

Engenharia de Software

Prof. Vinicius Cardoso Garcia

vcg@cin.ufpe.br :: [@vinicius3w](https://twitter.com/vinicius3w) :: viniciusgarcia.me

[IF977] Engenharia de Software

<http://bit.ly/vcg-es>

Licença do material

Este Trabalho foi licenciado com uma Licença
Creative Commons - Atribuição-NãoComercial-
Compartilhagual 3.0 Não Adaptada



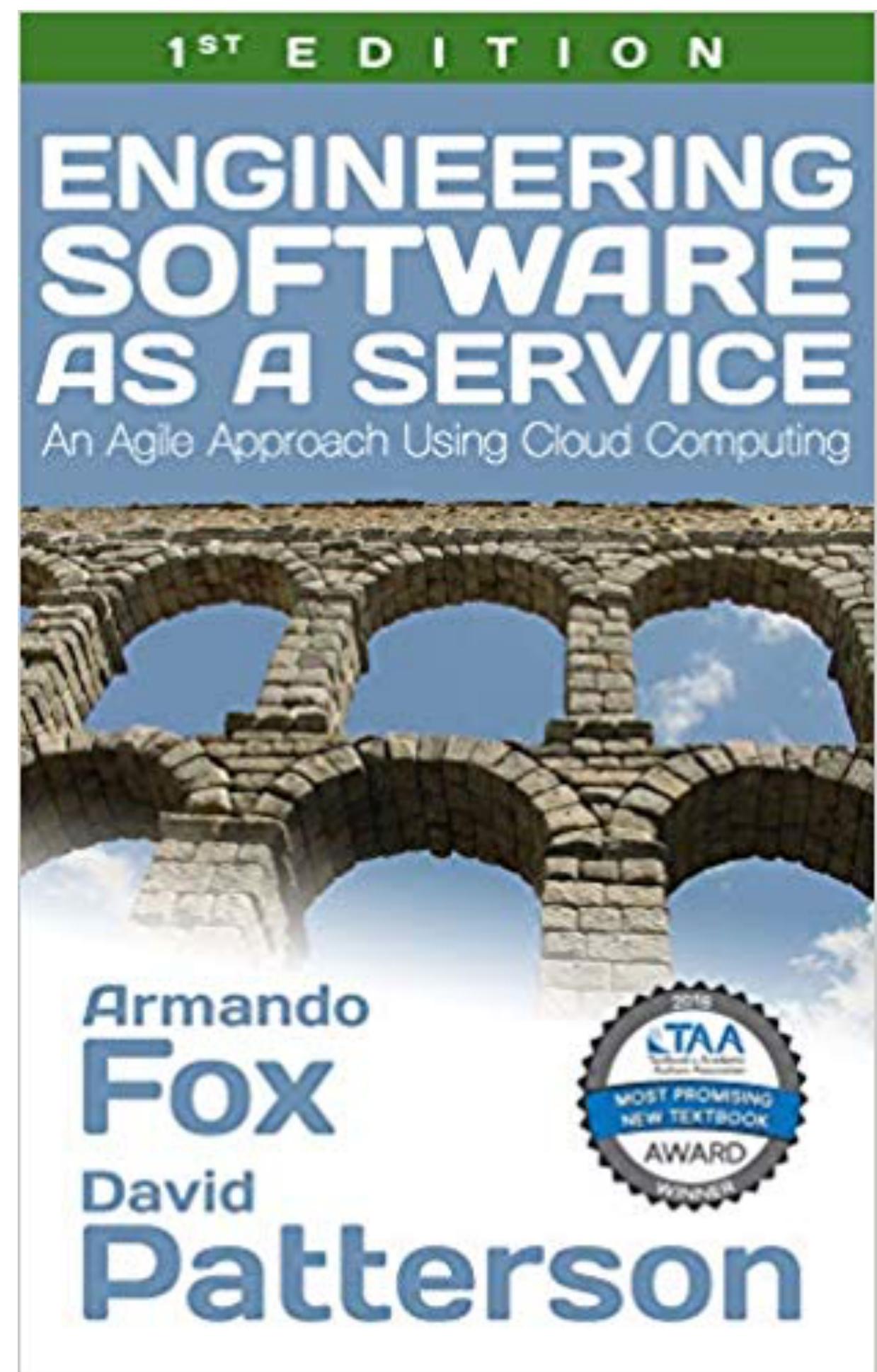
Mais informações visite

[http://creativecommons.org/licenses/by-nc-sa/
3.0/deed.pt](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt)



Referências

- A biblioteca do Desenvolvedor de Software dos dias de hoje
 - <http://bit.ly/31WYK5f>
- SWEBOK: Guide to the Software Engineering Body of Knowledge (SWEBOK)
 - <http://www.computer.org/web/swebok>
- Engineering Software as a Service: An Agile Approach Using Cloud Computing
 - <http://www.saasbook.info/>
- Marco Tulio Valente. Engenharia de Software Moderna
 - <https://engsoftmoderna.info/>



O que é isso? Que ano?

- Década de 1950
 - 90 minutos para processar uma reserva
 - 80% dos assentos não vendidos foi devido a erros de processamento manual



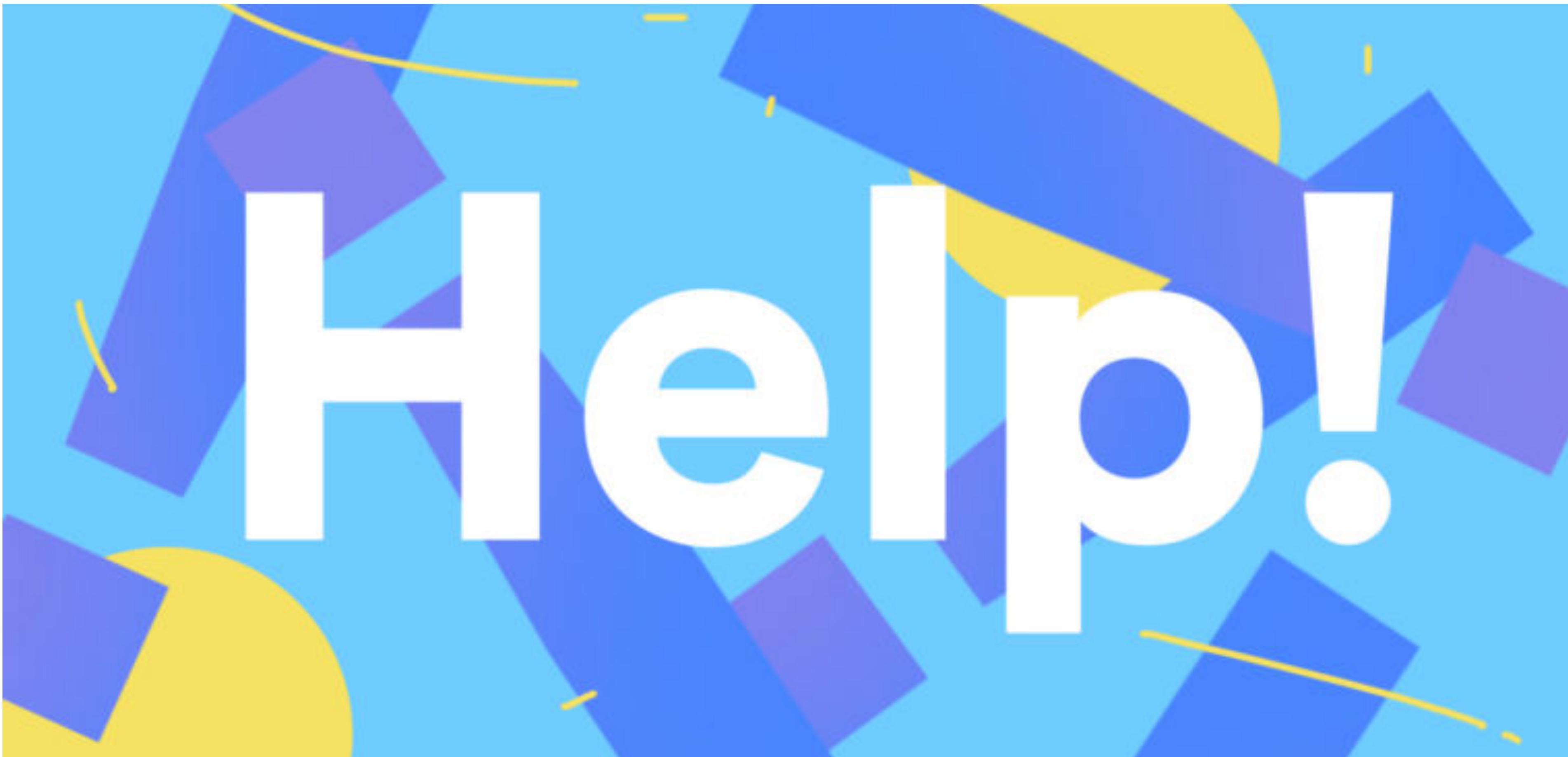


- 1957: contrato (\$ 40M; =~ \$ 300M hoje)
- 1961: dois 7090s, 1500 terminais, 20K reservas/day
- 1964: todas as reservas da AA via SABRE
- 1976: agentes de viagens recebem terminais
- Anos 80/90: acesso via Compu\$erve, AOL



Protocolo personalizado
baseado em FM sobre linhas
alugadas da AT & T



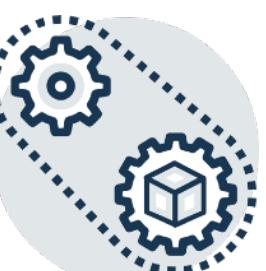


Como pedir por ajuda



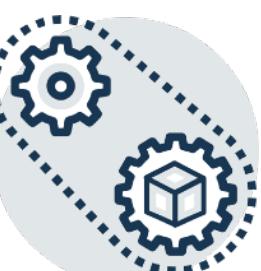
Depuração: "método científico"

1. Tão especificamente quanto possível, **o que você esperava que acontecesse?**
 - Havia alguma condição prévia assumida?
2. Tão especificamente quanto possível, **o que realmente aconteceu?**
 - Quanta informação você pode extrair das mensagens de erro?
 - Se não houver mensagem, você pode identificar o último ponto em que as coisas estavam da maneira que você espera?
3. Forme uma hipótese que possa explicar a discrepância
 - i.e. Variável incorreta ou nula → onde está configurada?
 - i.e. declaração não alcançada → qual foi o último lugar no caminho de código alcançado?
4. Crie uma maneira de testar a hipótese
 - **O emparelhamento pode mantê-lo honesto a cada passo**



RAPoS: usando o “método científico” para os passos 1 & 2

- **Depurar é um fato da vida. Todos fazemos isso.**
- **Leia a mensagem de erro.** Realmente leia.
- **Faça uma pergunta informada a um colega.**
- **Publique no StackOverflow, fóruns de classes, etc.**
 - Outros são tão ocupados quanto você. Ajude-os a ajudá-lo, fornecendo informações mínimas, mas completas
 - Pode ser mais rápido se o erro for altamente específico para algo no curso
- **ou Pesquise usando StackOverflow, Google, etc.**
 - Erros **esotéricos** podem afetar **versões específicas** de bibliotecas, código, sistemas operacionais etc. Mas a maioria dos erros **não é esotérica**.



Leia as mensagens de erro

- Algum programador achou que essa mensagem de erro seria útil no contexto
- De onde veio a mensagem? (que parte do código? que ferramenta?)
- Qual foi a última coisa relacionada que você tentou que funcionou conforme o esperado? O que mudou?
- Um colega pode reproduzir? (Você está trabalhando com um colega, certo?)



Coisas que provavelmente não ajudarão

- A publicação de uma mensagem de erro longa e sem explicação
- Fazendo uma pergunta sem mensagem de erro ou explicação
- Não descrevendo o que você já tentou
- "Acho que encontrei um bug no Node / React / etc"
- "Acho que encontrei um bug na lição de casa"?
Talvez! mas outros podem reproduzir?



Express.js



Construindo API's em Node + Express

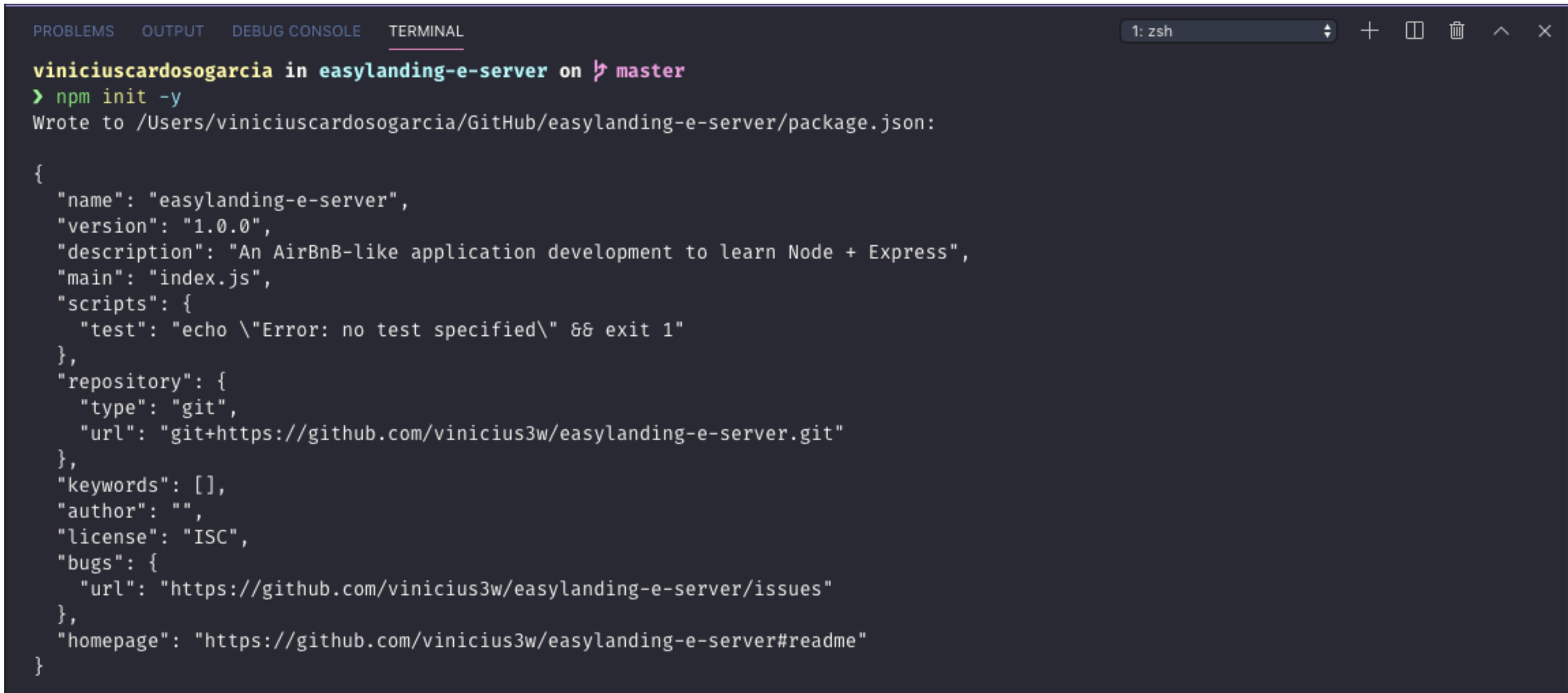


Preparando o ambiente

- Instalação Node
 - <https://nodejs.org/>
- Instalação MongoDB
 - <https://docs.mongodb.com/manual/installation/>
- Instalação NPM
 - <https://www.npmjs.com/get-npm>
- Instalação Yarn
 - <https://yarnpkg.com/pt-BR/docs/install>
- Post sobre ES6, ES7 e ES8
 - <https://blog.rocketseat.com.br/as-melhores-features-do-es6-es7-e-es8/>
- Download Insomnia (para testar API REST): <https://insomnia.rest/download/>

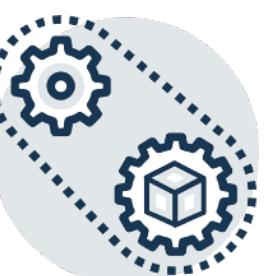


Inicializando o projeto



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: zsh + - ×
viniciuscardosogarcia in easylanding-e-server on ✘ master
> npm init -y
Wrote to /Users/vinicio cardosogarcia/GitHub/easylanding-e-server/package.json:

{
  "name": "easylanding-e-server",
  "version": "1.0.0",
  "description": "An AirBnB-like application development to learn Node + Express",
  "main": "index.js",
  "scripts": {
    "test": "echo \\"$Error: no test specified\\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/vinicio3w/easylanding-e-server.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/vinicio3w/easylanding-e-server/issues"
  },
  "homepage": "https://github.com/vinicio3w/easylanding-e-server#readme"
}
```

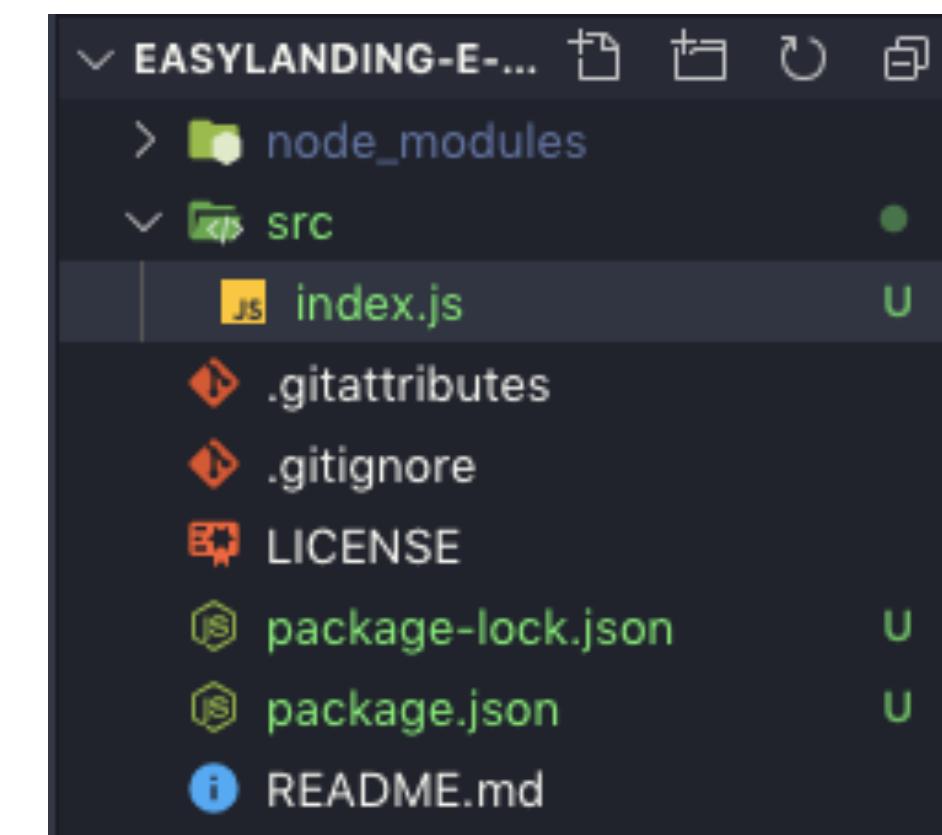


Instalando o express

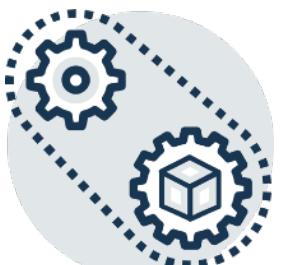
- Agora você poderá ver uma pasta `node_modules` criada na raiz do seu projeto
- A adição de `--save` salvará o pacote na sua lista de dependências, localizada no `package.json`, no diretório do projeto
- O pacote `body-parser` vai ajudar fazendo com que o node entenda as requisições recebendo as informações em JSON e também entendendo os parâmetros na URI
- Crie a pasta `scr` e o arquivo `index.js`

```
viniciuscardosogarcia in easylanding-e-server on ✘ master [?]
> npm install express --save
npm notice created a lockfile as package-lock.json. You should commit this file.
+ express@4.17.1
added 50 packages from 37 contributors and audited 126 packages in 9.808s
found 0 vulnerabilities
```

```
viniciuscardosogarcia in easylanding-e-server on ✘ master [?] took 14s
> npm install body-parser --save
+ body-parser@1.19.0
updated 1 package and audited 158 packages in 5.089s
found 0 vulnerabilities
```

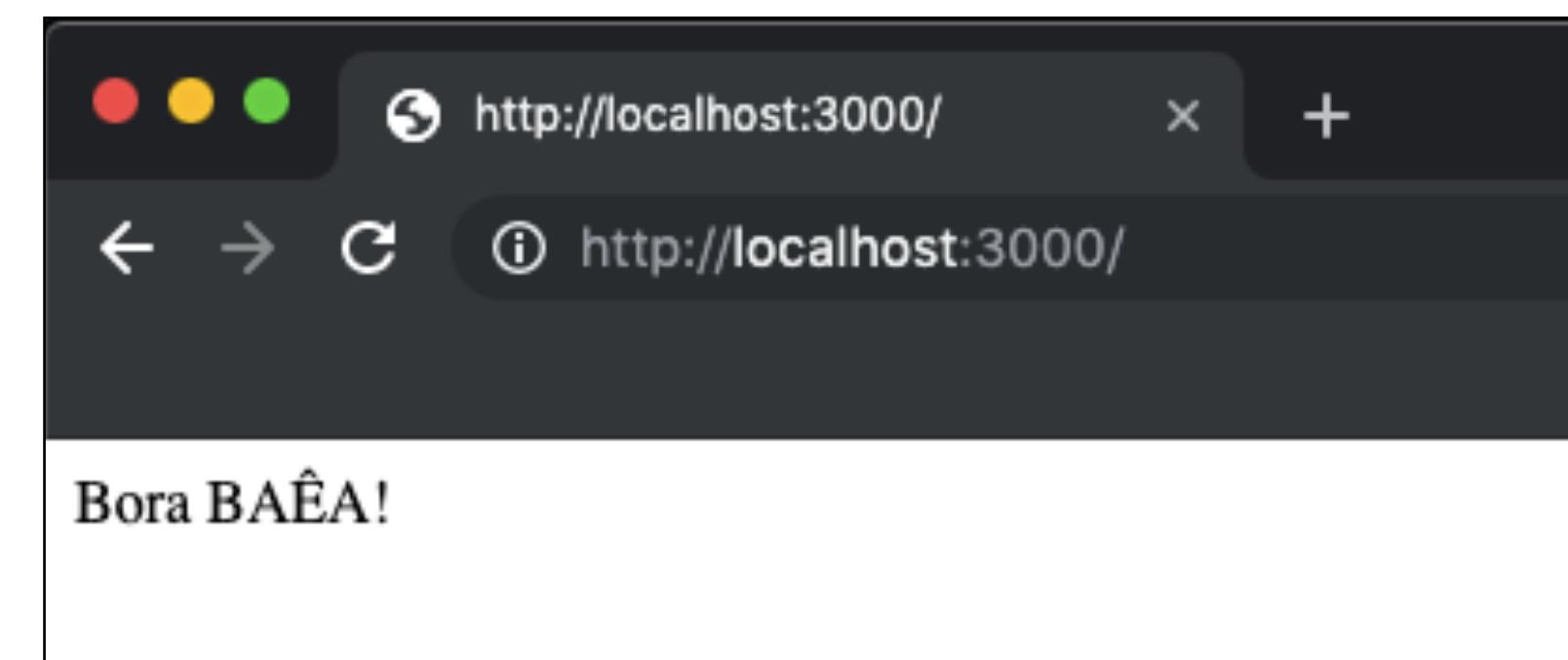


Definindo o express



```
index.js  X
src > index.js > ...
1 // Importando o express e o body-parser usando constante e require do ES6
2 const express = require('express');
3 const bodyParser = require('body-parser');
4
5 // Criando a aplicação
6 const app = express();
7
8 /** Indicar para a minha aplicação que vou usar o body-parser
9  * A primeira função é informar que vou utilizar a função json para entender as
10 * informações que vão chegar em formato JSON.
11 * A segunda é para ela entender quando eu passar parâmetros na URL.
12 */
13 app.use(bodyParser.json());
14 app.use(bodyParser.urlencoded({ extended: false}));
15
16 // Uma rota simples na raiz
17 app.get('/', (req, res) => {
18   res.send('Bora BAÊA!');
19 });
20
21 // Vou informar qual porta a aplicação vai "escutar"
22 app.listen(3000, () => {
23   console.log('Aplicação de exemplo escutando na porta 3000!');
24 });
```

```
viniciuscardosogarcia in easylanding-e-server on ↗ master [?] took 8s
> node src/index.js
Aplicação de exemplo escutando na porta 3000!
[]
```



Criando a conexão com o Banco

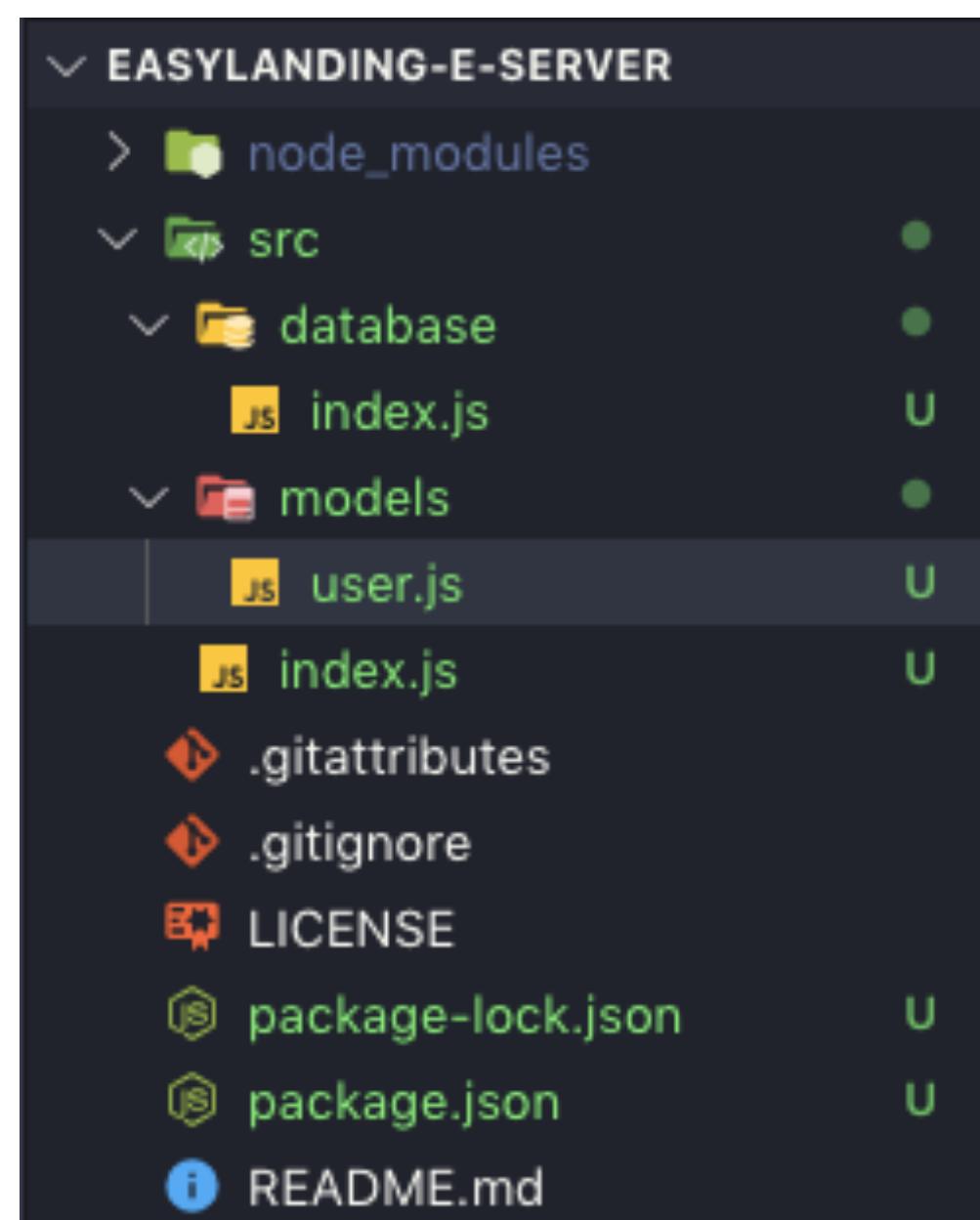
```
src > database > index.js > ...
1 //Importar o mongoose
2 const mongoose = require('mongoose');
3
4 //Indicar a URL de conexão com o banco de dados e como vai se dar a conexão
5 mongoose.connect('mongodb://localhost/easylandingdb', {useMongoClient: true});
6
7 //Precisamos indicar qual a classe de promise o mongoose vai usar
8 mongoose.Promise = global.Promise;
9
10 module.exports = mongoose;
```

JavaScript: Resolvendo o problema da programação assíncrona com o uso de Promises

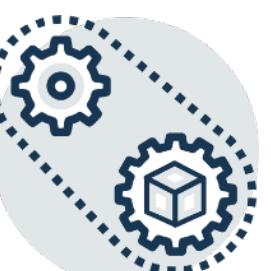
Documentação do objeto Promise



Criando o Model



```
src > models > User.js > ...
1 // Importar o mongoose
2 const mongoose = require('../database');
3
4 // Definição do Schema (arquitetura de informação no DB para a entidade User)
5 const UserSchema = new mongoose.Schema({
6   username: {
7     type: String,
8     required: true,
9   },
10  email: {
11    type: String,
12    unique: true,
13    required: true,
14    lowercase: true,
15  },
16  password: {
17    type: String,
18    required: true,
19    select: false,
20  },
21  createdAt: {
22    type: Date,
23    default: Date.now,
24  },
25});
26
27 const User = mongoose.model('User', UserSchema);
28
29 module.exports = User;
```

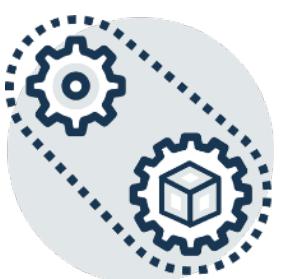


Controlador de autenticação

```
src > controllers > SessionController.js > ...
1 const express = require('express');
2 //Importando o nosso modelo User
3 const User = require('../models/User');
4
5 // Para podermos definir e trabalhar com as nossas rotas
6 const router = express.Router();
7 /**
8 * Rota de cadastro. Uso do async (a partir do node 9) para tratar promises com
9 * o await.
10 */
11
12 router.post('/register', async (req, res) => {
13   try {
14     const user = await User.create(req.body); //await para esperar o create
15     return res.send({ user });
16   } catch(err){
17     return res.status(400).send({ error: 'Registration failed' });
18   }
19 });
20
21 /**
22 * Definindo o router para ser utilizado dentro do app. Toda vez que o usuário
23 * acessar /session nosso router vai ser chamado. Com isso teremos uma rota
24 * /session/register
25 */
26 module.exports = app => app.use('/session', router);
```

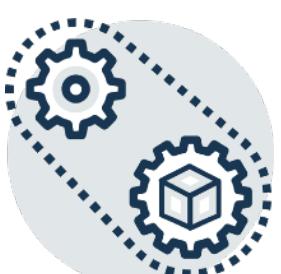
```
src > index.js > ...
1 //Importando o express e o body-parser usando constante e require do ES6
2 const express = require('express');
3 const bodyParser = require('body-parser');
4
5 //Criando a aplicação
6 const app = express();
7
8 /** Indicar para a minha aplicação que vou usar o body-parser
9 * A primeira função é informar que vou utilizar a função json para entender as
10 * informações que vão chegar em formato JSON.
11 * A segunda é para ela entender quando eu passar parâmetros na URL.
12 */
13 app.use(bodyParser.json());
14 app.use(bodyParser.urlencoded({ extended: false}));
15
16 //Referenciando o nosso controlador de autenticação
17 require('../controllers/SessionController')(app);
18
19 //Uma rota simples na raiz
20 app.get('/', (req, res) => {
21   res.send('Bora BAÊA!');
22 });
23
24 // Vou informar qual porta a aplicação vai "escutar"
25 app.listen(3000, () => {
26   console.log('Aplicação de exemplo escutando na porta 3000!');
27 });
```

Design Pattern com Async/Await em Node.js



Testando no Insomnia

POST base_url /session/register	Send	400 Bad Request	TIME 32.6 ms	SIZE 31 B
Body ▾ Auth ▾ Query Header Docs	Preview ▾	Header 6 Cookie Timeline	1▼ { 2 "error": "Registration failed" 3 }	
POST base_url /session/register	Send	200 OK	TIME 89.5 ms	SIZE 158 B
JSON ▾ Auth ▾ Query Header 1 Docs	Preview ▾	Header 6 Cookie Timeline	1▼ { 2 "user": { 3 "_id": "5d62f54f79ece056b597977d", 4 "username": "vinicius", 5 "email": "vcg@cin.ufpe.br", 6 "password": "123456", 7 "createdAt": "2019-08-25T20:53:35.781Z", 8 "__v": 0 9 } 10 }	



Criptografando dados sensíveis

```
> npm install bcryptjs
+ bcryptjs@2.4.3
added 1 package from 6 contributors and audited 198 packages in 3.213s
found 0 vulnerabilities

src > models > User.js > ...
1 // Importar o mongoose
2 const mongoose = require('../database');
3 const bcryptjs = require('bcryptjs');
4
```

```
// Função do mongoose para dizer o que é para acontecer antes de ...
// Ao criar um usuário, o hash da senha vai acontecer automaticamente
UserSchema.pre('save', async function(next) {
    // quantas vezes o hash será gerado (o número de rounds)
    const hash = await bcryptjs.hash(this.password, 10);
    this.password = hash;

    next();
});

const User = mongoose.model('User', UserSchema);

module.exports = User;
```

The screenshot shows a POST request to `base_url/session/register`. The JSON input is:

```
1 {  
2   "username": "jackbauer",  
3   "email": "jackbauer@cin.ufpe.br",  
4   "password": "123456"  
5 }
```

The response is a 200 OK with the following details:

- TIME 184 ms
- SIZE 219 B
- Just Now

The response body is a JSON object containing the user's information and a hashed password:

```
1 {  
2   "user": {  
3     "_id": "5d65800d3dec0371919167f5",  
4     "username": "jackbauer",  
5     "email": "jackbauer@cin.ufpe.br",  
6     "password": "$2a$10$Ay28IAwhXVR16cWIP2YFteIQUZOsKpjdg8tEEYNQVVAECoix12.3C",  
7     "createdAt": "2019-08-27T19:10:05.769Z",  
8     "__v": 0  
9   }  
10 }
```

Melhorando o retorno de erro de usuário existente

```
src > controllers > SessionController.js > router.post('/register') callback
12   router.post('/register', async (req, res) => {
13     const { email } = req.body;
14     try {
15       //Se o email já existir, não deixar cadastrar o usuário e informar o erro
16       if (await User.findOne({ email })) {
17         return res.status(400).send({ error: 'User already exists' });
18       }
19       const user = await User.create(req.body); //await para esperar o create
20       //Não queremos retornar o password após o cadastro, mesmo criptografado
21       user.password = undefined;
22       return res.send({ user });
23     } catch(err){
24       return res.status(400).send({ error: 'Registration failed' });
25     }
26   });

```

POST `base_url` /session/register

Send **400 Bad Request** TIME 100 ms SIZE 31 B Just Now

JSON Auth Query Header 1 Docs

```
1 {  
2   "username": "jackbauer",  
3   "email": "jackbauer@cin.ufpe.br",  
4   "password": "123456"  
5 }
```

Preview Header 6 Cookie Timeline

```
1 {  
2   "error": "User already exists"  
3 }
```



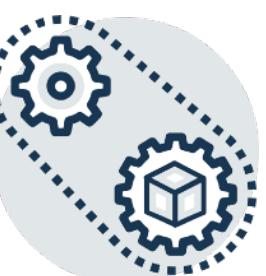
Construindo API's em Node + Express

Autenticação



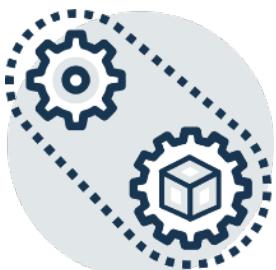
JSON Web Token (JWT) – O que é?

- Convenção **aberta** (RFC 7519)
- Possibilita de uma maneira **compacta** e **auto-contida** transmitir com **segurança** informação entre **duas partes** no formato de um objeto JSON
- Assinadas digitalmente por **criptografia** utilizando uma **chave secreta** (HMAC) ou por **chaves pública/privada** utilizando RSA ou ECDSA.



JSON Web Token (JWT) – Para que serve?

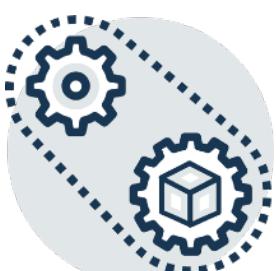
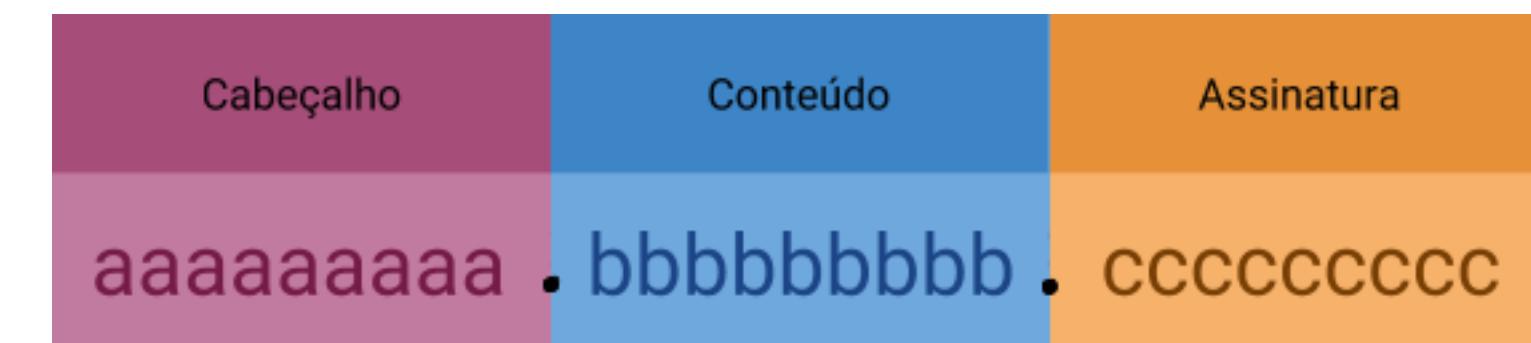
- **Autenticação**
 - O token é utilizado para verificar a identidade de um usuário e suas permissões
 - Normalmente incluem identificadores e informações não sensíveis do usuário
- **Troca de informação**
 - Por ser um meio seguro para duas aplicações conversarem, graças a maneira que os tokens são assinados digitalmente, eles garantem a identidade das partes envolvidas e se a informação não foi alterada no meio da caminho



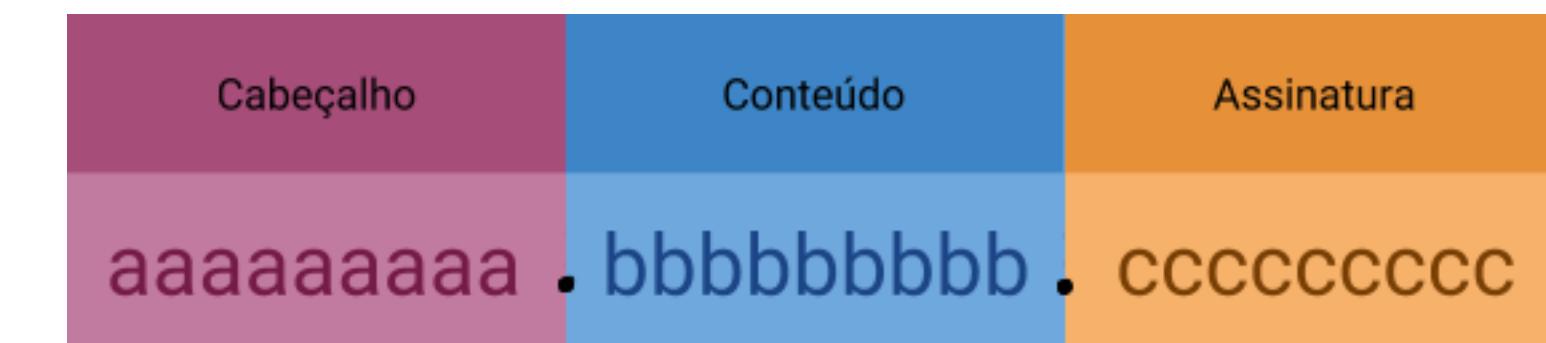
JSON Web Token (JWT) – Como funciona?

- **Cabeçalho**

- Comumente consiste em duas partes: O tipo de token, JWT na maioria das vezes, e o tipo de algoritmo de assinatura utilizado (HMAC SHA256 ou RSA)
- Essas informações são codificadas em base64 formando a primeira parte do token



JSON Web Token (JWT) – Como funciona?



- **Conteúdo**

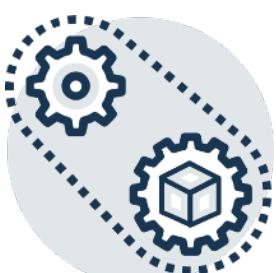
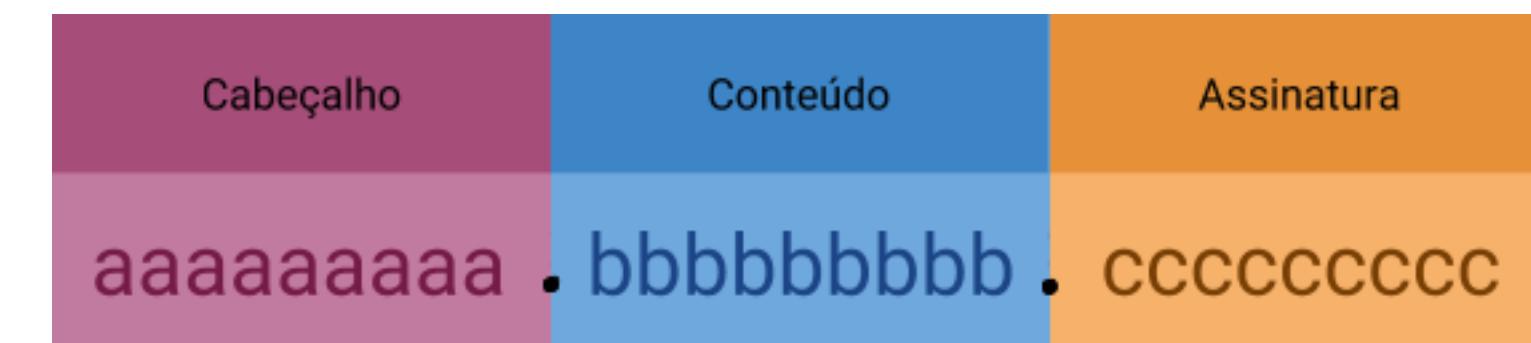
- Informações sobre a entidade que aquele token representa, informações de gestão e emissão do token, e por exemplo os dados de um usuário
 - **Registrado:** Podemos definir como informações comuns da estrutura do token, as propriedades do tipo registrado podem ter apenas 3 caracteres. As mais comuns são:
 - “iss”: Nome/URI do emissor do token.
 - “exp”: Tempo de validade do token.
 - **Público:** Como o nome já define bem, qualquer informação que pode ser considerada pública e não sensível, que faça sentido enviar nesse token, como por exemplo o nome ou algum identificador do usuário.
 - **Privado:** Qualquer informação privada que seja acordada entre as duas partes que vão utilizar esse token.



JSON Web Token (JWT) – Como funciona?

- **Assinatura**

- Basicamente temos a junção de todas as outras partes geradas: Cabeçalho codificado + Conteúdo codificado
- Todo esse conteúdo é criptografado com o método escolhido resultando então na assinatura

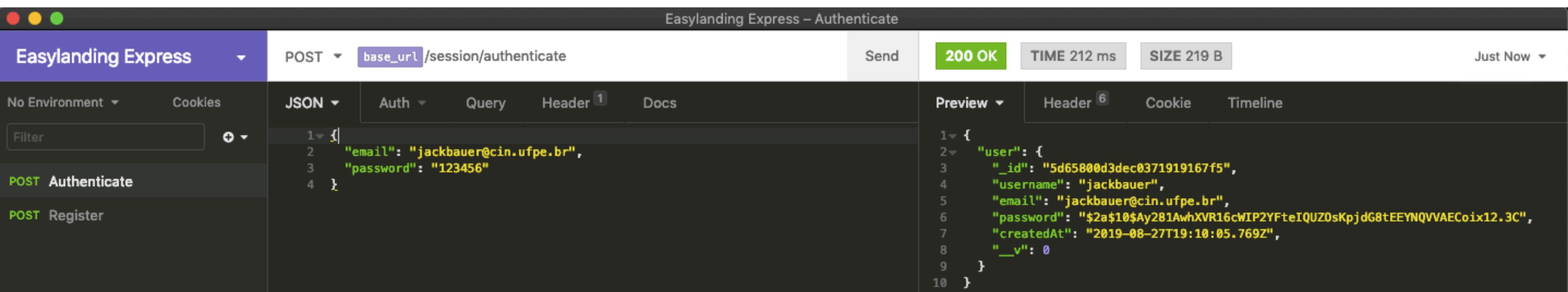


Adicionando a rota de autenticação

```
src > controllers > SessionController.js > router.post('/authenticate') callback
 4 | //Importando o módulo de encriptação e descriptação
 5 | const bcryptjs = require('bcryptjs');
 6 |
 7 | // Para podermos definir e trabalhar com as nossas rotas
 8 | const router = express.Router();
 9 |
10 | /**
11 | * Rota de cadastro. Uso do async (a partir do node 9) para tratar promises com
12 | * o await.
13 | */
14 |> router.post('/register', async (req, res) => { ...
15 | });
16 |
17 |
18 |
19 |
20 | /**
21 | * Rota de autenticação
22 | */
23 |> router.post('/authenticate', async(req, res) => {
24 |   const { email, password } = req.body;
25 |   //O password não vem no retorno de consultas, por isso usamos o método select
26 |   const user = await User.findOne({ email }).select('+password');
27 |   if (!user)
28 |     return res.status(400).send({ error: 'User not found' });
29 |   if (!await bcryptjs.compare(password, user.password))
30 |     return res.status(400).send({ error: 'Invalid password' });
31 |
32 |   res.send({ user });
33 | });
34 |
35 | You, a few seconds ago • Uncommitted changes
```



Testando no Insomnia



Easylanding Express – Authenticate

POST `base_url` /session/authenticate

Send **200 OK** TIME 212 ms SIZE 219 B Just Now

No Environment Cookies

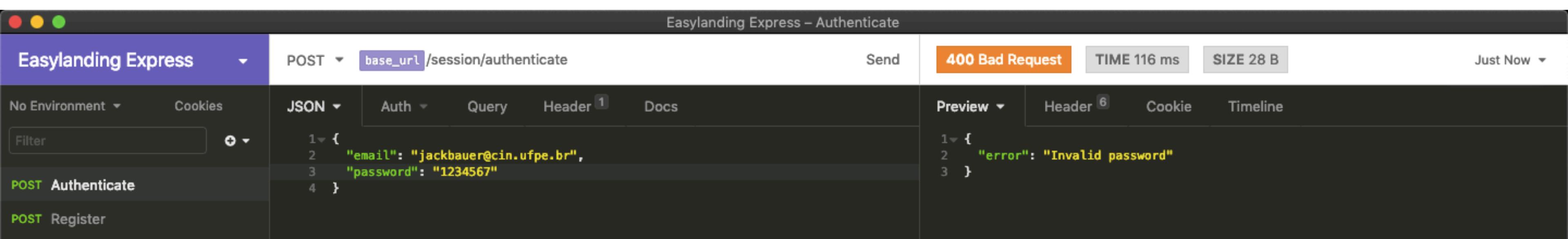
JSON Auth Query Header 1 Docs

Filter

```
1 {  
2   "email": "jackbauer@cin.ufpe.br",  
3   "password": "123456"  
4 }
```

Preview Header 6 Cookie Timeline

```
1 {  
2   "user": {  
3     "_id": "5d65800d3dec0371919167f5",  
4     "username": "jackbauer",  
5     "email": "jackbauer@cin.ufpe.br",  
6     "password": "$2a$10$Ay281AwhXVR16cWIP2YFteIQUZOsKpjdg8tEEYNQVVAECoix12.3C",  
7     "createdAt": "2019-08-27T19:10:05.769Z",  
8     "__v": 0  
9   }  
10 }
```



Easylanding Express – Authenticate

POST `base_url` /session/authenticate

Send **400 Bad Request** TIME 116 ms SIZE 28 B Just Now

No Environment Cookies

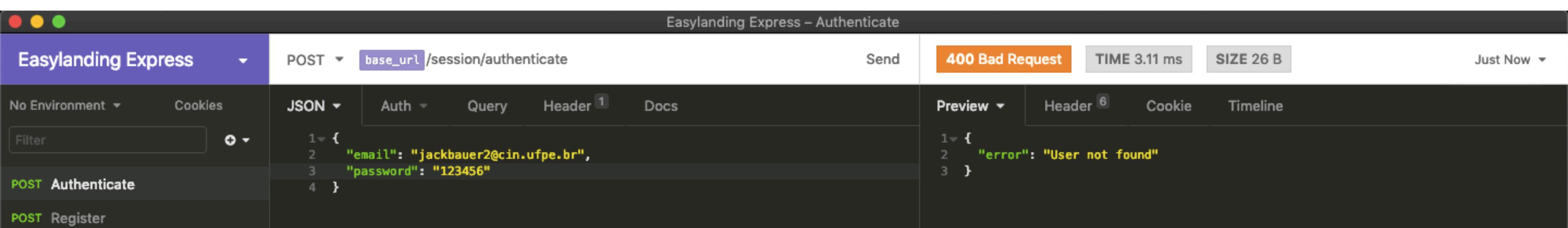
JSON Auth Query Header 1 Docs

Filter

```
1 {  
2   "email": "jackbauer@cin.ufpe.br",  
3   "password": "1234567"  
4 }
```

Preview Header 6 Cookie Timeline

```
1 {  
2   "error": "Invalid password"  
3 }
```



Easylanding Express – Authenticate

POST `base_url` /session/authenticate

Send **400 Bad Request** TIME 3.11 ms SIZE 26 B Just Now

No Environment Cookies

JSON Auth Query Header 1 Docs

Filter

```
1 {  
2   "email": "jackbauer2@cin.ufpe.br",  
3   "password": "123456"  
4 }
```

Preview Header 6 Cookie Timeline

```
1 {  
2   "error": "User not found"  
3 }
```



Adicionando token

```
viniciuscardosogarcia in easylanding-e-server on ✪ master [!] took 4s
> npm install jsonwebtoken
+ jsonwebtoken@8.5.1
updated 1 package and audited 215 packages in 3.195s
found 0 vulnerabilities
```

```
src > controllers > SessionController.js > router.post('/authenticate') callback
1 const express = require('express');
2 //Importando o nosso modelo User
3 const User = require('../models/User');
4 //Importando o módulo de encriptação e decriptação
5 const bcryptjs = require('bcryptjs');
6 //Importando o jwt
7 const jwt = require('jsonwebtoken');
8
9 const authConfig = require('../config/auth.json');
10
```

```
src > controllers > SessionController.js > router.post('/authenticate') callback
34 /**
35 * Rota de autenticação
36 */
37 router.post('/authenticate', async(req, res) => {
38   const { email, password } = req.body;
39   //O password não vem no retorno de consultas, por isso usamos o select
40   const user = await User.findOne({ email }).select('+password');
41   if (!user)
42     return res.status(400).send({ error: 'User not found' });
43   if (!await bcryptjs.compare(password, user.password))
44     return res.status(400).send({ error: 'Invalid password' });
45   user.password = undefined;
46   /**
47    * O primeiro parâmetro é o que vai ser efetivamente único, por isso usamos
48    * o id. O segundo parâmetro é um hash que deve ser único da aplicação,
49    * por isso vamos criar esse hash e salvar em /config/auth.json. O terceiro
50    * parâmetro é o tempo para o token expirar.
51   */
52   const token = jwt.sign({ id: user.id }, authConfig.secret, {
53     expiresIn: 86400, //um dia em segundos
54   });
55   res.send({ user, token });
56});
```

Easylanding Express – Authenticate

Easylanding Express POST base_url/session/authenticate Send 200 OK TIME 257 ms SIZE 327 B Just Now

No Environment Cookies

Filter

POST Authenticate

POST Register

JSON Auth Query Header 1 Docs

Preview Header 6 Cookie Timeline

```
1 {
2   "email": "jackbauer@cin.ufpe.br",
3   "password": "123456"
4 }
```

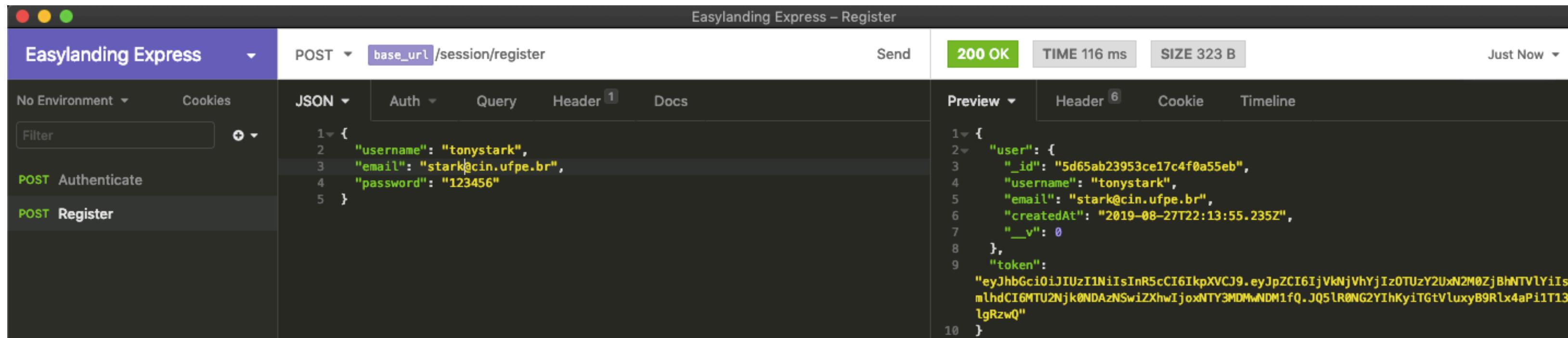
```
1 {
2   "user": {
3     "_id": "5d65800d3dec0371919167f5",
4     "username": "jackbauer",
5     "email": "jackbauer@cin.ufpe.br",
6     "createdAt": "2019-08-27T19:10:05.769Z",
7     "__v": 0
8   },
9   "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVkJU4MDBkM2R1YzAzNzE5MTkxNjdNSIsImhdCIEMTU2Njk0MTQ4NiwiZXhwIjoxNTY3MDI3ODg2fQ.BFbzHffrYjMdUyMG37SzPe9FeL1YbiTj9M8lGR8G60s"
```

```
src > config > auth.json > secret
1 {
2   "secret": "2bb2a795c1fe0ed42aaaf4aaee951a969"
3 }
```

Mas, ao cadastrar um novo usuário, não deveríamos gerar um token?

```
/**  
 * O primeiro parâmetro é o que vai ser efetivamente único, por isso usamos  
 * o id. O segundo parâmetro é um hash que deve ser único da aplicação,  
 * por isso vamos criar esse hash e salvar em /config/auth.json. O terceiro  
 * parâmetro é o tempo para o token expirar.  
 */  
  
function generateToken(params = {}){  
  return jwt.sign(params, authConfig.secret, {  
    expiresIn: 86400, //um dia em segundos  
  });  
}  
  
/**  
 * Rota de autenticação  
 */  
router.post('/authenticate', async(req, res) => {  
  const { email, password } = req.body;  
  //O password não vem no retorno de consultas, por isso usamos o método select  
  const user = await User.findOne({ email }).select('+password');  
  if (!user)  
    return res.status(400).send({ error: 'User not found' });  
  if (!await bcryptjs.compare(password, user.password))  
    return res.status(400).send({ error: 'Invalid password' });  
  user.password = undefined;  
  res.send({  
    user,  
    token: generateToken({ id: user.id })  
  });  
});
```

```
/**  
 * Rota de cadastro. Uso do async (a partir do node 9) para tratar promises com  
 * o await.  
 */  
router.post('/register', async (req, res) => {  
  const { email } = req.body;  
  try {  
    //Se o email já existir, não deixar cadastrar o usuário e informar o erro  
    if (await User.findOne({ email })) {  
      return res.status(400).send({ error: 'User already exists' });  
    }  
    const user = await User.create(req.body); //await para esperar o create  
    //Não queremos retornar o password após o cadastro, mesmo criptografado  
    user.password = undefined;  
    return res.send({  
      user,  
      token: generateToken({ id: user.id })  
    });  
  } catch(err){  
    return res.status(400).send({ error: 'Registration failed' });  
  }  
});
```



The screenshot shows the Easylanding Express API testing interface. A POST request is made to `/base_url/session/register`. The JSON payload is:

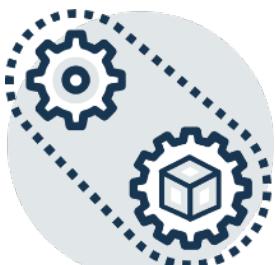
```
1 {  
2   "username": "tonystark",  
3   "email": "stark@cin.ufpe.br",  
4   "password": "123456"  
5 }
```

The response status is 200 OK, with a time of 116 ms and a size of 323 B. The response body is:

```
1 {  
2   "user": {  
3     "_id": "5d65ab23953ce17c4f0a55eb",  
4     "username": "tonystark",  
5     "email": "stark@cin.ufpe.br",  
6     "createdAt": "2019-08-27T22:13:55.235Z",  
7     "__v": 0  
8   },  
9   "token":  
10    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVhYjIzOTUzY2UxN2M0ZjBhNTVlYiIsImldC16MTU2Njk0NDAzNSwiZXhwIjoxNTY3MDMwNDM1fQ.JQ5lR0NG2YIhKyiTGTvluxyB9Rlx4aPi1T13IlgRzwQ"
```

E a validação do token?

- Até o momento não precisamos validar o token
- Para se cadastrar e autenticar, o usuário não estava logado
 - Não possuía um token ativo



32



Middleware para validar o token

```
> node_modules
  +-- src
    +-- config
      +-- auth.json
    +-- controllers
      +-- ProjectController.js
      +-- SessionController.js
    +-- database
      +-- index.js
    +-- middlewares
      +-- auth.js
    +-- models
      +-- User.js
      +-- index.js
    +-- .gitattributes
    +-- .gitignore
    +-- LICENSE
    +-- package-lock.json
    +-- package.json
    +-- README.md
```

```
src > controllers > ProjectController.js > ...
1  const express = require('express');
2  const authMiddleware = require('../middlewares/auth');
3
4  const router = express.Router();
5
6  router.use(authMiddleware);
7
8  router.get('/', (req, res) => {
9    res.send({ ok: true, user: req.userId});
10 });
11
12 module.exports = app => app.use('/projects', router);
```

GET `base_url /projects`

Send **200 OK** TIME 13.8 ms SIZE 45 B Just Now

Body Auth Query Header **1** Docs

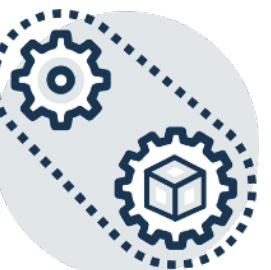
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ik

New Header

Preview Header Cookie Timeline

```
1 { "ok": true, "user": "5d65800d3dec0371919167f5" }
```

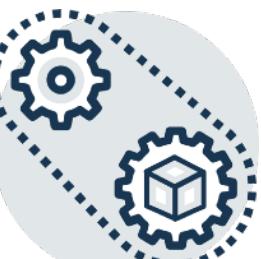
```
src > middlewares > auth.js > <unknown> > module.exports > jwt.verify() callback
1  const jwt = require('jsonwebtoken');
2  const authConfig = require('../config/auth.json');
3
4  // O next só vai ser chamado para ir para o próximo passo
5  // Vamos interceptar o usuário aqui no middleware
6  module.exports = (req, res, next) => {
7    // Vou buscar o header de autorização (authorization) na requisição
8    const authHeader = req.headers.authorization;
9
10   // Se o header não foi enviado, já retorno um erro
11   if (!authHeader)
12     return res.status(401).send({ error: 'No token provided' });
13
14   // Verificar se o token está no formato correto: Bearer <token>
15   const parts = authHeader.split(' ');
16   if (!parts.length == 2)
17     return res.status(401).send({ error: 'Token error' });
18
19   const [ scheme, token ] = parts;
20   if (!/^Bearer$/i.test(scheme))
21     return res.status(401).send({ error: 'Token malformed' });
22
23   jwt.verify(token, authConfig.secret, (err, decoded) => {
24     if (err) return res.status(401).send({ error: 'Token invalid' });
25     req.userId = decoded.id;
26     return next();
27   });
28 }
```





Construindo API's em Node + Express

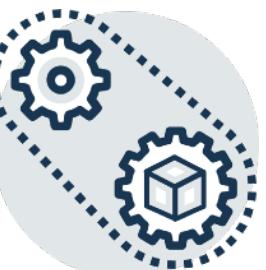
CRUD e relacionamentos com MongoDB



Criando as entidades

```
src > models > Project.js > ...
1 // Importar o mongoose
2 const mongoose = require('../database');
3 const bcryptjs = require('bcryptjs');
4
5 // Definição do Schema (arquitetura de informação no DB para a entidade Project)
6 const ProjectSchema = new mongoose.Schema({
7   title: {
8     type: String,
9     required: true,
10 },
11   description: {
12     type: String,
13     required: true,
14 },
15   user: {
16     type: mongoose.Schema.Types.ObjectId,
17     ref: 'User',
18     required: true
19 },
20   tasks: [
21     type: mongoose.Schema.Types.ObjectId,
22     ref: 'Task'
23 ],
24   createdAt: {
25     type: Date,
26     default: Date.now,
27 },
28 });
29
30 const Project = mongoose.model('Project', ProjectSchema);
31
32 module.exports = Project;
```

```
src > models > Task.js > ...
1 const mongoose = require('../database');
2 const bcryptjs = require('bcryptjs');
3
4 const TaskSchema = new mongoose.Schema({
5   title: {
6     type: String,
7     required: true,
8 },
9   project: {
10   type: mongoose.Schema.Types.ObjectId,
11   ref: 'Project',
12   required: true,
13 },
14   assignedTo: {
15   type: mongoose.Schema.Types.ObjectId,
16   ref: 'User',
17   required: true,
18 },
19   completed: {
20   type: Boolean,
21   required: true,
22   default: false,
23 },
24   createdAt: {
25   type: Date,
26   default: Date.now,
27 },
28 });
29
30 const Task = mongoose.model('Task', TaskSchema);
31
32 module.exports = Task;
```



Trabalhando no controlador

```
//Create
router.post('/', async (req, res) => {
  try {
    const project = await Project.create({ ... req.body, user: req.userId });
    return res.send({ project });
  } catch (err) {
    return res.status(400).send({ error: 'Error creating new project' });
  }
});
```

POST `base_url /projects` Send **200 OK** TIME 60.1 ms SIZE 200 B 4 Minutes Ago

JSON Bearer Query Header 1 Docs

```
1 {  
2   "title": "Novo Projeto",  
3   "description": "Descrição do Projeto"  
4 }
```

Preview Header 6 Cookie Timeline

```
1 {  
2   "project": {  
3     "tasks": [],  
4     "_id": "5d680d958835771f88a0715b",  
5     "title": "Novo Projeto",  
6     "description": "Descrição do Projeto",  
7     "user": "5d65800d3dec0371919167f5",  
8     "createdAt": "2019-08-29T17:38:29.674Z",  
9     "__v": 0  
10  }  
11 }
```

```
//List
router.get('/', async (req, res) => {
  try {
    const projects = await Project.find();
    return res.send([projects]);      You, a few seconds ago + Uncommitted changes
  } catch (error) {
    return res.status(400).send({ error: 'Error listing projects' });
  }
});
```

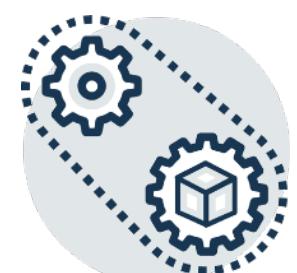
```
//Show
router.get('/:projectId', async (req, res) => {
  try {
    //Eager loading
    const project = await Project.findById(req.params.projectId).populate('user');
    return res.send({ project });
  } catch (error) {
    return res.status(400).send({ error: 'Error listing project' });
  }
});
```

```
200 OK TIME 111 ms SIZE 313 B
```

Preview Header 6 Cookie Timeline

```
1 {  
2   "projects": [  
3     {  
4       "tasks": [],  
5       "_id": "5d6813c96e316a214f3cf088",  
6       "title": "Novo Projeto",  
7       "description": "Descrição do Projeto",  
8       "user": {  
9         "_id": "5d65800d3dec0371919167f5",  
10        "username": "jackbauer",  
11        "email": "jackbauer@cin.ufpe.br",  
12        "createdAt": "2019-08-27T19:10:05.769Z",  
13        "__v": 0  
14      },  
15      "createdAt": "2019-08-29T18:04:57.700Z",  
16      "__v": 0  
17    }  
18  ]  
19 }
```

```
//Delete
router.delete('/:projectId', async (req, res) => {
  try {
    //Eager loading
    await Project.findByIdAndRemove(req.params.projectId);
    return res.send();
  } catch (error) {
    return res.status(400).send({ error: 'Error removing project' });
  }
});
```



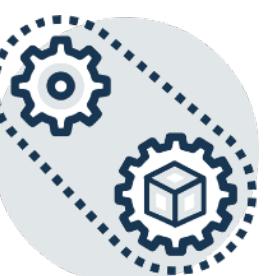
Adicionando as tasks

The screenshot shows a POST request to `base_url /projects`. The request body is a JSON object representing a project with tasks:

```
//Create
router.post('/', async (req, res) => {
  try {
    const { title, description, tasks } = req.body;
    const project = await Project.create({title, description, user: req.userId});
    await Promise.all(tasks.map(async task => {
      const projectTask = new Task({ ...task, project: project._id });
      await projectTask.save();
      project.tasks.push(projectTask);
    }));
    //Como estamos adicionando novas tasks no projeto, precisamos atualizar ele
    await project.save();
    return res.send({ project });
  } catch (err) {
    return res.status(400).send({ error: 'Error creating new project' });
  }
});
```

The response is a 200 OK status with a response time of 192 ms and a size of 604 B. The response body is a JSON object containing the created project with its tasks:

```
1< {
2  "project": {
3    "tasks": [
4      {
5        "completed": false,
6        "_id": "5d681b630a9fb3248c32f45a",
7        "title": "Outra tarefa",
8        "assignedTo": "5d65800d3dec0371919167f5"
9      },
10     {
11       "completed": false,
12       "title": "Nova tarefa",
13       "assignedTo": "5d65800d3dec0371919167f5"
14     }
15   ]
16 },
17 "title": "Outro Novo Projeto",
18 "description": "Descrição do Projeto",
19 "user": "5d65800d3dec0371919167f5",
20 "createdAt": "2019-08-29T18:37:23.917Z",
21 "v": 0
22 },
23 "_id": "5d681b630a9fb3248c32f459",
24 "title": "Nova tarefa",
25 "assignedTo": "5d65800d3dec0371919167f5",
26 "project": "5d681b630a9fb3248c32f458",
27 "createdAt": "2019-08-29T18:37:23.916Z",
28 "v": 0
29 }
30 }
```



Melhorando a exibição das tasks

```
//List
router.get('/', async (req, res) => {
  try {
    //Eager loading
    const projects = await Project.find().populate(['user', 'tasks']);
    return res.send({ projects });
  } catch (error) {
    return res.status(400).send({ error: 'Error listing projects' });
  }
});

//Show
router.get('/:projectId', async (req, res) => {
  try {
    //Eager loading
    const project = await Project.findById(req.params.projectId).populate(['user', 'tasks']);
    return res.send({ project });
  } catch (error) {
    return res.status(400).send({ error: 'Error listing project' });
  }
});
```

200 OK TIME 7.15 ms SIZE 714 B

Preview Header 6 Cookie Timeline

```
1< {
2  "project": {
3    "tasks": [
4      {
5        "completed": false,
6        "_id": "5d681b630a9fb3248c32f45a",
7        "title": "Outra tarefa",
8        "assignedTo": "5d65800d3dec0371919167f5",
9        "project": "5d681b630a9fb3248c32f458",
10       "createdAt": "2019-08-29T18:37:23.917Z",
11       "__v": 0
12     },
13    {
14      "completed": false,
15      "_id": "5d681b630a9fb3248c32f459",
16      "title": "Nova tarefa",
17      "assignedTo": "5d65800d3dec0371919167f5",
18      "project": "5d681b630a9fb3248c32f458",
19      "createdAt": "2019-08-29T18:37:23.916Z",
20      "__v": 0
21    },
22  ],
23  "_id": "5d681b630a9fb3248c32f458",
24  "title": "Outro Novo Projeto",
25  "description": "Descrição do Projeto",
26  "user": {
27    "_id": "5d65800d3dec0371919167f5",
28    "username": "jackbauer",
29    "email": "jackbauer@cin.ufpe.br",
30    "createdAt": "2019-08-27T19:10:05.769Z",
31    "__v": 0
32  },
33  "createdAt": "2019-08-29T18:37:23.906Z",
34  "__v": 1
35 }
36 }
```



Atualizar projeto

```
//Update
router.put('/:projectId', async (req, res) => {
  try {
    const { title, description, tasks } = req.body;
    const project = await Project.findByIdAndUpdate(req.params.projectId, {
      title,
      description
    }, { new: true }); //o mongoose vai retornar o valor atualizado
    //Vamos apagar todas as tasks antes de criá-las novamente
    project.tasks = [];
    await Task.remove({project: project._id});           You, a few seconds ago • Unc
    await Promise.all(tasks.map(async task => {
      const projectTask = new Task({ ...task, project: project._id });
      await projectTask.save();
      project.tasks.push(projectTask);
    }));
    //Como estamos adicionando novas tasks no projeto, precisamos atualizar ele
    await project.save();
    return res.send({ project });
  } catch (err) {
    return res.status(400).send({ error: 'Error updating project' });
  }
});
```



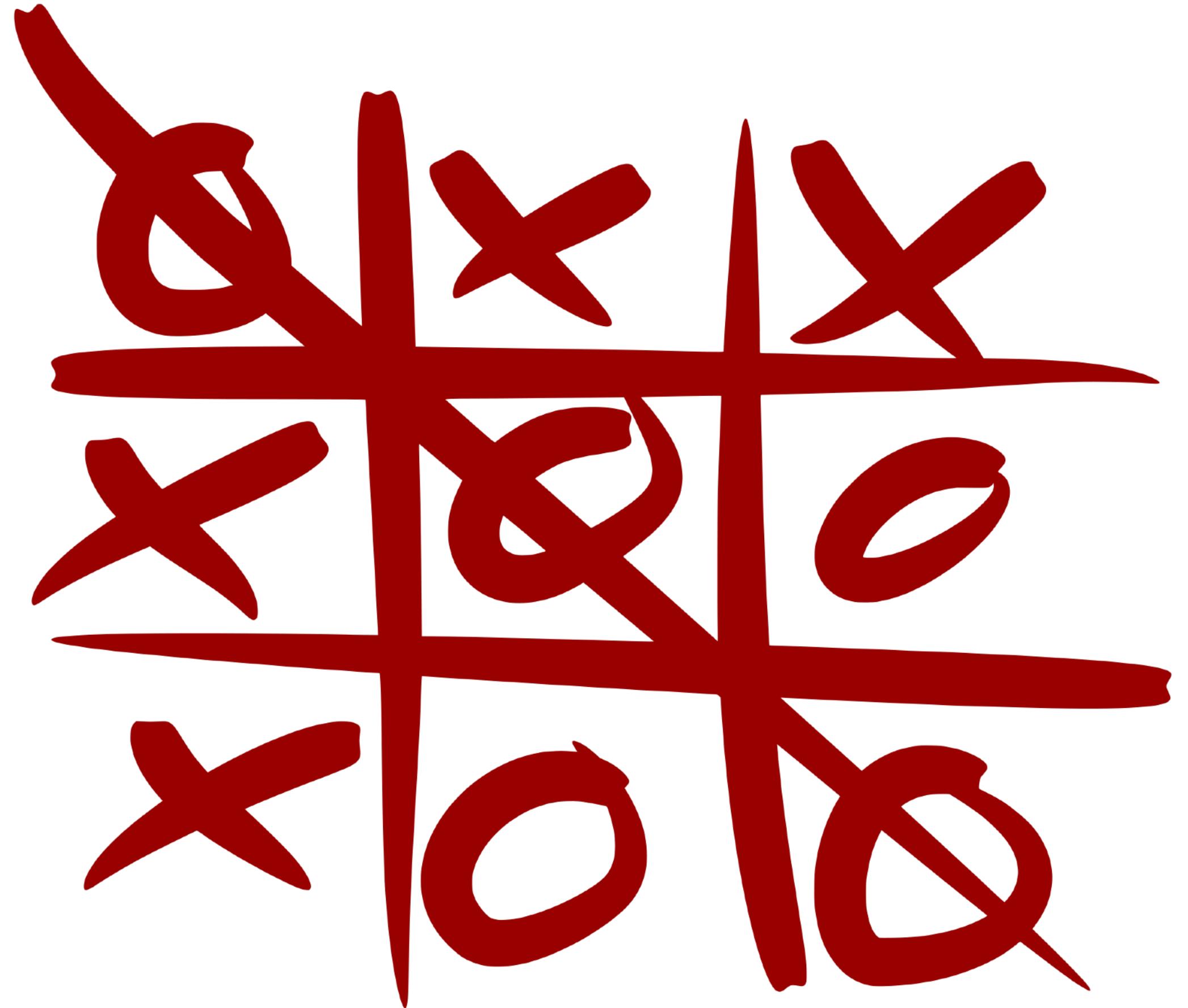


Desafio

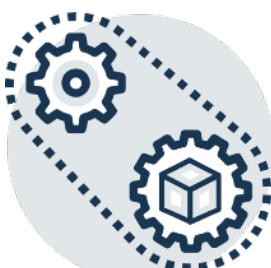


40





Juntando tudo: um jogo da velha RESTful



41



Questões de projeto

- Que estado precisará ser preservado nas requisições (ou seja, na sessão)?
- Quais recursos e operações queremos?
 - Recurso primário
 - Operação a ser realizada
 - Dados necessários, efeitos colaterais, tipos de erros

