**CSC650 - Progress Report**

Graduate Student Group

October 11, 2019

American University

**This work complies with the AU honor code. We did not give or receive unauthorized help on this assignment.**

**TABLE OF CONTENTS**

**Client Overview**

**team**

**task overview**

**Progress Report**

**Implementation and design**

**Project changes**

**Issues and challenges**

**Summary and conclusion**

**Client Overview**

Our client for this project is Kristof Aldenderfer, Adjunct Professor for Physics and Director of the Design and Build Lab (DABL) at American University. He can be contacted at [kristof@american.edu](mailto:kristof@american.edu) or 202-885-6431.

# **Team**

Our team members are:

* Myles Russell mr7085a@student.american.edu
* Abdul Tarawally at4667a@student.american.edu
* Isabella Sims is3923a@student.american.edu
* Jiachen Yao jy3695a@student.american.edu
* Cliff Gallagher sg2563a@student.american.edu
* Li Jie Su [ls8513a@student.american.edu](mailto:ls8513a@student.american.edu)

# **Task Overview**

We are developing a queuing and scheduling system in the Design and Build Lab (DABL) which will permit students to register for workshops and activities based on the specific tools (3D printers, laser-cutter, etc.) that they have completed training for. Ultimately, this will limit students to reserve tools that they have been certified to use. Students will be able to view the DABL website to learn availability of the tools for scheduling an appointment. This allows students to reserve tools online in comparison to the current system of first-come-first-serve. All reservation data will be used to evaluate machine use to determine specified analytics of tool use.

# 

# **Progress Report**

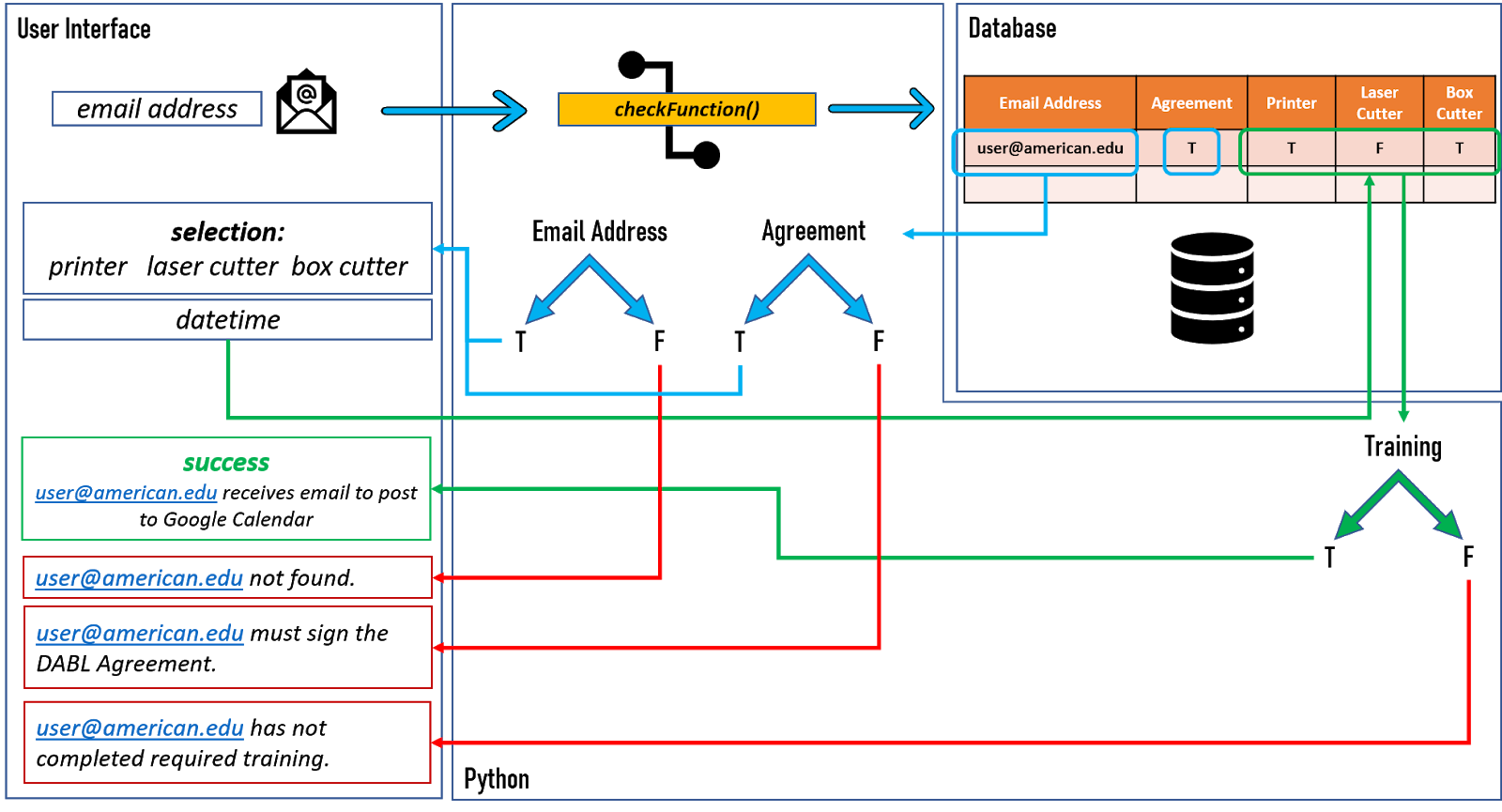
***Summary***

During the initial phase of the project, all functional requirements were finalized and non-functional requirements were generally established. The project then moved to the next phase focused on system requirements. A workflow was established and time was allocated to the development stage to fulfill a *proof-of-concept* implementation that was completed during that time period.

***Development Progress***

From a development standpoint, the team created a scheduling application that demonstrated the ability to reserve a time on the DABL website. Additionally, a GUI implementation was provided to illustrate a basic user interaction on the front-end of the application. The *proof-of-concept* application (referred to hereafter simply as *application*) achieved the general concept of allowing users to reserve a specific tool at a specific date and time. Although we assess that future requirements will evolve the database, the team provided a workable foundation for actual application. Additionally, the logic for scripts have been implemented to query database regarding whether student has a reservation at the time they login into DABL, has completed the requisite training to use a specific tool. This is key since we can transfer these into the required *php* scripts needed to integrate with *Wordpress*.

Some additional - and vital - points in the exercise of creating the application allowed the developers to hash out many overarching concepts of how each of the separate components will interact with one another. One critical part that was researched and implemented effectively was the Google calendar and email confirmation API that tells users once their reservation request was successful. This will translate easily into the *Wordpress* code and served to provide as a foundation to build from.



***Flow chart of what has been built.***

**Implementation and design**

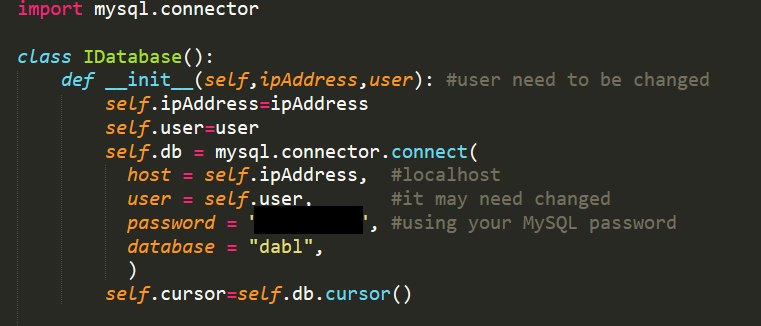
***System and Program Design***

Primarily, the team has used *Python*, *MySQL*, and *WordPress* to build the application. MySQL is the database chosen to support the application and it will be integrated into DABL. Python interacts with the database, processes all data from users to query and insert into the database. Users may access the DABL website to reserve equipment and review the calendar through *WordPress*.

***Python interface with MySQL database***

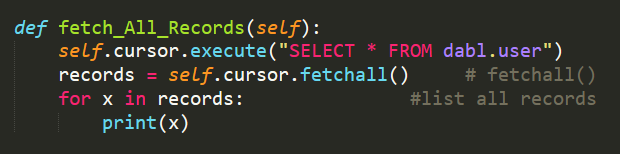
The team imported and applied the *mysql.connector* module to interact with the MySQL database. The team built several functions in the *IDatabase* class to support interaction with the database.

**\_\_init\_\_(self,ipAddress,user)**



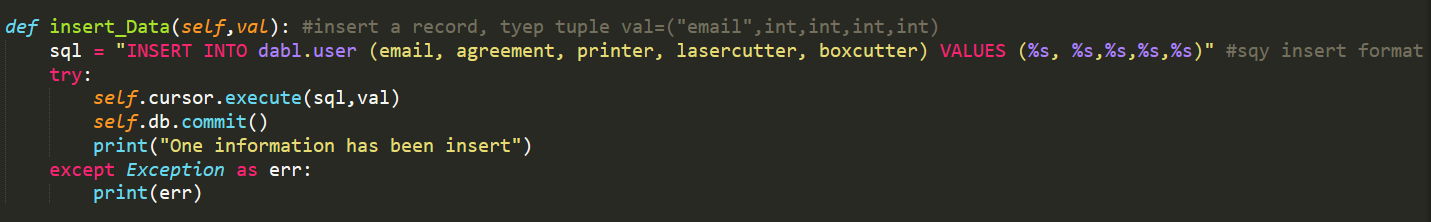
First, the user connects to the database through a function by using the established IP address and username. The *password* parameter should be the password of the database and the *database* parameter should be the name of the database. When the function executes the command of the database, the cursor is created.

**fetch\_All\_Records(self)**



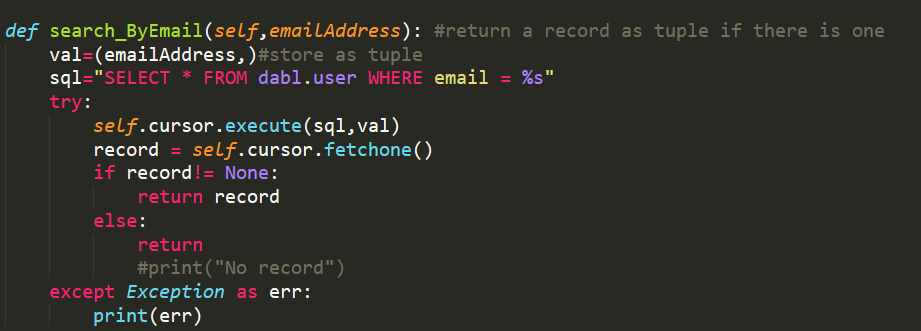
Using the cursor to execute MySQL command and getting all records in the database to save as records.

**insert\_Data(self,val)**



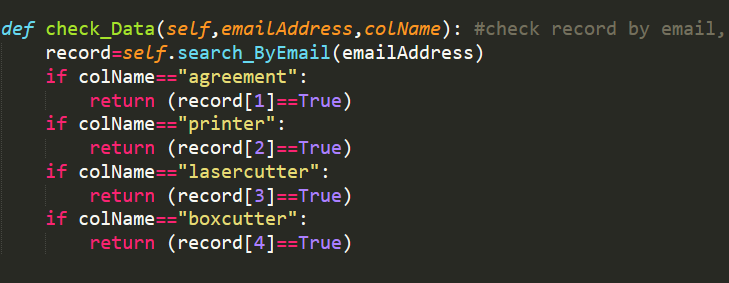
Obtaining the *data* parameter by *user* or *admin* and insert it into the database.

**search\_ByEmail(self,emailAddress)**



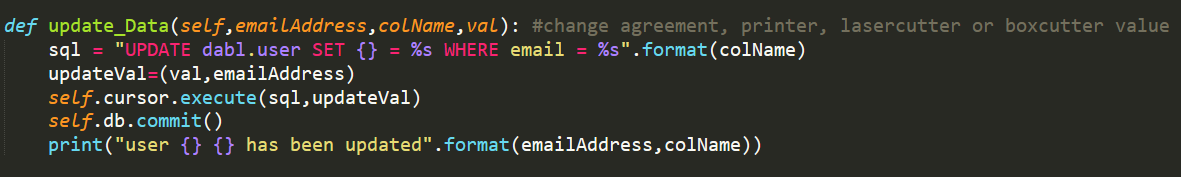
Checking the email address parameter is in the database or not, if the email address is in the database, then return the record.

**check\_Data(self,emailAddress,colName)**



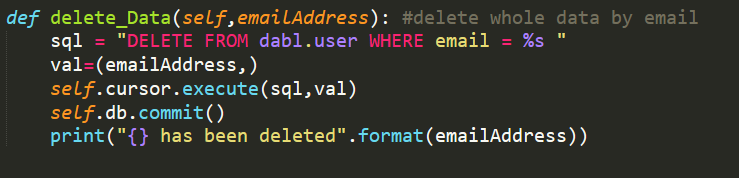
Checking the data by email address agreement, *3D printer,* *laser cutter,* or *boxcutter* and return the data is *true* or *false*.

**update\_Data(self,emailAddress,colName,val)**



Let *admin* or *user* update their data in the database. By checking their respective email, and updating their respective agreement, they can access the DABL tools.

**delete\_Data(self,emailAddress)**



Finally, the *admin* deletes the *user* data through the mail.

***Backend function for User***

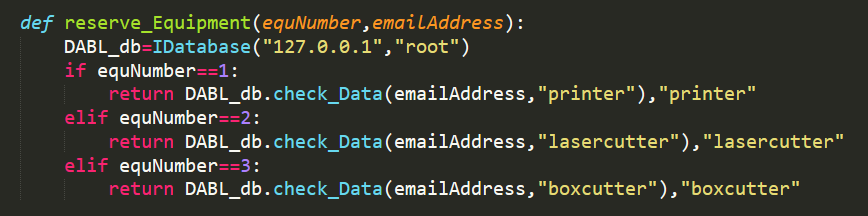
Based on all these functions we build in *IDatabase* class, we can let users or admin interact with the database. Preliminary, we built the backend function for users to reserve equipment in DABL through the frontend website, but we just test all these functions on our local console. After users reserve one of the equipment and choose the time section, we will send a confirmation email and arrange a schedule on DABL. We will embed all these functions into the Wordpress website.

**is\_login\_Success(emailAddress)**



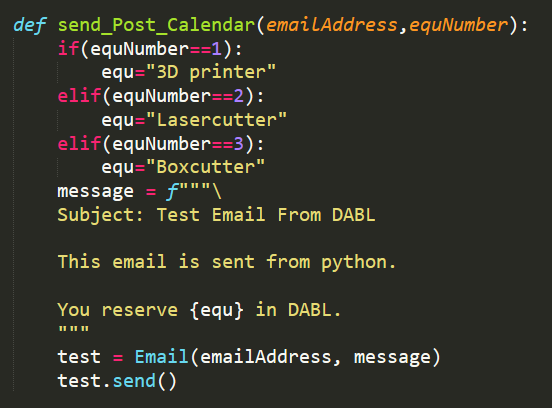
First, we will let users type in their email address to check they are in the database or not. Create a DABL\_db object to connect the database. The IP address should be the address where the database locate. The database will move to the remote database in the future.

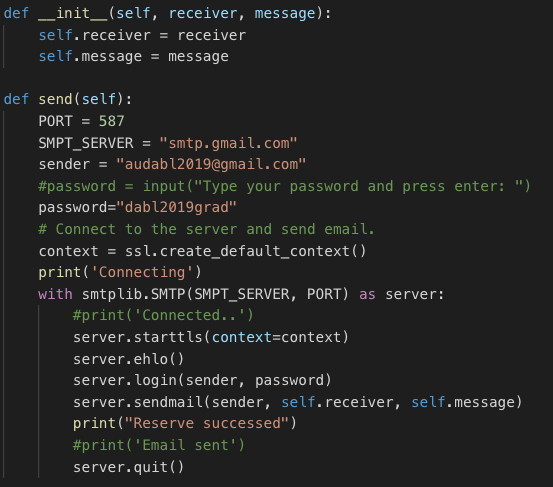
**reserve\_Equipment(equNumber,emailAddress)**



After checking the email in the database is or not. This reserve function will connect to the database and check the user finished the training or not. And return it is true or false.

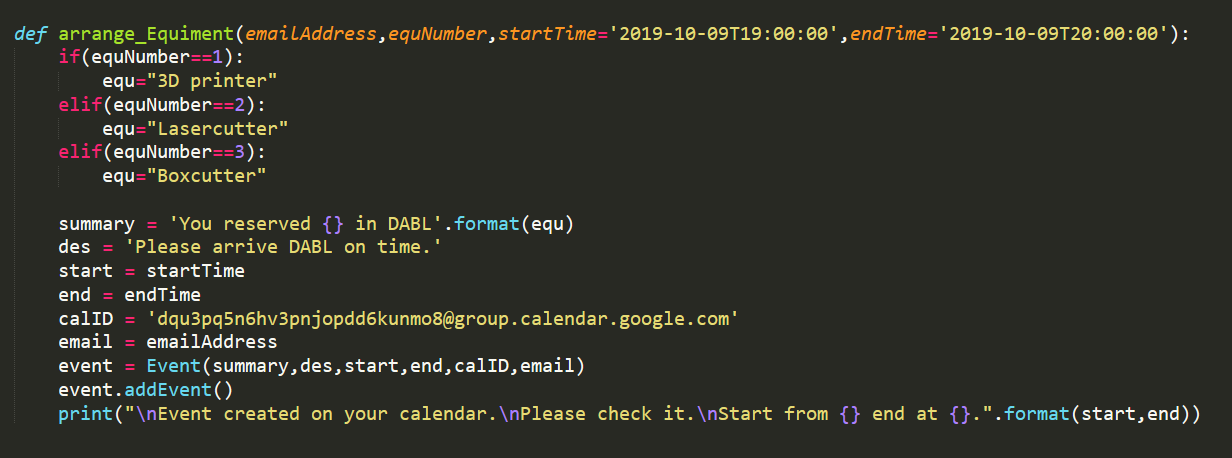
**send\_Post\_Calendar(emailAddress,equNumber)**



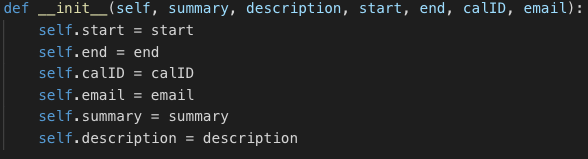
If the user signed the agreement and finished the training, this function sends a confirmation email to the user’s email address through the GOOGLE API.

Email object takes two parameters, receiver email address and message. It will establish a connection to SMTP server and sent an email from our test email address to receiver email address with the message.

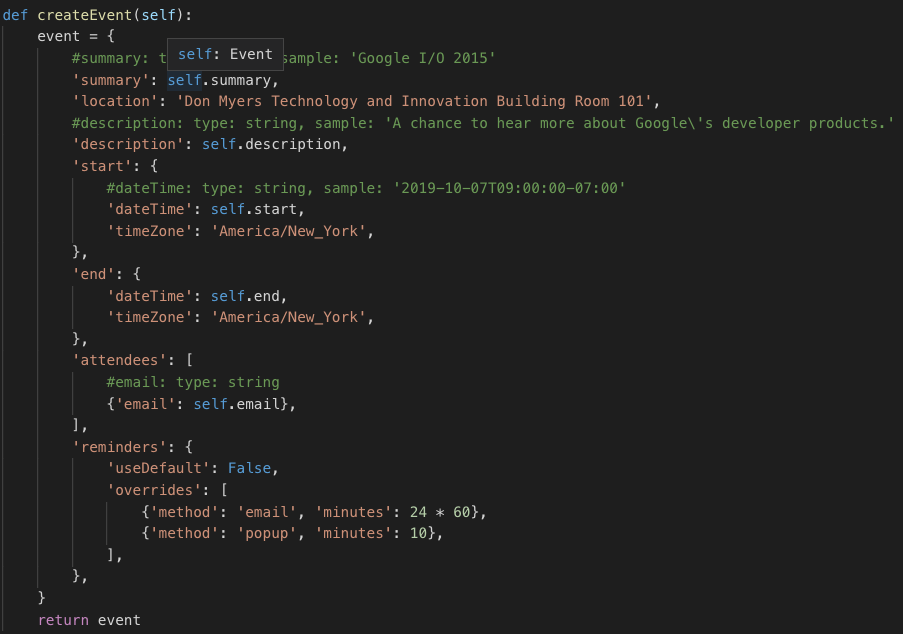
**arrange\_Equiment(emailAddress,equNumber,startTime,endTime)**

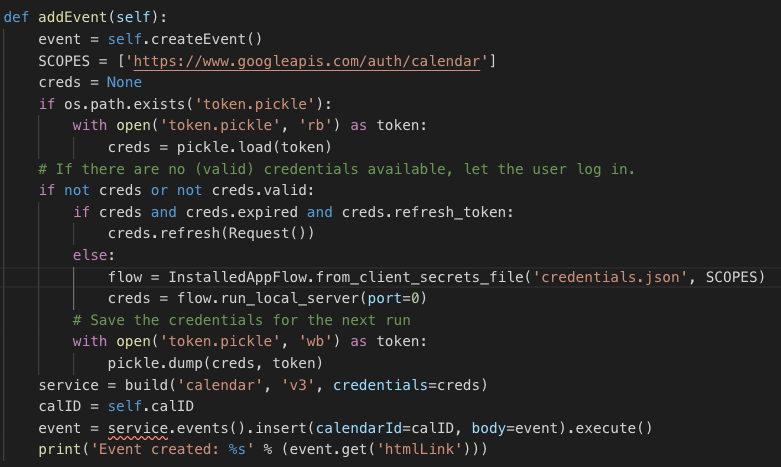


After we sent a confirmation email to the user, this function arranges the schedule to the user’s calendar and the calendar of DABL equipment. The functionality of creating event and posting to google calendar was implemented in Event object.



The object takes some required information for google calendar event object.

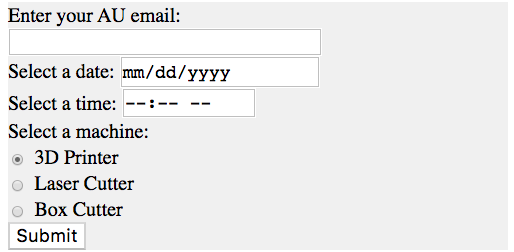




After the event object created, we post the event to google calendar with addEvent function. Basically, the function first authorizes user’s google account and then add the event to one of the calendar under the account based on the calendar ID.

**Front End Design**

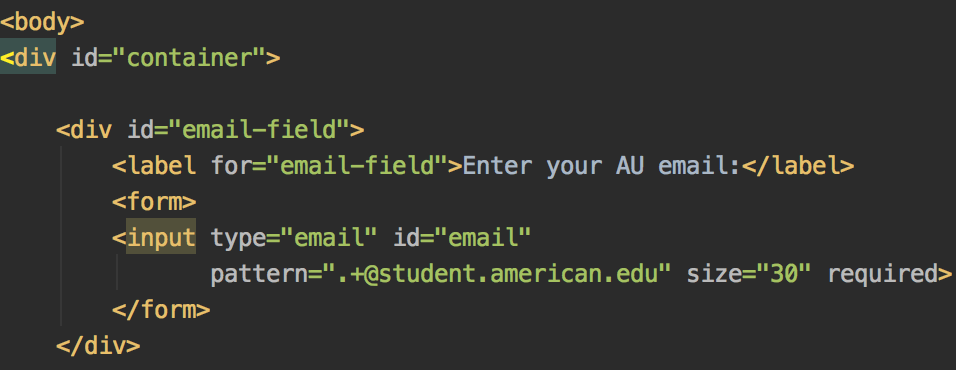
On the front end, students will be able to schedule a time for a *3D printer*, a *laser cutter*, or a *box cutter* from the current DABL website. We are designing a *widget* to accept user input for *email addresses*, *dates*, *times*, and *machines of interest*. A prototype of the widget can be seen as follows:



The current widget is simple but will require more work as we connect our back end to our front end and start designing PHP scripts to do so. This is to be addressed in our next sprint. The widget can be easily integrated once we acquire the AWS clone wordpress allows custom HTML blocks and embeds. The code for our current widget version is as follows:

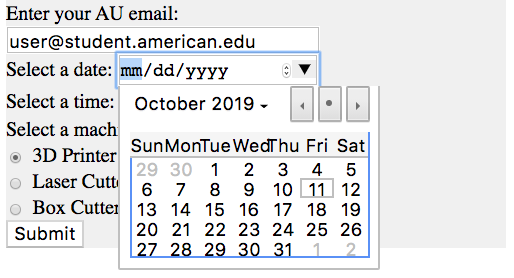


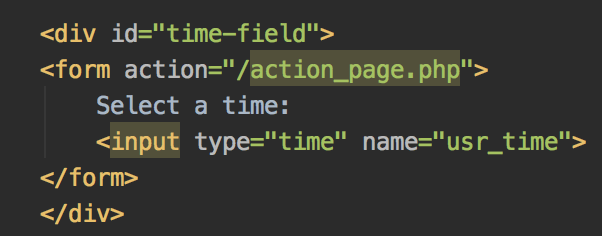
The email field is designed to only accept student email accounts:



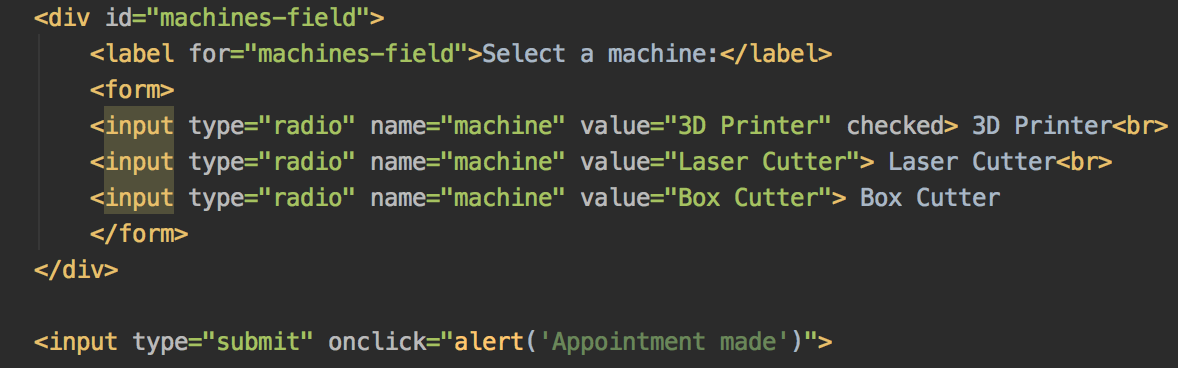
The date field produces a clickable calendar for easy scheduling:



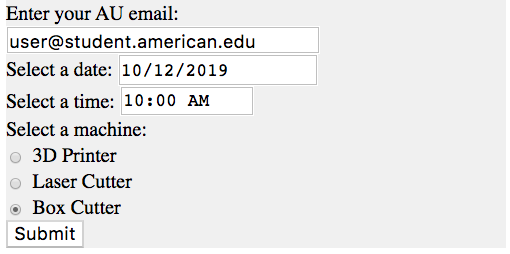




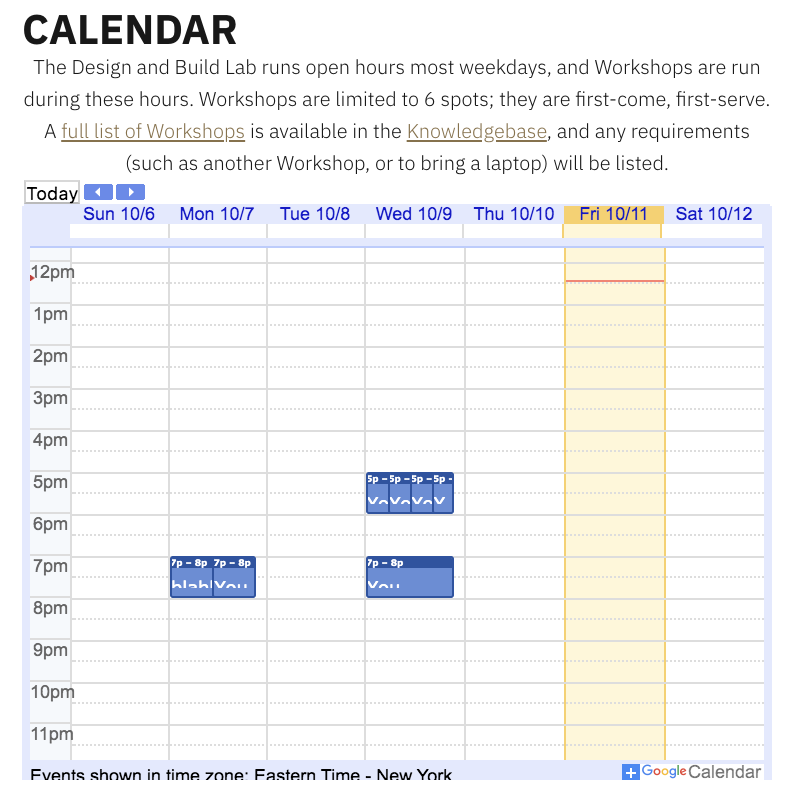
There are three radio buttons to represent the three different machine options. In our next sprint, it will be designed to display only the calendar of the machine selected:



An example of all fields completed before submission:



For testing purposes, we have copied the page source of the current DABL website into a mock *Wordpress* site. Working with *Wordpress* has been less restrictive than we anticipated. Wordpress allows for *Google Calendar* embeds, which has proved to be simple and effective. For our demo, we swapped the real DABL calendar with a mock one we created and were able to verify the functionality of our backend by creating events on the linked calendar:



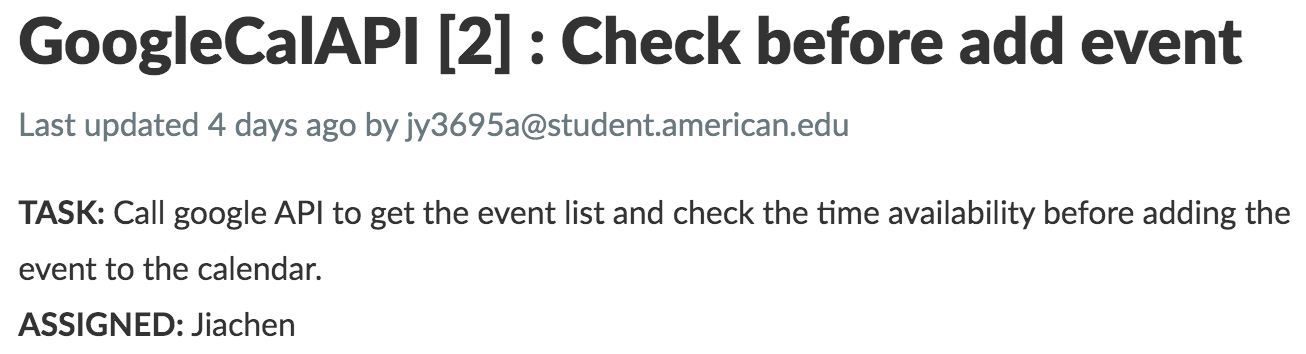
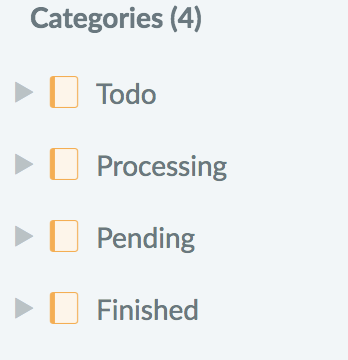
**Project Changes**

Since our last report, we have made quite a few changes. At the more refined level, the most prominent change was the decision to abandon DABL IDs and use emails instead. Because we found that we are allowed to store AU email addressesses, this was clearly the better option. From a UI perspective, the email address method of identification will be much more pleasant as students will not have to remember a new form of ID to schedule a machine. This will also be easier for us: we no longer need to email every student in the database a unique ID or do any sort of hashing. Additionally, using emails makes the software more maintainable for our client as they will not need to deal with assigning new students DABL IDs or troubleshoot errors like students losing the email with their assignments.

We have also started Tetra as a ticketing system since our last report. We are using this delegate and keep track of tasks assigned to team members. The four categories represent the following:

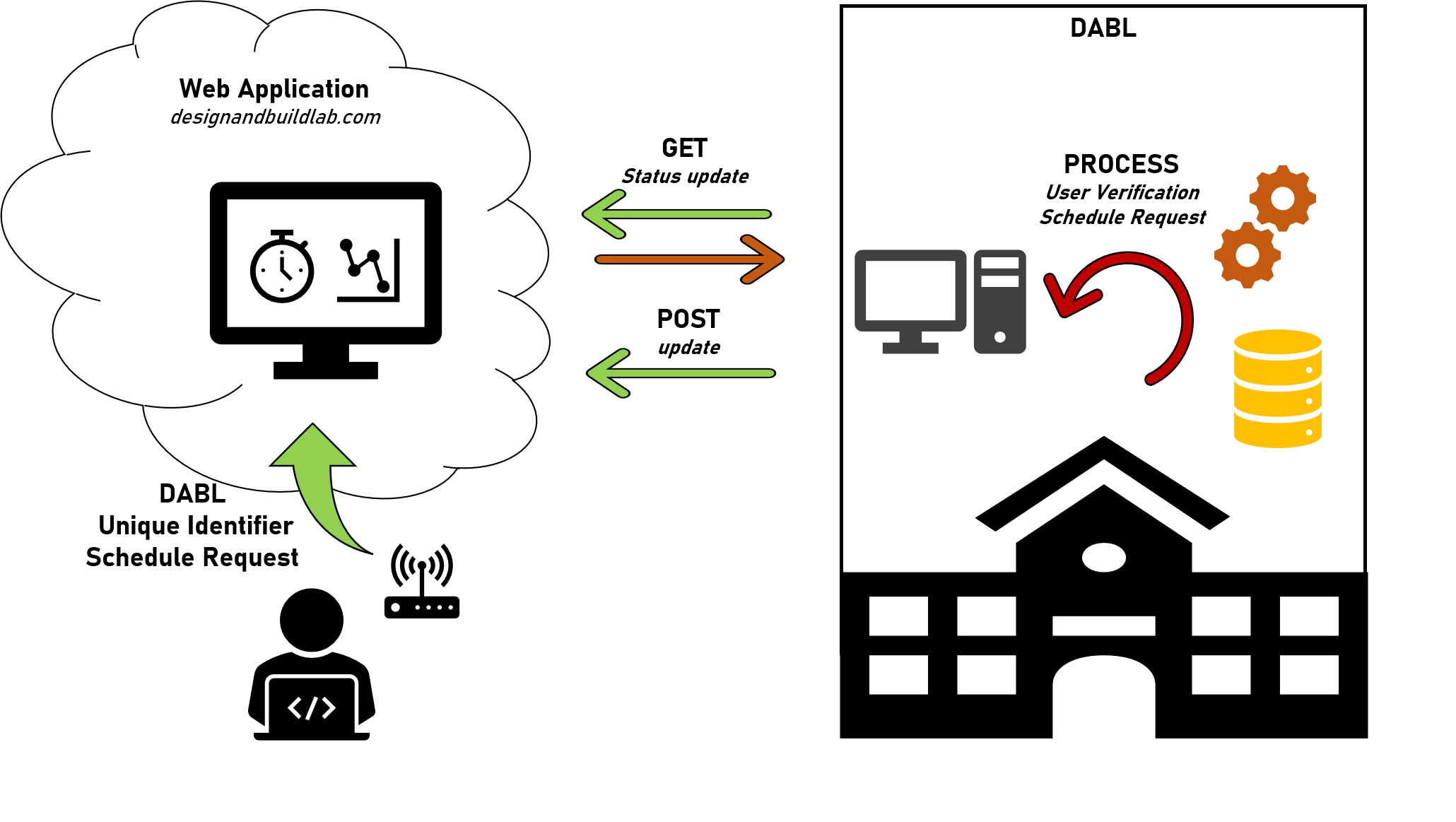
1. *Todo*
   1. Tasks that have been created but not assigned or started
2. *Processing*
   1. Tasks that have been assigned, started, but not completed
3. *Pending*
   1. Tasks that have been assigned and completed but need approval from the team to be considered completed
4. *Finished*
   1. Tasks that have been assigned, completed, and checked by every team member. Tasks are moved here once the group has reached a consensus that the code or idea is ready to be merged into our current working version.

An example of the format of each ticket and image of our categories are pictured below:



Tettra has been a useful tool for version control and has added structure to our daily standup meetings.

At a higher level, project changes have occurred more in the sense of refining requirements than in seeing major shifts in what the group pictured. More specifically, the project requirements that did evolve, did so in such a way as to make the project more manageable and sustainable in the long run. Moreover, project requirements were mutually designed through several conversations with the customer.

The biggest change from our initial requirements has been the establishment of American University privacy regulation and practices to determine what we can and cannot use with regards to user information. Once these were identified, the requirements regarding component interfacing with the DABL website changed to allow for storage of required user information in the *Amazon Web Services (AWS)* instance that runs the DABL website. This avoided some of the later issues the team would have encountered when trying to invoke communication both securely and succinctly between the initial requirements that required DABL to have a standalone computer POSTing and querying the website. [see original framework below] 

The next change worth noting is the *administrative* page requirement for DABL employees to sign users up. This is an additional page that the team had not considered until it became evident that the user information required to perform the requirements could be stored in the *AWS instance*.

The final significant change to the project is the *emphasis* on the DABL tool check requirement. Originally, the requirements desired an interactive side akin to monitoring use of DABL tools and presenting the status on the DABL website. This is still desired but has been lessened to a *nice-to-have* due to infrastructure complexity in communicating with the devices. The team and customer concluded that if the requirement could be figured out and time allowed, that it should be implemented, however it was not necessary in the time frame allotted for the project.

In summary, through consistent communication with the customer regarding the problem set, the team was able to come to a workable and sustainable solution that will result in the extension of their web application through the implementation of database storage and administrative user authentication.

**Issues and Challenges**

Our biggest issues from a technical standpoint have been remedied through working closely with the customer. Most of these have been described above. From a project standpoint, our greatest difficulty was the lack of access to the website source code. This forced the team into a position to speculate our future requirements without a full understanding. Fortunately, the team was able to overcome this by creating a demo to explore what component interaction and logic of the requirements would look like. This helped, as previously discussed in the report a great deal and provided the team a step in the right direction going forward. Notably, access to a cloned version of the DABL website has been provided to the team, which will allow them to begin working with the actual source code.

One significant change we may implement in the future that has been new for the team, is the use of *Tettra*. Although this tool is sufficient and provides a solid planning platform, the team is finding it is not as compatible to software development as other tools that are more intuitive and visually appealing.

**Summary and Conclusion**

The team has formulated and solidified customer requirements, as well built a *proof-of-concept* application to demonstrate its functionality, despite lack of access to the actual source code. Furthermore, the team has established project management tools and practices to ensure project and developer success.

Near-future requirements will see the evaluation of the recently-acquired source code, as well as the re-planning required in order to implement previously determined program logic and workflow. It is inevitable that the team will encounter subtle differences in working with a different programming language - *php* - however this will not likely inhibit the team going forward or shift requirements in any substantial manner. The team is on the right track to complete the project on time, but provide real and sustainable requirements to the DABL.