**RECURSOS**

-Node.js

https://www.alura.com.br/artigos/como-instalar-node-js-windows-linux-macos

-Comando para habilitar execução de scripts Node.js no firewall do windows

Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned

-Validando instalação Node.js no CMD

npm --version

-Instalação GIT

https://www.youtube.com/watch?v=Am46OOLgV4s

-Instalação Postman

https://www.alura.com.br/artigos/postman-como-instalar-dar-seus-primeiros-passos

-Instalação FFMpeg

<https://www.youtube.com/watch?v=WDCJzPfWx6o>

-Wireshark para analise de pacotes http

-Criptografia com biblioteca - OpenSSL - Certificados digitais  
https://slproweb.com/products/Win32OpenSSL.html

**SERVIDORES FRONTEND BACKEND**

**-Download e configuração backend**

git clone https://github.com/alura-cursos/api-alurabooks.git

-Local recursos de backend

cd api-alurabooks

-instalando as dependências listadas no arquivo package.json

npm install

-executando o backend e o disponibilizando através de um servidor no endereço http://localhost:8000

npm run start-auth

**-Download e configuração do frontend**

git clone https://github.com/alura-cursos/curso-react-alurabooks.git

-Local recursos de frontend

cd curso-react-alurabooks

-selecionando a versão correta

git checkout aula-5

-instalando as dependências

npm install

-compilando o frontend e o disponibilizando através de um servidor no endereço http://localhost:3000

npm start

**OBSERVAÇÕES**

Por que para executar o servidor backend e frontend são comandos distintos?

Backend npm run start-auth

Frontend npm start

A diferença entre os dois comandos (npm run start-auth no backend e npm start no frontend) é relacionado a como cada projeto está configurado no arquivo package.json de cada um deles.

-Servidor backend utilizado

https://www.npmjs.com/package/json-server

**URL, ENDEREÇOS DA WEB, PORTAS E SERVIDORES**

URL (Uniform Resource Locator) Uma URL é o enderecamento que permite localizar recursos na web, como páginas, scripts e serviços.   
  
Exemplo de URL:

http://localhost:3000/

Componentes da URL:

Protocolo (HTTP/HTTPS/FTP/etc.): Define o meio de comunicacao.

Servidor + Porta (ex: localhost:3000): Indica onde o recurso está e qual porta o acessa.

Caminho (/): Representa o recurso dentro do servidor (no caso, a página inicial).

Portas e Servidores

A porta é um canal de acesso dentro de um servidor.

O que permite um servidor hospedar um ou mais serviços.

E esses serviços são distinguidos devido a porta de acesso:

Front-end: localhost:3000

Back-end: localhost:8000

Portas comuns HTTPS:

80: HTTP (padrão, geralmente não aparece na URL)

443: HTTPS (seguro, padrão para sites com criptografia)

Portas de uso livre (para desenvolvimento): a partir de 1023 (ex: 3000, 8000).

Por que não vemos a porta no Google.com?

Porque estamos usando a porta padrão do HTTPS (443), que é automaticamente assumida pelo navegador.

Nome de Domínio vs Endereço IP

Servidores são acessados por IPs, mas usamos nomes (ex: google.com) por serem mais fáceis de lembrar. O mapeamento entre nome e IP é feito pelo DNS (Domain Name System), assunto que será explorado posteriormente.

DNS - [DNS: o que é, como funciona e qual escolher | Alura](https://www.alura.com.br/artigos/dns-o-que-e-qual-escolher)

**testes**

-Teste de conectividade com um determinado site.

Ping

-Visualizar a rota que os pacotes de dados seguem até alcançar o destino desejado.

traceroute (ou tracert no Windows)

-Obter informações do seu registro DNS

nslookup www.site.com.br

-Verificando front e back no ar

netstat -ano | findstr :3000

netstat -ano | findstr :8000

**TCPIP - HTTP - METODOS**

TCP, IP e HTTP são protocolos essenciais para a comunicação na internet.

TCP (Transmission Control Protocol) garante a entrega confiável dos dados

IP (Internet Protocol) é responsável por endereçar e encaminhar os pacotes de dados.

HTTP (Hypertext Transfer Protocol) é o protocolo utilizado para transferência de informações entre clientes e servidores web, utilizando métodos como GET, POST, PUT, etc.

Elaboração:

TCP/IP: É um conjunto de protocolos que define como a informação é enviada e recebida na internet.

O TCP garante que os dados sejam entregues na ordem correta e sem erros, enquanto o IP cuida de como os dados são endereçados para chegar ao destino correto.

HTTP: É o protocolo que permite a transferência de dados na web, como páginas, imagens e outros recursos.

Ele usa o TCP/IP para realizar essa comunicação.

Métodos HTTP: São as instruções que indicam o tipo de ação que o cliente quer realizar com o servidor (CRUD).

Os principais métodos incluem:

-GET: ler Solicita dados do servidor.

-POST: criar Envia dados para o servidor, geralmente para criar ou modificar um recurso.

-PUT: update Substitui o recurso no servidor por uma nova representação.

-DELETE: apagar Remove um recurso do servidor.

-PATCH: Modifica parcialmente um recurso no servidor.

-HEAD: Solicita apenas o cabeçalho da resposta, sem o corpo.

-OPTIONS: Solicita as opções de comunicação disponíveis para um recurso.

-TRACE: Executa um teste de loopback e mostra o caminho da requisição.

-CONNECT: Estabelece um túnel com o servidor.

Em resumo, TCP e IP são os protocolos de transporte e rede que permitem que os dados sejam enviados e recebidos na internet, enquanto HTTP é o protocolo que permite a comunicação entre clientes e servidores web, utilizando métodos para indicar a ação a ser realizada.

Métodos de requisição HTTP

https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Reference/Methods

**VERIFICANDO REQUISIÇÕES NO PROTOCOLO HTTP**

- Verificando requisições no protocolo http

Com servidor backend e frontend em execução,

backend

localhost:8000

frontend

localhost:3000

Ao inspecionar ou teclando f12 - será habilitado ferramentas de desenvolvedor

Ao cadastrar um novo usuário, é enviado uma nova requisição ao servidor

Ou ao realizar login usuário, também é enviado uma requisição ao servidor

no monitoramento, na aba rede (networking)

Pode-se capturar o pacote enviado e verificar o formato da requisição

cabeçalho

conteúdo

visualização

resposta

Respostas:

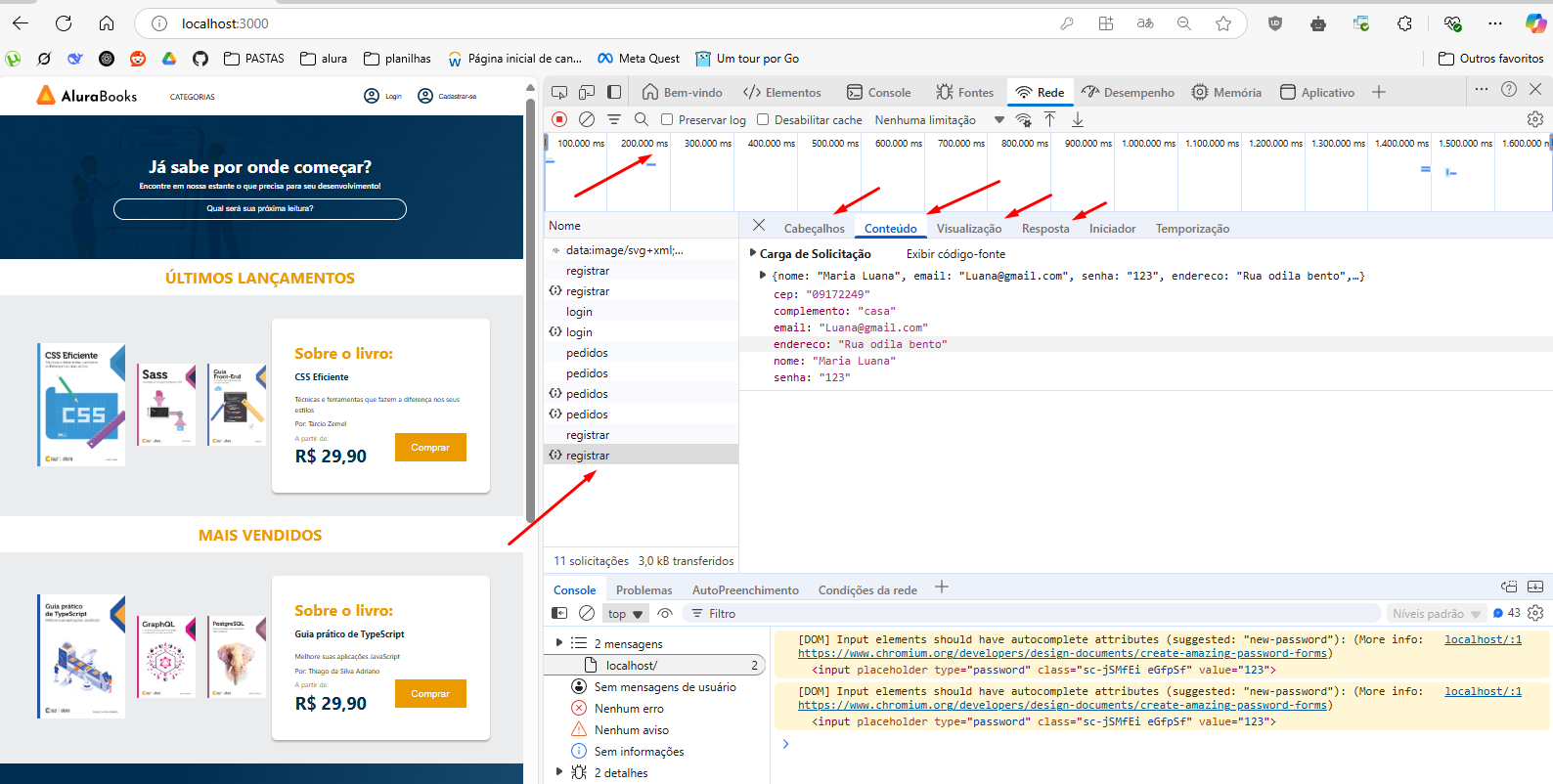
100 – 199 Respostas Informativas

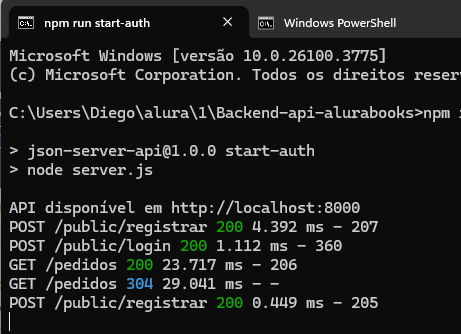
200 – 299 Respostas bem-sucedidas

300 – 399 Mensagens de redirecionamento | **304** **Not Modified** acesso implícito para o recurso em cache.

400 – 499 Respostas de erro do cliente

500 – 599 Respostas de erro do servidor





**REQUISIÇÕES HTTP NO FRONTEND E BACKEND**

- Temos 1 host - Servindo dois servidores - backend e frontend

- Os serviços são separados por portas, cada uma, realiza operações diferentes

Backend - <http://localhost:8000>

Frontend - <http://localhost:3000>

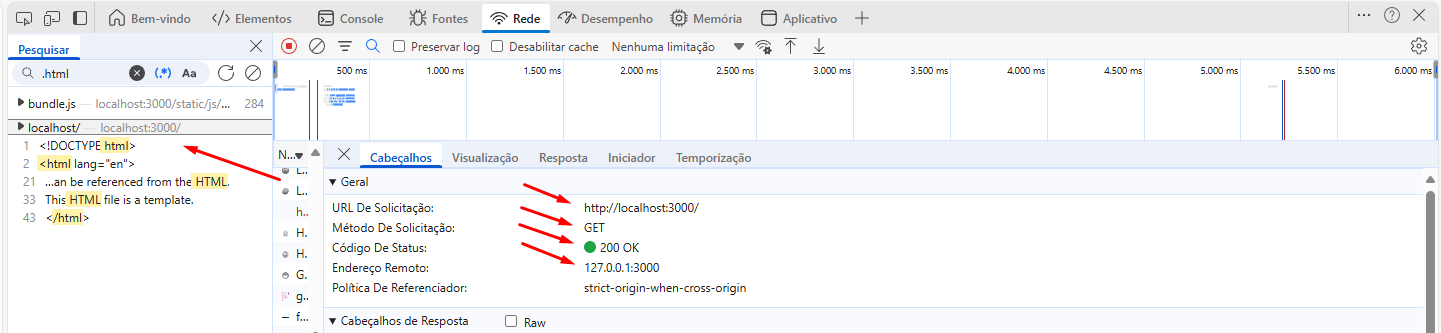
Quando acessamos um site pela primeira vez, é enviado um **GET ao front-end** - requisitando, o **index.html ou ‘/**’ do site

Para esse servidor, nós enviaríamos uma mensagem com o seguinte conteúdo: "GET /HTTP/1.1".

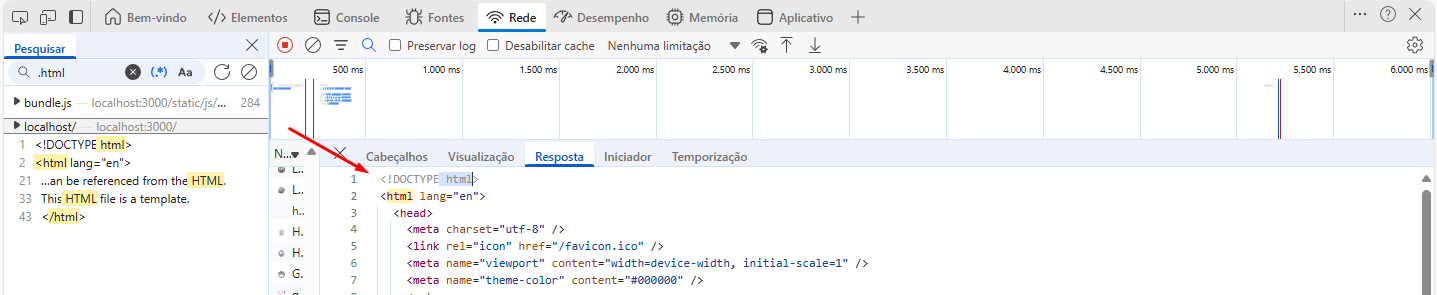
O "HTTP" é o protocolo que estamos usando e "1.1" é a sua respectiva versão.

Esse método GET está sendo usado para obtermos informações Index.html e para renderização do site.

**GET - Requisito ao servidor**



**RESPOSTA - Do servidor ao cliente**



Agora, imagine que criamos um usuário e gostaríamos de fazer o login desse usuário no Projeto AllBooks, na plataforma.

Enviaríamos uma mensagem para o servidor do back-end com o seguinte conteúdo:

POST /public/login HTTP/1.1

Content-Type: application/json

Content-length: 42

{"email": "nome@email.com", "senha": "123"}

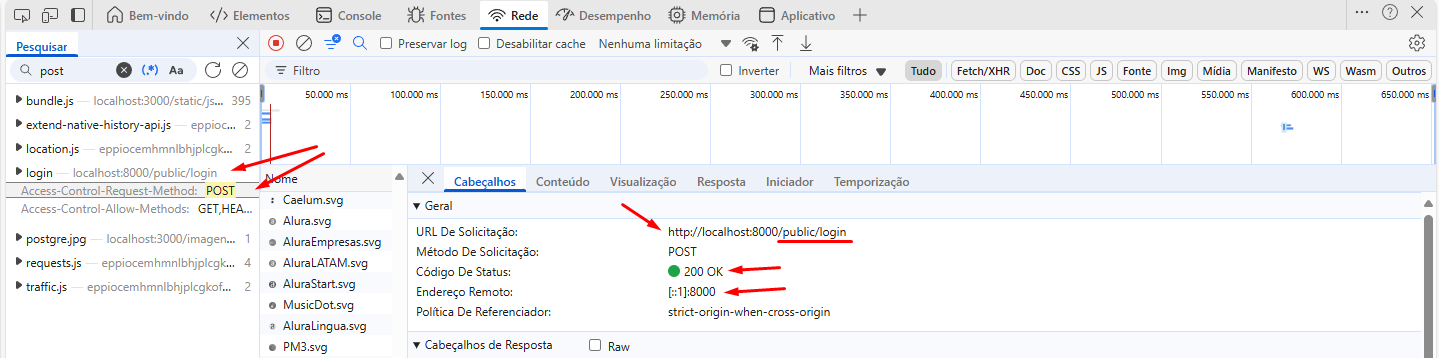
Primeiro, teríamos o metódo **"POST"**, e depois a indicação do recurso que gostaríamos de acessar, o **"/public/login"**, seguido da versão do protocolo HTTP com "1.1". Temos um tipo de conteúdo e o tamanho desse conteúdo.

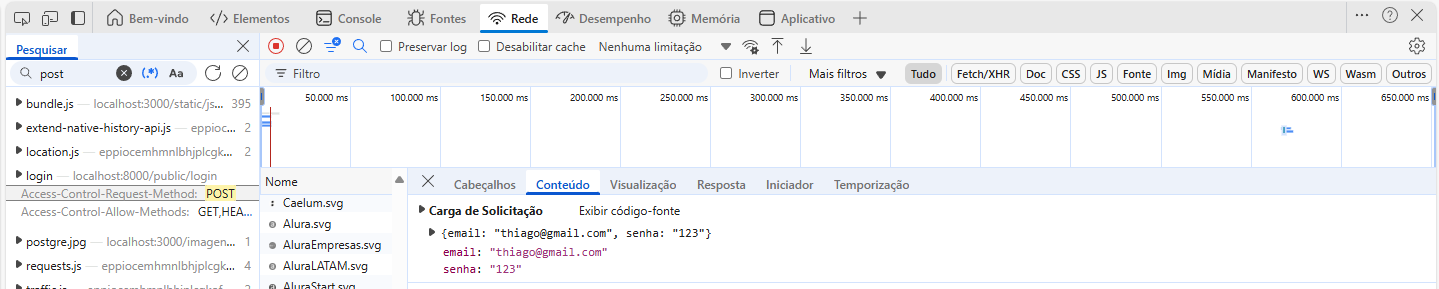
Dentro desse conteúdo, **estamos mandando uma mensagem no formato de um JSON (JavaScript Object Notation),** por isso **"Application/JSON", com um tamanho de 42,** e contendo os dados necessários para que o usuário consiga logar nesse site.

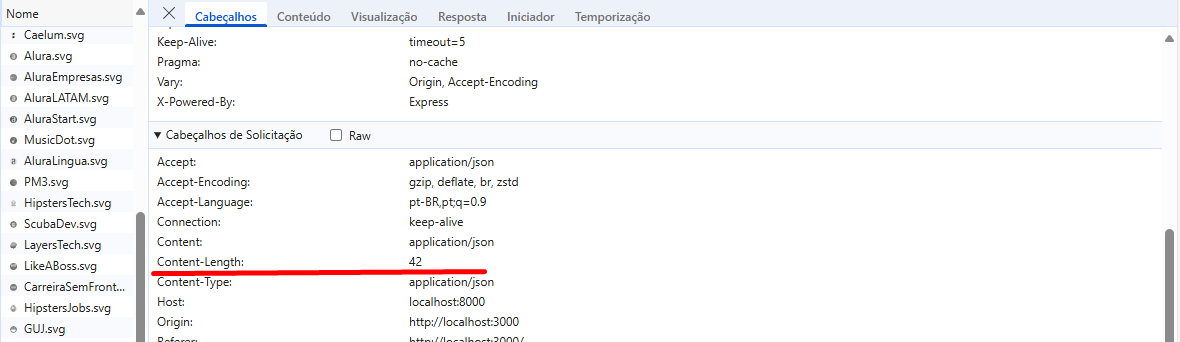
**O e-mail do usuário e a senha cadastrada.**

**Por sua vez, o servidor também responderia.**

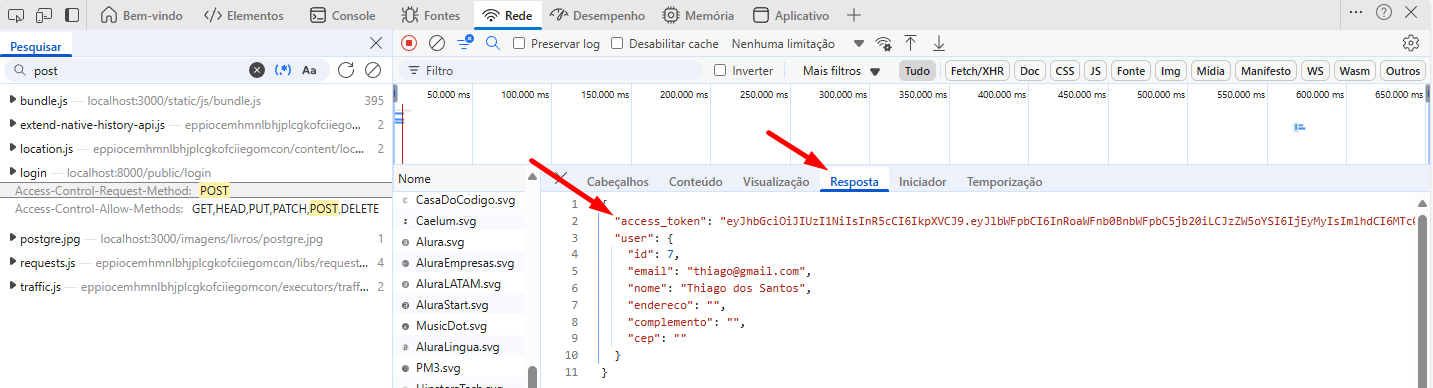
**E nessa mensagem de resposta, ele teria um "HTTP/1.1 200 OK", um código de status 200 indica que a requisição foi recebida e está sendo respondida com êxito, e uma mensagem novamente contendo um cabeçalho, um "Content-type" e um tamanho de conteúdo.**







Dentro desse tamanho de conteúdo, que também seria um JSON, receberíamos um token de acesso, dizendo que o usuário está cadastrado, a senha é válida, se refere a esse usuário, e ele pode sim acessar a página dele dentro do site do Allbooks.



E o que percebemos na estrutura dessas mensagens? Percebemos na primeira linha, temos um POST indicando qual a nossa intenção da mensagem, depois temos um metadado da mensagem, e na sequência temos a mensagem.

Dessa forma, estruturamos uma mensagem HTTP em duas partes, a primeira parte se refere ao cabeçalho, ou os headers, e a segunda parte é o seu conteúdo, ou melhor dizendo, o seu corpo, ou body.

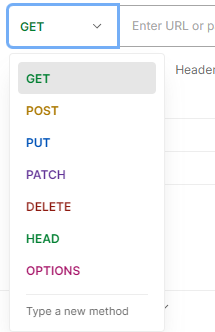
No corpo enviamos o que queremos mandar para o servidor, e no cabeçalho temos algumas informações sobre essa mensagem que estamos enviando.

**POSTMAN**

- Explorando o funcionamento desses métodos HTTP usando o Postman.   
- O Postman permite fazer testes de servidores web e também de APIs.

-Os metódos HTTP são às intenções de iteração com às paginas WEB

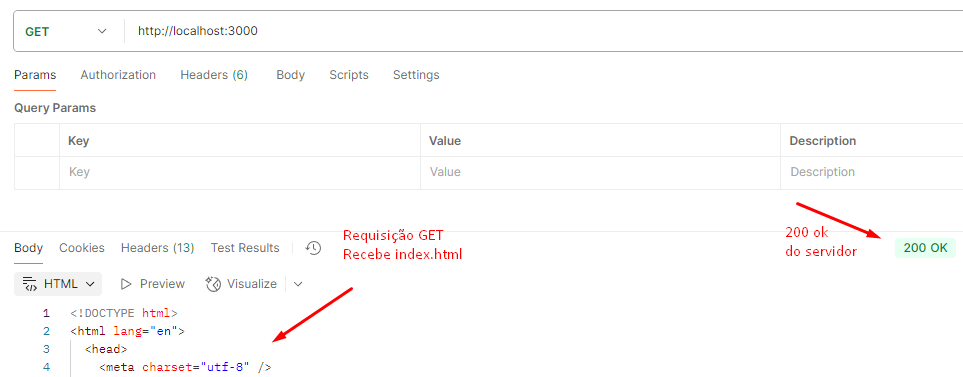
**MÉTODO GET NO HTTP**

Temos o “GET” "POST", "PUT" e uma série de outros métodos do HTTP.  
   
  
Podemos também entrar com uma request, isto é, com uma URL para a qual vamos mandar uma requisição, e o botão "Send" serve para enviarmos essa requisição.

Começaremos com o "GET".   
  
Mandaremos uma mensagem ao "http://localhost:3000", que é o endereço do nosso front-end.



No campo inferior, recebemos a mensagem do servidor.

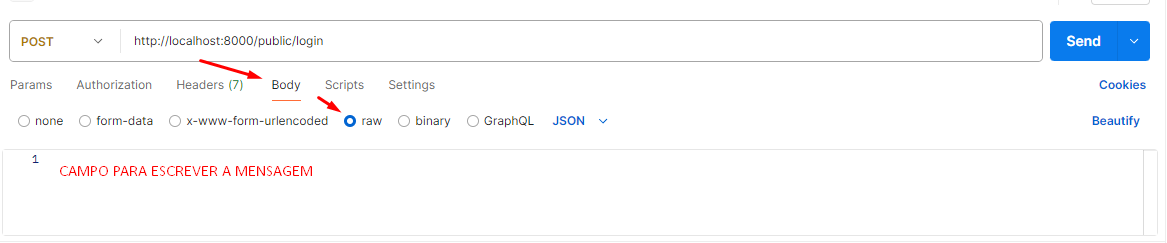
  
  
Nessa mensagem temos exatamente o código HTML que podemos usar para renderizar essa página no nosso navegador.

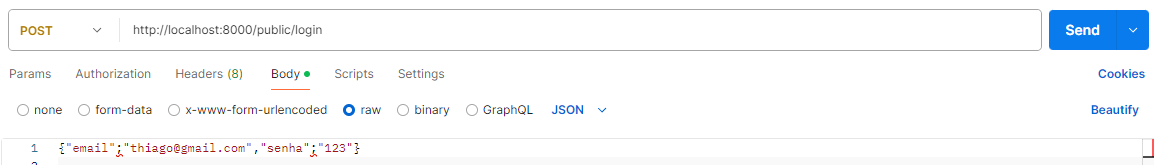
**MÉTODO POST NO HTTP**

Mudaremos nossa request, e vamos mandar uma requisição POST ao Backend direto ao recurso de login.

É necessário conhecer a estrutura do site, para enviar a requisição post ao local correto no backend.  
  
<http://localhost:8000/public/login>

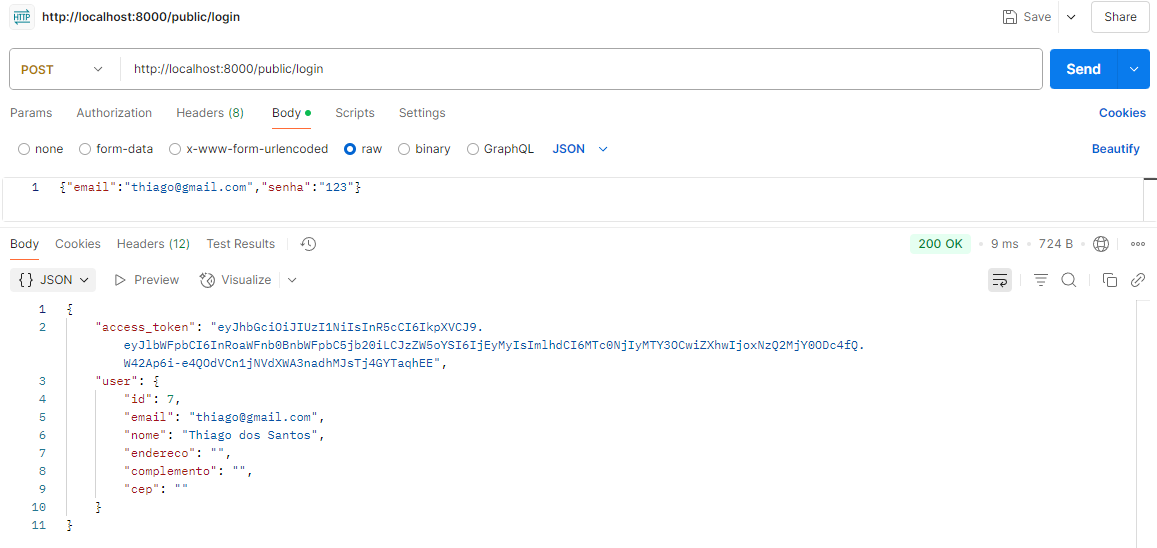


Vamos mudar o corpo da mensagem, alterando none "none" por "raw", que será um JSON.   
A alteração de none para raw - é apenas indicativo de que não será enviado uma mensagem vazia(none) e raw indica que iremos escrever uma mensagem que será enviada no post.  
  
{"email":"lcs@alura.com","senha":"123"}



Após o digitar a mensagem do JSON de um Send e veja o retorno abaixo com 200 ok



Feito isso, o campo body exibe o retorno do backend; Com o token de acesso do usuário  


**MÉTODO POST NO HTTP - Continuação**

- HTTP (Hypertext Transfer Protocol) - É o protocolo que faz comunicação entre Cliente(navegador) x Servidores web.

Toda vez que um site é acessado, o navegador faz requisições HTTP para o servidor web pedindo páginas, imagens, dados e outros recursos.

- Stateless (Sem Estado) - O HTTP é um protocolo stateless, ou seja:

Cada requisição é independente e o servidor não guarda memória da requisição anterior.

Exemplo: você faz login em um site, o servidor responde "login feito", na próxima requisição ele não sabe mais quem você é, a menos que algo mantenha essa informação (via token ou cookie).

-Requisição - É o pedido que o navegador faz ao servidor.   
Uma requisição HTTP possui várias partes:

1 Método (GET, POST, PUT, DELETE, etc.)

2 Cabeçalhos (headers) com informações adicionais, como autenticação

3 Corpo (body) contendo dados (usado em POST, por exemplo)

-Token - Um token é uma "chave digital" usada para identificar e autenticar o usuário.

É comum em APIs e aplicações modernas.

Exemplo: você faz login e recebe um token (como um JWT).

Esse token é enviado em cada requisição no cabeçalho Authorization.

O servidor valida o token e permite o acesso.

-Cookies - São pequenos arquivos salvos pelo navegador e enviados automaticamente em cada requisição ao servidor.

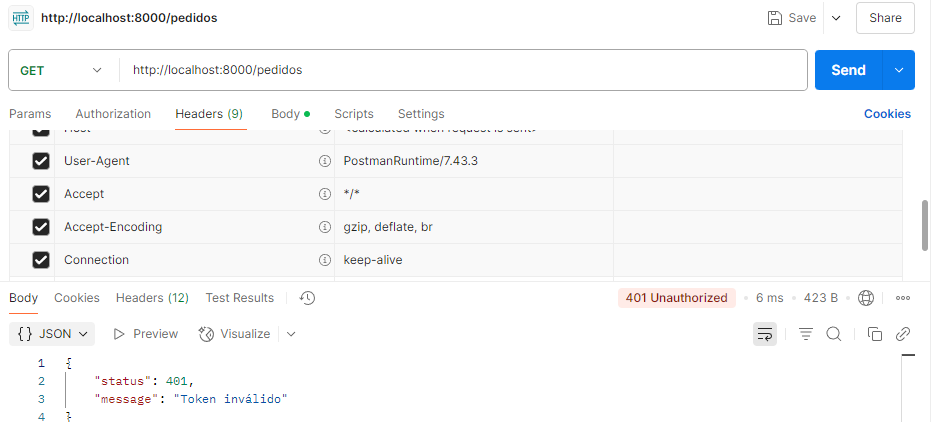
Servem para manter o estado em aplicações web, como:

Autenticação de sessão (manter usuário logado) e Armazenamento de preferências (idioma, tema, etc.)

Por que isso é importante?   
Como visto anteriormente, foi enviado um POST contendo um JSON que carregava os dados de login e senha do usuário.

Anteriormente acessamos o recurso <http://localhost:8000/public/login> e houve êxito na autenticação.

Nesse processo o servidor responde com 200 ok.   
E gera um token de acesso.   
Esse token é enviado pelo servidor e pode ser armazenado pelo cliente.   
  
  
Se acessarmos outro recurso da aplicação, como a página de pedidos, não conseguiremos autenticar diretamente.   
Isso acontece porque o recurso acessado no método POST anterior ficou restrito àquela única requisição.   
Como o HTTP é stateless, os dados da sessão não são mantidos automaticamente.



Ou seja, para acessar outros recursos da aplicação, nesse caso a pagina de pedidos, que é um recurso apenas para usuários autenticados. É importante que essas requisições sejam enviadas com o dados de acesso.   
Para que a requisição seja respondida com sucesso.

Uma requisição (por exemplo, GET para a página de pedidos), o token de acesso pode ser informado no cabeçalho da requisição, permitindo a autenticação sem a necessidade de realizar login novamente.  
  
1 Obtendo token - envie um post ao <http://localhost:8000/public/login> e copie o token na resposta

2 Envie um get ao recurso <http://localhost:8000/pedidos>

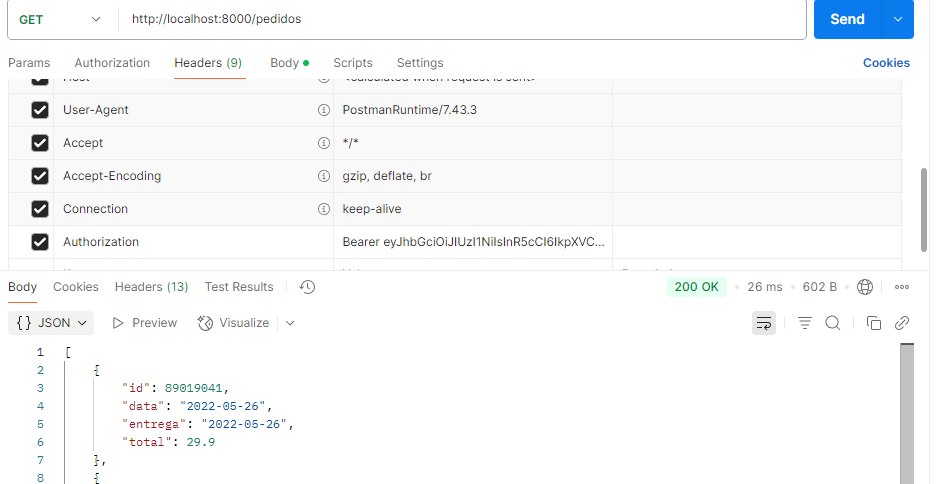
3 no get, edite o cabeçalho “HEADERS”

3 Incremente uma nova linha,

Primeiro campo escreva “Authorization”

Segundo campo escreva “Bearer yJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InRoaWFnb0BnbWFpbC5jb20iLCJzZW5oYSI6IjEyMyIsImlhdCI6MTc0NjIyNDc0MCwiZXhwIjoxNzQ2MjY3OTQwfQ.EigWjF\_kjYjJ3xTFuLjN8FOtEoDeV48OSB2Y2XN0BjU”  
  
Bearer tokenID

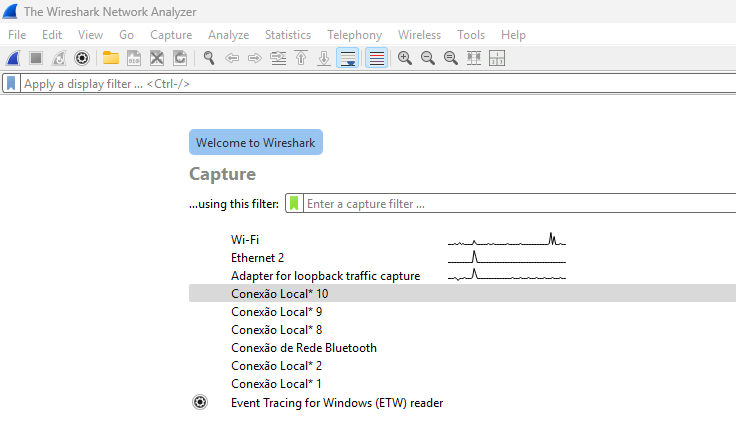
Resultado:

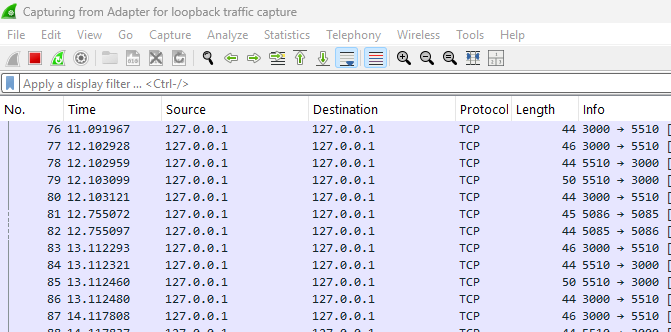


**ANALÍSE DE PACOTES COM WIRESHARK**

-Wireshark - Permite verificar pacotes na rede, verificar mensagens e etc...  
-Inicie os servidores - Backend e Frontend

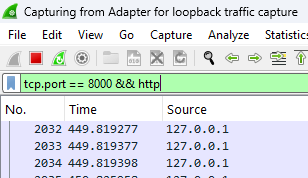
-Ao abrir, podemos selecionar a interface de rede a ser monitorado



-Iremos selecionar a interface de loopback - Permite que o dispositivo teste seus próprios componentes de rede ou interface. 

-Com a interface, de loopback na tela, podemos aplicar filtro para capturar pacotes específicos, nesse contexto. Iremos depurar o protocolo HTTP. Afim, de identificar se ao realizar logins, ou no trafego de dados senciveis se o conteúdo desses pacotes podem ser capturados. Se isso ocorrer, é uma falha. Sendo necessário correção de segurança.

Filtro: tcp.port == 8000 && http

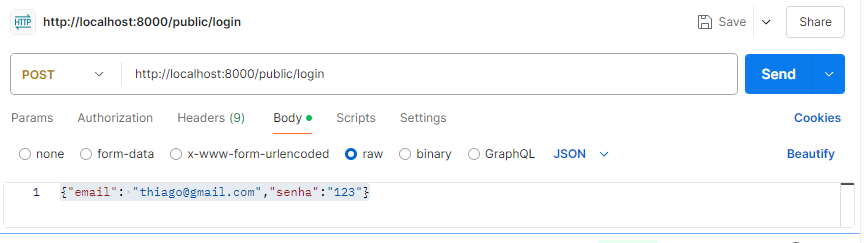


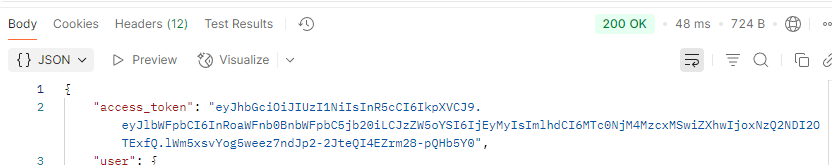
Filtragem de Conexões TCP - na porta 8000 - protocolo - HTTP

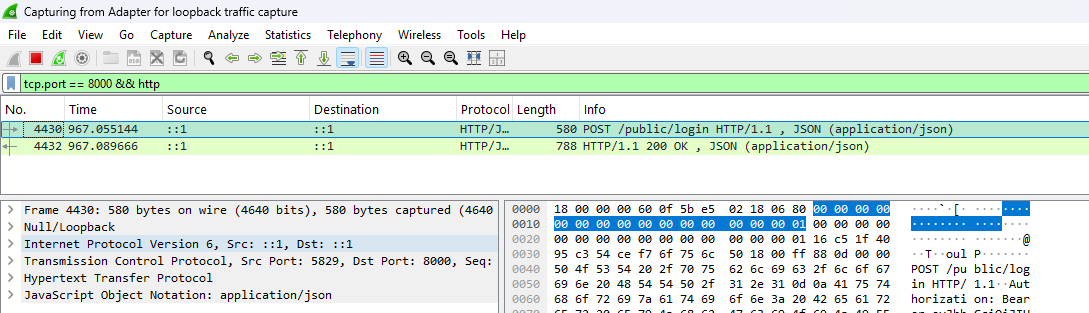
A tela, pode aparecer em branco, isso ocorre, pois não houve nenhuma requisição ao backend na porta 8000

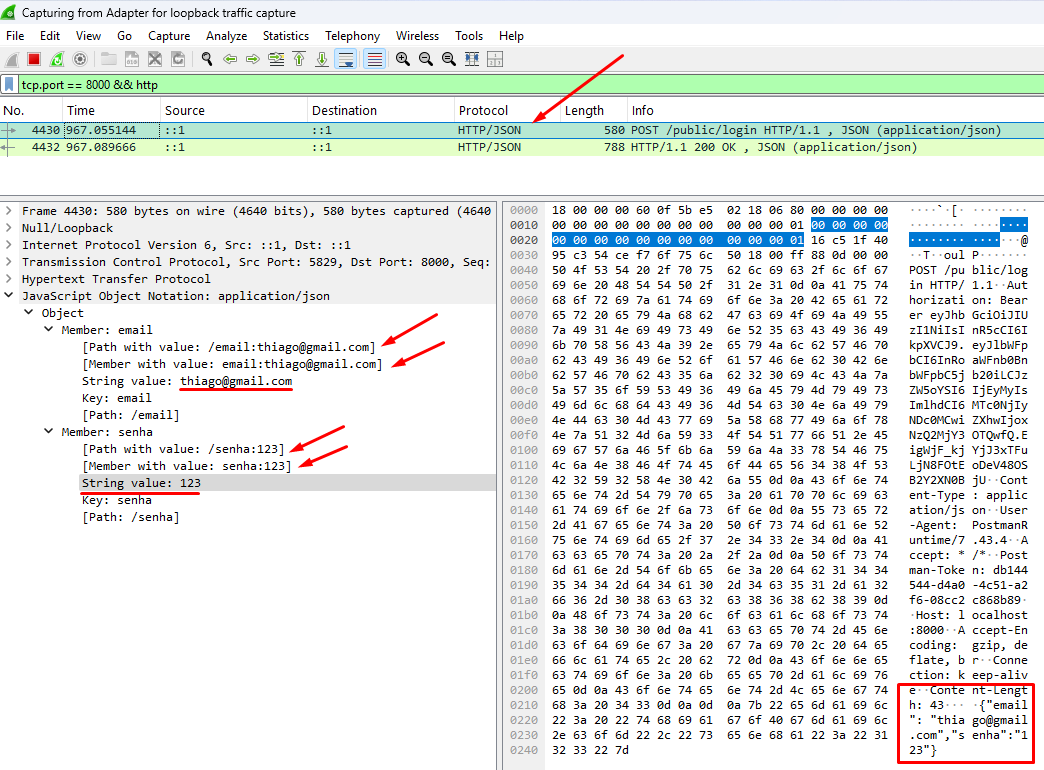
No postman, iremos enviar uma requisição ao backend ao recurso /public/login e monitorar o comportamento com o wireshark  
-Tipo POST  
-Endereço backend - http://localhost:8000/public/login  
-JSON RAW - mensagen: {"email": "thiago@gmail.com","senha":"123"}

Requisição:

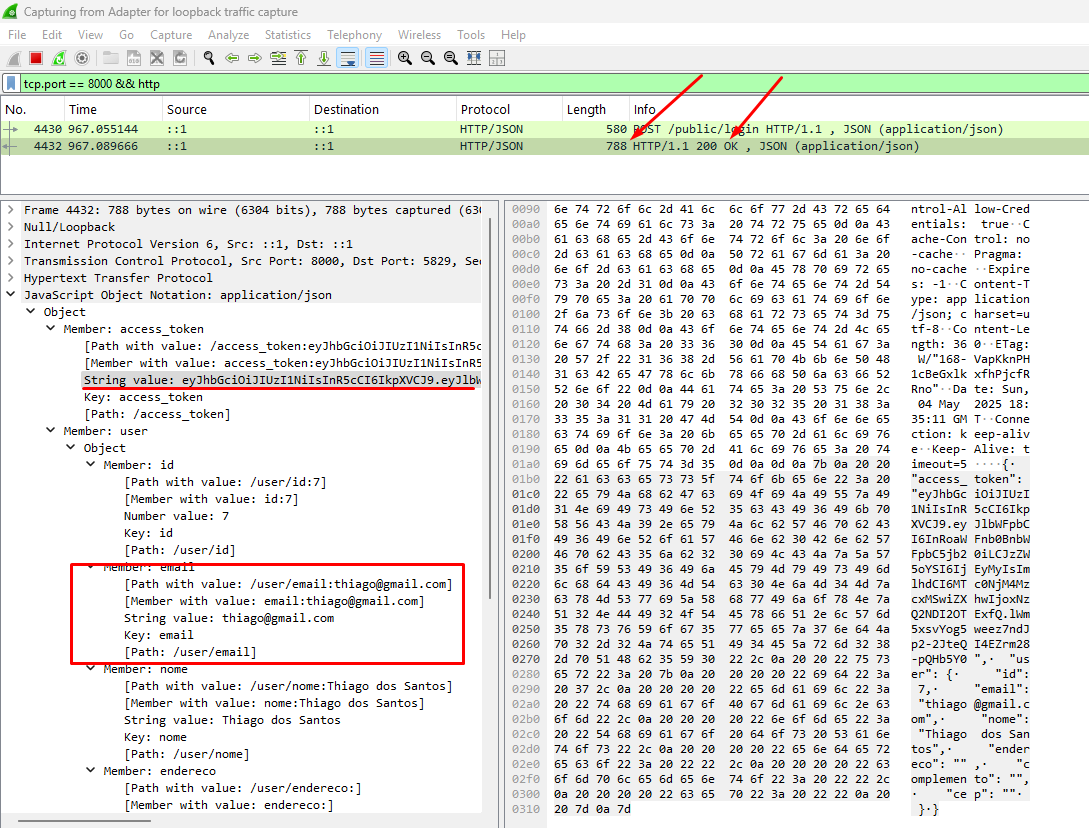


Reposta:  


Pacote capturado no wireshark:  


Conteúdo do pacote - Requisição enviada:  


Conteúdo do pacote - Resposta do servidor:



O comportamento observado revela que:

1 - O protocolo http não tem uma camada de segurança

2 - A ausência de segurança permite ver a requisição e resposta

3 - É possível capturar email e senha, ou o token de acesso

Com esses dados é possível realizar os seguintes ataques: Man in the middle, e roubo de sessão com o token de acesso, e envio de requisições ao servidor com o token de acesso e acessar outros recursos da aplicação.  
  
- Para garantir a segurança da aplicação é necessário modernizar o processo que envolve o HTTP

**HTTP/ HTTPS/ SSL/TLS -**

SSL (Secure Sockets Layer)

- Protocolo de segurança para criptografar e autenticar comunicações entre cliente e servidor.

- Substituído pelo TLS, mas o termo ainda é usado.

TLS (Transport Layer Security)

- Sucessor do SSL, com melhorias em segurança e eficiência.

- Padrão atual para criptografia de dados em trânsito.

- Usa criptografia assimétrica (autenticação) e simétrica (troca de dados).

- Opera sobre TCP, frequentemente chamado de SSL/TLS no uso comum.

Socket HTTPS

- Conexão de rede que usa HTTPS (HTTP sobre TLS/SSL).

- Socket é uma abstração para comunicação bidirecional.

- No HTTPS, o socket é configurado com TLS para criptografia, formando um "socket seguro".

- Exemplo: ao acessar `https://exemplo.com`, o navegador usa um socket seguro com TLS.

Relação:

- SSL/TLS: Protocolos de criptografia que garantem segurança.

- Socket HTTPS: Implementação de um socket de rede usando TLS para proteger HTTP (HTTPS).

- TLS (ou SSL) é a base que permite ao socket HTTPS operar com segurança.

Fluxo Típico:

1. Cliente inicia conexão com servidor via socket.

2. TLS realiza handshake: negociação de chaves e verificação de certificado.

3. Socket torna-se seguro, e dados HTTP são transmitidos como HTTPS, criptografados por TLS.

**IMPLEMENTAÇÃO SSL/TSL/HTTPS**

-OpenSSL - O OpenSSL é uma biblioteca de código aberto que fornece ferramentas e funções para implementar protocolos de segurança, como SSL (Secure Sockets Layer) e TLS (Transport Layer Security), usados para criptografar comunicações na internet. Ele oferece recursos para:

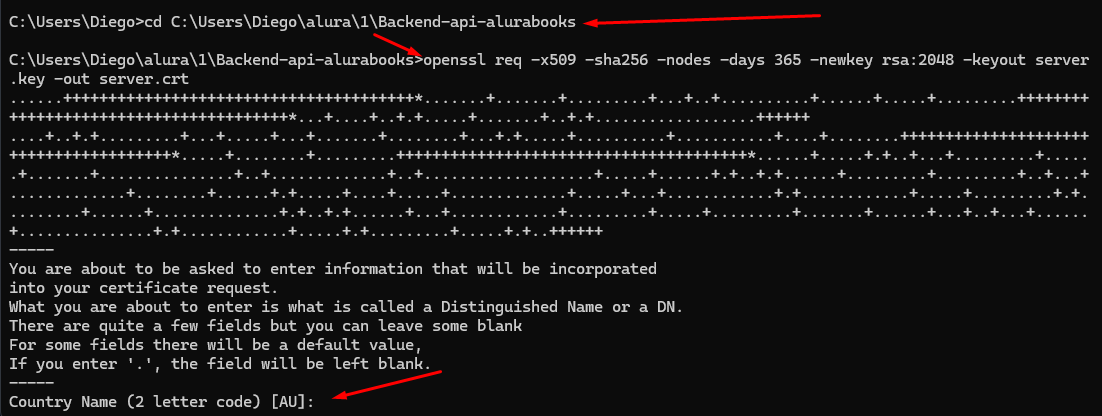
Criptografia: Algoritmos como AES, RSA, e SHA para proteger dados.

Gerenciamento de certificados: Criação, validação e manipulação de certificados digitais.

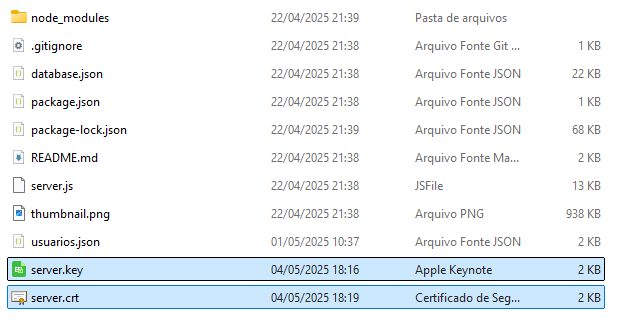
Conexões seguras: Estabelecimento de comunicações seguras em aplicações, como HTTPS em navegadores ou VPNs.

1 Com o OpenSSL devidamente configurado, vá até a pasta do servidor backend  
2 Rode o comando: openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout server.key -out server.crt

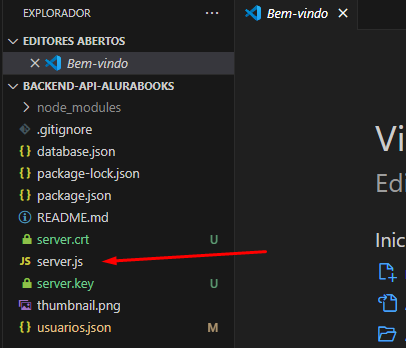
- Neste código temos uma requisição para gerar uma chave, por isso o server.key, e um certificado, um server.crt



- Após inserir o comando, será chamado um prompt para configuração do certificado

- Após inserir as informações do certificado é gerado os arquivos de server.crt e server.key  
  


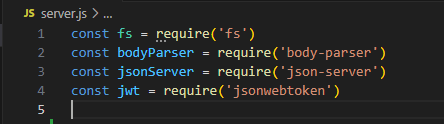
3 Após a geração dos arquivos, abra a pasta do backend no VScode



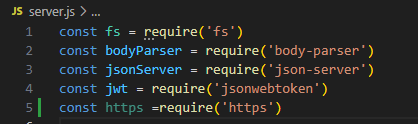
4 abra o arquivo server.js

- Importe o modulo https

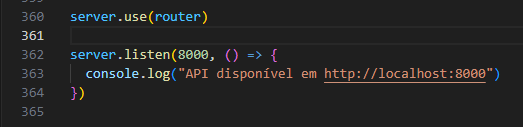
antes



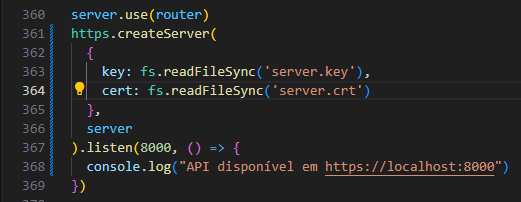
Depois



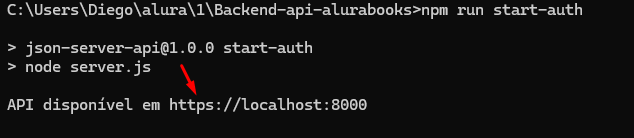
Antes



Depois



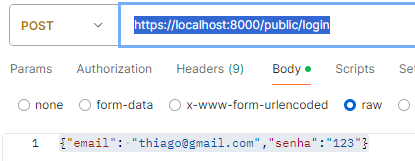
Reinicie os servidores

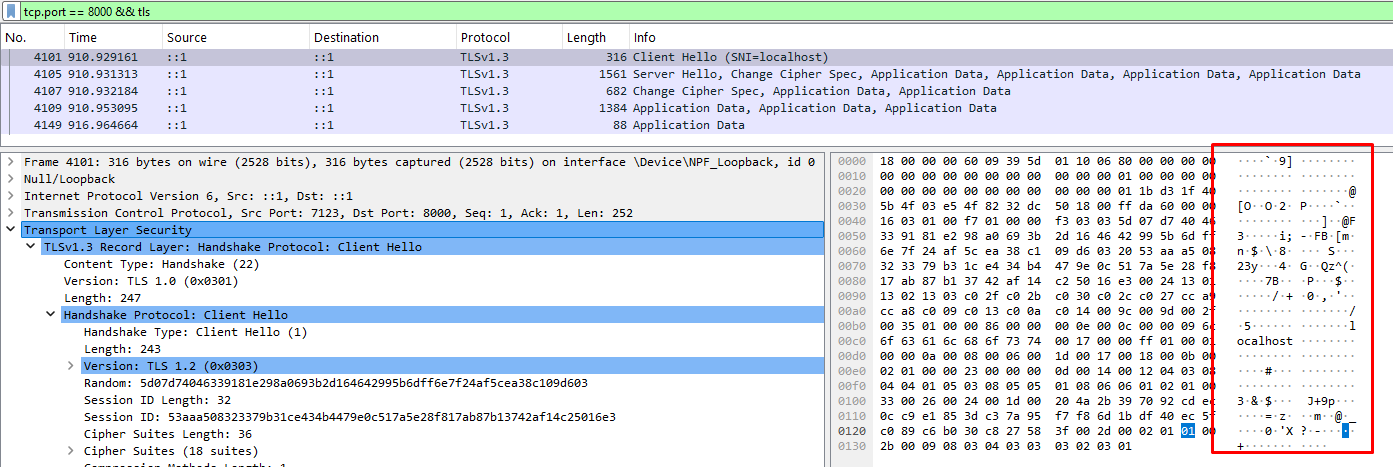


- Como visto, no log da API, já mostra saida via https

- Voltemos ao wireshark e postman, e repetiremos os testes de envio requisição post e validar criptografia do usuário e senha

- filtro wireshark: tcp.port == 8000 && tls





**VERIFICANDO CERTIFICADO E CHAVE CRIPTOGRAFICA**

- Arquivos:

server.crt  
server.key

Há um comando específico dentro do OpenSSL para verificar o conteúdo de cada um desses arquivos.

Comando OpenSSL para verificar o conteúdo de texto dentro do arquivo server.crt:

openssl x509 -in server.crt -text  
  
- As informações do arquivo server.crt, temos o país, a organização, temos assinatura digital, chave pública, chave privada e o início do certificado.

Comando OpenSSL para verificar a chave criptográfica arquivo server.key

openssl rsa -in server.key -text -noout

- O rsa indica o algoritmo de criptografia utilizado com essa chave. Será exibida uma série de informações, indicando que se trata de uma chave privada com 2048 bits.

**FUNCIONALIDADE**

Como funcionam a chave e o certificado digital na prática

A criptografia transforma dados em informações ilegíveis, que só podem ser decifradas com a chave correta.

Na criptografia assimétrica:

- Chave privada: Fica guardada no servidor e é usada para descriptografar mensagens.

- Chave pública: Está no certificado digital do servidor e é compartilhada com clientes para criptografar mensagens.

- Certificado digital: É a identidade do servidor, contendo informações como organização, API, chave pública e assinatura digital. É apresentado aos clientes para estabelecer conexões seguras.

Processo:

1. O cliente usa a chave pública (do certificado) para criptografar uma mensagem.

2. A mensagem criptografada é enviada ao servidor.

3. O servidor usa a chave privada para descriptografar a mensagem.

Esse mecanismo garante segurança na comunicação, como no protocolo HTTPS, que usa camadas adicionais como SSL e TLS (a serem exploradas em outra aula).

**MANIPULAÇÃO DE PARÂMETROS NO HTTP**

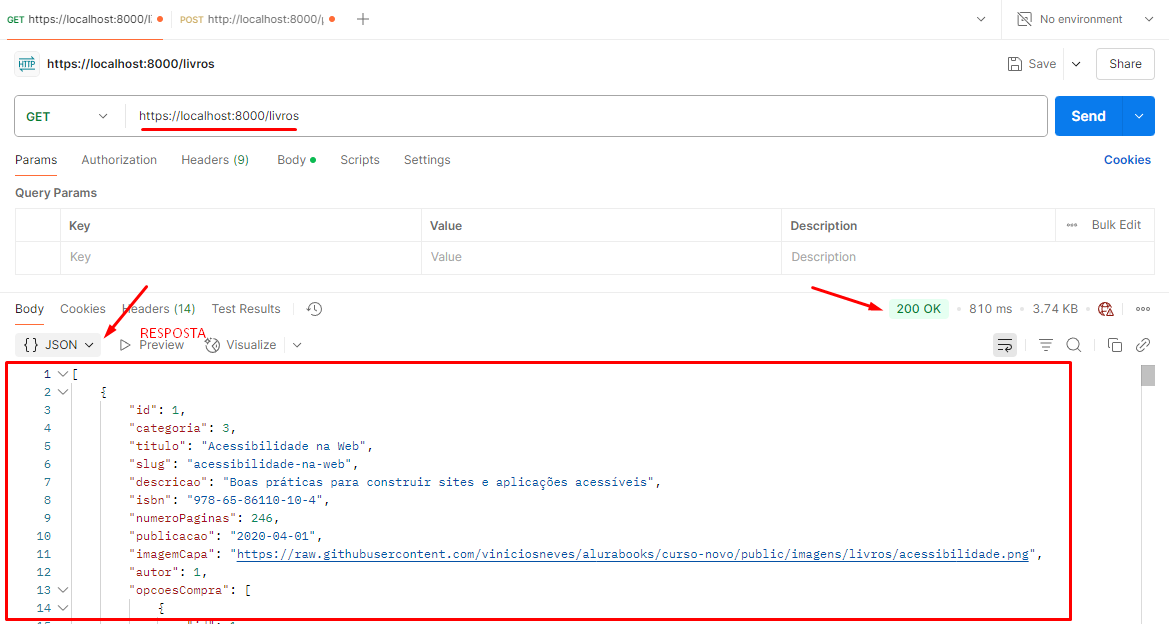
- HTTP métodos GET e POST

- Manipuação de URL

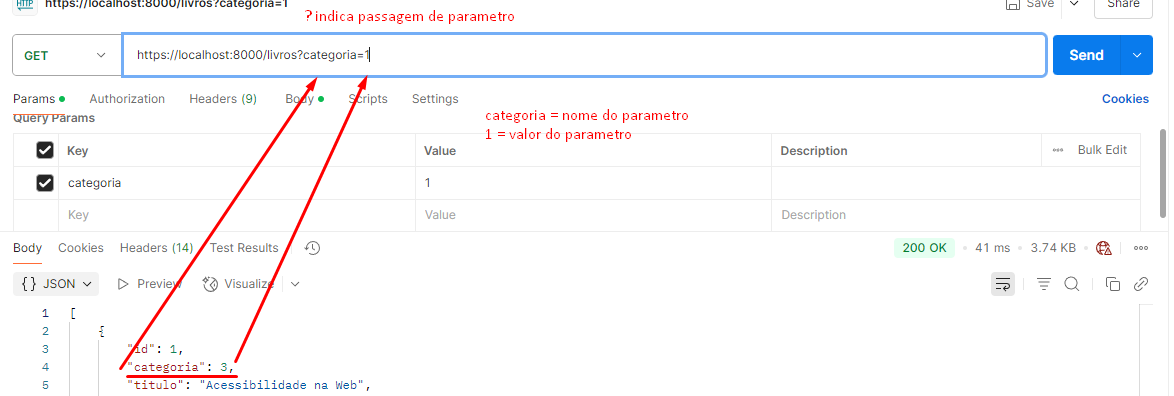
- Inserção de dados com POST

REQUISIÇÕES GET HTTP

1 A aplicação dispõe de uma sessão chamada livros e com o método GET podemos fazer uma requisisão no recurso da aplicação - <https://localhost:8000/livros> para obter informações dos livros cadastrados; Essa ação tem por finalidade apenas de validar como os livros estão cadastrados e organizados.



Como verificado, ao enviar a requisição get ao recurso /livros a aplicação retorna as informações dos livros cadastrados. Podemos verificar na resposta, como os livros estão organizados, e cadastrados.  
  
Um ponto de atenção é que os livros estão organizados por categorias. Sabendo disso, podemos usar essa informação e passar um paramêtro via get - post

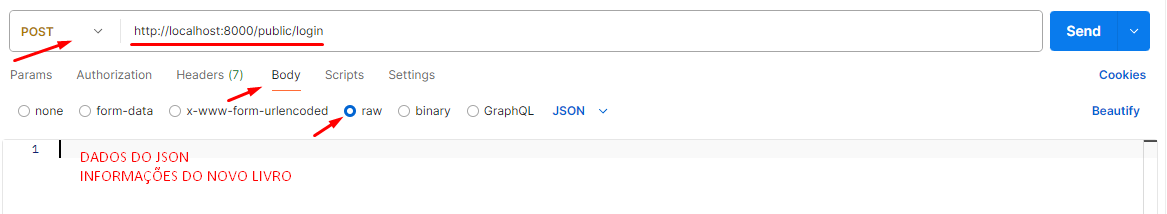


Assim a resposta que teremos é os livros cadastrados na categoria 1 - veja as demais categorias

INSERINDO DADOS COM POST

- Inserindo novo livro na categoria

- O novo livro será alocado em <https://localhost:8000/livros>



======= Insira as informações no campo e send  
{

"id": 55,

"categoria": 3,

"titulo": "DevOps",

"slug": "novo-livro",

"descricao": "Livro de Introducao a DevOps",

"isbn": "978-65-1111-11-1",

"numeroPaginas": 200,

"publicacao": "2023-01-01",

"imagemCapa": "heetps://raw.githubusercontent.com/viniciosneves/alurabooks/cursp-novo/public/imagens/livros/acessibilidade.png",

"autor": 1,

"opcoesCompra": [{

"id": 1,

"titulo": "E-book",

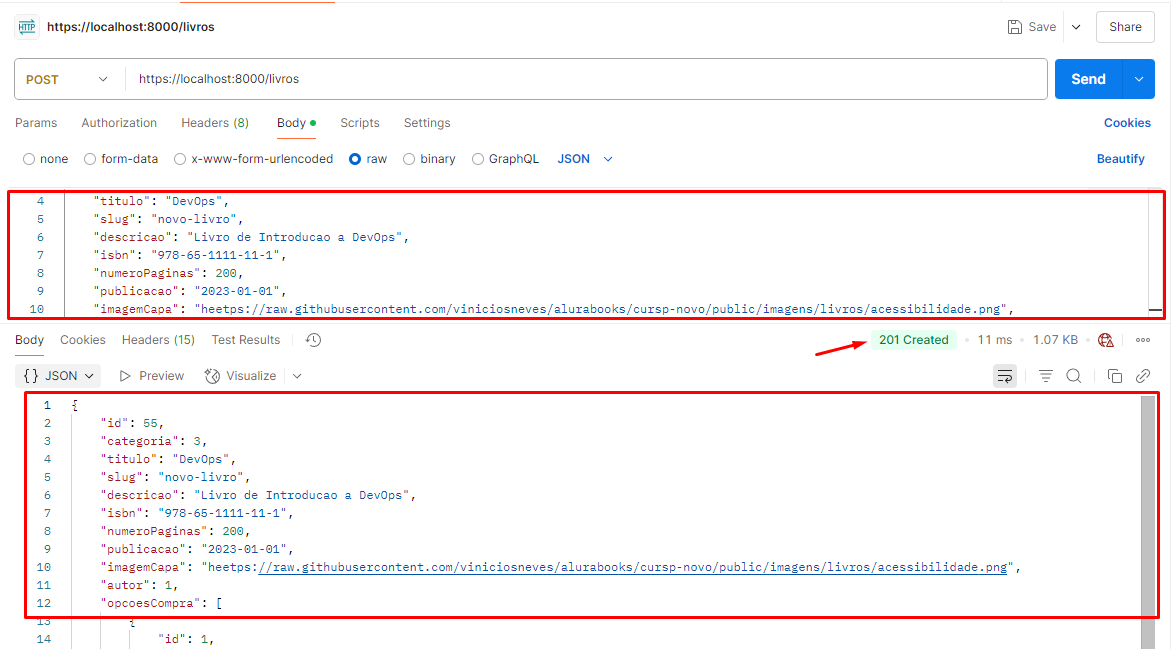
"preco": 29.9,

"formatos": [".pdf", ".pub", ".mob"]

}],

"sobre": "Compre esse livro e aprenda sobre DevOps."

}

  
  
O novo livro foi cadastrado, e pode ser verificado com o metodo get