



Day 33



Change Detection

- Angular updates components when certain event occurs
 - DOM event fires eg, button clicked, input receiving focus, etc
 - `setTimeout()`, `setInterval()` completes
 - HTTP request completes
 - Variable the is bound to `@Input()` changes

- All attribute bindings are updated, the view is redrawn

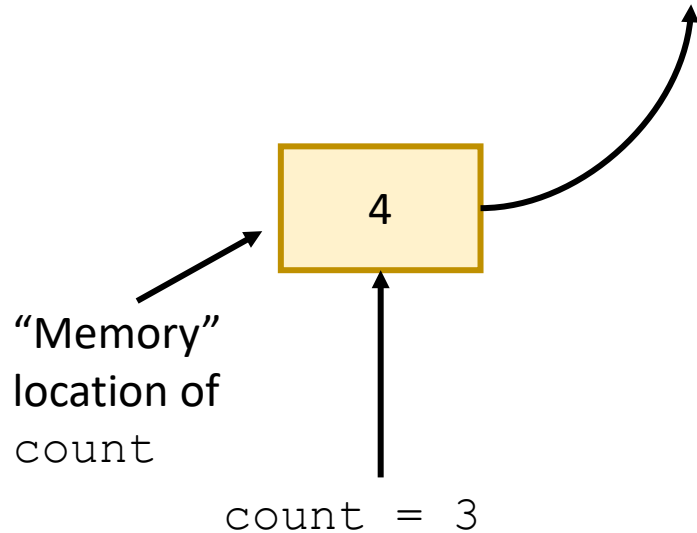
```
<app-list [data]="count"></app-list>
```

- `count` is reassigned to data attribute when change detection occurs
 - view is updated and component gets the latest `count` value
- Use the `OnChanges` to listen to updates to `@Input()`
 - Use case is for performing actions on dependent values eg. calculating the total cost when the shopping cart changes
 - `ngOnChanges` is only triggered if the variable change eg. assigned a new value



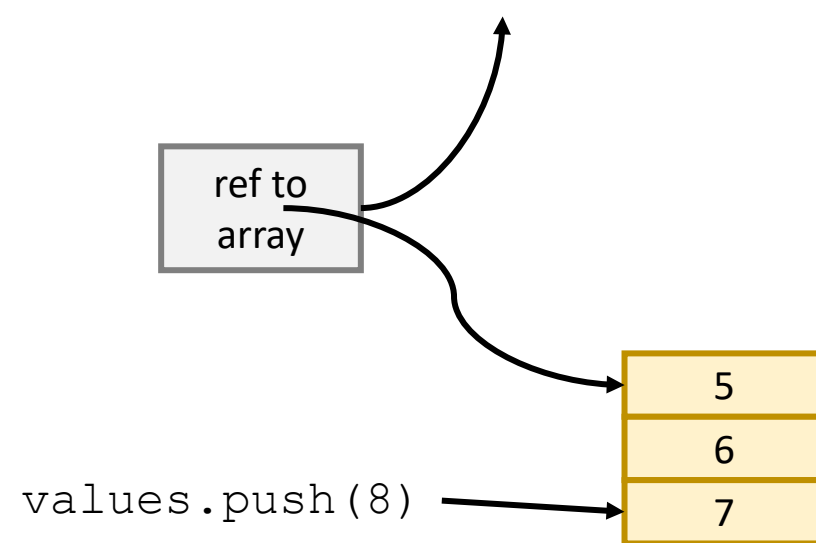
Attribute Binding

```
<app-list [data]="count"></app-list>
```



When `count` is reassigned a new value, change detection occurs: view is updated and the value is pushed to the component

```
<app-list [data]="values"></app-list>
```



When a new element is push to `values`, the array changes but the variable to the array did not change View is update but the new element is not pushed to the component



Typescript Types and Attribute Binding

- Value types
 - number
 - string
 - boolean
 - null, undefined
- Reference types
 - array
 - object
 - function
- Need to reassign a new reference for change detection to occur

`app.component.html`

```
<app-counter [list]="values"></app-counter>
```

`app.component.ts`

```
@Component({})  
export class AppComponent {  
  values: number[] = []  
  
  addToList(newValue: number) {  
    this.values = [ ...this.values, newValue ]  
  }  
}
```

Recreating the
reference type with
the new value



```
this.values.push(newValue)
```

Array with new
element will not be
pushed to component



Example - ngOnChanges Lifecycle Hook

```
@Component({...})
export class AppCounter implements OnInit, OnChanges {
  @Input()
  data: number = 0
  values: number = []
  total: number = 0
```

```
  ngOnInit() {
    this.total += this.data
    this.values.push(this.data)
  }
```

```
  ngOnChanges(changes: SimpleChanges) {
    this.total += this.data
    this.values.push(this.data)
  }
```

app.component.html

```
<app-counter [data]="value"></app-counter>
```

app-counter.component.html

```
<ul>
  <li *ngFor="let v of values">{{ v }}</li>
  <li> Total: {{ total }}</li>
</ul>
```

Update to value causes values and total to be updated

Update the dependents
when binding changes

Update the dependents
when component is created



Content Projection

- Content projection allows a parent component to project its content into a child component
- Use cases
 - Spinners
 - Additional form controls
- Type of projection
 - Single slot projection
 - Multiple slot project
 - Conditional projection
 - Will not be covering multiple and conditional projection

```
<app-registration>  
  <button type="button" (click)="process()">  
    Register  
  </button>  
</app-registration>
```

The diagram shows a registration form with the following elements:

- Name:
- Mobile:
- Email:
- ☒ Receive newsletter
-

Two arrows point from the text 'This is from the parent component' to the 'Register' button and the 'Receive newsletter' checkbox, indicating that these elements are projected from the parent component into the child component.

This is from
the parent
component



ng-content

app-component.html

```
<app-registration>
  <button type="button" (click)="process()">
    Register
  </button>
</app-registration>
```

app-registration.html

```
<form ...>
  ...
  <ng-content></ng-content>
</form>
```

Content goes here



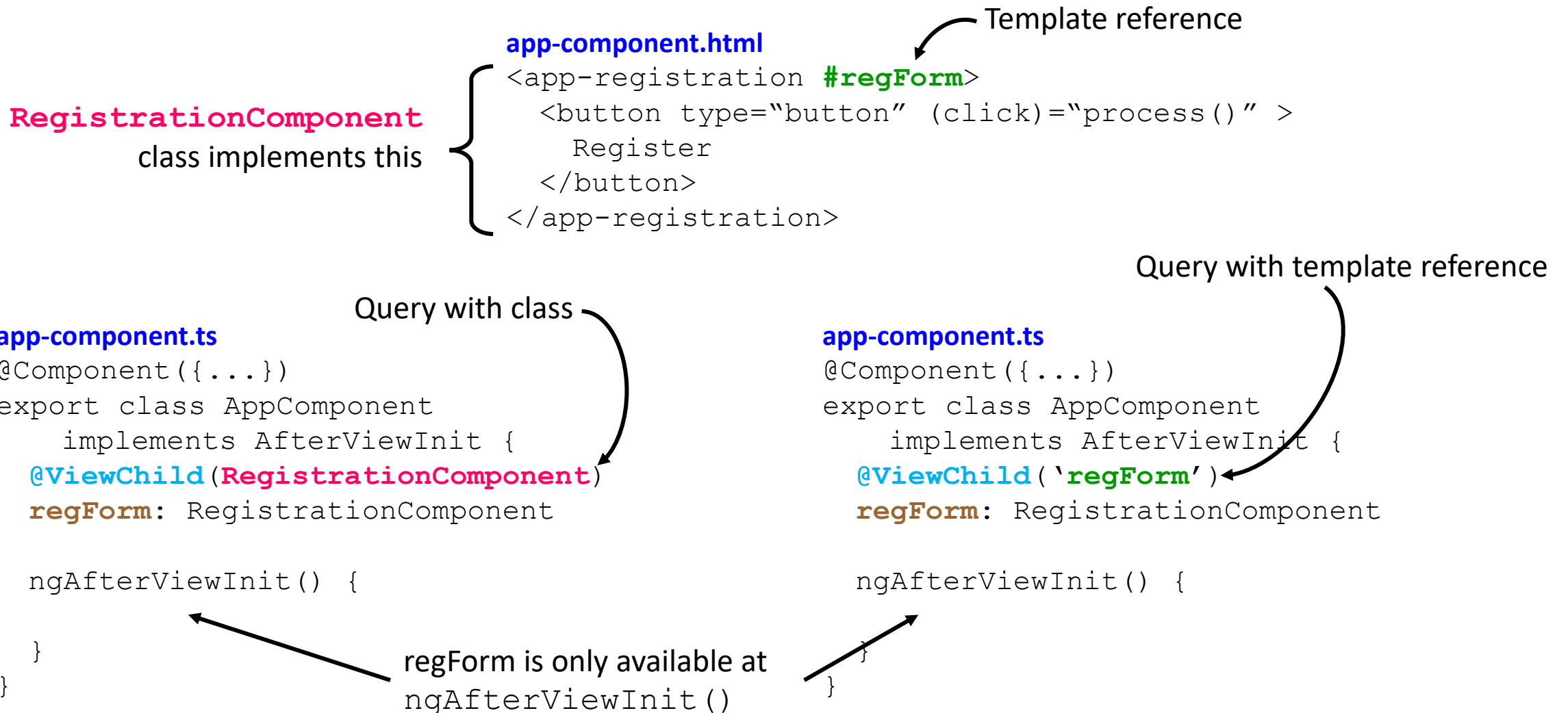


@ViewChild

- A way to allow a parent to access its child
 - Called a view query
 - Can only query immediate children
- Parent get a reference of the child component's instance
 - Manipulate the child component programmatically other than using attribute and event binding
 - Returns null if child is not found within the parent
- Component instance is only available after `AfterViewInit` lifecycle
- Use case: may be used with content projection to access values from a form's controls




Class Selector and Element Selector





Example - Content Projection and ViewChild

```
@Component({...})  
export class RegistrationComponent implements OnInit {
```

```
  @Input()  Attribute to add a  
  title: string title to the form
```

```
  get values(): Registration {  
    return this.form.value as Registration  
  }
```

```
  form!: FormGroup  
  constructor(private fb: FormBuilder) { }
```

```
  ngOnInit() {  
    this.form = this.fb.group({...})  
  }  
}
```

A property to get
the form's values



```
<h2>{{ title }}</h2>  
<form [formGroup]="form">  
  ...  
  <ng-content></ng-content>  
</form>
```

Content is projected to here





Example - Content Projection and ViewChild

```
@Component({...})  
export class AppComponent implements AfterViewInit {
```

```
  @ViewChild(RegistrationComponent)  
  regForm: RegistrationComponent
```

```
  ngAfterViewInit() {  
    this.regForm.title = "New Product Registration"  
  }
```

```
  processRegistration() {  
    const newReg: Registration = this.regForm.values  
    ...  
  }
```

Instead of using attribute binding,
programmatically set the component
attribute. Component is only accessible
on AfterViewInit

When the button (projected) is
pressed, get the values from the
component

```
<app-registration>  
  <button type="button" (click)="processRegistration()">  
</app-registration>
```



Mobile Application



Native

- Access native API
- Cannot run on multiple platform
- Run offline
- Distribute via app store



Web

- Cannot access native API
- Multiple platform support
- Cannot run offline
- Distribute via the web



Hybrid

- Can access native API
- Multiple platform support
- Run offline
- Distribute via the web



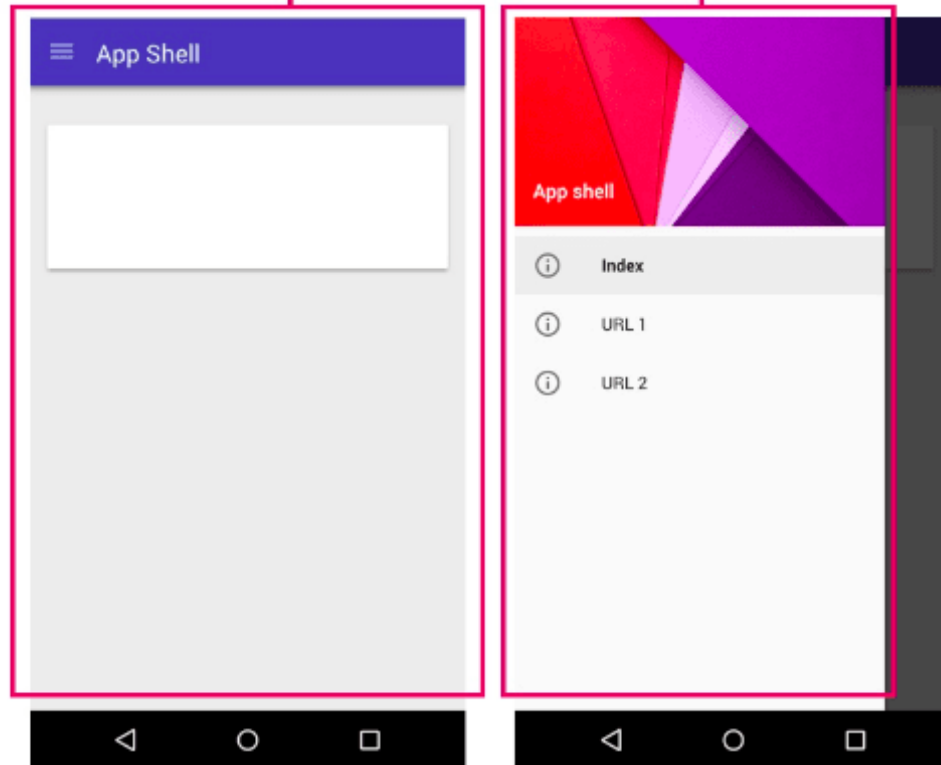
Progressive

- Limited by the browser
- Multiple platform support
- Run offline
- Distribute via the web



Progressive Web Application

application shell



Cached shell loads **instantly** on repeat visits.

content



Dynamic content then populates the view



Progressive Web Application Features

- Install as 'native' application on the phone, tablet's screen
- Has a service worker running in the background of the application
- Service worker
 - Caches the web application, HTML, CSS, JavaScript, images, etc so that the web application can run offline
 - Intercept outgoing or incoming HTTP request; can return previously cached data or modify request or result
- Receive web notifications
 - Web notifications allows backend to push data/notifications to registered web application even if the web application is not opened
 - Eg. WhatsApp web notification



Application Manifest

- A JSON file that provides information about an application
 - Eg. name, icon, splash screen, screen orientation, etc.
- The purpose of the manifest is to allow web application to be installed on a mobile device's home screen
 - If supported
- Is a core building block of PWA
 - Service worker
 - Push notification
 - Offline cache



Application Manifest

- Create a JSON file with one or more of the following properties
 - `name` - application's name
 - `short_name` - for use in space constrained
 - `start_url` - the URL that start/bootstraps the application
 - `scope` - what are the pages to be included with this application; typical value is /
 - `display` - how the application should be displayed. Valid values are
 - `standalone` - no browser control
 - `fullscreen` - similar to standalone
 - `background_color` - RGB
 - `theme_color` - top bar color RGB
 - `icons` - an array of icons
 - `orientation` - valid values include
 - `landscape`, `portrait`, `any`



Example - manifest.json

```
{
  "name": "My Awesome App",
  "short_name": "AwApp",
  "theme_color": "#2196f3",
  "background_color": "#2196f3",
  "display": "standalone",
  "scope": "/",
  "start_url": "/",
  "icons": [
    {
      "src": "images/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    }, ...
  ]
}
```



Application Manifest Generator

<https://manifest-gen.netlify.app/>

<https://www.simicart.com/manifest-generator.html/>

Unzip in your application's `src` or `assets` directory depending on how the manifest is generated

```
src
├── app
│   ├── app.component.css
│   ├── app.component.html
│   ├── app.component.spec.ts
│   ├── app.component.ts
│   ├── app.module.ts
│   └── material.module.ts
├── assets
├── environments
│   ├── environment.prod.ts
│   └── environment.ts
├── favicon.ico
├── images
│   └── icons
│       ├── icon-128x128.png
│       ├── icon-144x144.png
│       ├── icon-152x152.png
│       ├── icon-192x192.png
│       ├── icon-384x384.png
│       ├── icon-512x512.png
│       ├── icon-72x72.png
│       └── icon-96x96.png
├── index.html
├── main.ts
├── manifest.json
├── polyfills.ts
├── styles.css
├── test.ts
├── tsconfig.app.json
├── tsconfig.spec.json
└── typings.d.ts
```



Add Application Manifest to Application

- Add `<link>` to `src/index.html`

```
<link rel="manifest" href="/manifest.json">
```

- Include `manifest.json` and `images` directory as part of the build
 - Edit `angular.json`

```
"apps": [  
  {  
    "assets": [ "src/assets", "src/favico.ico",  
      "src/manifest.json", "src/images"  
    ],  
  }  
]
```



Application Identity

- `id` property give the PWA application an identity
- The mobile phone will treat the PWA as a new application even if the PWA has already been installed
- `id` should be the same as `start_url` attribute

```
"start_url": "/",  
"id": "/"
```

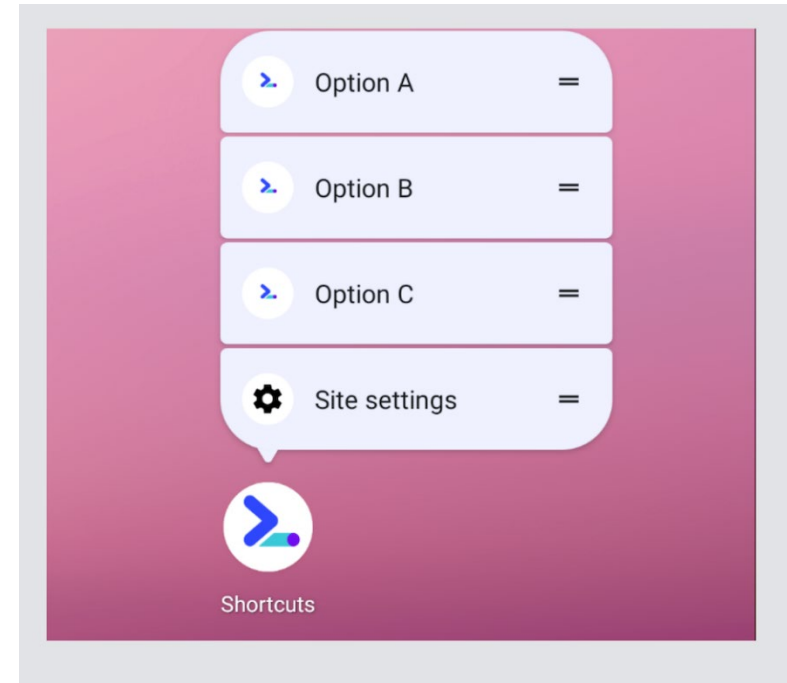


Shortcuts

- Short cuts is a static list of deep links into the PWA application

```
{
  "start_url": "http://server.com",
  "shortcuts": [
    {
      "name": "Option A",
      "short_name": "Option A",
      "description": "...",
      "url": "/option?selection=A",
      "icons": [
        {
          "src": "/icons/icon.png",
          "sizes": "192x192"
        }
      ]
    },
    ...
  ]
}
```

A single shortcut entry




The following URL will be fetched with the shortcut is selected


http://server.com/option?selection=A



Empty Service Worker


 **index.html**

```
<head>
  <script src="reg_sw.js"></script>
  ...
```

 **sw.js**

```
console.info('Idle service worker')
```

Register an empty service worker in the Angular's `index.html` file

 **reg_sw.js**

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js')
    .then(reg => {
      console.info('Service worker registered ', reg)
    })
    .catch(error => {
      console.error('Cannot register server worker ', error)
    })
}
```



Application Manifest Installed

Your email *

Please enter a valid email

Your phone *

Please enter a 8 digit phone number

Register Me

Application

- Manifest
- Service Workers
- Clear storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies

Cache

- Cache Storage
- Application Cache

Frames

- top

Name My Awesome App

Short name AwApp

Presentation

Start URL /


Theme color #2196f3


Background color #2196f3


Orientation

Display standalone

Icons

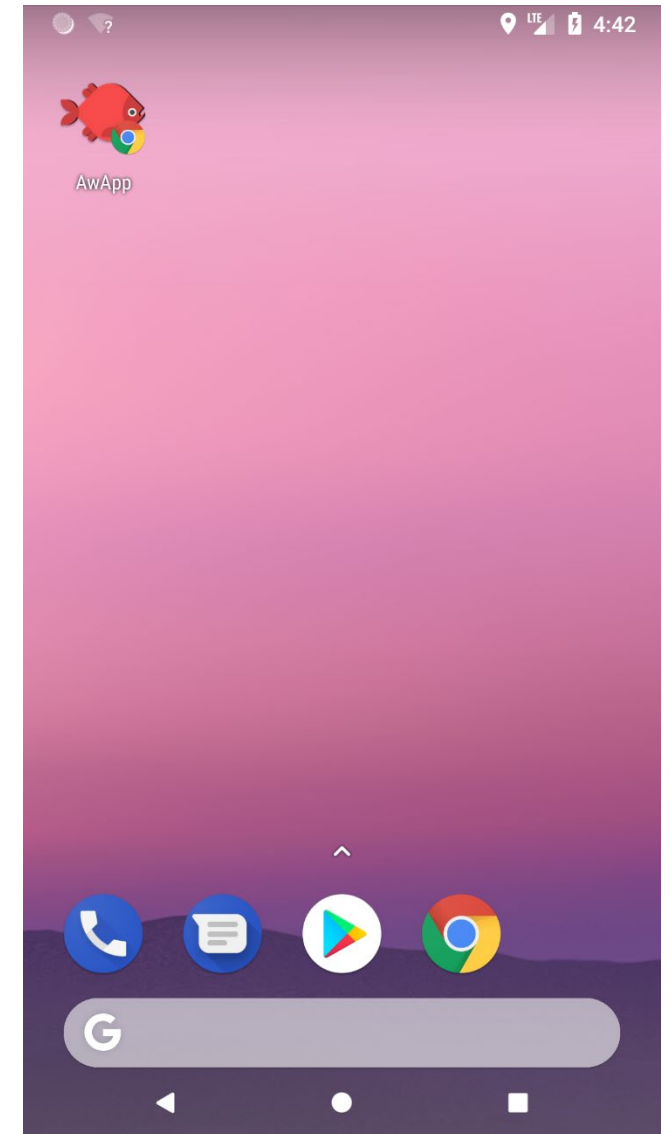
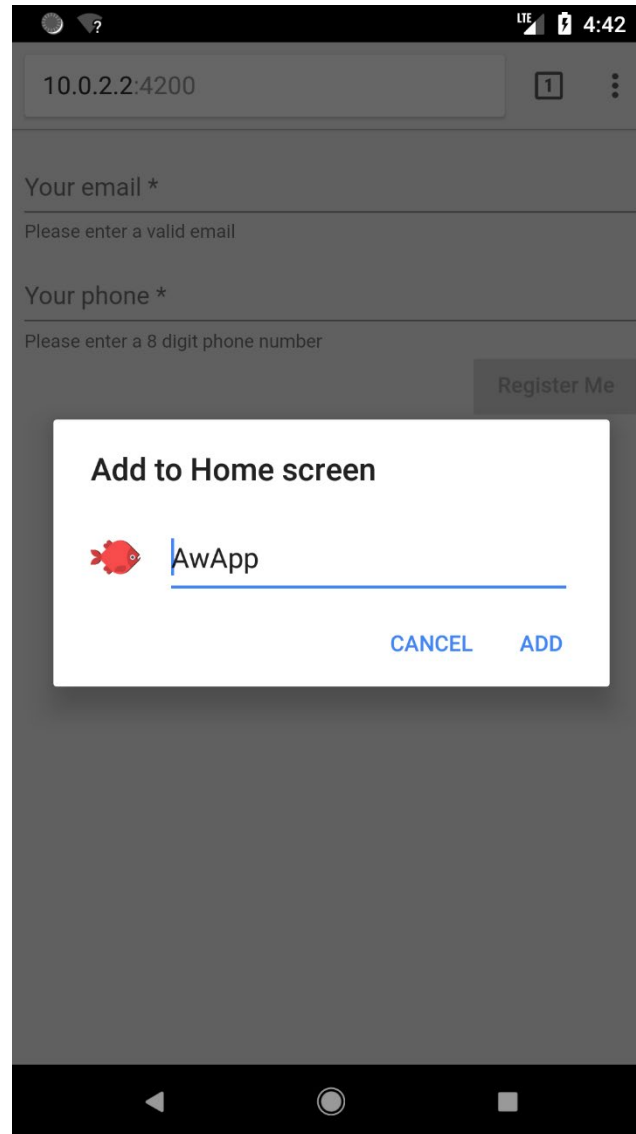
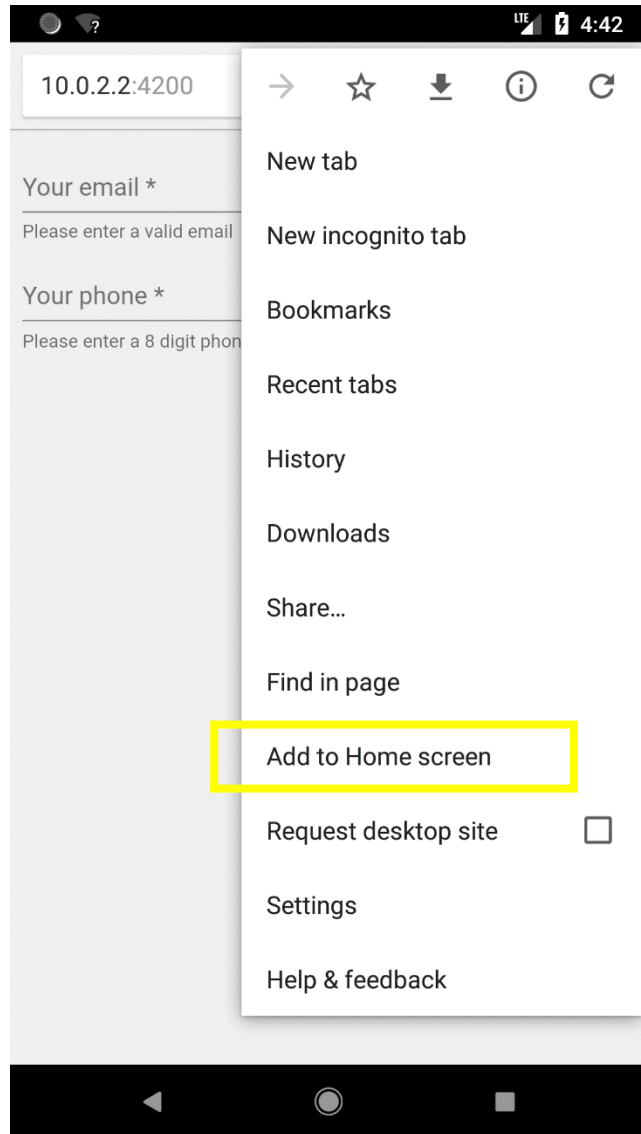
72x72
image/png 

96x96
image/png 

128x128
image/png 



On Mobile Phone





Inspect Android Chrome

- Enable developer mode on Android phone
- Attach phone to notebook/computer
- Allow Android phone to be debugged via USB
- Open Chrome browser on notebook/computer
 - `chrome://inspect/#devices`

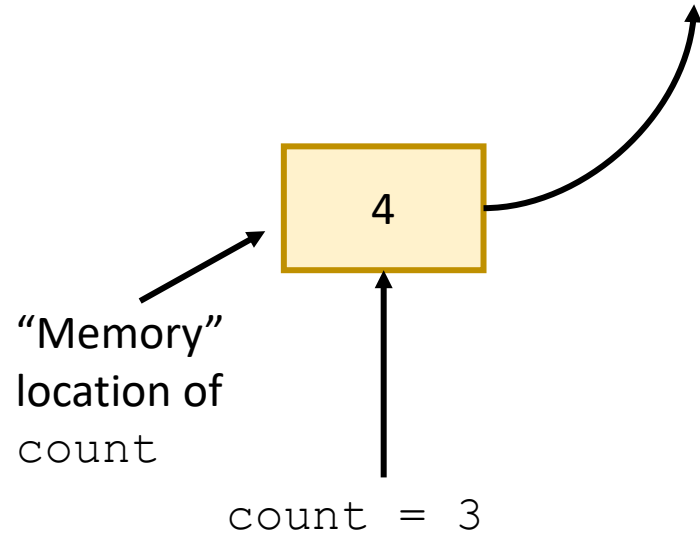


Unused



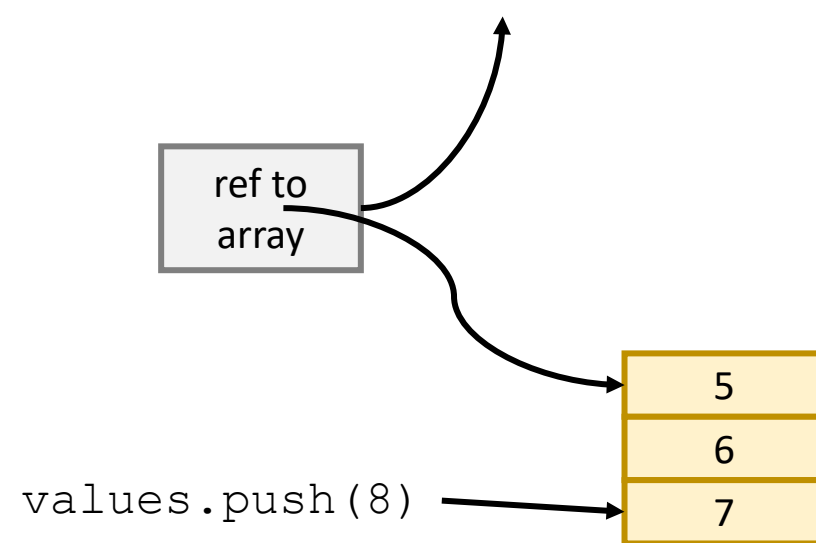
Attribute Binding

```
<app-list [data]="count"></app-list>
```



When `count` is reassigned a new value, change detection occurs: view is updated and the value is pushed to the component

```
<app-list [data]="values"></app-list>
```



When a new element is push to `values`, the array changes but the variable to the array did not change View is update but the new element is not pushed to the component



Typescript Types and Attribute Binding

- Value types
 - number
 - string
 - boolean
 - null, undefined
- Reference types
 - array
 - object
 - function
- Need to reassign a new reference for change detection to occur

`app.component.html`

```
<app-counter [list]="values"></app-counter>
```

`app.component.ts`

```
@Component({})  
export class AppComponent {  
  values: number[] = []  
  
  addToList(newValue: number) {  
    this.values = [ ...this.values, newValue ]  
  }  
}
```

Recreating the
reference type with
the new value



```
this.values.push(newValue)
```

Array with new
element will not be
pushed to component



Angular Material



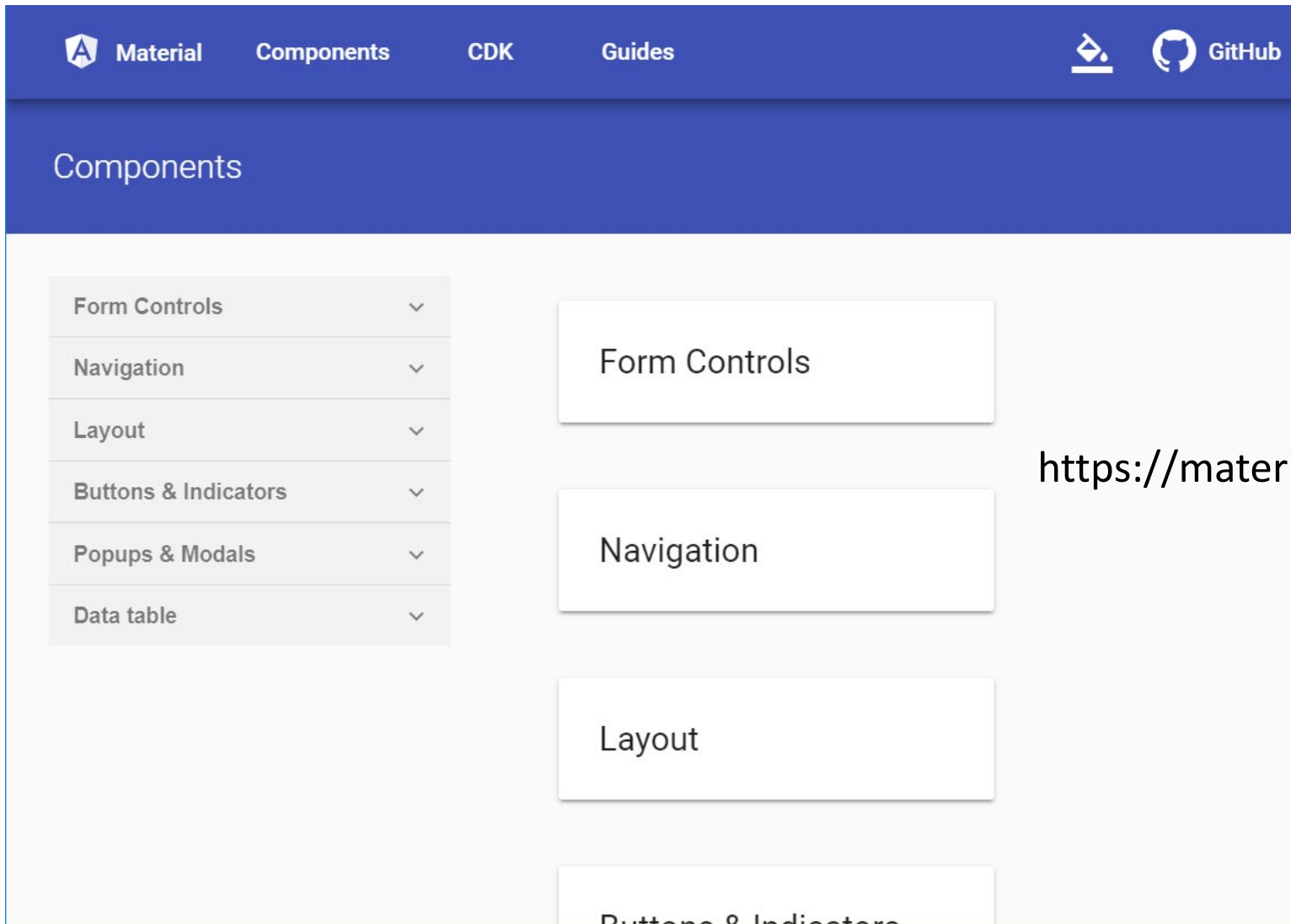
Angular Material

- Set of UI components for Google's Material Design Specification
- Consist of Component Development Kit and Material components
 - Dependent on animations and hammer.js
- Angular Materials Setup
 - See <https://material.angular.io/guide/getting-started> for details

```
ng add @angular/material
```



Angular Materials Documentation Page



<https://material.angular.io/components/categories>



Using Material Components

- Material component consist of one or more material markup
 - Eg `<button mat-button>`, `<mat-icon>`
- Almost every component is in a separate module
 - Need to import module to unlock component
- **Example:** `mat-button`, `mat-raised-button`, `mat-fab`
 - From `MatButtonModule`



Using Angular Material Components

```
import { MatButtonModule } from '@angular/material/button';
import { MatIconModule } from '@angular/material/icon';
const MATERIAL = [ MatButtonModule, MatIconModule ];
@NgModule({
  imports: MATERIAL,
  exports: MATERIAL
})
export class MaterialModule { }
```

**Keep Material module imports in
a separate module**

```
import { MaterialModule } from '../material.module';
@NgModule({
  imports: [ BrowserModule, MaterialModule ],
  ...
})
export class AppModule { }
```

**Import
MaterialModule
into the AppModule**



Example - Buttons

Press

```
<button type="button"
      mat-button>
  Press
</button>
```

Press

```
<button type="button"
      mat-raised-button>
  Press
</button>
```

Press

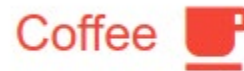
```
<button type="button"
      color="primary"
      mat-raised-button>
  Press
</button>
```



```
<button type="button"
      mat-icon-button>
  <mat-icon>favorite</mat-icon>
</button>
```



```
<button type="button"
      color="accent"
      mat-raised-button
      mat-icon-button>
  <mat-icon>favorite</mat-icon>
</button>
```



```
<button type="button"
      color="warn"
      mat-button>
  <mat-icon>
    free_breakfast
  </mat-icon>
</button>
```



Form Field

```
<form [formGroup]="rsvpForm"
      (ngSubmit)="process(form)">
```

```
<mat-form-field>
```

```
  <input type="email" FormControlName="email"
        placeholder="Please enter your email"
        matInput>
```

```
</mat-form-field>
```

```
</form>
```

mat-form-field are used to
wrap input **and** textarea

Please enter your email

Blur

Please enter your email

Focus



Radio Button

☐ Yes ☐ No

```
<mat-radio-group formControlName="attending">  
  <mat-radio-button value="yes">  
    Yes  
  </mat-radio-button>  
  <mat-radio-button value="no">  
    No  
  </mat-radio-button>  
</mat-radio-group>
```



Select

Guest



```
<mat-form-field>
  <mat-select placeholder="Guest" formControlName="guest" >
    <mat-option *ngFor="let g of [0, 1, 2, 3]"
      [value]="g">
      {{ g }}
    </mat-option>
  </mat-select>
</mat-form-field>
```



Checkbox

☐ Yes, I wish to receive newsletter

```
<mat-checkbox formControlName="newsletter">  
  Yes, I wish to receive newsletter  
</mat-checkbox>
```



Checkbox - Multiple Values

Diet:

☐ Fish ☐ Meat ☐ Vegetables

```
labels = [ 'Fish', 'Meat', 'Vegetables' ]  
dietArray: FormArray  
rsvpForm: FormGroup  
  
ngOnInit() {  
  this.dietArray = this.fb.array(  
    this.labels.map(()=>this.fb.control(''))  
  )  
  this.rsvpForm = this.fb.group({  
    ...  
    diet: this.dietArray  
  })  
}
```



Checkbox - Multiple Values

Diet:

☐ Fish ☐ Meat ☐ Vegetables

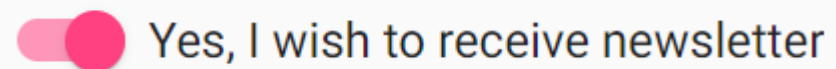
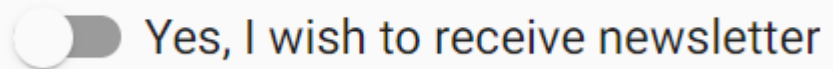
```
<mat-checkbox [formControl]="diet"
  *ngFor="let diet of dietArray.controls; let i = index">

  {{ labels[i] }}

</mat-checkbox>
```




Slide Toggle



```
<mat-slide-toggle formControlName="newsletter">  
  Yes, I wish to receive newsletter  
</mat-slide-toggle>
```



Date Picker

- Date picker requires a date implementation, supported implementations (<https://material.angular.io/components/datepicker/overview#choosing-a-date-implementation-and-date-format-settings>)

- Native
- date-fns - <https://github.com/date-fns/date-fns>
- Luxon - <https://moment.github.io/luxon/>

- Install an implementation

```
npm install --save @angular/material-luxon-adapter
```

- Add to `app.module.ts`

```
import { MatLuxonDateModule }  
    from '@angular/material-luxon-adapter';
```



Date Picker

```
<mat-form-field appearance="fill">
```

```
  <input matInput
    [matDatepicker]="datepicker"
    placeholder="Date of birth"
    FormControlName="dob">
```

```
  <mat-datepicker-toggle matSuffix
    [for]="datepicker">
</mat-datepicker-toggle>
```

```
  <mat-datepicker #datepicker>
</mat-datepicker>
```

```
</mat-form-field>
```

Date of birth



Date of birth



JAN 1970 ▾

< >

Su	Mo	Tu	We	Th	Fr	Sa
JAN				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31



Template Driven Form



Form

RSVP

Form

Controls

Button to submit

Name:

Email:

Phone:

Attending: ☐ YES ☐ NO

A diagram of an RSVP form. The form is enclosed in a dashed blue border. It contains labels for Name, Email, and Phone, each followed by a text input field with placeholder text. Below these is an "Attending" section with two radio buttons labeled "YES" and "NO". At the bottom is a "Send" button. Annotations include an arrow pointing to the top of the dashed box labeled "Form", an arrow pointing to the top of the "Name" input field labeled "Controls", an arrow pointing to the top of the "Email" input field labeled "Controls", an arrow pointing to the top of the "Phone" input field labeled "Controls", an arrow pointing to the "YES" radio button labeled "Controls", and an arrow pointing to the "Send" button labeled "Button to submit".



Form

`<form>` ← Form

Name: `<input type="text" name="name">`

...

Attending:

`<input type="radio" name="attending" value="yes"> YES`

`<input type="radio" name="attending" value="no"> NO`

`<button type="submit">`
Send
`</button>`

← Button to submit

`</form>`

Controls



Angular Forms

- Angular creates an `NgForm` component for every `<form>`
 - Fires an event call `ngSubmit` in response to the submit
 - Form component is called `ngForm`

Event fired by `NgForm` component
when button (submit) is pressed

`ngForm`

```
<form (ngSubmit)="processForm()">
```

```
  <button type="submit">  
    Send  
  </button>
```

```
</form>
```

A `NgForm` is instantiated
and mapped to a form



Forms - NgModel

- Annotate form fields `<input>` with `ngModel` directive
- Allow `NgForm` to manage them as controls
- Every field must have a unique name

```
<input type="text" name="email" ngModel>
```




Form

```
<form (ngSubmit)="processForm()">
```

```
  Name: <input type="text" name="name" ngModel>
```

```
  ...
```

```
  Attending:
```

```
  <input type="radio" name="attending" value="yes" ngModel> YES
```

```
  <input type="radio" name="attending" value="no" ngModel> NO
```

```
  <button type="submit">
```

```
    Send
```

```
  </button>
```

```
</form>
```



Template Reference

- A variable that references a HTML element

```
<h1 #h1Element>hello, world</h1>
```

```
<form #form>...</form>
```

- Assign the template reference on a `<form>` to `ngForm` object
 - Access to the form when it is submitted

```
<form #form="ngForm" (ngSubmit)="processForm(form)">  
  ...  
</form>
```



Processing Form

```
<form #form="ngForm" (ngSubmit)="processForm(form)">
```

```
Name: <input type="text" name="name" ngModel>
```

```
export class RSVPComponent {
```

```
  processForm(form: NgForm) {
```

```
    const name = form.value.name;
```

```
    ...
```

```
    form.reset();
```

```
  }
```

```
}
```

Pass the form into
the event handler

Clears the values in the form

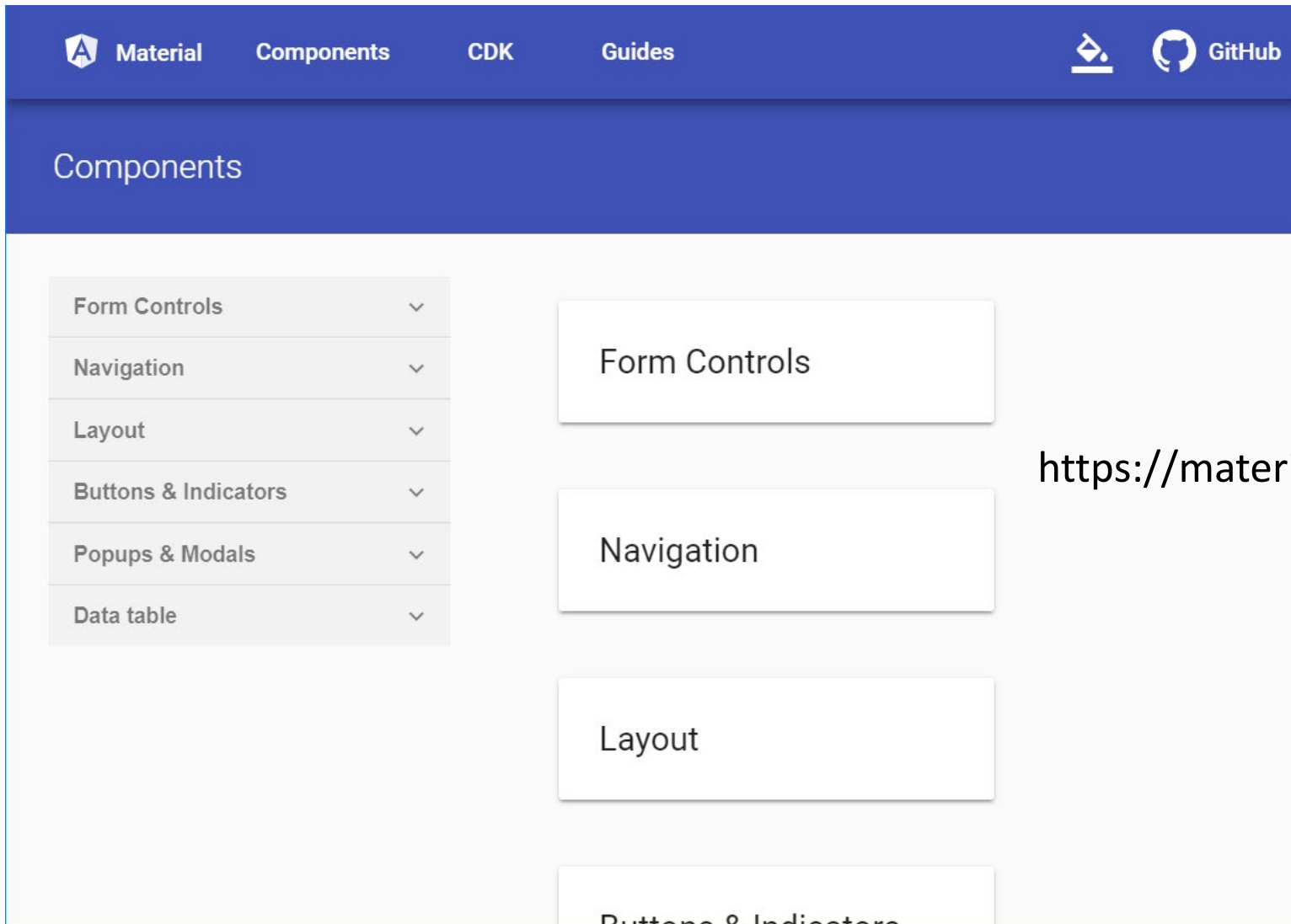


Angular Material

- Set of UI components for Google's Material Design Specification
- Consist of Component Development Kit and Material components
 - Dependent on animations and hammer.js
- Angular Materials Setup
 - See <https://material.angular.io/guide/getting-started> for details
- Steps
 - Install `@angular/cdk`, `@angular/material`, `@angular/animations`, `hammerjs`
 - Import `BrowserAnimationsModule` in `app.module.ts`
 - Select a theme from `@angular/material/prebuilt-themes`
 - Import `hammer.js` in `main.ts` for gesture support
 - Add Material icons



Angular Materials Documentation Page



<https://material.angular.io/components/categories>



Using Material Components

- Material component consist of one or more material markup
 - Eg `<button mat-button>`, `<mat-icon>`
- Almost every component is in a separate module
 - Need to import module to unlock component
- **Example:** `mat-button`, `mat-raised-button`, `mat-fab`
 - From `MatButtonModule`



Using Angular Material Components

```
import { MatButtonModule } from '@angular/material/button';
import { MatIconModule } from '@angular/material/icon';
const MATERIAL = [ MatButtonModule, MatIconModule ];
@NgModule({
  imports: MATERIAL,
  exports: MATERIAL
})
export class MaterialModule { }
```

**Keep Material module imports in
a separate module**

```
import { MaterialModule } from '../material.module';
@NgModule({
  imports: [ BrowserModule, MaterialModule ],
  ...
})
export class AppModule { }
```

**Import
MaterialModule
into the AppModule**



Example - Buttons

Press

```
<button type="button"
      mat-button>
  Press
</button>
```

Press

```
<button type="button"
      mat-raised-button>
  Press
</button>
```

Press

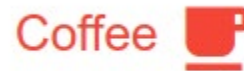
```
<button type="button"
      color="primary"
      mat-raised-button>
  Press
</button>
```



```
<button type="button"
      mat-icon-button>
  <mat-icon>favorite</mat-icon>
</button>
```



```
<button type="button"
      color="accent"
      mat-raised-button
      mat-icon-button>
  <mat-icon>favorite</mat-icon>
</button>
```



```
<button type="button"
      color="warn"
      mat-button>
  <mat-icon>
    free_breakfast
  </mat-icon>
</button>
```




Form Field

```
<form #form  
  (ngForm)="process(form)">
```

```
<mat-form-field>
```

```
  <input type="email" name="email"  
    placeholder="Please enter your email"  
    ngModel matInput>
```

```
</mat-form-field>
```

```
</form>
```

mat-form-field are used to
wrap input **and** textarea

Please enter your email

Blur

Please enter your email

Focus



Radio Button

☐ Yes ☐ No

```
<mat-radio-group name="attending" ngModel>  
  <mat-radio-button value="yes">  
    Yes  
  </mat-radio-button>  
  <mat-radio-button value="no">  
    No  
  </mat-radio-button>  
</mat-radio-group>
```



Select

Guest



```
<mat-form-field>
  <mat-select placeholder="Guest" name="guest" ngModel>
    <mat-option *ngFor="let g of [0, 1, 2, 3]"
      [value]="g">
      {{ g }}
    </mat-option>
  </mat-select>
</mat-form-field>
```



Checkbox

☐ Yes, I wish to receive newsletter

```
<mat-checkbox name="newsletter" ngModel>  
  Yes, I wish to receive newsletter  
</mat-checkbox>
```



Checkbox - Multiple Values

Diet:

☐ Fish ☐ Meat ☐ Vegetables

MatCheckboxChange
is the event object

```
<mat-checkbox ngModel (change)="diet[0] = $event.checked">
  Fish
</mat-checkbox>
<mat-checkbox ngModel (change)="diet[1] = $event.checked">
  Meat
</mat-checkbox>
<mat-checkbox ngModel (ngModel)="diet[2] = $event.checked">
  Vegetables
</mat-checkbox>
```

`diet` = [false, false, false]

Array elements will be set to true
when checkbox is selected



Date Picker

- Datepicker component uses moment for date and time
 - See <https://momentjs.com>
- Will have to install moment adapter

```
npm install --save @angular/material-moment-adapter
```

- Add to app.module.ts

```
import { MatMomentDateModule }  
    from '@angular/material-moment-adapter';
```



Date Picker

```
<mat-form-field>
```

```
  <input matInput  
    [matDatepicker]="datepicker"  
    placeholder="Date of birth"  
    ngModel name="dob">
```

```
  <mat-datepicker-toggle matSuffix  
    [for]="datepicker">  
</mat-datepicker-toggle>
```

```
  <mat-datepicker #datepicker>  
</mat-datepicker>
```

```
</mat-form-field>
```

Date of birth



Date of birth



JAN 1970 ▾

< >

Su	Mo	Tu	We	Th	Fr	Sa
JAN				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31