**Criterion C: Development**

**Ways through which I identified where bugs happened- when they did:**

1. Putting an alert in the beginning of a function, to see if the code enters it.
2. Checking issues using the “Console” and “Network” tabs, after pressing CTRL+SHIFT+I.
3. Running a piece of code individually, to test if it is the one containing the problem.
4. Consulting online sources (they can be found in Appendix 4).
5. Structuring the code in a different way, but following the same concept.

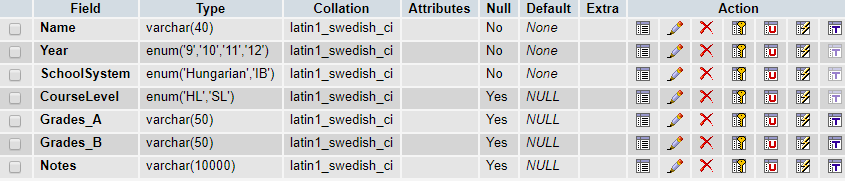
**Explanation of code:**

The database “credentials” table, storing username and password (type varchar):



For the password, hashing was used, thus in case the database is breached, the password will not be “stolen”. The password below in the database’s “credentials” table is hashed (i.e. not the real password):

The “students” table, storing student credentials:

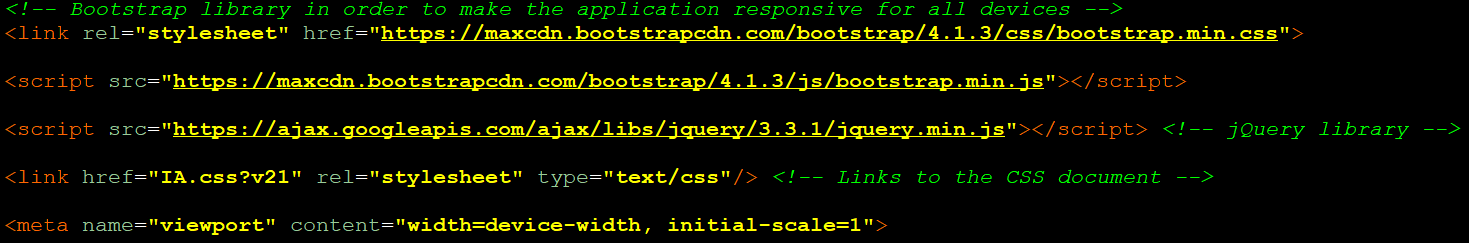


Students’ names are type varchar. School system, year and course level are type enum, having only two, and three valid choices respectively. Grades and notes are type varchar, storing up to 10000 characters for convenience. This way the user, is aware of the possibilities/limitations of the fields (SC5).

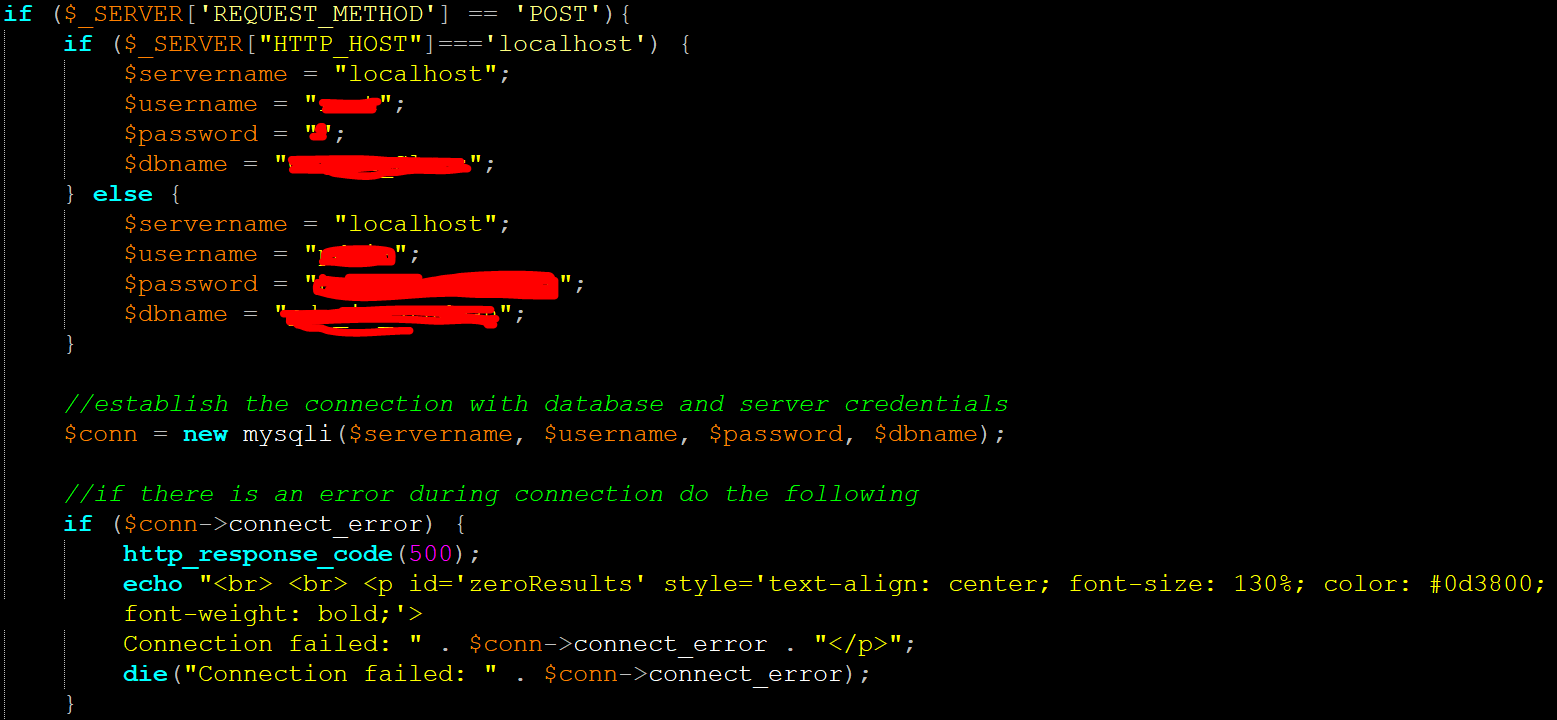
There are 6 linked documents of code, separated by purpose for distinguishability. They contain:

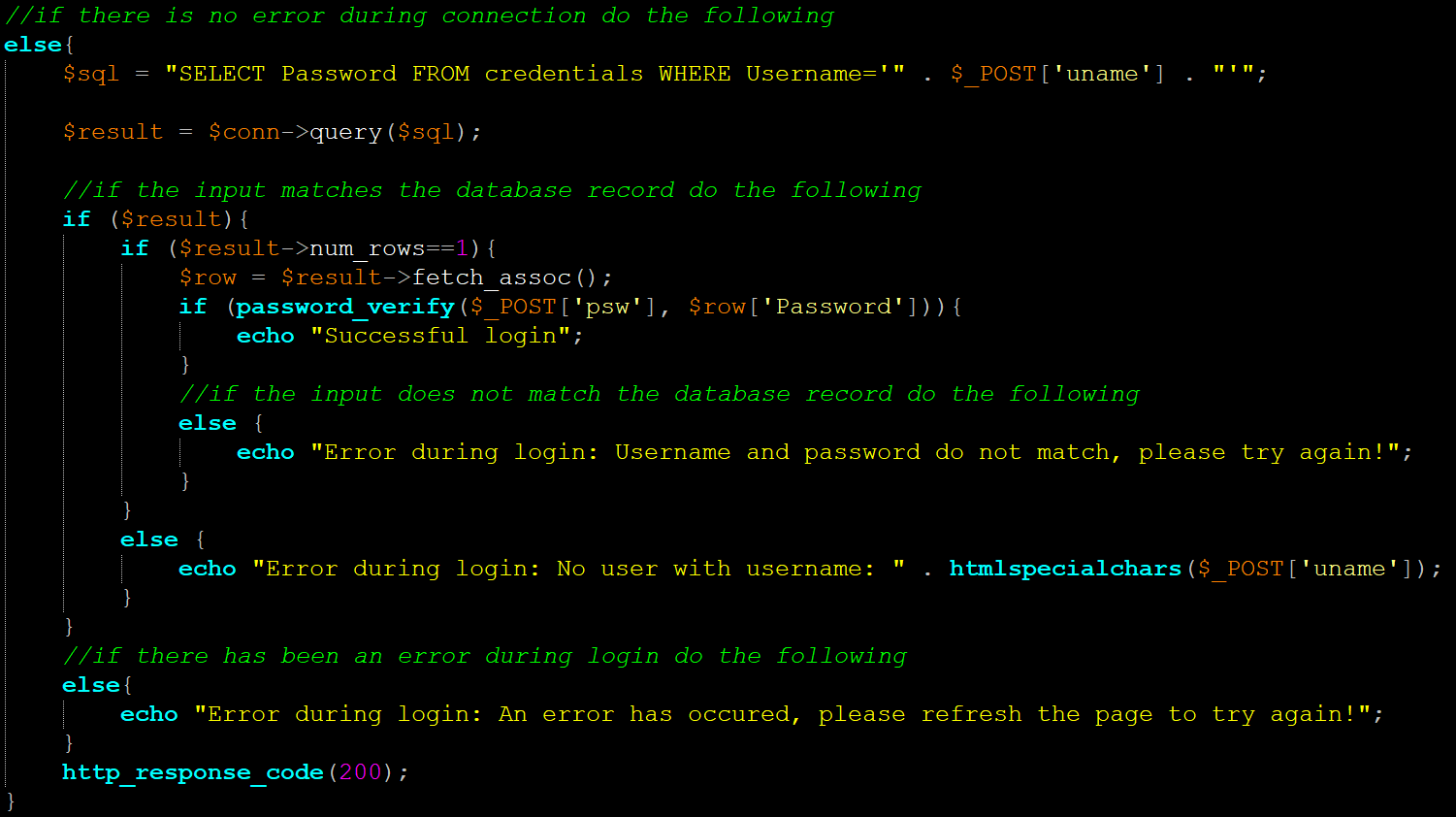
* IA.css: CSS code
* IA.php: HTML code
* IA\_scripts.js: JavaScript code
* login.php: login-related PHP code
* getstudents.php: search/edit-related PHP code
* addRemoveRecord.php: add/remove-related PHP code

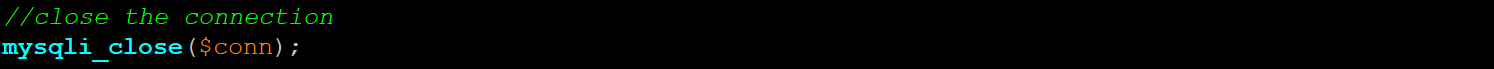


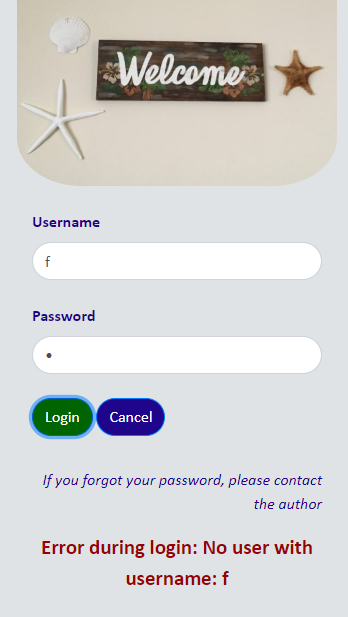
The application is responsive and adapts depending on the accessing device (achieved using Bootstrap classes). AJAX and JQuery were used too. The links to the libraries, CSS, JavaScript and PHP documents respectively:



For the login (SC2), PHP was used to connect to the database and execute SQL commands. It is feasible to compare the login inputs’ values to the respective ones in the database table. To learn SQL and relate it to PHP, I studied the sources “*MySQL by Examples for Beginners”* by *ntu.edu, 2014* and everything in the section “*mysqli”* by *php.net, n.d.*[[1]](#footnote-1).Following is the PHP code for connecting to the database. The same method is used later (but not explained again):

Noteworthy are the usages of htmlspecialchars() and mysqli\_real\_escape\_string() to prevent malicious inputs. If the connection is successful:

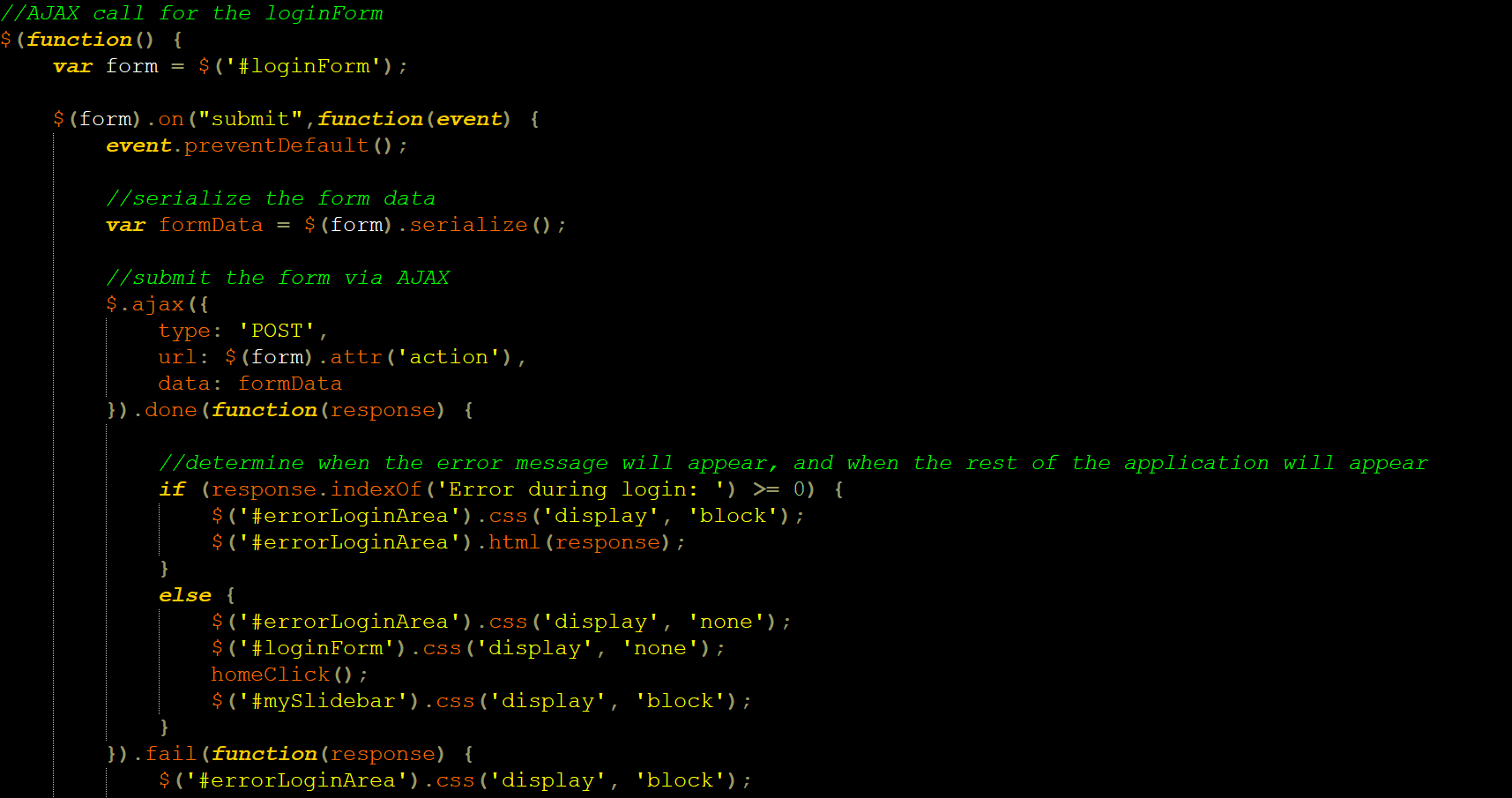


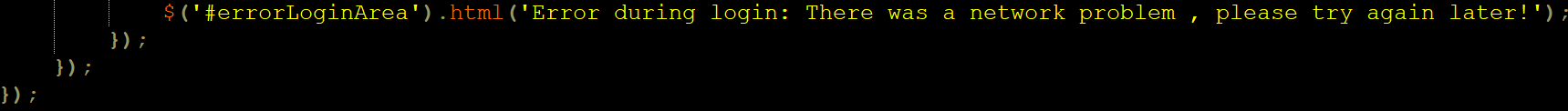


**Login error:**



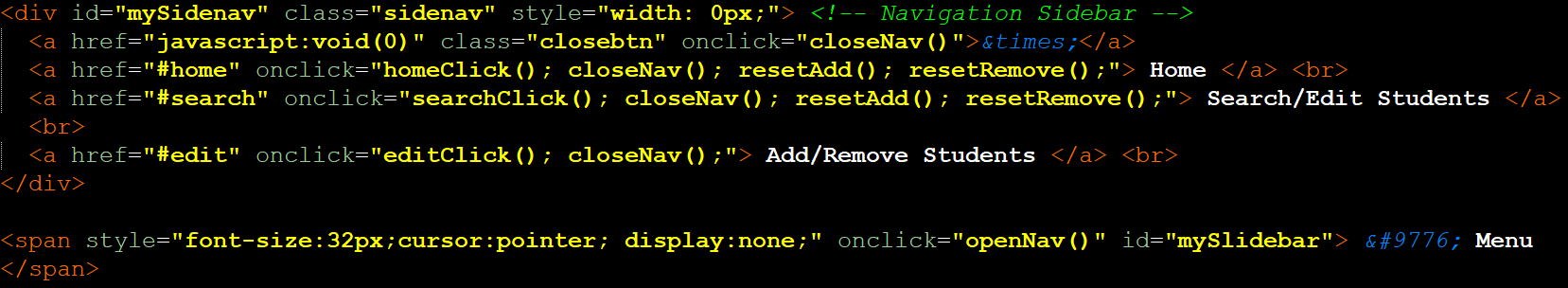
AJAX is used to submit forms asynchronously, at the background. If the login is not successful, the response will contain “Error during login”. If it does, the rest of the application remains invisible and the informing error message appears (SC8). If not, the rest are made visible [source for AJAX form submissions: “*How to Create an AJAX Contact Form*”by *West, 2013*].

AJAX for loginForm submission:

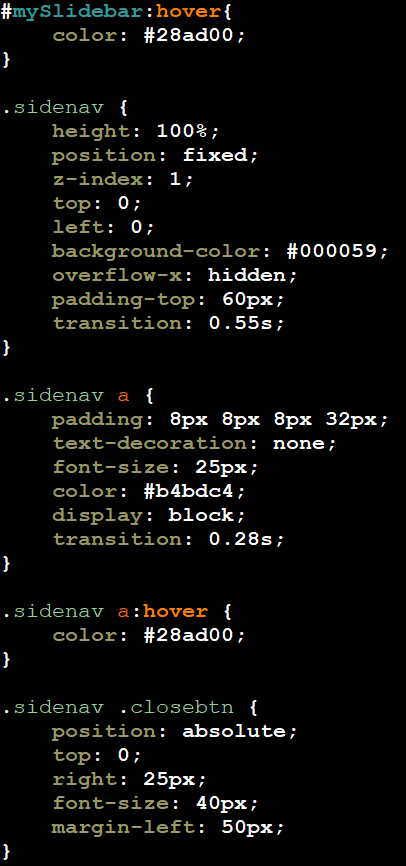


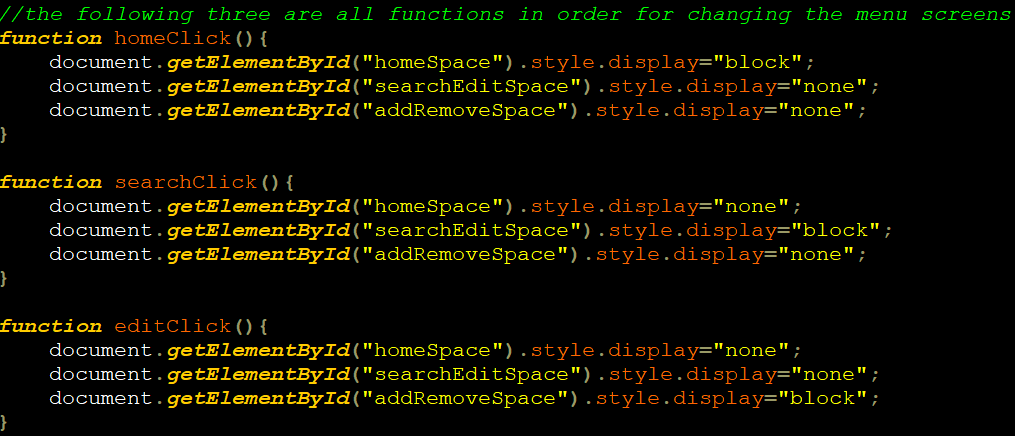
The homeClick() is explained in the “menu” section. The preventDefault() method is used to stop the browser from reloading the page, creating a Single Page Application.

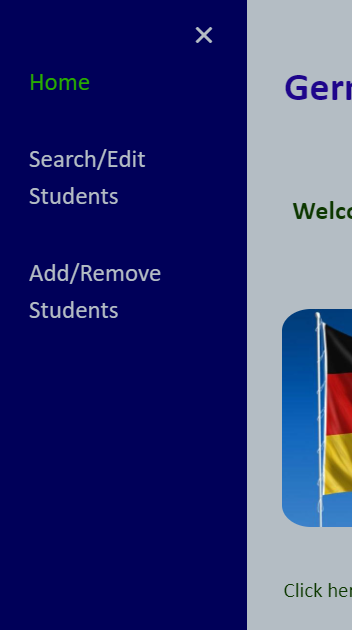
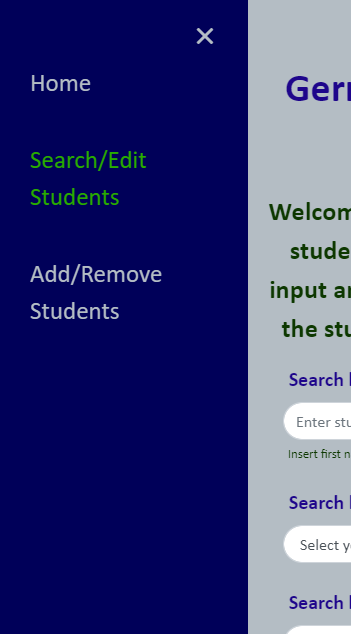
The menu separates the application into sections (SC4) [source for menu:*“How TO - Side Navigation”* by *w3schools, n.d.*].

HTML:

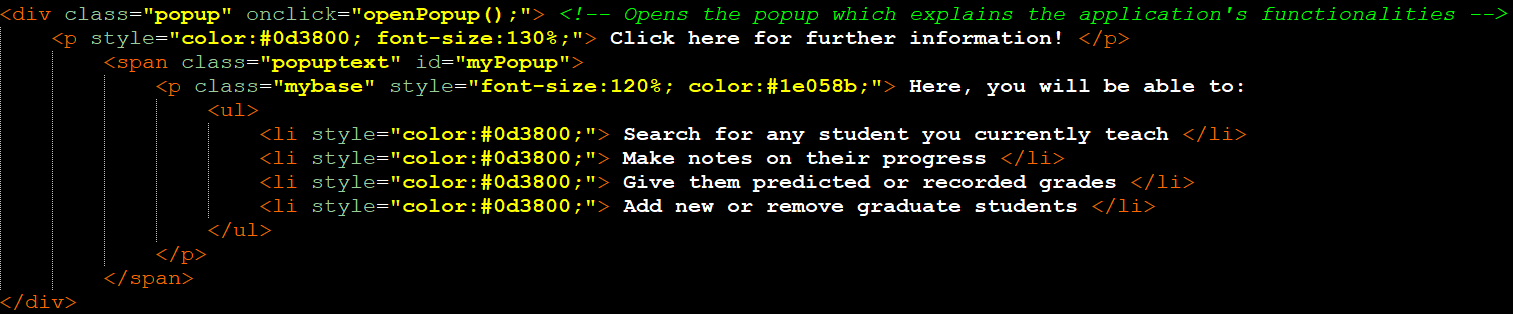
CSS:



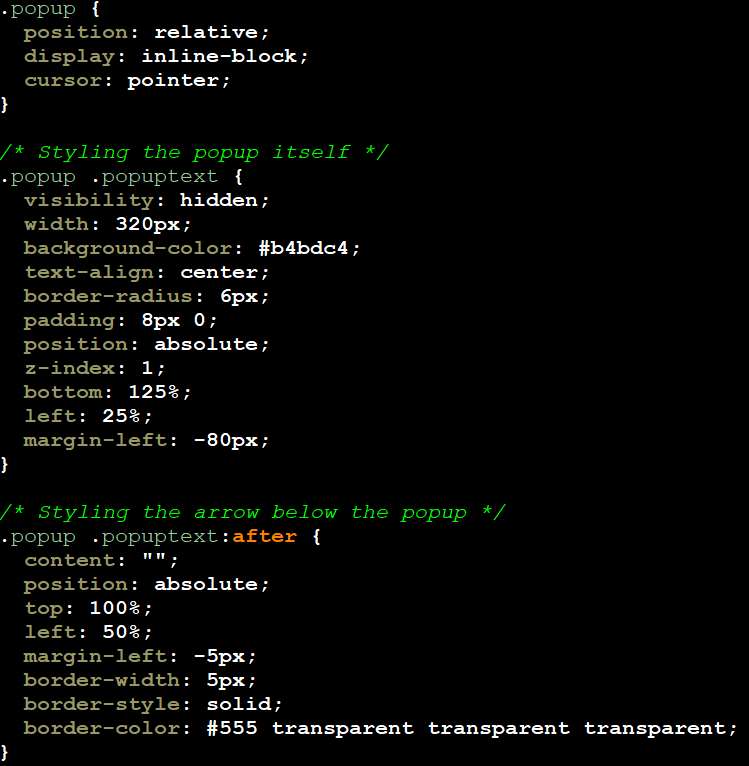
JavaScript functions, called onclick in the corresponding menu link:

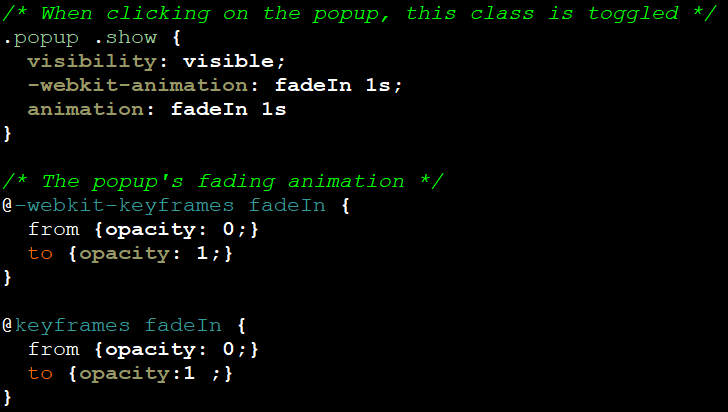
**Menu:**

There exists a popup with instructions in home screen (SC5) [source for popup: *“How TO - Popup”* by *w3schools.com, n.d.*]. The popup appears when a particular sentence is clicked, and fades given the sentence is re-clicked.

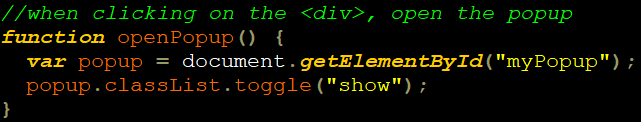
HTML part:

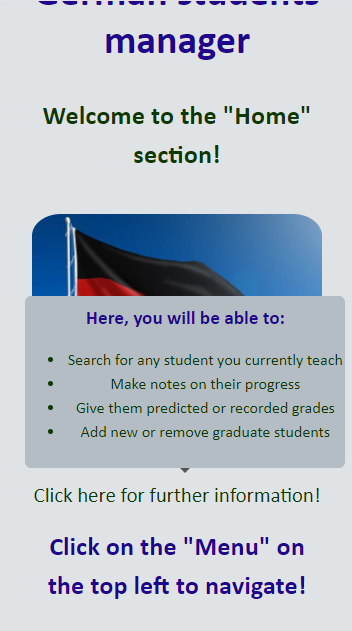
CSS part:





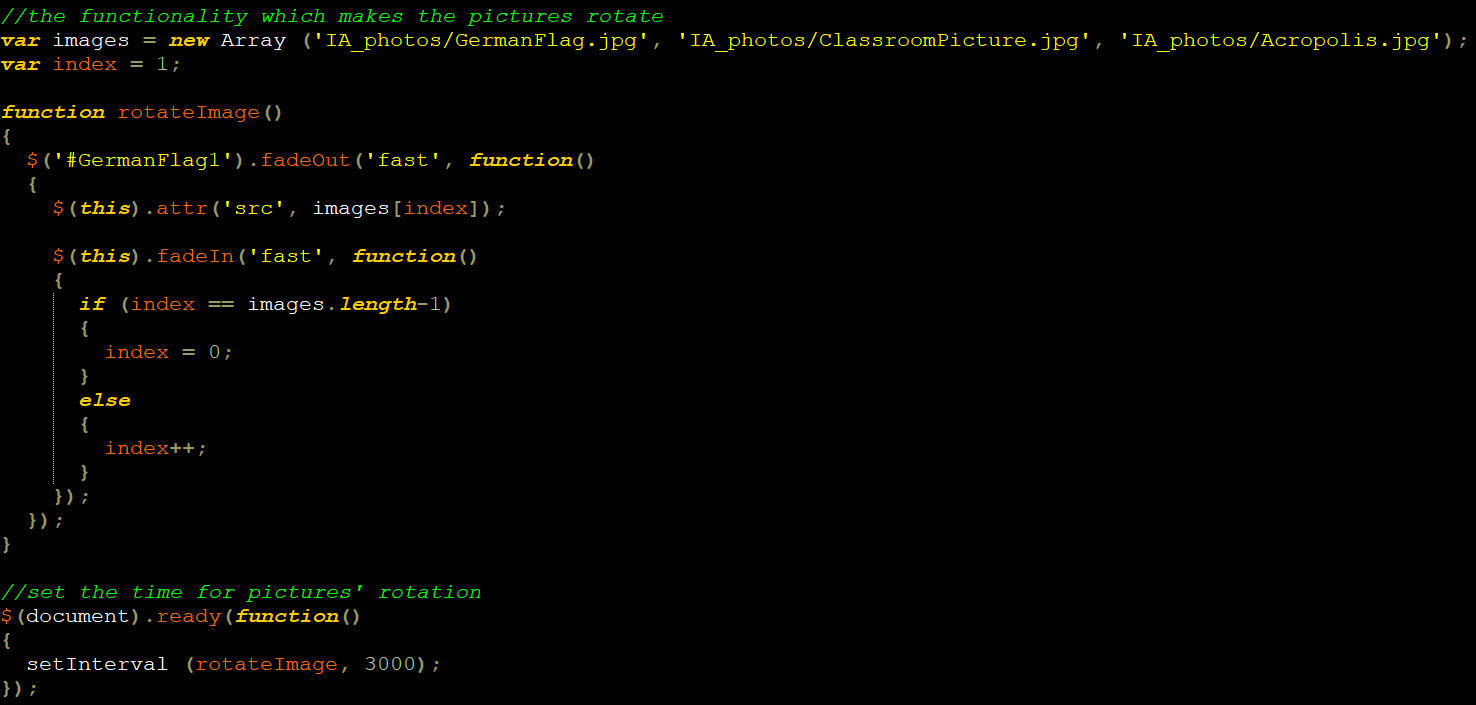
JavaScript part:



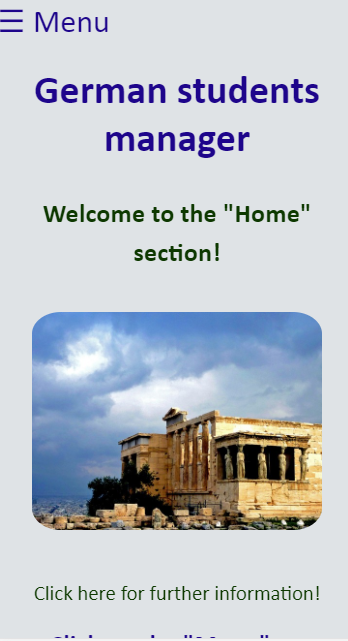
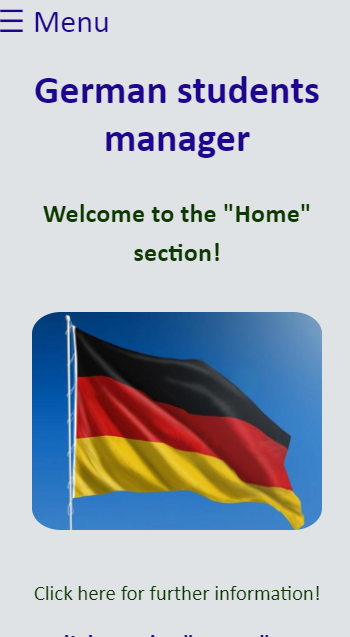


**Popup:**

Here are also 3 rotating pictures [source for image rotator: *“how to create a simple automatic image rotator using jQuery”* by *burnmind, 2010*].

JavaScript code:

An array holds the pictures (with same dimensions). There is a first existing picture (German flag), which fades out after 3 seconds, getting replaced by the next one in the array and so on.

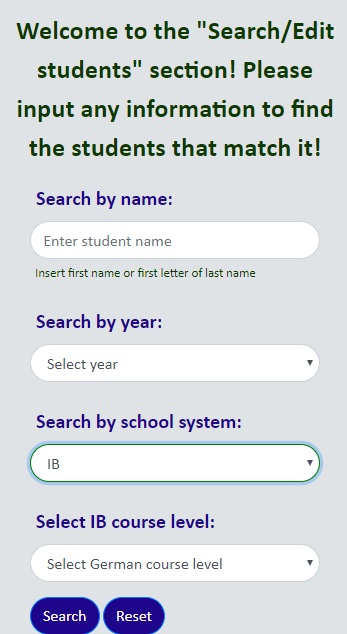
**Images:**



There is also a screen for searching and editing students (SC6, 9). Search inputs are created with HTML. Three inputs (name, year, school system) are always visible. While the school system is “IB”, an additional “course level” input appears.

Following is the JavaScript code. It is dynamic because the same functionality is used in the add/remove screen, but will not be re-explained:



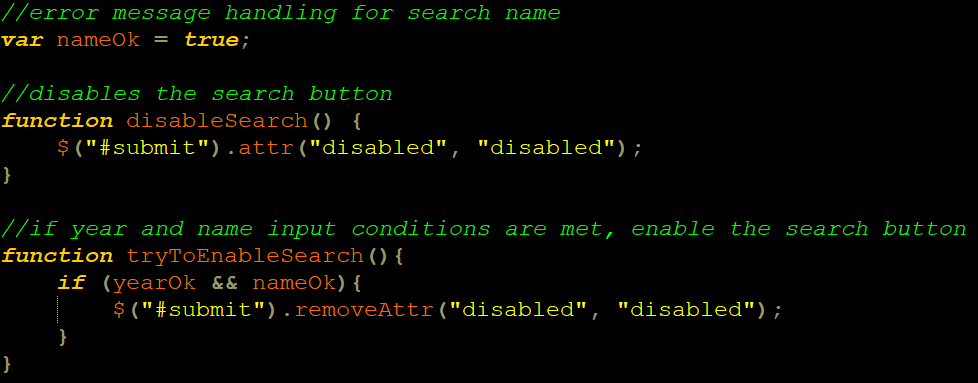


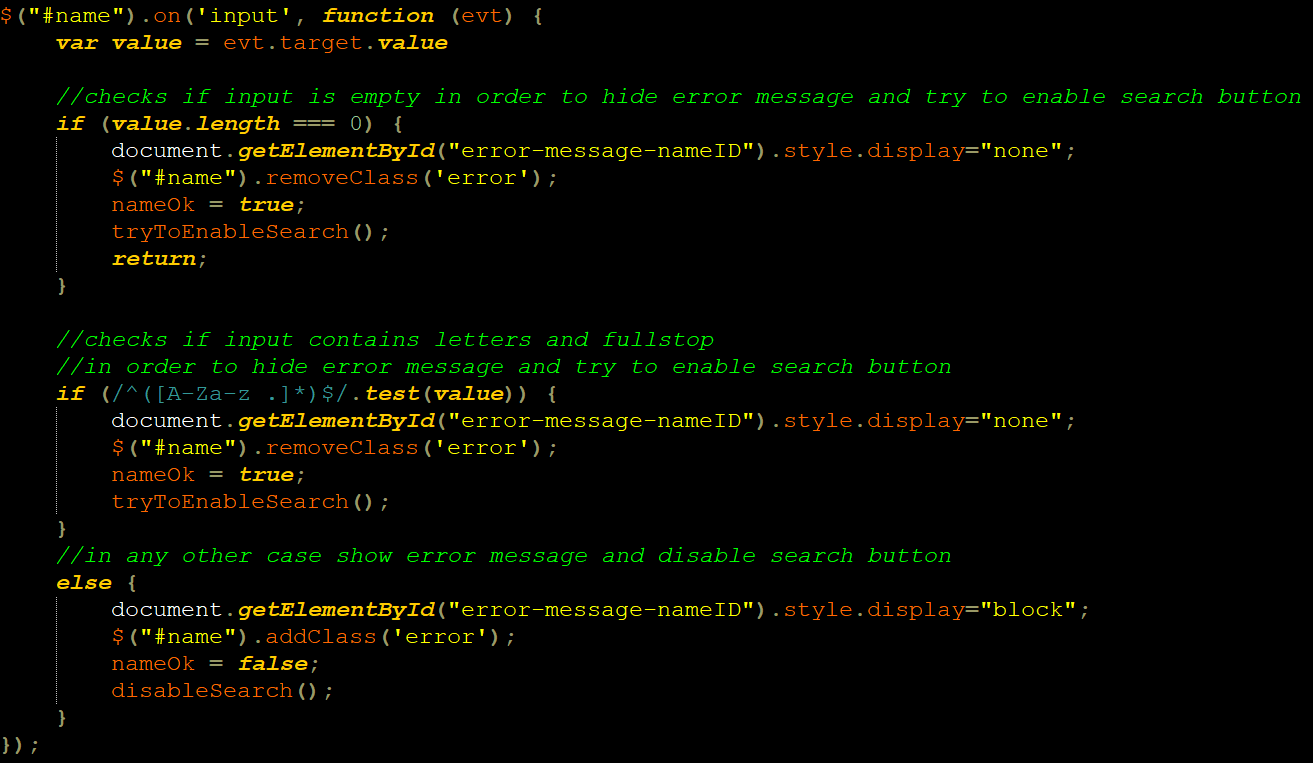
**“Course level” appears only while the school system is “IB”:**

There are two cases where submission is prevented. The user is then informed (SC8):

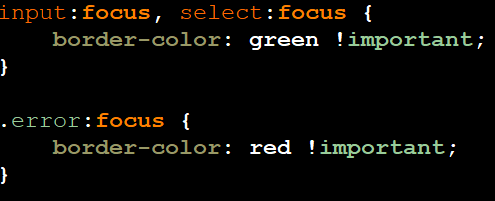
1. Something other than letters/full-stops/spaces is inputted as name.

2. All inputs are empty.

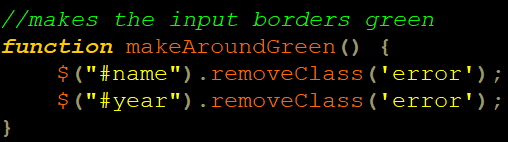
Following is the JavaScript codes for all cases. Matching concepts are used for the add/remove screen, but will not be re-explained. The first case disables the search button and adds the “error” class to the input. I consulted the source “*How to check contents of input in “real time””* in *stackoverflow* by *Koops, 2017*, expanding on the code:



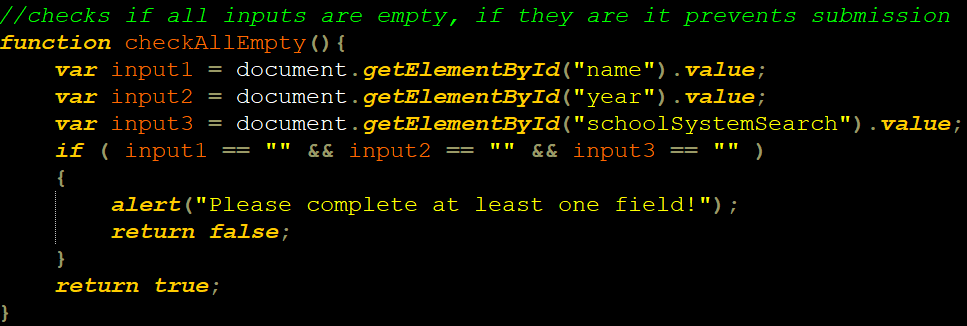
All inputs’ borders are originally green while focused, but red if they contain the class “error”. CSS code:

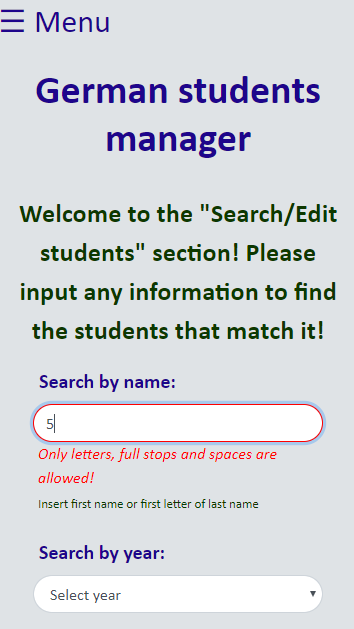


The following JavaScript function, and other simple ones, are called upon clicking the reset button:



Following the JavaScript code for the third case, showing an alert and preventing form submission:

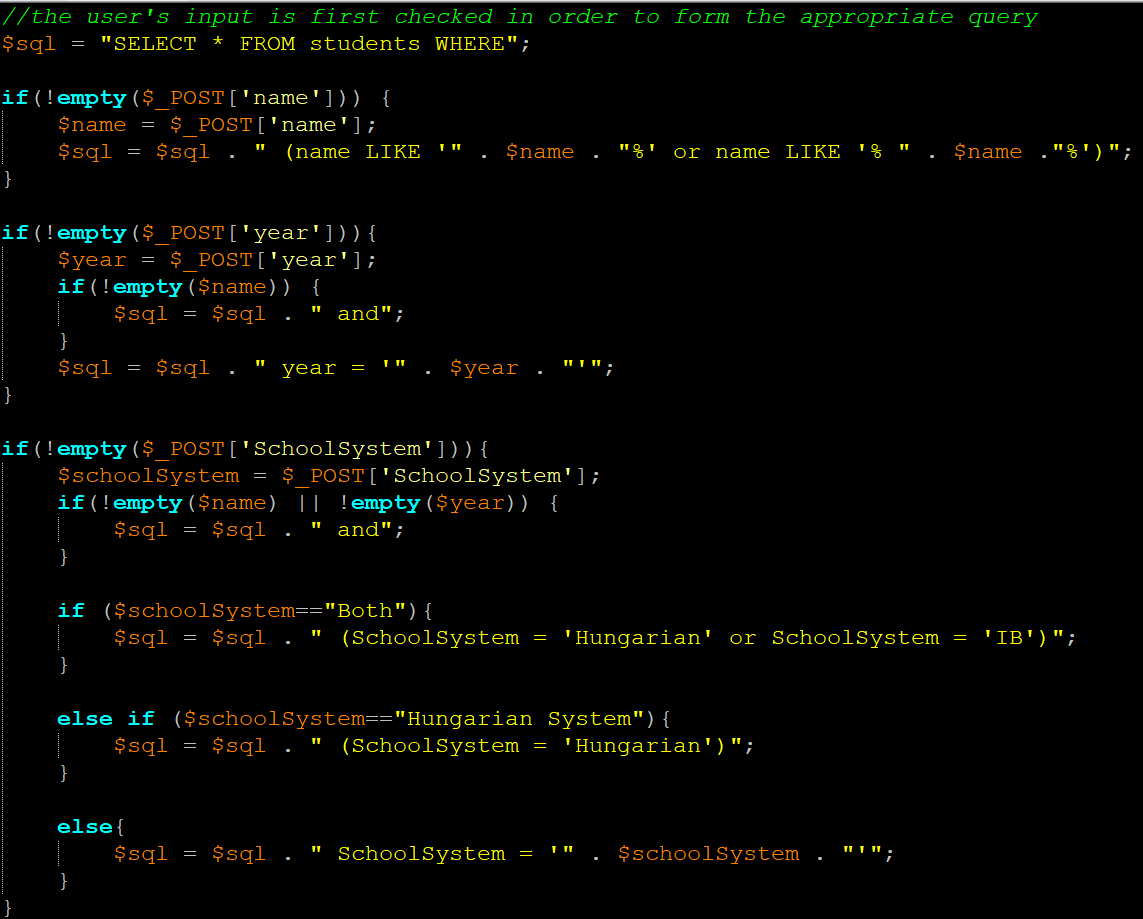


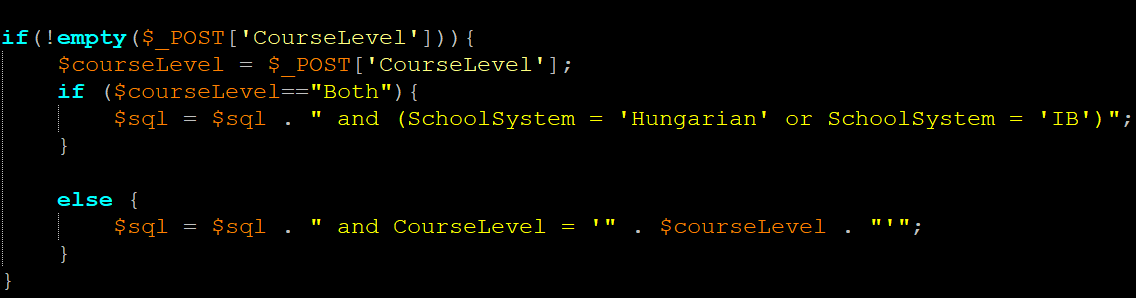
****

**When wrong input is given**

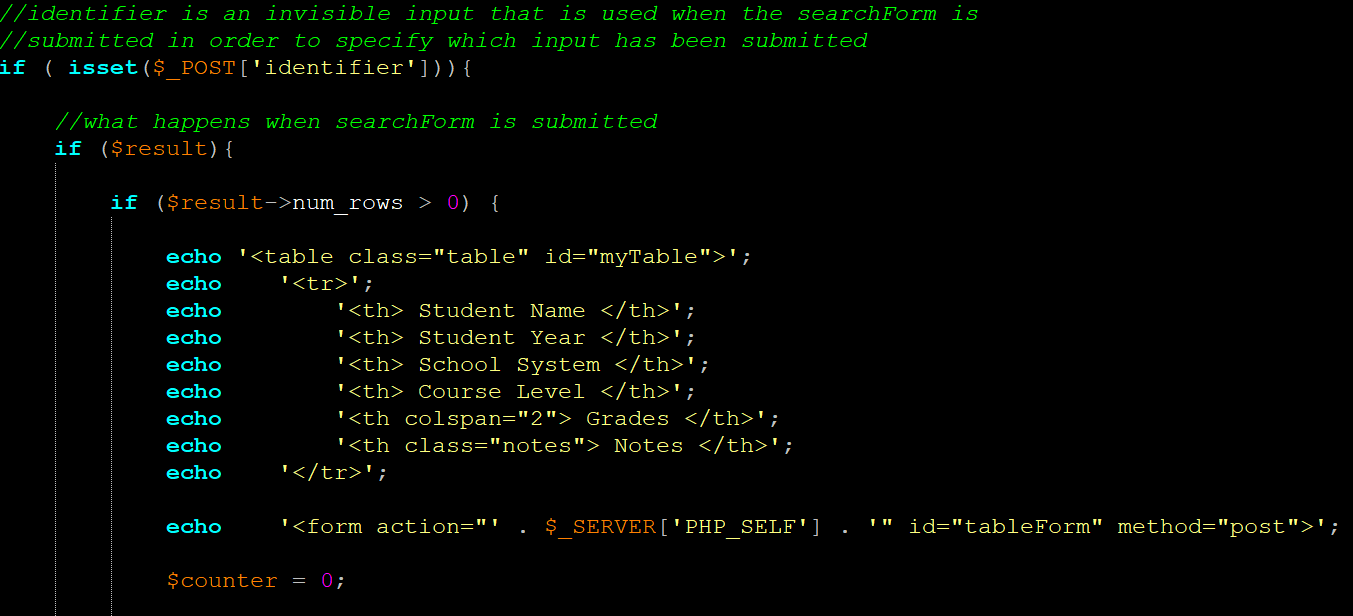
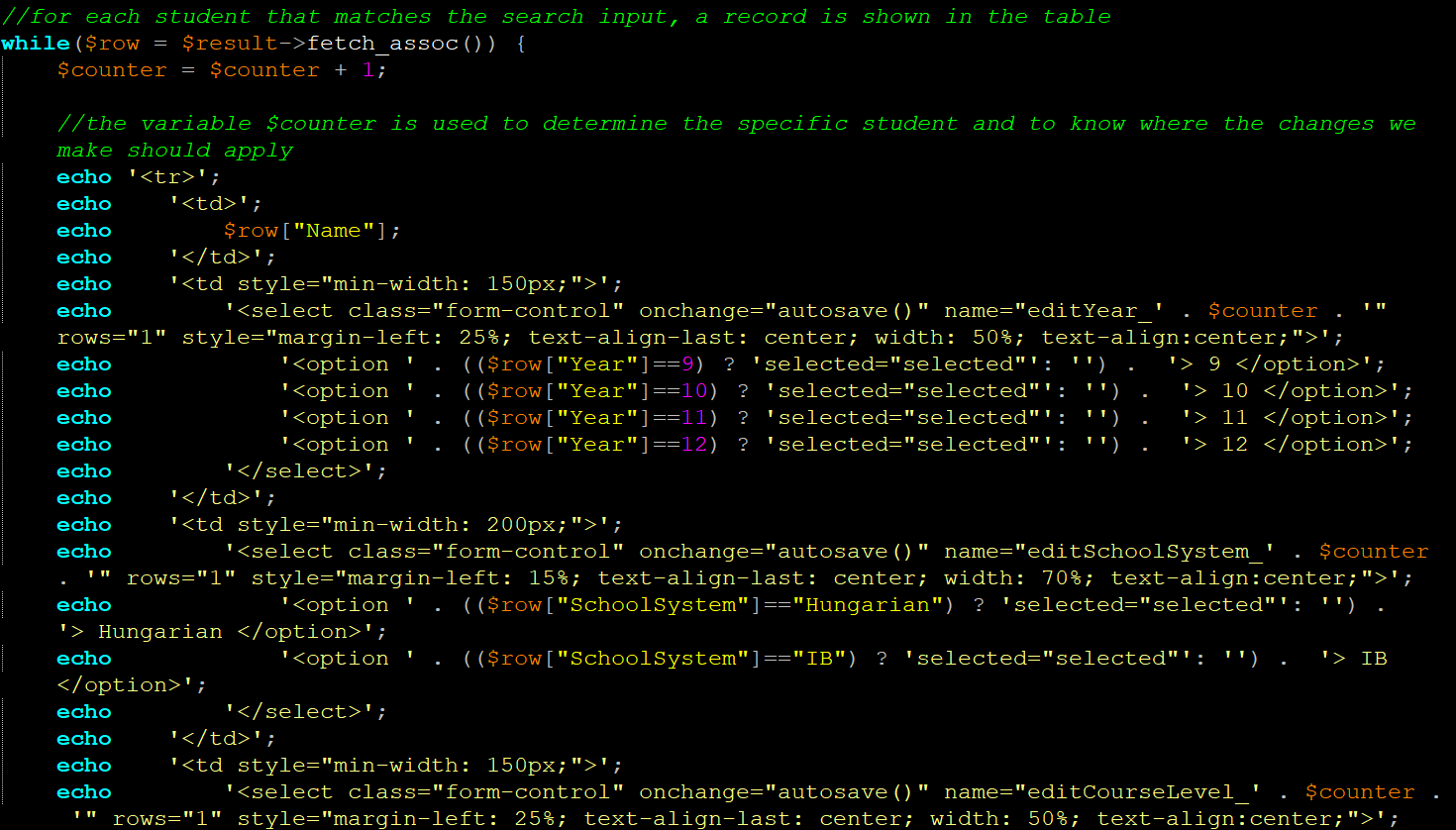
**as name:**

There exist one static form for searching, and one dynamically generated for editing. For the search form, through conditions, a variable $sql takes the respective value in the form of an SQL command, then executes the query. PHP code:





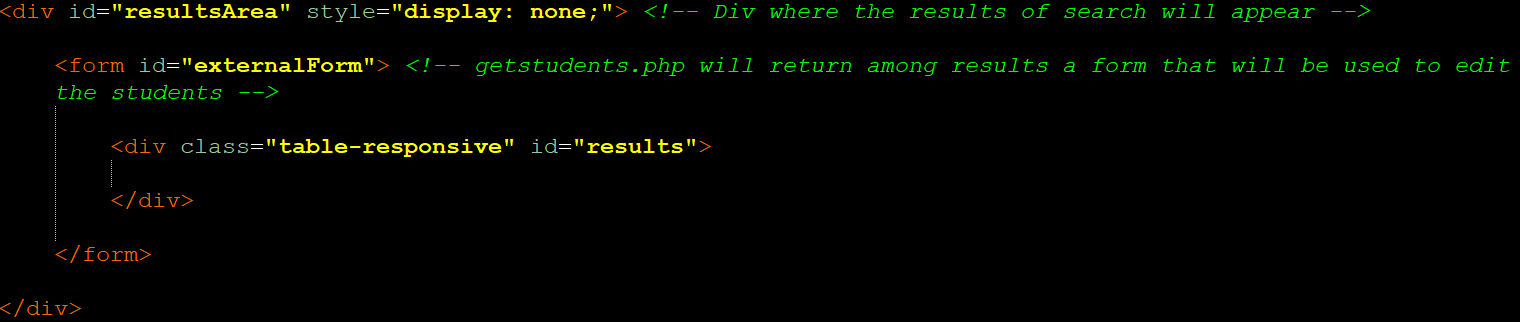
To specify which of the two forms has been submitted, an invisible input with name “identifier” is used. If this input is set, and the variable $result has a value, the table is dynamically generated using $counter. PHP code:



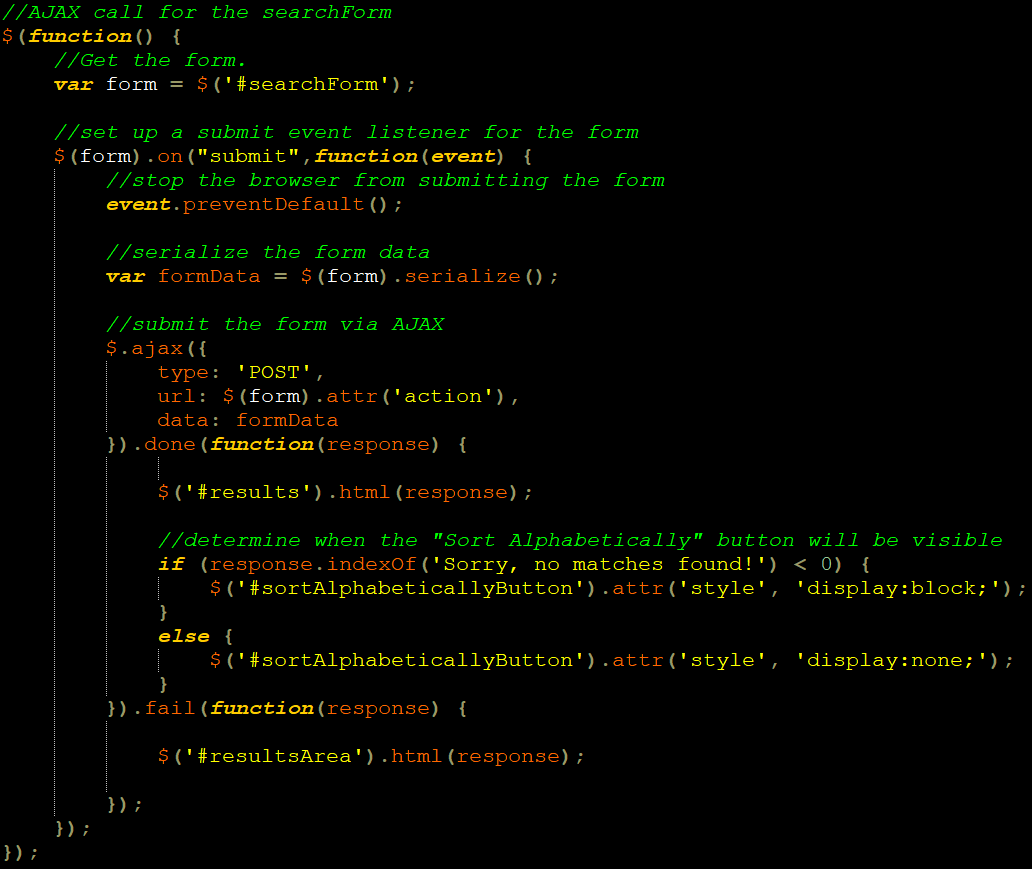
The $counter will always add 1 until there are no students left that match the search, only printing relevant results. If no results are found, a message appears stating this. The $counter is also used for editing to specify which record we are referring to.

Using the AJAX technique described before, the table appears in the “results” div:

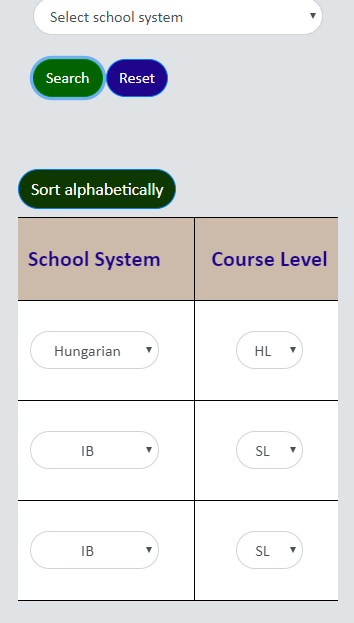
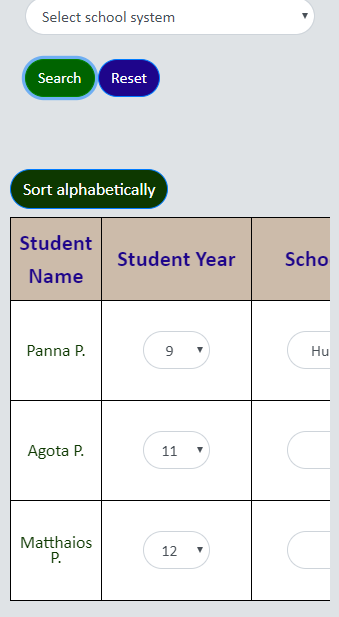
HTML:

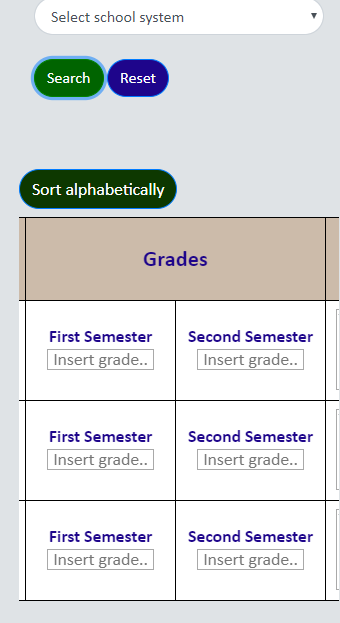
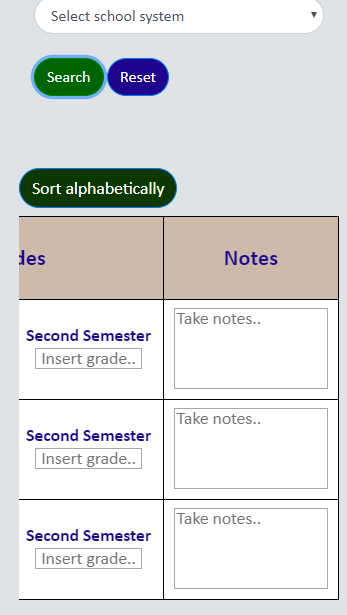


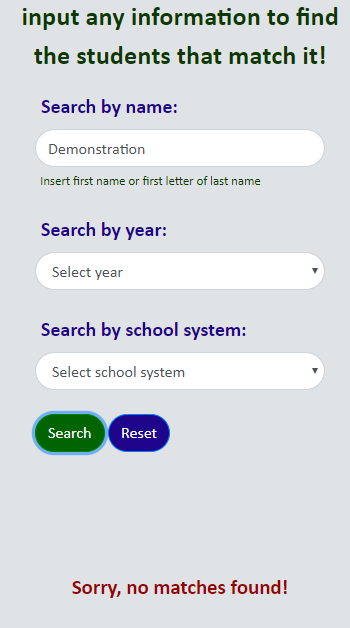
AJAX for searchForm submission:

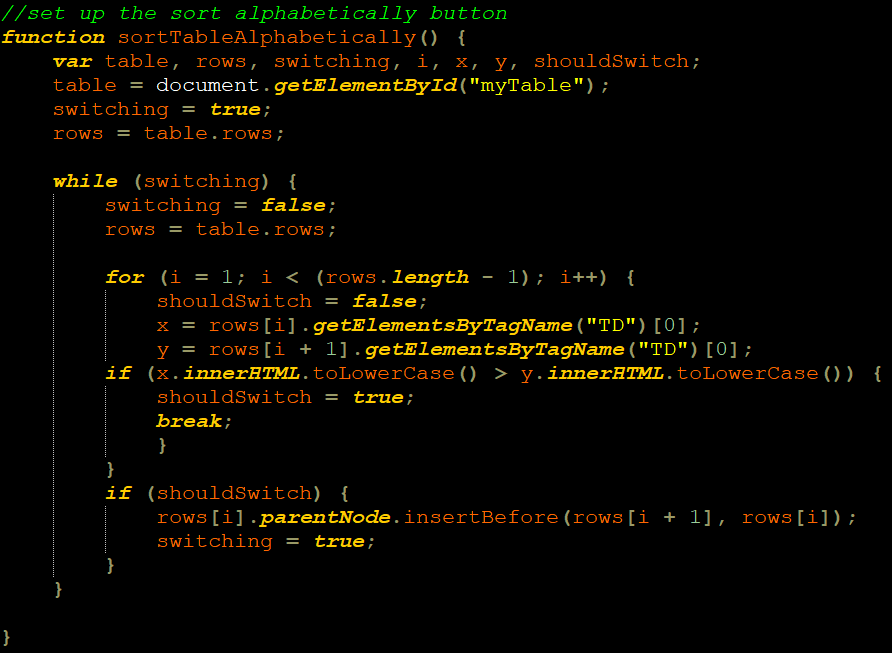


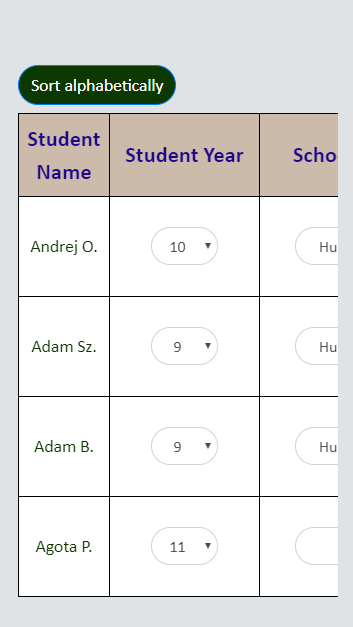
Only the students’ first name and the first letter of last name are present, for privacy purposes. **Demonstrations (the table is scrollable):**



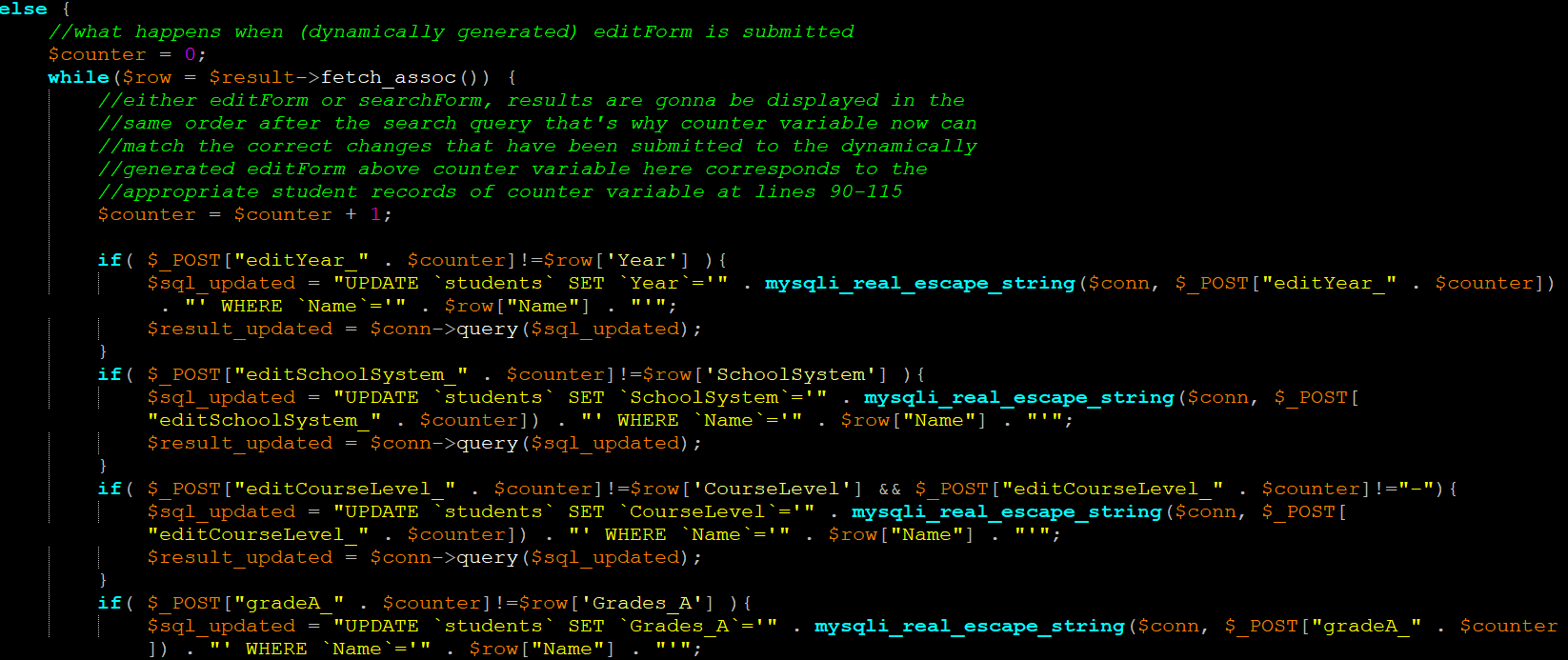


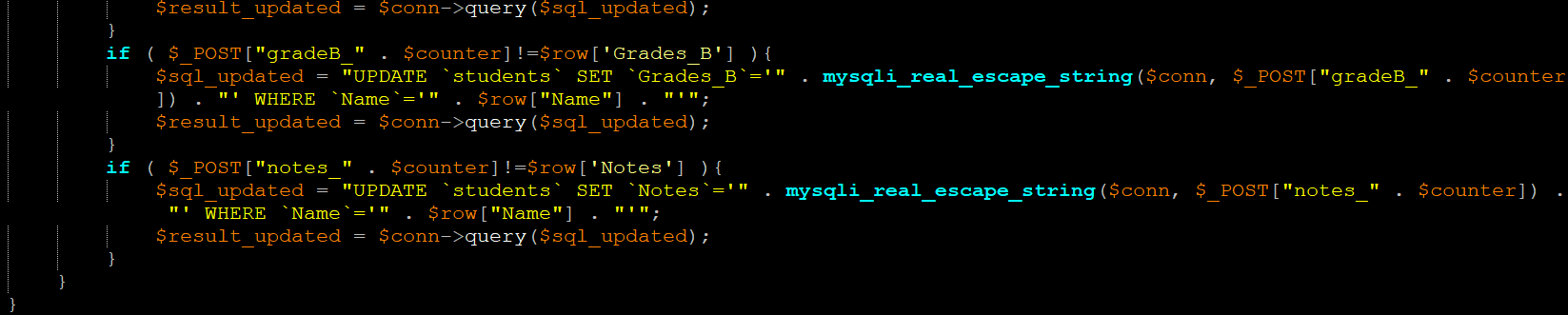
**Error:**

The user can sort results alphabetically [source for sorting: “*How TO - Sort a Table*” by *w3schools, n.d.*]. JavaScript code:

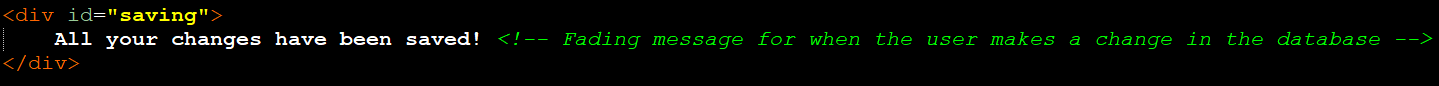
**After clicking**

**“Sort alphabetically” button:**

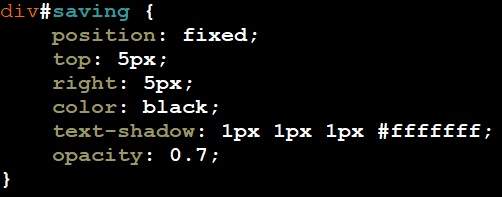
There is the possibility to edit records (SC9). It is checked whether a value in the application’s table is identical to the respective value in the database. This is done to prevent overloading the database with unnecessary query executions. If not, the variable $sql\_updated takes the respective SQL command value and executes update query. Following is the PHP code executed after submitting the “externalForm”, which is the form used for editing:

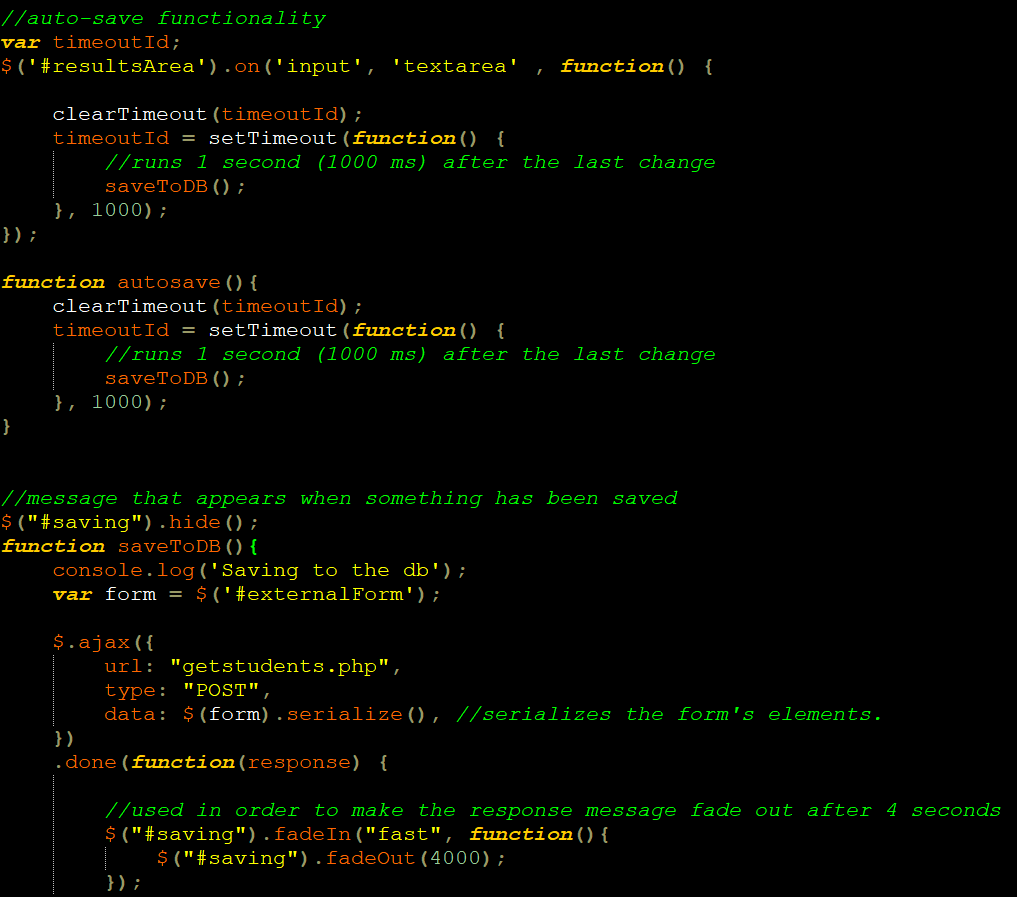


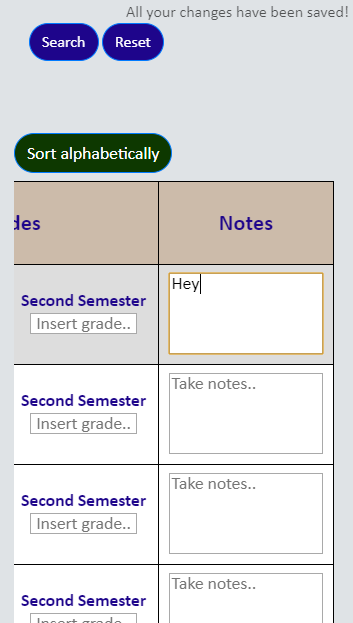
After any updates, the user is notified with a fading notification (SC7) [source for fading functionality: “*jQuery Effects - Fading*” by *w3schools, n.d.*]. It appears when something has been inputted in any textarea of the resultsArea (area where table textareas are).

Notification’s HTML:

Notification’s CSS:

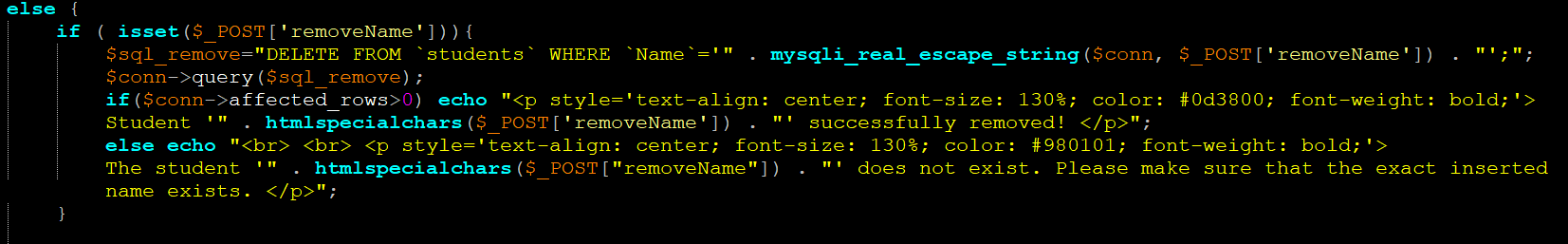


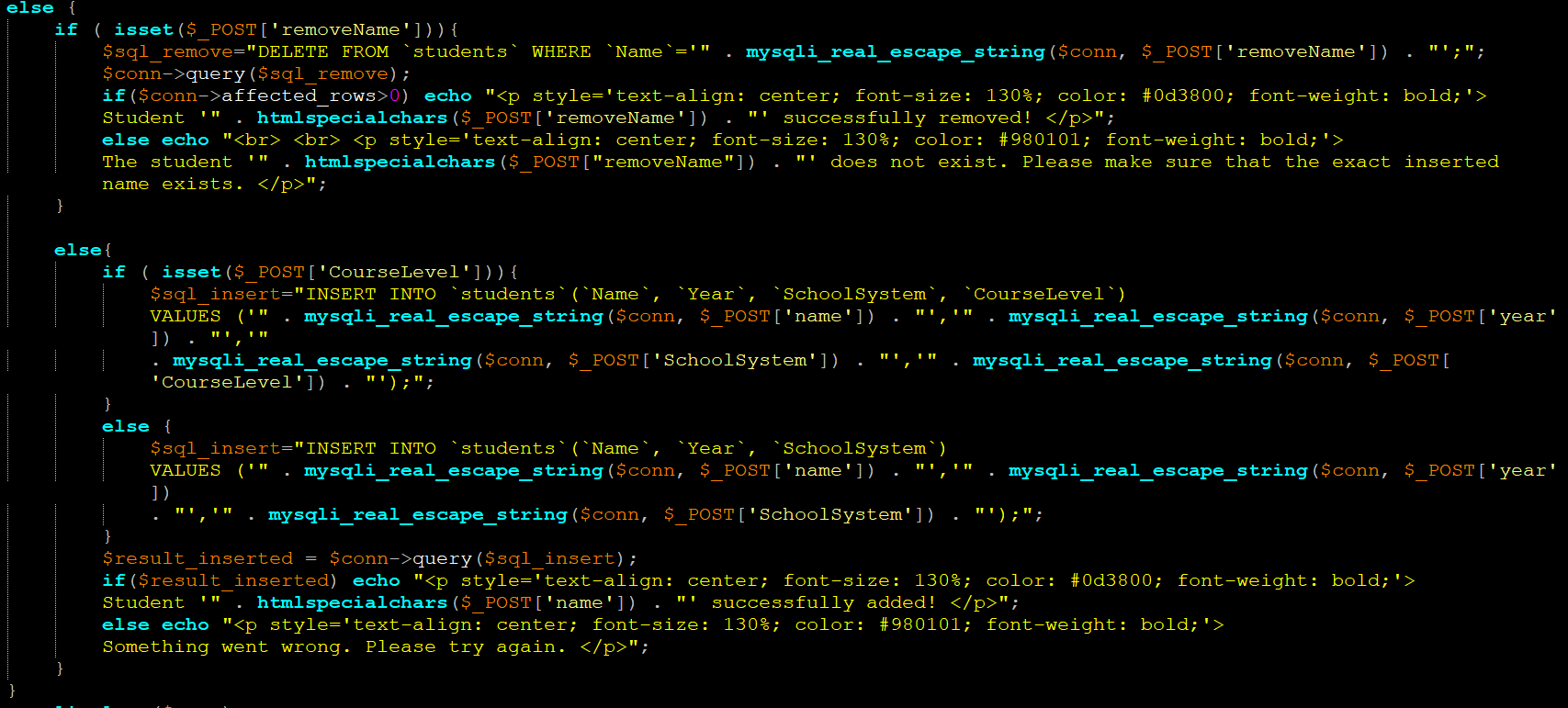
As shown above, “externalForm” is the form used for editing [source for auto-save: “*Save 1 secon* *d after Form Change”* by *Eric, n.d.*]. The JavaScript code:

**Editing and message on top right:**

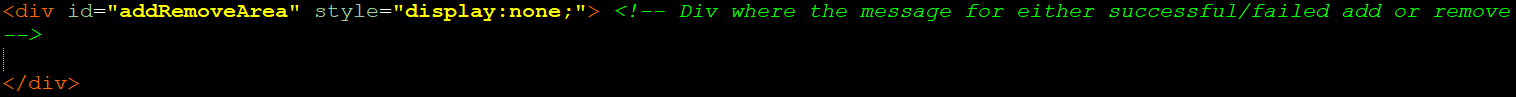
**Edit successful:**



The user can remove/add students (SC11, 12). PHP codes, in respective order:

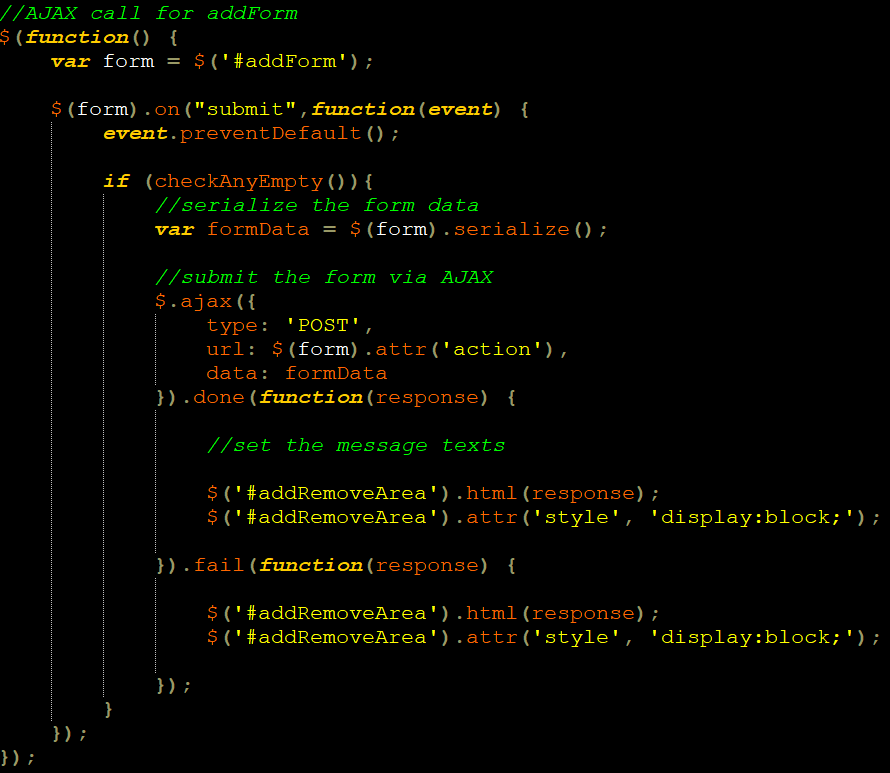


If the “removeName” input is set, removal code is executed. If the “courseLevel” input is set, adding code is executed. The reason is that “removeName” is the only input provided when removing students. If students are added/removed or if something goes wrong, the user is informed. Since this may realistically happen, addition of students whose credentials already exist in the database is allowed. Here, aforementioned techniques are used, making the “course level” option appear only while the “school system” is IB, and disabling the add/remove buttons while any input is invalid.

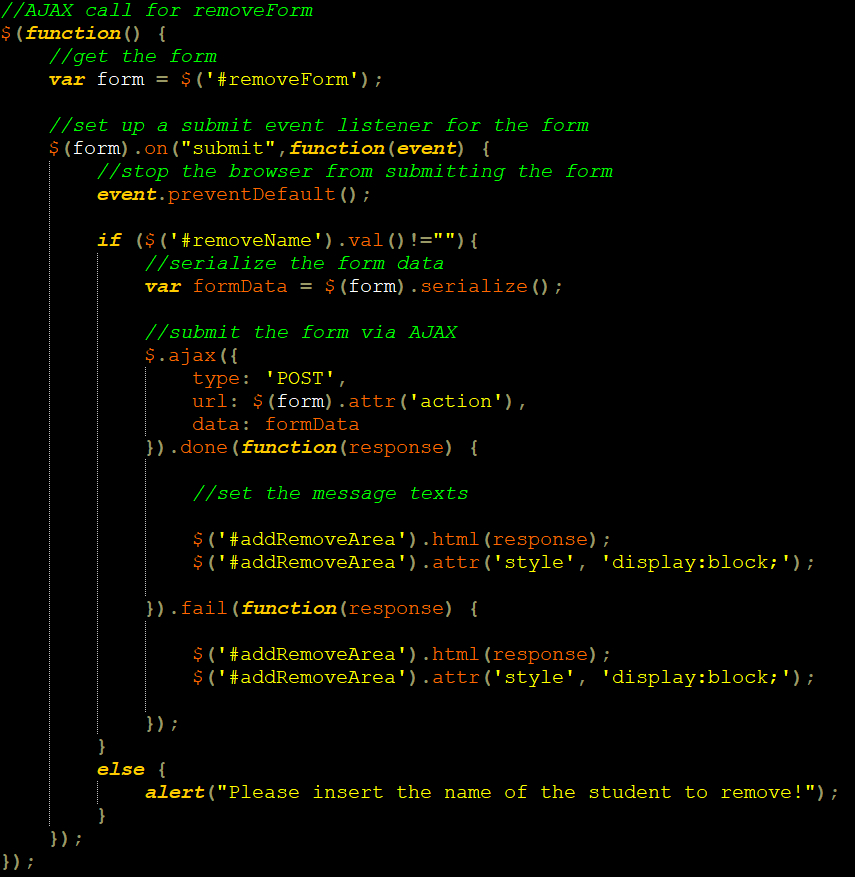
The notifications of add/remove success/failure will always appear in the “addRemoveArea” div:

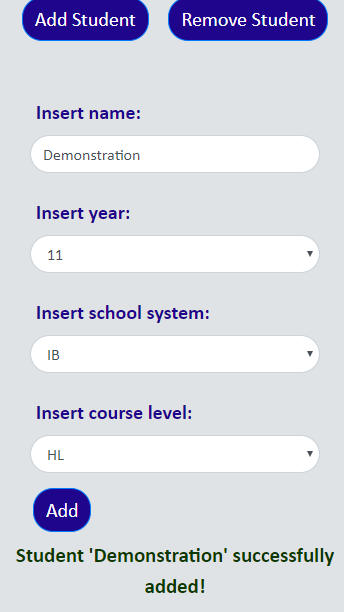
This notification disappears upon changing sections, and the inputs are resetted. The following functions are called upon clicking on anything that will change sections:



AJAX for addForm submission:

AJAX for removeForm submission:

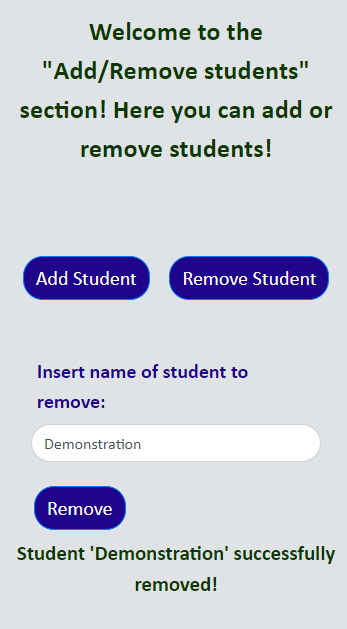


**Addition notification:**

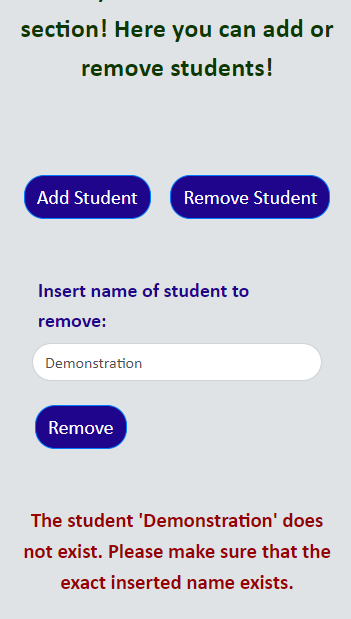
**Addition successful:**





**Removal notification:**

**Removal successful:**

**Notification when no**

**students found:**

In order to record the video for Criterion D, the software “*NVIDIA GeForce Experience*” by *NVIDIA Corporation, 2016* will be used.

**Word count:** 1185 (excluding code, screenshots, footnotes and image references, and references inside square brackets)

1. Appendix 4 contains the “References” where **all** the sources can be found [↑](#footnote-ref-1)