

Roteiro Didático de Aula
Aula 03 – React Native
12/11/2020

Assuntos que serão abordados:

Consumo de API

Referências

Consumo de API

Primeiramente iremos criar um novo projeto do zero, sendo assim, criaremos um diretório em um local de nossa escolha e abriremos ela no Vs Code utilizando no CMD (sem necessidade de ser admin) o comando:

```
code .
```

Com o VsCode aberto, abra o terminal dele e execute:

```
yarn init -y
```

Com esse comando geramos um package.json com informações básicas do projeto, o `-y` é para aceitar tudo para enviar o pack.

Instalando o Express

Docs: <https://expressjs.com/pt-br/>

Com:

```
yarn add express
```

O Express é um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicativos web e móvel.

E podemos adicionar o cors também

```
yarn add cors
```

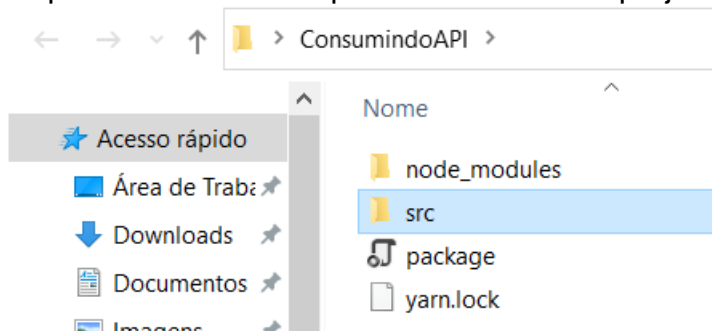
O CORS (Cross-origin Resource Sharing) é um mecanismo utilizado pelos navegadores para compartilhar recursos entre diferentes origens. O CORS é uma especificação do W3C e faz uso de headers do HTTP para informar aos navegadores se determinado recurso pode ser ou não acessado.

Talvez seja necessário para seu pleno funcionamento adicionar os tipos:

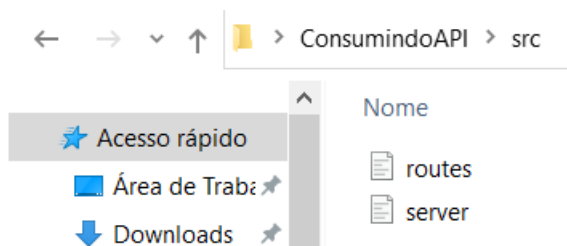
```
yarn add @types/cors
```

Talvez seja necessário reiniciar o Vs Code para ele computar os tipos, não é uma regra mas é uma boa pratica para evitar dores de cabeça futuras

Depois disso crie uma pasta *src* na raiz do projeto



Crie dois arquivos, um com o nome de *server.ts* e o outro com o nome de *routes.ts* dentro da pasta *src*



Então dentro do *server.ts*:

```
import express from 'express';

import routes from './routes'; //importar as rotas para **caso tenha erro com ela, pode
deixa-la comentada e depois, antes de rodar o get, venha e descomente**

import cors from 'cors'; //importar o cors para nossa aplicação ser visível por outros
fronts-end

const app = express(); //transformar o express() em uma variável manipulvel
app.use(cors()); //falar para express usar o cors
app.use(express.json()); //falar para express usar json
app.use(routes); //caso de erro n início, comente tbm //falar para express usar as ro
tas
```

```
app.listen(3333); //falar para express usar ouvir a porta 3333, podemos usar qualquer porta
//localhost:3333
```

Como o Node não entende tudo o que está nesse código, assim como o próprio typescript, vamos corrigir isso, instalando o typescript na aplicação, com o comando:

```
yarn add typescript
```

E então depois, executar o comando:

```
yarn tsc --init
```

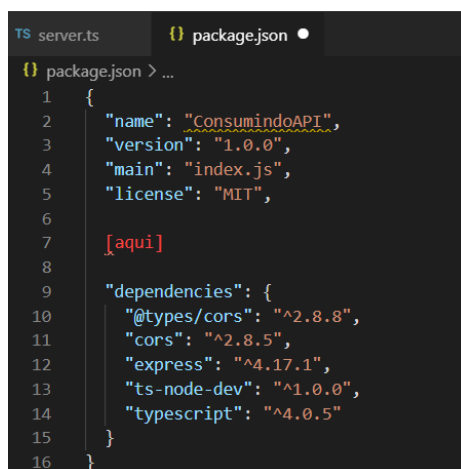
Que irá criar um arquivo chamado *tsconfig.json* que são as configs dele, abrir e mudar o “target” para “es2017” para aumentar a versão na hora da conversão, salvar e fechar.

Então instalar um pacote com o comando:

```
yarn add ts-node-dev
```

É um pacote para executar nosso projeto usando tsc e node.

Agora o arquivo *package.json*, abra e edite após a license e antes das *Dependencies* ou *devDependencies*



```
{} package.json > ...
1 {
2   "name": "ConsumindoAPI",
3   "version": "1.0.0",
4   "main": "index.js",
5   "license": "MIT",
6
7   [aqui]
8
9   "dependencies": {
10    "@types/cors": "^2.8.8",
11    "cors": "^2.8.5",
12    "express": "^4.17.1",
13    "ts-node-dev": "^1.0.0",
14    "typescript": "^4.0.5"
15  }
16 }
```

o código a ser inserido é esse:

```
"scripts": {
  "start": "tsnd --transpile-only --ignore-watch node_modules --respawn src/server.ts"
},
// transpile-only apenas transpila e não ve erros
// ignore-watch node_modules não olha a pasta node modules
// respawn reinicia o servidor
```

Isso vai fazer a aplicação iniciar com o comando

Vamos testar!

Caso aconteça esse erro:

```
13  
14   app.listen(3334);  
15   //localhost:3334  
16
```

Error: listen EADDRINUSE: address already in use :::3333 você deve mudar a porta para 3334 no arquivo server.ts

Caso não aconteça erro algum, vamos ver o que aconteceu acessando pelo nosso navegador a pagina <http://localhost:3333/> ou caso tenha dado o erro acima <http://localhost:3334/>

E se tudo deu certo... deve ter dado errado... aparecerá o erro:

Cannot GET /

Mas já era um erro esperado pois não criamos rotas ainda. Mas está funcionando... já pode pegar um café e chorar discretamente.

Vamos então fazer uma simples alteração no arquivo server.ts

```
app.get('/', async (request, response)=>{  
  return response.send("Hello World")  
});
```

Vamos salvar, recarregar e ver isso no navegador:

Hello World

Isso foi uma rota que nós criamos de maneira simples de forma didática, porem não utilizamos usualmente as rotas diretamente no arquivo server pois no final teremos uma grande bagunça... iremos então dividir as rotas em outro arquivo, mas ainda falando sobre as rotas, temos outros métodos além desse que utilizamos:

GET: Buscar ou listar informações (acabamos de utilizar)

POST: Criar alguma nova Informação

PUT: Atualizar uma informação existente

DELETE: Deletar uma informação

temos também uma estrutura básica que se segue:

//Corpo (request body): dados para criação, ou atualização de um registro

//Routes Params: identificar qual recurso att ou delet

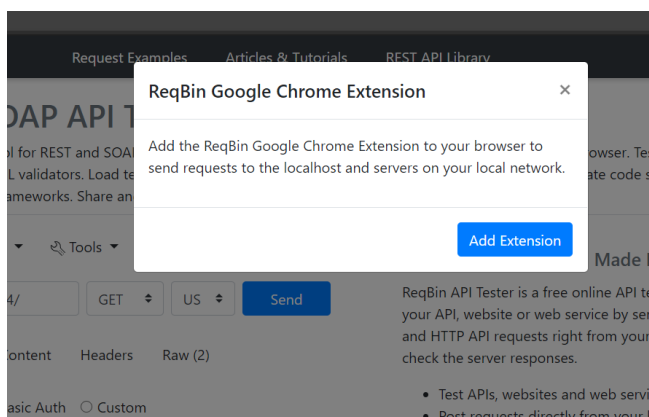
//Query Params: Paginação, filtro, ordenação

Mas... como nem tudo são flores... os navegadores só conseguem resolver o método get, então para os outros, vamos usar o site:

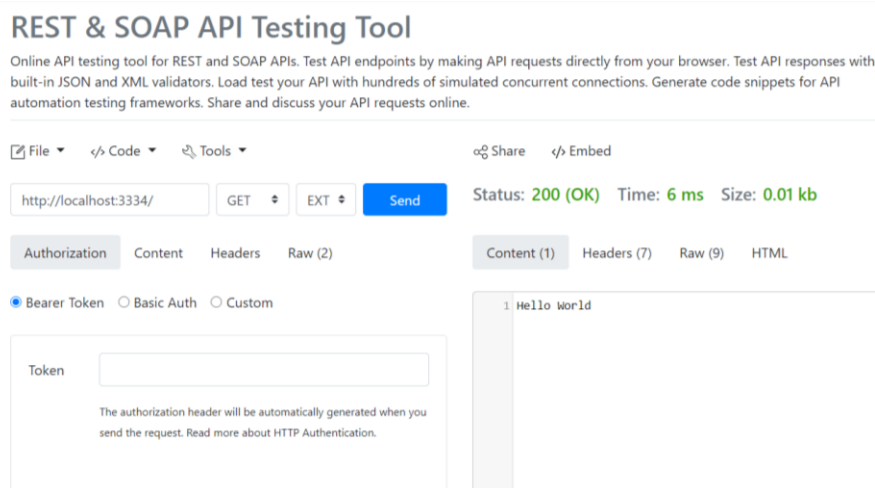
<https://reqbin.com/>

o reqbin é um substituto do postman

ao tentarmos utilizar o localhost ele irá pedir para instalar uma extensão



após instalar a extensão e atualizar a pagina do site, já será possível fazer as requisições



Pare o servidor com o `ctrl + c`, irá aparecer:

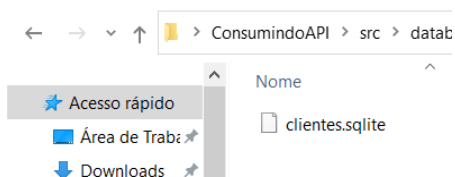
Deseja finalizar o arquivo em lotes (S/N)?

Digite S e pressione enter

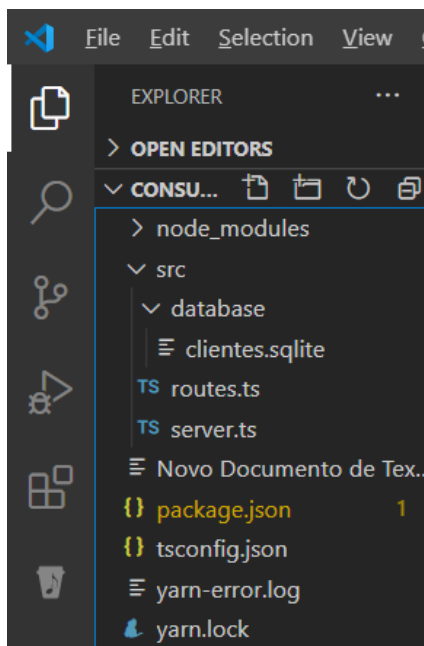
Agora chegou a hora de utilizarmos um banco de dados, então vamos instalar o sqlite, com o comando:

```
yarn add sqlite3
```

Agora dentro da pasta `src`, vamos criar uma chamada `database` onde vamos colocar o db de clientes (vou deixar no classroom esse arquivo também:



Temos que ter essa imagem agora:



Com isso feito, vamos criar um arquivo, dentro da pasta `database` com o nome de `connection.ts` e escrever esse código:

```
const sqlite3 = require('sqlite3').verbose();// aqui estamos pegando essa função que chama o BDe jogando em uma variável

const db = new sqlite3.Database("./src/database/clientes.sqlite", sqlite3.OPEN_READWRITE ,
(err: { message: any; }) => {
  //aqui estamos conectando com o BD
  if (err) { //tratativa de erro
    console.error(err.message);
  }
  console.log('Connected to database. ');
  console.log(db)
});

export default db;//exportando o DB
```

Com isso feito, vamos criar nossas rotas, vamos mover esse get que esta em server.ts para o nosso arquivo de rotas routes.ts e escrever um pouco de SQL

```
import express from 'express';
import db from './database/connection';//chamado o db do connection.ts

const routes = express.Router();//transformando em uma variável manipulável

routes.get('/clientes', async (request, response)=>{ //transformamos nossa função em async

  const dataBase = await db;//pq a requisição para o db precisa ser await, para nossa aplicação poder esperar acontecer, e jogamos em uma variável tbm

  dataBase.serialize(() => {
    dataBase.each('SELECT * FROM clientes', function(err: { message: any; }, row: any){
//aqui que estamos fazendo a busca no db e mostrando, no console
      if (err) { //tratativa de erros
        console.error(err.message);
      }
      console.log(row)//vai retornar tudo que temos no banco de dados
    });
    return response.send(200) //mostrar uma mensagem de 'ok'
  });
});

export default routes;
```

E como vemos isso funcionar?

Vamos fazer o servidor rodar de novo, com o comando
`yarn start`

MAS ANTES, TEMOS QUE DESCOMENTAR AS PARTES COM ROTAS EM `SERVER.TS`

```
server.ts • TS connection.ts TS routes.ts () package.json
src > TS server.ts > ...
1 import express from 'express';
2
3 import routes from './routes'; //importar as rotas para **caso tenha erro com ela, pode deixa-la comentada e depois, antes de rodar o get, venha
4
5 import cors from 'cors'; //importar o cors para nossa aplicação ser visível por outros fronts-end
6
7 const app = express(); //transformar o express() em uma variável manipulável
8 app.use(cors()); //falar para express usar o cors
9 app.use(express.json()); //falar para express usar json
10 app.use(routes); //caso de erro n início, comente tbm //falar para express usar as rotas
11
12
13 app.listen(3334); //falar para express usar ouvir a porta 3333, podemos usar qualquer porta
14 //localhost:3334
15
16 //app.get('/', async (request, response)=>{
17 //   return response.send("Hello World")
18 // });
19
```

Não esqueça de salvar!!

Acesse o site de requisições e coloque o endereço:

<http://localhost:3333/clientes>

Vamos ter essa imagem:

Request Examples Articles & Tutorials REST API Library Curl Online

http://localhost:3333 GET EXT Send Status: 200 (OK) Time: 20 ms Size: 0.00 kb

Authorization Content Headers Raw (2) Content (1) Headers (7) Raw (9)

☒ Bearer Token ☐ Basic Auth ☐ Custom

Token

The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication.

1 OK

Com isso vamos saber que tudo funcionou!!

Agora vamos fazer e testar um post, mas antes disso precisamos adicionar um código no nosso arquivo de rotas routes.ts

```
routes.post('/novocliente', async (request, response)=>{
  //uma nova rota

  const dataBase = await db;

  dataBase.run(`INSERT INTO clientes (name, telefone, cep, n_casa) VALUES(?,?,?,?)`, [
    `${request.body.name}`, `${request.body.telefone}`, `${request.body.cep}`, `${request.body.n_casa}`], function(err: { message: any; }, row: any){
    // nessa parte estamos escrevendo sqlite e usando concatenação de strings para podermos
    // jogar as variaveis do corpo no lugar as interrogações no comando SQLite
    if (err) { //tratativa de erro
      console.error(err.message);
    }
    console.log(row);
    return response.send(200) //resposta ok
  });
});
```

Agora vamos testar, no site, troque o `/clientes` para `/novocliente`, mudar de get para post e clicar em content, e preencha um json com as informacoes, como no exemplo:

```
{
  "name": "Caique",
  "telefone": 11000000000,
  "cep": 96325874,
  "n_casa": 200
}
```

Lembre-se de ser exatamente assim, mas trocando o nome pelo o seu!!

Agora clique em send, vamos ter esse resultado, com um ok novamnte:

http://localhost:33: POST EXT Send Status: 200 (OK) Time: 73 ms Size: 0.00 kb

Authorization Content (6) Headers Raw (11) Content (1) Headers (8) Raw (10)

JSON (application/json)

```
1 {
2   "name": "Caique",
3   "telefone": 1100000000,
4   "cep": 96325874,
5   "n_casa": 200
6 }
```

```
1 OK
```

Agora para vermos vamos dar outro get e olhar o console:

```
Database {}
undefined
express deprecated res.send(status): Use res.sendStatus(status) instead src\routes.ts:29:25
express deprecated res.send(status): Use res.sendStatus(status) instead src\routes.ts:19:25
{
  id: 11,
  name: 'vivian',
  telefone: 113698524,
  cep: 3694910,
  n_casa: 789
}
{
  id: 12,
  name: 'Caique',
  telefone: 1100000000,
  cep: 96325874,
  n_casa: 200
}
{
  id: 13,
  name: 'Caique',
  telefone: 1100000000,
  cep: 96325874,
  n_casa: 200
}
```

Com isso vemos que tudo correu normalmente!!

Referências

Rocketseat. (s.d.). *Expo*. Fonte: <https://www.youtube.com/watch?v=ZaDpDIPr25M>

Yarn. (s.d.). Fonte: <https://yarnpkg.com/getting-started/install>