Postman

Overview

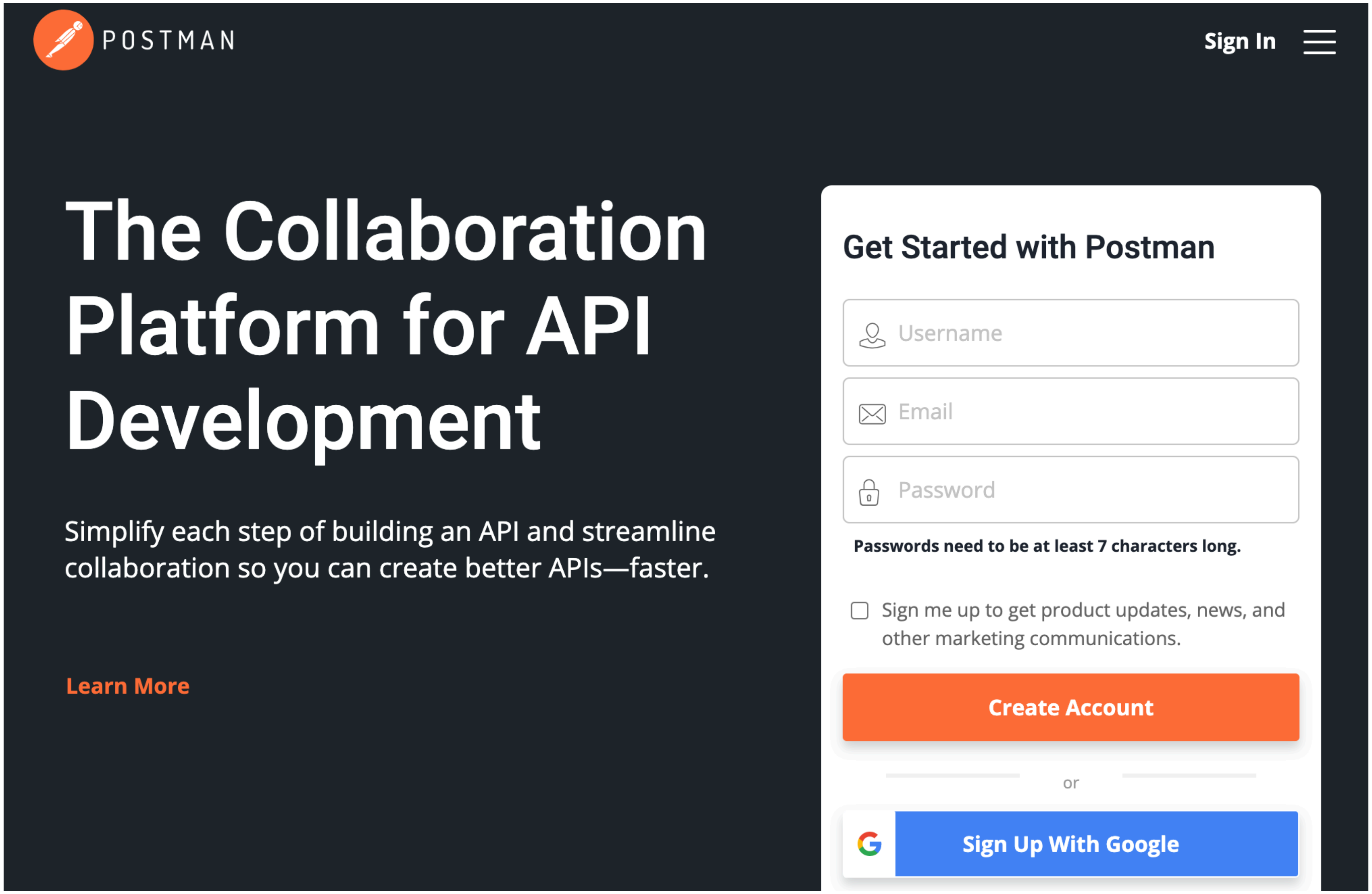
- Cosa è?
  - Uno programma per Server side testing

- Automatizza delle richieste HTTP

A cosa serve?

- Development
- Test
- Share
- Document

Account



The Collaboration Platform for API Development

Simplify each step of building an API and streamline collaboration so you can create better APIs—faster.

Learn More

Get Started with Postman

Username

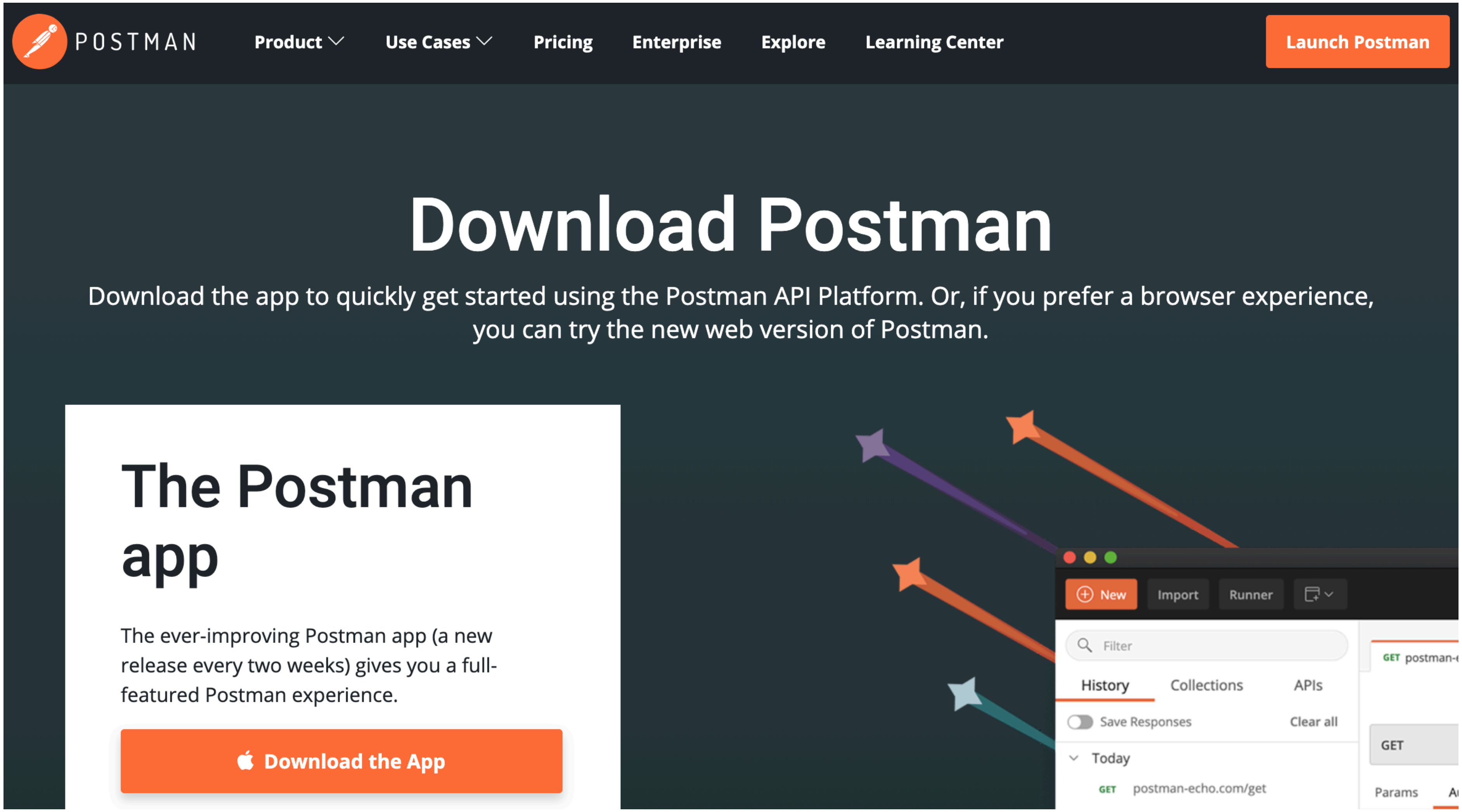
Email

Password

Passwords need to be at least 7 characters long.

☐ Sign me up to get product updates, news, and other marketing communications.

Install



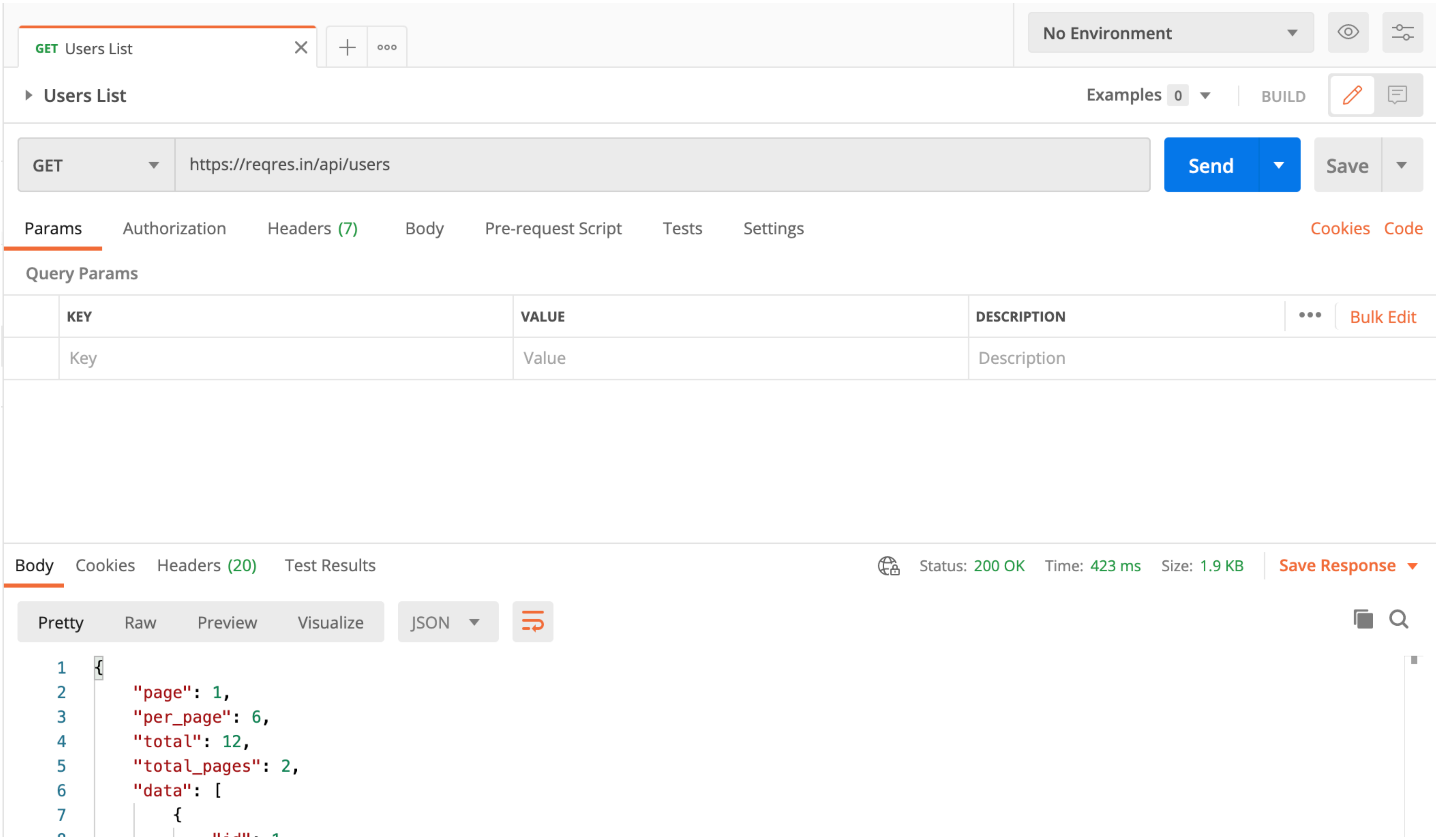
Download Postman

Download the app to quickly get started using the Postman API Platform. Or, if you prefer a browser experience, you can try the new web version of Postman.

The Postman app

The ever-improving Postman app (a new release every two weeks) gives you a full-featured Postman experience.

Richieste



GET https://reqres.in/api/users

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 423 ms Size: 1.9 KB

```
{
    "page": 1,
    "per_page": 6,
    "total": 12,
    "total_pages": 2,
    "data": [
        {
```

Main Features

- Collections
  - group of saved requests you can organize into folders
- Workspace
  - organize and collaborate on API projects
- Environment
  - a set of variables
- Script
  - JavaScript code executed before and after the req/resp

Express

Overview

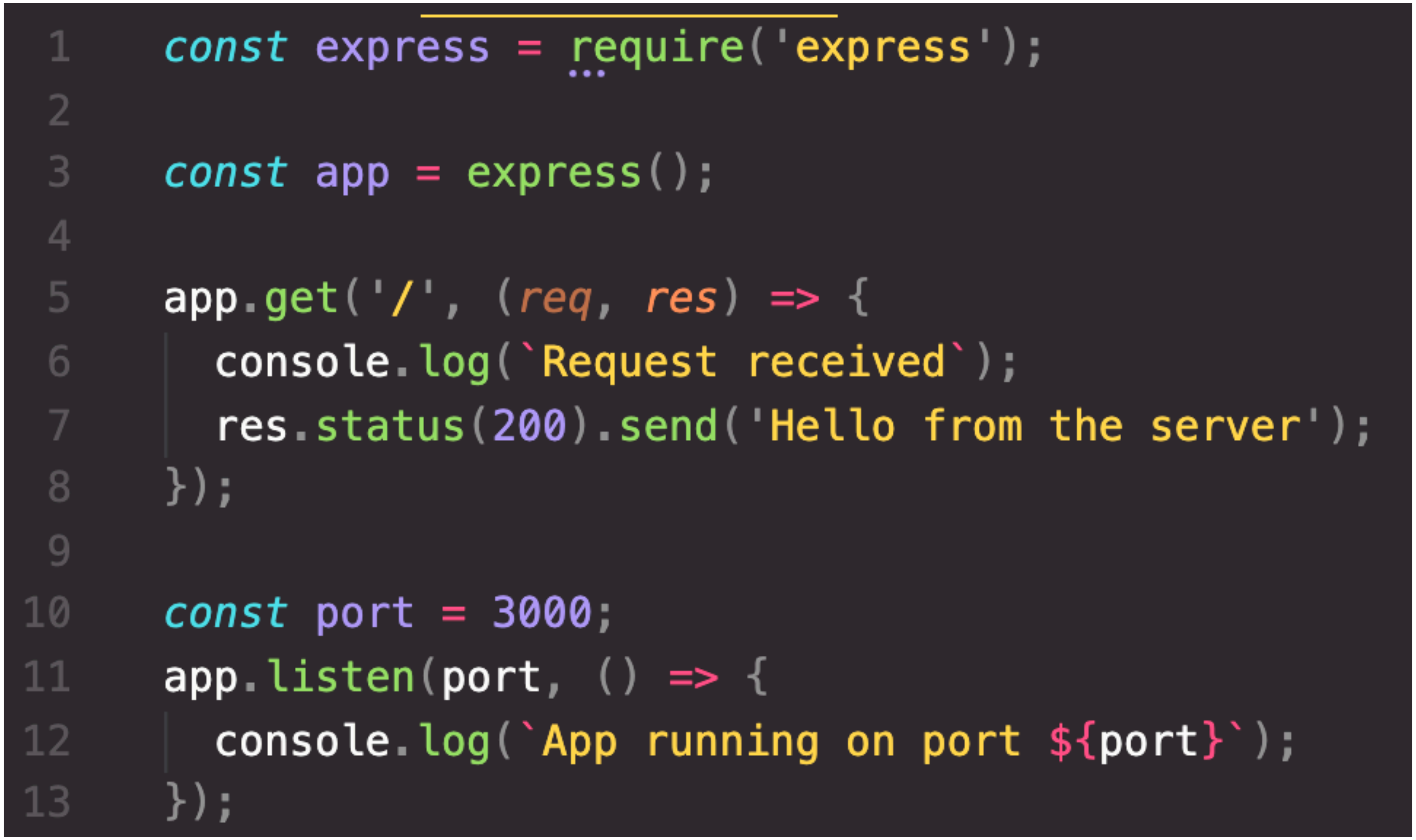
- Minimal node js framework

- Features
  - complex routing
  - req/resp handling
  - middleware
  - server side rendering

- Rapid app development
- MVC Architecture

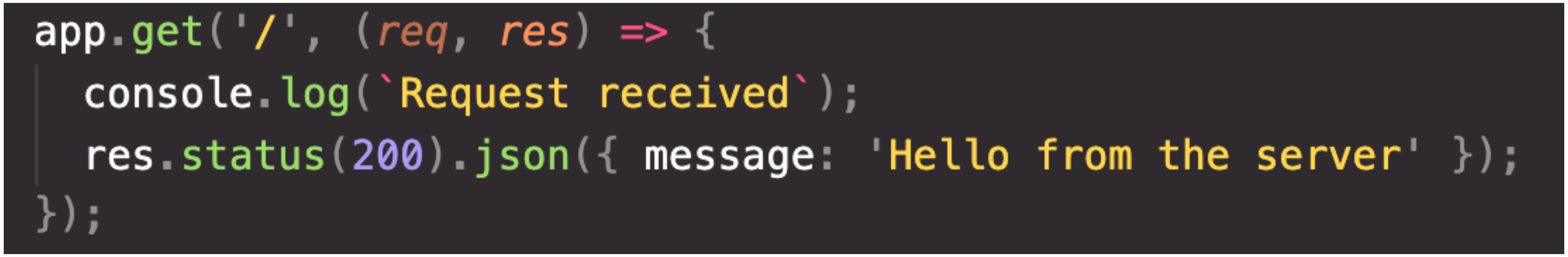
Università di Roma Tor Vergata

basic app



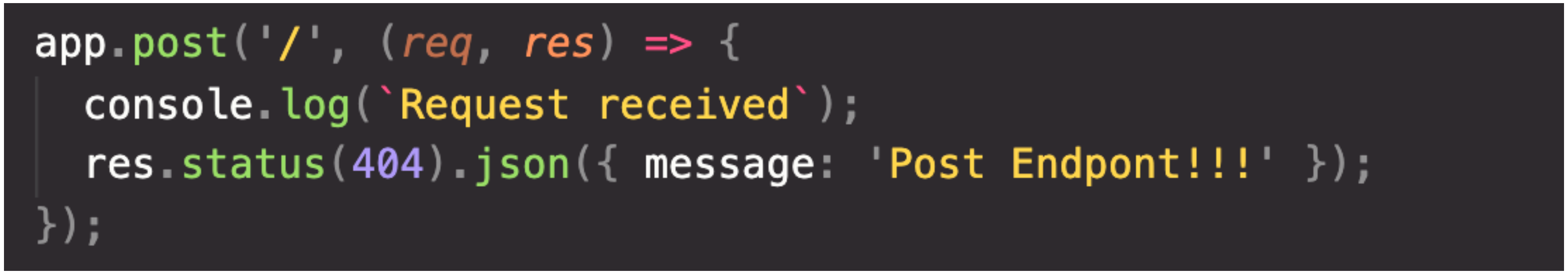
```
const express = require('express');

const app = express();

app.get('/', (req, res) => {
  console.log(`Request received`);
  res.status(200).send('Hello from the server');
});

const port = 3000;
app.listen(port, () => {
  console.log(`App running on port ${port}`);
});
```

json response



```
app.get('/', (req, res) => {
  console.log(`Request received`);
  res.status(200).json({ message: 'Hello from the server' });
});
```

Simple post



```
app.post('/', (req, res) => {
  console.log(`Request received`);
  res.status(404).json({ message: 'Post Endpont!!!' });
});
```

Architettura REST

- **REST** (*RE*presentational *S*tate *T*ransfer)

- insieme di linee guida o principi per la realizzazione di una architettura di sistema
  - uno stile architetturale

- non si riferisce ad un sistema concreto
- non si tratta di uno standard

Principi REST

- Identificazione delle risorse
- Utilizzo esplicito dei metodi HTTP
- Risorse autodescrittive
- Collegamenti tra risorse
- Comunicazione senza stato

Esempio risorse

`products` `users` `orders`

```
http://my-url/addNewProduct
             /getProduct
             /updateProduct
             /deleteProduct

             /getProductbyOrder
             /getOrderbyUser
```

Esempio risorse

`products` `users` `orders`

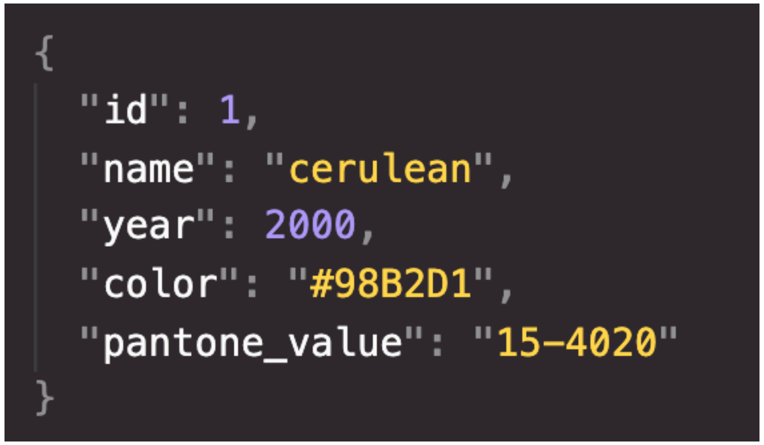
```
http://my-url/addNewProduct
             /getProduct
             /updateProduct
             /deleteProduct


             /getProductbyOrder
             /getOrderbyUser
```

CRUD Operations

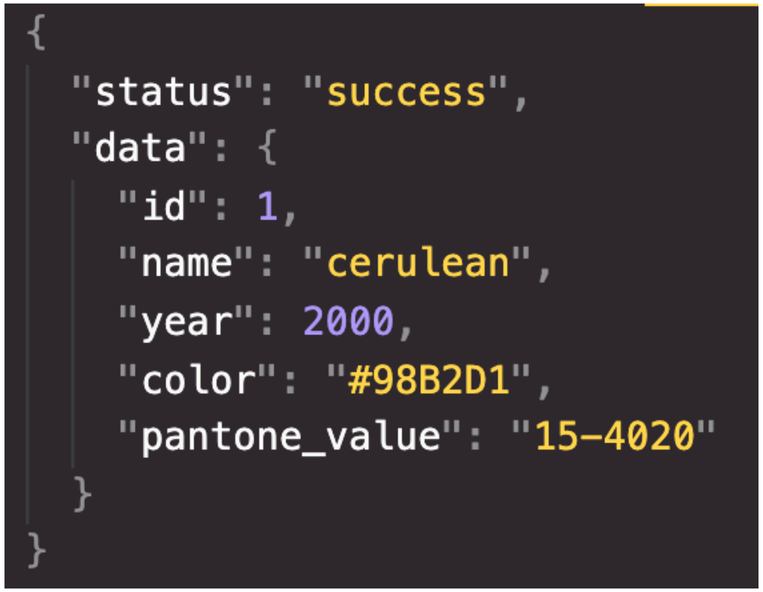
`/addNewProduct`	`POST /products`
`/getProduct`	`GET /products/3`
`/updateProduct`	`PUT /products/3`
	`PATCH /products/3`
`/deleteProduct`	`DELETE /products/3`
`/getProductbyOrder`	`GET /orders/4/products`
`/getOrderbyUser`	`GET /users/9/orders`

JSON formatting

JSEND
https://github.com/omniti-labs/jsend



```
{
  "id": 1,
  "name": "cerulean",
  "year": 2000,
  "color": "#98B2D1",
  "pantone_value": "15-4020"
}
```



```
{
  "status": "success",
  "data": {
    "id": 1,
    "name": "cerulean",
    "year": 2000,
    "color": "#98B2D1",
    "pantone_value": "15-4020"
  }
}
```

1. JSON API - JSON API covers creating and updating resources as well, not just responses.
2. JSend - Simple and probably what you are already doing.
3. OData JSON Protocol - Very complicated.

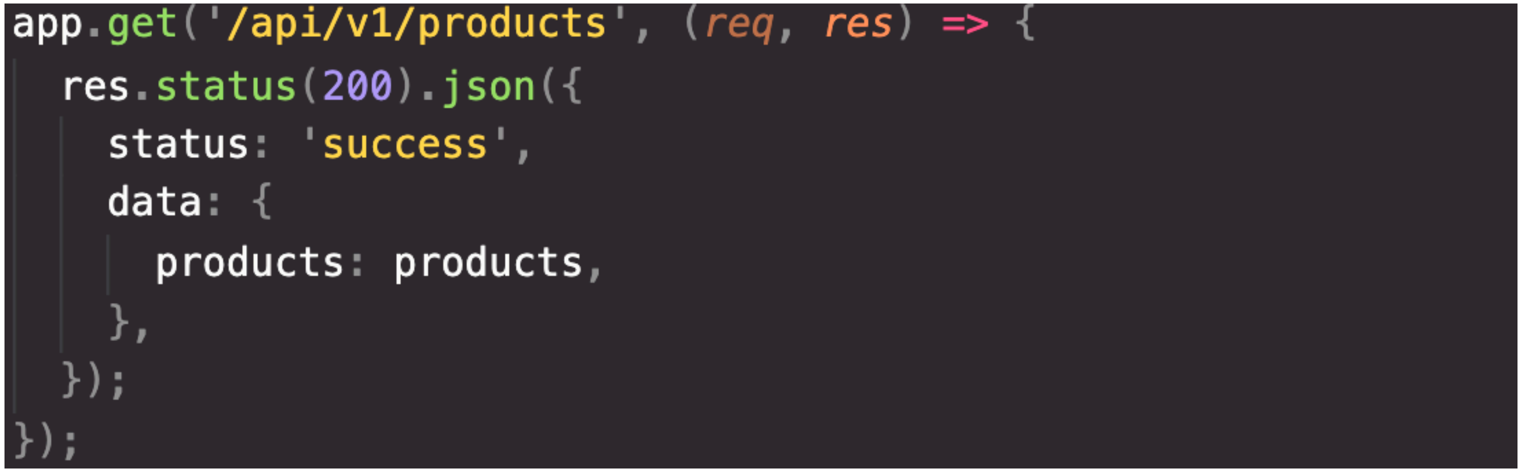
Stateless!!!

- Lo stato va manteniuto nel client
  - Il server per rispondere non deve ricordare una richiesta precedente

- Esempio
  - paging:
    - https://reqres.in/api/users?page=1
    - ~~https://reqres.in/api/users?page=nextpage~~
  - login
    - ogni richiesta è autenticata singolarmente

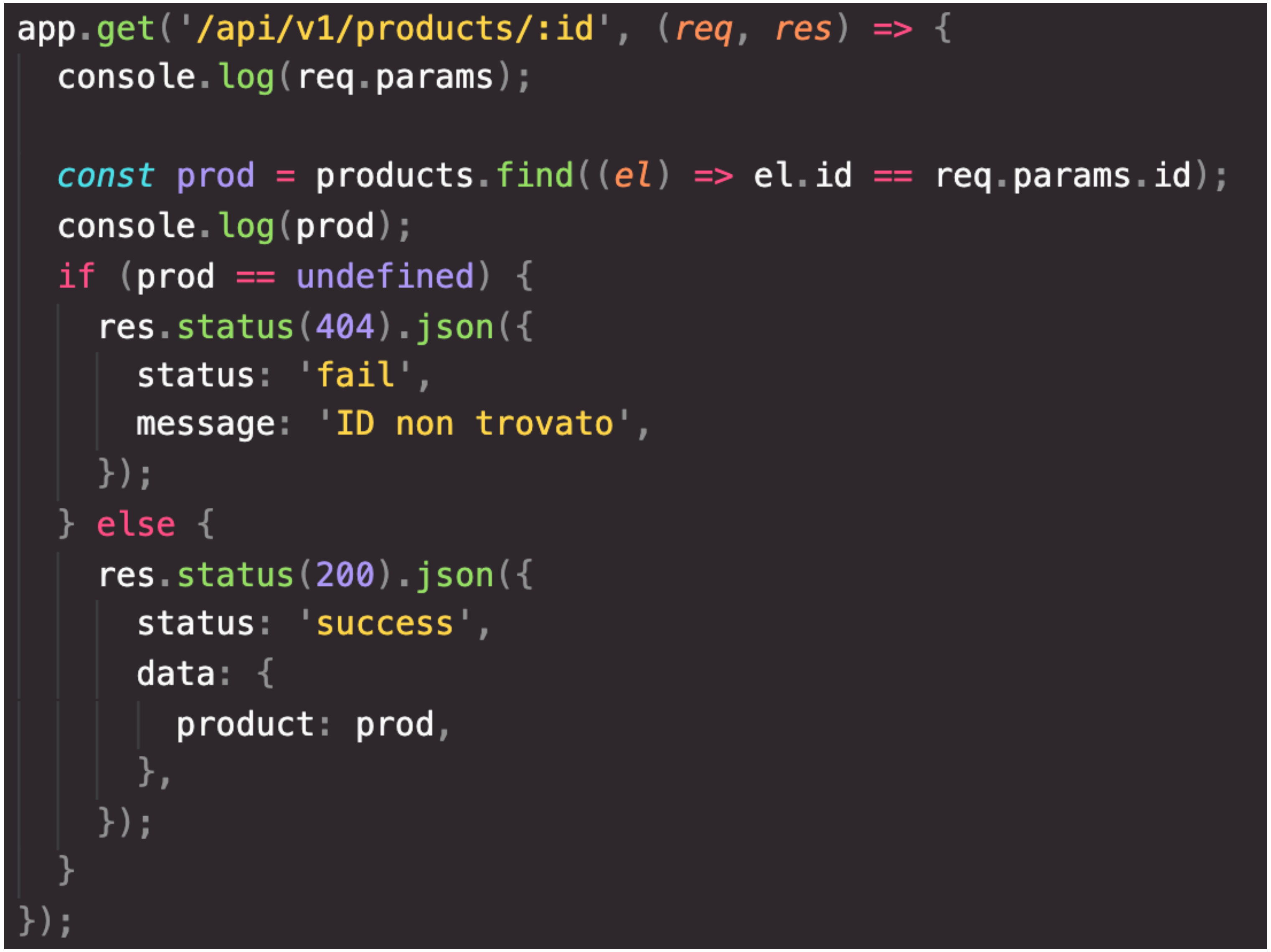
CRUD API

GET



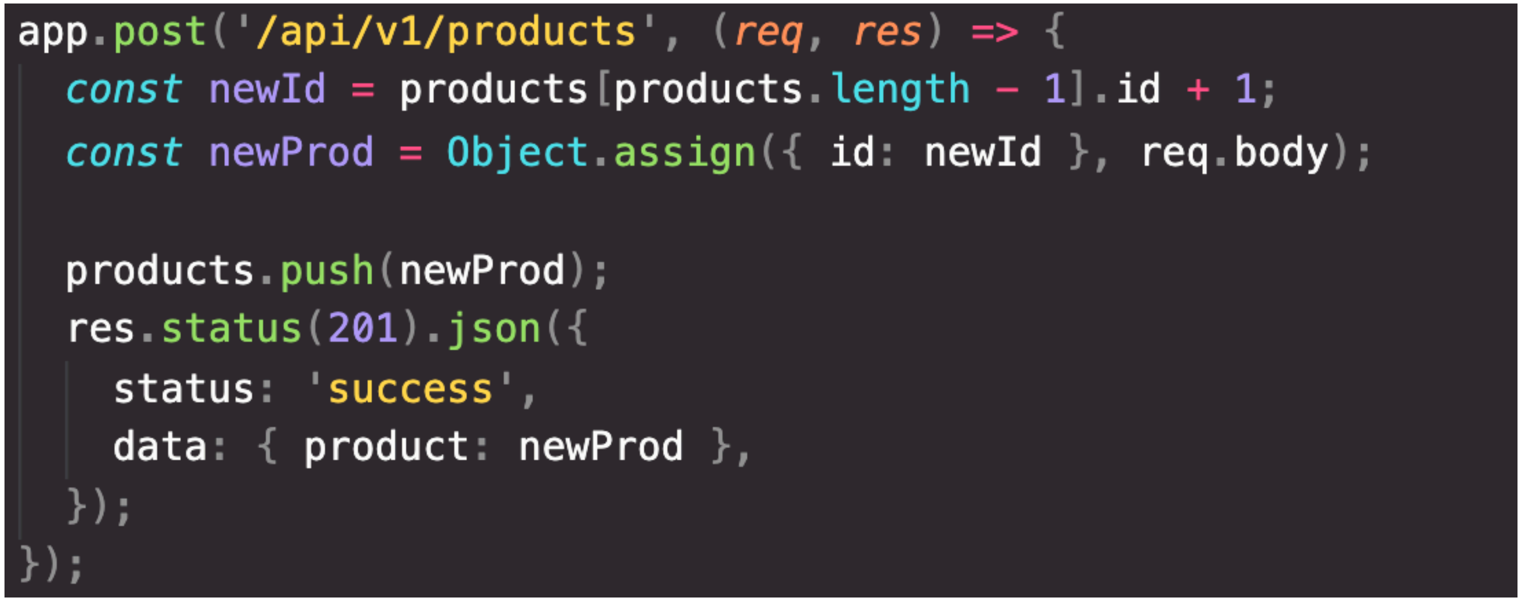
```
app.get('/api/v1/products', (req, res) => {
  res.status(200).json({
    status: 'success',
    data: {
      products: products,
    },
  });
});
```

GET



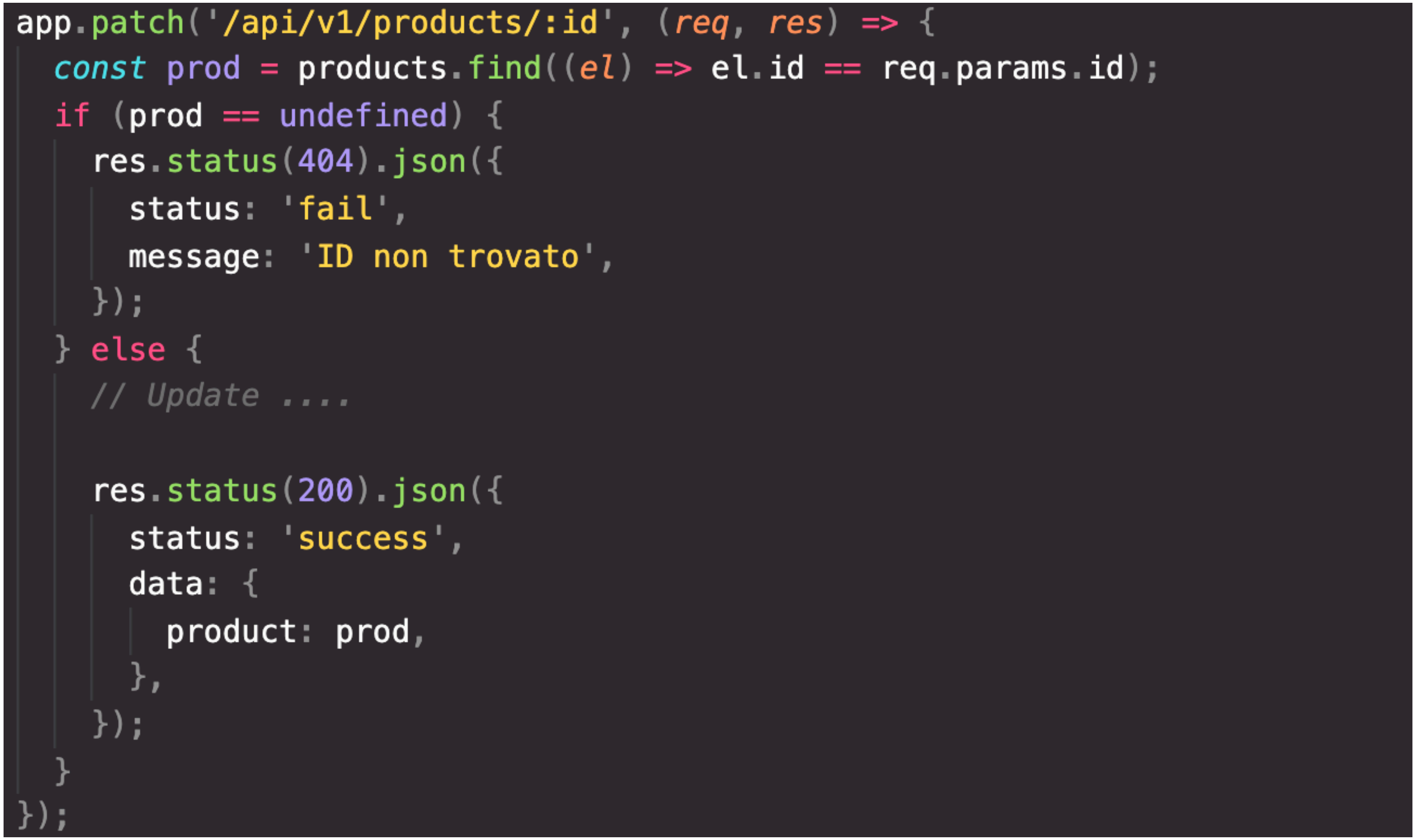
```
app.get('/api/v1/products/:id', (req, res) => {
  console.log(req.params);

  const prod = products.find((el) => el.id == req.params.id);
  console.log(prod);
  if (prod == undefined) {
    res.status(404).json({
      status: 'fail',
      message: 'ID non trovato',
    });
  } else {
    res.status(200).json({
      status: 'success',
      data: {
        product: prod,
      },
    });
  }
});
```

POST



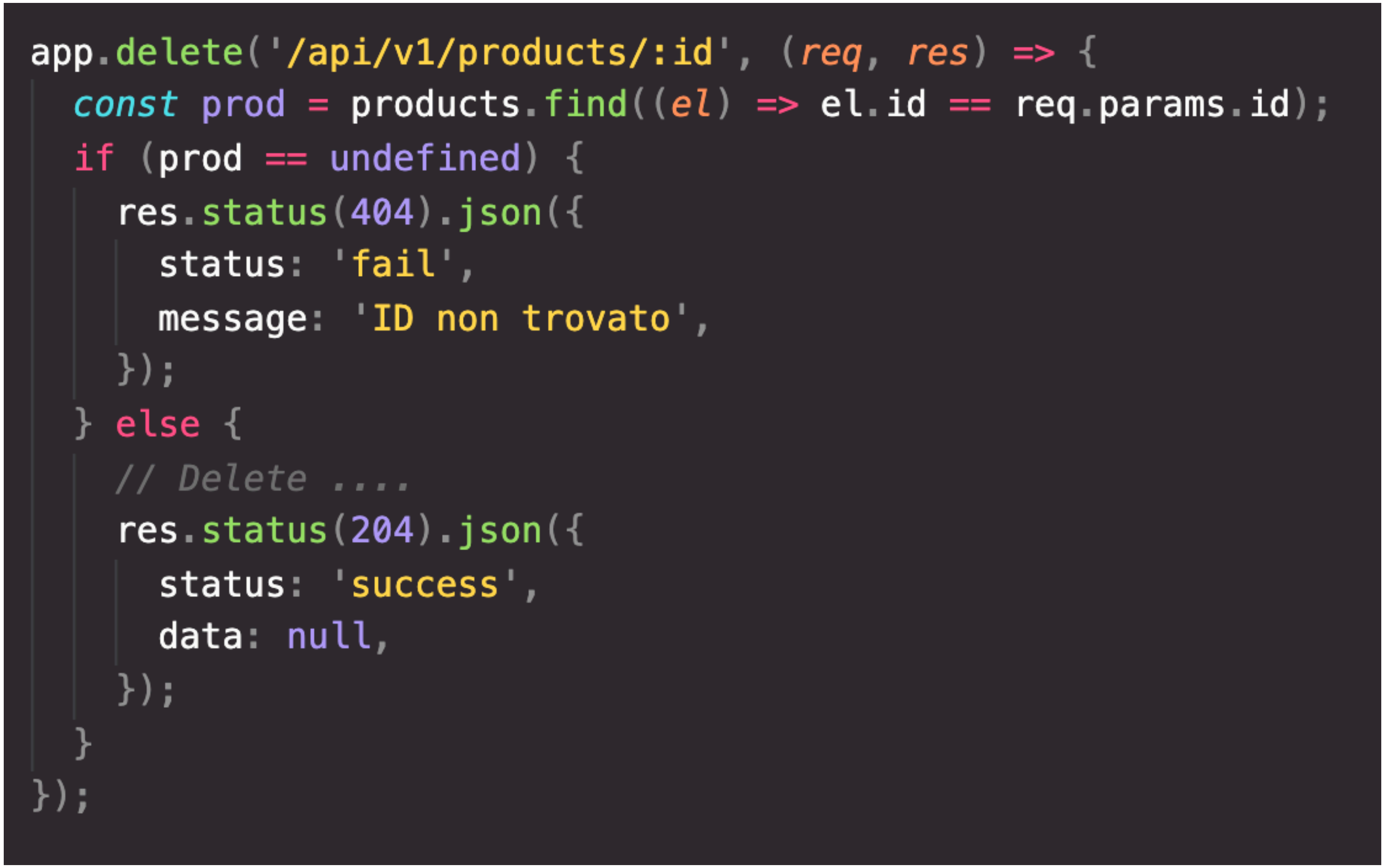
```
app.post('/api/v1/products', (req, res) => {
  const newId = products[products.length - 1].id + 1;
  const newProd = Object.assign({ id: newId }, req.body);

  products.push(newProd);
  res.status(201).json({
    status: 'success',
    data: { product: newProd },
  });
});
```

PUT/PATCH



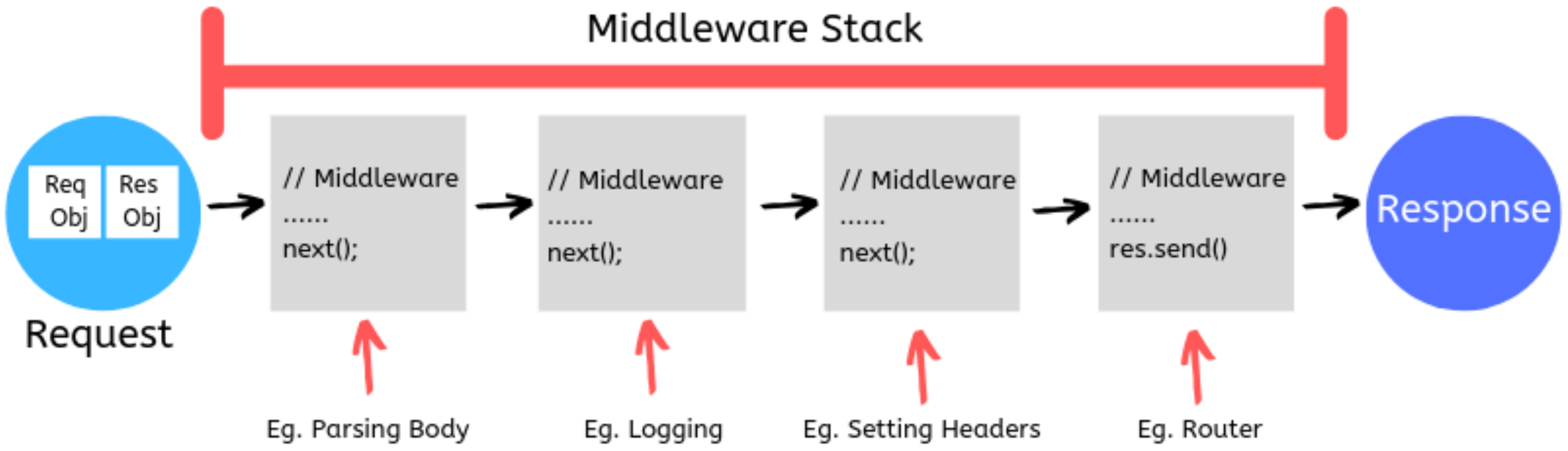
```
app.patch('/api/v1/products/:id', (req, res) => {
  const prod = products.find((el) => el.id == req.params.id);
  if (prod == undefined) {
    res.status(404).json({
      status: 'fail',
      message: 'ID non trovato',
    });
  } else {
    // Update ....

    res.status(200).json({
      status: 'success',
      data: {
        product: prod,
      },
    });
  }
});
```

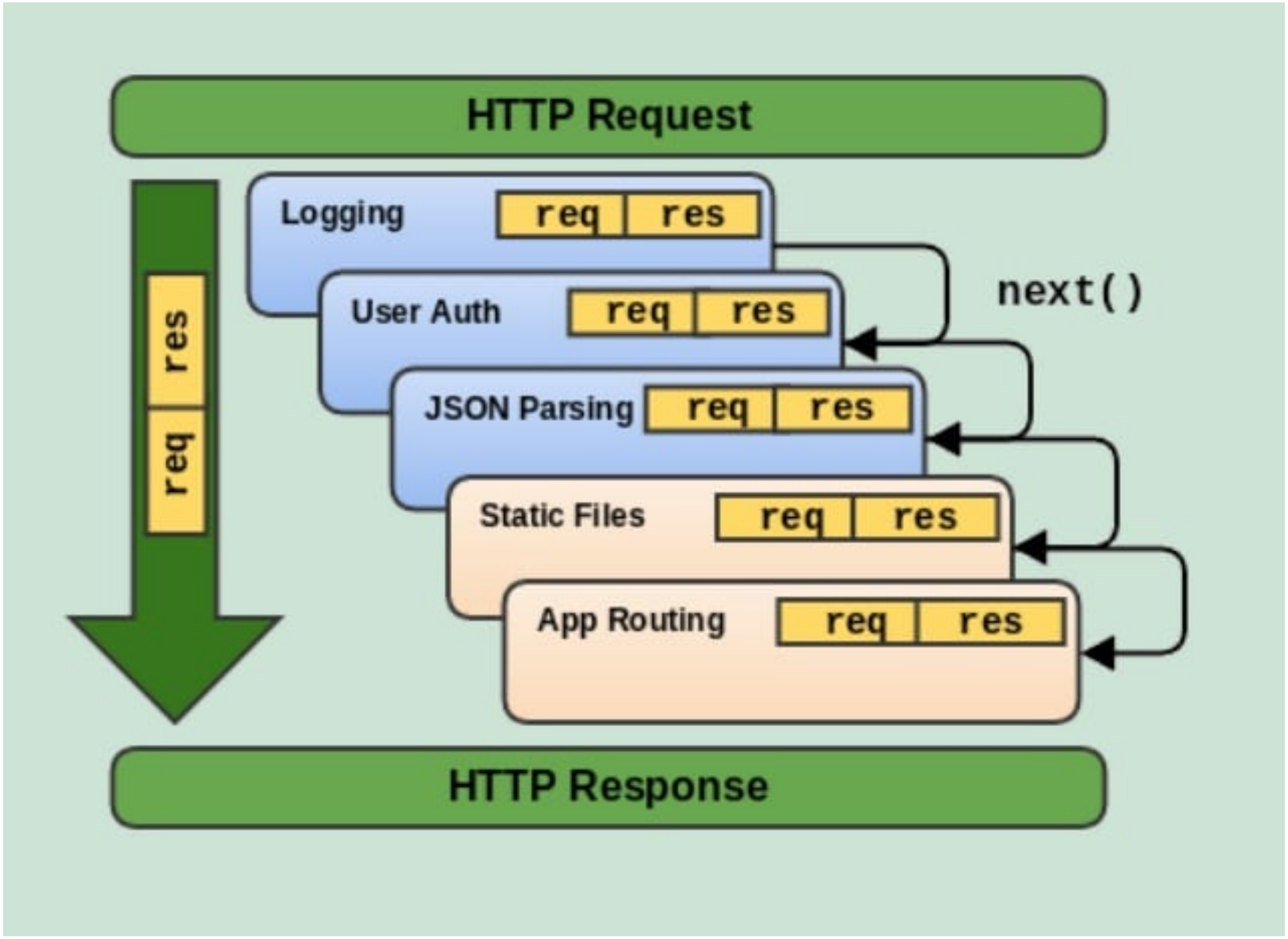
DELETE



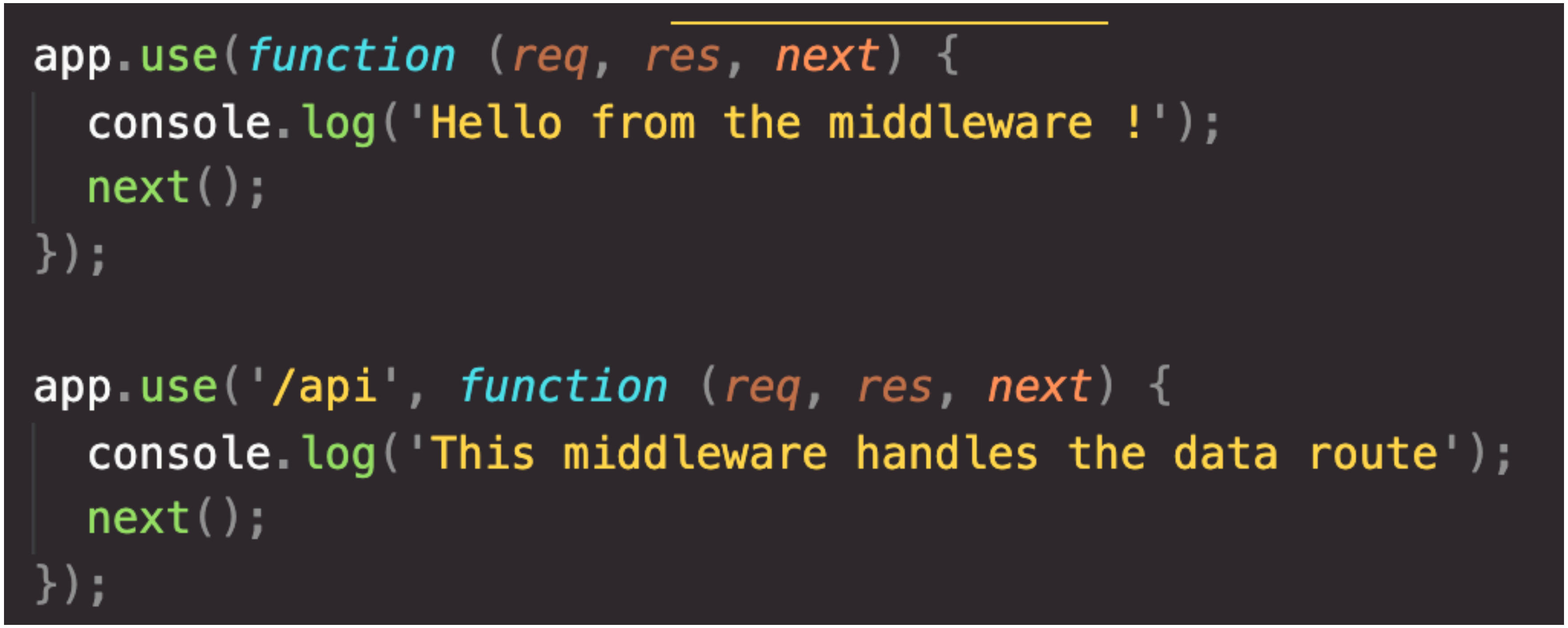
```
app.delete('/api/v1/products/:id', (req, res) => {
  const prod = products.find((el) => el.id == req.params.id);
  if (prod == undefined) {
    res.status(404).json({
      status: 'fail',
      message: 'ID non trovato',
    });
  } else {
    // Delete ....
    res.status(204).json({
      status: 'success',
      data: null,
    });
  }
});
```

Middleware


Middleware Stack
Req Obj
Res Obj
Request
// Middleware
......
next();
// Middleware
......
next();
// Middleware
......
next();
// Middleware
......
res.send()
Response
Eg. Parsing Body
Eg. Logging
Eg. Setting Headers
Eg. Router

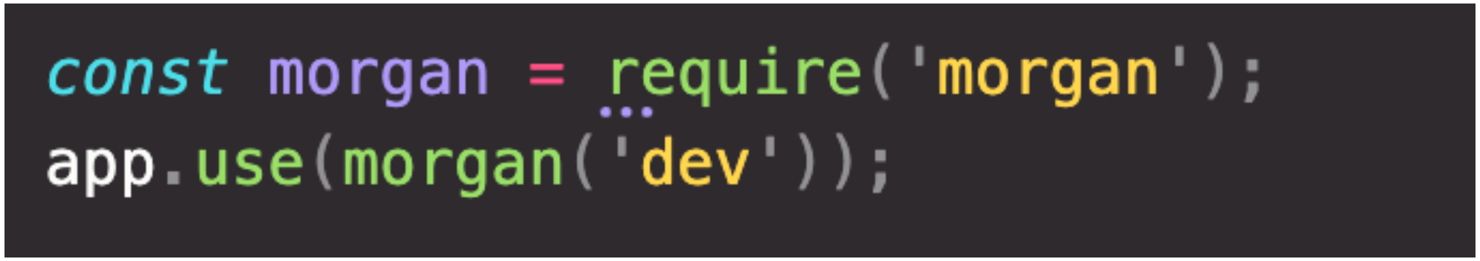

HTTP Request
res
req
Logging
req
res
User Auth
req
res
JSON Parsing
req
res
Static Files
req
res
App Routing
req
res
next()
HTTP Response

Custom Middleware



```
app.use(function (req, res, next) {
  console.log('Hello from the middleware !');
  next();
});

app.use('/api', function (req, res, next) {
  console.log('This middleware handles the data route');
  next();
});
```

Third Party middleware



```
const morgan = require('morgan');
app.use(morgan('dev'));
```

npm install morgan

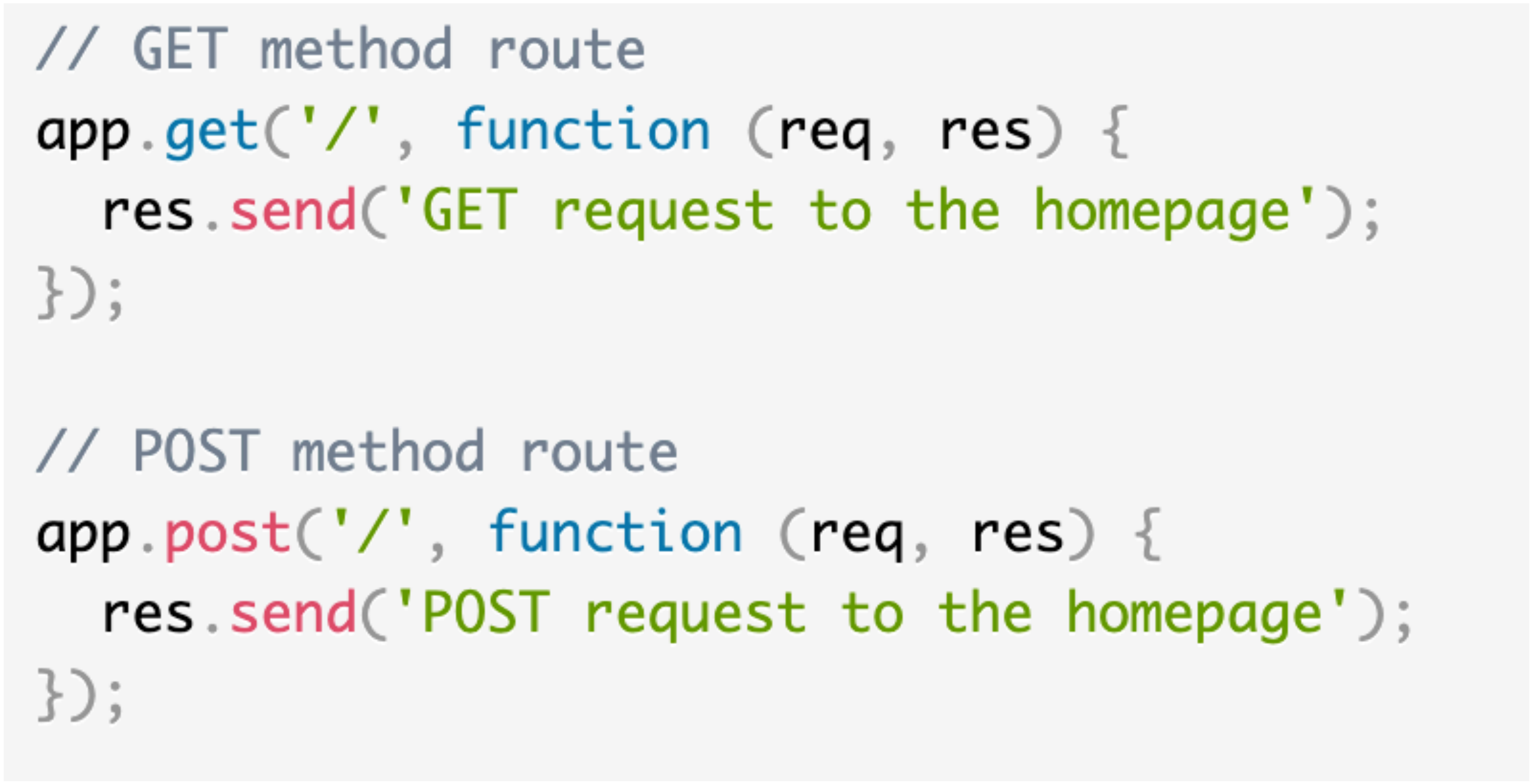
https://github.com/expressjs/morgan

Routing

https://expressjs.com/en/guide/routing.html

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes

get(), post(), put()...



```
// GET method route
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
});

// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

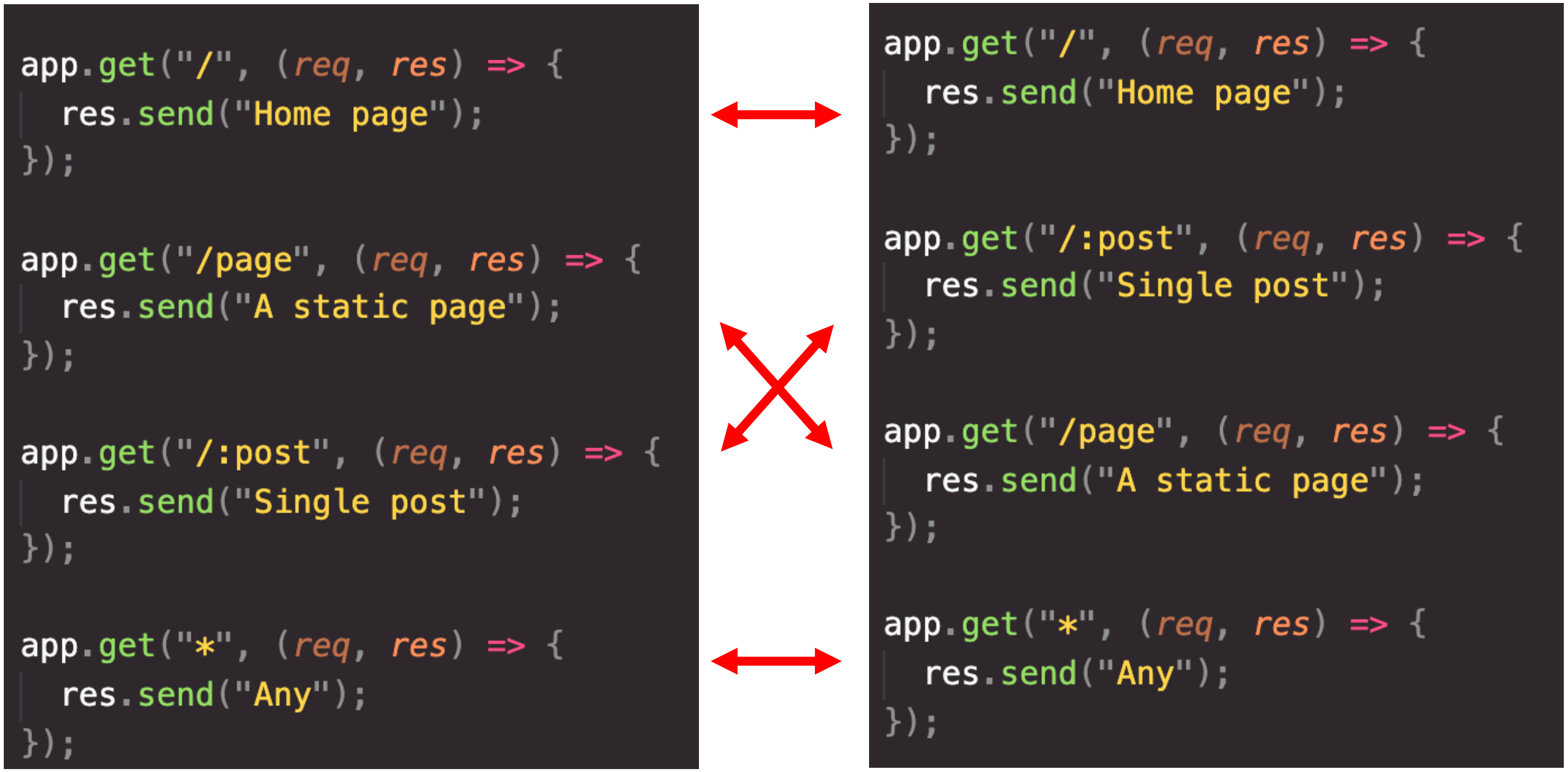
- get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search e connect

multiple handlers

```
var cb0 = function (req, res, next) {
  console.log('CB0');
  next();
}

var cb1 = function (req, res, next) {
  console.log('CB1');
  next();
}

var cb2 = function (req, res) {
  res.send('Hello from C!');
}

app.get('/example/c', [cb0, cb1, cb2]);
```

Un array di funzioni callback possono gestire una route.

Order of routes



```
app.get("/", (req, res) => {
  res.send("Home page");
});

app.get("/page", (req, res) => {
  res.send("A static page");
});

app.get("/:post", (req, res) => {
  res.send("Single post");
});

app.get("*", (req, res) => {
  res.send("Any");
});
```

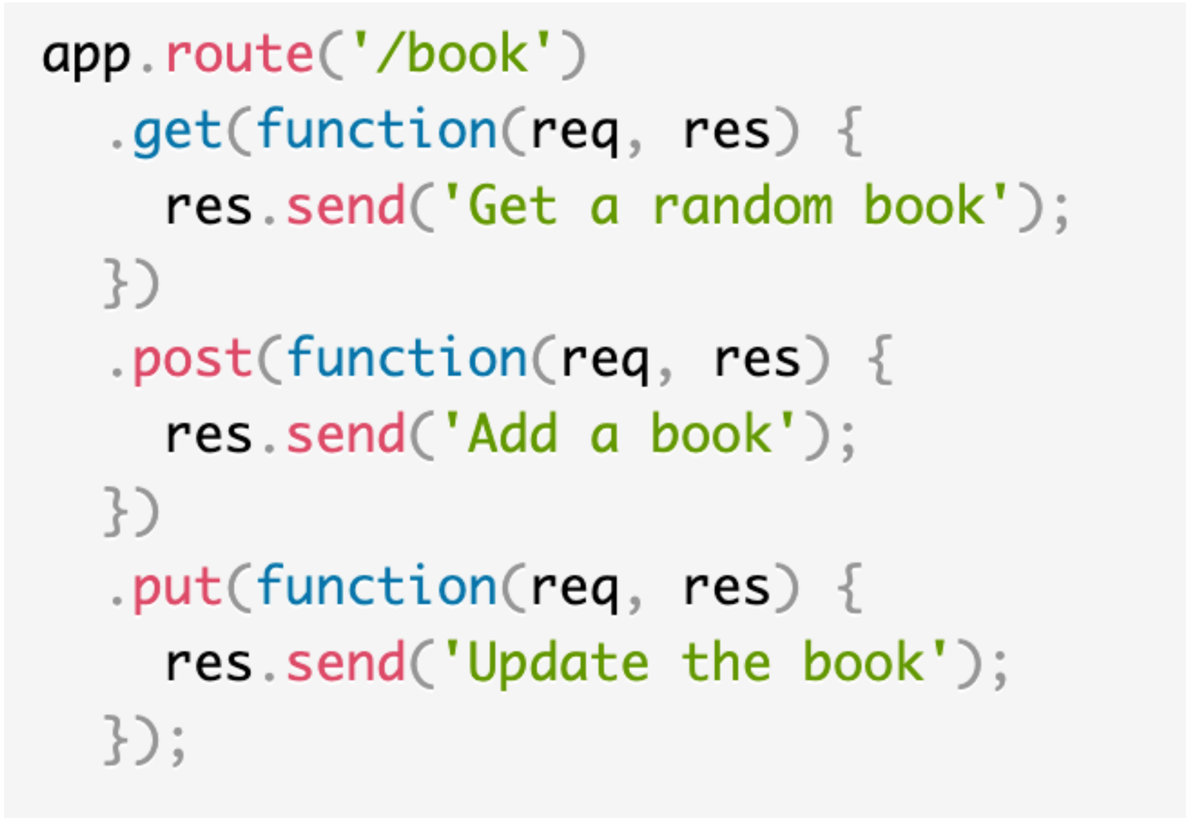
```
app.get("/", (req, res) => {
  res.send("Home page");
});

app.get("/:post", (req, res) => {
  res.send("Single post");
});

app.get("/page", (req, res) => {
  res.send("A static page");
});

app.get("*", (req, res) => {
  res.send("Any");
});
```

- parametric path inserted just before a literal one takes the precedence over the literal one

https://gabrieleromanato.name/order-of-routes-when-using-parameters-in-expressjs

route()

- È possibile creare handler di route concatenabili per un percorso di route, utilizzando app.route().



```
app.route('/book')
  .get(function(req, res) {
    res.send('Get a random book');
  })
  .post(function(req, res) {
    res.send('Add a book');
  })
  .put(function(req, res) {
    res.send('Update the book');
  });
```

Routers

```
var express = require('express');
var router = express.Router();

// middleware that is specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});
// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
```

La classe express.Router crea handler di route modulari e montabili.

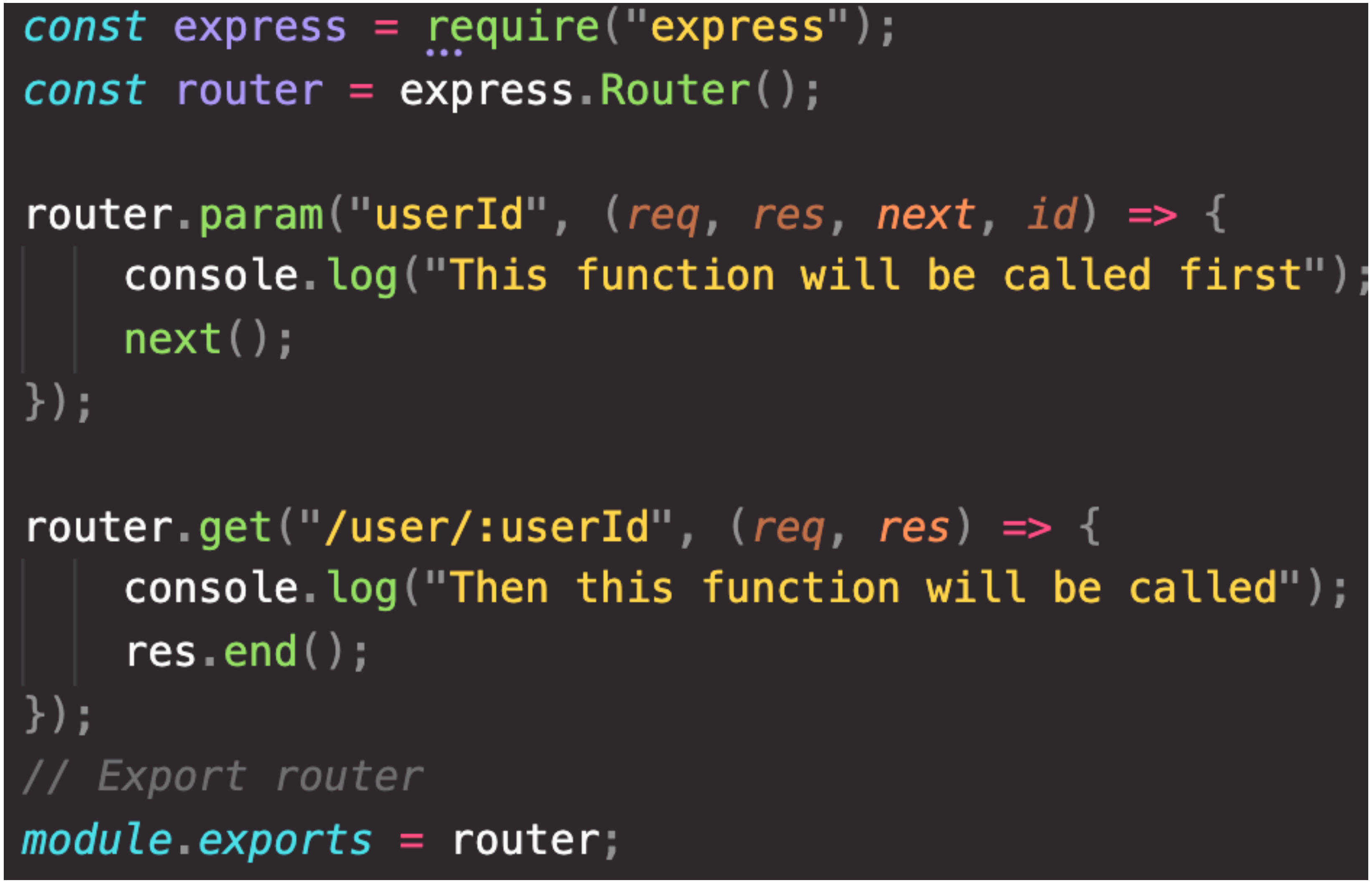
Un'istanza Router è un middleware e un sistema di routing completo; per questa ragione, spesso si definisce "mini-app".

/birds/

/birds/about

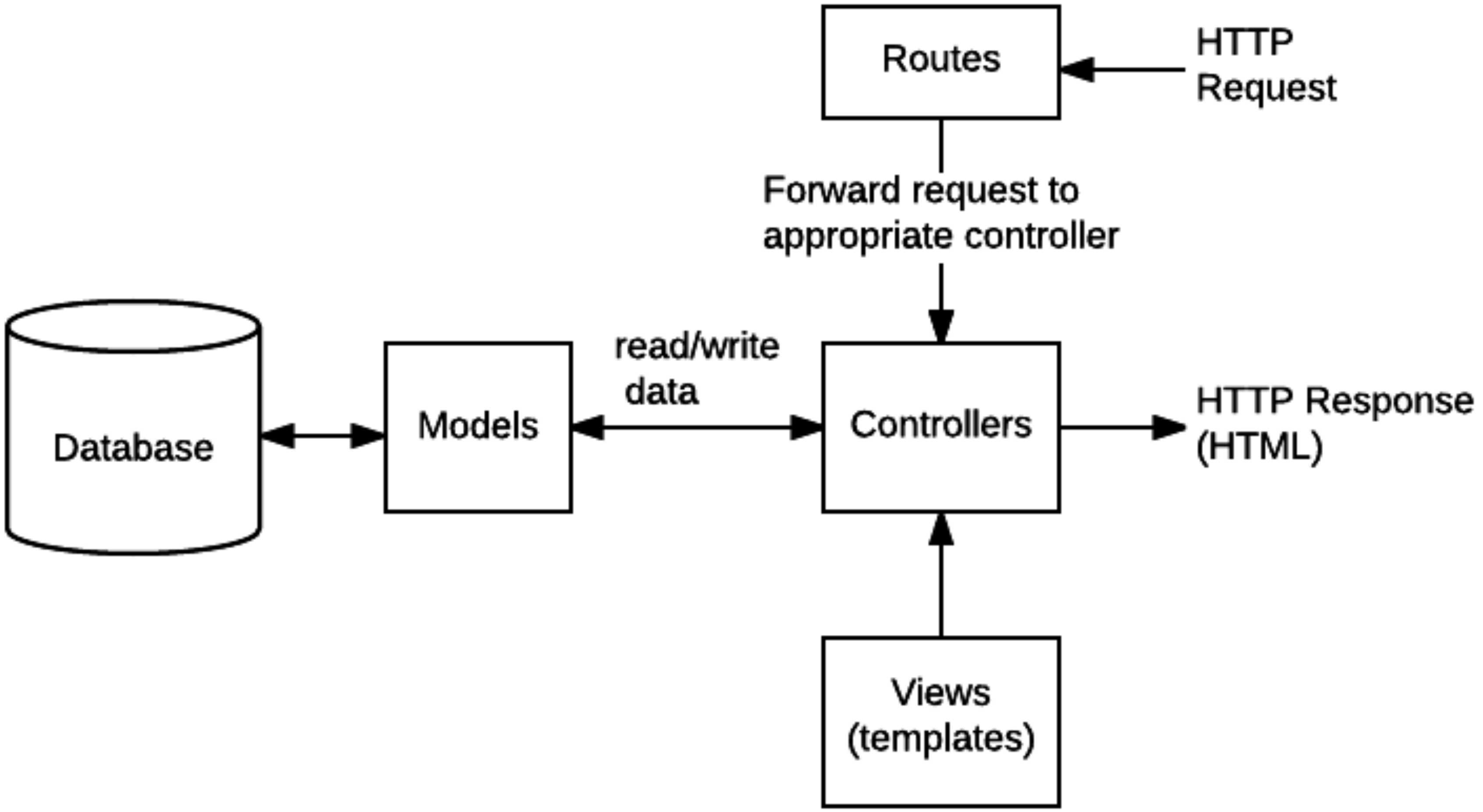
```
var birds = require('./birds');
...
app.use('/birds', birds);
```

param()



```
const express = require("express");
const router = express.Router();

router.param("userId", (req, res, next, id) => {
    console.log("This function will be called first");
    next();
});

router.get("/user/:userId", (req, res) => {
    console.log("Then this function will be called");
    res.end();
});
// Export router
module.exports = router;
```

MVC Architecture

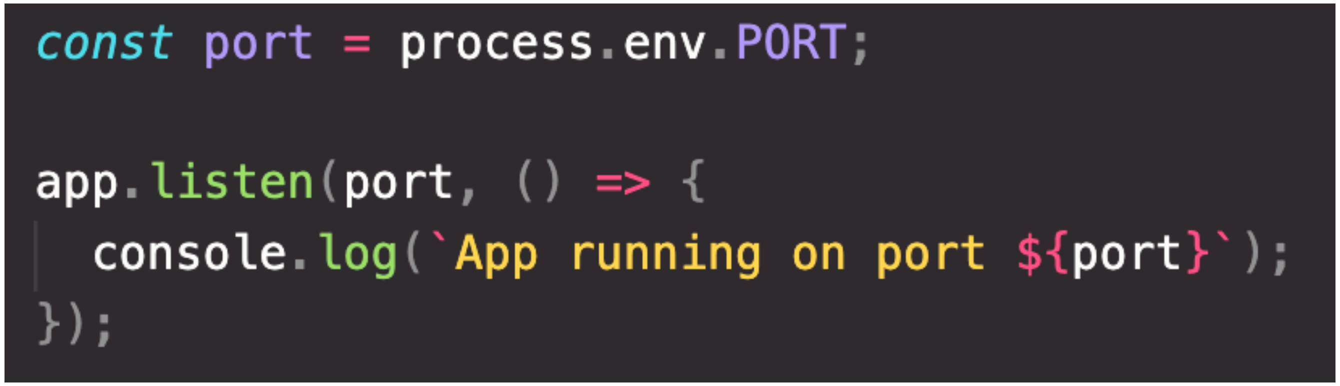


Environment

- Environment variables allowing apps to behave differently based on the environment
- Externalize all environment specific parameters

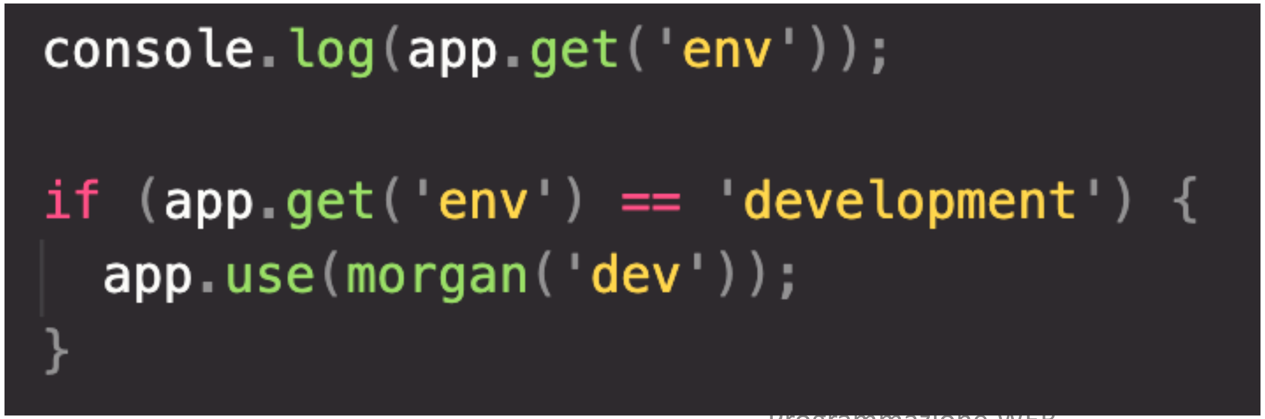
- Examples
  - Which HTTP port to listen on
  - What path and folder your files are located in, that you want to serve
  - Pointing to a development, staging, test, or production database

Examples



```
const port = process.env.PORT;

app.listen(port, () => {
  console.log(`App running on port ${port}`);
});
```

- PORT=8626 node server.js
- PORT=8626 NODE_ENV=development node server.js



```
console.log(app.get('env'));

if (app.get('env') == 'development') {
  app.use(morgan('dev'));
}
```

.env file

.env

```
PORT=8765
NODE_ENV=development

USERNAME=loreti
PASSWORD=12345
```

server.js

```javascript
const dotenv = require('dotenv');
dotenv.config();
```

```javascript
const dotenv = require('dotenv');
dotenv.config({ path: '/custom/path/to/.env' });
```

config module

config.js

```
const dotenv = require('dotenv');
dotenv.config();

module.exports = {
  username: process.env.USER_NAME,
  password: process.env.PASSWORD,
  port: process.env.PORT,
};
```

server.js

```
const { port, username, password } = require('./config');
```

Static Files

```
app.use(express.static('public'));
```

Per gestire i file statici, quali immagini, file CSS e file JavaScript, utilizzare la funzione middleware integrata express.static in Express.

Fornire il nome della directory che contiene gli asset statici alla funzione middleware express.static per iniziare a gestire i file direttamente. Ad esempio, utilizzare il seguente codice per gestire le immagini, i file CSS e i file JavaScript nella directory denominata public:

```
app.use(express.static('public'));
app.use(express.static('images'));
```

```
app.use('/static', express.static('public'));
```