=================================

AC Meeting Scheduler

Lexie Gaines-Smith, Will Longabaugh, Marie Margishvili, Sam Tang

v1.0 ?

=================================

The Meeting Scheduler is a web application for professors to be able to schedule outside of class meetings with students. It was built using HTML, CSS, Javascript, Jquery, Bootstrap and Firebase.

**Tools**:

Jquery: a javascript library used in our application for form actions and dynamic CSS manipulation. We chose to use this instead of pure javascript because of its sleeker, more compact format.

Bootstrap: a HTML, CSS and Javascript framework which is generally intended for use in front end development. for this application we mainly used Bootstrap’s modal elements and CSS classes.

Firebase: A real-time backend database service that allowed us to upload JSON files with our data’s structure and then edit and keep track of edits in real time. This is how the timeslots’ availability is stored and retrieved. The service has a Javascript API, as well as Java and Objective-C.

For more information on HTML,CSS, or Javascript go [here](http://www.w3schools.com/) or [here](http://www.codecademy.com/)

[For more information on Jquery](http://api.jquery.com/)

[For more information on Bootstrap](http://getbootstrap.com/)

[For more information on Firebase](https://www.firebase.com/)

**Structure**:

*home.html*- This is an page to split users between the students and professors functionality. Clicking Students leads to index.html. Clicking Professors leads to login.html.

*index.html*-the students page. It allows students to choose a professor and then see which slots that professor has set to available. Students can then select timeslots and enter Name, Email and Reason for Visit in the modal form that is triggered by a click on an available slot. script.js contains the scripts run on this page to handle calls to the Firebase and the modal form.

*login.html*- a dummy authentication scheme set-up. The entries into the username field are checked against an array of professor usernames which are all possible professors names as stored in the Firebase. If they are valid the submit script will send the user to profsPage.html with the username being passed as a query string.

*profsPage.html*- allows logged in professors to set or unset half hour timeslots availability status. profScript.js contains the scripts that run on this page to check the firebase and change the CSS classes of timeslots.

*style.css*- contains the majority of the CSS classes for this application.

*bootstrap.css*-contains the CSS classes used to indicate availability of timeslots such as success and danger.

**profScript.js**

**Overview:**

The script is split into two main parts:

1. It fills already-scheduled appointments from Firebase.
2. It allows professors to change the status of timeslots – unavailable timeslots can be made available, and available timeslots can be cancelled and made unavailable.

These both occur on the loading of the page with jQuery’s *.ready()* event. The first thing that happens when the page is loaded is the acquisition of the correct professor’s name from the query string (which is the username submitted from the login screen).

**Table Filling:**

To fill every day of the week, *fillDay()* is called once for every day, Monday through Friday.

*function fillDay(professor, day)*:

* Parameters are the professor’s name and the day we’re filling, both as strings.
* We construct a Firebase reference to the professor’s day with a URL and the function parameters. The reference is stored in the var ‘dayRef’.
* We call *dayRef.on(“child\_added”)*. This returns all of the child data of the day we’re looking at for the professor we’re looking at as Firebase DataSnapshots.
  + The direct children to dayRef will always be every timeslot in a day, regardless of whether the timeslot is Available, Taken, or Unavailable. As such, all timeslots will be considered. It also places an event listener on each Firebase timeslot which is triggered whenever children data is added to any timeslot. Each timeslot DataSnapshot is contained in the var ‘snapshot’.
* We then have an if statement with *snapshot.hasChildren()* as our condition. The DataSnapshot snapshot will have children if a timeslot is Available or Taken.
  + We look at the children of snapshot. If there is a child with the value Available, we set the text of that timeslot in the HTML table to ‘Available’ and add the class ‘available’ to it.
  + Else if there’s a child with the value Taken, we set the text of that timeslot in the HTML table to the name of the student who registered for the timeslot and add the class ‘danger’ to it.

**Timeslot Status Changes:**

Click event handlers are attached to every timeslot with *$('.half-hour-prof').mousedown().*

*$('.half-hour-prof').mousedown()*:

* Pertinent variables are set from the id of the timeslot clicked on and the professor. We get a Firebase reference to the correct place (‘profRef’), and strings of the time (‘time’) and day (‘day’).
* If the timeslot HTML element doesn’t have the class ‘danger’ or the class ‘success’ (which are attached to timeslots which are Taken and Available), the HTML element gets the class success, its text is set to available, and a child is added to the correct Firebase timeslot. This child will be an object with available, name, email, and reason properties; the available property is set to ‘Available’, and the rest are set to ‘--‘.
  + Intuitively, this means that if a timeslot is Unavailable, it is made Available.
* If the timeslot HTML element has either class ‘danger’ or class ‘success’, the HTML element is cleared of that class, its text is removed, and the Firebase representation of the timeslot is cleared.
  + Intuitively, this means that if a timeslot is Available or Taken, it is made Unavailable.

There is also code which allows a professor to click and drag their mouse over a number of timeslots, changing all of their statuses in the same way clicking on them would.

* This is done with *$(document).mousedown()*, which tracks whether the professor is clicking anywhere on the document, and *$('.half-hour-prof').mouseover()*, which tracks which timeslot the professor’s mouse is on.
* *$(document).mousedown()* sets a boolean ‘isDown’ to true. *mousedown()* triggers when the mouse button is pressed.
* If ‘isDown’ is true, and the professor’s mouse is over a timeslot, then *$('.half-hour-prof').mouseover()* does the exact same things *$('.half-hour-prof').mousedown()* does.
* ‘isDown’ is set to false again when the mouse button is released (this is done with *$(document).mouseup()*).

**profScript Tips:**

* You can run into trouble with mistaking Firebase references for DataSnapshots and vice versa. Keep track of which one you’re using, because they have different functions, and are initialized differently.
* The script relies upon consistency between the Firebase and the HTML. Remember to update the HTML and Firebase at the same time for most changes. If you update the Firebase but not the HTML you will not be able to undo changes you make (on this loading of the page). If you update the HTML but not the Firebase your changes won’t be saved.
* Learn about the different event types for the *.on()* function. Some of them immediately return data, and some of them don’t. All of them trigger asynchronously when their specified event occurs.

**Script.js**

**Overview:**

The script is split into two main parts:

1. It allows a student to select a professor and displays the professor’s calendar by pulling the specific professors available and busy slots from Firebase.
2. It allows students to make an appointment in an available timeslot or cancel an appointment they have previously made.

These both occur on the loading of the page with jQuery’s *.ready()* event. The first thing that a student sees is an empty calendar, on the upper left side of which there’s a drop-down menu for selecting a professor.

**Professor Selection:**

*$("select").change():*

* The function reads in the what is selected from the dropdown menu with *val()* function
* If it’s not the default ‘select a professor’ button, then it clears the calendar (thus, in case you switch from one professor to another, the first professors hours are cleared away)
* After clearing the calendar, it calls *FillDay* on each workday of the week, passing in the professor and the day as parameters.

**Table Filling:**

To fill every day of the week, *function* ***fillDay****(professor, day)* is called once for every day, Monday through Friday. It works exactly the way described above in the profScript.js section.

**Timeslot Status Changes:**

*$('.half-hour').click():*

* Pertinent variables are set from the id of the timeslot clicked on and the professor. We get a Firebase reference to the correct place (‘profRef’).
* If the timeslot HTML element has the class ‘success’ (which is attached to timeslot ‘Available’), then a small box pops up where a student can enter information (name, email and reason for visit)
* If the timeslot HTML element has the class ‘danger’ (which is attached to timeslot ‘Taken’), clicking it makes the class ‘danger’ be replaced with class success (hence, intuitively, the appointment is cancelled), and the time-slot is marked as available. Furthermore, the information about the student (their name, email and reason) is set to ‘—‘ (becomes empty again).

**Updating the Firebase:**

After clicking an available time-slot, a box pops up, where a student can enter their name, email and reason. The box has three fields: “Name,” “Email” and “Reason” as well as two buttons: “Submit” and “Cancel.”

*$('#submit-form').validate():*

* This function makes sure certain rules are followed when filling in the pop-up box. Specifically, it ascertains that:
  + All the fields are filled
  + Every entry is longer than 2 letters
  + The email is in valid format

*$('#submit-button').click():*

* When a submit button is clicked, if the timeslot is available, then the name, email and reason for visit are read from the pop-up box by *val()* function
* The slot’s availability (the class success) is then replaced with unavailability (the class danger), after which the pop-up box is hidden
* Finally, firebase gets updated with the appropriate name, email and reason, while Availability slot is replaced from ‘Available’ to ‘Taken.’

*$('#cancel-button').click():*

* When the cancel function is clicked, the pop up box is hidden away and nothing gets changed or updated.

**Styling/Bootstrap**

**Overview**

The scheduler application uses Twitter Bootstrap in conjunction with custom CSS to style the documents. We used Boostrap’s built-in classes for table cell coloring based on availability, for the submission form to reserve a time slot, and for navigation via buttons

**Availability Indication**

Bootstrap has built-in classes to change the background color of an element. A table cell will contain the class ‘danger’ (turning it red) if the time slot is taken, ‘success’ (turning it green) if the time slot is available, or neither if the time slot is altogether unavailable.

**Table Cells**

Each clickable table cell contains a class ‘half-hour’ or ‘half-hour-prof’ class which defines the dimensions and text size of the cells. When a time slot gets reserved or made available, the ‘danger’ or ‘success’ classes are appended via jQuery.

**Modal Form**

Using Bootstrap’s modal form class lets students reserve time slots by clicking on available times. Once they do, the modal form appears on-screen and a reference to the table cell they clicked on is passed into the currentTime variable, which is used to create a Firebase reference to the correct time slot in the database.

Clicking submit in the modal form will append the ‘danger’ class to the table cell, retrieve the form data and write to Firebase, and set the text of the table cell to the name of the student who requested the meeting. The form is then cleared and the modal element is hidden. Clicking cancel will hide the modal.

**Buttons**

The student page has a number of buttons which use Bootstrap styling. The week navigator uses the ‘btn-group’ class to give them a continuous appearance and each button in the group is styled using ‘btn btn-default’.

Although they currently are not functional, incorporating the navigator would be straightforward. For each user in the database, there would be a list of child attributes indicating the week, with the days of the week making up a sublist of children for each week. In the script and profScript files, there would need to be a variable like ‘currentWeek’ containing the current week, which would be used to generate a reference ‘weekRef’ to the correct week in Firebase. The currentWeek variable would be updated each time the left or right-arrow buttons are pressed in the button group and the table would be repopulated using the Firebase reference.

The cancel button in the modal form has a type of ‘reset’, which clears the form when the modal is closed.

**Possible Extensions:**

-*Outside Calendar Integration*: We hoped to add in functionality to connect with a outside calendar service like Google Calendars. Google has a [Calendar API](https://developers.google.com/google-apps/calendar/) that appears to have the functionality necessary to allow for this feature to be added. If that becomes the case then the professor interface will potentially become unnecessary as all professor actions could be performed from within their own Calendar.

-*Variable Sized Timeslots*: The use of the HTML table for our timeslot grid enforced a rather strict half an hour availability structure. A possible option for extending this feature to work for timeslots that are irregularly sized is to underlay the visible table with another table having smaller timeslots and then assigning a certain number of them when clicked. Any addition to this feature would have to be an edit of profPage.html,index.html as well was profScript.js and script.js.

- *Secure Authentication*: Currently we employ a dummy authentication scheme, where all of the possible professors from the database are stored as values in an array. Every username value entered is checked for validity against that array and then if it is found passed as the “professor” value to the professor’s page. A real authentication scheme could be employed by adding database calls to the scripts in login.html to check for validity that way.

-*Multiple Weeks*: Right now we only have this functionality set up to work for a single week but future version might want to create multiple week functionality. The Student’s UI has a placeholder element for the addition of this feature. This could be done through a variety of methods. Potentially with addition of the Google Calendar functionality or a secure authentication scheme the application can tie into some built in date function to asses which week the application needs to pull. This might require that the Firebase include a month signifier in its data structure.

-*Student-Professor Synchronicity*: A future change that might be useful would be a way to ensure that student could receive real time changes was a professor makes them. This could be some done with an additional Firebase function added to script.js.