**http status code**

1xx Informational response

100 Continue -everything so far is OK, the client can continue the request or ignore if it's finished.

2xx Success

200 OK - The request has succeeded

201 Created - The request has succeeded and new resource was created

202 Accepted - has been received but not yet acted upon

204 No Content - Tno content to send for this request

205 Reset

206 Partial

3xx Redirection

300 Multiple Choice

301 Moved Permanently - URI of the requested resource has been changed permanently

302 Found - has been changed temporarily

303 See Other - direct the client to get the requested resource at another URI with a GET request.

304 Not Modified - Thas not been modified, so the client can use cached version

307 Temporary - direct the client to get resource at another URI with same method that was used in the prior request

308 Permanent - resource is now permanently located at another URI, specified by the Location: HTTP Response header

4xx Client errors

400 Bad Request - server could not understand the request due to invalid syntax

401 Unauthorized - Asemantically this response means "unauthenticated"

402 Payment Required - This response code is reserved for future use

403 Forbidden - does not have access rights to the content, unlike 401, the client's identity is known to the server.

404 Not Found - The server can not find requested resource.

405 Method Not Allowed - The request method is known by the server but has been disabled and cannot be used

406 Not Acceptable

408 Request Timeout - sent on an idle connection by some servers

409 Conflict - conflicts with the current state of the server

410 Gone - content has been permanently deleted from server

411 Length Required - Content-Length header field is required

413 Payload Too Large - payload large than limits defined by server

414 URI Too Long - URI longer than server allows

**webpack**

The webpack library allows to bundle javascript files into one big file,

bundle other static resources like images, handle bar files, css files, etc.

field "module" several loaders are defined, to be able to load static resources like images, css files and handlebars files.

"HtmlWebpackPlugin" plugin was used to simplify the creation

of HTML files to serve the webpack bundles.

npm run-script -> arbitrary command from a package’s "scripts", same as npm-build

npm-build: executes build instruction on package.json

415 Unsupported Media Type - The media format is not supported by server

416 Requested Range Not Satisfiable

417 Expectation Failed

426 Upgrade Required

428 Precondition Required

429 Too Many Requests - sent too many requests ("rate limiting").

431 Request Header Fields Too Large

451 Unavailable For Legal Reasons

5xx Server errors

500 Internal Server Error - error server doesn't know how to handle.

501 Not Implemented – The request method is not supported

502 Bad Gateway

503 Service Unavailable - The server is not ready to handle the request

507 Insufficient Storage

510 Not Extended

511 Network Authentication Required

**http methods**

GET - deve retornar apenas dados.

HEAD - GET sem conter o corpo da resposta.

POST - submeter uma entidade a um recurso específico, causando mudança de estado do recurso ou efeitos colaterais no servidor.

PUT - substitui todas as atuais representações do recurso d destino pela carga de dados da requisição.

DELETE - remove um recurso

CONNECT - faz túnel para servidor identificado pelo recurso.

OPTIONS - descrever as opções de comunicação com o recurso.

TRACE - executa um teste loop-back com o caminho para o recurso.

PATCH - aplica modificações parciais a recurso.

**css**

**.class -** elemento com classe

**.class1.class2** - elemento com as 2 classes

**#id** - elemento com id

**element -** elementos com tag

**element.class -** elementos com tag e classe

**element1,element2 -** elementos com tag elemento1 ou elemento2

**element1 element2 -** todos os elementos element2 que estejam dentro de um element1

**element1>element2 -** todos os elementos element2 em que o pai seja um element1

more specific rules override more general ones, in tie, last one to appear wins(overwirte)

**DOM**

Document Object Model, “DOM”, is an interface to web pages. Allows read and manipulate the page’s content, structure, and styles. The Document interface represents any web page loaded in the browser and serves as an entry point into the web page's content, which is the DOM tree. The Window interface represents a window containing a DOM document. document.readyState - property describes the loading state of the document (loading, interactive, complete) interactive - the document has finished loading but sub resources like imeges and stylesheets and frames are still loading complete - trigger load event

**nodeJs**

-g or --global, it installs the package as global.

--production flag (or NODE\_ENV environment variable is "production"), npm will not install modules listed in devDependencies

-P, --save-prod: Package will appear in your dependencies.

This is the default unless -D or -O are present.

-D, --save-dev: Package will appear in your devDependencies.

-O, --save-optional: Package will appear in your optionalDependencies.

--no-save: Prevents saving to dependencies.

Two aditional flags when saving packages to package.json:

-E, --save-exact: Saved dependencies will be configured with an exact

version rather than using npm’s default semver range operator.

-B, --save-bundle: Saved dependencies will also be added to your bundleDependencies list.

req.on('data', chunk => {

    body.push(chunk)

}).on('end', () => {

    body = Buffer.concat(body).toString();

    res.statusCode = 200;

    res.end(handler.call())

})

// handler – function that uses body to do things

element = document.querySelector(selectors);

elementList = document.querySelectorAll(selectors);

element = document.getElementById(id);

element.addEventListener(type,listener);

element.className

eventos: change, click, copy, focus, hashchange,

input, keypress, mouseenter, mouseleave

**selector .class, #id, <tg>**

**javascript**

**let array1 = [1, 2, 3, 4, 5];**

let found = array1.**find**(nr > nr > 2); // expected output: 3

let biggerThanThree = array1.**filter**( nr => nr >= 3 ) // expected output: [ 3, 4, 5 ]

let smallerThanThree = array1.**filter**(function(nr){ return nr < 3 }) // expected output: [ 1, 2 ]

let reducer = (accumulator,currentValue) => accumulator + currentValue;

let reduce1 = array1.**reduce**(reducer); // expected output: 15

let reduce2 = array1.**reduce**(reducer, 5); // expected output: 20

let slice = array1.**slice**(1,4);// /arr.slice([start[,end]]) // expected output: [2, 3, 4]

let array1COPY = array1.**slice**(); // copy

let triples = array1.**map**( function(nr) { return nr \* 3} ); // expected output:  [ 3, 6, 9, 12, 15 ]

array1.**map**(x => [x \* 2]); // expected output: [[2], [4], [6], [8]]

array1.**flatMap**(x => [x \* 2]); // expected output: [2, 4, 6, 8]

**o = new Object(); o.prop = 'exists'**;

o.**hasOwnProperty**('prop');   // expected output:  true

array1.**forEach**(element => console.log(element)); // expected output: "1" "2" "3" "4" "5"

// arrayOfStrings = stringToSplit.split(separator);

**let monthString = 'Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec';**

let arrayOfStrings = monthString.**split**(","); // expected output: [Jan, Feb, Mar, Apr, May, Jun, …, ]

arrayOfStrings.**includes**('Jan'); // expected output: true

fn.call(thisArg, 1, 2, 3)  // both are used in functions like **var.function**

fn.apply(thisArg, [1, 2, 3]) // (for arguments use apply)

**handlebars**

<table>

    <thead>

        <tr>

            {{#each headings as |column|}}

                <th>{{column}}</th>

            {{/each}}

        </tr>

    </thead>

    <tbody>

        {{#each items as |row|}}

            <tr>

                {{#each row as |cell|}}

                    <td>{{cell}}</td>

                {{/each}}

            </tr>

        {{/each}}

    </tbody>

</table>

let slicedArgs = Array.prototype.slice.call(arguments);

function createObj() {

    const items = []

    return {

        addItem : (e) => Array.prototype.push.call(items, e),

        addItem2: (e) => items.push(e),

        addItems: (...e) => items.push(e),

    };

}

function filterProperties(propNames, obj) {

    let res = {};

    propNames.forEach(p => {

        if (obj.hasOwnProperty(p))

            res[p] = obj[p];

    });

    return res;

}

function filterPropertiesN(propNames, objs) {

    return objs.map((obj) => filterProperties(propNames, obj))

}

function mw(req, rsp, next) {

    let ogJson = rsp.json;

    rsp.json = function(obj) {

        let filter = req.query.filter;

        if(filter !== undefined) {

            let splitedfilter = filter.split(',');

            let newObj = {};

            if(obj instanceof Array)

<form id="searchGamesForm">

    <input type="text" name="gameName" placeholder="name" value="" />

    <input type="submit" />

</form>

input types: checkbox, email, hidden, image, password, number

                newObj = filterPropertiesN(splitedfilter, obj);

            else

                newObj = filterProperties(splitedfilter, obj);

            ogJson.call(rsp, newObj);

        } else {

            ogJson.call(rsp, obj);

        }

    };

    next();

}

function get(url, cb) { fetch(url) . then(resp => { cb(null, resp) }) . catch( err =>cb(err)) }

**passport**

server.use(session({ //express-session

    secret: props.session.secretKey,

    resave: false,

    saveUninitialized: false

}));

server.use(passport.initialize()); // initialize: definidas estratégias de des/serializacao

server.use(passport.session()); do use,local strategy, authentication process

authMw: function(req, rsp, next) {

    if (req.isAuthenticated()) {

        return next();

    } else {

        res.redirect("/")

    }

}

const validateAuthenticationMw = require ( 'validate-auth' )( LOGIN\_URL )

app.all("/private", validateAuthenticationMw, '<path action>')

app.all("/private/\*", validateAuthenticationMw, '<path action>')

const express = require("express"); const server = express();

module.exports = function () {

    let expressServer = {

        get: function (url, mw) { server.get(url, mw); },

        put: function (url, mw) { server.put(url, mw); },

        delete: function (url, mw) { server.delete(url, mw); },

        post: function (url, mw) { server.post(url, mw); },

        start: function(host, port) { server.listen(port, console.log("Server running on port: " + port)); }

    }

    return expressServer; }

let router = function(router, services, authentication) {

    router.post('/login', authentication.notAllowAuthenticatedRequests, services.login);

    router.delete('/logout', authentication.allowAuthenticatedRequests, services.logout);

    router.get('/games/', authentication.allowAuthenticatedRequests, services.getMostPopularGames);

    router.post('/groups', authentication.allowAuthenticatedRequests, services.createGroup);

    return router;

}

module.exports = router;

**yama-web-api**

async function addMusicToList(req, rsp){

    try {

        /\* validadores de parâmetros \*/

        let playlist = yamaService.addMusicToList(req.params.playlistId, req.params.musicId);

        res.statusCode = 201;

        res.setHeader('Content-type', 'application/json');

        res.end(JSON.stringify(playlist));

    } catch(err) {

        res.statusCode = 500;

        res.setHeader('Content-type', 'application/json');

        res.end(JSON.stringify(err));

    }

}

function groups(data, routesManager) {

    routesManager.setMainContent(templates.groups(data));

    const formCreateGroup = document.querySelector("#createGroup");

    formCreateGroup.addEventListener('submit', handleSubmit);

    function handleSubmit(e) {

        e.preventDefault();

        const formName = document.querySelector("#formName");

        const formDescription =

 document.querySelector("#formDescription");

        routesManager.changeRoute('createGroup',

{ name: formName.value, description: formDescription.value });

    }

    const rows = document.querySelectorAll("[id\*=gameid\_]");

    rows.forEach(row => {

        row.addEventListener('click', handleRowClick);

    });

    function handleRowClick(e) {

        routesManager.changeRoute(this.getAttribute("data-href"));

    }

}

**yama-db**

async function addMusicToList(playListId, musicId){

    let playlist = await yamaDb.getPlaylistById(playlistId);

    if(!playlist) {

        let err =  new Error("No playlist with that id");

        err.status = 404;

        throw err;

    }

    if(playlist.allowRepeats || !playlist.musics.find(music.id === musicId)) {

        yamaDb.addGameToList(playlistId, musicId);

    } else {

        let err =  new Error("Cannot add repeated music for this playlist");

        err.status = 500;

        throw err;

    }

}

**yama services**

async function getArtistsNames(nameStart, resultsCount) {

    let options = {

        method: "GET",

        headers: new Headers("Accept", "application/json")

    }

    let response = await fetch(`/artists/${nameStart}?max=${resultsCount}`, options);

// orginal fetch

fetch(url, options).then(function(response) {

    // handle HTTP response

  }, function(error) {

    // handle network error

})

fetch = function() {

    let originalFetch = fetch;

    return async (p, o) => {

        try {

            let cachedResp = fetch.cache[p];

            if (!cachedResp) {

                let resp = await originalFetch(p, o);

                if(resp.headers.has("Cache-Control")

                && (resp.headers.has("Cache-Control") === "private"

                || resp.headers.has("Cache-Control") === "public")){

                    fetch.cache[p] = resp;

                }

                return resp;

            } else {

                return cachedResp;

            }

        } catch(e){

            console.log(e)

        }

    }

}();

fetch.cache = {};

v

    if(response.ok){

// POST /groups/{id}/games

async function AddGamesToGroup(req, rsp) {

    let options = {

        method: "POST",

        headers: new Headers("Accept", "aplication/json"),

        body: [], // array with gamesIds

    }

    let res = await fetch('/groups/{id}/games', options);

    if(res.ok)

        let body = await res.json();

    else

        return res.status;

// Status Codes: 400 - Bad Request  404 - Not Found ,

// 500 - Internal Server Error

}

function addGamesToGroup(req, rsp) {

    let groupId = req.params.groupId;

    let gamesIds = req.body.gamesIds;

    /\* validadores \*/

   let res = await ciborg-services.AddGamesToGroup(groupId, gamesIds);

}

function addGamesToGroup(groupId, gamesIds) {

    let games = await board-games-data.getGames(gamesIds);

    addedGamesPromisses = games.map(game => ciborg-db.addGameToGroup(groupId, game));

    return Promisse.all(addedGamesPromisses);

}

        let body = await response.json();

        return body;

    } else { return response.status; }

}

**yama html**

document.querySelector("#artists").addEventListener("change", listener);

async function listener(e) {

    let input = document.querySelector("#artists");

    let div = document.querySelector("#suggestions");

    if(input.value.length < 3) {

        if(input.className != "hiddenAutoComplete") {

            input.className = "hiddenAutoComplete";

        }

        div.innerHtml = "";

    } else {

        if(input.className != "visibleAutoComplete") {

            input.className = "visibleAutoComplete";

        }

        let artists = await getArtistsNames(input.value, 10);

        let htmlList = "";;

        artists.forEach(e => {

            artistList += `<div>${e.name}</div>`;

        });

        div.innerHtml = artistList;

    }

}

**API fetch()**

async function getAllBundlesBookTitles () {

**waterfall**

function promisify(fn) {

    return async function() {

        let args = arguments;

        return new Promise(function(resolve, reject) {

            let cb = (err, res) => {

                if(err){

                    reject(err);

                } else {

                    resolve(res);

                }

            };

            args = [].slice.call(args);

            args.push(cb);

            fn.apply(null, args);

        });

    }

}

    let response = await fetch ( 'http://localhost:9200/b4/bundle/\_search' )

    let bundles = await response . json ();

    const bookUrlBase = 'http://localhost:9200/books/book/'

**copia função, callback ultimo argumento e retorna promise**

function promisify(fn) {

    return async function() {

        let args = arguments;

        return new Promise(function(resolve, reject) {

            let cb = (err, res) => {

                if(err){

                    reject(err);

                } else {

                    resolve(res);

                }

            };

            args = [].slice.call(args);

            args.push(cb);

            fn.apply(null, args);

        });

    }

}

v

    const requests = bundles

        . flatMap ( bundle => bundle . books )

        . map ( b => fetch ( bookUrlBase + b . id ))

    const responses = await Promise . all ( requests );

    const books = await Promise . all ( responses . map ( async resp => await resp . json ()))

    const booksTitles = books . map ( b => b . \_source . title )

    return booksTitles ;

}

async function getAllBundlesBookTitles\_withoutAwaitKeyWord () {

    const url = 'http://localhost:9200/b4/bundle/\_search'

    const bookUrlBase = 'http://localhost:9200/books/book/'

    return fetch (url)

        .then(response => response . json())

        .then(bundles => bundles

            . flatMap ( bundle => bundle . books )

            . map ( b => fetch ( bookUrlBase + b . id ))

**calcular tempos de execução**

function profile(fn) {

    let newFn = function () {

        newFn.execs = [];

        let start = new Date().getTime();

        let result = fn.apply(null, arguments).then(res => {

            let end = new Date().getTime();

            newFn.execs.push(end - start);

            return res;

        });

        return result;

    }

    newFn.avgDur = function () {

        return (newFn.execs.reduce(

(accum, curr) => accum + curr, 0) / newFn.execs.length);

    };

    return newFn;

}

        )

        .then(response => response . map (resp => resp . json()))

**copia função que retorna uma promise e adiciona callback**

function nodify(fn) {

    let original = fn;

    let newFn = function(){

        let args = [].slice.call(arguments)

        let cb = args[args.length - 1];

        args = args.slice(0, args.length - 1)

        original

            .apply(null, args)

            .then(res => cb(null, res)).catch(err => cb(err))

    }

    return newFn

}

        .then(books => books . map ( b => b . \_source . title ))

}

new Promise(function(resolve, reject) {

    try {

        setTimeout(resolve, 100, 'foo');

    } catch(err) {

        reject(err);

    }

});

Promise.all([promise1, promise2, promise3]).then(function(values) {

  console.log(values);//array of values

});

<div class="container">

    <h1 class="title">List of all owned groups</h1>

    <div >

        <table class="table table-striped table-dark table-hover">

        <tr class="thead-dark">

            <th>Name</th>

            <th>Description</th>

        </tr>

        {{#if payload}}

            {{#each payload}}

            <tr id="gameid\_{{id}}"class="clickable-row" data-href="#group/{{id}}">

                <td>{{name}}</a></td>

                <td>{{description}}</td>

            </tr>

            {{/each}}

        {{else}}

            <tr>

                <td colspan="2" class="no-records">No records found!</td>

            </tr>

        {{/if}}

    </table>

    </div>

    <div>

    <form id="createGroup" action="/groups" method="POST" class="ciborg-form">

    <h4 class="formTitle">Add new group</h4>

        <div class="input-block">

            <label for="formName">Name</label>

            <input type="text" id="formName" class="form-control" >

        </div>

        <div class="input-block">

            <label for="formDescription">Description</label>

            <textarea name="formDescription" id="formDescription" cols="97" rows="10" class="form-control"></textarea>

        </div>

        </br>

        <div class="input-group">

            <input type="submit" class="btn submitBtn" value="Create Group" >

        </div>

    </form>

    </div>

</div>