



UNISANTA

Backend com Express

UNISANTA



Índice

API com Express.....	3
Criando o workspace.....	4
Criando o projeto da API.....	5
Definindo o modelo de dados.....	8
Criando a estrutura de rotas.....	9
Criando a rota para /api/post.....	11
Testando o endpoint /api/post.....	12
Criando a rota para /api/getAll.....	13
Testando o endpoint /api/getAll.....	14
Criando a rota para /api/delete/id.....	15
Testando o endpoint /api/delete/id.....	16
Criando a rota para /api/update/id.....	17
Testando o endpoint /api/update/id.....	18
Histórico de revisões.....	19



API com Express

Nesta aula vamos desenvolver o programa que vai rodar no Backend, permitindo a manipulação de dados no Banco de Dados criado na aula anterior.

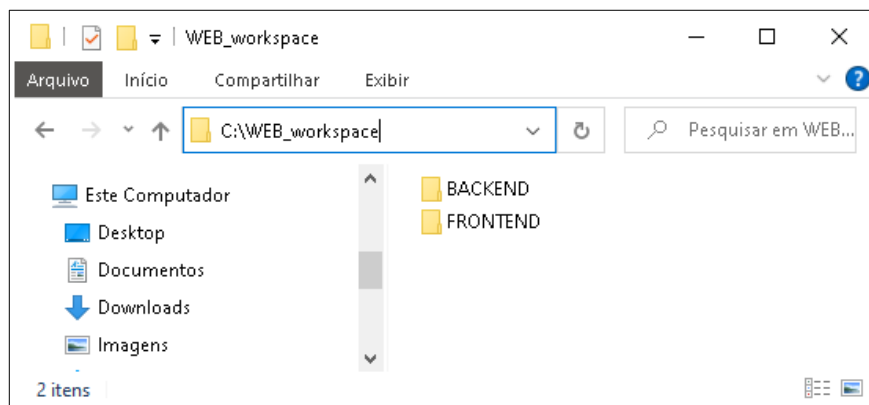
O Backend será desenvolvido no formato de uma WEB API, disponibilizando endpoints para a realização de operações CRUD (Create, Read, Update e Delete) no Banco de Dados.



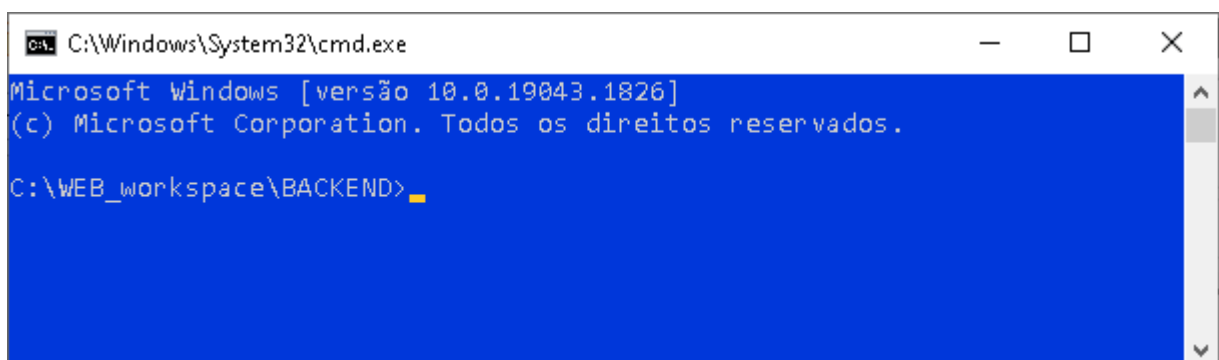
Criando o workspace

Acesse a pasta "C:\WEB_workspace" de seu computador, criada no início de nossos estudos.

Dentro dessa pasta crie uma nova pasta chamada de BACKEND (no mesmo nível da pasta FRONTEND criada anteriormente). Ex:



Acesse o prompt de comando do Windows dentro da pasta BACKEND. É aqui onde você executará os primeiros comandos de criação da API:



Criando o projeto da API

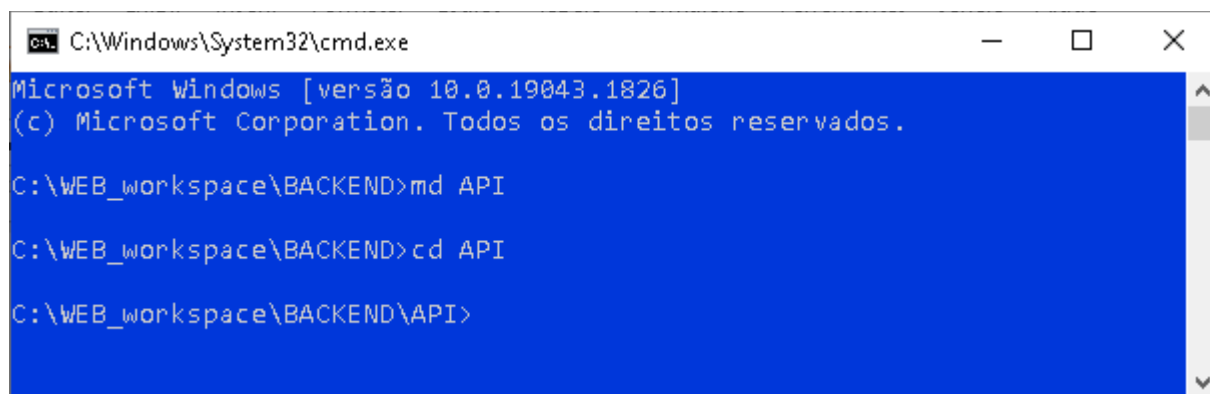
Abra o prompt de comando dentro da pasta BACKEND, e execute o seguinte comando para criar uma pasta chamada API para armazenar o código de nosso projeto:

md API

Entre na pasta API utilizando o seguinte comando:

cd API

Segue estado do prompt após executarmos os comandos acima:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [versão 10.0.19043.1826]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\WEB_workspace\BACKEND>md API

C:\WEB_workspace\BACKEND>cd API

C:\WEB_workspace\BACKEND\API>
```

Execute o seguinte comando para criar a estrutura inicial do projeto (respondendo com ENTER para todas as perguntas realizadas):

npm init

Após a execução desse comando, o arquivo **package.json** é criado dentro da pasta API.



Agora vamos adicionar os módulos do framework "**express**" ao nosso projeto:

```
npm i express --save
```

Agora vamos adicionar o módulo "**nodemon**" para monitorar as alterações em nosso código fonte e reiniciar o servidor automaticamente sempre que for detectada uma alteração no código durante o desenvolvimento:

```
npm i nodemon -g --save-dev
```

Agora vamos adicionar o módulo "**mongoose**" para permitir acesso ao Banco de Dados:

```
npm i mongoose --save
```

Execute o VsCode para permitir a edição do código dentro da pasta atual:

```
code .
```

Crie um arquivo chamado de "**index.js**" com o seguinte conteúdo:

```
const express = require('express');
const app = express();
app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader('Access-Control-Allow-Methods', 'HEAD, GET, POST, PATCH, DELETE');
  res.header(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept"
  );
  next();
});
app.use(express.json());
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server Started at ${PORT}`)
})
// Obtendo os parametros passados pela linha de comando
var userArgs = process.argv.slice(2);
var mongoURL = userArgs[0];

//Configurando a conexao com o Banco de Dados
var mongoose = require('mongoose');
mongoose.connect(mongoURL, {
  useNewUrlParser: true, useUnifiedTopology:
    true
});
mongoose.Promise = global.Promise;
const db = mongoose.connection;
db.on('error', (error) => {
  console.log(error)
})
db.once('connected', () => {
  console.log('Database Connected');
})
```



O código acima vai inicializar a API respondendo pela porta 3000. Em seguida vai iniciar uma conexão com o Banco De Dados utilizando a URL de conexão passada como parâmetro através da linha de comando.

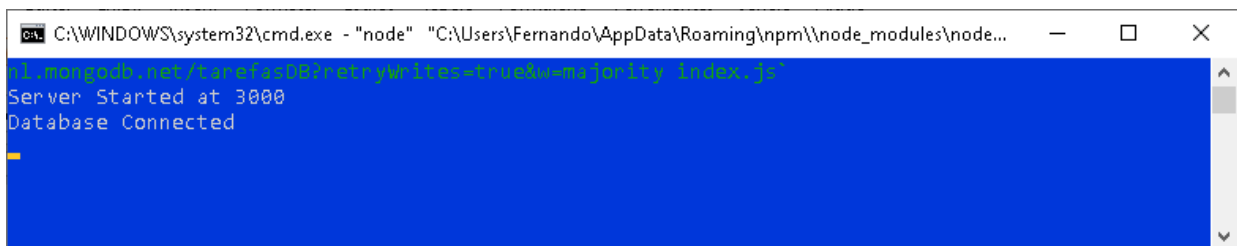
Caso a conexão com o Banco de Dados ocorra com sucesso, irá apresentar a mensagem "Database Connected", caso contrário apresentará uma mensagem indicando qual foi o erro ocorrido.

Execute o servidor através do seguinte comando:

```
nodemon index "coloque sua URL de conexão aqui"
```

Obs: Coloque sua URL de conexão com o banco de dados "entre aspas"

Deverá aparecer o seguinte resultado no console:



```
C:\WINDOWS\system32\cmd.exe - "node" "C:\Users\Fernando\AppData\Roaming\npm\node_modules\nodemon\index.js"
Server Started at 3000
Database Connected
```



Definindo o modelo de dados

Antes de criarmos as rotas dos endpoints, vamos definir o modelo dos dados a serem armazenados no MongoDB. Para isso vamos criar um arquivo chamado de **"API/models/tarefa.js"** com o seguinte conteúdo:

```
const mongoose = require('mongoose');
const schemaTarefa = new mongoose.Schema({
  descricao: {
    required: true,
    type: String
  },
  statusRealizada: {
    required: true,
    type: Boolean
  },
}, {
  versionKey: false
})
module.exports = mongoose.model('Tarefa', schemaTarefa)
```

O schema acima define o armazenamento dados na Collection "tarefas".

Dentro da Collection, cada tarefa (Document) possuirá:

- Um campo **"descricao"** do tipo **string**
- Um campo **"statusRealizada"** do tipo **boolean**.

Criando a estrutura de rotas

A nossa API deverá ser capaz de disponibilizar micro-serviços para manipular informações no Banco de Dados. Esses serviços permitirão realizar as operações de Gravação (CREATE), Leitura (READ), Atualização (UPDATE) e Remoção (DELETE) de informações no Banco. Esse conjunto de operações costuma-se chamar de CRUD, que são as iniciais das siglas em Inglês.

Para conseguir isso, devemos especificar um endereço (endpoint) para cada serviço disponibilizado. Adicionalmente ao endpoint, devemos também associar o método esperado para cada tipo de requisição.

Segue tabela com os endpoints dos serviços a serem disponibilizados por nossa API:

Endpoint	Método HTTP	Operação CRUD	Objetivo
/api/post	POST	CREATE	Grava uma nova tarefa
/api/getAll	GET	READ	Obter todas as tarefas (itens)
/api/update/id	PATCH	UPDATE	Atualiza a tarefa identificada pelo id
/api/delete/id	DELETE	DELETE	Elimina a tarefa identificada pelo id

Para cada endpoint deverá ser criada uma rota para tratar as solicitações realizadas pelo cliente. Para isso, vamos criar um arquivo chamado de "**API/routes/routes.js**", com o seguinte conteúdo, para iniciar a preparação da estrutura de rotas.

```
const express = require('express');  
  
const router = express.Router()  
  
module.exports = router;  
  
const modeloTarefa = require('../models/tarefa');
```

Agora vamos direcionar para o arquivo de rotas, o tratamento de todos os endpoints



iniciados pelo prefixo "/api" Para isso adicione as linhas abaixo destacadas em verde no arquivo **index.js**:

```
const express = require('express');
const app = express();
app.use(express.json());
const routes = require('./routes/routes');
app.use('/api', routes);
app.listen(3000, () => {
  console.log(`Server Started at ${3000}`)
})
// Obtendo os parametros passados pela linha de comando
var userArgs = process.argv.slice(2);
var mongoURL = userArgs[0];

//Configurando a conexao com o Banco de Dados
var mongoose = require('mongoose');
mongoose.connect(mongoURL, {
  useNewUrlParser: true, useUnifiedTopology:
    true
});
mongoose.Promise = global.Promise;
const db = mongoose.connection;
db.on('error', (error) => {
  console.log(error)
})
db.once('connected', () => {
  console.log('Database Connected');
})
```



Criando a rota para /api/post

Agora sim podemos inserir a rota para o método "POST" ao final do arquivo **"routes.js"**:

```
router.post('/post', async (req, res) => {
  const objetoTarefa = new modeloTarefa({
    descricao: req.body.descricao,
    statusRealizada: req.body.statusRealizada
  })
  try {
    const tarefaSalva = await objetoTarefa.save();
    res.status(200).json(tarefaSalva)
  }
  catch (error) {
    res.status(400).json({ message: error.message })
  }
})
```

A rota acima vai tratar as requisições HTTP realizadas pelo método POST para o endpoint **"/api/post"**.

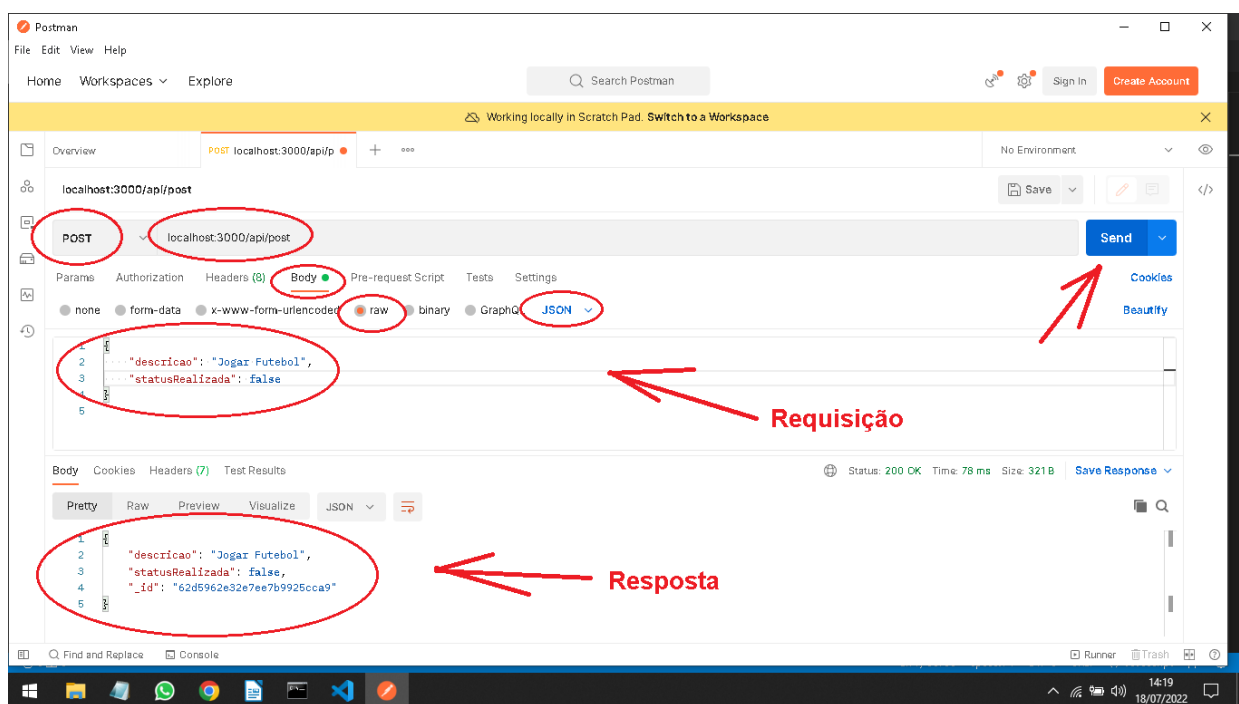
O código cria um objeto do tipo **modeloTarefa**, utilizando os parâmetros de **"descricao"** e **"statusRealizada"** passados através do body da requisição HTTP.

Em seguida o código solicita a gravação do objeto no Banco de Dados e retorna a resposta para o cliente.

Testando o endpoint /api/post

Para testar o endpoint, podemos realizar a requisição HTTP utilizando o programa **Postman**.

Transmita uma requisição utilizando o método POST para o endpoint "**localhost:3000/api/post**", passando um JSON com a "descricao" e "statusRealizada" da tarefa no corpo (body) da requisição. Ex:



Entre em sua conta no **MongoDB Atlas** para constatar que o dado foi corretamente gravado.



Criando a rota para /api/getAll

Insira a seguinte rota ao final de "**routes.js**":

```
router.get('/getAll', async (req, res) => {  
  try {  
    const resultados = await modeloTarefa.find();  
    res.json(resultados)  
  }  
  catch (error) {  
    res.status(500).json({ message: error.message })  
  }  
})
```

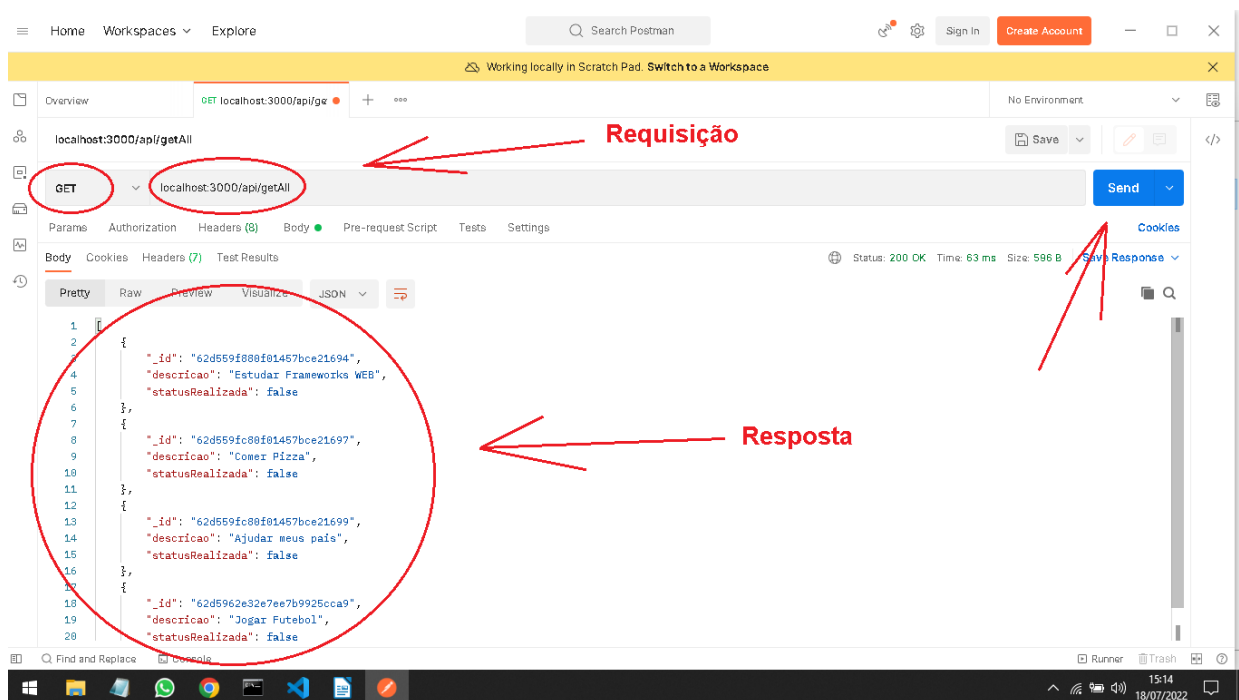
A rota acima vai tratar as requisições HTTP realizadas pelo método GET para o endpoint "/api/getAll".

O código acima utiliza o método "find" para ler todos os registros do Banco de Dados e retorna como resposta no formato de um json para o cliente.



Testando o endpoint /api/getAll

Utilizando o Postman, transmita uma requisição utilizando o método GET para o endpoint "localhost:3000/api/getAll". Ex:





Criando a rota para /api/delete/id

Insira a seguinte rota ao final de "**routes.js**":

```
router.delete('/delete/:id', async (req, res) => {
  try {
    const resultado = await modeloTarefa.findByIdAndDelete(req.params.id)
    res.json(resultado)
  }
  catch (error) {
    res.status(400).json({ message: error.message })
  }
})
```

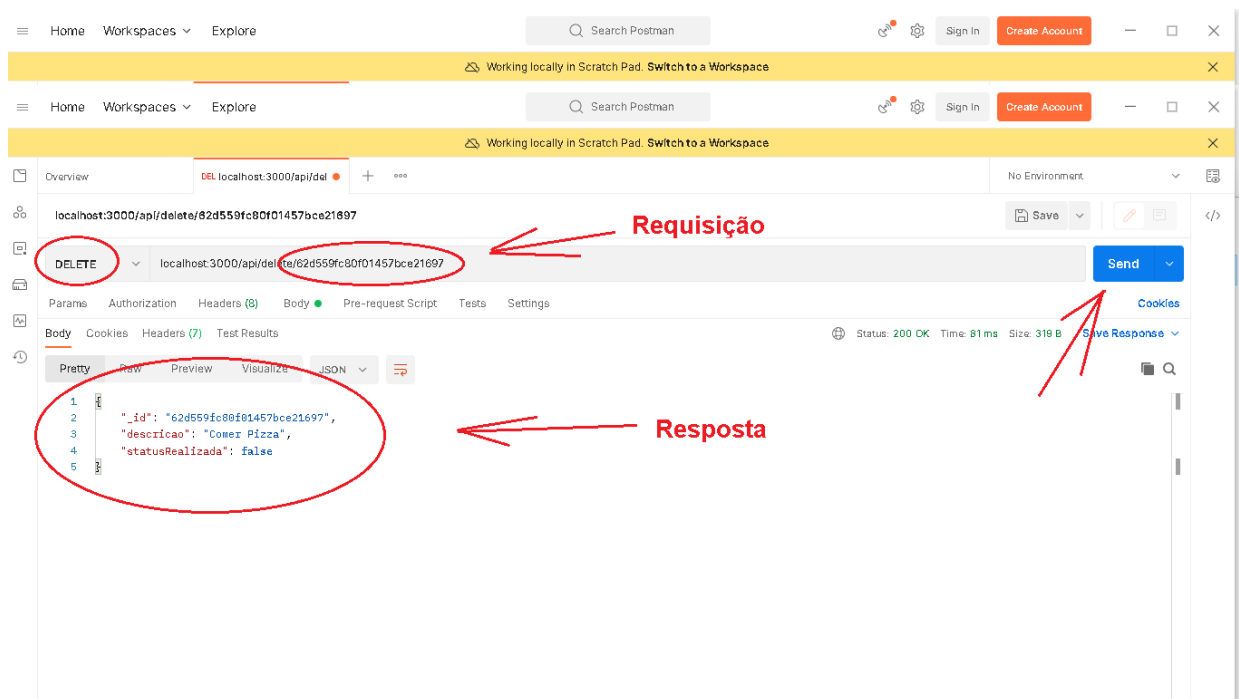
A rota acima vai tratar as requisições HTTP realizadas pelo método DELETE para o endpoint "/api/delete/id".

O código acima utiliza o método "findByIdAndDelete" para localizar e excluir o Document cujo "id" foi especificado como parte do endpoint.



Testando o endpoint `/api/delete/id`

Utilizando o Postman, transmita uma requisição utilizando o método DELETE para o endpoint "`localhost:3000/api/delete/id`", substituindo o "`id`" pelo código do "`id`" associado à tarefa a ser excluída no Bando de Dados. Ex:



Repare que o "`id`" utilizado na requisição foi o "`id`" da tarefa "Comer Pizza".

Após realizar a requisição, entre em sua conta no **MongoDB Atlas** para constatar que a tarefa foi corretamente eliminada.



Criando a rota para /api/update/id

Insira a seguinte rota ao final de "**routes.js**":

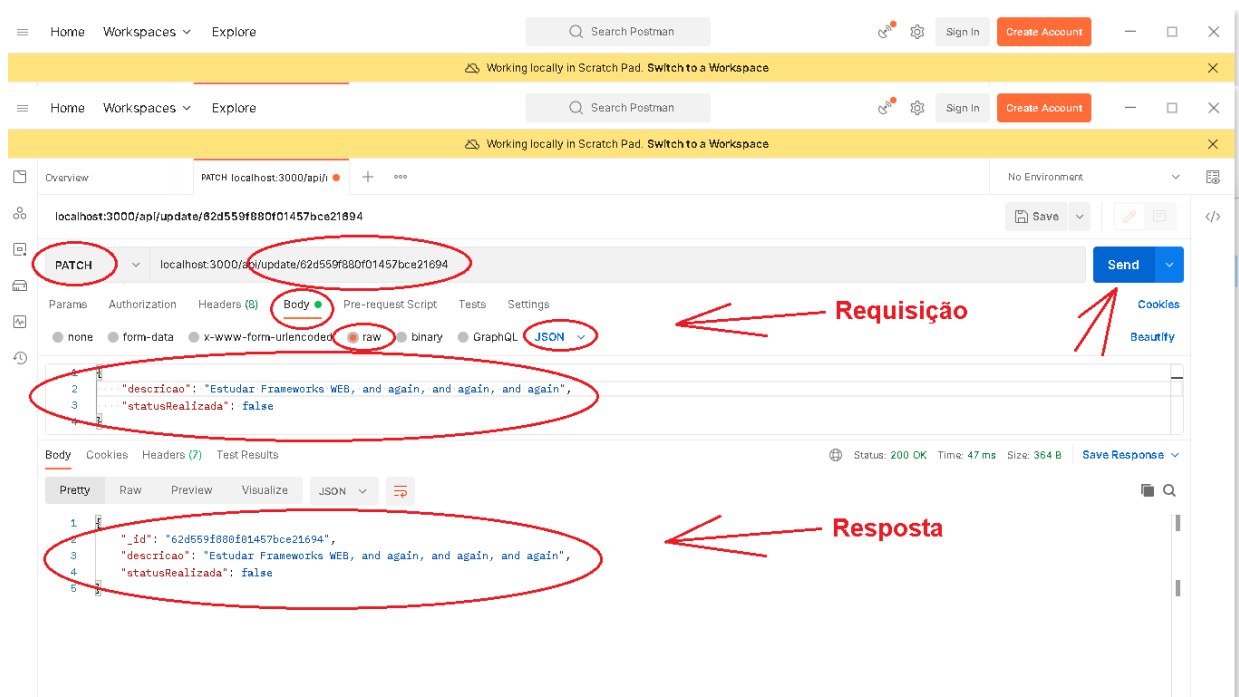
```
router.patch('/update/:id', async (req, res) => {
  try {
    const id = req.params.id;
    const novaTarefa = req.body;
    const options = { new: true };
    const result = await modeloTarefa.findByIdAndUpdate(
      id, novaTarefa, options
    )
    res.json(result)
  }
  catch (error) {
    res.status(400).json({ message: error.message })
  }
})
```

A rota acima vai tratar as requisições HTTP realizadas pelo método PATCH para o endpoint "/api/update/id".

O código acima utiliza o método "findByIdAndUpdate" para localizar e atualizar o Document cujo "id" foi especificado como parte do endpoint. Ele então atualiza o registro encontrado com a "novaTarefa" recebida como parâmetro no body da requisição.

Testando o endpoint /api/update/id

Utilizando o Postman, transmita uma requisição utilizando o método PATCH para o endpoint "`localhost:3000/api/update/id`", substituindo o "`id`" pelo código do "`id`" associado à tarefa a ser alterada no Bando de Dados e passando um JSON com a nova "`descricao`" e "`statusRealizada`" da tarefa no corpo (body) da requisição. Ex:



Repare que o "`id`" utilizado na requisição é o "`id`" da tarefa "Estudar Frameworks WEB".

Perceba que a resposta apresenta uma nova descrição para a tarefa identificada com o mesmo "`id`" anterior.

Após realizar a requisição, entre em sua conta no **MongoDB Atlas** para constatar que a tarefa foi corretamente alterada.



Histórico de revisões

Revisão: 00

Data: 18/07/2022

Descrição das alterações:
Documento original