

# Introdução NODEJS: Criando um projeto

Prof. Me. Ewerton José da Silva

# O que é o node js?

Node js é muito popular entre os desenvolvedores, utilizado para desenvolver sistemas utilizando uma linguagem familiar para a maioria dos desenvolvedores, o javascript. O nodejs se tornou famoso por ser simples de aprender e por construir aplicações de baixo custo e de alta escala.

# O que é uma rest api?

REST significa Representational State Transfer. Em português, Transferência de Estado Representacional. Trata-se de uma abstração da arquitetura da Web. Resumidamente, o REST consiste em princípios/regras/constraints que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas. Desta forma, permitindo, por exemplo, que aplicações se comuniquem.

## REST X RESTFUL

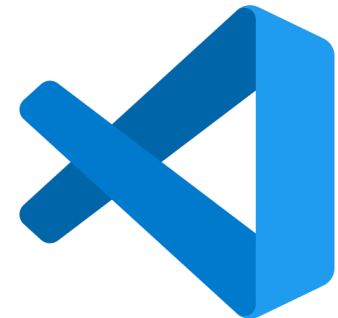
# Métodos de requisição HTTP que você precisa saber

Antes de começarmos a falar sobre o desenvolvimento do projeto em si, é necessário que conheçamos algumas metodos http importantes do protocolo (mas calma, não é nada tão complexo de se entender).

- GET- Utilizado para o busca de dados de algum lugar.
- POST- Utilizado para enviar dados para algum lugar.
- PUT/PATCH- Atualiza os dados com os novos dados enviados.
- DELETE- Remove dados .

# Ferramentas utilizadas

- Node - <https://nodejs.org/en/blog/release/v14.17.3/>
- VSCode
- Thunder Client
- Mysql e mysql workbench
- GitHub Desktop

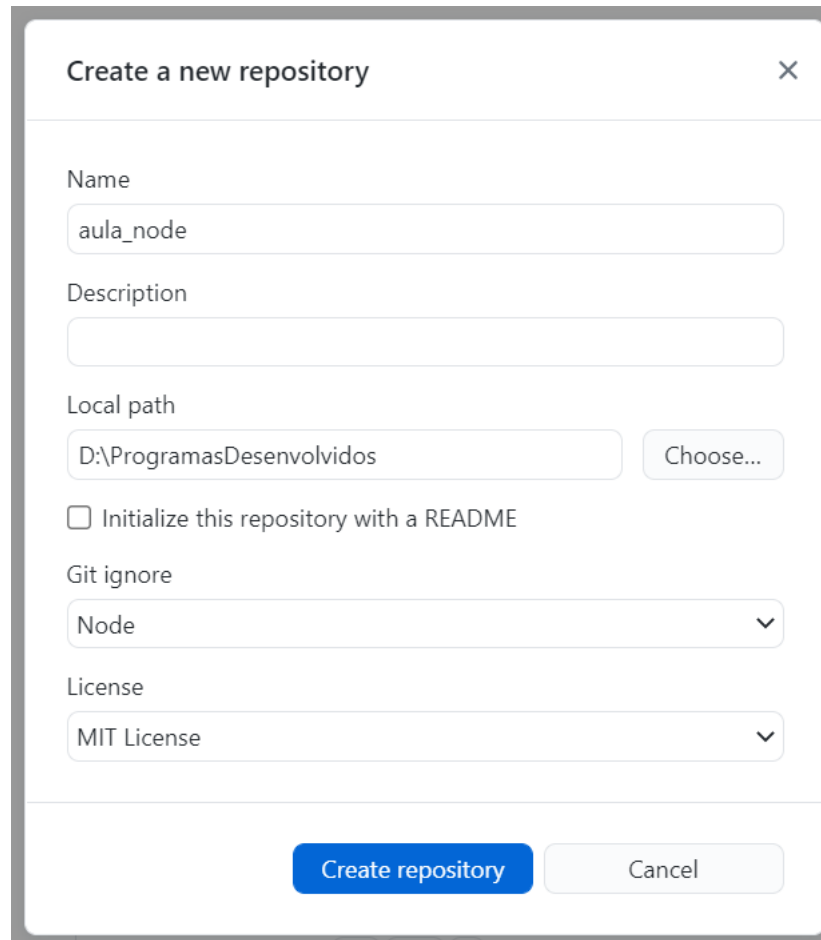


Verifique se o node e o npm estão corretamente instalados, ao inserir os comando apontados abaixo, devemos ter como retorno a versão instalada.

```
C:\Users\Ewerton>node -v ←  
v14.17.4
```

```
C:\Users\Ewerton>npm -v ←  
8.5.5
```

# Crie um repositório com o GitHub Desktop



The image shows a screenshot of the 'Create a new repository' dialog box in GitHub Desktop. The dialog has a title bar with a close button (X) in the top right corner. The main content area contains several input fields and options:

- Name:** A text input field containing the text 'aula\_node'.
- Description:** An empty text input field.
- Local path:** A text input field containing the path 'D:\ProgramasDesenvolvidos', followed by a 'Choose...' button.
- Initialize this repository with a README:** An unchecked checkbox.
- Git ignore:** A dropdown menu with 'Node' selected.
- License:** A dropdown menu with 'MIT License' selected.

At the bottom of the dialog, there are two buttons: a blue 'Create repository' button and a grey 'Cancel' button.

# Criando o projeto inicial

Agora que você já tem o nodejs e o seu editor de texto instalado em sua máquina, podemos começar o desenvolvimento do nosso projeto.

Abra o cmd se você estiver utilizando windows ou o terminal se for linux. Acesse a pasta do seu projeto criado com o GitHub Desktop e dentro da pasta execute o seguinte comando.

```
npm init -y
```



Após a execução é possível observar que um novo arquivo chamado package.json com as informações parecidas com as que estão na imagem abaixo foi adicionado ao projeto, esse arquivo é necessário para o nosso ponto de partida, nele conterá algumas informações importantes referente ao nosso projeto como descrição do projeto versão, scripts que podemos configurar e outros.

```
D:\ProgramasDesenvolvidos\aula_node>npm init -y
Wrote to D:\ProgramasDesenvolvidos\aula_node\package.json:

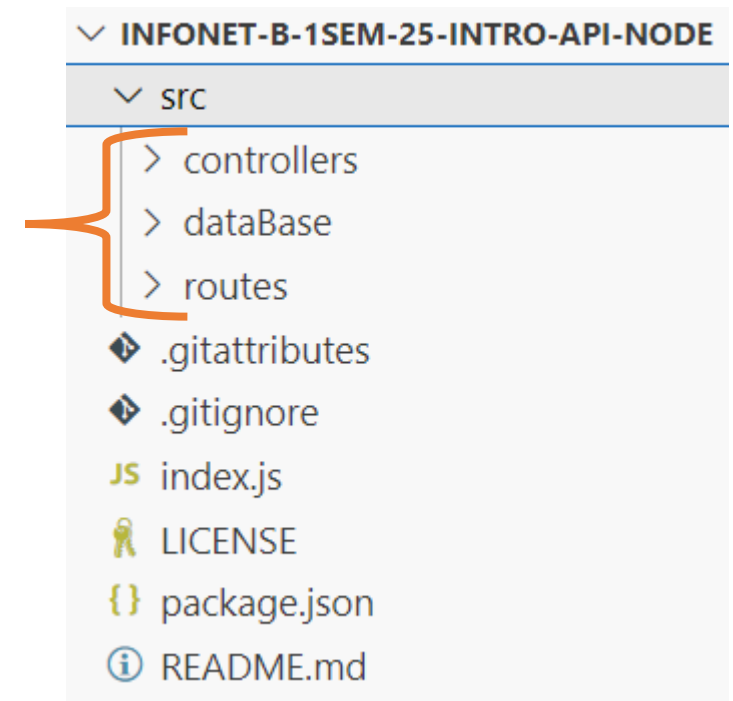
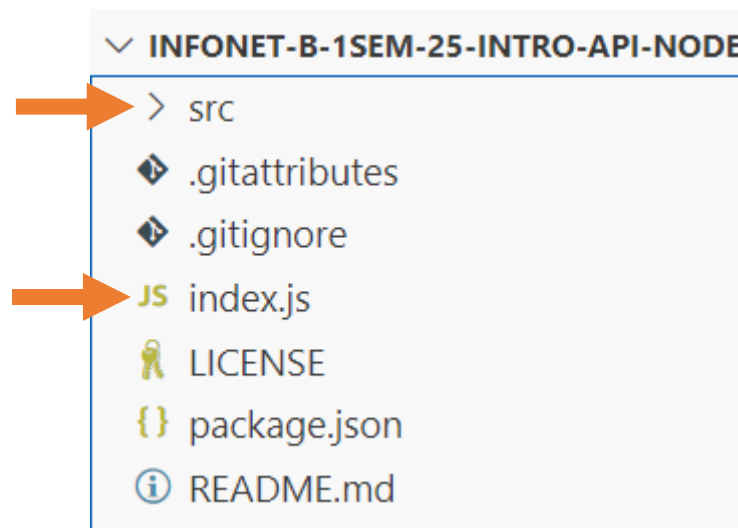
{
  "name": "aula_node",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

# Organizando estrutura de pastas

Após gerar o arquivo package.json você pode abrir a pasta do seu projeto diretamente no VS Code ou no seu editor de texto, se ainda estiver no cmd (ou no terminal) basta executar o comando “code .” para que abra uma janela do vs code na pasta atual.

Criaremos agora algumas pastas e arquivos para organizar melhor o projeto. A estrutura final com as pastas criadas ficará da seguinte forma.

- Um arquivo com o nome “index.js”
- Uma pasta com o nome “src”
- Três pastas dentro de src: controllers, database e routes



# Dependências necessárias para o projeto

Precisamos de algumas dependências importantes para que nosso projeto funcione corretamente e para isso precisaremos instalá-los, segue abaixo a lista do que precisaremos adicionar ao nosso projeto, basta executar dentro da pasta do nosso projeto os seguintes comandos:

- `npm i express`
- `npm i mysql2`
- `npm i cors`
- `npm i nodemon`
- `npm i dotenv`

# Express

O Express.js é um framework web para o Node.js que simplifica o desenvolvimento de aplicativos web e APIs. Ele oferece uma camada abstrata sobre o HTTP, facilitando a criação de servidores web e o tratamento de solicitações e respostas HTTP de maneira eficiente. O Express tem várias finalidades e benefícios importantes no desenvolvimento de aplicativos Node.js:

1. Roteamento: O Express permite definir rotas que mapeiam URLs específicas para funções que devem ser executadas quando essas URLs são acessadas. Isso facilita a criação de APIs RESTful e o tratamento de diferentes solicitações HTTP (GET, POST, PUT, DELETE, etc.).
2. Middleware: O Express utiliza um sistema de middleware que permite adicionar funções intermediárias para processar solicitações e respostas HTTP antes que elas atinjam suas rotas finais. Isso é útil para tarefas como autenticação, validação de dados, registro de logs e manipulação de erros.
3. Gerenciamento de Sessão e Cookies: O Express oferece suporte para gerenciamento de sessões e cookies, o que é importante para criar aplicativos web autenticados e personalizados.
4. Suporte a Templates: O Express pode ser usado com mecanismos de modelos como o EJS e o Handlebars para gerar dinamicamente conteúdo HTML a ser enviado como resposta para as solicitações. Isso é útil para criar páginas da web dinâmicas.

5. Suporte a APIs RESTful: O Express é frequentemente usado para criar APIs RESTful que permitem que aplicativos front-end e outros serviços se comuniquem com o servidor de maneira eficiente e consistente.
6. Roteamento de Páginas: Embora seja comumente usado para criar APIs, o Express também pode ser usado para criar aplicativos web tradicionais com roteamento de páginas, onde as páginas HTML são geradas e renderizadas no servidor.
7. Extensibilidade: O Express é altamente extensível, o que significa que você pode adicionar pacotes e middleware de terceiros para estender suas funcionalidades de acordo com as necessidades do seu projeto.
8. Comunidade Ativa: O Express tem uma comunidade ativa e é amplamente adotado na indústria, o que significa que você pode encontrar uma ampla variedade de recursos, pacotes e exemplos para ajudar no desenvolvimento.
9. Desempenho: O Express é conhecido por sua eficiência e bom desempenho. Ele é leve e não impõe muita sobrecarga, tornando-o uma escolha sólida para criar aplicativos escaláveis.

# cors (Cross-Origin Resource Sharing)

É uma middleware que permite controlar a política de segurança do navegador relacionada às solicitações de recursos entre diferentes origens (origins) em aplicativos web. Ela é especialmente útil quando você está construindo APIs RESTful ou serviços web que precisam ser acessados por clientes de diferentes origens, como navegadores em domínios diferentes.

Aqui está uma explicação mais detalhada de para que serve a biblioteca "cors" no Node.js:

1. Controle de Segurança de Origem Cruzada (CORS): O CORS é um mecanismo de segurança implementado pelos navegadores da web para evitar solicitações não autorizadas entre diferentes origens. Uma origem é definida pela combinação de esquema (HTTP/HTTPS), domínio (por exemplo, example.com), e porta (opcional). Quando um navegador faz uma solicitação para um servidor em uma origem diferente, ele envia um cabeçalho "Origin" com a solicitação para informar ao servidor de destino sobre a origem da solicitação.



2. Problemas de Segurança: Sem a configuração apropriada, os navegadores web bloqueiam solicitações feitas a partir de origens diferentes (mesmo protocolo, domínio e porta diferentes) por razões de segurança. Isso significa que, por padrão, um cliente da web não pode acessar recursos de um servidor em uma origem diferente.
3. Solução com o "cors": A biblioteca "cors" é usada para habilitar e configurar políticas de CORS no servidor Node.js. Ela permite especificar quais origens (domínios) têm permissão para acessar recursos em seu servidor, bem como quais métodos HTTP (GET, POST, PUT, DELETE, etc.) e cabeçalhos são permitidos nas solicitações CORS.
4. Configuração Flexível: O "cors" fornece opções flexíveis para configurar as políticas de CORS de acordo com as necessidades do seu aplicativo. Você pode configurar origens permitidas, cabeçalhos personalizados, métodos permitidos e até mesmo definir se deseja habilitar cookies e credenciais em solicitações CORS.
5. Middleware Express: O "cors" é frequentemente usado como um middleware em aplicativos Express.js. Você pode adicioná-lo facilmente ao seu aplicativo Express para habilitar o controle de CORS em todas as rotas ou de forma seletiva, dependendo das necessidades do seu aplicativo.

# mysql2

A biblioteca mysql2 é uma biblioteca para Node.js que oferece suporte para conectar e interagir com bancos de dados MySQL de forma assíncrona e promissificada. Ela é uma alternativa à biblioteca mysql mais antiga e é amplamente usada para acessar bancos de dados MySQL em aplicativos Node.js. A seguir são apresentadas algumas das principais finalidades e funcionalidades da biblioteca mysql2:

1. **Conexão ao Banco de Dados:** A biblioteca `mysql2` permite estabelecer conexões com bancos de dados MySQL a partir do Node.js. Isso é essencial para realizar operações de leitura e escrita no banco de dados a partir do seu aplicativo Node.js.
2. **Operações Assíncronas:** A biblioteca `mysql2` é projetada para funcionar de forma assíncrona, o que significa que você pode realizar consultas e operações de banco de dados sem bloquear a execução do seu programa. Isso é importante para garantir que o aplicativo seja responsivo, especialmente em cenários com muitas solicitações concorrentes.
3. **Promessas:** Uma característica importante do `mysql2` é que ele oferece suporte a promessas, permitindo que você use `async/await` para tornar o código mais legível e manutenível ao lidar com operações de banco de dados assíncronas.
4. **Manipulação de Resultados:** A biblioteca `mysql2` fornece métodos para recuperar e manipular os resultados das consultas SQL. Você pode obter dados em vários formatos, como objetos JavaScript, arrays e outros tipos de estrutura de dados.

5. Transações: O `mysql2` oferece suporte a transações, o que permite que você execute várias consultas SQL como uma única unidade atômica. Isso é importante para garantir que as operações de banco de dados sejam consistentes e seguras.
6. Consultas Preparadas: Você pode usar consultas preparadas com o `mysql2` para evitar injeção de SQL e melhorar a segurança das operações de banco de dados.
7. Pool de Conexões: O `mysql2` suporta a criação de pools de conexões, que são úteis para gerenciar e reutilizar conexões com eficiência, melhorando o desempenho do aplicativo.
8. Compatibilidade com MySQL: A biblioteca `mysql2` é compatível com o MySQL e oferece suporte às funcionalidades mais recentes do banco de dados, garantindo que você possa aproveitar os recursos mais avançados do MySQL em seu aplicativo Node.js.

# nodemon

O Nodemon é uma ferramenta que facilita o desenvolvimento de aplicativos Node.js, especialmente durante o processo de codificação e depuração. Ele é usado em combinação com o Node.js para monitorar alterações nos arquivos do seu projeto e automaticamente reiniciar o servidor Node.js sempre que você fizer alterações no código-fonte. Isso elimina a necessidade de você manualmente parar e reiniciar o servidor a cada vez que fizer uma modificação no código.

# Configurando NodeMon

Agora abra o arquivo “package.json” e defina uma nova chave para o nodemon, conforme na imagem da direita.

Com essa configuração, estamos preparando o NodeMon para ser executado somente no ambiente de desenvolvimento...

```
23     homepage : https://github.com
24     "dependencies": {
25         "cors": "^2.8.5",
26         "express": "^4.18.2",
27         "mysql2": "^3.6.0",
28         "nodemon": "^3.0.1"
29     }
30 }
31
```

```
23     homepage : https://github.com
24     "dependencies": {
25         "cors": "^2.8.5",
26         "express": "^4.18.2",
27         "mysql2": "^3.6.0"
28     },
29     "devDependencies" : {
30         "nodemon": "^3.0.1"
31     }
32 }
```

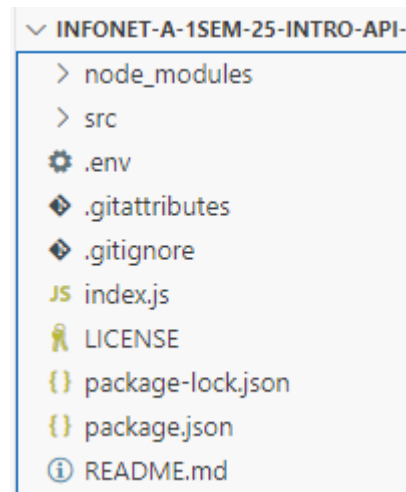
... agora adicionaremos dois novos scripts, na chave “scripts”, um para execução do projeto pelo node e outro que servirá para a execução do nosso projeto com o nodemon. Com o nodemon configurado para ser executado no modo desenvolvimento não temos que reiniciar nossa aplicação toda vez que salvarmos uma alteração em nosso código.

```
5   |   "main": "index.js",  
    |   ▶ Debug  
6   |   "scripts": {  
7   |     "start": "node index.js",  
8   |     "dev" : "nodemon index.js",  
9   |     "test": "echo \"Error: no test specified\" && exit 1"  
10  |   },  
11  |   "repository": "
```

# .env

Crie um arquivo na raiz do projeto com o nome “.env” e insira o código apontado na imagem.

Nas linha onde as setas estão apontando insira as informações referente ao seu acesso de banco de dados.



```
.env
1  #Banco
2  BD_USUARIO=us_aula_node
3  BD_SENHA=123456
4  BD_SERVIDOR=10.67.22.216
5  # BD_SERVIDOR=192.168.15.8
6  BD_PORTA=3306
7  BD_BANCO=bd_aula_node
8
9  #Email
10 MAIL_HOST=host e-mail
11 MAIL_PORT=porta e-mail
12 MAIL_USER=e-mail usuario
13 MAIL_PASS=senha usuario
14
15 #Dominio
16 DOMINIO=http://dominio:porta
17
18 #Porta
19 PORT=3333
```

Um arquivo .env em um projeto Node.js com Express e MySQL serve como um repositório para variáveis de ambiente, ou seja, informações de configuração que podem variar dependendo do ambiente em que sua aplicação está sendo executada (desenvolvimento, teste, produção).



Para que serve:

**Segurança:** Permite armazenar informações sensíveis, como senhas de banco de dados, chaves de API e outros segredos, fora do código-fonte, evitando que sejam expostas em repositórios públicos.

**Configuração flexível:** Facilita a alteração de configurações sem precisar modificar o código da aplicação. Basta alterar os valores no arquivo `.env` e reiniciar o servidor.

**Organização:** Centraliza as configurações em um único lugar, tornando o código mais limpo e organizado.

**Gerenciamento de ambientes:** Permite definir configurações diferentes para cada ambiente, garantindo que a aplicação se comporte de forma adequada em cada contexto.

Aplicações em uma API Node.js com Express e MySQL:

Credenciais do banco de dados: Armazenar o nome do host, nome de usuário, senha e nome do banco de dados MySQL.

Chaves de API: Guardar chaves de acesso para serviços externos, como APIs de pagamento, envio de e-mail ou redes sociais.

Portas e URLs: Definir a porta em que o servidor Express irá rodar e a URL da API.

Variáveis de ambiente específicas: Configurar variáveis que controlam o comportamento da aplicação em diferentes ambientes, como níveis de log, flags de recursos ou configurações de cache.

# .env

**Importante:** O arquivo .env faz parte do seu .gitignore! Este arquivo contém configurações específicas do ambiente e não deve ser enviado para o repositório Git.

*Porém em nosso exemplo usado em aula iremos permitir que este arquivo faça parte do repositório para facilitar a configuração e acesso do banco de dados por todos os integrantes do grupo. Para isso comente o trecho do código referente ao arquivo “.env” no “.gitignore”*

```

❖ .gitignore
75  # dotenv environment variable files
76  #.env ←
77  .env.development.local
78  .env.test.local
79  .env.production.local
80  .env.local
```

# Preparando o arquivo principal do projeto

Tudo que precisamos fazer no nosso arquivo index.js é adicionar algumas linhas, inicialmente importamos o express, que será responsável pelo nosso controle de rotas além de que determinamos em qual porta a nossa aplicação será executada, nesse caso na porta 3333.

```
JS index.js > ...
1  require('dotenv').config();
2  const express = require('express');
3  const cors = require('cors');
4
5  const app = express();
6  app.use(cors());
7  app.use(express.json());
8
9  const porta = process.env.PORT || 3333;
10
11 app.listen(porta, () => {
12   |   console.log(`Servidor iniciado em http://localhost:${porta}`);
13 });
```

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

Esta linha carrega as variáveis de ambiente do arquivo .env para o process.env do Node.js.

O pacote dotenv lê o arquivo .env na raiz do projeto e disponibiliza as variáveis definidas nele como propriedades do objeto process.env.

Isso permite que você acesse as variáveis de ambiente em seu código usando process.env.NOME\_DA\_VARIAVEL.

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

Esta linha importa o módulo express, que é um framework web para Node.js. express facilita a criação de APIs e aplicativos web.

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

Esta linha importa o módulo cors, que é um middleware do express.

CORS (Cross-Origin Resource Sharing) é um mecanismo de segurança do navegador que restringe solicitações HTTP de um domínio para outro. Esse middleware permite que sua API responda a solicitações de outros domínios.

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

Esta linha cria uma instância do aplicativo Express.

app é o objeto que você usará para definir rotas, middleware e outras configurações do seu servidor.

```
app.use(cors());
```

Esta linha usa o middleware cors no seu aplicativo Express.

Isso permite que o servidor responda a solicitações de diferentes origens, evitando erros de CORS.

```
app.use(express.json());
```

Esta linha usa o middleware express.json().

Este middleware analisa as solicitações HTTP com corpo JSON e disponibiliza os dados analisados no objeto req.body.

É essencial para APIs que recebem dados JSON.

```
const porta = process.env.PORT || 3333;
```

Esta linha define a porta em que o servidor irá rodar.

`process.env.PORT` tenta obter o valor da variável de ambiente `PORT`.

Se `process.env.PORT` não estiver definido, a porta padrão será 3333.

Isso permite que a porta seja configurada por meio de variáveis de ambiente, o que é útil para implantação em diferentes ambientes.

```
app.listen(porta, () => { console.log(`Servidor iniciado em  
http://localhost:${porta}`); });`
```

Esta linha inicia o servidor Express na porta definida.

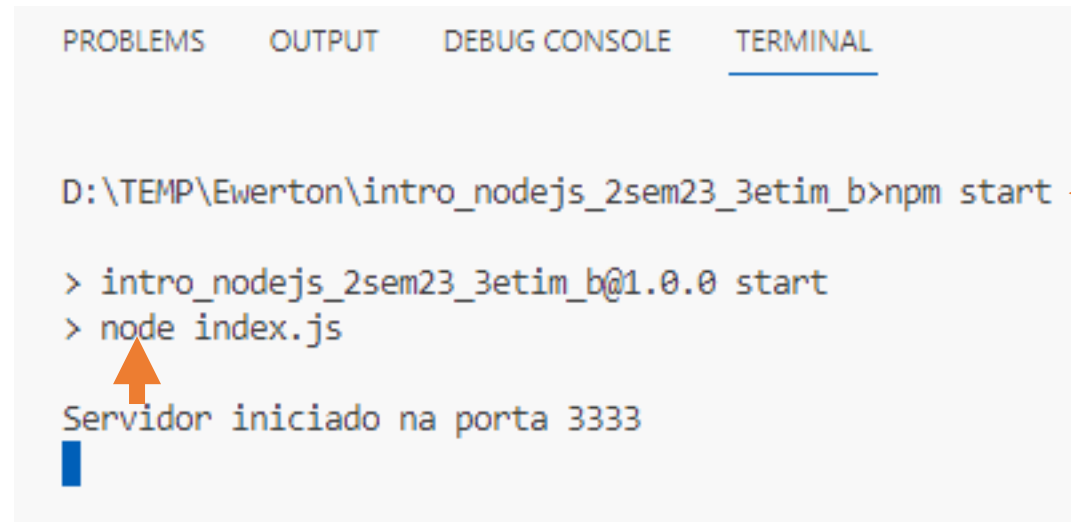
O método `app.listen()` recebe a porta e uma função de callback.

A função de callback é executada quando o servidor é iniciado com sucesso.

Neste caso, a função de callback exibe uma mensagem no console informando que o servidor está rodando e a URL em que ele está acessível.

# Iniciando nosso servidor

- Agora que temos tudo configurado no nosso arquivo index podemos executar a nossa aplicação, para isso basta executar o comando abaixo no seu terminal dentro da pasta do projeto.
- npm start
- npm run dev



The screenshot shows a VS Code terminal window with the 'TERMINAL' tab selected. The terminal displays the following text:

```
D:\TEMP\Ewerton\intro_nodejs_2sem23_3etim_b>npm start
```

An orange arrow points to the right of the command line. Below this, the terminal shows the output of the command:

```
> intro_nodejs_2sem23_3etim_b@1.0.0 start
> node index.js
```

Another orange arrow points to the output line. The final line of the terminal output is:

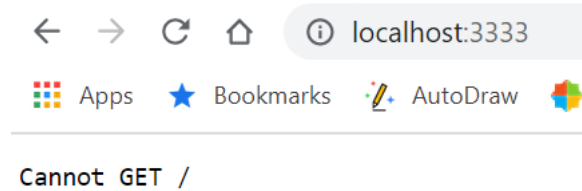
```
Servidor iniciado na porta 3333
```

A blue cursor is visible at the end of the last line.



- No navegador insira o endereço a seguir:

`http://localhost:3333/`



- Nada acontece e você verá a seguinte mensagem.
- Isso aconteceu por que “startamos” nossa aplicação e não configuramos nenhuma rota, mas o que é uma rota?

Obs: Utilize “Ctrl + c” para encerrar o processo referente a nossa aplicação. Esse processo é necessário sempre que formos atualizar o código tendo iniciado o servidor com o comando “npm start”, de agora em diante iremos utilizar “npm run dev” para otimizar a atualização do código sem a necessidade de reiniciar o serviço manualmente.

# Entendendo o que é um rota

Uma rota assim como o nome sugere, é um caminho(url) pelo qual quando chamada poderá gerar alguma ação.

Exemplos de rotas

- /api/dados → Quando invocada poderá retornar os dados da nossa aplicação
- /api/deleteltem/1 → Quando invocada poderá remover um determinado item, nesse caso o item 1

Mas as rotas não trabalham sozinhas, elas trabalham com outros componentes que veremos mais adiante, os chamados controllers.

Inicie o servidor com o comando “npm run dev”, assim sempre que salvamos o código no editor o servidor é reiniciado automaticamente.

```
D:\TEMP\Ewerton\intro_nodejs_2sem23_3etim_b>npm run dev
```

```
> intro_nodejs_2sem23_3etim_b@1.0.0 dev
```

```
> nodemon index.js
```

```
[nodemon] 3.0.1
```

```
[nodemon] to restart at any time, enter `rs`
```

```
[nodemon] watching path(s): *.*
```

```
[nodemon] watching extensions: js,mjs,cjs,json
```

```
[nodemon] starting `node index.js`
```

```
Servidor iniciado na porta 3333
```

# Criando a primeira rota

A sintaxe de uma rota é parecido com isto que estamos vendo abaixo, inicialmente definimos a nossa rota “/”, que é a rota inicial da nossa aplicação, é ela que será chamada quando você digitar apenas “localhost:3333” no seu navegador, além disso nossa rota possui uma função que recebe dois parâmetros, o request e o response .

```
app.get('/', (request, response) => {  
    response.send("Hello world")  
})
```

## REQUEST

Podemos manipular alguns dados de uma requisição, podemos obter informações do tipo de navegador que fez a requisição, se foi de um dispositivo móvel ou desktop e outras coisas.

## RESPONSE

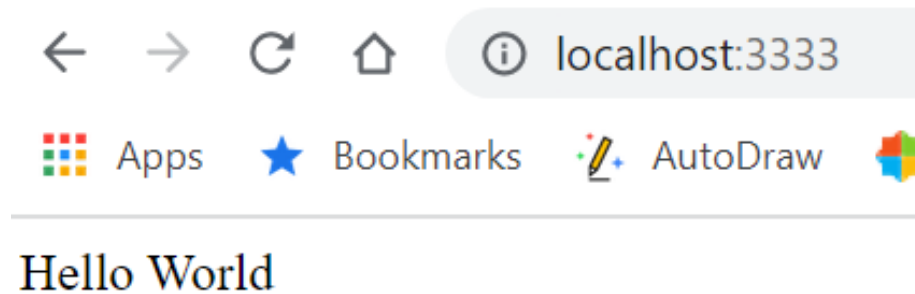
O response é responsável por retornar algo para a página que fez a requisição para a rota, alguns dos métodos principais métodos do response podem ser vistos abaixo.

- `response.json({message: "retorno de objeto json"})` → Retorna um objeto json para a página
- `response.send(" ")` → Envia algo para a página como um texto

O arquivo index.js ficará parecido com o exemplo abaixo após as modificações.

```
JS index.js > ...
1  require('dotenv').config();
2  const express = require('express');
3  const cors = require('cors');
4
5  const app = express();
6  app.use(cors());
7  app.use(express.json());
8
9  const porta = process.env.PORT || 3333;
10
11  app.listen(porta, () => {
12    | console.log(`Servidor iniciado em http://localhost:${porta}`);
13  });
14
15  app.get('/', (request, response) => {
16    | response.send('Hello World');
17  });
```

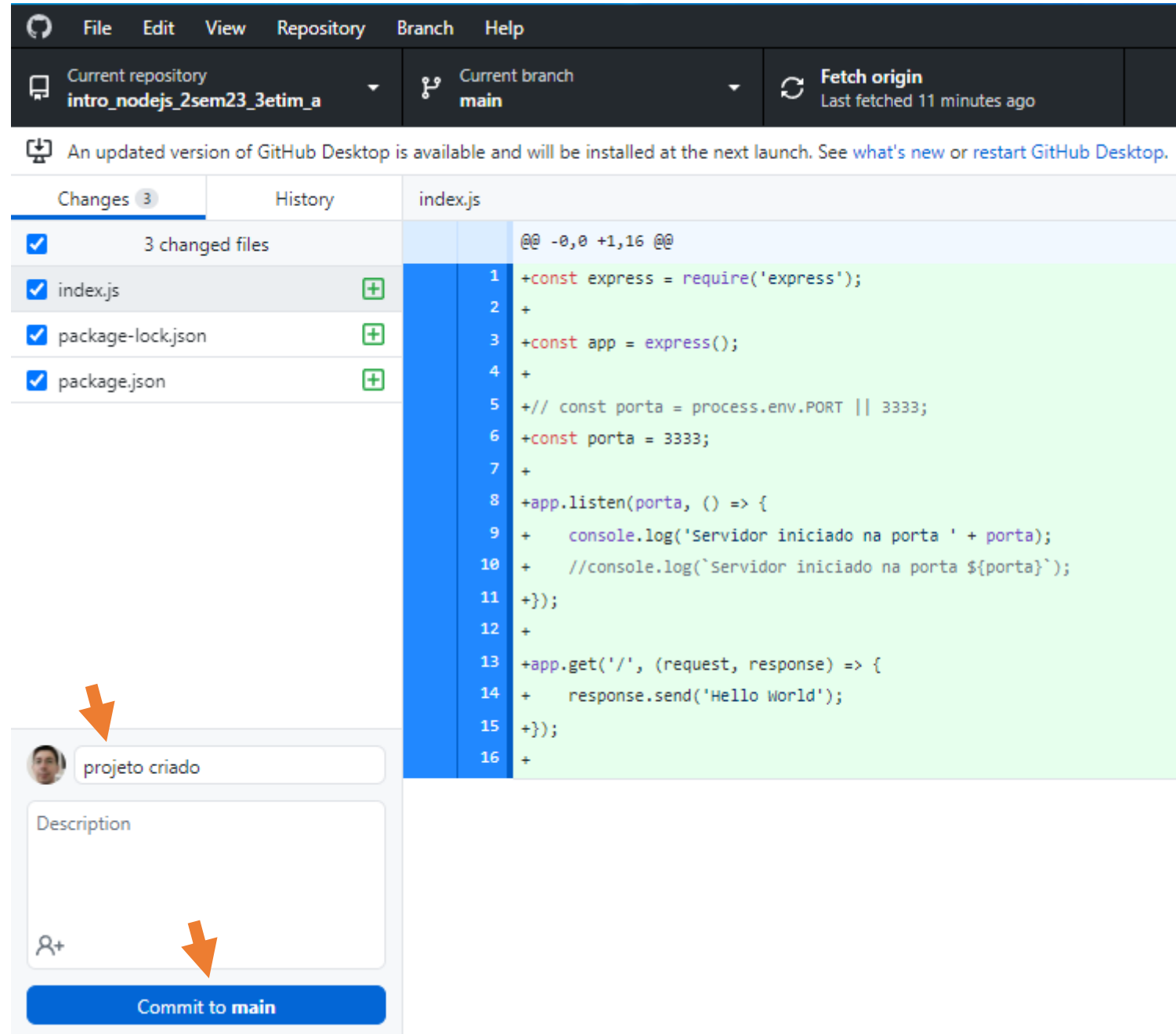
Ao recarregar sua página no endereço “localhost:3333”, poderá observar que a mensagem “Hello world” é apresentada na tela



# Commit

Para guardar as alterações realizadas em nosso repositório é necessário acessar o GitHub Desktop e realizar um Commit.

De uma nome e uma descrição para auxiliar a entender o que foi realizado.





Commit é um termo que vem do controle de versão de software e é uma parte fundamental do processo de colaboração em projetos de desenvolvimento de software. No contexto do GitHub Desktop, o commit é uma ação que você realiza para salvar as alterações que fez em seu código-fonte ou em outros arquivos em seu repositório local. Seu funcionamento passa por algumas ações como:

1. Realização de Alterações: Antes de fazer um commit, você geralmente faz alterações em seus arquivos, como adicionar ou modificar código, atualizar a documentação ou fazer qualquer outra alteração necessária no projeto.
2. Staging (Preparação): Após fazer as alterações, você precisa "preparar" as alterações que deseja incluir no commit. Isso é feito no GitHub Desktop selecionando os arquivos específicos que você deseja incluir no commit. Esse processo é chamado de "staging" e permite que você escolha quais alterações serão registradas no commit.

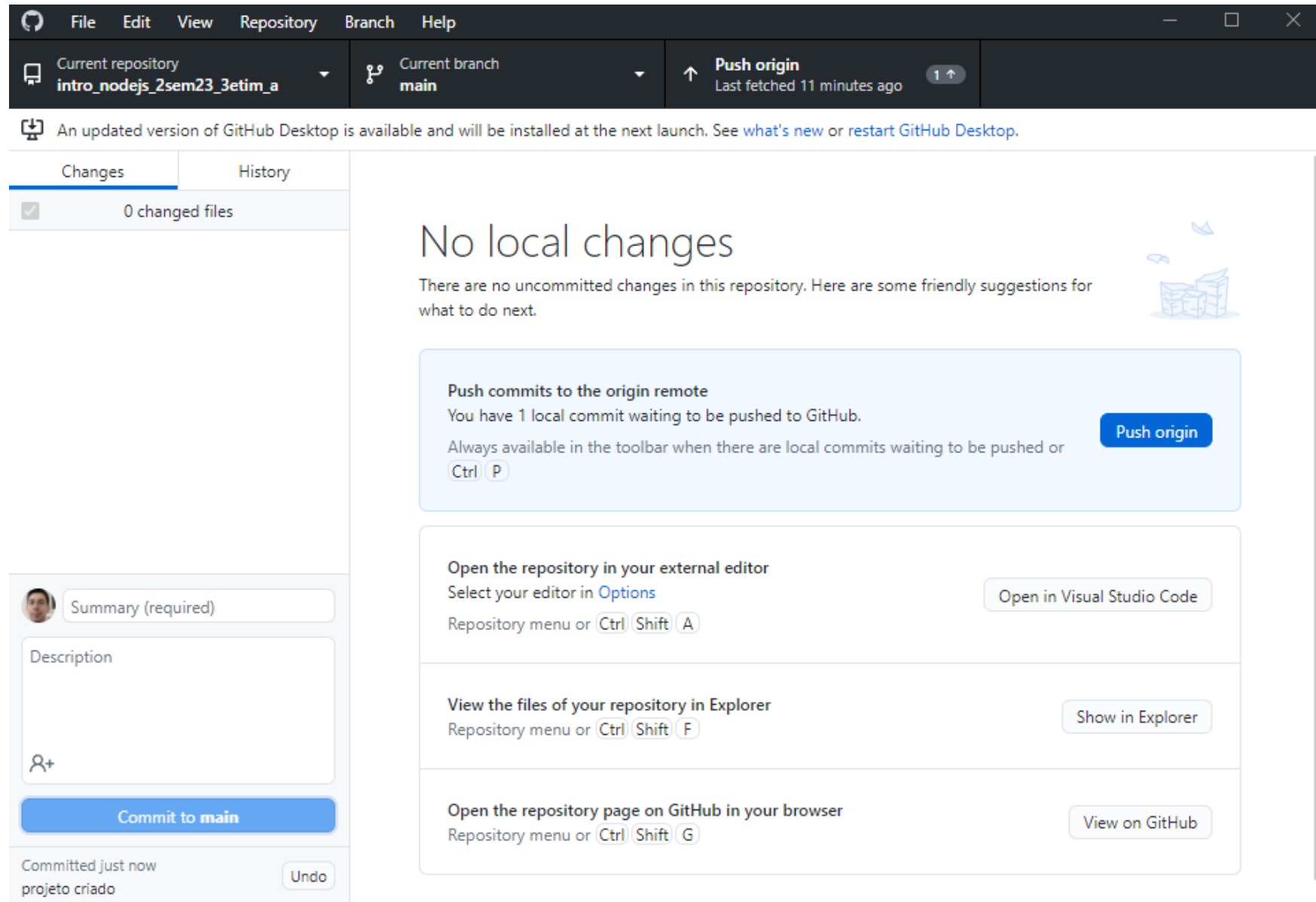
3. Commit: Depois de preparar suas alterações (staging), você pode criar um commit. Um commit é como um instantâneo das alterações que você fez naquele momento. Ele inclui uma mensagem de commit que descreve o que foi feito nas alterações. A mensagem do commit deve ser informativa e concisa para que outras pessoas possam entender as alterações sem precisar examinar o código em detalhes.
4. Envio para o Repositório: Após criar um commit, ele é salvo no seu repositório local. Você pode pensar nele como uma etapa intermediária entre as alterações feitas no seu código e a sua integração com o repositório remoto no GitHub.com.
5. Sincronização: Para enviar suas alterações para o repositório remoto no GitHub.com, você precisará realizar uma sincronização (push). Isso atualiza o repositório remoto com os commits que você fez localmente.

# Resumindo

Os commits são uma maneira fundamental de acompanhar o histórico de alterações em um projeto de software, permitindo que você e outros colaboradores vejam o que foi feito e quando. Eles também facilitam a resolução de problemas e a colaboração em equipe, pois você pode comentar sobre commits específicos e entender as alterações que foram feitas ao longo do tempo.

# push

Após o commit  
realize o Push  
para gravar o  
commit na sua  
conta do gitHub



O "push" é uma operação que você realiza no GitHub Desktop (ou em qualquer cliente Git) para enviar as alterações que você fez em seu repositório local para o repositório remoto hospedado no GitHub.com. Isso é fundamental para manter seu repositório remoto atualizado com as últimas alterações feitas localmente. Seu funcionamento passa por algumas ações como:

1. Realização de Alterações Locais: Antes de fazer um push, você geralmente faz alterações em seus arquivos, adiciona novos commits (conforme explicado na resposta anterior) e faz outras edições no código ou em outros arquivos do projeto.

2. **Commit Localmente:** Cada commit que você cria registra um conjunto específico de alterações em seu repositório local. Os commits são salvos apenas em sua máquina até que você decida enviá-los para o repositório remoto.
3. **Push para o Repositório Remoto:** Quando você se sente pronto para compartilhar suas alterações com outros colaboradores ou simplesmente deseja manter seu repositório remoto atualizado, você realiza um push. Isso envia os commits que você fez localmente para o repositório remoto correspondente no GitHub.com.
4. **Atualização do Repositório Remoto:** Após um push bem-sucedido, o repositório remoto no GitHub.com será atualizado com os commits que você enviou. Outros colaboradores que têm acesso a esse repositório remoto poderão ver e colaborar com as alterações que você fez.

# Resumindo

O push é uma parte essencial do fluxo de trabalho de colaboração com o Git e o GitHub. Ele permite que várias pessoas trabalhem juntas em um projeto, enviando suas alterações para um repositório compartilhado. Isso também é usado para implantar código em servidores de produção, atualizando-os com as últimas alterações do repositório.

Lembre-se de que, ao realizar um push, você precisará estar conectado à sua conta do GitHub (ou outra plataforma de hospedagem de código) e ter as permissões adequadas para enviar alterações para o repositório remoto.