```
<table>
    <h5>Games:</h5>
    <thead class="thead-dark">
        <tr>
            {{#each header as |column|}}
                <th scope="col" class="test-class">{{column}}</th>
            {{/each}}
        </tr>
    </thead>
    <tbody>
        {{#if elements}}
            {{#each elements as |row|}}
                <tr>
                    <td name="gameId">{{row.id}}</td>
                    <td>
                        <button id="{{@index}}" name="remove" type="button" >R</button>
                    </td>
                </tr>
            {{/each}}
        {{else}}
            <tr>
                <td colspan="5" class="no-records">No records found!</td>
            </tr>
        {{/if}}
    </tbody>
</table>
```

```
//user o this é user o proprio objeto em vez de nome da propriedade
var template = Handlebars.compile("Handlebars <b>{{doesWhat}}</b>");
template({ doesWhat: "rocks!" })
```

```
{
  person: {
    firstname: "Yehuda",
    lastname: "Katz"
  }
}
{{#with person}}
{{firstname}} {{lastname}}
{{/with}}
{{person.firstname}} {{person.lastname}}
```

```
let headers = new Headers();
let options = {
    method: "GET".
    headers: headers
}
fetch(url, options).then(resp =>
resp.json())
.then(body => console.log(body));
Response.headers
Response.ok
Response.status
Response.url
Response.json()
Response.formData()
Headers.getAll()
Headers.get(name)
Headers.append(name, value)
```

```
.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
In short: more specific rules override
more general ones
em caso de empate a ultima a aparecer ganha
Bootstrap é uma CSS Framework
The CSS box model is essentially a box that
wraps around every HTML element. It consists
of: margins, borders, padding, and the actual
content.
```

```
REQUEST:
Accept: application/json, application/x-www-form-urlencoded,
application/xml, text/html, text/css, text/csv, text/plain,
application/javascript, image/jpeg, image/png, image/gif, */*
Accept-Encoding: gzip, deflate
Content-Type: (ver Accept)
Cookie: UserID=JohnDoe;
User-Agent: quem enviou pedido
RESPONSE:
Content-Encoding: (ver Accept-Encoding)
Location: utilizado em redirects
Set-Cookie: UserID=JohnDoe;
```

```
    function profile(fn) {
      let newFn = function () {
        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.execs = [];
      newFn.avgDur = function () {
        return (newFn.execs.reduce((accum, curr) =>
    accum + curr, 0) / newFn.execs.length);
      };
      return newFn;
    }

  fetch = function() {
    let originalFetch = fetch;
    return async (p, o) => {
      try {
        let cachedResp = fetch.map[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.map[p] = resp;
          }
          return resp;
        } else {
          return cachedResp;
        }
      } catch(e){
        console.log(e)
      }
    }
  }();
  fetch.map = {};
```

```
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
window.location - alteraçao forca refresh
location.hash - alteraçao nao forca refresh
element = document.querySelector(selectors);
elementList = document.querySelectorAll(selectors);
element = document.getElementById(id);
element.addEventListener(type,listener, useCapture);
useCapture – true se quiser usar capturing, default false usa bubling
element.className – mudar a class
document.querySelector("input").disabled = true
eventos: change, click, copy, focus, hashchange,
input, keypress, mouseenter, mouseleave
onload - occurs when an object has been loaded, most often used within the
<body> element to execute a script once a web page has completely loaded
all content (including images, script files, CSS files, etc.)
<form id="searchGamesForm">
    <input type="text" name="gameName" placeholder="name" value="" />
    <input type="submit" />
</form>
input types: checkbox, email, hidden, image, password, number

event bubbling - eventos começam do elementos mais abaixo no DOM e
propagam-se para os elementos de cima
Event Capturing -  is the event starts from top element to target element.
event.stopPropagation(); - prevents further propagation of the current event
in the capturing and bubbling phases
```

```
array.slice() - copy
arrat.slice(init, end) - sub arrany
end index not included in array
array.find((element, index, array)=>{})
- retorna o objeto se exstir ou undefined
array.findIndex((element, index, array)=>{})
- retorna index da 1ª ocorrencia ou -1
array.indexOf(arg) - igual a findIndex
array.includes(arg, fromIndex) - devolve true se encontrar
array.map((element, index, array)=>{})
array.filter((element, index, array)=>{return boolean for no filtering})
array.reduce((accumulator, element, index, array)
reduce implementado manualmente:
function reduce(array, func, start)
{
    let current = start
    for(let element of array)
    {
        current = func(current, element)
    }
    return current
}
=>{accumulator+element}, valor inicial)
array.sort((a,b)=>{
                0 - iguais
                1 - a > b
                -1 - b < a
})
const array1 = [1, 2, 3];
const firstElement = array1.shift();
//Array [2, 3]
var arr = [1, 2];
arr.unshift(-2, -1); // = 5
// arr is [-2, -1, 0, 1, 2]
retorna comprimento de novo array
```

```
Safe methods - GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval.
These methods ought to be considered "safe". This allows user agents to represent other methods, such as POST,
PUT and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested.
Naturally, it is not possible to ensure that the server does not generate side-effects as a result of performing a GET request;
in fact, some dynamic resources consider that a feature. The important distinction here is that the user did not request
the side-effects, so therefore cannot be held accountable for them.
Idempotent Methods - Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of
N > 0 identical requests is the same as for a single request. The methods GET, HEAD, PUT and DELETE share this property. Also,
the methods OPTIONS and TRACE SHOULD NOT have side effects, and so are inherently idempotent.
```

```
addMusicToList: async function(playListId, musicId){
            let playList = await
yamaDb.getPlaylistById(playListId); //devolve a playList pelo ID
ou undefined se nao existir
            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)//adiciona uma musica a playlist
            } else {
                let err =  new Error("Cannot add
repeated music for this playlist");
                err.status = 500;
                throw err;
            }
}

function promisify(fn) {
  let oldFn = 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);
      oldFn.apply(null, args);
    });
  }
}

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;
    }
}
```

```
In the Node.js module system, each file is treated as a separate module.
Functions and objects are added to the root of a module by specifying additional properties on the
special exports object
The module.exports property can be assigned a new value (such as a function or object).
The core modules are defined within Node.js's source and are located in the lib/ folder.
If the NODE_PATH environment variable is set to a colon-delimited list of absolute paths, then Node.js
will search those paths for modules if they are not found elsewhere.
The module wrapper:
Before a module's code is executed, Node.js will wrap it with a function wrapper (below)
(function(exports, require, module, __filename, __dirname) {
// Module code actually lives in here
});
-It keeps top-level variables (defined with var, const or let) scoped to the module rather than the global
object.
-It helps to provide some global-looking variables that are actually specific to the module, such as:
The module and exports objects that the implementor can use to export values from the module.
The convenience variables __filename and __dirname, containing the module's absolute filename and
directory path.
```

```
Passport is authentication middleware for Node.js.
The verify callback invokes “done” to supply Passport with the user that
authenticated
be sure to use session() before passport.session() to ensure that the login
session is restored in the correct order, passport session poe o objeto do
user no request
let passportInitializer = (bcrypt, localStrategy, CiborgError) => {
    function initialize(passport, getUserById) {
        const authenticateUser = async(userId, password, done) => {
          try {
            const userData = await getUserById(userId);
            const user = userData.body;
            let isMatch = await bcrypt.compare(password, user.password);
            if (isMatch) {
              return done(null, user);
            } else {
              throw new CiborgError(null,
                'Error in passport initializer.',
                'Password Incorrect.',
                '500' // Internal Server Error
              );
            }
          } catch (err) {
            if (!(err instanceof CiborgError)) {
              err = new CiborgError(err,
                'Error in passport initializer.',
                'Unable to login.',
                '500' // Internal Server Error
              );
            }
            console.log(err)
            done(err);
          }
        };
        passport.use(new localStrategy({
          usernameField: "userId",
          passwordField: "password"
        }, authenticateUser));
        passport.serializeUser((user, done) => done(null, user.userId));
        passport.deserializeUser((id, done) => done(null, id));
    }
    return initialize;
}
```

1xx Informational response
100 Continue -everything so far is OK, the client can continue the request or ignore if it's finished.
101 Switching Protocol - indicates the protocol the server is switching to
102 Processing (WebDAV) - indicates the server received and is processing the request, but no response available yet.
103 Early Hints
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
203 Non-Authoritative - meta-information set is not exact set as available from the origin server but collected from a local or a third-party copy.
204 No Content - no content to send for this request
205 Reset
206 Partial
207 Multi-Status (WebDAV) - conveys information about multiple resources
208 Multi-Status (WebDAV)
226 IM Used (HTTP Delta encoding)
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 - has not been modified, so the client can use cached version
305 Use Proxy - deprecated
306 unused - deprecated
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
407 Proxy Authentication Required - This is similar to 401 but authentication is needed to be done by a proxy.
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
412 Precondition Failed
413 Payload Too Large - payload larger than limits defined by server
414 URI Too Long - URI longer than server allows
415 Unsupported Media Type - The media format is not supported by server
416 Requested Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot - joke
421 Misdirected Request
422 Unprocessable Entity (WebDAV)
423 Locked (WebDAV)
424 Failed Dependency (WebDAV)
425 Too Early
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
504 Gateway Timeout - server is acting as gateway and cannot get a response in time.
505 HTTP Version Not Supported
506 Variant Also Negotiates
507 Insufficient Storage
508 Loop Detected (WebDAV)
510 Not Extended
511 Network Authentication Required

Strict mode makes several changes to normal JavaScript semantics: ('use strict';)
-Eliminates some JavaScript silent errors by changing them to throw errors.
-Fixes mistakes that make it difficult for JavaScript engines to perform optimizations: strict mode code can sometimes be made to run faster than identical code that's not strict mode.
-Prohibits some syntax likely to be defined in future versions of ECMAScript.
-Changes simplifying eval and arguments, changes making it easier to write "secure" JavaScript
e.g.: error -> delete Object.prototype;

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.

-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.

```
configs:
server.use(session({ //express-session
   secret: props.session.secretKey,
   resave: false,
   saveUninitialized: false
}));
server.use(passport.initialize());
server.use(passport.session());
initialize: foram definidas estratégias de serializacao e desserializacao do user,
local strategy, authentication process
authMw: function(req, rsp, next) {
   if (req.isAuthenticated()) {
      return next();
   } else {
      res.redirect("/")
   }
}
```

```
ENV FILE
BAR=bar1
exports FOO=foo2
var env = require('node-env-file');
process.env.FOO = "defaultfoo";
```

```
var fs = require('fs');
fs.readFile('DATA', 'utf8', function(err, contents) {
   console.log(contents);
});
```

Middleware - functions that have access to the request object (req), the response object (res), and the next middleware
Passar erros -> next(err)
Apanhar erros do outro lado -> (err, req, res, next)

code executed by setTimeout() is called from an execution context separate from the function from which setTimeout was called.
Default this is global scope, set via apply or bind

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

```
describe('Service-groups tests:', function() {
   it('Should return list with 3 groups', function(done) {
      resP.then((res) => { //ou catch em caso de erro
         assert.equal(3, res.body.length);
         done();//tem de ser chamada apenas 1 vez
      });
   });
});
```

before - before all tests on describe
after - after all tests on describe
beforeEach - before each test on describe
afterEach - after each test on describe

```
new Promise(function(resolve, reject) {
   try {
      setTimeout(resolve, 100, 'foo');
   } catch() {
      reject(e);
   }
});
Promise.all([promise1, promise2, promise3]).then(function(values) {
 console.log(values);//array of values
});
```

States:
pending (pendente): Estado inicial, que não foi realizada nem rejeitada.
fulfilled (realizada): sucesso na operação.
rejected (rejeitado): falha na operação.
settled (estabelecida): Que foi realizada ou rejeitada.

```
construtor - new Function('a', 'b', 'return a + b');
[].slice.call(arguments) - por arguments em array
func.apply(thisArg [, argsArray]);
func.call([thisArg[, arg1, arg2, ...argN])
fixar o this de uma função - func.bind(thisArg)
```

```
Arrow function this:
function Point(x, y){
   this.x = x
   this.y = y

   this.add = function(p) {
      return new Point(this.x + p.x, this.y + p.y)
   }

   //this.add = p => new Point(this.x + p.x, this.y + p.y)
}
let p1 = new Point(5,4)
console.log(p1.add(p1))
//Point { x: 10, y: 8, add: [Function] }
let p2 = new Point(6,5)
console.log(p2.add(p2))
//Point { x: 12, y: 10, add: [Function] }
let f3 = p1.add
console.log(f3.call(p1,p2))
//Point { x: 11, y: 9, add: [Function] }
```

```
new Date()
new Date().getTime() - since 1970/01/01

Iterate obj props:
for(let p in obj) {
   console.log(`${p} : ${obj[p]}`)
}
Iterate array:
for(let element of array) {
   newArray.push(func(element))
}
deleting property: delete obj.prop
```

```
const express = require('express')
const app = express()
app.get('/games/:name', function (req, res) {
 res.send('Hello World')
})

app.get('/', mw1, mw2, rootHandler)
express.Router() - A Router instance is a complete middleware and routing system
const routerBundleApi = require('./web-api')(express.Router(),service)
app.use('/bundleApi', routerBundleApi)

module.exports = function(router ,service){
   router.get('/bundles', getBundles)
   router.get('/', rootHandler)
   return router;
   function rootHandler(req, res){
      res.send("RootHandler")
   }
   function getBundles(req, res){
      res.send("getBundles")
   }
}
express-session stores only a session identifier on the client within a cookie and stores the session data on the server

var bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.use('/', express.static('public'))
este middleware permite a pedidos ao servidor aceder a conteudos estaticos como imagens e outro ficheiros
http://localhost:3000/images/kitten.jpg
se dentro da pasta public extistir este ficheiro

resp.set('Set-Cookie', cookieName) - para os headers
res.status(code) - podes fazer status(200).end(), encadear
let queryParam = req.query.param;
let urlParam = req.params.param;
let body = req.body;
let headerValue = req.header('User-Agent')
req.baseUrl
req.hostname
req.method
app.listen(port, console.log("Running on port: " + port));
```

```
req.on('data', chunk => {
   body.push(chunk)
}).on('end', () => {
   body = Buffer.concat(body).toString();
   res.statusCode = 200;
   res.end(handler.call())
})
```

*handler – função que faz as cenas com o resultado do body

JavaScript is often described as a prototype-based language — to provide inheritance, objects can have a prototype object, which acts as a template object that it inherits methods and properties from. Each object has a private property which holds a link to another object called its prototype. That prototype object has a prototype of its own, and so on until an object is reached with null as its prototype. (o.[[Prototype]])
prototype is a reference to another object and contains common attributes/properties across all instances of the object
Arrow functions do not have a prototype property
Object.create(null) - criar objeto sem prototype
Object.create(Person) - criar objeto que herda do prototype Person
Object.prototype.constructor
Object.keys(obj)
Object.values(obj)

```
function Person(name, job, yearOfBirth){
   this.name= name;
   this.job= job;
   this.yearOfBirth= yearOfBirth;
}
Person.prototype.calculateAge= function(){
   console.log('The current age is: '+(2019- this.yearOfBirth));
}
let Person1= new Person('Jenni', 'clerk', 1986);
console.log(Person1) // there is no calculateAge in obj
let Person2= new Person('Madhu', 'Developer', 1997);
console.log(Person2) // there is no calculateAge in obj
Person1.calculateAge();
Person2.calculateAge();
```

object1.hasOwnProperty('property1') - whether the object has the specified property as its own property (as opposed to inheriting it).
Object.create(obj) method creates a new object, using an existing object as the prototype
Object.entries() - returns an array of a given object's own enumerable string-keyed property [key, value]
The Object.assign() method copies all enumerable own properties from one or more source objects to a target object

During an indexing operation, for example, it converts raw data, such as log files or message files, into internal documents and stores them in a basic data structure similar to a JSON object. equipped with a rich and powerful HTTP RESTful API that enables you to perform fast searches in near real time

```
GET /twitter/_search
{
   "query" : {
      "term" : { "user" : "kimchy" }
   }
}
{
                  //...
                  "hits":{
   "total" : {
      "value": 1,
      "relation": "eq"
   },
   "max_score": 1.3862944,
   "hits" : [
      {
         "_index" : "twitter",
         "_type" : "_doc",
         "_id" : "0",
         "_score": 1.3862944,
         "_source" : {
            "user" : "kimchy",
            "message": "trying out Elasticsearch",
            "date" : "2009-11-15T14:12:12",
            "likes" : 0
         }
      }
   ]
   }
}
GET twitter/_doc/0
{
   "_index" : "twitter",
   "_type" : "_doc",
   "_id" : "0",
   "_version" : 1,
   "_seq_no" : 10,
   "_primary_term" : 1,
   "found": true,
   "_source" : {
      "user" : "kimchy",
      "date" : "2009-11-15T14:12:12",
      "likes": 0,
      "message" : "trying out Elasticsearch"
   }
}
```