Next.js

Stepan Rutz

30.11.2023

Next.js

- React + Serverside Rendering (SSR) + Static Website Generation
- Based on
  - HTML
  - Javascript
  - Typescript
  - JSX
  - React
  - CSS
  - Webpack
  - HTTP

Learning resources

- Javascript/CSS/HTML
  - Javascript
  - CSS
  - HTML
- Typescript
  - Typescript
  - Typescript-Playground
- React / JSX
  - React.dev
- Next.js
  - Next.js Documentation
  - Next-Learn
- Data
  - Dummy Json Data

Good Tutorials and Podcasts

Tutorials

- Net Ninja Modern React
- 12 React mistakes
- CSS Flexbox
- CSS Grid

Podcasts

- Syntax
- Podrocket

Alternativen

- Zu React ... Angular, Vue.js, SvelteKit, Solid.js

Erweiterungen, Gui Libs

- Rest/Http: Axios
- UI: Carbon, Chakra-UI, React-Bootstrap, TailwindCSS, Mantine, Bulma
- Fonts,Icons: React-Icons, Google-Fonts
- Testing: Jest, Cypress, Vitest
- Development: Eslint, Prettier

Workshop 1, Typescript und React

Inhalte:

1. Komponenten in React und JSX erstellen
2. CSS-Klassen mit CSS-Modules
3. Wiederholung
   - Destructuring, Spread und Rest
   - ?: Operator
   - && und ??
   - Arrays map, reduce, find
4. Typsichere Props (ReactNode, PropsWithChildren)
5. Typsichere Styles (CSSProperties)
6. Typisierung von States (Primitives und Objekte) (useState)
7. Typisierung von Event-Handlern
8. Typisierung von References (useRef)

Typsichere Props

```
import { ReactNode } from "react"
export type ExampleProps = {
    stringValue: string
    numericValue: number
    booleanValue: boolean
    optionalValue?: string
    children: ReactNode
}
export function Example(props: ExampleProps) {
    const {
        stringValue,
        numericValue,
        booleanValue,
        optionalValue = "standard" // initialize default
    } = props
    return (
        <div>stringValue={stringValue}...
            <div>{props.children}</div>
        </div>
    )
}
```

Typsichere Events

```
import { MouseEvent } from "react"

export function Example() {
    const divclick = (e: MouseEvent<HTMLDivElement>) => {
        const nativeEvent = e.nativeEvent
    }

    return (
        <div onClick={divclick}>Click me</div>
    )
}
```

Typsichere Refs

```
import { useEffect, useRef } from "react"

export function Example() {

    const myref = useRef<HTMLDivElement>(null)

    useEffect(() => {
        console.log(myref.current)
    }, [])

    return (
        <div ref={myref}>Click me</div>
    )
}
```

Typsichere State-Objekte

```
import { MouseEvent, useState } from "react"

type UserForm = {
    firstname: string; lastname: string; email: string
}

export function Example() {
    const[userForm,setUserFrom] = useState<UserForm>()
    const onDivClick = (e: MouseEvent<HTMLDivElement>) => {
        const newUserForm: UserForm = {
            firstname: "Karl",
            lastname: "Hansen",
            email: "hans@karl.de"
        }
        setUserFrom(newUserForm)
    }
    return (<div onClick={onDivClick}>
        Email={userForm?.email??"nicht gesetzt"}</div>)
}
```

Standalone React Projekte erstellen

Wenn man etwas einfach testen will oder nur React benötigt, dann empfiehlt sich Vite. Hier Beispiel für Typescript

```
npm create vite@latest \
  001_helloworld -- --template react-ts
cd 001_helloworld
npm install
```

danach mit HMR entwickeln:

```
npm run dev
```

Next.js

Was ist SSR und Next.js in a nutshell

Pseudocode

```
/* handle request on the server */
const app = express()
app.handleRequest(request => {
    const s = loadScript(request)
    if (is_serverside(s)) {
        /* here all of node.js is available
         * but nothing from the browser (=client) */
        return runAndRenderIntoString(s)
    } else {
        /* script is not executed but send to the
         * browser (=client) */
        return s
    }
})
```

Workshop 2, Next.js

Anlegen einer Next.js Anwendung mit

```
npx create-next-app@latest
```

```
npx next telemetry status
```

```
npx next telemetry disable
```

Agenda Next.js Basics

- CSS Modules verwenden
- Routing
  - Erstellen eines Layouts (Flexbox)
  - Nagivation erstellen
  - Links verwenden
  - Routen definieren
  - Parallel Routes
  - Intercepting Routes
  - Special Pages (Error, Loading etc)
- Icons einbinden
- Font einbinden
- Bilder einbinden
- APIs mittels route.ts

Workshop

Step1:

- Anwendung leeren
  - Löschen der Grafiken aus public/
  - Löschen des Inhaltes von page.tsx und ersetzen durch einfaches Headertag (oder ähnlich)
  - page.module.css löschen
  - Löschen des Inhalts von global.css
- Stand: Branch: 01_empty_everything

Step2:

- Ersetzen des Inhalts von global.css durch

`https://static.stepanrutz.com/nextjs/global.css`

`und von Layout.tsx durch`

`https://static.stepanrutz.com/nextjs/layout.tsx`

- Metadata anpassen
- Stand: Branch: 02_simple_layout

Step3:

- Weitere Routen anlegen

`src/app/imprint/page.tsx`
`src/app/contact/page.tsx`
`src/app/about/page.tsx`

- Links einbauen in Client-Side Component Dazu die
  durch eine eigene Componente ersetzen und dort Links einbauen.
- Stand: branch: 03_links

Step4:

- Aktive Links mit einer extra Klasse versehen und diese anders rendern.
- Mittels
  `const pathname = usePathname()` auf den aktuellen Pfad zugreifen und ggf. eine andere Klasse setzen.
- Stand: branch: 04_links_with_style

Step5:

Middleware hinzufügen. Diese ist für - Redirects - Loggin - Authentifizierungen

Neue Datei middleware.ts

- Stand: branch: 05_middleware

Step6:

Das Tag

`<Image>`

benutzen . . .
Achtung: Das ist kompliziert. Einbinden von Image und ImageWrapper

- Stand: branch: 06_image

Step7:

- Fonts.
- Öffnen der Datei

`next/dist/compiled/@next/font/dist/google`

und einfach Font auswählen und importieren.
In Layout.tsx konfigurieren und aktivieren

- Stand: branch: 07_font

Step8:

- Plugins installieren.
  Install mit npm install
  und einbinden in einer beliebigen (Client-)-Komponente
- Stand: branch: 08_youtube

Step9:

- Posts von dummyjson.com laden
- Neue Route /posts definieren in
- Ggf. Routen deklarativ rendern
- Die neue Route lädt die Posts clientseitig
- Layout für diese Route in page.module.css definieren
- Stand: branch: 09_posts

Step10:

- Dynamic Routes
- Anlegen einer client-Component für einzelne Posts
- Links zu den einzelnen Posts hinzufügen.
- Dynamische Route anlegen /posts/[id]
- Style für die neue Post-Komponente
- Post-Typedefinition in eigene Datei
- In der dynamischen Route die neue Komponente aufrufen
- (Aufs Caching eingehen)
- Stand: 10_dynamic_routes

Workshop 2 (continued), Next.js

- Suspense (and fallback)
- Caching (prerendered static content)

  ```
  export const dynamic = "force-dynamic"
  fetch option next.cache: "no-cache"
  fetch option next.revalidate: 3600
  ```
- There are 2 caches on the server
  - Full route cache
  - Data caches
- and 1 Cache for Routes on the client

Workshop 3, Next.js und React

- Interaktion (useState, useEffect)
- Warenkorb implementieren (useContext)
- 3rd Party GUI Komponenten einbinden
- Modale Dialogs, Toasts

Workshop 3, Next.js + Carbon

- Serverside Rendering
- Serverside Datafetching
- Markdown

How to add carbon

Carbon benutzt SASS

Projekt anpassen:

```
npm i --save-dev sass
npm install -S @carbon/react
```

Styles inkludieren:

globals.css umbenennen zu **globals.scss** und oben folgendes einfügen:

```
@use '@carbon/react';
```

Und je nach Komponente eine Client-Boundary schaffen (`"use client"`)

Workshop 3, Step 1

Standard Next.js Projekt leeren

- Start with Default next.js App (Typescript, ESLint, App Router, Src Directory, Default Alias). Skip Tailwind
- Empty globals.css, replace with contents from https://static.stepanrutz.com/nextjs/global.css
- Empty page.tsx, add some small defaults
- Delete page.module.css
- Replace contents of layout.tsx with https://static.stepanrutz.com/nextjs/layout.tsx
- Delete SVG images in public/
- Branch for above: 01_empty_nextjs

Workshop 3, Step 2

##Carbon hinzufügen

- Install deps

```
npm i --save-dev sass\n
npm i -S @carbon/react
```

- Rename globals.css to globals.scss and change import in layout.tsx
- Branch for above: 02_dependencies

Workshop 3, Step 3

Carbon Navbar implementieren

- Style für “nav” Element löschen
- Top-Margin von 48px für das “main” Element setzen
- Compile error in ./node_modules/@carbon/icons-react/es/generated/bucket-8.js
- Neue Route /imprint anlegen (neues File src/app/imprint/page.tsx)
- Narbar.tsx anlegen und in layout.tsx einbinden anstelle von altem
- Branch for above: 03_carbon_navbar

Workshop 3, Step 4

Carbon Datatable

- Neue page /datatable anlegen und in der Navbar verlinken
- Style für die neue Page definiere mit overflow-y: auto;
- Code aus

`https://react.carbondesignsystem.com/?path=/story/components-datatable-`
einfügen.
Rows und Headers wie folgt:

```
const headers = [ { header: 'Name', key: 'name', },
 { header: 'Age', key: 'age', },
 { header: 'Location', key: 'location', }, ];

const templateRow = { id: '_', name: 'John Doe', age: 30, location: 'Ne
const rows: typeof templateRow[] = []
rows.push(row)
```

Workshop 4, Next.js

- APIs mittels route.ts (revisited)
- Serverside Rendering
- Serverside Datafetching
- Markdown
- Testing

How to add Jest

Das Testframework JEST zu einer bestehenden Next.js Anwendung hinzufügen funktioniert wie folgt:

- Dev-Dependencies hinzufügen

```
npm install --save-dev @testing-library/jest-dom \
  @testing-library/react \
  @testing-library/user-event \
  @types/jest jest jest-environment-jsdom
```

- Jest Config files anlegen

jest.config.ts

Inhalt von jest.config.ts im Project-root

```
const nextJest = require('next/jest')
const createJestConfig = nextJest({
  dir: './',
})
const customJestConfig = {
  setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
  testEnvironment: 'jest-environment-jsdom',
}
module.exports = createJestConfig(customJestConfig)
```

jest.setup.is

Inhalt von jest.setup.js im Project root

```
import '@testing-library/jest-dom'
```

und danach Test files anlegen inkl. JSX. Als Tests werden folgende Files erkannt:

- *.test.js*, .test.jsx, *.test.ts*, .test.tsx

Ausführen von Tests

Die Tests ausführen mit

```
npx jest
```

Beispiel Test mit JSX:

```
import '@testing-library/jest-dom'
import {cleanup, fireEvent, render, screen}
  from '@testing-library/react';
import MenuEntryCard from "@/app/menu/MenuEntryCard";
import {MenuEntry} from "@/data/menuentry";
import {CartProvider} from "@/components/CartProvider";

test("reacttest", () => {
    const entry: MenuEntry = {
        key: "one", description: "hello",
        label: "huhu",
        price: 100, image: "abc.jpg"
    }
    render(
        <CartProvider>
            <MenuEntryCard entry={entry} fadeInDelay={100} />
        </CartProvider>
    )
    const element = screen.getByText('hello')
    expect(element).toBeInTheDocument()
})
```