



Full-stack Application Development

Introduction to React

Where to Find The Code and Materials?

<https://github.com/iproduct/fullstack-typescript-react>



Agenda

1. MVC flavours
2. Single Page Applications (SPA)
3. SIMPLE Webpack Project Bootstrapping
4. Why React - simple and superfast, component oriented development using pure JavaScript (ES 6), virtual DOM, one-way reactive data flow, MVC framework agnostic
5. React by example – JSX syntax
6. React by example – JavaScript syntax
7. Lets do some code :)
8. Top level API
9. ES6 class syntax

Agenda

10. JSX in depth – differences with HTML, transformation to JavaScript, namespaced components,
11. Expressions, child expressions and comments, props mutation anti-pattern, spread attributes, using HTML entities, custom attributes, if-else, immediately-invoked function expressions.
12. React Components Lifecycle Callbacks and ES6 class syntax
13. Events in React, managing DOM events
14. Components composition in depth – ownership, [*this.props.children*](#), [*React.Children*](#) utilities, child reconciliation, stateful children and dynamic children using keys
15. Transferring props

MVC Comes in Different Flavors



What is the difference between following patterns:

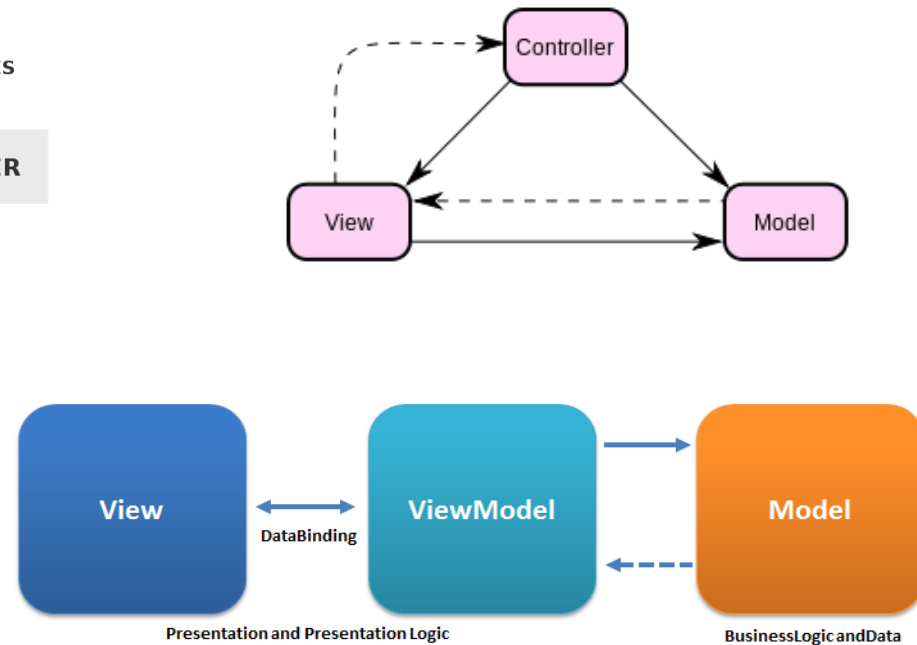
- Model-View-Controller (MVC)
- Model-View-ViewModel (MVVM)
- Model-View-Presenter (MVP)

MVC Comes in Different Flavors - 2

- MVC



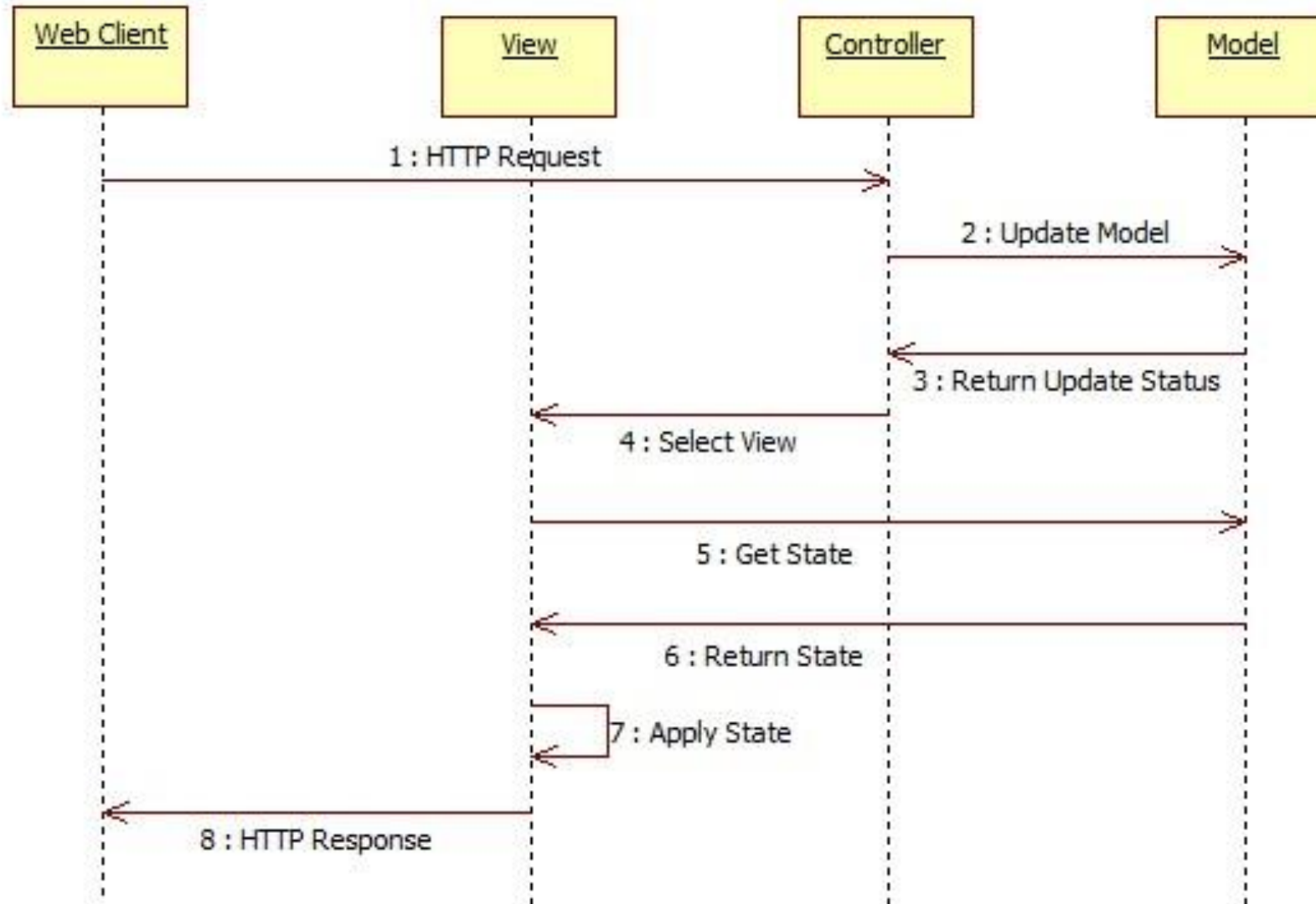
- MVVM



- MVP

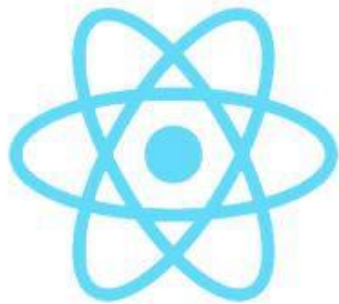


Web MVC Interactions Sequence Diagram



Why React?

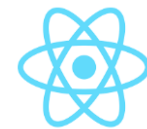
- **React.js** is a JavaScript library for creating user interfaces by Facebook and Instagram – the V in MVC.
- **Solves well one problem**: building large applications with data that changes over time
- Simple and **superfast** – one-way reactive data flow



React

Why React?

- **Declarative** and **one-way reactive data flow** – simply express how your app should look, and React will automatically manage all UI updates when your underlying data changes
- **Component oriented SPA** development using pure JavaScript (**ES 6**) – React is all about building composable and reusable components - code reuse, testing, and separation of concerns
- **Virtual DOM** – allows decoupling of components from DOM, rendering done as last step
- Allows **isomorphic (client + server side) rendering**
- **MVC framework agnostic** – Flux, Redux, Reflux, ...
- Available at: <https://facebook.github.io/react>



React.js by Example – JSX Syntax

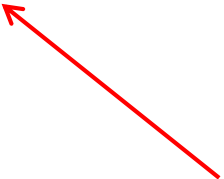
```
import React from "react";

import ReactDOM from "react-dom";

import Hello from "./hello";

ReactDOM.render(
  <Hello name="World" />,
  document.getElementById('app')
);
```

```
import React from "react";
export default
class Hello extends React.Component
{
  render() {
    return (
      <div className="hello">
        <h2>
          Hello, {this.props.name}!
        </h2>
      </div>
    );
  }
}
```



JavaScript syntax extension (JSX)
that looks similar to XML

Hello React TypeScript Example - index.tsx

- `npx create-react-app 07-ts-react --template typescript`

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
```

```
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

```
serviceWorker.unregister();
```

// More about service workers: <https://create-react-app.dev/docs/making-a-progressive-web-app/>

Hello React TypeScript Example – App.tsx (Function Component)

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
import { Hello } from './Hello';
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>Edit <code>src/App.tsx</code> and save to reload.</p>
        <a className="App-link" href="https://reactjs.org" target="_blank" rel="noopener noreferrer">
          Learn React
        </a>
        <Hello compiler="TypeScript" framework="React" />
      </header>
    </div>
  );
}
export default App;
```

Hello React TypeScript Example – Hello.tsx (Class Component)

```
import React from 'react';
```

```
export interface HelloProps {  
  compiler: string;  
  framework: string;  
}
```

// 'HelloProps' describes the shape of props. State is never set so we use the '{}' type.

```
export class Hello extends React.Component<HelloProps, {}> {  
  render() {  
    return (  
      <div>  
        <h1>Hello from {this.props.compiler} and {this.props.framework}!!! </h1>  
      </div>  
    );  
  }  
}
```

React TypeScript Component with PropTypes Runtime Validation

```
import React from 'react';
import PropTypes from 'prop-types';

export interface HelloProps {
  compiler: string;
  framework: string;
}

export class Hello extends React.Component<HelloProps, {}> {
  static propTypes = {
    compiler: PropTypes.string.isRequired,
    framework: PropTypes.string.isRequired,
  };
  render() {
    return (
      <div>
        <h1>Hello from {this.props.compiler} and {this.props.framework}!</h1>
      </div>
    );
  }
}
```

Comments Demo Example – Pure JavaScript

```
import React from "react";
import ReactDOM from "react-dom";

let CommentBox = React.createClass({displayName: 'CommentBox',
  render: function() {
    return (
      React.createElement('div', {className: "commentBox"},
        "Hello, world! I am new CommentBox."
      )
    );
  }
});

ReactDOM.render(
  React.createElement(CommentBox, null),
  document.getElementById('app')
);
```

Lets Do Some React Code :)

Official React Comments tutorial:

<https://facebook.github.io/react/docs/tutorial.html>

React comment box example available @GitHub:

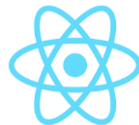
<https://github.com/reactjs/react-tutorial>

React.js documentation and API:

<https://facebook.github.io/react/docs>

Top Level API

- **React** – the entry point to the React library. If you're using one of the prebuilt packages it's available as a global; if you're using CommonJS modules you can `require()` it.
- **ReactDOM** – provides DOM-specific methods that can be used at the top level of your app and as an escape hatch to get outside of the React model if you need to. Most of your components should not need to use this module.
- **ReactDOMServer** – the `react-dom/server` package allows you to render your components on the server: **`ReactDOMServer.renderToString(ReactElement element)`**
- @: [*https://facebook.github.io/react/docs/top-level-api.html*](https://facebook.github.io/react/docs/top-level-api.html)



React ES6 Demo Example

```
export class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {count: props.initialCount};
    this.tick = this.tick.bind(this);
  }
  tick() {
    this.setState({count: this.state.count + 1});
  }
  render() {
    return (
      <div onClick={this.tick}>
        Clicks: {this.state.count}
      </div>
    );
  }
}
Counter.propTypes = { initialCount: PropTypes.number };
Counter.defaultProps = { initialCount: 0 };
```

React Components

- **Virtual DOM** – everything is a component (e.g. `<div>` in JSX), rendering done as last step
- **Components are like functions** – of three arguments:
- **this.props** – these is the external interface of the component, passed as attributes – allow the parent component (“owner”) to pass state and behavior to embedded (“owned”) components. Should never be mutated within component – immutable.
- **this.props.children** – part of the component interface but passed in the body of the component (component tag)
- **this.state** – internal state of the component should be mutated only using **React.Component.setState(nextState, [callback])**

React Components as Pure Functions

```
function HelloMessage(props) {  
    return <div>Hello {props.name}</div>;  
}
```

```
ReactDOM.render(<HelloMessage name="React User" />, mountNode);
```

- OR using ES6 => syntax:

```
const HelloMessage = (props) => <div>Hello {props.name}</div>;
```

```
ReactDOM.render(<HelloMessage name="React User" />, mountNode);
```

What Components Should Have State?

- Most components should just render data from props. However, sometimes you need to **respond to user input**, a **server request** or the **passage of time** => then use state.
- Try to keep as many of your components as possible **stateless** – makes easier to reason about your application
- Common pattern: create several **stateless components** that just **render data**, and have a **stateful component above them** in the hierarchy that **passes its state to its children via props**.
- **Stateful component** encapsulates all of the **interaction logic**
- **Stateless components** take care of **rendering data in a declarative way**

JSX Syntax

- With JSX: `Hello!`
- In pure JS: `React.createElement('a',
 {href: 'https://facebook.github.io/react/'}, 'Hello!')`
- JSX is optional – we could write everything without it, but it is not very convenient:

```
var child1 = React.createElement('li', null, 'First Text Content');
```

```
var child2 = React.createElement('li', null, 'Second Text Content');
```

```
var root = React.createElement('ul', { className: 'my-list' }, child1, child2);
```

JS Syntax Using Factories

- We can use factories to simplify the component use from JS:

```
var Factory = React.createFactory(ComponentClass);
```

```
...
```

```
var root = Factory({ custom: 'prop' });
```

```
ReactDOM.render(root, document.getElementById('example'));
```

- For standard components like <div> there are factories built-in:

```
var root = React.DOM.ul(  
    { className: 'my-list' },  
    React.DOM.li(null, 'Text Content')  
);
```

JS Syntax in Depth

- Since **JSX is JavaScript**, identifiers such as **class** and **for** are discouraged as XML attribute names. Instead, React DOM components expect DOM property names like **className** and **htmlFor**, respectively.

```
var myDivElement = <div className="foo" />;
```

```
ReactDOM.render(myDivElement, document.getElementById('example'));
```

- To render it use **Uppercase variable** → **comp.displayName**:

```
var MyComponent = React.createClass({/*...*/});
```

```
var myElm = <MyComponent someProperty={true} />;
```

```
ReactDOM.render(myElm, document.getElementById('example'));
```


JavaScript Expressions

- Attribute Expressions:

```
var person = <Person name= {app.isLoggedIn ? app.currentUser : ""} />;
```

- Boolean Attributes:

```
<input type="button" disabled />;
```

```
<input type="button" disabled={true} />;
```

- Child Expressions:

```
var content = <Container>
```

```
    {app.isLoggedIn ? <Nav /> : <Login />}
```

```
</Container>;
```

JSX Spread Attributes

- Mutating props is bad – should be treated as immutable
- Spread Attributes:

```
var props = {};
```

```
props.foo = x;
```

```
props.bar = y;
```

```
var component = <Component {...props} />;
```

- Order is important – property value overriding:

```
var props = { foo: 'default' };
```

```
var component = <Component {...props} foo={'override'} />;
```

```
console.log(component.props.foo); // 'override'
```

HTML Entities in JSX

- Double escaping (all content is escaped by default – XSS):

`<div>First · Next</div>` - **OK**

`<div>{'First · Next'}</div>` - **Double escaped**

- Solution 1: type (and save) it in UTF-8:

`<div>{'First · Next'}</div>`

- Solution 2: use Unicode

`<div>{'First \u00b7 Next'}</div>`

`<div>{'First ' + String.fromCharCode(183) + ' Next'}</div>`

- Solution 3: use mixed arrays with strings and JSX elements:

`<div>['First ', ·, 'Next']</div>`

- Solution 4 (last resort): type (and save) it in UTF-8:

`<div dangerouslySetInnerHTML={{__html: 'First · Next'}} />`

HTML Entities in JSX

- If you pass properties to native HTML elements that do not exist in the HTML specification, React will not render them.

- Custom attributes - should be prefixed with **data-** :

```
<div data-custom-attribute="foo" />
```

- Custom elements (with a hyphen in the tag name) support arbitrary attributes:

```
<x-my-component custom-attribute="foo" />
```

- Web Accessibility attributes starting with **aria-** are rendered:

```
<div aria-hidden={true} />
```

Immediately-Invoked Function Expressions

```
return (  
  <section>  
    <h1>Color</h1>  
    <h3>Name</h3> <p>{this.state.color || "white"}</p>  
    <h3>Hex</h3><p>  
      {(() => {  
        switch (this.state.color) {  
          case "red": return "#FF0000";  
          case "green": return "#00FF00";  
          default: return "#FFFFFF";  
        }  
      })()}  
    </p>  
  </section>  
);
```

React Component Lifecycle Callbacks (1)

- React components **lifecycle** has 3 phases:
 - **Mounting**: A component is being inserted into the DOM.
 - **Updating**: A component is being re-rendered to determine if the DOM should be updated.
 - **Unmounting**: A component being removed from the DOM.
- **Mounting** lifecycle callbacks:

constructor()

static getDerivedStateFromProps() - if the state depends on changes in props

render()

componentDidMount() - is invoked immediately after mounting occurs. Initialization that requires DOM nodes should go here.

React Component Lifecycle Callbacks (2)

static `getDerivedStateFromProps()` - invoked right before calling the render method, both on the initial mount and on subsequent updates. It should return an object to update the state, or null to update nothing. This method exists for rare use cases where the state depends on changes in props over time.

Updating Lifecycle Callbacks

static getDerivedStateFromProps(props, state)

shouldComponentUpdate(object nextProps, object nextState): boolean – invoked when a component decides whether to update - optimization comparing **this.props** with **nextProps** and **this.state** with **nextState** and return **false** if React should skip updating.

render()

getSnapshotBeforeUpdate(prevProps, prevState) - invoked right before the most recently rendered output is committed to e.g. the DOM. It enables your component to capture some information from the DOM (e.g. scroll position) before it is potentially changed. Any value returned by this lifecycle will be passed as a parameter to `componentDidUpdate()`.

componentDidUpdate(object prevProps, object prevState) – invoked immediately after updating occurs.

Unmounting Lifecycle Callbacks:

- **Unmounting**

componentWillUnmount() – invoked immediately before a component is unmounted and destroyed. Cleanup should go here.

- **Error Handling**

static getDerivedStateFromError(error)

componentDidCatch(error, info)

- **Mounted** composite components also support:

component.forceUpdate() – can be invoked on any mounted component when you know that some deeper aspect of the component's state has changed without using **this.setState()**.

React Hooks – New in React 16!

[\[https://reactjs.org/docs/hooks-intro.html\]](https://reactjs.org/docs/hooks-intro.html)

- **Hooks** are a new addition in React 16.8. They let you use state and other React features without writing a class.
- **Basic Hooks**
 - **useState**: `const [state, setState] = useState(initialState);`
 - **useEffect**: `useEffect(() => {
 const subscription = props.source.subscribe();

 return () => { subscription.unsubscribe() };
});`
 - **useContext** – allows to access resources application wide
- **Additional Hooks** – `useReducer`, `useCallback`, `useMemo`, `useRef`, `useImperativeHandle`, `useLayoutEffect`, `useDebugValue` – will be discussed later during the course

React Hooks Example

[\[https://github.com/iproduct/course-node-express-react/tree/master/04-mybooks-lab4\]](https://github.com/iproduct/course-node-express-react/tree/master/04-mybooks-lab4)

```
const GOOLE_BOOKS_API_BASE = "https://www.googleapis.com/books/v1/volumes?q=";

function App() {
  const [books, setBooks] = useState(mockBooks);
  return (
    <React.Fragment>
      <Nav searchBooks={onSearchBooks} />
      <div className="section no-pad-bot" id="index-banner">
        <div className="container">
          <Header />
          <BookList books={books} />
        </div>
      </div>
      <Footer />
    </React.Fragment>
  );

  async function onSearchBooks(searchText) {
    const booksResp = await fetch(GOOLE_BOOKS_API_BASE + encodeURIComponent(searchText));
    const booksFound = await booksResp.json();
    console.log(booksFound.items);
    setBooks(booksFound.items.map(gbook => ({
      'id': gbook.id,
      'title': gbook.volumeInfo.title,
      'subtitle': gbook.volumeInfo.subtitle,
      'frontPage': gbook.volumeInfo.imageLinks && gbook.volumeInfo.imageLinks.thumbnail
    })))));
  }
}
```

Component Properties Validation (1)

```
React.createClass({  
  propTypes: {  
    // Optional basic JS type properties  
    optionalArray: React.PropTypes.array,  
    optionalBool: React.PropTypes.bool,  
    optionalFunc: React.PropTypes.func,  
    optionalNumber: React.PropTypes.number,  
    optionalObject: React.PropTypes.object,  
    optionalString: React.PropTypes.string,  
    optionalSymbol: React.PropTypes.symbol,
```

Component Properties Validation (2)

// Anything that can be rendered: numbers, strings, elements
or // an array (or fragment) containing these types.

optionalNode: React.PropTypes.node,

// A React element.

optionalElement: React.PropTypes.element,

// You can also declare that a prop is an instance of a class.

optionalMessage: React.PropTypes.instanceOf(Message),

Component Properties Validation (3)

// You can ensure that your prop is limited to specific enum.

```
optionalEnum: React.PropTypes.oneOf(['News', 'Photos']),
```

// An object that could be one of many types

```
optionalUnion: React.PropTypes.oneOfType([
```

```
  React.PropTypes.string,
```

```
  React.PropTypes.number,
```

```
  React.PropTypes.instanceOf(Message)
```

```
]),
```

Component Properties Validation (4)

// An array of a certain type

optArray: React.PropTypes.arrayOf(React.PropTypes.number),

// An object with property values of a certain type

optObject: React.PropTypes.objectOf(React.PropTypes.number),

// An object taking on a particular shape

optionalObjectWithShape: React.PropTypes.shape({

 color: React.PropTypes.string,

 fontSize: React.PropTypes.number

}),

Component Properties Validation (5)

// You can chain any of the above with `isRequired`

requiredFunc: React.PropTypes.func.isRequired,

// A required value of any data type

requiredAny: React.PropTypes.any.isRequired,

// You can also specify a custom validator => return an Error

customProp: function(props, propName, componentName) {

if (!/matchme/.test(props[propName])) {

return new Error('Invalid prop `' + propName + '` supplied
to' +

`'` + componentName + `'. Validation failed.'

); }}

/* ... */

Events in React

- **SyntheticEvent(s)** - event handlers are passed instances of SyntheticEvent – cross-browser wrapper around native events
- Same interface: **stopPropagation()**, **preventDefault()**
- **Event pooling** – all SyntheticEvent(s) are pooled = objects will be reused and all properties will be nullified after the event callback has been invoked (performance) -not for async access:
- Example: <https://facebook.github.io/react/docs/events.html>
- If you want event to be persistent, call: **event.persist()**
- Event types: **Clipboard, Composition, Keyboard, Focus, Form, Mouse, Selection, Touch, UI Events, Wheel, Media, Image, Animation, Transition**

Component Ownership

- **Multiple Components** – allow separation of concerns and reusability
- **Ownership** – an owner is the component that **sets the props** of owned components.
- When a component **X** is created in component **Y**'s **render()** method, it is said that **X** is owned by **Y**.
- Only defined for React components – different from parent-child DOM relationship.
- **Child Reconciliation** – the process by which React updates the DOM with each new render pass. In general, children are reconciled according to the order in which they are rendered.

Reconciliation Example

- // Render Pass 1
- <Card>
- <p>Paragraph 1</p>
- <p>Paragraph 2</p>
- </Card>
- // Render Pass 2
- <Card>
- <p>Paragraph 2</p>
- </Card>

Stateful Children Reconciliation – Keys

```
var ListItemWrapper = React.createClass({

  render: function() {

    return <li>{this.props.data.text}</li>;

  }

});

var MyComponent = React.createClass({

  render: function() {

    return (

      <ul>

        {this.props.results.map(function(result) {

          return <ListItemWrapper key={result.id} data={result}/>;

        })}

      </ul>

    );
  }

});
```

React.Children Utilities

- **React.Children.map**(object children, function fn [, object thisArg]): array – invoke fn on every immediate child contained within children with this set to thisArg
- **React.Children.forEach**(object children, function fn [, object thisArg]) – same as map, but does not return an array
- **React.Children.count**(object children): number - returns children count
- **React.Children.only**(object children): object – returns the only child in children. Throws otherwise
- **React.Children.toArray**(object children): array – returns the children as a flat array with keys assigned to each child

Transferring Props

```
function FancyCheckbox(props) {  
  let { checked, ...other } = props;  
  
  let fancyClass = checked ? 'FancyChecked' : 'FancyUnchecked';  
  
  // `other` contains { onClick: console.log } but not the checked property  
  
  return (  
    <div {...other} className={fancyClass} />  
  );  
}  
  
ReactDOM.render(  
  <FancyCheckbox checked={true} onClick={console.log.bind(console)}>  
    Hello world!  
  </FancyCheckbox>,  
  document.getElementById('example')  
);
```

Forms in React – Controlled Components

- **Interactive Props** - form components support a few props that are affected via user interactions:
 - value - supported by `<input>` and `<textarea>` components
 - checked - supported by `<input>` of type checkbox or radio
 - selected - supported by `<option>` components
- Above form components allow listening for changes by setting a callback to the **onChange** prop:

```
handleAuthorChange(e) { this.setState({author: e.target.value}); }
```

```
<input type="text" value={this.state.author} placeholder="Your name"  
      onChange={this.handleAuthorChange}/>
```

- **Controlled component** does not maintain its own internal state – the component renders purely based on **props**

Refs to Components

- **Refs (references)** – allow to find the **DOM markup** rendered by a component, and invoke methods on **component instances** returned from **render()**
- Example uses: absolute positioning, using React components in larger non-React applications, transition existing code to React.

```
var myComponentInstanceRef =  
ReactDOM.render(<MyComp />, myContainer);  
myComponentInstanceRef.doSomething();
```

- **ReactDOM.findDOMNode(componentInstance)** – this function will return the DOM node belonging to the outermost HTML element returned by render.

The ref Callback Attribute

```
render: function() {  
  
  return (  
  
    <TextInput ref={ function(input) {  
  
      if (input != null) {  
  
        input.focus();  
  
      }  
  
    }} />  
  
  );  
  
},  
  
  • OR using ES6 => :  
  
render: function() {  
  
  return <TextInput ref={(c) => this._input = c} />;  
  
},  
  
componentDidMount: function() {  
  
  this._input.focus();  
  
},  
}
```

The ref String Attribute

- Assign a **ref** attribute to anything returned from **render** such as:

```
<input ref="myInput" />
```

- In some other code (typically event handler code), access the backing instance via **this.refs** as in:

```
var input = this.refs.myInput; // better  
this.refs['myInput'];
```

```
var inputValue = input.value;
```

```
var inputRect = input.getBoundingClientRect();
```

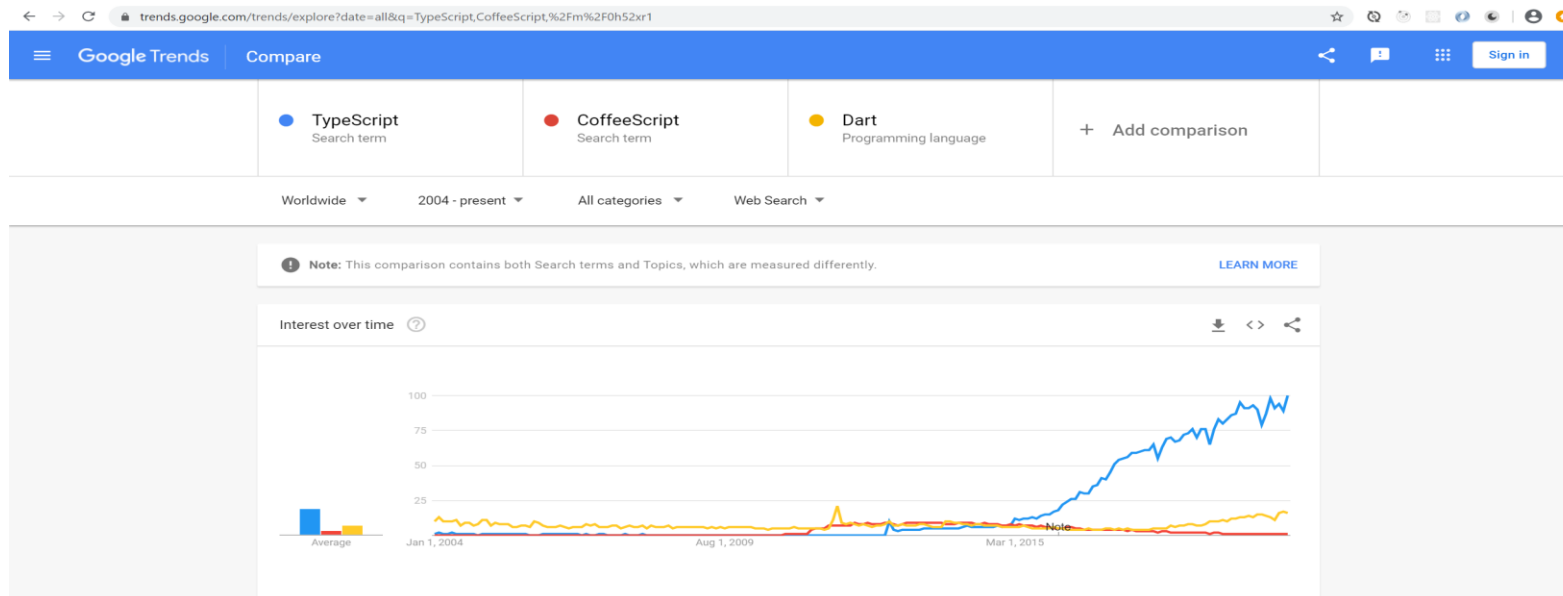
The ref String Attribute - Example

```
var MyComponent = React.createClass({
  handleClick: function() {
    React.findDOMNode(this.refs.myTextInput).focus();
  },
  render: function() {
    return (
      <div>
        <input type="text" ref="myTextInput" />
        <input type="button" value="Focus the text input"
onClick={this.handleClick} />
      </div>
    );
  }
});
```

TypeScript

<http://www.typescriptlang.org/>

- Typescript → since October 2012, Anders Hejlsberg (lead architect of C# and creator of Delphi and Turbo Pascal)
- Targets large scale client-side and mobile applications by compile time type checking + @Decorators -> Microsoft, Google
- TypeScript is strictly superset of JavaScript, so any JS is valid TS



Source: [Google Trends comparison](#)

TypeScript Hello World I

- Installing Typescript: `npm install -g typescript`
- Create new directory: `md 02-ts-demo-lab`
- Create a new TS project using npm: `npm init`
- Write a simple TS function – file `greeter.ts` :

```
function greeter(person: string) {  
    return 'Hello, ' + person + ' from Typescript!';  
}
```

```
const user = 'TypeScript User';  
document.body.innerHTML = greeter(user);
```

TypeScript Hello World II

- Compile TypeScript to JavaScript (ES5): **tsc greeter.ts**
- Include script in **index.html** :

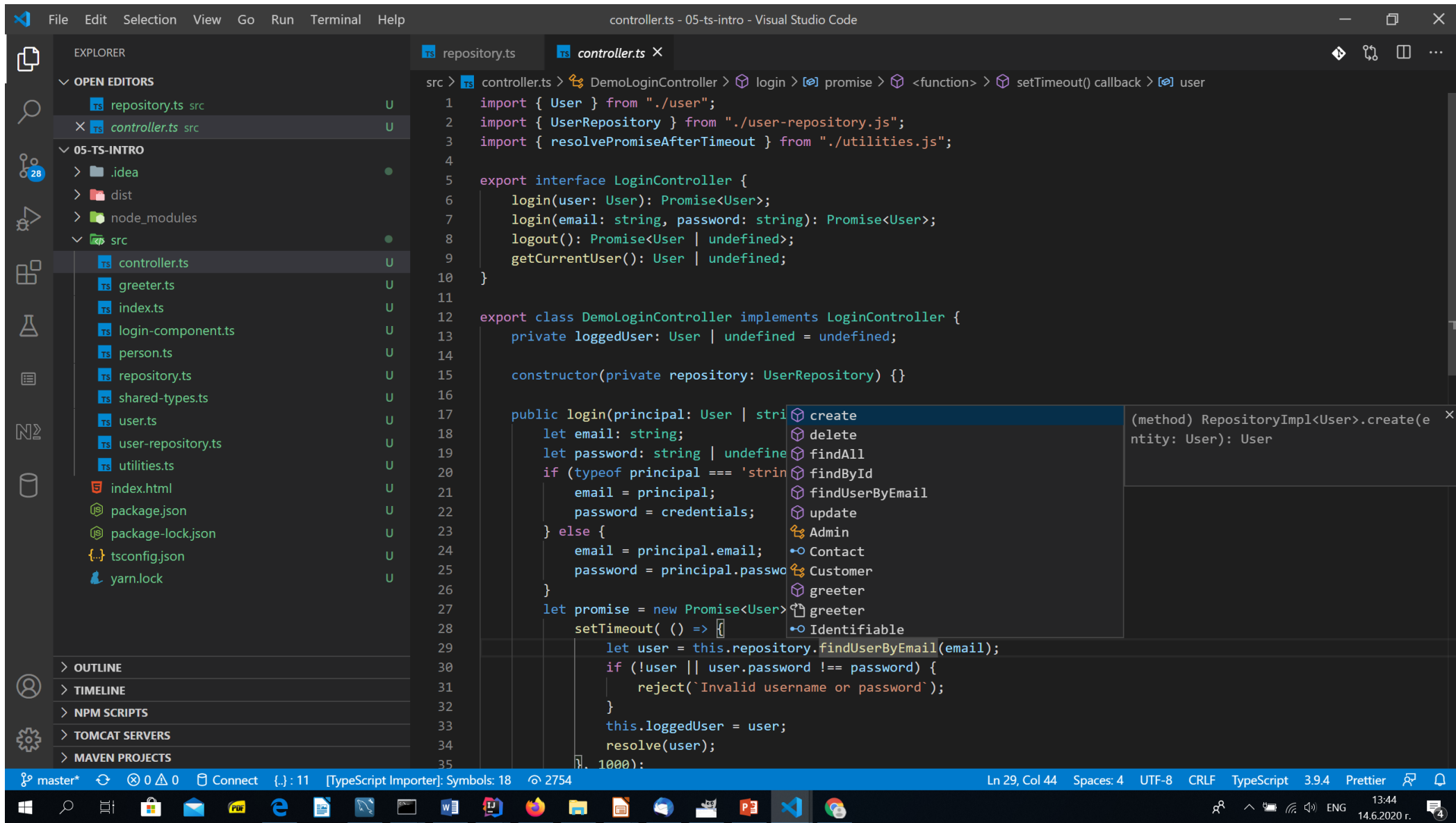
```
<html>
<head>
  <title>ES6 Simple Demo 01</title>
</head>
<body>
  <script src="greeter.js"></script>
</body>
</html>
```

- And open it in web browser – thats all :)
- If you make changes - use watch flag: **tsc -w greeter.ts**

Configuring, Building and Deploying TypeScript Project

- Node package manager (**npm**) configuraton – **package.json**
- TypeScript configuration – **tsconfig.json**, to generate it run: **tsc --init**
- Configuring System.js module loader – **systemjs.config.js**
- Using external JS librarries – **@types** and npm packages
- TypeScript compiler options:
<http://www.typescriptlang.org/docs/handbook/compiler-options.html>
- Linting TypeScript code – **tslint.json**: <https://palantir.github.io/tslint/rules/>
- Developing simple TS project – **Login Demo**

Visual Studio Code



Visual Studio Code

```
10 export default class UserView extends Component<Props> {  
    2 references  
11     render() {  
12         return (  
13             |  
14         );  
15     }  
16  
    0 references  
17     private getYearJoined(user: User) {  
18         return _.padStart(user.dateJoined.format('yyyy'), 6);  
19     }  
20  
    0 references  
21     private getDisplayName(user: User) {
```










VS Code: Popular Angular/TS Plugins I

- **TSLint** - linter for the TypeScript language, help fixing error in TS code. Must have when working with TS. Will be merged with ESLint.
- **ESLint** - linter for the ECMAScript and TypeScript languages, help fixing error in TS code. TSLint announced that it will merge with ESLint:
<https://github.com/typescript-eslint/tslint-to-eslint-config>
- **ES7 React/Redux/GraphQL/React-Native snippets** - This extension provides you JavaScript and React/Redux snippets in ES7 with Babel plugin features for VS Code. **Supported languages:** JavaScript (.js), JavaScript React (.jsx), TypeScript (.ts), TypeScript React (.tsx)
- **JavaScript (ES6) code snippets** – This extension contains code snippets for JavaScript in ES6 syntax for Vs Code editor – supports both JavaScript and TypeScript.
- **Typescript React code snippets** – code snippets for React with Typescript.
- **React + Typescript Code Snippets** – opinionated code snippets for React with Typescript










VS Code: Popular Angular/TS Plugins II

- **TypeScript Hero** - Sorts and organizes your imports according to convention and removes imports that are unused (Ctrl+Alt+o on Win/Linux or Ctrl+Opt+o on MacOS).
- **Path Intellisense** - VSCode has a very good auto import capability, but sometime you still need to import some files manually, and this extension helps a lot in these cases.
- **TypeScript Importer** - Automatically searches for TypeScript definitions in workspace files and provides all known symbols as completion item to allow code completion.
- **Debugger for Chrome** - allows to debug using chrome and add your breakpoints in VSCode.










Let's Install Some VSCode Plugins III

	Auto Import 1.5.3 Automatically finds, parses and provides code actions and code completion for all available imports. Works with Typescript and TSX steoates	959K ⭐ 4.5
★ 	Beautify 1.5.0 Beautify code in place for VS Code HookyQR	4.9M ⭐ 4.5
★ 	Docker 0.8.2 Makes it easy to create, manage, and debug containerized applications. Microsoft	4.7M ⭐ 4.5
	Document This 0.7.1 Automatically generates detailed JSDoc comments in TypeScript and JavaScript files. Joel Day	685K ⭐ 3.5
	ES7 React/Redux/GraphQL/React-Native snippets 2.8.0 Simple extensions for React, Redux and GraphQL in JS/TS with ES7 syntax dsznajder	1.5M ⭐ 4.5
★ 	ESLint 1.9.1 Integrates ESLint JavaScript into VS Code. Dirk Baeumer	9.5M ⭐ 4.5
★ 	Git History 0.6.5 View git log, file history, compare branches or commits Don Jayamanne	2.5M ⭐ 4.5
★ 	GitLens — Git supercharged 10.2.2 Supercharge the Git capabilities built into Visual Studio Code — Visualize code authorship at a glance via Git blame annotations and code lens, seamlessly navigate and ex... Eric Amodio	5.4M ⭐ 5
	JavaScript (ES6) code snippets 1.8.0 Code snippets for JavaScript in ES6 syntax charalampos karypidis	3.5M ⭐ 5

Let's Install Some VSCode Plugins IV

	JSON to TS 1.7.5 Convert JSON object to typescript interfaces MariusAlchimavicius	190K ⭐ 4.5
	Latest TypeScript and Javascript Grammar 0.0.53 This is development branch of VSCode JS/TS colorization. Please file any issues you find against https://github.com/Microsoft/TypeScript-TmLanguage/issues Microsoft	292K ⭐ 3.5
	Material Icon Theme 4.1.0 Material Design Icons for Visual Studio Code Philipp Kief	4.4M ⭐ 5
	Move TS - Move TypeScript files and update relative imports 1.12.0 extension for moving typescript files and folders and updating relative imports in your workspace stringham	291K ⭐ 4.5
	Node Exec 0.5.1 Execute the current file or your selected code with node.js. Miramac	194K ⭐ 5
★ 	npm Intellisense 1.3.0 Visual Studio Code plugin that autocompletes npm modules in import statements Christian Kohler	2.1M ⭐ 4.5
	Nx Console 12.0.0 Nx Console for Visual Studio Code nrwl	437K ⭐ 3.5
	Paste JSON as Code 12.0.46 Copy JSON, paste as Go, TypeScript, C#, C++ and more. quicktype	500K ⭐ 4.5
	Peacock 3.7.2 Subtly change the workspace color of your workspace. Ideal when you have multiple VS Code instances and you want to quickly identify which is which. John Papa	754K ⭐ 4.5

Let's Install Some VSCode Plugins V

	React + Typescript Code Snippets 2.1.0 Code snippets for react in typescript weffe	1K
	refactorix 0.3.6 TypeScript refactoring tools for Visual Studio Code Christian Oetterli	110K ★ 5
	Simple React Snippets 1.2.2 Dead simple React snippets you will actually use Burke Holland	511K ★ 5
	TSLint 1.2.3 TSLint support for Visual Studio Code Microsoft	1.9M ★ 3
	TypeScript Hero 3.0.0 Additional toolings for typescript Christoph Bühler	593K ★ 3.5
	TypeScript Importer 2.0.1 Automatically searches for TypeScript definitions in workspace files and provides all known symbols as completion item to allow code completion. pmneo	276K ★ 4.5
	Typescript React code snippets 1.3.1 Code snippets for react in typescript infeng	98K ★ 4.5
	Typescript React/Redux Snippets 0.2.0 Typescript React Snippets Alexander Botteram	15K
	TypeScript Toolbox 0.5.0 Add and Optimize Imports, Generate Getters / Setters and Constructors DSKWRK	143K ★ 3.5

Introduction to TypeScript I

- Functions, interfaces, classes and constructors.
- Common types – **Boolean, Number, String, Array, Tuple, Enum, Any, Void, Null, Undefined**.
- **--strictNullChecks** flag
- Type assertions: `let length: number = (<string> data).length;`
`let length: number = (data as string).length;`
- **Duck typing** = structural similarity based typing
- Declaring variables – **let**, **const** and **var**. Scopes, variable capturing, Immediately Invoked Function Expressions (IIFE), closures and **let**.

Declaring Contracts using Interfaces I

```
export interface User {  
  id: number;  
  firstName: string;  
  lastName: string;  
  email: string;  
  password: string;  
  contact?: Contact;      ← Optional properties  
  roles: Role[];  
  getSalutation(): string; ← Required methods  
}
```


Declaring Contracts using Interfaces II

```
import { User } from './users';
export interface UserRepository {
  addUser(user: User): void;
  editUser(user: User): void;
  ...
  findAllUsers(): User[];
}
export class DemoUserRepository implements UserRepository {
  private users = new Map<number, User>();
  public addUser(user: User): void {
    if (this.findUserByEmail(user.email)) {
      throw `User with email ${user.email} exists.`;
    }
    user.id = this.getNextId();
    this.users.set(user.id, user);
  }
  ...
}
```

Declaring Contracts using Interfaces II

- Properties, optional properties and readonly properties:

```
interface UserRepository {  
    readonly size: number;  
    addUser: (user: User) => void;  
}
```

- Function types:

```
interface RoleFinder {  
    (user: User) : Role[];  
}
```

- Array (indexable) types. Dictionary pattern:

```
interface EmailUserMap {  
    [key: string]: User;  
}
```

Class Types

```
export class Customer implements User {  
    public id: number; // set automatically by repository  
    constructor(public firstName: string,  
        public lastName: string,  
        public email: string,  
        public password: string,  
        public contacts?: Contact,  
        public roles: Array<Role> = [ Role.CUSTOMER ]) {  
    }  
    public get salutation() {  
        return `${this.firstName} ${this.lastName}  
            in role ${Role[this.roles[0]]}`;  
    }  
}
```

Default value
↓

Extension of Interfaces. Hybrid Types.

```
export interface Person {  
    id: number;  
    firstName: string;  
    lastName: string;  
    email: string;  
    contact?: Contact;  
}
```

```
export interface User extends Person{  
    password: string;  
    roles: Role[];  
    getSalutation(): string;  
}
```

Classes

- Constructors, constructor arguments as properties
- Public/private/protected properties
- Get and set accessors

```
export interface User extends Person{  
    password: string;  
    roles: Role[];  
    readonly salutation: string;  
} ...  
public get salutation() {  
    return `${this.firstName} ${this.lastName}`;  
}
```

- Static and instance sides of a class. Abstract classes

Functions and Function Types

- Optional, default and **rest (...)** parameters

```
export class Person {  
  public restNames: string[];  
  constructor(public firstName: string, ...restNames: string[]) {  
    this.restNames = restNames;  
  }  
  public get salutation() {  
    let salutation = this.firstName;  
    for (let name of this.restNames) {  
      salutation += ' ' + name;  
    }  
    return salutation;  
  }  
}  
console.log(new Person('Ivan', 'Donchev', 'Petrov').salutation);
```

Function Lambdas (=>) and Use of this

```
export class LoginComponent {  
    constructor(private jqElem: string, private loginC: LoginController){  
        const keyboardEventHandler = (event: JQueryKeyEventObject) => {  
            if (event.keyCode === 13) {  
                loginEventHandler();  
            }  
        };  
        const loginEventHandler = () => {  
            this.loginC.login(usernameInput.val(), passwordInput.val())  
                .then(() => {  
                    this.showCurrentUser();  
                }).catch(err => {  
                    this.showError(err);  
                });  
            return false;  
        };  
        ...  
    }  
}
```

Type Guards & Method Overloading

```
export class DemoLoginController implements LoginController {
  public login(email: string, password: string): Promise<User>;
  public login(user: User): Promise<User>;
  public login(principal: User | string, credentials?: string)
    : Promise<User> {
    let email: string, password: string;
    if (typeof principal === 'string') {
      email = principal;
      password = credentials;
    } else {
      email = principal.email;
      password = principal.password;
    }
    let promise = new Promise<User>( (resolve, reject) => { ... });
    return promise;
  }
}
```


Using Enums

- Defining enumeration:

```
enum Role {  
    ADMIN = 1, CUSTOMER  
}
```

- In generated code an enum is compiled into an object that stores both forward (name -> value) and reverse (value -> name) mappings.
- Getting enumerated name by value:

```
public get salutation() {  
    return `${this.name} in role ${Role[this.roles[0]]}`;  
}
```

JavaScript Module Systems – ES6

```
// lib/math.js
```

```
export function sum (x, y) { return x + y }
```

```
export var pi = 3.141593
```

```
// someApp.js
```

```
import * as math from "lib/math"
```

```
console.log("2 $\pi$  = " + math.sum(math.pi, math.pi))
```

```
// otherApp.js
```

```
import { sum, pi } from "lib/math"
```

```
console.log("2 $\pi$  = " + sum(pi, pi))
```

Modules in TypeScript

- **Namespaces** and **modules** – former internal and external modules – prefer **namespace X { ... }** instead **module X { ... }**
- ES6 modules (preferred) – using **export** and **import**:

```
export interface Person { ... }
```

```
export interface User extends Person { ... }
```

```
export interface Contact { ... }
```

```
export enum Role { ADMIN, CUSTOMER }
```

```
export class Customer implements User { ... }
```

```
import { User } from './users';
```

```
import { resolvePromiseAfterTimeout } from './utilities';
```

```
import $ from 'jquery';
```

```
import * as repository from './user-repository'; // default import
```

```
import './my-module.js'; // import a module for side-effects only
```

```
let module = await import('/modules/my-module.js'); // dynamic
```

Interoperability with External JS Libraries

- Ambient type declarations and ambient modules (*.d.ts) – typings, @types – Example **node.d.ts** (simplified):

```
declare module "url" {  
    export interface Url {  
        protocol?: string;  
        hostname?: string;  
        pathname?: string;  
    }  
    export function parse(urlStr: string, parseQueryString?,  
slashesDenoteHost?): Url;  
}  
/// <reference path="node.d.ts"/>  
import * as URL from "url";  
let myUrl = URL.parse("https://github.com/iproduct");
```

Generic Type Parameters I

- Writing generic functions, interfaces and classes – Examples:

```
interface Repository {  
    findById<T>(id: number): T;  
    findAll<T>(): Array<T>;  
}
```

//OR

```
interface Repository<T> {  
    findById: (id: number) => T;  
    findAll(): Array<T>;  
}
```

Generic Type Parameters II

```
export class RepositoryImpl<T> implements Repository<T> {  
    private data = new Map<number, T>();  
    public findById(id: number): T {  
        return this.data.get(id);  
    }  
    public findAll(): T[] {  
        let results: T[] = [];  
        this.data.forEach(item => results.push(item));  
        return results;  
    }  
}
```

- Bounded generics
- Generic constructors

Advanced TypeScript Topics

- Advanced types
- Type Guards and Differentiating Types
- Type Aliases
- Symbols
- Declaration merging
- Decorators
- Using mixins in TypeScript

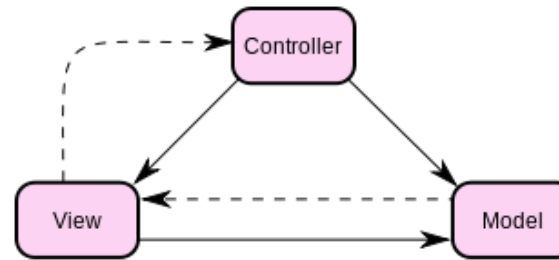
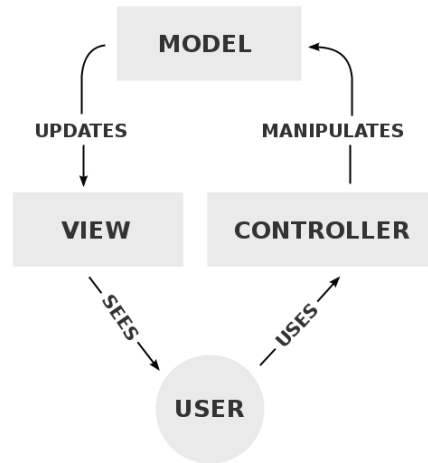
Exercise: Users Login Demo

<https://github.com/iproduct/fullstack-typescript-react/tree/master/05-ts-intro>

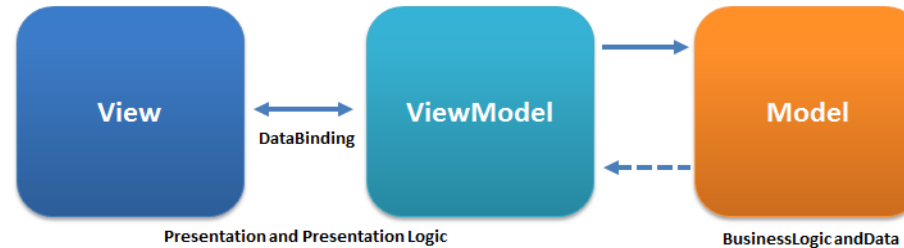


MVC Comes in Different Flavors

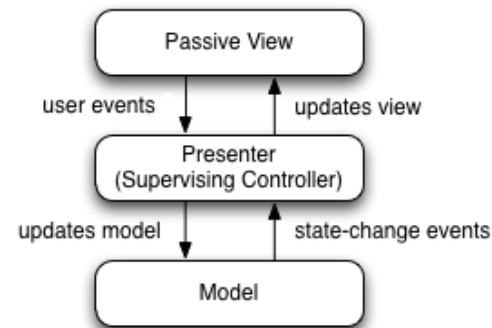
- MVC



- MVVM



- MVP

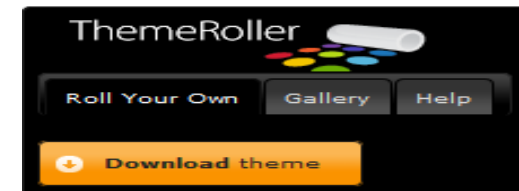


jQuery

- **jQuery** is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development [<http://jquery.com/>]
- **Lightweight Footprint** - about 31KB in size (Minified and Gzipped)
- **Easy-to-use but powerfull** – Ajax, Attributes, Callbacks Object, Core, CSS, Data, Deferred Object, Dimensions, Effects, Events, Forms, Internals, Manipulation, Miscellaneous, Offset, Plugins, Properties, Selectors, Traversing, Utilities
- **Widespread JS library** with many **third-party plugins**

jQuery [2]

- Supports CSS 3 selectors and much more
- [jQuery 3.x browser support](#) – IE: 9+, Chrome, Edge, Firefox, Safari: Current/ -1, Opera: Current, Safari iOS: 7+, Android 4.0+
- Supports own layout and presentation widgets – jQueryUI
 - Interactions – Drag/Droppable, Resizable, Selectable, Sortable
 - Widgets – Accordion, Autocomplete, Button, Datepicker, Dialog, Menu, Progressbar, Slider, Spinner, Tabs, Tooltip
 - Effects – Add Class, Color Animation, Effect, Hide, Remove Class, Show, Switch Class, Toggle, Toggle Class
 - Utilities – Position, Widget Factory
- Supports custom themes (CSS)



Using JavaScript Libraries in TypeScript: JQuery I

```
import { LoginController } from './controller';  
import '../node_modules/jquery/dist/jquery.js';
```

```
export class LoginComponent {  
  private messagesElement: JQuery;  
  constructor(private jqElementSelector: string, private loginController: LoginController) {  
  
    const keyboardEventHandler = (event: JQuery.Event) => {  
      if (event.keyCode === 13) {  
        loginEventHandler();  
      }  
    };  
  
    const loginEventHandler: any = () => {  
      this.loginController.login(usernameInputElem.val() as string, passwordInputElem.val() as string)  
        .then(() => {  
          this.showCurrentUser();  
        }).catch(err => {  
          this.showError(err);  
        });  
      return false;  
    };  
  }  
};
```

(- continues -)

Using JavaScript Libraries in TypeScript: JQuery II

```
const formElem = $("<form class='form-inline' role='form'>").addClass('form-inline');
const usernameInputElem: JQuery<HTMLElement> =
    $("<input id='username' type='email' placeholder='email'>")
    .addClass('form-control').bind('keypress', keyboardEventHandler);
const passwordInputElem: JQuery<HTMLElement> =
    $("<input id='password' type='password' ", keyboardEventHandler);
const loginButtonElem: JQuery<HTMLElement> =
    $('<button>Login</buttton>').addClass('btn btn-primary')
    .click(loginEventHandler);

// build the login form
formElem.append(usernameInputElem);
formElem.append(passwordInputElem);
formElem.append(loginButtonElem);
this.messagesElement = $('<div id="message" class="well well-lg">');
$(jqElementSelector).append(formElem).append(this.messagesElement);

this.showCurrentUser();
}

public showCurrentUser(): void {
    const user = this.loginController.getCurrentUser();
    this.messagesElement.html(user ? `Welcome ${user.salutation}` : `No user is logged in.`);
}
```

Webpack Builder & Bundler Tool

webpack - Mozilla Firefox

File Edit View History Bookmarks Tools Help

we X Home Media Lib AdWc AdWc M [ANS] P How t P Mar 2 P Green HTML Web Softw Monit 25 Googl T CALL P Open BIA Flying FX Impro Graph f > + v

https://webpack.js.org

Most Visited Getting Started Latest Headlines Scientific American To... Портфолио от услуг... Amazon.de: OMEN 17... Портфолио от услуг... Guest IPT курс Web 2.0 и Ajax GIMP Plugin Registry -... Greg Murray's Blog

Sponsor webpack and get apparel from the [official shop](#) or get stickers [here](#)! All proceeds go to our [open collective](#)!

webpack DOCUMENTATION CONTRIBUTE VOTE BLOG

bundle your scripts

MODULES WITH DEPENDENCIES

STATIC ASSETS

10:08 PM 4/1/2018

Creating New Project: NPM + WebPack + React

```
mkdir my-project
cd my-project
npm init
npm install --save-dev webpack webpack-cli webpack-dev-server
touch index.html src/index.js webpack.config.js
npm install --save react react-dom
npm install --save-dev @types/react @types/react-dom
npm install --save-dev typescript ts-loader source-map-loader
// OR npm install typescript awesome-typescript-loader --save-dev
npm install css-loader style-loader css-to-string-loader --save-dev
npm install file-loader url-loader html-loader clean-webpack-plugin --save-dev
npm install extract-text-webpack-plugin html-webpack-plugin --save-dev
npm i --save-dev eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin eslint-plugin-react
```

In package.json:

```
"scripts": {
  "start": "webpack-dev-server --inline --hot --open",
  "watch": "webpack --watch",
  "build": "webpack -p"
},
```

Minimal tsconfig.json

```
{
  "compilerOptions": {
    "outDir": "./dist/",
    "sourceMap": true,
    "noImplicitAny": true,
    "module": "commonjs",
    "target": "ESNext",
    "jsx": "react"
  },
  "include": [
    "src/**/*" // *** The files TypeScript should type check ***
  ],
  "exclude": ["node_modules", "build"] // *** The files to not type check ***
}
```

- For more complete TS config see:

<https://www.sitepoint.com/react-with-typescript-best-practices/>

React Component File: /src/components/Hello.tsx

```
import * as React from 'react';
```

```
export interface HelloProps {  
  compiler: string;  
  framework: string;  
}
```

// 'HelloProps' describes the shape of props. State is never set so we use the '{}' type.

```
export class Hello extends React.Component<HelloProps, {}> {  
  render() {  
    return (  
      <h1>  
        Hello from {this.props.compiler} and {this.props.framework}!  
      </h1>  
    );  
  }  
}
```

React Main File: /src/index.tsx

```
import * as React from "react";
import * as ReactDOM from "react-dom";

import { Hello } from "../components/Hello";

ReactDOM.render(
  <Hello compiler="TypeScript" framework="React" />,
  document.getElementById("demo")
);
```

Simple webpack.config.js I

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const { CleanWebpackPlugin } = require('clean-webpack-plugin');
module.exports = {
  mode: 'development',
  entry: {
    app: './src/index.tsx',
  },
  devtool: 'inline-source-map',
  devServer: {
    contentBase: './dist',
    compress: true,
    hot: true,
    port: 9000
  },
  plugins: [
    new CleanWebpackPlugin({ cleanStaleWebpackAssets: false }),
    new HtmlWebpackPlugin({
      template: path.join(__dirname, 'index.html'),
      title: 'Hello React',
    }),
  ],
}
```

Simple webpack.config.js II

```
output: {
  filename: '[name].bundle.js',
  path: path.resolve(__dirname, 'dist'),
},
resolve: {
  extensions: ['.wasm', '.ts', '.tsx', '.mjs', '.js', '.json'],
},

module: {
  rules: [
    {
      test: /\.ts(x?)$/,
      exclude: /node_modules/,
      use: [
        {
          loader: "awesome-typescript-loader"
        }
      ]
    }
  ]
},
```

Simple webpack.config.js III

```
// optimization
optimization: {
  splitChunks: {
    cacheGroups: {
      default: false,
      vendors: false,
      // vendor chunk
      vendor: {
        // sync + async chunks
        chunks: 'all',
        // import file path containing node_modules
        test: /node_modules/
      }
    }
  }
};
```

Webpack Loaders and Plugins

- Loaders are transformations (functions running in node.js) that are applied on a resource file of your app
- You can use loaders to load [ES6/7](#) or [TypeScript](#)
- Loaders can be chained in a pipeline to the resource. The final loader is expected to return JavaScript
- Loaders can be synchronous or [asynchronous](#)
- Loaders accept [query parameters](#) – loader configuration
- Loaders can be [bound to extensions / RegExps](#)
- Loaders can be published / installed through [npm](#)
- [Plugins](#) – more universal than loaders, provide [more features](#)

Webpack Loaders

- [ts-loader](#), [awesome-typescript-loader](#) -TypeScript => ES 5 or 6
- [babel-loader](#) - turns ES6 code into vanilla ES5 using Babel
- [file-loader](#) - emits the file into the output folder and returns the url
- [url-loader](#) - like file loader, but returns Data Url if file size <= limit
- [extract-loader](#) - prepares HTML and CSS modules to be extracted into separate files (alt. to ExtractTextWebpackPlugin)
- [html-loader](#) - exports HTML as string, requiring static resources
- [style-loader](#) - adds exports of a module as style to DOM
- [css-loader](#) - loads css file resolving imports and returns css code
- [sass-loader](#) - loads and compiles a SASS/SCSS file
- [postcss-loader](#) - loads and transforms a CSS file using PostCSS
- [raw-loader](#) - lets you import files as a string

Webpack Main Plugins

- [ExtractTextWebpackPlugin](#) - extracts CSS from your bundles into a separate file (e.g. app.bundle.css) → **mini-css-extract-plugin**
- [CompressionWebpackPlugin](#) - prepare compressed versions of assets to serve them with Content-Encoding
- [I18nWebpackPlugin](#) - adds i18n support to your bundles
- [HtmlWebpackPlugin](#) - simplifies creation of HTML files ([index.html](#)) to serve your bundles
- [ProvidePlugin](#) - automatically loads modules, whenever used
- [UglifyJsPlugin](#) – tree transformer and compressor which reduces the code size by applying various optimizations
- ~~[CommonsChunkPlugin](#)~~ - generates chunks of common modules shared between entry points and splits them to separate bundles → **SplitChunksPlugin**

Resources

- TypeScript Quick start –
<http://www.typescriptlang.org/docs/tutorial.html>
- TypeScript Handbook –
<http://www.typescriptlang.org/docs/handbook/basic-types.html>
- Official Webpack repo @ GitHub –
<https://github.com/webpack/webpack>
- Webpack: An Introduction (Angular website):
<https://v5.angular.io/guide/webpack>

Thank's for Your Attention!



Trayan Iliev

IPT – Intellectual Products & Technologies

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>