HTTP клиент и GIT клиент – Insomnia

Сборщик Docker

NVM – для быстрого переключения с одной версии Node.js на другую

npm i –g typescript – установка typescript

npm i -D @types/react - установка в проект react

npx create-next-app название --use-nmp – чтобы использовать npm

# Разворачиваем приложение

npx create-next-app top-app --use-nmp

cd top-app

code .

## Настраиваем поддержку typescript

touch tsconfig.json

npm i –D typescript @types/react @types/node

Теперь переименовываем app.js и index.js в app.tsx и index.tsx

npm run dev

И переходим в браузере на <http://localhost:3000/>

Все, в проекте настроен typescript

## Структура проекта

В папке pages удаляем api;

Папка public – для хранения svg и favicon;

В папке styles оставляем только globals, а все остальные стили будем хранить в папках компонентов;

Подключаем пакет classnames, пригодится позже:

npm i -D @types/classnames

В tsconfig.json:

- меняем на "allowJs": false, так теперь если попадется файл .js , то мы получим ошибку;

- меняем на "strict": true, даем нам более строгий режим и проверки на обязательность свойств в typescript;

- добавляем "strictPropertyInitialization": false, - теперь нам необязательно инициализировать каждое свойство класса;

## Настройка Eslint

Устанавливаем eslint и зависимости:

npm i –D eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin

Создаем файл конфигурации - .eslintrc

{

  "root": true,

  "parser": "@typescript-eslint/parser",

  "plugins": [

    "@typescript-eslint"

  ],

  "rules": {

    "semi": "off",

    "@typescript-eslint/semi": [

      "warn"

    ],

    "@typescript-eslint/no-empty-interface": [

      "error",

      {

        "allowSingleExtends": true

      }

    ]

  },

  "extends": [

    "eslint:recommended",

    "plugin:@typescript-eslint/eslint-recommended",

    "plugin:@typescript-eslint/recommended"

  ]

}

## Настройка Stylelint

Устанавливаем stylelint:

npm i –D stylelint stylelint-config-standard stylelint-order stylelint-order-config-standard

Создаем файл конфигурации - .stylelintrc.json

В package.json, в  "scripts" добавляем:

    "stylelint": "stylelint \"\*\*/\*.css\" --fix" – чтобы по вызову команды фиксить любую папку и любой файл с расширением .css

## Настройка отладки приложения

Настраиваем конфигурацию: Run – AddConfiguration – Node.

{

  // Use IntelliSense to learn about possible attributes.

  // Hover to view descriptions of existing attributes.

  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387

  "version": "0.2.0",

  "configurations": [

    {

      "type": "node",

      "request": "attach",

      "name": "Launch Program",

      "skipFiles": [

        "<node\_internals>/\*\*"

      ],

      "port": 9229

    }

  ]

}

Устанавливаем cross-env

npm i cross-env --save-dev

В package.json, в  "scripts" добавляем:

"debug": "cross-env NODE\_OPTIONS='--inspect' next dev",

Есть еще возможность запускать отдельной командой, но это не лучше:

npm --node-options=--inspect=0.0.0.0:9229 run dev

Далее запускаем дебагер: F5 либо Run – StartDebugging

Далее ставим брэкпоинт. Обновляем приложение на localhost:3000, и видим в VSCode,

Всё, приложение развернуто.

# Работа с компонентами NEXT

## Компонент HEAD

Содержит метатеги, title, link.

В каждом компоненте можно импортировать и вставить HEAD

Title возьмет из самого вложенного.

Link rel=”icon” скомпонует их все. Но если задать свойство key (key={1}) обоим link, то возьмет также из самого вложенного.

## Компонент DOCUMENT

В папке pages необходимо создать файл \_document.tsx

Необходимо сделать его классовым компонентом, т.к. extend от Document.

Тут необходимо отобразить структуру приложения, импортировав соответствующие компоненты из 'next/document'.

Нужно также задать lang для html и пробросить стандартные пропсы и контекст.

import Document, { Html, Head, Main, NextScript, DocumentContext, DocumentInitialProps } from 'next/document';

class MyDocument extends Document {

  static async getInitialProps(ctx: DocumentContext): Promise<DocumentInitialProps> {

    const initialProps = await Document.getInitialProps(ctx);

    return { ...initialProps };

  }

  render(): JSX.Element {

    return (

      <Html lang="ru">

        <Head />

        <body>

          <Main />

          <NextScript />

        </body>

      </Html>

    );

  }

}

export default MyDocument;

# Подключаем шрифты и цвета

## Выбираем шрифты

Можно например на <https://fonts.google.com/>

Выбираем нужные виды найденного шрифта, и ссылка link формируется автоматически.

Вставляю сформированные link в App в HEAD. Проверяю\меняю на самозакрывающиеся теги.

## Выбираем цвета

Цвет выбираем из макета в Figma и выносим в переменные (можно и font-family вынести), например так:

//globals.css

:root {

  --black: #000;

  --white: #FFF;

  --primary: #01A7FD;

  --red: #900;

  --green: #377A04;

  --green-light: #6AE908;

  --font-family: "Inter", sans-serif;

}

# Работа с SVG

## **Настройка SVG**

Можно просто сделать через img и в src указать файл, который заранее импортирован в папку с компонентом.

Но, можно сделать через полноценный React – компонент.

Для этого создаем в корне файл next.config, в который добавляем правило для webpack(config):

- если есть js|ts файл с расширение .svg, то используется специальный загрузчик use: ['@svgr/webpack']

module.exports = {

  webpack(config) {

    config.module.rules.push({

      test: /\.svg$/,

        issuer: { and: [/\.(js|ts|md)x?$/] },

        use: [

          {

            loader: "@svgr/webpack",

          },

        ],

      });

      return config;

    },

  };

Создаем новый файл custom-next-env.d.ts, где нужно задекларировать модуль:

declare module "\*.svg" {

  const content: React.FC<React.SVGAttributes<SVGAElement>>;

  export default content;

}

После этого в файле next-env.d нужно поправить tsconfig.json, где вместо next-env.d.ts (который нельзя модифицировать) включить наш новый custom-next-env.d.ts

  "include": [

    "custom-next-env.d.ts",

    "\*\*/\*.ts",

    "\*\*/\*.tsx"

  ],

  "exclude": [

    "node\_modules",

    "next-env.d.ts"

  ]

После этого нужно установить зависимость в проект: npm I –D @svgr/webpack

Всё.

Поменять цвет можно в модуле css

.primary svg,

.ghost:hover svg {

  fill: var(--white);

}

## **tabIndex и onKeyDown**

Для установки на SVG компоненте события по таб нужно сделать так:

tabIndex={() => { isEditable ? 0 : -1; }}

Для установки на SVG компоненте события по нажатию пробела нужно сделать так:

- импортировать KeyboardEvent из React

import { useEffect, useState, KeyboardEvent } from 'react';

- назначить событие так

onKeyDown={(e: KeyboardEvent<SVGAElement>) => isEditable && handleSpace(i + 1, e)}

  const handleSpace = (i: number, e: KeyboardEvent<SVGAElement>) => {

    if (e.code !== "Space" || !setRating) {

      return;

    }

    setRating(i);

  };

Для плавного перехода мышью по SVG нужно SVG-компонент обернуть в <span> и перенести на него все классы и события кроме tabIndex и onKeyDown.

  const constructRating = (currentRarting: number) => {

    const updatedRating = ratingArray.map((r: JSX.Element, i: number) => {

      return (

        <span

        className={cn(styles.star, {

          [styles.filled]: i < currentRarting,

          [styles.editable]: isEditable

        })}

        onMouseEnter={() => changeDisplay(i + 1)}

        onMouseLeave={() => changeDisplay(rating)}

        onClick={() => onClick(i + 1)}

        >

        <StarIcon

          tabIndex={isEditable ? 0 : -1}

          onKeyDown={(e: KeyboardEvent<SVGAElement>) => isEditable && handleSpace(i + 1, e)}

          />

          </span>

      );

    });

    setRatingArray(updatedRating);

  };

# Хуки

## Проверка хуков с eslint

Устанавливаем плагин: npm i eslint-plugin-react-hooks --save-dev

Добавляем в .eslintrc "plugin:react-hooks/recommended"

  "extends": [

    "eslint:recommended",

    "plugin:@typescript-eslint/eslint-recommended",

    "plugin:@typescript-eslint/recommended",

    "plugin:react-hooks/recommended"

  ]

}

## UseState

Если в стейте нужен массив JSX элементов, например таких , то

Записываем таким образом

const [ratingArray, setRatingArray] = useState<JSX.Element[]>(new Array(5).fill(<></>));

Здесь в стейте для быстрого заполнения прописываем newArray, указываем количество элементов, и чем он должен быть заполнен (фрагментами).

# HOC-компонент

## Типизированный HOC-компонент

export const HOCexample = <T extends Record<string, unknown>>(Component: FunctionComponent<T>) => {

  return function WithWrapperComponent(props: T) {

    return (

      <div className= 'hoc-wrapper'>

      <Component {...props} />

      </div>

    );

  }

}

Запись T extends Record<string, unknown>> - это тип пропсов передаваемого компонента. Мы могли указать вместо Record<string, unknown> просто объект { } или Obj, но это более правильная запись любого объекта.

Пример обертки из проекта:

const Layout = ({ children }: LayoutProps): JSX.Element => {

  return (

    <>

      <Header />

      <div>

        <Sidebar />

        <main>{children}</main>

      </div>

      <Footer className={styles.footer} />

    </>

  );

};

export const withLayout = <T extends Record<string, unknown>>(Component: FunctionComponent<T>) => {

  return function withLayoutComponent(props: T): JSX.Element {

    return (

      <Layout>

        <Component {...props} />

      </Layout>

    );

  };

};