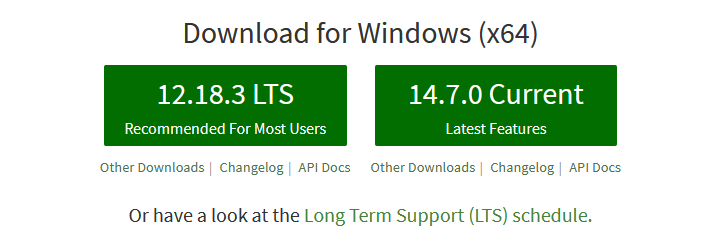
# Node e NPM (Contextualização)

O primeiro passo para podermos utilizar a Omnistack (Node.js, ReactJS e React Native) é instalar o Node.js, que vem acompanhado do NPM. Para visualizar o site do Node.js e suas versões:

**https://nodejs.org/en/**

Como você já deve ter visto, na página principal do Node são apresentadas duas versões: LTS e Current.

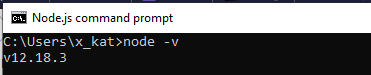


A primeira se refere à versão do Node que possui Long Term Support (LTS), são as mais confiáveis e é a que recomendamos utilizar na NLW. Já a segunda se refere à versão do Node mais atual e experimental, o que não é recomendada para desenvolvimento ainda.

Escolhida a versão LTS do Node, precisamos decidir o método de instalação. É importante ressaltar que apesar de na tela inicial do Node.js eles recomendarem a forma de instalação direta (famosa janela que só clicamos no Next), iremos utilizar nesse guia os gerenciadores de pacote (exceto Linux).

Não só pelo fato de facilitar possíveis desinstalações e atualizações do Node, mas também por serem muito úteis para trabalhar com diversos outros pacotes.

Pronto dev, agora que já sabemos que iremos instalar a versão LTS do Node.js utilizando um gerenciador de pacote, bora para o passo-a-passo de cada sistema operacional.



**WINDOWS**

Para o Windows utilizaremos o gerenciador de pacotes [Chocolatey] (https://chocolatey.org/), porém antes dos passos de instalação vamos falar brevemente sobre qual shell você deve usar.

* CMD: também conhecido como Command Prompt, ele é um dos shells mais antigos da atualidade (foi construído para ser compatível com o MS-DOS) e, apesar da sua fama, hoje em dia tem sido cada vez menos utilizado.
* Powershell: novo shell apresentado pela Microsoft por volta de 2005, ele apresenta diversas melhorias em relação ao CMD, tornando-o popular atualmente e consequemente a nossa escolha para a NLW#02.

Escolhido o shell, vamos começar a instalação:

Busque no campo de busca do Windows por Windows Powershell, clique com o botão direito em cima do programa e escolha a opção Executar como administrador. O Powershell trabalha com um esquema de autorizações (conhecido como `Execution Policy`) para execução de scripts e, por isso, precisamos verificar se o presente no sistema está compatível com o que o Chocolatey precisa. Execute o seguinte comando:

Get-ExecutionPolicy

Caso ele retorne `Restricted`, execute o comando:

Set-ExecutionPolicy RemoteSigned

E escolha a opção `[A] Sim para Todos. Caso o comando acima apresente erro, tente usar:

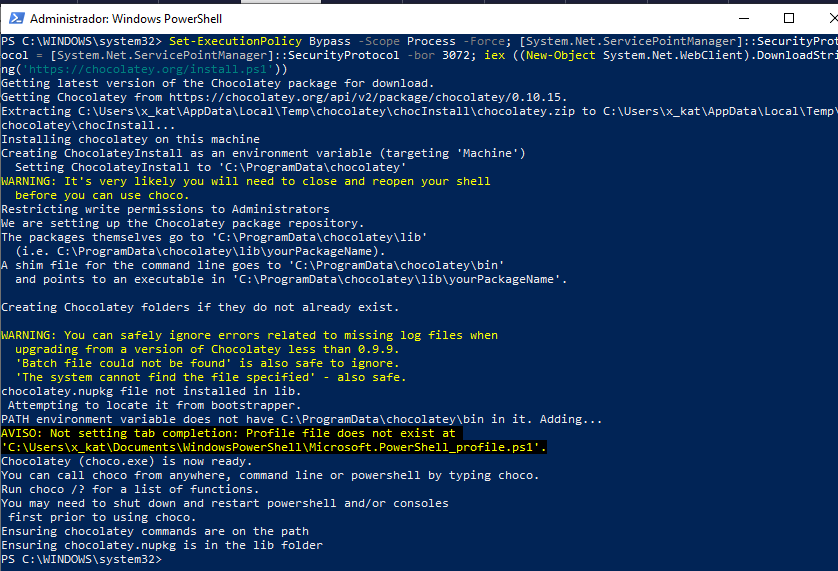
`Set-ExecutionPolicy Bypass -Scope Process`

Verifique se alteração de permissão ocorreu com sucesso executando novamente o comando:

Get-ExecutionPolicy

Alterada a permissão, basta instalar o Chocolatey com o comando:

Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))



Após o fim da instalação, feche e abra o Powershell como administrador novamente e execute:

**choco –v**



Caso ele retorne a versão do Chocolatey, a instalação foi um sucesso. Para finalizar, basta instalar a versão LTS mais recente do Node com o seguinte comando:

**cinst nodejs-lts**

E escolha a opção `[A]ll - yes to all`. Após o fim da instalação, feche e abra o Powershell como administrador novamente e execute:

**node –v**



Caso retorne as versões do Node e npm, sua instalação foi um sucesso.

**YARN 1**

Para instalar o Yarn 1 no Windows siga os seguintes passos, execute o comando no Powershell (como admin):

**cinst yarn**

E escolha a opção `[A]ll - yes to all`. Feche e abra o terminal novamente, em seguida rode o comando:

**yarn --version**

****

Caso retorne a versão do Yarn (acima de 1.0, abaixo de 2.0), a instalação ocorreu com sucesso.

**EXPO**

Para instalar o Expo é bem simples e o passo é o mesmo nos 3 sistemas operacionais. Com o Node e Yarn instalados, abra o terminal (no Windows, sem ser como admin) e execute:

**yarn global add expo-cli**

Caso você prefira utilizar o npm, basta executar:

**npm install expo-cli --global**

Para verificar se a instalação ocorreu com sucesso, execute:

**expo –version**

**VISUAL STUDIO CODE**

Para instalar o editor de texto Visual Studio Code em qualquer um dos 3 sistemas operacionais, basta [acessar o site](https://code.visualstudio.com/), baixar e rodar o executável. Com a instalação finalizada, abra o programa.

Para finalizar, vamos adicionar algumas configurações no Visual Studio Code. Para isso, basta pressionar `Ctrl + Shift + P` e escolher a opção `Open Settings (JSON)`. Na janela que foi aberta, adicione as configurações abaixo:

É preciso tomar alguns cuidados ao realizar essas alterações. Verifique se a configuração adicionada já não existe no arquivo. Se sim, apenas atualize o valor. Verifique também se a todas as linhas de configuração, exceto a última, terminam com vírgula, para não gerar erro. Por fim, caso queira substituir completamente a sua configuração pela abaixo, envolva com chaves `{}` todo o código disponibilizado.

```json

"workbench.iconTheme": "material-icon-theme",

"workbench.startupEditor": "newUntitledFile",

"explorer.compactFolders": false,

"editor.renderLineHighlight": "gutter",

"workbench.editor.labelFormat": "short",

"extensions.ignoreRecommendations": true,

"javascript.updateImportsOnFileMove.enabled": "never",

"typescript.updateImportsOnFileMove.enabled": "never",

"breadcrumbs.enabled": true,

"editor.parameterHints.enabled": false,

"explorer.confirmDragAndDrop": false,

"explorer.confirmDelete": false,

"emmet.syntaxProfiles": { "javascript": "jsx" },

"emmet.includeLanguages": { "javascript": "javascriptreact" },

"javascript.suggest.autoImports": true,

"typescript.suggest.autoImports": true,

"workbench.colorTheme": "Omni"

```

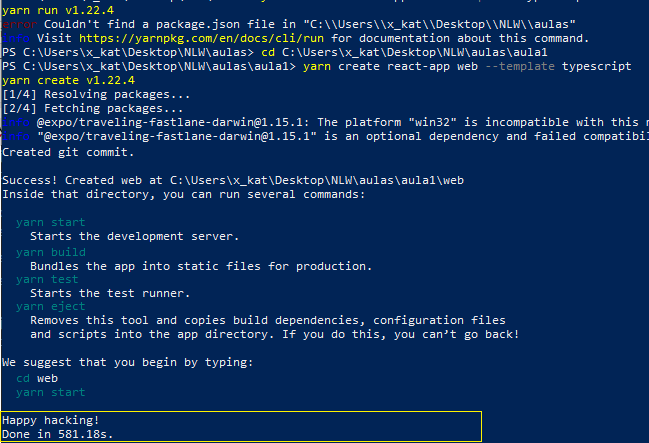
**CRIAR PROJETO**

Duas maneiras:

**yarn create react-app web --template typescript**

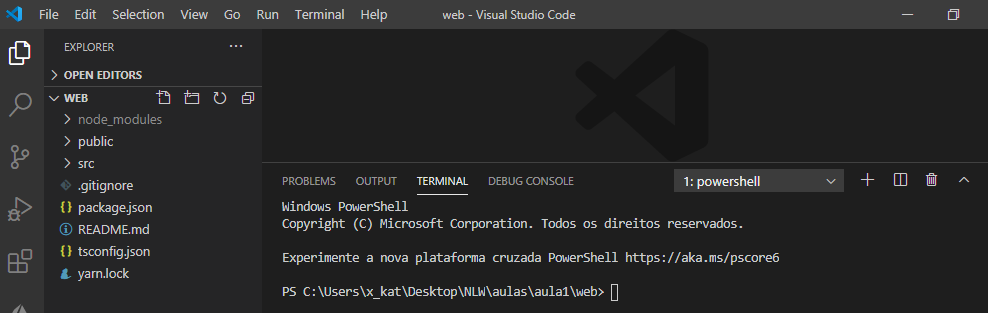
OU

**npx create-react-app web --template typescript**



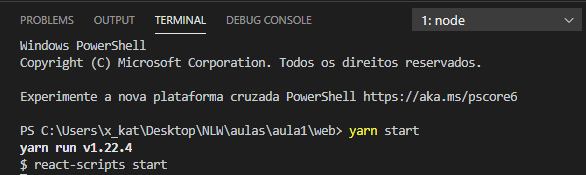
Esse comando, na minha máquina, durou igualmente 10 min... Na demonstração demorava 22s. Dependendo da máquina e velocidade de download dos pacotes.

Abra o VS Code e habilite o Toggle Terminal (Crtl + ‘ )

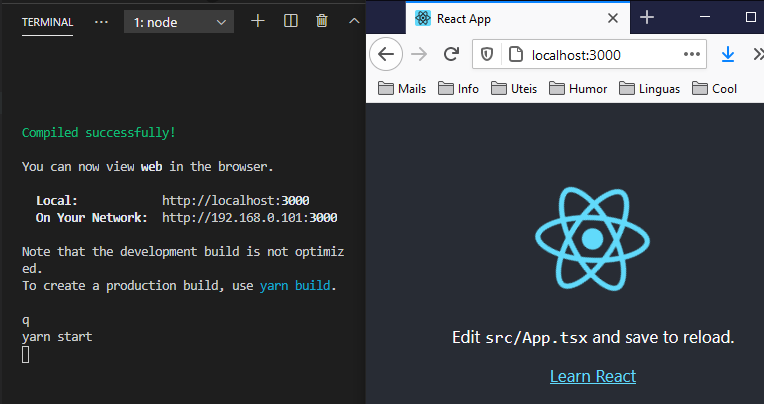


Inicie o yarn com

**yarn start**

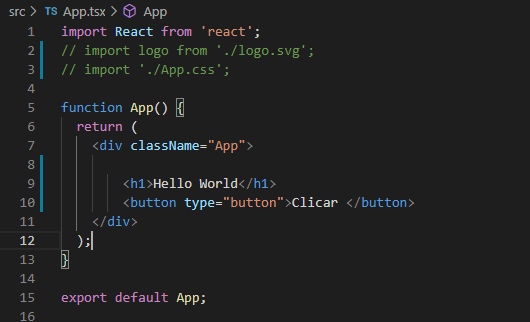


E já inicia o aplicativo

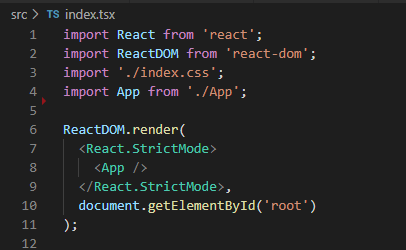


Nos arquivos extensão tsx (Typescript com XML) coloca-se as funções que irá renderizar os componentes. A sintaxe utilizada é chamada de JSX (Javascript com XML) e é neste arquivo onde ficará os componentes que poderá ser utilizado em várias partes do código. Note que o nome da função deve estar em maiúscula. E de que sempre devemos importar a biblioteca:

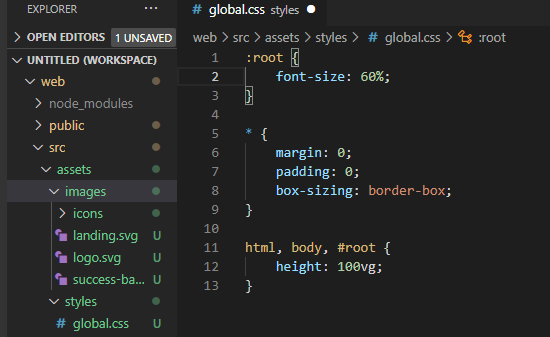
**import React from 'react';**



Na index.tsx tem as chamadas das funções. Note que se o nome da função fosse em minúscula, o React entenderia como se fosse tag HTML.

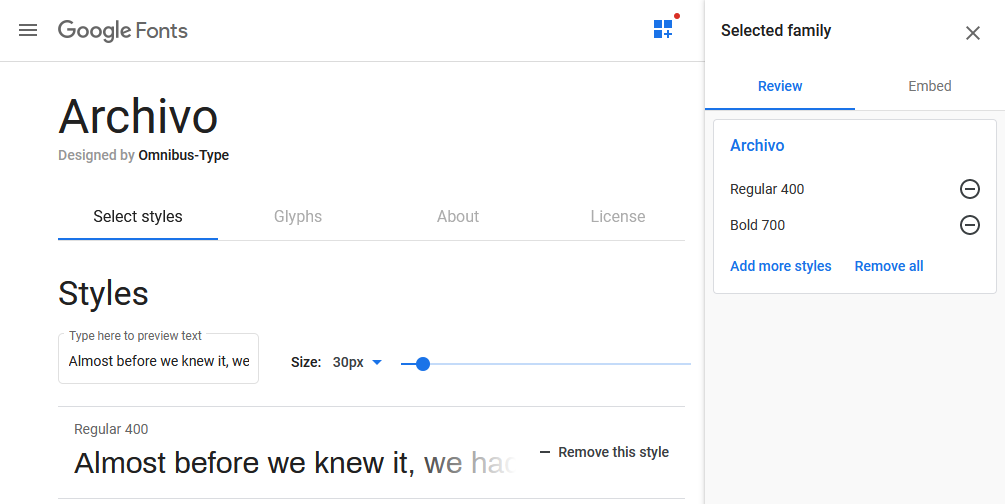


Crie uma pasta assets dentro de src e coloque a pasta imagens fornecida pelo treinamento. Crie também uma pasta style e um arquivo global.css. O height 100vh corresponde a sempre 100% da tela.



Para associar fontes ao projeto só precisa ir no site do Google Fonts:

**https://fonts.google.com/**

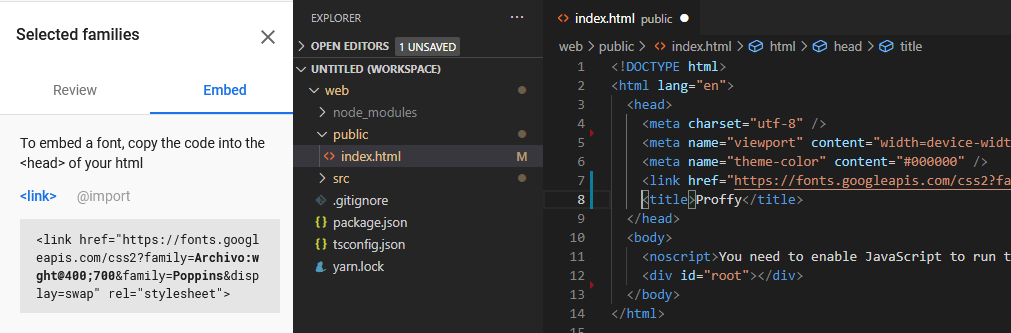


Depois de selecionar as fontes abaixo, clicamos em Embedd e copiamos o link e colamos em public/indez.html:

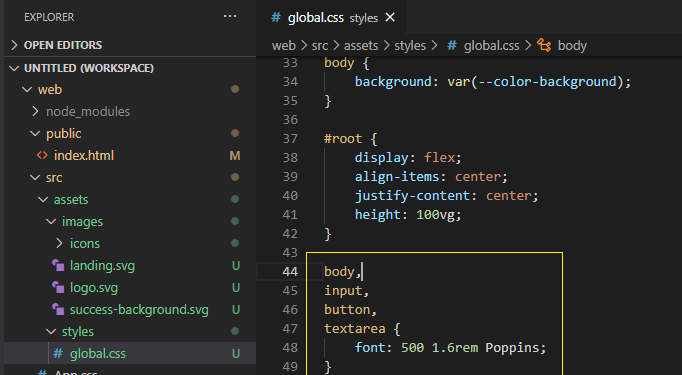
**archivo Regular 400**

**archivo Bold 700**

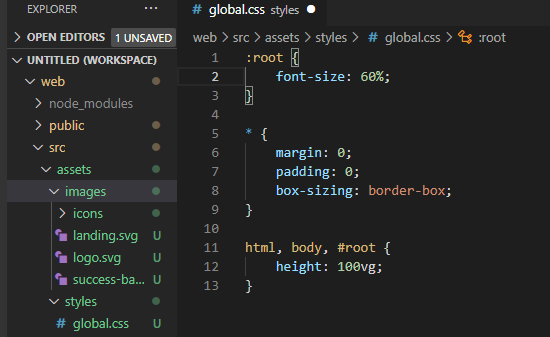
**poppins Regular 400**

****

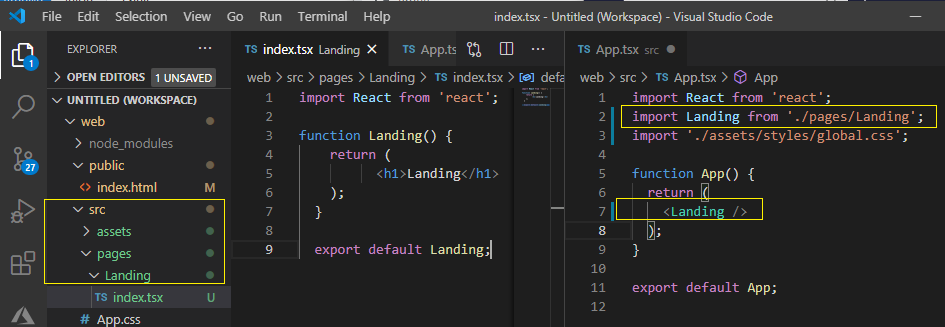
Depois de incluirmos as fontes podemos definir o tamanho delas responsivo com o unidade rem:

****

O 1.6 significa que a fonte desses elementos será 60% maior do que o tamanho padrão da fonte (16px) para dar mais destaque. Esse ajuste é criado, pois como no início do style havia sido definido font-size de 60% (16 x 0.60 = 9.6px). E o rem é utilizado para caso futuramente precisasse aumentar a fonte, só seria necessário alterar o valor do root.

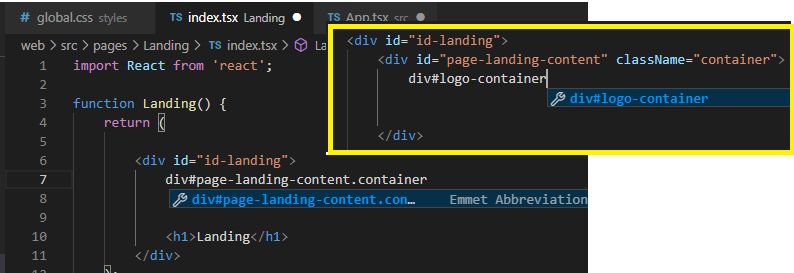


Próximo passo, criamos uma página Landing, e a importamos na página App.tsx, substituindo o html estático com Hello Word e o botão que havia.



Repare que a referência de arquivos dentro da aplicação deve estar sempre precedido por ./ ou ../ senão o React confunde com pacotes e não consegue encontrar os arquivos.

É possível acelerar a codificação de páginas agora com a sintaxe abreviada Emmet que utiliza com padrões de sintaxe do css para gera o id, class, e outros atributos como uma engenharia reversa:



Essa habilidade foi ativada no início da aula no arquivo settings.json com os parâmetros:

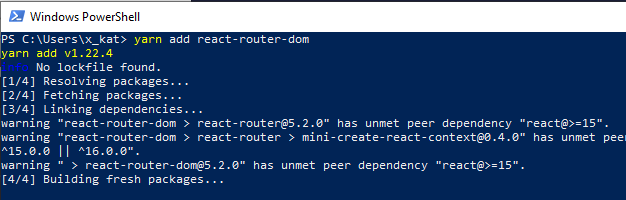
**"emmet.syntaxProfiles": { "javascript": "jsx" },**

**"emmet.includeLanguages": { "javascript": "javascriptreact" },**

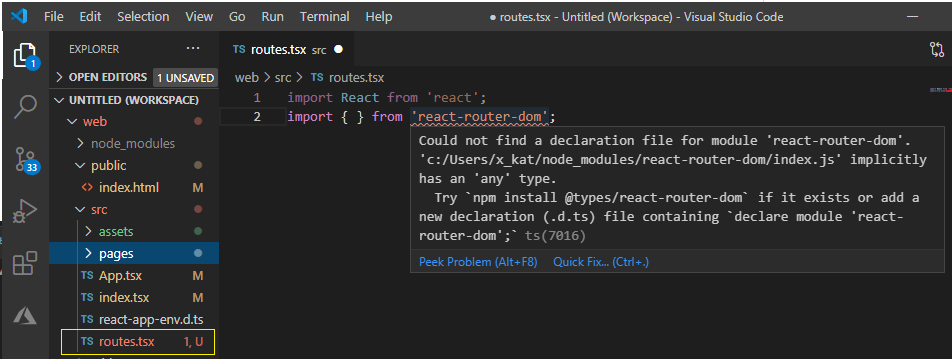
Repare que as classes são definidas pelo atributo classname e não mais somente com o class. Isso por que o class é palavra reservada no React.

Agora para navegar entre páginas temos que instalar:

**yarn add react-router-dom**

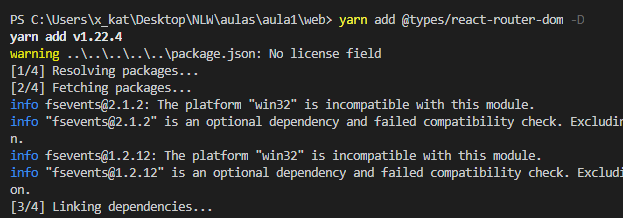


Incluímos outro arquivo routes.tsx para configurar as rotas e na hora de importar emite um aviso para instalar outro pacote,

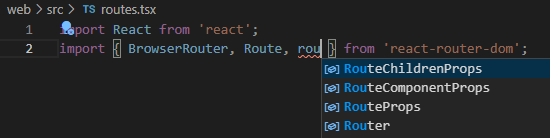


Colocamos o parâmetro –D para indicar que é uma dependência de desenvolvimento e não vai ser utilizada na produção

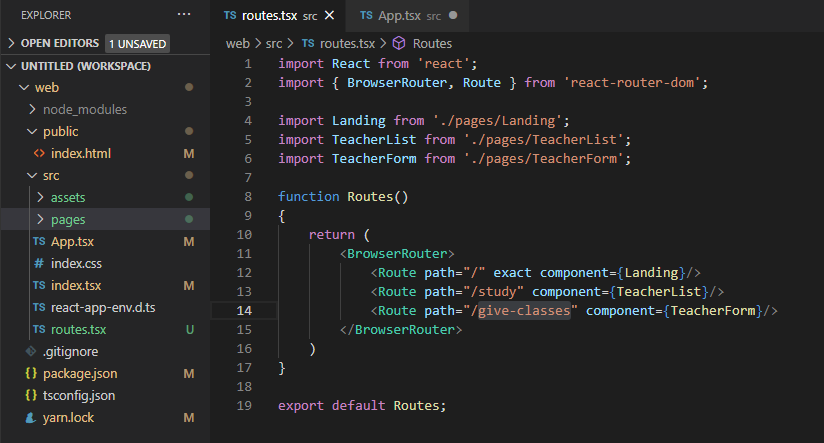
**yarn add @types/react-router-dom –D** OU **npm install @types/react-router-dom -D**



Depoi de instalarmos esse pacote aparece o Inteligesense quando digitamos:

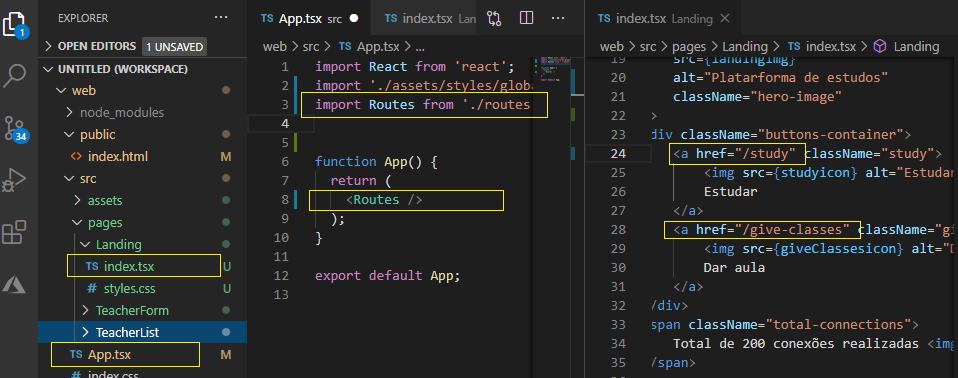
****

O código da função Routes ficará assim:

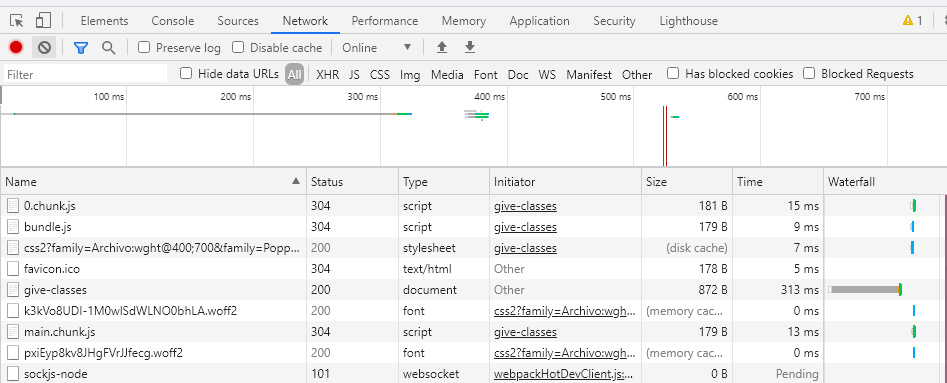


Repare na primeira rota que navega para raiz tem uma propriedade a mais exact. Isso é para diferenciar das outras rotas que também contém a barra para frente, então a página inicial apareceria em todas as outras páginas.

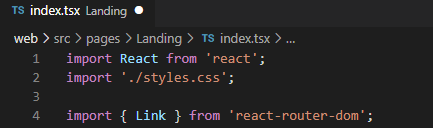
Se alterarmos agora a página inicial App.tsx e a página Landing.tsx as rotas já estarão funcionando.



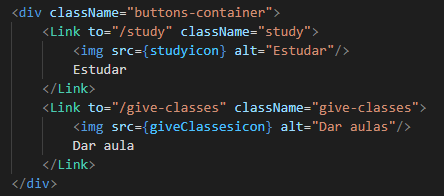
Porém de formos em Network do browser podemos ver que a cada navegação está havendo o Refresh e carregamento da página inteira (css, scripts) e não queremos isso.



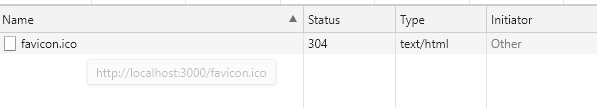
Para resolver isso temos que importar de react-router-dom o componente Link e substitui-lo pelas anchor tags do HTML.



E invés de href fica to:



Ao salvar e testar não está recarregando mais nada ao mudar de página. Esse é o conceito de SPA(Single Page Application). Não tem que ficar recarregando recursos compartilhados da mesma aplicação.

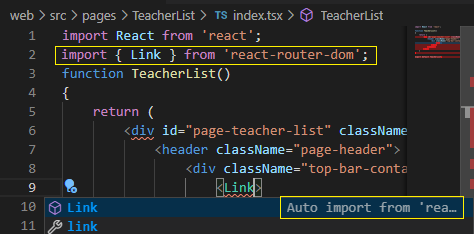


Um recurso interessante que foi ativado no settings.json no começo dessa aula com os parâmetros:

**"javascript.suggest.autoImports": true,**

**"typescript.suggest.autoImports": true,**

Isso ativa a possibilidade de importar as referência de outros objetos enquanto estamos digitando:



Para criar um componente é o mesmo processo de criar as páginas:

* Cria-se um folder com o Nome do componente
* Cria-se um arquivo index.tsx
* Cria-se um arquivo de estilo local style.css
* Faz as importações de import React from 'react'; import './styles.css';

Uma das diferenças é a criação personalizada de atributos que é feita adicionando uma interface, onde define-se o nome e tipo do atributo, como a seguir:

interface PageHeaderProps

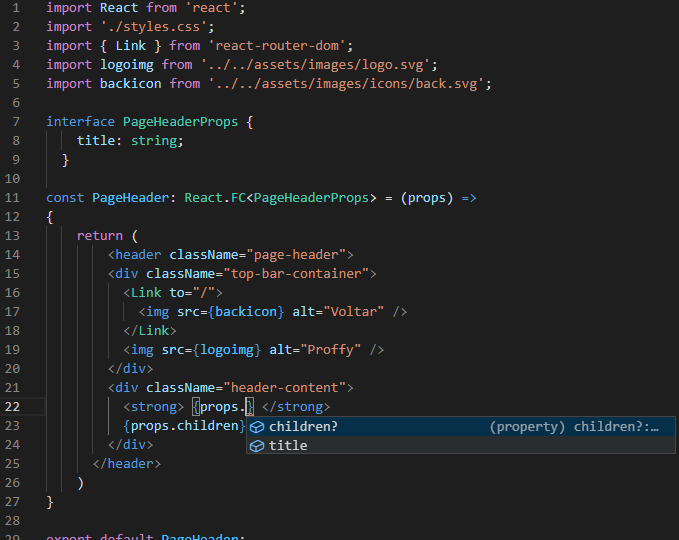
{

title: string;

}

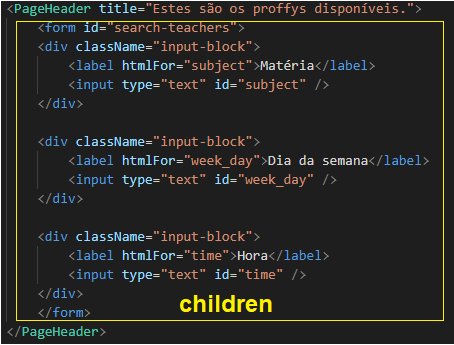
Se fosse permitido nulos no parâmetro title teria que colocar um sinal de interrogação após o nome (title?).

Além disso, é necessário transformar a duas função em um objeto constante ficando como abaixo:



O React.FC é uma abreviação de React.FuncionalComponent. As duas formas são aceitas, quando você passa a interface PageHeaderProps está se declarando as propriedades que esse componente pode aceitar. Sempre quando se deseja inserir um objeto Javascript dentro do HTML coloca-se em chaves {props} ou {logimg}. O props refere-se a construção lambda, essa denominação é definida pelo programador.

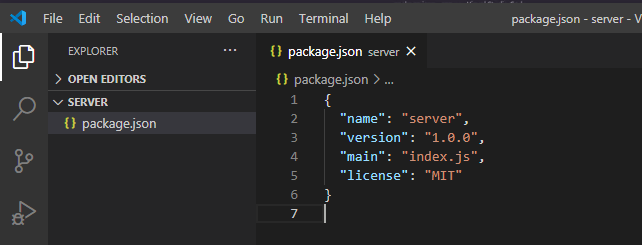
O props.children refere-se a um propriedade que todos os componentes possuem. E significa que tudo que você colocar dentro de um componente será considerado como “children” e será renderizado se você chamar props.children



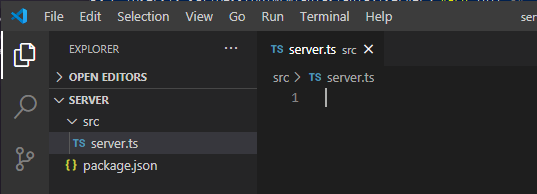
Iremos agora criar uma Web API REST. Crie uma pasta server, abra o PowerShell nela e digite:

**yarn init -y**

Após isso abra o VS Code e foi criado a dependências da aplicação (packages.json)

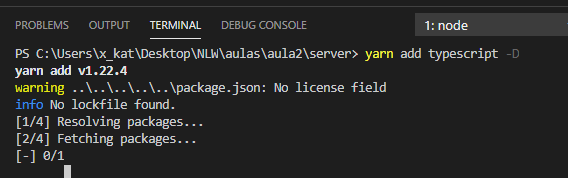


Crie um pasta src e um arquivo server.ts



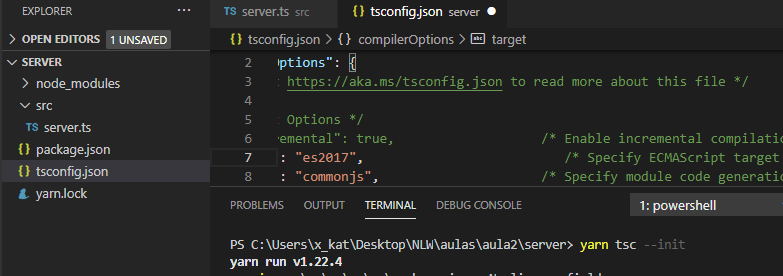
Abra um terminal no VS Code e instale a biblioteca typescript com o comando:

**yarn add typescript –D**



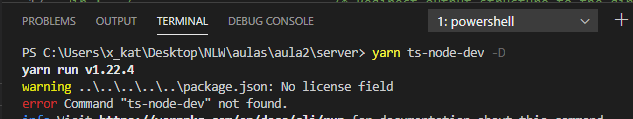
Gere o arquivo de configuração do Typescript com o comando:

**yarn tsc –init**



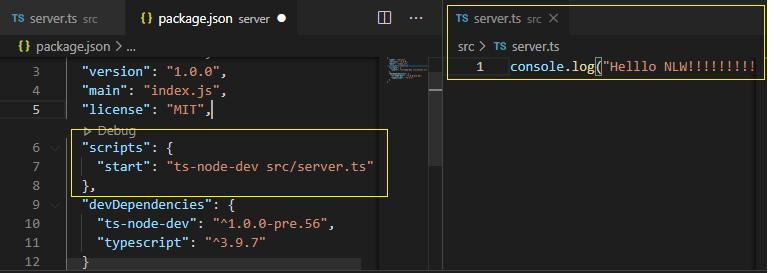
Definimos a versão do ECMAScript para es2017 pois é a versão que o Node.js entende. Se fosse uma versão de browser como o IE teria que ser uma versão mais antiga. Instalamos a também a dependência:

**yarn add ts-node-dev -D**



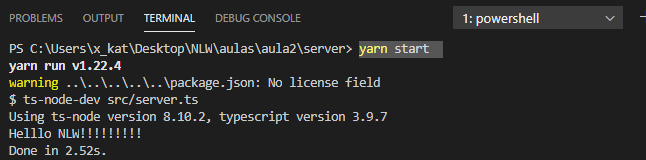
Que executa o servidor Node, fazendo ele entender Typescript, e monitora se teve alteração no Script. Se houver, ele restart automaticamente o servidor. Por padrão, sem a extensão, teríamos que parar e reiniciar o Node manualmente se houvesse alterações.

Para testar o serviço, altere o arquivo packages.json e crie uma seção scripts e no arquivo server.ts uma mensagem console.log genérica:



Observe que esta configuração "start": "ts-node-dev src/server.ts" poderia ser abreviada como "start": "tsnd src/server.ts". Para executar no terminai digite:

**yarn start**



No script do iremos inserir algumas flags e ficará assim:

**"scripts": {**

**"dev": "tsnd -**

**--transpile-only** >> Converte Javacript para Typescript, não verifica se tem erros, acelara desenvolvimento

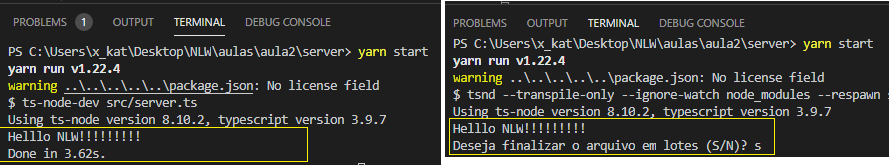
**--ignore-watch node\_modules** >> Não converte código dentro da pasta node\_modules

**--respawn** > Funcionamento de se alterar o código faz um restart, senão continua rodando e só sai se der Crtl+C

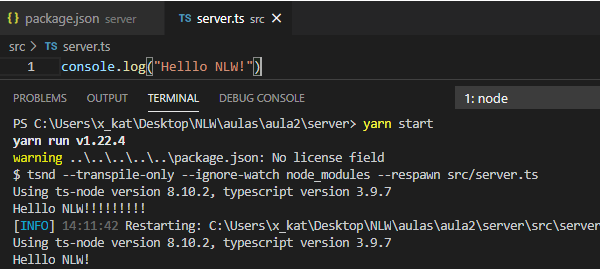
**src/server.ts",**

**},**

Abaixo um comparativo das saídas de usando o flag respawn, exemplificando que ele continua rodando...

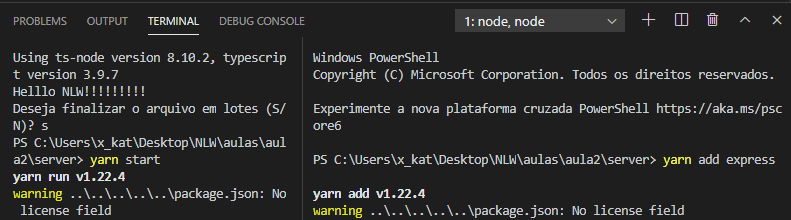


Outro exemplo é que se alterar a saída, ele atualiza automaticamente:



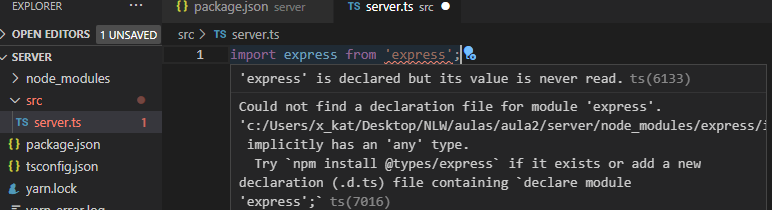
Por causa dessa funcionalidade vamos abrir outro terminal e vamos instalar outra dependência o express:

**yarn add express**



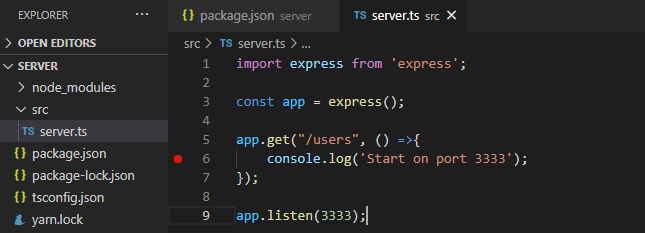
O express é micro-framework que traz algumas funcionalidades prontas que evitam ficar fazendo configurações.

Ao importar o express no arquivo server.ts dá um erro de dependecia já visto anteriormente

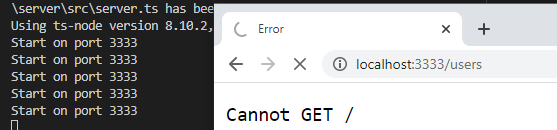


Quando estamos trabalhando com typescript, alguns pacotes adicionados são também em typescript e outros não. Os que não são a comunidade precisar criar esses pacotes de tipagem (com @type) que precisam ser instaladas também.

**npm install @types/express –D**



Executando o método get acima, já temos a comunicação da aplicação mas os resultados não saem no navegador.



Para dar a saída temos que incluir os 2 parâmetros que são injetados pelo método app.get: o request e o response. O request traz informações sobre a requisição (o cabeçalho e o corpo, o usuário, e-mail, senha, dados recebidos pelo Front-End). O response é a resposta da API para a aplicação. O código fica assim:

import express from 'express';

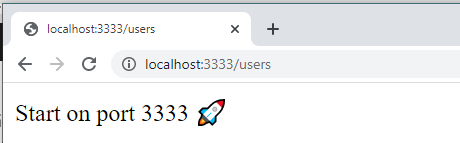
const app = express();

app.get("/users", (request, response) =>{

return response.send('Start on port 3333 🚀')

});

app.listen(3333);



Porém como trabalhar com formato JSON precisamos fazer algumas alterações:

import express from 'express';

const app = express();

app.get("/users", (request, response) =>{

const users =[

{name: 'Diogo', age:25},

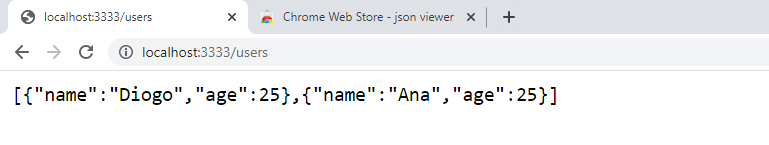
{name: 'Ana', age:25},

]

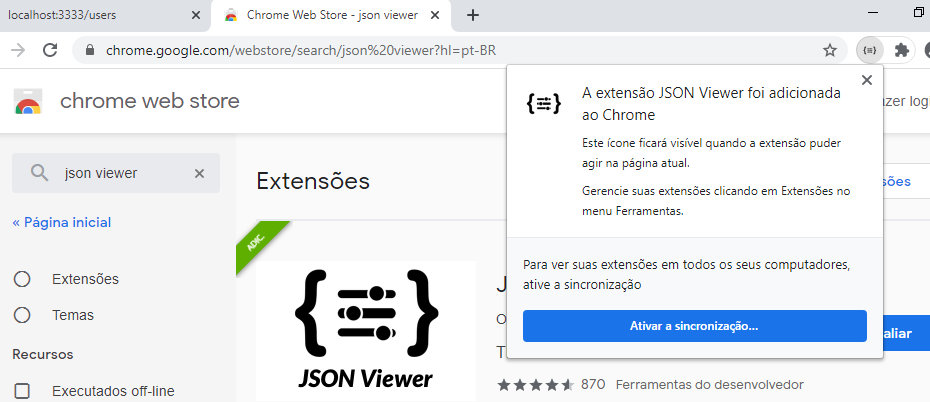
return response.json(users)

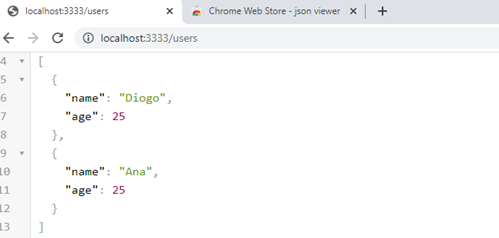
});

app.listen(3333);



Para visualizarmos melhor os resultados podemos instalar um plugin no navegador Chrome, JSON Viewer:

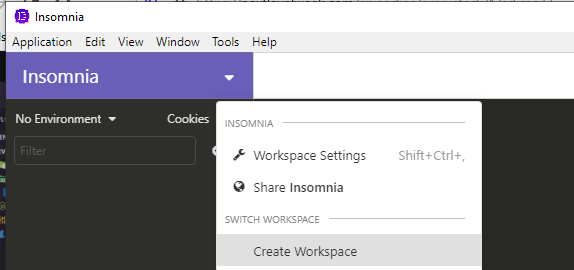




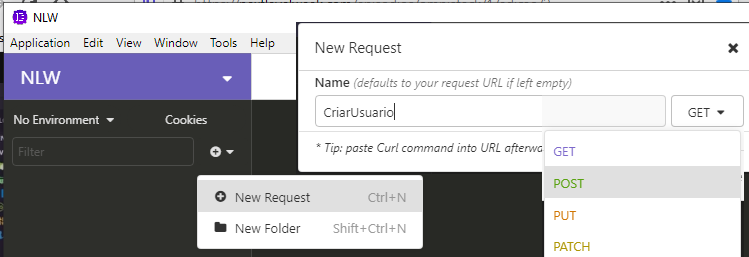
Rotas, Recursos e Métodos HTTP

Cada endereço é uma rota (Ex.: <http://localhost:3333/users> ou <http://localhost:3333/contact>). O recurso é qualquer coisa que vem depois da URL base, ou seja, /users e /contact são os recursos. Os métodos HTTP de uma API mais comuns são o GET, POST, PUT e DELETE responsáveis pelas operações de CRUD. Como o pacote express que estamos utilizando parece ter limitações(???) quanto as outras requisições que não seja GET vamos utilizar outra ferramenta: O Insomnia Core. E vamo criar um Worksapace para o NLW.

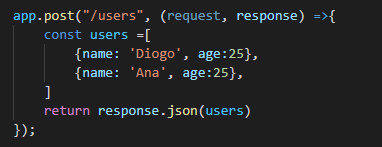
**https://insomnia.rest/download/**

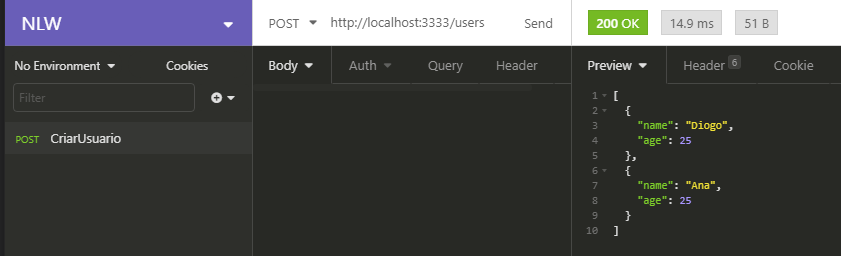


Depois criamos uma nova requisição CriarUsuario pelo método POST:



Depois de alterarmos o método de GET para POST conseguimos obter o resultado desejado:





Parâmetros

Quando fazemos requisição existem 3 tipos de parâmetros:

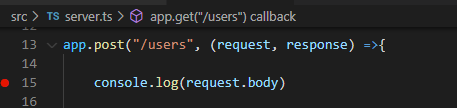
**RequestBody:** Para criar ou atualizar um registro, uma informação vem no corpo da requisição.

**Route Params:** Identifica um recusrso na nossa rota qunado for atualizar ou deletar um registro específico.

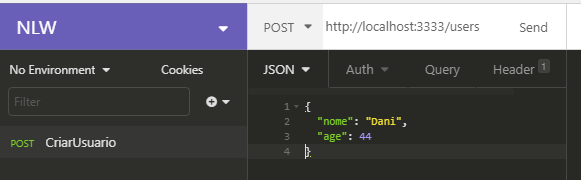
**Query Params:** Usados principalmente em listagens, para fazer filtros, paginação, ordenação, etc.

**RequestBody**

Se colocarmos um request.body no método POST:

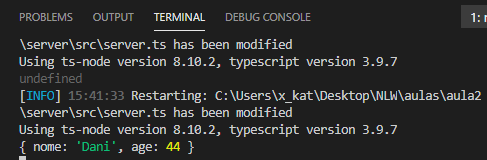


E no Insomnia enviarmos um novo registro de usuário e colocamos método POST:



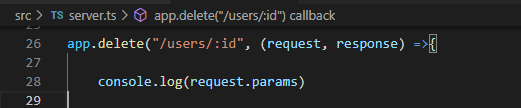
No terminal retorna undefined pois o express não entende JSON por padrão. Corrigimos isso adicionando ao código o módulo que interpreta o JSON:

**app.use(express.json());**



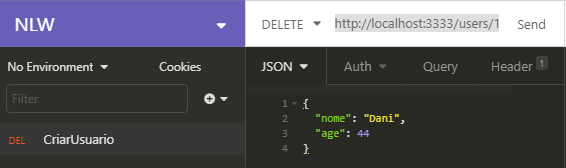
**Route Params**

Se colocarmos um request.params no método DELETE:

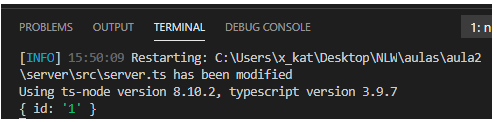


E no Insomnia enviarmos uma rota com a barra e um id e selecionamos método DELETE:

**http://localhost:3333/users/1**

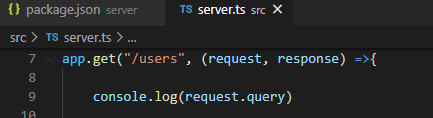


No terminal retorna

****

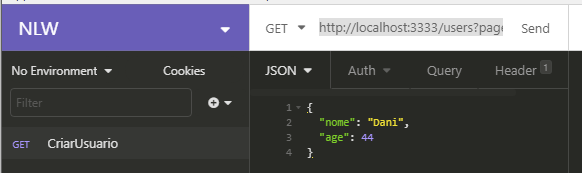
**Route Query**

Se colocarmos um request.query no método GET:

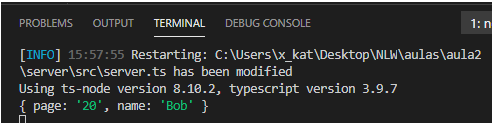


E no Insomnia enviarmos uma rota com a query:

**http://localhost:3333/users?page=20&name=Bob**



No terminal retorna

****

**Casos de Uso**

Identificando aonde vai ter acesso a Back-End pelo layout:

**Conexões**

* Rota para listar o total de conexões realizadas

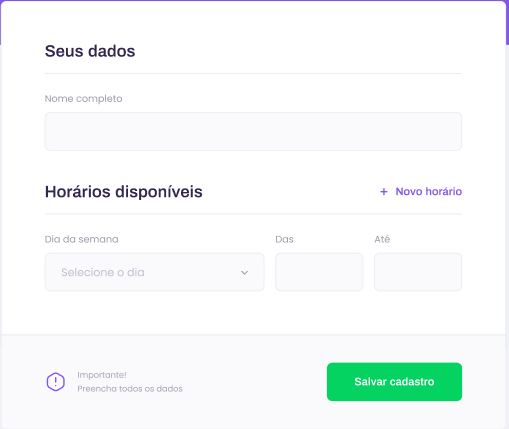


* Rota para criar uma nova conexão (Entrar em contato). Juntamente irá abrir o WhatZap do professor



**AULAS**

* Rota para criar uma aula;



* Rota para listar aulas
  + Filtra por matéria, dia da semana e horário



**Armazenamento interno do Mobile**

* Professores favoritos



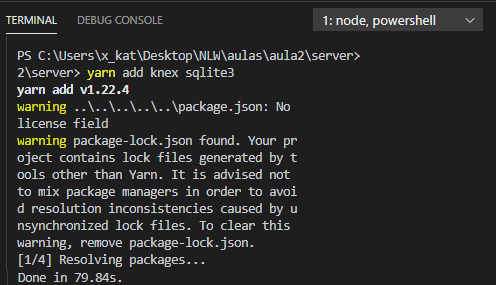
**ROTA PARA CRIAR UMA AULA - PÁGINA DE CADASTRO.**

**Databases**

Instalar

* knex (Query-Builer, escreve comando SQL em Javascript)
* sqlite3 (driver para que o Node consiga se conectar ao sqlite)

**yarn add knex sqlite3**

****

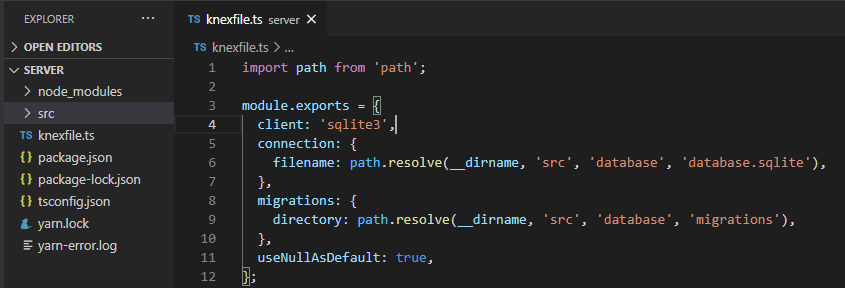
Adicione uma pasta dentro de src com um arquivo connections.ts:



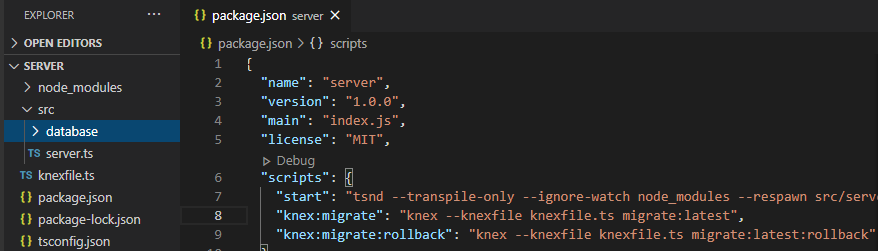
A função path.resolve é útil pois não precisamos colocar as barras do caminho. Esta função irá gerar um arquivo sqlite dentro da pasta database (que é o valor gravado em \_dirname também). O sqlite não sabe o que fazer com valores não preenchidos por isso setamos use NullAsDefault como true.

**MIGRATIONS**

Como o knex só entende Javascript e não Typescript precisamos criar um arquivo na raiz do projeto chamado knexfile.ts:



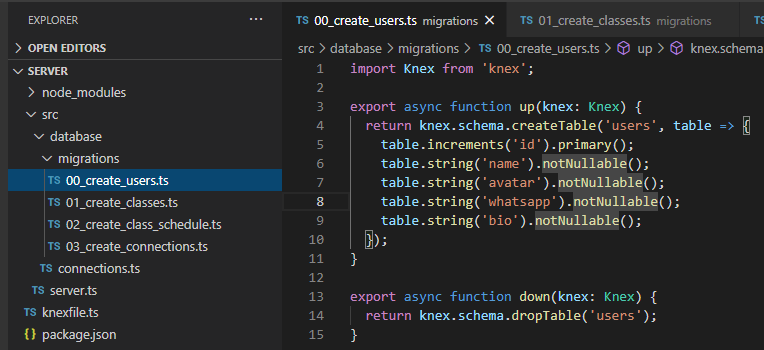
Depois incluímos um script no arquivo packages.json, que faz o redirecionamento dos comando knex através do arquivo anterior. Para cada comando que a gente tiver necessidade de usar pelo Typescript, podemos declarar neste script.



* make: Cria uma nova migração, com o nome da migração sendo adicionado.
* latest: Executa todas as migrações que ainda não foram executadas.

Controlam **as** versões dentro do banco de dados, similar ao Git. Por isso que vamos montar a estrutura do BD utilizando essa ferramenta. Quando outro desenvolvedor for trabalhar no mesmo projeto o Migrations reconstrói o banco na sua versão mais atualizada.

Poderíamos utilizar um comando knex que cria a Migrations tudo em Javascript (???), mas como estamos utilizando Typescript vamos cria-los manualmente. Primeiro, é interessante nomear os arquivos com número, pois isso definirá uma ordem de execução do Migrations. Crie uma pasta migrations dentro de database com os seguintes arquivos para gerar os objetos do DB:



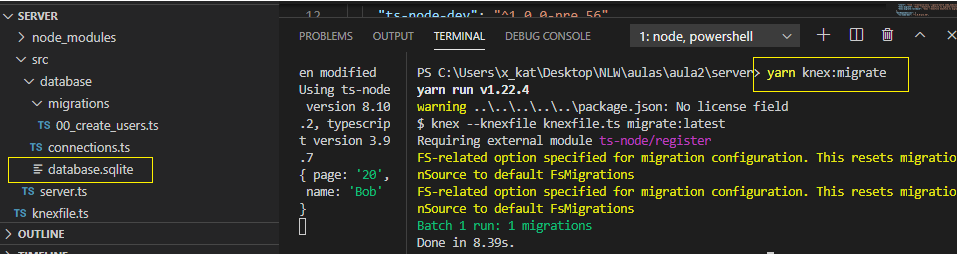
* Up: Executa o especificado (pelo parâmetro config.name) ou a próxima migração cronológica que ainda não foi executada.
* Down: Desfaz o especificado (pelo parâmetro config.name) ou a última migração que foi danificada

Para maiores informações de implementação desses objetos pode-ser obtida na documentação oficial:

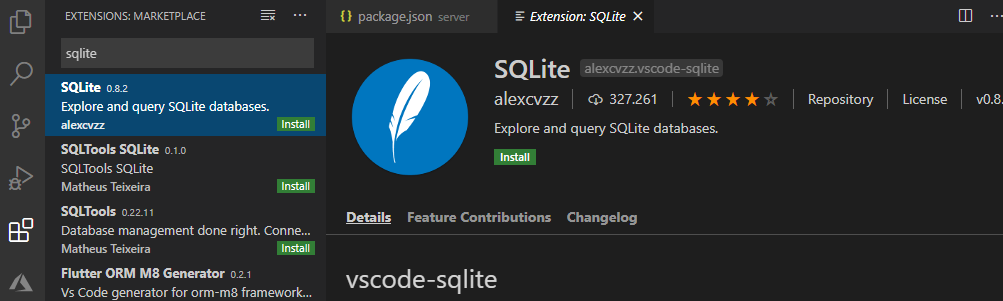
**http://knexjs.org/#Migrations-API**

Rodando o comando adaptado para criar a tabela de usuários:

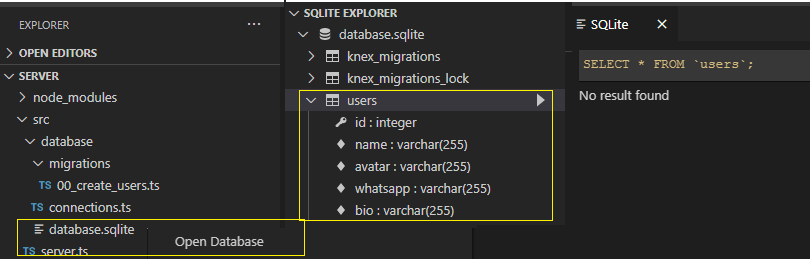
**yarn knex:migrate**



Para visualizar os dados gravados em database.sqlite precisamos instalar a Extensão abaixo. (OBS.: Essa extensão dá muitos problemas na hora de visualizar os dados e tem que ficar **dando RELOAD no VS Code e reiniciando o yarn start**)

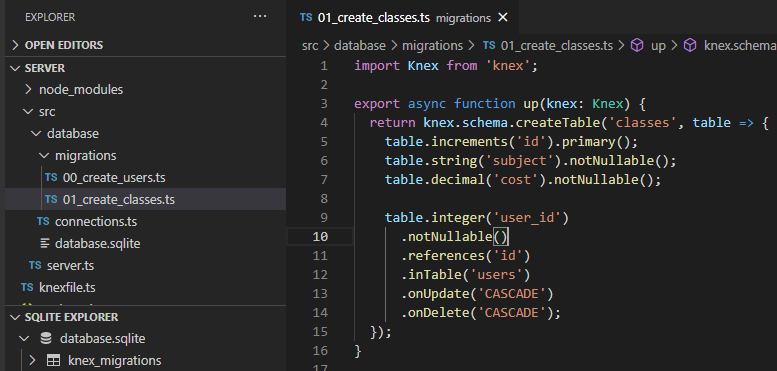


Ao clicar com o botão direito no arquivo database.sqlite, podemos abrir uma nova janela com nossa conexão sqlite com a tabela gerada. As outras tabelas migrations são as que guardam o histórico de alterações desse BD. Se outro desenvolvedor executar o comando yarn knex:migrate executará só as migrações não executadas.



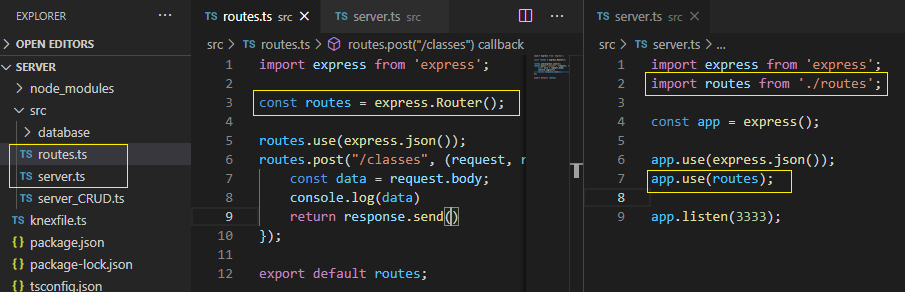
Se algo der errado, o mais fácil e deletar o arquivo database.sqlite e tentar gerar novamente tudo. No código abaixo, podemos ver a implementação de tabelas relacionadas e Trigger em Cascade para Deletar e Fazer update das tabelas. A cada alteração do banco, para atualizar o arquivo database.sqlite precisamos rodar o comando:

**yarn knex:migrate**

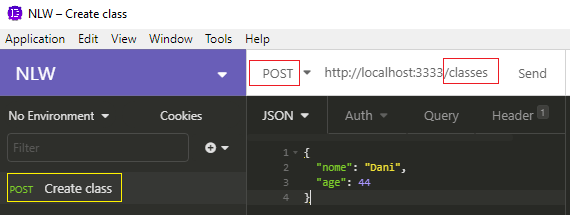


Organizando Código

Por critério de organização iremos criar um arquivo routes.ts para retirar o código de rotas que estava dentro de server:



Como está em um arquivo separado utilizamos express.Router() (anteriormente as rotas eram construídas diretamente através de express). Isso possibilita as rotas serem chamadas e utilizadas em diversas partes do projeto. Para testar as alterações voltamos ao Insomnia e fazemos as seguintes alterações:



Ao enviar temos a resposta no terminal:



Agora altramos os dados do objeto para cadastrar uma classe:

{

"name": "Shyoutaou Shyou",

"avatar": "https://avatars3.githubusercontent.com/u/66930143?s=460&u=9a46318c1563414a627c432d89b8ae53bf359430&v=4",

"whatsapp":"986719596",

"bio": "Software developer, whose likes to be in tune with the technology flow.",

"subject": "Mathematic",

"cost": 250,

"schedule": [

{"week\_day": 1, "from": "8:08", "to": "12:00"},

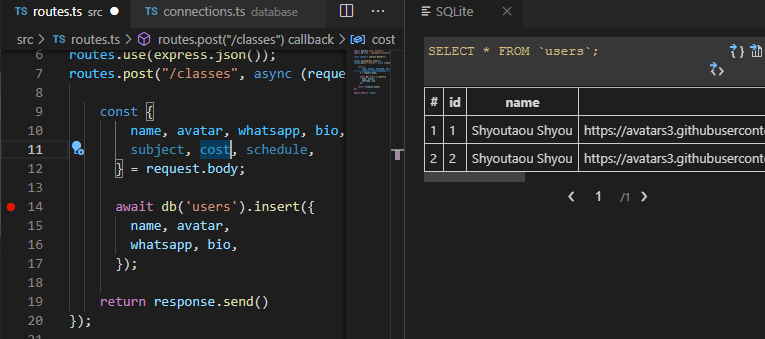
{"week\_day": 3, "from": "10:08", "to": "18:00"},

{"week\_day": 4, "from": "8:08", "to": "12:00"}

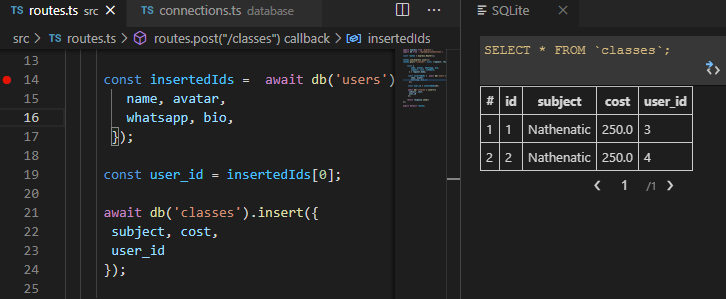
]

}

Fazemos as alterações abaixo no arquivo routes.ts para testar a gravação n taela users:

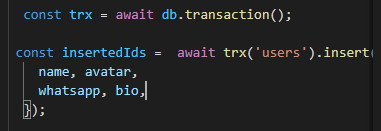


Para inserir classes o mesmo procedimentos, só temos que recuperar o user\_id ao salvar um usuário:



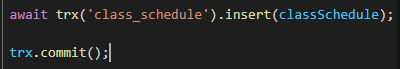
Para habilitar as operações em transações e commit igual no SQL declaramos uma constante:

**const trx = await db.transaction();**



E substituímos por toda ocorrência de db que havia antes. No final damos o commit:

**trx.commit();**



Para tratar erro com try-catch pode ser feito assim:

**const trx = await db.transaction();**

**try**

**{**

**Code here**

**trx.commit();**

**}**

**catch (error)**

**{**

**trx.rollback();**

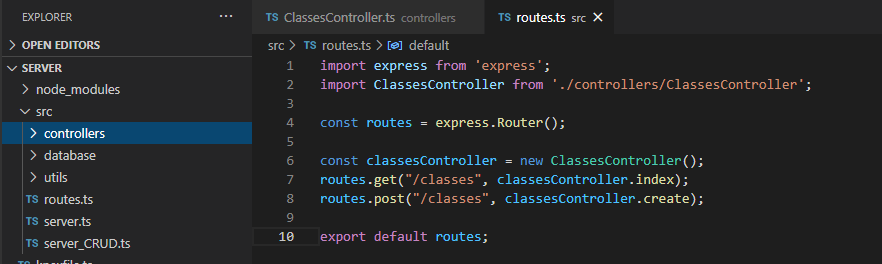
**console.log(error);**

**return response.status(400).json({error: "Unexpected error"})**

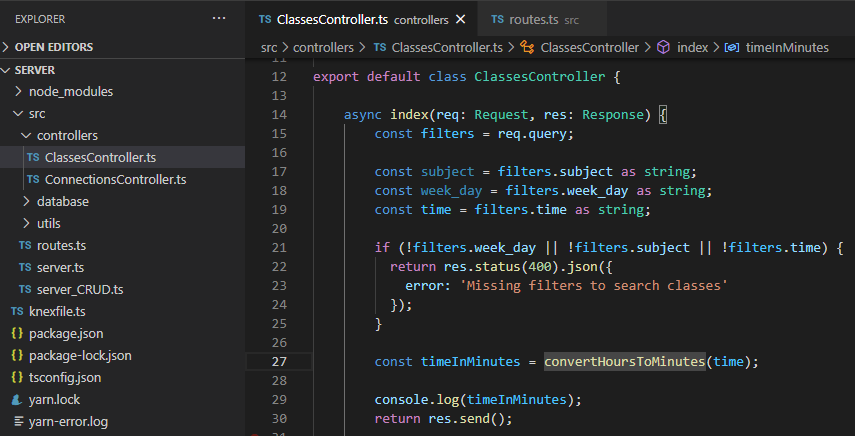
**}**

**ROTA PARA LISTAR AULAS - PÁGINA DE LISTAGEM**

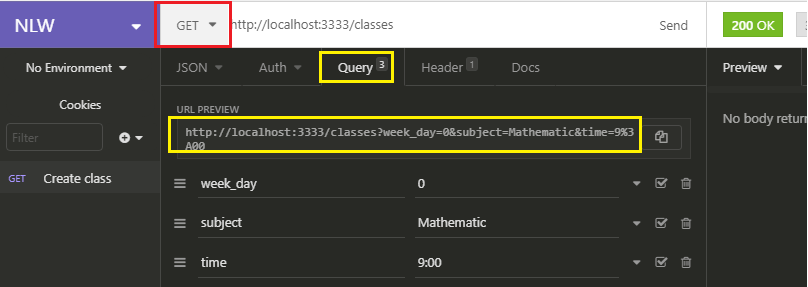
Foi feito um refatoramento de código, retirando tudo que estava em route.ts (que gravava no BD) e deixado somente o que é relativo a rotas neste arquivo. Repare que já foi adicionado a rota do método GET que será usado na página de filtragem.



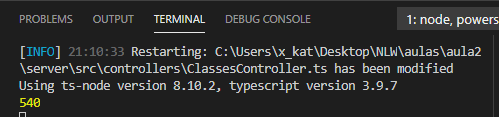
As partes de código de gravação e filtragem estão agora em outra pasta chamada controllers que segue o modelo MVC e que foi visto uma implementação similar quando vimos a parte de componentes anteriormente. Foi criado o arquivo ClassesController.ts e codificado um teste para página de filtro utilizando o método GET:



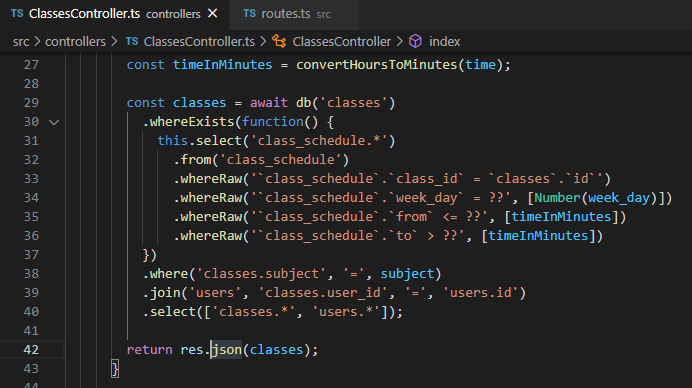
Para testar no Insomnia, crie o 3 parâmetros utilizados (week\_day, subject, time) com o auxílio da interface e clique Enviar



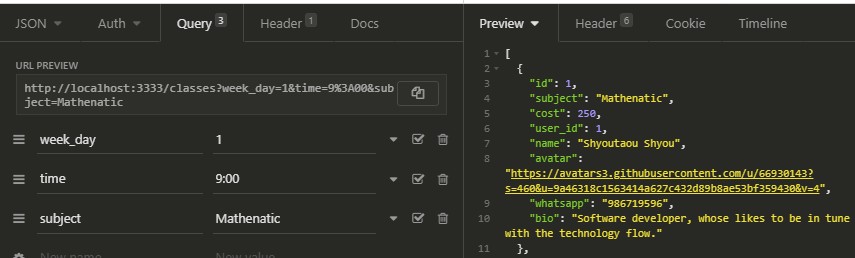
Se retornar o valor dos minutos no terminal está tudo OK:



Adicionando a filtragem e retornando um objeto JSON:

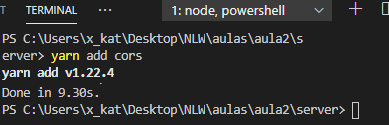


Testando no Insomnia:



Para finalizar, adicionamos o pacote cors, que permite que aplicações em endereços diferentes, no caso, nosso Front-End esteja no localhost:3000 e seja acessado pela API Bach End em localhost:3333. Por padrão, só permite o acesso de alicações no mesmo endereço a API.

**yarn add cors**

****