# Day 32

# Sample Form

Mandatory

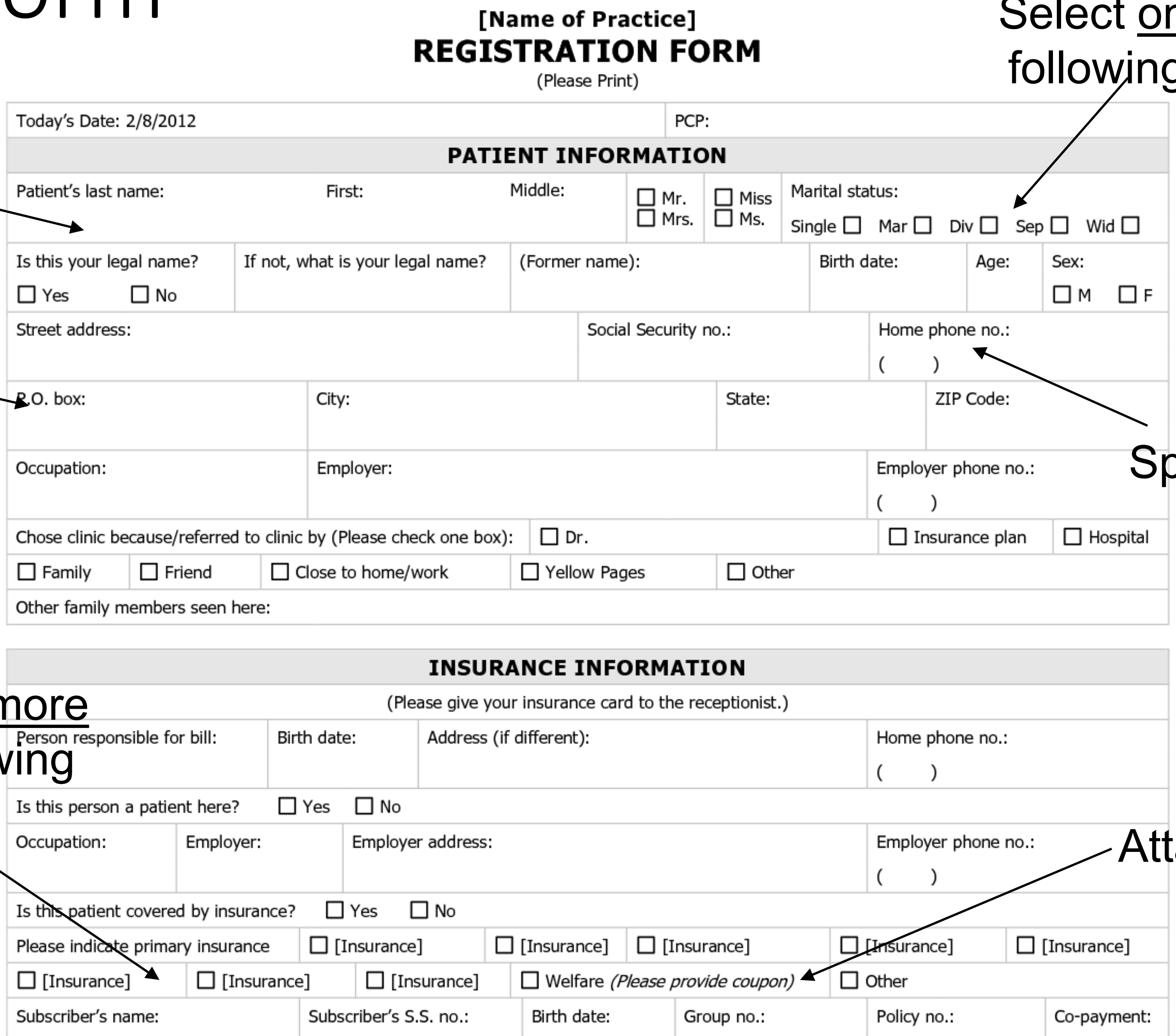
Fill in the blanks

Select one or more from the following

Select one of the following values

Specific value type

Attachments



**[Name of Practice]**

**REGISTRATION FORM**

(Please Print)

| | |
|---|---|
| Today's Date: 2/8/2012 | PCP: |

**PATIENT INFORMATION**

| | | | | | |
|---|---|---|---|---|---|
| Patient's last name: | First: | Middle: | ☐ Mr. ☐ Mrs. | ☐ Miss ☐ Ms. | Marital status: Single ☐ Mar ☐ Div ☐ Sep ☐ Wid ☐ |

| | | | | | |
|---|---|---|---|---|---|
| Is this your legal name? ☐ Yes ☐ No | If not, what is your legal name? | (Former name): | Birth date: | Age: | Sex: ☐ M ☐ F |

| | | |
|---|---|---|
| Street address: | Social Security no.: | Home phone no.: ( ) |

| | | | |
|---|---|---|---|
| P.O. box: | City: | State: | ZIP Code: |

| | | |
|---|---|---|
| Occupation: | Employer: | Employer phone no.: ( ) |

| | | | |
|---|---|---|---|
| Chose clinic because/referred to clinic by (Please check one box): | ☐ Dr. | ☐ Insurance plan | ☐ Hospital |

| | | | | |
|---|---|---|---|---|
| ☐ Family | ☐ Friend | ☐ Close to home/work | ☐ Yellow Pages | ☐ Other |

Other family members seen here:

**INSURANCE INFORMATION**

(Please give your insurance card to the receptionist.)

| | | | |
|---|---|---|---|
| Person responsible for bill: | Birth date: | Address (if different): | Home phone no.: ( ) |

Is this person a patient here? ☐ Yes ☐ No

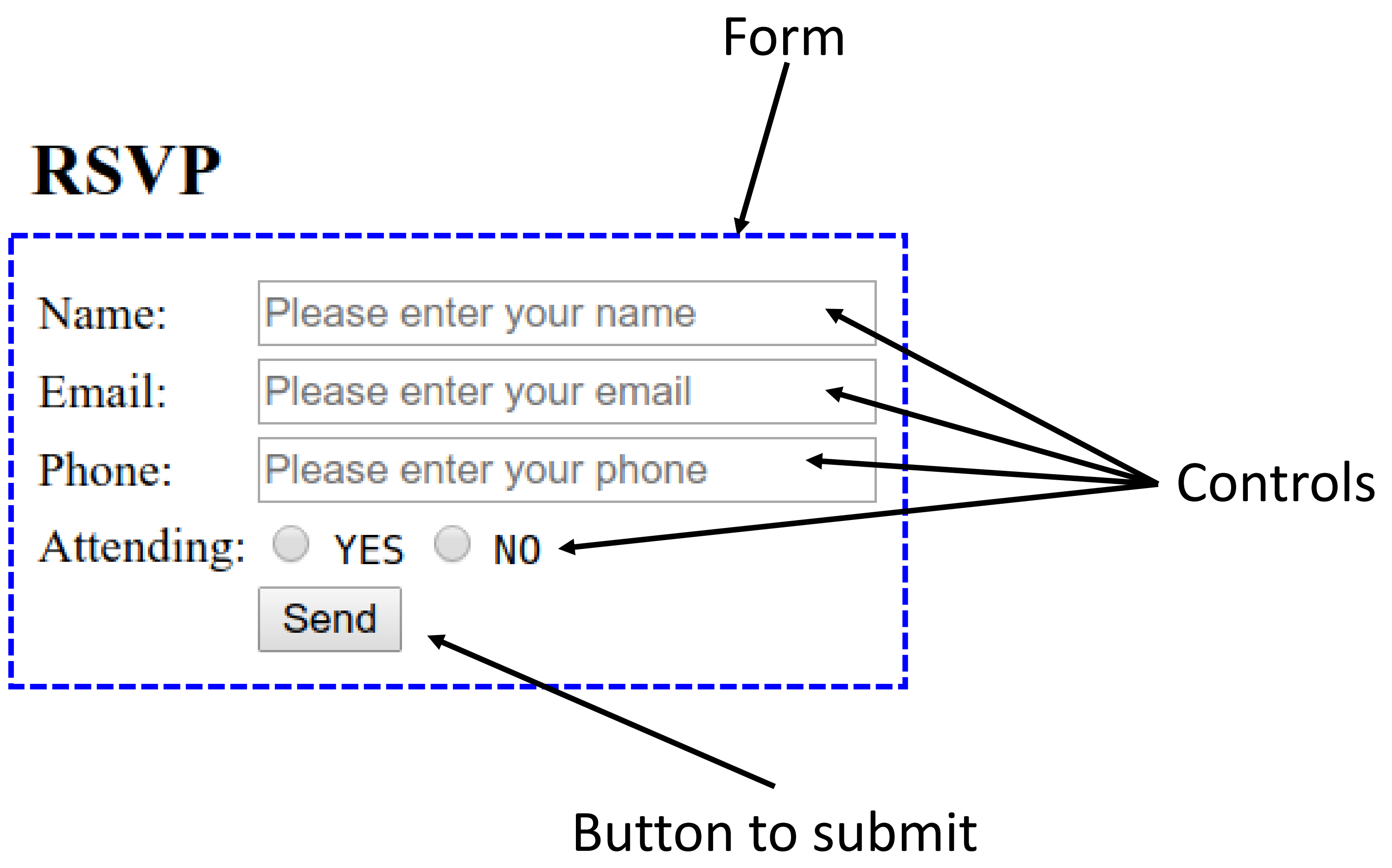
| | | | |
|---|---|---|---|
| Occupation: | Employer: | Employer address: | Employer phone no.: ( ) |

Is this patient covered by insurance? ☐ Yes ☐ No

| | | | | | |
|---|---|---|---|---|---|
| Please indicate primary insurance | ☐ [Insurance] | ☐ [Insurance] | ☐ [Insurance] | ☐ [Insurance] | ☐ [Insurance] |
| ☐ [Insurance] | ☐ [Insurance] | ☐ [Insurance] | ☐ Welfare *(Please provide coupon)* | ☐ Other | |

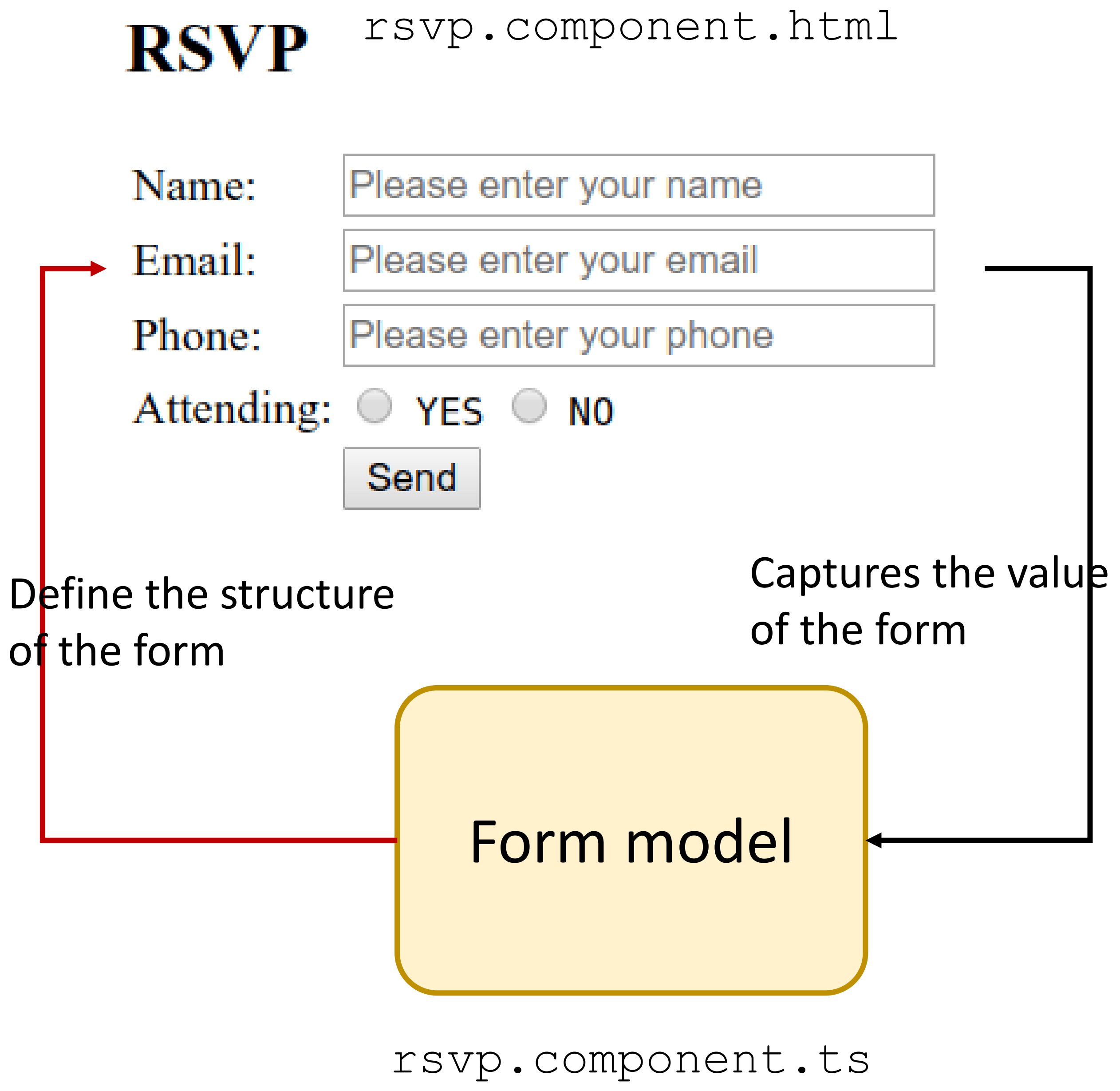
| | | | | | |
|---|---|---|---|---|---|
| Subscriber's name: | Subscriber's S.S. no.: | Birth date: | Group no.: | Policy no.: | Co-payment: |

# Terminology - Form



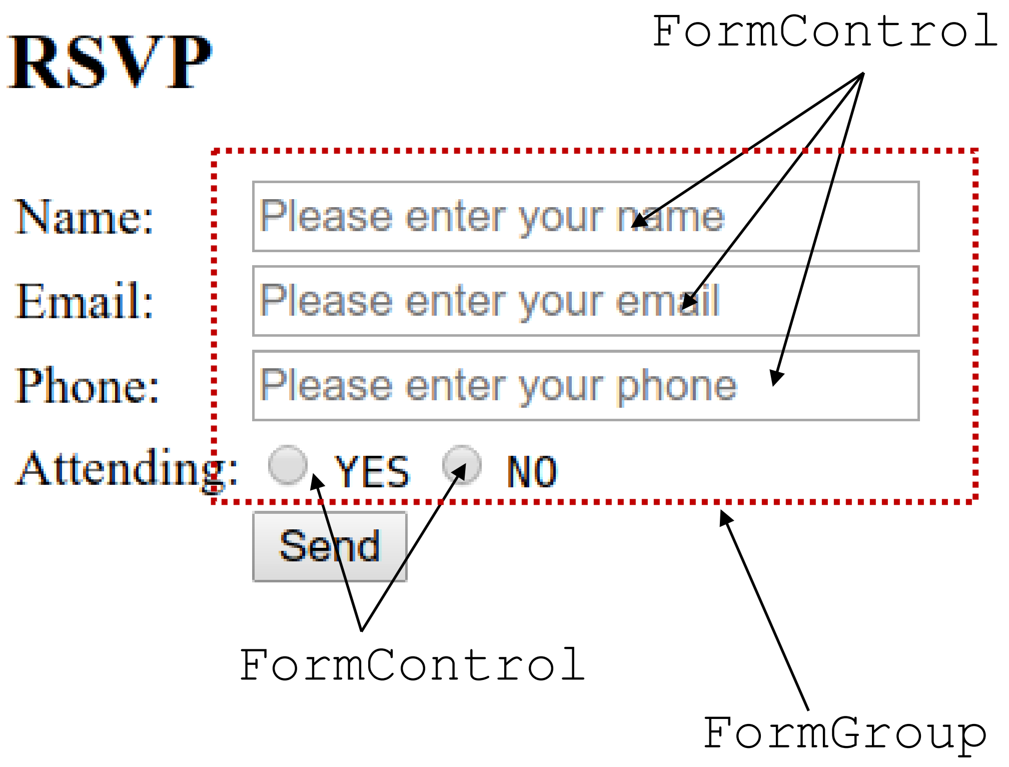
# Forms

- Angular has 2 types of forms
  - Template driven
  - Reactive
- Template driven
  - Form (HTML) defines the form
  - Good for static forms
- Reactive
  - Form's logic is in the component (TypeScript)
  - More complex but more flexible



# Terminology - Angular

- Control - **`FormControl`**
  - Represents a from control
  - Eg. input, textarea, checkboxes, etc
- Group - **`FormGroup`**
  - Holds one or more controls, array or group
  - Used for logical grouping eg customer info
- Array - **`FormArray`**
  - Grouping controls, groups and/or array



# Configuring Reactive Forms Module

- Need to be added the module

```
import { ReactiveFormsModule } from ‘@angular/forms’;

@NgModule({
  imports: [
    BrowserModule,
    ReactiveFormsModule
  ]
})
export class AppModule {
  ...
```

## Create the 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>

</form>
```

Controls

Button to submit

# Component Lifecycle

- Angular creates, renders and destroy the component
- Lifecycle hooks allow you to perform certain operations at these key moments
  - Eg. load data before the component is destroyed
- Implement one or more of these lifecycle interfaces
  - `OnInit` - called just after the component is created but before displaying
  - `OnDestroy` - called just before the component is destroyed
  - `OnChanges` - called when the `@Inputs` are updated because of attribute binding
  - `AfterViewInit` - called after the component's view has been created

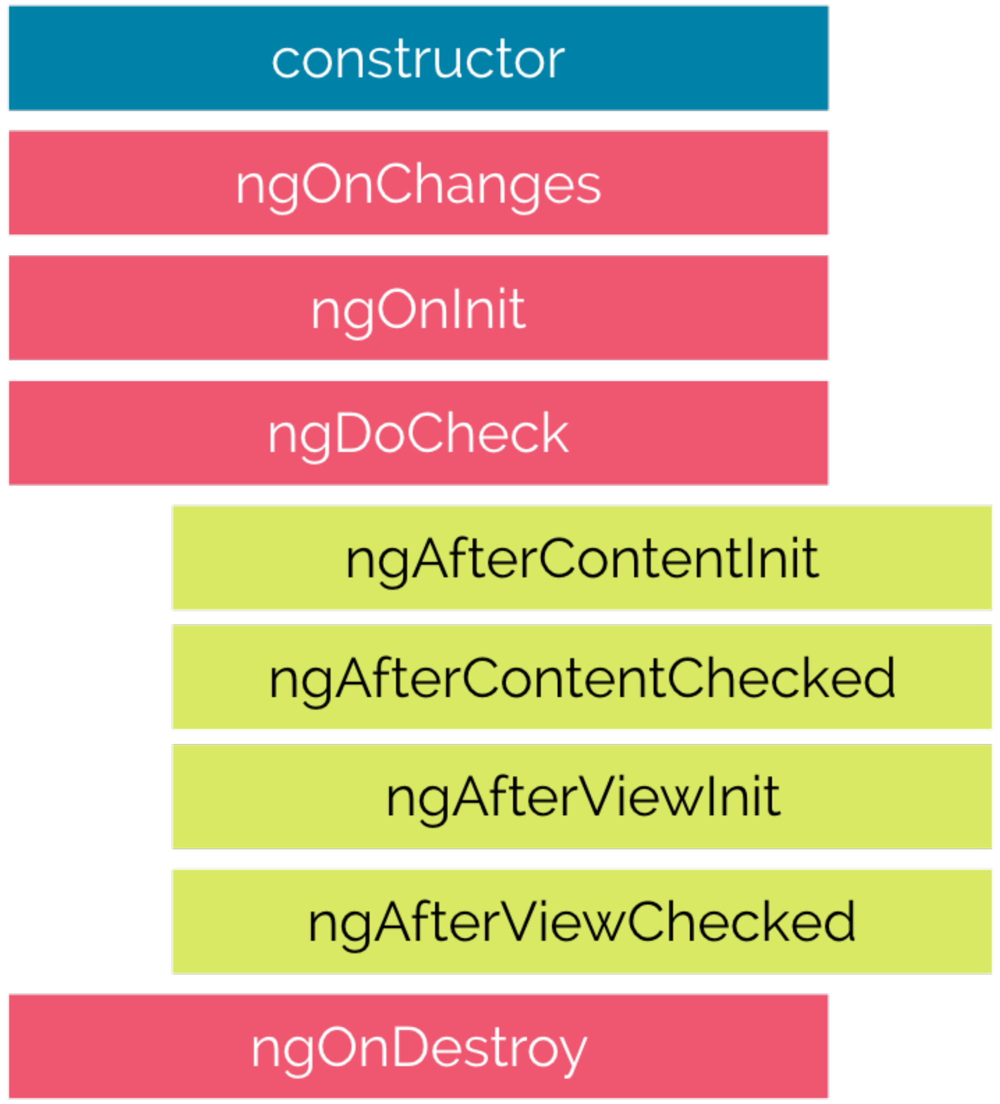


Image from https://codecraft.tv/courses/angular/components/lifecycle-hooks/

# Using Lifecycle Hooks

```
import { OnInit, OnDestroy } from '@angular/core';

export class AppComponent implements OnInit, OnDestroy {
   form!: FormGroup
   sub$! Subscription

   ngOnInit() {
      //From OnInit interface
      this.form = this.fb.group({ ... })
      this.sub$ = this.anObservable.subscribe(...)
   }

   ngOnDestroy() {
      //From OnDestroy interface
      this.sub$.unsubscribe()
   }
}
```

`ngOnInit` will be called just after component is created

Initialize components eg. create form, subscribe to observables

`ngOnDestroy` will be called just before component is destroyed

Clean up resources that can cause memory leaks before component is destroyed

# Define the Form Model

```
import { FormBuilder, FormGroup } from ‘@angular/forms’

@Component({ ... }
export class RSVPComponent implements OnInit {
  rsvpForm: FormGroup

  constructor(private fb: FormBuilder) { }

  ngOnInit() {
    this.rsvpForm = this.fb.group({
      name: this.fb.control<string>(‘’),
      email: this.fb.control<string>(‘’),
      phone: this.fb.control<string>(‘’),
      attending: this.fb.control<string>(‘’)

    })
  }
```

Helper service for building controls, groups and arrays

Create a group

Control name

Control type

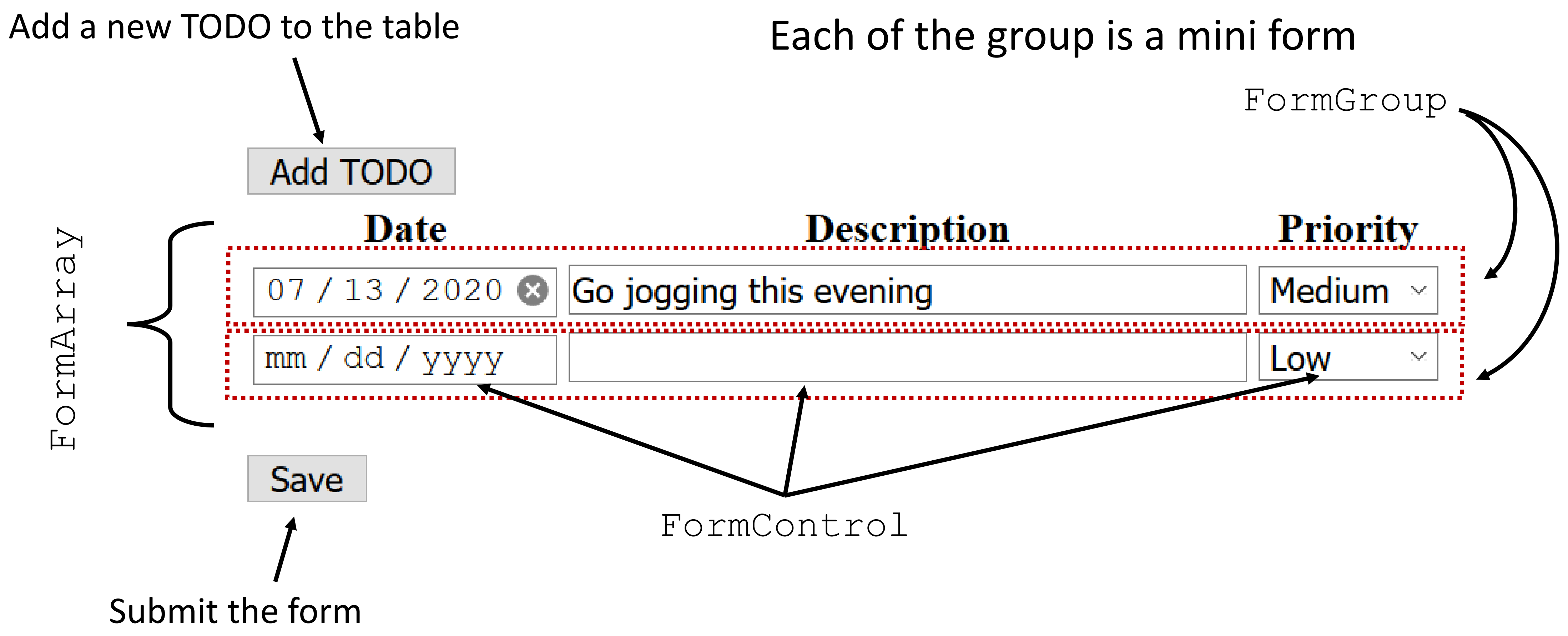
# Map FormGroup to the Form

Bind the `rsvpForm` (`FormGroup`) to the form by binding it to the `formGroup` attribute

Map each form field to the controls defined in the `formGroup`

```
<form [formGroup]="rsvpForm">

  Name: <input type="text" formControlName="name">
  Email: <input type="email" formControlName="email">
  Phone: <input type="tel" formControlName="phone">

  Attending:
  <input type="radio" value="yes" formControlName="attending"> YES
  <input type="radio" value="no" formControlName="attending"> NO

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

</form>
```

# Repeat Groups



# TODO Form

Add a new 'row' when this button is pressed

```
<form (ngSubmit)="processForm()">
  <button type="button">Add TODO</button>
    <tbody>
      <tr>
        <td> <input type="date"> </td>
        <td> <input type="text"> </td>
        <td>
          <select>
            <option value="low">Low</option>
            ...
          </select>
        </td>
      </td>
    </tbody>
  <button type="submit">Save</button>
</form>
```

Fragment of HTML table

Repeat these

Note: HTML form truncated for brevity

# Defining the Model

```
@Component({ })
export class TodoComponent implements OnInit {
  todoForm: FormGroup
  todoArray: FormArray
  constructor(private fb: FormBuilder) { }
  ngOnInit() {
    this.todoArray = this.fb.array([])
    this.todoForm = this.fb.group({ todos: this.todoArray })
  }
  addTodo() {
    const todoGroup = this.fb.group({
      date: this.fb.control<Date>(new Date()),
      description: this.fb.control<string>(''),
      priority: this.fb.control<string>('')
    })
    this.todoArray.push(todoGroup)
  }
}
```

Add array to the form (group)

Construct the group (mini form) and add it to the array

# Mapping the Model to the Form

```
<form [formGroup]="todoForm" (ngSubmit)="processForm()">
  <button type="button" (click)="addTodo()">Add TODO</button>
    <tbody formArrayName="todos">
      <tr *ngFor="let todo of todoArray.controls" [formGroup]="todo">
        <td> <input type="date" formControlName="date"> </td>
        <td> <input type="text" formControlName="description"> </td>
        <td>
          <select formControlName="priority">
            <option value="low">Low</option>
            ...
          </select>
        </td>
      </td>
    </tbody>
  <button type="submit">Save</button>
</form>
```

Each array element is a mini form

Loop thru the `todoArray`. Each element is a `FormGroup`. Bind the `FormGroup` to the `formGroup` directive; use `formControlName` to bind the controls to the form fields

Note: HTML form truncated for brevity

# Reading the Form Values

```
@Component({ ... }
export class RSVPComponent implements OnInit {
  rsvpForm: FormGroup

  constructor(private fb: FormBuilder) { }

  ngOnInit() { ... }

  processForm() {
    const rsvp = this.rsvpForm.value as RSVP
    // do something with rsvp
    ...
  }
```

Get the values from the group

## Validation

- The process of validating that the values are correct before processing
- Types of validation
  - Syntactic - valid format, length of an entry, appropriate number range, mandatory field, etc.
  - Semantic - user name is not available, withdrawing more than the balance, etc.
- Syntactic validation are performed on the client/browser
  - Ensure that data is clean before submitting to server for processing
- Semantic validation are performed on the server
  - Typically require checking against database

# Angular Form Validation

- Performs syntactic validation
- Comes with a set of build in validators
  - `required` - mandatory field
  - `requiredTrue` - required a checkbox to be checked
  - `email` - the entry is in a valid email format
  - `min`, `max` - validate a number range
  - `minLength`, `maxLength` - validates the minimum and maximum length
  - `pattern` - use regular expression to validate a field

# Form Validation

```
import { FormBuilder, FormGroup, Validators } from '@angular/forms'

@Component({ ... }
export class RSVPComponent implements OnInit {
   rsvpForm: FormGroup

   constructor(private fb: FormBuilder) { }

   ngOnInit() {
      this.rsvpForm = this.fb.group({
         name: this.fb.control<string>('', [ Validators.required ] ),
         email: this.fb.control<string>(''
               , [ Validators.required, Validators.email ]),
         phone: this.fb.control<string>(''),
         attending: this.fb.control<string>('', [ Validators.required ])

      })
   }
```

# Form Validity

- Is the form or a control is valid or invalid
  - `FormGroup.valid`
  - `FormGroup.invalid`
  - `FormControl.valid`
  - `FormControl.invalid`

- Is a field valid or invalid
  - `get()` returns the control from a group

```
rsvpForm.get('email').valid
```

- Does a field has a specific type of error

```
rsvpForm.get('email')
     .hasError('email')
```

STACKUP

# Affordance - Hints and Error Messages

```
<form [formGroup]="rsvpForm">
  Name: <input type="text" formControlName="name">
  <div *ngIf="rsvpForm.get('name').hasError('required')">
    Please enter your name
  </div>
  Email: <input type="email" formControlName="email">
  <div *ngIf="rsvpForm.get('email').hasError('required')">
    Please enter your email
  </div>
  <div *ngIf="rsvpForm.get('email').hasError('email')">
    Please enter a valid email
  </div>
  ...
  <button type="submit" [disabled]="rsvpForm.invalid">
    Send
  </button>

</form>
```

Display a message if a control has a specific error

Only enable the submit button if the entire form is valid

# Angular Deploy

- Angular has addons for deploying to various CDNs
    - Firebase hosting - `@angular/fire`
    - Azure - `@azure/ng-deploy`
    - Now - `@zeit/ng-deploy`
    - Github pages - `angular-cli-ghpages,` free
- Deploy directly, not via addons, to CDN
    - Vercel - used in this course
    - Netlify
    - S3
- Manual by installing webserver

## Publishing to GitHub Pages

- GitHub Pages is a web hosting website for static pages
  - There are limitations
  - See https://help.github.com/articles/what-is-github-pages/#usage-limits
- Accessed with the following

```
https://<username>.github.io/<repo_name>
```

- GitHub will serve pages from **`gh-pages`** branch of your repository
- Add deployment to project
  - See https://www.npmjs.com/package/angular-cli-ghpages

```
ng add angular-cli-ghpages
```

# Deploying to Github Pages

- Compiled Angular is pushed to `gh-pages` branch
- Angular application is hosted under

```
https://<username>.github.io/<repo_name>
```

- Need to ‘shift’ Angular’s base from `/` to `/<repo_name>/`
  - Otherwise loading resources will fail

## Deploying to Github Pages

```
ng deploy --base-href=/myapp/
```

Must have the last /

```
ng deploy
```

angular.json

```
...
"deploy": {
    "builder": "angular-cli-ghpages:deploy",
     "options": {
       "baseHref": "/myapp/"
    }
}
```

# Deploying to Github Pages with Custom Domain

OPTIONAL

```
ng deploy --cname=myapp.acme.com
```

- Do not need to use `--base-href`
- `--cname` option will create a file called `CNAME` in the compiled Angular application
  - Contains your domain name
- Set custom domain manually if you forget to set custom domain during deployment
  - Settings (top right corner), Pages, Custom Domain
- Need to set CNAME record in DNS
  - You must own the domain!

```
myapp CNAME <username>.github.io
```

# Web Server

- Serves content over the web using HTTP
  - Eg. HTML pages, videos, audios
- Compile Angular application and serve from a web server
  - Create an 'empty' Spring Boot application, copy files to `src/main/resources/static` directory
  - Use a web server like Nginx or Caddy
- Nginx - [https://nginx.org/en/download.html](https://nginx.org/en/download.html)
- CaddyServer - [https://caddyserver.com/download](https://caddyserver.com/download)

# Caddyfile - Caddy Configuration File

**Caddyfile**

```
:8080 {

  encode zstd gzip

  root * ./static

  try_file {path} {path}/index.html =404

  file_server

  handle_errors {
     try_files /404.html
     file_server
  }
}
```

Server from port 8080. Use `{$PORT}` for binding to environment variable

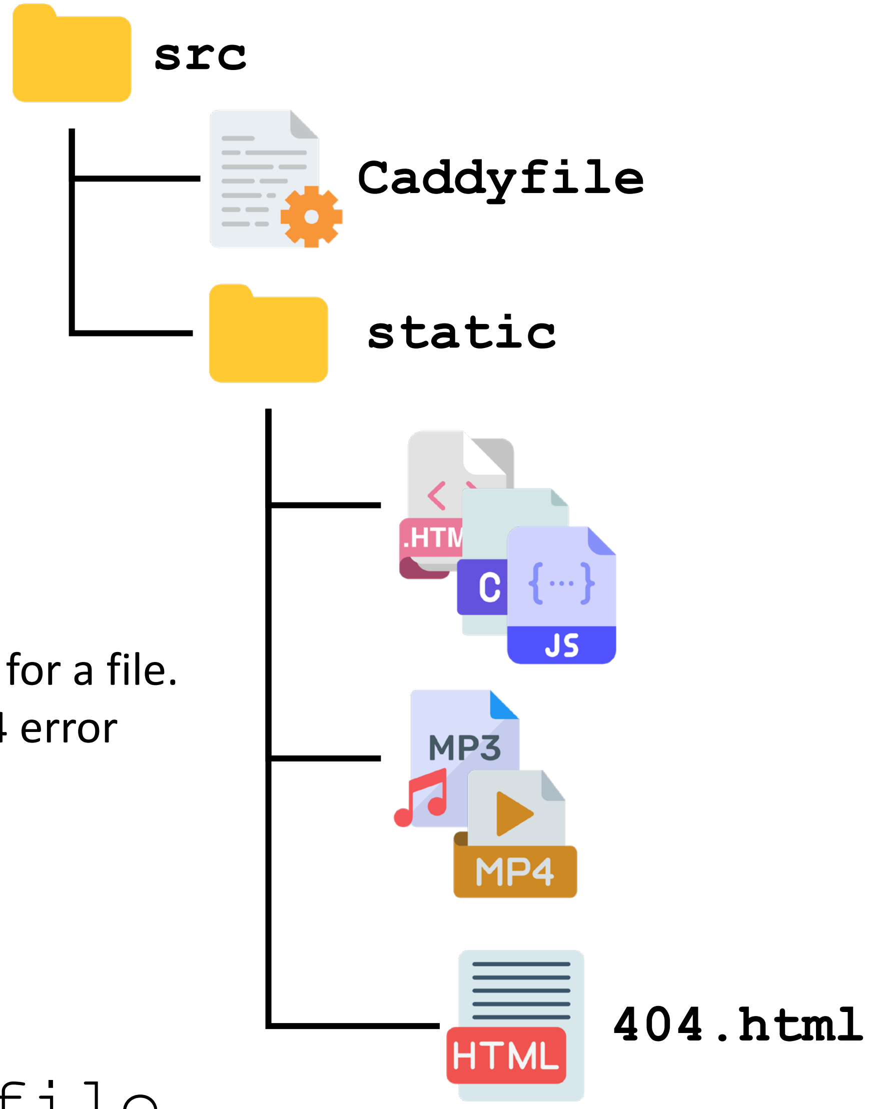
Compress the file

Get all files from `static` directory

Different options to look for a file. If not found emit the 404 error

Serve the file

If there are any errors, return the 404.html file



START

```
caddy run -config Caddyfile
```

# Unused

## Custom Validators

```
import { ValidatorFn, AbstractControl, ValidationErrors }
   from '@angular/forms'

const nonWhiteSpace = (ctrl: AbstractControl) => {
   if (ctrl.value.trim().length > 0)
      return (null)
   return { nonWhiteSpace: true } as ValidationErrors
}
```

Custom validators are functions with a single `AbstractControl` parameter

Returns null if no error

Returns a object indicating what error(s) have occurred `formCtrl.hasError()` method checks this

```
   emailCtrl = this.fb.control('', [
         Validators.required, Validators.email,
         Validators.nonWhiteSpace
   ])
```