NEWS · OCTOBER 30, 2025

React Components: Das Web UI Model verstehen (Teil 3 von 5)

By Dr. Holger Flick

Der konzeptionelle Wandel

Erinnern Sie sich an das erste Mal, als Sie verstanden haben, wie Delphis Komponentenmodell funktionierte? Sie konnten einen `TButton` auf ein Form ziehen, seine Properties setzen, einen `OnClick` Handler schreiben, und plötzlich hatten Sie ein funktionierendes User Interface. Komponenten waren wiederverwendbar. Sie konnten eigene Komponenten erstellen und sie überall verwenden. Die VCL machte Desktop-Entwicklung intuitiv.

React ist dieselbe Idee für Web-Anwendungen. Anstatt Desktop Controls haben Sie Web Components. Anstatt Forms haben Sie Pages. Aber das fundamentale Konzept—Interfaces aus zusammensetzbaren, wiederverwendbaren Teilen zu erstellen—ist identisch.

Dieser Artikel handelt nicht davon, Ihnen beizubringen, React Code zu schreiben. Es geht darum, Ihnen zu helfen, das mentale Model zu verstehen, damit Sie bewerten können, ob Ihre Delphi-Anwendungen zu diesem Ansatz übertragen werden können.

Komponenten verstehen

In Delphi könnten Sie ein kundenspezifisches Panel mit einem Label und Button erstellen:

```
type
  TCustomerPanel = class(TPanel)
  private
    FCustomerName: string;
    FLabelCaption: TLabel;
    FButtonAction: TButton;
  published
    property CustomerName: string read FCustomerName write SetCustomerName;
  end;
```

In React ist eine Komponente einfach eine Function, die zurückgibt, was angezeigt werden soll:

```
interface CustomerPanelProps {
  customerName: string;
  onActionClick: () => void;
}

function CustomerPanel({ customerName, onActionClick }: CustomerPanelProps) {
  return (
    <div>
      <label>{customerName}</label>
      <button onClick={onActionClick}>Action</button>
    </div>
  );
}
```

Der wichtigste Unterschied: In Delphi erstellen Sie ein Objekt, das über die Zeit existiert. In React beschreiben Sie, was bei bestimmten Inputs gerendert werden soll. Es ist ein deklaratives Model anstatt eines imperativen.

Kernkonzepte (Delphi-Übersetzung)

- **Component** = Ein Custom Control (wie `TCustomerPanel`)
- **Props** = Properties, die übergeben werden (wie `CustomerName` Property)
- **State** = Interne Fields, die sich ändern können (wie `FSelectedIndex`)
- **Events** = Event Handler (`onActionClick` = `OnButtonClick`)

Der größte mentale Wandel: Anstatt den Bildschirm manuell zu aktualisieren (wie `Label1.Caption := 'New Text'` aufzurufen), ändern Sie die Daten und React aktualisiert automatisch, was angezeigt wird.

State Management - Daten, die sich ändern

In Delphi haben Components interne Fields, die sich ändern:

```
type
  TCounter = class(TCustomControl)
  private
    FCount: Integer;
  public
    procedure Increment;
  end;

procedure TCounter.Increment;
begin
  Inc(FCount);
  Invalidate; // Trigger repaint
end;
```

React hat ein ähnliches Konzept namens "State":

```
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </div>
  );
}
```

Wenn `setCount` aufgerufen wird, rendert React automatisch die Komponente mit dem neuen Wert neu. Sie rufen nicht manuell `Invalidate()` auf—React übernimmt das.

Die wichtigste Erkenntnis: Anstatt das UI imperativ zu aktualisieren ("ändere diesen Label-Text"), beschreiben

Sie deklarativ, wie das UI für jeden gegebenen State aussehen soll. Wenn sich der State ändert, findet React heraus, was aktualisiert werden muss.

Event Handling - Auf User-Aktionen reagieren

Delphi Events:

```
procedure TMyForm.ButtonClick(Sender: TObject);
begin
  ShowMessage('Clicked!');
end;
```

React Events sind ähnlich:

```
function MyButton() {
  const handleClick = () => {
    alert('Clicked!');
  };

  return <button onClick={handleClick}>Click Me</button>;
}
```

Die Syntax ist anders, aber das Konzept ist identisch: Eine Function an ein Event anhängen, und diese Function läuft, wenn das Event ausgelöst wird.

Props vs State: Der wichtigste Unterschied

Props (Properties):

- Daten, die von Parent zu Child übergeben werden
- Read-only innerhalb der Component
- Wie Delphi Published Properties

State (Interne Fields):

- Daten, die innerhalb der Component verwaltet werden
- Können von der Component modifiziert werden
- Wie Delphi Private Fields

In Delphi könnten Sie haben:

```
type
  TCustomerEdit = class(TEdit)
  private
    FCustomerId: Integer;  // State (internal)
  published
    property CustomerId: Integer read FCustomerId write FCustomerId; // Can be prop
  end;
```

In React:

```
function CustomerDisplay({ customerId }: { customerId: number }) {
  const [customerName, setCustomerName] = useState('');
  // customerId ist ein Prop (vom Parent)
  // customerName ist State (intern)

  return <div>{customerName}</div>;
}
```

Component Composition (Komposition von Komponenten) - Bausteine

Genau wie in Delphi, wo Sie komplexe Forms aus Panels, Group Boxes und Controls zusammensetzen, werden React-Anwendungen durch das Zusammensetzen von Komponenten erstellt:

```
Page
   Header
      Logo
      Navigation
   Content
      CustomerList
         CustomerCard (wiederholt)
      Sidebar
   Footer
```

Jeder Teil ist eine eigenständige Komponente, die überall wiederverwendet werden kann.

Listen und Iteration

In Delphi könnten Sie eine Liste durchlaufen, um ein Grid zu füllen:

```
for I := 0 to Customers.Count - 1 do
  AddRow(Customers[I]);
```

In React transformieren Sie Arrays direkt in UI-Elemente:

```
function CustomerList({ customers }: { customers: Customer[] }) {
  return (
    <div>
      {customers.map(customer => (
        <div key={customer.id}>
          {customer.name}
        </div>
      ))}
    </div>
  );
}
```

Die `.map()` Function transformiert jeden Customer in eine Komponente. React übernimmt das Rendering.

Bedingtes Rendering

In Delphi: `Panel1.Visible := IsAdmin;`

In React:

```
{isAdmin && <AdminPanel />}
{hasPermission ? <EditButton /> : <ReadOnlyView />}
```

Sie beschreiben, was unter welchen Bedingungen angezeigt werden soll. React findet heraus, wann die Seite oder nur ein Teil davon aktualisiert werden muss.

Was das für Ihre Anwendungen bedeutet

Das React Component Model lässt sich gut auf Delphis Ansatz abbilden:

- Beide verwenden wiederverwendbare, zusammensetzbare Bausteine
- Beide haben Properties/Props und internen State
- Beide handhaben Events
- Beide fördern Separation of Concerns

Der wichtigste Unterschied: React ist deklarativ (beschreiben, was gezeigt werden soll), während Delphi imperativ ist (dem Computer Schritt für Schritt sagen, was zu tun ist). Das braucht etwas Gewöhnung, aber viele Entwickler finden, dass es zu saubererem Code führt, sobald sie sich angepasst haben.

Ihre Migration bewerten

Beim Betrachten Ihrer Delphi-Anwendungen fragen Sie:

- Wie viel Ihres UI besteht aus Standard Controls, die auf React Components abgebildet werden könnten?
- Wie viel ist Custom-gezeichnet oder stark angepasst?
- Wie eng gekoppelt ist Ihr UI an Ihre Business Logic?

Standard CRUD Forms lassen sich einfach übersetzen. Komplexe Custom Controls könnten mehr Überlegung brauchen. Die gute Nachricht: Ihre Business Logic (Berechnungen, Validierungen, Regeln) kann oft fast direkt verschoben werden, sobald Sie die TypeScript-Syntax verstehen.

Wann professionelle Hilfe Sinn macht

React Components konzeptionell zu verstehen ist eine Sache. Eine vollständige Migration zu architektieren ist eine andere. Sie müssen berücksichtigen:

- Component-Struktur und Wiederverwendbarkeit
- State Management über große Anwendungen hinweg
- Form Validation Strategien
- Error Handling Patterns
- Performance-Optimierung
- Testing-Ansätze

Das sind strategische architektonische Entscheidungen, nicht nur Syntax-Änderungen.

Ich helfe Delphi-Teams bei der Planung und Ausführung dieser Migrationen:

- Analysiere Ihre bestehenden Forms und Data Entry Screens
- Entwerfe eine Komponenten-Architektur, die zu Ihrer Anwendung passt
- Erstelle wiederverwendbare 'Component Libraries'
- Schule Ihr Team in React Patterns und Best Practices
- Führe die Migration komplexer Business Logic an
- Etabliere Muster für Ihr Team zum Folgen

Das Ziel ist, Ihrem Team eine solide Grundlage und klare Patterns zu geben, damit sie nach der initialen Migrationsarbeit weiterbauen können.

Was als nächstes kommt

In Teil 4 betrachten wir die Next.js Applikationsstruktur — wie Projekte organisiert sind, wie Routing funktioniert und wo Ihre Business Logic lebt. Sie werden sehen, wie sich das mit Delphis Projektstruktur vergleicht und wie Ihre Database Operations übertragen werden.

Wir bauen auf Teil 5 hin, wo wir tatsächliche Migrationsstrategien besprechen werden: was zuerst migriert werden soll, wie man es schrittweise macht und wann professionelle Hilfe hinzugezogen werden sollte.

Bereit, Ihre spezifischen Delphi-Anwendungen zu besprechen? Ich würde gerne Ihre Forms und Screens überprüfen, um Ihnen eine realistische Einschätzung zu geben, was eine Migration beinhalten würde. Auch ein kurzes Gespräch kann Ihnen helfen, den Umfang und Aufwand zu verstehen. Kontaktieren Sie mich!

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —