Using JavaScript Frameworks

React Bascis

React

JavaScript library for building UIs

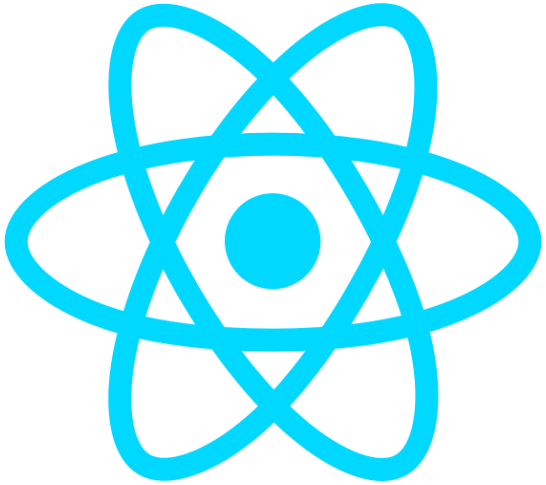
- Focus is on code, not templates
- The V in MVC

React Components are composable

One-way reactive data flow

- Two-way data binding is supported, but is not the default

Handles DOM manipulations efficiently using a virtual DOM



JSX / TSX

JSX is a JavaScript syntax extension that looks similar to XML

TSX must be compiled to JavaScript

```
// Input (JSX):
var app = <Nav color="blue" />;
// Output (JS):
var app = React.createElement(Nav, {color:"blue"});
```

Exploring JSX / TSX

When a JSX element is transpiled, the result is a **createElement** call in the outputted code

JSX elements are not HTML and they are not string content, the elements are nested function calls

JSX looks like HTML, which allows the developer to think of the UI structure as a traditional HTML markup structure, but under the hood the elements are only function calls

TSX is a convention for JSX written in TypeScript

React Component

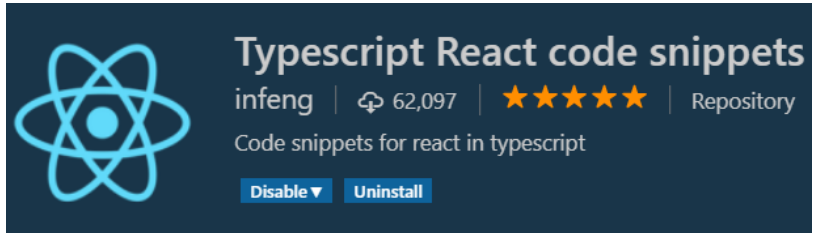
A React app is a set of nested Components

Components utilize three kinds of data: props, state and context

Props, state and context indicate the relationship of the component to a particular data

Renders content using render()

Create additional React Components



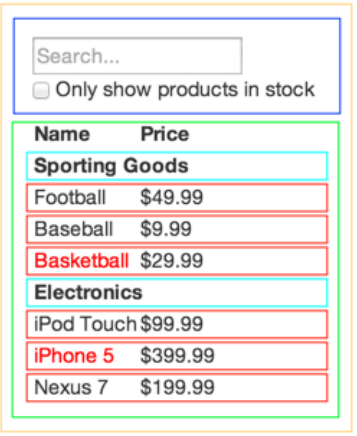
Add File with ext *.tsx

Use Triggers from extension

Add Import!

Trigger	Content
tsrcc→	class component skeleton
tsrcfull→	class component skeleton with Props, State, and constructor
tsrcjc→	class component skeleton without import and default export lines
tsrpcc→	class purecomponent skeleton
tsrpcjc→	class purecomponent without import and default export lines
tsrpfc	pure function component skeleton
tsrsfc	stateless functional component
conc→	class default constructor with props and context
cwm→	componentWillMount method
ren→	render method
cdm→	componentDidMount method
cwrp→	componentWillReceiveProps method
scu→	shouldComponentUpdate method
cwu→	componentWillUpdate method
cdu→	componentDidUpdate method
cwum→	componentWillUnmount method

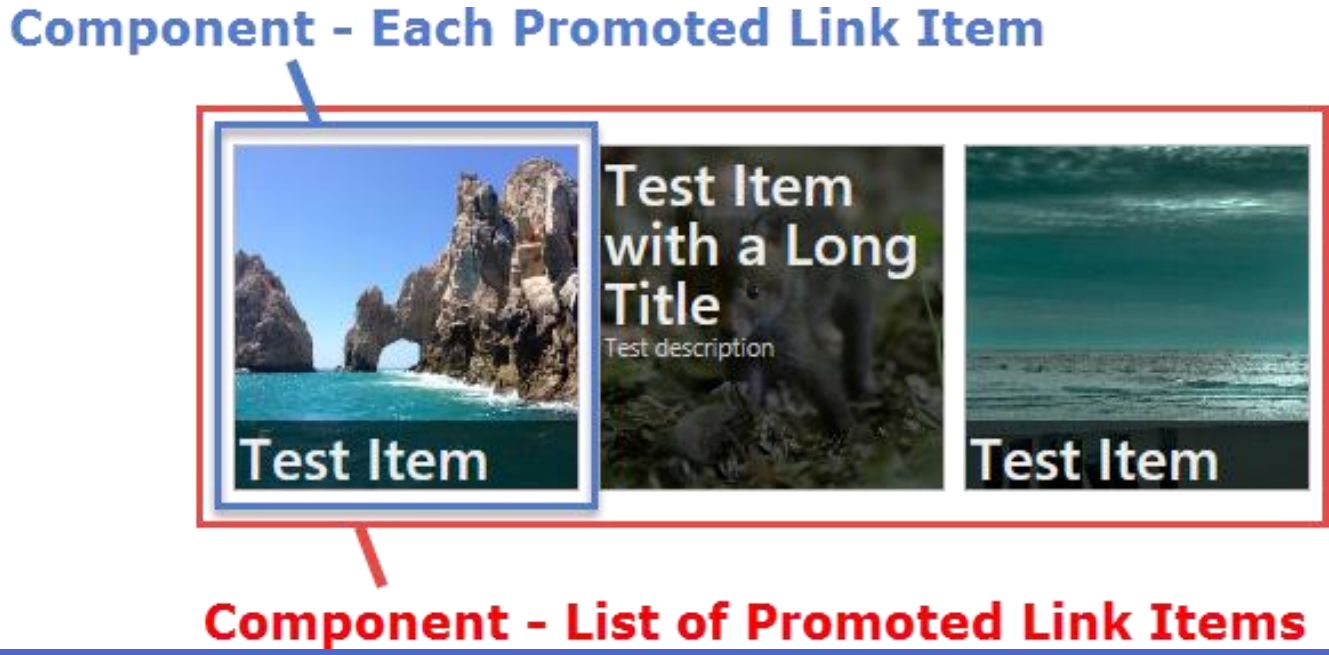
Thinking in React



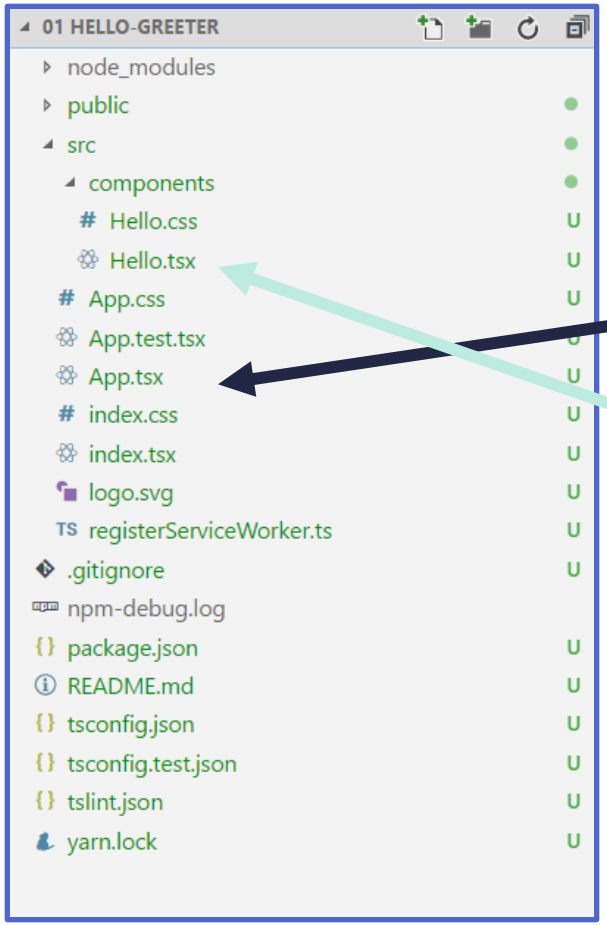
Split UI into multiple components – a component should do only one thing

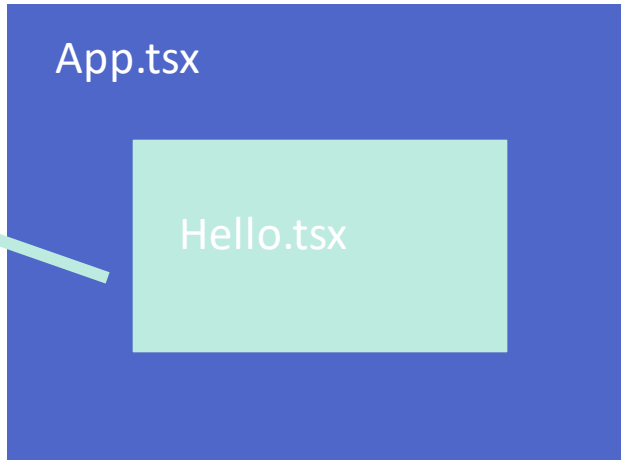
Build in phases – start with a static version and add functionality

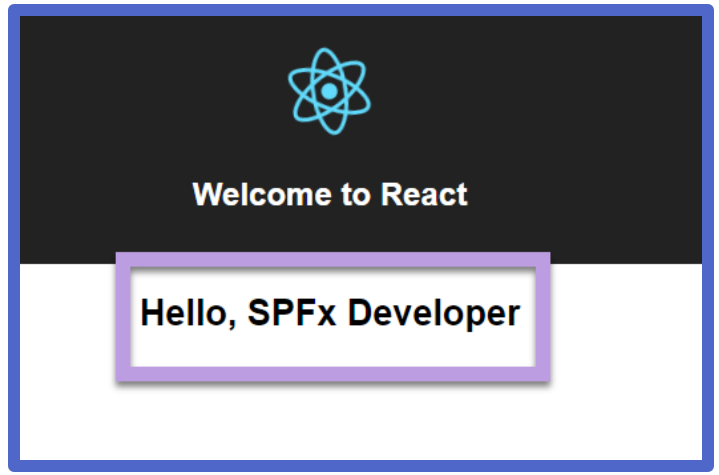
Do not set state directly – use setState()



Nesting Components – Hello Greeter





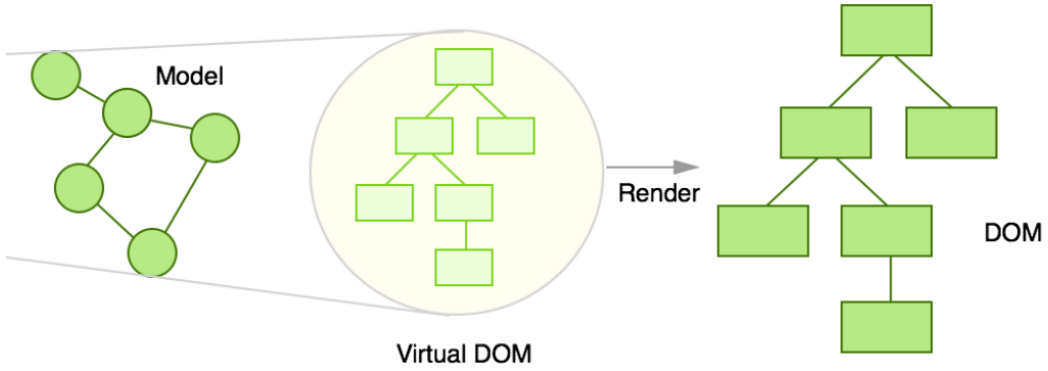


Virtual DOM

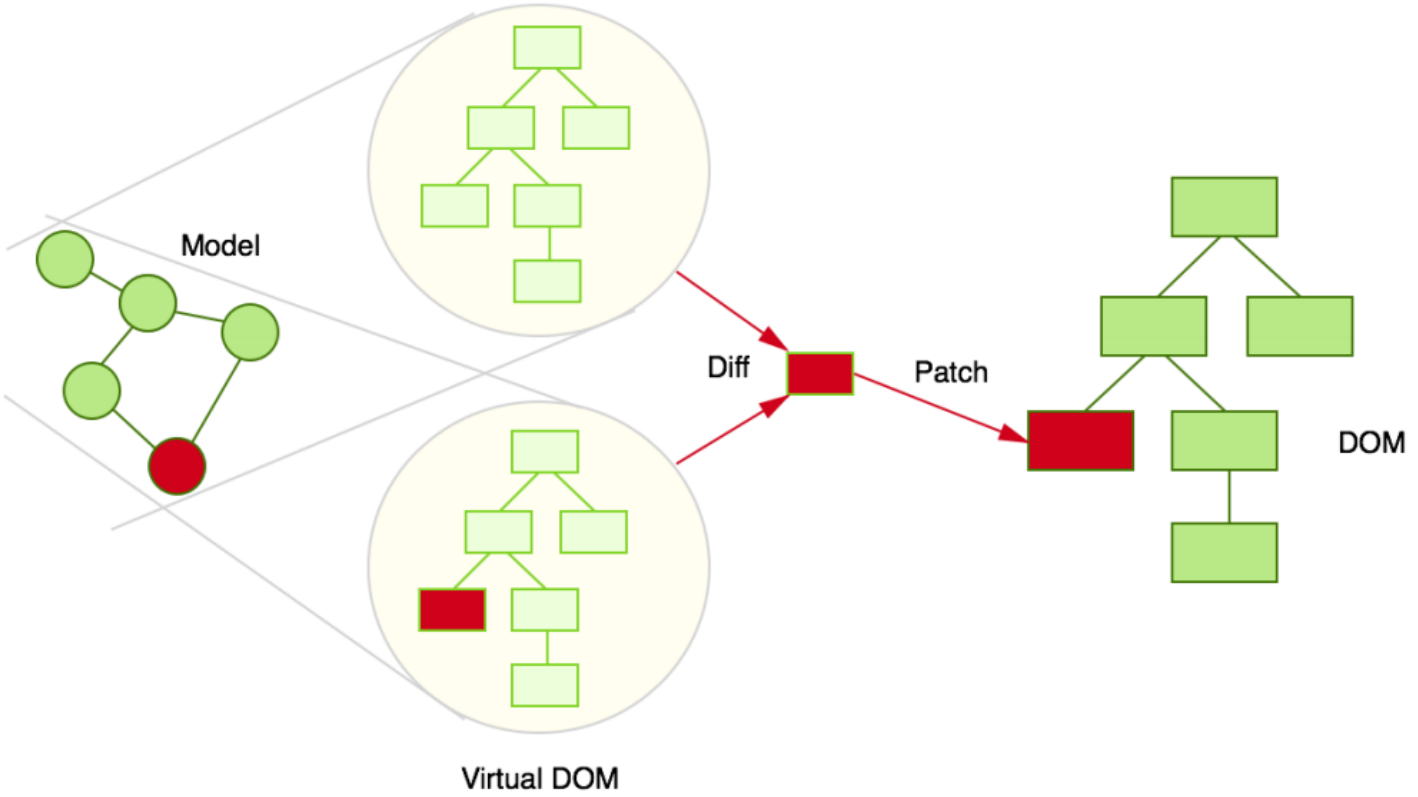
Keep track of state in DOM is hard

The DOM API is slow when re-render everything on change

Virtual DOM calculates Delta – applies Delta to DOM



Virtual DOM - Change Detection



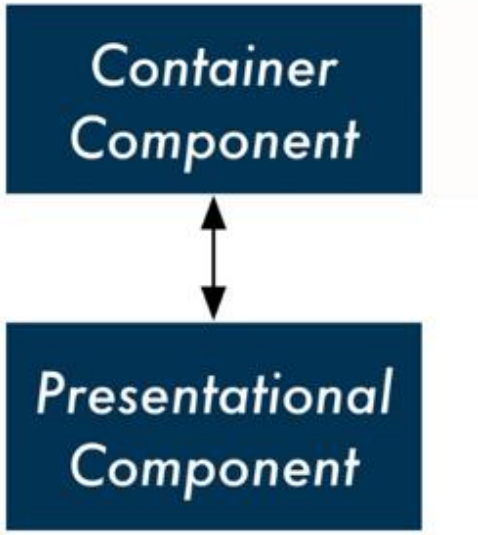
Container vs Presentational Components

Container Components corresponds to the „Page“ holding all other „Artifacts“

Presentational Components represent an aspect of a view

Containers hold one or more Presentational Components

Containers manage State



App.tsx

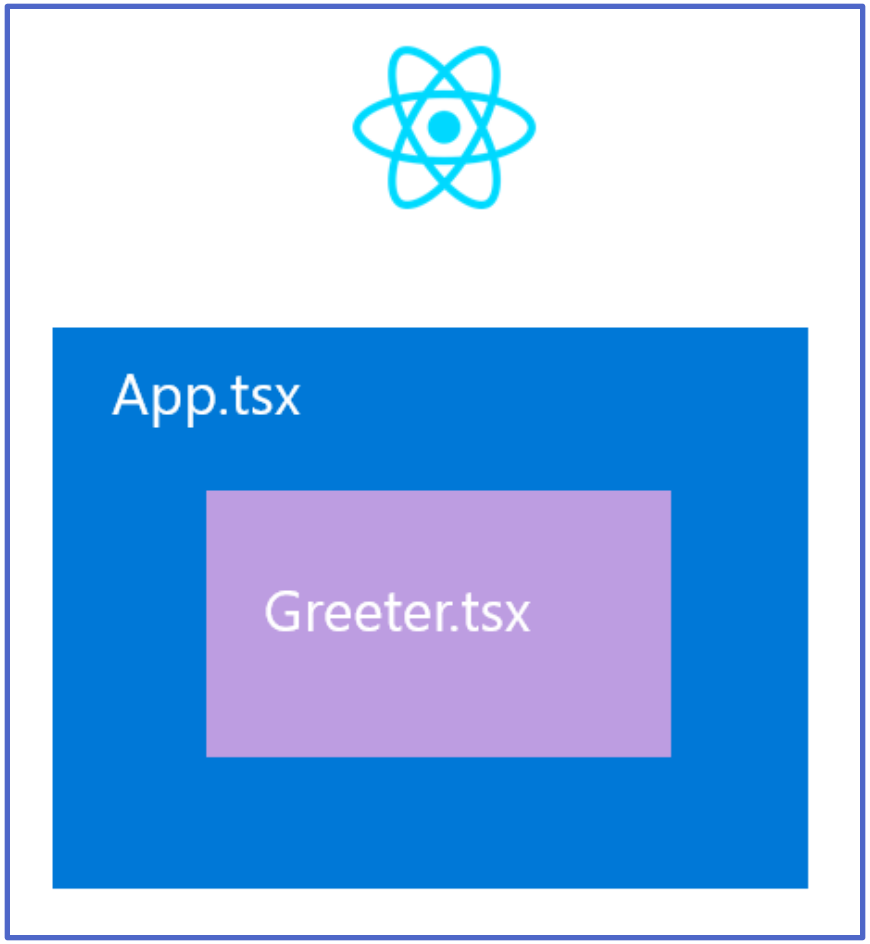
App.tsx

```tsx
import { Hello } from './components/Hello';
import * as React from 'react';
import './App.css';

const logo = require('./logo.svg');

class App extends React.Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          <Hello></Hello>
        </p>
      </div>
    );
  }
}

export default App;
```



Hello.tsx

```tsx
import * as React from 'react'
import './Hello.css'

export class Hello extends React.Component<any, any> {

    user = {
        firstName: 'SPFx',
        lastName: 'Developer'
    };

    render() {
      return (
        <div>
            <h2>
                Hello, {this.formatName(this.user)}
            </h2>
        </div>
      );
    }

    formatName(user:any) {
        return user.firstName + ' ' + user.lastName;
    }
  }

  export default Hello;
```

Utility Function

Iteration

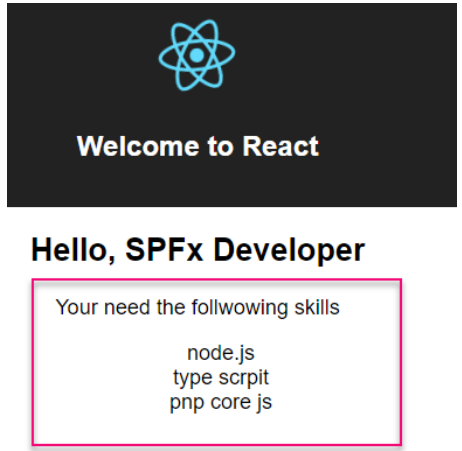
Use map operator

Assign a unique „key“ to each iterated item to allow minimal DOM change

```
export interface SkillsProps {
    id: number;
    name: string;
}

export default class Skills extends React.Component<SkillsProps, any> {
  skills: SkillsProps[] = [{id: 1, name: 'node.js'},
    {id: 1, name: 'type script'}, {id: 1, name: 'pnp js core'}];

  render() {
    return (
      <div>
        <div>Your need the follwowing skills</div>
        <ul className="{li-skill}">
            {
                this.skills.map( (item) =>{
                 return <li key={item.id}>{item.name}</li>
            })}
        </ul>
      </div>
    );
  }
}
```



Handling Events

Attach Handler using

- Arrow Function or
- bind() expression

```
export default class Skills extends React.Component<any, any> {
  skills: Skill[] = [{id: 1, name: 'node.js'},
    {id: 1, name: 'type script'}, {id: 1, name: 'pnp core js'}];

  render() {
    return (
      <div>
        <div>Your need the follwowing skills</div>
        <ul className="{li-skill}">
            {
                this.skills.map( (item) =>{
                  //return <li key={item.id} onClick={this.skillClicked.bind(this,item)}>{item.name}</li>
                  return <li key={item.id} onClick={()=>this.skillClicked(item)}>{item.name}</li>
                })}
        </ul>
      </div>
    );
  }

  skillClicked(skill: Skill){
    console.log(`You clicked skill ${skill.name}`)
  }
}
```

Using CSS

Import css using „import './Skills.css‘;”

Use className to attack CSS class

```
render() {
  return (
    <div className="container">
      <div>Your need the follwowing skills</div>
      <ul>
          {
              this.skills.map( (item) =>{
                  return <li key={item.id} onClick={()=>this.skillClicked(item) } className="li-skills" >{item.name}</li>
              })}
      </ul>
    </div>
  );
}
```

```
.container {
    clear: both;
    width: 300px;
    text-align: center;
}

ul{
    background: lavender;
}

.li-skills{
    border-bottom: 1px solid black;
    list-style-type: none;
}
```

Component Lifecycle Methods

render() – process the virtual DOM and update the display

getInitialState() – initial state value

getDefaultProps() – fallback for when props are not supplied

setState(state) – triggers UI updates

componentWillMount() – Invoked once, on both client & server before rendering occurs.

componentDidMount() – Invoked once, only on the client, after rendering occurs. **LOAD DATA HERE**

shouldComponentUpdate() – Return value determines whether component should update.

componentWillUnmount() – Invoked prior to unmounting component.

Angular

What is a Single Page Application (SPA)

A web application that is implemented using a single *.html-file

- Loads the content (template) of the URL-Segments (Routes) into a Div-Like container

Providing a more fluent user experience - similar to a desktop application

In a SPA, either all necessary assets like HTML, JavaScript, and CSS –

- Are retrieved with a single page load, or
- Resources are dynamically loaded and added to the page, usually in response to user actions

Most SPA‘s are implemented using JavaScript Frameworks like

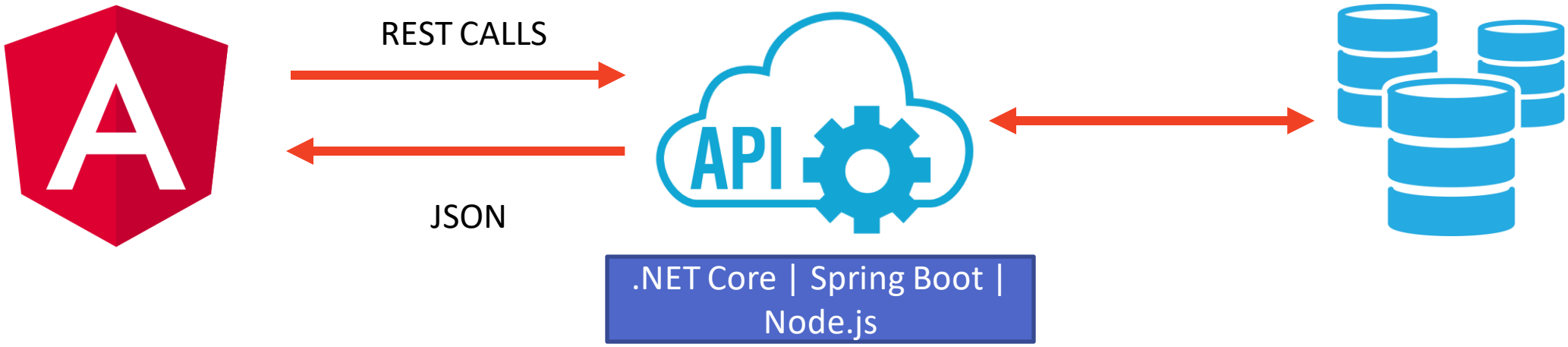
- Angular -> Google
- React -> Facebook
- Vue.js -> Started by Evan You, Ex-Google from Angular JS

Angular SPA Architecture

Provides the Front-End / User Interface

- Runs in a Container / Blob Storage / on a Web Server
- Is secured be Token Based Authentication ->
  - Forwards the token in the Header of the HttpRequest to the REST Api that consumes the same Identity Provider

Typically consumes one or more REST Apis that return JSON



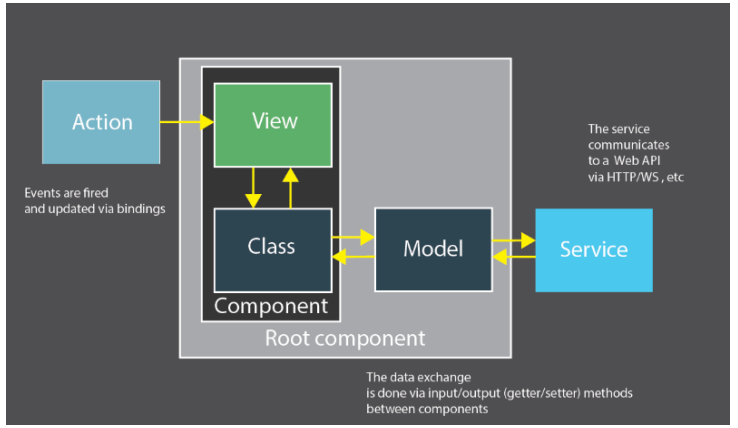
Component

An Angular App consists of one or more [nested] components

It defines:

- A TypeScript Class that acts as Controller
  - Defines Metadata like a selector using a Decorator -> @Component
- View: HTML | Inline
- Styles: CSS | SCSS
- @Input | @Output are used to exchange data with parent Components
- …

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent implements OnInit {
```



Routing

Routing is the process to switch from one view to another

- ... From one component to another
- Routing is achieved using Angular Router (V3)

Main Routing is configured in app-routing.module.ts

- Each module can have its own routing

Router Links are used for navigation

```
const routes: Routes = [
  {
    path: '',
    component: HomeComponent,
  },
  {
    path: 'skills',
    component: SkillsListComponent,
  },
  {
    path: 'skills/:id',
    component: SkillsEditComponent
  },
```

```
<nav>
  <a routerLink="/">Home</a>
  <a routerLink="/skills">Skills</a>
</nav>
<div>
    <router-outlet></router-outlet>
</div>
```

Angular Technology Stack

Runtime / Package Management

- Node.js, NPM

Language

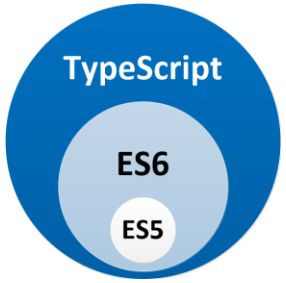
- TypeScript (ES 6, Dart)

Templating / Dependencies

- Angular CLI

Bundling

- Webpack



Common Editors

Any Editor that has integrated Support for Node.js

Editor choice is often result of available Plugins

- Visual Studio Code
  - Make sure you have Angular Language Service Extension installed
- IntelliJ IDEA & WebStorm from JetBrains
- Stackblitz
  - Online Editor used for prototyping and Online Questions
- Any other editor of choice that supports Node based Development

What is Angular CLI

Command Line Interface used to manage Angular projects

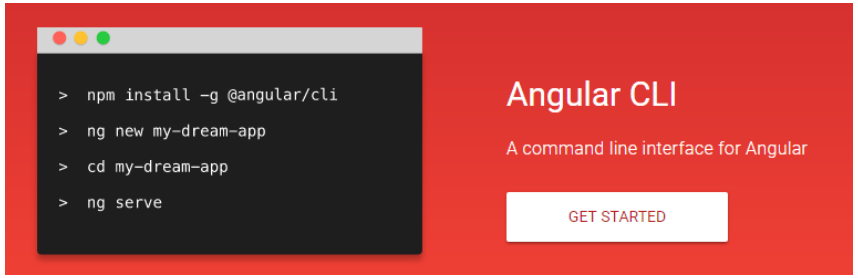
Installation

- npm install –g @angular/cli

Common Commands

- ng new
- ng generate
- ng serve

Documentations @ https://cli.angular.io/



Service

A service is responsible for

- data operations or
- provides reusability for utility methods

When working with data it typically utilizes the built in HttpClient defined in @angular/common/http

Can be Stateless or Statefull

```
@Injectable()
export class DemoService {
  constructor(private httpClient: HttpClient) {}

  getItems(): Observable<DemoItem[]> {
    return this.httpClient.get<DemoItem[]>("/assets/demos.json");
  }
}
```

```
@Component({
  selector: 'app-demo-container',
  templateUrl: './demo-container.component.html',
  styleUrls: ['./demo-container.component.scss']
})
export class DemoContainerComponent implements OnInit {
  constructor(
    private router: Router,
    private demoService: DemoService,
    public ms: MenuService
  ) {}
```

Injection & Instances

Services can be injected using "Providers" at:

- Module
- Component, or
- @Injectable Decorator (default) – ng6+
  - providedIn: "root" | "platform" | lazy loaded module

By default Services are Singletons per module

If injected to module and Component

- two instances are created

```
@Injectable({
  providedIn: "root"
})
export class O365Service {
  constructor(
    private adalSvc: MsAdalAngular6Service,
    private httpClient: HttpClient
  ) {
    this.user = this.adalSvc.userInfo;
  }
```

```
@NgModule({
  declarations: [
    DemoContainerComponent,
  ],
  imports: [
    CommonModule,
    FormsModule,
    RouterModule.forChild(demoRoutes),
    MaterialModule,
    HttpClientModule
  ],
  providers: [DemoService, VouchersService, PersonService]
})
```

Data Binding

Angular provides four databinding patterns

- String Interpolation – Expression Binding
  - <h1>{{ title }}</h1>
- Property binding – Attribute binding
  -
- Event binding
  - <button (click)="onClick(param)">
- Two-way binding – mix of Property Binding and Event Binding
  - <input type="text" [(ngModel)]="firstName">
- Form Controls provided by Reactive Forms - covered later

String Interpolation - Expressions

Just like Expressions it uses double curly brackets {{ variable }}

- can also use simple programming constructs:
  - ie x==true ? ...then ... : else

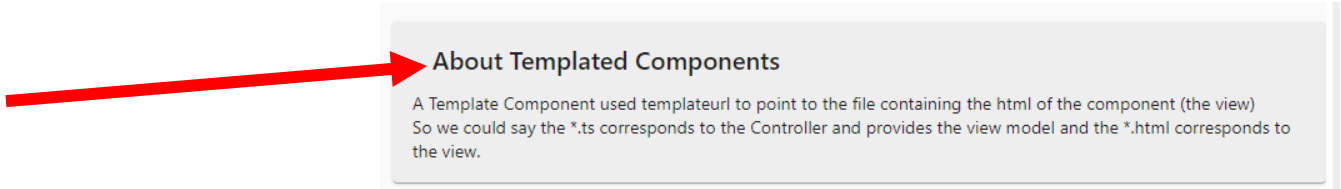
Value must be defined in Component as public property

- private props cannot be accessed in the template

```
export class TemplateComponent implements OnInit {
  title = 'About Templated Components';

  constructor() {}
```

```
<mat-card-header>
  <mat-card-title>{{ title }}</mat-card-title>
</mat-card-header>
```



Property Binding

Also called Attribute Binding

- Just warp the HTML attribute in square brackets [prop]

Properties of the component can be bound to any HTML attribute (without {{ curly braces }})

Replace most of Angular 1.x Directives like ng-hide -> [hidden]=„prop"

```
<div class="wrapper">
  <span [style.visibility]="hide ? 'hidden' : 'visible'">Msg 1</span>
  <span [style.visibility]="!hide ? 'hidden' : 'visible'">Msg 2</span>
</div>
```

Common DOM Events

Mouse events:

- onclick, onmousedown, onmouseup
- onmouseover, onmouseout, onmousemove

Key events:

- onkeypress, onkeydown, onkeyup
- Only for input fields

Interface events:

- onblur, onfocus
- onscroll

Form events

- onchange – for input fields
- onsubmit
  - Allows you to cancel a form submission
  - Useful for form validation

Event Binding

Event binding allows you to listen for certain events such as keystrokes, mouse movements, clicks, and touches

Can pass parameters e.g.:

- method(param)
- $event is a special param that is a reference to the event metadata

```
toggleDisplay() {
  this.hide = !this.hide;
}
```

```
<button mat-raised-button (click)="toggleDisplay()">Toggle</button>

<div class="wrapper">
  <span [style.visibility]="hide ? 'hidden' : 'visible'">Msg 1</span>
  <span [style.visibility]="!hide ? 'hidden' : 'visible'">Msg 2</span>
</div>
```

Two Way Binding

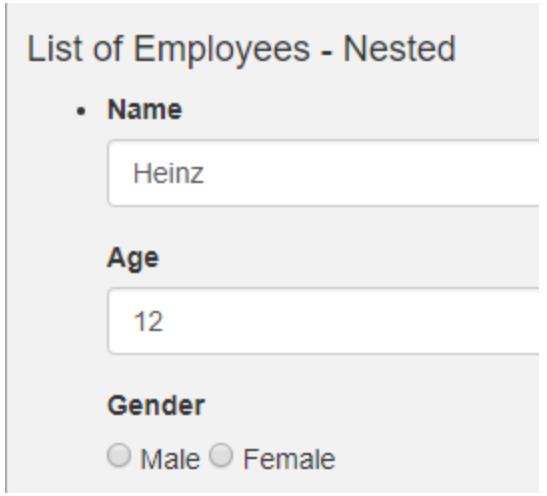
Achieved using [(ngModel)]='property'

- ( ) ... Event Binding
- [ ] ... Property Binding

Can be used directly, but in most cases in combination with forms

- Forms require FormsModule from '@angular/forms‘

Reactve Forms offer an alternative Binding Approach using a Reactive FormControl




```
export class BindingComponent implements OnInit {
  constructor(private ps: PersonService) {}

  persons: Person[];
  selectedPerson: Person = { id: 0, name: '', age: 0, gender: '' };
```

```
<mat-form-field>
  <label for="lblTwoWay">Two Way (Model) Binding</label> <br />
  <input matInput type="text" [(ngModel)]="selectedPerson.name"/>
</mat-form-field>
```