Criando e configurando Shadow CLJS

John Doe

December 7, 2020

Contents

1 Conectando Shadow CLJS no terminal **1**

2 Criando as pastas e documentos **1**

3 Instalando dependencias do react **4**

4 Instalando e configurando tailwind **4**

5 Lispcast: Understandig Reframe **5**
5.1 01 Reframe Stack Overview 5
5.2 02 Getting Set Up 5

1 Conectando Shadow CLJS no terminal

- SPC m C: cider connect cljs
- selecionar localhost com a porta 9000
- selecionar shadow
- selecionar a build :app

2 Criando as pastas e documentos

Primeiramente iniciamos um projeto node com o comando abaixo:

```
yarn init -y
```

Com esse comando a pasta `package.json` é criada:

```
ls
```

Então criamos a pasta onde ficará o source do nosso código:

```
mkdir -p src/cljs/app
```

E colocamos o inicio da código:

```
(ns app.core)

(defn ^export init []
 (js/console.log "Oi mundo" ))
```

Depois criamos uma pasta onde ficarão as configurações da biblioteca `shadow.cljs`:

No arquivo shadow-cljs.edn configuramos com os seguintes parâmetros:

- :source-paths is where you put source code
- :dependencies is ClojureScript deps of this projects
- :builds is where you specify build configurations
- :app is a build id named by us, we will need it to run specific compilations from command line tools or from APIs
- :output-dir decides where the generated files are saved
- :asset-path is the base path of files of the hot updated code
- :target is short for "compilation target", :browser is for browser apps
- :modules specifies the modules we want to generate, :main also means the bundle will be named main.js
- :init-fn specifies the main function of the whole program
- :devtools specifies configurations for development tools
- :dev-http tells shadow-cljs to serve the folder target/ on port 8080.
- :main vai ser o nome do JavaScript compilado

- :devtools :http-root onde os arquivos da build e o arquivo html fica,

```
mkdir -p resources/public/js
```

```
{:source-paths ["src/cljs"]
 :nrepl {:port 9000}
 :dependencies [[cider/cider-nrepl "0.25.5"]]
 :builds {:app {:target :browser
                :output-dir "resources/public/js"
                :modules {:main {:init-fn app.core/init}}
                :devtools {:http-root "resources/public"
                           :http-port 3000}}}
 }
```

Depois disso adicionamos a biblioteca no projeto Node com o comando:

```
yarn add shadow-cljs
```

Depois disso no arquivo package.json, adicionar o seguinte código, dentro das chaves, não esquecer de colocar a vírgula no fim do primeiro bloco do JSON:

```
"scripts": {"start": "shadow-cljs watch app"}
```

Depois disso adicionamos um arquivo HTML para ser o host do nosso arquivo JS compilado.

```
<!DOCTYPE html>

<html lang="pt-br">
    <head>
        <meta charset="UTF-8"/>
        <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
        <title >Document</title>
    </head>
    <body>
        <script src="/js/main.js">
        </script>
    </body>
</html>
```

Depois podemos iniciar o programa com com o script `start` escrito acima:

```
yarn start
```

3 Instalando dependencias do react

```
yarn add react react-dom create-react-class
```

4 Instalando e configurando tailwind

Full Stack Clojure Contact Book - [4] Front End Preparation `https://www.youtube.com/watch?v=Jf94HNmCXyU`

- Instalando autoprefixer 9.8.6, postcss e postcss-cli para corrigir bug de compilação do tailwind

```
yarn add tailwind onchange autoprefixer postcss postcss-cli
```

- Configurando

```
mkdir -p resources/public/css
```

Precisamos criar 2 arquivos Este primeiro na raiz do arquivo

```
module.exports = {
  purge: [],
  theme: {
    container: {
      center: true
    },
    extend: {},
  },
  variants: {},
  plugins: [],
}
```

Este segundo onde o CSS vai ficar:

```
@tailwind base;

@tailwind components;

@tailwind utilities;
```

Depois adicionar o seguinte script no package.json:

```
"build-styles": "tailwind build src/cljs/tailwind.css -o resources/public/css/main.css"
```

5 Lispcast: Understandig Reframe

5.1 01 Reframe Stack Overview

- DOM (Document Object Model): API do browser/javascript para mudar os elementos da página, tanto o HTML e quanto os eventos. É difícil manter o estado da aplicação e o DOM sincronizados.
- O React foi criado para lidar com os eventos de forma reativa com o DOM Virtual, e melhorou a forma como lidar com os eventos.
- O Reagent é um wrapper para o React criado para o ClojureScript, usa o Hiccup para gerar HTML. Cria os átomos que dizem quando a view precisa ser renderizada novamente.
- Re-Frame é um framework para aplicações. Ajuda a capturar e manter as intenções do usuário em um único local. Além disso, controla os efeitos que estão fora do DOM (requisições API, falar com bancos de dados).

5.2 02 Getting Set Up

```
git clone https://github.com/lispcast/understanding-re-frame.git
&& cd understanding-re-frame
&& git checkout -f 001
```