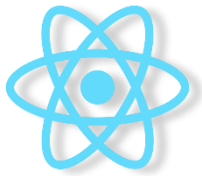
Mit Liebe vom OstBlock™ präsentiert

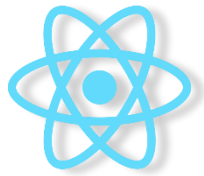
Dominic Fiegert, Slawo Grabos, Juri Schneider, Marius Holeksa, Artur Minich, Tamas Szitas

Was ist React?



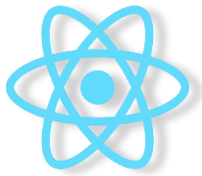
- Eine beliebte Open Source JavaScript Library, die mithilfe von HTML Data Viewing ausführt
- Eine effiziente, selbsterklärende und flexible Bibliothek zum Erstellen von Benutzeroberflächen
- Die Benutzeroberfläche wird aus isolierten Code-Stücken (Komponenten) zusammengestellt
- Bei Änderung der Daten updatet React nur die betroffenen Komponenten
- Wird von Facebook, Instagram und Community-Entwickler verwendet

Komponenten



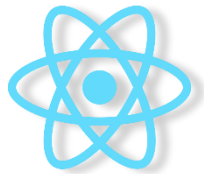
- Mit Komponenten können Benutzeroberflächen in unabhängige, wiederverwendbare Teile aufgebrochen und isoliert betrachtet werden
- Komponenten nehmen beliebige Eingaben entgegen (props) und geben React-Elemente zurück
- Jede React-Komponente ist entkapselt und kann unabhängig arbeiten
- Funktionskomponenten sind vom Prinzip her JavaScript-Funktionen
- Klassenkomponenten sind komplexer als Funktionskomponenten

Entwicklungshistory



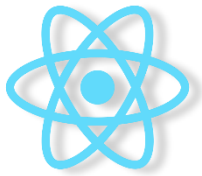
- 2011 von Jordan Walke, Softwareentwickler bei Facebook, entwickelt
- Erster Anwendungsfall im Facebook Newsfeed in 2011
- Später von Instagram verwendet
- Facebook entschied sich im Mai 2013, es als Open Source-Software zur Verfügung zu stellen
- In 2015 wurde React Native angekündigt, welches die Entwicklung mit Android und iOS vereinfacht

Eigenschaften



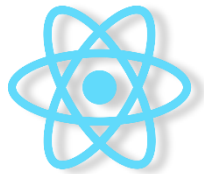
- Funktioniert wie ein View, das ein komponentenbasiertes System nutzt
- Die Komponenten sind als Custom HTML Tags bezeichnet
- Die Unterkomponenten können nicht von externen Warteschlangen beeinflusst werden
- Die Bibliothek geht sicher, dass eine klare Trennung der verschiedenen Komponenten zu erkennen ist

Unterschied Bibliothek / Framework



- Bei einer Bibliothek handelt es sich um eine Sammlung von Klassen und Funktionen
- Der Entwickler ruft innerhalb seines Codes die Klassen und Funktionen direkt auf
- Ein Framework ist eine spezielle Form der Bibliothek
- Das Framework ruft vorgesehene Funktionen wenn nötig selbstständig auf

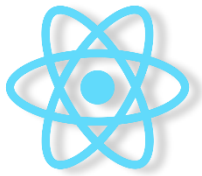
Single-Page-Webanwendung



- Eine Webanwendung, die aus einem einzigen HTML-Dokument besteht und dessen Inhalte dynamisch nachgeladen werden
- Durch die Nutzung einer einzigen Webseite kann die Client-Server-Kommunikation reduziert werden
- Durch vollständige Umsetzung der Präsentationsschicht auf dem Client ist es möglich, mithilfe der Web-Storage Funktion persistente Zwischenspeicher anzulegen
- Für dem Fall, dass der Nutzdatenserver nicht erreichbar ist, können Daten aus dem Zwischenspeicher verwendet werden

  → Anwendung kann weiterhin auf dem Client betrieben werden, ohne dass eine Verbindung zum Server bestehen muss

Model View Controller (MVC)



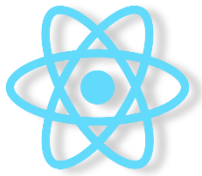
- Der MVC Controller beeinflusst direkt die Daten im gegebenen Model

- View und Model sind unabhängig voneinander

  → Programmierer können unabhängig voneinander gleichzeitig an verschiedenen Stellen der Anwendung arbeiten

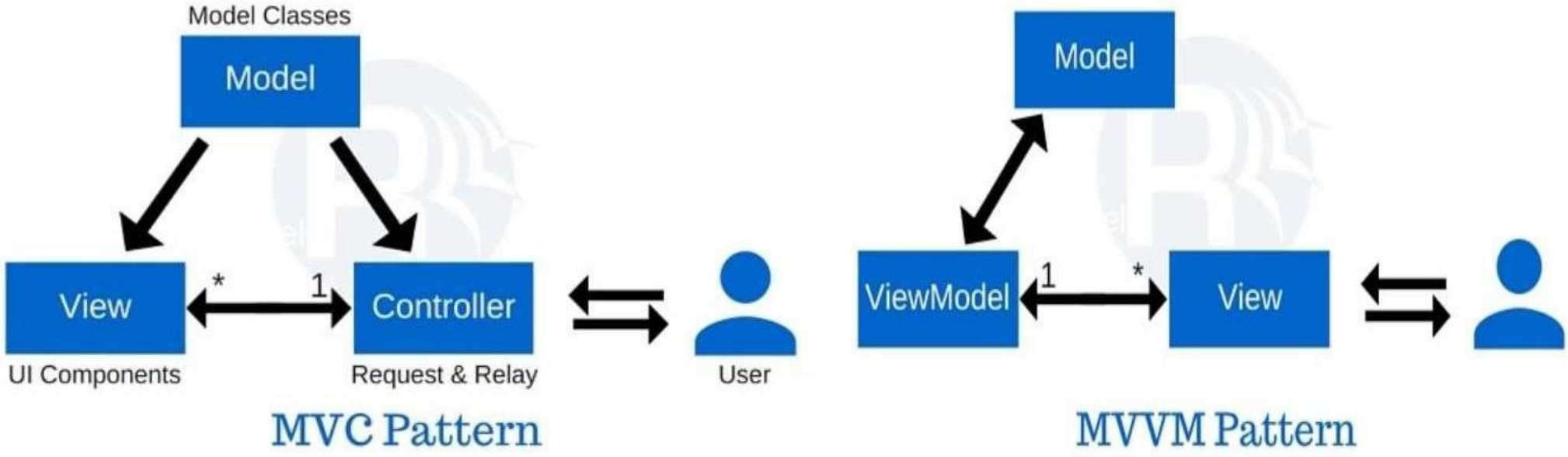
- Da üblicherweise jedes Model seinen eigenen Controller hat, kann dies bei größeren Anwendungen zu Unübersichtlichkeit und komplexen Strukturen führen

Model View ViewModel (MVVM)

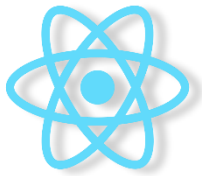


- MVVM ViewModel basiert überwiegend auf dem Frontend der Webanwendung
- Das MVVM Format ist speziell dafür ausgelegt, um eine direkte Kommunikation zwischen View und Model zu ermöglichen
- MVVM Single-Page-Webanwendungen können schnell und flüssig reagieren und ununterbrochen Daten in der Datenbank speichern
- Im Gegensatz zum MVC Controller nimmt das View Model wesentlich mehr Speicher, aufgrund der Datenbindung, ein.

MVC oder MVVM

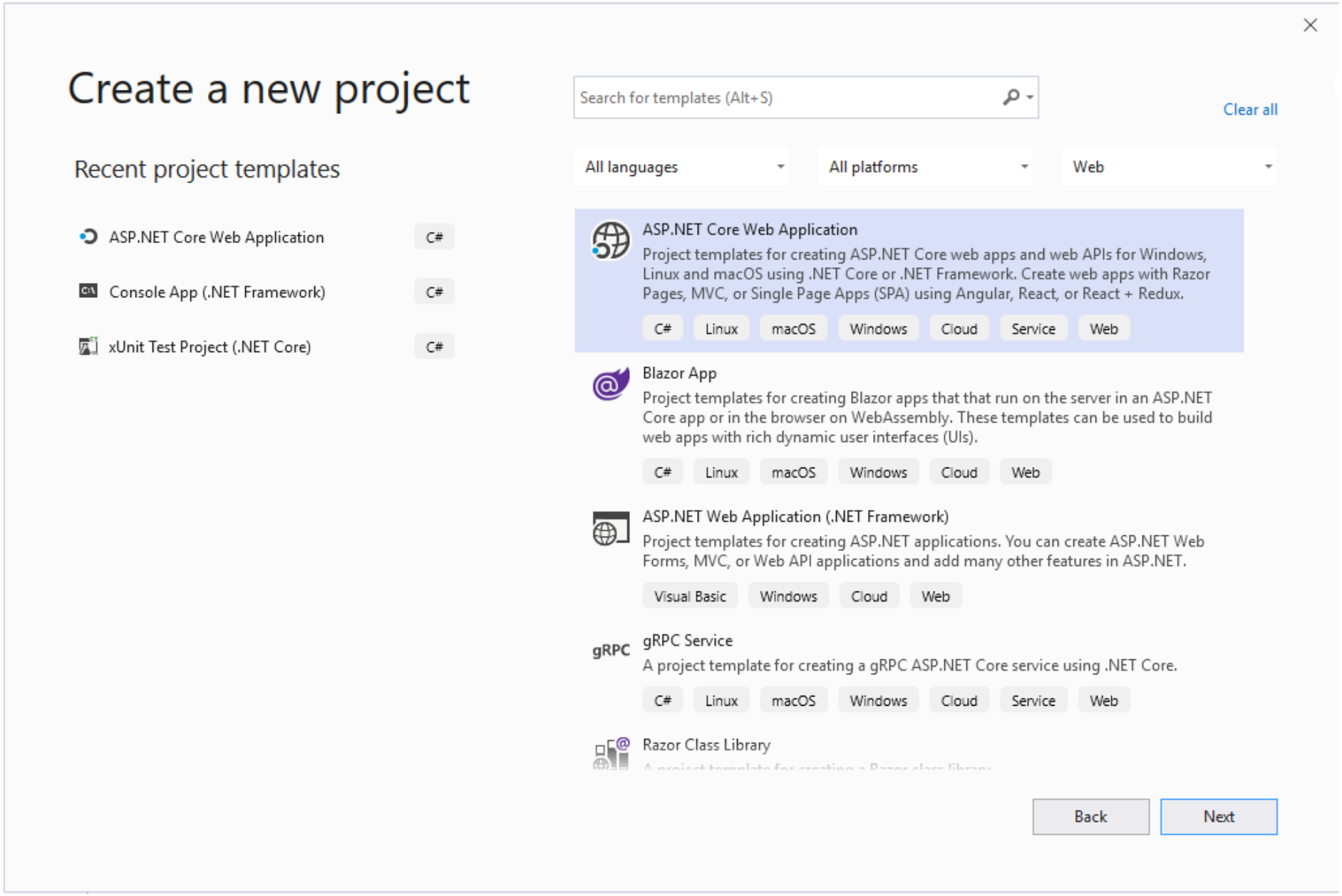


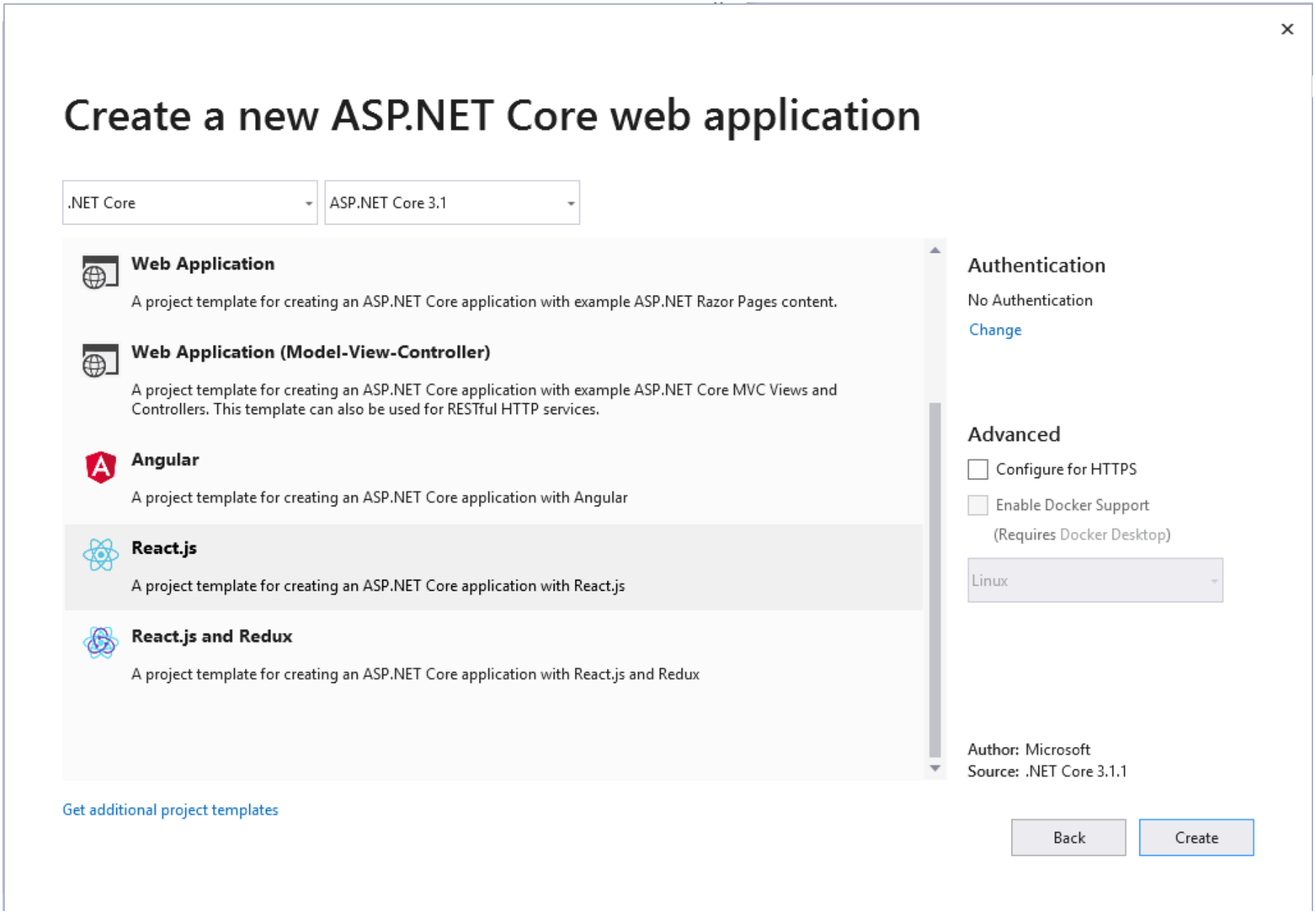
Installation #1



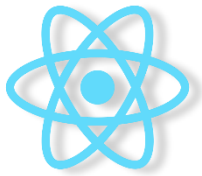
Mit Visual Studio 2019:

- Neues ASP.Net Core Web Applikation Projekt anlegen
- React auswählen
- Fertig


Create a new project
Search for templates (Alt+S)
Clear all
Recent project templates
ASP.NET Core Web Application
C#
Console App (.NET Framework)
C#
xUnit Test Project (.NET Core)
C#
All languages
All platforms
Web
ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
C# Linux macOS Windows Cloud Service Web
Blazor App
Project templates for creating Blazor apps that that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web
ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.
Visual Basic Windows Cloud Web
gRPC Service
A project template for creating a gRPC ASP.NET Core service using .NET Core.
C# Linux macOS Windows Cloud Service Web
Razor Class Library
Back
Next


Create a new ASP.NET Core web application
.NET Core
ASP.NET Core 3.1
Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
Web Application (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
Angular
A project template for creating an ASP.NET Core application with Angular
React.js
A project template for creating an ASP.NET Core application with React.js
React.js and Redux
A project template for creating an ASP.NET Core application with React.js and Redux
Authentication
No Authentication
Change
Advanced
Configure for HTTPS
Enable Docker Support
(Requires Docker Desktop)
Linux
Author: Microsoft
Source: .NET Core 3.1.1
Get additional project templates
Back
Create

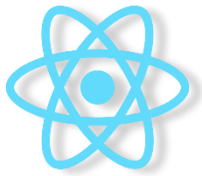
Installation #2



CREATE-REACT-APP

- Für die Installation wird NodeJS benötigt
- Durch NodeJS steht npm (Node Package Manager) zur verfügung
- Zentraler Lager für zu lokal oder global installierende Pakete (axios, uuid, bootstrap usw.)
- npx create-react-app my_app_name_was_auch_immer
- npx = ermöglicht die einfache Nutzung von CLI Tools und Installation von Pakete
- create-react-app = Der bevorzugte Weg, eine React-Anwendung (Boilerplate) zu generieren
- Webpack, Babel, React-scripts vorkonfiguriert
- my_app_name_was_auch_immer = Der Name des eigenen Projekts

Create-React-App

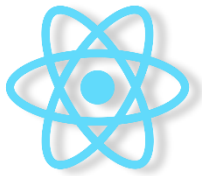


MINGW64:/d/Learn Material/School projects

```
ITA1-TN12@PC0410 MINGW64 /d/Learn Material/School projects
$ npx create-react-app my_app_name_was_auch_immer
npx: Installierte 99 in 13.775s

Creating a new React app in D:\Learn Material\School projects\my_app_name_was_auch_immer.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

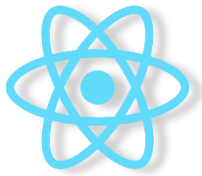
```
+ react-scripts@3.4.0
+ react-dom@16.13.0
+ react@16.13.0
+ cra-template@1.0.2
added 1566 packages from 747 contributors and audited 918658 packages in 111.774s
```

Zu Verfügung stehende Befehle



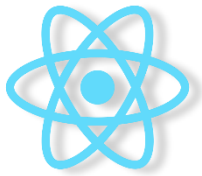
```
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd my_app_name_was_auch_immer
  npm start

Happy hacking!
```

Create-React-App: npm start



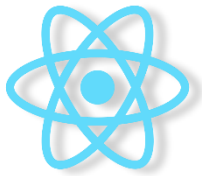
```
ITA1-TN12@PC0410 MINGW64 /d/Learn Material/School projects/my_app_name_was_auch_immer
$ npm start

> my_app_name_was_auch_immer@0.1.0 start D:\Learn Material\School projects\my_app_name
> react-scripts start

i 「wds」: Project is running at http://172.21.0.174/
i 「wds」: webpack output is served from
i 「wds」: Content not from webpack is served from D:\Learn Material\School projects\my_
i 「wds」: 404s will fallback to /
Starting the development server...

Compiled successfully!

You can now view my_app_name_was_auch_immer in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.21.0.174:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Die einzelne HTML-Datei



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <base href="%PUBLIC_URL%/" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <title>React Demo Project</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
  </body>
</html>
```

Erstes Beispiel – Index.JS



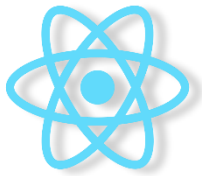
React aus dem React Paket importieren
ReactDOM aus dem React-Dom Paket importieren

ReactDOM.render(was, wohin):

- Eine Funktion, welche eine Komponente in das gewünschte HTML-Element einfügt
- Rendert das App Element (das oberste Komponente) in das einzige HTML Element (#root)

```
import 'bootstrap/dist/css/bootstrap.css';
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

const rootElement = document.getElementById('root');

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  rootElement);

registerServiceWorker();
```

Erstes Beispiel – App.JS – ROUTER ENTFERNEN



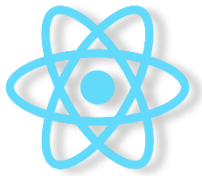
React und Component aus dem React Paket importieren
Route aus dem React-Router Paket importieren:

- React-Router ermöglicht die Nutzung von z.B.
  /home | /counter | /fetch-data
- Beim Aufruf des Pfades wird die angegebene Komponent gerendert

Layout ist eine Komponente, um die Struktur der Webseite festzustellen

```
import React, { Component } from 'react';
import { Route } from 'react-router';
import { Layout } from './components/Layout';
import Home from './components/Home';
import { FetchData } from './components/FetchData';
import { Counter } from './components/Counter';

import './custom.css'

export default class App extends Component {

  render () {
    return (
      <Layout>
        <Route exact path='/' component={Home} />
        <Route path='/counter' component={Counter} />
        <Route path='/fetch-data' component={FetchData} />
      </Layout>
    );
  }
}
```

Erstes Beispiel – Hello World

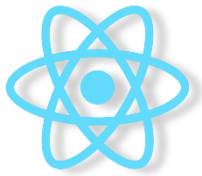


Klassenkomponente:

- Eine klassenbasierte Komponente MUSS eine render() Methode haben, welche in diesem Fall durch „return“ eine <h1> HTML-Element (?) zurückgibt
- Die Komponente muss exportiert werden, damit sie in anderen Komponenten wieder verwendet werden kann
- Klassenkomponente sind „ältere“ Komponente, die sich auch stateful-, container- oder smart component nennen. Der Grund dafür ist React vor 2018, denn nur Klassenkomponente durften das State Objekt besitzen und damit arbeiten.

```
import React, { Component } from 'react';

class Home extends Component {

    render() {
        return <h1>Hello, world!</h1>
    }

}

export default Home;
```

Erstes Beispiel – Hello World

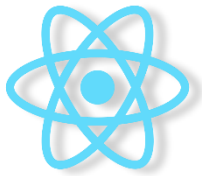


Funktionskomponente:

- Komponente, die eigentlich eine Funktion ist, welche ein HTML-Element (?) zurückgibt.
- Sie nennt sich auch stateless-, presentational- oder dumb component.
- Funktionskomponenten sind vor 2018 nur für die Erstellung von HTML-Elementen (?) verwendet worden, da sie kein lokales State haben konnten.

```
import React from 'react';

const Home = () => <h1>Hello, world!</h1>

export default Home;
```

Ist das doch HTML?

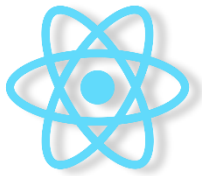


Hello World!

```
import React from 'react';

const Home = () => <h1 className="red-text">Hello World!</h1>

const Home = () => React.createElement('h1', { className: "red-text" }, "Hello World!")

export default Home;
```

- React hat einen einzigartigen Syntax: JSX (JavaScript XML)
- JSX ist ein „syntactical sugar“, da es im Hintergrund beim Kompilieren in React.createElement() umgewandelt wird
- In React ist ALLES JavaScript!

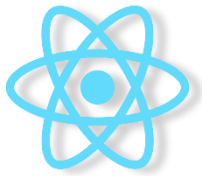
Props



- Props (Kurzform von Properties) sind weitere Eigenschaften, die von der Elternkomponente in die Kind Komponenten „runtergebohrt“ werden können.
- Die zwei Props sind „name“ und „today“, die ab jetzt in der Home Komponente zur Verfügung stehen

```
export default class App extends Component {

  render () {
    return (
        <Layout>
            <Home name="ITA-1" today={new Date().toLocaleDateString()} />
      </Layout>
    );
  }
}
```

Props in der Kind Komponente



```jsx
const Home = (props) => (
    <div>
        <h1 className="red-text">Hello, {props.name}!</h1>
        <h2>Das heutige Datum ist: {props.today}</h2>
    </div>
)
```

Hello, ITA-1!

Das heutige Datum ist: 9.3.2020

Home

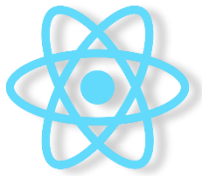
props

```
name:  "ITA-1"
today:  "9.3.2020"
new prop    : ""
```

rendered by

App

Component Level State

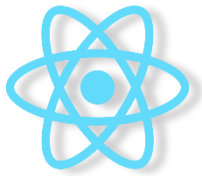


Lokales State in Klassenkomponenten

- State bezieht sich auf die interne Logik der Komponente und ist von außen nicht erreichbar
- Das State kann alle Typen enthalten
- In diesem Beispiel besteht das State aus einem Objekt, welches standardmäßig als Schlüssel „currentCount“ und als Wert „0“ hat.
- Durch die „incrementCounter()“ Methode wird die „this.setState()“ React-eigene Methode verwendet, um den Wert des Counters zu inkrementieren.
- Immutable State!

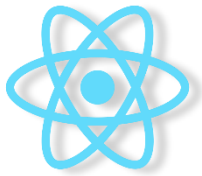
```
import React, { Component } from 'react';

export class Counter extends Component {

  constructor(props) {
    super(props);
    this.state = { currentCount: 0 };
    this.incrementCounter = this.incrementCounter.bind(this);
  }

  incrementCounter() {
    this.setState({
      currentCount: this.state.currentCount + 1
    });
  }
```

Render Funktion des Counters

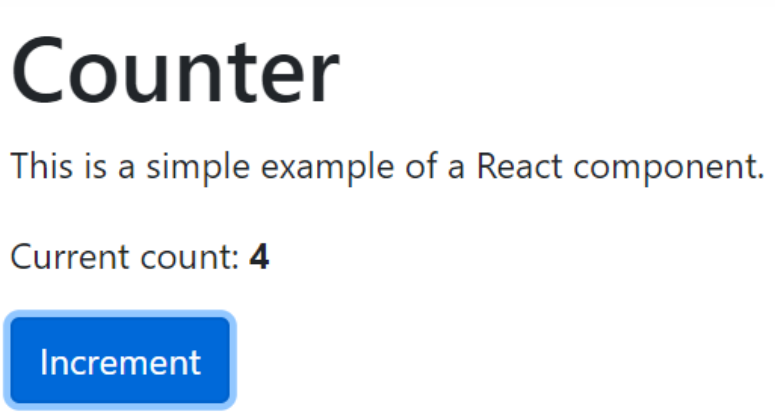


```
render() {
  return (
    <div>
      <h1>Counter</h1>

      <p>This is a simple example of a React component.</p>

      <p aria-live="polite">Current count: <strong>{this.state.currentCount}</strong></p>

        <button
          className="btn btn-primary"
          onClick={this.incrementCounter}>
          Increment
        </button>
    </div>
  );
```

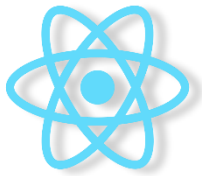
Das virtuelle DOM



- Wenn das State sich ändert (die Taste wird gedrückt und der Counterwert erhöht sich um 1) wird die Komponente neu gerendert.
- React rendert die Komponenten neu, wenn sich das State oder die Props ändern, also wenn sich irgendwas in der Anwendung ändert, welches eine Auswirkung auf das entsprechende Komponente hat.
- Um zu wissen, welches Teil der Anwendung neugerendert werden muss, benutzt React das sogenannte virtuelle DOM. React vergleicht den jetzigen Zustand des DOMs mit dem bei jeder Veränderung erstellten virtuellen DOM und analysiert.
- Wo sich eine Komponente verändert hat und nur diese Komponente wird aktualisiert.



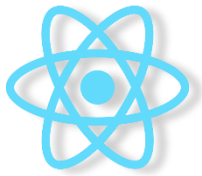
Das Abenteuer ins Rabbit-Hole… #1



Component Lifecycle:

- Eine Komponente verfügt über mehrere Lifecycle-Methods
  - Komponente wird gerendert
  - Komponente ist gerade gerendert worden
  - Hat sich die Komponente gerade aktualisiert?
  - Soll die Komponente sich neurendern?
  - Hat die Komponente eine Exception gefangen?
  - Wird die Komponente gerade vom DOM entfernt?
  - Ist die Komponente gerade vom DOM entfernt worden?

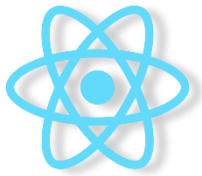
Das Abenteuer ins Rabbit-Hole… #2



Globales State Management

- Die Erstellung einer React Komponente ist ziemlich einfach:
  - Klassen- oder Funktionskomponente erstellen und JSX zurückliefern
- Wenn wir eine große Anwendung haben, kann die Struktur der Anwendung schnell unübersichtlich werden. Welche Komponente sollen über ein State verfügen? Wie tief in der Hierarchie muss ich Props runterschicken, wenn ich eine gewisse Eigenschaft nur in einer Kind Komponente brauche, die Eigenschaft sich aber in der in einer der obersten Parentkomponenten befindet? Dieses Phänomen führt zum „prop-drilling“.
- Lösung: Ein globales, zentrales Store in der Anwendung zu haben, worauf jede Komponente zugreifen kann, um Daten, Methoden oder Eigenschaften für sich zu holen.
  - Redux
  - React-eigene Context API
  - MobX

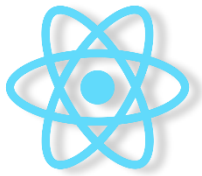
Das Abenteuer ins Rabbit-Hole… #3



State Management und React Hooks:

- Mittlerweile können auch Funktionskomponente über ein State verfügen
- React Hooks sind Funktionen, die in Funktionskomponente benutzt werden können
- UseState Hook:
  - Ermöglicht, dass eine Funktionskomponente ein (oder mehrere) State hat
  - Die UseState Funktion nimmt eine Default State entgegen und liefert ein Array zurück, dessen erstes Element das State ist, die zweite jedoch eine neue Funktion ist, womit das State geändert werden kann
  - const [todo, setTodo] = useState({ titel: ‘ ‘, istErdledigt: false })
  - const [darkMode, setDarkMode] = useState(false)
- UseEffect Hook:
  - Ermöglicht die Arbeit mit dem Lifecycle der Funktionskomponente innerhalb einer kompakten Funktion

Das Abenteuer ins Rabbit-Hole… #4



Styling von Komponenten:

- Zuweisung eines Objektes mit Styling Attributen (backGroundColor, paddingLeft, etc.) – Achtung! JavaScript Syntax!
- Inline Styling mit JSX
- 3rd Party Library:
  - styled-components
  - Radium
  - materialize-css
  - Materialize UI
  - Reactstrap
- Globales CSS
- Modulares CSS