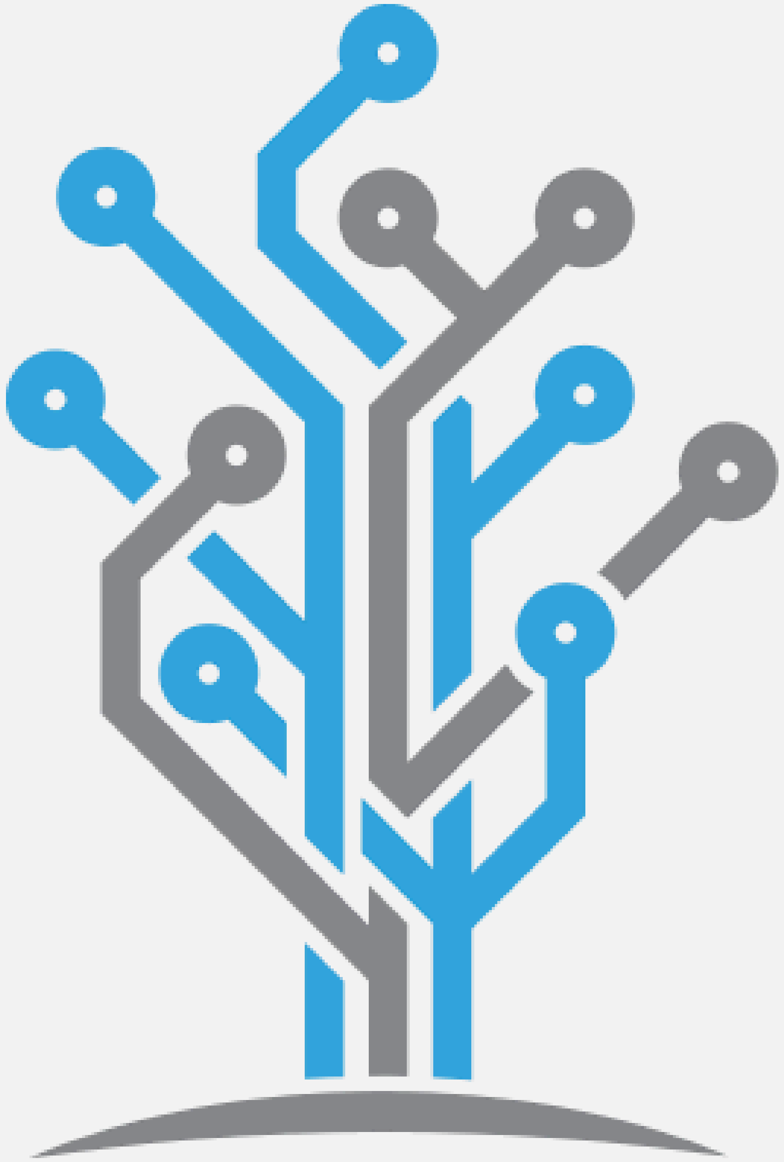
NodeJS
środowisko i technologia ServerSide

PAWEŁ ŁUKASZUK

1064



Callback

A Callback is simply a function passed as an argument to another function which will then use it (call it back)

Callback function allows other code to run in the meantime.

Callback example

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

var myCallbackFunction = function (err, data) {
   console.log(data.toString());
}

fs.readFile("input.txt", myCallbackFunction);

console.log("Program Ended");
```

Nested callback

```
const makeBurger = () => {
    getBeef(function (beef) {
        cookBeef(beef, function (cookedBeef) {
            getBuns(function (buns) {
                // Put patty in bun
            })
        })
    })
}
```

Nested callback - example

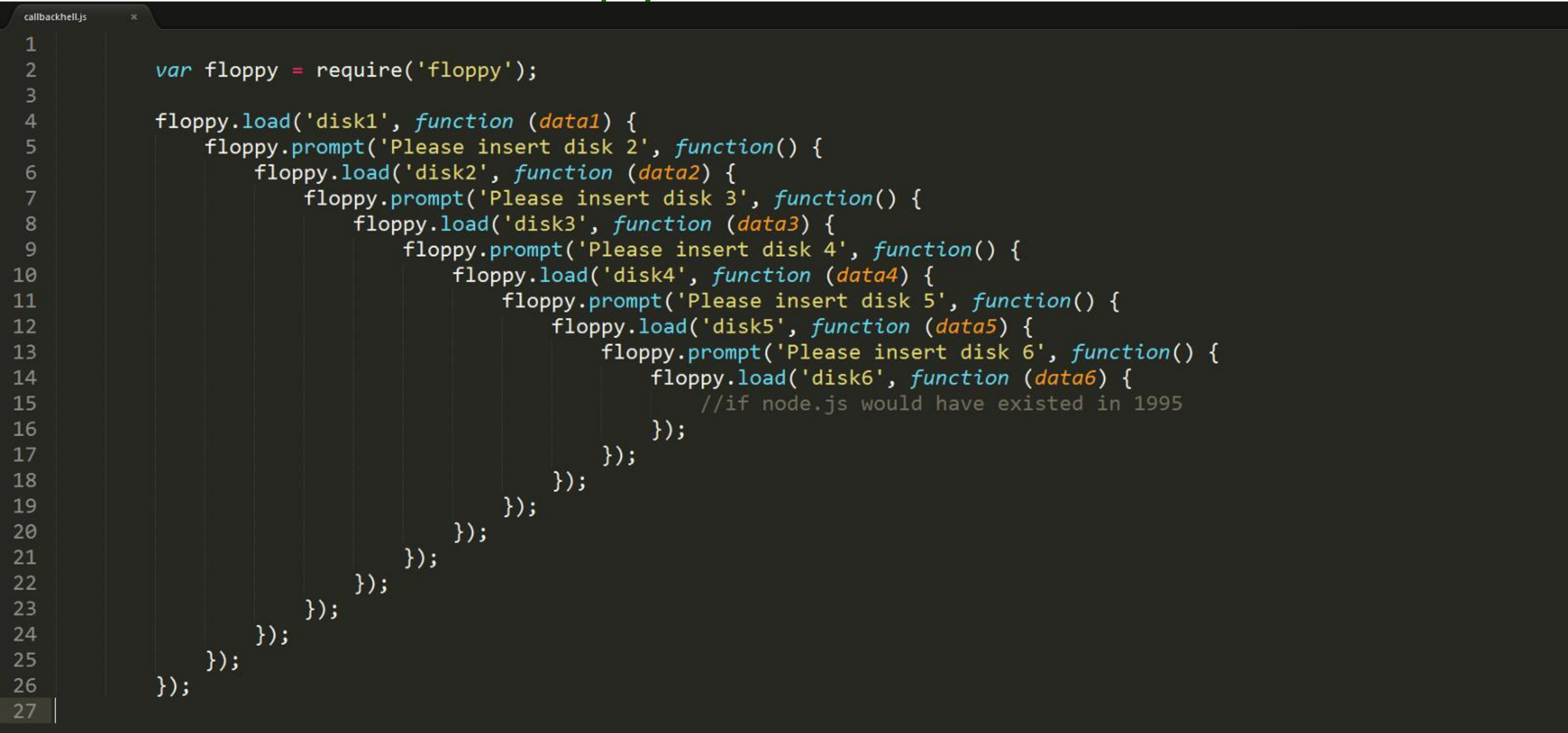
We would like to add couple of songs to a playlist on Spotify.

Here are the steps that we need:

- Retrieve temporary access token
- Retrieve user’s id using the access token that we just got
- Create a brand new empty playlist
- Try to look for the song on Spotify for every song on the list
- Since we got the user’s id from step 2 as well as the playlist’s id from step 3, we should now be able to add songs to the playlist on Spotify

```
post("https://accounts.spotify.com/api/token", {}, urlencode({
    grant_type: 'authorization_code',
    code: getParam(tab.url, 'code'),
    redirect_uri: "https://www.lukaszuk.net/spotify.html",
    client_id: ":)",
    client_secret: ":)",
}), function (response) {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    get("https://api.spotify.com/v1/me", {
        Authorization: tokenType + ' ' + accessToken
    }, null, function (response) {
        var userId = response.id;
        post("https://api.spotify.com/v1/users/" + userId + "/playlists", {
            Authorization: tokenType + ' ' + accessToken,
            "Content-type": "application/json"
        }, JSON.stringify({
            name: localStorage.playlistTitle
        }), function (response) {
            var playlistId = response.id;
            var songs = JSON.parse(localStorage.songs);
            var i = 0;
            for (key in songs) {
                get("https://api.spotify.com/v1/search", {
                    Authorization: tokenType + ' ' + accessToken
                }, "q=" + songs[key].title + "%20album:" + songs[key].album + "%20artist:
" + songs[key].artist + "&type=track", function (response) {
                    if (response.tracks.items.length) {
                        var uri = response.tracks.items[0].uri;
                        post("https://api.spotify.com/v1/users/" + userId + "/playlists/"
 + playlistId + "/tracks", {
                            Authorization: tokenType + ' ' + accessToken,
                            "Content-type": "application/json"
                        }, JSON.stringify({
                            uris: [uri]
                        }), function (response) {
                            // song has been added to the playlist
                        });
                    }
                });
            };
        });
    });
});
```

Callback hell / pyramide of doom



```
var floppy = require('floppy');

floppy.load('disk1', function (data1) {
    floppy.prompt('Please insert disk 2', function() {
        floppy.load('disk2', function (data2) {
            floppy.prompt('Please insert disk 3', function() {
                floppy.load('disk3', function (data3) {
                    floppy.prompt('Please insert disk 4', function() {
                        floppy.load('disk4', function (data4) {
                            floppy.prompt('Please insert disk 5', function() {
                                floppy.load('disk5', function (data5) {
                                    floppy.prompt('Please insert disk 6', function() {
                                        floppy.load('disk6', function (data6) {
                                            //if node.js would have existed in 1995
                                        });
                                    });
                                });
                            });
                        });
                    });
                });
            });
        });
    });
});
```

Solution #1 - comments

General rule says that you should avoid putting comments in your code.

Sometimes using comments is justified and can bring benefits.

```
// function get(url, header, param, success) {...}
// function post(url, header, param, success) {...}

// Retrieve temporary access token
post("https://accounts.spotify.com/api/token", {}, urlencode({
    grant_type: 'authorization_code',
    code: getParam(tab.url, 'code'),
    redirect_uri: "https://www.lukaszuk.net/sfy.html",
    client_id: ":)",
    client_secret: ":)",
}), function (response) {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    // Retrieve user's id using the access token that we just got
    get("https://api.spotify.com/v1/me", {
        Authorization: tokenType + ' ' + accessToken
    }, null, function (response) {
        var userId = response.id;
        // Create a brand new empty playlist
        post("https://api.spotify.com/v1/users/" + userId + "/playlists", {
            Authorization: tokenType + ' ' + accessToken,
            "Content-type": "application/json"
        }, JSON.stringify({
            name: localStorage.playlistTitle
        }), function (response) {
            var playlistId = response.id;
            var songs = JSON.parse(localStorage.songs);
            var i = 0;
            // Try to look for the song on Spotify for every song on the list
            for (key in songs) {
                get("https://api.spotify.com/v1/search", {
                    Authorization: tokenType + ' ' + accessToken
                }, "q=" + songs[key].title + "%20album:" + songs[key].album + "%20artist:" +
songs[key].artist + "&type=track", function (response) {
                    if (response.tracks.items.length) {
                        var uri = response.tracks.items[0].uri;
                        // Since we got the user's id from step 2 as well as the playlist's i
d from step 3, we should now be able to add songs to the playlist on Spotify
                        post("https://api.spotify.com/v1/users/" + userId + "/playlists/" + p
laylistId + "/tracks", {
                            Authorization: tokenType + ' ' + accessToken,
                            "Content-type": "application/json"
                        }, JSON.stringify({
                            uris: [uri]
                        }), function (response) {
                            // song has been added to the playlist
                        });
                    }
                });
            };
        });
    });
});
```

Solution #2 - smaller functions

Splitting long function into multiple smaller functions is always good idea.

Small pieces of code are:

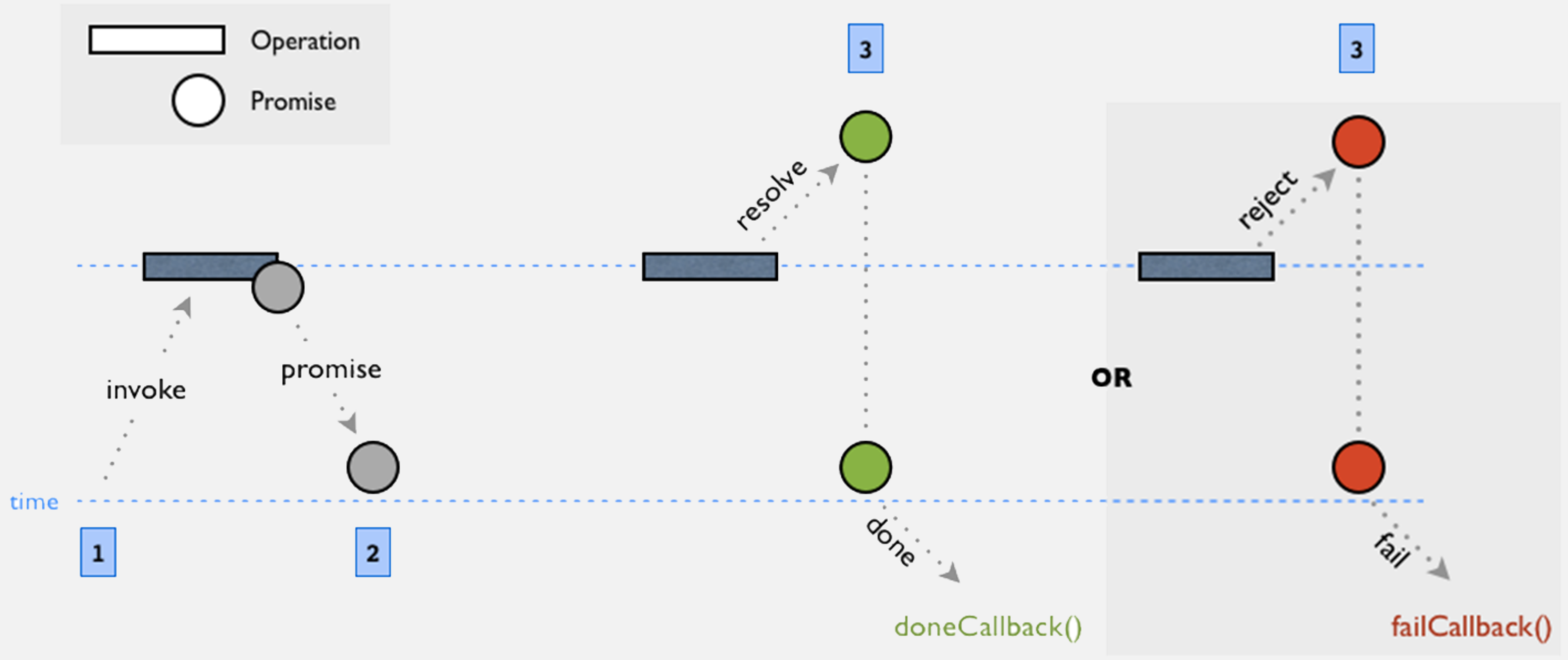
- easier to read
- easier to understand
- easier to change
- …

```
// function get(url, header, param, success) {...}
// function post(url, header, param, success) {...}

var tokenType, accessToken, userId, playlistId, songs = JSON.parse(localStorage.songs);

function retrieveAccessToken(callback) {
    post("https://accounts.spotify.com/api/token", {}, urlencode({
        grant_type: 'authorization_code',
        code: getParam(tab.url, 'code'),
        redirect_uri: "https://www.lukaszuk.net/sfy.html",
        client_id: ":)",
        client_secret: ":)",
    }), function (response) {
        callback(response);
    });
}

function retrieveUserId(response, callback) {
    tokenType = response.token_type;
    accessToken = response.access_token;
    get("https://api.spotify.com/v1/me", {
        Authorization: tokenType + ' ' + accessToken
    }, null, function (response) {
        callback(response);
    });
}

function createANewPlaylist(response, callback) {
    userId = response.id;
    post("https://api.spotify.com/v1/users/" + userId + "/playlists", {
        Authorization: tokenType + ' ' + accessToken,
        "Content-type": "application/json"
    }, JSON.stringify({
        name: localStorage.playlistTitle
    }), function (response) {
        callback(response);
    });
}

function searchASong(key, callback) {
    get("https://api.spotify.com/v1/search", {
        Authorization: tokenType + ' ' + accessToken
    }, "q=" + songs[key].title + "%20album:" + songs[key].album + "%20artist:" + songs[key].
artist + "&type=track", function (response) {
        callback(response);
    });
}

function addASongToThePlaylist(uri, callback) {
    post("https://api.spotify.com/v1/users/" + userId + "/playlists/" + playlistId + "/track
s", {
        Authorization: tokenType + ' ' + accessToken,
        "Content-type": "application/json"
    }, JSON.stringify({
        uris: [uri]
    }), function (response) {
        callback(response);
    });
}

function addAllSongsToPlayList(response, callback) {
    playlistId = response.id;
    var i = 0;
    for (key in songs) {
        searchASong(key, function (response) {
            if (response.tracks.items.length) {
                addASongToThePlaylist(response.tracks.items[0].uri, function (response) {
                    i++;
                });
            }
        });
    }
    callback(i);
}

retrieveAccessToken(function (response) {
    retrieveUserId(response, function (response) {
        createANewPlaylist(response, function (response) {
            addAllSongsToPlayList(response, function (total) {
                console.log("There are " + total + " out of " + songs.length + " songs been
added to the playlist!!!");
            });
        });
    });
});
```

Solution #3 - using promises

Promise - class that allows you to create objects, representing value or failure of async operation.

Promise represents an operation that is not yet finished, but it is expected to end in the future.



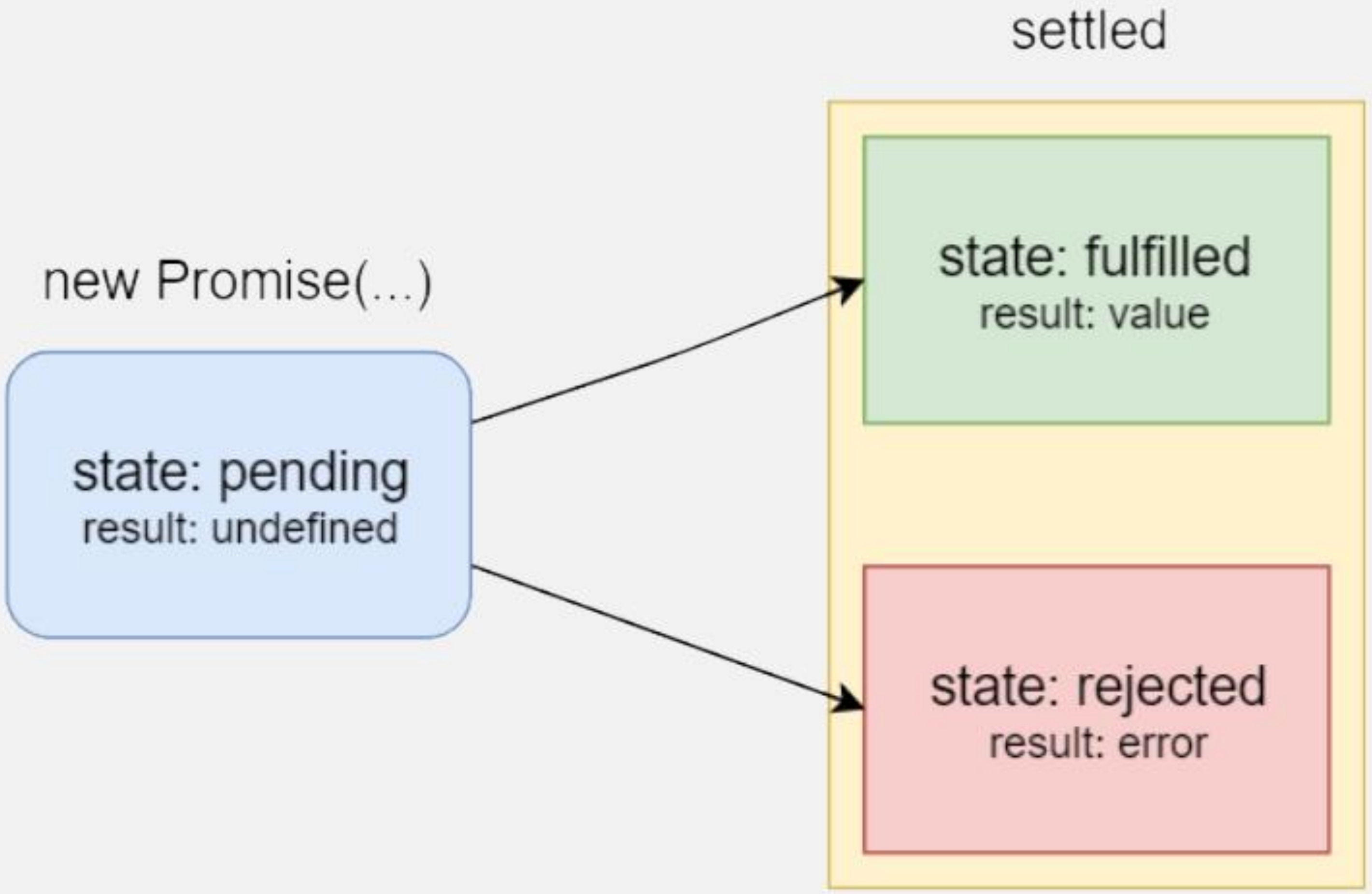
Callback vs Promise

- Callback is a function, Promise is an object
- Callback accepts parameters, Promise return value
- Callback supports success and error, Promise handles nothing but pass on values
- Callback can be called many times, Promise is only called once

Promise

Promise states:

- pending
- settled:
  - fulfilled
  - rejected



Create Promise

```
const myPromise = new Promise((resolve, reject) => {
    /* some logic */
    if (/* some condition */) {
        resolve('all works fine');
    } else {
        reject('error');
    }
});
```

Converting Callbacks into Promises

In practice, callbacks would probably be written for you already.

If you use Node, each function that contains a callback will have the same syntax:

- the callback would be the last argument
- the callback will always have two arguments. And these arguments are in the same order. (Error first, followed by whatever you’re interested in).

If your callback has the same syntax, you can use libraries like:

es6-promisify or Node.js util.promisify.

Converting Callbacks into Promises

```
//callback
const fs = require('fs');



const data = 'ala ma kota';
fs.writeFile('some-file.txt’,
   data,
   () => {
       console.log('file saved’);
   }
)
```

```
//promise
const util = require('util');
const fs = require('fs');

const writePromise = util.promisify(fs.writeFile);
const data = 'ala ma kota';
writePromise('some-file.txt', data)
    .then(() => {
        console.log('file saved');
    });
```

Converting Callbacks into Promises

```
const fs = require('fs');

const data = 'ala ma kota';
try {
    fs.writeFile('some-file.txt',
    data,
      () => {
        console.log('file saved');
      }
    )
}
catch(error){
    console.log('error: ', error);
}
```

```
//promise
const util = require('util');
const fs = require('fs');

const writePromise = util.promisify(fs.writeFile);
const data = 'ala ma kota';
writePromise('some-file.txt', data)
    .then(() => {
        console.log('file saved');
    })
    .catch((error) => {
        console.log('error: ', error);
    });
```

Promise syntax

```
retrieveAccessToken(tab.url)
    .then(retrieveUserInfo)
    .then(createAPlaylist)
    .then(getAllSongsInfo)
    .then(prepareToaddAllSongsToPlaylist)
    .then(addAllSongsToPlaylist)
    .catch(error => {
        // error handling
    });
```

```
const retrieveAccessToken = url => {
    return new Promise(resolve => {
        post("https://accounts.spotify.com/api/token", {}, ur
lencode({
            grant_type: 'authorization_code',
            code: getParam(url, 'code'),
            redirect_uri: "https://www.lukaszuk.net/sfy.html",
            client_id: ":)",
            client_secret: ":)",
        }), response => {
            resolve(response);
        });
    })
};

const retrieveUserInfo = response => {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    return new Promise(resolve => {
        get("https://api.spotify.com/v1/me", {
            Authorization: tokenType + ' ' + accessToken
        }, null, response => {
            response['token_type'] = tokenType
            response['access_token'] = accessToken;
            return resolve(response);
        });
    });
};

const createAPlaylist = response => {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    var userId = response.id;
    return new Promise(resolve => {
        post("https://api.spotify.com/v1/users/" + userId + "
/playlists", {
            Authorization: tokenType + ' ' + accessToken,
            "Content-type": "application/json"
        }, JSON.stringify({
            name: localStorage.playlistTitle
        }), response => {
            response['token_type'] = tokenType
            response['access_token'] = accessToken;
            response['userId'] = userId;
            return resolve(response);
        });
    });
};

const searchASong = response => {
    return new Promise(resolve => {
        get("https://api.spotify.com/v1/search", {
            Authorization: response.token_type + ' ' + respon
se.access_token
        }, buildSearchQuery(response.song), responseFromSearc
h => {
            resolve(responseFromSearch.tracks.items[0]);
        });
    });
};


const getAllSongsInfo = response => {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    var playlistId = response.id;
    var userId = response.userId;
    var songs = JSON.parse(localStorage.songs);
    var allSearchPromises = [];
    for (key in songs) {
        response['song'] = songs[key];
        allSearchPromises.push(searchASong(response));
    }
    return Promise.all(allSearchPromises).then(function (resp
onse) {
        response['token_type'] = tokenType;
        response['access_token'] = accessToken;
        response['playlistId'] = playlistId;
        response['userId'] = userId;
        return response;
    });
};

const prepareToaddAllSongsToPlaylist = response => {
    var songs = [];
    for (key in response) {
        if (isNumeric(key)) {
            songs.push(response[key].uri);
        }
    }
    return new Promise(resolve => {
        response['songs'] = songs;
        resolve(response);
    });
};

const addAllSongsToPlaylist = response => {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    var playlistId = response.playlistId;
    var userId = response.userId;
    var songs = response.songs;
    return new Promise(resolve => {
        post("https://api.spotify.com/v1/users/" + userId + "
/playlists/" + playlistId + "/tracks", {
            Authorization: tokenType + ' ' + accessToken,
            "Content-type": "application/json"
        }, JSON.stringify({
            uris: songs
        }), function (response) {
            resolve(response);
        });
    });
};

function isNumeric(n) {
    return !isNaN(parseFloat(n)) && isFinite(n);
}

function buildSearchQuery(song) {
    return "q=" + song.title +         "%20album:" + song.alb
um +
        "%20artist:" + song.artist + "&type=track";
}

retrieveAccessToken(tab.url)
    .then(retrieveUserInfo)
    .then(createAPlaylist)
    .then(getAllSongsInfo)
    .then(prepareToaddAllSongsToPlaylist)
    .then(addAllSongsToPlaylist)
    .catch(error => {
        progress.innerHTML += "[WARNING] " + error + "<br>";
    });
```

Solution #4 – using async/await

Data return from async functions is automatically wrapped in a promise.

Using await keyword will automatically extract data from promise.

Await keyword can be used only inside functions marked as async

Async/await

```
// PROMISE
function asyncAction() {
    return new Promise((resolve, reject) => {
        const successTimeout = Math.random() * 10000;
        const errorTimeout = Math.random() * 10000;

        setTimeout(() => {
            resolve('success');
        }, successTimeout);

        setTimeout(() => {
            reject('error');
        }, errorTimeout);
    });
}
```

Async/await - syntax

```
// PROMISE
function doWork() {
    asyncAction()
        .then(data => {
            console.log(data);
        });
}

doWork();
```

Async/await - syntax

```
// PROMISE
function doWork() {
    asyncAction()
        .then(data => {
            console.log(data);
        });
}

doWork();
```

```
// ASYNC/AWAIT
async function doWork() {
    const data = await asyncAction();
    console.log(data);
}

doWork();
```

Async/await - syntax with error handling

```
// PROMISE
function doWork() {
    asyncAction()
        .then(data => {
            console.log(data);
        })
        .catch(error => {
            console.log(error);
        });
}

doWork();
```

```
// ASYNC/AWAIT
async function doWork() {
    try {
        const data = await asyncAction();
        console.log(`message = ${data}`);
    } catch (error) {
        console.log(`message = ${error}`);
    }
}

doWork();
```

```
const retrieveAccessToken = url => {
    return new Promise(resolve => {
        post("https://accounts.spotify.com/api/token", {}, ur
lencode({
            grant_type: 'authorization_code',
            code: getParam(url, 'code'),
            redirect_uri: "https://lukaszuk.net/sfy.html",
            client_id: ":)",
            client_secret: ":)",
        }), response => {
            resolve(response);
        });
    })
};

const retrieveUserInfo = response => {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    return new Promise(resolve => {
        get("https://api.spotify.com/v1/me", {
            Authorization: tokenType + ' ' + accessToken
        }, null, response => {
            response['token_type'] = tokenType
            response['access_token'] = accessToken;
            return resolve(response);
        });
    });
};

const createAPlaylist = response => {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    var userId = response.id;
    return new Promise(resolve => {
        post("https://api.spotify.com/v1/users/" + userId + "
/playlists", {
            Authorization: tokenType + ' ' + accessToken,
            "Content-type": "application/json"
        }, JSON.stringify({
            name: localStorage.playlistTitle
        }), response => {
            response['token_type'] = tokenType
            response['access_token'] = accessToken;
            response['userId'] = userId;
            return resolve(response);
        });
```

```
    });
};

const searchASong = response => {
    return new Promise(resolve => {
        get("https://api.spotify.com/v1/search", {
            Authorization: response.token_type + ' ' + respon
se.access_token
        }, buildSearchQuery(response.song), responseFromSearc
h => {
            resolve(responseFromSearch.tracks.items[0]);
        });
    });
};

const getAllSongsInfo = response => {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    var playlistId = response.id;
    var userId = response.userId;
    var songs = JSON.parse(localStorage.songs);
    var allSearchPromises = [];
    for (key in songs) {
        response['song'] = songs[key];
        allSearchPromises.push(searchASong(response));
    }
    return Promise.all(allSearchPromises).then(function (resp
onse) {
        response['token_type'] = tokenType;
        response['access_token'] = accessToken;
        response['playlistId'] = playlistId;
        response['userId'] = userId;
        return response;
    });
};

const prepareToaddAllSongsToPlaylist = response => {
    var songs = [];
    for (key in response) {
        if (isNumeric(key)) {
            songs.push(response[key].uri);
        }
    }
    return new Promise(resolve => {
        response['songs'] = songs;
```

```
        resolve(response);
    });
};

const addAllSongsToPlaylist = response => {
    var tokenType = response.token_type;
    var accessToken = response.access_token;
    var playlistId = response.playlistId;
    var userId = response.userId;
    var songs = response.songs;
    return new Promise(resolve => {
        post("https://api.spotify.com/v1/users/" + userId + "
/playlists/" + playlistId + "/tracks", {
            Authorization: tokenType + ' ' + accessToken,
            "Content-type": "application/json"
        }, JSON.stringify({
            uris: songs
        }), function (response) {
            resolve(response);
        });
    });
};

function isNumeric(n) {
    return !isNaN(parseFloat(n)) && isFinite(n);
}

function buildSearchQuery(song) {
    return "q=" + song.title + "%20album:" + song.album +
        "%20artist:" + song.artist +  "&type=track";
}

const beginToAddSongsToPlaylist = async () => {
    let response = await retrieveAccessToken(tab.url);
    response = await retrieveUserInfo(response);
    response = await createAPlaylist(response);
    response = await getAllSongsInfo(response);
    response = await prepareToaddAllSongsToPlaylist(response)
;
    response = await addAllSongsToPlaylist(response);
};

beginToAddSongsToPlaylist();
```

Promises vs async/await

```
//PROMISE
retrieveAccessToken(tab.url)
    .then(retrieveUserInfo)
    .then(createAPlaylist)
    .then(getAllSongsInfo)
    .then(prepareToaddAllSongsToPlaylist)
    .then(addAllSongsToPlaylist)
    .catch(error => {
        progress.innerHTML += "[WARNING] " + error + "<br>";
    });
```

```
//ASYNC-AWAIT
const beginToAddSongsToPlaylist = async () => {
    let response = await retrieveAccessToken(tab.url);
    response = await retrieveUserInfo(response);
    response = await createAPlaylist(response);
    response = await getAllSongsInfo(response);
    response = await prepareToaddAllSongsToPlaylist(response);
    response = await addAllSongsToPlaylist(response);
};

beginToAddSongsToPlaylist();
```

Promise.all vs async/await

```
var p1 = new Promise((resolve, reject) => { … });
var p2 = new Promise((resolve, reject) => { … });
var p3 = new Promise((resolve, reject) => { … });

Promise.all([p1, p2, p3]).then(values => {
  console.log(values);
  // result of all promises […, …, …]
});
```

:(

Promise.any vs async/await

```
var p1 = new Promise((resolve, reject) => { … });
var p2 = new Promise((resolve, reject) => { … });
var p3 = new Promise((resolve, reject) => { … });

Promise.any([p1, p2, p3]).then((value) => {
    console.log(value);
    // result which fulfils first
})
```

```
:(
```

Promise.race vs async/await

```
var p1 = new Promise((resolve, reject) => { … });
var p2 = new Promise((resolve, reject) => { … });
var p3 = new Promise((resolve, reject) => { … });

Promise.race([p1, p2, p3]).then((value) => {
    console.log(value);
    // result which fulfils or rejects first
})
```

```
:(
```

Async/Await

Pros of async/await approach:

- similar pattern is available in other languages: C#, F#, Python, Rust, Scala
- concise and clean
- error handling using common javascript approach
- more accessible intermediate values
- easier debugging

Cons of async/await approach:

- looks less like JavaScript
- slightly less functionality

Error handling with Async/Await

```
// PROMISES
const axios = require('axios’);
axios.get(url)
    .then((response) => {
        console.log(response.data.name);
    })
    .catch(error => {
        console.log(error);
    });
```

```
// ASYNC/AWAIT
const axios = require('axios');
(async function () {
    try {
        const response = await axios.get(url);
        console.log(response.data.name);
    } catch (error) {
        console.log(error);
    }
})();
```

Async/Await summary

Async/await is really syntactic sugar for promises
because it still uses promises under the hood.

It’s not either/or
You can use both promise chains and async await,
and there’s nothing wrong with that :)