CSEE5590/CS490 APS
Programming for Web/Cloud/Mobile Applications

Lab Assignment 2

Gulnoza Khakimova (8) and Jackie Batson (3)

Task 1 - Search and view videos using YouTube API

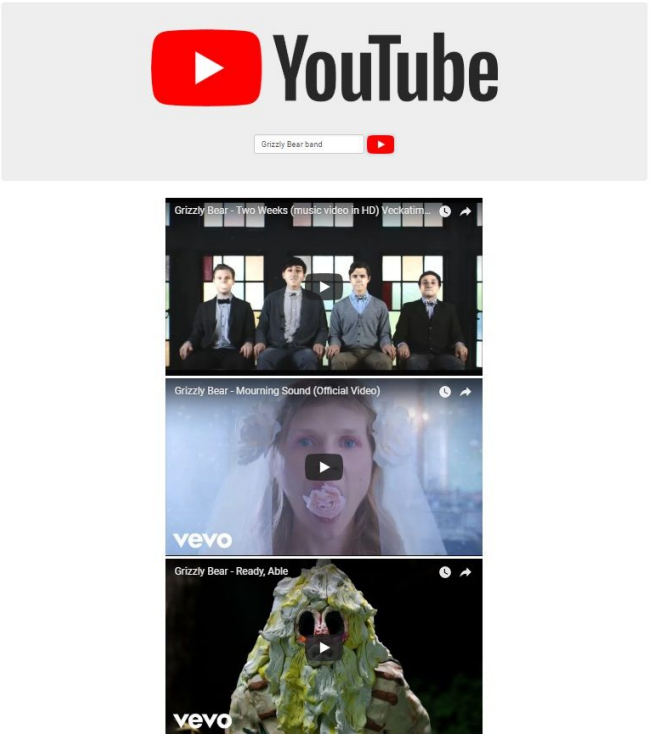
Objective

The goal of Task 1 was to create a simple one page web application where a user can search for videos and then be able to view the requested videos.

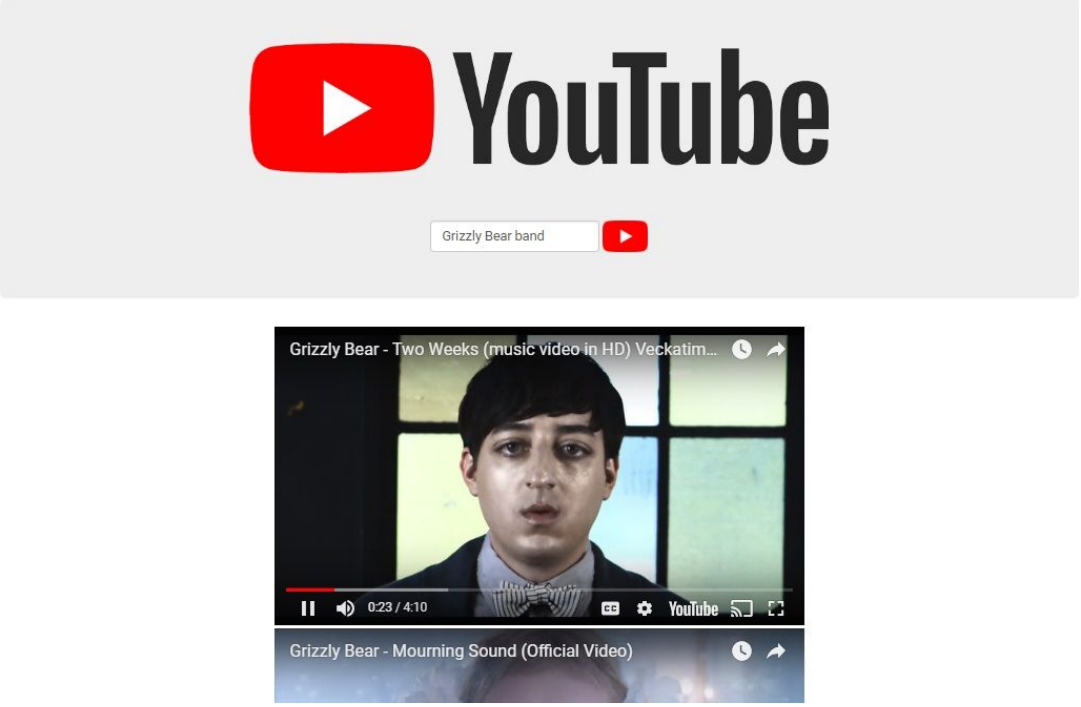
Features and Steps

The design is a simple layout with a logo, search bar, and submit button in the style of the YouTube logo.

The user enters a search term in the search bar and presses the submit button to see the returned results in the same page. The results are defaulted to return 5 videos, not playlists or channels.

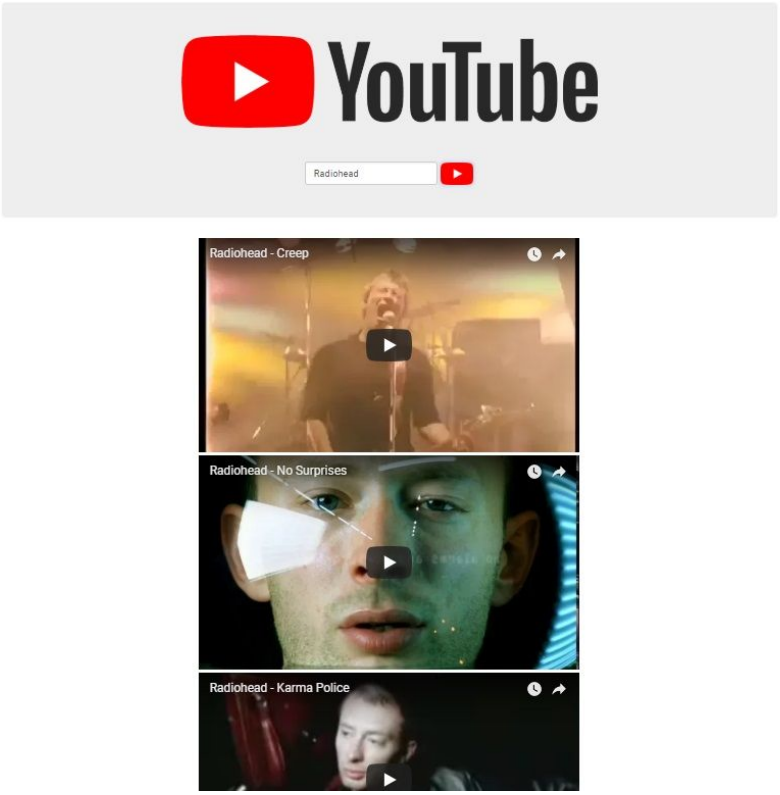


The user can watch the full video on the page and has full access to the controls.



The video can also be made full-screen.

A new search can be conducted with the results updating dynamically, without refreshing the page.



Code

The code for Task 1 is simple and requires only three files: index.html, style.css, and script.js. The application runs completely in the client, with no back end. The app makes use of AngularJS and the YouTube API in order to provide dynamic content.

index.html contains a Bootstrap jumbotron containing the YouTube logo and an inline form containing the search input and the submit button. The videos are then displayed dynamically using Angular's ng-repeat directive on the resulting videoList.

```
<!DOCTYPE html>
<html lang="en" ng-app="youTubeApp">
<head>
    <meta charset="UTF-8">
    <title>YouTube Search</title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
          integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
    <!-- Google Fonts Roboto CSS-->
    <link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
    <link rel="stylesheet" href="style.css">

</head>
<body ng-controller="AppCtrl">

<div class="container">
    <div class="jumbotron text-center">
        <img src="../resources/youtube_full_color_light_logo/digital_and_tv/yt_logo_rgb_light.png" alt="" id="logo">
        <div class="form-inline">
            <div class="form-group justify-content-center">
                <input type="text" class="form-control" id="query" placeholder="Search">
                <input type="image" class="form-control myButton"
                       src="../resources/youtube_full_color_icon/digital_and_tv/yt_icon_rgb.png" id="iconButton"
                       ng-click="getResults()">
            </div>
        </div>
    </div>

    <div class="text-center" ng-repeat="video in videoList">
        <iframe width="560" height="315" ng-src="{{getVidUrl(video.id.videoId)}}" frameborder="0"
                allow="autoplay; encrypted-media" allowfullscreen></iframe>
    </div>
</div>

<!-- Bootstrap and jQuery-->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
        integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
        crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
        crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"
        integrity="sha384-smHYKdLADwkXOn1EmN1qk/HfnUcbVRZyYmZ4qpPea6sjB/pTJ0euyQp0Mk8ck+5T"
        crossorigin="anonymous"></script>
<!-- AngularJS -->
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.min.js"></script>
<script src="../config/tokens.js"></script>
<script src="script.js"></script>
</body>
```

style.css is simple, showing styles and placement for the logo and submit button.

```css
* {
    font-family: 'Roboto', sans-serif;
}
body {
    margin-top: 100px;
}
.form-group {
    margin-top: 50px;
}
.myButton {
    padding: 0;
    background-color: transparent;
    border: none;
}
#logo {
    width: 60%;
    height: auto;
}
```

script.js contains several functions.

```js
angular.module('youTubeApp', [])
    .config(function($sceDelegateProvider) {
        $sceDelegateProvider.resourceUrlWhitelist([
            'self',
            'https://www.youtube.com/**'
        ])
    })
    .controller('AppCtrl', function ($scope, $http) {
        $scope.videoList = new Array();

        $scope.getResults = function () {
            var query = document.getElementById("query").value;
            var key = "AIzaSyC-OZ3iu8r3p6SfBwr3H120QUXxR0MXarQ";
            var baseUrl = "https://www.googleapis.com/youtube/v3/search";
            if (query != null && query != "") {
                var url = baseUrl +
                    "?key=" + key +
                    "&part=snippet" +
                    "&type=video" +
                    "&videoEmbeddable=true" +
                    "&q=" + query;

                $http.get(url)
                    .success(function (response) {
                        $scope.videoList = response.items;
                    });
            } else {
                alert("Please enter a search query.");
            }
        };

        $scope.getVidUrl = function (id) {
            return "https://www.youtube.com/embed/" + id;
        }
    });
```

First, an angular module is defined called ‘youTubeApp. This is used in the html file. The module is configured using $sceDelegateProvider. This module lists sites that are deemed ‘safe’ to load content from. Since our site is loading content from an external site, youtube.com, we must set this as part of the resource whitelist.

The controller does the work of the application. First, an array is declared for the videoList. The getResults() function is called when the user presses the submit button. It gets the query value that the user entered into the search box and uses that to set the url, which includes the base url for the youtube api and an access key. Angular’s $http module is used to build and send the HTTP request to the supplied url. On a successful response event, the videoList is set to the items object in the response, which contains the videos and their ids. The getVidUrl(id) function is called in the html when the videos are being displayed. It creates the embed link to the video to show on the page.

Configuration

1. IDE: WebStorm version 2018.1.4.
2. HTML 5
3. CSS 3
4. JavaScript
5. AngularJS 1.6.1
6. Bootstrap 3.3.7
7. YouTube API

Limitations

The app currently only returns 5 videos. The specification did not say what to return other than videos, but the user might possibly want to view playlists and channels. These are also possible using the API.

References

1. Class notes and use cases
2. AngularJS-sceDelegateProvider: https://docs.angularjs.org/api/ng/provider/$sceDelegateProvider
3. AngularJS-http: https://docs.angularjs.org/api/ng/service/$http
4. Bootstrap: http://getbootstrap.com
5. YouTube API: https://developers.google.com/youtube/v3/getting-started

Task 2 - Develop a Tic Tac Toe game with AngularJS

Objective

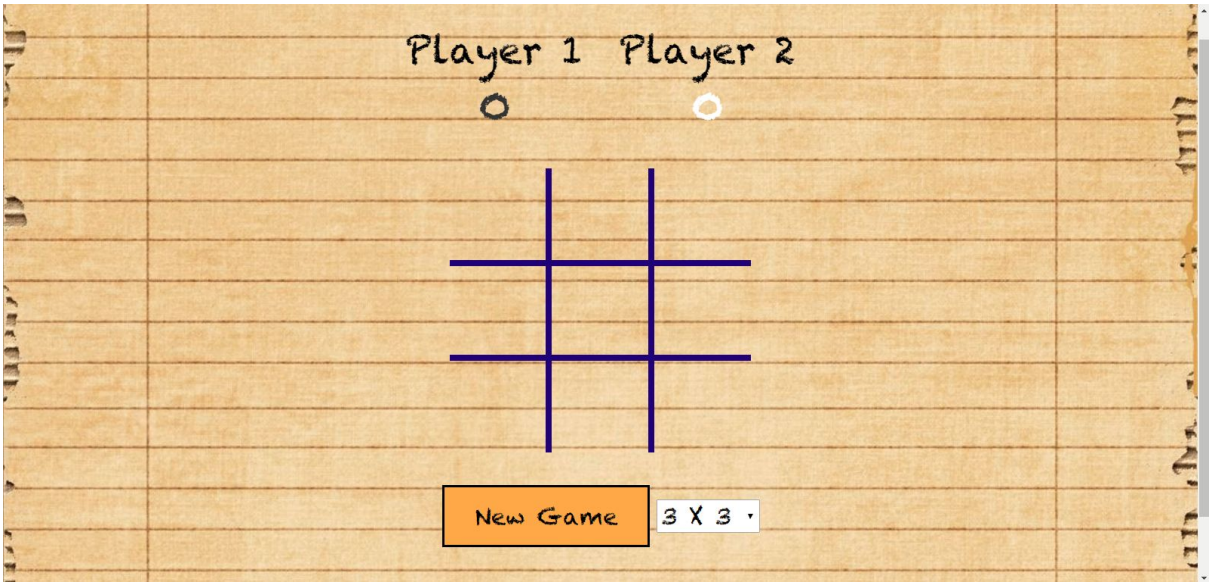
Objective of Task 2 was to create a Tic Tac Toe game using AngularJS. Game board should be minimum 3x3 and have at least two players. Application will count points of each player in order to determine who is winning. recursive method needs to be used to fill blocks until one of the players wins.

Features

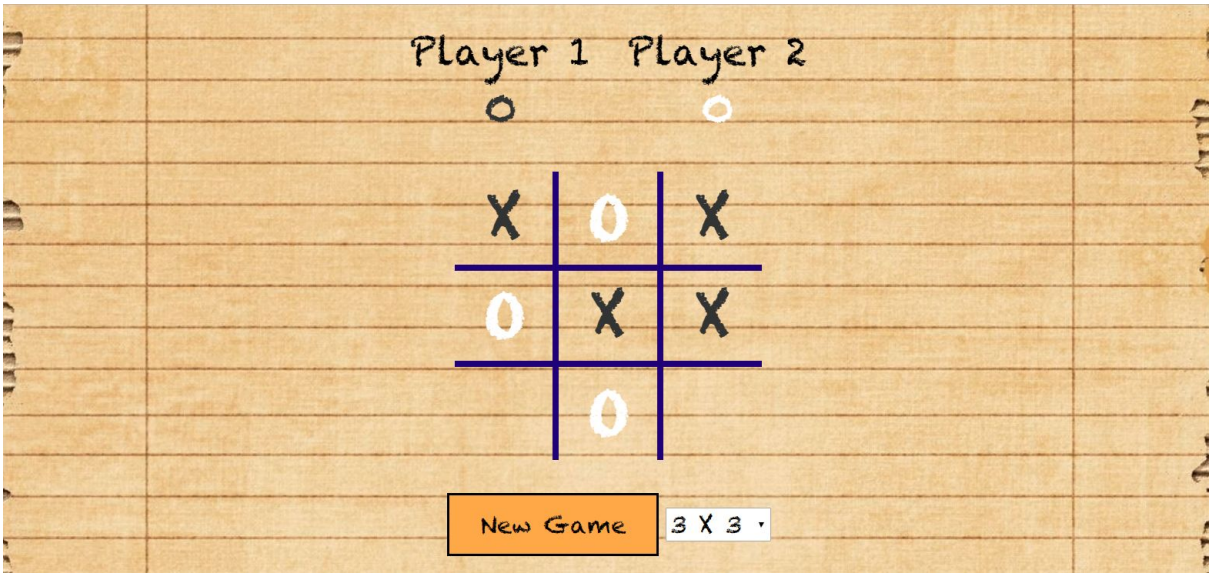
Tic Tac Toe game is traditional game with 3x3 board and two players, however player can change dimensions of the board by picking the size from drop down list. Available dimension are: 3x3, 4x4 and 5x5. Application will count number of wins for each player. Users can start new game by pressing “New Game” button.

Steps

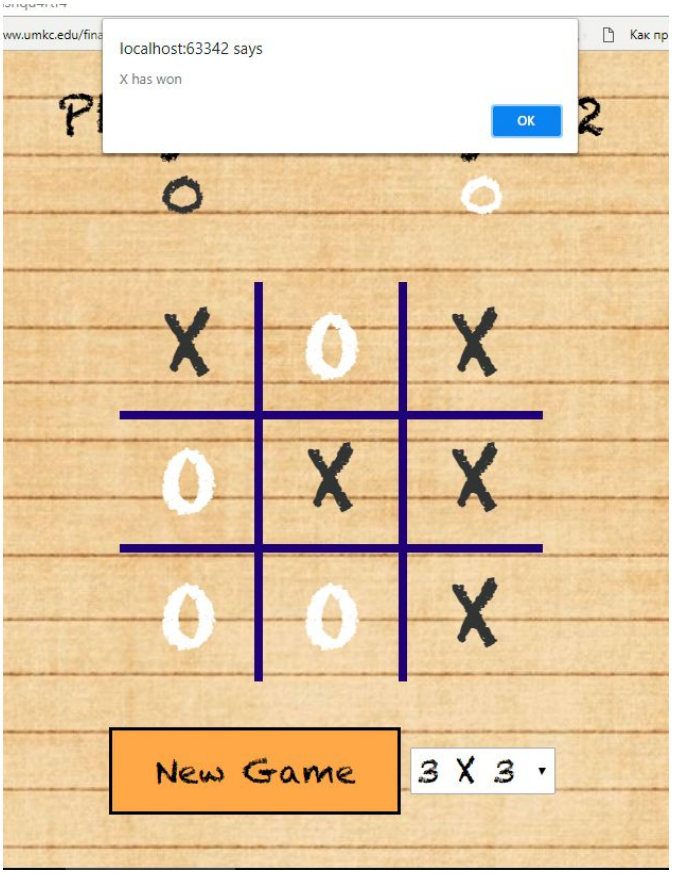
Game begins from picking dimension of the board from a drop down list, which has: 3x3, 4x4 and 5x5 size boards, default dimension is 3x3 board. Player 1 uses “X” symbol and Player 2 uses “O” symbol to play. Player 1 will start first:



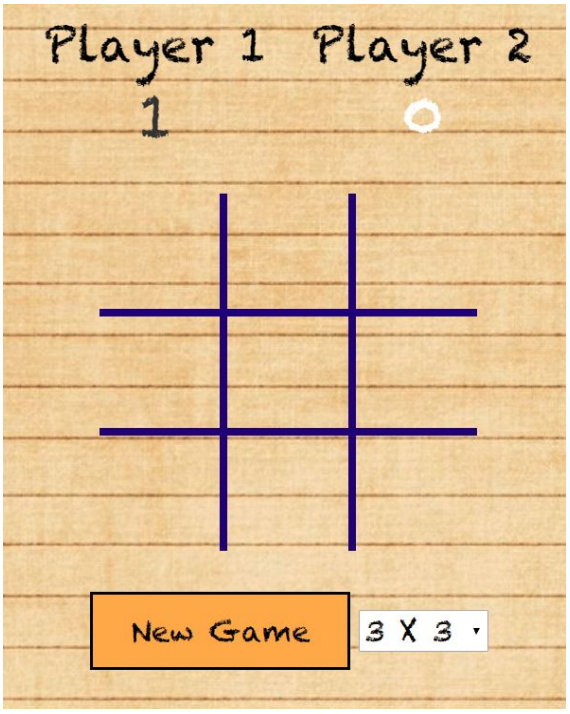
Game continues until board is filled out or one of the players wins:



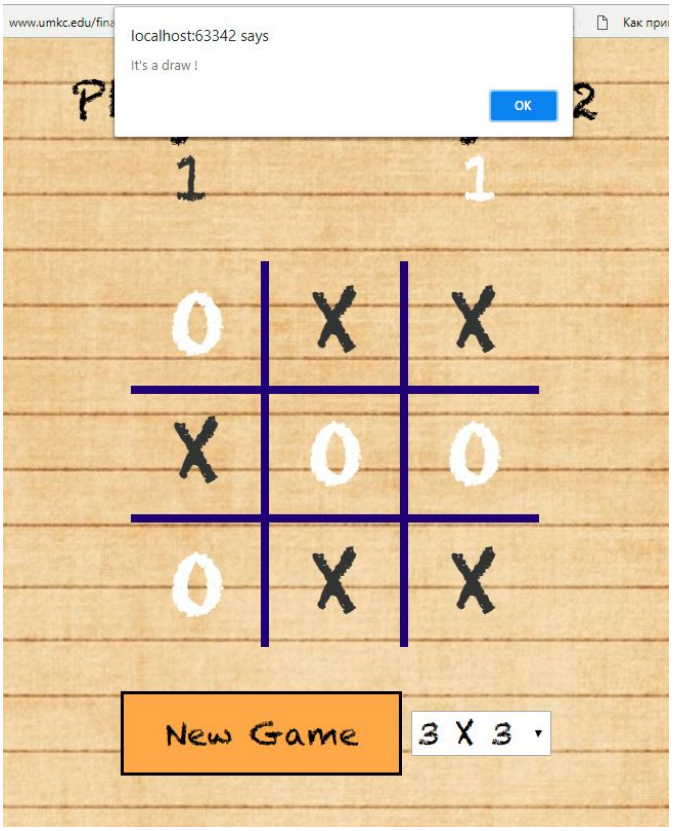
If one of the players wins - alert will pop up saying that Player 1 or 2 won the game:



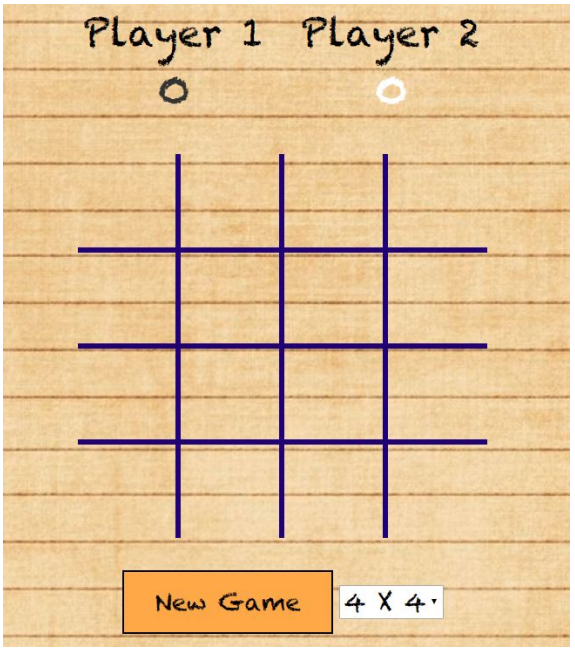
After clicking “OK” button in alert new game will start and score will be incremented:

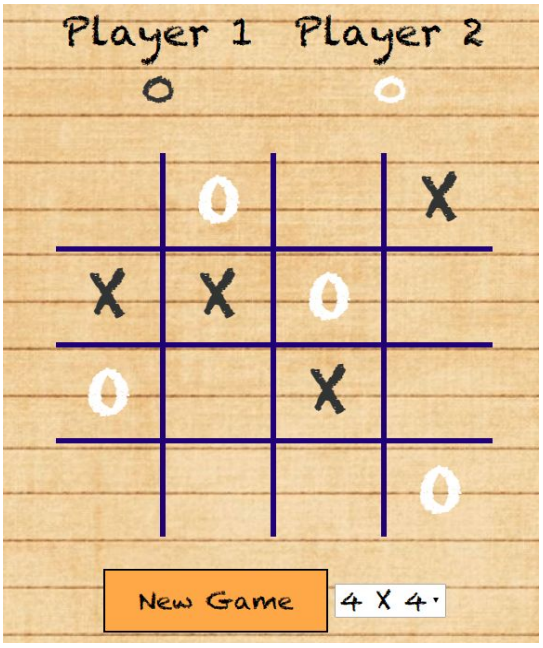


In following example we can see that the was a draw and alert will pop up as well:

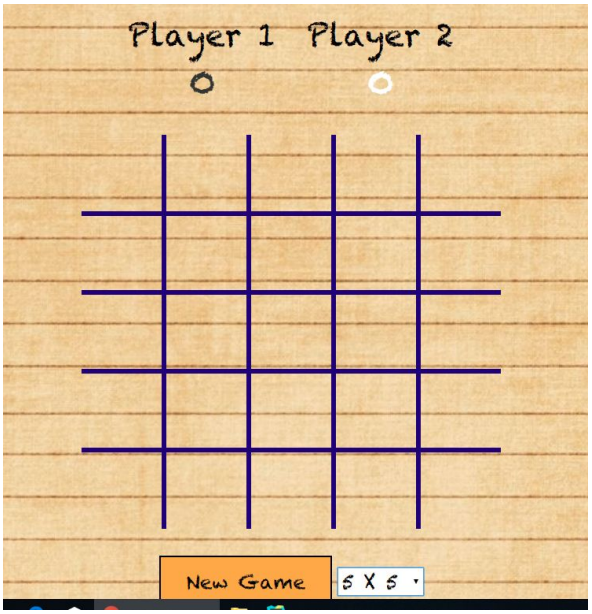


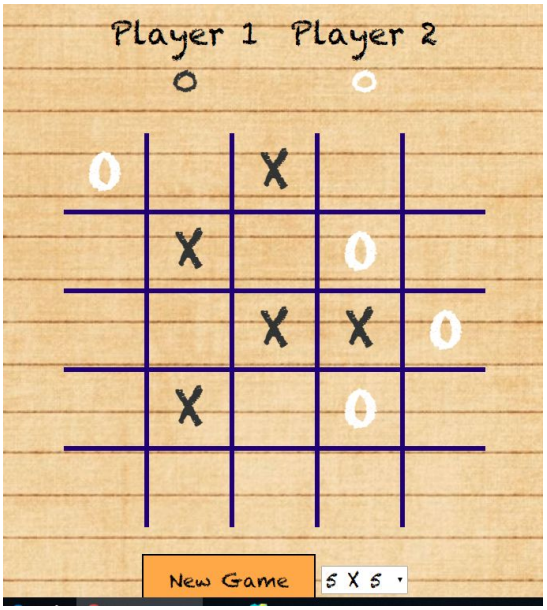
Example of 4x4 board:





Example of 5x5 board:





Code behind

WebStorm IDE was used to implement Tic Tac Toe Game using Angular JS. It has .html, .css and .js classes. Image called “Notebook.jpeg” was used for background image of the application.

Index.html

This class defines user interface of the application. It contains table with id = “board” which uses imported font style: Chalkduster. Using *ng-repeat* to create a table of given size. Also defining button “New Game” and selector for board dimension:

```
<!DOCTYPE>
<html>
<head>
    <meta charset="utf8" />
    <title> Tic-Tac-Toe </title>
    <link rel="stylesheet" type="text/css" href="app.css" />
</head>
<body>
<div ng-app="app" ng-controller="TicTacToeCtrl">

    <table id="board" align="center">
        <tr>
            <td style="font-family: Chalkduster; font-size: 40px"> Player 1</td>
            <td width="30">  </td>
            <td style="font-family: Chalkduster; font-size: 40px"> Player 2 </td>
        </tr>
        <tr>
            <td class="score" id="player1">
                {{game.scores.X}}
            </td>
            <td>  </td>
            <td class="score" id="player2">
                {{game.scores.O}}
            </td>
        </tr>
    </table>


    <table align="center" id="tictactoe">
        <tr ng-repeat="(row_index, row) in game.newArray track by $index">
            <td ng-repeat="(column_index, column) in game.newArray track by $index"
                ng-click="game.mark(row_index, column_index)"
                class="{{game.data[row_index+''+column_index]}}">
                {{game.data[row_index+''+column_index]}}
            </td>
        </tr>
    </table>

    <button class="button button1" style="font-size: 25px;font-family: Chalkduster" ng-click="game.init()">New Game</button>

    <select ng-model="game.grid_size"
            ng-options="option.value as option.label for option in grid_options"
            ng-change="game.init()" style="font-family: Chalkduster; font-size: 25px" >
    </select>

</div>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.min.js"></script>
<script src="app.js"></script>
</body>
</html>
```

app.js

Logic and functionality of the application is defined in this class which uses AngularJS. It is called when game is started or new game button is clicked. Defining scores and marks (“X” and “O”) of players in following code:

```
var app = angular.module('app', []);

app.factory('$tictactoe', ['$timeout', function($timeout) {
    return function(grid_size) {

        this.grid_size = grid_size;

        this.init = function() {
            this.scores = {
                X: 0,
                O: 0
            };
            this.marks = {
                X: 'X',
                O: 'O',
                count: 0
            };
            this.newArray = new Array(this.grid_size);
            this.data = {};
        };

        this.empty = function() {
            this.data = {};
            this.marks.count = 0;
        };
```

this.empty = function() is used to restart the game and clean the board.

this.mark = function(row_index, column_index) function is responsible for login of the Tic Tac Toe game, increments score if one of the players won and displays alert or just diplayes alert of it was draw.

```
    this.mark = function(row_index, column_index) {

        var self = this;
        if(self.data[row_index + '' + column_index]) {
            return;
        }
        self.marks.count++;
        var current_mark = self.marks.count % 2 === 1 ? self.marks.X : self.marks.O;
        self.data[row_index + '' + column_index] = current_mark;
        $timeout(function(){
            if (self.Win(current_mark)) {
                alert(current_mark+" has won");
                self.scores[self.marks.count % 2 === 1 ? 'X' : 'O']++;
                self.empty();
            } else if (self.marks.count == self.grid_size * self.grid_size) {
                alert("It's a draw !");
                self.empty();
            }
        });
    };
```

this.win = function(mark) is called when one of the players wins the game:

```
    this.Win = function(mark) {

        var CountVertical = 0,
            CountHorizontal = 0,
            CountLeftRight = 0,
            CountRightLeft = 0;
        for (var i = 0; i < this.grid_size; i++) {
            CountVertical = 0;
            CountHorizontal = 0;
            for (var j = 0; j < this.grid_size; j++) {
                if (this.data[i + '' + j] == mark) {
                    CountHorizontal++;
                }
                if (this.data[j + '' + i] == mark) {
                    CountVertical++;
                }
            }
            if (this.data[i + '' + i] == mark) {
                CountLeftRight++;
            }
            if (this.data[(this.grid_size - 1 - i) + '' + i] == mark) {
                CountRightLeft++;
            }
            if (CountHorizontal == this.grid_size || CountVertical == this.grid_size) {
                return true;
            }
        }
        if(CountLeftRight == this.grid_size || CountRightLeft == this.grid_size) {
            return true;
        }
        return false;
    };

    this.init();

    return this;

};
```

app.controller is used to define size of the board, when user select dimensions of the board from drop down list - table will be resized accordingly:

```
app.controller('TicTacToeCtrl', ['$scope', '$tictactoe', function($scope, $tictactoe) {

    $scope.grid_options = [{
        value: 3,
        label: '3 X 3'
    }, {
        value: 4,
        label: '4 X 4'
    }, {
        value: 5,
        label: '5 X 5'
    }];

    $scope.game = new $tictactoe(3);

}]);
```

.css

This style sheet is responsible for classes, parameters and ids of the user interface which defines margin, size, layouts and fonts of different elements of the application:

```css
@font-face {
    font-family: Chalkduster;
    src: url(Chalkduster.ttf);}

body {
    background-image: url("Notebook.jpeg");
    height: 100%;
    background-position: center;
    background-repeat: no-repeat;
    background-size: cover;
    text-align: center;
    margin: 50px auto;
    font-family: Chalkduster;
    font: normal 20px Chalkduster ms;
}

#tictactoe {
    border-collapse: separate;
    cell-spacing: 0px;
    margin: 30px auto;
    border-spacing: 0px;
}

#tictactoe td {
    border: 3px solid #000080;
    height: 100px;
    width: 100px;
    text-align: center;
    font-family: Chalkduster;
    font: bold 60px Chalkduster ms;
}

#tictactoe tr:first-child td {
    border-top: none
}

#tictactoe tr:last-child td {
    border-bottom: none
}

#tictactoe tr td:first-child {
    border-left: none;
```

.button

```css
#tictactoe td.X {

    font-family: Chalkduster;
    font-size: 60px;
    color: #333;
}

#tictactoe td.O {
    font-family: Chalkduster;
    font-size: 60px;
    color: white;
}

#board .score {
    text-align: center;
    font-size: 40px
}

#board .score#player1 {
    font-family: Chalkduster;
    font-size: 50px;
    color: #333;
}

#board .score#player2 {
    font-family: Chalkduster;
    font-size: 50px;
    color: white;
}
.button {
    background-color: #f0ad4e;
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 13px;
```

```css
    display: inline-block;
    font-size: 13px;
    margin: 4px 2px;
    cursor: pointer;
    -webkit-transition-duration: 0.4s; /* Safari */
    transition-duration: 0.4s;
}

.button1 {
    background-color: #f0ad4e;
    color: black;
    border: 2px solid black;
}
.button1:hover {
    box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24),0 17px 50px 0 rgba(0,0,0,0.19);
    background-color: black;
    color: white;
}
```

Configuration

8. IDE: WebStorm version 2018.1.4.
9. HTML 5

10. CSS 3
11. JavaScript
12. AngularJS 1.6.1

Limitation

Application has all features of traditional Tic Tac Toe game, the only limitation of our application is that there are only 2 player can play the game. Game could have more that 2 players.

References

1. https://www.w3schools.com
2. In class lectures
3. In Class Use cases and ICPs

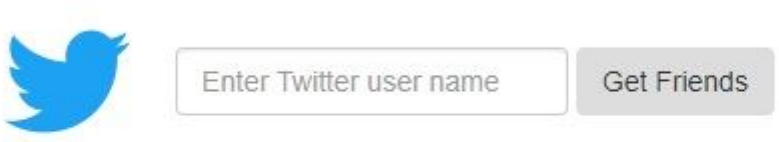
Task 3 - Twitter friends list visualization with D3.js

Objective

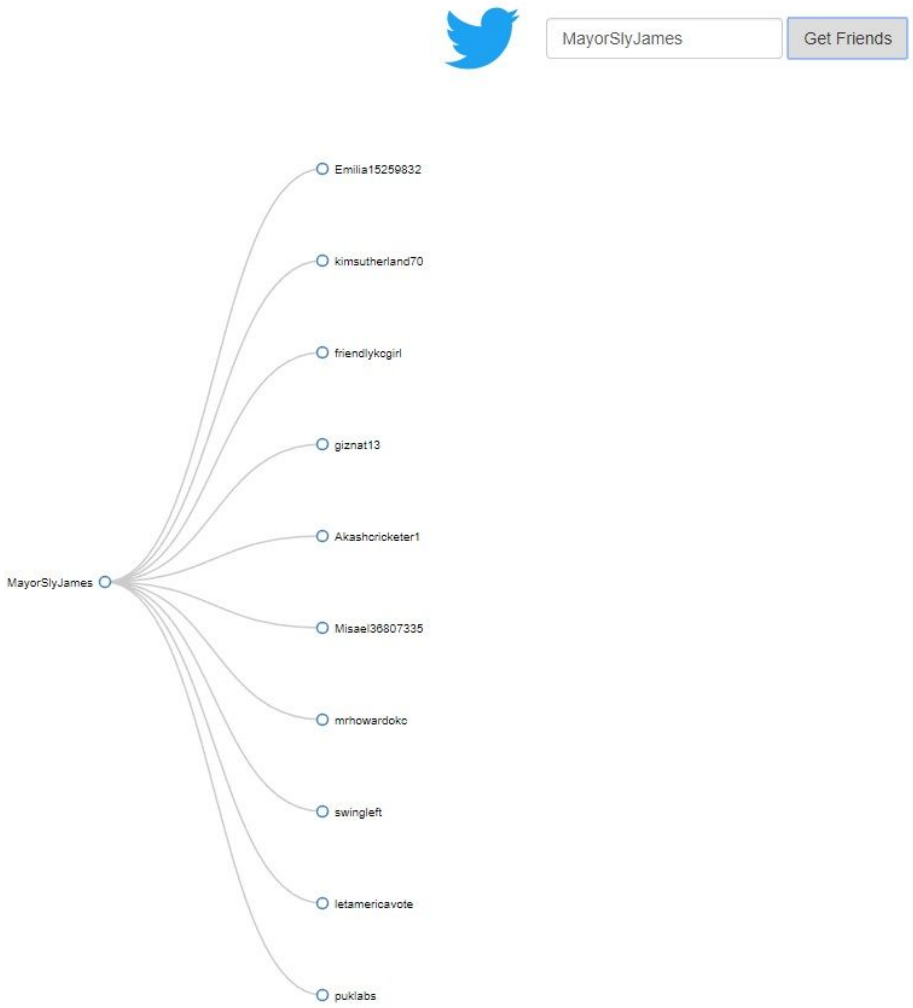
The goal of Task 3 was to use the Twitter API to find a user's friends list, then visualize that list using D3.js, Data Driven Documents. D3 is a visualization tool that uses HTML, CSS, and SVG (scalable vector graphics) to create dynamic graphs and charts of data.

Features and Steps

This app has a simple search bar and submit button.

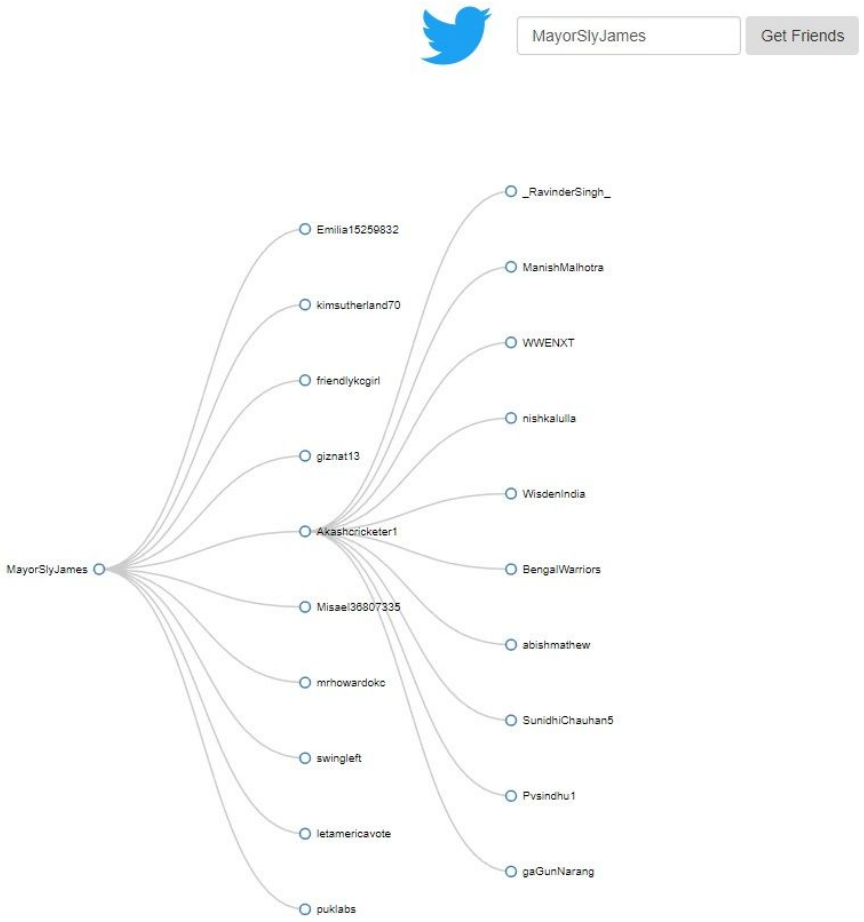


After a username is entered and the "Get Friends" button is pressed, the user and their friends are displayed in a tree.

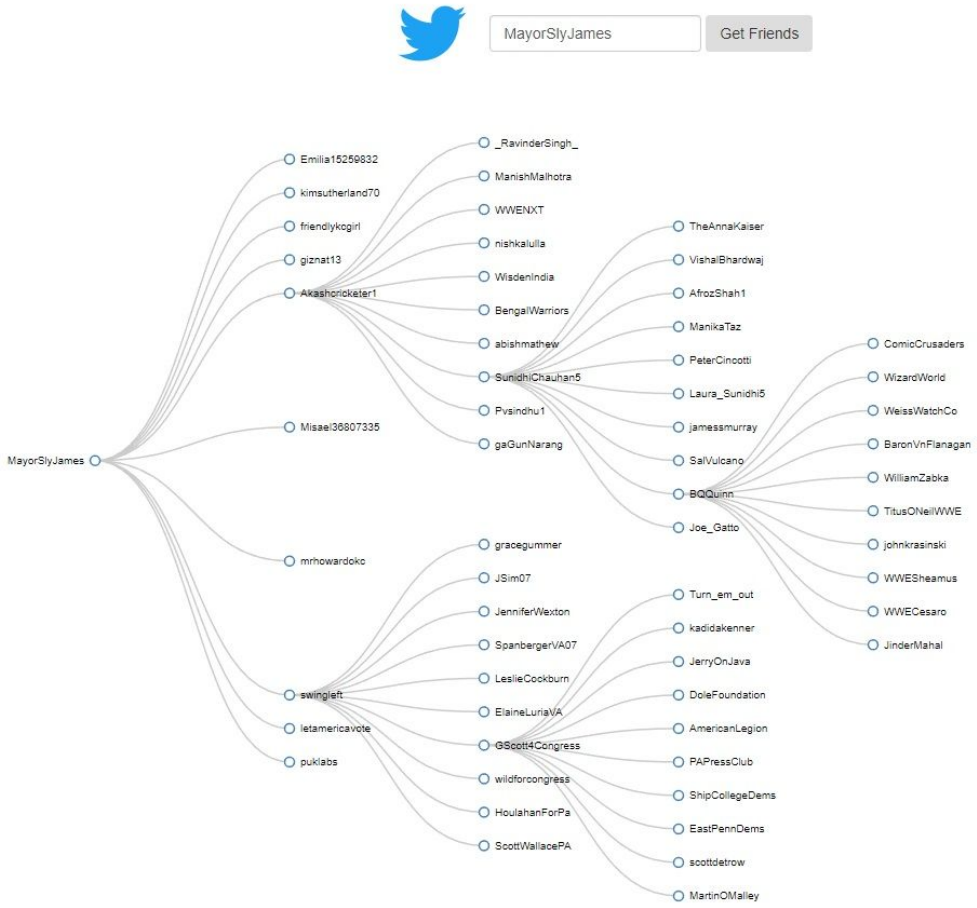


Currently, 10 friends are returned, but this value can be changed. Ten makes the visualization easier to read.

When one of the friends is clicked, another call is made to retrieve their friends, and the tree is updated.



And this process can keep going…



Code

The code for Task 3 was made complicated by the fact that the Twitter API does not implement CORS and thus does not service client-side requests. Only server-side requests are allowed by the API, so we created a proxy server to handle our requests using Node js. The D3.js code is also difficult to understand; we relied heavily on the examples provided in class for a dynamic tree structure.

The index.html file is very simple and consists only of the logo, input form, and button.

```
<html lang="en" ng-app="twitterApp">
<head>
    <meta charset="UTF-8">
    <title>Twitter Tree</title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
          integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
    <link rel="stylesheet" href="style.css">
</head>
<body ng-controller="TwitterCtrl">

<div class="container">

    <form class="form-inline">
        <img src="../resources/Twitter%20Logos/Twitter%20Logos/Twitter_Logo_Blue/Twitter_Logo_Blue.png" id="logo">
        <input type="text" class="form-control" ng-model="screenName" placeholder="Enter Twitter user name">
        <button class="btn btn-light" ng-click="getFriends(screenName)">Get Friends</button>
    </form>

</div>

<!-- Bootstrap and jQuery-->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
        integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
        crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
        crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"
        integrity="sha384-smHYKdLADwkXOn1EmN1qk/HfnUcbVRZyYmZ4qpPea6sjB/pTJ0euyQp0Mk8ck+5T"
        crossorigin="anonymous"></script>
<!-- AngularJS -->
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.2/angular.min.js"></script>
<script src="http://d3js.org/d3.v3.min.js"></script>
<script src="../config/tokens.js"></script>
<script src="twitter.js"></script>
<script src="script.js"></script>
</body>
```

Likewise, the stylesheet contains simple styles for the tree nodes:

```
.container {
    padding-top: 50px;
}

#logo {
    height: 100px;
    width: auto;
}

.node {
    cursor: pointer;
}

.node circle {
    fill: #fff;
    stroke: steelblue;
    stroke-width: 1.5px;
}

.node text {
    font: 10px sans-serif;
}

.link {
    fill: none;
    stroke: #ccc;
    stroke-width: 1.5px;
}
```

There are two javascript files, one for the server and one for the client. The server code uses the Twitter library for Node.js to make the call to the API. A Twitter client is created using credentials that are stored as environment variables for the server. The request is then sent using the library and the response handled and passed back to the client.

```
var Twitter = require('twitter');
var express = require('express');
var cors = require('cors');
var app = express();

// set a new Twitter client using the twitter module.  Credentials are stored as environment variables.
var client = new Twitter({
    consumer_key: process.env.TWITTER_CONSUMER_KEY,
    consumer_secret: process.env.TWITTER_CONSUMER_SECRET,
    bearer_token: process.env.TWITTER_BEARER_TOKEN
});

app.use(cors());

// uses the twitter client to send the request to the twitter API.  The response is sent with the list of friends
// returned
app.get('/getFriends/:screenName', function (req, res) {
    client.get('friends/list', { screen_name: req.params.screenName, count: 10 }, function(error, list, response) {
        if (error) throw error;
        res.send(JSON.stringify(list.users));
    });
});

// creates a node js server
var server = app.listen(8081, function () {
    var host = server.address().address;
    var port = server.address().port;

    console.log("Twitter app listening at http://%s:%s", host, port)
});
```

The client side js is more complex because it contains the D3 code. The AngularJS module is declared and the Twitter API site is whitelisted as in Task 1.

```
var app = angular.module('twitterApp', [])
    .config(function($sceDelegateProvider) {
        // declare twitter api as a safe site to load data from
        $sceDelegateProvider.resourceUrlWhitelist([
            'self',
            'https://api.twitter.com/**'
        ])
    });

    app.run(function ($http) {
        // Add CORS header
        $http.defaults.headers.common['Access-Control-Allow-Origin'] = '*';
    })

    app.controller('TwitterCtrl', function($scope, $http) {
        // Called when the submit button is pressed by the user
        // makes a call to the proxy server to get the friends of the screen name.  The response contains the friend
        // data.  The data are used to create a tree visualization
        $scope.getFriends = function(screenName) {

            var req = $http.get('http://127.0.0.1:8081/getFriends/' + screenName)
                .then(function (data) {
                    // data.data contains an array of friends from the response
                    $scope.friendsList = data.data;
                    // get the data in a tree-like json, with each node having children
                    var tree = getFriendTree(data.data, screenName);
                    // draw the tree
                    buildTree(tree);
                    })
        };

        // Creates a json object for the tree, using the screen name and appending an array of children.
        // The queried user is the root.
        var getFriendTree = function(arr, root) {
            var jsonTree = { "name" : root, "children" : [] };
            angular.forEach(arr, function(friend) {
                jsonTree.children.push({"name": friend.screen_name });
            });
            return jsonTree;
        };
```

A CORS header is added to the $http responses. The getFriends(screenName) function is called when the user presses the submit button. The screen name gets passed to the server that then handles the request. This method also reads the response and sets the data to the friendsList. Finally, the method gets and builds the friend tree.

The getFriendTree(arr, root) function simply takes the friends list data provided by the Twitter response and converts it into a tree object, with a name and children array for each node. The tree is then used to build the graphical tree.

The buildTree(tree) function uses the tree structure we created in ICP6. It largely uses the code provided here: https://bl.ocks.org/mbostock/4339083

```
// Uses D3.js to build a tree from the json object
var buildTree = function(treeJSON) {
    var margin = {top: 20, right: 120, bottom: 20, left: 120},
        width = 960 - margin.right - margin.left,
        height = 800 - margin.top - margin.bottom;

    var i = 0,
        duration = 750,
        root;

    var tree = d3.layout.tree()
        .size([height, width]);

    var diagonal = d3.svg.diagonal()
        .projection(function(d) { return [d.y, d.x]; });

    var svg = d3.select("body").append("svg")
        .attr("width", width + margin.right + margin.left)
        .attr("height", height + margin.top + margin.bottom)
        .append("g")
        .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

    d3.json(treeJSON, function(error) {
        if (error) throw error;

        root = treeJSON;
        root.x0 = height / 2;
        root.y0 = 0;

        function collapse(d) {
            if (d.children) {
                d._children = d.children;
                d._children.forEach(collapse);
                d.children = null;
            }
        }

        root.children.forEach(collapse);
        update(root);
    });

    d3.select(self.frameElement).style("height", "800px");

    function update(source) {

        // Compute the new tree layout.
        var nodes = tree.nodes(root).reverse(),
            links = tree.links(nodes);
```

...continues on github.

When a node is clicked, another call is made to the API to get the friends list for that user.

```
// Get new children on click.
function click(d) {
        var req1 = $http.get('http://127.0.0.1:8081/getFriends/' + d.name)
            .then(function (data) {
                $scope.friendsListNew = data.data;
                d.children = [];
                angular.forEach(data.data, function(friend) {
                    d.children.push({"name": friend.screen_name});
                });
                update(d);
            })
}
```

Limitations

The Twitter API is limited in that it does not allow client-side requests from the browser. We were able to get around this thanks to the Twitter library developed for Node.js. We also feel limited in our knowledge of D3.js, since this was not really discussed in class or our tutorials. We had to rely heavily on existing code for this part. Our tree only shows 10 friends, even though users may have many friends, but it might be difficult to visualize more with a tree. Finally, our tree is not currently collapsible.

Configuration

1. IDE: WebStorm version 2018.1.4.
2. HTML 5
3. CSS 3
4. JavaScript
5. AngularJS 1.6.1
6. Bootstrap 3.3.7
7. Node.js
8. Twitter library for Node.js

References

1. Class notes, use cases, and ICPs
2. AngularJS-sceDelegateProvider: https://docs.angularjs.org/api/ng/provider/$sceDelegateProvider
3. AngularJS-http: https://docs.angularjs.org/api/ng/service/$http
4. Bootstrap: http://getbootstrap.com
5. Twitter API: https://developer.twitter.com/en/docs/basics/getting-started
6. Twitter library for Node.js: https://www.npmjs.com/package/twitter
7. D3.js: https://d3js.org/
8. D3.js collapsible tree: https://bl.ocks.org/mbostock/4339083