Lab Assignment 1 Oauth and Mashup Web Application

Team Details

Muhammad Ali

Class id: 61

Gregory Brown

Class id: 5

Team 2-2

Objective

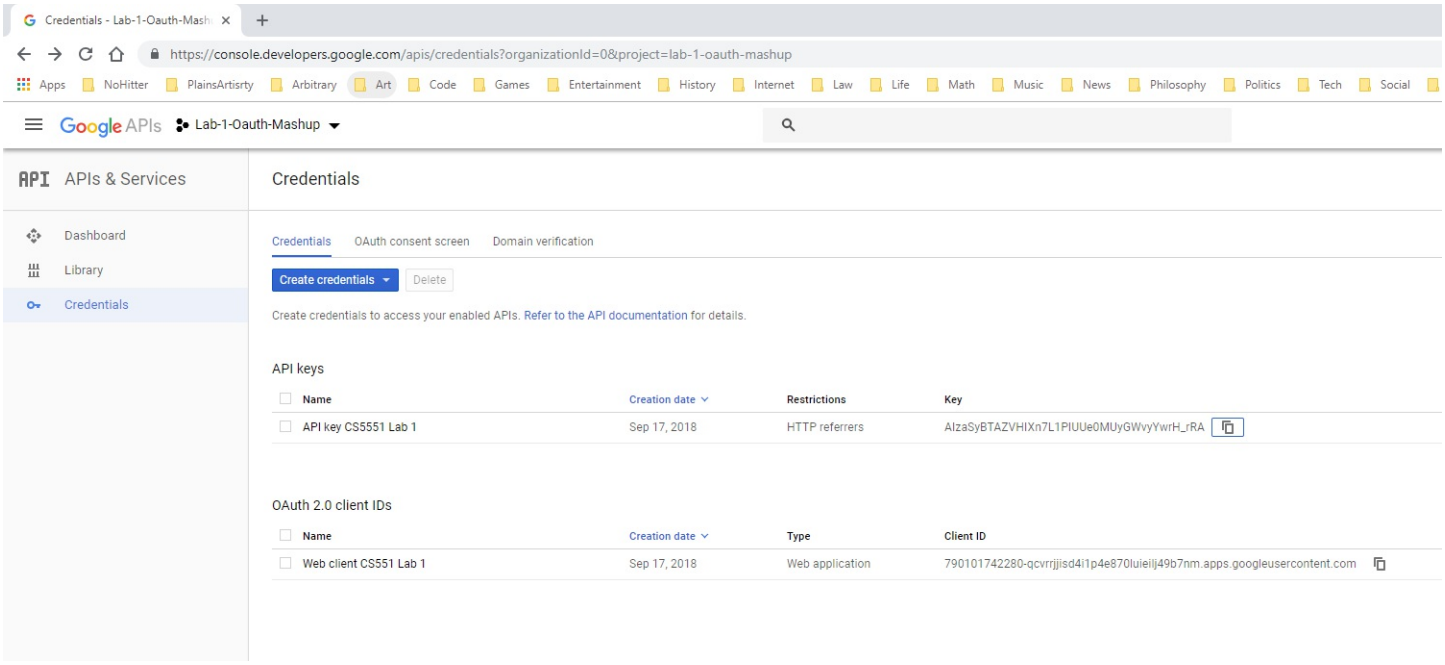
Create a web application which uses OAuth 2.0 social login, as well as custom localStorage login, a mashup of APIs, and pages about our class projects and to contact us.

1.Login and Register Pages

a) OAuth Login & Registration

We have used googlees OAuth2.0 service to provide the social login. This requires getting an OAuth2.0 client ID as shown in the screenshot below.

Note that we also obtained an API key and enabled it for use with Googles Knowledge Graph Search API.



Next we included provide our clientID and set up buttons for login and logout. These buttons refer to ‘onSignIn()’ and 'signOut() functions. Notably the sign in button is handled by the class name of the , “g-signin2” and it is not necessary to create a button object.

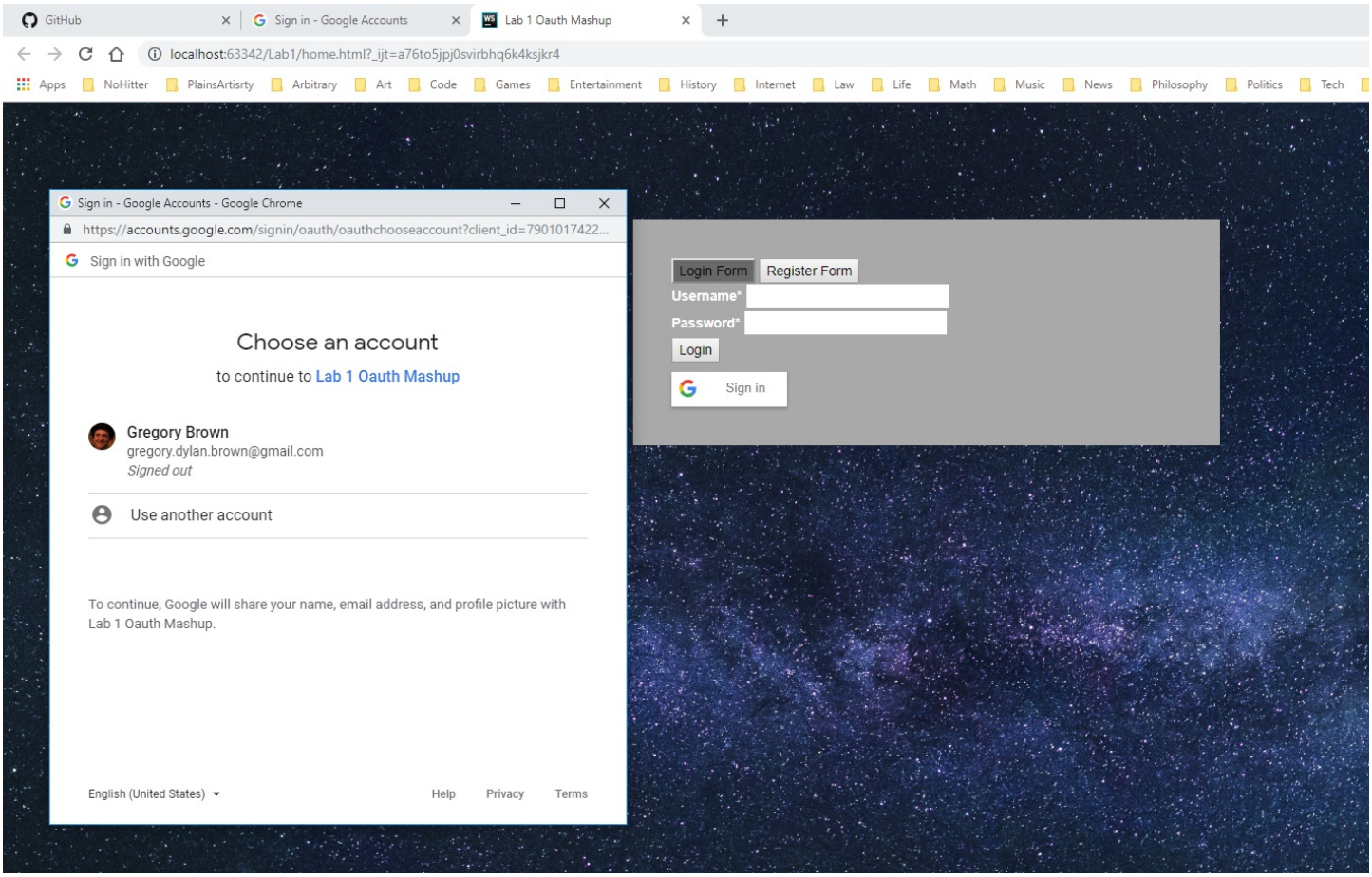
```
<meta name="google-signin-client_id" content="790101742280-qcvrrjjisd4i1p4e870luieilj
...
<div class="g-signin2" data-onsuccess="onSignIn"></div>
...
<button onclick="signOut()" class="button">Sign Out</button>
```

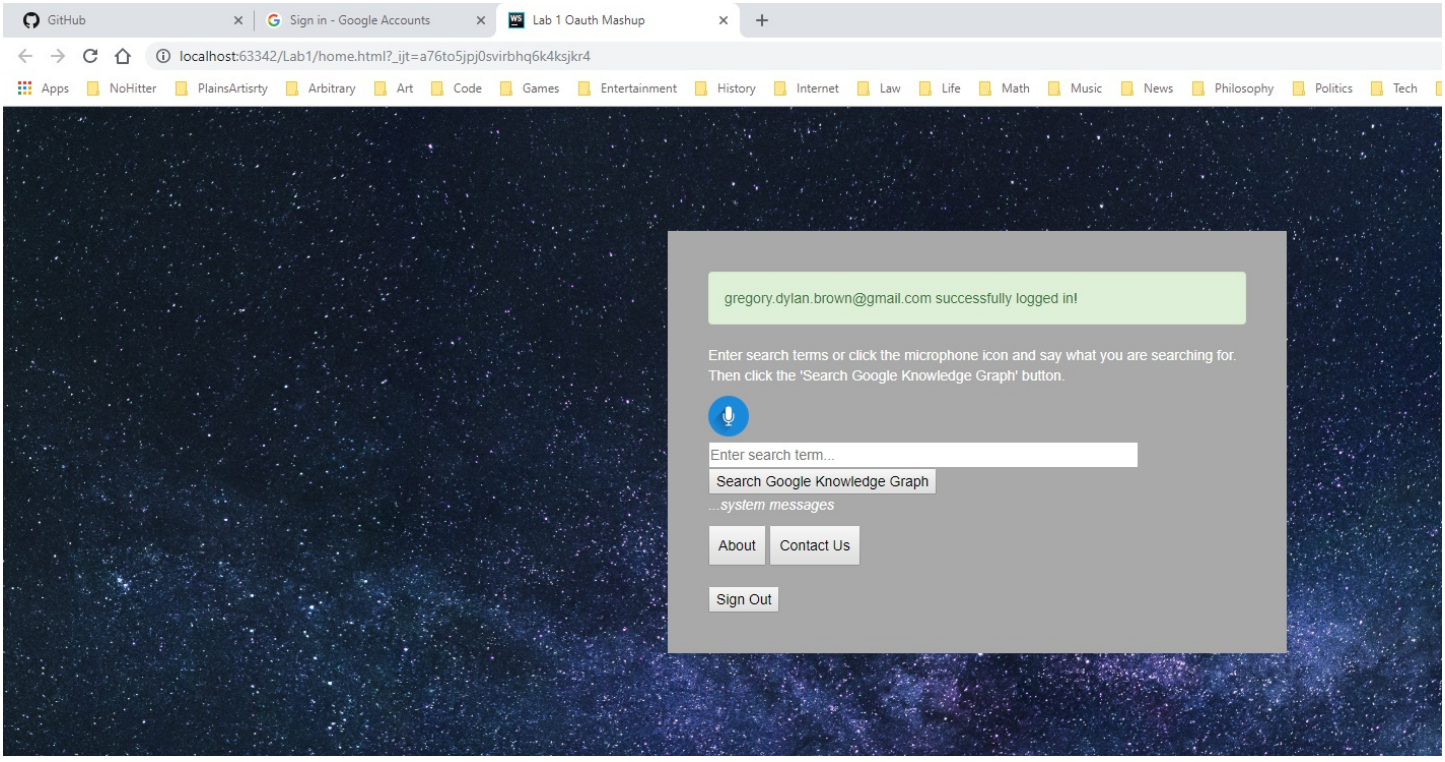
Finally we need to provide the functionality of hiding the parts of the web app which should not be available to the user until they sign in, and re-hiding them after the user signs out. This is handled in our ‘onSignIn(googleUser)’ function and our ‘signOut()’ function. The sign in

function also handles displaying successful sign in information.

```
function onSignIn(googleUser){
    "use strict";
    var profile=googleUser.getBasicProfile();
    $(".g-signin2").css("display","none");
    $(".login").css("display","none");
    $(".content").css("display","block");
    $("#sucess").text(profile.getEmail()+" successfully logged in!");
}

function signOut(){
    "use strict";
    var auth2 = gapi.auth2.getAuthInstance();
    auth2.signOut().then(function(){
        $(".g-signin2").css("display","block");
        $(".login").css("display","block");
        $(".content").css("display","none");

    });
}
```

Below is a screenshot of the google sign-in in action followed by a screenshot of the page once a user is successfully signed in:





b) Local Storage

To create a local storage based login system, we have created two

functions: ‘login()’ and ‘register()’. We use two additional functions to allow the user to swap back and forth between the login form and the registration form. A user must first register in order to be able to login with the local storage. Their registered ‘email’, ‘un’ or username, and ‘pw’ or password are then stored and the user is returned to the login form view.

```
$scope.login = function() {
    if (document.getElementById("username").value !== "" && document.getElementById("password").value !== "") {
        if (document.getElementById("username").value === localStorage.getItem("un") && document.getElementById("password").value === localStorage.getItem("pw")) {
            $(".g-signin2").css("display","none");
            $(".login").css("display","none");
            $(".content").css("display","block");
            $("#sucess").text(localStorage.getItem("email")+" successfully logged in!");
        }
        else {
            document.getElementById("logError").innerHTML = "Username or Password does not match!";
        }
    }
    else {
        document.getElementById("logError").innerHTML = "Please provide username and password!";
    }
};

$scope.register = function() {
    if (document.getElementById("un").value !== "" && document.getElementById("email").value.length !== "" && document.getElementById("pw").value.length !== "") {
        localStorage.setItem("un", (document.getElementById("un").value));
        localStorage.setItem("pw", (document.getElementById("pw").value));
        localStorage.setItem("email", (document.getElementById("email").value));
        window.location.href = "home.html";
    }
    else {
        document.getElementById("regError").innerHTML = "Please provide username, email, and password!";
    }
};
```

After a user has registered, they can then login by providing their ‘username’ and ‘password’ which are compared to the ‘un’ and ‘pw’ variables. These functions are called by the login and register buttons at the bottoms of their respective forms in the html.

2.Home Page

For our home page we created a mashup web application using the Google Knowledge Graph Search API and the Web Speech API.
We began by implementing the Google Knowledge Graph Search. The HTML is simple:

```
<input type="text"  ng-model="searchTerm" style="width: 80%;color: black;"  placeholder="Enter search term...">
<input type="button" ng-click="gKnowledgeSearch()" style="color: black" class="button" value="Search Google Knowledge Graph">
```

We create a text field to take the input search terms, and then create a button to call ‘gKnowlegeSearch()’ Function. In order to call the Google Knowledge Graph Search API, we set up params providing the search term and our API key for our web app. If we get a valid response we use local storage to display that information on the home page. We can tell if the response is validbecause we are searching for only the first response and an invalid result will have ‘response.itemListElement.length’ of 0. If not we let the user know that Google Knowledge Graph Search API returned no results for the search term.

```
$scope.gKnowledgeSearch = function() {
    var service_url = "https://kgsearch.googleapis.com/v1/entities:search";
    var params = {
        "query": $scope.searchTerm,
        "limit": 1,
        "indent": true,
        "key" : "AIzaSyBTAZVHIXn7L1PIUUe0MUyGWvyYwrH_rRA"
    };
    $.getJSON(service_url + "?callback=?", params, function(response) {
        if(response.itemListElement.length === 1){
            localStorage.setItem("resName", response.itemListElement[0].result.name);
            localStorage.setItem("resDesc", response.itemListElement[0].result.detailedDescription.articleBody);
            document.getElementById("resName").innerHTML = localStorage.getItem("resName");
            document.getElementById("resDesc").innerHTML = localStorage.getItem("resDesc");
        }else{
            document.getElementById("resName").innerHTML = "No result for search term:  " + $scope.searchTerm;
            document.getElementById("resDesc").innerHTML = "";
        }
    });
};
```

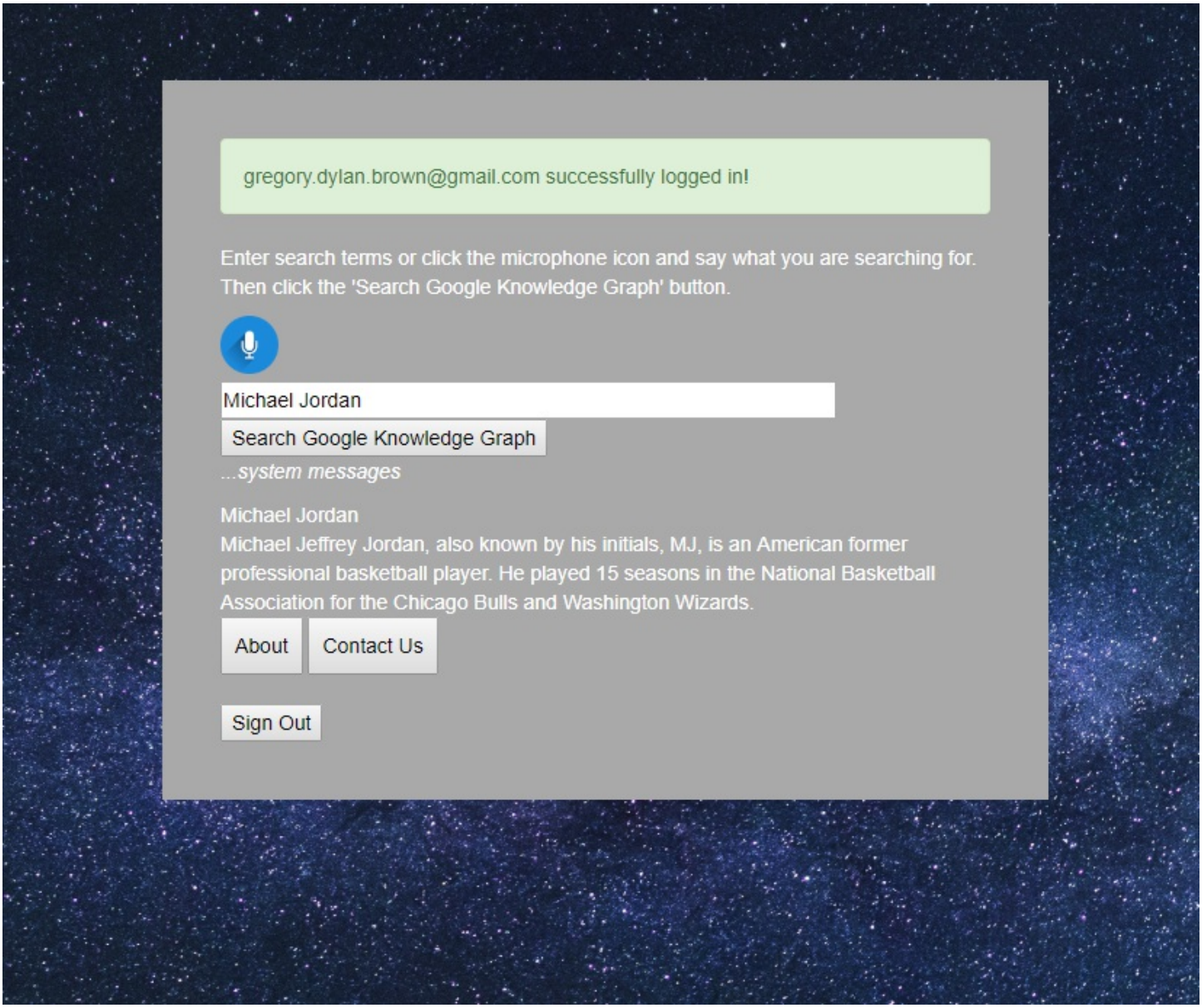
Next we include the Web Speech API function. The HTML for this is similar to that for the Google Search API; we create a button which will initiate listening, by calling our ‘listen()’ function. This function simply starts the ‘SpeechRecognition’. Once the user finishes speaking the the ‘SpeechRecognition’ is ended by the ‘onspeechend()’ function. The

‘onresult()’ function is where we can set the word recognized to our search term variable which will then be passed to the Google Search API. Here we also display the recognized term(s) to the user. The ‘onnomatch()’ and ‘onerror()’ functions handle issues with the speech recognition.

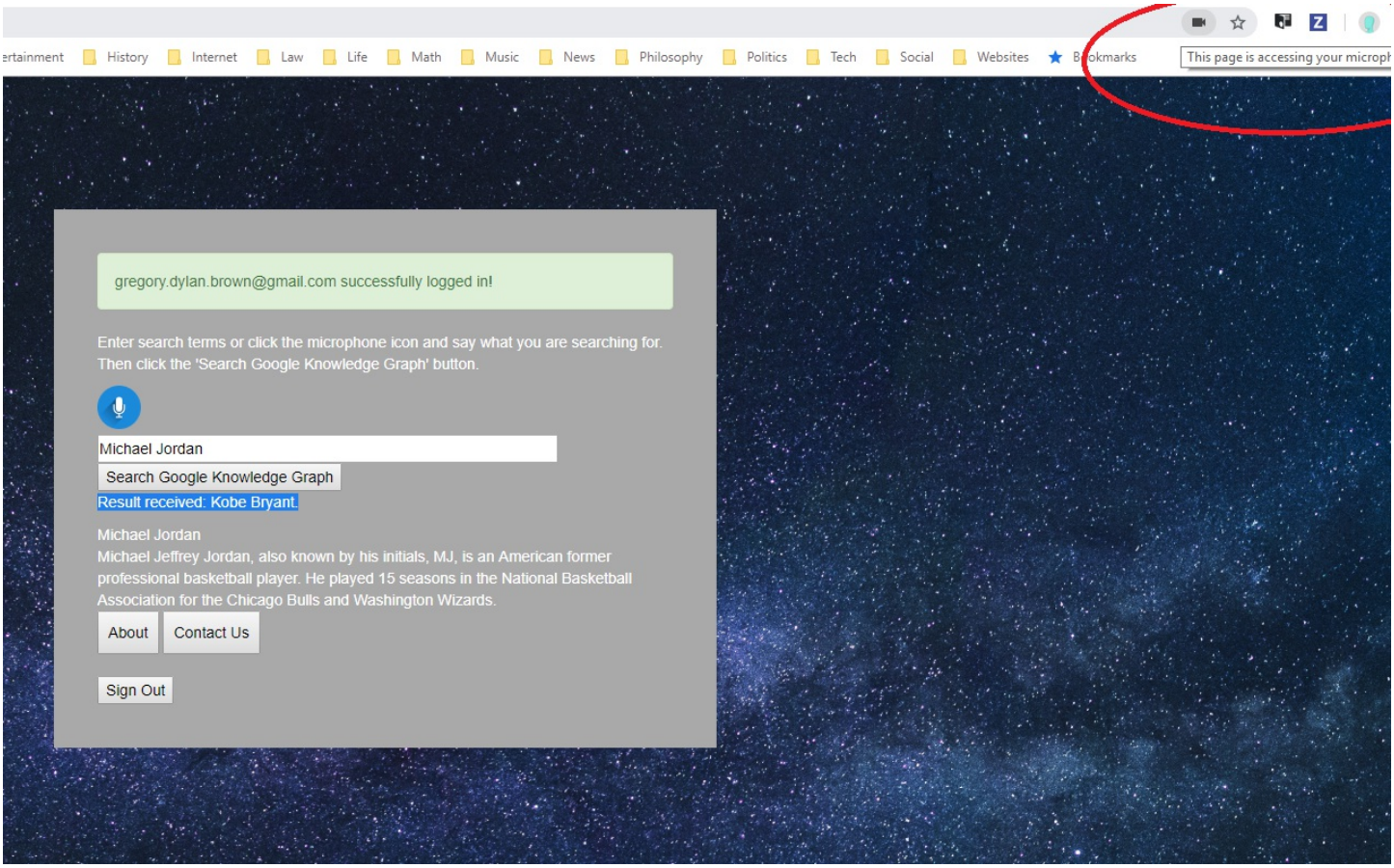
```
var SpeechRecognition = SpeechRecognition || webkitSpeechRecognition;
var SpeechGrammarList = SpeechGrammarList || webkitSpeechGrammarList;
var SpeechRecognitionEvent = SpeechRecognitionEvent || webkitSpeechRecognitionEvent;

var words = [];
var grammar = "#JSGF V1.0; grammar words; public <word> = " + words.join(" | ") + " ;"

var recognition = new SpeechRecognition();
var speechRecognitionList = new SpeechGrammarList();
speechRecognitionList.addFromString(grammar, 1);
recognition.grammars = speechRecognitionList;
recognition.lang = "en-US";
recognition.interimResults = false;
recognition.maxAlternatives = 1;

var diagnostic = document.querySelector(".output");

$scope.listen = function() {
    recognition.start();
};

recognition.onresult = function(event) {
    var last = event.results.length - 1;
    var word = event.results[last][0].transcript;

    $scope.searchTerm = word;
    diagnostic.textContent = "Result received: " + word + ".";
};

recognition.onspeechend = function() {
    recognition.stop();
};

recognition.onnomatch = function(event) {
    diagnostic.textContent =
        "Didn't recognise that word.";
};

recognition.onerror = function(event) {
    diagnostic.textContent = "Error occurred in recognition: " + event.error;
};
```

Below are screenshots which demonstrate the use of the mashup web app. First is a simple Google Knowledge Graph Search using the text box. As you can see the name of the entity and the detailed description are

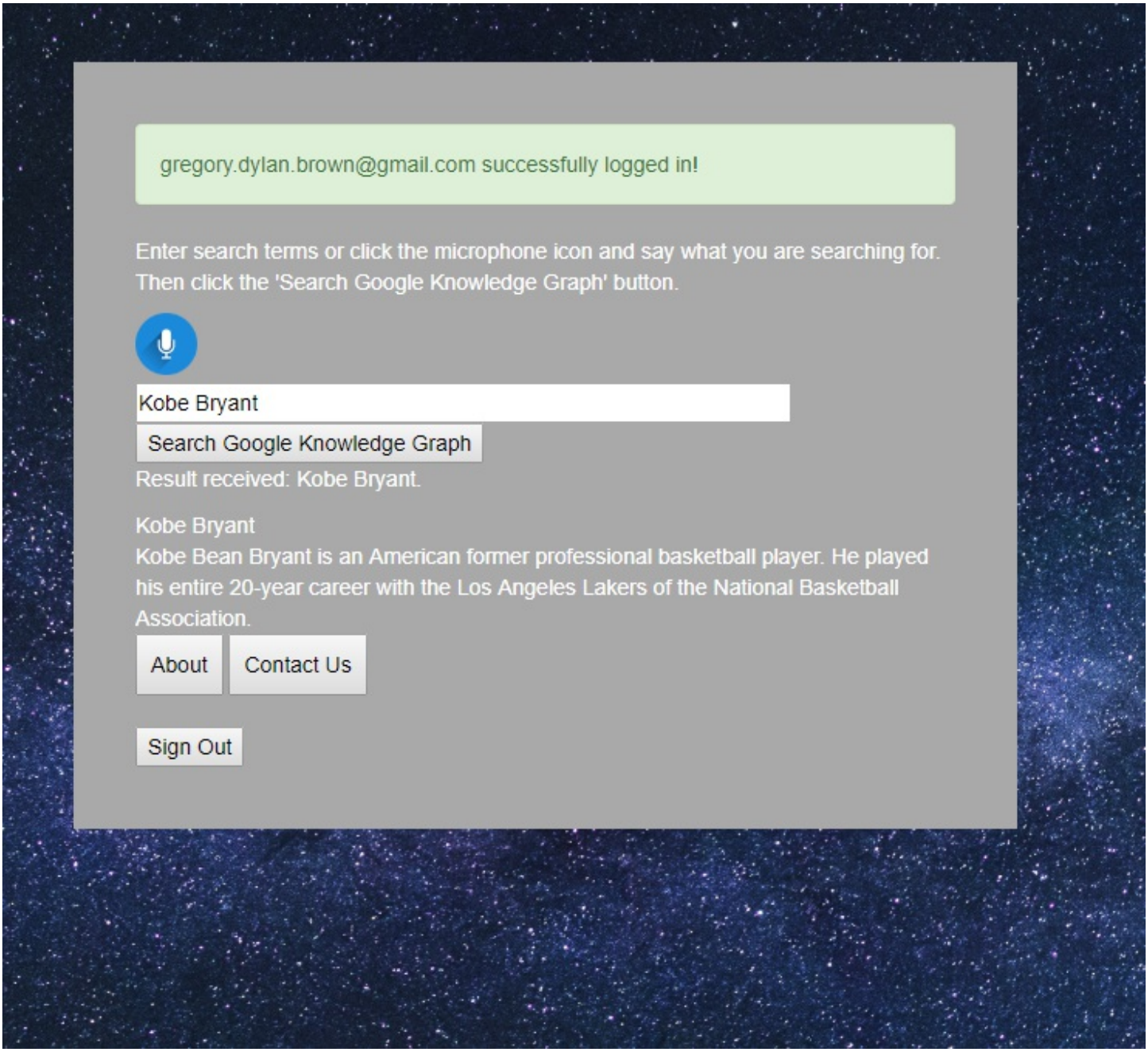
displayed.



This next image demonstrates the use of the speech recognition button by clicking the microphone icon. Please note that in this image I highlighted the diagnostic text showing that the spoken search term was recognized, and that I have circled in red the notification from the browser that audio is being recorded. On first use, a user will have to allow the browser to record audio.



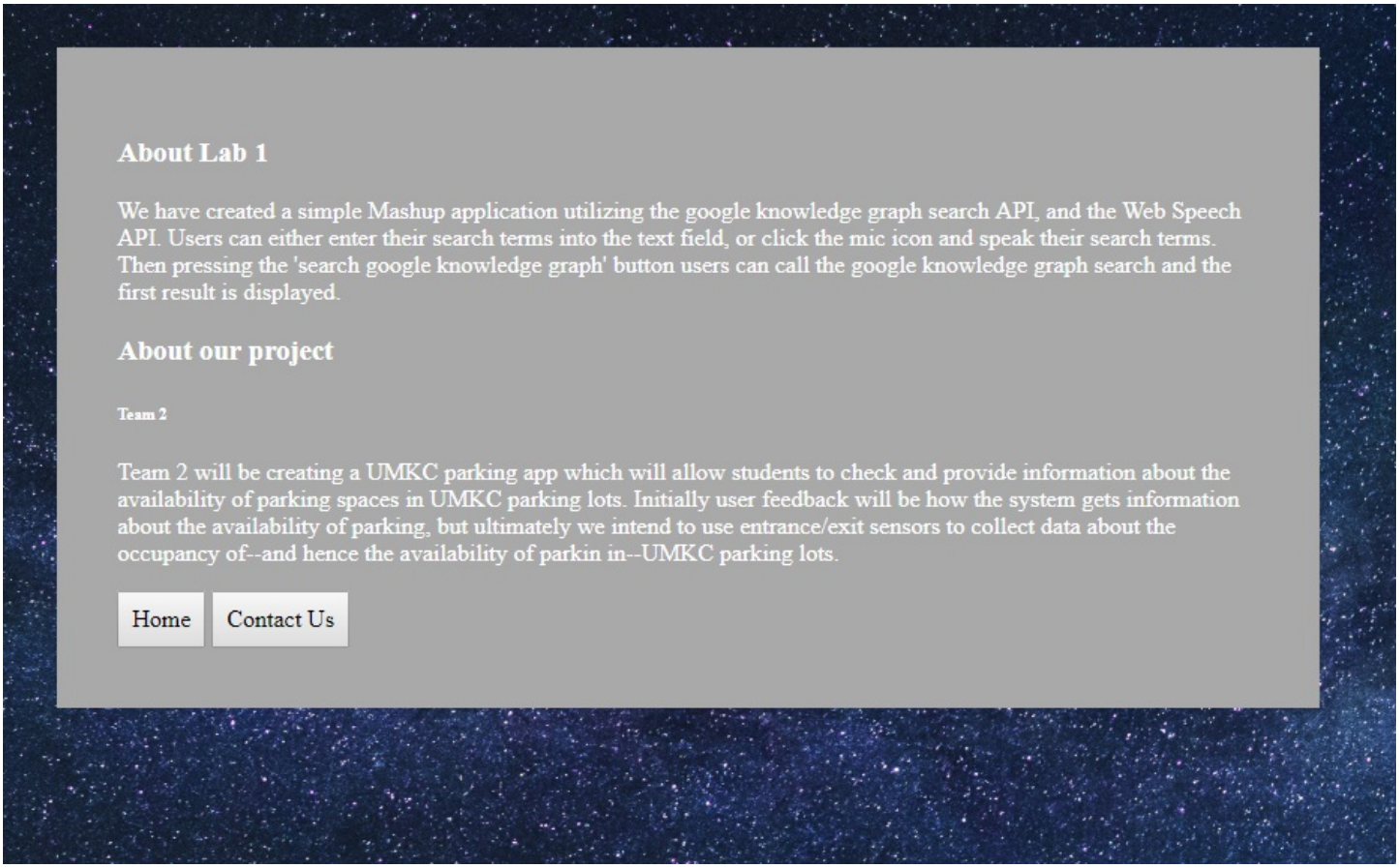
Finally, after clicking the ‘Search Google Knowledge’ button the spoken term is searched and the results are displayed as shown below:



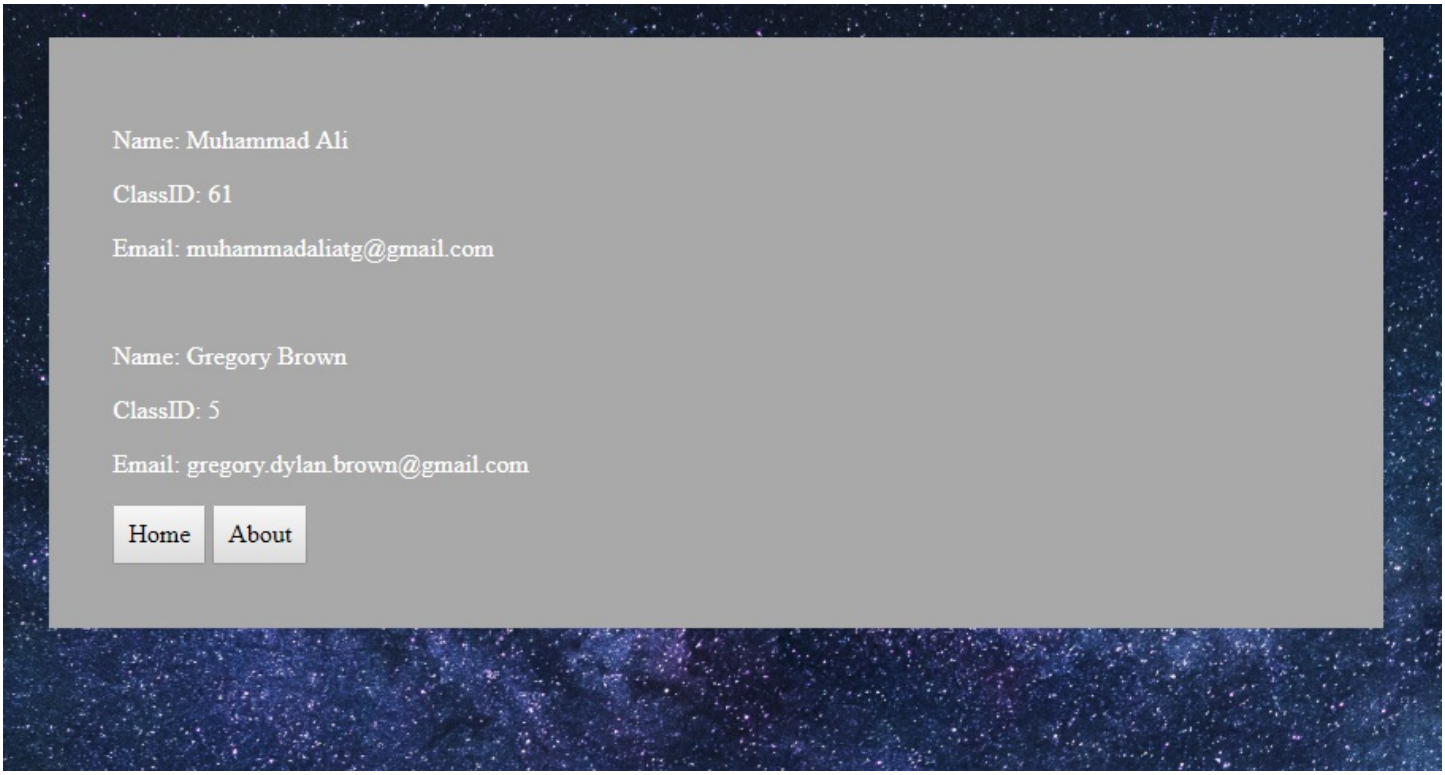
3.About and Contact Pages

We have created an “about” page which explains what was done in this lab, and about our plan for the semester project. Additionally we created a “contact us” page which has contact information and information about us, i.e. name and classID#. These pages are linked from the home page, as can be seen in the screenshot of the home page, and they link to one another.

About Page:



Contact us Page:



4.User Interface

We used custom css3 and bootstrap to help make the web application look nice. We have included non-copyrighted background image and a non-copyrighted image of microphone. Below is our style.css, but we also make use of several classes provided by bootstrap such as ‘col-md-4’, ‘col-md-offset-4’, and ‘alert alert-success’:

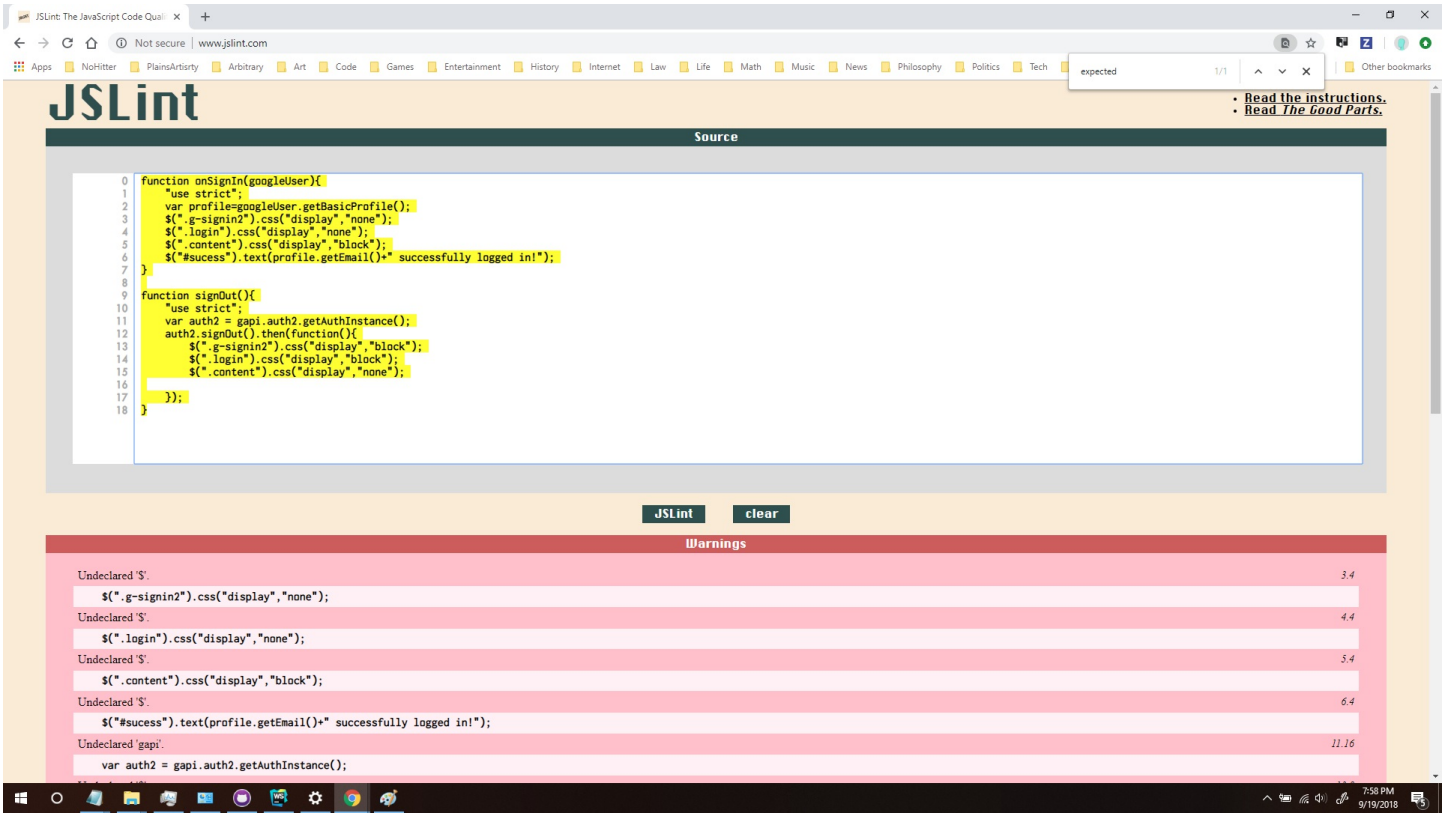
```
body {
    background-color: silver;
    background-image: url("http://localhost:63342/Lab1/source
    background-size: cover;
    color: white;
}

.login{
    background-color: darkgrey;
    color: white;
    padding: 40px;
```

```
    margin-top: 20%;
}

.content{
    display: none;
    background-color: darkgrey;
    color: white;
    padding: 40px;
    margin-top: 20%;
}

.container {
    background-color: darkgrey;
    width: 40%;
    min-width: 300px;
    margin-left: auto;
    margin-right: auto;
    margin-top: 10%;
    margin-bottom: auto;
    padding: 40px;
}

a.button {
    -webkit-appearance: button;
    -moz-appearance: button;
    appearance: button;
    text-decoration: none;
    color: black;
    padding: 10px;
}
.button{
    color: black;
```

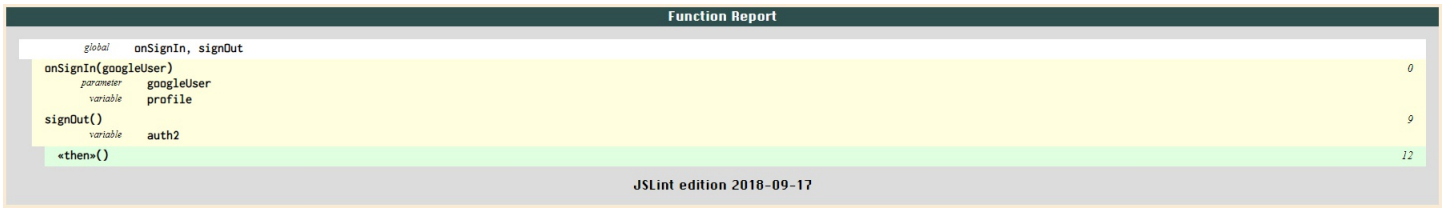
5.Quality Check

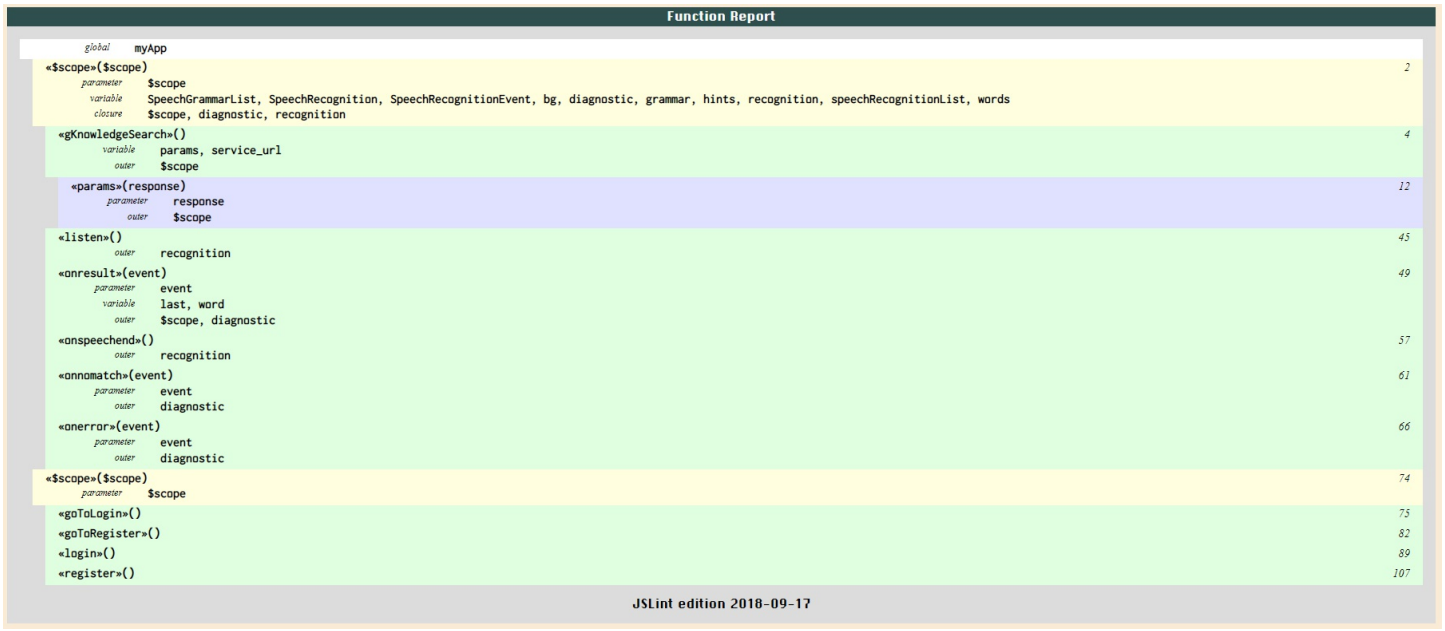
We used JSLint to find issues with our code, ultimately we solved all the issues except for ‘undeclared ___’ issues and issues where the line is over

80 characters. Most of the warning were about use of single quotes instead of double quotes.



Above is an example of final check of script.js. Below are the function reports for script.js and function.js



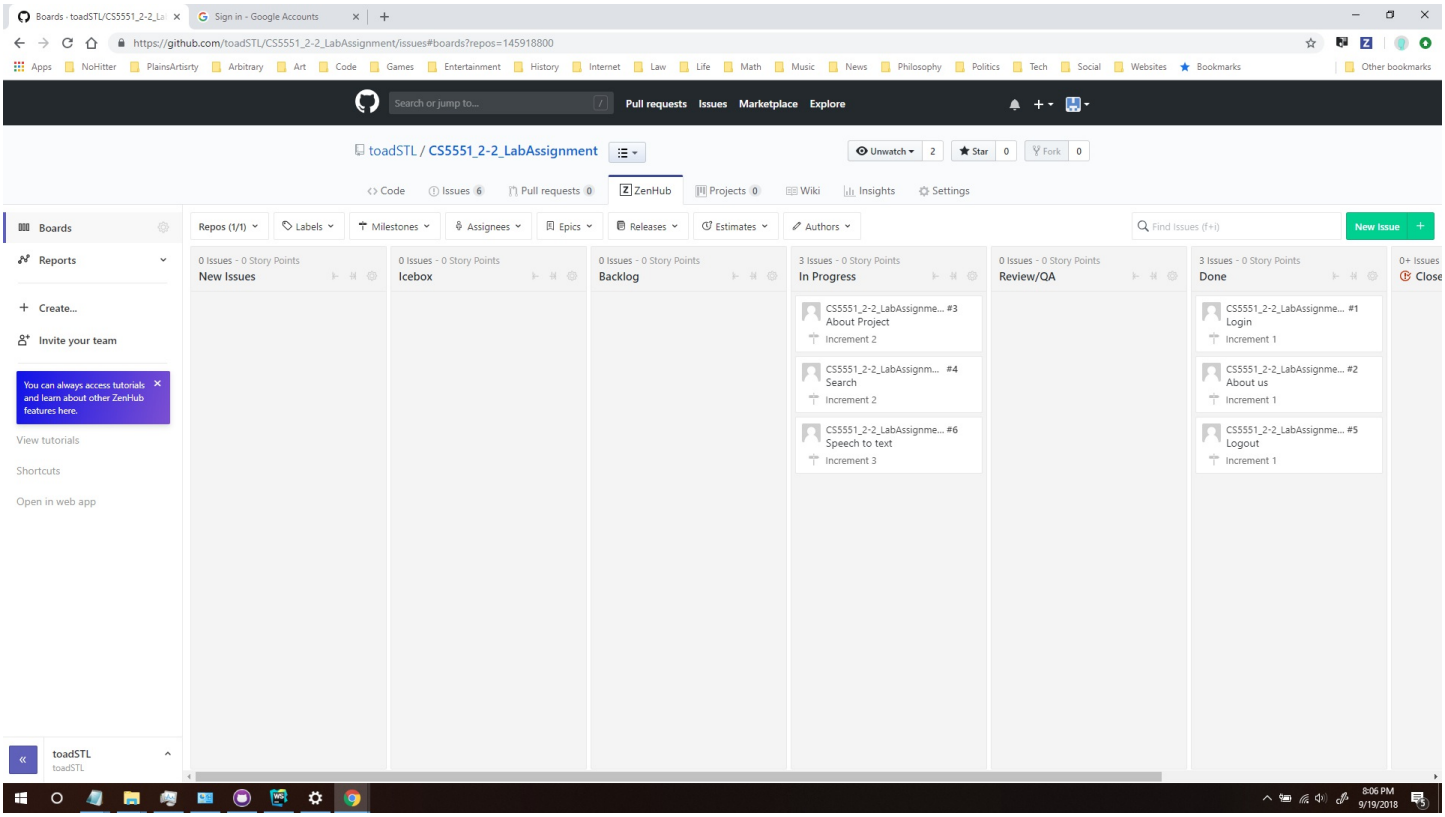


6.Create GitHub Account

As can be seen in the code section of GitHub, we created a repository for this project and have put our code in the ‘source’ directory and our images for the wiki in the ‘documentation’ directory.

7.Create ZenHub Tool Account

For this lab assignment we used ZenHub to create 6 issues, and 3 milestones: increment 1, increment 2, and increment 3.



Unfortunately we did not update our milestones as we progressed, and because of the due date, we had to change the due date of the milestones, in order to see their end in these burndown charts. Below are the burndown charts of each of the three milestones:

Increment 1 | Create new Milestone | Edit Milestone

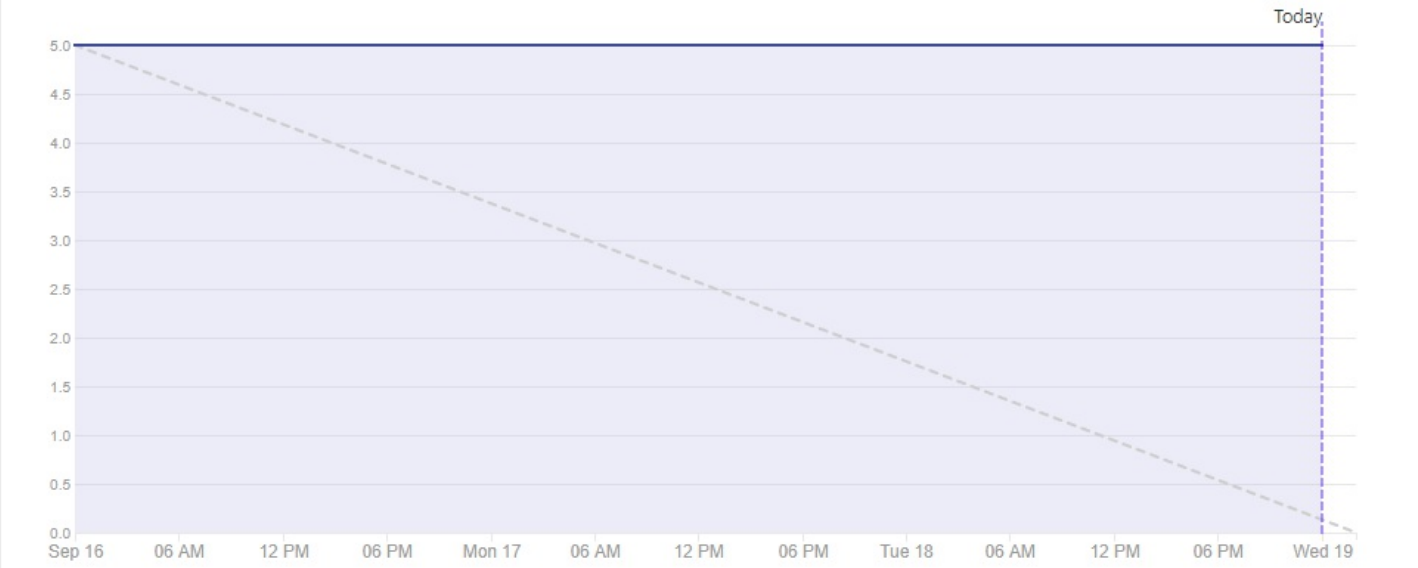
Increment 1

Sign in via google

Start **Sep 16, 2018** Change Due by **Sep 19, 2018 - Due today** Change

Labels | Hide Pull Requests | Burn Pipelines

Burndown report



5 Total Story Points

0 Completed / **5** Remaining

3 Total Issues and Pull Requests

0 Completed / **3** Remaining

Remaining Issues and Pull Requests **Story points**

Code | Issues 6 | Pull requests 0 | ZenHub | Projects 0 | Wiki | Insights | Settings

Increment 2 | Create new Milestone | Edit Milestone

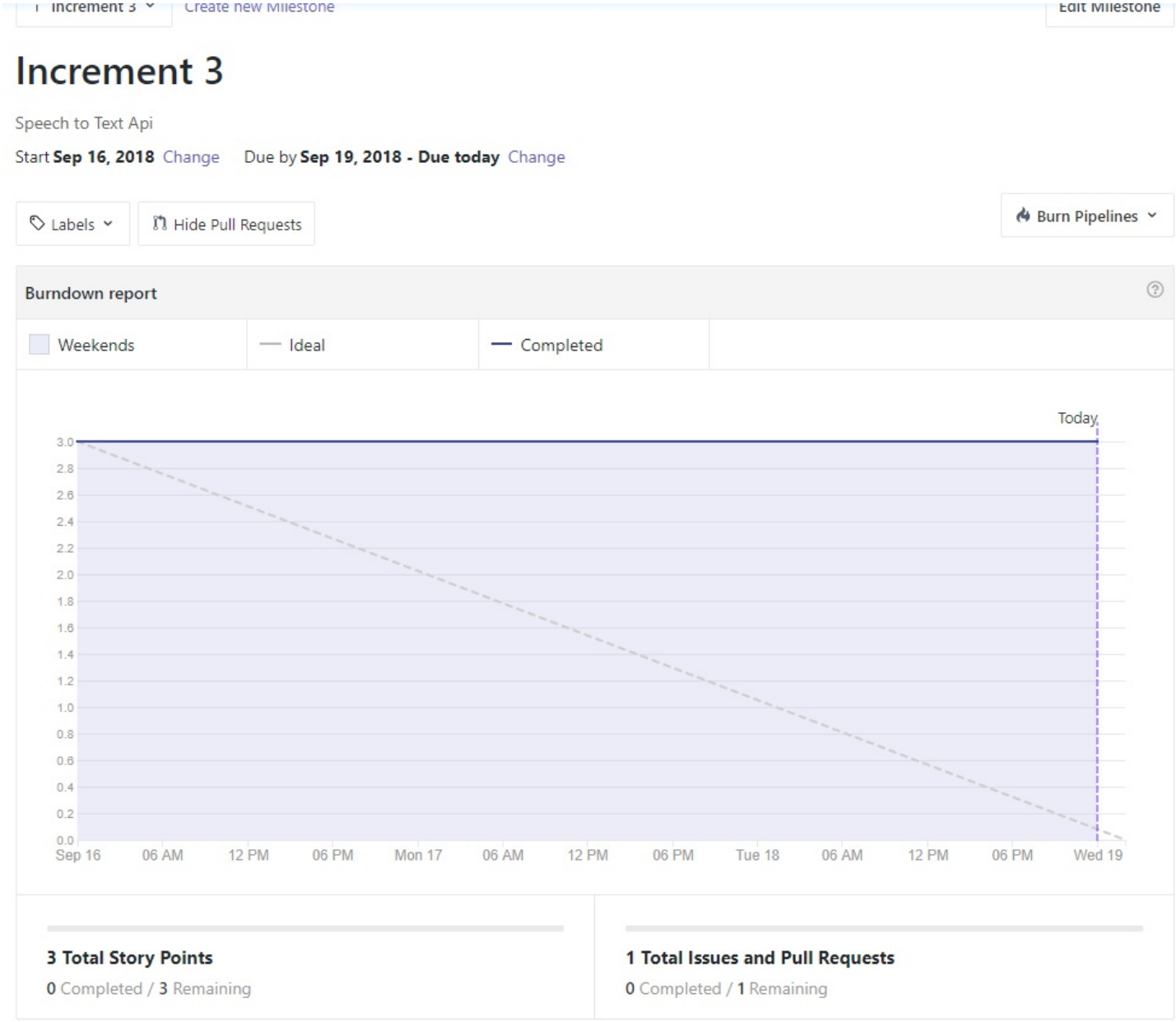
Increment 2

Google knowledge search api

Start **Sep 16, 2018** Change Due by **Sep 19, 2018 - Due today** Change

Labels | Hide Pull Requests | Burn Pipelines

Burndown report



8.Create Wiki page

This is the wiki page we have created for this lab assignment. See contributions below (NB: we worked locally together, so all commits/pushes to the repository are from Greg, hence we have not included a contribution chart. For future labs we will remedy this.):

Muhammad’s Contributions:

OAuth login via google

Local storage register and login

About page

Contact page

CSS

Greg’s contributions:

Home page-including:

- Speech to text using Web Speech API (Please note only works in chrome)*
- Google Knowledge Graph API

UML Class Diagram

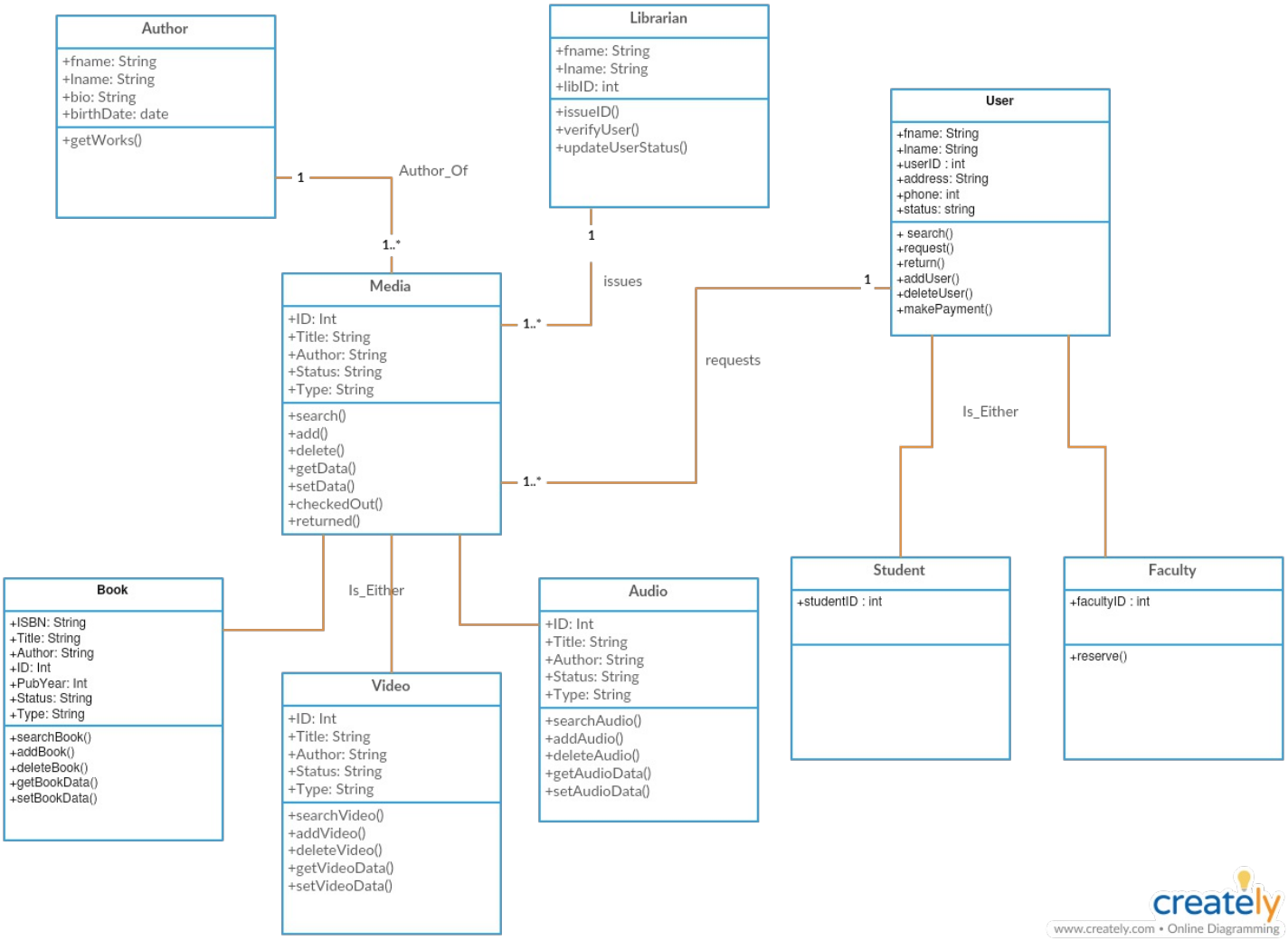
ZenHub Issues and Burndown

Use JSLint to clean up and improve javascript

*actually there is a workaround for firefox explained here.

9.Creatly Tool

Additionally we created the UML class diagram for a school library:



References

Google Oauth2.0

Web Speech API

Google Knowledge Graph API

Ruthvic’s Example