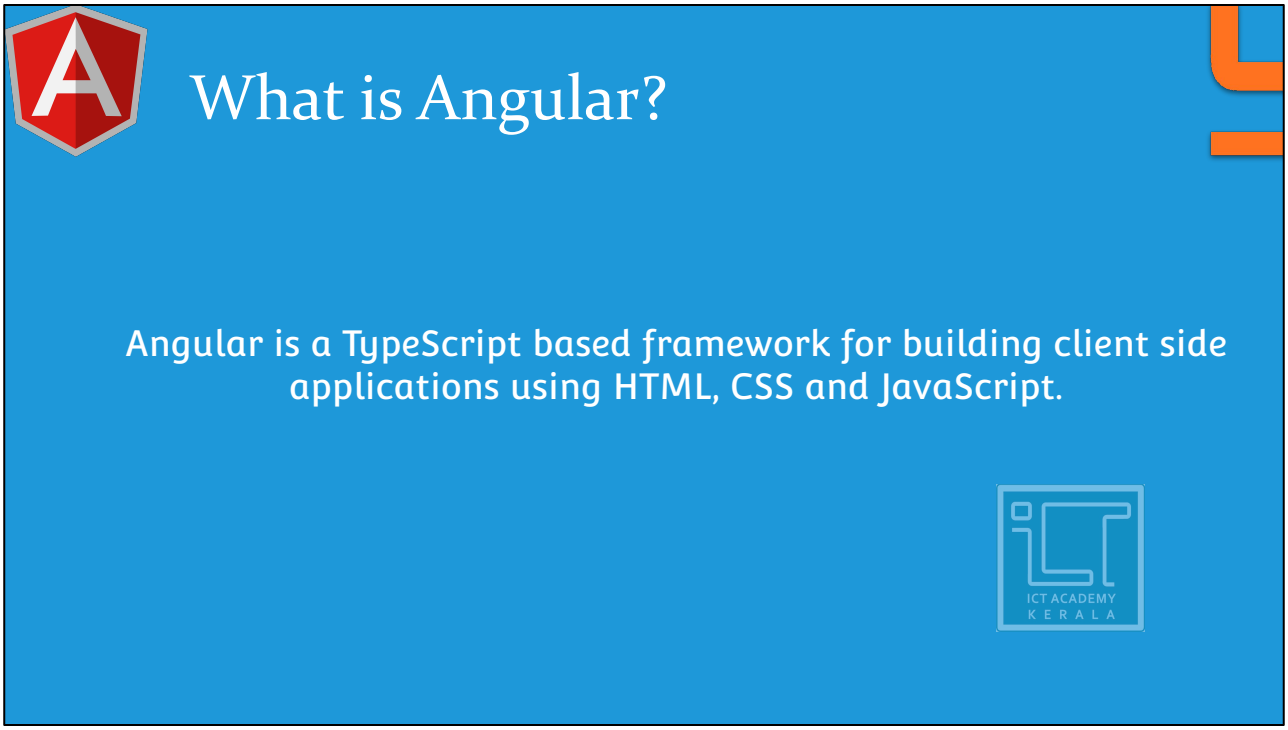
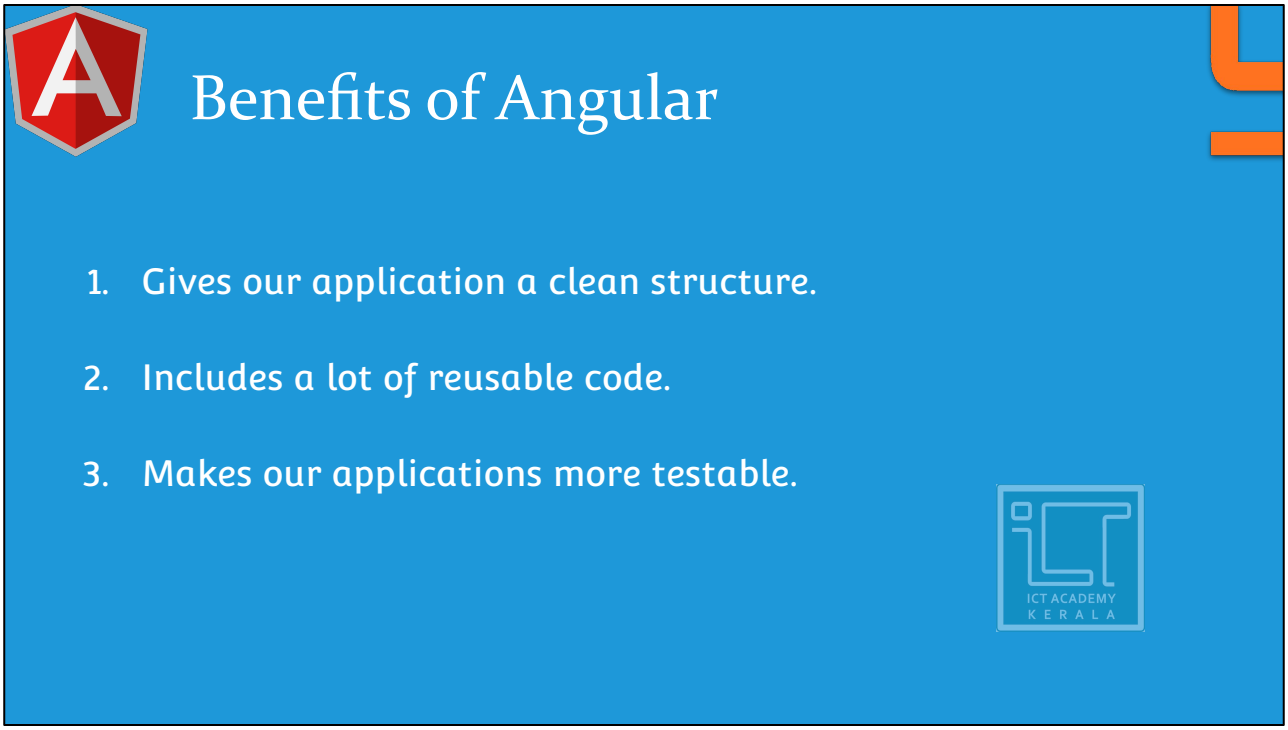


TypeScript:

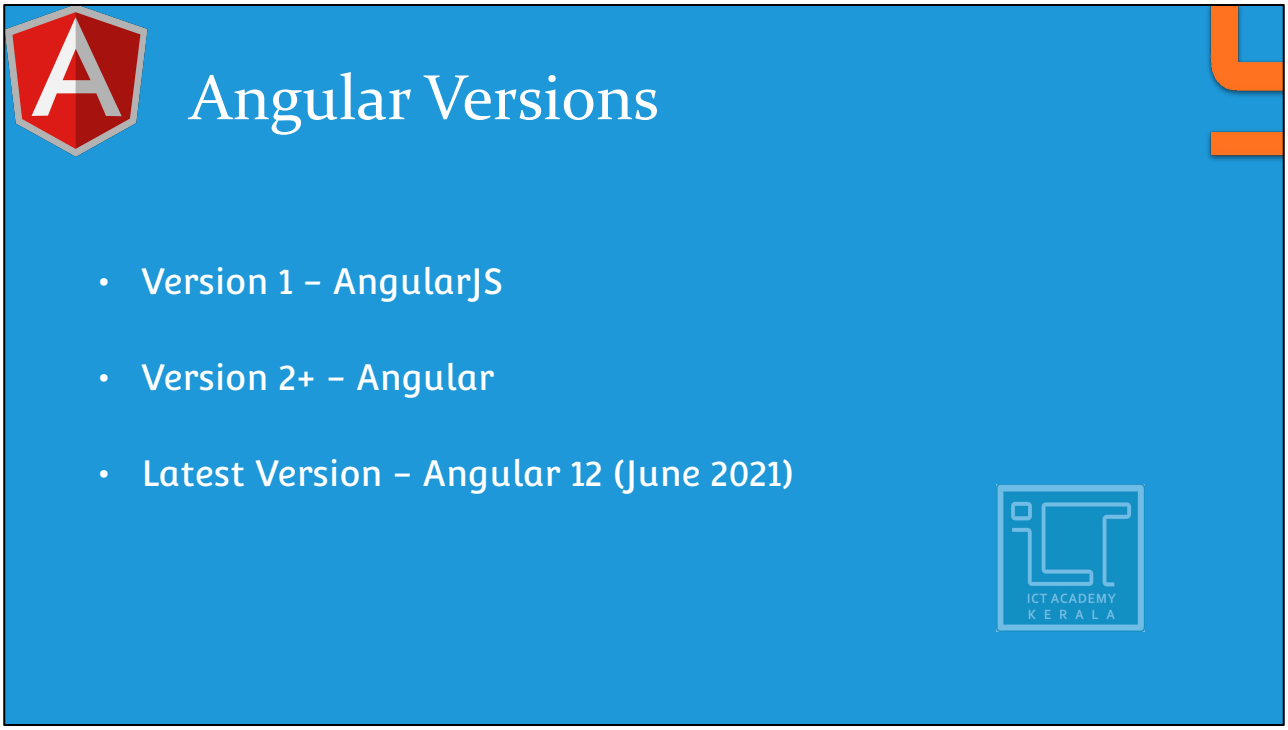
- Compiles to JavaScript
- Angular itself is run with typeScript

Why do we need Angular?

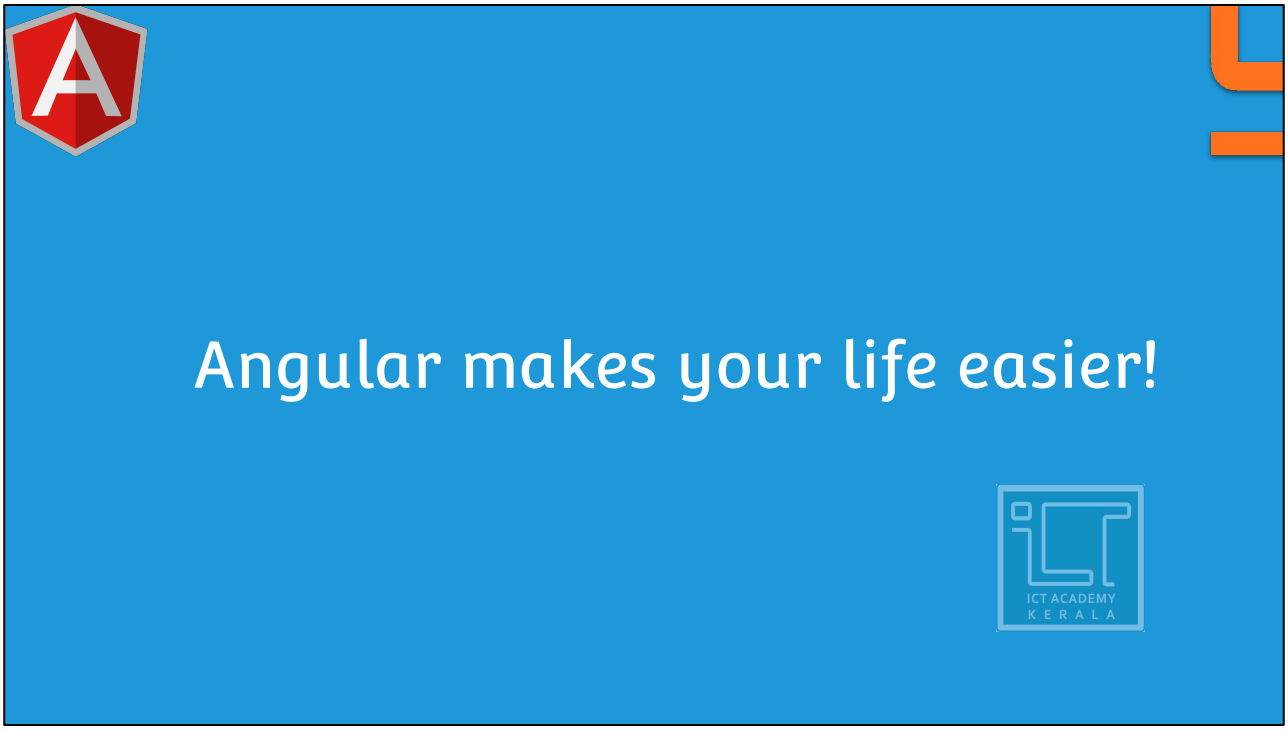
- Can use vanilla JavaScript/JQuery code.
- But for complex code, vanilla JS/JQuery gets harder to maintain.
- A way to properly structure our applications.
- To make web application development easier.

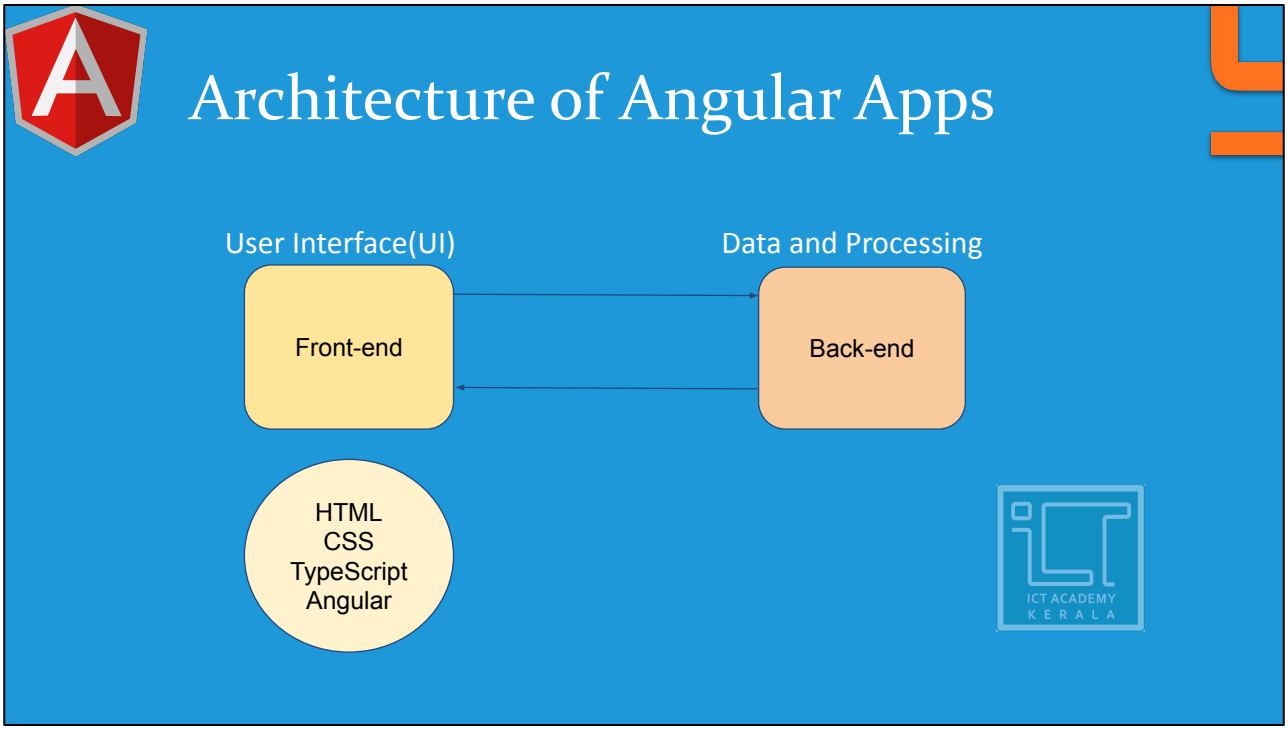


Easy to understand and write



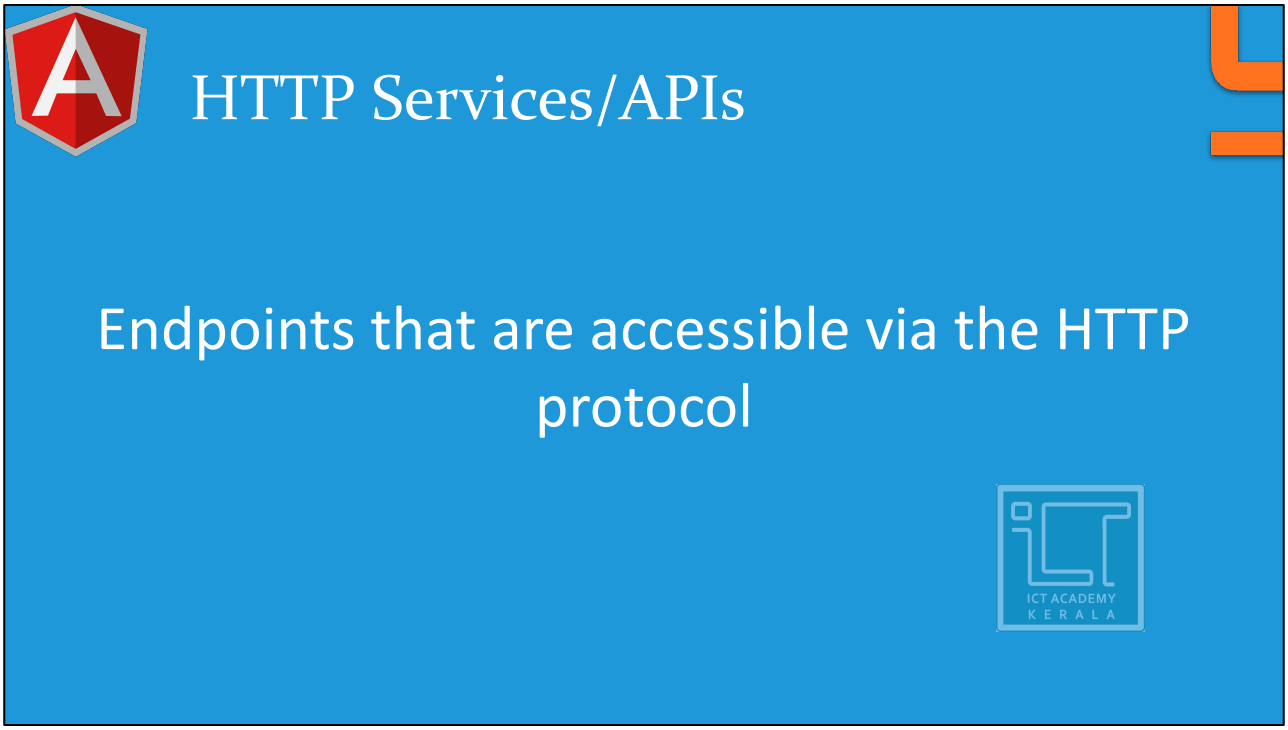
Easy to understand and write

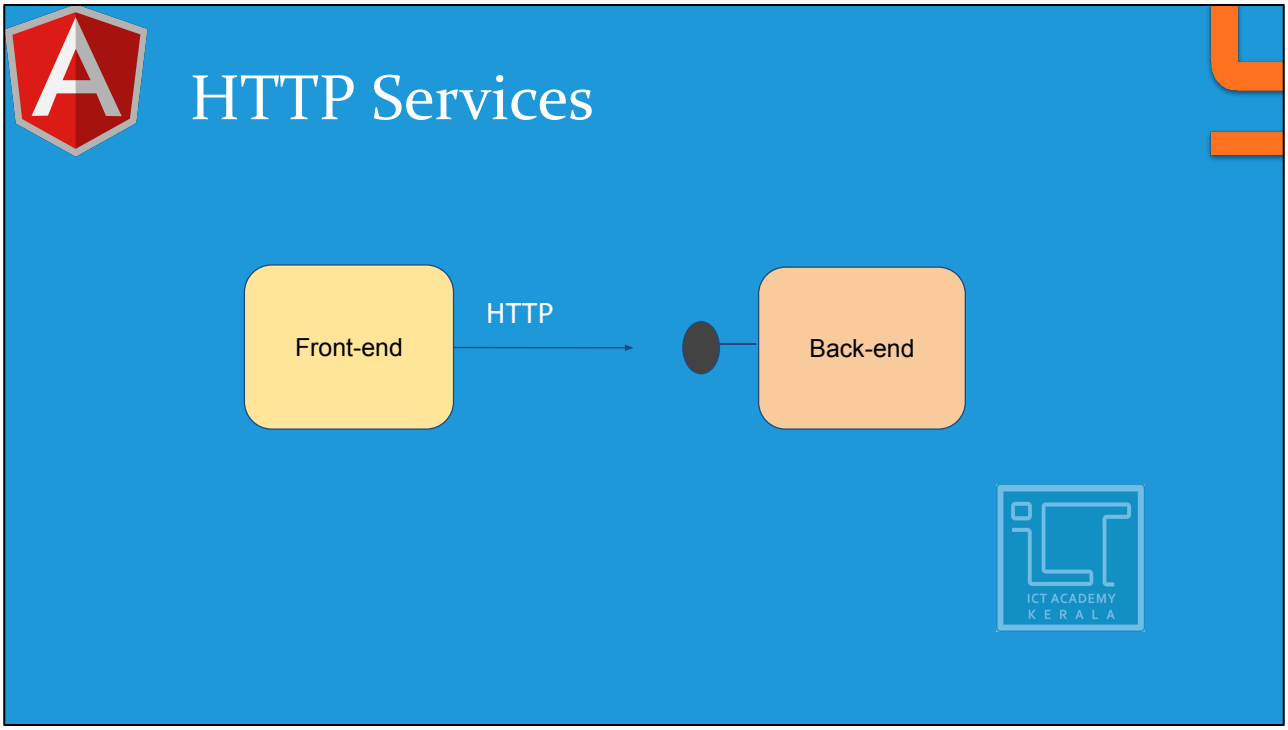

Angular makes your life easier!
ICT ACADEMY
KERALA



Front-end/client side : where it runs in web browser and user interacts with. It contains the UI which is coded in HTML/CSS,TypeScript and Angular.

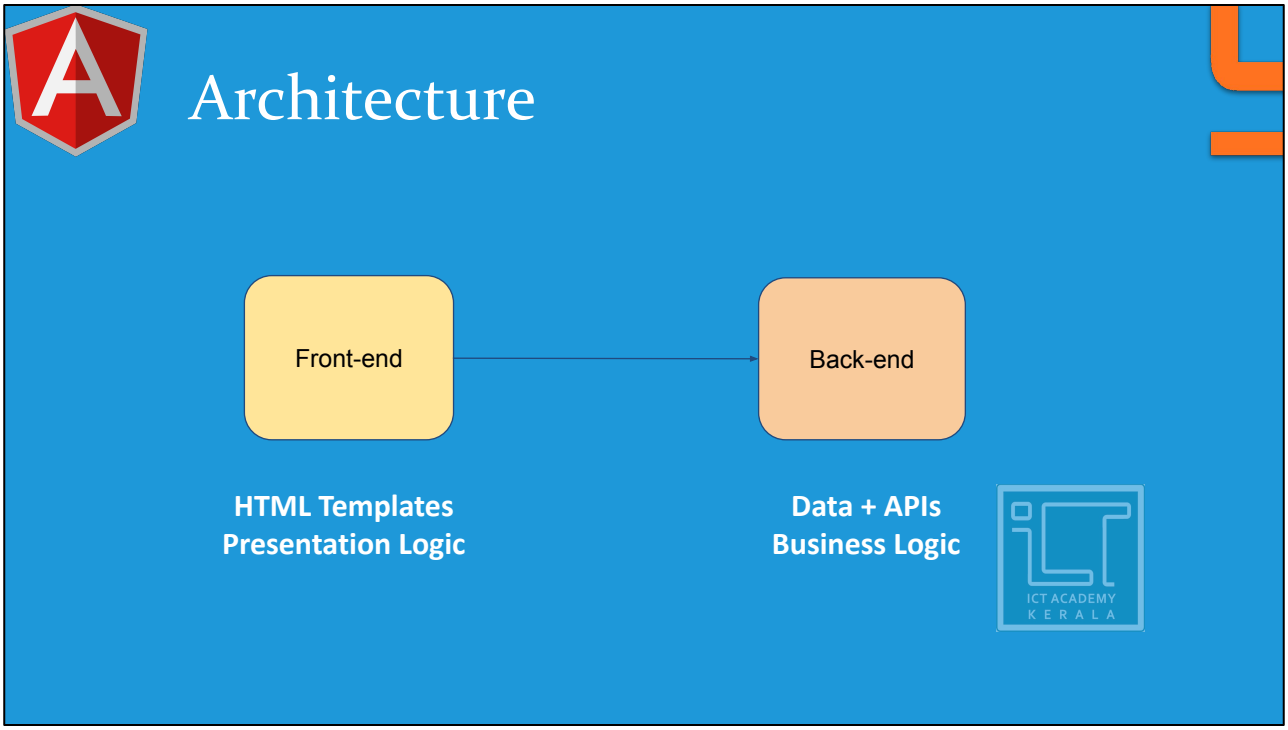
Back-end : sits on a web server/multiple web server in cloud. Where data received from client is stored and processed.
*We do not store the data in the client/angular, because the data gets easily disappear when the user clears the browser history or moves to a diff computer.
So here we have one or more databases, bunch of HTTP services/API's to make this data available to the clients.

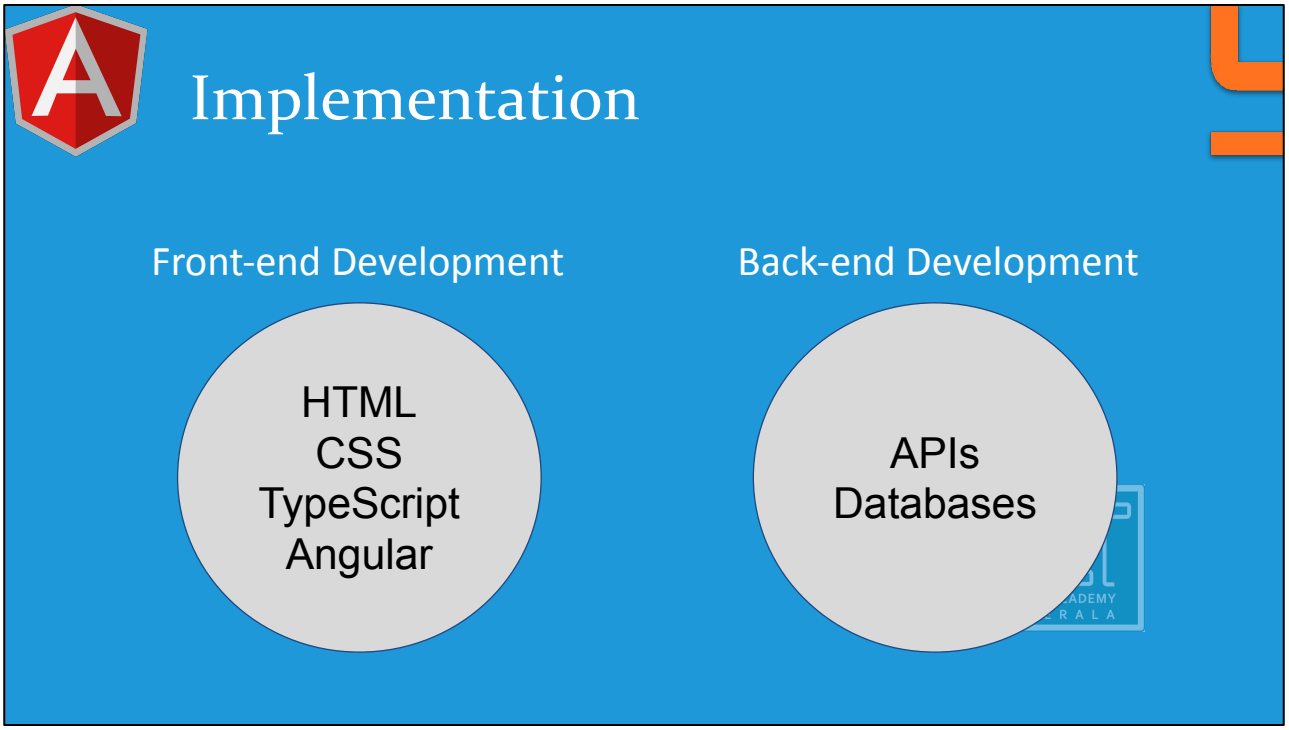

HTTP Services/APIs
Endpoints that are accessible via the HTTP protocol
ICT ACADEMY
KERALA

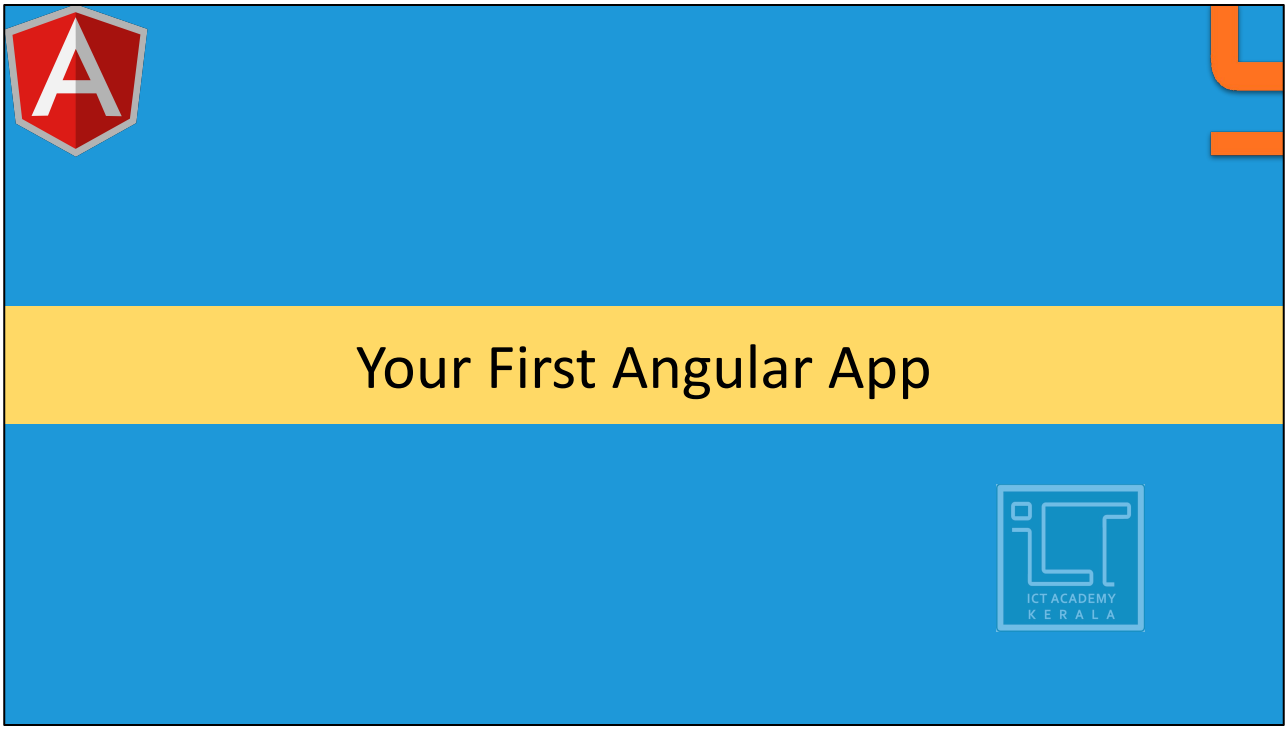


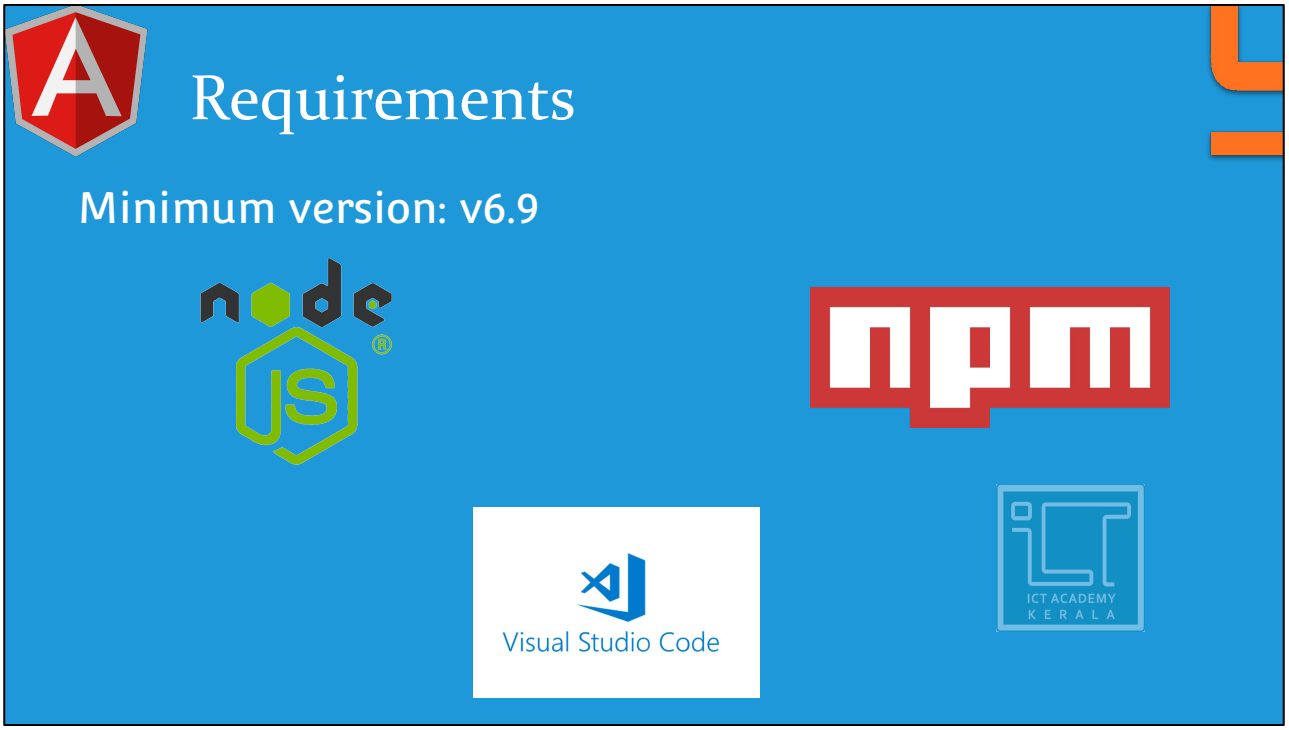
So we can call them using http req to get or to save the data.

Let's take a metaphor. Client send HTTP Req to Endpoint to get the data

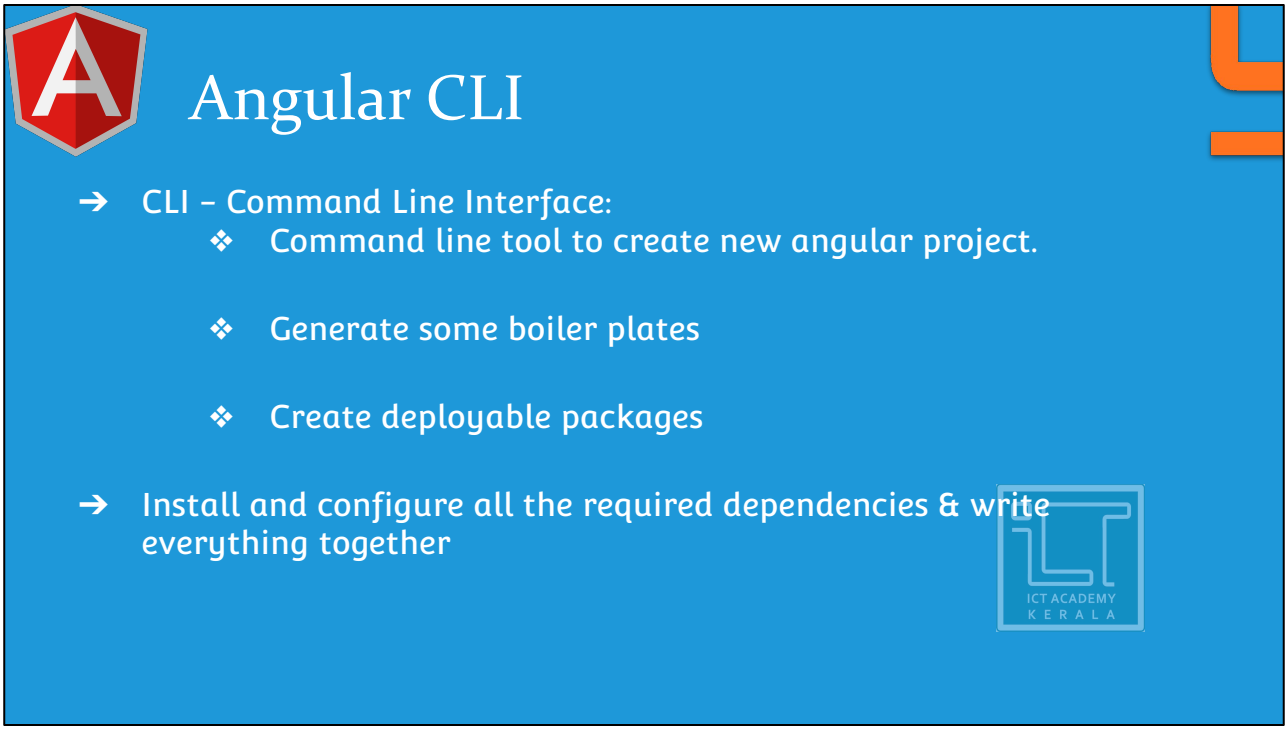

Architecture
Front-end
Back-end
HTML Templates
Presentation Logic
Data + APIs
Business Logic
ICT ACADEMY
KERALA


Implementation
Front-end Development
HTML
CSS
TypeScript
Angular
Back-end Development
APIs
Databases


Your First Angular App
ICT ACADEMY
KERALA

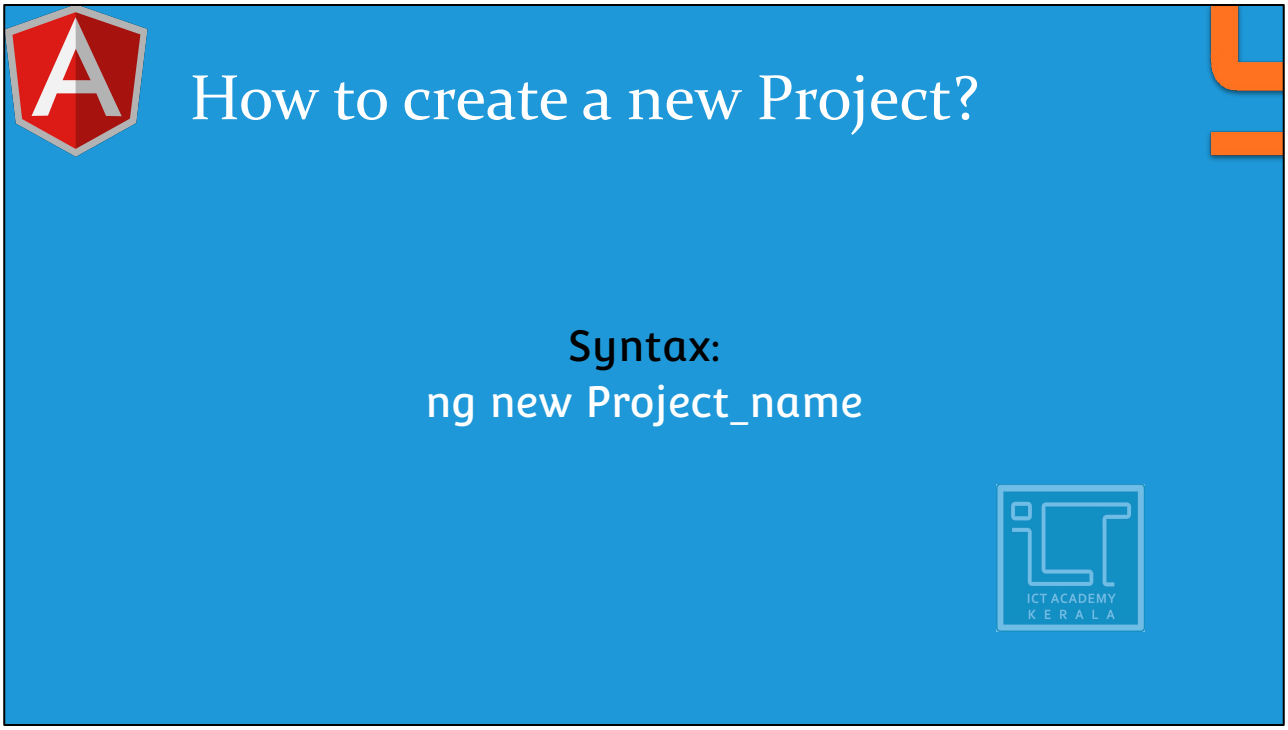


Node js: Runtime environment for executing Javascript code outside browser
Npm: node package manager is used to install external packages
Visual Studio Code: Code editor

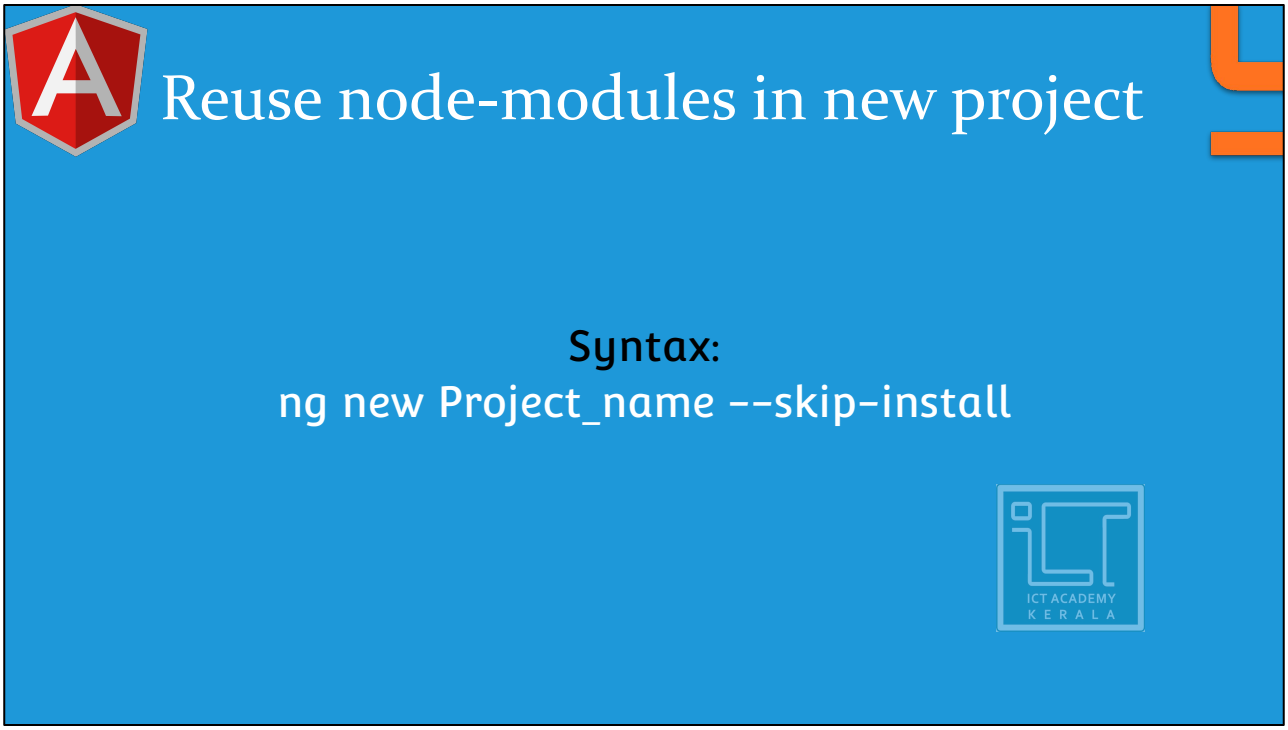


Angular cli: library we need to install. Command line tool used to create new angular project/generate some boiler plates.

This will install the latest version of Angular. ng stands for aNGular. **Important:** You need strong internet connection!!



Important: You need strong internet connection!! 1. Project_name should start with capital letter. 2. No space



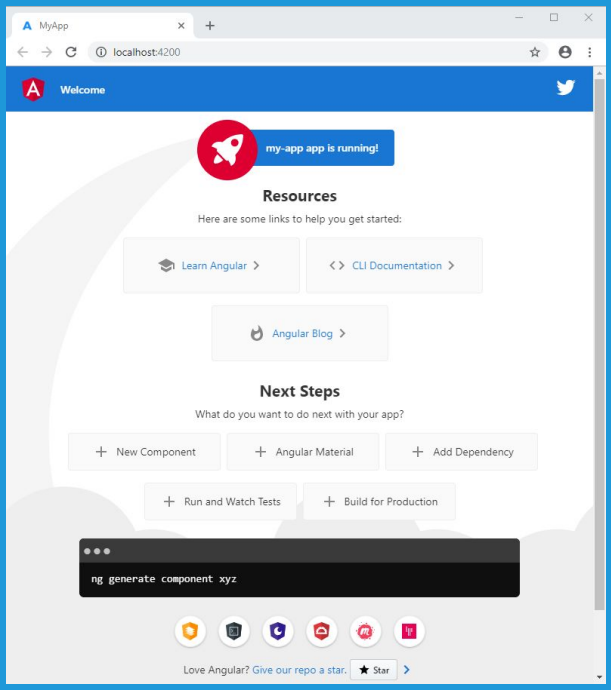
- Skip installing the node modules.
- Copy paste the node modules from the previous projects.
- Reduce the time of installation

How to run a project?

- Connecting to server: ng serve
- For the first time it will ask whether we need to send the data to google(y/n) : n
- After successful compilation:
  - Web browser – localhost:4200
- 4200 : dedicated port for angular

By default the angular project will run in the port : 4200

Default Template



How to change the port number?

Syntax:

```
ng serve --port 5000
```

Structure of Angular Project

1. e2e :
   - end to end
   - write end to end text for applications
   - Automated text that simulate a real user

Example of e2e: launch our browser, navigate to the homepage or application, clicking links, fill out forms etc., then there is something on the page

Structure of Angular Project

2. Node modules:

- We store the third party libraries
- Dependencies for angular project

Structure of Angular Project

3. src : actual source code of our application

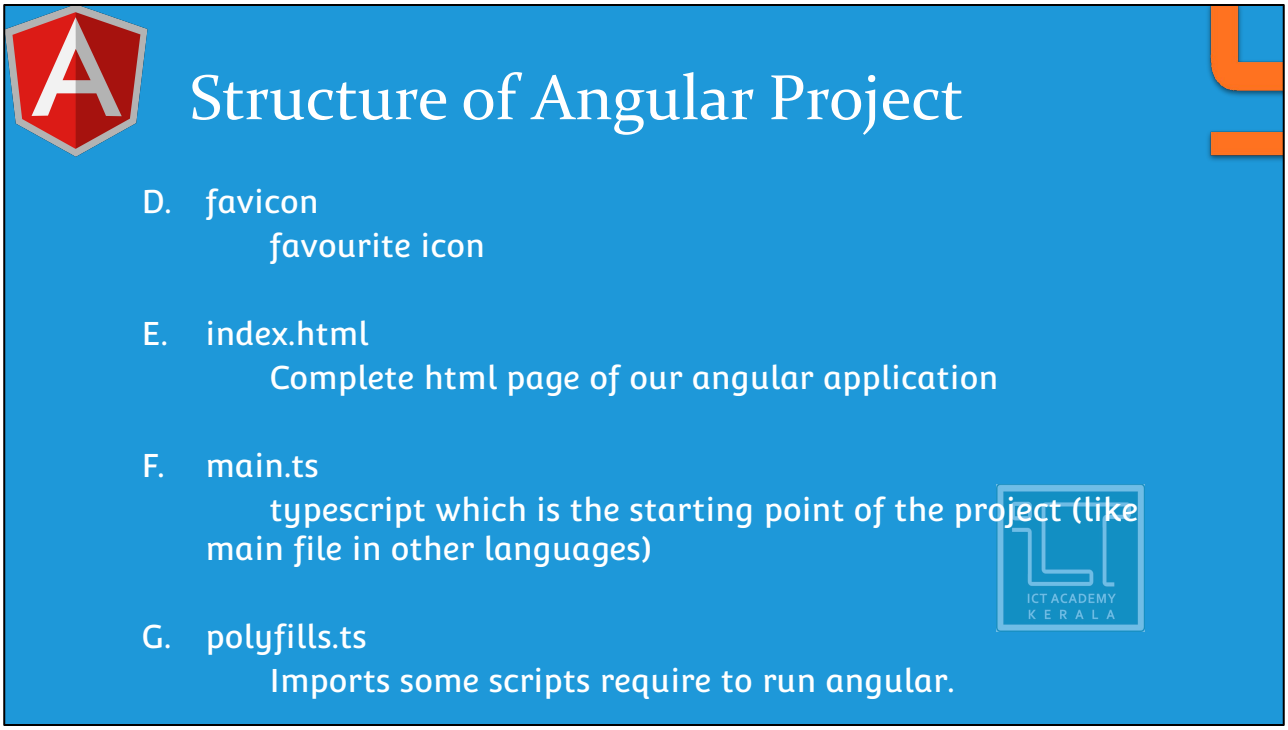
A. app
- Module,component and routing

B. assets
- Store the static assets of our project

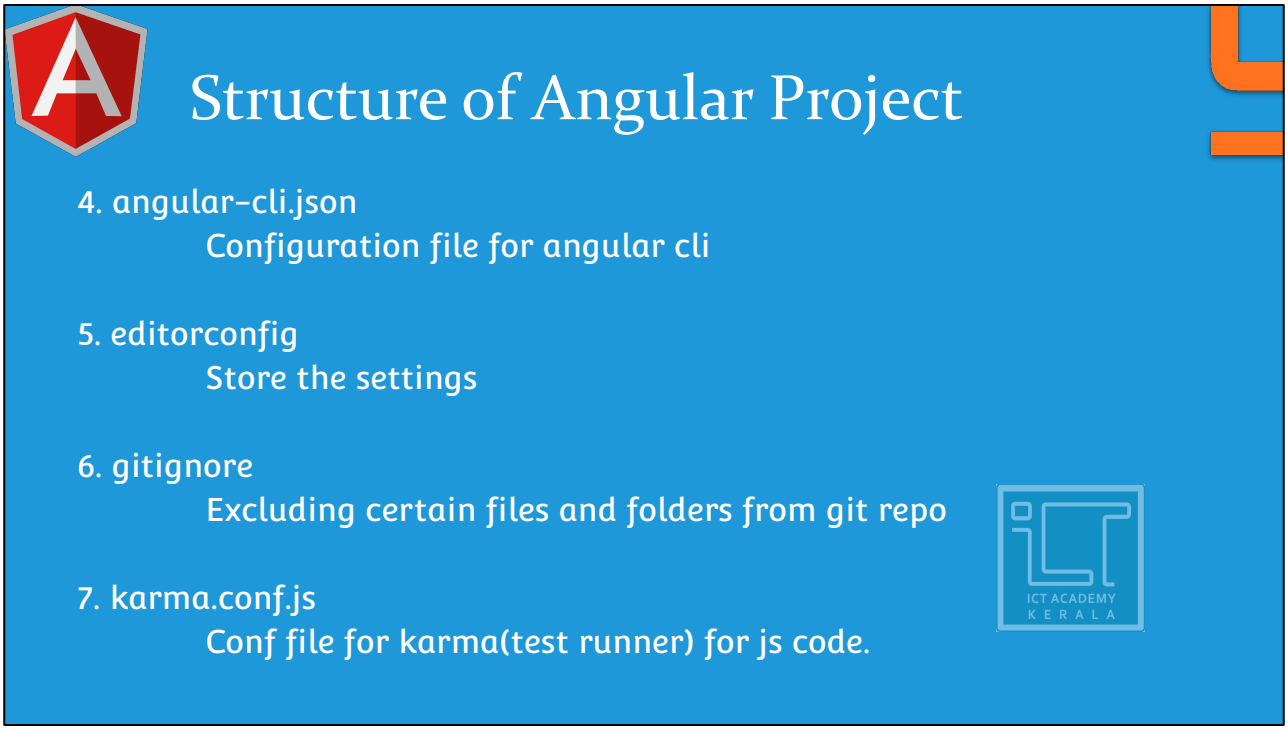
C. environment
- Store configuration settings
  - production
  - development

App: Every application has at least 1 module and 1 component.

Assets: Image file, text file, audio file, video file



Polyfills: angular framework uses some features of javascript which is not available in current version of js not supported by many browsers. Fills the gap between features of js



Editorconfig: Make sure all developers use the same settings in their editors while working as a team

Structure of Angular Project

8. package.json

Metadata of all the libraries and related files that your application is dependent upon

9. protractor.conf.js

Tool for running end-end test for angular

10. tsconfig.json

Settings file for typescript compiler

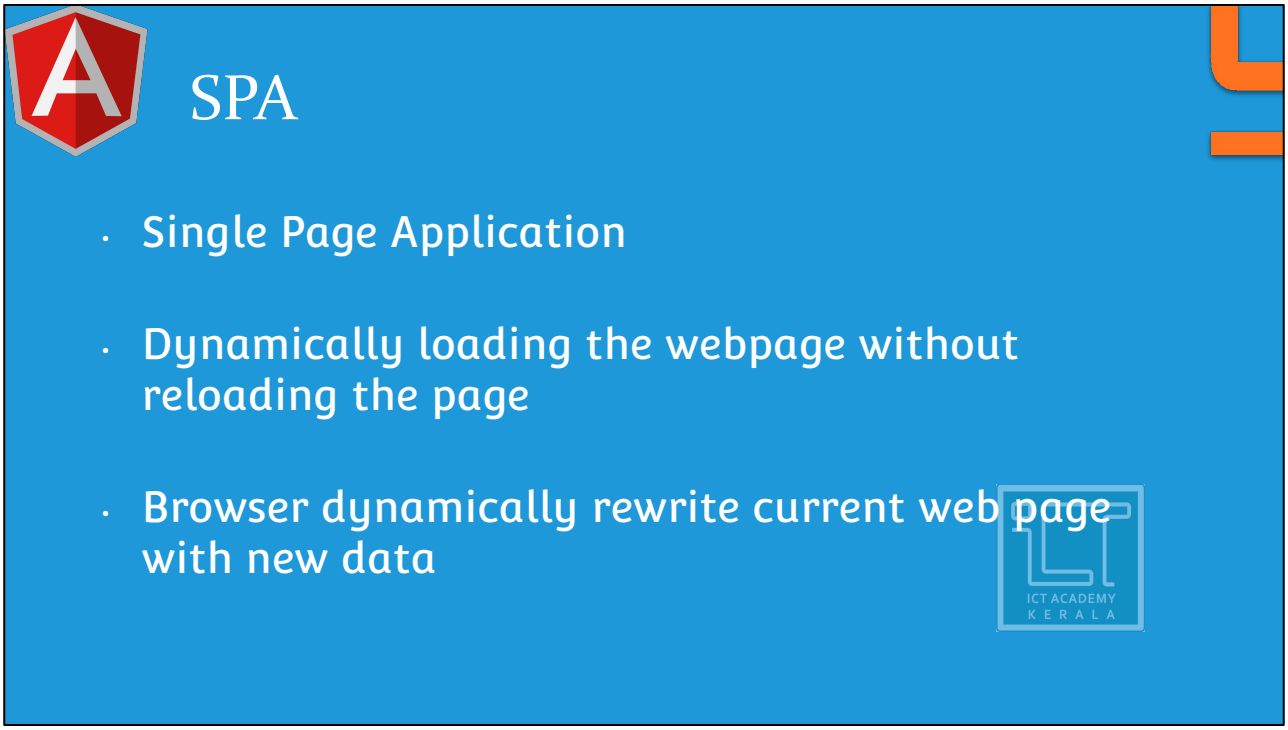
11. tslint.json

Settings for tslint

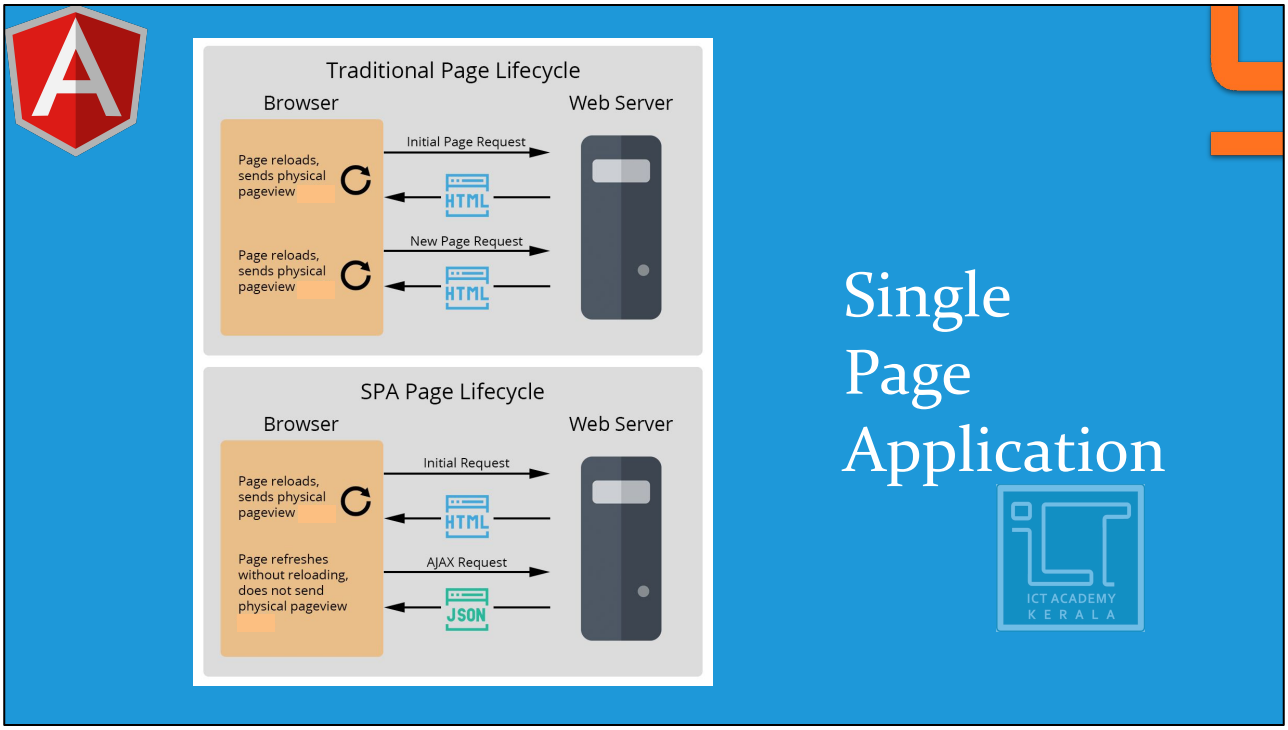
Tslint: static analysis tool for typescript code. Checks your typescript code for readability, maintainability and functionality errors.

Real Applications

1. Clarity.design



Instead of loading entire new pages, browsers dynamically rewrite the current web page with new data

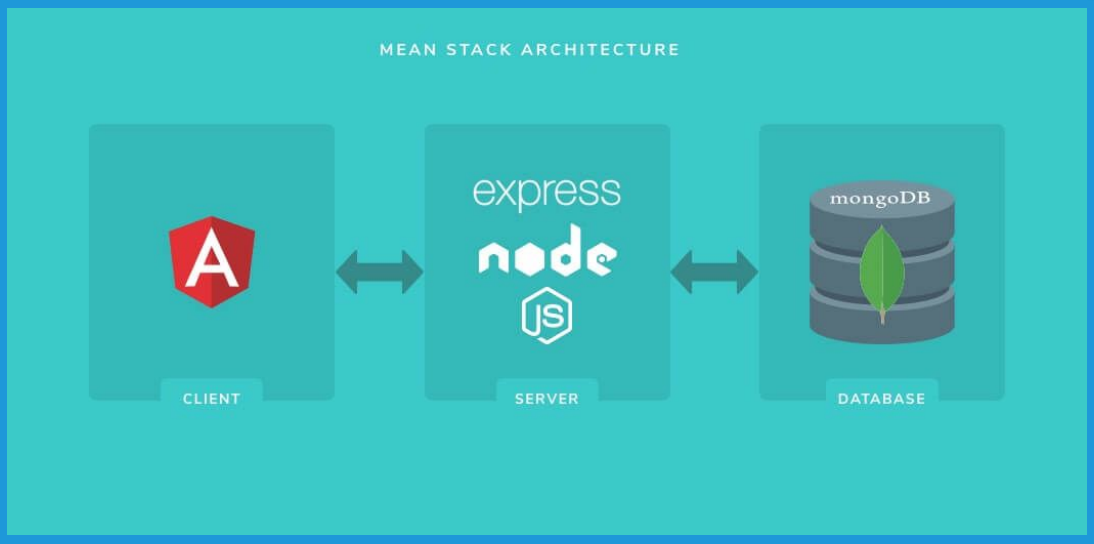


Single Page Application

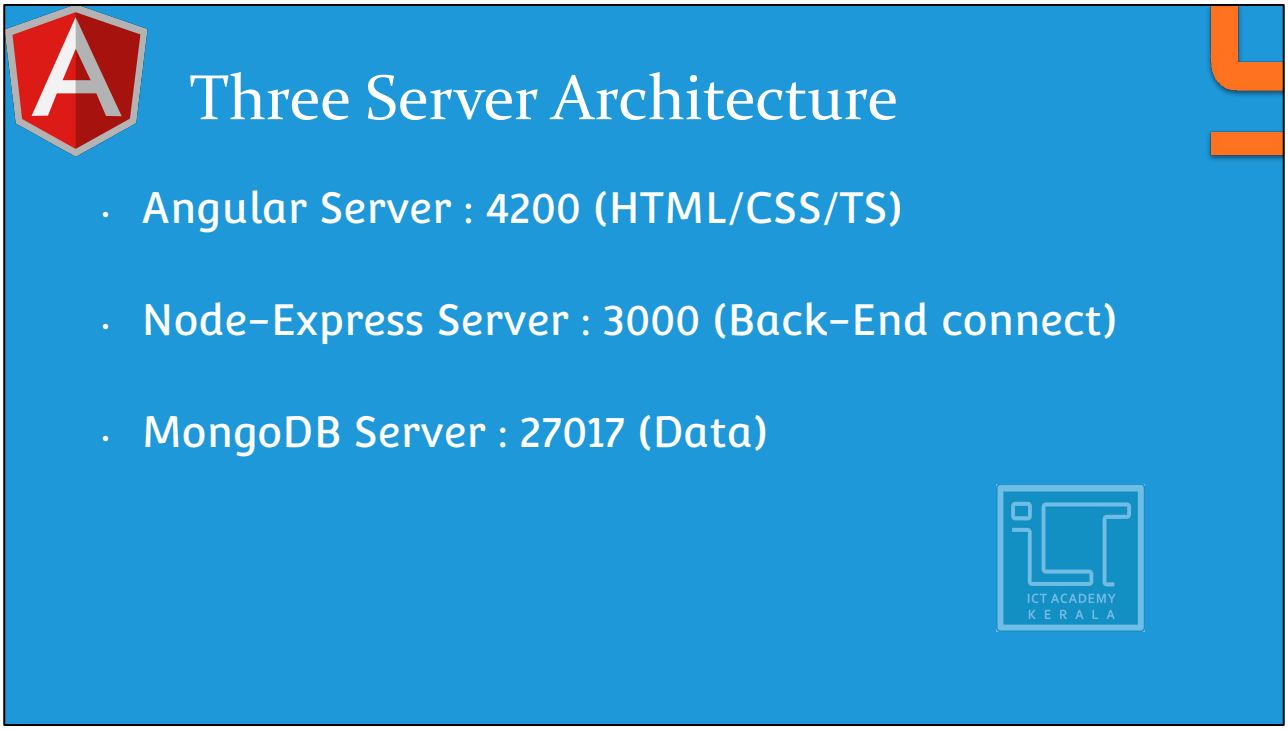
Advantages of SPA

- Improves user experience
- Less amount of data transferred through network

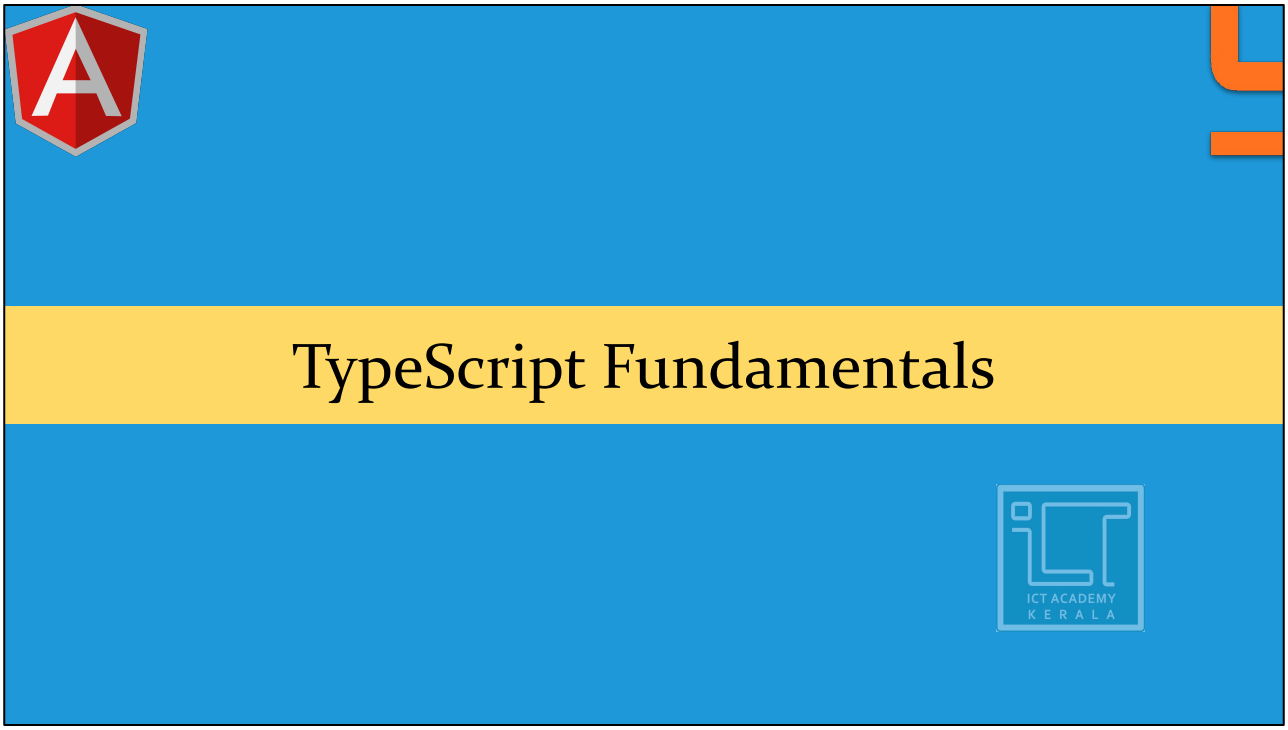
MEAN Stack Architecture

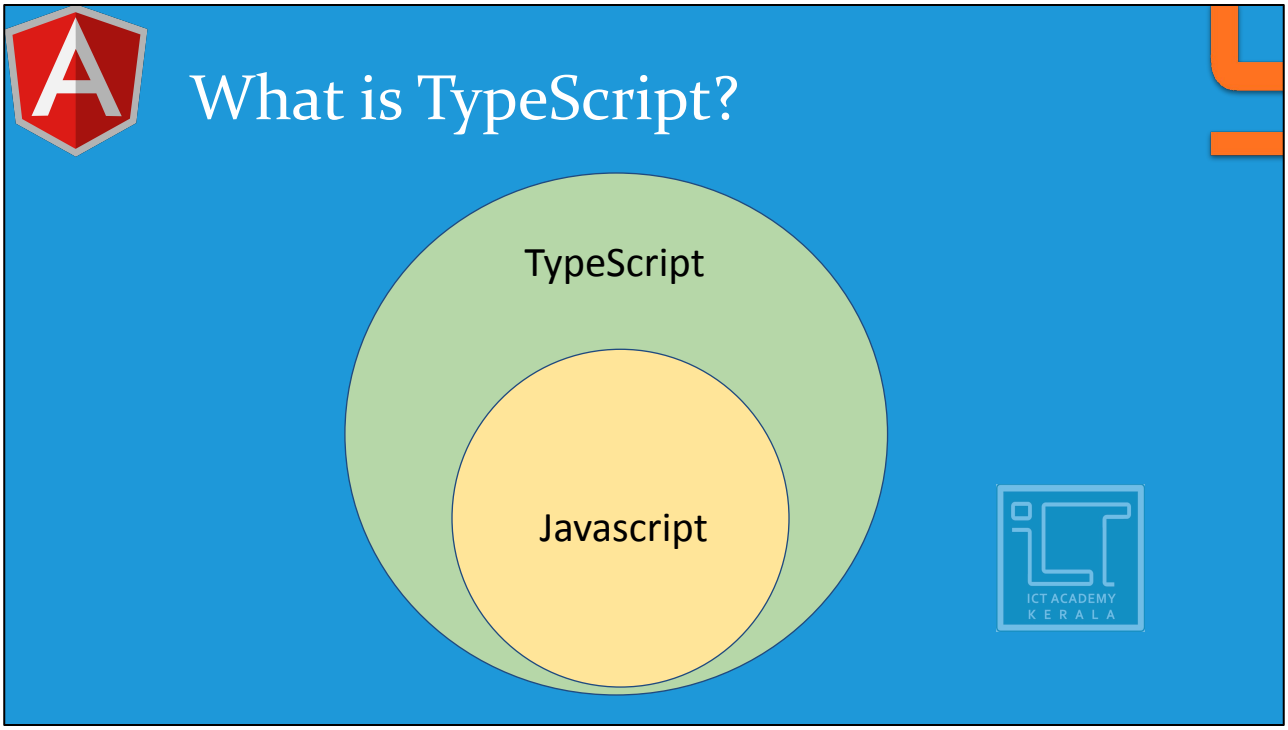


1. Browser - Angular - Node,express - MongoDB : Request
2. MongoDB - Node, Express - Angular - Browser : Response

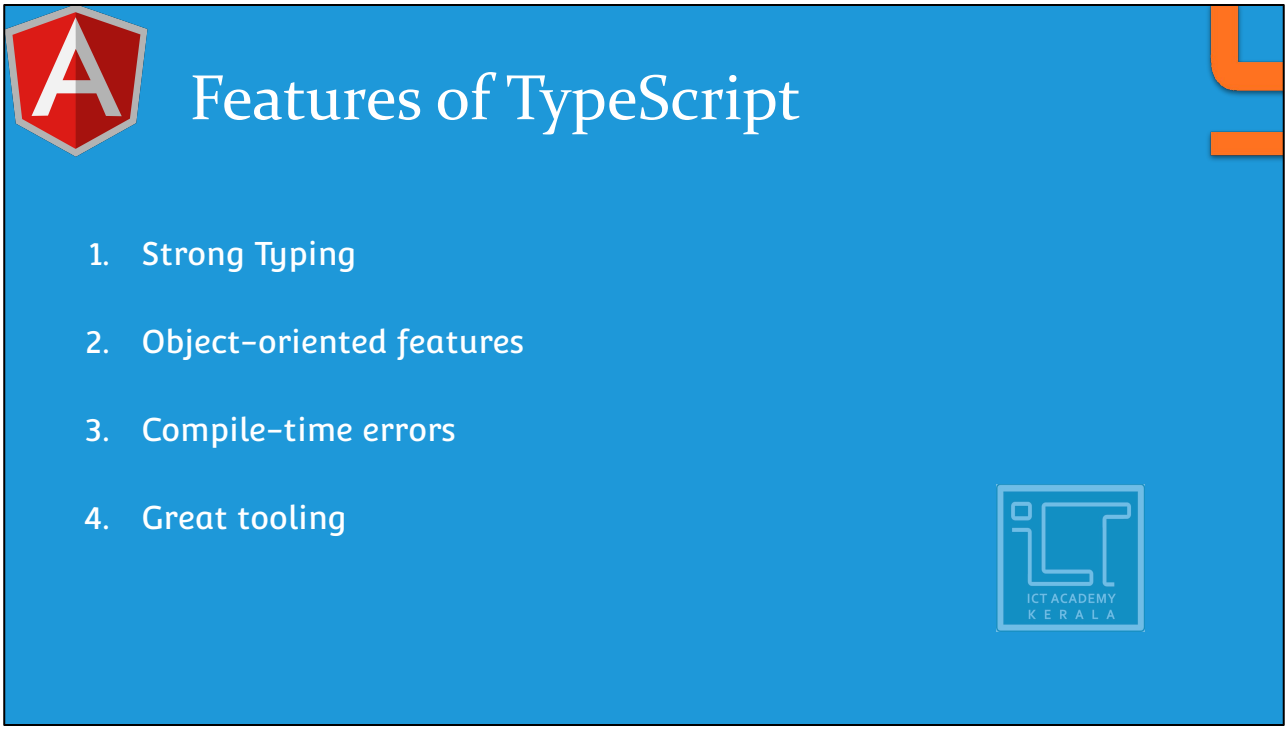


Angular is more dynamic framework. Eg: clarity.design - SPA example without reloading

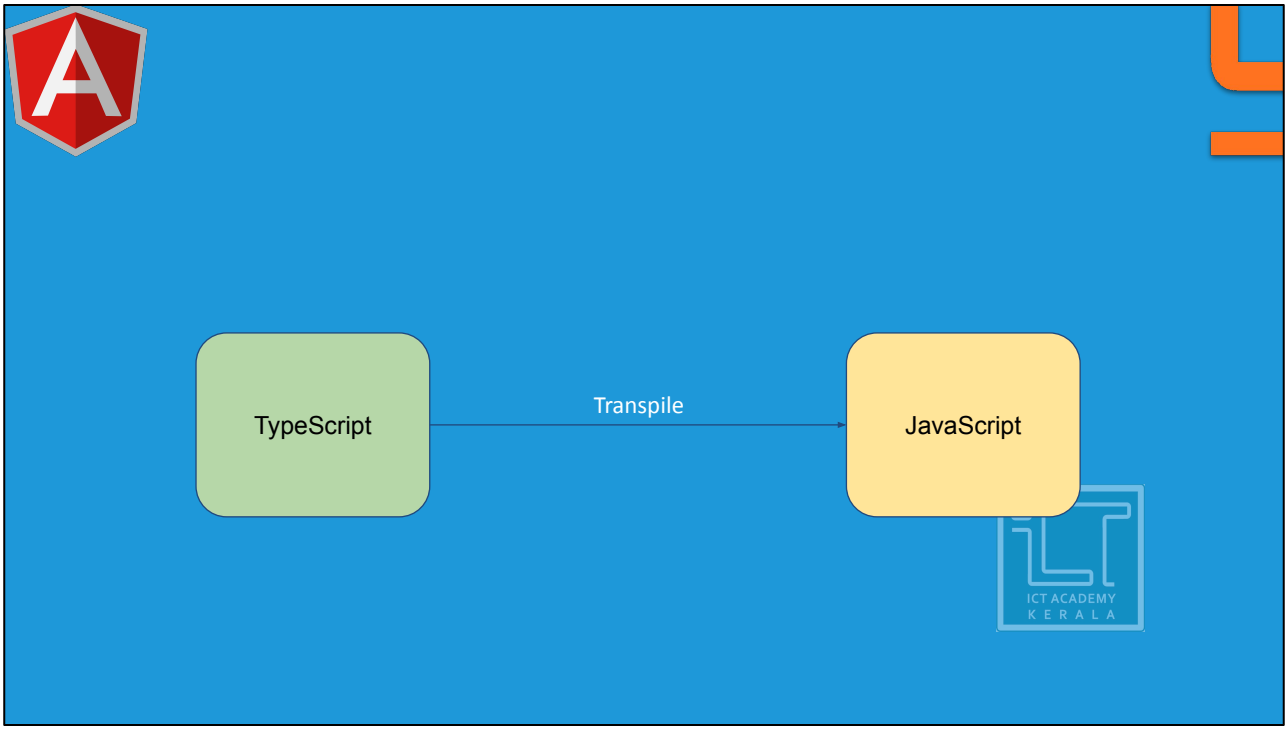

TypeScript Fundamentals
ICT ACADEMY
KERALA

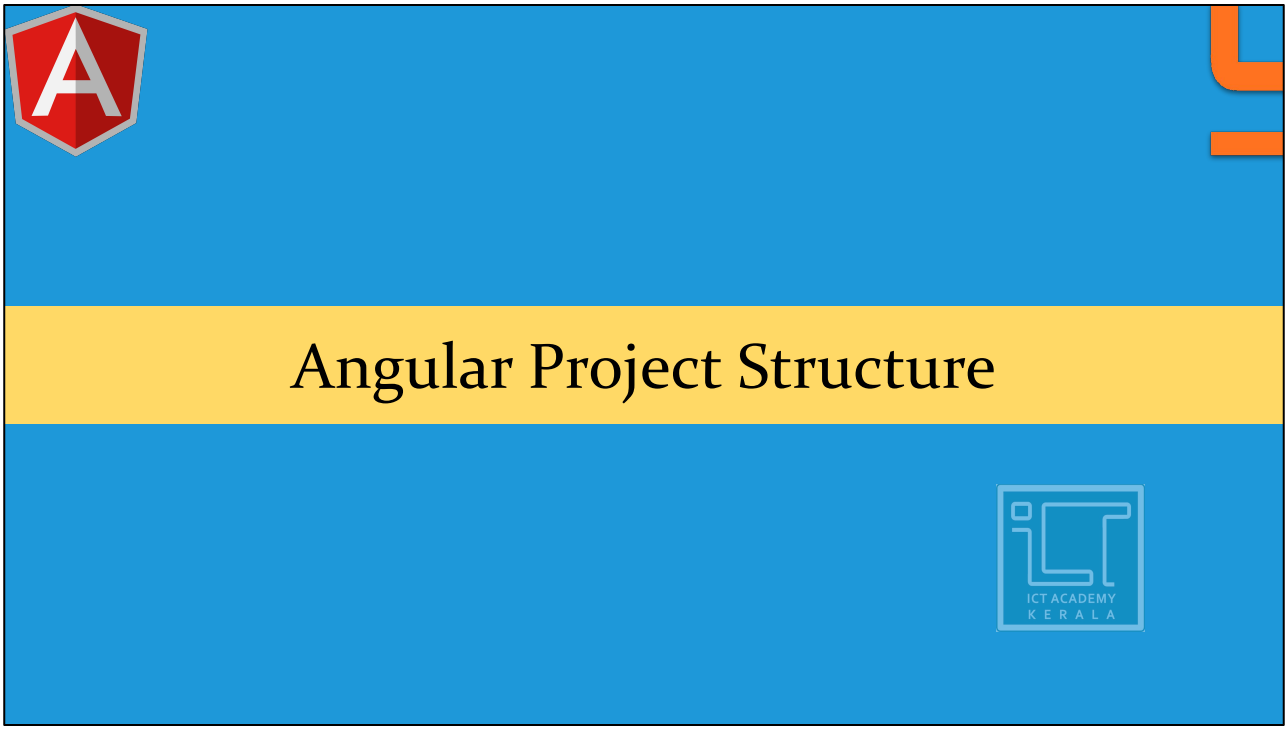


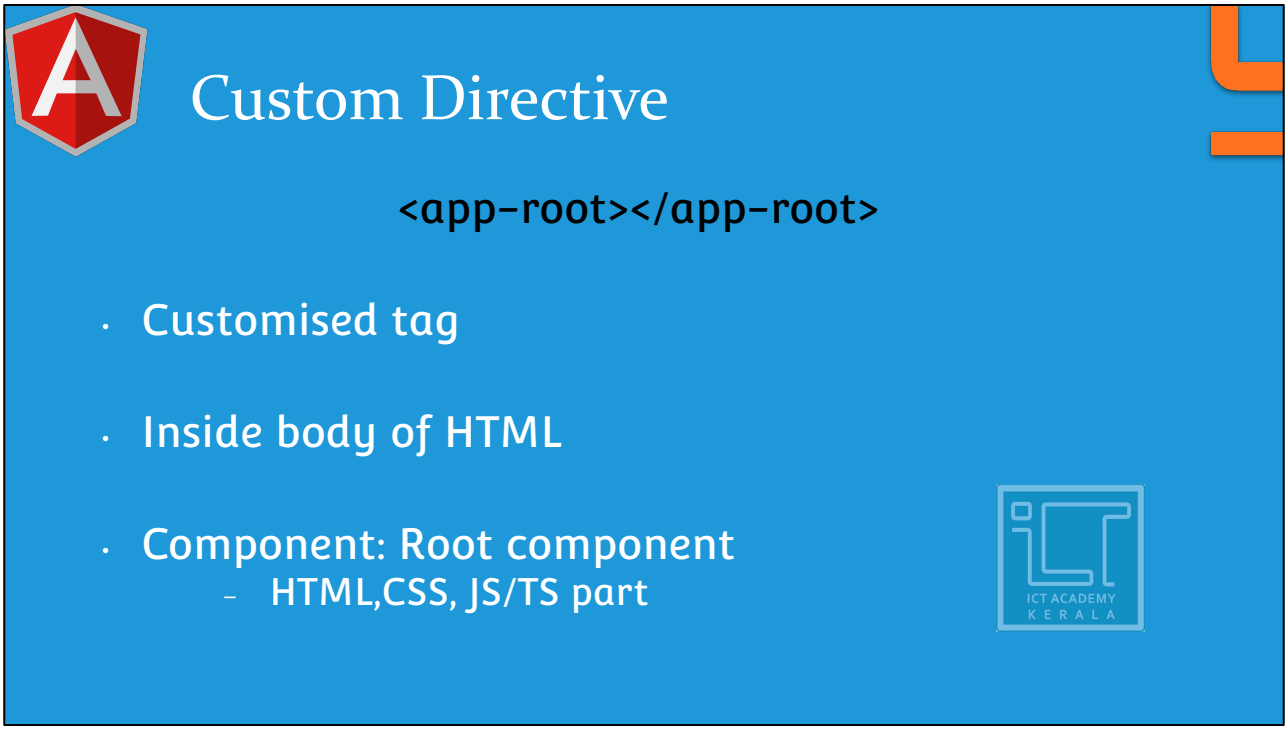
Typescript is the superset of javascript: Any valid JS code is also valid TS code. But TS has additional features that do not exist in current version of JS.



1. Specifying type of a variable is not necessary.
2. Classes,interfaces, constructors, access modifiers: public/private, generic types : used in TS(missed out in JS)
3. Catch errors at compile time instead @ run time(not all errors but most of them)
4. Intellisense


TypeScript
Transpile
JavaScript
ICT ACADEMY
KERALA


Angular Project Structure
ICT ACADEMY
KERALA



When **selector** of ts file is used as a tag in html file we call it as a **custom directive**

App/Root Component

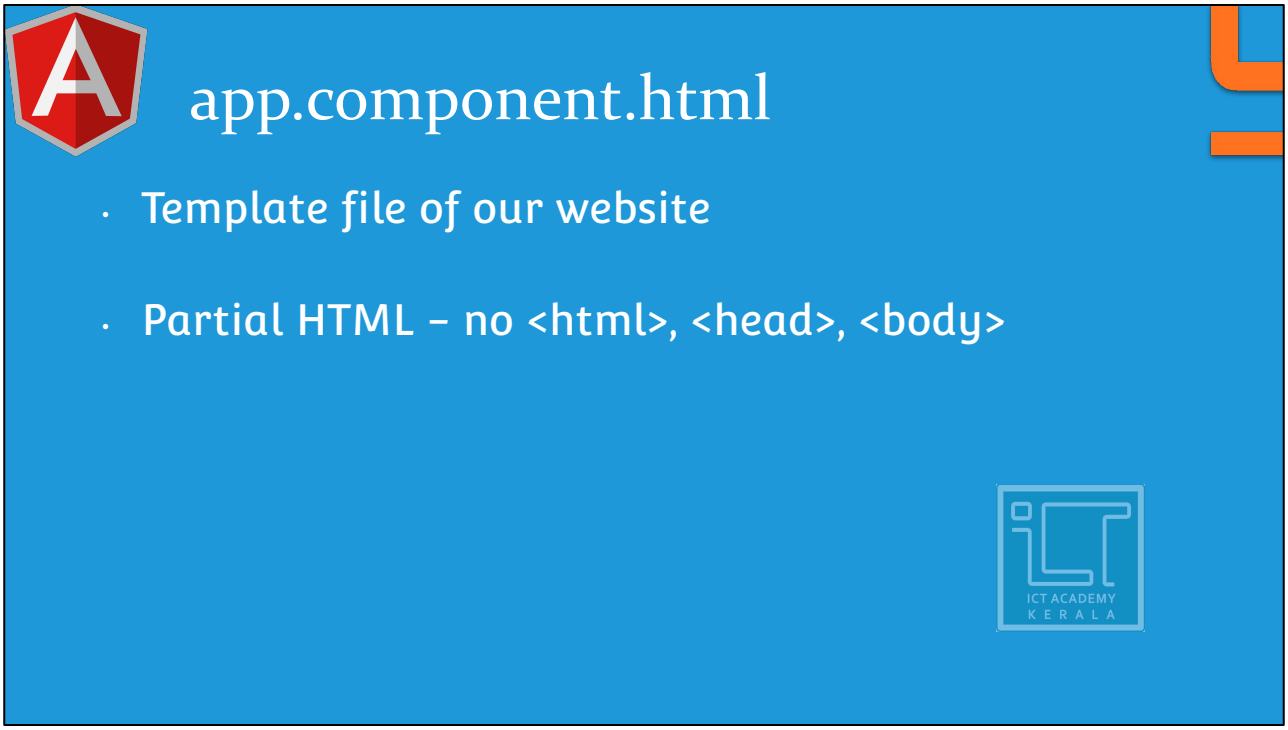
- app.component.css – styles
- app.component.html – structure/template/view
- app.component.ts – logic

app.component.ts

- class AppComponent : every component will have a TS file within which it should contain a class which can be exported. Also contain a variable : title

- @Component: Component Decorative, specify the details about the component
  - selector : name of the component
  - templateUrl: HTML for the component (path)
  - styleUrls: CSS for the component (path)

styleUrls : [‘./style.css’, ‘./index.css’] - can add multiple css separated using , hence represented as an array.



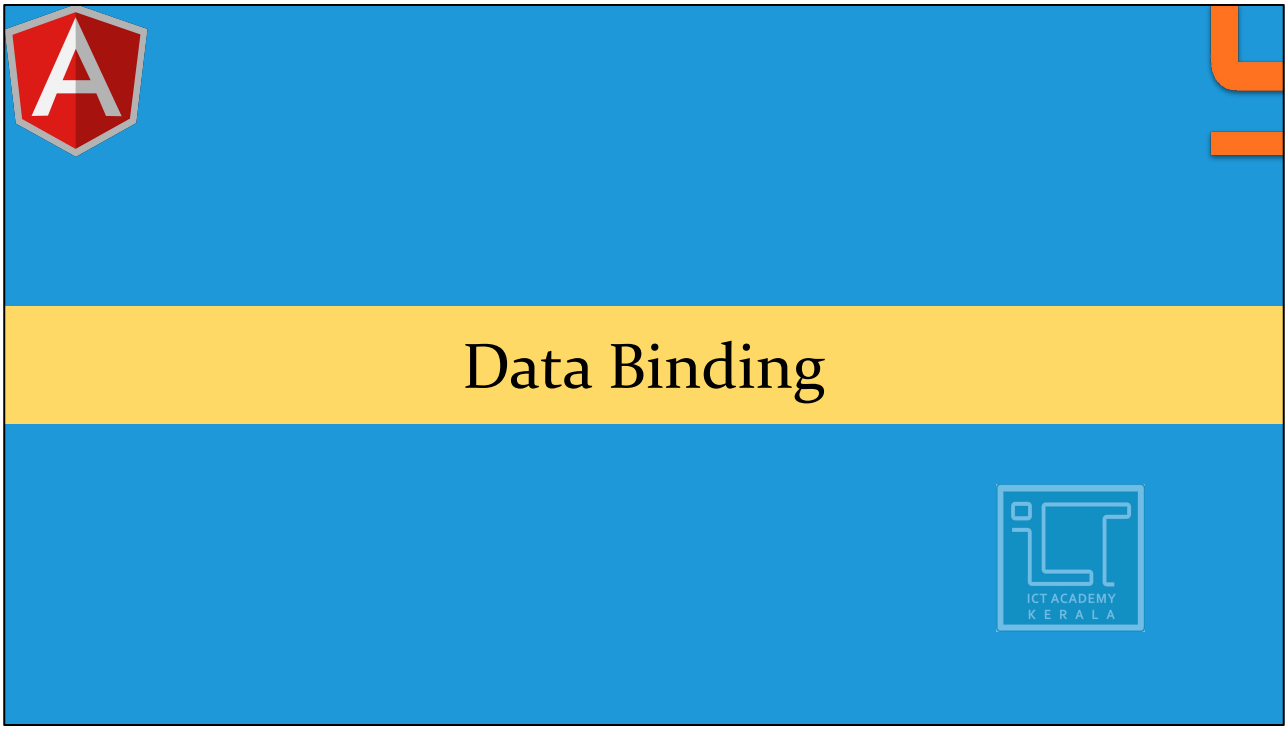
Index.html is the complete html file which consist of the html, head and body tag.

Added Advantage of Angular

When you change the code and save server will automatically restart and will be reflected in the browser

styles.css

- Common styles to all the webpages can be written inside the styles.css file.
- Global styles


Data Binding
ICT ACADEMY
KERALA

Data Binding

Any data which is written inside the class of the component TS file can be accessed inside the component HTML file and vice-versa.

Data – variable, function, arrays, etc.

Interpolation { {} }

For applying the data binding we use the format called Interpolation { { } }

One-Way Binding

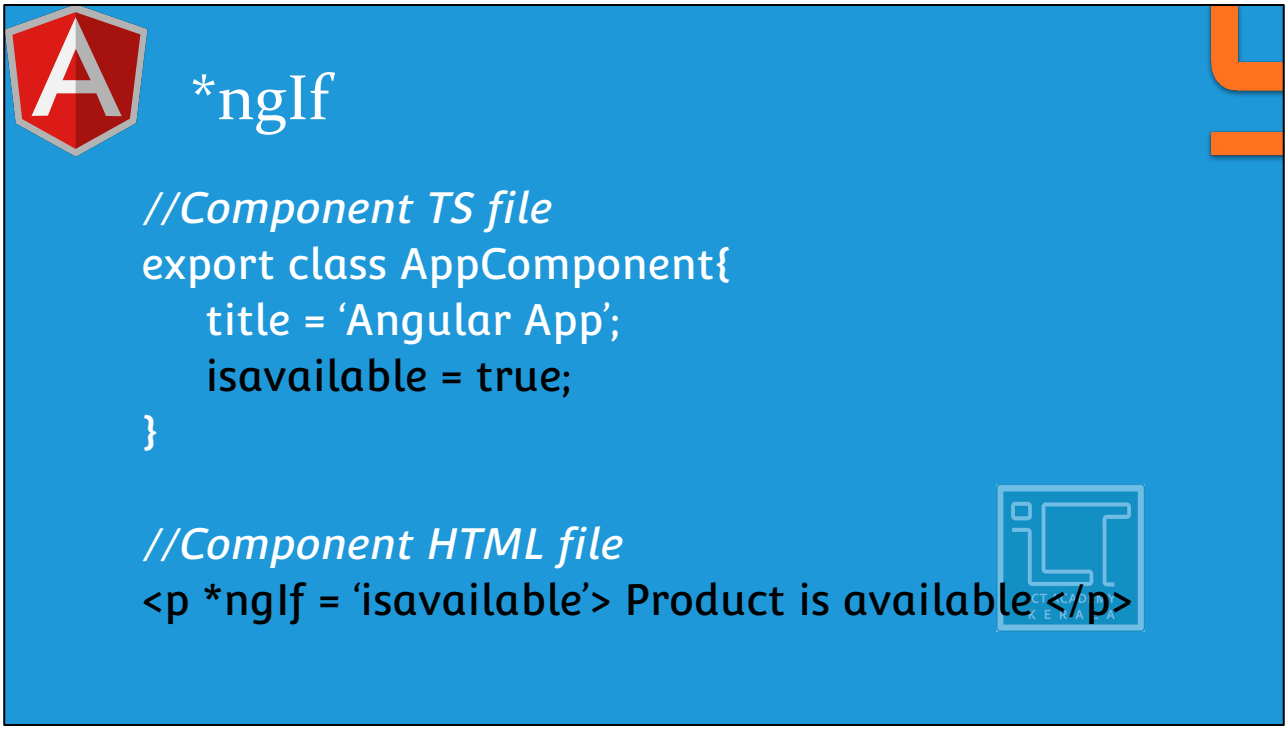
When we use the variable only from class to html

Built-in Directives

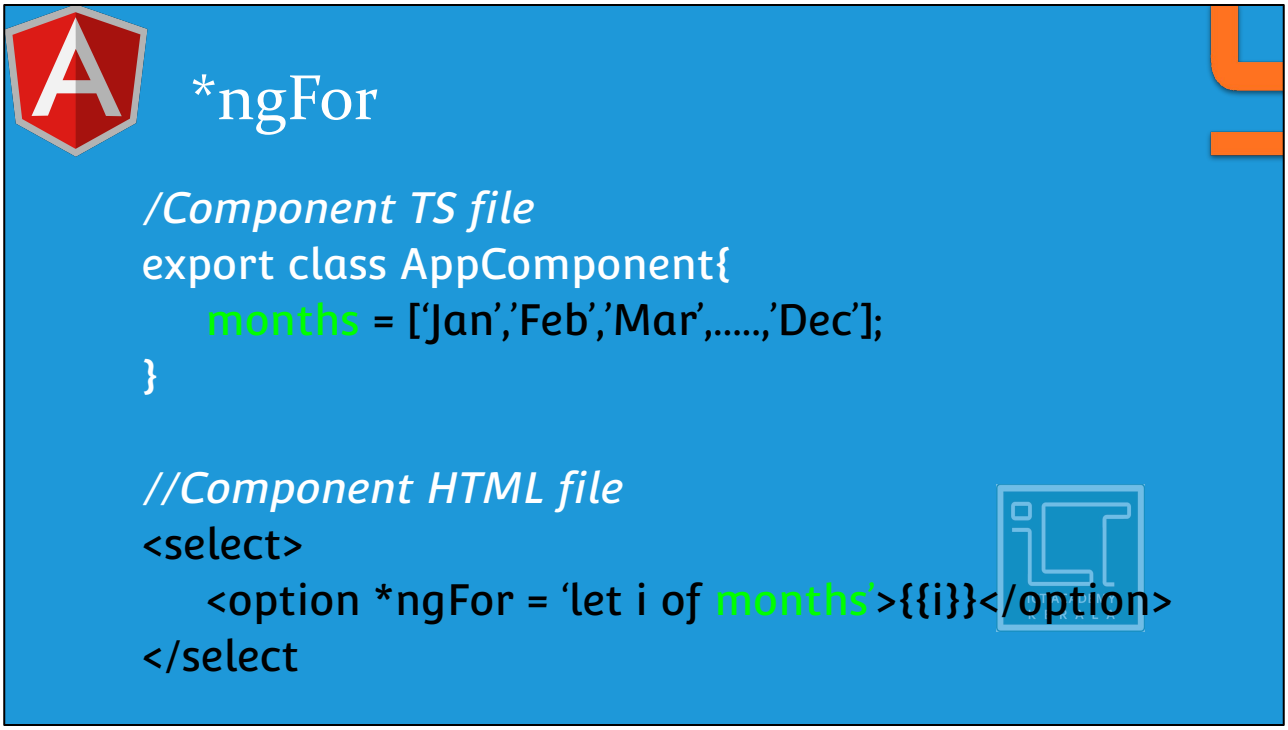
- Structural Directives
  - *ngIf – if Logic (based on condition)
  - *ngFor – for logic (based on lopping principle)

  * indicates that it is a structural directive

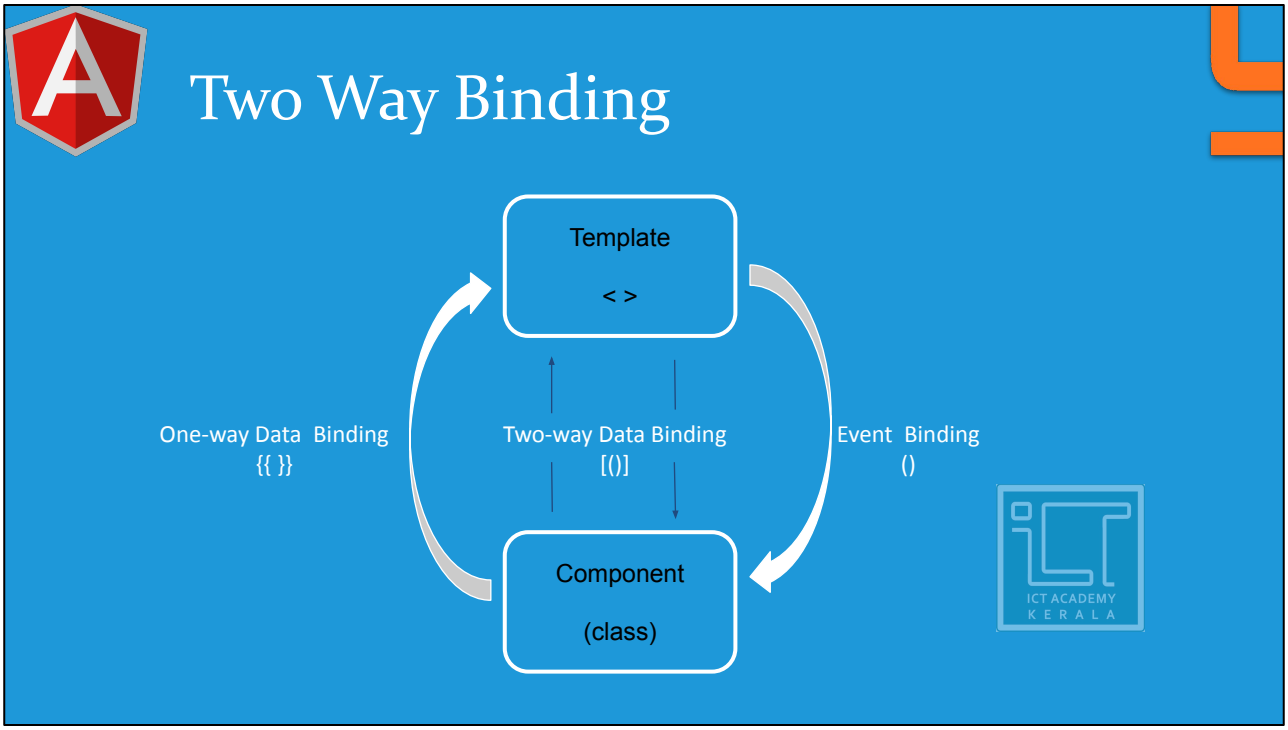
  We use *ngIf and *ngFor for structuring our HTML hence known as structural directive



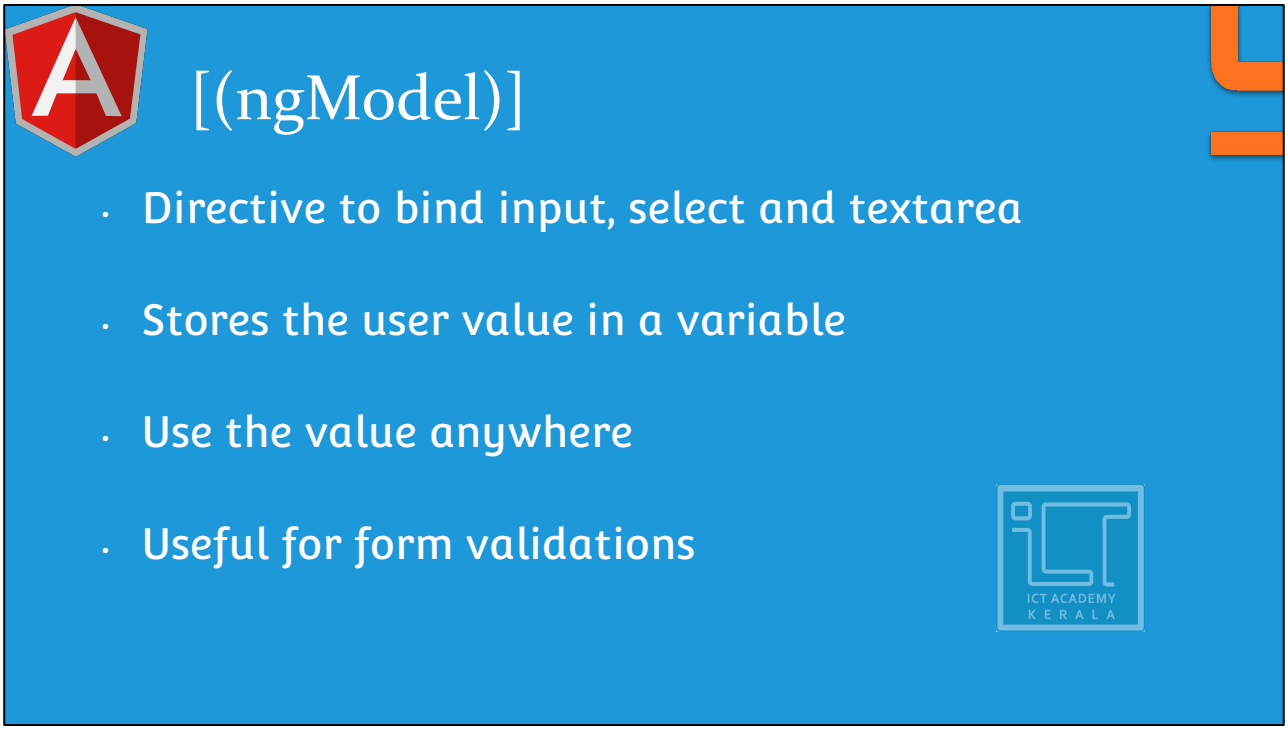
Only if 'isavailable = true' then display the contents in <p> tag. Else don't show. Try making the 'isavailable = false' and check.



Dropdown of months.



Two-way data **binding** refers to sharing data between a component class and its template. If you change data in one place, it will automatically reflate at the other end.



ngModel is a directive which binds input, select and textarea, and stores the required user value in a variable and we can use that variable whenever we require that value. It also is used during validations in a form.

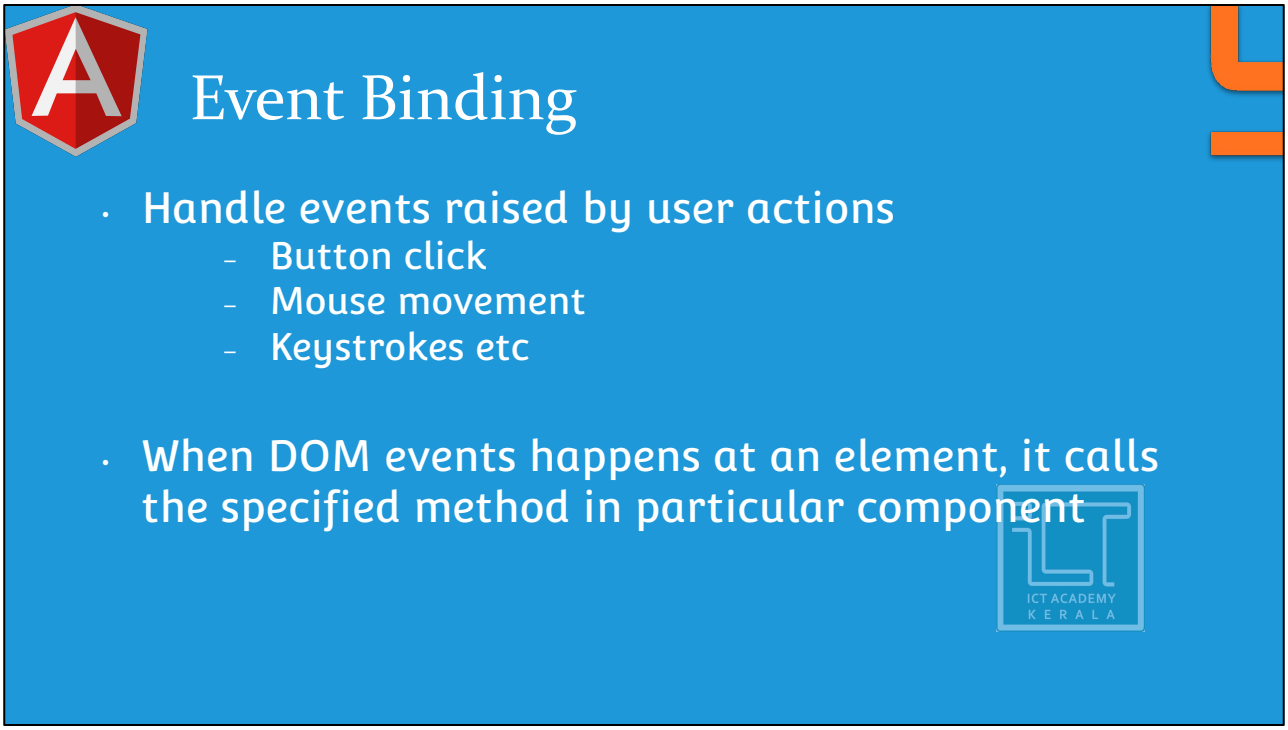
app.module.ts

- settings file
- Include the below lines in this file for using ngModel in our project.

```
import { FormsModule } from '@angular/forms';

imports: [
FormsModule
]
```

Two-way Binding Example

```
//Component TS file
export class AppComponent{
    name = 'Meera'; //create a variable
}

//Component HTML file
<h3>My name is {{name}}</h3> //one-way binding
<input type="text" [(ngModel)]="name"> //two-way
```



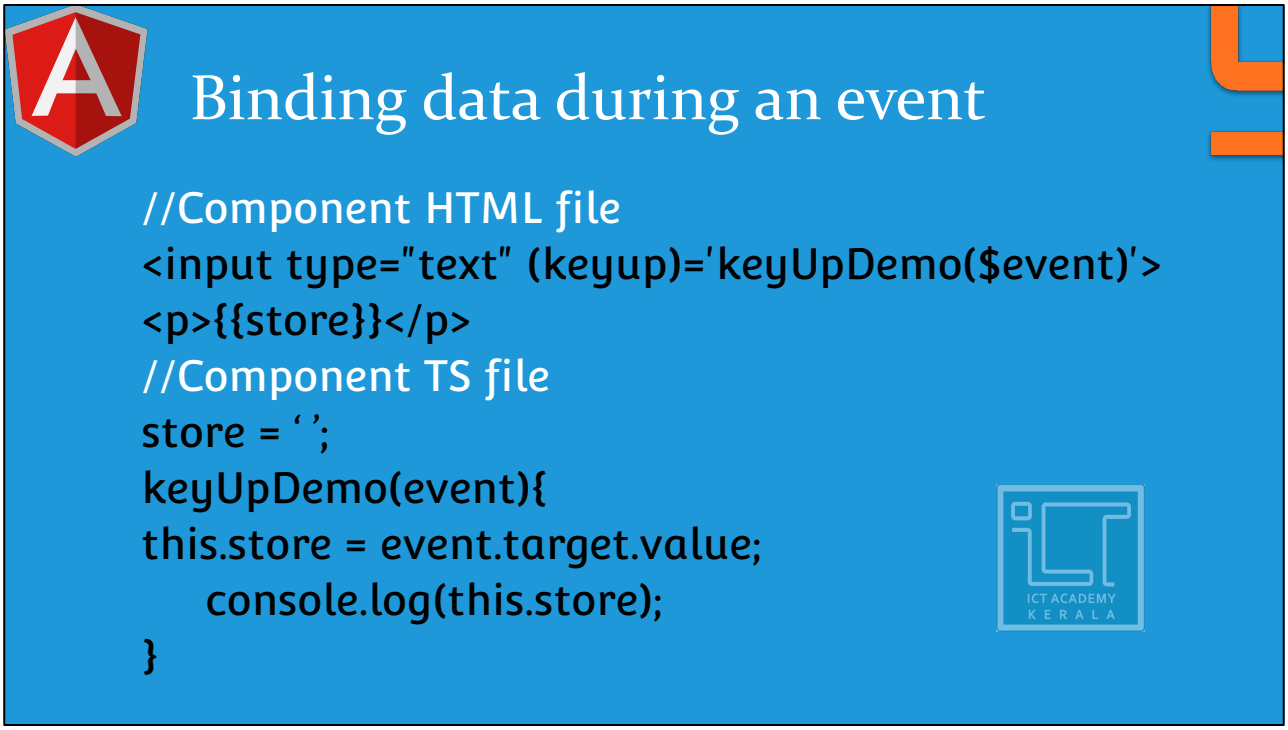
In **Angular** 8, **event binding** is used to handle the **events** raised by the user actions like button click, mouse movement, keystrokes, etc. When the DOM **event** happens at an element(e.g. click, keydown, keyup), it calls the specified method in the particular component.

Event Binding Example #1

```
//Component HTML file
<button (click)=clickDemo()>Click Me</button>

//Component TS file
clickDemo(){
    alert('Hey you clicked me');
}
```

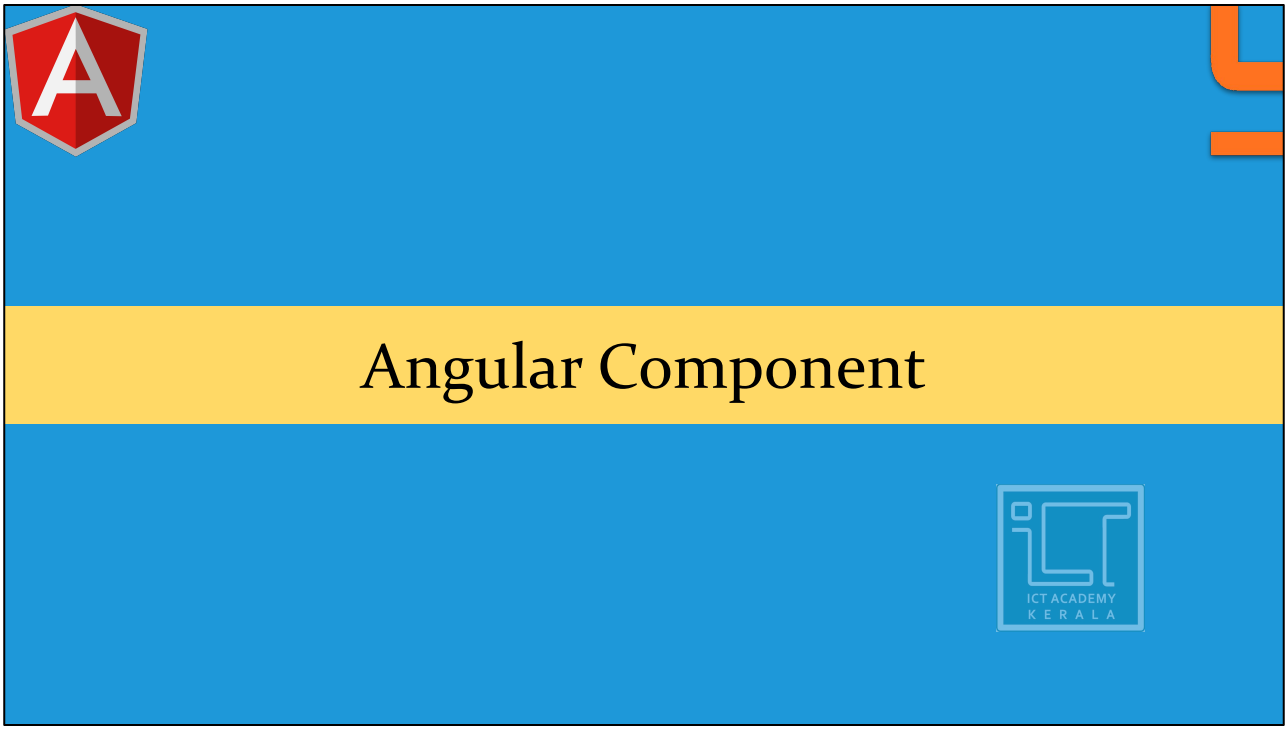
Event Binding Example #2

```
//Component HTML file
<input type="text" (keyup)=keyUpDemo()>

//Component TS file
keyUpDemo(){
    console.log('Key up occurred');
}
```



When the user input data needs to be sent from html to the class, we need another method.

1. Where we capture the user inputs via an argument inside the function handler(keyUpDemo) called **$event**. **event** is a keyword. - HTML
2. keyUpDemo needs to accept the argument as parameter - TS
3. Values entered into the input box needs to accessed and stores in a variable using **this.store = event.target.value; -** TS
4. This data can be printed in html page by using interpolation ** {{store}} -** HTML


Angular Component
ICT ACADEMY
KERALA

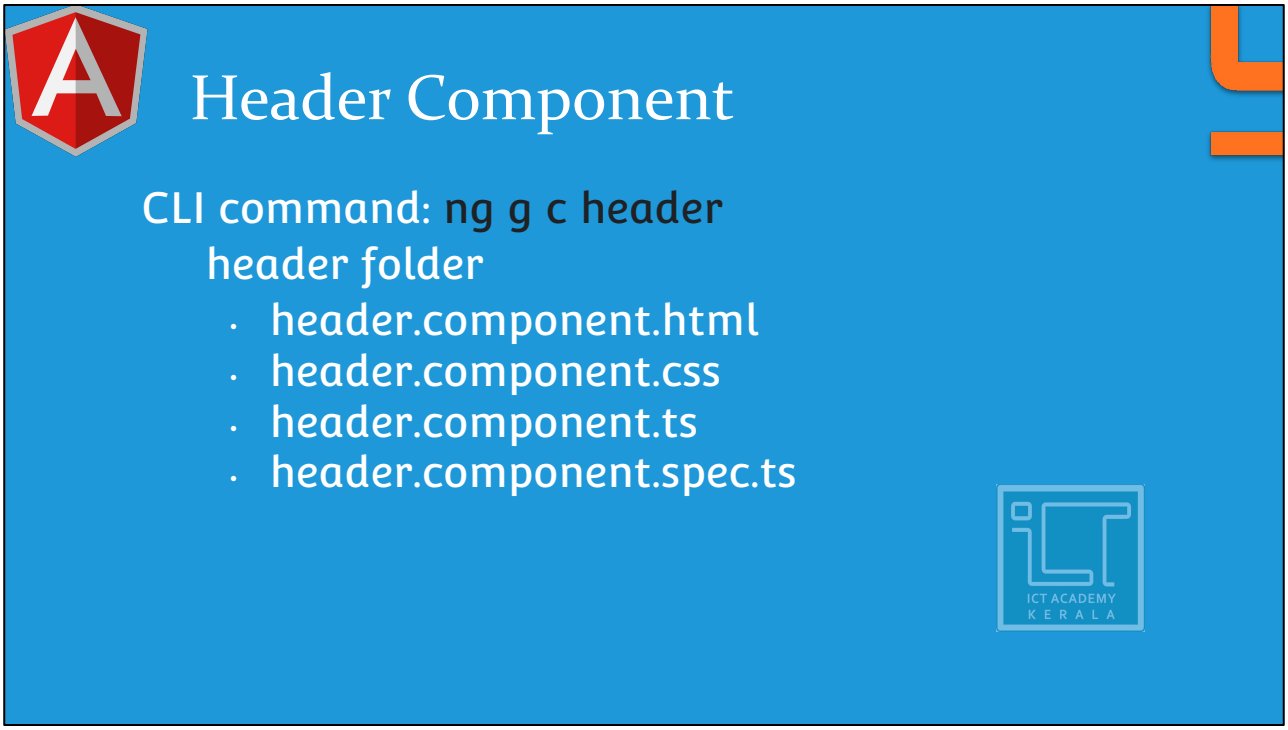
Create New Component

Angular CLI command:

ng generate component component_name

Shortcut:

ng g c component_name



Type the cli command in new terminal

header.component.html

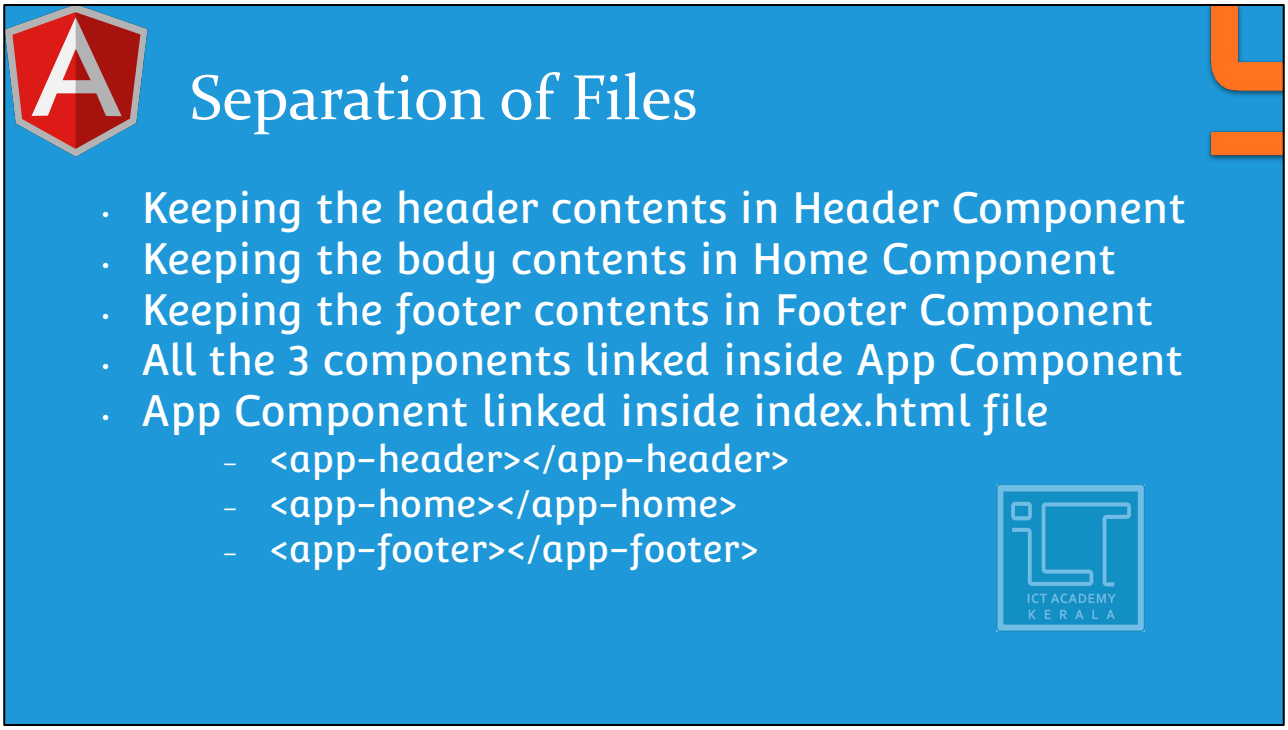
```
<div id="links">
    <a href="#">Home</a>
    <a href="#">About</a>
    <a href="#">Contact</a>
</div>
```

header.component.css

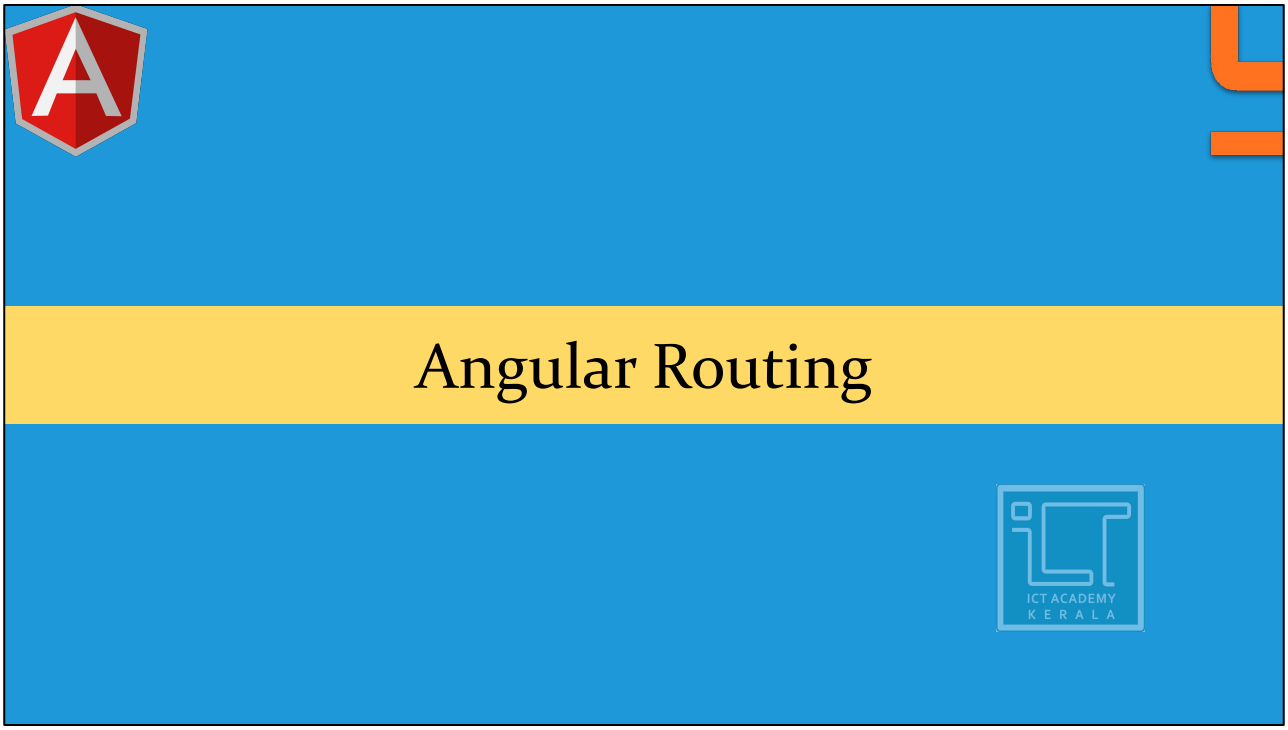
```
#links{
    background-color: orange;
    width: 100%;
}

a{
    color: white;
}
```

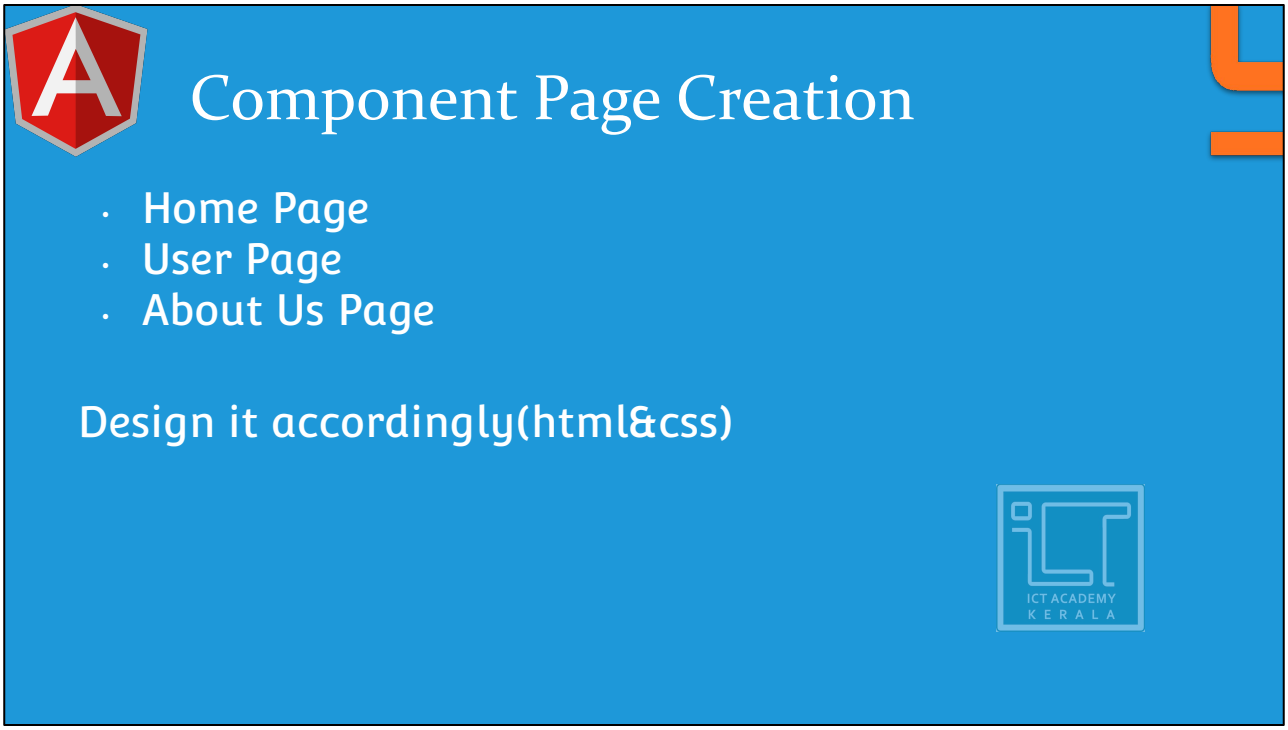
Linking

- Link the <app-header></app-header> in the top of app.component.html file.
- NOT in the index.html file



Cut all the html, css, ts contents from app.component and paste it inside the home.component


Angular Routing
ICT ACADEMY
KERALA



Component creation: ng g c user, ng g c about

Linking the path in header

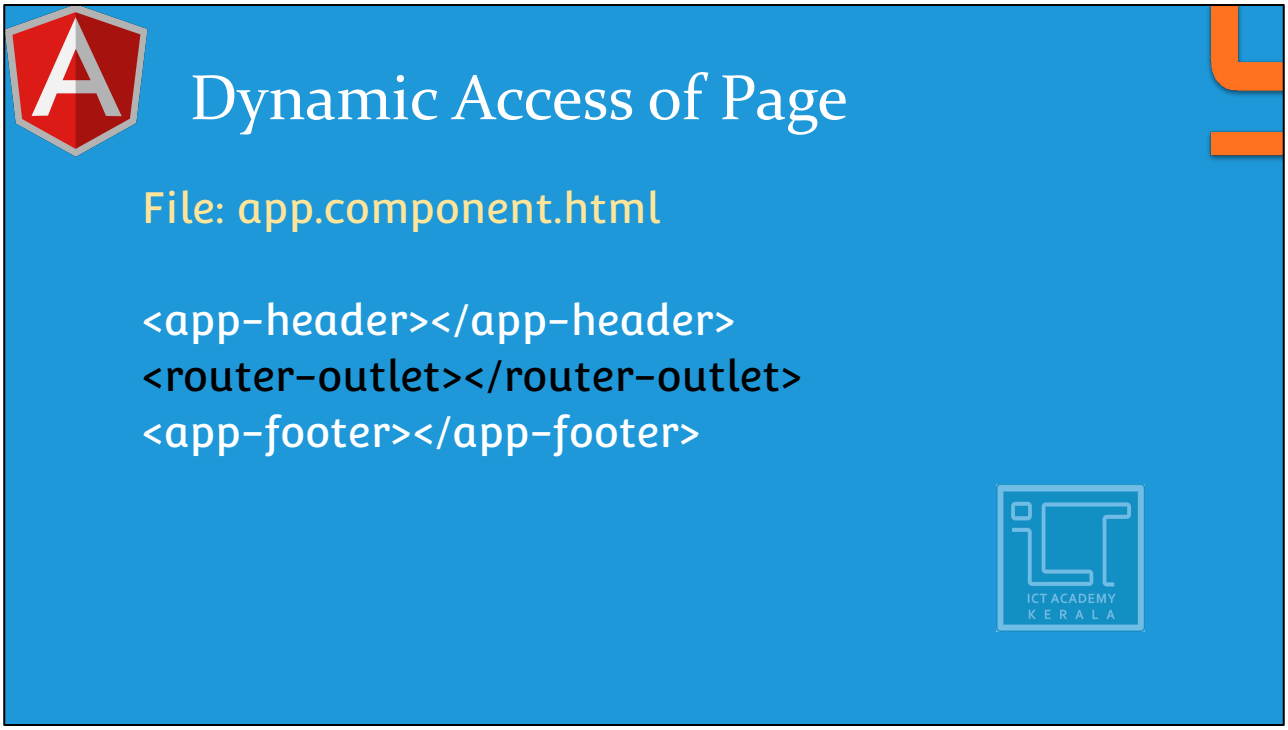
File: header.component.html

```
<div id="links">
    <a href=" ">Home</a>
    <a href="user">Users</a>
    <a href="about">About Us</a>
</div>
```

Path & Component for Routing

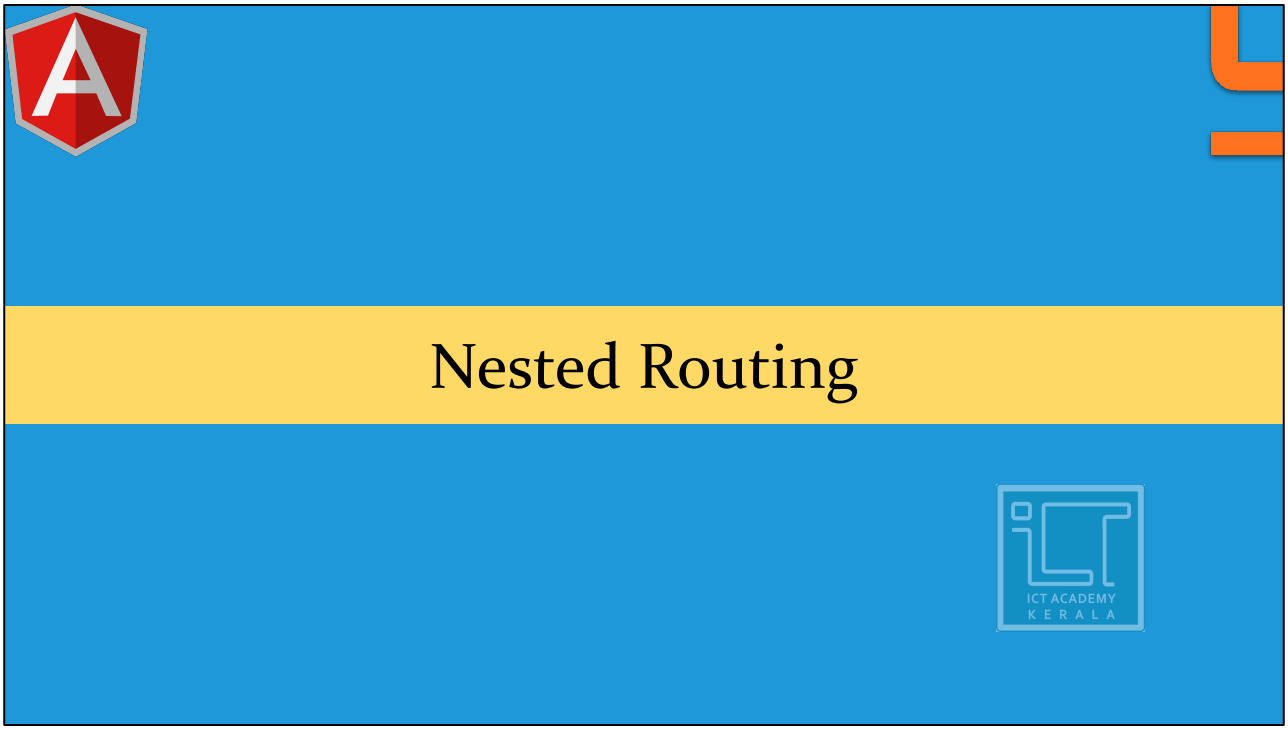
File: app-routing.module.ts

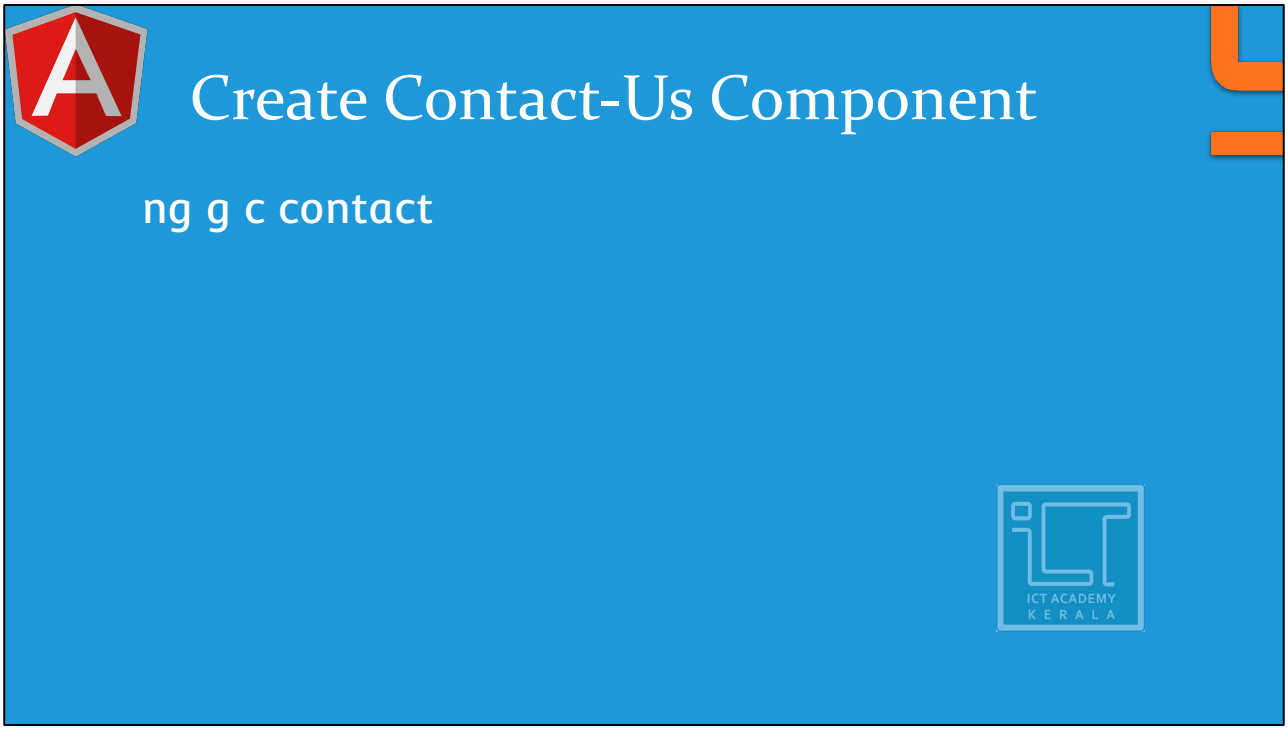
```
const routes: Routes = [{ path: " ", component:
HomeComponent },
{ path: "user", component: UserComponent },
{ path: "about", component: AboutComponent }];
```



Instead of statically fixing the root component, we need to dynamically access the pages when we click any link. For that we use the module called <router-outlet> where it fills with the corresponding page when we click a link.

To make our website a SPA(without reloading), we need to replace href with routerLink for linking the path of the component.

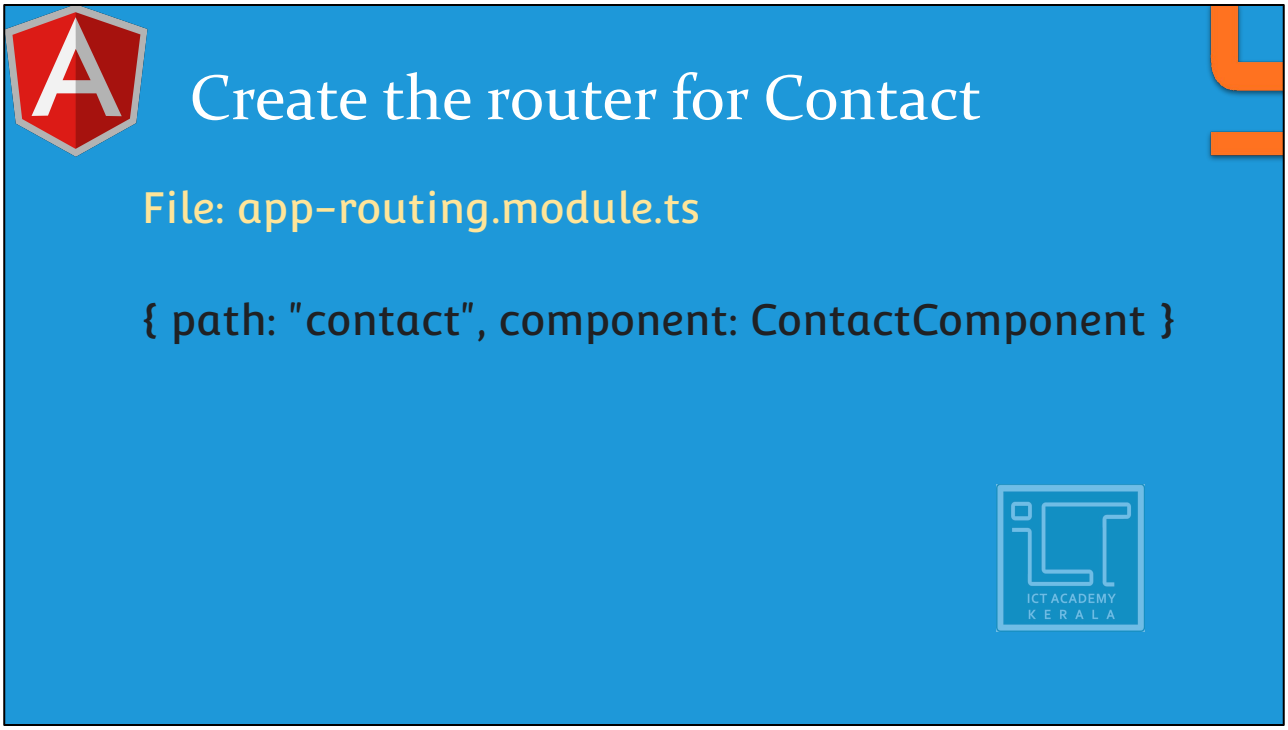

Nested Routing
ICT ACADEMY
KERALA


Create Contact-Us Component
ng g c contact
ICT ACADEMY
KERALA

Adding and Linking into header

File: header.component.html

```
<a routerLink="contact">Contact Us</a>
```



Routing definitions

Contents inside Contact

File: contact.component.html

```
<div class="main">
   <div class="left">
      <a href="">Address</a>
      <a href="">Phone</a>
   </div>
   <div class="right">
   </div>
</div>
```

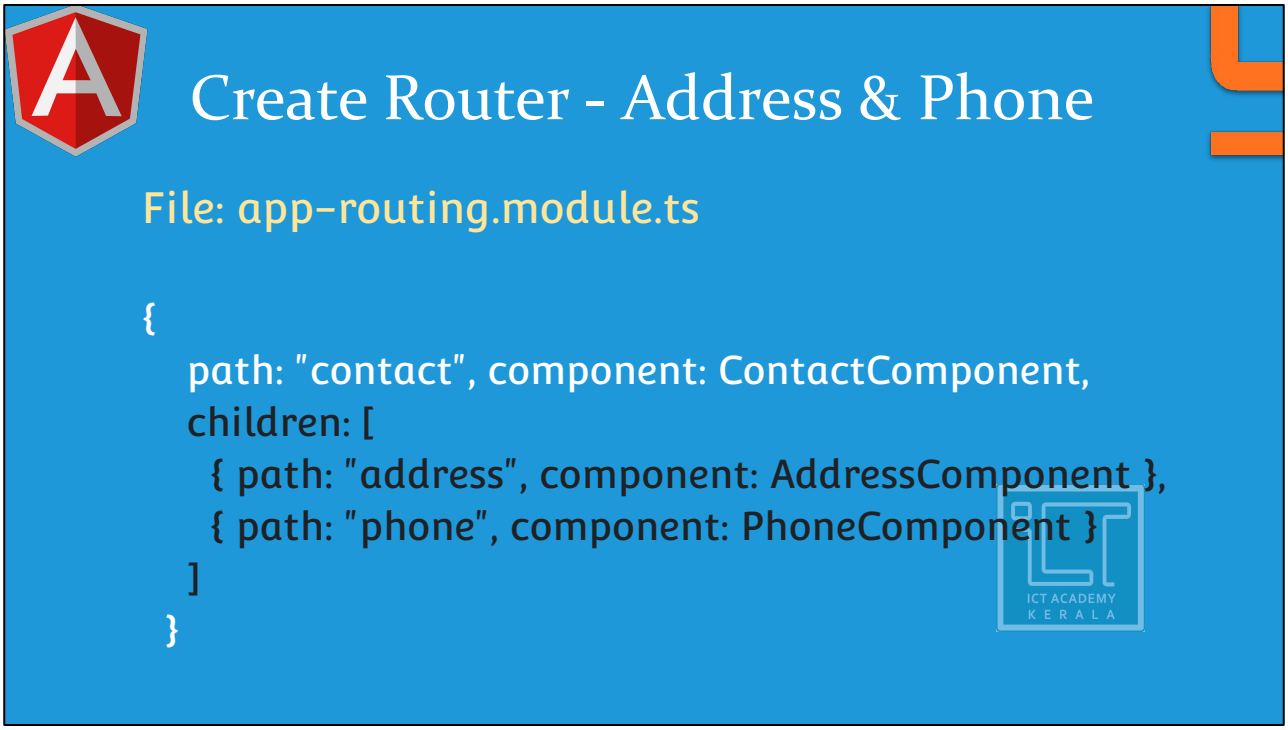
Address and Phone Component

CLI command
ng g c address
ng g c phone

Modifying into SPA

File: contact.component.html

```
<div class="main">
    <div class="left">
        <a routerLink="address">Address</a>
        <a routerLink="phone">Phone</a>
    </div>
    <div class="right">
    </div>
</div>
```

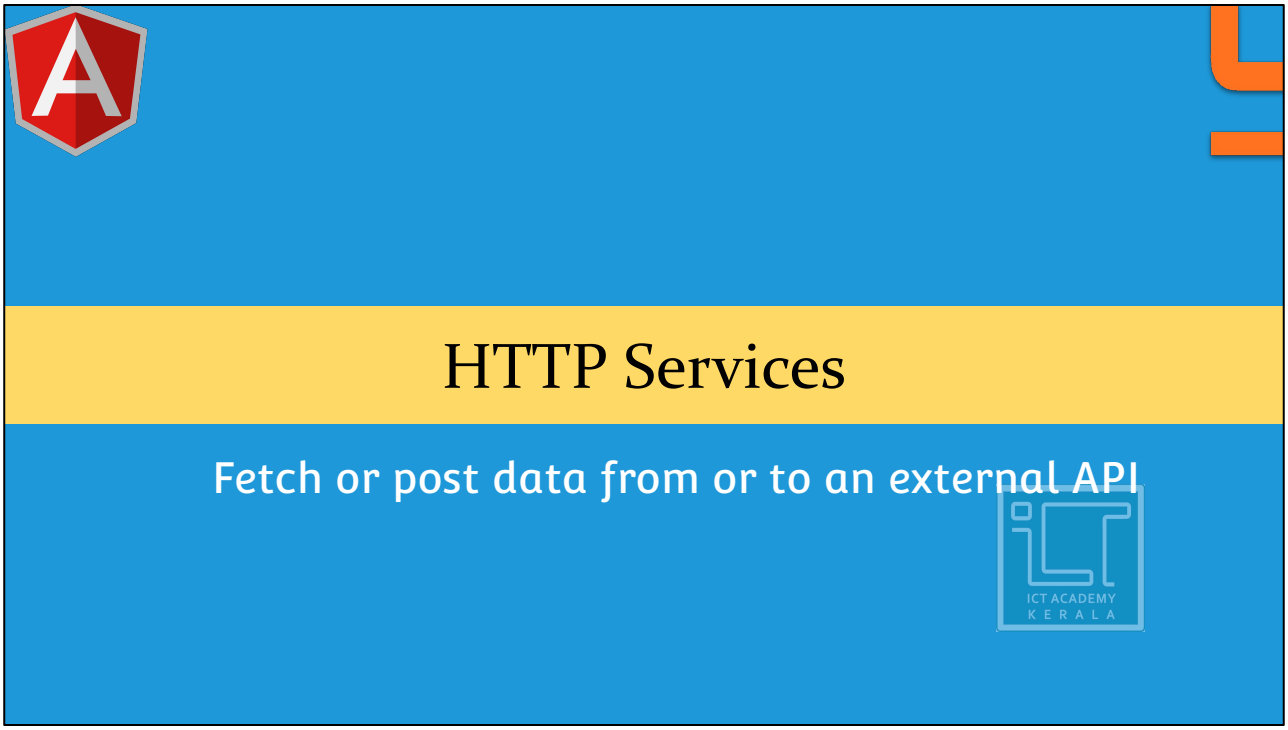


Since Address and Phone component is inside the Contact component, they are created as the children router inside the contact router.

Dynamically accessing pages

File: contact.component.html

```
<div class="main">
   <div class="left">
      <a routerLink="address">Address</a>
      <a routerLink="phone">Phone</a>
   </div>
   <div class="right">
      <router-outlet></router-outlet>
   </div>
</div>
```


HTTP Services
Fetch or post data from or to an external API
ICT ACADEMY
KERALA

HTTP Client Module

Import the HttpClientModule class from the http library

File: app.module.ts

```
import {HttpClientModule} from '@angular/common/http';

imports[
    HttpClientModule
    ]
```

constructor()

- When we are loading a component inside a browser.
- First function is executed when component is getting created.
- All external dependencies, Object creation are done

ngOnInit()

- Angular life-cycle hookup functions:
  ngOnInit, ngOnChanges

- ngOnInit: This function get executed when component is initialised in browser.

Create an Object of httpClient

- In order to create a http request to the API from a component (here, users component)
- For that we can make use of the constructor()
  - All external dependencies
  - Object creation

Object Creation

```
constructor(public http: HttpClient) { }
```

- public: object to be accessed inside another function
- http : object name (any name)
- HttpClient : class name

ngOnInit

- Produce an http request
- get() – to produce a get request to the external URL, inside the HttpClient class

  this.http.get('https://jsonplaceholder.typicode.com/users')

- Since the response is an array we need a variable to store this array – httpdata;

- We need to subscribe for the data

```
.subscribe((users)=>{
    this.httpdata=users;})
```

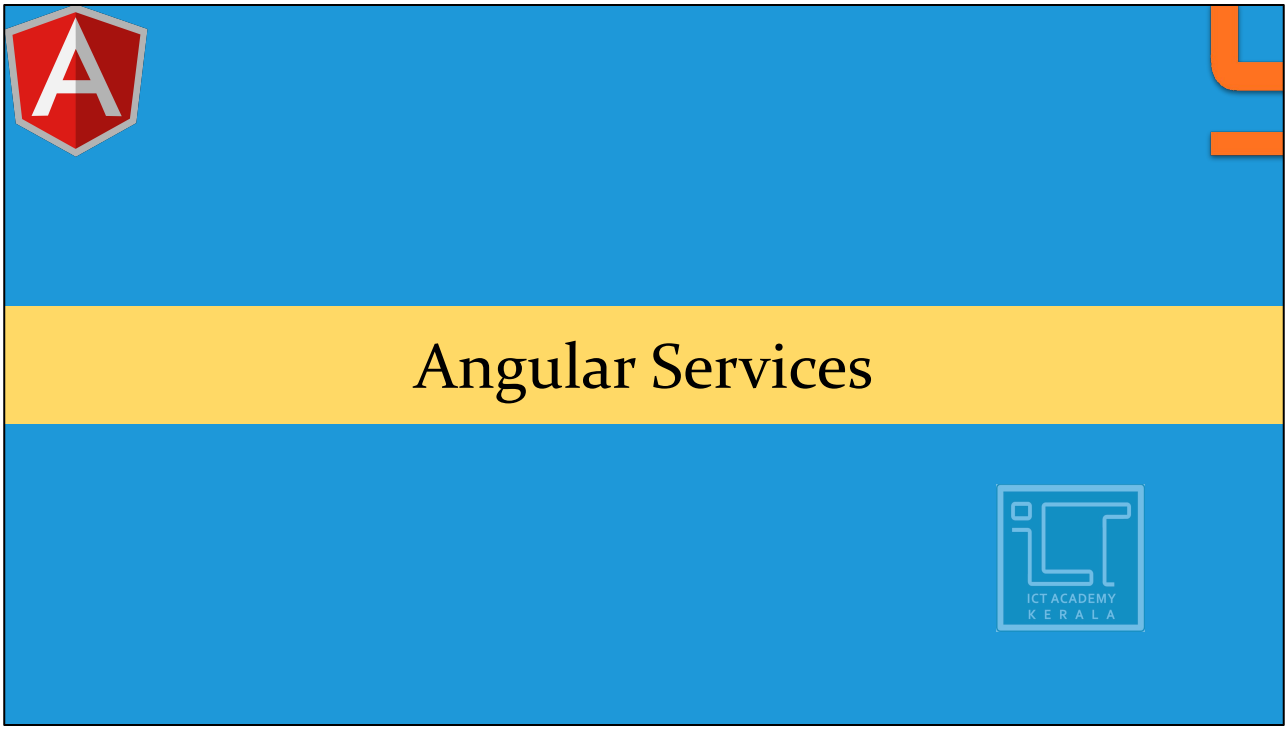
Httpdata variable to be declared before the constructor function
Users is a parameter inside a callback function

```
export class UserComponent implements OnInit {
  httpdata;
  constructor(public http: HttpClient) { }

  ngOnInit() {

    this.http.get('https://jsonplaceholder.typicode.com/users')
      .subscribe((users) => {
        this.httpdata = users;
        console.log(this.httpdata);
      })
  }

}
```

Displaying in Web page

File: user.component.html

```
<div>
   <ul *ngFor="let i of httpdata">
      <li>Name: {{i.name}}</li>
      <li>Address: {{i.address.street}}</li>
   </ul>
</div>
```


Angular Services
ICT ACADEMY
KERALA

Services

All the codes which needs to be reused in future by any other components are kept inside a separate file.

- How to create a service:

  ng generate service service_name

Services

Class with specific purposes

Used for features that are:

- Independent from any particular component
- Provide shared data or logic across components
- Encapsulate external interactions

Services Demo

Task: Both Users and Students component need to use the same users list.

1. Create a students component
2. Modify the students html
3. Add the students link in Header component
4. Add the students path in app.routing.module
5. Create a Service
6. Add the API get request code in the service

people.service.ts

```
1. import { HttpClient } from '@angular/common/http';
2. constructor(public http: HttpClient)
3. getpeople() {
return this.http.get('https://jsonplaceholder.typicode.com/users');
}
```

users.component.ts

```
export class UserComponent implements OnInit {
  httpdata;
  constructor(public people: PeopleService) { }

  ngOnInit() {
    this.people.getpeople()
      .subscribe((user) => {
        this.httpdata = user;
      })

  }
```

users.component.html

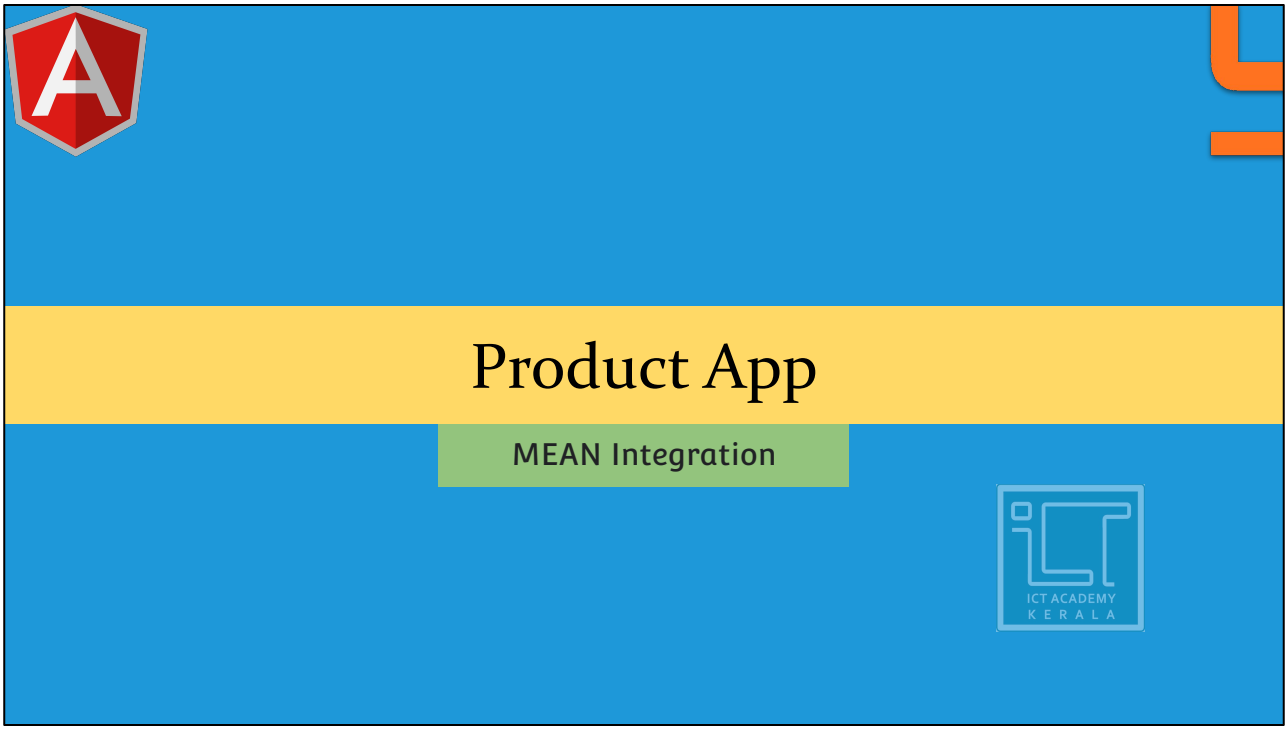
```
<div>
    <h3>User Page</h3>
    <ul *ngFor="let i of httpdata">
        <li>Name: {{i.name}}</li>
        <li>Address: {{i.address.street}}</li>
    </ul>
</div>
```

students.component.ts

```
export class StudentsComponent implements OnInit {
  httpdata;
  constructor(public people: PeopleService) { }

  ngOnInit() {
    this.people.getpeople()
      .subscribe((student) => {
        this.httpdata = student;
      })
  }
```

students.component.html

```
<div>
    <h3>Students Page</h3>
    <ul *ngFor="let i of httpdata">
        <li>Name:{{i.name}}</li>
        <li>Address:{{i.address.street}}</li>
    </ul>
</div>
```


Product App
MEAN Integration
ICT ACADEMY
KERALA

1. Create the Project

- ng new Products
- ng new Products --skip-install

3. Component Creation

1. Create header component – ng g c header
2. Add the header selector in app.component.html
3. Create product_list component – ng g c product_list
4. Define the routing path of product_list in app.routing.module.ts
5. Add the <router-outlet> in app.component.html

Header Component

1. Add a String variable title in *TS file*

   title:String = “Product Management”
2. Access this title in *HTML file* using <h1>

   <h1>{{title}}</h1>
3. Create an Anchor Tag inside a div in *HTML file*

   <a routerLink="/">Products

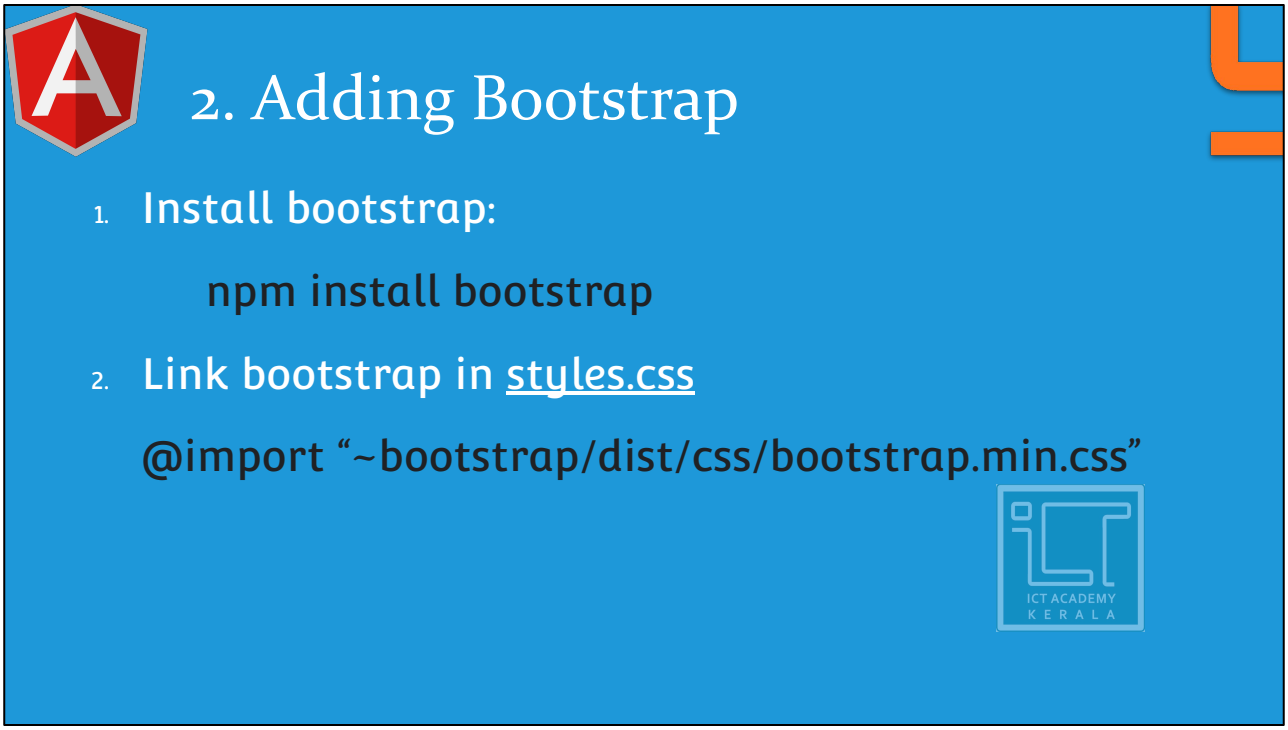
Product List Component

1. Add a String variable title in *TS file*

   title:String = “Product List”
2. Access this title in *HTML file* inside a div

```
<div class="card-header">
    {{title}}
</div>
```
3. Create a table for listing the products

“ ” specifies root(/)



Other ways to link bootstrap is:

1. CDN method directly in index.html
2. angular.json

Video 2

Product Listing Backend

1. Create a separate folder – Product-Backend
2. Create package.json – npm init
3. Install express with node_modules – npm install express --save
4. Create app.js file – add the express structure
5. Create a database model – ProductData.js
6. Create 3-4 documents in Mongodb Compass directly

app.js

```
const express = require('express');
const ProductData = require('./src/model/ProductData');
const app = new express();

app.get('/products', function (req, res) {
    ProductData.find()
        .then(function (products) {
            res.send(products);
        })
});

app.listen(3000);
```


DB Model - ProductData.js
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/ProductDb');
const Schema = mongoose.Schema;
var ProductSchema = new Schema({
productCode: String,
productName: String,
availability: String,
price: Number,
starRating: Number,
imageURL: String
});
var Productdata = mongoose.model('product', ProductSchema);
module.exports = Productdata;

CORS Importance

If a request from any front-end application need to be accepted by the backend then we needs to specify a package called : cors

1. Install cors
2. Require cors
3. app.use(cors())
4. Mention the origin as * meaning from any source
5. Mention the http request methods


CORS
1 const express = require('express');
2 const cors = require('cors');
3 const ProductData = require('./src/model/ProductData');
4
5 const app = new express();
6 app.use(cors());
7
8 app.get('/products', function (req, res) {
9 res.header("Access-Control-Allow-Origin","*");
10 res.header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE")
11 ProductData.find()
12 .then(function (products) {
13 res.send(products);
14 })
15 });
16
17 app.listen(3000);

2, 6, 9, 10 - LOC

Video 2

HttpClientModule

If a frontend need a data from backend then it require http request.

- Import HttpClientModule from http library

File: app.module.ts

```
import {HttpClientModule} from '@angular/common/http';
imports[
    HttpClientModule
    ]
```

Angular Services

For creating a http request,

1. create a new service – ng g s product
2. import the HttpClient from http library
3. create an object of HttpClient class
4. create a function to define the http get request

Angular Services

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class ProductService {

  constructor(public http: HttpClient) { }
  getProducts() {
    return this.http.get('http://localhost:3000/products');
  }
}
```

Model Schema for Product

1. Create a new file under productlist

   product.model.ts

2. Inside this file we create a schema for single product

   *Similar to mongodb Scheme

product.model.ts

```
export class ProductModel {
    constructor(
        public productCode: Number,
        public productName: String,
        public availability: String,
        public price: Number,
        public starRating: number,
        public imageURL: String
    ) { }
}
```

productlist.component.ts

1. Capture the data from backend to a products array variable which is of type ProductModel. *Import the same*
2. Set the image properties – width & margin
3. Create an object of ProductService along with its class inside constructor. *Import the same*
4. Subscribe the data which needs to be kept in products array
5. Stringify the data and then Parse it and keep it in products array since it's of ProductModel type

productlist.component.ts

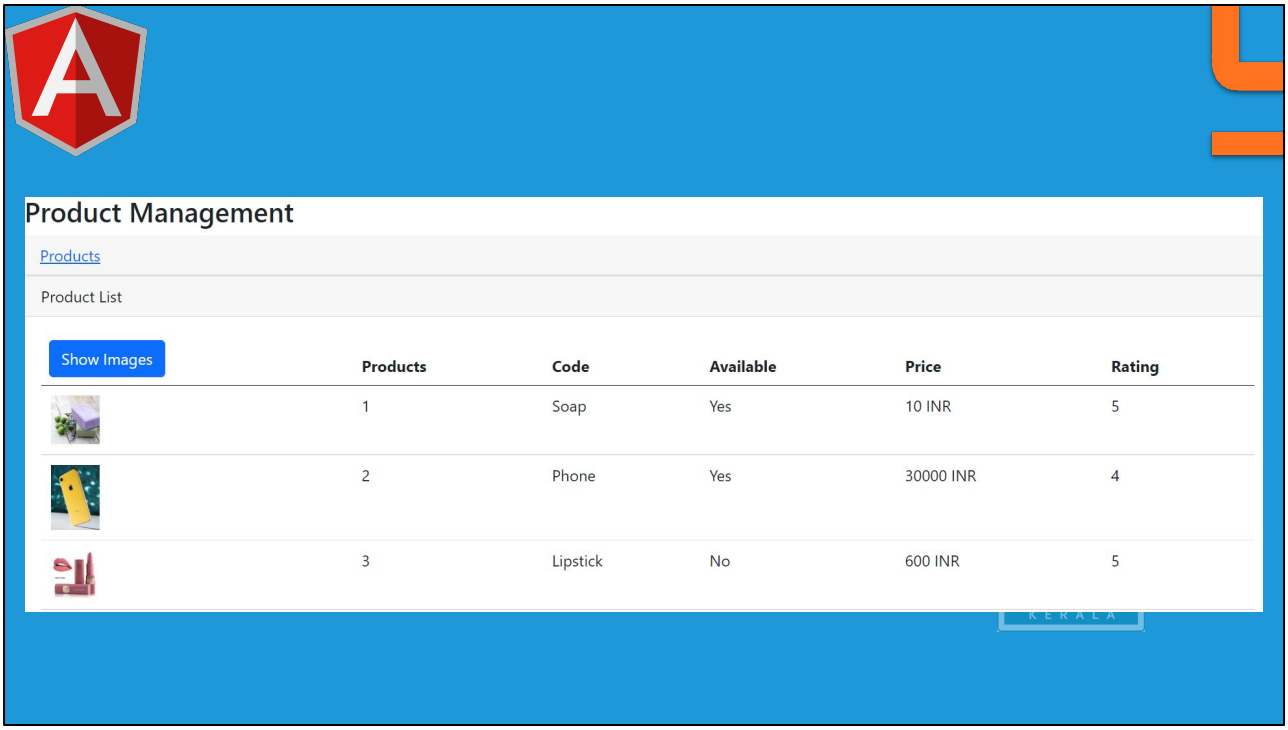
```
import { Component, OnInit } from '@angular/core';
import { ProductModel } from './product.model';
import { ProductService } from '../product.service';

@Component({
  selector: 'app-productlist',
  templateUrl: './productlist.component.html',
  styleUrls: ['./productlist.component.css']
})
export class ProductlistComponent implements OnInit {
  title: String = "Product List";

  products: ProductModel[];
  imageWidth: number = 50;
  imageMargin: number = 2;

  constructor(private productService: ProductService) { }

  ngOnInit(): void {
    this.productService.getProducts().subscribe((data) => {
      this.products = JSON.parse(JSON.stringify(data));
    })
  }
```

Product listing Frontend

- Inside product.component.html, inside table body, we need to loop through the products array and display the items in browser(frontend)
- For looping we require ngFor
- Image properties should be accessed in HTML using Property Binding [ ]

productlist.component.html

```
<tbody>
    <!-- Data from the database should display here -->
    <tr *ngFor='let i of products'>
        <td><img [src]="i.imageURL" alt="image"
        [style.width.px]='imageWidth'
        [style.margin.px]='imageMargin'>
        </td>
        <td>{{i.productCode}}</td>
        <td>{{i.productName}}</td>
        <td>{{i.availability}}</td>
        <td>{{i.price}}</td>
        <td>{{i.starRating}}</td>
    </tr>
</tbody>
```


Product Management
Products
Product List
Show Images
Products
Code
Available
Price
Rating
1
Soap
Yes
10 INR
5
2
Phone
Yes
30000 INR
4
3
Lipstick
No
600 INR
5

Video 2

header.component.html

Add an additional anchor tag with Add Products

```
<div class="card-header">
    <a routerLink="/">Products</a>
    <a routerLink="/add">Add Products</a>
</div>
```

Add-Product Component

- Add a new component new-product
  - ng g c new-product

Add the Routing

Add the path and corresponding new-product component in the app-routing.module.ts file

```
const routes: Routes = [
  { path: "", component: ProductlistComponent },
  { path: "/add", component: NewProductComponent }
];
```

Creation of Product Form

```
<div class="card">
    <form action="">
        <!-- Header -->
        <div class="card-header">
            {{title}}
        </div>

        <!-- Body -->
        <div class="card-body">
            <div class="table-responsive">
                <table class="table">
                    <tbody>
                        <tr>
                            <td>Product Code</td>
                            <td><input type="number"></td>
                        </tr>
                        <tr>
                            <td>Product Name</td>
                            <td><input type="text"></td>
                        </tr>
                        <tr>
```

Create a FORM for the user to Add Product.

new-product.component.html

New-product.component.html

ngModel should get each value from the productModel using the object variable created inside new-product.component.ts

NewProduct()

- This function will get executed when the user submit the form
- Call this function in the form submit
- When form is getting submitted, function inside productService will get called. Ie. addProduct()

new-product.component.ts

new-product.component.ts

```
constructor(private productService: ProductService, private router: Router)
productItem = new ProductModel(null, null, null, null, null, null);
ngOnInit(): void {
}

NewProduct() {
  this.productService.addProduct(this.productItem);
  console.log("Called");
  alert("New Product Added");
  this.router.navigate(['/']);
}
```

Import {Router} from '@angular/router';

new-product.component.ts

- Create a title variable
- Call the product service inside the constructor

```
title: String = 'Add Product';
constructor(private productService: ProductService) { }
```

“https://localhost:3000/insert” needs to be mentioned in the Back-End(Node,Express)

addProduct(item)

- This function will accept an item(form details from user)
- This function is used to send the form data to the Backend – DB
- The /insert with POST method should be defined in our Backend – Node/Express

product.service.ts

productItem

- productItem needs to be send to the Backend to DB
- This will be formed when user fill the form
- New ProductModel
- During the creation of productItem, all the fields will be null
- When user fills the forms, these null will be replaced with corresponding values

new-product.component.ts

new-product.component.ts

- Create a new object of the ProductModel with initial values as null.
- Create a new function to call the addProduct() from product.service.ts and pass the above object
- Console some messages for debugging
- Create an alert for the user
- After adding a new product redirect to home page (/)
  - For redirecting, you need Router module

Accessing the data in HTML

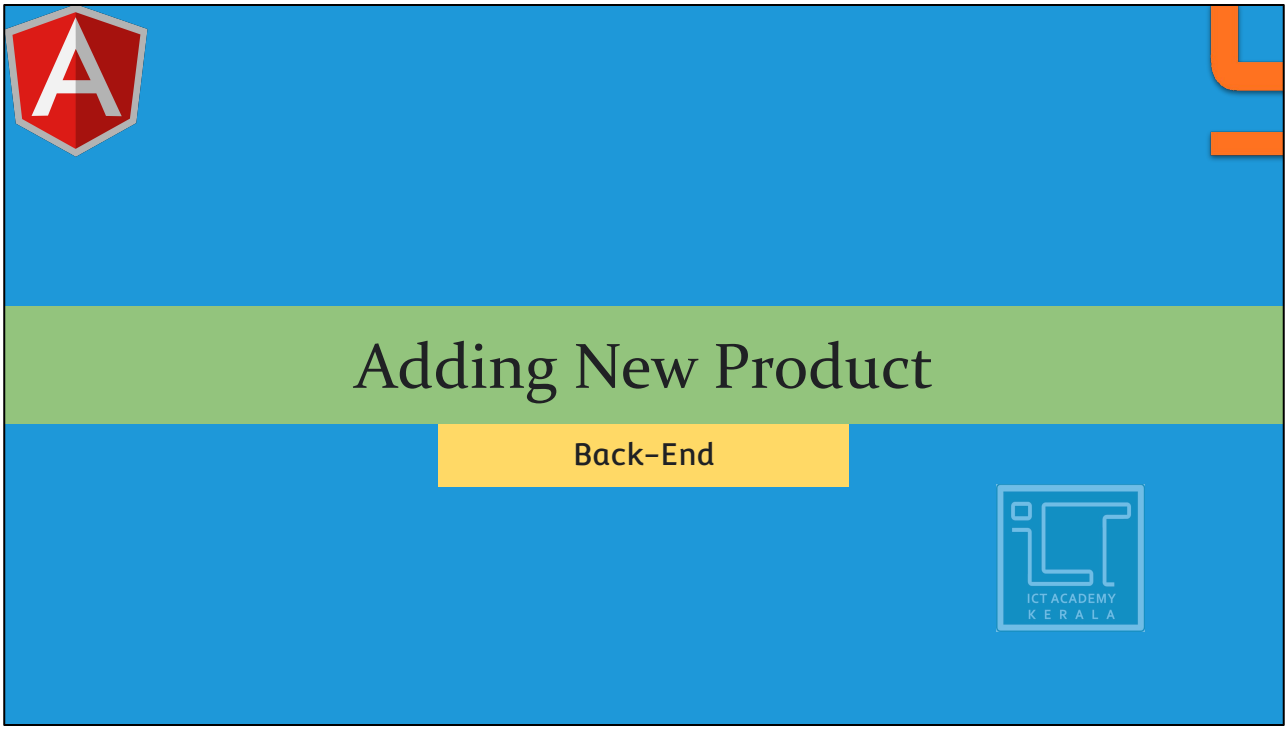
1. Every input box should have 2 parameters
   a. name
   b. [(ngModel)]
      i. Import FormsModule for using ngModel

```
<tr>
    <td>Product Code</td>
    <td><input type="number" name="pcode" [(ngModel)]="productItem.productCode"></td>
</tr>
```

Accessing data from Backend

For accessing the data from backend we need use the SUBSCRIBE function.

```
addProduct(item) {
  console.log("Accessed");
  return this.http.post('http://localhost:3000/insert', { item })
    .subscribe(data => { console.log(data) });
}
```



Video 2

Post Request

```
app.post('/insert', function (req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE");
    console.log(req.body);
    var product = {
        productCode: req.body.item.productCode,
        productName: req.body.item.productName,
        availability: req.body.item.availability,
        price: req.body.item.price,
        starRating: req.body.item.starRating,
        imageURL: req.body.item.imageURL
    }

    var products = new ProductData(product);
    products.save();
})
```

Body-Parser

Since we are using req.body we need to

- require body-parser
- Train the system to use bodyparser.json()

```
const bodyparser = require('body-parser');
app.use(bodyparser.json());
```

Parameters

- Item – data passed from Front-End
- All the items accessed from the front-end saved to an object named ‘products’
- ‘products’ created as a new object as per the DB Model
- Then saved to the DB using save();