

Aula 2 – NodeJS e ExpressJS

PROFESSOR: HARLEY MACÊDO DE MELLO

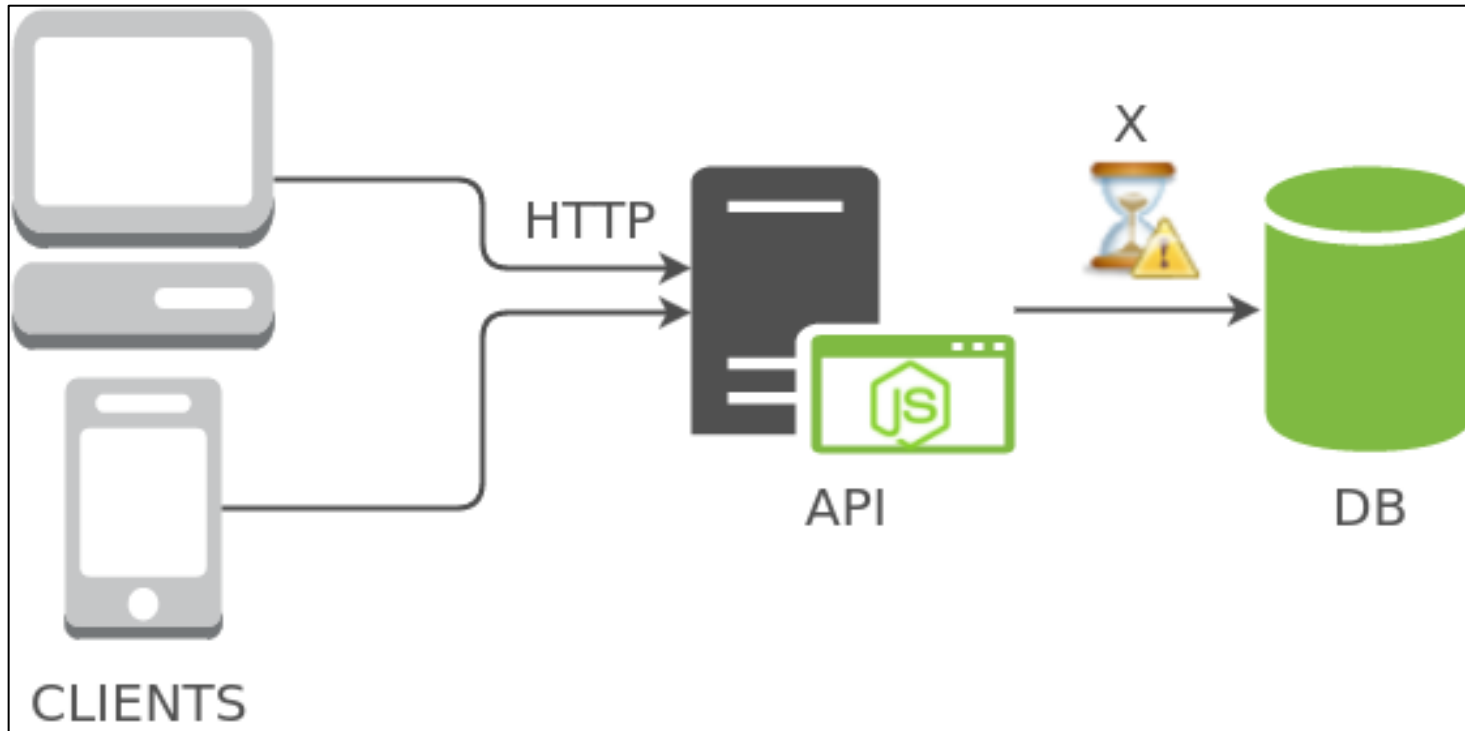
Roteiro

- NodeJS
- ExpressJS
- Rotas
- Middleware
- Bibliotecas importantes
- API de terceiros

NodeJS

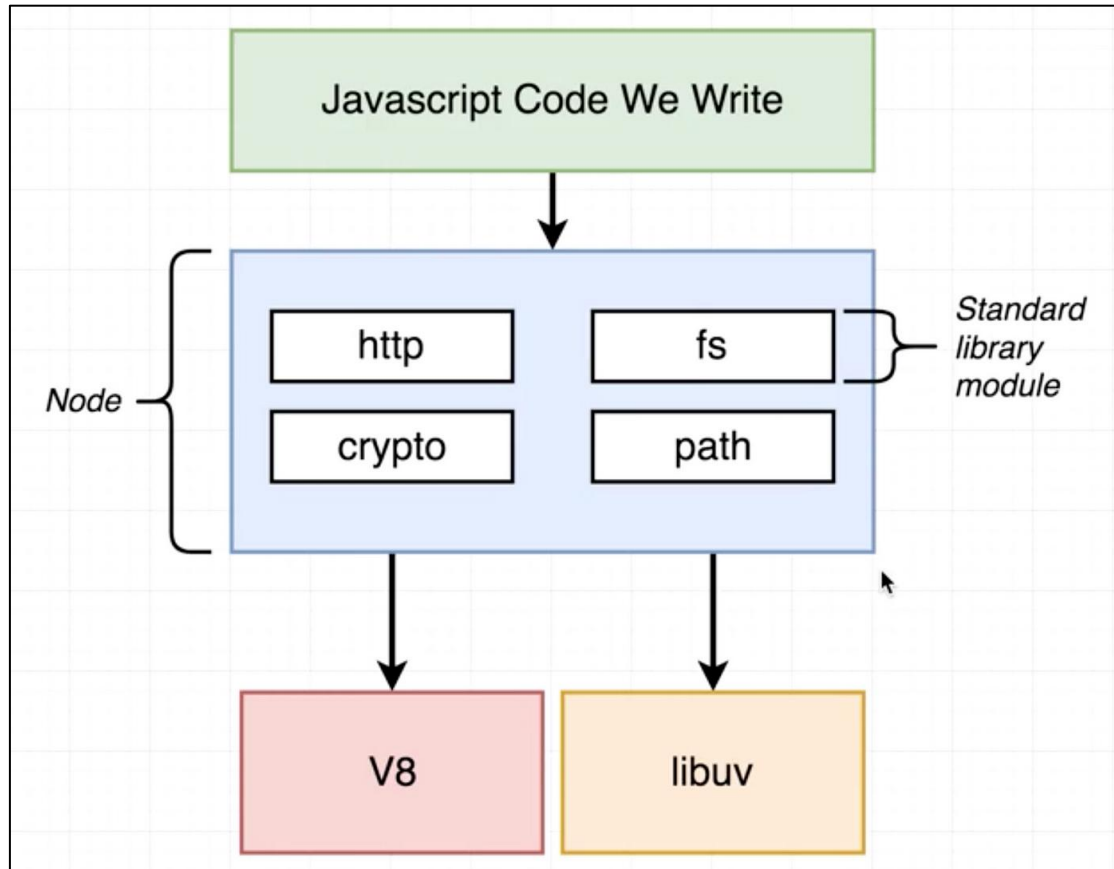
- Um ambiente de servidor open source
- Permite executar Javascript no servidor
- Pode gerar páginas dinâmicas
- Manipular arquivos no servidor
- Coletar e enviar dados de clientes
- Manipular banco de dados

NodeJS



Representação de um servidor NodeJS.

NodeJS



Representação das bibliotecas do NodeJS.

NodeJS

- Arquivo package.json
 - Todo projeto NodeJS deve ter o arquivo
 - Informações sobre a aplicação
 - Versão, descrição, repositório e dependências
 - 'npm init' cria uma estrutura inicial

NodeJS

```
1  {
2    "name": "t1projetobasico",
3    "version": "1.0.0",
4    "description": "Um projeto para exemplificar um a estrutura do ExpressJS",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [
10     "Projeto",
11     "Exemplo",
12     "ExpressJS"
13   ],
14   "author": "Harley Macedo",
15   "license": "ISC",
16   "dependencies": {
17     "express": "^4.18.2"
18   }
19 }
```

Arquivo package.json
preenchido.

NodeJS

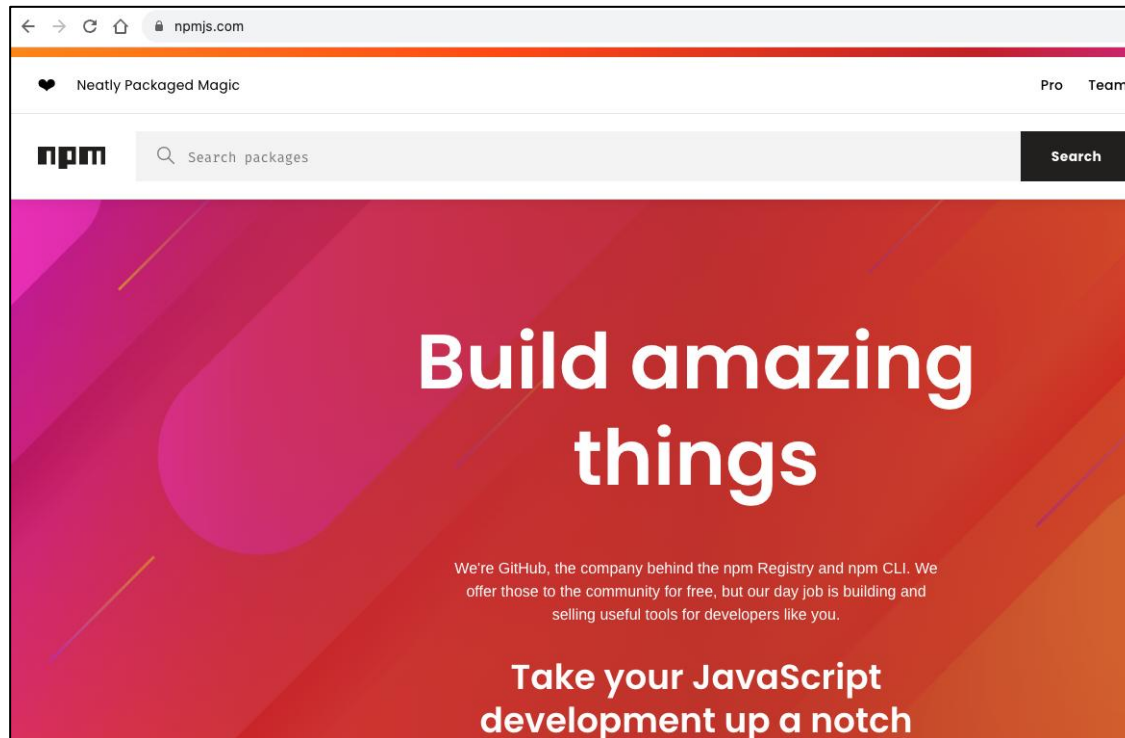
```
"scripts": {  
  "test": "jest",  
  "dev": "nodemon index.js",  
  "start": "node index.js",  
  "compile": "tsc"  
},
```

Arquivo package.json
preenchido.

NodeJS

- NPM
 - Node Package Manager
 - Módulos Node
 - 'npmjs.com' contem milhares de pacotes
 - O comando 'npm install pacote' instala pacotes
 - O comando 'npm uninstall pacote' desinstala pacotes
 - É possível verificar várias informações sobre os pacotes

NodeJS

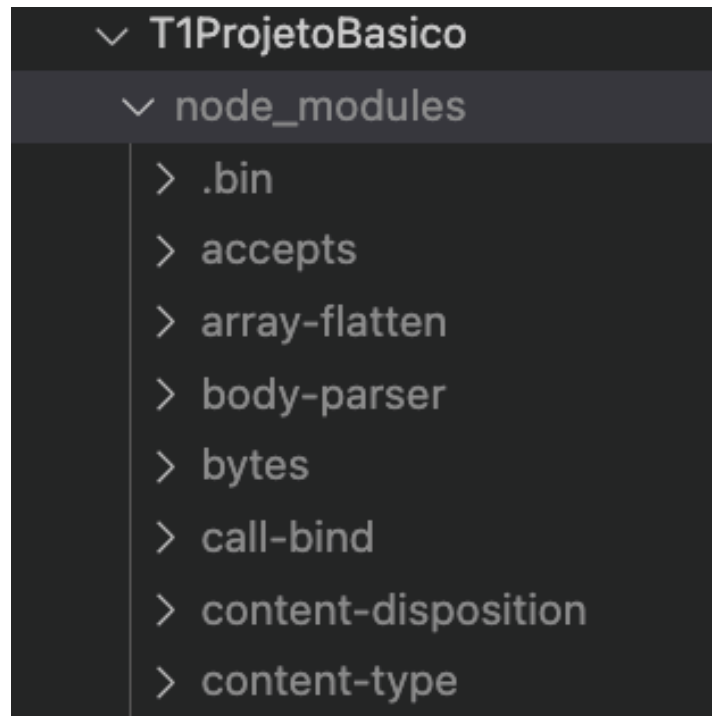


Site com repositório com milhares de pacotes(bibliotecas) node.

NodeJS

- Pasta node_modules
 - Contêm os pacotes utilizados pelo projeto
 - Pasta contém muitos arquivos e pastas
 - Ao enviar o projeto para outra máquina, não precisa enviar esta pasta
 - Os pacotes devem ser baixados na nova máquina com o comando 'npm install'
 - Todas as dependências descritas no package.json serão baixadas e atualizadas

NodeJS



Pasta node_modules ficam as bibliotecas que o projeto utiliza.

NodeJS

- Bibliotecas nativas
 - O NodeJS já contém muitas bibliotecas nativas
 - Como exemplo: http, fs, dns, crypto, os, net
 - Com as bibliotecas http e fs já pode-se criar um servidor http
 - Atualmente se usa frameworks, como o ExpressJS, que abstrai algumas bibliotecas

NodeJS

```
1 | //Importando bibliotecas do NodeJS
2 | const http = require('http')
3 | const fs = require('fs').promises
4 |
5 | //Mapeando URLs e definindo resposta
6 | const server = http.createServer((req, res) => {
7 |     switch (req.url) {
8 |         case '/recomendacao':
9 |             res.setHeader("Content-Type", "application/json");
10 |             res.writeHead(200)
11 |             res.end( JSON.stringify({recomendacao: 'Lendas da paixão'}) )
12 |             break
```

Servidor básico
com NodeJS puro.

NodeJS

- Nodemon
 - Módulo que observa alterações no projeto
 - Reinicia o servidor atual
 - Agiliza o desenvolvimento
 - 'npm install nodemon'
 - 'nodemon index.js'

ExpressJS

- Framework minimalista
- Gerencia rotas
- Facilmente configurado
- Função de servidor
- Abstrai o uso de módulos nativos

ExpressJS

- Comandos e etapas para criar um projeto básico:
 - 'npm init' para definir arquivo de projeto 'package.json'
 - Criar o arquivo 'index.js', que será nosso arquivo principal
 - 'npm install express' para instalar o módulo ExpressJS
 - Criar o código do projeto
 - 'node index.js' executa o projeto
 - Atalho 'ctrl + c' parar a execução do projeto

ExpressJS

```
1 //Importando e instanciando o ExpressJS
2 const express = require('express')
3 const app = express()
4
5 //Uso de middleware
6 app.use(express.static('public'));
7
8 //Definindo rotas com suas devidas respostas
9 app.get('/recomendacao', (req, res) => {
10 |   res.json({recomendacao: 'Lendas da paixão'})
11 | })
12 app.get('/lancamentos', (req, res) => {
13 |   res.json({recomendacao: ['7 anos no Tibet', 'Tempo de glória']})
14 | })
15 app.get('/cadastro', (req, res) => {
16 |   res.sendFile(__dirname + '/public/index.html')
17 | })
18
19 //Ouvinte das requisições
20 app.listen(3000)
```

Projeto básico com ExpressJS.

Rotas

- São os pontos de acesso de nossa API
- Programadas para enviar alguma resposta ao usuário
- Usam os verbos do padrão REST
- Os mais comuns são GET, POST, PUT e DELETE

Rotas

- Verbo 'GET': Obter informações do servidor, contêm informações na URL
- Verbo 'POST': Guardar novas informações, contêm informações no body da requisição
- Verbo 'PUT': Atualizar informações, contêm informações na URL e no body
- Verbo 'DELETE': Excluir informações, contêm informações na URL

Rotas

- Parâmetros
 - São informações passadas na URL
 - Uso do ':' para identificação pelo ExpressJS
 - São permitidos vários parâmetros
 - Usar o padrão 'servidor/:email/:agendar
 - Para obter o email, basta usar 'req.params.email'

Rotas

- Testador de rotas
 - Fazer requisições REST para os endpoints da nossa API
 - As requisições podem precisar de informações Body ou no Header
 - Usar um software testador de rotas
 - Thunder Client ou Postman são os mais utilizados
 - Fácil de criar requisições REST e configurar as informações

Rotas

```
25 //Rota para obter um professor através do nome
26 professorRouter.get('/professor/:nome', (req, res) => {
27     try {
28         const professorEncontrado = professores.find( (item) => {
29             return item.nome === req.params.nome
30         });
31         res.json(professorEncontrado)
32     } catch (error) {
33         res.json({erro: true, mensagem: 'Não foi possível recuperar os dados.'})
34     }
35 })
```

Rota GET com
parâmetro.

Rotas

- Configuração de respostas
 - `'res.download()'` solicitação para download
 - `'res.send()'` termina o processo de resposta
 - `'res.json()'` envia resposta do tipo JSON
 - `'res.redirect()'` redireciona uma solicitação
 - `'res.render()'` renderiza um modelo de view
 - `'res.sendFile()'` envia arquivo HTML estático

Rotas

```
1  //Imports gerais
2  const express = require('express')
3  const app = express()
4  const {engine} = require('express-handlebars')
5
6  //Definindo motor de view
7  app.engine('handlebars', engine({defaultLayout: null}) )
8
9  //Sets
10 app.set('view engine', 'handlebars')
11
12 //Resposta JSON
13 app.get('/json', (req, res) => {
14   res.json({mensagem: 'Resposta JSON enviada'})
15 })
16
17 //Resposta send
18 app.get('/send', (req, res) => {
19   res.send('Resposta de texto enviada')
20 })
```

Respostas com ExpressJS.

Rotas

```
22 //Resposta de download
23 app.get('/download', (req, res) => {
24     res.download('recursos/relatorio.docx')
25 })
26
27 //Resposta de redirecionamento
28 app.get('/redirect', (req, res) => {
29     res.redirect('/json')
30 })
31
32 app.get('/sendFile', (req, res) => {
33     res.sendFile(__dirname + '/public/login.html')
34 })
35
36 //Resposta render com handlebars
37 app.get('/render', (req, res) => {
38     res.render('sobre', {dado: 'Campus Crato'})
39 })
40
41 //Ouvinte de requisição
42 app.listen(3000)
```

Respostas com ExpressJS.

Rotas

The screenshot displays the Thunder Client interface. At the top, a GET request is configured to `http://localhost:3000/professor/Maria`. The 'Query' tab is active, showing a table for query parameters with columns 'parameter' and 'value'. The 'Response' tab is also active, showing the status '200 OK', size '39 Bytes', and time '3 ms'. The response body is a JSON object: `{ "nome": "Maria", "area": "Programação" }`.

parameter	value

```
1 {
2   "nome": "Maria",
3   "area": "Programação"
4 }
```

Fazendo requisição através do Thunder client.

Rotas

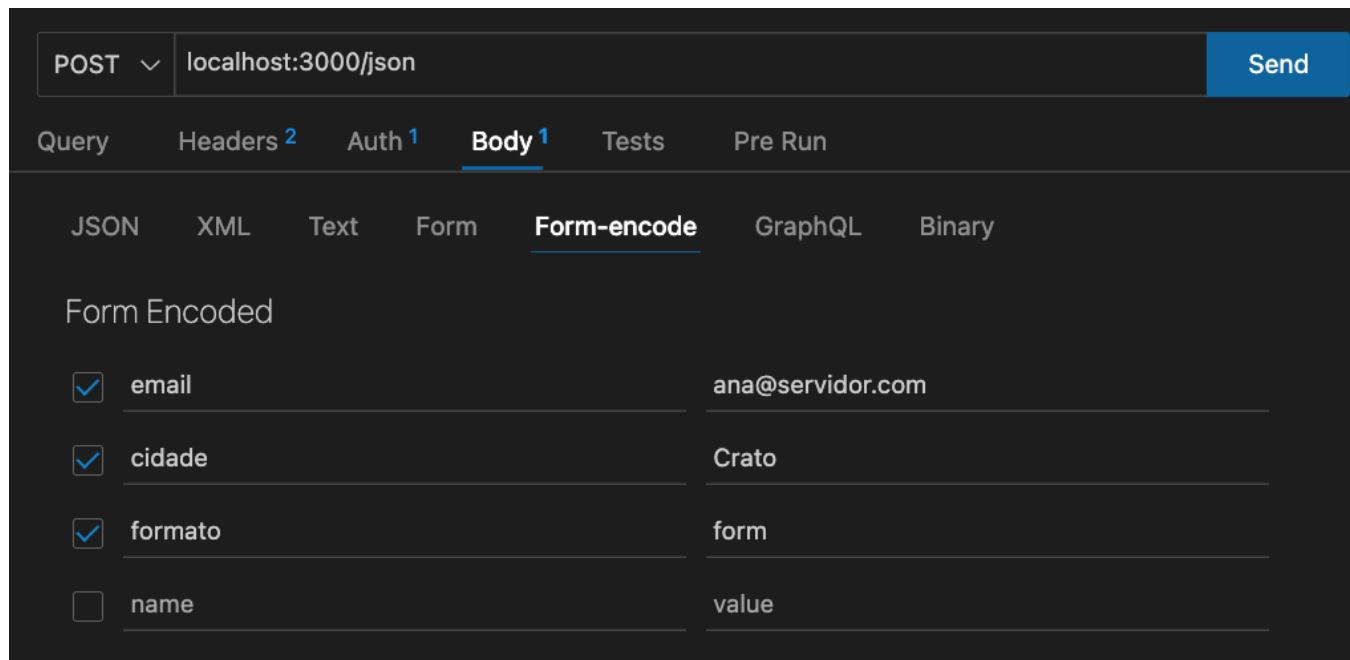
- Configuração de requisição
 - Habilitar os middlewares de requisição, para reconhecer envio de dados
 - `'express.json()'` habilita requisição no formato JSON
 - `'express.urlencoded({extended: true})'` habilita requisição no formato URLEncoded
 - Estas requisições podem ser testadas com um cliente de http como o Thunder Client

Rotas

```
5  app.use(express.json())
6  app.use(express.urlencoded({extended: true}))
7
8  //Rota que recebe dados JSON
9  app.post('/json', (req, res) => {
10 |      res.json({dadoEnviado: req.body})
11 |  })
12
13 //Rota que recebe dados de urlencoded
14 app.post('/body', (req, res) => {
15 |      res.json({dadoEnviado: req.body})
16 |  })
```

Recebendo dados em
formato json e form.

Rotas



The screenshot shows a REST client interface with a dark theme. At the top, the method is set to POST and the URL is localhost:3000/fjson. A 'Send' button is on the right. Below the URL bar, there are tabs for Query, Headers, Auth, Body, Tests, and Pre Run. The 'Body' tab is selected, and within it, the 'Form-encode' sub-tab is active. The 'Form Encoded' section contains a table with four rows, each representing a form field. The first three rows are checked, and the last one is not.

Form Encoded	
<input checked="" type="checkbox"/> email	ana@servidor.com
<input checked="" type="checkbox"/> cidade	Crato
<input checked="" type="checkbox"/> formato	form
<input type="checkbox"/> name	value

Preparando requisição
com dados enviados
por form.

Rotas

```
Status: 200 OK   Size: 368 Bytes   Time: 35 ms

Response  Headers 6  Cookies  Results  Docs
1  {
2    "dadoEnviado": {
3      "email": "ana@servidor.com",
4      "cidade": "Crato",
5      "formato": "form"
6    },
7    "headers": {
8      "content-length": "48",
9      "accept-encoding": "gzip, deflate, br",
10     "accept": "*/*",
11     "user-agent": "Thunder Client (https://www.thunderclient.com)",
12     "content-type": "application/x-www-form-urlencoded",
13     "authorization": "Basic aGFybGV50jEyMzQ1Ng==",
14     "host": "localhost:3000",
15     "connection": "close"
16   }
17 }
```

Resultado da requisição
feita com envio de dados.

Rotas

```
1 //Imports gerais
2 const express = require('express')
3 const app = express()
4
5 app.use(express.json())
6 app.use(express.urlencoded({extended: true}))
7
8 //Rota que recebe dados JSON
9 app.post('/json', (req, res) => {
10 |   res.json({dadoEnviado: req.body})
11 | })
12
13 //Rota que recebe dados de urlencoded
14 app.post('/body', (req, res) => {
15 |   res.json({dadoEnviado: req.body})
16 | })
17
18 //Ouvinto de requisição
19 app.listen(3000)
```

Rotas que recebem dados JSON ou URLEncoded.

Rotas

The screenshot displays a REST client interface with a POST request to `localhost:3000/body`. The 'Body' tab is active, showing 'Form-encoded' data with three fields: 'email' (checked, value: `ana@servidor.com`), 'cidade' (checked, value: `Crato`), and 'name' (unchecked, value: `value`). The 'Response' tab shows a `200 OK` status, `61 Bytes` size, and `16 ms` time. The response body is a JSON object: `{ "dadoEnviado": { "email": "ana@servidor.com", "cidade": "Crato" } }`.

Method	URL	Status	Size	Time
POST	localhost:3000/body	200 OK	61 Bytes	16 ms

Field	Value
email	ana@servidor.com
cidade	Crato
name	value


```
{
  "dadoEnviado": {
    "email": "ana@servidor.com",
    "cidade": "Crato"
  }
}
```

Requisição com dados no formulário.

Middleware

- É uma função que intercepta requisições
- Registrados como 'app.use(nomeMiddleware)'
- Pode modificar a requisição e a resposta
- Pode encerrar o ciclo de requisição
- Pode chamar o próximo middleware
- Pode executar qualquer código

Middleware

- Middlewares registrados com o 'use' interceptam todas as rotas
- Middlewares registrados em uma rota, só interceptam aquela rota
- Em uma rota, podem ser inseridos vários middlewares
- A ordem dos middlewares influenciam na sua execução

Middleware

```
5 //Imports dos middlewares
6 const confereHorario = require('./middlewares/horarioPermitido')
7 const registrarAtividade = require('./middlewares/logAtividade')
8
9 //Anexando o middleware para todas as rotas
10 app.use(registrarAtividade)
11
12 //Rota raiz, que será interceptada pelo middleware confereHorario
13 app.get('/', (req, res) => {
14   res.json({mensagem: 'App em execucao'})
15 })
16
17 //Middleware específico, que vai interceptar apenas esata rota
18 app.get('/rota2', confereHorario, (req, res) => {
19   res.json({mensagem: 'App em execucao2'})
20 })
```

Middleware importados e anexados de forma geral e específica.

Middleware

```
1 //Função de middleware que verifica se já mais de 8 horas para liberar acesso
2 const confereHorario = (req, res, next) => {
3   let date = new Date()
4   let hora = date.getHours()
5   if (hora >= 8) {
6     return res.json({mensagem: 'Nesse horário não é permitido fazer essa requisição'})
7   }
8   next()
9 }
10
11 module.exports = confereHorario
```

Função de Middleware.

Middleware

```
1 //Função de middleware que imprime dados da requisição quando alguma rota é acessada
2 const registrarAtividade = (req, res, next) => {
3   let date = new Date()
4   let hora = date.getHours()
5   let minutos = date.getMinutes()
6   let segundos = date.getMinutes()
7   console.log(`Atividade: ${req.ip}, ${req.url}, ${hora}:${minutos}:${segundos}`)
8   next()
9 }
10
11 module.exports = registrarAtividade
```

Função de Middleware.

Bibliotecas importantes

- Biblioteca de PDF
 - 'pdfkit' ou 'html-pdf'
 - 'pdfkit' tem muitas funcionalidades
 - Pode-se adicionar imagens, tabelas e formas geométricas
 - Pode-se escolher fontes e cores
 - 'npm install pdfkit'

Bibliotecas importantes

```
4  const PDFKIT = require('pdfkit')
5  const fs = require('fs')
6
7  //Rota para gerar o PDF e disponibilizar para download
8  app.get('/relatorio/pdf', (req, res) => {
9    try {
10      const pdf = new PDFKIT()
11      pdf.text('Relatório com os filmes mais assistidos')
12      pdf.end()
13      pdf.pipe(fs.createWriteStream('relatorio.pdf')).on('finish', () => {
14        res.download('./relatorio.pdf')
15      })
16    } catch (error) {
17      res.json({mensagem: 'Erro na geração do relatório'})
18    }
19  })
```

Biblioteca de geração de PDF em uso.

Bibliotecas importantes

- Biblioteca de upload
 - Biblioteca 'multer'
 - Recebe dados de um file de um formulário
 - Transfere esse arquivo para pasta definida
 - Enctype do form precisa ser 'multipart/formdata'
 - 'npm install multer'

Bibliotecas importantes

```
8 //Importando o multer
9 const multer = require('multer')
10
11 // Configuração de armazenamento
12 const storage = multer.diskStorage({
13   destination: function (req, file, cb) {
14     cb(null, 'uploads/')
15   },
16   filename: function (req, file, cb) {
17     // Extração de nome e extensão do arquivo original
18     const extensaoArquivo = file.originalname.split('.')[1]
19     const novoNomeArquivo = file.originalname.split('.')[0]
20     cb(null, `${novoNomeArquivo}.${extensaoArquivo}`)
21   }
22 })
23
24 //Instanciando o multer
25 const upload = multer({ storage })
26
27 //Definindo rotas com suas devidas respostas
28 app.get('/cadastro', (req, res) => {
29   res.sendFile(__dirname + '/public/index.html')
30 })
31 app.post('/recebeDados', upload.single('file'), (req, res) => {
32   res.json({mensagem: 'Arquivo armazenado'})
33 })
```

Fazendo upload de arquivo com o multer.

Bibliotecas importantes

- Biblioteca de qrcode
 - Biblioteca 'qrcode'
 - Geração de QR Code
 - Guardar informações numéricas, alfanuméricas ou bytes
 - Pode-se definir tamanho da imagem
 - Pode-se definir cores
 - 'npm install qrcode'


API de terceiros

- Algumas funcionalidades podem ser construídas através de serviços
- Popularização de serviços para desenvolvedores
- Centralizar ou dividir em diversos serviços de nuvem
- AWS, MS Azure, Google Cloud e Twilio despontam como os principais

API de terceiros

- Serviços como
 - Envio de email
 - Processamento de perguntas
 - Processamento de endereços e mapas
 - Processamento de pagamentos
 - Tradução de textos
 - Armazenamento de arquivos
 - Processamento de imagens

API de terceiros



[API Hub](#) [Organizations](#) [Apps](#) [My APIs](#)




Image To Text OCR

By [AI Touch](#) | Updated 2 months ago | [Visual Recognition](#)

Popularity

9.7 / 10

Latency


247ms

Service Level

100%

[Endpoints](#) [About](#) [Tutorials](#) [Discussions](#) [Pricing](#) [Subscribed](#)

[Back to All Plans](#)



Subscription Created Successfully

Start using your API - [API Documentation](#)

API de terceiros

```
15 app.get('/imagem0cr', async (req, res) => {
16   const encodedParams = new URLSearchParams();
17   encodedParams.set('imageUrl', 'https://enfoquevisual.com.br/cdn/shop/products/AVS-012.jpg')
18   const options = {
19     method: 'POST',
20     url: 'https://image-to-text-ocr1.p.rapidapi.com/ocr',
21     headers: {
22       'content-type': 'application/x-www-form-urlencoded',
23       'X-RapidAPI-Key': process.env.X_RapidAPI_Key,
24       'X-RapidAPI-Host': 'image-to-text-ocr1.p.rapidapi.com'
25     },
26     data: encodedParams,
27   }
28   try {
29     const response = await axios.request(options)
30     console.log(response.data.text)
31     res.json({texto: response.data.text})
32   } catch (error) {
33     console.error(error)
34   }
35 }
```