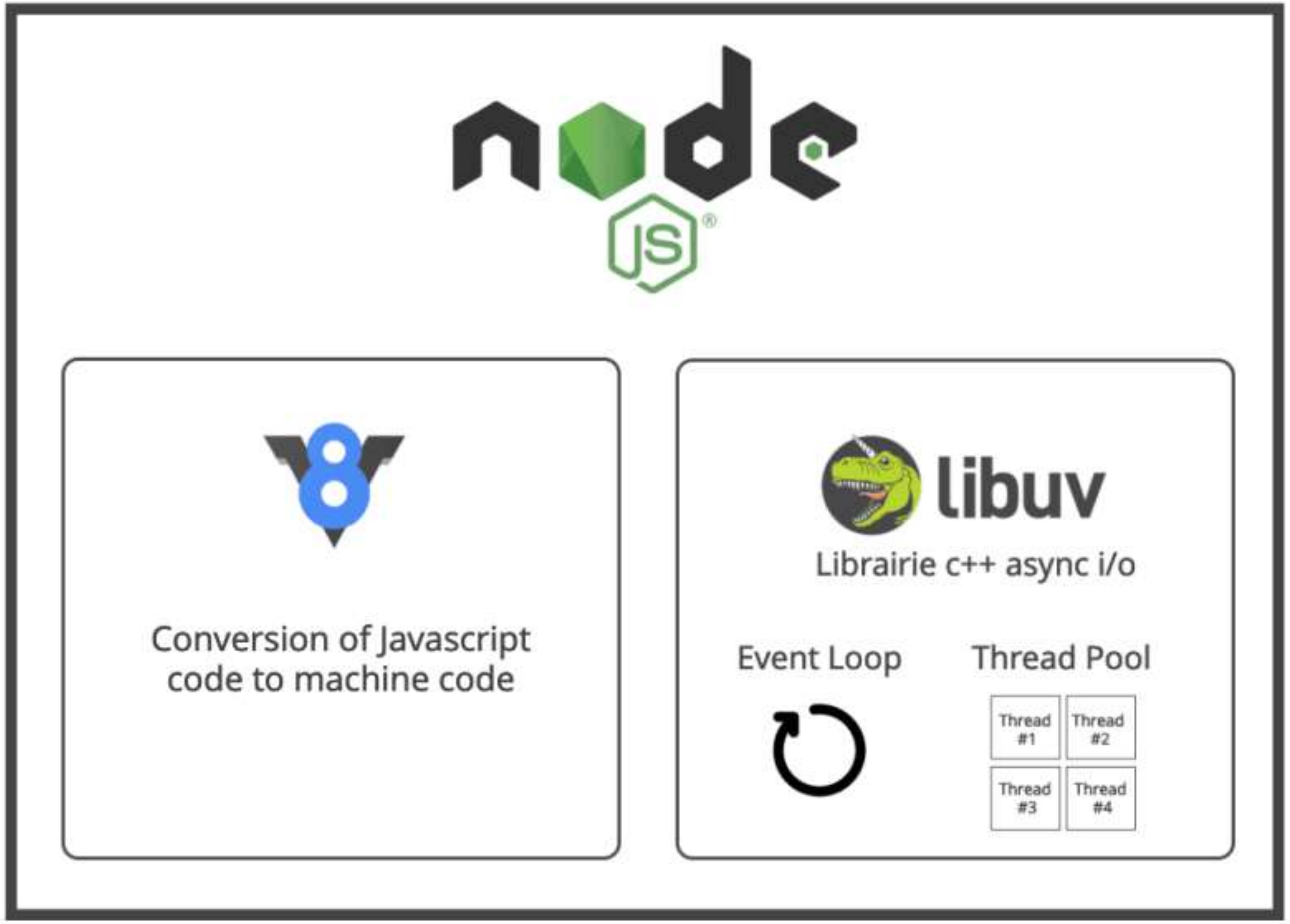
NODE.JS & DATABASES

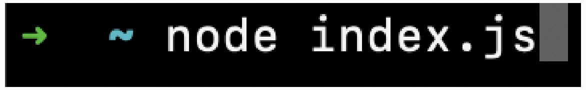
Node.js

Node.js® is an open-source, cross-platform JavaScript runtime environment.



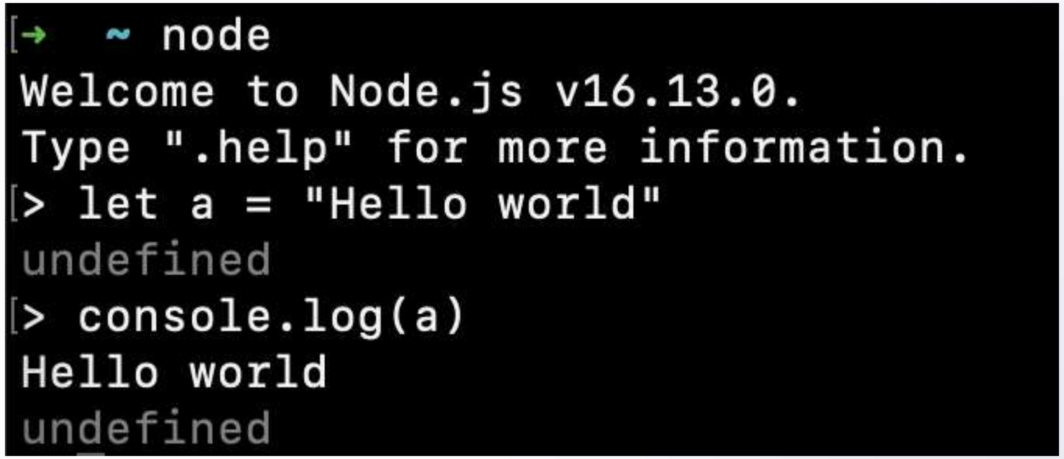
Node's CLI and REPL

Node CLI is installed with Node JS.
You can run it with node <fileName> command.
For example:



```
➜  ~ node index.js
```

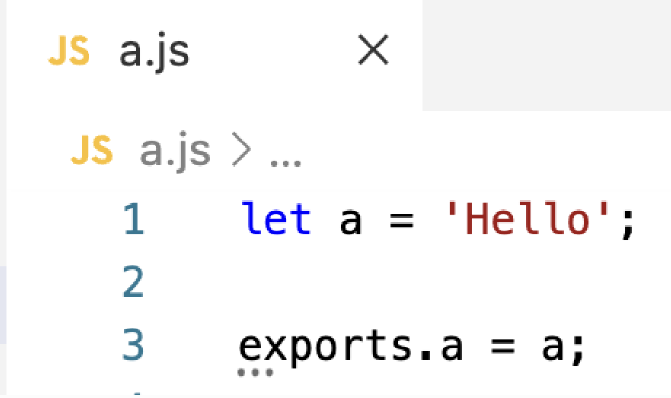
REPL stands for **read-eval-print loop**
It's very similar to browse's console.
To use REPL just type node in terminal and start use it.



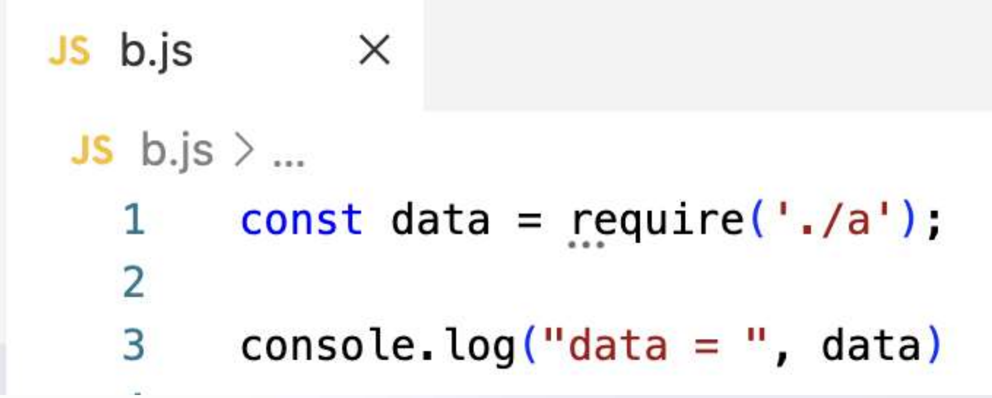
```
[➜  ~ node
Welcome to Node.js v16.13.0.
Type ".help" for more information.
[> let a = "Hello world"
undefined
[> console.log(a)
Hello world
undefined
```

Node's module system

In Node.JS files are called modules.
Let's assume we have 2 modules a.js and b.js and we want to import a.js into b.js



```
let a = 'Hello';

exports.a = a;
```



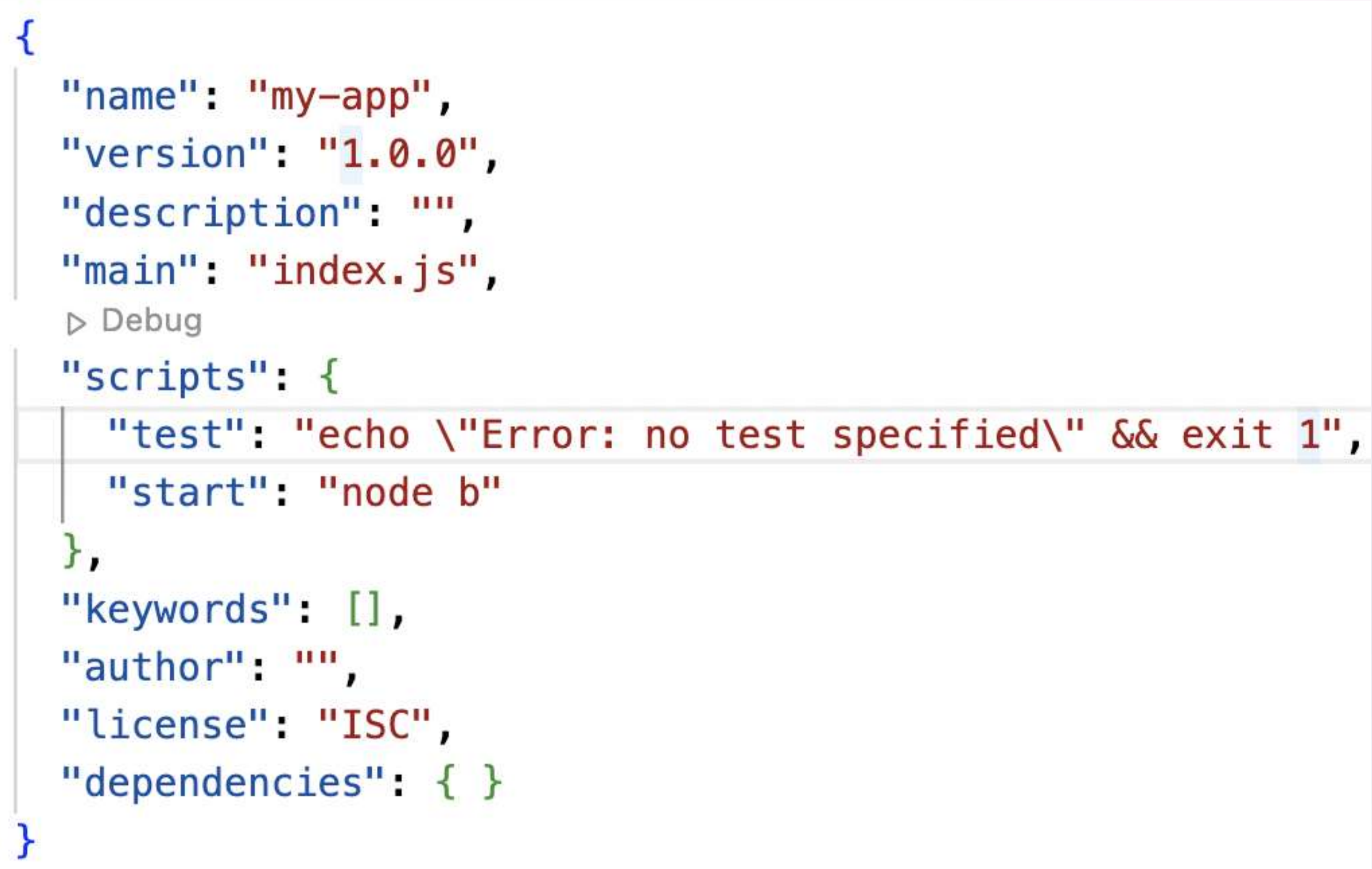
```
const data = require('./a');

console.log("data = ", data)
```

To export data we use **exports** keyword and to import file we use **require** function.
In require function, we pass the file name. If the file name doesn't start with ./ or / require will try to find that file in core modules, then in node_modules directory.

NPM

Node use NPM for package managers. NPM will automatically be installed with Node.

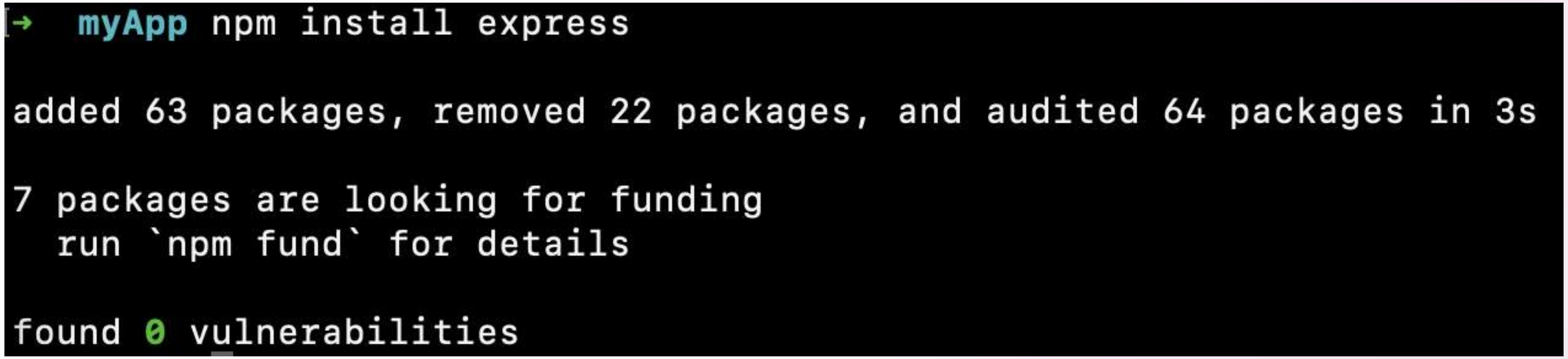
To start a new node project we must run the **npm init** command and then answer the questions. It will create package.json file which contains information about application.



```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node b"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": { }
}
```

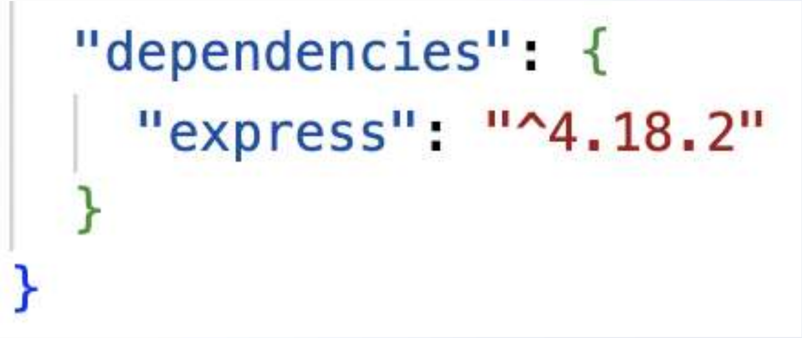
NPM packages

To install package we use install npm's command. For example let's install express package.



```
→  myApp npm install express

added 63 packages, removed 22 packages, and audited 64 packages in 3s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

This command will add a new package in the node_modules directory and add a new dependency in the package.json dependency section.



```
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

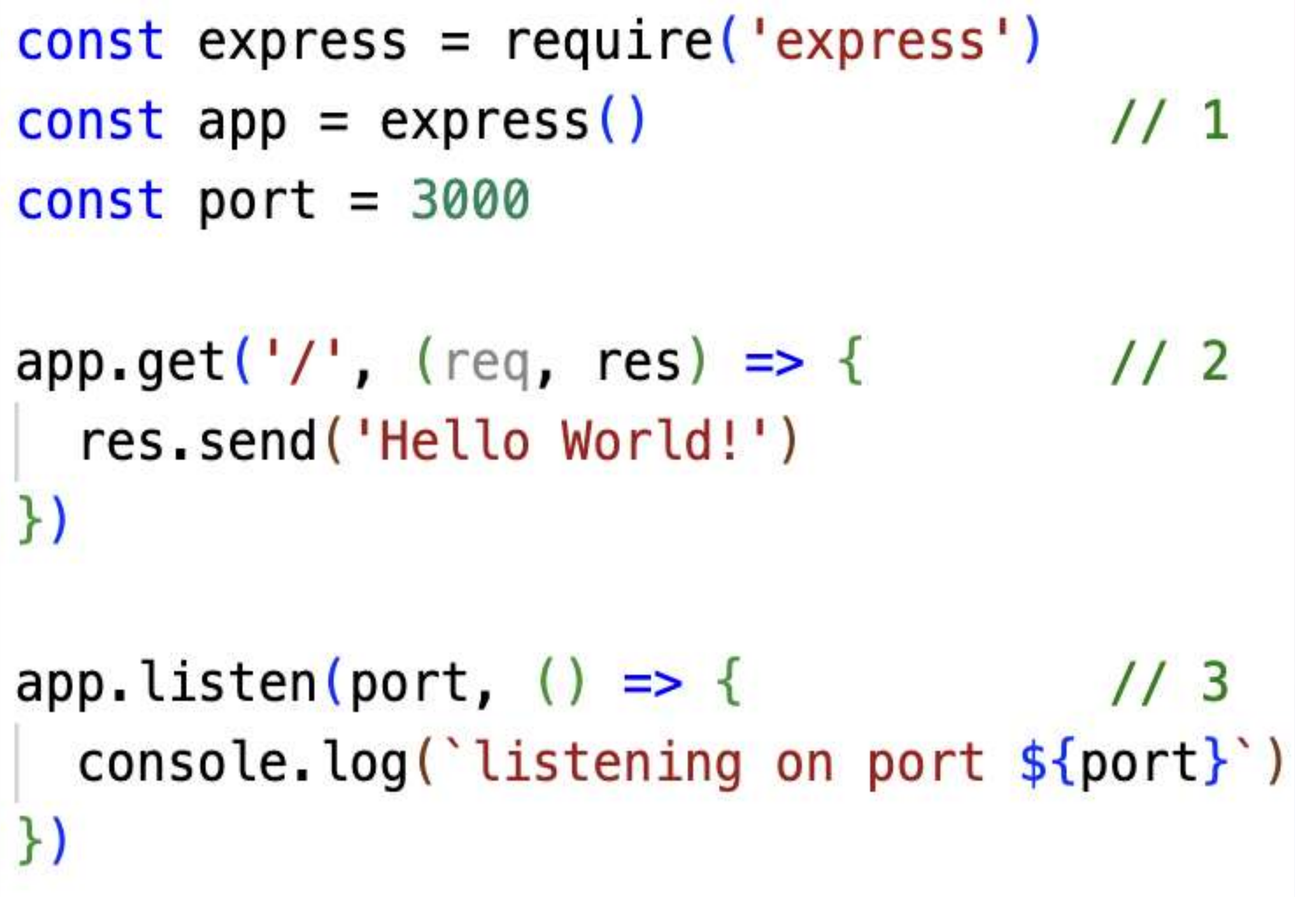
Express JS

- ExpressJS is a web framework for Node.js.
- The core concepts of the express are Routing and middlewares.

Simple Express APP

To start a basic Express app we just need to:

1. create an instance of the express app
2. set route and middleware,
3. start to listen to some ports.

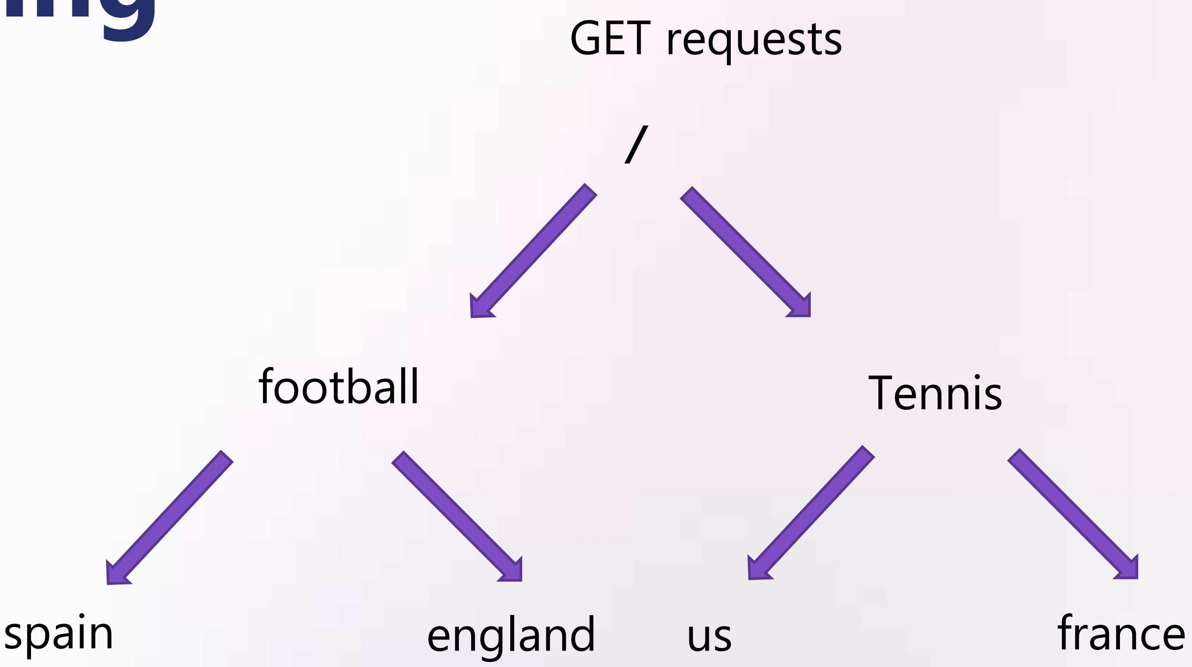


```
const express = require('express')
const app = express()                 // 1
const port = 3000

app.get('/', (req, res) => {          // 2
  res.send('Hello World!')
})

app.listen(port, () => {              // 3
  console.log(`listening on port ${port}`)
})
```

Routing

Routing lets us create a routing tree for each HTTP method that defines how requests should be handled.

To create a server with a routing tree from the picture we can write this kind of code:



GET requests

/

football

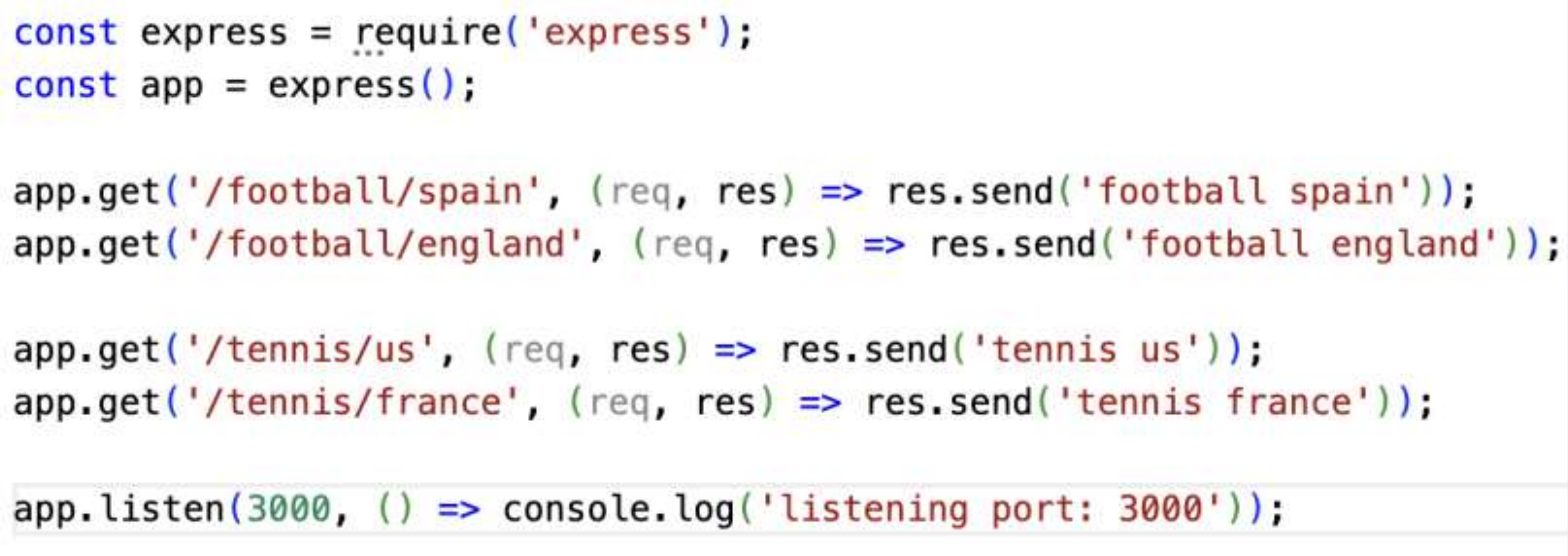
Tennis

spain

england

us

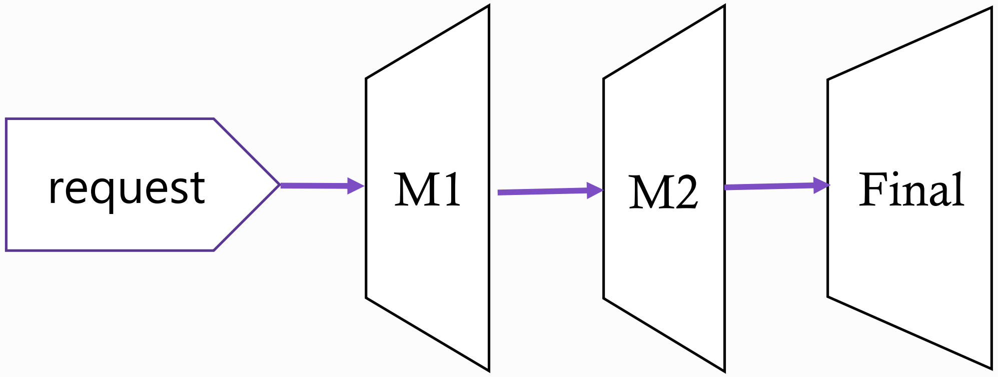
france



```
const express = require('express');
const app = express();

app.get('/football/spain', (req, res) => res.send('football spain'));
app.get('/football/england', (req, res) => res.send('football england'));

app.get('/tennis/us', (req, res) => res.send('tennis us'));
app.get('/tennis/france', (req, res) => res.send('tennis france'));

app.listen(3000, () => console.log('listening port: 3000'));
```

Middlewares

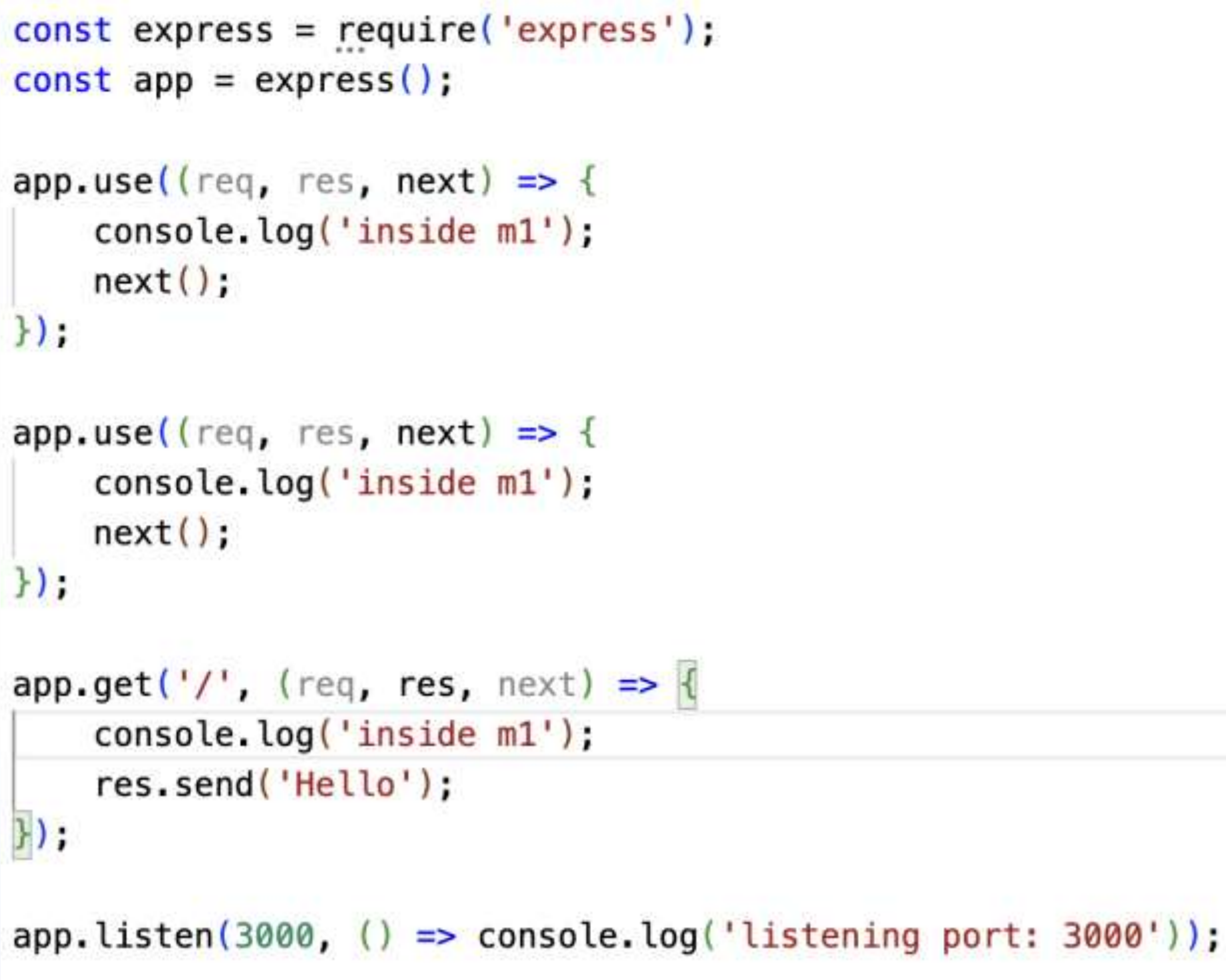
When the Express server receives a request, it will pass that request through a bunch of registered middlewares.



Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function.

If we forgot to call next function from m1, or m2 middleware, the next middleware will not execute and the request will hang up.

There is no need to call next function inside last middleware.



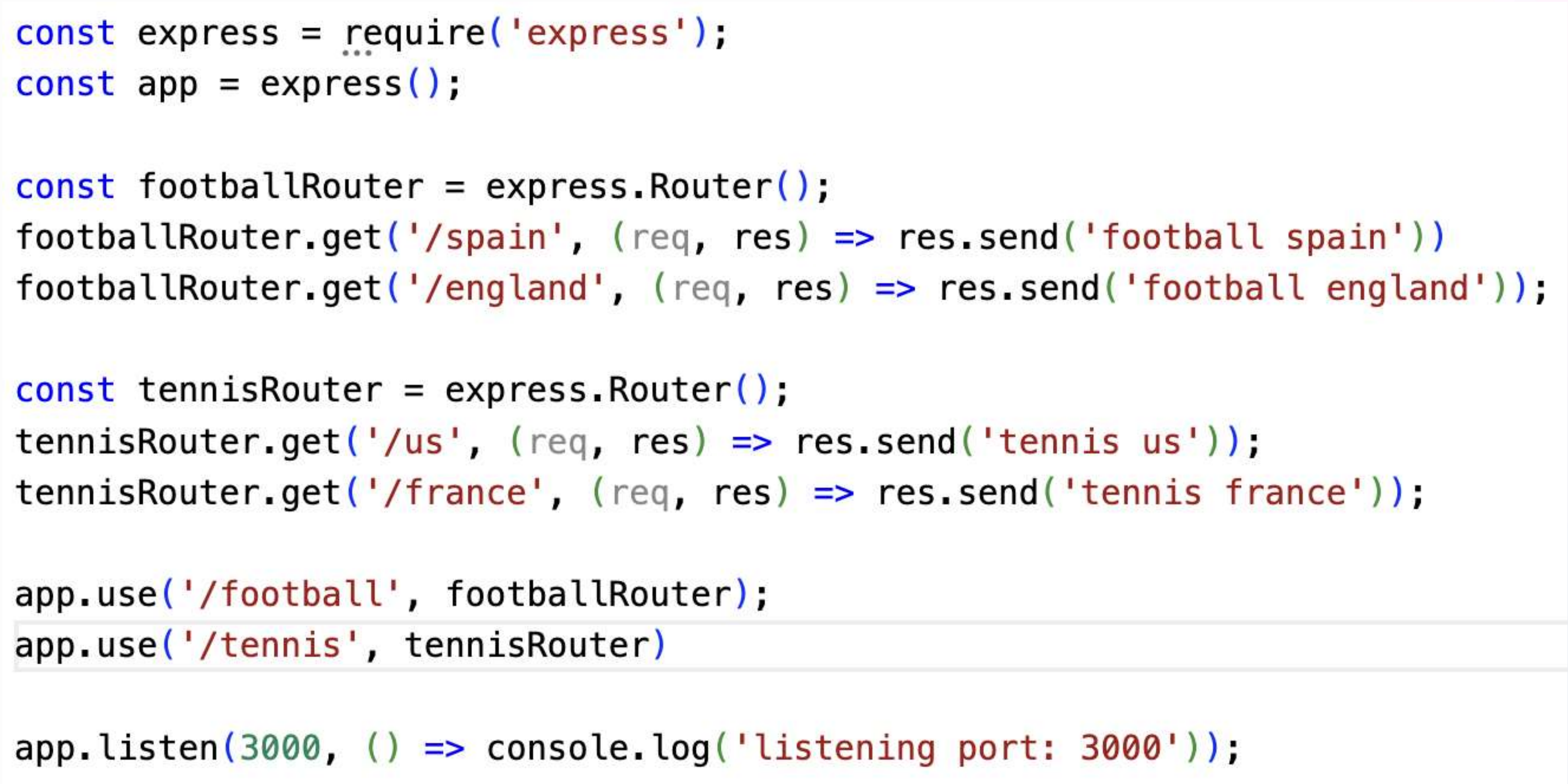
```
const express = require('express');
const app = express();

app.use((req, res, next) => {
    console.log('inside m1');
    next();
});

app.use((req, res, next) => {
    console.log('inside m1');
    next();
});

app.get('/', (req, res, next) => {
    console.log('inside m1');
    res.send('Hello');
});

app.listen(3000, () => console.log('listening port: 3000'));
```

sourcemind

Router

Express router can help us to logically group related routes.

Let's see how we can group routes that we created in the Routing section

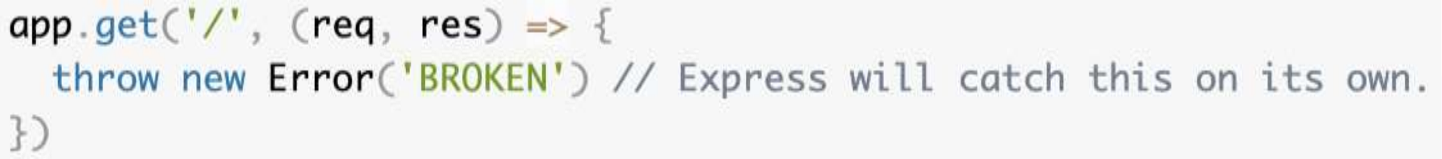


```
const express = require('express');
const app = express();

const footballRouter = express.Router();
footballRouter.get('/spain', (req, res) => res.send('football spain'))
footballRouter.get('/england', (req, res) => res.send('football england'));

const tennisRouter = express.Router();
tennisRouter.get('/us', (req, res) => res.send('tennis us'));
tennisRouter.get('/france', (req, res) => res.send('tennis france'));

app.use('/football', footballRouter);
app.use('/tennis', tennisRouter)

app.listen(3000, () => console.log('listening port: 3000'));
```

Error handling

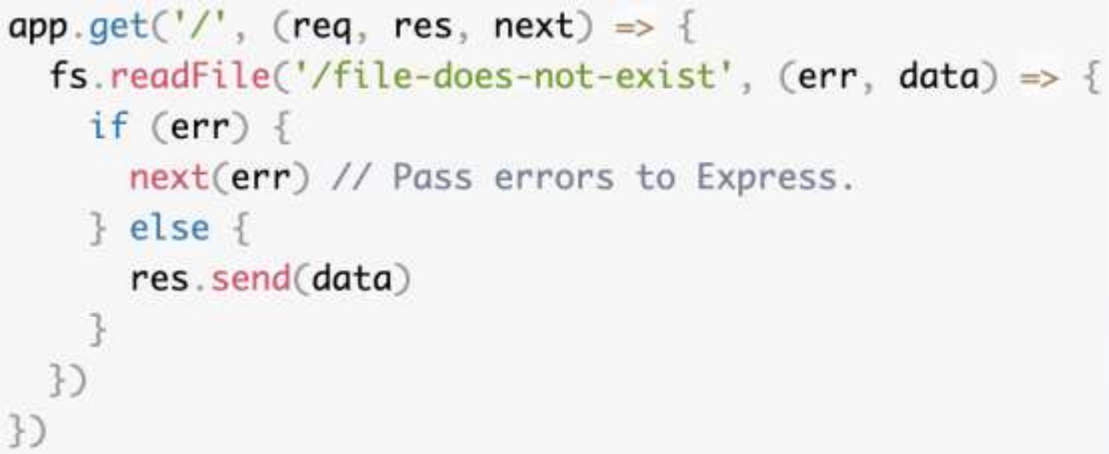
Error Handling refers to how Express catches and processes errors that occur both synchronously and asynchronously.

Express comes with a default error handler so you don’t need to write your own to get started.
Errors that occur in synchronous code inside route handlers and middleware require no extra work.



```
app.get('/', (req, res) => {
  throw new Error('BROKEN') // Express will catch this on its own.
})
```

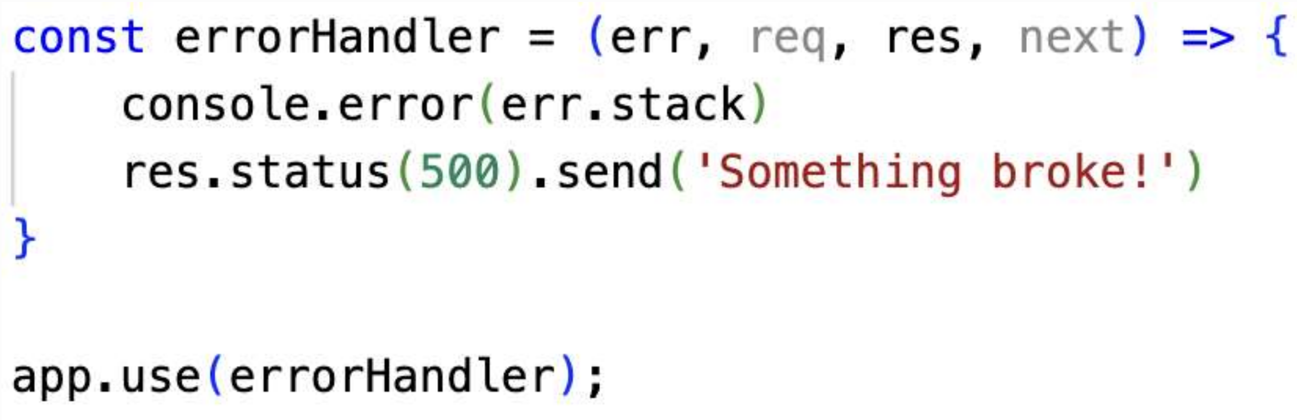
We must pass errors returned from asynchronous functions to the next() function



```
app.get('/', (req, res, next) => {
  fs.readFile('/file-does-not-exist', (err, data) => {
    if (err) {
      next(err) // Pass errors to Express.
    } else {
      res.send(data)
    }
  })
})
```

Writing error handlers

Define error-handling middleware functions in the same way as other middleware functions, except error-handling function have four arguments instead of three: (err, req, res, next).

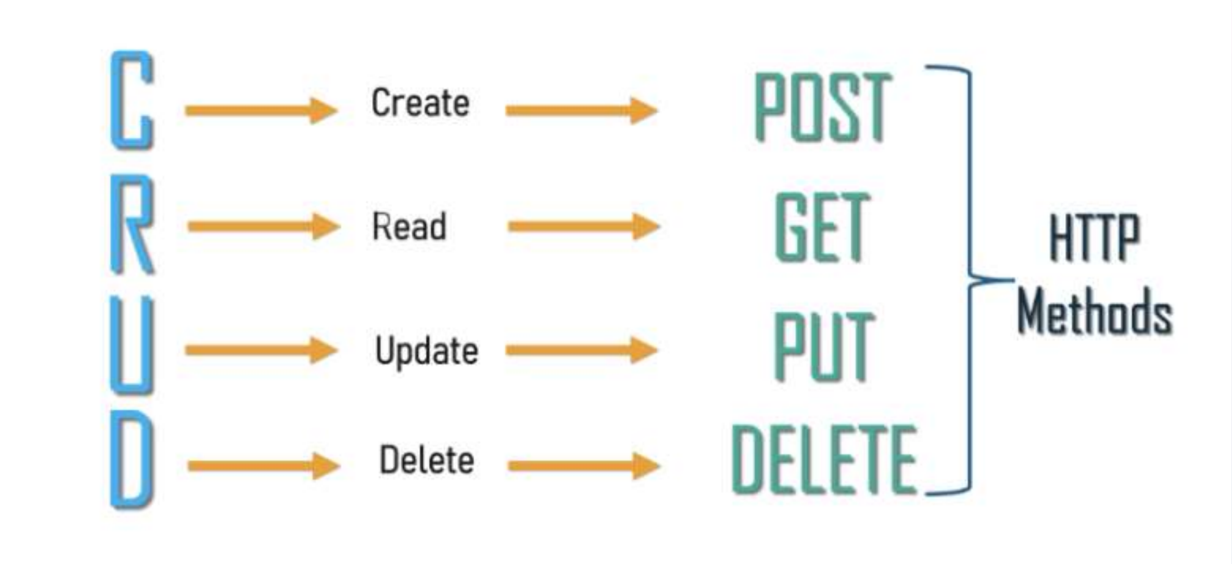


```
const errorHandler = (err, req, res, next) => {
    console.error(err.stack)
    res.status(500).send('Something broke!')
}

app.use(errorHandler);
```

Error handling middleware must be the last middleware

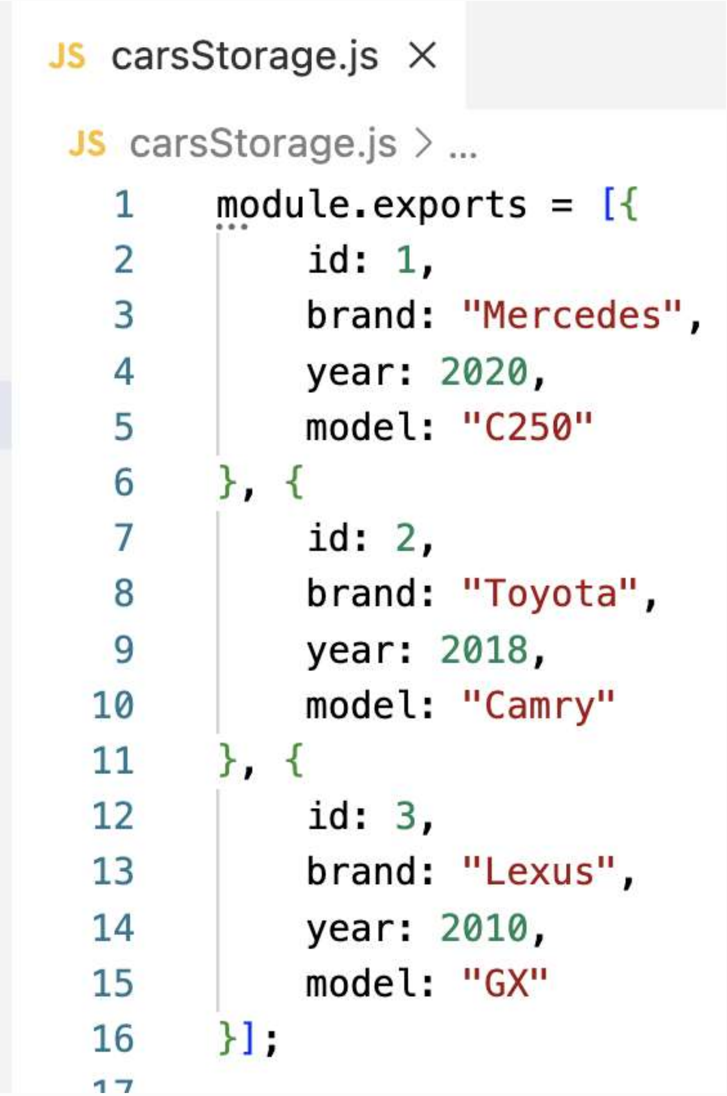
CRUD

CRUD stands for create, read update delete.
Suppose we have a server and in that server, we store information about cars. The server must provide CRUD API for its clients that can be used for creating, reading, updating and deleting cars.
Each CRUD action has a corresponding HTTP method.

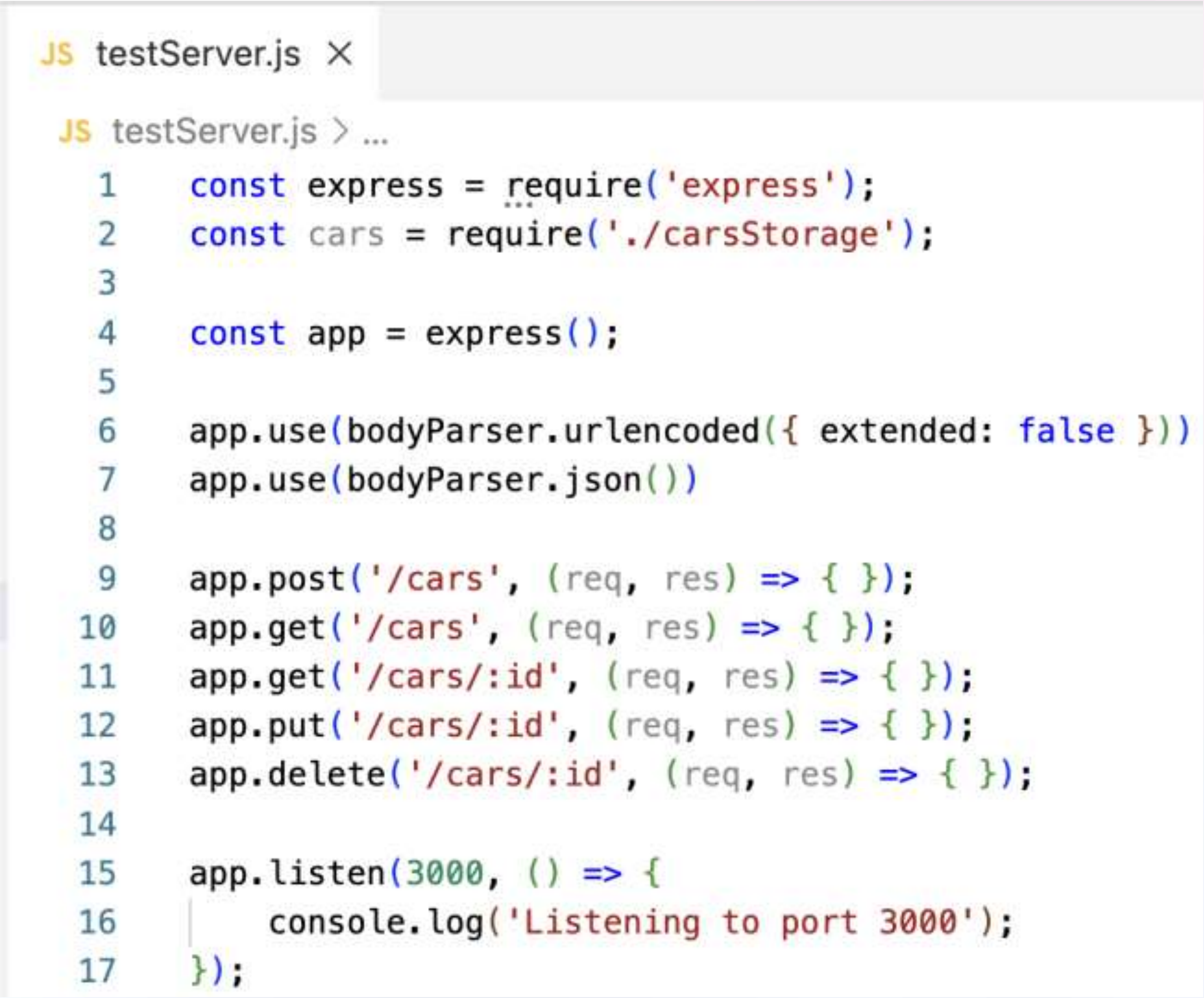


sourcemind

Implementing CRUD API



```
module.exports = [{
    id: 1,
    brand: "Mercedes",
    year: 2020,
    model: "C250"
}, {
    id: 2,
    brand: "Toyota",
    year: 2018,
    model: "Camry"
}, {
    id: 3,
    brand: "Lexus",
    year: 2010,
    model: "GX"
}];
```



```
const express = require('express');
const cars = require('./carsStorage');

const app = express();

app.use(bodyParser.urlencoded({ extended: false }))
app.use(bodyParser.json())

app.post('/cars', (req, res) => { });
app.get('/cars', (req, res) => { });
app.get('/cars/:id', (req, res) => { });
app.put('/cars/:id', (req, res) => { });
app.delete('/cars/:id', (req, res) => { });

app.listen(3000, () => {
    console.log('Listening to port 3000');
});
```

sourcemind

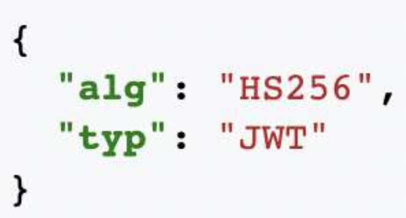
JWT

JSON web token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

typical look like these:
eyJhbGciOiJIUzI1NiIsInR.TcwMDUsImV4cCI6MTY2ODYxNzEyNX0.z0IkJbYv2DPmsxLozDG_I5RkocLkVp7Tw9nT7BT8v1M

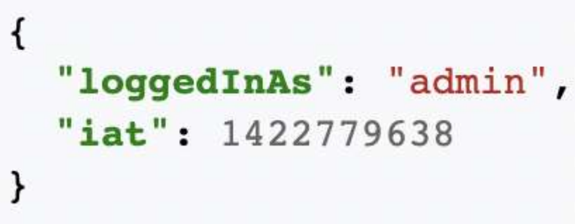
JWT has 3 parts separated by `.` (dot)

Header - Identifies which algorithm is used to generate the signature.



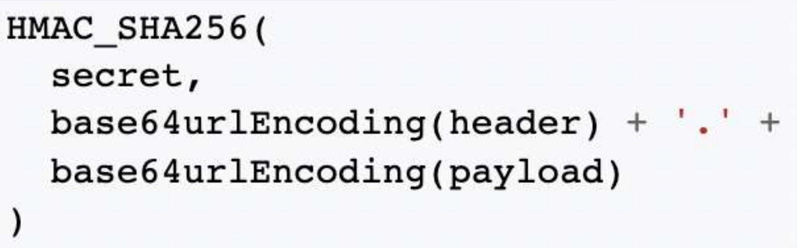
```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload - Contains a set of claims. The JWT specification defines seven Registered Claim Names. We can also add custom claims.



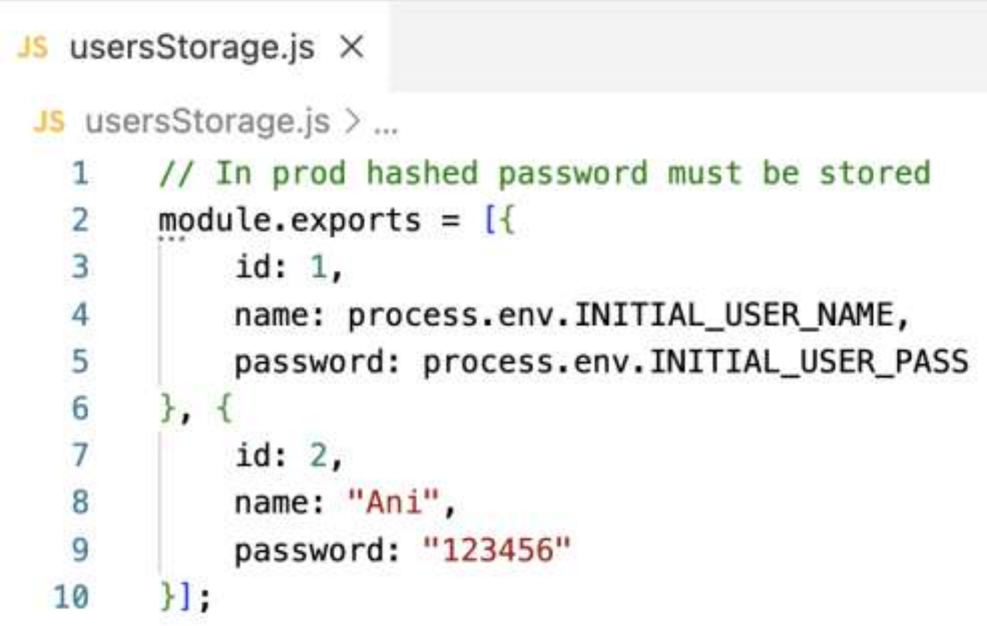
```
{
  "loggedInAs": "admin",
  "iat": 1422779638
}
```

Signature - The signature is calculated by encoding the header and payload using Base64url Encoding RFC 4648 and concatenating the two together with a period separator. That string is then run through the cryptographic algorithm specified in the header.

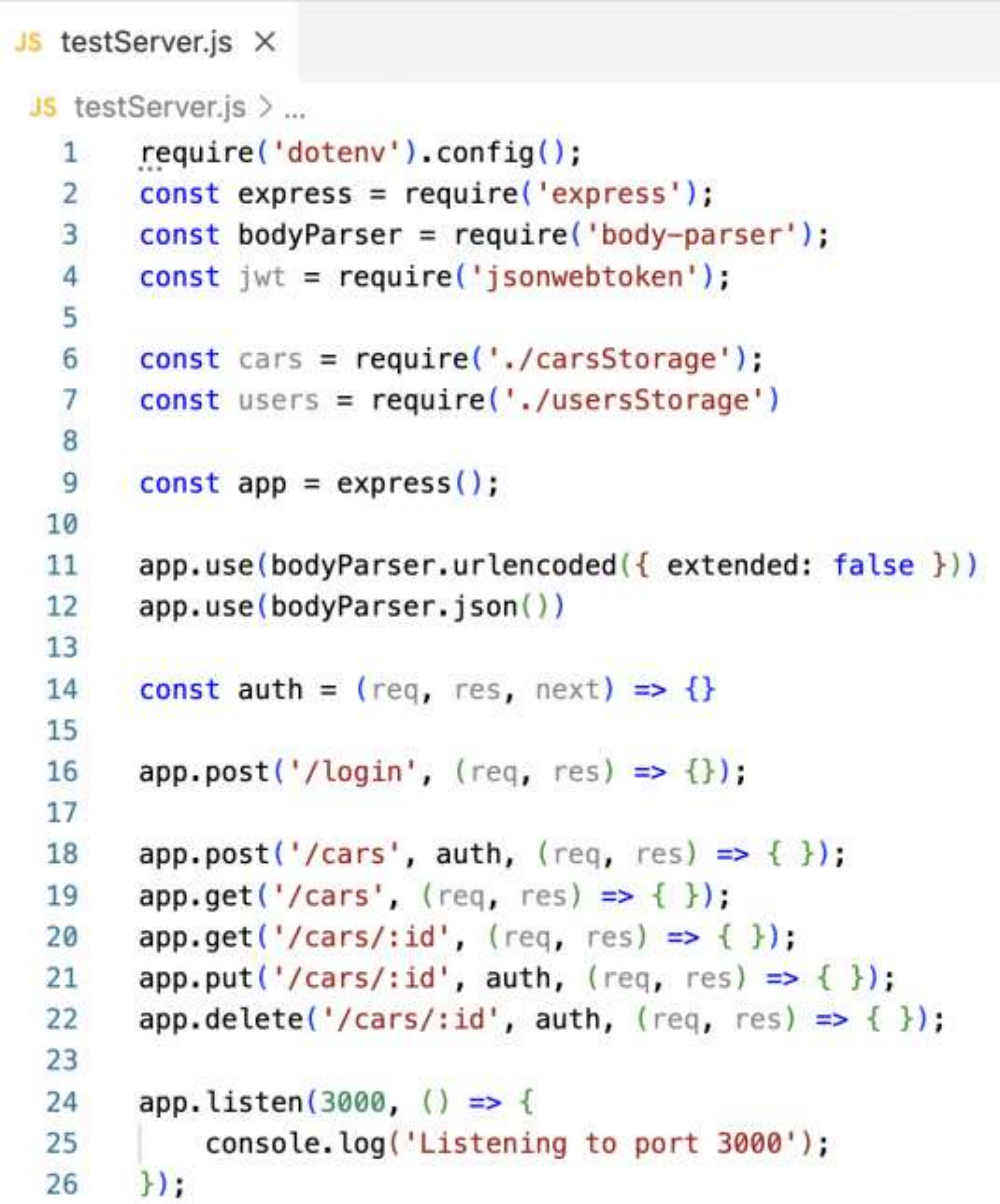


```
HMAC_SHA256(
  secret,
  base64urlEncoding(header) + '.' +
  base64urlEncoding(payload)
)
```

Authorization with JWT



```javascript
// In prod hashed password must be stored
module.exports = [{
    id: 1,
    name: process.env.INITIAL_USER_NAME,
    password: process.env.INITIAL_USER_PASS
}, {
    id: 2,
    name: "Ani",
    password: "123456"
}];
```



```javascript
require('dotenv').config();
const express = require('express');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');

const cars = require('./carsStorage');
const users = require('./usersStorage')

const app = express();

app.use(bodyParser.urlencoded({ extended: false }))
app.use(bodyParser.json())

const auth = (req, res, next) => {}

app.post('/login', (req, res) => {});

app.post('/cars', auth, (req, res) => { });
app.get('/cars', (req, res) => { });
app.get('/cars/:id', (req, res) => { });
app.put('/cars/:id', auth, (req, res) => { });
app.delete('/cars/:id', auth, (req, res) => { });

app.listen(3000, () => {
    console.log('Listening to port 3000');
});
```