

Headers HTTP e Autenticação

Relembrando a última aula

- Instalamos o SQLite
- Instalamos as dependências do Prisma
 - Prisma e `@prisma/client`
- Instanciamos o `PrismaClient` no nosso Controller
- Utilizamos os métodos do Prisma para
 - Buscar todos
 - Buscar um
 - Inserir um novo registro

Para essa aula

- Commitar as alterações da aula passada
- Implementar métodos
 - Deletar
 - Alterar
- Tratar possíveis erros nas operações com Try Catch

Após isso

- Headers HTTP
 - O que são
 - Para que servem
 - Como acessar
- Autenticação
 - Conceito e prática
- Tokens
 - JWT
 - Assinar
 - Secret Key

Terminando implementação

Deletar usuário

```
const deletedUser = await prisma.user.delete({  
  where: {  
    id: "84e37ff7-0f7e-41c6-9cad-d35ceb002991",  
  },  
});
```

Alterar usuário

```
const updatedUser = await prisma.user.update({  
  where: {  
    id: "84e37ff7-0f7e-41c6-9cad-d35ceb002991",  
  },  
  data: {  
    name: "Arindam",  
  },  
});
```

Tratamento de erros durante operações

Try Catch

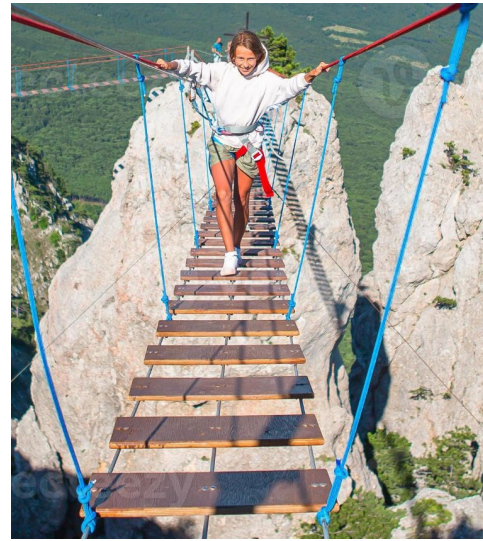
O que é o Try Catch

Estrutura de **controle de fluxo** (como if, por exemplo) usada para **capturar e tratar erros** que podem ocorrer durante a execução do código.

Analogia:

o **try** é a tentativa de passar uma ponte

o **catch** é uma rede de contenção pro caso de escorregar da ponte



Para que serve?

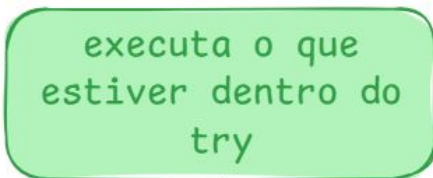
- Evita que a aplicação trave quando encontra problemas
- Fornece uma experiência melhor pro usuário (mensagens mais claras)
- Permite que o programa continue executando, mesmo após o erro
- Registra os erros
- Implementa fluxos alternativos quando algo falha

Como fica no código?

**Código de exemplo apenas*

```
16  function inserirUsuario(req, res) {
17    //essa funcao pode dar algum problema?
18    //vamos circundar ela com try catch
19
20    //tente isso
21    try {
22      await this.prisma.user.create(/*dados do usuario*/)
23    } catch (error) {
24      //aqui vem o tratamento necessario
25      console.log(error)
26    }
27  }
```

começo da função



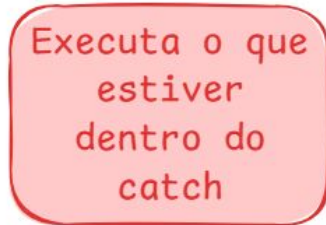
executa o que
estiver dentro do
try



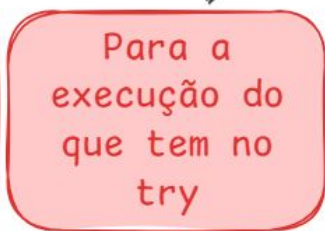
ocorreu algum erro?

sim

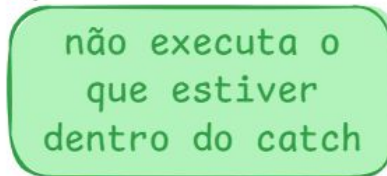
não



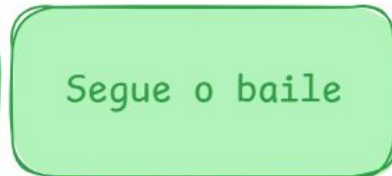
Executa o que
estiver
dentro do
catch



Para a
execução do
que tem no
try



não executa o
que estiver
dentro do catch



Segue o baile

Throw

Método que permite **lançar erros manualmente**

É como ativar um alarme quando detecta algo suspeito

Analogia: Você é um segurança

Throw: apertar o botão de emergência pois viu algo suspeito

Catch: quem vai ouvir o alarme

Try: Local que você está guardando (de onde você vai acionar o botão)

```
1  function exemploTryCatch(entrada) {
2      try {
3          const valor_de_entrada = parseInt(entrada);
4
5          if (isNaN(valor_de_entrada)) {
6              throw new Error("Entrada nao eh um numero valido");
7          }
8
9          const resultado = valor_de_entrada / 10;
10
11         console.log(`resultado: ${resultado}`);
12     } catch (error) {
13         console.log(error.message);
14     }
15 }
```

Vamos executar esse código pra ver o que acontece

Posso capturar diferentes tipos de erros?

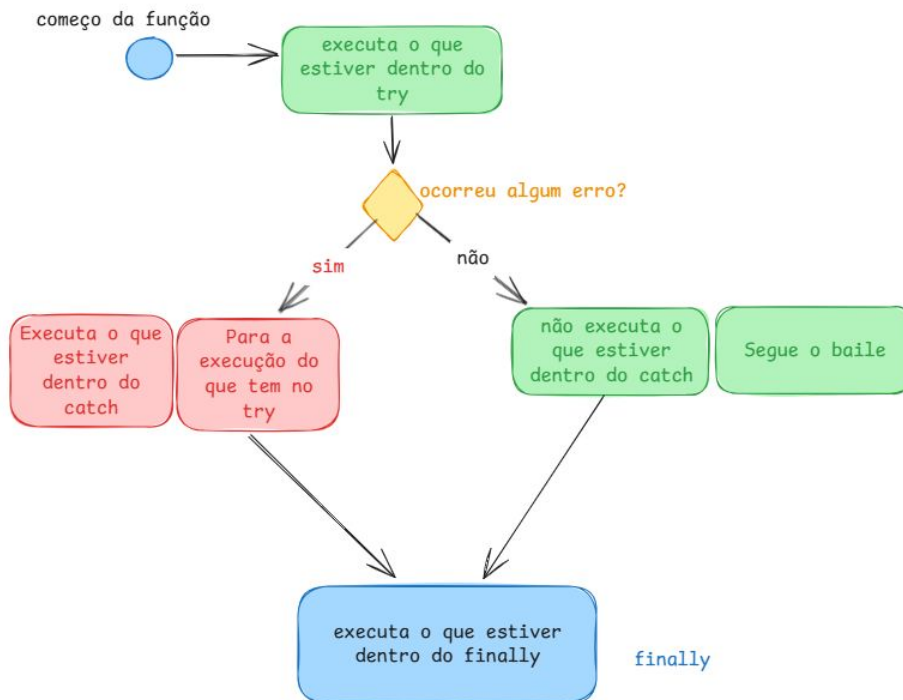
Sim!

Nesse caso, estamos
identificando o tipo do
erro

```
try {  
    let numero = parseInt(prompt("Digite um número:"));  
    let resultado = 10 / numero;  
    console.log(resultado);  
} catch (error) {  
    if (error instanceof TypeError) {  
        console.log("Erro de tipo de dado!");  
    } else if (error instanceof ReferenceError) {  
        console.log("Variável não definida!");  
    } else {  
        console.log("Erro desconhecido:", error.message);  
    }  
}
```


Finally

*Bloco de código que é executado **sempre**, mesmo com erro*



Quando devo usar try catch?

- Operações com arquivos (eles podem não existir)
- Conexão de rede (pode falhar)
- Conversão de tipos de dados (podem ser inválidos)
- Operações matemáticas (divisão por zero)
- Acesso a APIs externas (podem estar fora do ar)
- Acesso ao banco de dados (pode estar com a conexão fechada, corrompido, etc)

Vamos adicionar try catch em todas as
nossas operações do banco de dados

Headers HTTP

O que são

Metadados que acompanham toda requisição e resposta na web.

Carregam *informações extras* sobre a comunicação entre o cliente e o servidor

Analogia: Enviando uma carta pelo correio

Conteúdo da carta: body

Carteiro: Método HTTP

Envelope: Headers (dizem *como* interpretar a mensagem, não o conteúdo dela)

Que tipo de informação vem neles?

Autenticação: quem está fazendo a requisição?

Autorização: esse usuário tem permissão?

Tipo de conteúdo: JSON, HTML, imagem, etc

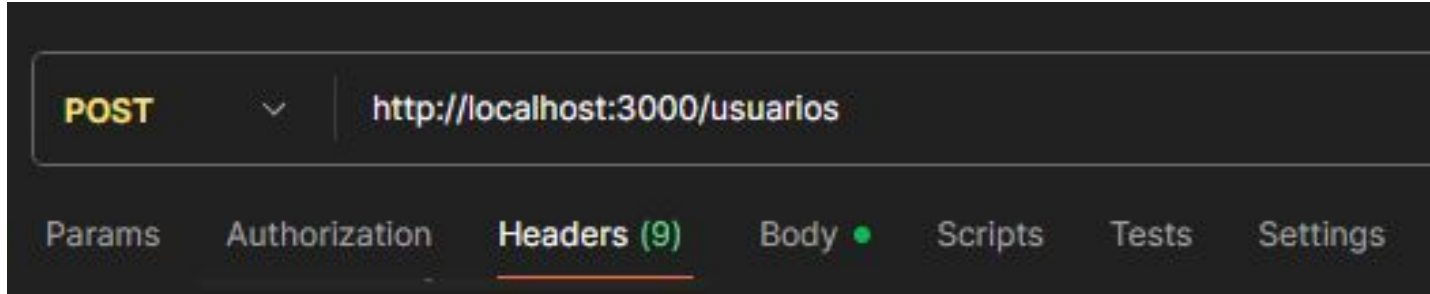
Compressão: possui dados comprimidos?

Cache: pode guardar dados em cache?

Segurança: proteção contra ataques

Informação do cliente: navegador, dispositivo, idioma, etc

Visualizando Headers no Postman



Capturando Headers no Node

```
const headers = req.headers;
```


Params

Authorization

Headers (9)


Body ●










Scripts

Tests

Settings

Headers

 Hide auto-generated headers

	Key		Value
<input checked="" type="checkbox"/>	Cache-Control		no-cache
<input checked="" type="checkbox"/>	Postman-Token		<calculated when request is sent>
<input checked="" type="checkbox"/>	Content-Type		application/json
<input checked="" type="checkbox"/>	Content-Length		<calculated when request is sent>
<input checked="" type="checkbox"/>	Host		<calculated when request is sent>
<input checked="" type="checkbox"/>	User-Agent		PostmanRuntime/7.46.1
<input checked="" type="checkbox"/>	Accept		/*/*
<input checked="" type="checkbox"/>	Accept-Encoding		gzip, deflate, br
<input checked="" type="checkbox"/>	Connection		keep-alive

Exemplo real de uso

Imagine que você precisa baixar um determinado arquivo *em partes*

Motivo: conexão lenta, então precisamos baixar arquivos de tamanho menor

Como fazer: usar o header ***Range***



Range

bytes 0-9999

Nesse caso, nosso servidor vai receber isso e poder responder com apenas uma parte do recurso, conforme solicitado pelo cliente

Autenticando usuários por token

Header: ***Authorization***

Usado para dizer “Quem é você”, quando realiza uma requisição

Carrega informações como: senha, token, chave, etc

<tipo> <chave>

Basic 123456abc

Por que eu preciso de tokens e autenticação?

- Nem todo mundo pode acessar os dados

Imagine se todo mundo pudesse acessar a API da XP investimentos, por exemplo, e saber quanto dinheiro você tem investido lá, e pudesse manipular esse dinheiro

Tipos de Token

Basic

Funciona com **usuário e senha** codificados em Base64

Base64 é um método de codificação que transforma dados em uma sequência de caracteres de texto ASCII

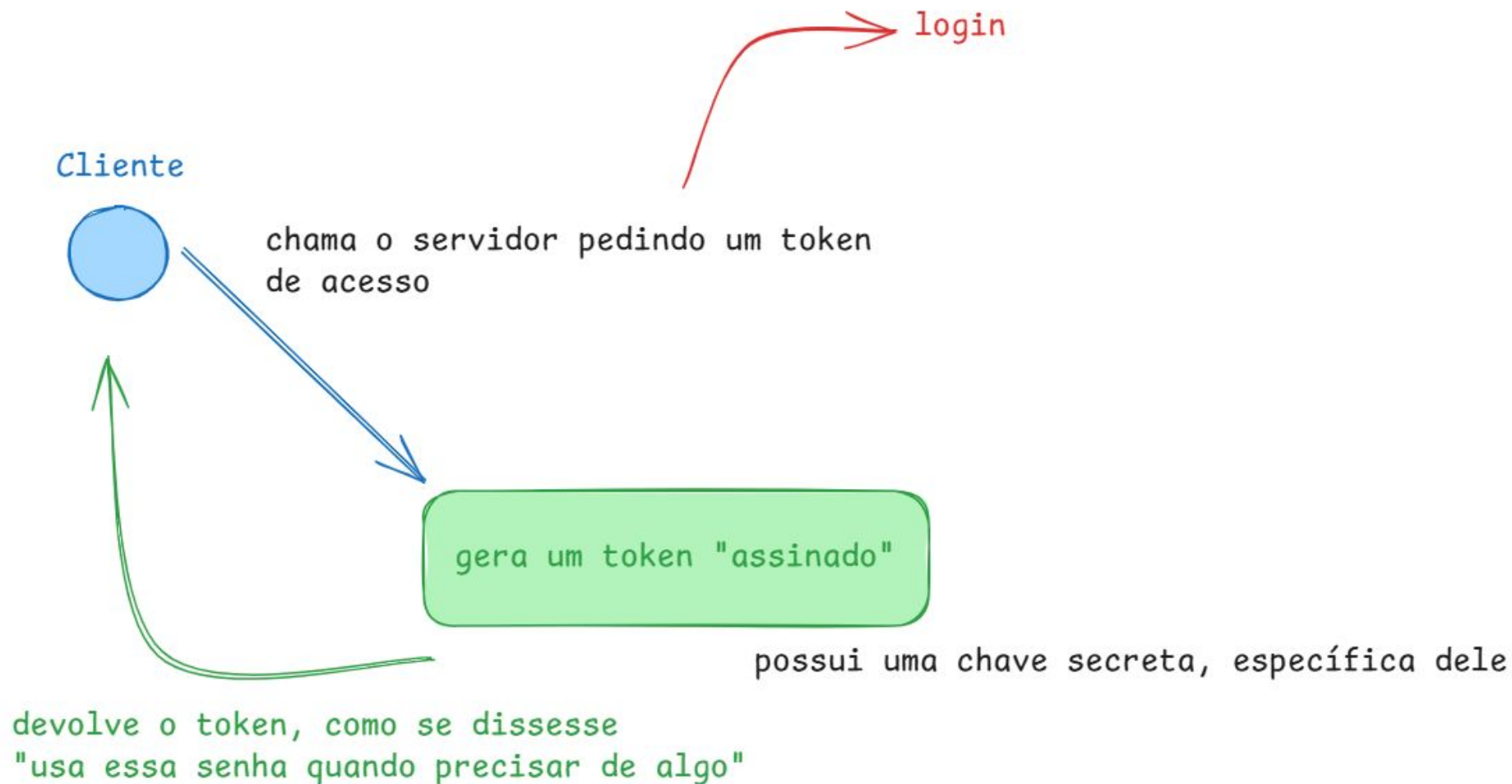
usuário: admin

senha: 123

em base64: YWRtaW46MTIz

Basic YWRtaW46MTIz

Bearer



JWT - Json Web Token

Token assinado que garante autenticidade: se o conteúdo mudar, a assinatura não vai mais bater

FORMAÇÃO DO JWT

Formação

header.payload.signature

Header: diz qual algoritmo foi usado na assinatura

Payload: onde ficam os dados do usuário (email, nome, id, role, etc)

Signature: garantia de que o token não foi adulterado

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true,  
  "iat": 1516239022  
}
```

SECRET

Valid secret

a-string-secret-at-least-256-bits-long

JWT.IO

Vendo na prática

Gerar um token através do jsonwebtoken

Validar informações desse token no jwt.io

Gerar um token pelo jwt.io

verificar os dados desse token via código

Vamos autenticar a nossa API com Basic
Token