**CSC650 - Progress Report II**

Graduate Student Group

November 1, 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**

**Plan For Next Sprint**

**Summary and Conclusion**

**Appendix**

**PHP scripts**

**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
* Li Jie Su 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**

This past sprint has yielded several productive outcomes. First, progress was made in terms of obtaining required accesses to *Design and Build Lab* (DABL) infrastructure as well as gaining a deeper understanding of the network layout between the components that lay within the American University (AU) network (147.9.\*.\*) and those on the Amazon Web Services (AWS). Throughout several meetings with DABL stakeholders, the group established and tested accesses to a newly-built AWS clone of the DABL website, and identified new accesses that will be required. Namely, we identified certain network security implementations that will prohibit directional connectivity based on incoming or outgoing connections. This impacts any ability for our project to move to a database located in the cloud as we cannot overcome AU privacy policy considerations. Therefore, the administrative application concept was discussed and developed into a standalone application contained to the DABL local desktop. This serves to secure the application by nature of being hosted on the AU network - which limits inbound connections.

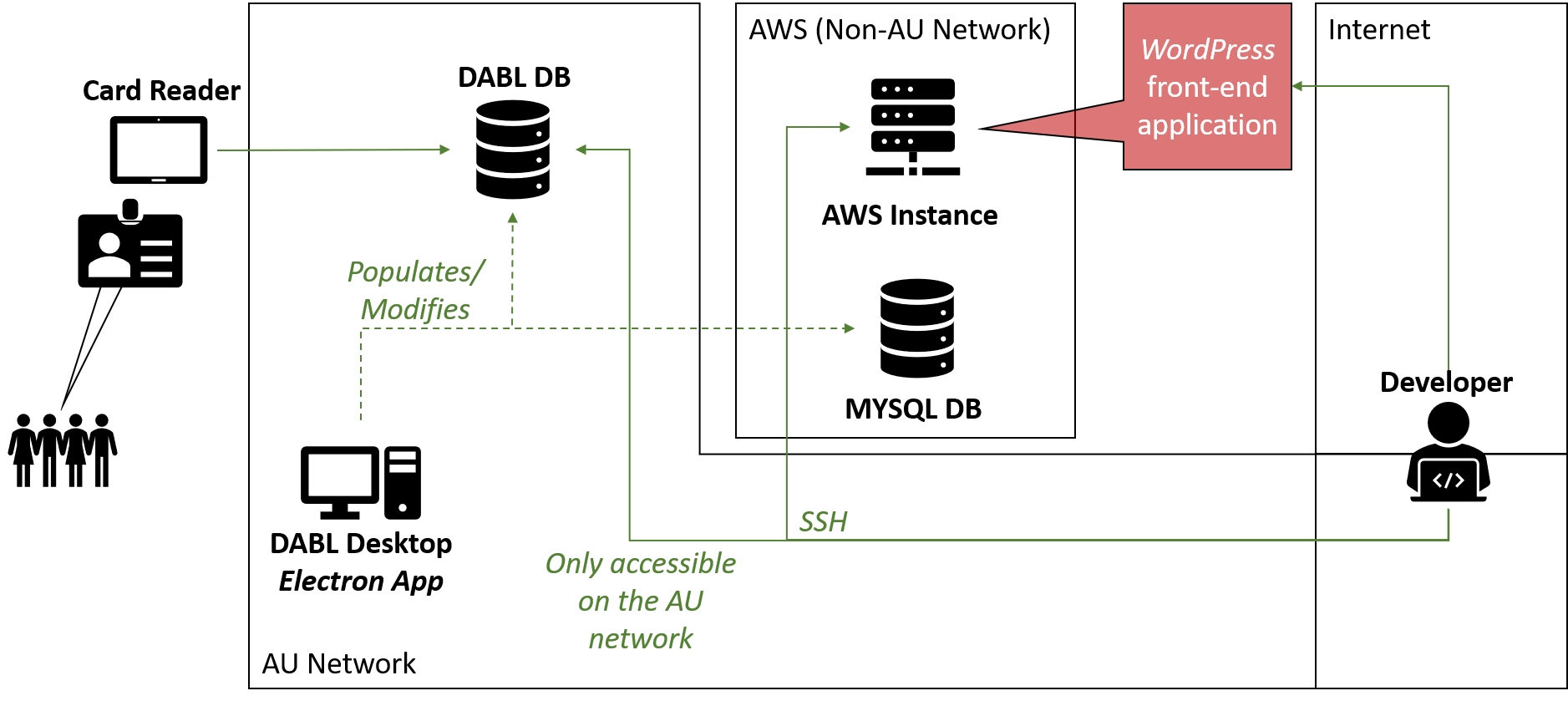
Next we converted the existing *python* application to *php* in order to be integrated into the existing *WordPress* framework. *WordPress* does not have the ability to incorporate *python* and we concluded that it would be more intuitive for follow-on developers to work with. Lastly, we incorporated the front-end widget into the newly acquired AWS clone. Because this is based on *HTML* and *javascript*, the front-end portion was easily integrated using the *WordPress* web application.

The following points summarize this past sprint’s areas of effort:

1. **DABL Infrastructure Access**
   * Significant syncing and access discussions/growth with stakeholder several key accesses were attained
2. **Electron Administrative App**
   * A standalone node.js-based application that will run on the DABL computer and populate local and AWS databases with member information.
3. **Python to php conversion of initial proof-of concept**
   * The critical functionality and logic governing stakeholder requirements
4. **Front-end widget for scheduling on the DABL website**
   * Serves as the interactive component for users visiting and scheduling via the DABL website.

# **Implementation And Design**

The image below provides an overview of the different interactions and components of the network. On the left, DABL users use their AU identification (AUID) to tag into DABL. If they are new users, they will require a DABL administrator to create a new user that populates the desktop database, used to store AU policy-appropriate information, as well as the AWS DABL clone database used to verify users for scheduling tools via the website. Additionally, the graphic depicts the developer having access to AWS from the open internet via the *WordPress* web application. This is because the AWS clone - though not located physically at AU - only accepts inbound connections from the AU network. Otherwise, the developer must be on the AU network in order to *Secure Shell* (*SSH*) into the respective components.



**Python to PHP conversion**

The *php* frame uses c*omposer* as its own package management system, so the server will require installation the package. The group has already converted three main functions needed for the *DABL* scheduling system and will require implementation onthe *WordPress* front-end since *php* scripts work very differently from *python* scripts, there are still some issues we have to deal with.

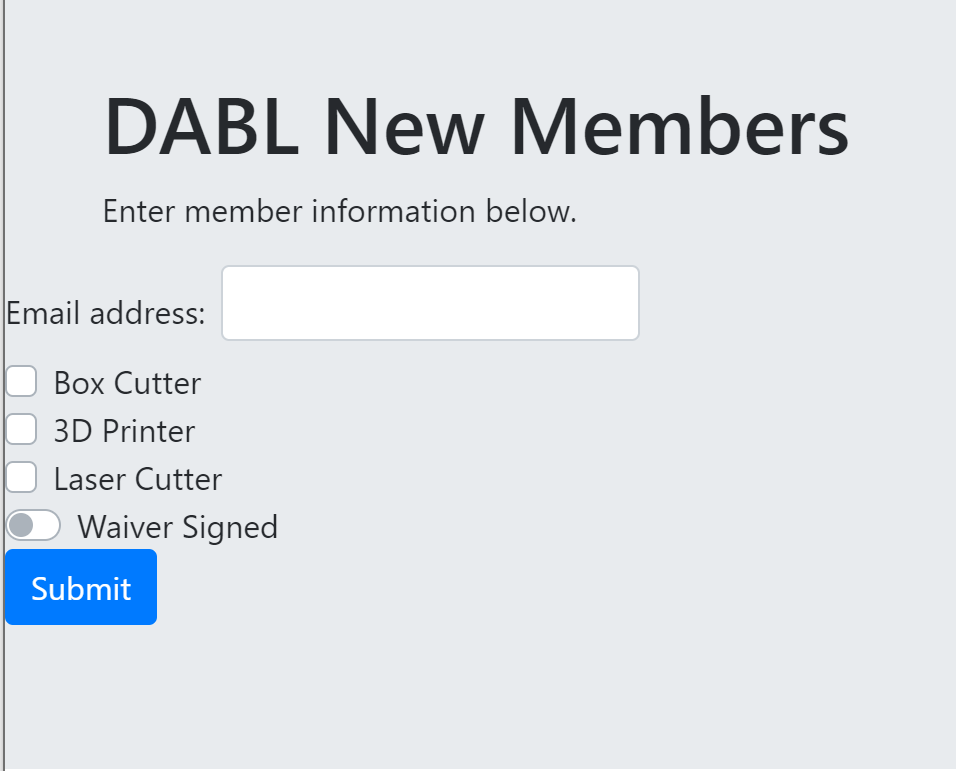
Because we received connectivity to the AWS *MySQL* database late in the sprint, the group was unable to test the final step of connecting the *php* application to AWS or local desktop databases.

The steps completed during this sprint:

* **Interface with database**: Creates a new user and changes the authentication type to *standard* for *admin* since the *php* *MySQL* has not supported the SHA2 password hash. Thus, if the administrative user needs some function, it will need to create a new user on the database.
* **Sends email**: Requires the *php* mailer package. This is simply cloned from GitHub. One issue encountered was that *localhost* may block this function if running locally. This *php* script still needs to be tested as an integrated component with *WordPress*.
* **Schedule via Google calendar**: The *composer* package from Google provides into the functionality required to populate the respective Google calendar. It works differently from the *python* script, in that it needs something an authentication code which we are still working to acquire.

**Electron Frame Application**

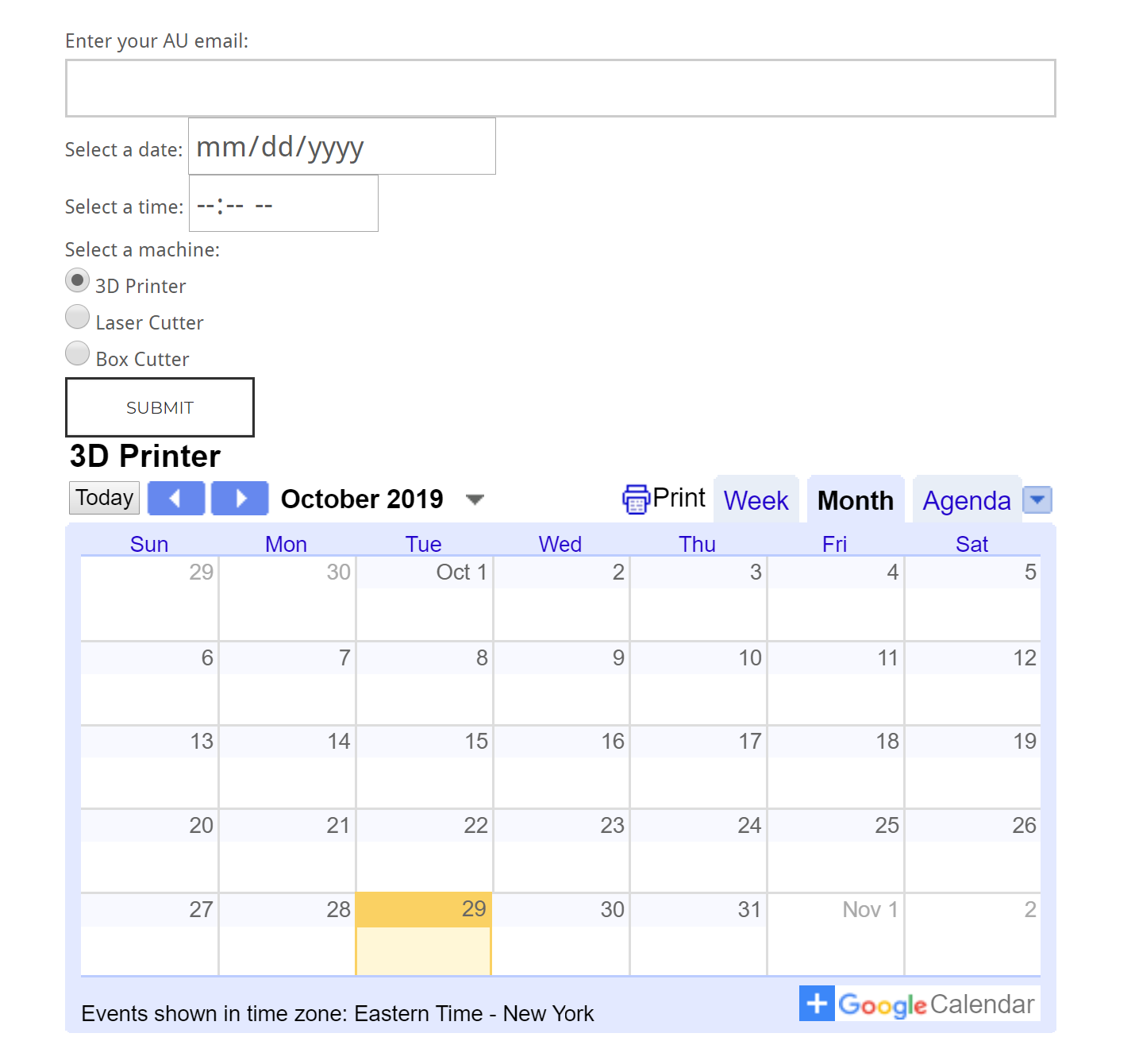
The decision to create and implement a standalone application was determined when the group ruled out creating a single database. This was primarily due to the requirement for AUIDs to be stored on an AU-network database for the card reader to reference. Our intention is to abstract the appearance of two databases to the administrative user. In other words, to the administrator, it should appear as though there is one. This is achieved by creating a front-end where the user can perform two actions: new user creation, and existing user modification. The application will handle interactions performed against both databases. 

The application is guarded securely through three layers. The outermost layer is the physical location of the desktop - the DABL room located on the AU campus. Next, the local desktop resides on the AU network, only accessible from an internal *internet protocol (IP)* address. Lastly, the application is an executable placed on the desktop. This provides outbound security as well to the databases since we can only allow connections to the respective databases from the static IP address allocated to that desktop. Because these security measures are in place, the group has decided the need for administrative user and password login is not required of the application.

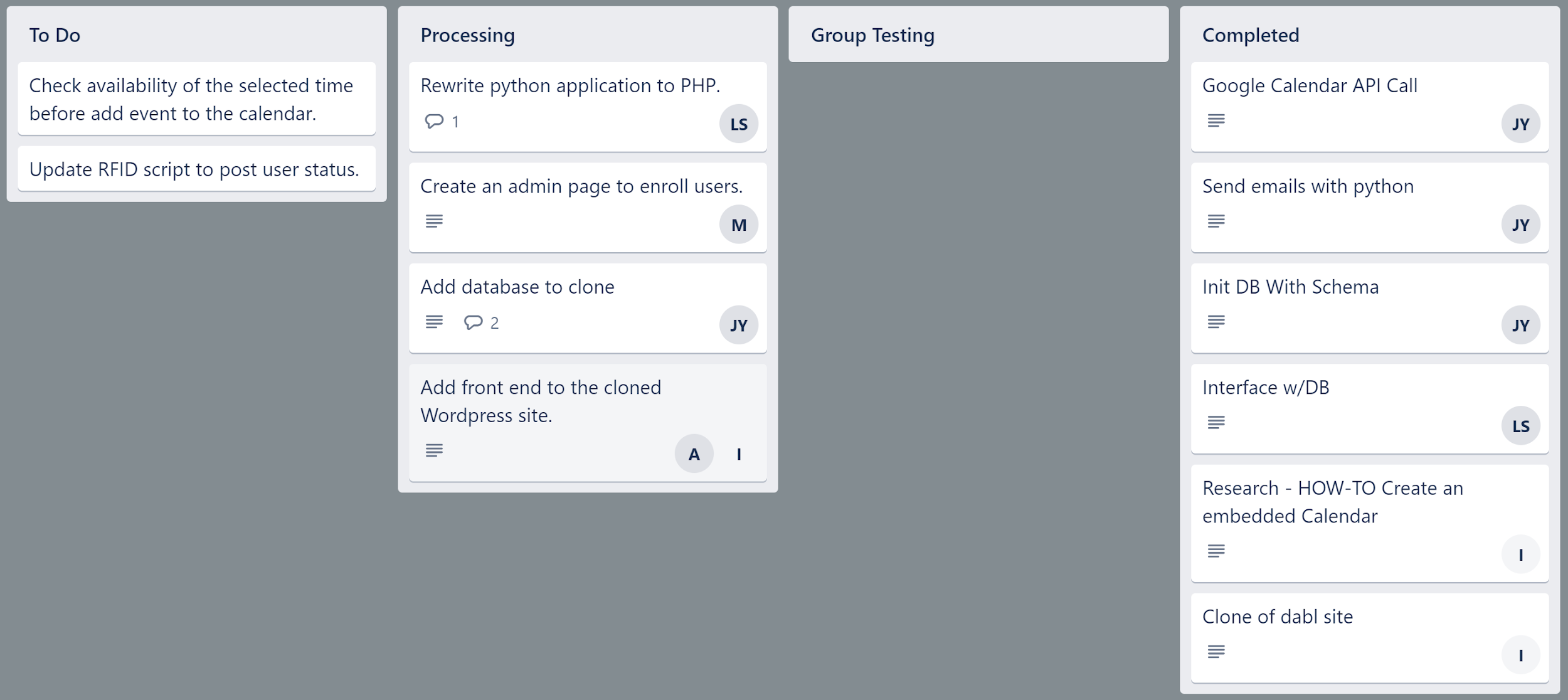
Thus far, the application only consists of a *graphical user interface* *(GUI)* that allows user creation. Similarly to the *php* application, the group must still integrate with both databases.

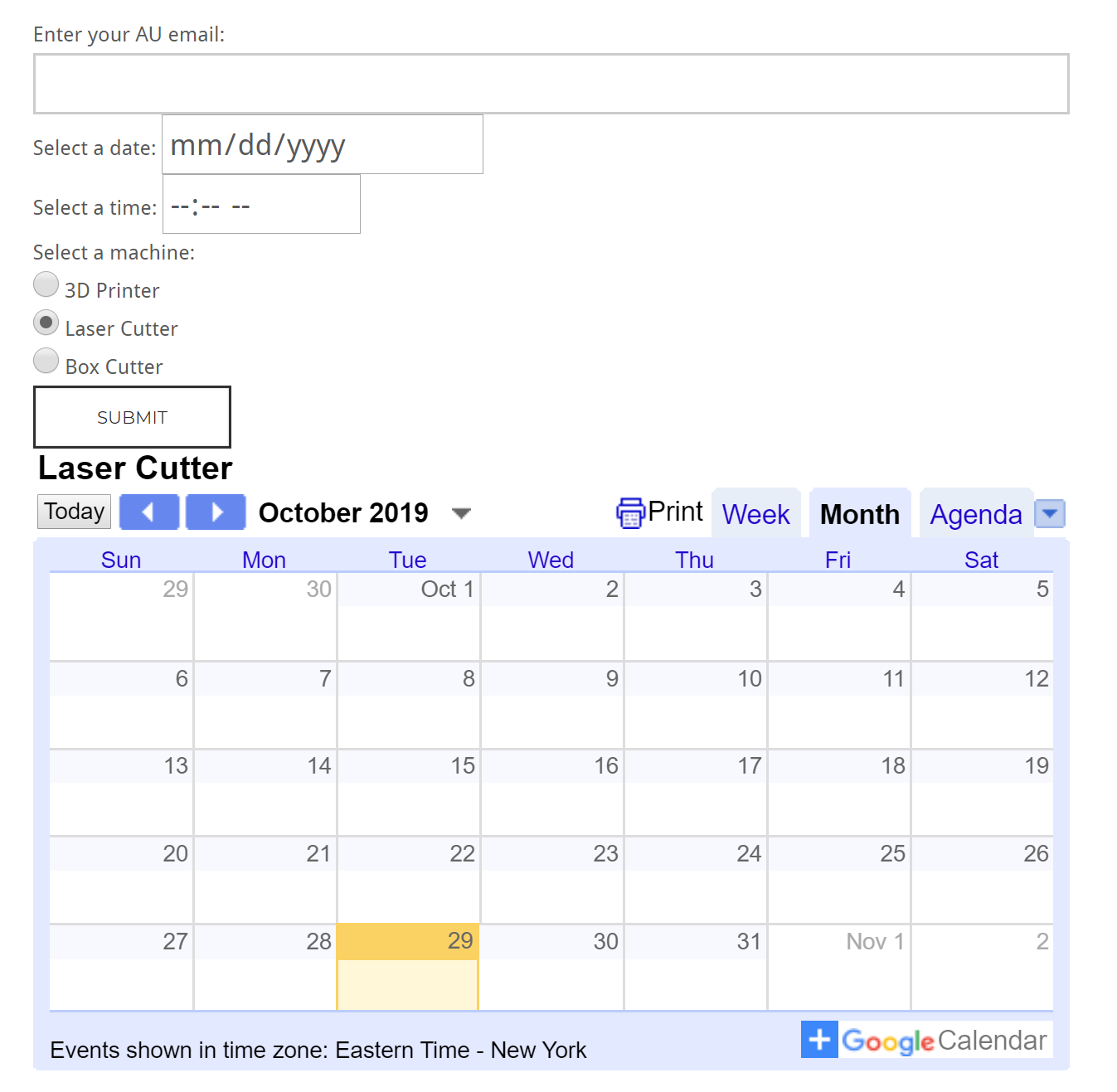
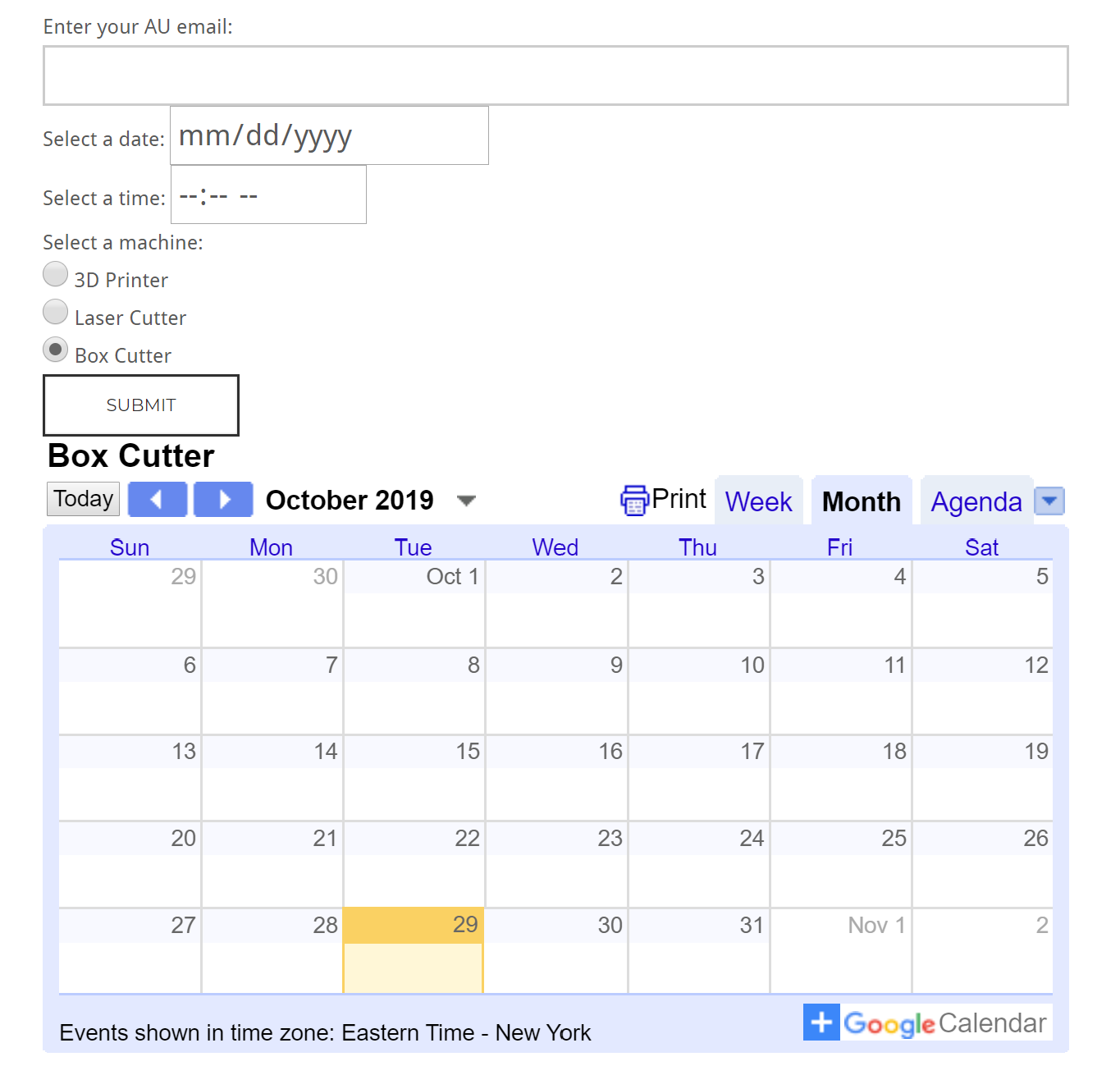
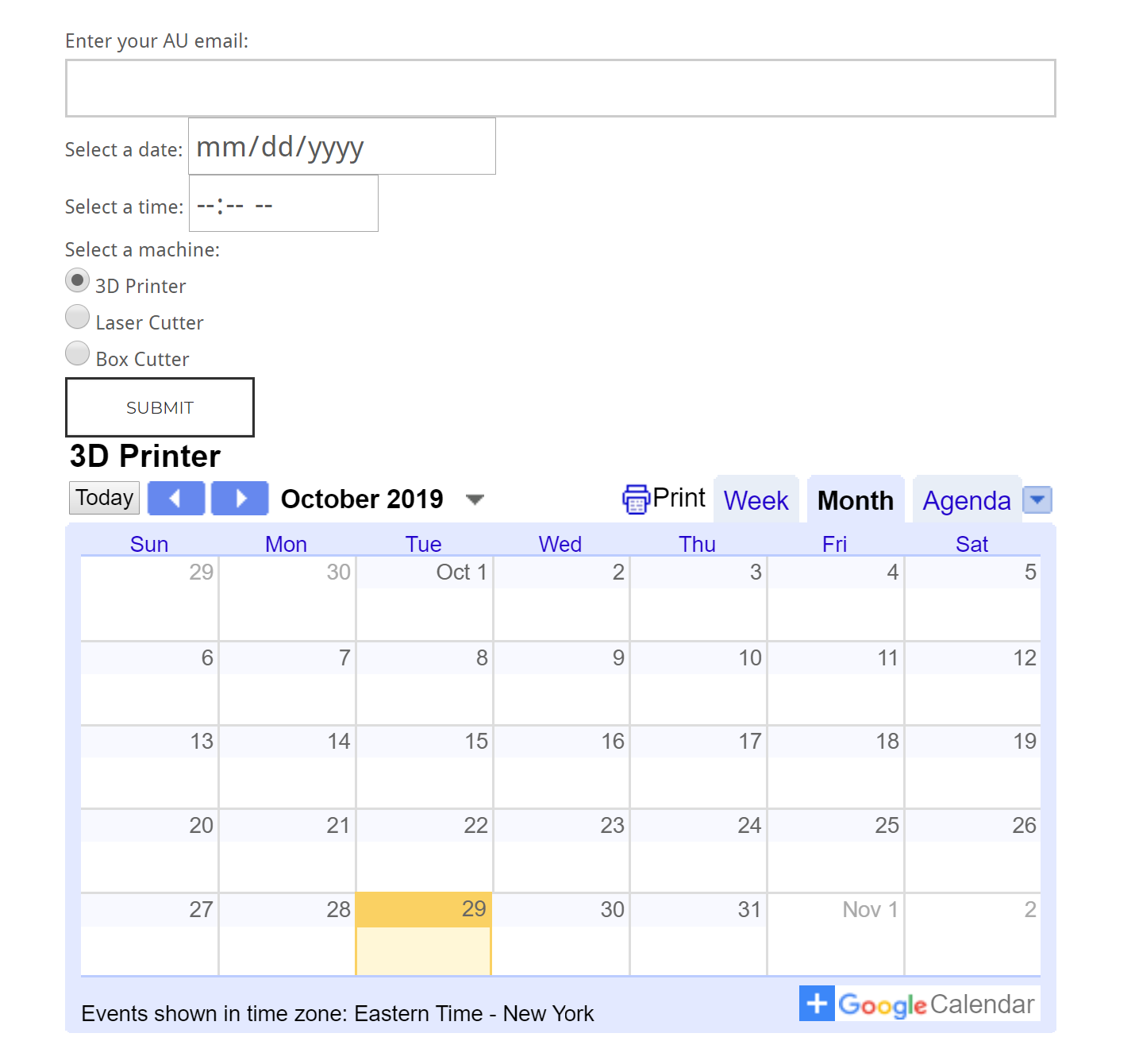
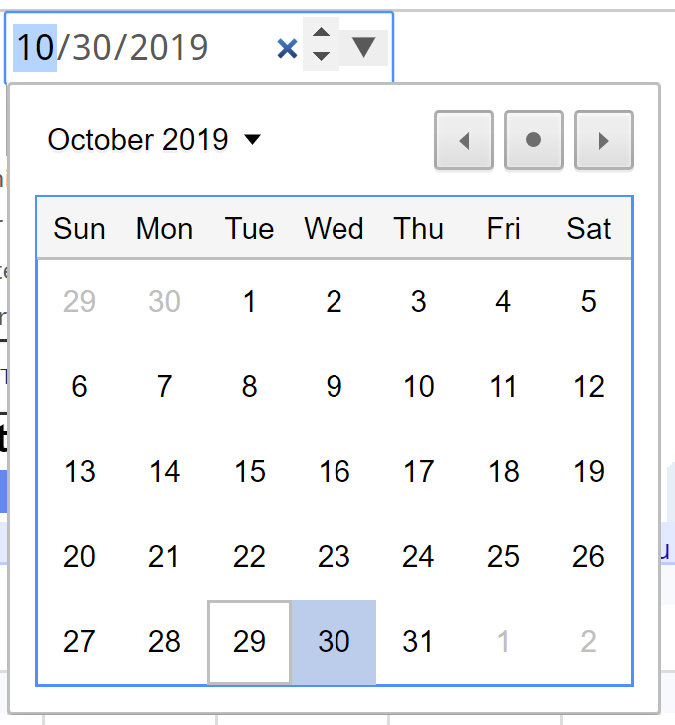
**Front-End Widget Implementation**

Our widget has fields for email, date, time, and machine. Each machine radio button toggles the display to its corresponding calendar. The email field has built-in pattern recognition to prevent submission without the @student.american.edu address.



# **Project Changes**

Since our last update, we have changed our python script to PHP in order to interface with the wordpress website. Additionally, we have moved from tettra to trello and are continuing our usual ticketing system there. We have moved from a proof of concept to actual implementation with our front end which can be seen below:



The largest difference is the toggling between three different calendars. In the previous version, we had one calendar, we now have one calendar per machine.

# **Issues And Challenges**

There were three primary issues that we encountered during this last sprint. Ultimately, they resulted in having to push several tickets to the next sprint so they were somewhat significant. Please see the following explanations for more detail.

**AWS**

The first issue we encountered occurred with AWS. The DABL website is a *WordPress* website hosted on AWS. Consequently, we had to gain access to AWS in order to add content to the web application. Meetings with our client paved the way to obtain an RSA key file to access the server. Nevertheless, Because the group did not have experience using AWS, it took some time to figure out how to SSH into the AWS instance using the key file. Currently, for security reasons, the server only allows

SSH inbound connections from an AU IP address, which means that we must connect to the AU network in order to build anything on AWS not accessible via the *WordPress* interface.

**MySQL**

The second challenge we encountered was identifying connectivity to the MySQL database allocated to *WordPress*. Our initial approach was to access the database with a *WordPress* plugin. This would have been our desired approach since allowing the *WordPress* platform carry out tasks will maintain uniformity as well as be easier on follow-on developers for continuity.

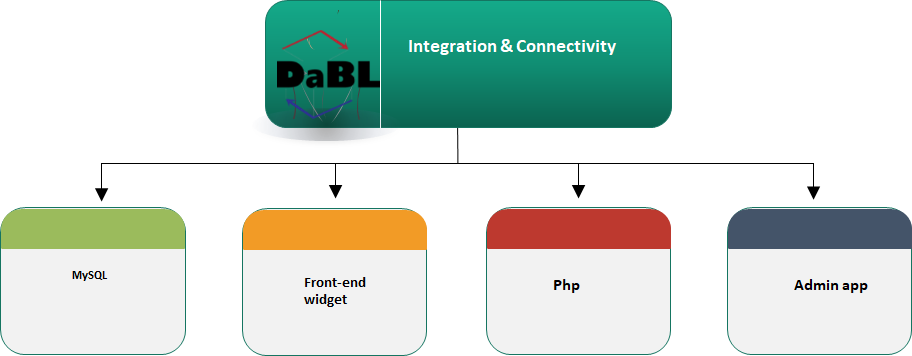
This approach indicated that the group would require the password of the AWS administrative account, which the client declined to provide us. Throughout an engagement with our client on how to approach this issue, the group opted to gain access to the MySQL directly (by SSHing into the AWS instance), so that we can add databases and tables manually to the server. A possible issue that we may encounter is that using the *WordPress* interface allows the developer to not worry about certain ways that *WordPress* does things, such as set up a database for interaction with the front-end. This may - though hopefully not - play out to where we have to do additional configurations within the *WordPress config* files in order to make sure everything integrates functionally.

**PHP**

The final issue occured with regards to the *php* implementation to the *WordPress* front-end. We know that we can use an *inline php* script directly to HTML to perform *php* functions. However, this results in exposing the MySQL *php* code to the user if they were to *inspect* the web page. In other words, this is not secure because the *php* script may contain some server information that is viewable by checking the source code through the browser. Therefore, the group is identifying other ways to work between *WordPRess* and *php* in the upcoming sprint.

**Plans For Next Sprint**

**Integration and connectivity**



Plans for the next sprint will primarily focus on bringing together all of the above component subsystems into one system so that the system will be able to deliver its functionality, while at the same time testing, debugging, risk assementment and maintenance etc will also be refactoring in. Below is a brief description on how we plan the system to to talk to each other.

The Php will be interacting with the front-end widget which serve as an interactive component for the user to schedule either 3D printers, laser cutter of box cutter. The frontend widget interact with a Mysql database ( which is populated with users information) to check whether the user is certified or have received training to use either of the tools and also checks if the user has a waiver signed. The admin app will run on the DABL computer and populate the local and Mysql databases with members information.

# **Summary and Conclusion**

In summary, the group performed a productive, though not fully successful sprint as we continue to encounter issues related to our evolving assumptions and understanding of the infrastructure, as well as the requisite accesses in order to test concepts and implementations. Nevertheless, the group is genuinely fortunate to have a customer that is both highly accessible, technically savvy, and flexible enough to provide guidance without micromanaging the project. This is critical to our success as both a student and potential future software engineers.

Ultimately, the problems that we have encountered have also highlighted agile methodologies that allowed us to successfully pivot to alternate requirements that do not deter from the client’s vision of the end result. Consequently, the requirements are beginning to codify into the tangible product. This and the following sprint will see the group complete these separate components and begin to focus on the integration aspect of the project.

# 

# **Appendix**

Send an email php script

1.<?php

2.usePHPMailer\PHPMailer\PHPMailer;

3.usePHPMailer\PHPMailer\Exception;

4.

5.requireDIR.'/PHPMailer/src/Exception.php';

6.requireDIR.'/PHPMailer/src/PHPMailer.php';

7.requireDIR.'/PHPMailer/src/SMTP.php';

8.

9.$mail=newPHPMailer;

10.$mail->isSMTP();

11.$mail->SMTPDebug=2;//0=off(forproductionuse)-1=clientmessages-2=clientandservermessages

12.$mail->Host="smtp.gmail.com";//use$mail->Host=gethostbyname('smtp.gmail.com');//ifyournetworkdoesnotsupportSMTPoverIPv6

13.$mail->Port=587;//TLSonly

14.$mail->SMTPSecure='tls';//sslisdepracated

15.$mail->SMTPAuth=true;

16.$mail->Username="ls8513a@student.american.edu";

17.$mail->Password="Taipinopwinopwin";

18.$mail->setFrom("no-reply@american.edu","no-reply@american.student.edu");

19.$mail->addAddress("opwinopwin@gmail.com","Jerry");

20.$mail->Subject='PHPMailerGMailSMTPtest';

21.$mail->msgHTML("testbody");//$mail->msgHTML(file\_get\_contents('contents.html'),DIR);//Readan

HTMLmessagebodyfromanexternalfile,convertreferencedimagestoembedded,

22.$mail->AltBody='HTMLmessagingnotsupported';

23.//$mail->addAttachment('images/phpmailer\_mini.png');//Attachanimagefile

24.

25.if(!$mail->send()){

26.echo"MailerError:".$mail->ErrorInfo;

27.}else{

28.echo"Messagesent!";

29.}

30.

31.?>

Interface with database php script

1.<?php

2.

3.classIDatabase{

4.privatestatic$servername="127.0.0.1";

5.privatestatic$username="phpUser";

6.privatestatic$password="0000";

7.privatestatic$dbname="dabl";

8.

9.functionconnectDB(){

10.$servername=self::$servername;

11.$username=self::$username;

12.$password=self::$password;

13.$dbname=self::$dbname;

14.

15.$conn=newmysqli($servername,$username,$password,$dbname);

16.if($conn->connect\_error){

17.die("Connectionfailed:".$conn->connect\_error);

18.}

19.else{

20.//echo"Connectedsuccessfully\n";

21.return$conn;

22.}

23.returnNull;

24.}

25.functionfetch\_All(){

26.$conn=self::connectDB();

27.$sql="SELECT\*FROMdabl.user";

28.$result=$conn->query($sql);

29.if($result->num\_rows>0){

30.while($row=$result->fetch\_assoc()){

31.echo"email:".$row["EMAIL"]."Agreement:".$row["AGREEMENT"]."Printer:".$row["PRINTER"]."Lasercutter:".$row["LASERCUTTER"]."Boxcutter:".$row["BOXCUTTER"]."\n";

32.}

33.}

34.else{echo"Noresult";}

35.$conn->close();

36.}

37.

38.functioninsert\_Data($email,$agreement,$printer,$lasercutter,$boxcutter){

39.$conn=self::connectDB();

40.$sql="INSERTINTOdabl.user(email,agreement,printer,lasercutter,boxcutter)VALUES('$email','$agreement','$printer','$lasercutter','$boxcutter')";

41.if($conn->query($sql)===TRUE){

42.echo"Oneinformationhasbeeninsert";

43.}else{

44.

45.echo"Error:".$sql."<br>".$conn->error;

46.}

47.}

48.functionsearch\_ByEmail($email){

49.$conn=self::connectDB();

50.$sql="SELECT\*FROMdabl.userWHEREemail='$email'";

51.$result=$conn->query($sql);

52.while($row=mysqli\_fetch\_array($result))

53.{

54.$agreement=$row["AGREEMENT"];

55.$printer=$row["PRINTER"];

56.$lasercutter=$row["LASERCUTTER"];

57.$boxcutter=$row["BOXCUTTER"];

58.//echo$row["EMAIL"]."Agreement:".$row["AGREEMENT"]."Printer:".$row["PRINTER"]."Lasercutter:".$row["LASERCUTTER"]."Boxcutter:".$row["BOXCUTTER"]."\n";

59.}

60.if($result!=Null){return$result;}

61.else{echo"Norecord";}

62.

63.}

64.functioncheck\_Data($email,$colname){

65.$conn=self::connectDB();

66.$sql="SELECT\*FROMdabl.userWHEREemail='$email'";

67.$result=$conn->query($sql);

68.while($row=mysqli\_fetch\_array($result))

69.{

70.$agreement=$row["AGREEMENT"];

71.$printer=$row["PRINTER"];

72.$lasercutter=$row["LASERCUTTER"];

73.$boxcutter=$row["BOXCUTTER"];

74.}

75.switch($colname){

76.case"agreement":

77.if($agreement==1){return1;}

78.else{return0;}

79.case"printer":

80.if($printer==1){return1;}

81.else{return0;}

82.case"lasercutter":

83.if($lasercutter==1){return1;}

84.else{return0;}

85.case"boxcutter":

86.if($boxcutter==1){return1;}

87.else{return0;}

88.}

89.}

90.functionupdate\_Data($emailAddress,$colName,$val){//didn'twork

91.$conn=self::connectDB();

92.$sql="UPDATEdabl.userSET'$colName'='$val'WHEREemail='$emailAddress'";

93.$conn->query($sql);

94.$conn->commit();

95.}

96.functiondelete\_Data($emailAddress){//didn'twork

97.$conn=self::connectDB();

98.$sql="DELETEFROMdabl.userWHEREemail='$emailAddress'";

99.$conn->query($sql);

100.$conn->commit();

101.}

102.

103.}

104.

105.$obj=newIDatabase;

106.

107.$obj->fetch\_All();

108.#$obj->insert\_Data("taipinKanSu@gmaill.com",1,1,1,1);

109.//$obj->search\_ByEmail("t25366021@yahoo.com.tw");

110.$obj->delete\_Data("taipinKanSu@gmail.com");

111.//$check\_result=$obj->check\_Data("t25366021@yahoo.com.tw","agreement");

112.//echo$check\_result;

113.

114.?>

Schedule on Google calendar

1.<?php

2.require'C:/Users/KanSu/vendor/autoload.php';

3.

4.if(php\_sapi\_name()!='cli'){

5.thrownewException('Thisapplicationmustberunonthecommandline.');

6.}

7.

8./\*\*

9.\*ReturnsanauthorizedAPIclient.

10.\*@returnGoogle\_Clienttheauthorizedclientobject

11.\*/

12.functiongetClient()

13.{

14.$client=newGoogle\_Client();

15.$client->setApplicationName('GoogleCalendarAPIPHPQuickstart');

16.$client->setScopes(Google\_Service\_Calendar::CALENDAR\_READONLY);

17.$client->setAuthConfig('credentials.json');

18.$client->setAccessType('offline');

19.$client->setPrompt('select\_accountconsent');

20.

21.//Loadpreviouslyauthorizedtokenfromafile,ifitexists.

22.//Thefiletoken.jsonstorestheuser'saccessandrefreshtokens,andis

23.//createdautomaticallywhentheauthorizationflowcompletesforthefirst

24.//time.

25.$tokenPath='token.json';

26.if(file\_exists($tokenPath)){

27.$accessToken=json\_decode(file\_get\_contents($tokenPath),true);

28.$client->setAccessToken($accessToken);

29.}

30.

31.//Ifthereisnoprevioustokenorit'sexpired.

32.if($client->isAccessTokenExpired()){

33.//Refreshthetokenifpossible,elsefetchanewone.

34.if($client->getRefreshToken()){

35.$client->fetchAccessTokenWithRefreshToken($client->getRefreshToken());

36.}else{

37.//Requestauthorizationfromtheuser.

38.$authUrl=$client->createAuthUrl();

39.printf("Openthefollowinglinkinyourbrowser:\n%s\n",$authUrl);

40.print'Enterverificationcode:';

41.$authCode=trim(fgets(STDIN));

42.

43.//Exchangeauthorizationcodeforanaccesstoken.

44.$accessToken=$client->fetchAccessTokenWithAuthCode($authCode);

45.$client->setAccessToken($accessToken);

46.

47.//Checktoseeiftherewasanerror.

48.if(array\_key\_exists('error',$accessToken)){

49.thrownewException(join(',',$accessToken));

50.}

51.}

52.//Savethetokentoafile.

53.if(!file\_exists(dirname($tokenPath))){

54.mkdir(dirname($tokenPath),0700,true);

55.}

56.file\_put\_contents($tokenPath,json\_encode($client->getAccessToken()));

57.}

58.return$client;

59.}

60.

61.

62.//GettheAPIclientandconstructtheserviceobject.

63.$client=getClient();

64.$service=newGoogle\_Service\_Calendar($client);

65.

66.//Printthenext10eventsontheuser'scalendar.

67.$calendarId='primary';

68.$optParams=array(

69.'maxResults'=>10,

70.'orderBy'=>'startTime',

71.'singleEvents'=>true,

72.'timeMin'=>date('c'),

73.);

74.$results=$service->events->listEvents($calendarId,$optParams);

75.$events=$results->getItems();

76.

77.if(empty($events)){

78.print"Noupcomingeventsfound.\n";

79.}else{

80.print"Upcomingevents:\n";

81.foreach($eventsas$event){

82.$start=$event->start->dateTime;

83.if(empty($start)){

84.$start=$event->start->date;

85.}

86.printf("%s(%s)\n",$event->getSummary(),$start);

87.}

88.}

89.$event=newGoogle\_Service\_Calendar\_Event(array(

90.'summary'=>'GoogleI/O2015',

91.'location'=>'800HowardSt.,SanFrancisco,CA94103',

92.'description'=>'AchancetohearmoreaboutGoogle\'sdeveloperproducts.',

93.'start'=>array(

94.'dateTime'=>'2015-05-28T09:00:00-07:00',

95.'timeZone'=>'America/Los\_Angeles',

96.),

97.'end'=>array(

98.'dateTime'=>'2015-05-28T17:00:00-07:00',

99.'timeZone'=>'America/Los\_Angeles',

100.),

101.'recurrence'=>array(

102.'RRULE:FREQ=DAILY;COUNT=2'

103.),

104.'attendees'=>array(

105.array('email'=>'lpage@example.com'),

106.array('email'=>'sbrin@example.com'),

107.),

108.'reminders'=>array(

109.'useDefault'=>FALSE,

110.'overrides'=>array(

111.array('method'=>'email','minutes'=>24\*60),

112.array('method'=>'popup','minutes'=>10),

113.),

114.),

115.));

116.

117.$calendarId='primary';

118.$event=$service->events->insert($calendarId,$event);

119.printf('Eventcreated:%s\n',$event->htmlLink);