Week 09: Routing & Forms

Agenda

- Client-side routing
- Advanced forms
- Demo

Client-side routing

- In SPA applications we need **fake** routing
- We want all features of the old-style routing (search params, ..)
- The most used solution is **React Router DOM**

React Router DOM

- Definition of the routes with **Route Objects** or Route Elements
- Dynamic routes with `useParams` hook to access the parameters
- Nested routes with `Outlet` and layouts
- `Link` and `Navlink`
- Search parameters with `useSearchParams` hook
- Though there are other types, use `BrowserRouter` (definitely in this subject)

React router DOM - Route Object

Routes definition with `RouteObject`:

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";

const router = createBrowserRouter([
  {
    path: "/preferences",
    Component: PreferencesPage,
  },
  {
    path: "/products",
    Component: ProductListPage,
  },
]);

const App: FC = () => {
  return <RouterProvider router={router} />;
};
```

React router DOM - Nested routes

```
const router = createBrowserRouter([
  {
    path: "/preferences",
    Component: PreferencesLayout,
    children: [
      { path: "users", Component: UsersPreferences },
      { path: "products", Component: ProductsPreferences },
    ],
  },
]);
```

```
const PreferencesLayout: FC = () => (
  <div>
    <PreferencesNavbar />
    <Outlet /> {/* imported from React router dom */}
  </div>
);
```

React router DOM - Dynamic routes

```
const router = createBrowserRouter([
  {
    path: "/products/:productId",
    Component: ProductDetail,
  },
]);
```

```
const ProductDetail: FC = () => {
  const { productId } = useParams(); /* imported from React router dom */
  const { product } = useProductDetail(productId);

  return; /* html part with product detail */
};
```

Need for **parsing params** and checking if they are **valid**.

React router DOM - Links

- `Link` replaces `<a>`
- `NavLink` is suitable for navigation such as menu or navbar

```
import { Link } from "react-router-dom";

const SomeComponent: FC = () => {
  return (
    <div>
      {/* Absolute path */}
      <Link to="/preferences">Go to preferences</Link>

      {/* Relative path */}
      <Link to="../products" relative="path">
        Go to products
      </Link>
    </div>
  );
};
```

React router DOM - Navigate

```
import { Navigate } from "react-router-dom";

const UserInfo: FC = () => {
  const { user } = useUser();

  if (typeof user === 'undefined') {
    // Redirect
    return <Navigate to="/" />;
  }

  return (
    <>{user.name}</>
  );
};
```

React router DOM - Navigate programmatically

```
import { useNavigate } from "react-router-dom";

const OrderDetails: FC = () => {
  const navigate = useNavigate();

  const onSubmit = () => {
    // Redirect
    navigate('../confirmation', { relative: 'path' });
  }

  return (
    <>Some content</>
  );
};
```

Advanced forms

- In React we typically want **interactiveness**
- We would like to inform users about the form's **invalid data**
- There are **controlled** and **uncontrolled** inputs

Controlled inputs

```
const ProductForm: FC = () => {
  const [value, setValue] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log("Submitted form with: ", value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input value={value} onChange={(e) => setValues(e.target.value)} />
      <button>Submit</button>
    </form>
  );
};
```

Uncontrolled inputs

```
const ProductForm: FC = () => {
  const inputRef = useRef<HTMLInputElement>();

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log("Submitted form with: ", inputRef.current?.value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input ref={inputRef} />
      <button>Submit</button>
    </form>
  );
};
```

React hook form

- Can abstract the work with the form
- Helps us with easy data validation
- Has performance and can be used on large forms
- Provides `useForm` hook
- Typescript support and data validation
- Provides `Controller` for complicated inputs
- **9/10 tutors** recommend using this library :-).

React hook form - Input registration

```
import { useForm } from "react-hook-form";

const ProductForm: FC = () => {
  const { register, handleSubmit } = useForm();

  const submitHandler = (values) => {
    console.log("Submitted form with: ", inputRef.current.value);
  };

  return (
    <form onSubmit={handleSubmit(submitHandler)}>
      <input {...register("title")} />
      <button>Submit</button>
    </form>
  );
};
```

React hook form - Typescript

```tsx
type CreateProduct = {
  title: string;
  description: string;
};

const ProductForm: FC = () => {
  const { register, handleSubmit } = useForm<CreateProduct>();

  const submitHandler: SubmitHandler<CreateProduct> = (values) => {
    console.log("Submitted form with: ", values.title, values.description);
  };

  return; /* form */
};
```

React hook form - Validation

We can use **zod** (you already know) and **zodResolver**:

```
const productSchema = z.object({ title: z.string().min(3) });
type CreateProduct = z.infer<typeof productSchema>;

const ProductForm: FC = () => {
  const { register, handleSubmit, formState } = useForm<CreateProduct>({
    resolver: zodResolver(productSchema),
  });

  return (
    <form onSubmit={handleSubmit(submitHandler)}>
      <input {...register("title")} />
      {formState.errors.title && <p>{formState.errors.title.message}</p>}
      <button>Submit</button>
    </form>
  );
};
```

React hook form - Controller

Suitable when input is **complicated** or the library has **limited API**:

```
const ProductForm: FC = () => {
  const { control, handleSubmit } = useForm<CreateProduct>();

  return (
    <form onSubmit={handleSubmit(submitHandler)}>
      <Controller
        control={control}
        name="deliveryDate"
        render={({ field }) => (
          <ReactDatePicker onChange={field.onChange} selected={field.value} />
        )}
      />
      <button>Submit</button>
    </form>
  );
};
```

Let's code!

- The assignment zip can be found in the interactive syllabus.
- All the necessary packages are **already installed** for you.

That's it for today!