Fluxo de Dados no Backend

Labenu_

O que vamos ver hoje?

- Fluxo e validação de dados no backend
- Status codes HTTP mais comuns
- Refatoração de um endpoint criado na aula anterior

Introdução

Labenu_

Fluxo de Dados

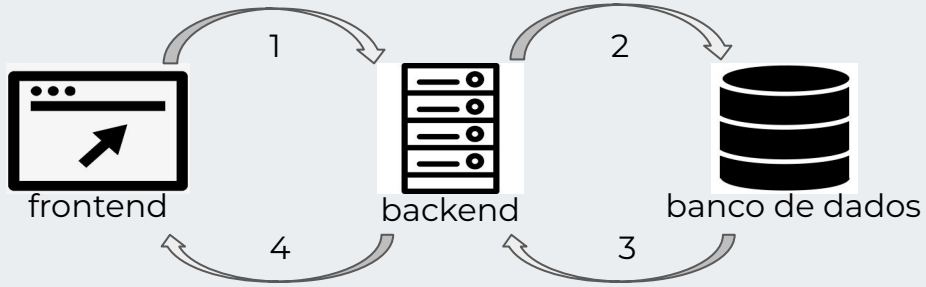
Agora que já vimos o essencial da sintaxe do Express, podemos dar mais atenção à *lógica* dos nossos endpoints. Por exemplo:

- O que acontece quando um id inválido é passado na url?
- O que acontece quando o token de autenticação não é informado?

Fluxo de Dados

Essas perguntas ilustram dois momentos importantes que ainda estamos ignorando no fluxo da nossa aplicação:

1. Validação das entradas da requisição
2. Consulta à base de dados
3. Validação dos resultados da consulta
4. Envio da resposta



Fluxo de Dados

No caso de endpoints mais simples, o uso de blocos **if/else** pode ser suficiente para garantir uma resposta adequada

```
app.get("/playlists", (req: Request, res: Response) => {

 const token = req.headers.authorization

 if(!token){

   res.status(401).send(
     "Verifique se você está passando o header 'Authorization'"
   )
 } else{
   res.send(playlists)
 }
})
```

Fluxo de Dados

Muitos endpoints, no entanto, exigem que a execução do código seja interrompida caso alguma verificação falhe, o que não acontece com o mero envio da resposta.

```
app.delete('/users/:id', (req: Request, res: Response) => {

  if (!req.headers.authorization) {
     res.status(401).end() // essa linha NÃO encerra a lógica
  }

  const index: number = users.findIndex(
     user => user.id === Number(req.params.id)
  )

  users.splice(index, 1) // essa linha será executada

  res.status(200).end()

})
```

Fluxo de Dados

Poderíamos utilizar a *keyword* ***return*** para interromper a execução do nosso código. Porém, existe uma alternativa para que ele fique mais limpo e reaproveitável: a *keyword* ***throw***

```
app.delete('/users/:id', (req: Request, res: Response) => {
  try {
     if (!req.headers.authorization) {
        throw new Error()
     }

     const index: number = users.findIndex(
        user => user.id === Number(req.params.id)
     )

     users.splice(index, 1) // essa linha NÃO será mais executada

     res.status(200).end()
  } catch {
     res.status(401).end()
  }
})
```

Fluxo de Dados

- Quando atiramos um erro dentro de um bloco **try**, a execução do bloco é interrompida e retomada no bloco **catch**, dentro do qual teremos acesso ao erro

- A classe Error, assim como a classe Date, é nativa do JS. Ela também recebe um parâmetro (opcional) ao ser instanciada: uma mensagem (string)

```
try {
     throw new Error("Something went wrong")

     console.log("Success!") // essa linha não será executada
} catch (err) {
     console.log(err.message)  // imprime a mensagem "Something went wrong"
}
```

Exercício 1

O que será impresso no console caso a função abaixo seja chamada com o número 10 como argumento? E com o número 20?

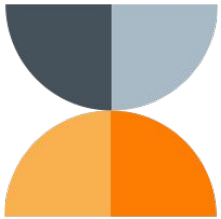
```
const validateAge = (age:number)=> {
 try {
   console.log("mensagem 1")

   if(age < 18){
     throw new Error("mensagem 2")
   }
   console.log("mensagem 3")
 } catch (error: any) {
   console.log(error.message)
 }
}
```

Pausa para relaxar 😴

10 min

As principais etapas do fluxo de dados no backend são:



1. Validação das entradas da requisição
2. Consulta à base de dados
3. Validação dos resultados da consulta
4. Envio da resposta

Status HTTP

Labenu_

Status HTTP

Como já vimos ~~há 3000 anos~~ em módulos anteriores, o ***status*** é um código numérico que resume o resultado de uma requisição

Status HTTP

Nos módulos de back, os códigos de status que utilizaremos com mais frequência serão os seguintes:

- 200 - Ok
  - status genérico de sucesso

- 201 - Created

Status HTTP

- 400 - Bad Request
  - status genérico de requisição inválida: método ou caminho inválidos, JSON inválido...

- 401 - Unauthorized
  - credenciais ausentes ou inválidas

- 403 - Forbidden
  - usuário não tem as permissões necessárias

Status HTTP

- 404 - Not Found
  - rota não encontrada
- 409 - Conflict
  - tentativa de criar um registro já existente
- 422 - Unprocessable Entity
  - requisição com parâmetros inválidos: idade negativa, senha muito curta, parâmetros ausentes...
- 500 - Internal Server Error
  - status genérico de erro no back

Exercício 2

Adicione validações ao endpoint de adicionar música à uma playlist. Elas devem responder as requisições com status:

- 401, caso o headers de **Authorization** não seja informado
- 404, caso não exista playlist com o **id** informado
- 409, caso já exista música com o **nome** e **artista** informados
- 422, caso algum dado da música esteja ausente
- 500, em caso de algum erro não previsto

Resumo

Labenu_

Resumo 📓

As principais etapas do fluxo de dados no backend são:

1. Validação das entradas da requisição
2. Consulta à base de dados
3. Validação dos resultados da consulta
4. Envio da resposta

Resumo 📓

Status codes mais utilizados no curso

Sucesso

200 *OK*
201 *Created*

Erro de servidor

500 *Internal Server Error*

Erro de cliente

400 *Bad Request*
401 *Unauthorized*
403 *Forbidden*
404 *Not Found*
409 *Conflict*
422 *Unprocessable Entity*

Dúvidas? 🧐

Labenu_

Obrigado!