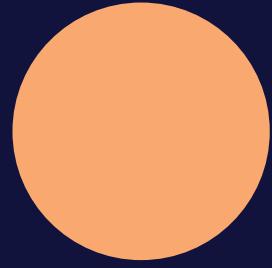


PRACTICAL REACT WITH TYPESCRIPT

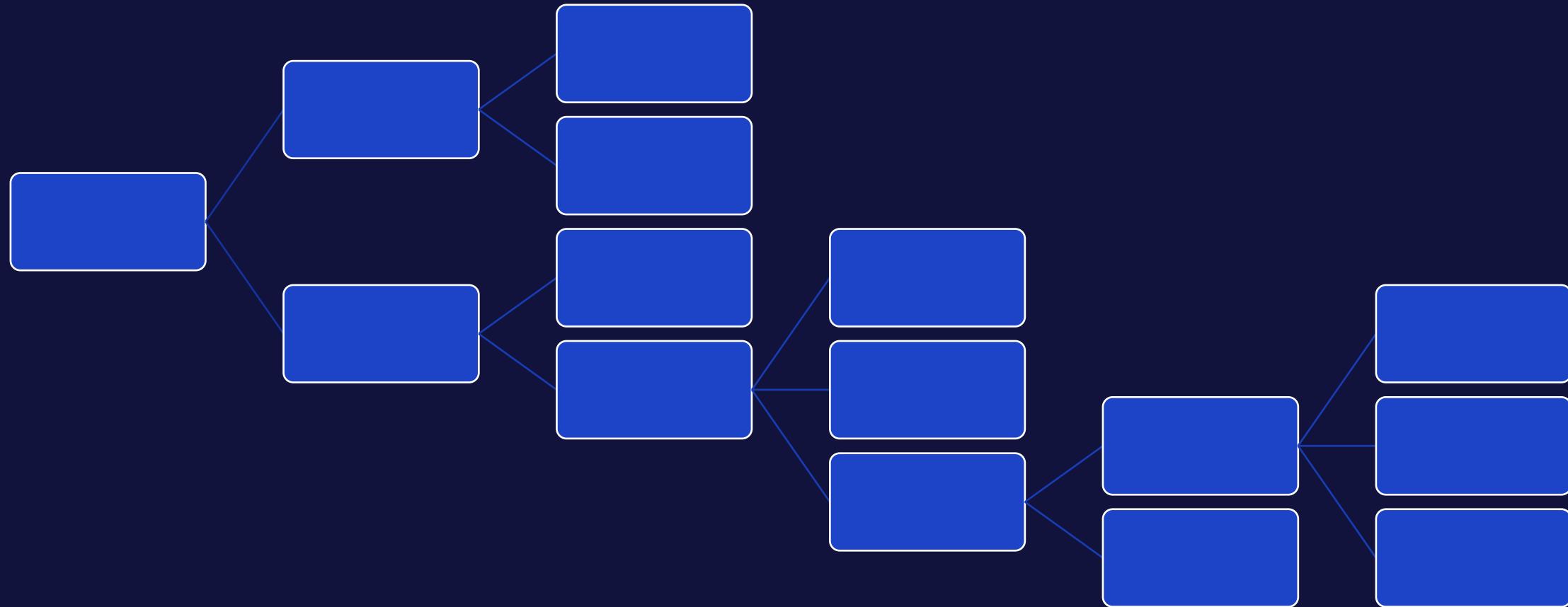
bouvet



Setup

Agenda

Anatomy of React



<> TextField



-
-
-
-

<> BooleanField

•

•

•



<> Reusable fields



.

Anatomy of a component

```
export interface TextFieldProps {  
  label: string  
}
```

props

```
export const TextField = ({ label }: TextFieldProps) => {  
  const id = useId()  
  const [value, setValue] = useState("")  
  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```

component

```
export interface TextFieldProps {  
  label: string  
}  
  
export const TextField = ({ label }: TextFieldProps) => {  
  const id = useId()  
  const [value, setValue] = useState("")  
  
  return (  
    <div>  
      <label
```

Anatomy of a component

```
export interface TextFieldProps {  
  label: string  
}  
use*                                     hooks  
  
export const textField = ({ label }: TextFieldProps) => {  
  const id = useId()  
  const [value, setValue] = useState("")  
  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```

Anatomy of a component

```
export interface TextFieldProps {  
  label: string  
}  
export default function TextField({label, id}: TextFieldProps) => {  
  const [value, setValue] = useState("")  
  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```

Anatomy of a component

```
export interface TextFieldProps {  
  label: string  
}  
  
export const TextField = ({ label }: TextFieldProps) => {  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```

jsx

Anatomy of a component

```
export interface TextFieldProps {  
  label: string  
}  
  
export const TextField = ({ label }: TextFieldProps) => {  
  const id = useId()  
  const [value, setValue] = useState("")  
  
  return (children  
    <div>  
      <label htmlFor={id}>label</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```



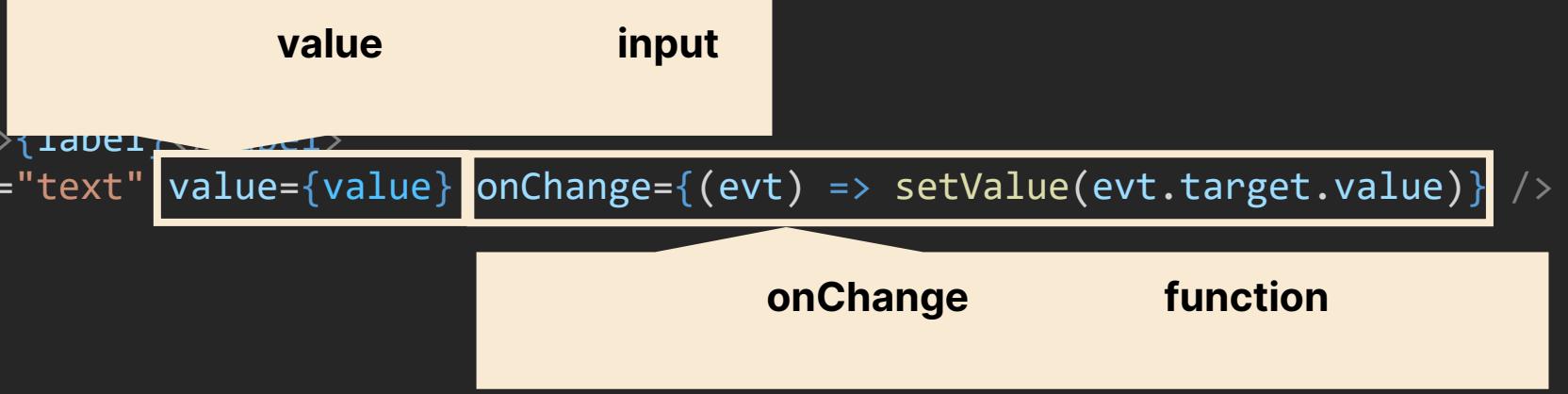
children	label	label	TextField
children	label	label	TextField

Anatomy of a component

```
export interface TextFieldProps {  
  label: string  
}  
export const [value, setValue] = useState("")  
const id = useState()  
useState  
destructure  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```

Anatomy of a component

```
export interface TextFieldProps {  
  label: string  
}  
  
export const TextField = ({ label }: TextFieldProps) => {  
  const id = useId()  
  const [value, setValue] = useState("")  
  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```



The code defines a `TextField` component that takes a `label` prop. It uses `useId` and `useState` to generate a unique ID and store the current value of the text input respectively. The component returns a `div` containing a `label` and an `input` element. The `input` has its `value` set to the current state and its `onChange` event handler set to a function that updates the state when the value changes.

Anatomy of a component

- **Component:**
- **Props**
- **Hooks**
- **State**
- **Children**
- **JSX**
- **{}**

JSX

Anatomy of an event

```
export interface TextFieldProps {  
  label: string  
}  
  
export const TextField = ({ label }: TextFieldProps) => {  
  const id = useId()  
  const [value, setValue] = useState("")  
  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)}>  
      <p>{value}</p>  
    </div>  
  )  
}
```



input*

input

onChange

input

Anatomy of an event

```
export interface TextFieldProps {  
  label: string  
}  
  
export const TextField = ({ label }: TextFieldProps) => {  
  const id = useId()  
  const [value, setValue] = useState("")  
  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange-  
        {evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```



The diagram illustrates the flow of an event. A curved arrow originates from the `onChange` prop in the `TextField` component's render function and points to the `evt` parameter in the `onChange` handler. This handler is enclosed in a yellow box labeled "event handler". Inside this box, another arrow points from the `evt` parameter to the `setValue` call, which is labeled "setter".

Anatomy of an event

```
export interface TextFieldProps {  
  label: string  
}
```

State change

re-render
updated data

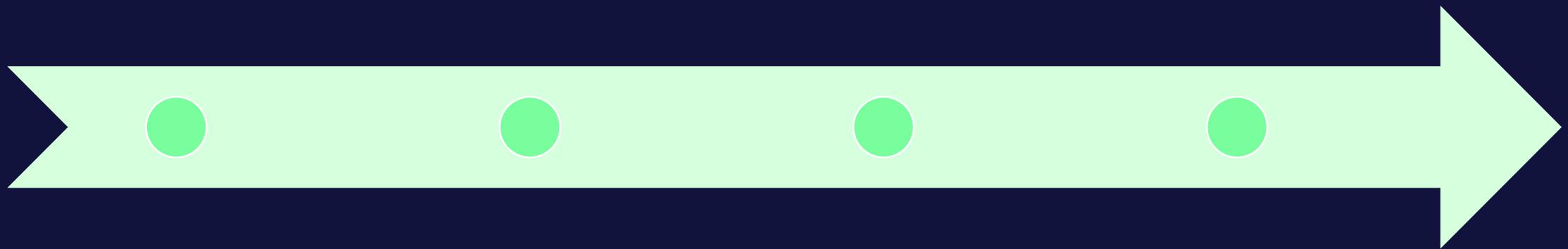
```
export const TextField = ({ label }: TextFieldProps) => {  
  const id = useId()  
  const [value, setValue] = useState("")  
  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```

Anatomy of an event

```
export interface TextFieldProps {  
  label: string  
}  
  
export const TextField = ({ label }: TextFieldProps) => {  
  const id = useId()  
  const [value, setValue] = useState("")  
  
  return (  
    <div>  
      <label htmlFor={id}>{label}</label>  
      <input id={id} type="text" value={value} onChange={(evt) => setValue(evt.target.value)} />  
      <p>{value}</p>  
    </div>  
  )  
}
```

The diagram illustrates the state flow in a React component. A curved arrow originates from the `value` prop in the `TextFieldProps` interface and points to the `value` state variable in the `TextField` component's code. Another curved arrow originates from the `value` state variable and points to the `value` prop of the `<input>` element. A callout box labeled "value" points to the `value` prop of the `<input>` element. Another callout box labeled "updating the UI" points to the `onChange` handler, indicating that changes to the input field trigger an update to the UI state.

Anatomy of an event

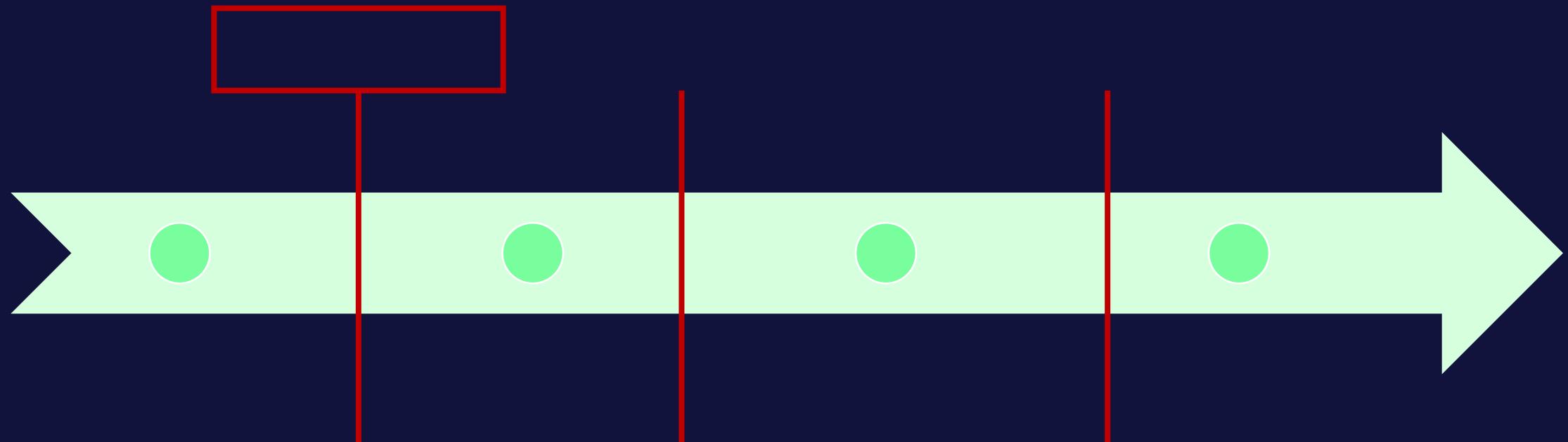


Component lifecycle

-
-
-
-
-
-

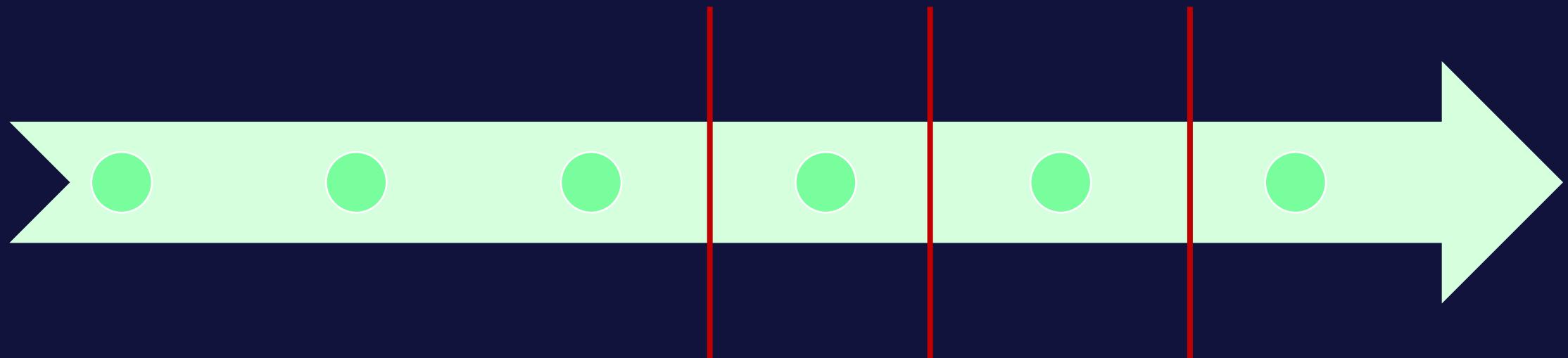
Component lifecycle

TextField



Component lifecycle

TextField



<> ClickUntil

•

•

•

•

•

•

<> NumericField



<> ClickUntilForm



Styling

-
-
-
-

<> Style TextField



.

<> Style components

-
-
-

EoD 1

Organizing our repository

- • • • • • • • •

Immutability

-
-



< > UserDetails

- 卷之三

<> GroupDetails

-
-
-

Loops



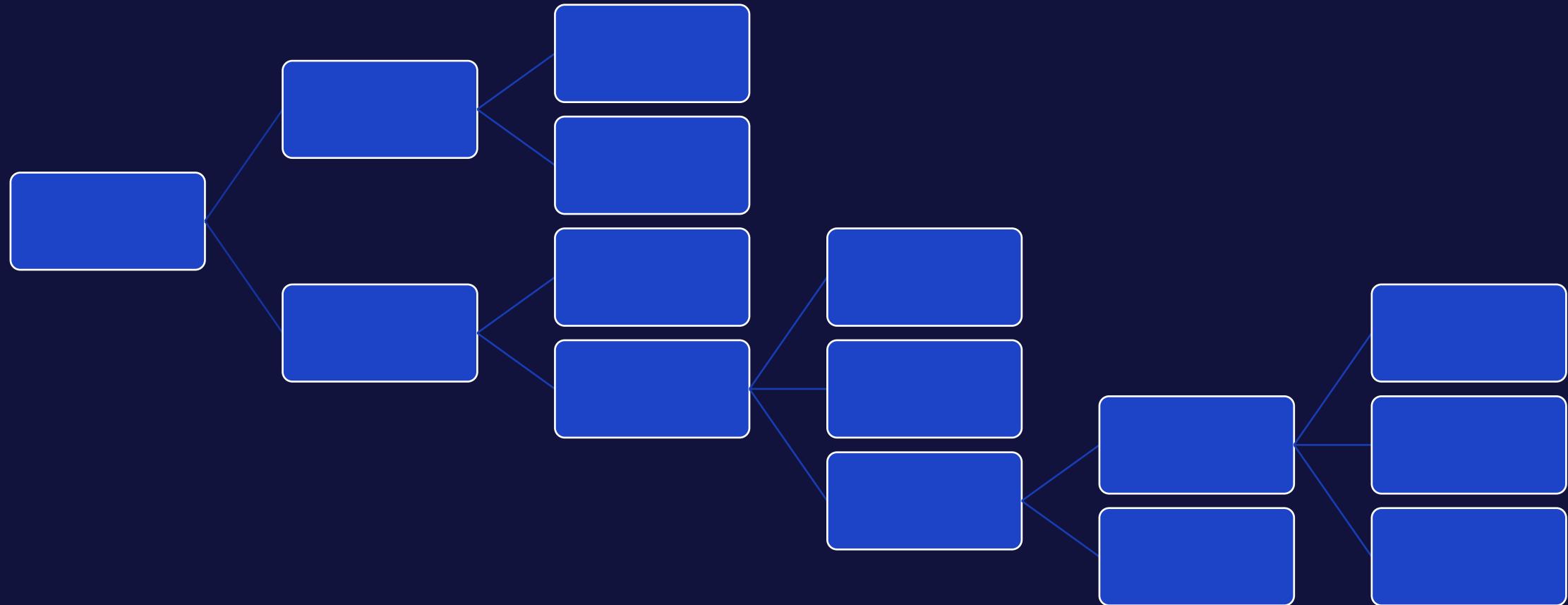
useMemo

.

Routing

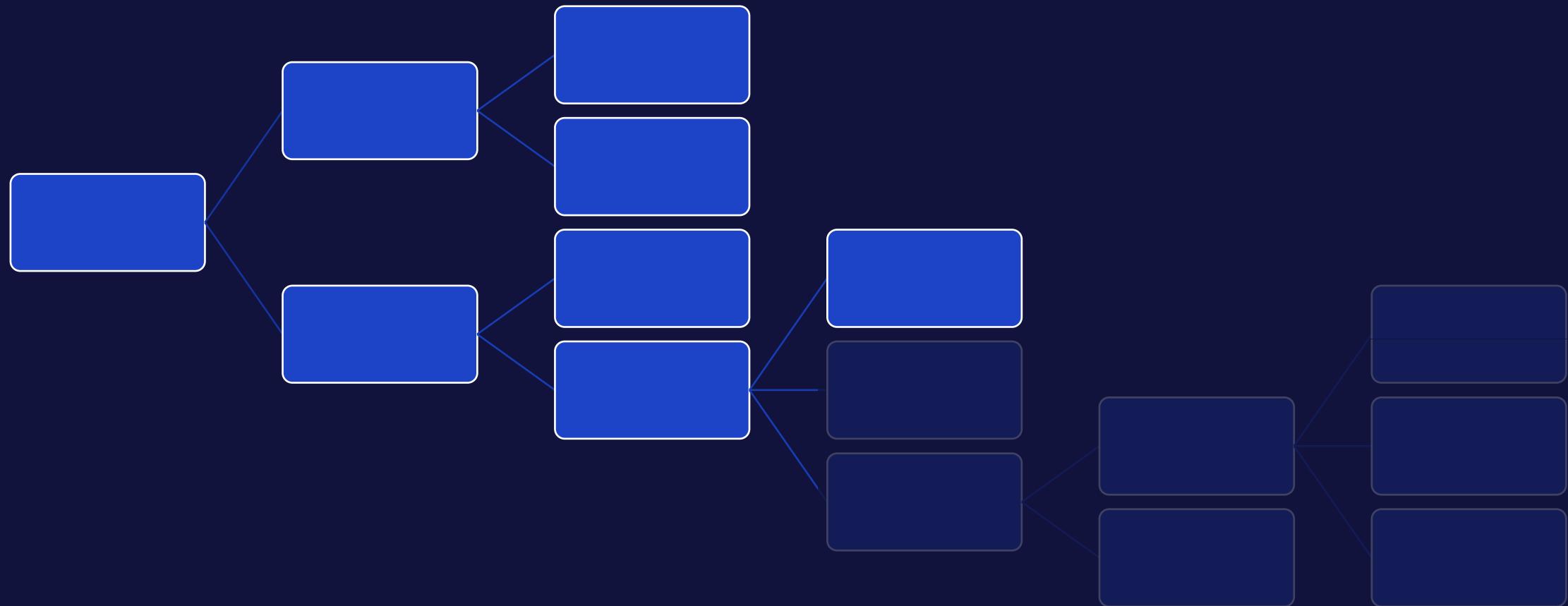
-
-

Anatomy of routing



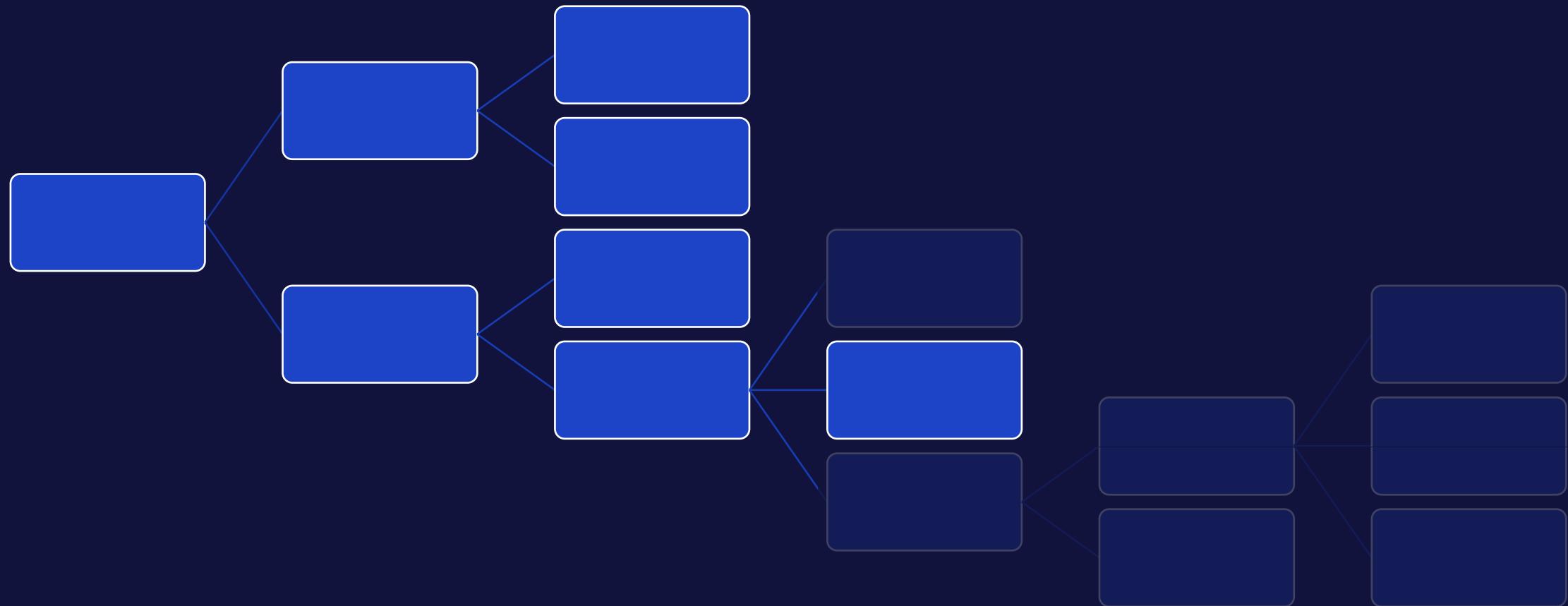
Anatomy of routing

/home



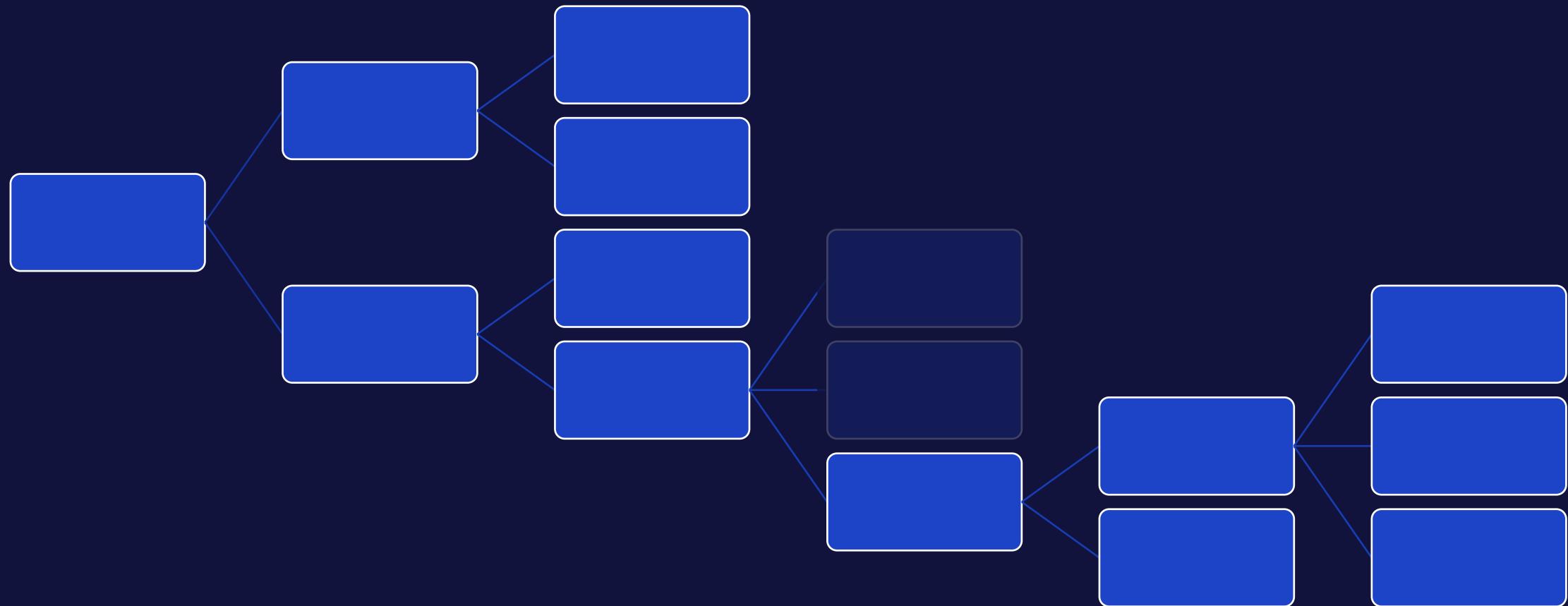
Anatomy of routing

/login



Anatomy of routing

/profile



<> HomePage and UserDetailsPage



-
-
-
-

<> MainLayout



•

•

<> UsersTable Page and UserDetails Page

-
-
- /users
-

EoD 2

Hooks



Most common react hooks

- **useId:**
- **useState:**
- **useMemo:**
- **useEffect:**
- **useRef:** **does not**

<> useRandom



.

<> useWindowTitle

•

•

Error handling

-
- errorElement

<> useThrowIfNetworkUnavailable

-
-

TypeScript Generics

.

TypeScript Generics

```
interface ContainerFor<TType> {  
  id: string  
  value: TType  
}
```

```
const stringContainer: ContainerFor<string> = {  
  id: "123",  
  value: "foo"  
}  
  
const booleanContainer: ContainerFor<boolean> = {  
  id: "123",  
  value: true  
}  
  
interface Name {  
  firstName: string  
  lastName: string  
}  
const objectContainer: ContainerFor<Name> = {  
  id: "123123",  
  value: {  
    firstName: "Test",  
    lastName: "Testington"  
  }  
}
```

TypeScript Generics

```
interface ContainerFor<TType extends string | boolean> {  
  id: string  
  value: TType  
}
```

```
const stringContainer: ContainerFor<string> = {  
  id: "123",  
  value: "foo"  
}  
  
const booleanContainer: ContainerFor<boolean> = {  
  id: "123",  
  value: true  
}
```

TypeScript Generics

```
interface GenericInterface<T> {
  id: string
  value: T
}

type GenericType<T> = {
  id: string
  value: T
}

const genericFn = <T, T2>(firstArg: T, secondArg: T2) => {
  //...
}

function genericFn2<T, TReturn>(firstArg: T): TReturn {
  //...
}
```

TypeScript Generics with React

```
export interface GenericComponentProps<TType> {
  label: string
  value: TType
}

export const GenericComponent = <TType,>(props: GenericComponentProps<TType>) => {
  // ..
}
```

<> ChoiceField

•

•

•

•

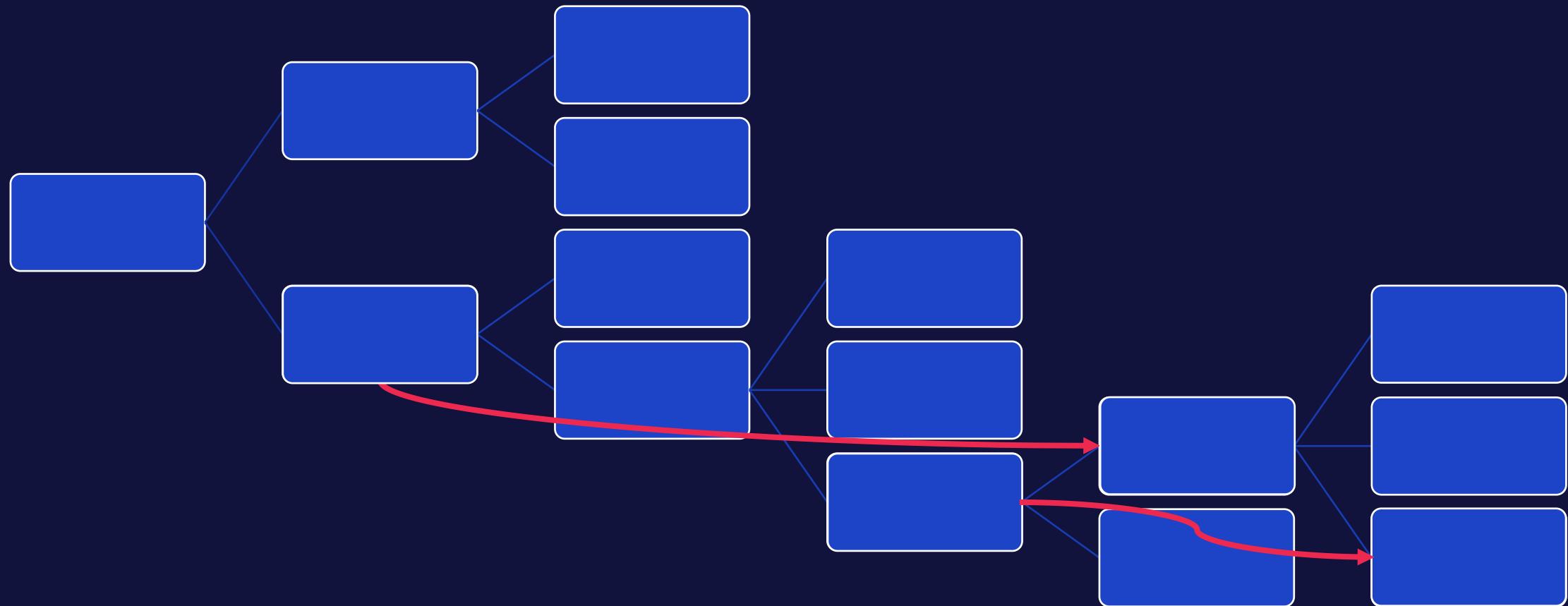
<> ChoiceField

-
-
-
-
-

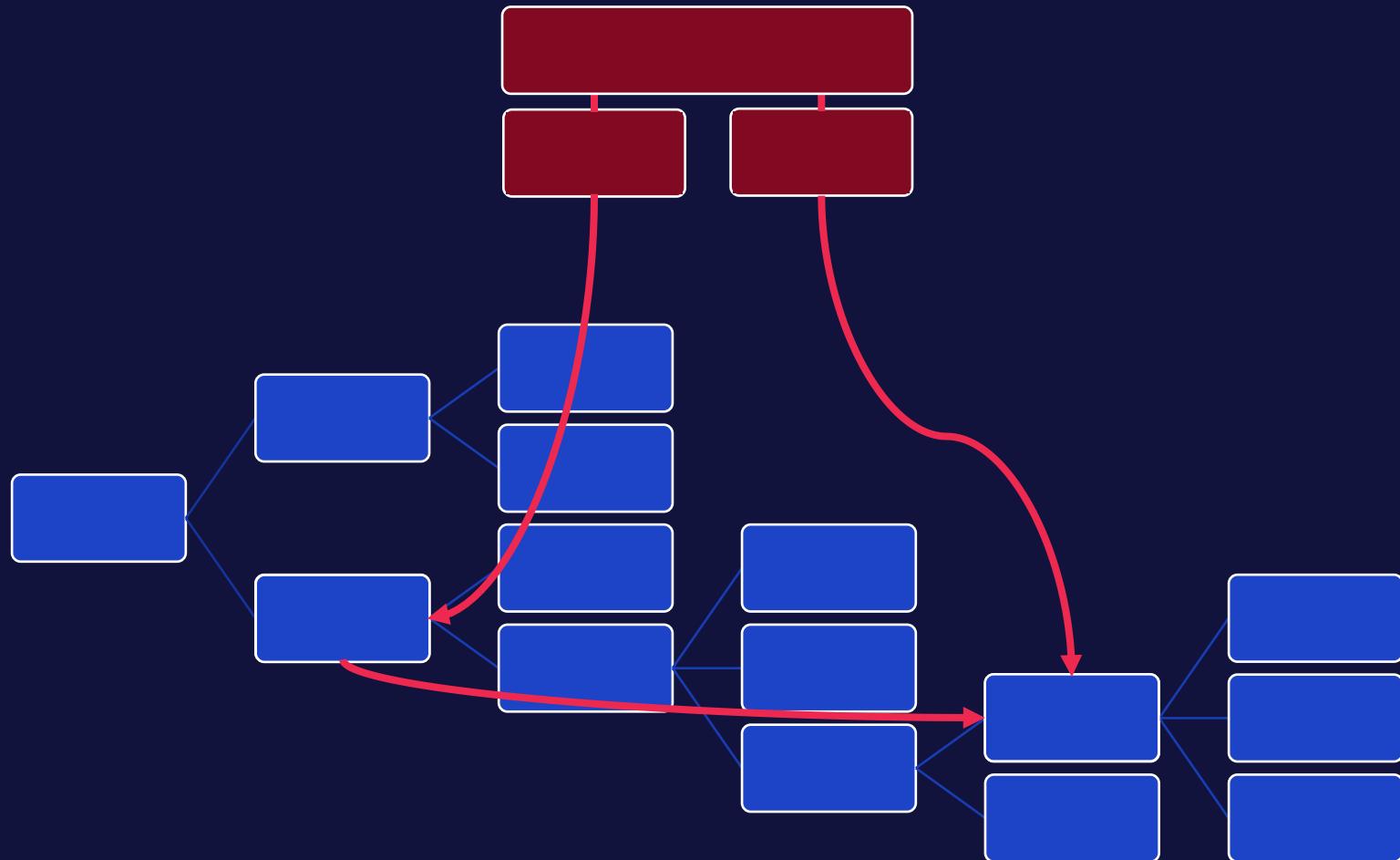
Contexts



Anatomy of Context



Anatomy of Context



<> NavContext



.

<> DisableFieldsContext

-
-
-
-

State management scopes

-
-
-
-

Communication with a server

-
-
-

Generating clients

-
-
-

<> ServerSideUserTable

-
-
-
-

UserDetails



-
-
-
-

Testing



Optimizations

- **Avoid unnecessary re-renders**

-
-
-
-
-
-
-
-
-

Resources

- • • • • • • • •