Sumário

[Instalando Ferramentas 2](#_Toc151876854)

[Extensões VSCode 2](#_Toc151876855)

[Links úteis para o projeto 2](#_Toc151876856)

[Instalando ferramentas do Back-End 2](#_Toc151876857)

[Tags HTML 3](#_Toc151876858)

[Entendendo o CSS 4](#_Toc151876859)

[Programas e Extensões do Chrome 5](#_Toc151876860)

[Programas 5](#_Toc151876861)

[Extensões do Chrome 5](#_Toc151876862)

**DDD**

# Instalando Ferramentas

## Extensões VSCode

**Eslint**

Instale a extensão [ESLint](https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint) no seu Visual Studio Code.

Abra a Paleta de Comandos:

Se estiver no Windows ou Linux: CTRL + SHIFT + P

Abra as configurações em JSON buscando por: Abrir as Configurações do usuário (JSON)

Adicione dentro do JSON o código abaixo:

JSON

"editor.codeActionsOnSave": {

"source.fixAll.eslint": true

},

*“serve para corrigir todos os arquivos da aplicação com apenas um comando”*

**Prisma**

Abra as configurações em JSON buscando por: Abrir as Configurações do usuário (JSON)

Adicione dentro do JSON o código abaixo:

JSON

"[prisma]": {

"editor.formatOnSave": true

},

*“serve para deixar os comandos das tabelas alinhados”*

npm install eslint -D

npm i @rocketseat/eslint-config -D

**PostCSS Language Suport**

**Tailwind CSS IntelliSense**

## Links úteis para o projeto

<https://www.figma.com/community/file/1240070456276424762>

<https://react-native.rocketseat.dev/>

<https://github.com/rocketseat-education/nlw-12-spacetime-ignite>

<https://efficient-sloth-d85.notion.site/Trilha-Ignite-e2ed19139b544a46984a28b65dcd4aae>

## Criando Back-End

O **NodeJs** deve estar instalado, para verificar se está instalado e as versões digite:

**node -v**

**npm -v**

Como o projeto é **Full-Stack,** primeiro criamos a pasta do projeto (exemplo: time-line) e dentro da pasta criamos a pasta do Back-End chamada **server.**

Para criar um projeto Javascript abra a pasta server e digite:

**npm init -y**

Para instalar o **Typescript**:

**npm i typescript -D**

**npm i @types/node -D**

**npx tsc –init**

**npm i tsx -D**

No arquivo **typescript.config** alterar:

"target": "es2020",

Para melhor compatibilidade com o nodejs.

Como o node não compila o Typescript temos que converter a execução do Typescript para Javascript, para isso criamos a pasta **src** e o arquivo da aplicação que neste caso **server.ts**.

server

src

server.ts

Dentro do arquivo **package.json**

"scripts": {

    "dev": "tsx watch src/server.ts"

  },

Feito isso podemos rodar a aplicação com:

**npm run dev**

**Instalando Eslint no Back-End:**

Para utilizar as configurações do Eslint feitas pele rocketseat fazemos o seguinte comando:

**npm i @rocketseat/eslint-config -D**

Dentro da pasta do projeto criamos um arquivo chamado **.eslintrc.json** e fazemos a seguinte configuração:

{

    "extends": [

        "@rocketseat/eslint-config/node"

    ]

}

*“quando fazer o front-end com react, mudamos de node para react o final”*

Dentro do arquivo package.json

  "scripts": {

    "dev": "tsx watch src/server.ts",

    "lint": "eslint src --ext .ts --fix"

  },

Para corrigir arquivos pelo Eslint:

**npm run lint**

**Instalando o** **Framework Fastify**:

**npm i fastify**

Com o Fastify instalado podemos configurar o arquivo **server.ts** para rodar a **API** da aplicação:

import fastify from "fastify"

const app = fastify()

app.get('/hello', () => {

    return 'Hello World'

})

app.listen({

    port: 3333,

}).then(() => {

    console.log('😁 HTTP server running on http://localhost:3333')

})

**Prisma + SQLite**

Prisma é uma ferramenta que intercepta a informação do back-end com o banco de dados. Ela tem suporte a praticamente todos os bancos de dados, ou seja, no ambiente de produção poderemos mudar de SQLite para algum outro se preciso.

Para instalar o **Prisma**:

**npm i prisma -D**

Depois de instalado se quiser verificar os possíveis comandos dele basta digitar:

**npx prisma -h**

Por padrão o banco de dados utilizado pelo prisma é o Postgresql, mas vamos utilizar o SQLite, então podemos inicializa-lo da seguinte forma:

**npx prisma init --datasource-provider SQLite**

Após fazer o comando acima vai ser gerado um arquivo **.env**, nele fica configurado a URL do banco de dados da aplicação específico para o ambiente em que estamos usando. Esta configuração pode ser alterada se quisermos ou se mudarmos para um ambiente de produção.

Também foi criada uma pasta chamada prisma, com o arquivo schema.prisma. Nele podemos definir as tabelas do banco de dados.

**Criando a tabela com o prisma**

Dentro do arquivo schema.prisma:

generator client {

  provider = "prisma-client-js"

}

datasource db {

  provider = "sqlite"

  url      = env("DATABASE\_URL")

}

model User {

  id   String @id @default(uuid())

  name String

}

//model cria tabela

//@id indica que é uma chave primária

//uuid inseri um id único universal

Agora temos que executar um comando para **criar a tabela dentro de um arquivo do banco de dados:**

**npx prisma migrate dev**

Após fazer este comando o prisma vai pedir um nome para a nova migration, para que ele salve um histórico das criações e alterações de tabelas, funciona como um sistema de controle versões do banco de dados:

**√** Enter a name for the new migration: ... *create users table*

Feito isso o Prisma cria uma pasta chamada migrations, dentro dela tem um arquivo do histórico de alterações e também o arquivo **sql**, já com os comandos criados de forma automática pra criação da tabela.

Dentro da pasta prisma foi criado um arquivo do banco de dados *“dev.db”*, que podemos acessa-lo através de alguma ferramenta de acesso a bancos sql, ou pela própria ferramenta do prisma.

**Acessar o banco de dados** com a ferramenta do **Prisma**:

**npx prisma studio**

Este comando abre uma aba no navegador onde é possível salvar dados no banco de dados.

**Acessar o banco de dados** pela **API**:

Para isso precisamos instalar uma ferramenta do prisma com o seguinte comando:

**npm i @prisma/cliente**

Agora alteramos o arquivo **server.ts** para acessar:

import fastify from 'fastify'

import { PrismaClient } from '@prisma/client'

// importe prisma client

const app = fastify()

const prisma = new PrismaClient()

// faça conexão com o baco com o comando new

app.get('/users', async () => {

  const users = await prisma.user.findMany()

  // faça a função ficar asincrona, faça a conexão com users como await, passa a propriedade findMany e adicione users a rota com get.

  return users

})

app

  .listen({

    port: 3333,

  })

No navegador acessamos <http://localhost:3333/users>

## Criando Front-End

Na pasta inicial do projeto (time-line) criamos a pasta do **Front-End** chamada **web**. Abrimos a pasta web no terminal para instalar o Framework Next-Js**. Para instalar o Next-Js** utilizamos o seguinte comando:

**npx create-next-app@latest web --use-npm**

*“web é o nome específico para este projeto”*

Após fazer este comando ele vai fazer uma pergunta se realmente quer instalar, pressione Y e enter.

Na sequência teremos mais uma serie de perguntas especificas para cada projeto.

√ Would you like to use TypeScript? ... No / **Yes**

√ Would you like to use ESLint? ... No / **Yes**

√ Would you like to use Tailwind CSS? ... No / **Yes**

√ Would you like to use `src/` directory? ... No / **Yes**

√ Would you like to use App Router? (recommended) ... No / **Yes**

√ Would you like to customize the default import alias (@/\*)? ... No / **Yes**

√ What import alias would you like configured? ... @/\*

Projeto criado, podemos roda-lo com o comando:

**npm run dev**

Vai abrir a página padrão de um projeto React com Next-Js. Agora podemos fazer a nossas configurações especificas. Primeiro vamos limpar este projeto:

* Excluir o README.md
* Excluir os arquivos da pasta public
* Excluir o arquivo favicon.ico da pasta src/app
* Dentro da pasta src/app vamos limpar o arquivo global.css deixando apenas as primeiras linhas:

@tailwind base;

@tailwind components;

@tailwind utilities;

* Dentro da pasta src/app vamos limpar html dentro do arquivo page.tsx:

export default function Home() {

  return (

    <h1>Hello World</h1>

  )

}

**Dica:** para que o html preencha as tags automaticamente dentro do React-Js adicionamos duas configurações a nossa configuração do VSCode.

Abra as configurações em JSON buscando por: Abrir as Configurações do usuário (JSON)

Adicione dentro do JSON o código abaixo:

JSON

"emmet.syntaxProfiles": {

"javascript": "jsx"

},

"emmet.includeLanguages": {

"javascript": "javascriptreact"

},

**Instalando Eslint no Front-End**

Usamos o comando:

**npm i @rocketseat/eslint-config -D**

Em seguida alteramos o arquivo **.eslintrc.json**

{

  "extends": [

    "next/core-web-vitals",

    "@rocketseat/eslint-config/node"

  ]

}

**Instalando Prettier**

Usamos o seguinte comando:

**npm i prettier-plugin-tailwindcss -D**

Na pasta do projeto criamos o seguinte arquivo:

**prettier.config.js**

# Tags HTML

*<!-- -->* : Cria comentários.

*<h1> </h1> ao <h6> </h6>:* Coloca títulos na página sendo que h1 é o primeiro da hierarquia de títulos.

*<p> </p>:* Cria parágrafos.

*<hr>:* Cria uma linha horizontal.

*<br>:* Quebra de linha.

*<img src=”” alt=””>:* Inseri uma imagem. Para não precisar ficar digitando o caminho e o nome da imagem basta clicar entre as aspas do src, segurar Ctrl e apertar espaço para achar o caminho do arquivo e entre as aspas do alt coloque uma descrição curta da imagem.

*<i> </i>:* O texto fica em itálico. **OBS: não semântico.**

*<em> </em>:* O texto fica em itálico e indica ênfase.

*<address> </address>:* O texto fica em itálico e indica que é um endereço.

*<b> </b>:* O texto fica em negrito. **OBS: não semântico.**

*<strong> </Strong>:* O texto fica em negrito e indica que ele está em destaque.

*<u> </u>:* O texto fica sublinhado. **OBS: não semântico.**

*<ins> </ins>:* O texto fica sublinhado e indica texto inserido.

*<del> </del>:* O texto fica riscado e indica texto excluído.

*<mark> </mark>:* Adiciona um marca-texto. *Obs: para mudar a cor de um marca-texto basta usar css dentro da tag.* ***Exemplo: <mark style=”background-color: lime;”>***

*<small> </small>:* Deixa o texto pequeno.

*<sup> </sup>:* Deixa o texto sobrescrito.

*<sub> </sub>:* Deixa o texto subscrito.

<*code> </code>:* Deixa as fontes monoespaçadas, ou seja, todas as letras ocupam o mesmo espaço.

*<pre> </pre>:* Formata todo texto conforme escrito no documento HTML respeitando até mesmo as quebras de linha.

*<q> </q>:* Coloca o texto entre aspas indicando uma citação.

*<blockquote> </blockquote>:* Indica que o texto é uma citação. *Obs: podendo colocar o link da citação na tag.* ***Exemplo: <blockquote cite=”http//:www........”> Bla bla bla <blockquote>.***

*<abbr> </abbr>:* Usado para mostrar o significado de uma abreviação quando passar o mouse por cima dela.

*<ol> </ol>:* Cria listas ordenadas.

*<ul> </ul>:* Cria listas não ordenadas.

*<li>:* Cria as linhas de uma lista ordenada ou não ordenada.

*<dl> </dl>:* Cria lista de definições.

*<dt> </dt>:* Cria um termo para uma lista de definições.

*<dd> </dd>:* Cria a definição de um termo de uma lista de definições.

*<a href="........"* *> </a>:* Cria um link.

*<Picture> </Picture>:* Cria vários sources de imagens ao mesmo tempo.

*<audio src=””> </audio>:* Adiciona um áudio.

*<vídeo src=””> </vídeo>:* Adiciona um vídeo**. OBS. Essa opção gera custos autos de hospedagem. Veja em anotações.**

<span> </span>: é um container genérico em linha para conteúdo fraseado. Ele pode ser usado para agrupar elementos para fins de estilo (usando os atributos class ou id ), ou para compartilhar valores de atributos, como lang. <span> é muito parecido com o elemento <div>, entretando <div> é um elemento de nível de bloco enquanto <span> é um elemento em linha.

<div> </div>: é um container genérico para conteúdo de fluxo. Ele pode ser utilizado para agrupar elementos para fins de estilos (usando class ou id), ou para compartilhar valores de atributos, como lang. Ele deve ser utilizado somente quando não tiver outro elemento de semântica (tal como <article> ou <nav>).

# Entendendo o CSS

**Regra:** Uma regra CSS é uma declaração que segue uma sintaxe própria e que define como será aplicado estilo a um ou mais elementos HTML. Um conjunto de regras CSS formam uma Folha de Estilos.

**Seletores:** Um seletor CSS é a primeira parte de uma regra CSS. É um padrão de elementos e outros termos que informam ao navegador quais elementos HTML devem ser selecionados para que os valores de propriedade CSS dentro da regra sejam aplicados a eles.

**Propriedade:** é o atributo do elemento HTML ao qual será aplicada a regra (por exemplo: font, color, background, etc...).

**Valor:** é a característica específica a ser assumida pela propriedade (por exemplo: letra tipo arial, cor azul, fundo verde, etc...)

**Declarações:** É o conjunto entre propriedade e valor. Qualquer motor do CSS calcula quais declarações serão aplicadas para todos um único elemento da página em ordem adequadamente, a fim de exibí-lo com o estilo correto.

**@:** São regras que dever ser sempre colocadas nas primeiras linhas do arquivo CSS ou da tag style e não deve ser colocada dentro de seletores.

Exemplo:

**@import** url('https://fonts.googleapis.com/css2?family=Work+Sans:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&display=swap');

**body {**

**font-family: 'Work Sans', sans-serif;**

**font-weight: bolder;**

**font-size: 3em;**

**font-style: italic;**

**}**

# Programas e Extensões do Chrome

## Programas

**Visual Studio Code:** (VS Code) é um editor de código aberto desenvolvido pela Microsoft.

**Gimp:** É um programa de código aberto voltado, essencialmente, para edição e criação de imagens raster e, em menor escala, para desenho vetorial. O software é repleto de recursos, de fácil uso e uma boa alternativa gratuita ao mais conhecido dos editores, o gigante Adobe Photoshop.

**HandBrake:** É um programa multiplataforma e multitarefa em código aberto de conversão de arquivos de vídeo e DVD e Bluray para MPEG-4, disponível para Mac OS X, Linux e Windows. Com ele é possível compactar bastante um vídeo.

**Git:** Programa para versionamento local.

**GitHub Desktop:** Programa para versionamento online direto no site do GitHub.

## Extensões do Chrome

**Todas as extensões devem ser baixadas no site**: <https://chrome.google.com/webstore/category/extensions?hl=pt-br>

**Web Developer:** Essa extensão foi criada para concentrar diversas ferramentas simples e úteis em uma barra adicional no Chrome. O conceito veio da barra de ferramentas do desenvolvedor.Ela é repleta de utilidades para quem quiser usar em seu trabalho diário, não só para os programadores, mas também para quem opera no webdesign. Dentre os recursos estão alguns como:adição de contornos aos elementos (tudo com apenas um clique);detecção de imagens quebradas;controle de habilitação/desabilitação de popups, plugins e JavaScript.

**ColorZilla:** É uma extensão de navegador muito versátil que permite descobrir um código de cor específico para um determinado site. Além disso, esta ferramenta acaba sendo indispensável para designers gráficos, desenvolvedores web ou front-end.

**Fonts Ninja**: Identifica fontes de qualquer site, que não esteja dentro de uma imagem.

**Window Resizer:** permite emular diversas resoluções de ecrã para testar as suas páginas de Internet. Após instalada é necessário fazer uma permissão, para isso abra o menu de extensões, gerencias extensões e clique em saiba mais do ícone do Window Resizer e ligue “permitir acesso a URLs de arquivo”.