**What is @NgModule?**

This defines how different parts of the application fit together. Using Modules to encapsulate different features and functionalities can help with performance of your lazy load the modules. Each module will get compiled into it's own JavaScript file and the browser will only retrieve that js file Just In Time. Modules have declarations, imports, exports, providers and a bootstrap.

**What are component decorators?**

A component is an encapsulation of typescript, html, css. Angular combines these things into a specific type of directive. The Component directive allows the user to specify the metadata for 'selector', 'template' 'styles', 'providers' .

**What are Directives?**

They allow us to attach behavior to a DOM element.

*Structural directives?*

These change the structure of the DOM. \*ngIf, \*ngFor, \*ngSwitch

*Attribute directives?*

These allow you to change the appearance of a dom element.

*Component directives?*

They have their own template.

**\*ngIf:** Removes or recreates a portion of the DOM tree

**\*ngFor:** Turns the html element into a template and instantiates a view for each element in a list.

\*ngFor=”let p of persons”.

**\*ngSwitch:** Conditionally swaps the contents of the embedder templates basted on a condition

<div ng-switch="myVar">

<div ng-switch-when="dogs"> Dogs</div>

</div>

**@Pipe:** Enables the transform data with the HTML. Pipes can be chained. Custom one’s must implement PipeTransform. Upper/Lower, date, Currency, Percent {{ 2 | customPipe: 10}}

**@Injectable:** Marks a class as available for Injections. Can be provided in modules or components. Usually referred to as a ‘service’. Can be used to share data across components.

**@Input():** Allows a parent component to communicate information to a child component. Not this info flow is one way. Changes in child are not reflected in parent. You can use a {setter} to intercept the input and act upon it.

**@Output():** A child component would use this to emit information and make that info available to a parent component.

**@HostBinding –** Allows you to set properties on the element or component that hosts the directive.

**@HostListner –** Allows you to listen for events on the host element or component.

**ElementRef –** Allows direct access to the native DOM element.

**[ngClass]:** Binds the presence of css classes on the element. {‘active’: isActive}

**[ngStyle]:**  Allows you to assign styles to an html element using css.

**Reactive Forms:** Model driven approach to handling form inputs. Built around observable streams where inputs and values are provided as streams accessed synchronously

**Template Forms:** No longer recommended. Uses [(ngModel)] binding for value controls.

**Routing:** Requires a router-outlet. If hosting on IIS you need to configure redirect to index.html. Rout parameter are formatted like heros/:id and would be used like /heros/42. Order of routes matter. We can use route guards for authentication.

**HttpInterceptor**: Gives the ability to intercept https calls and handle errors or add headers.

**ngOnChanges():** Used to observe all changes to properties on a component.

**Ng-template:**

**HttpUrlEncodingCodec:** Encodes the URL strings. I overwrote this for our dotnet core REST API once.

**HttpXsrfTokenExtractor:** Used to extract an xsrf token so it can be combined into the next request.

**Template Ref Variables**: This is the hashtag in the html that lets you reference the element directly in html as a template or you can use the @ViewChild(‘name’) syntax in the .ts file to access that element.

**multi: true means** that one provider token provides an array of elements. For example all directives for router support routerLink, router-outlet are provided by ROUTER\_DIRECTIVES.

If a new provider is registered with the token ROUTER\_DIRECTIVES, then it overrides the previously registered directives. If multi: true (on the first registered and the new provider) is set, the new directives are added to the previously registered directives instead of overriding.

**Selectors:**

**•** element-name: Select by element name.

• .class: Select by class name.

• [attribute]: Select by attribute name.

• [attribute=value]: Select by attribute name and value.

• :not(sub\_selector): Select only if the element does not match the sub\_selector.

• selector1, selector2: Select if either selector1 or selector2 matches.

**What is Template reference variables?**

A template reference variable can be used in the html of a component and allows the component html to reference the element the variable is defined on. It is defined via a # then the name. It can also be used in the Typescript via a ViewChild('name') which gives you access to that element from within the typescript.

**What are all the uses of a service?**

Services should encapsulate business logic. They are used to separate the concerns of the UI from the Business. This separation makes unit testing easier. They can be used to store data.

**What is Pure and Impure Pipes?**

Pure pipes don't store state. Impure pipes store state. AsyncPipe is an example of an impure pipe.

**What is ngZone?**

This is the execution context that Angular uses in order to manage asynchronous activities. If you do work outside the zones scope you need to rescope your actions into the zone if you want Angular to do it's change detection.

**How is the core module different from other modules?**

The core module can only be imported once.

**What is important about the Async |.**

It will close the Observable itself.

**// Example - Output() Event Emitter**

@Component({

template: `<button (click)="add()">`,

selector: 'my-child-button'

})

export class MyChildButtonComponent {

@Input() amount = 0;

@Output() valueChange = new EventEmitter();

public add() {

this.amount++;

this.valueChange.emit(this.amount);

}

**// Example - delegate and expression**

public errorHandler = (error?: HttpResponseError) => {

console.error(error);

this.router.navigate(['/error']);

}

**// Example - ViewChild**

<app-siteminder-login #siteminderLogin [email]="email" [pword]="p"></app-siteminder-login>

@ViewChild('siteminderLogin') siteminderLogin;

**// Example - HTTPClient subscription**

add(account: AccountModel): Observable<AccountModel> {

const headers = new HttpHeaders({ 'Content-Type': 'application/json' });

const body = JSON.stringify(statementRecord);

const url = `${this.url}/Accounts`;

return this.http.post<AccountModel>(this.url, body, { headers: headers });

}

private addAccount() {

this.accountService

.add(this.accountModel)

.pipe(takeUntil(this.unsubscribe))

.subscribe(this.responseHandler, this.errorHandler, this.completeHandler);

}

**// Example - Validator**

export const zipMaskedValidator = (control: AbstractControl): { [key: string]: boolean } => {

if (control.value === undefined || control.value === null) {

return undefined;

}

const numbersOnlyValue = control.value.replace(/[^0-9\.]/g, '');

if (control.value && numbersOnlyValue.length !== 5) {

return { invalidLength: true };

}

};

@Directive({

selector: '[sha-zip-masked][ngModel]',

providers: [{ provide: NG\_VALIDATORS, useExisting: ZipMaskedValidatorDirective, multi: true }]

})

export class ZipMaskedValidatorDirective implements Validator {

constructor() { }

validate(control: AbstractControl): ValidationErrors | null {

return CustomValidators.zipMasked(control);

}

}

export function passwordContainsEmailValidator(otherControl: AbstractControl): ValidatorFn {

return (control: AbstractControl): ValidationErrors => {

// check control vs other control here

}

**// Example – for each**

var someArray = [9, 2, 5];

someArray.forEach((item, index) => {

console.log(item); // 9, 2, 5

console.log(index); // 0, 1, 2

});

**// Example – HostListner and HostBinding**

@HostBinding('style.color') color: string;

@HostBinding('style.border-color') borderColor: string;

@HostListener('keydown') newColor() {

const colorPick = Math.floor(Math.random() \* this.possibleColors.length);

this.color = this.borderColor = this.possibleColors[colorPick];

}