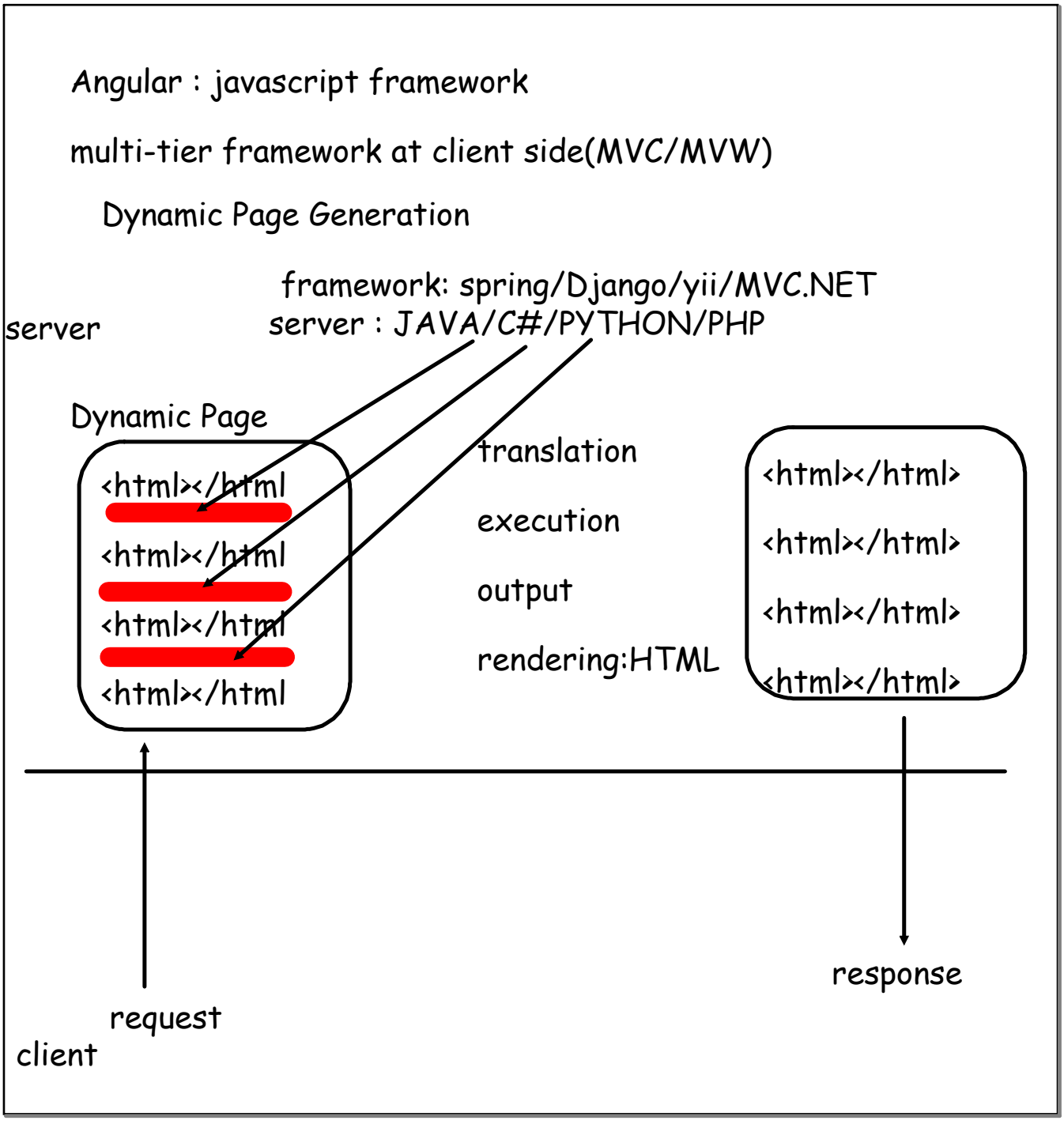
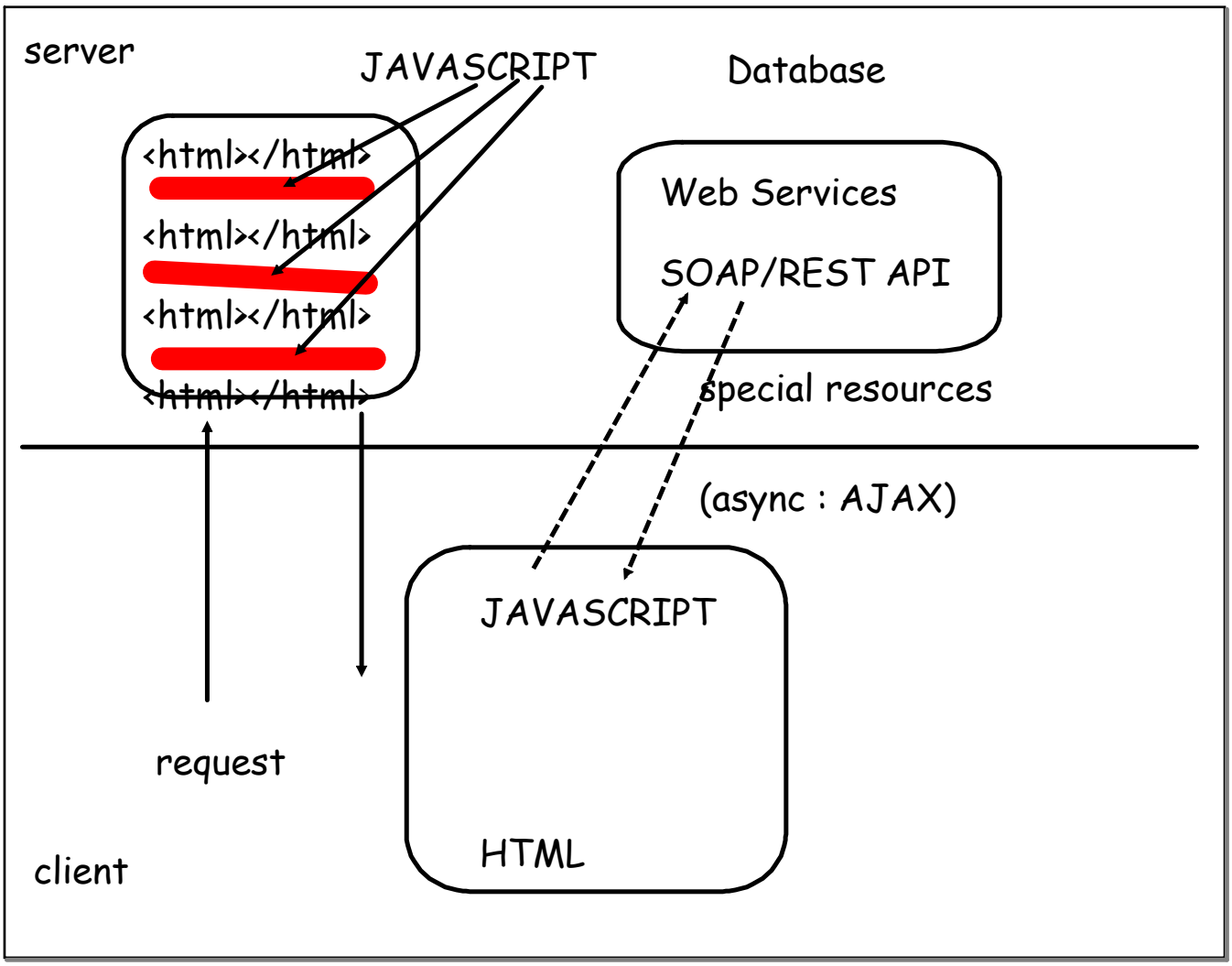

Angular : javascript framework
multi-tier framework at client side(MVC/MVW)
Dynamic Page Generation
framework: spring/Django/yii/MVC.NET
server : JAVA/C#/PYTHON/PHP
server
Dynamic Page
<html></html
<html></html
<html></html
<html></html
translation
execution
output
rendering:HTML
<html></html>
<html></html>
<html></html>
<html></html>
response
request
client


server
JAVASCRIPT
Database
<html></html>
<html></html>
<html></html>
<html></html>
Web Services
SOAP/REST API
special resources
(async : AJAX)
JAVASCRIPT
request
client
HTML

ANGULAR JS 1/2

library : like jquery

new methods /

model/controller/classes/ service to interact with server

Extension to existing HTML Component

new attributes for HTML

adding more logic behind HTML Components

ANGULAR 4 : corresponds to any server side framework

Base Script of ANGULAR : TS(Type script) + ES6

Resources : Client Side JS Community

NODE JS (Projects)

library and set of standard : repository of tool

ES6

TS

Angular

React + REdux

Server Side JS (NODE JS)

TS + ES6--> Transpiled -->ES5 (client)

ES6 : (Babel/Treceaur)

TS (typescript) :

Collabaration : MS + Google

Node Js : tool (npm tool : command (cli)) (Node package manager)

manage/fetch resources from repository

initialize new projects :

Typescript : Type System : named datatypes

External JAVASCRIPT : ext : .ts

TS

1. Type systems

2. classes(ES5/ES6: OOPs : prototype)

3. decorators(annotations)

4. imports (load/include res)

5 utilities : strong programming language

Type System

Classes

1string

2number(float)

3boolean(true/false)

4.enums (enumerated datatypes)

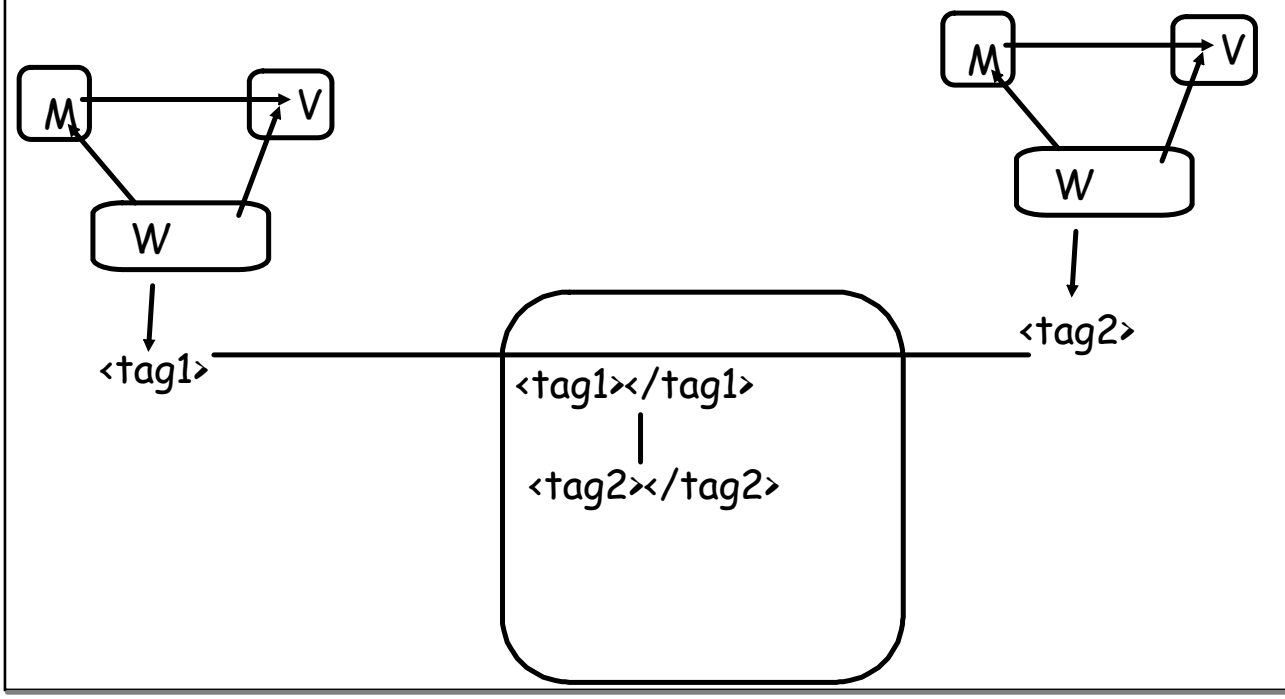
5. Array

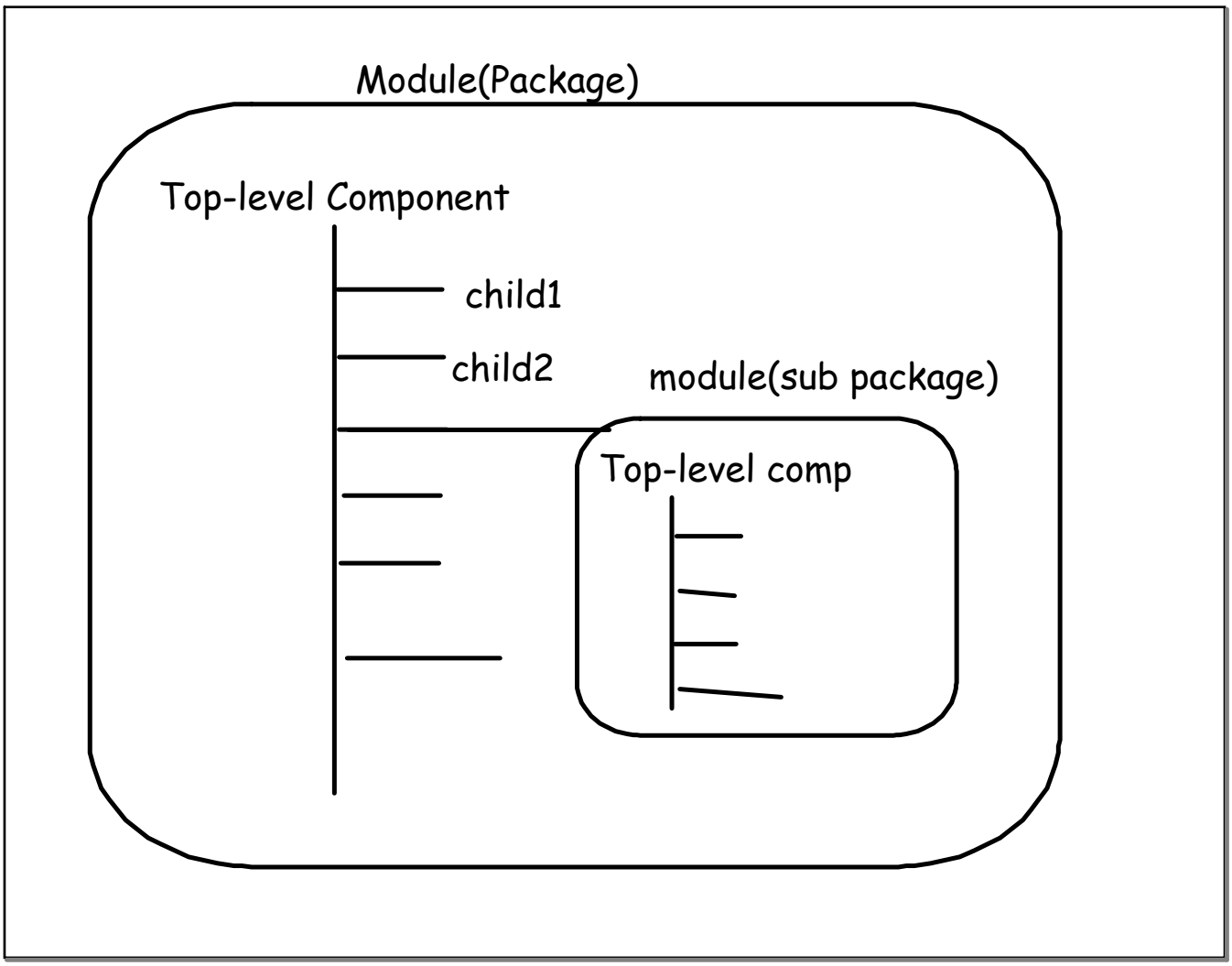
6. any(default datatype of ES5/ES6)

7. void : return type of function

Create new HTML Component : Component Oriented Programming

new HTML Tag : multi-tier (MVW) :




Module(Package)
Top-level Component
child1
child2
module(sub package)
Top-level comp

Angular Tools : angular/cli : command

tools/command : build/manage/organize/update/build-launch

Download cli

npm:

1. Standard project structure :
2. Download all dependancies/library

npm install -g @angular/cli : download and install angular tools

ng new first-app : creates new project named first-app

by default structure :

create one web page : Home page (index.html)

Top level Module (package)

One top level component inside the Module(TAG)

e2e : contain resources for end to end testing

node_modules : all dependency and library

src : all our source-code

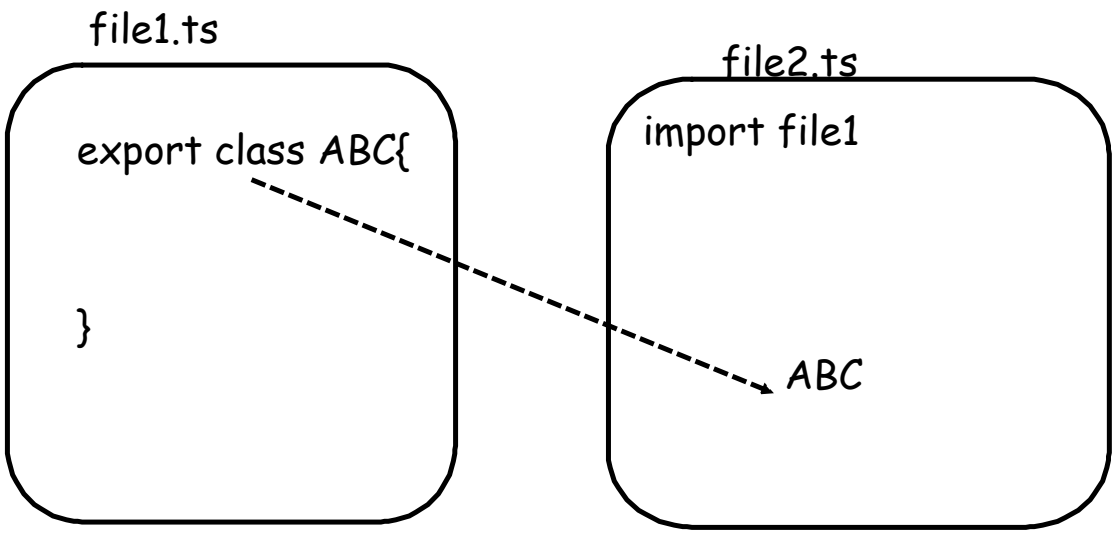
angular.json : angular project configuration file

package.json : npm configuration (dependency/library)

tsconfig.,json : typescript/ES6 config

Module : Container

class to represent : app.module.ts



import {<class name1>,<class name2>} from <file name>

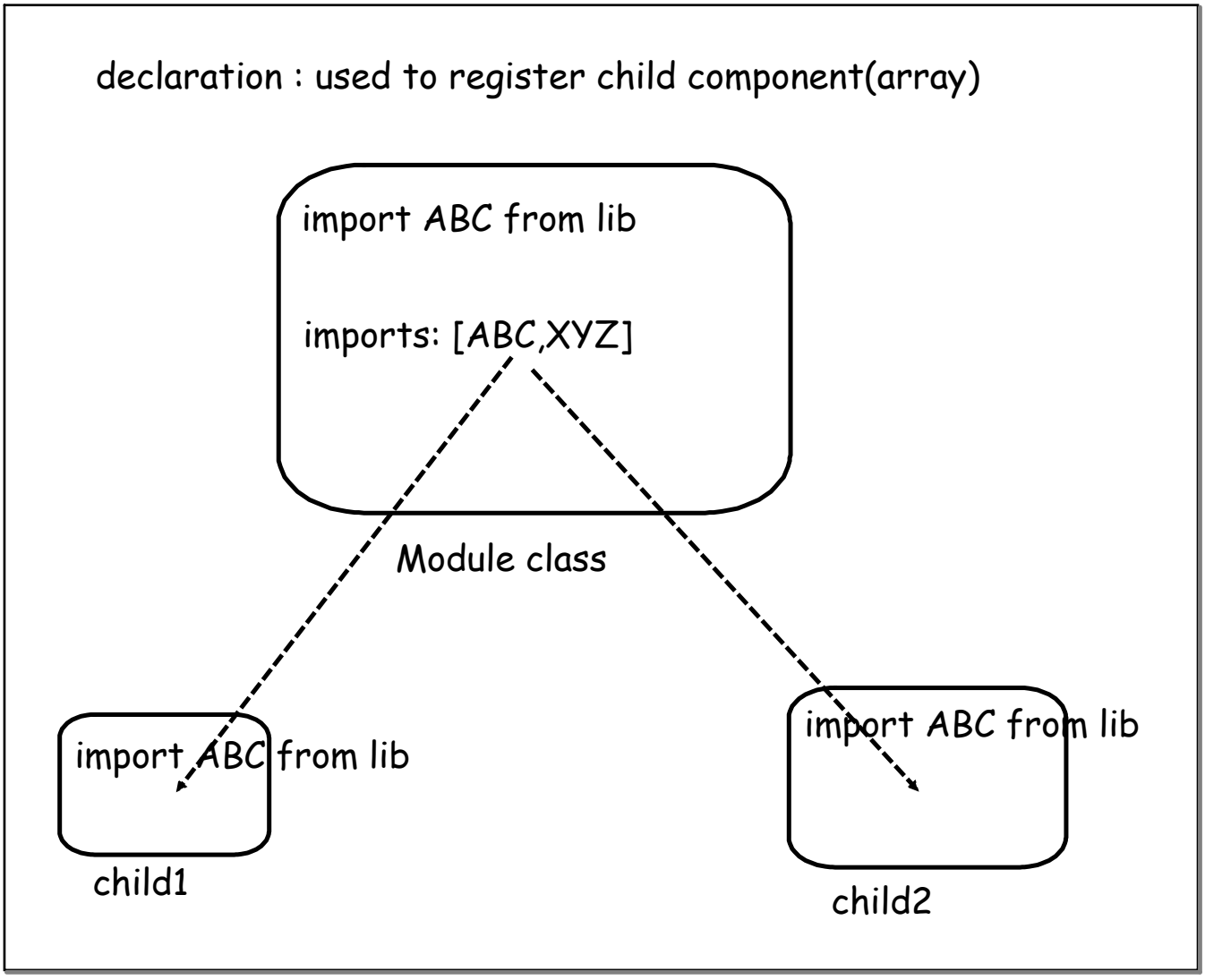
Decorator/Annotation :

can be applied on classes/member

```
@NgModule({
    key1 : value1,
    key2:value2

})
class ABC{
}
```

1. Mark any entity for some special purpose/service : identity
2. provide some additional data(meta -data)


declaration : used to register child component(array)
import ABC from lib
imports: [ABC,XYZ]
Module class
import ABC from lib
child1
import ABC from lib
child2

providers : used to register the shared services

services : cut-accross task req to be performed in multiple component

bootstrap : defines the root component

app.component.ts : class backing your component tag(logic & data)

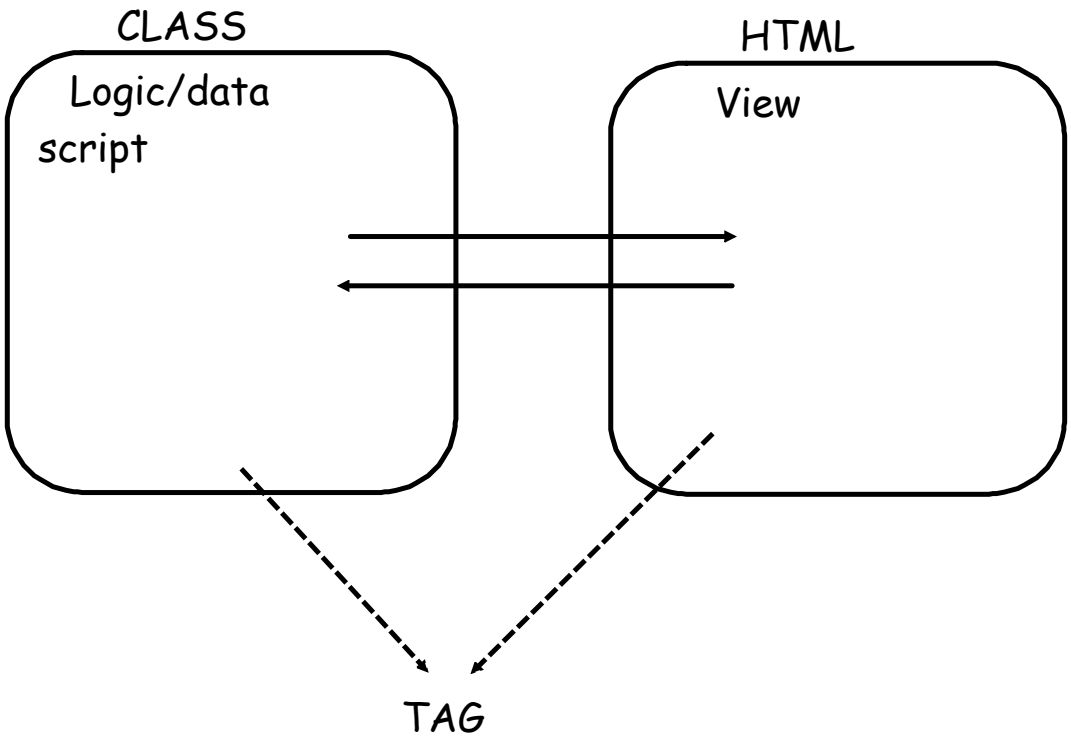
app.component.html : file backing your component view(presented)

app.component.css : style sheet for that particular html :compone nt oriented

app.component.spec.ts : unit testing that component

selector : tag-name

ng generate component my-comp: create a new component my-comp



↘ HTML has direct access over data members if class

ANGULAR JS 1 : two way data access

ANGULAR 4: one way access activated : configure explicity activate two way

Directives :

ANGULAR JS 1/2:

new attributes to HTML comp :

directives

structural directives

: control: add/remove the DOM Object from DOM Tree

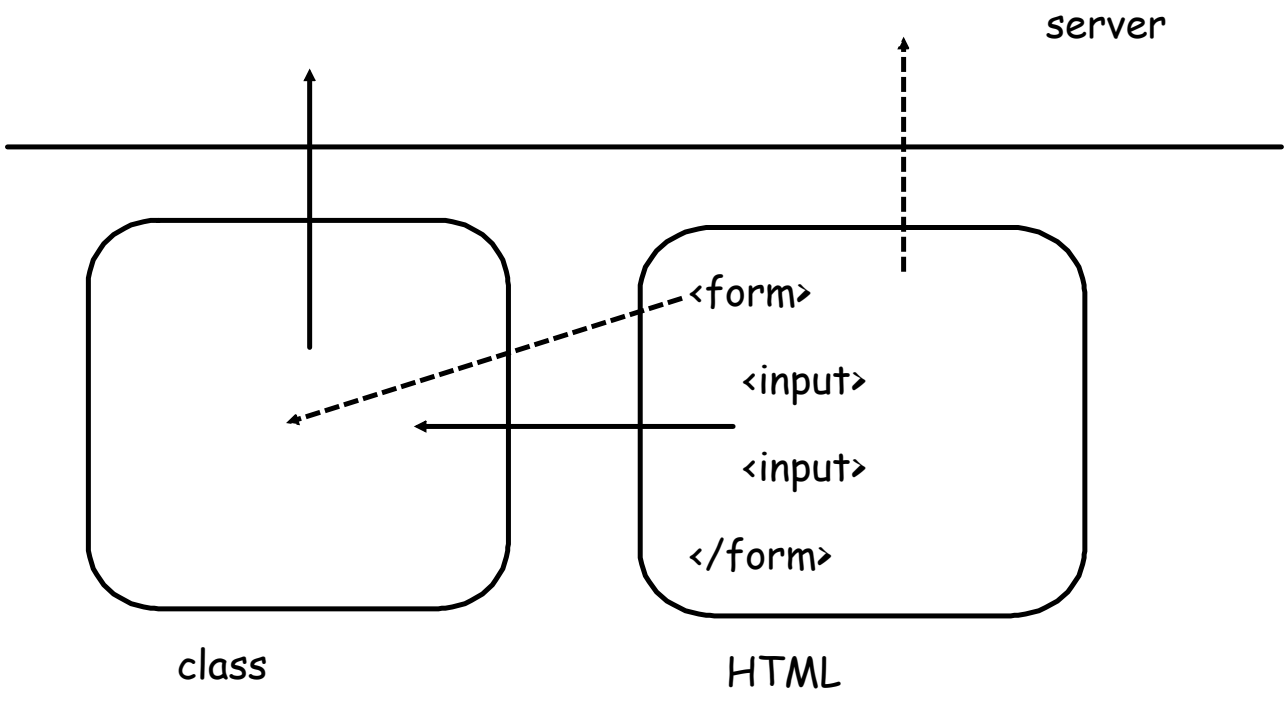
ngIf : conditionally add/remove (control visibility) of DOM Components

ngSwitch

ngFor

Events :

Angular : inbuilt event delegates(function) : HTML EVENT



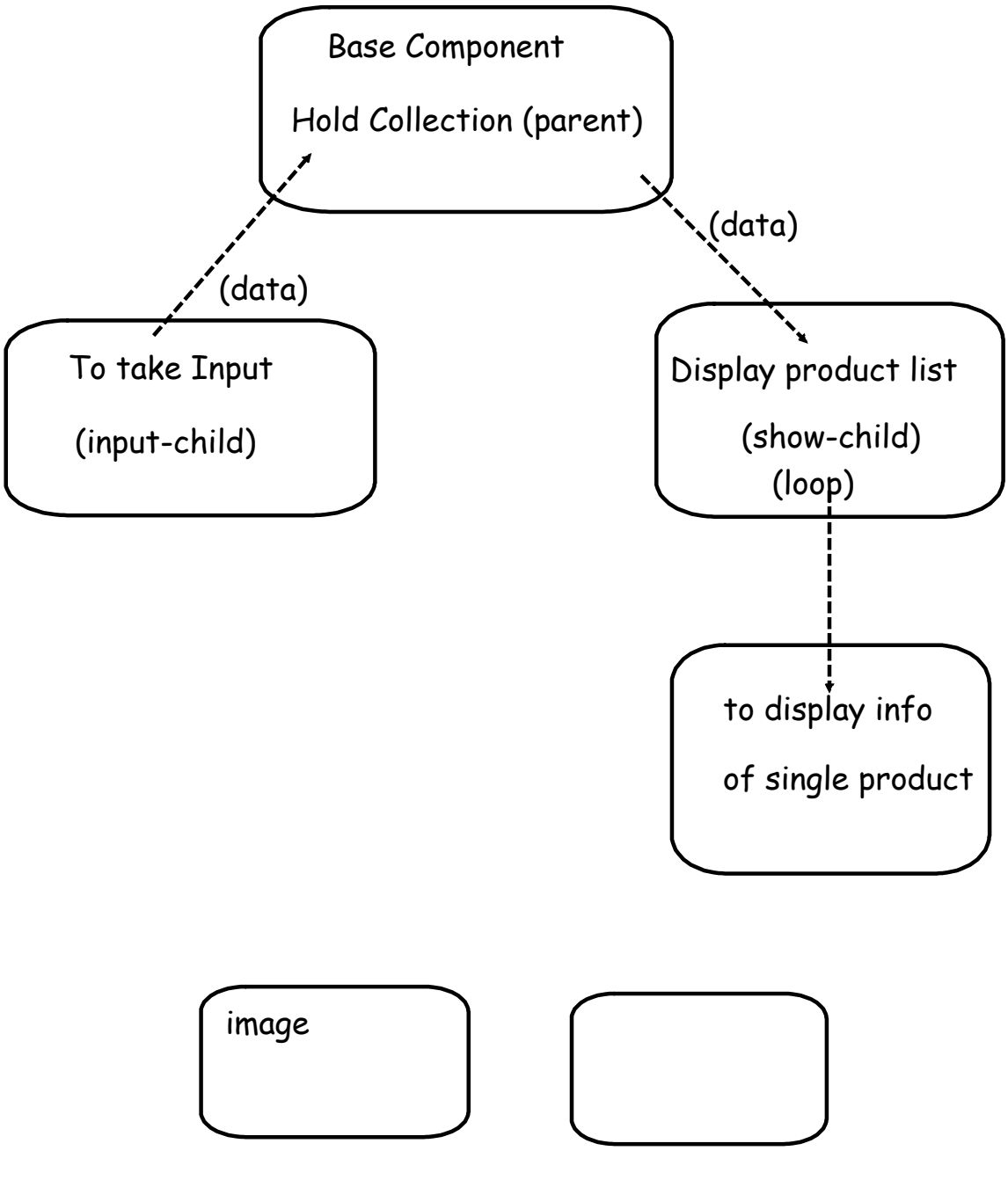
Functional Component : MVW/MVC

1. Hold a collection/list of some products : MODAL

create a modal : product: data structure : class (different properties)

2. Display all products

3. Take input of new product , add it to existing collection and reflecting on display



Nov 20-5:18 PM

Forms in Angular:

library support to deal with forms

inbuilt modules : container/package holding multiple resources : group of component / group of classes / predefined global variable

Two module for forms dealing

1. FormsModule

2. ReactiveFormsModule

Two important classes

1. FormControl : correspond to inputHTML DOM object component of a form eg : text/dropdown/checkbox

2. FormGroup : corresponds to HTML Dom of form(<form>)

FormGroup holds collection of FormControls

Two important inbuilt global objects :

1. ngModel : object of FormControl (directive) : used as an attribute of HTML Component (input)

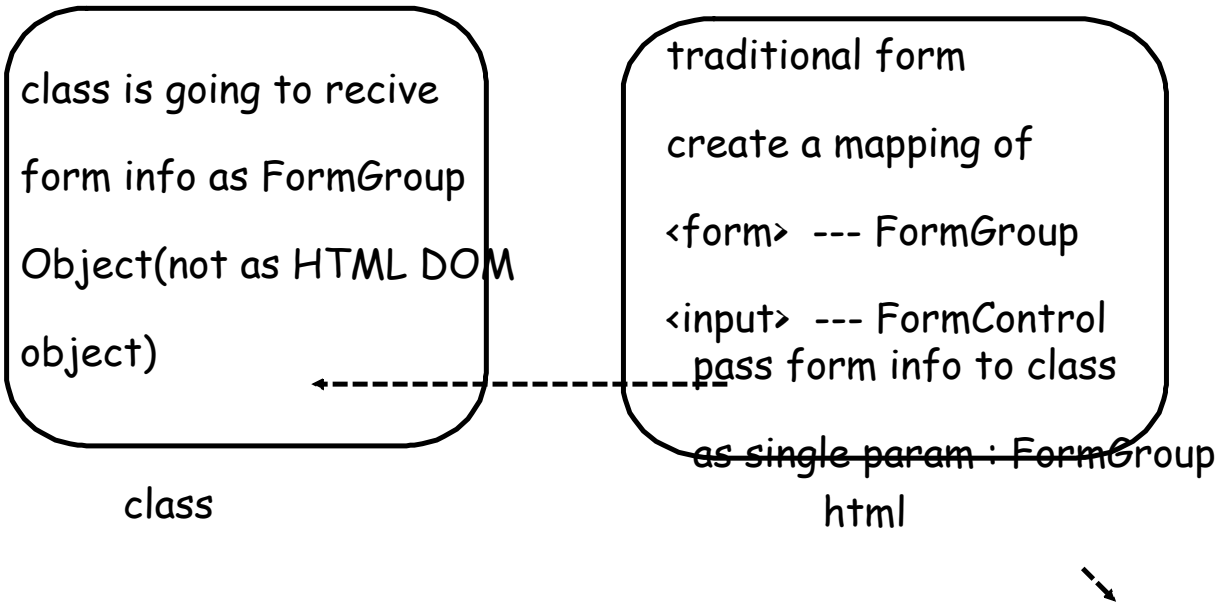
2. ngForm : object of FormGroup (directive) : for <form> component

Two different ways to handle the form

1. Template

2. Model

Template

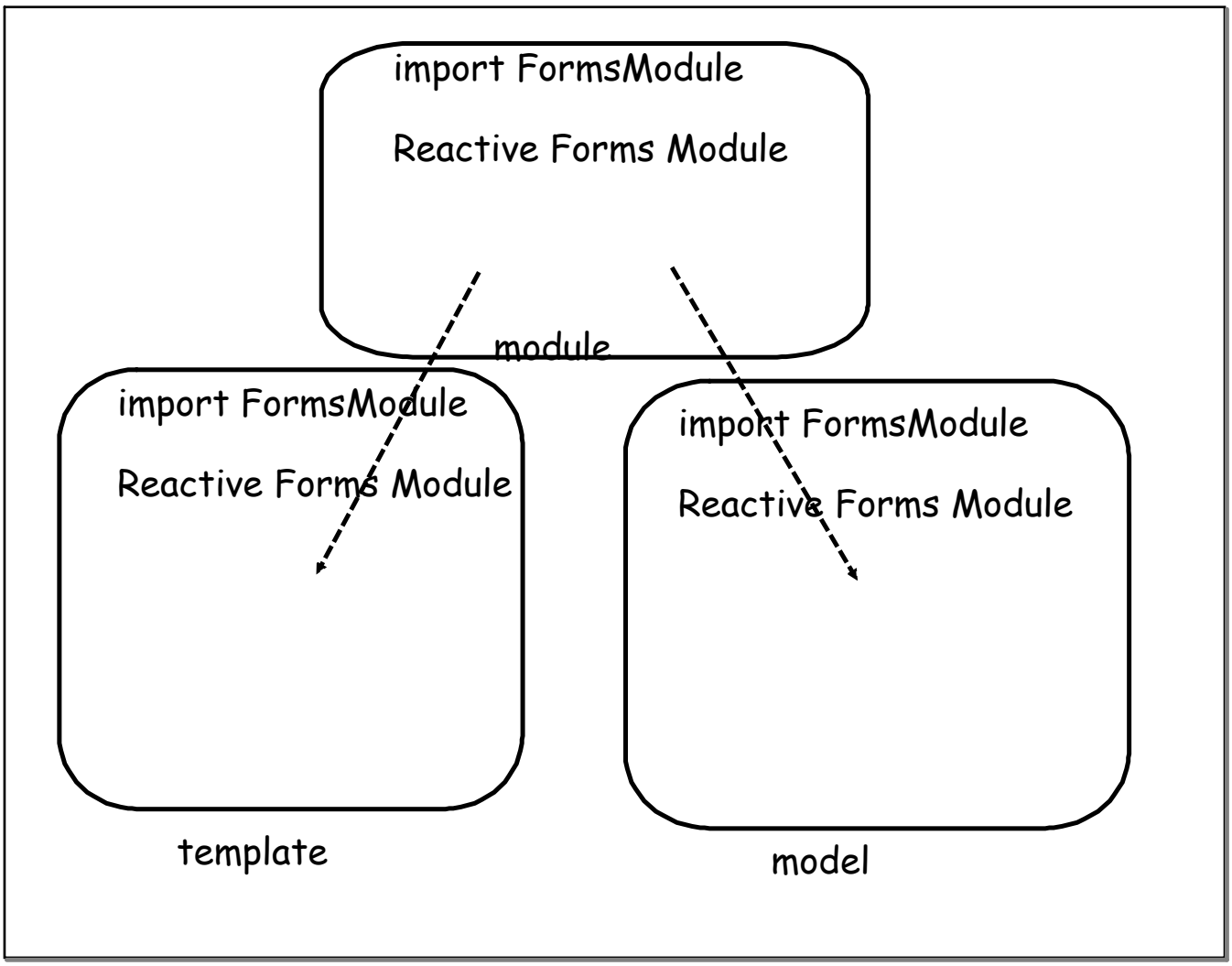


Traditional : angular id : transfered each comp info individually

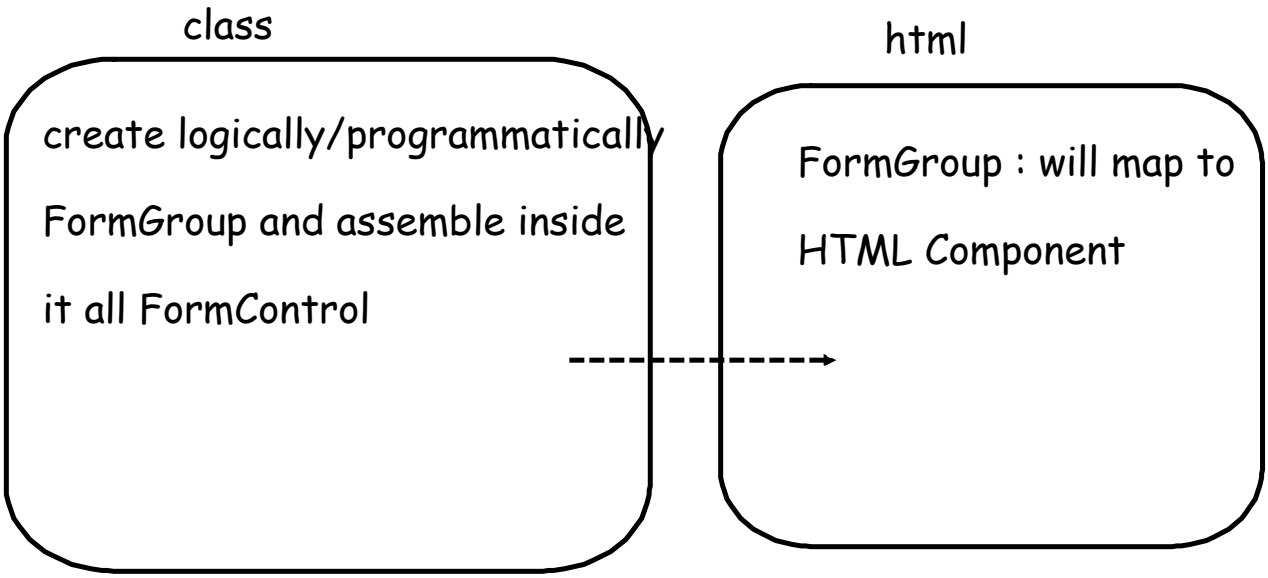
save(a,b,c,....)

Template : transfer complete form group as single argument

save(g); form group


import FormsModule
Reactive Forms Module
module
import FormsModule
Reactive Forms Module
import FormsModule
Reactive Forms Module
template
model

Model



Model :

no need to transfer formgroup info from html to class

class will have direct/auto access over HTML data

To create form group : Helper class (FormBuilder)

Reactive Forms Validations:

1. required (mandatory)

2. pattern (input should match)

Angular : does enforces validation

sets the validation status : needs to checked

features of FormControl :

1 valid : inbuilt variable : set to true if all constraint/validation applied on form control are satisfied

2. dirty : inbuilt variable : set to true when any modification is done on form control after loading

3. hasError(<type of constraint>) : return boolean true : if a particular constraint is not met

Form Group :

1 valid : inbuilt variable : set to true if all constraint/validation applied on all the form control are satisfied

2. dirty : inbuilt variable : set to true when any modification is done on any form control after loading

3. hasError(<type of constraint>) : return boolean true : if a particular constraint is not met on any of the formControl

Two way data transfer : implement explicit (FormsModule)

Directive : Pipes : Services

Directive : custom directives~component (attributes)

1. Access over events of parent/host HTML Component
2. Receive values from HTML Component
3. have access over other html attributes

4 Pillars

Component | Directive | Service | Routing

Pipes / Filter

Data Transformation

(<src data> | <tranformation rule>) : new formatted data

1. chaining of pipes

2. extension : specify transformation more specifically

   <transformation> : <extension>

inbuilt Pipes

custom Pipes

lowercase

uppercase

decimal

percent

date

currency

slice

json

Pipe : class

pipe : find square root

Service :

to create resources that are shareable among component and other service

To implement : dependency Injection framework

target the logic of muti-tier:

1. Seperation of concerns (validation/server interaction/ log)
2. Dependency Injection (loose Coupling)

DI framework : singleton object of service

broadcasting

Dependancy Injection : Loose Coupling

service-component :

Relationship shall not be hard-coded : controlled from outside

how to apply

1. programmatic style
2. DI Framework api

Email ~ SMS(new Service)

inbuilt services :

Http Service:

interact with server (async) : takes use of AZAX in backend

import HttpModule

all http methods : get/post/put/delete/patch... : rest apis

json-server

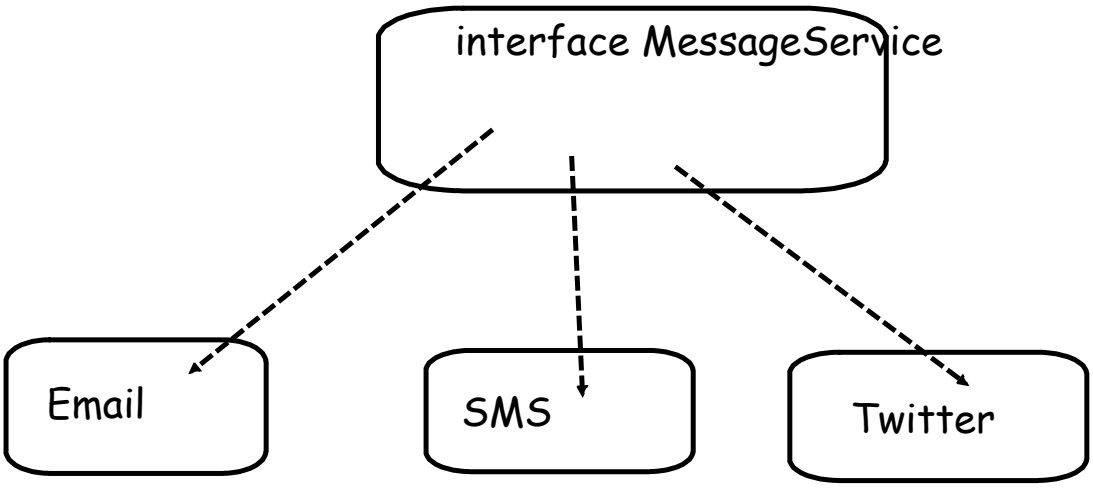
1. create own database : custom json file (inside app)

2. tool by npm : json-server : npm install -g json-server

3. run json-server : json-server --watch <path and name of json file>

4. when server runs : show a url (http://localhost:3000/<name of json file>)

Directive ~ class



Common presentation : interface (method/utilities)

Router Mechanism

Unit Testing in Angular

Angular : SPA (Single Page Application)

All resource would be available under one url

exclusive html : index.html

all other html : template of component

UI/services : comp can be added and removed as tags

different type of services : presented independently

switch to a new interface : Routers

1. if user refresh the page : get back to original/first

2. bookmark on contact

3. share a url of about us

Routing : create diff urls for diff comp:

switch form 1 url to another : async

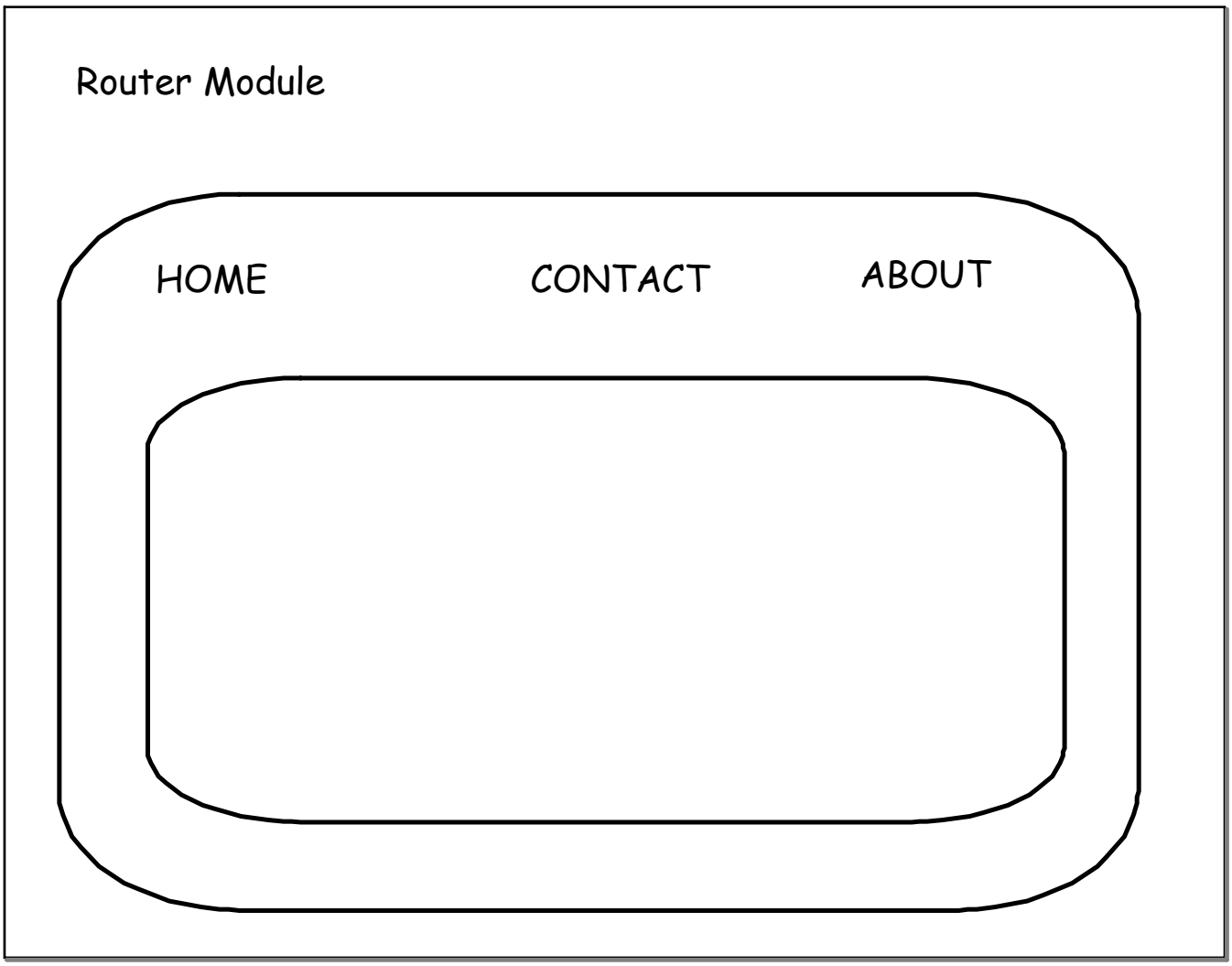
eg : HOME

CONTACT

ABOUT

REG

PROD :


Router Module
HOME
CONTACT
ABOUT

-

1. import of Router Module
2. create array of Route Object
3. import array
4. Receive the injected Router
5. Directive [routerLink] to refer to url ext
6. Tag <router-outlet> as container

to transfer data with navigation :route object should be config

Unit Testing in Angular

Automated Testing...

Extra Effort : code extra (test code)

while development (TDD)

Unit Test :

testing in isolation

easy to write

Angular : (without involving template)

Integration test

unit test extended for involving external resources

End to End Test

Entire app as whole

Unit Test Angular:

Tools :

Jasmine and Karma (Unit test)

Angular testing tool : TestBed (Integration testing)

E2E : Protractor

Jasmine : Framework/apis to test javascript

Does not involve DOM

Karma : Test Runner

by default integrated in project

1. Code Test cases : file : shall extension spec.ts

1. describe("suite-name",function to contain test cases)

:define a test suite(group of related test: container)

2. it("spec-name/description",function containing test logic)

: define a test/spec

3. expect()

expect(actual result we have received)

4. group of matcher function

what type of matching is required : matching of result with multiple values...

expect(result).toBe('value');

expect(result).toContain('value');

expect(result).toEqual('value');

expect(result).toBeNull('value');

expect(result).toBeTruthy('value');

npm install -g rxjs@6 rxjs-compat@6 --save (save dependancy in project)

import { map } from 'rxjs/operators'

this.http.get("").map(()=>{});

~

this.http.get("").pipe(map(()=>{}));

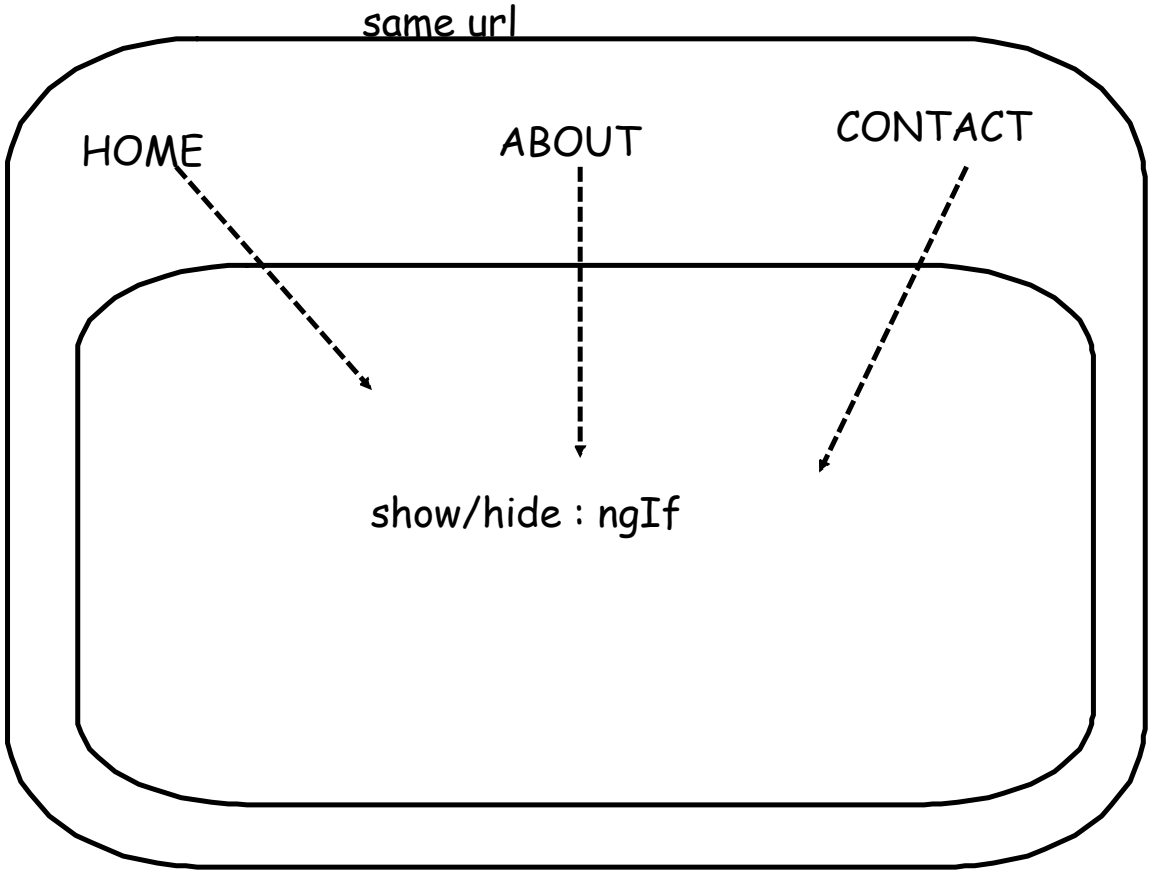
karma to scan all file : spec.ts

Routing Mechanism

navigate between diff components

Angular : SPA : Single Page Application

Tradition : seperate pages | Angular : seperate Comp



1. Refresh : get root comp back(initial) : maintain the state is req

2. not be bookmark page :req

3. share the url of service

Routing Mechanism :

provide diff url for diff comp

get changed aysnc

easy navigation

Unit Testing : TDD : Test Driven Development

Manual : Tradition

Automated testing: tools that can test resources

Levels of testing....

1. Unit testing :

Test a individual element in isolation : without any external res/ depandancy

component : without html

2. Integrated Testing

extension : unit + external resources

3. End to End Testing

Entire application

Extra Effort : extra code: test code

Angular Testing Tools

unit testing : Jasmine and Karma

Integrated Testing : Angular testing utility (TestBed + Jasmine + Karma)

End to End : Protractor

installed in every angular projected

Jasmine : Testing framework for testing javascript code

Karma : Test runner : environment

Jasmine Api :

==> all test case shall be coded in seperate file ext : spec.ts

Apis:

1. describe() : define a test suite : container to hold a group of related test | spec

describe("suite-name",function that hold test cases)

2. it() : define the test case : spec

it("spec-name/description", function that hold logic of test)

3. expect()

expect(actual result)

4. Group of functions : Matcher function

fun(expected value) (to be successfull)

Karma : scan your app for spec.ts :

going to run test on all of them

```
expect(result).toBe(excepted);

expect(result).toContain(excepted);

expect(result).toEqual(excepted);

expect(result).toBeNull();

expect(result).toBeTruthy();
```