ITCS212 Web Programming

Project 2: Movie Search System

Submit to

Dr. Jidapa Kraisangka

Dr. Wudhichart Sawangphol

Presented by

Thanawat Riamliw 6188019

Worameth Siritanakorn 6188061

Tharit Chantanalertvilai 6188068

Thanyanit Jongjitragan 6188075

Mahidol University Salaya

Faculty of Information and Communication Technology

What have we done for Web Accessibility?

We have applied the Bootstrap to our web page, so it will be responsive. Also, we make the text content to be readable and understandable by reducing the opacity of the background, making it easier for users to see content including separating foreground and background.

How is our webpage friendly?

It is easy to use. There is no complex function on our webpage. There is no complex picture to distract us. The UI is not overwhelming.

How can we interact with all web services?

We call omdbapi from client side script and other three 3rd party APIs (Spotify, Youtube, Twitter) from server-side script by following steps.

<u>Calling from client- side script</u>

1. We call our movie information API by using opensource API called OMDb.

```
$(document).ready(function () {
  // do a HTTP get request through omdb data api v3
  $.get(
    "http://www.omdbapi.com/",
    {
      apikey: "9ba1bfdc",
      t: inputQuery,
      r: "json",
    },
```

FYI: http://www.omdbapi.com/

2. Then we have anonymous function for receiving the data and use jQuery to decorate the website.

```
//A callback function for handling the data we get from our get request
function (data) {
  let output = "";
  console.log(data);

  output += '<img class="card-img-top my3" src="' + data.Poster + '">'
            + '<div class="card-body">'
              + '<h4 class="card-title">' + data.Title + '</h4>'
              + '<p class="card-text">'
                + '<b>Released: </b>' + data.Released + '<br>'
                + '<b>Type: </b>' + data.Type + '<br>'
                + '<b>Runtime: </b>' + data.Runtime + '<br>'
                + '<b>Genre: </b>' + data.Genre + '<br>'
                + '<b>Actors: </b>' + data.Actors + '<br>'
                + '<b>Plot: </b>' + data.Plot + '<br>'
                + '<b>Language: </b>' + data.Language + '<br>'
                + '<b>Ratings: </b>' + data.Ratings[0].Value + '<br>'
            + '</div>';

  $("#omdb").append(output);
```

Calling from Server-side script

We call these APIs from server side with a few following steps

1. jQuery call the different route of localhost:3000/<twitter/spotify/youtube> as following

- Twitter

```
function twitterSearch(inputQuery){
  $("#twitter").html("");
   console.log("TwitterSearch is activated")
   $.get("http://localhost:3000/twitter", {
    q: inputQuery,
```

- Spotify

```
function spotifySearch(inputQuery){
  $("#twitter").html("");
   console.log("SpotifySearch is activated")
   $.get("http://localhost:3000/spotify", {
    q: inputQuery,
```

- Youtube

```
function youtubeSearch(inputQuery)
{
  // clear youtube output
  $('#youtube').html('');

  // q parameter
  input = inputQuery + " movie trailer";
  console.log(input)
  // insert parameters and get data
  $.get(
      "http://localhost:3000/youtube", {
          q: input,
      },
```

2. We have each route handled by using Node.js + Express.js

Call Twitter API by using external Twit package. We search/tweets q is the keyword that user type, count is how many tweet we have, and we want recent tweets.

```
const Twit = require("twit");

app.get("/twitter", (req, res) => {
  const keyword = req.query.q;
  console.log("twitter keyword " + keyword);
  const T = new Twit(twitterConfig);
  T.get(
    "search/tweets",
    {
      q: keyword,
      count: 3,
      result_type: "recent",
    },
    (error, data, response) => {
      console.log(data);
      res.send(data);
    }
  );
});
```

Calling Spotify API using external node package called node-spotify-api. In spotify.search we want playlist according to songKeyword and we want 2 search results.

```
const Spotify = require("node-spotify-api");

app.get("/spotify", (req, res) => {
  const keyword = req.query.q;
  const songKeyword = keyword + "soundtrack";
  console.log("spotify keyword " + keyword);

  const spotify = new Spotify({
    id: process.env.SPOTIFY_CLIENT_ID,
    secret: process.env.SPOTIFY_CLIENT_SECRET,
  });

  spotify.search({ type: 'playlist', query: songKeyword, limit:2 }, function(err, data) {
    if (err) {
      return console.log('Error occurred: ' + err);
    }

    res.send(data)
  });
});
```

Calling Youtube API use external node package called axios. We want video snippet from our keyword and we want only 1 result.

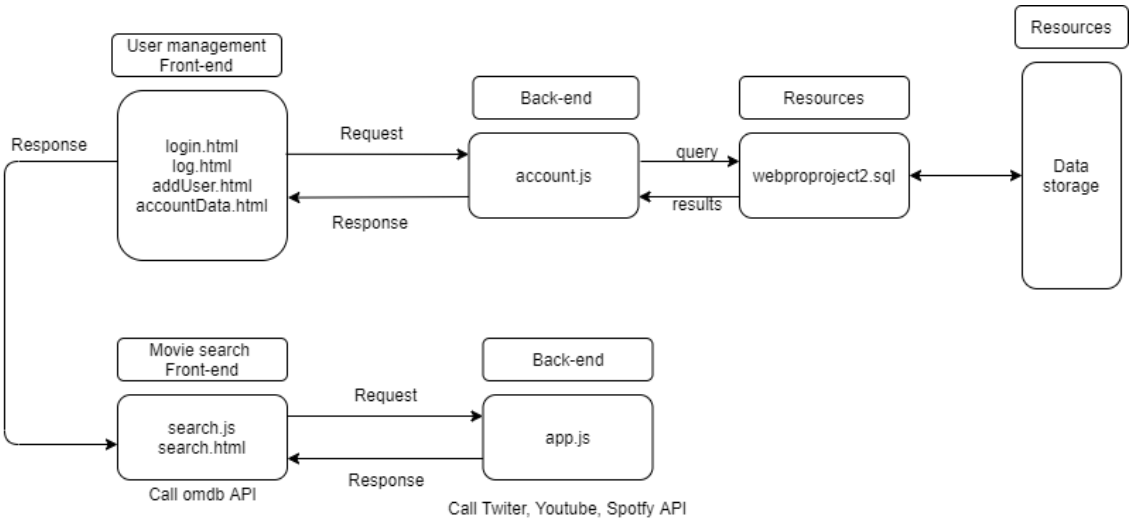
```
const axios = require("axios");

  const callYoutube = async () => {
    try {
      return await axios.get(youtubeURL, {
        params: {
          part: "snippet",
          type: "video",
          // OUR API_KEY
          key: "AIzaSyBTOAXguG1MOrbcuu5v-lZ6WNObJZuxfSY",
          // our search query
          q: keyword,
          maxResults: 1,
        },
      });
    } catch (error) {
      console.error(error);
    }
  };

  callYoutube().then(result => res.send(result.data.items[0]))
});
```

3. The data return from these handlers from second step will return to each jQuery call on the callback anonymous function “function(data)” or “function(result)”.
4. We use jQuery to manipulate these data and make beautiful website

For instance for both 3. and 4. in this Youtube API calling

```
$.get(
    "http://localhost:3000/youtube", {
      q: input,
    },
    function(result) {
        let item = result;
          // get output
          var output = '<img class="card-img-top my3" src="' + item.snippet.thumbnails.high.url + '">'
                + '<div class="card-body">'
                  + '<h4 class="card-title">' + item.snippet.title + '</h4>'
                  + '<p class="card-text"> URL: <a href="https://www.youtube.com/embed/' + item.id.videoId + '">https://www.youtube.com/embed/' + item.id.
                  videoId + '</a></p><br>';
                + '</div>';
        // display results
        $('#youtube').append(output);
    }
)
```

The architecture of our system



<u>Relationship between app.js and search.js with search.html in a few important steps</u>

1. Users connect to search.html and then search for information related to their beloved movies.
2. The user's input will be keyword for omdb API to be called and also act as request sending to app.js
3. App.js call Youtube, Sportify and Twitter API and then send data back to search.js
4. Search.js manage all those data and then use jquery to manage those data and turn it into glamorous html element.

<u>Relationship between user management and account.js (frontend - backend)</u>

1. Users will login in log.html if User login success, user will enter to search.html and if Admin login in administrator.html success, it will bring you to AdministatorsTools.html.
2. In administatorsTools.html, you can add, edit ,list and delete all accounts.
3. account.js will call data from webproproject2.sql to show in every page that use account data.

<u>Relationship between account.js and resources sql file</u>

1. Account.js will query webproproject2.sql for data table.
2. webproproject2 will send result to account.js for display in all file html.

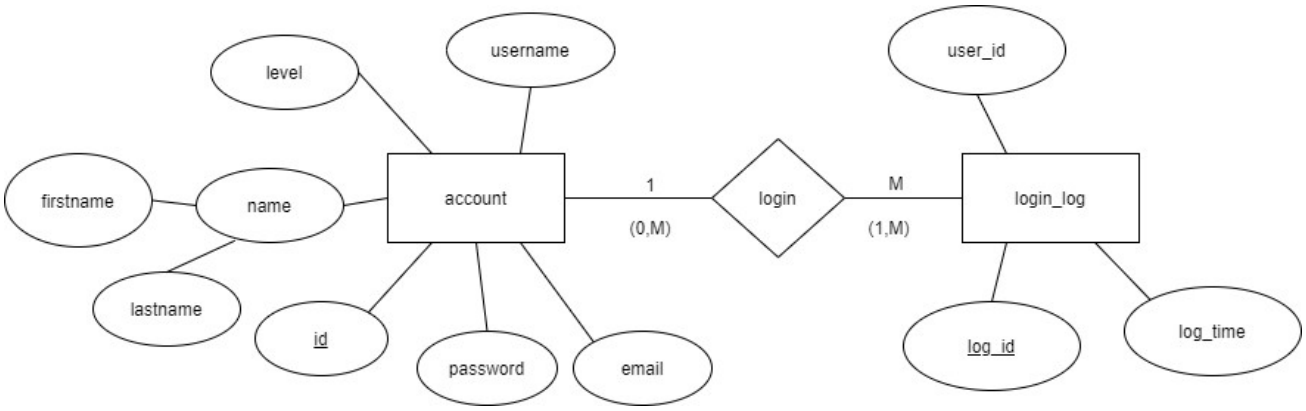
<u>Relationship between user management system and movie search</u>

1. Start from login button to connect from login page to movie search system.

<u>Relationship between sql file and data storage</u>

1. Connect by sql query INSERT , SELECT, UPDATE, DELETE

ER Diagram



The structure of our code

For linking between search.html and search.js is method search() on button tag in search.html and function declaration in search.js

```
<input class="form-control mr-sm-2" type="text" placeholder="Search" id="query">
<button class="btn blue-gradient" type="button" style="border-radius: 30px;" onClick="search()">Search</button>
```

```
function search() {
  // Get value from our input tag
  let inputQuery = document.getElementById("query").value;
  console.log(inputQuery);
  omdbSearch(inputQuery);
  youtubeSearch(inputQuery);
  spotifySearch(inputQuery);
  twitterSearch(inputQuery);
}
```

For calling web services we build the relationship between frontend and backend $.get(localhost:(PORT)/(yourAPI)) and app.get('/(yourAPI) ')

for example $.get(localhost:3000/spotify) and app.get('/spotify')

```
function spotifySearch(inputQuery){
  $("#spotify").html("");
   console.log("Spotify Search is activated")
   $.get("http://localhost:3000/spotify", {
    q: inputQuery,
```

```
app.get("/spotify", (req, res) => {
  const keyword = req.query.q;
  const songKeyword = keyword + " soundtrack";
  console.log("spotify keyword " + keyword);
```

For more detailed interaction, the interactions between are in topic of "How can we interact with web service "

Management System

```
// Connect to MySQL
var dbConn = mysql.createConnection({
    host: process.env.MYSQL_HOST,
    user: process.env.MYSQL_USERNAME,
    password: process.env.MYSQL_PASSWORD,
    database: process.env.MYSQL_DATABASE
});
dbConn.connect();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/* PAGE ROUTE */

// DEFAULT ROUTE
// const defaultPage = "adminTool.html";
const defaultPage = "administrators.html"
router.get('/', function (req, res) {
    res.sendFile(path.join(__dirname + '/' + defaultPage));
});
router.get('/' + defaultPage, function (req, res) {
    res.sendFile(path.join(__dirname + '/' + defaultPage));
});

// OTHER PAGE ROUTES

router.get('/addUser.html', function (req, res) {
    res.sendFile(path.join(__dirname + "/addUser.html"));
});

router.get('/accountData.html', function (req, res) {
    res.sendFile(path.join(__dirname + "/accountData.html"));
});

router.get('/administrators.html', function (req, res) {
    res.sendFile(path.join(__dirname + "/administatrors.html"));
});
```

Connect to MySQL for database by using host user password and database name in env file.

```
app.js > router.get() callback

// DEFAULT ROUTE
// const defaultPage = "adminTool.html";
const defaultPage = "administrators.html"
router.get('/', function (req, res) {
    res.sendFile(path.join(__dirname + '/' + defaultPage));
});
router.get('/' + defaultPage, function (req, res) {
    res.sendFile(path.join(__dirname + '/' + defaultPage));
});

// OTHER PAGE ROUTES

router.get('/addUser.html', function (req, res) {
    res.sendFile(path.join(__dirname + "/addUser.html"));
});

router.get('/accountData.html', function (req, res) {
    res.sendFile(path.join(__dirname + "/accountData.html"));
});

router.get('/administrators.html', function (req, res) {
    res.sendFile(path.join(__dirname + "/administatrors.html"));
});

router.get('/administatorsTool.html', function (req, res) {
    res.sendFile(path.join(__dirname + "/administatorsTool.html"));
});

router.get('/APIProject.html', function (req, res) {
    res.sendFile(path.join(__dirname + "/APIProject.html"));
});

router.get('/log.html', function (req, res) {
    res.sendFile(path.join(__dirname + "/log.html"));
});
```

This code for getting database to display in any file html. A route method is derived from one of the HTTP methods, and is attached to an instance of the express class. The following code is an example of routes that are defined for the GET and the POST methods to the root of the app.

```
addUser.html   administatorsTool.html   style.css   administrators.html   log.html   login.html   account.js
account.js > ...

// LOGIN
// to search.html
// require account level <= 2 (be a user or admin)
router.get('/login/login/:uname/:pw', function (req, res) {
    let uname = req.params.uname;
    let pw = req.params.pw;
    let sql = "SELECT * FROM account WHERE username = ? AND password = ?";

    // Missing username or password
    if (!uname || !pw) {
        res.statusMessage = 'Missing username or password.';
        return res.status(400).send();
    }
    dbConn.query(sql, [uname, pw], function (error, results) {
        // Fail to retrieve data, probaby due to the sql error
        if (error) {
            return res.send({
                error: true, message: error.sqlMessage
            });
        }

        let tmp;
        if (results.length == 0) {
            res.statusMessage = "Username or password is incorrect.";
            return res.status(401).send();
        }
        else if (results[0].level == '1' || results[0].level == '2') {
            tmp = results[0].id;
        }
        else {
            res.statusMessage = "This account doesn't have the right to access this site.";
            return res.status(402).send();
        }
        dbConn.query("INSERT INTO login_log (`log_id`, `log_time`, `user_id`) VALUES (NULL, CURRENT_TIMESTAMP, ?)",
        tmp, function (error, results) {
            // console.log("insert new log" + tmprs);
            res.redirect("/administatorsTool.html");
```

When you input correct username and password for admin, it will take you to administatorTools.html

```
// UPDATE
// Edit existing account BY ID
router.put('/accountData/update/', function (req, res) {
    let id = req.body.user.id;
    let email = req.body.user.email;
    let uname = req.body.user.username;
    let fname = req.body.user.firstname;
    let lname = req.body.user.lastname;
    let user = req.body.user;
    if (!id) {
        return res.status(400).send({ error: true, message: 'Don\'t know who to update.'});
    }
    else if (!user || !uname || !email || !fname || !lname /*|| !lv*/) {
        return res.status(400).send({ error: true, message: 'Missing data. Cannot update an account.' });
    }
    else
        dbConn.query("UPDATE account SET ? WHERE id = ?", [user,id], function (error, results) {
            if (error) {
                return res.send({
                    error: true, message: error.sqlMessage
                });
            }
            return res.send({
                error: false, data: results.affectedRows, message: 'Successfully updated account ID: '+id
            })
        });
```

This code for updating account data.

```
                    error: true, message: error.sqlMessage
                });
            }
            return res.send({
                error: false, data: results.affectedRows, message: 'Successfully updated account ID: '+id
            })
        });
});

// DELETE
// Delete an account BY EMAIL
router.delete('/accountData/delete', function (req, res) {
    let id = req.body.user.id;
    if (!id) {
        return res.status(400).send({ error: true, message: 'Don\'t know who to delete.' });
    }
    else
        dbConn.query('DELETE FROM account WHERE id = ?',id , function (error, results) {
            if (error) {
                return res.send({
                    error: true, message: error.sqlMessage
                });
            }
            return res.send({
                error: false, data: results.affectedRows, message: 'Successfully delete account: '+id
            });
        });
});

// SELECT using input string
// get all accounts that colply to the search filter
// Empty string input means SELECT ALL
router.get('/accountData/selectuser/', function (req, res) {
    let sql = "SELECT * FROM account";
    dbConn.query(sql, function (error, results) {
```

Delete an account BY EMAIL input email and it will delete the account data that match to email.

```
account.js > router.get('/login/login/:uname/:pw') callback > dbConn.query() callback

////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
/* LOGIN LOG */

// SELECT
// get all logs that comply with the search filter
// If input is not empty, call this
router.get('/log/getlog/:lid/:ft/:tt/:id', function (req, res) {
    let lid = valid(req.params.lid);
    let ft = valid(req.params.ft);
    let tt = valid(req.params.tt);
    let id = valid(req.params.id);

    let sql = "SELECT * FROM `login_log` JOIN `account` ON account.id = login_log.user_id WHERE (id >= 0)";
    if (lid != '') sql += "AND (log_id = " + lid + ") ";
    if (id != '') sql += "AND (id = " + id + ") ";
    if (ft != '' && ft != '%') sql += "AND (log_time >= '" + ft + "') ";
    if (tt != '' && tt != '%') sql += "AND (log_time <= '" + tt + "') ";
    dbConn.query(sql, function (error, results) {
        if (error) {
            return res.send({
                error: true, message: error.sqlMessage
            });
        }
        //console.log(results);
        return res.send({
            error: false, data: results, message: 'Successfully retrieved data from DB', sql: sql
        });
    });
});
```

..

In Login Log, it will select account data that login to our web site by checking data that input filter for display data that match to data in filter.

```
// SELECT using input string
// get all accounts that colply to the search filter
// Empty string input means SELECT ALL
router.get('/accountData/selectuser/', function (req, res) {
    let sql = "SELECT * FROM account";
    dbConn.query(sql, function (error, results) {
        if (error) {
            return res.send({
                error: true, message: error.sqlMessage
            });
        }
        //console.log(results);
        return res.send({
            error: false, data: results, message: 'Successfully retrieved data from DB'
        });
    });
});

// If input is % (aka %25), change it to ''
function valid(query){
    if (typeof query !== 'undefined' && query){
        if(query == '%')
            return '';
        return query;
    }
    else
        return '';
}
router.get('/accountData/selectuser/:id/:email/:fname/:lname/:uname/:lv1/:lv2', function (req, res) {
    let id = valid(req.params.id);
    let email = valid(req.params.email);
    let uname = valid(req.params.uname);
    let fname = valid(req.params.fname);
    let lname = valid(req.params.lname);
    let lv1 = valid(req.params.lv1);
    let lv2 = valid(req.params.lv2);
```

SELECT using input string and get all accounts that comply to the search filter
Empty string input means SELECT ALL checkboxes.