Netflix, AirBnB, and DropBox. The React team also created React Native, which is used to build cross-platform native mobile applications. Unlike this library, React Native is fully compiled to the device's native code, while still allowing you to use HTML/CSS/JS to create the app.

4 TypeScript Explanation

a. Syntax

Since TypeScript is a superset of JavaScript, it shares the same syntax, except with typing systems added. Due to this, this explanation of TypeScript will overlap with JavaScript quite a bit.

Variables are declared using the keywords const, let, or var. You can also specify a type using the ”:” symbol. The const keyword declares a *constant* variable, which cannot be reassigned. Variabled declared with let and var can be reassigned. The var keyword is a relic of the past, and shouldn’t be used. Unlike const and let, which are *block scoped*, a variable declaration with var is hoisted to the top of the function, and declared there. This leads to readability and reliability issues, as it can be harder to nail down errors. Here’s an example of the declaration syntax, and a demonstration of the var issue.

```
const PI: number = 3.14159
let firstname: string = "Joe"
var balance: number = 3116.44

function fee() {
    console.log(x)  // logs "undefined"
    var x = 10
}

function foo() {
    console.log(x)  // throws "ReferenceError: x is not defined"
    let x = 10
}
```

JavaScript uses curly braces to define scopes. You *can* use semicolons to end each statement, but it isn’t required. I personally prefer not to (unless I have multiple statements on one line, which requires semicolons), but it just comes down to preference.

Also shown in the example is a *function definition.* Functions can also take parameters, which you can type if you’d like. TypeScript is mostly a suggestive extension. You will get warnings if you don’t type your parameters, but your code will still execute fine. You can also type the return value of a function, as shown below.

```
function getStrLength(input: string): number {
    return input.length
}
```

You can also declare a function as a variable using the *arrow function* syntax. This syntax is less readable, but can be convenient in some cases.

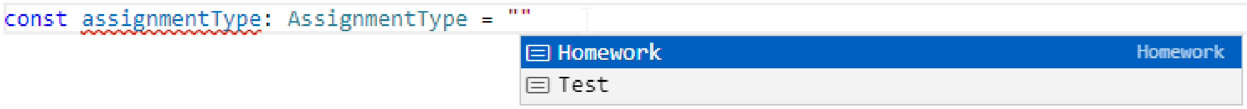
```
const getStrLength = (input: string) => input.length
```

Classes are defined/used in a similar way to *Java*. You can use the `class` keyword to define a class, and `new` to define an instance of a class. You must also use a TypeScript *interface* to define the object's properties. *Interfaces* are an integral part of TypeScript, and are implemented heavily throughout my project. Also, like Java, there are no destructors in JavaScript.

```
interface Car {
    make: string,
    model: string,
    year: number
}

class Car {
    constructor(make: string, model: string, year: number) {
        this.make = make
        this.model = model
        this.year = year
    }

    getInfo() {
        return `This is a ${this.year} ${this.make} ${this.model}.`
    }
}

let myCar = new Car("Toyota", "Corolla", 2022)
console.log(myCar.getInfo()) // "This is a 2022 Toyota Corolla."
```

TypeScript also allows you to define a `type` , which can further restrict which values are allowed for a variable.

```
type AssignmentType = "Homework" | "Test"
```



```
const assignmentType: AssignmentType = ""
```

TypeScript has pretty good intuition, but it doesn’t always know the type of a variable/attribute. To work around this, you can use the as keyword to assert a type. Doing so will avoid compilation errors, and give you useful auto-completion in your IDE.

```
const parentNode = e.target.parentElement as HTMLInputElement
```

b. Control Structures

TypeScript’s control structures are fairly basic for a programming language. It has the standard if-else two-way selection statement, as well as if-elseif-else for multi-way selection. TypeScript also supports switch-case statements.

```
// two-way selection statement
if (x > y) {
    /* ... code ... */
} else {
    /* ... code ... */
}

// multi-way selection statement
if (x > y) {
    /* ... code ... */
} else if (x == y) {
    /* ... code ... */
} else {
    /* ... code ... */
}

// switch-case
let day: DayOfWeek = "tue"
let isWeekend: boolean
switch (day) {
    case "sat":     isWeekend = true;   break
    case "sun":     isWeekend = true;   break
    default:        isWeekend = false;  break
}
```

There is also support for two-way ternary operators, which can even be done in variable/property declarations.

```
const person = {
    name: "joe",
    mood: isWeekend ? "Happy" : "Sad"
}
```

c. Data Structures

JavaScript provides a variety of built-in data structures. We've already covered classes pretty thoroughly, but here's some other structures.

One data structure is the *Array*, which can store a collection of values, and access them individually. The *Array* also has many different methods, such as map(), which returns a transformed version of the array depending on the function you provide it.

```
const myArray = [1, 2, 3, 4]
console.log(myArray[2]) // Output: 3

const myArraySquared = myArray.map(item => {
    return item * item
})  // [1, 4, 9, 16]
```

JavaScript also has *Tuples*, which are similar to arrays, but have a fixed length, and each element can be typed specifically (TypeScript feature).

```
let myTuple: [string, number] = ["Josh", 22]
console.log(myTuple[0]); // Output: "John"
```

Objects are a key data structure in JavaScript, allowing you to store data in a fixed format. Objects, unlike arrays, have named elements which can be referenced directly. Objects can also contain functions, and can be nested with other objects.

```
const myObj = {
    name: "josh",
    age: 22,
    city: "Toledo"
}
```

You can use TypeScript's *interfaces* system to define the shape of an object. The interface, when applied, enforces good data structure, and can help with autocompletion.

```
interface Student {
    name: string,
    age: number,
    city?: string   // ? indicates optional
}
const josh: Student = {
    name: "josh",
    age: 22,
    city: "Toledo"
}
```

JavaScript also has *maps*, which are collections of key-value pairs. Maps are similar to objects, but maps can have any type of key, and provide methods for adding/removing items.

```
let myMap = new Map()
myMap.set("name", "Josh")
myMap.set("age", 22)
myMap.set("city", "Toledo")
console.log(myMap.get("age")) // Output: 22
```

d. I/O

JavaScript is natively used in the browser, and is primarily event-driven. There are two ways to listen for user inputs: Event Listeners and On-Element Event Triggers.

Event Listener

```
function incrementCount() {
    setCount(count + 1)
}

const buttonNode = document.getElementById('increment_button')
buttonNode.addEventListener('click', incrementCount)

return (
    <div>
        <h1>Counter: {count}</h1>
        <button id='increment_button'>+</button>
    </div>
)
```

On-Element Event Trigger

```
function incrementCount() {
    setCount(count + 1)
}

return (
    <div>
        <h1>Counter: {count}</h1>
        <button onClick={incrementCount}>+</button>
    </div>
)
```

When an element's event is triggered, it's also passed an event object. The event object contains some information and methods, such as preventDefault(), and also a target object, which is a reference to the actual DOM element that triggered the event. You can manipulate the target's properties, get it's parent node, sibling nodes, and more. You can use this functionality to create repeatable event functions.

For example, let's say we had a React webpage that loads *after* retrieving an array of content from a fetch. The following code is used to transform that data into an array of JSX Elements:

```
const displayContent = APIResult.map(item => {
    return (
        <div>
            <p>{item.content}</p>
            <button onClick={hideThisDiv}>Hide</button>
        </div>
    )
})
```

In this case, we'd use the event.target attribute to handle the Hide button's event. This way, even if the amount of content is unknown, the function will work per-item.

```
function hideThisDiv(event: React.MouseEvent<HTMLButtonElement>) {
    const buttonNode = event.target as HTMLButtonElement
    const containerNode = buttonNode.parentElement as HTMLDivElement
    containerNode.hidden = true
}
```

e. Parameter Passing

In JavaScript, primitive data types such as strings, numbers, and booleans are passed by value. This means that the value of a variable is copied into the function, instead of the memory reference itself, and cannot be manipulated inside of the function. Objects and arrays are passed by reference, and *can* be manipulated by the function. There is no way to specify/change this behavior.

f. Scope Rules

JavaScript, outside of the var keyword, is entirely block-scoped with no exceptions. A function definition can take place after it's usage, but variables (including arrow functions) must be declared before usage. You can declare a global-like variable by just declaring it outside of any explicit scope, but you'd need to import it into other files if you'd like to use it.

g. Stack / Heap

JavaScript is a very high-level language, as the memory management is abstracted away from the developer. Primitive variables, such as strings and numbers, are stored on the *stack*. When they are referenced, the interpreter searches down the stack for the most recent value of the variable, and grabs it. Functions, arrays, and objects are stored on the *heap*. This is why you can create variables (references) of objects, and manipulating them changes the original object. You are only copying the reference to the object, not the object itself.

h. Error Handling Procedures

JavaScript uses the common `try`/`catch` syntax for error handling. Inside of the `try` block, you enter code that may throw an `Error` (or, you can throw an `Error` yourself using `throw new Error()`). The catch block can take an `error` parameter, which'll catch the error `Error` thrown. The `Error` object has a `message` property, which has a detailed description of the error. You can also use a `finally` block to define code that'll run after the `try`/`catch`, regardless of if an error occurs. JavaScript normally will continue execution, despite any errors, however this does not apply to the `try` block. When an error occurs inside of a `try` block, execution is immediately rerouted to the `catch` block.

```
function divide(numerator: number, denominator: number): number {
    try {
        if (denominator === 0) {
            throw new Error("Divide by zero error")
        }
        return numerator / denominator
    } catch (error) {
        console.log(error.message)
        return NaN
    } finally {
        console.log("divide by zero completed.")
    }
}

let result = divide(10, 0) // output: "Divide by zero error"
```

5 TypeScript Evaluation

For this portion, I will be comparing TypeScript with Java, Python, and Visual Basic.

a. Readability

JavaScript itself is a very readable language, when the writer follows standard practices (such as indentation). TypeScript further improves this by adding additional documentation through it's typing system. Also, the JSDoc system mentioned earlier, dramatically improves the programming experience and decreases the need to look at a manual to understand functions.

Compared to Java

Java and JavaScript are very syntactically similar, however Java code is very declarative and verbose (looking at you, `public static void main (String args[])`…). Whether or not this helps/hurts readability depends on the complexity of the program, and the experience/knowledge of the developer. For a newcomer, seeing Java's boiler plate code may scare them away, since they aren't sure what those keywords actually do. However, generally in these analysises, declarative code means higher readability, since you can get a better understanding of the program by looking at the code. In my *personal opinion*, this complexity hurts Java's readability, but for experienced developers, they may see this as a readability boost. TypeScript wins, in my opinion, but seasoned Java developers may reasonably disagree.

Compared to Python

Python is known for it's readability. Unlike TypeScript, Python uses indentation for block-scoping. This enforces clean code practices, whereas TypeScript could technically be written entirely on one line. Python, like JavaScript, is a very high-level language, and doesn't require typing (though unlike JavaScript, it is a built-in option). Other than forced indentation, TypeScript and Python are very similar in terms of readability. This is nearly a tie, but due to enforced indentation, Python wins.

Compared to Visual Basic

Visual Basic is arguably more reknown for it's readability than Python. VB uses a more english-like syntax, using keywords such as `Then` and `End If`, to define scopes. Visual Basic doesn't have any indentation requirement. This system makes the language very readable, and quite approachable for beginners. Comparing this to JavaScript's curly-brace syntax, Visual Basic wins in terms of readability.

b. Writability

JavaScript is a very writable language, as there is very little boilerplate code. However, the curly brace syntax can be slightly inconvenient.

Compared to Java

Java is a difficult to write language. As mentioned before, there is a ton of boilerplate code, which slows down typing and can interrupt the flow of thought. Java also uses a strict typing system, which hurts writability, but helps reliability (and modularity). JavaScript, on the other hand, has very minimal boilerplate code, and is quite easy to write (especially if you're used to similar language syntax). TypeScript's (mostly optional) added typing system hurts writability, but helps reliability. TypeScript wins by a mile.

Compared to Python

Python's indentation-based syntax is much easier to write with than JavaScript's curly-braced syntax. With JavaScript, you *should* be indenting anyways, so there's more steps to take to create block-scoped than python. Both Python and TypeScript have optional typing systems. Python wins with a slight edge.

Compared to Visual Basic

Visual Basic's usage of keywords may make it more writable than JavaScript. As a developer, I don't like using english works to define block scopes, but it does flow more naturally than using curly brackets. However, just to note, Visual Basic has a vague newline requirement which can (and for me, has) made writing it quite confusing. Due to the naturality of the english-like syntax, Visual Basic wins.

c. Reliability

JavaScript alone is extremely unreliable, between it's lack of typing and *unique* type coercion system (check this out if you want a chuckle), which has many developers scratching their heads. TypeScript's added functionality does wonders for reliability, but doesn't help much with the abnormal type coercion system. An developer unfamiliar with JavaScript's wacky logic can easily make mistakes, but to a knowledgeable developer, the type coercion system can aid writability.

Compared to Java

Java is a king of reliability. The very strict typing system and verbose writing structure creates an opinionated developer experience, catching many errors at compilation (if not in the IDE). Java also has a large community and many tools available to improve reliability. Java wins.

Compared to Python

Python has a dynamic typing system, like JavaScript, which hurts reliability a lot. However, types are an option, so if a developer wishes, they can improve reliability by using a strict typing system. Python also has a diverse ecosystem of tools, many testing frameworks, and a lot of libraries which make it an easy to use language. While Python and TypeScript are pretty similar, Python's implicit typing system isn't enforced as tightly as TypeScript's, so TypeScript wins.

Compared to Visual Basic

Visual Basic isn't a very reliable language. Many Visual Basic IDEs don't catch most errors until execution. Visual Basic does *have* a typing system, but it isn't required and not very fleshed out. Visual Basic has a very limited feature set and low-level constructs, which make it harder to write robust code, especially for larger projects. TypeScript wins.

d. Cost

TypeScript is an extremely high-level language, and is interpreted, so it's performance isn't anything remarkable. However, advancements in both consumer hardware and interpreters have helped close the gap between it and full-compiled programs. However, the strong typing system makes maintenance cost significantly lower, and the concise syntax means quicker development time.

Compared to Java

Java has a high development cost due to it's verbose syntax, but for the same reason, less maintenance will be needed in the future. Out of all of these languages, Java has the best performance optimization, as it runs closer to machine code as these other languages. Java wins.

Compared to Python

Python has a low development cost due to it's simple syntax, but the dynamic typing system can certainly create a need for more maintenance in the future. Python is an interpreted language, like JavaScript, so performance is similar. Overall, TypeScript wins.

Compared to Visual Basic

Visual Basic is more writable than TypeScript, but it can have strange behavior that leads to a lot of debugging. VB's loose typing system makes it easy to make mistakes, which'll lead to maintenance issues. VB is also interpreted, like JavaScript. Overall, TypeScript wins.

6 Appendix

This Appendix contains the source code for this program as requested. However, there is a ton of code here, and a bunch of complex parts. Some of the code attached is boilerplate from Electron and Node, so it can be hard to distingush what code is actually written by me. I'd **highly recommend** looking through my GitHub reposity for this project instead, linked below. I've used this repository since the creation of this project, with detailed and regular commits, so it'd be much more convenient to use that. You can also easily distinguish between the boilerplate code and my actual contributions.

https://github.com/JoshSilveous/grading-application

Also, as a self-critique, I definitely made some wrong decisions throughout this process. I started off with a slower, methodical pace, but towards the end I began to rush out features. I had put at least 50 hours of work into this program, and had a ton of other class projects weighing over my head, so I got sloppy. This can be seen in the **ClassTable.tsx** component, which handles most of the UI. I should've split that component into multiple files, which would make it much clearer for other developers to understand. I also primarily used Event objects to handle button trigger updates, which leads to some reliability issues if I wish to expand the program in the future. I should have used React's Ref system instead.

This was my first time working with NodeJS, Electron, SQLite, and some other technologies, so I made some mistakes and development took a while. If you'd like to install the program, download **JoshSilveousFinal.zip** from the GitHub repository, extract it, and run the **installer.cmd** file.

Installer Source Code

```
@echo off

set THISDIR=%~dp0
set PACKAGE=josh-silveous-grading-application-win32-x64\package
set SHORTCUT=josh-silveous-grading-application-win32-
x64\JoshSilveousGradingApplication.lnk
set INSTALLSOURCE=%THISDIR%%PACKAGE%
set INSTALLDEST="C:\joshs-grading-application"
set SHORTCUTSOURCE=%THISDIR%%SHORTCUT%
set SHORTCUTDEST="C:\Users\Public\Desktop"



:: Check to see if ran as administrator
net session > NUL
cls
if %ERRORLEVEL% NEQ 0 (
    color 0C
    echo ERROR: Please re-run this installer as administrator.
    pause
    exit

) ELSE (
    color 0A
    echo PASSED Administrator check.


    color 0F
    echo Checking if installation files exist...

    if EXIST %INSTALLSOURCE% (
        color 0A
        echo Files found.

    ) ELSE (
        color 0C
        echo Files not found. Please re-download the installation.
        exit

    )

    cls
    color 0F
    echo Press any key to begin installation...
    pause

    echo Copying files...
    xcopy %INSTALLSOURCE% %INSTALLDEST% /E /I /Y
    cls
    echo Files copied


    echo Copying Desktop Shortcut..
    xcopy %SHORTCUTSOURCE% %SHORTCUTDEST%
    echo File copied
    cls

    echo Setting shortcut target
    cscript %THISDIR%\desktopicon.vbs
    echo Shortcut target successfully set
    cls


    color 0A
    echo Installation complete.
    echo:
    echo You can start the program using "JoshSilveousGradingApplication" on your desktop.
    echo:
    echo You may close this window.
    echo:
    echo:
    pause
)
```

Database Bridge Functions (SQLite Core)

```
import sqlite from 'better-sqlite3'
import fs from 'fs'
import path from 'path'

const DBFileDir = path.join(process.env.USERPROFILE, 'Documents',
'josh_grading_application_data')
const DBFilePath = path.join(DBFileDir, 'data.db')

// check if directory exists, create if not
if (!fs.existsSync(DBFileDir)) {
    fs.mkdirSync(DBFileDir, { recursive: true })
}

let db = sqlite(DBFilePath)

// Potential performance buff, may apply if needed
// see https://github.com/WiseLibs/better-
sqlite3/blob/master/docs/performance.md
// db.pragma('journal_mode = WAL');

// Enables enforcing foreign key constraints, such as cascade deleting
db.pragma('foreign_keys = ON')
export default db
```

Database Functions – Assignment

```
import db from './db_bridge'

function createAssignment(
    class_id: number,
    name: string,
    description: string,
    assignment_type: AssignmentInfo['assignment_type'],
    is_extra_credit: boolean,
    max_points: number
): number {

    /*
        order_position determines the order in which assignments will be layed
out in
        the table for the user. This is stored in the database so it can
persist among
        re-loads of the database. This chunk of code gets the number of current
        assignments in the class, to define the new assignment's position.
        I didn't end up implementing this feature in the design, so it doesn't actually do anything currently.
    */
    const sqlGetAssignmentNum = `
        SELECT * FROM Assignment WHERE class_id = ?;
    `
    const stmtGetAssignmentNum = db.prepare(sqlGetAssignmentNum)
    const order_position = stmtGetAssignmentNum.all(class_id).length


    const sqlInsert = `
        INSERT INTO Assignment VALUES
        (NULL, ?, ?, ?, ?, ?, ?, ?);
    `
    const stmtInsert = db.prepare(sqlInsert)
    const resInsertNewID = stmtInsert.run(
        name, description, assignment_type,
        is_extra_credit ? 1 : 0,
        max_points, order_position, class_id
    ).lastInsertRowid


    // Create entries in the `Grade` table for each assignment, with a default value of 0
    const sqlGetStudentsInClass = `
        SELECT student_id FROM Enrollment WHERE class_id = ?;
    `
    const stmtGetStudentsInClass = db.prepare(sqlGetStudentsInClass)
    const studentsInClass = stmtGetStudentsInClass.all(class_id)

    const sqlInsertGrade = `
        INSERT INTO Grade VALUES (?, ?, 0, 0);
    `
    const stmtInsertGrade = db.prepare(sqlInsertGrade)
    studentsInClass.forEach(student => {
        stmtInsertGrade.run(student.student_id, resInsertNewID)
    })


    return resInsertNewID as number
}

function editAssignment(
    assignment_id: number,
    name: string,
    description: string,
    assignment_type: AssignmentInfo['assignment_type'],
    is_extra_credit: boolean,
    max_points: number
): void {
    const sql = `
        UPDATE Assignment
        SET name = ?,
            description = ?,
            assignment_type = ?,
            is_extra_credit = ?,
            max_points = ?
        WHERE assignment_id = ?;
    `
    const stmt = db.prepare(sql)
    stmt.run(
        name, description, assignment_type,
        is_extra_credit ? 1 : 0,
        max_points, assignment_id
    )
}

function deleteAssignment(assignment_id: number): void {
    const sql = `
        DELETE FROM Assignment
        WHERE assignment_id = ?;
    `
    const stmt = db.prepare(sql)
    stmt.run(assignment_id)
}

function getAssignment(assignment_id: number): AssignmentInfo {
    const sql = `
        SELECT * FROM Assignment
        WHERE assignment_id = ?;
    `
    const stmt = db.prepare(sql)
    const res: AssignmentInfo = stmt.get(assignment_id)
    return res

}

function updateAssignmentOrder(assignment_id: number, order_position: number):
void {
    const sql = `
```

```
        UPDATE Assignment
        SET order_position = ?
        WHERE assignment_id = ?;
    `
    const stmt = db.prepare(sql)
    stmt.run(order_position, assignment_id)
}



declare global {
    interface AssignmentInfo {
        assignment_id: number,
        class_id: number,
        name: string,
        description: string,
        assignment_type: "HOMEWORK" | "TEST",
        is_extra_credit: boolean,
        max_points: number
    }
    interface assignment_func_exports {
        /**
         * Creates an assignment in the database.
         * @param class_id The ID of the class associated.
         * @param name The name of the assignment.
         * @param description The description of the assignment (can be blank,
must be provided).
         * @param assignment_type The type of assignment. "HOMEWORK" or "TEST".
         * @param is_extra_credit boolean, whether the assignment should impact
grade negatively.
         * @param max_points Total points the assignment is worth.
         * @returns The ID of the newly-created class.
         */
        createAssignment: (
            class_id: number,
            name: string,
            description: string,
            assignment_type: AssignmentInfo['assignment_type'],
            is_extra_credit: boolean,
            max_points: number
        ) => number,
        /**
         * Edit an assignment in the database.
         * @param assignment_id The ID of the assignment.
         * @param name The name of the assignment.
         * @param description The description of the assignment (can be blank,
must be provided).
         * @param assignment_type The type of assignment. "HOMEWORK" or "TEST".
         * @param is_extra_credit boolean, whether the assignment should impact
grade negatively.
         * @param max_points Total points the assignment is worth.
         */
        editAssignment: (
            assignment_id: number,
            name: string,
            description: string,
            assignment_type: AssignmentInfo['assignment_type'],
            is_extra_credit: boolean,
            max_points: number
        ) => void,
        /**
         * Delete an assignment from the database.
         * @param assignment_id The ID of the assignment.
         */
        deleteAssignment: (assignment_id: number) => void,
        /**
         * Get an assignment's data from the database
         * @param assignment_id The ID of the assignment.
         * @returns An object containing all of the assignment's data.
         */
        getAssignment: (assignment_id: number) => AssignmentInfo,
        /**
         * Updates the display order for the assignment in the class.
         * @param assignment_id The ID of the assignment.
         * @param order_position The display order for the assignment.
         */
        updateAssignmentOrder: (assignment_id: number, order_position: number)
=> void
    }
}

export default {
    createAssignment,
    editAssignment,
    deleteAssignment,
    getAssignment,
    updateAssignmentOrder
} as assignment_func_exports
```

Database Functions – Class

```
import db from './db_bridge'

function getClassData(class_id: number): ClassData {

    const sqlClassInfo = `
        SELECT * FROM Class WHERE class_id = ?;
    `
    let stmtClassInfo = db.prepare(sqlClassInfo)
    let retClassInfo = stmtClassInfo.get(class_id)


    const sqlAssignments = `
        SELECT * FROM Assignment WHERE class_id = ?;
    `
    const stmtAssignments = db.prepare(sqlAssignments)
    const retAssignments: AssignmentInfo[] = stmtAssignments.all(class_id)

    // Replace integers with booleans
    const retAssignmentsAdj = retAssignments.map(asgn => {
        return {...asgn, is_extra_credit: asgn.is_extra_credit ? true : false}
    })

    // Create a list of assignment_id IN Class
    let listAssignments = ""
    retAssignments.forEach((assignment, index) => {
        listAssignments += index ? ',' + assignment.assignment_id :
assignment.assignment_id
    })

    const sqlStudentNames = `
        SELECT Student.student_id, first_name, last_name
            FROM Enrollment INNER JOIN Student
                ON Enrollment.student_id = Student.student_id
            WHERE class_id = ?;
    `
    let stmtStudentNames = db.prepare(sqlStudentNames)
    let retStudentNames = stmtStudentNames.all(class_id)


    const studentInfo = retStudentNames.map(student => {
        const sqlGrades = `
            SELECT assignment_id, earned_points, is_exempt
                FROM Grade
                WHERE student_id = ?
                AND assignment_id IN (${listAssignments});
        `
        const stmtGrades = db.prepare(sqlGrades)
        const resGrades = stmtGrades.all(student.student_id).map(grade => {
            // convert is_exempt from number (1 or 0) to boolean (true or
false)
            return { ...grade, is_exempt: grade.is_exempt ? true : false }
        })

        return {
            student_id: student.student_id,
            first_name: student.first_name,
            last_name: student.last_name,
            grades: resGrades
        } as StudentGrades
    })


    return {
        id: class_id,
        name: retClassInfo.name,
        description: retClassInfo.description,
        assignments: retAssignmentsAdj,
        studentInfo: studentInfo
    } as ClassData

}

function createClass(name: string, description: string): number {
    const sql = `
        INSERT INTO Class (name, description)
        VALUES (?, ?);
    `
    const stmt = db.prepare(sql)
    const res = stmt.run(name, description)

    return res.lastInsertRowid as number
}

function deleteClass(class_id: number): void {
    const sql = `
        DELETE FROM Class WHERE class_id = ?;
    `
    const stmt = db.prepare(sql)
    stmt.run(class_id)
```

```
}

function getClassInfo(class_id: number): ClassInfo {
    const sql = `
        SELECT * FROM Class WHERE class_id = ?;
    `
    const stmt = db.prepare(sql)
    const res: ClassInfo = stmt.get(class_id)

    return res
}

function getClassList(): ClassInfo[] {
    const sql = `
        SELECT * FROM Class;
    `
    const stmt = db.prepare(sql)
    const res: ClassInfo[] = stmt.all()

    return res
}

function editClass(class_id: number, name: string, description: string): void {
    const sql = `
        UPDATE Class
        SET name = ?,
            description = ?
        WHERE class_id = ?;
    `
    const stmt = db.prepare(sql)
    stmt.all(name, description, class_id)
}

function getStudentsNotInClass(class_id: number): StudentInfo[] {
    const sqlInClass = `
        SELECT Student.student_id FROM Student
            INNER JOIN Enrollment
                ON Student.student_id = Enrollment.student_id
            WHERE class_id = ?;
    `
    const stmtInClass = db.prepare(sqlInClass)
    const resInClass = stmtInClass.all(class_id)

    let listOfStudentIDsInClass = ''
    resInClass.forEach((stu, stuIndex) => {
        if (stuIndex === 0) {
            listOfStudentIDsInClass += stu.student_id
        } else {
            listOfStudentIDsInClass += "," + stu.student_id
        }
    })

    const sqlNotInClass = `
        SELECT Student.student_id, first_name, last_name FROM Student
            WHERE student_id NOT IN (${listOfStudentIDsInClass});
    `
    const stmtNotInClass = db.prepare(sqlNotInClass)
    const resNotInClass = stmtNotInClass.all()

    return resNotInClass
}



declare global {
    // Type declarations must be global for type-checking in other files

    interface ClassData {
        id: number,
        name: string,
        description: string,
        assignments: AssignmentInfo[],
        studentInfo: StudentGrades[]
    }
    interface StudentGrades {
        student_id: number,
        class_id: number,
        first_name: string,
        last_name: string,
        grades: GradeObject[]
    }
    interface GradeObject {
        student_id: number,
        assignment_id: number,
        earned_points: number,
        is_exempt: boolean
    }
    interface ClassInfo {
        class_id: number,
        name: string,
        description: string
    }

    interface class_func_exports {
        /**
         * Gets verbose information about a class.
         * @param class_id the ID of the class being requested.
         * @returns Object containing all data about the class, it's
assignments, it's students, and their grades.
         */
        getClassData: (class_id: number) => ClassData,
        /**
         * Creates a class in the database.
         * @param name The name of the class. Max 50 chars.
         * @param description A brief description of the class. Max 200 chars.
         * @returns The newly-created `class_id`.
         */
        createClass: (name: string, description: string) => number,
        /**
         * Deletes a class from the database.
         * @param class_id The ID of the class.
         */
        deleteClass: (class_id: number) => void,
        /**
         * Gets surface-level information about the class.
         * @param class_id The ID of the class.
         * @returns Object containing `class_id`, `name`, and `description`.
         */
        getClassInfo: (class_id: number) => ClassInfo,
        /**
         * Gets surface-level information about all classes
         * @returns Array of objects containing `class_id`, `name`, and
`description`.
         */
        getClassList: () => ClassInfo[],
        /**
         * Edit a class's name & description. You must provide both parameters,
so if
         * you only want to change the name, provide back the original
description
         * (and vice versa).
         * @param class_id The ID of the class.
         * @param name The new name for the class. Max 50 chars.
         * @param description The new description for the class. Max 200 chars.
         */
        editClass: (class_id: number, name?: string, description?: string) =>
void,
        /**
         * Gets students that are not in a class (for list)
         * @param class_id The ID of the class.
         * @returns An array of objects containing the id and names of
students.
         */
        getStudentsNotInClass: (class_id: number) => StudentInfo[]
    }
}

export default {
    getClassData,
    createClass,
    deleteClass,
    getClassInfo,
    getClassList,
    editClass,
    getStudentsNotInClass
} as class_func_exports
```

Database Functions – Enrollment

```
import db from './db_bridge'

function addEnrollment(class_id: number, student_id: number): void {
    const sqlInsert = `
        INSERT INTO Enrollment
            VALUES (?, ?);
    `
    const stmtInsert = db.prepare(sqlInsert)
    stmtInsert.run(class_id, student_id)

    // Add a grade of '0' for each assignment in the class to the new user.
    const sqlGetAssignmentsInClass = `
        SELECT assignment_id FROM Assignment WHERE class_id = ?;
    `
    const stmtGetAssignmentsInClass = db.prepare(sqlGetAssignmentsInClass)
    const assignmentsInClass = stmtGetAssignmentsInClass.all(class_id)

    const sqlInsertGrade = `INSERT INTO Grade VALUES (?, ?, 0, 0);`
    const stmtInsertGrades = db.prepare(sqlInsertGrade)

    assignmentsInClass.forEach(assignment => {
        stmtInsertGrades.run(student_id, assignment.assignment_id)
    })

}

function deleteEnrollment(class_id: number, student_id: number): void {
    const sql = `
        DELETE FROM Enrollment
            WHERE class_id = ?
            AND student_id = ?;
    `
    const stmt = db.prepare(sql)
    stmt.run(class_id, student_id)
}



declare global {
    interface enrollment_func_exports {
        /**
         * Adds an enrollment to the database.
         * @param class_id The ID of the class.
         * @param student_id The ID of the student.
         */
        addEnrollment: (class_id: number, student_id: number) => void,
        /**
         * Deletes an enrollment from the database.
         * @param class_id The ID of the class.
         * @param student_id The ID of the student.
         */
        deleteEnrollment: (class_id: number, student_id: number) => void
    }
}

export default {
    addEnrollment,
    deleteEnrollment
} as enrollment_func_exports
```

Database Functions – Grade

```
import db from './db_bridge'

function editGradePoints(student_id: number, assignment_id: number,
earned_points: number): void {
    const sql = `
        UPDATE Grade
        SET earned_points = ?
        WHERE student_id = ?
        AND assignment_id = ?;
    `
    const stmt = db.prepare(sql)
    stmt.run(earned_points, student_id, assignment_id)
}

function editGradeExempt(student_id: number, assignment_id: number, is_exempt: 
boolean): void {
    const sql = `
        UPDATE Grade
        SET is_exempt = ${is_exempt ? 1 : 0}
        WHERE student_id = ${student_id}
        AND assignment_id = ${assignment_id};
    `
    const stmt = db.prepare(sql)
    stmt.run(
        is_exempt ? 1 : 0,
        student_id,
        assignment_id
    )
}

function applyBulkChanges(changes: PendingChange[]) {
    const sqlChangePoints = `
        UPDATE Grade SET earned_points = ?
            WHERE student_id = ?
            AND assignment_id = ?;
    `
    const sqlChangeExempt = `
        UPDATE Grade SET is_exempt = ?
            WHERE student_id = ?
            AND assignment_id = ?;
    `
    const stmtChangePoints = db.prepare(sqlChangePoints)
    const stmtChangeExempt = db.prepare(sqlChangeExempt)

    changes.forEach(change => {
        if (change.newEarnedPoints !== undefined) {
            stmtChangePoints.run(
                change.newEarnedPoints,
                change.student_id,
                change.assignment_id
            )
        }
        if (change.newIsExempt !== undefined) {
            stmtChangeExempt.run(
                change.newIsExempt ? 1 : 0,
                change.student_id,
                change.assignment_id
            )
        }
    })
}



declare global {
    interface PendingChange {
        student_id: number,
        assignment_id: number,
        newEarnedPoints?: number,
        newIsExempt?: boolean
    }

    interface grade_func_exports {
        /**
         * Change the points earned on an assignment for a student.
         * @param student_id The ID of the student.
         * @param assignment_id The ID of the assignment.
         * @param earned_points New value for earned_points.
         */
        editGradePoints: (student_id: number, assignment_id: number,
earned_points: number) => void,
        /**
         * Edit the exempt status on an assignment for a student.
         * @param student_id The ID of the student.
         * @param assignment_id The ID of the assignment.
         * @param is_exempt Whether or not the grade is exempt.
         */
        editGradeExempt: (student_id: number, assignment_id: number, is_exempt:
boolean) => void,
        /**
         * Applies a series of changes to the database (specifically grades)
         * @param changes An array of `pendingChange` objects describing the
changes to make.
         */
        applyBulkChanges: (changes: PendingChange[]) => void
    }
}

export default {
    editGradePoints,
    editGradeExempt,
    applyBulkChanges
} as grade_func_exports
```

Database Functions – Student

```
import db from './db_bridge'

function getStudentInfo(student_id: number): StudentInfo {
    const sql = `
        SELECT * FROM Student WHERE student_id = ?;
    `
    const stmt = db.prepare(sql)
    const res: StudentInfo = stmt.get(student_id)

    return res
}

function deleteStudent(student_id: number): void {
    const sql = `
        DELETE FROM Student
        WHERE student_id = ?;
    `
    const stmt = db.prepare(sql)
    stmt.run(student_id)
}

function createStudent(first_name: string, last_name: string): number {
    const sql = `
        INSERT INTO Student (first_name, last_name)
        VALUES (?, ?);
    `
    const stmt = db.prepare(sql)
    const res = stmt.run(first_name, last_name)

    return res.lastInsertRowid as number
}
```

```
function editStudent(student_id: number, first_name: string, last_name:
string): void {
    const sql = `
        UPDATE Student
        SET first_name = ?,
            last_name = ?
        WHERE student_id = ?;
    `
    const stmt = db.prepare(sql)
    stmt.run(first_name, last_name, student_id)
}

function getStudentEnrollments(student_id: number): ClassInfo[] {
    const sql = `
        SELECT Class.class_id, name, description
            FROM Class INNER JOIN Enrollment
                ON Class.class_id = Enrollment.class_id
            WHERE student_id = ?;
    `
    const stmt = db.prepare(sql)
    const res: ClassInfo[] = stmt.all(student_id)

    return res
}

function getStudentList(): StudentInfo[] {
    const sql = `
        SELECT * FROM Student;
    `
    const stmt = db.prepare(sql)
    const res: StudentInfo[] = stmt.all()

    return res
}



declare global {
    interface StudentInfo {
        student_id: number,
        first_name: string,
        last_name: string
    }
    interface student_func_exports {
        /**
```

```
         * Get's an object containing the student's information.
         * @param student_id The ID of the student.
         * @returns object containing `student_id`, `first_name`, and
`last_name`.
         */
        getStudentInfo: (student_id: number) => StudentInfo,
        /**
         * Removes a student from the database, including enrollments and
grades.
         * @param student_id The ID of the student.
         */
        deleteStudent: (student_id: number) => void,
        /**
         * Creates a new student in the database.
         * @param first_name First name of the student. Max 25 chars.
         * @param last_name Last name of the student. Max 25 chars.
         * @returns The newly-created student's ID.
         */
        createStudent: (first_name: string, last_name: string) => number,
        /**
         * Edit a student's name.
         * @param student_id The ID of the student.
         * @param first_name New irst name of the student. Max 25 chars.
         * @param last_name New last name of the student. Max 25 chars.
         */
        editStudent: (student_id: number, first_name: string, last_name:
string) => void
        /**
         * Gets an array of classes the student is in.
         * @param student_id The ID of the student.
         * @returns An array of objects containing the class's IDs, names, and
descriptions.
         */
        getStudentEnrollments: (student_id: number) => ClassInfo[],
        /**
         * Gets an array of all students in the database.
         * @returns An array of objects containing the students's IDs and
names.
         */
        getStudentList: () => StudentInfo[]
    }
}

export default {
    getStudentInfo,
    deleteStudent,
    createStudent,
    editStudent,
    getStudentEnrollments,
    getStudentList
} as student_func_exports
```

Database Functions – Setup

```
import db from './db_bridge'

function generateTables() {

    const sql = `
        CREATE TABLE IF NOT EXISTS Class (
            class_id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT(50) NOT NULL,
            description TEXT(200)
        );

        CREATE TABLE IF NOT EXISTS Assignment (
            assignment_id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT(50) NOT NULL,
            description TEXT(200),
            assignment_type TEXT(8) NOT NULL,
            is_extra_credit INTEGER NOT NULL,
            max_points INTEGER NOT NULL,
            order_position INTEGER,
            class_id INTEGER NOT NULL,
            CONSTRAINT fk_class_id
                FOREIGN KEY (class_id)
                REFERENCES Class(class_id)
                ON DELETE CASCADE
        );

        CREATE TABLE IF NOT EXISTS Student (
            student_id INTEGER PRIMARY KEY AUTOINCREMENT,
            first_name TEXT(25) NOT NULL,
            last_name TEXT(25) NOT NULL
        );

        CREATE TABLE IF NOT EXISTS Grade (
            student_id INTEGER,
            assignment_id INTEGER,
            earned_points INTEGER NOT NULL,
            is_exempt INTEGER NOT NULL,
            CONSTRAINT fk_student_id
                FOREIGN KEY (student_id)
                REFERENCES Student(student_id)
                ON DELETE CASCADE,
            CONSTRAINT fk_assignment_id
                FOREIGN KEY (assignment_id)
                REFERENCES Assignment(assignment_id)
                ON DELETE CASCADE,
            PRIMARY KEY (student_id, assignment_id)
        );

        CREATE TABLE IF NOT EXISTS Enrollment (
            class_id INTEGER,
            student_id INTEGER,
            CONSTRAINT fk_class_id
                FOREIGN KEY (class_id)
                REFERENCES Class(class_id)
                ON DELETE CASCADE,
            CONSTRAINT fk_student_id
                FOREIGN KEY (student_id)
                REFERENCES Student(student_id)
                ON DELETE CASCADE,
            PRIMARY KEY (class_id, student_id)
        );`

    db.exec(sql)
}

// for debug purposes
function dropTables() {
    const sql = `
        DROP TABLE IF EXISTS Enrollment;
        DROP TABLE IF EXISTS Grade;
        DROP TABLE IF EXISTS Student;
        DROP TABLE IF EXISTS Class;
        DROP TABLE IF EXISTS Assignment
    `

    db.exec(sql)
}



declare global {
    interface setup_func_exports {
        /**
         * Creates all tables in the database.
         */
        generateTables: () => void,
        /**
         * Drops all tables in the database.
         * Intended for debug purposes
         */
        dropTables: () => void
    }
}

export default {
    generateTables,
    dropTables
} as setup_func_exports
```

Index.ts (Electron Main Process)

```
import { app, BrowserWindow } from 'electron';
// This allows TypeScript to pick up the magic constants that's auto-generated
by Forge's Webpack
// plugin that tells the Electron app where to look for the Webpack-bundled app
code (depending on
// whether you're running in development or production).
declare const MAIN_WINDOW_WEBPACK_ENTRY: string;
declare const MAIN_WINDOW_PRELOAD_WEBPACK_ENTRY: string;

// Handle creating/removing shortcuts on Windows when installing/uninstalling.
if (require('electron-squirrel-startup')) {
    app.quit();
}

const createWindow = (): void => {
    // Create the browser window.
    const mainWindow = new BrowserWindow({
        height: 600,
        width: 800,
        webPreferences: {
            preload: MAIN_WINDOW_PRELOAD_WEBPACK_ENTRY,
        },
    });

    // and load the index.html of the app.
    mainWindow.loadURL(MAIN_WINDOW_WEBPACK_ENTRY);
    mainWindow.removeMenu()

    // Open the DevTools.
    // mainWindow.webContents.openDevTools();
};

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.on('ready', createWindow);

// Quit when all windows are closed, except on macOS. There, it's common
// for applications and their menu bar to stay active until the user quits
// explicitly with Cmd + Q.
app.on('window-all-closed', () => {
    if (process.platform !== 'darwin') {
        app.quit();
    }
});

app.on('activate', () => {
    // On OS X it's common to re-create a window in the app when the
    // dock icon is clicked and there are no other windows open.
    if (BrowserWindow.getAllWindows().length === 0) {
        createWindow();
    }
});

// In this file you can include the rest of your app's specific main process
// code. You can also put them in separate files and import them here.

import { ipcMain } from 'electron'



import db from './db'


db.setup.generateTables()


app.on('ready', () => {
    ipcMain.handle('closeApp', () => app.quit())
    ipcMain.handle('dbtest', dbtest)

    ipcMain.handle('db-setup-generateTables', db.setup.generateTables)
    ipcMain.handle('db-setup-dropTables', db.setup.dropTables)

    ipcMain.handle('db-test-insertTestData', db.test.insertTestData)

    ipcMain.handle('db-class-getClassData', (e, class_id: number) =>
db.class.getClassData(class_id))
    ipcMain.handle('db-class-createClass',
        (e, name: string, description: string) => db.class.createClass(name,
description)
    )
    ipcMain.handle('db-class-deleteClass', (e, class_id: number) =>
db.class.deleteClass(class_id))
    ipcMain.handle('db-class-getClassInfo', (e, class_id: number) =>
db.class.getClassInfo(class_id))
    ipcMain.handle('db-class-getClassList', () => db.class.getClassList())
    ipcMain.handle('db-class-editClass',
        (e, class_id: number, name: string, description: string) =>
db.class.editClass(class_id, name, description)
    )
    ipcMain.handle('db-class-getStudentsNotInClass',
        (e, class_id: number) => db.class.getStudentsNotInClass(class_id)
    )

    ipcMain.handle('db-student-getStudentInfo', (e, student_id: number) =>
db.student.getStudentInfo(student_id))
    ipcMain.handle('db-student-deleteStudent', (e, student_id: number) =>
db.student.deleteStudent(student_id))
    ipcMain.handle('db-student-createStudent',
        (e, first_name: string, last_name: string) =>
db.student.createStudent(first_name, last_name)
    )
    ipcMain.handle('db-student-editStudent',
        (e, student_id: number, first_name: string, last_name: string) =>
            db.student.editStudent(student_id, first_name, last_name)
    )
    ipcMain.handle('db-student-getStudentEnrollments',
        (e, student_id: number) => db.student.deleteStudent(student_id)
    )
    ipcMain.handle('db-student-getStudentList',
        () => db.student.getStudentList()
    )
    ipcMain.handle('db-assignment-createAssignment',
        (e,
            class_id: number,
            name: string,
            description: string,
            assignment_type: "HOMEWORK" | "TEST",
            is_extra_credit: boolean,
            max_points: number
        ) =>  db.assignment.createAssignment(
            class_id, name, description, assignment_type, is_extra_credit,
max_points
        )
    )
    ipcMain.handle('db-assignment-editAssignment',
    (e,
        assignment_id: number,
        name: string,
        description: string,
        assignment_type: "HOMEWORK" | "TEST",
        is_extra_credit: boolean,
        max_points: number
    ) =>  db.assignment.editAssignment(
        assignment_id, name, description, assignment_type, is_extra_credit,
max_points
    ))
    ipcMain.handle('db-assignment-deleteAssignment',
        (e, assignment_id: number) =>
db.assignment.deleteAssignment(assignment_id)
    )
    ipcMain.handle('db-assignment-getAssignment',
        (e, assignment_id: number) =>
db.assignment.getAssignment(assignment_id)
    )
    ipcMain.handle('db-assignment-updateAssignmentOrder',
        (e, assignment_id: number, order_position: number) =>
            db.assignment.updateAssignmentOrder(assignment_id, order_position)
    )

    ipcMain.handle('db-enrollment-addEnrollment',
        (e, class_id: number, student_id: number) =>
            db.enrollment.addEnrollment(class_id, student_id)
    )
    ipcMain.handle('db-enrollment-deleteEnrollment',
        (e, class_id: number, student_id: number) =>
            db.enrollment.deleteEnrollment(class_id, student_id)
    )

    ipcMain.handle('db-grade-editGradePoints',
        (e, student_id: number, assignment_id: number, earned_points: number)
=>
            db.grade.editGradePoints(student_id, assignment_id, earned_points)
    )
    ipcMain.handle('db-grade-editGradeExempt',
        (e, student_id: number, assignment_id: number, is_exempt: boolean) =>
            db.grade.editGradeExempt(student_id, assignment_id, is_exempt)
    )
    ipcMain.handle('db-grade-applyBulkChanges',
        (e, changes: PendingChange[]) =>
            db.grade.applyBulkChanges(changes)
    )

})
```

Index.scss (Some global styles)

```
/* https://coolors.co/5f5449-9b6a6c-b09398-cedfd9-ebfcfb */
@import 'colors';

body {

    font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
Helvetica,
        Arial, sans-serif;
    margin: auto;
    padding: 20px;
    background-color: $background;
    color: $background-text;
    height: calc(100vh - 40px);
}
#root {
    height: 100%;
}

button {
    display: flex;
    justify-content: space-between;
    align-items: center;
    width: fit-content;
    padding: 5px 10px;
    height: 40px;
    outline: none;

    background-color: $accent;
    border: 2px solid $color4;
    border-radius: 5px;
    color: $accent-text;
    cursor: pointer;

    font-size: 16px;
    font-weight: 700;

    transition: 0.2s ease background-color;
    transition: 0.2s ease scale;



    &:hover,
    &:focus {
        background-color: $accent-lighter;
        scale: 105%;
    }

    &:active {
        scale: 96%;
    }

    &>.icon {
        font-size: 24px;
    }

    &>.label {
        transform: translateY(1px)
    }

    &.addstudent {
        width: 154px;
    }
}
```

Preload.ts (IPC Bridge)

```
// See the Electron documentation for details on how to use preload scripts:
```

```
// https://www.electronjs.org/docs/latest/tutorial/process-model#preload-scripts
import { contextBridge, ipcRenderer } from 'electron'

contextBridge.exposeInMainWorld('app', {
    closeApp: () => ipcRenderer.invoke('closeApp')
} as Window['app'])

contextBridge.exposeInMainWorld('api', {
    // An example of a function that uses the ipc context bridge.
    // The contextBridge.exposeInMainWorld will make these functions accessible to the renderer.
    // The ipcRenderer.invoke grabs the function from the main file (index.ts).
    dbtest: () => ipcRenderer.invoke('dbtest')
} as Window['api'])

contextBridge.exposeInMainWorld('setup', {
    generateTables: () => ipcRenderer.invoke('db-setup-generateTables'),
    dropTables: () => ipcRenderer.invoke('db-setup-dropTables')
} as Window['setup'])

contextBridge.exposeInMainWorld('test', {
    insertTestData: () => ipcRenderer.invoke('db-test-insertTestData')
} as Window['test'])

contextBridge.exposeInMainWorld('class', {
    getClassData: (class_id) => ipcRenderer.invoke('db-class-getClassData', class_id),
    createClass: (name, description) => ipcRenderer.invoke('db-class-createClass', name, description),
    deleteClass: (class_id) => ipcRenderer.invoke('db-class-deleteClass', class_id),
    getClassInfo: (class_id) => ipcRenderer.invoke('db-class-getClassInfo', class_id),
    getClassList: () => ipcRenderer.invoke('db-class-getClassList'),
    editClass: (class_id, name, description) => ipcRenderer.invoke('db-class-editClass', class_id, name, description),
    getStudentsNotInClass: (class_id) => ipcRenderer.invoke('db-class-getStudentsNotInClass', class_id)
} as Window['class'])

contextBridge.exposeInMainWorld('student', {
    getStudentInfo: (student_id) => ipcRenderer.invoke('db-student-getStudentInfo', student_id),
    deleteStudent: (student_id) => ipcRenderer.invoke('db-student-
deleteStudent', student_id),
    createStudent: (first_name, last_name) => ipcRenderer.invoke('db-student-
createStudent', first_name, last_name),
    editStudent: (student_id, first_name, last_name) =>
        ipcRenderer.invoke('db-student-editStudent', student_id, first_name,
last_name),
    getStudentEnrollments: (student_id) => ipcRenderer.invoke('db-student-
getStudentEnrollments', student_id),
    getStudentList: () => ipcRenderer.invoke('db-student-getStudentList'),
} as Window['student'])

contextBridge.exposeInMainWorld('assignment', {
    createAssignment: (class_id, name, description, assignment_type,
is_extra_credit, max_points) =>
        ipcRenderer.invoke('db-assignment-createAssignment',
            class_id, name, description, assignment_type, is_extra_credit,
max_points
        ),
    editAssignment: (assignment_id, name, description, assignment_type,
is_extra_credit, max_points) =>
        ipcRenderer.invoke('db-assignment-editAssignment',
            assignment_id, name, description, assignment_type, is_extra_credit,
max_points
        ),
    deleteAssignment: (assignment_id) => ipcRenderer.invoke('db-assignment-
deleteAssignment', assignment_id),
    getAssignment: (assignment_id) => ipcRenderer.invoke('db-assignment-
getAssignment', assignment_id),
    updateAssignmentOrder: (assignment_id, order_position) =>
        ipcRenderer.invoke('db-assignment-updateAssignmentOrder',
assignment_id, order_position),
} as Window['assignment'])

contextBridge.exposeInMainWorld('enrollment', {
    addEnrollment: (class_id, student_id) => ipcRenderer.invoke('db-enrollment-
addEnrollment', class_id, student_id),
    deleteEnrollment: (class_id, student_id) => ipcRenderer.invoke('db-
enrollment-deleteEnrollment', class_id, student_id)
} as Window['enrollment'])

contextBridge.exposeInMainWorld('grade', {
    editGradePoints: (student_id, assignment_id, earned_points) =>
        ipcRenderer.invoke('db-grade-editGradePoints', student_id,
assignment_id, earned_points),
    editGradeExempt: (student_id, assignment_id, is_exempt) =>
        ipcRenderer.invoke('db-grade-editGradeExempt', student_id,
assignment_id, is_exempt),
    applyBulkChanges: (changes) =>
        ipcRenderer.invoke('db-grade-applyBulkChanges', changes),
} as Window['grade'])
```

Window.d.ts (Interface for IPC Bridge functions, defines types and adds JSDocs)

```
interface Window {

    /**
     * App-related functions
     */
    app: {
        /**
         * Closes out of the app
         */
        closeApp: () => Promise<void>
    }
    api: {
        /**
         * Returns an promise, then an object containing the class id, name,
description, and student info.
         * @param class_id the ID of the class requesting
         * @returns Object containing all data about the class, it's students,
and their grades.
         */
        dbtest: () => Promise<ClassData>
    }
    /**
     * Contains functions relating to setting up the database tables.
     */
    setup: {
        /**
         * Creates all tables in the database.
         */
        generateTables: () => Promise<void>,
        /**
         * Drops all tables in the database.
         * Intended for debug purposes
         */
        dropTables: () => Promise<void>
    }
    /**
     * Contains functions for testing purposes, not used in production.
     */
    test: {
        /**
         * Inserts some testing data into the database.
         * Debug purposes only.
         */
        insertTestData: () => Promise<void>
    }
    /**
     * Contains functions for interacting with classes in the database.
     */
    class: {
        /**
         * Gets verbose information about a class.
         * @param class_id the ID of the class being requested.
         * @returns Object containing all data about the class, it's 
assignments, it's students, and their grades.
         */
        getClassData: (class_id: number) => Promise<ClassData>,
        /**
         * Creates a class in the database.
         * @param name The name of the class. Max 50 chars.
         * @param description A brief description of the class. Max 200 chars.
         * @returns The newly-created `class_id`.
         */
        createClass: (name: string, description: string) => Promise<number>,
        /**
         * Deletes a class from the database.
         * @param class_id The ID of the class.
         */
        deleteClass: (class_id: number) => Promise<void>,
        /**
         * Gets surface-level information about the class.
         * @param class_id The ID of the class.
         * @returns Object containing `class_id`, `name`, and `description`.
         */
        getClassInfo: (class_id: number) => Promise<ClassInfo>,
        /**
         * Gets surface-level information about all classes
         * @returns Array of objects containing `class_id`, `name`, and 
`description`.
```

```
         */
        getClassList: () => Promise<ClassInfo[]>,
        /**
         * Edit a class's name & description. You must provide both parameters,
so if
         * you only want to change the name, provide back the original
description
         * (and vice versa).
         * @param class_id The ID of the class.
         * @param name The new name for the class. Max 50 chars.
         * @param description The new description for the class. Max 200 chars.
         */
        editClass: (class_id: number, name?: string, description?: string) =>
Promise<void>,
        /**
         * Gets students that are not in a class (for list)
         * @param class_id The ID of the class.
         * @returns An array of objects containing the id and names of
students.
         */
        getStudentsNotInClass: (class_id: number) => Promise<StudentInfo[]>
    }
    /**
     * Contains functions for interacting with students in the database.
     */
    student: {
        /**
         * Get's an object containing the student's information.
         * @param student_id The ID of the student.
         * @returns object containing `student_id`, `first_name`, and
`last_name`.
         */
        getStudentInfo: (student_id: number) => Promise<StudentInfo>,
        /**
         * Removes a student from the database, including enrollments and
grades.
         * @param student_id The ID of the student.
         */
        deleteStudent: (student_id: number) => Promise<void>,
        /**
         * Creates a new student in the database.
         * @param first_name First name of the student. Max 25 chars.
         * @param last_name Last name of the student. Max 25 chars.
         * @returns The newly-created student's ID.
         */
        createStudent: (first_name: string, last_name: string) =>
Promise<number>,
        /**
         * Edit a student's name.
         * @param student_id The ID of the student.
         * @param first_name New irst name of the student. Max 25 chars.
         * @param last_name New last name of the student. Max 25 chars.
         */
        editStudent: (student_id: number, first_name: string, last_name:
string) => Promise<void>
        /**
         * Gets an array of classes the student is in.
         * @param student_id The ID of the student.
         * @returns An array of objects containing the class's IDs, names, and
descriptions.
         */
        getStudentEnrollments: (student_id: number) => Promise<ClassInfo[]>,
        /**
         * Gets an array of all students in the database.
         * @returns An array of objects containing the students's IDs and
names.
         */
        getStudentList: () => Promise<StudentInfo[]>
    }
    /**
     * Contains functions for interacting with assignments in the database.
     */
    assignment: {
        /**
         * Creates an assignment in the database.
         * @param class_id The ID of the class associated.
         * @param name The name of the assignment.
         * @param description The description of the assignment (can be blank,
must be provided).
         * @param assignment_type The type of assignment. "HOMEWORK" or "TEST".
         * @param is_extra_credit boolean, whether the assignment should impact
grade negatively.
         * @param max_points Total points the assignment is worth.
         * @returns The ID of the newly-created class.
         */
        createAssignment: (
            class_id: number,
            name: string,
            description: string,
            assignment_type: AssignmentInfo['assignment_type'],
            is_extra_credit: boolean,
            max_points: number
        ) => Promise<number>,
        /**
         * Edit an assignment in the database.
         * @param assignment_id The ID of the assignment.
         * @param name The name of the assignment.
         * @param description The description of the assignment (can be blank,
must be provided).
         * @param assignment_type The type of assignment. "HOMEWORK" or "TEST".
         * @param is_extra_credit boolean, whether the assignment should impact
grade negatively.
         * @param max_points Total points the assignment is worth.
         */
        editAssignment: (
            assignment_id: number,
            name: string,
            description: string,
            assignment_type: AssignmentInfo['assignment_type'],
            is_extra_credit: boolean,
            max_points: number
        ) => Promise<void>,
        /**
         * Delete an assignment from the database.
         * @param assignment_id The ID of the assignment.
         */
        deleteAssignment: (assignment_id: number) => Promise<void>,
        /**
         * Get an assignment's data from the database
         * @param assignment_id The ID of the assignment.
         * @returns An object containing all of the assignment's data.
         */
        getAssignment: (assignment_id: number) => Promise<AssignmentInfo>,
        /**
         * Updates the display order for the assignment in the class.
         * @param assignment_id The ID of the assignment.
         * @param order_position The display order for the assignment.
         */
        updateAssignmentOrder: (assignment_id: number, order_position: number)
=> Promise<void>
    }
    /**
     * Contains functions for interacting with enrollments in the database.
     */
    enrollment: {
        /**
         * Adds an enrollment to the database.
         * @param class_id The ID of the class.
         * @param student_id The ID of the student.
         */
        addEnrollment: (class_id: number, student_id: number) => Promise<void>,
        /**
         * Deletes an enrollment from the database.
         * @param class_id The ID of the class.
         * @param student_id The ID of the student.
         */
        deleteEnrollment: (class_id: number, student_id: number) =>
Promise<void>
    }
    /**
     * Contains functions for updating student grades in the database.
     */
    grade: {
        /**
         * Change the points earned on an assignment for a student.
         * @param student_id The ID of the student.
         * @param assignment_id The ID of the assignment.
         * @param earned_points New value for earned_points.
         */
        editGradePoints: (student_id: number, assignment_id: number,
earned_points: number) => Promise<void>,
        /**
         * Edit the exempt status on an assignment for a student.
         * @param student_id The ID of the student.
         * @param assignment_id The ID of the assignment.
         * @param is_exempt Whether or not the grade is exempt.
         */
        editGradeExempt: (student_id: number, assignment_id: number, is_exempt:
boolean) => Promise<void>,
        /**
         * Applies a series of changes to the database (specifically grades)
         * @param changes An array of `pendingChange` objects describing the
changes to make.
         */
        applyBulkChanges: (changes: PendingChange[]) => Promise<void>
    }
}
```

_colors.scss (stores variables for CSS)

```scss
// https://coolors.co/bbdb9b-abc4a1-9db4ab-8d9d90-878e76

$background: #878E76;
$background-text: white;

$color1: #BBDB9B;
$color1-text: black;

$color2: #ABC4A1;
$color2-text: black;
$color2-lighter: #C7D8C0;
$color2-lighter2: #dfebdb;
$color2-darker: #9db493;

$color3: #9DB4AB;
$color3-darker: #668579;
$color3-text: black;

$color4: #8D9D90;
$color4-text: white;

$accent: #edec8f;
$accent-disabled: #c5c497;
$accent-lighter: #f0efc7;
$accent-text: #61603a;

$error-border: rgb(255, 79, 79);
$error-fill: rgb(255, 214, 214);
$error-text: red;

// $background: #FFD972;
// $background-text: black;

// $color1: #C7EAE4;
// $color1-text: black;

// $color2: #A7E8BD;
// $color2-text: black;
// $color2-lighter: #DFF7E7;

// $color3: #FCBCB8;
// $color3-text: black;
// $color4: #EFA7A7;
// $color4-text: black;
```

App.tsx (Controls classes and class switches)

```
import React from 'react'
import { ClassTable } from './ClassTable/ClassTable'
import './app.scss'
import newClassPopup from './PopupLib/newClassPopup'
import popup from '../Popup/popup'
import studentManagerPopup from './PopupLib/studentManagerPopup'



export default function App() {

    // An example of a function that uses the ipc context bridge.
    // This function, example(), is running from the main file, and has access
to node modules.
    const [currentClass, setCurrentClass] = React.useState<number>()
    const [currentClassInfo, setCurrentClassInfo] = React.useState<ClassInfo>()
    const [classList, setClassList] = React.useState<ClassInfo[]>()
    const [isLoaded, setIsLoaded] = React.useState<boolean>(false)
    const [newClassCreated, setNewClassCreated] =
React.useState<boolean>(false)
    const selectRef = React.useRef<HTMLSelectElement>()

    React.useEffect(() => {
        updateClassList()
    }, [])

    function updateClassList(): Promise<void> {
        return new Promise((resolve, reject) => {
            window.class.getClassList()
                .then(res => {

                    if(res.length === 0) {
                        newClassPopup.trigger()
                            .then(class_id => {
                                updateClassList().then(() => {
                                    setCurrentClass(class_id)
                                    setNewClassCreated(true)
                                    updateClassInfo(class_id)
                                })
                            })
                            .catch(() => {
                                // If user closes out of popup to create FIRST
class, exit out of application
                                window.app.closeApp()
                            })
                    }

                    setClassList(res)
                    setCurrentClass(res[0].class_id)
                    setIsLoaded(true)
                    resolve()
                    updateClassInfo(res[0].class_id)
                })
                .catch (() => reject())
        })
    }

    function updateClassInfo(class_id: number) {
        window.class.getClassInfo(class_id)
                .then(res => {
                    setCurrentClassInfo(res)
                })
    }

    let optionsDisplay
    if (classList) {
        optionsDisplay = classList.map(cls => {
            return (
                <option value={cls.class_id} key={cls.class_id}>
                    {cls.name}
                </option>
            )
        })
    }

    React.useEffect(() => {
        if (newClassCreated) {
            selectRef.current.selectedIndex = selectRef.current.options.length
- 2
            setNewClassCreated(false)
        }
    }, [newClassCreated])

    function handleClassChange(e: React.ChangeEvent<HTMLSelectElement>) {
        const selectNode = e.target as HTMLSelectElement
        const class_id = selectNode.value

        if (class_id === "create_new_class") {
            selectNode.value = currentClass.toString()
            newClassPopup.trigger()
                .then(class_id => {

                    updateClassList().then(() => {
                        setCurrentClass(class_id)
                        selectNode.value = class_id.toString()
                        setNewClassCreated(true)
                        updateClassInfo(class_id)
                    })
                })
        } else {
            setCurrentClass(parseInt(e.target.value))
            updateClassInfo(parseInt(e.target.value))
        }
    }

    function handleDelete() {
        popup.triggerPopup(
            <div className='delete_confirm'>
                <h3>Are you sure you'd like to delete this class?</h3>
                <div className='button_container'>
                    <button onClick={handleDeny}>No, take me back</button>
                    <button onClick={handleConfirm} className='delete'>Yes,
delete</button>
                </div>
            </div>
        , "warning")

        function handleDeny() {
            popup.closePopup()
        }
        function handleConfirm(){
            window.class.deleteClass(currentClass)
                .then(() => {
                    popup.closePopup()
                    selectRef.current.selectedIndex = 0
                    updateClassList()
                    setCurrentClass(parseInt(selectRef.current.value))
                })
        }
    }



    // descriptionContent defaults to "" if not provided
    let descriptionContent = ""
    if (currentClassInfo) {
        if (currentClassInfo.description) {
            descriptionContent = currentClassInfo.description
        } else {
            descriptionContent = ""
        }
    }

    function handleStudentManager() {
        studentManagerPopup.trigger()
    }

    return (<>
        {!isLoaded ? <h2>Loading...</h2> : <>
            <div className='classinfo_container'>
                <h2>Current Class: </h2>
                <select onChange={handleClassChange} ref={selectRef}>
                    {optionsDisplay}
                    <option value="create_new_class">+ Create New
Class</option>
                </select>
                <p>{descriptionContent}</p>
                <button className='delete' onClick={handleDelete}>Delete
Class</button>
                <button className='student_manager'
onClick={handleStudentManager}>Student Manager</button>
            </div>

            <ClassTable class_id={currentClass} />
        </>}
    </>)
}
```

App.scss (CSS sheet for styling the App)

```
@import '../colors';
.classinfo_container {
    display: flex;
    flex-direction: row;
    align-items: center;

    h2 {
        margin: 0;
        width: 170px;
        flex-shrink: 0;
    }
    select {
        height: 40px;
        font-size: 14px;
        max-width: 300px;
        flex-shrink: 0;
    }
    p {
        padding: 0px 10px;
        flex-shrink: 1;
    }
    button {
        flex-shrink: 0;
        margin-left: 15px;

        &.delete {
            color: $error-text;
        }
    }
    .delete_confirm {
        button {
            text-align: center;
            display: block;

            &.delete {
                margin-left: auto;
                color: $error-text;
            }
        }
    }
}
```

ClassTable.tsx (Main table display, and lower bar buttons)

```
import React from 'react'
import './ClassTable.scss'
import popup from '../../Popup/popup'
import addStudentPopup from '../PopupLib/addStudentPopup'
import editEnrollmentPopup from '../PopupLib/editEnrollmentPopup'
import newAssignmentPopup from '../PopupLib/newAssignmentPopup'
import editAssignmentPopup from '../PopupLib/editAssignmentPopup'

interface ClassTableProps {
    class_id: number
}


export function ClassTable(props: ClassTableProps) {

    const [classData, setClassData] = React.useState<ClassData>(null)
    const [asgnViewType, setAsgnViewType] = React.useState<"BOTH" | "HOMEWORK"
| "TEST">("BOTH")
    const gradeRefs = React.useRef<HTMLTableCellElement[]>([])
    const rowRefs = React.useRef<HTMLTableRowElement[]>([])

    React.useEffect(() => {
        updateClassData()
    }, [props.class_id])

    function updateClassData() {
        window.class.getClassData(props.class_id)
            .then(res => {
                setClassData(res)
            })
    }

    let tableIsEmpty = false
    if (classData) {
        if (classData.assignments.length === 0 && classData.studentInfo.length
=== 0) {
            tableIsEmpty = true
        }
    }

    // Set the buttonContainer's width to equal the table's width, for a
cleaner looking UI.
    // Unfortunantly, couldn't find a way to do this is pure CSS.
    const tableRef = React.useRef<HTMLTableElement>()
    const buttonContainerRef = React.useRef<HTMLDivElement | null>()
    React.useEffect(() => {
        if (buttonContainerRef.current && tableRef.current) {
            buttonContainerRef.current.style.width =
tableRef.current.clientWidth.toString() + 'px'
        }
        rowRefs.current.forEach(row => {
            updateTotal(parseInt(row.dataset.rownum))
        })
    }, [classData])



    /*  When a grade is changed, state is NOT updated. This is meant to
minimize re-renders, as well as
    allow the user to back out of changes. This function will add the change to
a `PendingChange[]` array,
    and will only apply changes to the database when "Save" is pressed.
    */
    let pendingChanges: PendingChange[] = []
    function handleGradeChange(e: React.ChangeEvent<HTMLInputElement>) {
        const gradeNode = e.target.parentElement.parentElement
        const percentNode = e.target.parentElement.childNodes[3] as
HTMLSpanElement
        const inputNode = e.target.parentElement.firstChild as HTMLInputElement
        const newGradeInt = parseInt(inputNode.value)
        const student_id = parseInt(inputNode.dataset.student_id)
        const assignment_id = parseInt(inputNode.dataset.assignment_id)
        const rowNum = parseInt(inputNode.dataset.rownum)
        const siblingExemptNode = inputNode.parentElement.childNodes[4] as
HTMLSpanElement

        const maxPoints =
parseInt(gradeNode.childNodes[0].childNodes[2].nodeValue)

        if (isNaN(newGradeInt) ||
            newGradeInt.toString() != inputNode.value.trim() ||
            inputNode.dataset.previousvalidinput === inputNode.value
        ) {
            // reset value to previous valid input if invalid
            inputNode.value = inputNode.dataset.previousvalidinput

        } else {
            if (inputNode.value !== inputNode.defaultValue) {
                gradeNode.classList.add('pending_change')
            } else {
                // check to make sure there isn't changes already pending from
exemptFlag
```

```
                if(siblingExemptNode.dataset.defaultstatus ===
siblingExemptNode.classList[1]) {
                    gradeNode.classList.remove('pending_change')
                }

            }

            inputNode.value = newGradeInt.toString()
            inputNode.dataset.previousvalidinput = newGradeInt.toString()
            pendingChanges.push(
                { student_id, assignment_id, newEarnedPoints: newGradeInt }
            )
            const undoBtn = buttonContainerRef.current.childNodes[2] as
HTMLButtonElement
            const saveBtn = buttonContainerRef.current.childNodes[3] as
HTMLButtonElement

            undoBtn.classList.remove('disabled')
            saveBtn.classList.remove('disabled')



            // Update the percentage shown
            percentNode.innerText = Math.round((newGradeInt / maxPoints) *
1000) / 10 + '%'
            updateTotal(rowNum)
        }
    }

    function handleAsgnViewChange(e: React.ChangeEvent<HTMLSelectElement>) {
        setAsgnViewType(e.target.value as "BOTH" | "HOMEWORK" | "TEST")
    }

    async function openAsgnSettings(assignment_id: number) {
        try {
            await promptSaveIfPendingChanges()
            await editAssignmentPopup.trigger(assignment_id)
            updateClassData()
        } catch {return}
    }

    async function handleStudentClick(student_id: number) {
        try {
            await promptSaveIfPendingChanges()
            await editEnrollmentPopup.trigger(student_id, props.class_id)
            updateClassData()
        } catch {return}
    }

    function handleExemptChange(e: React.MouseEvent<HTMLSpanElement>) {
        const spanNode = e.target as HTMLSpanElement
        const gradeNode = spanNode.parentElement.parentElement
        const student_id = parseInt(spanNode.dataset.student_id)
        const assignment_id = parseInt(spanNode.dataset.assignment_id)
        const rowNum = parseInt(spanNode.dataset.rownum)
        const siblingInputNode = spanNode.parentNode.childNodes[0] as
HTMLInputElement


        if (spanNode.className === "exempt disabled") {

            // if this is a changed value
            if (spanNode.dataset.defaultstatus === "disabled") {
                // AND the sibling input isn't changed
                if (siblingInputNode.defaultValue === siblingInputNode.value) {
                    gradeNode.classList.add('pending_change')
                }
            // if this is changing back to default value
            } else {
                // AND the sibling input isn't changed
                if (siblingInputNode.defaultValue === siblingInputNode.value) {
                    gradeNode.classList.remove('pending_change')
                }
            }

            spanNode.className = "exempt enabled"
            spanNode.title = "Remove Exemption"
            spanNode.innerText = "\u2691"
            pendingChanges.push(
                { student_id, assignment_id, newIsExempt: true }
            )

            const undoBtn = buttonContainerRef.current.childNodes[2] as
HTMLButtonElement
            const saveBtn = buttonContainerRef.current.childNodes[3] as
HTMLButtonElement

            undoBtn.classList.remove('disabled')
            saveBtn.classList.remove('disabled')
        } else {

            // if this is a changed value
            if (spanNode.dataset.defaultstatus === "enabled") {
                // AND the sibling input isn't changed
                if (siblingInputNode.defaultValue === siblingInputNode.value) {
                    gradeNode.classList.add('pending_change')
                }
            // if this is changing back to default value
            } else {
                // AND the sibling input isn't changed
                if (siblingInputNode.defaultValue === siblingInputNode.value) {
                    gradeNode.classList.remove('pending_change')
                }
            }

            spanNode.className = "exempt disabled"
            spanNode.title = "Make Exempt"
            spanNode.innerText = "\u2690"
            pendingChanges.push(
                { student_id, assignment_id, newIsExempt: false }
            )

            const undoBtn = buttonContainerRef.current.childNodes[2] as
HTMLButtonElement
            const saveBtn = buttonContainerRef.current.childNodes[3] as
HTMLButtonElement

            undoBtn.classList.remove('disabled')
            saveBtn.classList.remove('disabled')
        }
        updateTotal(rowNum)


    }

    function handleSaveChanges(e?: React.MouseEvent<HTMLButtonElement>) {

        // lose focus on button after select/enter. Better visually, and still
allows for keyboard control
        if (e) {
            let node = e.target as HTMLElement
            if (node.className === "undo_changes") {
                node = node as HTMLButtonElement
            } else {
                node = node.parentNode as HTMLButtonElement
            }
            node.blur()
        }


        if (pendingChanges.length !== 0) {
            window.grade.applyBulkChanges(pendingChanges)
            updateClassData()

            const allgradeNodes = gradeRefs.current
            allgradeNodes.forEach(gradeNode => {
                gradeNode.classList.remove('pending_change')

            })


            pendingChanges = []

            const undoBtn = buttonContainerRef.current.childNodes[2] as
HTMLButtonElement
            const saveBtn = buttonContainerRef.current.childNodes[3] as
HTMLButtonElement

            undoBtn.classList.add('disabled')
            saveBtn.classList.add('disabled')
        }
    }

    function handleUndoChanges(e?: React.MouseEvent<HTMLButtonElement>) {

        // lose focus on button after select/enter. Better visually, and still
allows for keyboard control
        if (e) {
            let node = e.target as HTMLElement
            if (node.className === "undo_changes") {
                node = node as HTMLButtonElement
            } else {
                node = node.parentNode as HTMLButtonElement
            }
            node.blur()
        }


        if (pendingChanges.length !== 0) {
            const allgradeNodes = gradeRefs.current
            allgradeNodes.forEach(gradeNode => {
                const percentNode = gradeNode.childNodes[0].childNodes[3] as
HTMLSpanElement
                const inputNode = gradeNode.childNodes[0].firstChild as
HTMLInputElement
                const oldValue = parseInt(inputNode.defaultValue)

                gradeNode.classList.remove('pending_change')
                const maxPoints =
parseInt(gradeNode.childNodes[0].childNodes[2].nodeValue)
                inputNode.value = inputNode.defaultValue

                // Update the percentage shown
                percentNode.innerText = Math.round((oldValue / maxPoints) *
1000) / 10 + '%'

                pendingChanges = []

                const undoBtn = buttonContainerRef.current.childNodes[2] as
HTMLButtonElement
                const saveBtn = buttonContainerRef.current.childNodes[3] as
HTMLButtonElement

                undoBtn.classList.add('disabled')
                saveBtn.classList.add('disabled')
            })

        }
    }


    async function handleAddStudent(e: React.MouseEvent<HTMLButtonElement>) {
        e.preventDefault()
        try {
            await promptSaveIfPendingChanges()


            let newStudentID = await addStudentPopup.trigger(props.class_id)
            window.enrollment.addEnrollment(props.class_id, newStudentID)
            updateClassData()

            // function cancels if user closes out of a popup
        } catch {return}
    }
    async function handleCreateAssignment(e:
React.MouseEvent<HTMLButtonElement>) {
        e.preventDefault()
        try {
            await promptSaveIfPendingChanges()
            await newAssignmentPopup.trigger(props.class_id)
            updateClassData()
            // create prompt for new assignment

            // function cancels if user closes out of a popup
        } catch {return}
    }



    /**
     * Resolves if user saves/undos changes, rejects if cancelled.
     */
    function promptSaveIfPendingChanges(): Promise<void> {
        return new Promise<void>((resolve, reject) => {

            // if pendingChanges isn't empty, tell user to save changes first
            if (pendingChanges.length !== 0) {
                function handlePopupSaveChanges() {
                    handleSaveChanges()
                    popup.closePopup()
                    resolve()
                }
                function handlePopupUndoChanges() {
                    handleUndoChanges()
                    popup.closePopup()
                    resolve()
                }

                const popupContent =
                    <div className="save-changes-popup">You have changes still
pending, would you like to
                        <span className="save"
onClick={handlePopupSaveChanges}> ✓ save </span>
                        or
                        <span className="undo"
onClick={handlePopupUndoChanges}> ✕ undo </span>
                        your changes?
                    </div>
                popup.triggerPopup(popupContent, "warning", () => reject())
            } else {
                resolve()
            }
        })
    }


    function updateTotal(rowNum: number) {
        if (!tableIsEmpty) {
            let totalStuPoints: number = 0
            let totalMaxPoints: number = 0

            const rowNodes = rowRefs.current[rowNum].childNodes as
NodeListOf<HTMLTableCellElement>
            // add together all scores and maxPoints for each assignment
            rowNodes.forEach((node, index) => {

                if (index !== 0 && index !== rowNodes.length - 1) {
                    const isExemptNode = node.firstChild.childNodes[4] as
HTMLSpanElement

                    if (isExemptNode.className === "exempt disabled") {
                        let thisGradeNode = node.firstChild.firstChild as
HTMLInputElement
                        totalStuPoints += parseInt(thisGradeNode.value)

                        if (classData && !classData.assignments[index -
1].is_extra_credit) {
                            let thisGradeMaxPoints =
node.firstChild.childNodes[2].nodeValue
                            totalMaxPoints += parseInt(thisGradeMaxPoints)
                        }

                    }
                }
            })
            const totalNode = rowNodes[rowNodes.length - 1]
            const totalStuPointsNode = totalNode.childNodes[0].childNodes[0] as
HTMLSpanElement
            const totalMaxPointsNode = totalNode.childNodes[0].childNodes[1] as
HTMLSpanElement
            const totalLetterNode = totalNode.childNodes[0].childNodes[2] as
HTMLSpanElement
```

```
            const totalPercentNode = totalNode.childNodes[0].childNodes[3] as
HTMLSpanElement

            const unformattedPercentGrade = totalMaxPoints === 0 ? 1 :
totalStuPoints / totalMaxPoints
            const formattedPercentGrade = totalMaxPoints === 0 ? '100%' :
                (Math.round(unformattedPercentGrade * 1000) / 10) + '%'

            totalStuPointsNode.innerText = totalStuPoints.toString()
            totalMaxPointsNode.innerText = "/ " + totalMaxPoints.toString()
            totalLetterNode.innerText = getLetterGrade(unformattedPercentGrade)
            totalPercentNode.innerText = formattedPercentGrade
        }
    }


    function getLetterGrade(percent: number): string {
        if (percent >= .97) { return "A+" }
        else if (percent >= .93) { return "A" }
        else if (percent >= .90) { return "A-" }
        else if (percent >= .87) { return "B+" }
        else if (percent >= .83) { return "B" }
        else if (percent >= .80) { return "B-" }
        else if (percent >= .77) { return "C+" }
        else if (percent >= .73) { return "C" }
        else if (percent >= .70) { return "C-" }
        else if (percent >= .67) { return "D+" }
        else if (percent >= .60) { return "D" }
        else { return "F" }
    }




    let assignmentDisplay
    let studentsDisplay
    if (classData && !tableIsEmpty) {
        gradeRefs.current = []  // reset refs to prevent overlap on re-render
        assignmentDisplay = classData.assignments.map(asgn => {
            if (asgnViewType === asgn.assignment_type || asgnViewType ===
"BOTH") {
                return (
                    <th
                        data-assignment_id={asgn.assignment_id}
                        key={asgn.assignment_id}
                        className="assignment"
                        title={asgn.description}
                        onClick={() => openAsgnSettings(asgn.assignment_id)}
                    >
                        <div className="content">
                            <h3>{asgn.name}</h3>
                            <span className="attributes">
                                <h4>{asgn.is_extra_credit ? "EXTRA CREDIT" :
""}</h4>
                                <h4>{asgn.assignment_type}</h4>
                            </span>
                        </div>
                    </th>
                )
            } else {return}
        })
        studentsDisplay = classData.studentInfo.map((stu, stuIndex) => {
            rowRefs.current = []  // reset refs to prevent overlap on re-render
            const gradesDisplay = stu.grades.map((grade, index) => {
                const max_points = classData.assignments[index].max_points
                const assignment_type =
classData.assignments[index].assignment_type
                if (asgnViewType === assignment_type || asgnViewType ===
"BOTH") {
                    return (
                        <td
                            className="grade_cell"
                            data-assignment_id={grade.assignment_id}
                            key={`${grade.assignment_id} ${stu.student_id}`}
                            ref={elem => {
                                if (elem) { gradeRefs.current.push(elem) }
                            }}
                        >
                            <span className="content_wrapper">
                                <input
                                    type="number"
                                    data-
assignment_id={grade.assignment_id.toString()}
                                    data-student_id={stu.student_id.toString()}
                                    data-rownum={stuIndex}
                                    data-
previousvalidinput={grade.earned_points.toString()}
                                    defaultValue={grade.earned_points.toString(
)} // Used to store old value, in case new value is invalid
                                    className="grade_cell_content_input"
                                    tabIndex={grade.assignment_id * 100 +
stu.student_id} // changes tab behavior to vertical
                                    id={grade.earned_points.toString()}
                                    onBlur={handleGradeChange}
                                    onKeyDown={(e) => {
                                        if (e.key == "Enter") {
                                            e.preventDefault()
                                            e.currentTarget.blur()
                                        }
                                    }}
                                />/ {max_points}<span className="percentage">
                                    {max_points === 0 ? 100 :
                                        Math.round((grade.earned_points /
max_points) * 1000) / 10
                                    }%
                                </span>
                                {grade.is_exempt ?
                                    <span
                                        onClick={handleExemptChange}
                                        data-defaultstatus="enabled"
                                        className="exempt enabled"
                                        title="Remove Exemption"
                                        data-
student_id={stu.student_id.toString()}
                                        data-
assignment_id={grade.assignment_id.toString()}
                                        data-rownum={stuIndex}
                                    >{"\u2691"}</span> :
                                    <span
                                        data-defaultstatus="disabled"
                                        onClick={handleExemptChange}
                                        className="exempt disabled"
                                        title="Make Exempt"
                                        data-
student_id={stu.student_id.toString()}
                                        data-
assignment_id={grade.assignment_id.toString()}
                                        data-rownum={stuIndex}
                                    >{"\u2690"}</span>
                                }
                            </span>
                        </td>
                    )
                } else {return}
            })
            return (
                <tr data-rownum={stuIndex} key={stu.student_id} ref={elem => {
                    if (elem) { rowRefs.current.push(elem) }
                }}>
                    <th className="studentname" onClick={() =>
handleStudentClick(stu.student_id)}>
                        {stu.first_name} {stu.last_name}
                    </th>
                    {gradesDisplay}
                    <td className="total_cell">
                        <span className="content_wrapper">
                            <span className="earned"></span>
                            <span className="max"></span>
                            <span className="letter"></span>
                            <span className="percent"></span>
                        </span>
                    </td>
                </tr>
            )

        })
    }




    return (<> {!classData ? <h2>Loading...</h2> : <>
        <div className='classtable_wrap'>
            <div
                className='tablewrap'
                hidden={tableIsEmpty}
            >
                <table className="class_table" ref={tableRef}>
                    <tbody>
                        <tr>
                            <th className="assignment_view">
                                <h3>View Type</h3>
                                <select onChange={handleAsgnViewChange}>
                                    <option value="BOTH">Both</option>
                                    <option value="HOMEWORK">Homework</option>
                                    <option value="TEST">Test</option>
                                </select>
                            </th>
                            {assignmentDisplay}
                            <th className="total">Total</th>
                        </tr>
                        {studentsDisplay}
                    </tbody>
                </table>
            </div>

            <div className='button_container' ref={buttonContainerRef}
            >
                <button className="addstudent" onClick={handleAddStudent}>
                    <span className="icon">+</span>
                    <span className="label">Add Student</span>
                </button>
                <button className="addassignment"
onClick={handleCreateAssignment}>
                    <span className="icon">+</span>
                    <span className="label">Create Assignment</span>
                </button>
                <button className="undo_changes disabled"
onClick={handleUndoChanges}>
                    <span className="icon">✕</span>
                    <span className="label">Undo Changes</span>
                </button>

                <button className="save_changes disabled"
onClick={handleSaveChanges}>
                    <span className="icon">✓</span>
                    <span className="label">Save Changes</span>
                </button>
            </div>
        </div>
    </>}</>)
}
```

ClassTable.scss (CSS sheet for styling the ClassTable)

```
@import '../../colors';

.tablewrap {
    width: fit-content;
    max-width: 100%;
    overflow: auto;
    max-height: calc(100vh - 200px)
}
```

```
.classtable_wrap {

    .class_table {
        background-color: $color4;
        margin: 5px 0px;

        tbody {
            color: $color2-text;


            th {
                text-align: left;
                padding: 5px 10px;
                min-width: 130px;
                max-width: 220px;
                background-color: $color3;
                color: $color3-text;
                font-size: 16px;

                &.assignment,
                &.total,
                &.studentname {
                    cursor: pointer;
                    transition: 0.15s ease background-color;

                    &:hover {
                        background-color: $color2;
                    }
                }

                &.assignment_view {
                    height: 100%;
                    display:flex;
                    align-items: center;
                    justify-content: center;
                    flex-direction: column;

                    h3 {
                        margin: 0 0 5px 0;
                    }
                    select {
                        width: 100%;
                        height: 23px;
                    }
                }

                &.assignment {
                    min-width: 160px;
                    .content {

                        h3 {
                            margin: 0;
                            height: 100%;
                            overflow-wrap:break-word;
                            hyphens: auto;
                        }
                        .attributes {
                            margin-top: auto;
                            display: flex;
                            justify-content: space-between;
                            // gap: 20px;
                            margin-top: auto;

                            h4 {
                                bottom: 5px;
                                right: 5px;
                                margin: 0;
                                font-size: 10px;
                            }
                        }
                    }

                }
            }

            td.grade_cell {
                background-color: $color2;
                padding: 5px 10px;
                font-size: 12px;


                .content_wrapper {
                    input {
                        background-color: transparent;
                        font-size: 16px;
                        border: none;
                        padding: 2px 5px;
                        width: 50px;
                        outline: 0px solid transparent;
                        transition: 0.1s ease background-color;
                        cursor: text;
                        margin-right: 5px;

                        background-color: $color2-lighter;
                        border-radius: 3px;

                        &:focus,
                        &:hover {
                            background-color: $color2-lighter2;
                        }
                    }

                    display: flex;
                    align-items: end;
                    position: relative;

                    .percentage {
                        font-size: 16px;
                        justify-self: flex-end;
                        align-self: flex-end;
                        margin-left: auto;
                    }

                    .exempt {
                        position: absolute;
                        right: -6px;
                        top: -10px;
                        cursor: pointer;
                        user-select: none;

                        &.disabled {
                            color: transparent;

                            // If user is hovering over the cell, and exempt is
disabled, make it black instead of transparent.
                            @at-root
tr>td.grade_cell:hover>.content_wrapper>.exempt.disabled {
                                color: black !important;
                            }


                        }
                    }
                }
```

```
        &.pending_change {
            box-shadow: inset 0 0 0 2px $color3-darker;
            background-color: $color2-darker;
        }
    }

    td.total_cell {
        background-color: $color2;
        padding: 5px 10px;
        font-size: 12px;


        .content_wrapper {
            display: flex;
            justify-content: flex-start;
            align-items: center;
            width: 200px;

            .earned {
                font-size: 16px;
                width: 30px;
                margin: 0px 2px 0px 0px;
                padding: 2px 5px;
            }

            .max {
                align-self: flex-end;
                width: 60px;
            }

            .letter {
                margin-right: auto;
                font-size: 25px;
            }

            .percent {
                font-size: 15px;
            }
        }
    }

}
```

```
    }


    .button_container {
        min-width: 715px;
        position: sticky;
    }
}

.button_container {
    display: flex;
    flex-direction: row;
    align-items: flex-start;
    gap: 5px;
    bottom: 20px;
    margin-top: 10px;
    max-width: 100%;

    &>* {
        width: 180px;
    }

    .undo_changes {
        margin-left: auto
    }
    button.addassignment {
        width: 200px;
    }

    button.pending_changes_animation {
        scale: 105%;
    }

    button.disabled {
        color: gray;
        background-color: $accent-disabled;
        cursor:not-allowed;

        &:hover, &:focus {
            scale: 100%;
            background-color: $accent-disabled;
        }
    }
}
```

```
div.save-changes-popup {


    .save,
    .undo {
        background-color: $accent;
        padding: 2px 4px;
        border-radius: 2px;
        display: inline-flex;
        margin: 0 3px;
        cursor: pointer;

        * {
            transform: translateY(-4px);
        }

        &:hover {
            background-color: $accent-lighter;
        }
    }
}
```

Popup.tsx (A reusable popup component I made)

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import './popup.scss'

const popupDomLocation = ReactDOM.createRoot(document.getElementById('popup-
root'))

function closePopup() {
    popupDomLocation.render(<></>)
}


function triggerPopup(content: JSX.Element, specialType?: "warning",
handleClose?: () => void) {


    const popupFinal =
    <div className="popup-background" >
            <div className={`popup-container ${specialType === "warning" ?
"warning" : ""}`}>
```

```
                <div className="popup-exit"
                    onClick={() => {
                        closePopup()
                        if (handleClose) {handleClose()}
                    }}
                >✕</div>
                {content}
            </div>
        </div>

    popupDomLocation.render(popupFinal)
}

interface popup_exports {
    /**
     * Creates a blocking popup on the screen. Overrides other popups.
     * @param content The content to hold within the popup.
     * @param specialType `"warning"` will make the popup tinted red.
     * @param handleClose A callback function that is ran when the popup is
closed by the user pressing the `x` button.
     */
    triggerPopup: (
        content: JSX.Element,
        specialType?: "warning",
        handleClose?: () => void
    ) => void,
    /**
     * Closes the current popup.
     */
    closePopup: () => void,
}

export default {
    triggerPopup, closePopup
} as popup_exports
```

Popup.scss (Styling for popups)

```
.popup-background {
    position: fixed;
    top: 0;
    left: 0;
    height: 100vh;
    width: 100vw;
    background-color: rgba(0, 0, 0, 0.4);
    z-index: 19;

    .popup-container {
        position: fixed;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
        width: auto;
        height: auto;
        background-color: white;
        border-radius: 20px;
        z-index: 20;
        color: black;
        padding: 40px 20px 20px 20px;

        &.warning {
            background-color: rgb(241, 189, 189);
        }

        .popup-exit {
            right: 13px;
            top: 9px;
            position: absolute;
            cursor: pointer;
            user-select: none;
        }
    }
}
```

AddStudentPopup.tsx

```
import React from 'react'
import './addStudentPopup.scss'
import popup from '../../Popup/popup'
import createStudentPopup from './createStudentPopup'




function validateName(input: string): boolean {
    if (input.trim().length === 0) { return false }
    if (input.length > 25) { return false }
    const bannedCharacters = [';', '"', ':', '<', '>', '(', ')', '{', '}', '[',
']', '*', '%']
    if (bannedCharacters.some(char => input.includes(char))) { return false }

    return true

}

async function trigger(class_id: number) {

    let studentOptions = await window.class.getStudentsNotInClass(class_id)
    console.log(studentOptions)
    return new Promise<number>((resolve, reject) => {

        const content =
            <div className='add_student_popup'>
                <h1>Add a Student to this class</h1>
                <div className='field_container'>
                    <label htmlFor='first_name'>Add Existing Student:</label>
                    <select>
                        {
                        studentOptions.map(stu => {return (
                                <option key={stu.student_id}
value={stu.student_id}>
                                    {stu.first_name} {stu.last_name}
                                </option>
                        )})
                        }
                    </select>
                    <button onClick={handleAddStudent}>Add</button>
                </div>

                <h3>- or -</h3>
                <button onClick={handleCreateStudent}>Create a new
Student</button>
            </div>

        popup.triggerPopup(content, null, () => reject("addStudentPopup closed
by user"))

        function handleAddStudent(e: React.MouseEvent<HTMLButtonElement>) {
            const buttonNode = e.target as HTMLButtonElement
            const currentValueNode = buttonNode.parentNode.childNodes[1] as
HTMLSelectElement
            const currentValue = parseInt(currentValueNode.value)
            popup.closePopup()
            resolve(currentValue)
        }

        function handleCreateStudent() {
            popup.closePopup()
            createStudentPopup.trigger()
                .then(newStudentID => resolve(newStudentID))
                .catch(error => reject(error))
        }


    })
}

declare global {

    interface AddNewStudent_exports {

        /**
         * Triggers a popup to create a new student, creates the student in the
DB, and returns the new `student_id`.
         * @returns A promise that resolves with the new student's ID, or
`"newStudentPopup closed by user"` if closed without creating a new student.
         */
        trigger: (class_id: number) => Promise<number>
    }
}
export default {
    trigger
} as AddNewStudent_exports

// this should be structured so that trigger() returns the student_id instead
as a promise.
// add 25 char limit to input
```

AddStudentPopup.scss

```
@import '../../colors';

.add_student_popup {
    display: flex;
    flex-direction: column;
    align-items: center;
    width: 500px;

    h1 {
        font-size: 20px;
        text-align: center;
    }

    .field_container {
        margin-top: 10px;
        display: flex;
        flex-direction: row;
        align-items: center;

        label {
            width: 180px;
            transform: translateY(-2px);
        }

        select {
            padding: 5px;
            min-width: 150px;
            height: 40px;
            cursor: pointer;
        }
        button {
            margin-left: 20px;
        }

    }

    h3 {
        margin: 10px;
    }

    button {
        margin-right: 0
    }

}
```

createStudentPopup.tsx

```
import React from 'react'
import './createStudentPopup.scss'
import popup from '../../Popup/popup'




function validateName(input: string): boolean {
    if (input.trim().length === 0) { return false }
    if (input.length > 25) { return false }

    const bannedCharacters = [';', '"', ':', '<', '>', '(', ')', '{', '}', '[', ']', '*', '%']
    if (bannedCharacters.some(char => input.includes(char))) { return false }

    return true

}

function trigger() {
    return new Promise<number>((resolve, reject) => {

        const content =
            <div className='new_student_popup'>
                <h1>Create A New Student</h1>
                <p className='error_message'></p>
                <div className='field_container'>
                    <label htmlFor='first_name'>First Name:</label>
                    <input type='text' id='first_name' maxLength={25}></input>
                </div>
                <div className='field_container'>
                    <label htmlFor='last_name'>Last Name:</label>
                    <input type='text' id='last_name' maxLength={25}></input>
                </div>
                <button onClick={handleCreate}>Create</button>
            </div>

        popup.triggerPopup(content, null, () => reject("createStudentPopup closed by user"))

        function handleCreate(e: React.MouseEvent<HTMLButtonElement>) {
            const buttonNode = e.target as HTMLButtonElement

            const errorNode = buttonNode.parentElement.childNodes[1] as HTMLParagraphElement
            const inputFirstNameNode = 
buttonNode.parentElement.childNodes[2].childNodes[1] as HTMLInputElement
            const inputLastNameNode = 
buttonNode.parentElement.childNodes[3].childNodes[1] as HTMLInputElement
            const inputFirstNameValue = inputFirstNameNode.value
            const inputLastNameValue = inputLastNameNode.value



            const firstNameIsValid = validateName(inputFirstNameValue)
            const lastNameIsValid = validateName(inputLastNameValue)
            if (!firstNameIsValid || !lastNameIsValid) {

                errorNode.innerText = 
                    "Incorrect input detected. Each field must be less than 25 
characters, and must not contain banned characters."

                if (!firstNameIsValid) {
                    inputFirstNameNode.className = 'error'
                } else {
                    inputFirstNameNode.className = ''
                }

                if (!lastNameIsValid) {
                    inputLastNameNode.className = 'error'
                } else {
                    inputLastNameNode.className = ''
                }
            } else {
                popup.closePopup()
                window.student.createStudent(inputFirstNameValue, 
inputLastNameValue).then(newStudentID => {
                    resolve(newStudentID)
                })
            }
        }

    })
}

declare global {

    interface CreateNewStudent_exports {

        /**
```

```
         * Triggers a popup to create a new student, creates the student in the
DB, and returns the new `student_id`.
         * @returns A promise that resolves with the new student's ID, or
`"newStudentPopup closed by user"` if closed without creating a new student.
         */
        trigger: () => Promise<number>
    }
}
export default {
    trigger
} as CreateNewStudent_exports

// this should be structured so that trigger() returns the student_id instead
as a promise.
// add 25 char limit to input
```

createStudentPopup.scss

```
@import '../../colors';

.new_student_popup {
    display: flex;
    flex-direction: column;
    width: 300px;

    h1 {
        font-size: 20px;
        text-align: center;
    }

    .error_message {
        font-size: 12px;
        text-align: center;
        color: $error-text;
        margin: 0 0 15px 0;
    }

    .field_container {
        display: flex;
        flex-direction: row;

        input {
            width: 150px;
            margin: 4px;
            border: 2px solid $background;
            background-color: $color2-lighter;
            border-radius: 8px;
            padding: 5px;
            outline: none;

            &:hover,
            &:focus {
                background-color: $color2-lighter2;
            }
            &.error {
                border: 2px solid $error-border;
                background-color: $error-fill;
            }
        }

        label {
            margin-right: 6px;
            width: 80px;
            transform: translateY(5px);
        }
    }

}
```

editAssignmentPopup.tsx

```
import React from 'react'
import './editAssignmentPopup.scss'
import popup from '../../Popup/popup'




function checkForBannedChars(input: string): boolean {

    const bannedCharacters = [';', '"', ':', '<', '>', '(', ')', '{', '}', '[',
']', '*', '%']
    if (bannedCharacters.some(char => input.includes(char))) { return false }

    return true

}
```

```
async function trigger(assignment_id: number) {

    const asgnData = await window.assignment.getAssignment(assignment_id)

    return new Promise<void>((resolve, reject) => {
        const content =
            <div className='edit_assignment_popup'>
                <h1>Edit Assignment</h1>
                <p className='error_message'></p>

                <div className='field_container'>
                    <label htmlFor='name'>Assignment Name:</label>
                    <input type='text' id='name' maxLength={50}
defaultValue={asgnData.name}/>
                </div>
                <div className='field_container'>
                    <label htmlFor='description'>
                        Description:<br />
                        <span className="subtext">optional</span>
                    </label>
                    <textarea
                        id='description'
                        maxLength={200}
                        defaultValue={asgnData.description}
                        onKeyDown={(e) => {
                        if (e.key === 'Enter') {e.preventDefault()}
                    }}/>
                </div>
                <div className='field_container radio'>
                    <label className='fieldname'>Assignment Type:</label>
                    <label htmlFor='HOMEWORK'>Homework</label>
                    <input
                        type='radio'
                        name='type'
                        id='HOMEWORK'
                        defaultChecked={asgnData.assignment_type ===
"HOMEWORK"}
                    />
                    <label htmlFor='TEST'>Test</label>
                    <input
                        type='radio'
                        name='type'
                        id='TEST'
                        defaultChecked={asgnData.assignment_type === "TEST"}
                    />
                </div>
                <div className='field_container extra_credit'>
                    <label htmlFor='is_extra_credit'>Extra Credit:</label>
                    <input type='checkbox' id='is_extra_credit'
defaultChecked={asgnData.is_extra_credit}/>
                </div>
                <div className='field_container'>
                    <label htmlFor='max_points'>Maximum Points:</label>
                    <input type='number' id='max_points'
defaultValue={asgnData.max_points}/>
                </div>

                <div className='button_container'>
                    <button onClick={handleDelete}
className='delete'>Delete</button>
                    <button onClick={handleSave}>Save</button>
                </div>
            </div>

        popup.triggerPopup(content, null, () => reject("editAssignmentPopup
closed by user"))

        function handleDelete() {
            popup.triggerPopup(
                <div className='delete_confirm'>
                    <h3>Are you sure you'd like to delete this assignment?</h3>
                    <div className='button_container'>
                        <button onClick={handleDeny}>No, take me back</button>
                        <button onClick={handleConfirm} className='delete'>Yes,
delete</button>
                    </div>
                </div>
            , "warning")

            function handleDeny() {
                popup.closePopup()
                trigger(assignment_id)
            }
            function handleConfirm(){
                window.assignment.deleteAssignment(assignment_id)
                    .then(() => {
                        popup.closePopup()
                        resolve()
                    })
            }
        }

        function handleSave(e: React.MouseEvent<HTMLButtonElement>) {
            const buttonNode = e.target as HTMLButtonElement

            const errorNode =
buttonNode.parentElement.parentElement.childNodes[1] as HTMLParagraphElement

            const inputNameNode =
buttonNode.parentElement.parentElement.childNodes[2].childNodes[1] as
HTMLInputElement
            const inputName: string = inputNameNode.value

            const inputDescriptionNode =
buttonNode.parentElement.parentElement.childNodes[3].childNodes[1] as
HTMLInputElement
            const inputDescription: string = inputDescriptionNode.value

            const inputTypeHomeworkNode =
buttonNode.parentElement.parentElement.childNodes[4].childNodes[2] as
HTMLInputElement
            const inputTypeIsHomework: boolean = inputTypeHomeworkNode.checked

            const inputTypeTestNode =
buttonNode.parentElement.parentElement.childNodes[4].childNodes[4] as
HTMLInputElement
            const inputTypeIsTest: boolean = inputTypeTestNode.checked

            const inputExtraCreditNode =
buttonNode.parentElement.parentElement.childNodes[5].childNodes[1] as
HTMLInputElement
            const inputIsExtraCredit: boolean = inputExtraCreditNode.checked

            const inputMaxPointsNode =
buttonNode.parentElement.parentElement.childNodes[6].childNodes[1] as
HTMLInputElement
            const inputMaxPoints: number = parseInt(inputMaxPointsNode.value)



            let errorMsg = "Error(s) detected:"
            let errorFound = false

            const inputNameIsValid = checkForBannedChars(inputName)
            const inputDescriptionIsValid = inputDescription.trim() === "" ? 
true : checkForBannedChars(inputDescription)
            const inputTypeIsChosen = inputTypeIsHomework || inputTypeIsTest
            const inputMaxPointsIsValid = !isNaN(inputMaxPoints)


            if (inputName.trim().length === 0) {
                errorMsg += "\nAssignment Name cannot be blank."
                errorFound = true
                inputNameNode.classList.add('error')
            } else if (!inputNameIsValid) {
                errorMsg += "\nAssignment Name cannot contain any of the 
following characters:\n;  :  <  >  (  )  {  }  [  ]  *  %"
                errorFound = true
                inputNameNode.classList.add('error')
            } else {inputNameNode.classList.remove('error')}


            if (!inputDescriptionIsValid) {
                errorMsg += "\nDescription cannot contain any of the following 
characters:\n;  :  <  >  (  )  {  }  [  ]  *  %"
                errorFound = true
                inputDescriptionNode.classList.add('error')
            } else {inputDescriptionNode.classList.remove('error')}


            if (!inputTypeIsChosen) {
                errorMsg += '\nYou must select either "Homework" or "Test"'
                errorFound = true
                inputTypeHomeworkNode.classList.add('error')
                inputTypeTestNode.classList.add('error')
            } else {
                inputTypeHomeworkNode.classList.remove('error')
                inputTypeTestNode.classList.remove('error')
            }

            if (!inputMaxPointsIsValid) {
                errorFound = true
                inputMaxPointsNode.classList.add('error')
                errorMsg += '\nYou must enter a value for Maximum Points'
            } else {
                inputMaxPointsNode.classList.remove('error')
            }
            if (errorFound) {
                errorNode.innerText = errorMsg
            } else {
                const inputType = inputTypeIsHomework ? "HOMEWORK" : "TEST"

                window.assignment.editAssignment(
                    assignment_id,
                    inputName,
                    inputDescription,
                    inputType,
                    inputIsExtraCredit,
                    inputMaxPoints
                    ).then(() => resolve())


                popup.closePopup()
            }

        }
    })
}

declare global {

    interface EditAssignment_exports {

        /**
         * Triggers a popup to create a new student, creates the student in the
DB, and returns the new `student_id`.
         * @param class_id The ID of the class to create the assignment in.
         * @returns A promise that resolves with the new student's ID, or
`"newStudentPopup closed by user"` if closed without creating a new student.
         */
        trigger: (class_id: number) => Promise<void>
    }
}
export default {
    trigger
} as EditAssignment_exports
```

editAssignmentPopup.scss

```
@import '../../colors';

.edit_assignment_popup {
    display: flex;
    flex-direction: column;
    width: 454px;

    h1 {
        font-size: 20px;
        text-align: center;
    }

    .button_container {
        display: flex;
        flex-direction: row;
        button {
            text-align: center;
            display: flex;
            justify-content: center;

        }
    }

    .error_message {
        font-size: 12px;
        text-align: center;
        color: $error-text;
        margin: 0 0 15px 0;
    }

    .field_container {
        display: flex;
        flex-direction: row;

        input[type='text'], input[type='number'], textarea {
            resize: none;
            width: 300px;
            margin: 4px;
            border: 2px solid $background;
            background-color: $color2-lighter;
            border-radius: 8px;
            padding: 5px;
            outline: none;

            &:hover,
            &:focus {
                background-color: $color2-lighter2;
            }
            &.error {
                border: 2px solid $error-border;
                background-color: $error-fill;
            }
        }
        textarea {
            height: 100px;
            font-family: Arial, Helvetica, sans-serif
        }

        label {
            margin-right: 6px;
            width: 140px;
            transform: translateY(5px);
            line-height: 16px;

            .subtext {
                font-size: 12px;
                font-style:italic;
            }
        }

        &.radio {
            margin-top: 10px;

            label {
                width: fit-content;
                transform: none;


                &.fieldname {
                    margin-right: 6px;
                    width: 140px;
                    transform: translateY(-1px);
                }
                &[for='TEST'] {
                    margin-left: 65px;
                }
            }
            input {
                scale: 150%;
                cursor: pointer;
                outline: 2px red;

                &.error::before {
                    content: "";
                    display: inline-block;
                    width: 16px;
                    height: 16px;
                    transform: translate(-1.6px,-1.6px);
                    background-color: #ff000054;
                    border-radius: 50%;
                }
            }
        }

        &.extra_credit {
            margin-top: 15px;
            margin-bottom: 10px;

            label {
                transform: translateY(-1px);
            }

            input {
                scale: 150%;
                cursor: pointer;
            }
        }
    }

    button {
        margin: 10px auto 0px auto;
    }

}

.delete_confirm {
    button {
        text-align: center;
        display: block;

        &.delete {
            margin-left: auto;
            color: $error-text;
        }
    }
}
```

editEnrollmentPopup.tsx

```
import React from 'react'
import './editEnrollmentPopup.scss'
import popup from '../../Popup/popup'
import editStudentPopup from './editStudentPopup'




function validateName(input: string): boolean {
    if (input.trim().length === 0) { return false }
    if (input.length > 25) { return false }

    const bannedCharacters = [';', '"', ':', '<', '>', '(', ')', '{', '}', '[', 
']', '*', '%']
    if (bannedCharacters.some(char => input.includes(char))) { return false }

    return true

}

function trigger(student_id: number, class_id: number) {
    return new Promise<void>((resolve, reject) => {

        const content =
            <div className='edit_enrollment_popup'>
                <h1>Edit Enrollment</h1>
                <div className='button_container'>
                    <button onClick={handleRemove} className='delete'>Remove 
From Class</button>
                    <button onClick={handleEdit}>Edit Student</button>
                </div>
            </div>

        popup.triggerPopup(content, null, () => reject("editEnrollmentPopup 
closed by user"))
```

```
        function handleRemove() {
            popup.triggerPopup(
                <div className='delete_confirm'>
                    <h3>Are you sure you'd like to remove this student from this class?</h3>
                    <p>This will delete all of their current grades for this class!</p>
                    <div className='button_container'>
                        <button onClick={handleDeny}>No, take me back</button>
                        <button onClick={handleConfirm} className='delete'>Yes, remove</button>
                    </div>
                </div>
            , "warning")

            function handleDeny() {
                popup.closePopup()
                trigger(student_id, class_id)
            }
            function handleConfirm(){
                window.enrollment.deleteEnrollment(class_id, student_id)
                    .then(() => {
                        popup.closePopup()
                        resolve()
                    })
            }
        }

        function handleEdit() {
            editStudentPopup.trigger(student_id)
                .then(() => {
                    resolve()
                })
                .catch (() => {
                    reject()
                })
        }

    })
}

declare global {

    interface EditEnrollment_exports {
        /**
         * Triggers a popup to edit a student's enrollment.
         * @returns A promise that resolves if changes are made, or rejects
otherwise.
         */
        trigger: (student_id: number, class_id: number) => Promise<void>
    }
}
export default {
    trigger
} as EditEnrollment_exports
```

editEnrollmentPopup.scss

```
@import '../../colors';

.edit_enrollment_popup {
    display: flex;
    flex-direction: column;

    .button_container {
        display: flex;
        flex-direction: row;
        button {
            text-align: center;
            display: flex;
            justify-content: center;

            &.delete {
                color: $error-text;
            }
        }
    }
    h1 {
        font-size: 20px;
        text-align: center;
    }

}

.delete_confirm {
    button {
        text-align: center;
        display: block;

        &.delete {
            margin-left: auto;
            color: $error-text;
        }
    }
    p {
        margin: 0;
        color: rgb(151, 0, 0);
    }
    h3 {
        margin: 0;
    }
}
```

editStudentPopup.tsx

```
import React from 'react'
import './editStudentPopup.scss'
import popup from '../../Popup/popup'




function validateName(input: string): boolean {
    if (input.trim().length === 0) { return false }
    if (input.length > 25) { return false }

    const bannedCharacters = [';', '"', ':', '<', '>', '(', ')', '{', '}', '[',
']', '*', '%']
    if (bannedCharacters.some(char => input.includes(char))) { return false }

    return true

}

function trigger(student_id: number) {

    return new Promise<void>(async (resolve, reject) => {

        try {
            const curStuInfo = await window.student.getStudentInfo(student_id)
            const content =
                <div className='edit_student_popup'>
                    <h1>Edit Student</h1>
                    <p className='error_message'></p>
                    <div className='field_container'>
                        <label htmlFor='first_name'>First Name:</label>
                        <input type='text' id='first_name' maxLength={25}
defaultValue={curStuInfo.first_name}></input>
                    </div>
                    <div className='field_container'>
                        <label htmlFor='last_name'>Last Name:</label>
                        <input type='text' id='last_name' maxLength={25}
defaultValue={curStuInfo.last_name}></input>
                    </div>
                    <div className='button_container'>
                        <button onClick={handleDelete}
className='delete'>Delete Student</button>
                        <button onClick={handleCreate}>Save</button>

                    </div>
                </div>

            popup.triggerPopup(content, null, () => reject("editStudentPopup
closed by user"))

            function handleCreate(e: React.MouseEvent<HTMLButtonElement>) {
                const buttonNode = e.target as HTMLButtonElement

                const errorNode =
buttonNode.parentElement.parentElement.childNodes[1] as HTMLParagraphElement

                const inputFirstNameNode =
buttonNode.parentElement.parentElement.childNodes[2].childNodes[1] as
HTMLInputElement
                const inputLastNameNode =
buttonNode.parentElement.parentElement.childNodes[3].childNodes[1] as
HTMLInputElement
                const inputFirstNameValue = inputFirstNameNode.value
                const inputLastNameValue = inputLastNameNode.value



                const firstNameIsValid = validateName(inputFirstNameValue)
                const lastNameIsValid = validateName(inputLastNameValue)
                if (!firstNameIsValid || !lastNameIsValid) {
                    errorNode.innerText =
                        "Incorrect input detected. Each field must be less than 
25 characters, and must not contain banned characters."

                    if (!firstNameIsValid) {
                        inputFirstNameNode.className = 'error'
                    } else {
                        inputFirstNameNode.className = ''
                    }

                    if (!lastNameIsValid) {
                        inputLastNameNode.className = 'error'
                    } else {
                        inputLastNameNode.className = ''
                    }
                } else {

                    // check if changes were made
                    if (inputFirstNameValue !== curStuInfo.first_name || 
inputLastNameValue !== curStuInfo.last_name) {
                        popup.closePopup()
                        window.student.editStudent(student_id, 
inputFirstNameValue, inputLastNameValue).then(() => {
                            resolve()
                        })
                    } else {
                        reject()
                    }

                }
            }

            function handleDelete() {
                popup.triggerPopup(
                    <div className='delete_confirm'>
                        <h3>Are you sure you'd like to delete this 
student?</h3>
                        <div className='button_container'>
                            <button onClick={handleDeny}>No, take me 
back</button>
                            <button onClick={handleConfirm} 
className='delete'>Yes, delete</button>
                        </div>
                    </div>
                , "warning")
                function handleDeny() {
                    popup.closePopup()
                    trigger(student_id)
                }
                function handleConfirm(){
                    window.student.deleteStudent(student_id)
                        .then(() => {
                            popup.closePopup()
                            resolve()
                        })
                }
            }
        } catch {
            reject(`error occured, can not retrieve StudentInfo for student_id
= ${student_id}`)
        }



    })
}

declare global {

    interface EditStudent_exports {

        /**
         * Triggers a popup to edit a student's name.
         * @returns A promise that resolves if changes are made, and rejects if
not.
         */
        trigger: (student_id: number) => Promise<void>
    }
}
export default {
    trigger
} as EditStudent_exports

// this should be structured so that trigger() returns the student_id instead
as a promise.
// add 25 char limit to input
```

editStudentPopup.scss

```
@import '../../colors';

.edit_student_popup {
    display: flex;
    flex-direction: column;
    width: 300px;

    h1 {
        font-size: 20px;
        text-align: center;
    }

    .error_message {
        font-size: 12px;
        text-align: center;
        color: $error-text;
        margin: 0 0 15px 0;
    }

    .field_container {
        display: flex;
        flex-direction: row;

        input {
            width: 150px;
            margin: 4px;
            border: 2px solid $background;
            background-color: $color2-lighter;
            border-radius: 8px;
            padding: 5px;
            outline: none;

            &:hover,
            &:focus {
                background-color: $color2-lighter2;
            }
            &.error {
                border: 2px solid $error-border;
                background-color: $error-fill;
            }
        }

        label {
            margin-right: 6px;
            width: 80px;
            transform: translateY(5px);
        }

    }

    .button_container {
        display: flex;

        button.delete {
            margin-right: auto
        }
    }

    .delete_confirm {
        button {
            text-align: center;
            display: block;

            &.delete {
                margin-left: auto;
                color: $error-text;
            }
        }
    }

}
```

newAssignmentPopup.tsx

```
import React from 'react'
import './newAssignmentPopup.scss'
import popup from '../../Popup/popup'





function checkForBannedChars(input: string): boolean {

    const bannedCharacters = [';', '"', ':', '<', '>', '(', ')', '{', '}', '[',
']', '*', '%']
    if (bannedCharacters.some(char => input.includes(char))) { return false }

    return true
```

```
}

function trigger(class_id: number) {
    return new Promise<number>((resolve, reject) => {

        const content =
            <div className='new_assignment_popup'>
                <h1>Create A New Assignment</h1>
                <p className='error_message'></p>

                <div className='field_container'>
                    <label htmlFor='name'>Assignment Name:</label>
                    <input type='text' id='name' maxLength={50} />
                </div>
                <div className='field_container'>
                    <label htmlFor='description'>
                        Description:<br />
                        <span className="subtext">optional</span>
                    </label>
                    <textarea id='description' maxLength={200} onKeyDown={(e)
=> {
                        if (e.key === 'Enter') {e.preventDefault()}
                    }}/>
                </div>
                <div className='field_container radio'>
                    <label className='fieldname'>Assignment Type:</label>
                    <label htmlFor='HOMEWORK'>Homework</label>
                    <input type='radio' name='type' id='HOMEWORK' />
                    <label htmlFor='TEST'>Test</label>
                    <input type='radio' name='type' id='TEST' />
                </div>
                <div className='field_container extra_credit'>
                    <label htmlFor='is_extra_credit'>Extra Credit:</label>
                    <input type='checkbox' id='is_extra_credit' />
                </div>
                <div className='field_container'>
                    <label htmlFor='max_points'>Maximum Points:</label>
                    <input type='number' id='max_points' />
                </div>


                <button onClick={handleCreate}>Create</button>
            </div>
        popup.triggerPopup(content, null, () => reject("newAssignmentPopup
closed by user"))

        function handleCreate(e: React.MouseEvent<HTMLButtonElement>) {
            const buttonNode = e.target as HTMLButtonElement

            const errorNode = buttonNode.parentElement.childNodes[1] as
HTMLParagraphElement

            const inputNameNode =
buttonNode.parentElement.childNodes[2].childNodes[1] as HTMLInputElement
            const inputName: string = inputNameNode.value

            const inputDescriptionNode =
buttonNode.parentElement.childNodes[3].childNodes[1] as HTMLInputElement
            const inputDescription: string = inputDescriptionNode.value

            const inputTypeHomeworkNode =
buttonNode.parentElement.childNodes[4].childNodes[2] as HTMLInputElement
            const inputTypeIsHomework: boolean = inputTypeHomeworkNode.checked

            const inputTypeTestNode =
buttonNode.parentElement.childNodes[4].childNodes[4] as HTMLInputElement
            const inputTypeIsTest: boolean = inputTypeTestNode.checked

            const inputExtraCreditNode =
buttonNode.parentElement.childNodes[5].childNodes[1] as HTMLInputElement
            const inputIsExtraCredit: boolean = inputExtraCreditNode.checked

            const inputMaxPointsNode =
buttonNode.parentElement.childNodes[6].childNodes[1] as HTMLInputElement
            const inputMaxPoints: number = parseInt(inputMaxPointsNode.value)



            let errorMsg = "Error(s) detected:"
            let errorFound = false

            const inputNameIsValid = checkForBannedChars(inputName)
            const inputDescriptionIsValid = inputDescription.trim() === "" ?
true : checkForBannedChars(inputDescription)
            const inputTypeIsChosen = inputTypeIsHomework || inputTypeIsTest
            const inputMaxPointsIsValid = !isNaN(inputMaxPoints)
            if (inputName.trim().length === 0) {
                errorMsg += "\nAssignment Name cannot be blank."
                errorFound = true
                inputNameNode.classList.add('error')
            } else if (!inputNameIsValid) {
                errorMsg += "\nAssignment Name cannot contain any of the 
following characters:\n;  :  <  >  (  )  {  }  [  ]  *  %"
                errorFound = true
                inputNameNode.classList.add('error')
            } else {inputNameNode.classList.remove('error')}


            if (!inputDescriptionIsValid) {
                errorMsg += "\nDescription cannot contain any of the following 
characters:\n;  :  <  >  (  )  {  }  [  ]  *  %"
                errorFound = true
                inputDescriptionNode.classList.add('error')
            } else {inputDescriptionNode.classList.remove('error')}


            if (!inputTypeIsChosen) {
                errorMsg += '\nYou must select either "Homework" or "Test"'
                errorFound = true
                inputTypeHomeworkNode.classList.add('error')
                inputTypeTestNode.classList.add('error')
            } else {
                inputTypeHomeworkNode.classList.remove('error')
                inputTypeTestNode.classList.remove('error')
            }

            if (!inputMaxPointsIsValid) {
                errorFound = true
                inputMaxPointsNode.classList.add('error')
                errorMsg += '\nYou must enter a value for Maximum Points'
            } else {
                inputMaxPointsNode.classList.remove('error')
            }




            if (errorFound) {
                errorNode.innerText = errorMsg
            } else {
                const inputType = inputTypeIsHomework ? "HOMEWORK" : "TEST"
                window.assignment.createAssignment(
                    class_id,
                    inputName,
                    inputDescription,
                    inputType,
                    inputIsExtraCredit,
                    inputMaxPoints
                ).then(newAsgnID => resolve(newAsgnID))

                popup.closePopup()
            }

        }
    })
}

declare global {

    interface AddNewAssignment_exports {

        /**
         * Triggers a popup to create a new student, creates the student in the DB, and returns the new `student_id`.
         * @param class_id The ID of the class to create the assignment in.
         * @returns A promise that resolves with the new student's ID, or `"newStudentPopup closed by user"` if closed without creating a new student.
         */
        trigger: (class_id: number) => Promise<number>
    }
}
export default {
    trigger
} as AddNewAssignment_exports
```

newAssignmentPopup.scss

```
@import '../../colors';

.new_assignment_popup {
    display: flex;
    flex-direction: column;
    width: 454px;
```

```
h1 {
    font-size: 20px;
    text-align: center;
}

.error_message {
    font-size: 12px;
    text-align: center;
    color: $error-text;
    margin: 0 0 15px 0;
}

.field_container {
    display: flex;
    flex-direction: row;

    input[type='text'], input[type='number'], textarea {
        resize: none;
        width: 300px;
        margin: 4px;
        border: 2px solid $background;
        background-color: $color2-lighter;
        border-radius: 8px;
        padding: 5px;
        outline: none;

        &:hover,
        &:focus {
            background-color: $color2-lighter2;
        }
        &.error {
            border: 2px solid $error-border;
            background-color: $error-fill;
        }
    }
    textarea {
        height: 100px;
        font-family: Arial, Helvetica, sans-serif
    }

    label {
        margin-right: 6px;
        width: 140px;
        transform: translateY(5px);
        line-height: 16px;

        .subtext {
            font-size: 12px;
            font-style:italic;
        }
    }

    &.radio {
        margin-top: 10px;

        label {
            width: fit-content;
            transform: none;


            &.fieldname {
                margin-right: 6px;
                width: 140px;
                transform: translateY(-1px);
            }
            &[for='TEST'] {
                margin-left: 65px;
            }
        }

        input {
            scale: 150%;
            cursor: pointer;
            outline: 2px red;

            &.error::before {
                content: "";
                display: inline-block;
                width: 16px;
                height: 16px;
                transform: translate(-1.6px,-1.6px);
                background-color: #ff000054;
                border-radius: 50%;
            }
        }
    }

    &.extra_credit {
        margin-top: 15px;
```

```
            margin-bottom: 10px;

            label {
                transform: translateY(-1px);
            }

            input {
                scale: 150%;
                cursor: pointer;
            }
        }
    }

    button {
        margin: 10px auto 0px auto;
    }

}
```

newClassPopup.tsx

```
import React from 'react'
import './newClassPopup.scss'
import popup from '../../Popup/popup'




function checkForBannedChars(input: string): boolean {

    const bannedCharacters = [';', '"', ':', '<', '>', '(', ')', '{', '}', '[',
']', '*', '%']
    if (bannedCharacters.some(char => input.includes(char))) { return false }

    return true

}

function trigger() {
    return new Promise<number>((resolve, reject) => {

        const content =
            <div className='new_class_popup'>
                <h1>Create A New Class</h1>
                <p className='error_message'></p>

                <div className='field_container'>
                    <label htmlFor='name'>Class Name:</label>
                    <input type='text' id='name' maxLength={50} />
                </div>
                <div className='field_container'>
                    <label htmlFor='description'>
                        Description:<br />
                        <span className="subtext">optional</span>
                    </label>
                    <textarea id='description' maxLength={200} onKeyDown={(e)
=> {
                        if (e.key === 'Enter') {e.preventDefault()}
                    }}/>
                </div>

                <button onClick={handleCreate}>Create</button>
            </div>

        popup.triggerPopup(content, null, () => reject("newClassPopup closed by
user"))

        function handleCreate(e: React.MouseEvent<HTMLButtonElement>) {
            const buttonNode = e.target as HTMLButtonElement

            const errorNode = buttonNode.parentElement.childNodes[1] as
HTMLParagraphElement

            const inputNameNode =
buttonNode.parentElement.childNodes[2].childNodes[1] as HTMLInputElement
            const inputName: string = inputNameNode.value

            const inputDescriptionNode =
buttonNode.parentElement.childNodes[3].childNodes[1] as HTMLInputElement
            const inputDescription: string = inputDescriptionNode.value

            let errorMsg = "Error(s) detected:"
            let errorFound = false

            const inputNameIsValid = checkForBannedChars(inputName)
            const inputDescriptionIsValid = inputDescription.trim() === "" ?
true : checkForBannedChars(inputDescription)

            if (inputName.trim().length === 0) {
                errorMsg += "\nAssignment Name cannot be blank."
                errorFound = true
                inputNameNode.classList.add('error')
            } else if (!inputNameIsValid) {
                errorMsg += "\nAssignment Name cannot contain any of the
following characters:\n;  :  <  >  (  )  {  }  [  ]  *  %"
                errorFound = true
                inputNameNode.classList.add('error')
            } else {inputNameNode.classList.remove('error')}

            if (!inputDescriptionIsValid) {
                errorMsg += "\nDescription cannot contain any of the following
characters:\n;  :  <  >  (  )  {  }  [  ]  *  %"
                errorFound = true
                inputDescriptionNode.classList.add('error')
            } else {inputDescriptionNode.classList.remove('error')}


            if (errorFound) {
                errorNode.innerText = errorMsg
            } else {
                window.class.createClass(
                    inputName,
                    inputDescription
                ).then(newClassID => resolve(newClassID))

                popup.closePopup()
            }

        }
    })
}

declare global {

    interface AddNewClass_exports {

        /**
         * Triggers a popup to create a new class.
         * @returns A promise which resolves with the new class_id.
         */
        trigger: () => Promise<number>
    }
}
export default {
    trigger
} as AddNewClass_exports
```

newClassPopup.scss

```
@import '../../colors';

.new_class_popup {
    display: flex;
    flex-direction: column;
    width: 454px;

    h1 {
        font-size: 20px;
        text-align: center;
    }

    .error_message {
        font-size: 12px;
        text-align: center;
        color: $error-text;
        margin: 0 0 15px 0;
    }

    .field_container {
        display: flex;
        flex-direction: row;

        input[type='text'], input[type='number'], textarea {
            resize: none;
            width: 300px;
            margin: 4px;
            border: 2px solid $background;
            background-color: $color2-lighter;
            border-radius: 8px;
            padding: 5px;
            outline: none;

            &:hover,
            &:focus {
                background-color: $color2-lighter2;
            }
            &.error {
                border: 2px solid $error-border;
                background-color: $error-fill;
            }
        }
        textarea {
            height: 100px;
            font-family: Arial, Helvetica, sans-serif
        }

        label {
            margin-right: 6px;
            width: 140px;
            transform: translateY(5px);
            line-height: 16px;

            .subtext {
                font-size: 12px;
                font-style:italic;
            }
        }

        &.radio {
            margin-top: 10px;

            label {
                width: fit-content;
                transform: none;


                &.fieldname {
                    margin-right: 6px;
                    width: 140px;
                    transform: translateY(-1px);
                }
                &[for='TEST'] {
                    margin-left: 65px;
                }
            }

            input {
                scale: 150%;
                cursor: pointer;
                outline: 2px red;

                &.error::before {
                    content: "";
                    display: inline-block;
                    width: 16px;
                    height: 16px;
                    transform: translate(-1.6px,-1.6px);
                    background-color: #ff000054;
                    border-radius: 50%;
                }
            }
        }

        &.extra_credit {
            margin-top: 15px;
            margin-bottom: 10px;

            label {
                transform: translateY(-1px);
            }

            input {
                scale: 150%;
                cursor: pointer;
            }
        }
    }

    button {
        margin: 10px auto 0px auto;
    }

}
```

studentManagerPopup.tsx

```
import React from 'react'
import './studentManagerPopup.scss'
import popup from '../../Popup/popup'
import editStudentPopup from './editStudentPopup'
import createStudentPopup from './createStudentPopup'


function trigger() {
    return new Promise<void>(async (resolve, reject) => {
        try {
            const stuData = await window.student.getStudentList()
            console.log(stuData)

            function editStudent(student_id: number) {
                editStudentPopup.trigger(student_id)
                    .then(() => {
                        trigger()
                    })
                    .catch(() => {
                        trigger()
                    })
            }
            function handleCreateStudent() {
                createStudentPopup.trigger()
                    .then(() => {
                        console.log('student created')
                        trigger()
                    })
                    .catch(() => {
                        trigger()
                    })
            }

            const stuListDisplay = stuData.map((stu, stuIndex) => {
                return (
                    <React.Fragment key={stu.student_id}>
                        {stuIndex !== 0 && <hr />}
                        <div className='student'>
                            <span>{stu.first_name} {stu.last_name}</span>
                            <button onClick={() =>
editStudent(stu.student_id)}>Edit</button>
                        </div>
                    </React.Fragment>
                )
            })

            const content =
                <div className='student_manager_popup'>
                    <h3>Student Manager</h3>
                    <div className='student_list'>
                        {stuListDisplay}
                    </div>
                    <button onClick={handleCreateStudent}>Create New
Student</button>
                </div>
            popup.triggerPopup(content, null, () => {resolve()})
        } catch {
            reject()
            return
        }

    })
}

declare global {

    interface StudentManager_exports {

        /**
         * Triggers a popup to edit a student's enrollment.
         * @returns A promise that resolves if changes are made, or rejects
otherwise.
         */
        trigger: () => Promise<void>
    }
}
export default {
    trigger
} as StudentManager_exports
```

studentManagerPopup.scss

```
.student_manager_popup {
    text-align: center;
    min-width: 400px;

    & > button {
        margin: 10px auto 0 auto
    }

    .student_list {
        min-width: 300px;
        display: flex;
        flex-direction: column;
        max-height: 70vh;
        overflow-y: auto;
        overflow-x: hidden;

        .student {
            display: flex;
            align-items: center;

            span {
                margin-right: 5px;
            }

            button {
                margin-left: auto;
            }
        }
        hr {
            width: 100%;
        }
    }
}
```