SunilOS

Angular 8

www.sunilos.com
www.raystec.com

Think IT Think Us RAYS

SunilOS

Contents

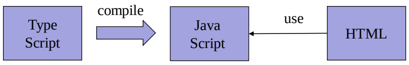
- ❑ Introduction & Overview
- ❑ MVC Architecture
- ❑ Installation and Configuration
- ❑ Getting Started
- ❑ Variable and Operators
- ❑ Control Statements
- ❑ Directives
- ❑ Module
- ❑ Component
- ❑ Pipe
- ❑ Services
- ❑ Router
- ❑ Http Client
- ❑ Forms

SunilOS

What is ANGULAR

- ❑ Angular is a JavaScript Framework
- ❑ Angular is used to build client-side applications using HTML
- ❑ Angular bootstraps JavaScript with HTML tags.
- ❑ Angular is used to make reach UI application.
- ❑ Angular enhances UI experience for User.

- ❑ Angular code is written in TypeScript language
  - o TypeScript is compiled into JavaScript
  - o JavaScript is used in HTML pages



SunilOS

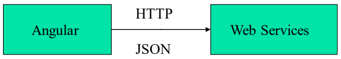
Angular enhances HTML

- ❑ Angular has set of directives to display dynamic contents at HTML page. Angular extends HTML node capabilities for a web application.
- ❑ Angular provides data binding and dependency injection that reduces line of code.
- ❑ Angular extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.
- ❑ Angular follows MVC Architecture

SunilOS

Angular – REST Web Services

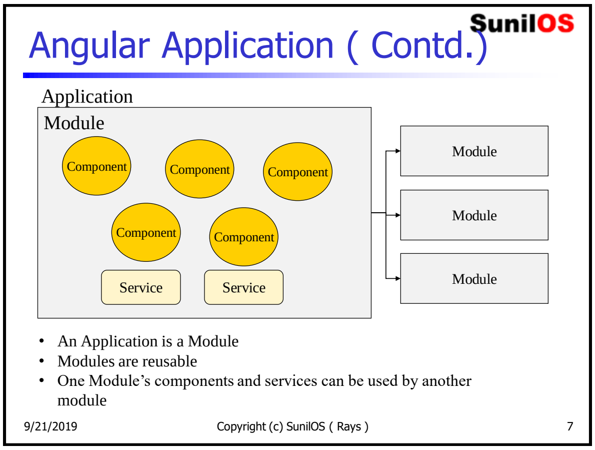
- ❑ Angular communicates with RESTFul web services in modern applications
- ❑ RESTful Web Services are accessed by HTTP call
- ❑ RESTful Web Services exchange JSON data



SunilOS

Angular Application

- ❑ An application is a Module
- ❑ Module contains components
- ❑ Component uses Services
- ❑ Services contains data and reusable business methods
- ❑ Basic building block of Angular is Component
- ❑ It is said that Angular follows Component/Service architecture. Internally it follows MVC Architecture



Angular Application (Contd.)

SunilOS

Application

Module

Component

Component

Component

Component

Component

Service

Service

Module

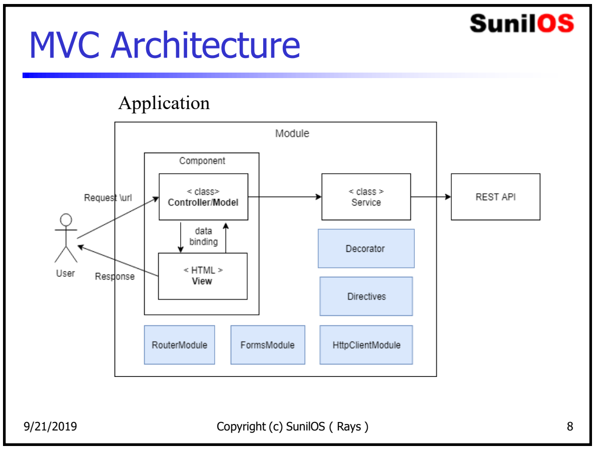
Module

Module

- An Application is a Module
- Modules are reusable
- One Module's components and services can be used by another module

9/21/2019 Copyright (c) SunilOS (Rays) 7

7



MVC Architecture

SunilOS

Application

Module

Component

< class> Controller/Model

data binding

< HTML > View

< class > Service

REST API

Decorator

Directives

RouterModule

FormsModule

HttpClientModule

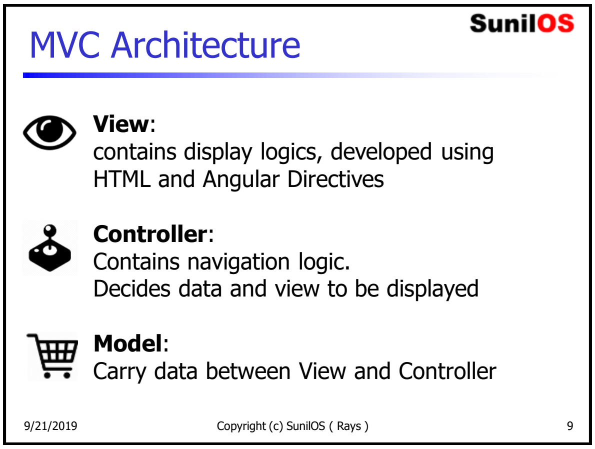
Request \url

Response

User

9/21/2019 Copyright (c) SunilOS (Rays) 8

8



MVC Architecture

SunilOS

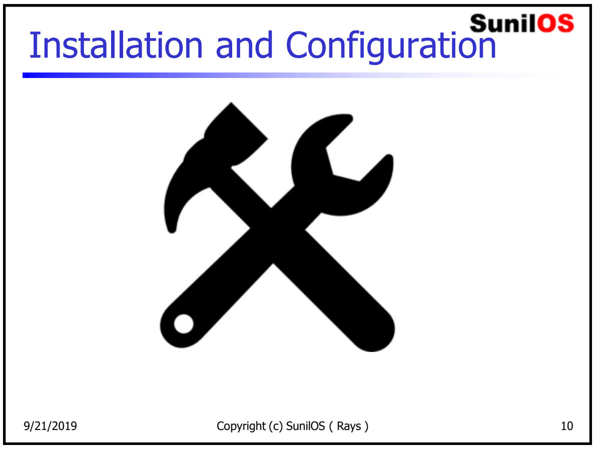
View:
contains display logics, developed using HTML and Angular Directives

Controller:
Contains navigation logic.
Decides data and view to be displayed

Model:
Carry data between View and Controller

9/21/2019 Copyright (c) SunilOS (Rays) 9

9

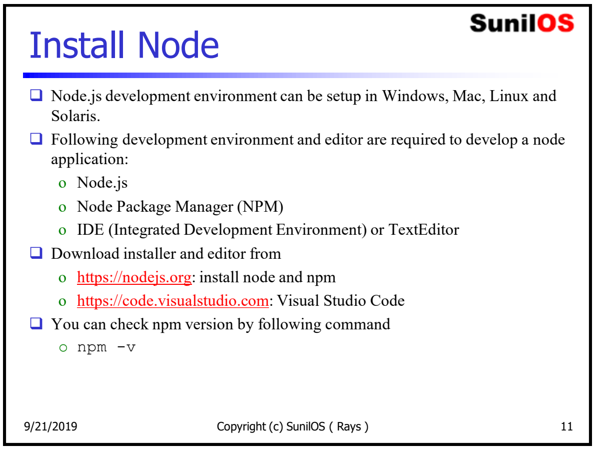


Installation and Configuration

SunilOS

9/21/2019 Copyright (c) SunilOS (Rays) 10

10



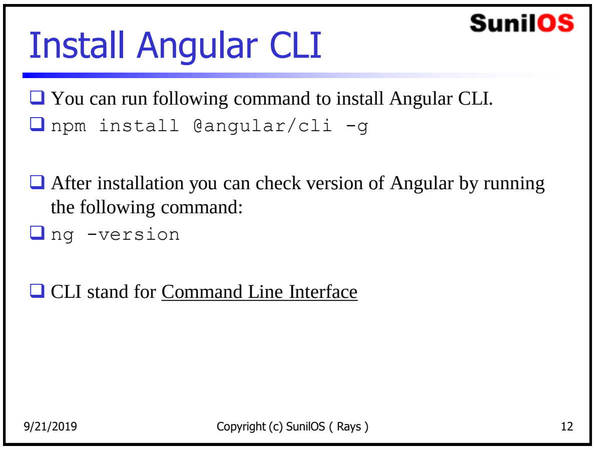
Install Node

SunilOS

- ❑ Node.js development environment can be setup in Windows, Mac, Linux and Solaris.
- ❑ Following development environment and editor are required to develop a node application:
  - o Node.js
  - o Node Package Manager (NPM)
  - o IDE (Integrated Development Environment) or TextEditor
- ❑ Download installer and editor from
  - o https://nodejs.org: install node and npm
  - o https://code.visualstudio.com: Visual Studio Code
- ❑ You can check npm version by following command
  - o `npm -v`

9/21/2019 Copyright (c) SunilOS (Rays) 11

11



Install Angular CLI

SunilOS

- ❑ You can run following command to install Angular CLI.
- ❑ `npm install @angular/cli -g`

- ❑ After installation you can check version of Angular by running the following command:
- ❑ `ng -version`

- ❑ CLI stand for Command Line Interface

9/21/2019 Copyright (c) SunilOS (Rays) 12

12

Angular Project

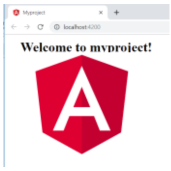
- ❑ Angular provides CLI command to generate project template

Create Project

- ❑ Angular project is created using command:
  - o `ng new project-name`
- ❑ We are assuming project name is SOS
  - o `ng new SOS`
  - o Above command will create project directory structure. Default component and configuration files is generated inside `c:/SOS` folder.
- ❑ Inside c:/SOS folder it will create following subfolders
  - o `c:/SOS/e2e`
  - o `c:/SOS/node_modules`
  - o `c:/SOS/src/app`
- ❑ All components will be created in `c:/sos/src/app` folder

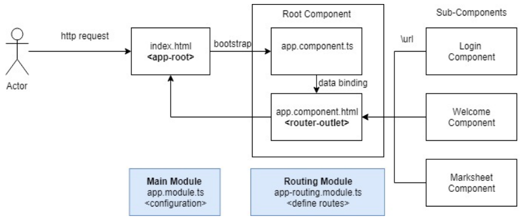
Run the project

- ❑ Run following command to run angular project
  - o `c:/SOS>ng serve -o`
- ❑ It will start angular server at default port number #4200 and make application accessible using http://localhost:4200



Welcome to myproject!

Project flow



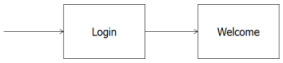
Project Key components

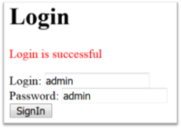
- ❑ File `app.module.ts` contains configuration of the application.
- ❑ File `app-routing.module.ts` contains url mapping of components. Components are accessed by their mapped urls.
- ❑ File `app.component.ts` contains definition of Root-Component
- ❑ File `index.html` is first page of application. It bootstraps root component.
- ❑ For new UI screens new components are created called sub components.

Getting Started

Login

- Let’s create an application which contains `Login` and `Welcome` pages.
- Login page has HTML form that accepts `Login-id` and `password`.
- When you submit `Login` page, credential is authenticated and Welcome page is displayed on successful login.





Login

Login is successful

Login: admin
Password: admin
SignIn

19

Login & Welcome Components

- Basic building block of Angular is Component
- One component is created for one View page.
- We have Login and Welcome 2 pages thus 2 components will be created.

Login
Welcome

20

Create components

- We are assuming that you have creates SOS project from previous slides and we will create Login and Welcome page in same project.
- Use following command in c:\SOS folder to generate Login component
  - `ng generate component Login`
    - or
  - `ng g c Login`
  - `ng g c Welcome`
- Above commands will generate Login and Welcome components and will automatically make entry in `app.module.ts` configuration file.

21

Generated Files

Welcome

- `c:/SOS/src/app/welcome/welcome.component.css`
- `c:/SOS/src/app/welcome/`**`welcome.component.html`**
- `c:/SOS/src/app/welcome/`**`welcome.component.ts`**
- `c:/SOS/src/app/welcome/welcome.component.spec.ts`

Login

- `c:/SOS/src/app/login/login.component.css`
- `c:/SOS/src/app/login/`**`login.component.html`**
- `c:/SOS/src/app/login/`**`login.component.ts`**
- `c:/SOS/src/app/login/login.component.spec.ts`

22

Define routes

- In order to access pages, define routes in `app-routing.module.ts`

```
const routes: Routes = [
  { path: 'login', component: LoginComponent},
  { path: 'welcome', component: WelcomeComponent}
];
```

```
#app.module.ts file
@NgModule({
  imports: [RouterModule.forRoot(routes)]
  exports: [RouterModule]
})
```

- Access pages using following URLs
  - http://localhost:4200/login
  - http://localhost:4200/welcome

23

Welcome page

- Lets initialize a variable `message` and display at html page

- File `welcome.component.ts`

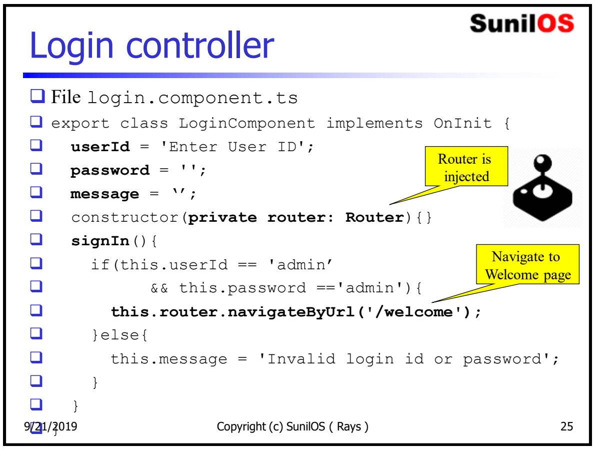
```
export class WelcomeComponent implements OnInit {
  message = 'Welcome to SunilOS’;
}
```

- File `welcome.component.html`

```
<H1>{{ message }}</H1>
```

- URL
  - http://localhost:4200/welcome

24



SunilOS

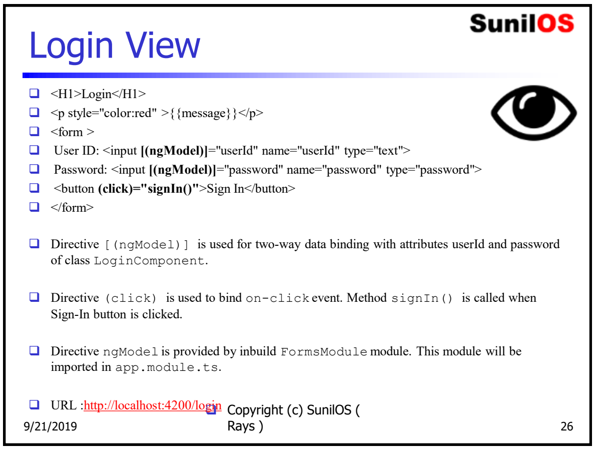
Login controller

- File `login.component.ts`

```
export class LoginComponent implements OnInit {
  userId = 'Enter User ID';
  password = '';
  message = '';
  constructor(private router: Router){}
  signIn(){
    if(this.userId == 'admin'
          && this.password =='admin'){
      this.router.navigateByUrl('/welcome');
    }else{
      this.message = 'Invalid login id or password';
    }
  }
}
```

Router is injected

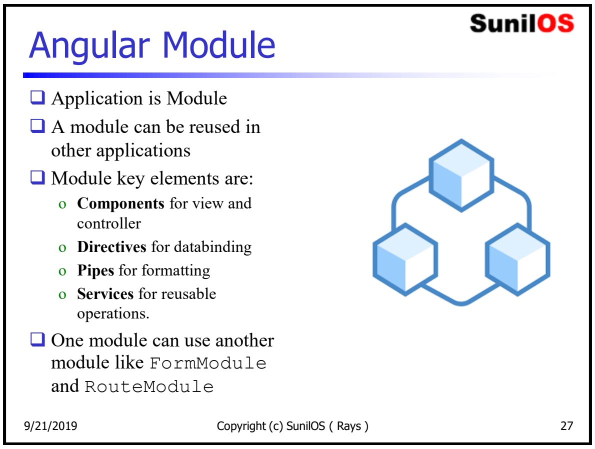
Navigate to Welcome page



SunilOS

Login View

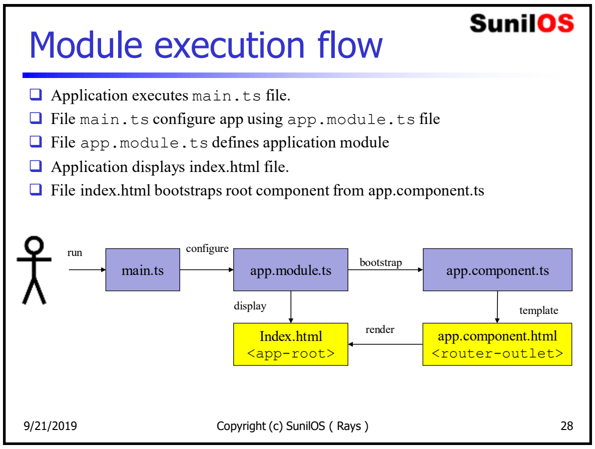
- <H1>Login</H1>
- <p style="color:red" >{{message}}</p>
- <form >
- User ID: <input **[(ngModel)]**="userId" name="userId" type="text">
- Password: <input **[(ngModel)]**="password" name="password" type="password">
- <button **(click)="signIn()"**>Sign In</button>
- </form>
- Directive `[(ngModel)]` is used for two-way data binding with attributes userId and password of class `LoginComponent`.
- Directive `(click)` is used to bind `on-click` event. Method `signIn()` is called when Sign-In button is clicked.
- Directive `ngModel` is provided by inbuild `FormsModule` module. This module will be imported in `app.module.ts`.
- URL :http://localhost:4200/login



SunilOS

Angular Module

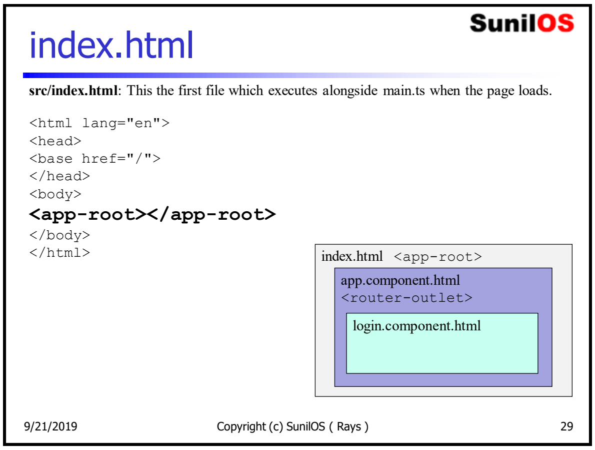
- Application is Module
- A module can be reused in other applications
- Module key elements are:
  - **Components** for view and controller
  - **Directives** for databinding
  - **Pipes** for formatting
  - **Services** for reusable operations.
- One module can use another module like `FormModule` and `RouteModule`



SunilOS

Module execution flow

- Application executes `main.ts` file.
- File `main.ts` configure app using `app.module.ts` file
- File `app.module.ts` defines application module
- Application displays index.html file.
- File index.html bootstraps root component from app.component.ts



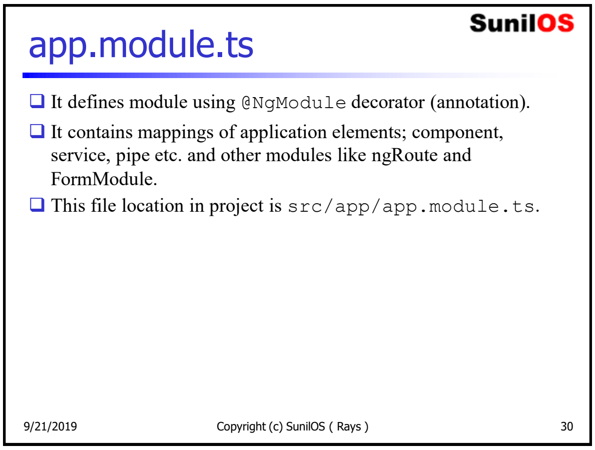
SunilOS

index.html

src/index.html: This the first file which executes alongside main.ts when the page loads.

```
<html lang="en">
<head>
<base href="/">
</head>
<body>
<app-root></app-root>
</body>
</html>
```

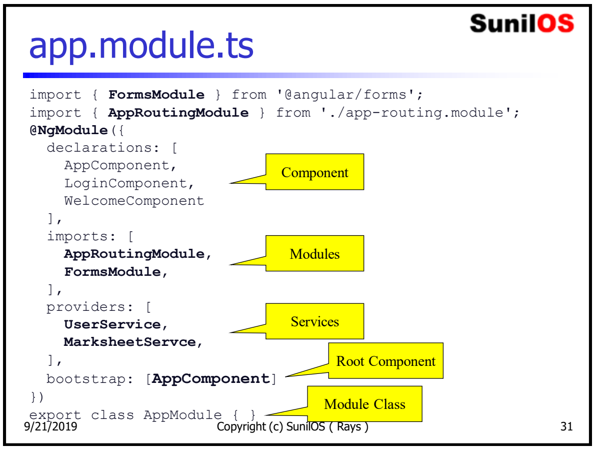
index.html <app-root>
app.component.html
<router-outlet>
login.component.html



SunilOS

app.module.ts

- It defines module using `@NgModule` decorator (annotation).
- It contains mappings of application elements; component, service, pipe etc. and other modules like ngRoute and FormModule.
- This file location in project is `src/app/app.module.ts`.



SunilOS

app.module.ts

```
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    WelcomeComponent
  ],
  imports: [
    AppRoutingModule,
    FormsModule,
  ],
  providers: [
    UserService,
    MarksheetServce,
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Component

Modules

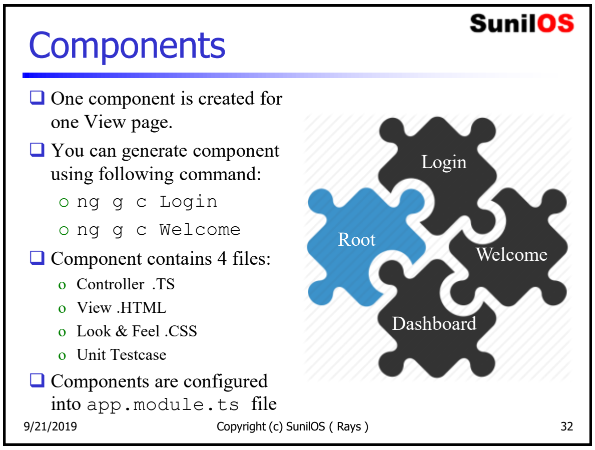
Services

Root Component

Module Class

9/21/2019 Copyright (c) SunilOS (Rays) 31

31



SunilOS

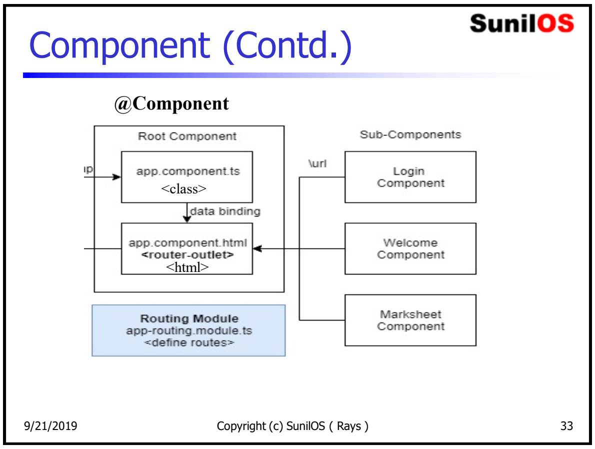
Components

- ❑ One component is created for one View page.
- ❑ You can generate component using following command:
  - o `ng g c Login`
  - o `ng g c Welcome`
- ❑ Component contains 4 files:
  - o Controller .TS
  - o View .HTML
  - o Look & Feel .CSS
  - o Unit Testcase
- ❑ Components are configured into `app.module.ts` file

Login
Root
Welcome
Dashboard

9/21/2019 Copyright (c) SunilOS (Rays) 32

32



SunilOS

Component (Contd.)

@Component

Root Component

app.component.ts
<class>

data binding

app.component.html
<router-outlet>
<html>

Routing Module
app-routing.module.ts
<define routes>

Sub-Components

\url

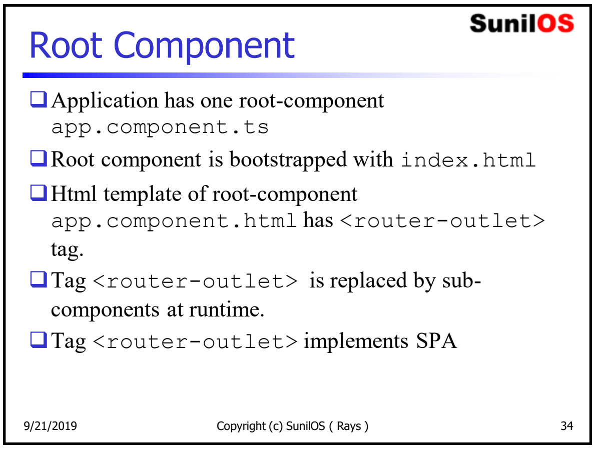
Login Component

Welcome Component

Marksheet Component

9/21/2019 Copyright (c) SunilOS (Rays) 33

33



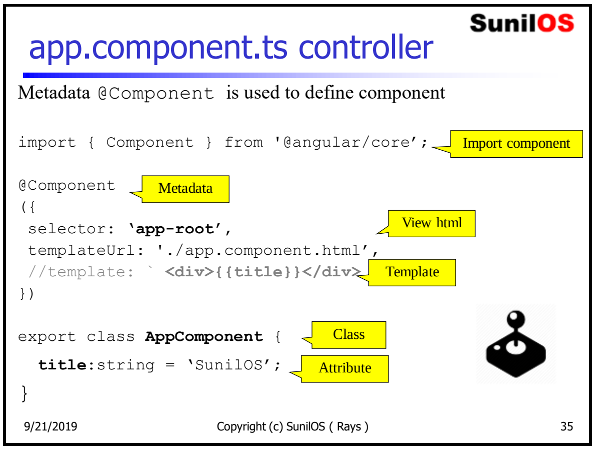
SunilOS

Root Component

- ❑ Application has one root-component `app.component.ts`
- ❑ Root component is bootstrapped with `index.html`
- ❑ Html template of root-component `app.component.html` has `<router-outlet>` tag.
- ❑ Tag `<router-outlet>` is replaced by sub-components at runtime.
- ❑ Tag `<router-outlet>` implements SPA

9/21/2019 Copyright (c) SunilOS (Rays) 34

34



SunilOS

app.component.ts controller

Metadata `@Component` is used to define component

```
import { Component } from '@angular/core';

@Component
({
 selector: 'app-root',
 templateUrl: './app.component.html',
 //template: ` <div>{{title}}</div>
})

export class AppComponent {
  title:string = 'SunilOS';
}
```

Import component

Metadata

View html

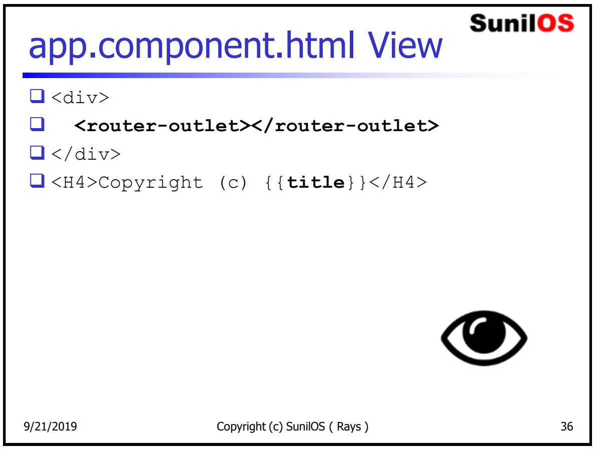
Template

Class

Attribute

9/21/2019 Copyright (c) SunilOS (Rays) 35

35



SunilOS

app.component.html View

```
❑ <div>
❑   <router-outlet></router-outlet>
❑ </div>
❑ <H4>Copyright (c) {{title}}</H4>
```

9/21/2019 Copyright (c) SunilOS (Rays) 36

36

Create sub-component- Login

SunilOS

- File `login.component.ts`

```
export class LoginComponent implements OnInit {
  userId = 'Enter User ID';
  password = '';
  message = '';
  constructor(private router: Router){}
  signIn(){
    if(this.userId == 'admin'
          && this.password =='admin'){
      this.router.navigateByUrl('/welcome');
    }else{
      this.message = 'Invalid login id or password';
    }
  }
}
```

Router is injected

Navigate to Welcome page

37

Login View

SunilOS

```
<H1>Login</H1>
<p style="color:red" >{{message}}</p>
<form >
 User ID: <input [(ngModel)]="userId" name="userId" type="text">
 Password: <input [(ngModel)]="password" name="password" type="password">
 <button (click)="signIn()">Sign In</button>
</form>
```

- Directive `[(ngModel)]` is used for two-way data binding with attributes userId and password of class `LoginComponent`.
- Directive `(click)` is used to bind `on-click` event. Method `signIn()` is called when Sign-In button is clicked.
- Directive `ngModel` is provided by inbuild `FormsModule` module. This module will be imported in `app.module.ts`.
- URL :http://localhost:4200/login

38

Define Login-route

SunilOS

- In order to access pages, define routes in `app-routing.module.ts`

```
const routes: Routes = [
  { path: 'login', component: LoginComponent},
  { path: 'welcome', component: WelcomeComponent}
];
@NgModule({
  imports: [RouterModule.forRoot(routes)]
  exports: [RouterModule]
})
```

- Access pages using following URLs
  - http://localhost:4200/login

39

Define variable

SunilOS

- Optional keyword let is used to define a variable
  - `let name = "ram" ;`
  - `let price = 100.10;`
- Optionally you can define data type of a variable. Data type is followed by variable name and separated by colon (:) character
  - `let name:string = "ram" ;`
  - `let price:number = 100.10;`
  - `let flag:Boolean = true;`
- Just like JavaScript you can alternately use `var` keyword to define a variable.
  - `var name = "ram" ;`
  - `var price = 100.10;`

40

Scope of class attributes

SunilOS

- An instance/static variable, defined in a class, is called attribute or member variable
- Scope of a variable can be `public` or `private`. Default scope is `public`

```
export class LoginComponent implements OnInit {
  public userId:string = 'Enter User ID';
  private password:string = '';
  message:string  = 'No message';
  ..
}
```

- Attributes are called inside methods using `this` keyword.

41

Define Mehods

SunilOS

- An instance/static method can be defined in a class.
- Scope of a method can be `public` or `private`. Default scope is `public`
- Here is example `signin()` method of login components:

```
signIn(){
    if(this.userId == 'admin'
          && this.password =='admin'){
      this.router.navigateByUrl('/welcome');
    }else{
      this.message = 'Invalid login id or password';
    }
  }
```

42

Static attributes and methods

- Keyword `static` is used to defined attributes and methods
  - **static** PI:number = 3.14;
- Memory is assigned only one its lifetime

```
static max(a:number, b:number){
  if(a> b){ return a; }
   else{ return b }
}
```

- Static methods are defined to use static attributes and can called with class name.

Constants

- Constant variable is defined using const keyword
- **const** PI:number = 3.14;

Constructor

- Class has only one constructor
- It is called at the time of class instantiation
- Services are injected in controller using constructor arguments:

```
export class LoginComponent implements OnInit {
  constructor(private router: Router) {
  }
}
```

- You can pass one or more arguments to constructor

Class and Interface

- Angular uses Typescripts
- TypeScript is Object Oriented
- TS provides inheritance of *classes* and implementation of *Interfaces*
- A class can inherit another class using `extents` keyword
- Interface has abstract methods
- One class may implement multiple interfaces using `implements` keyword.

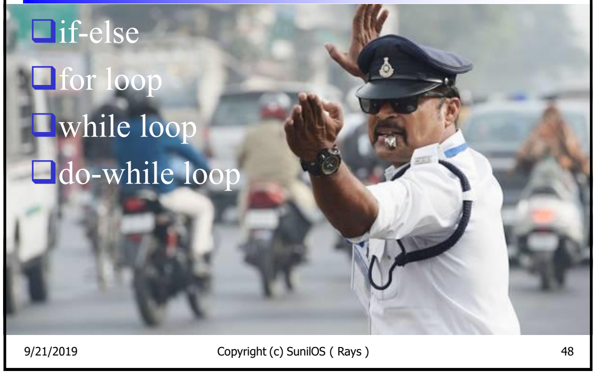
OnInit interface

- A component must have to implement `OnInit` interface

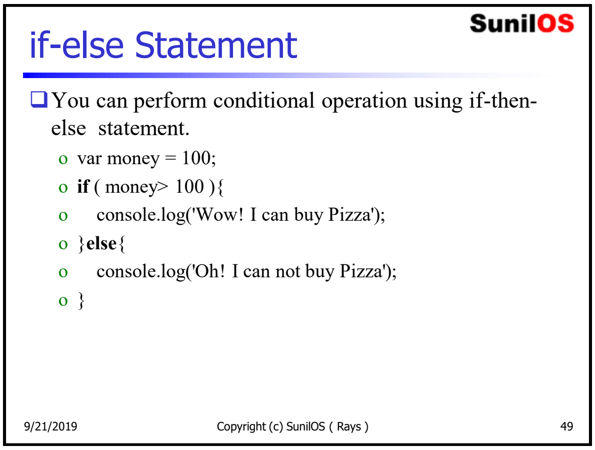
```
export class LoginComponent implements OnInit {
 ngOnInit() { .. }
}
```

- Interface `OnInit` has one abstract method `ngOnInit()`
- This method is called after component instantiation
- You may write code inside this method to initialize your component

Control Statements



- if-else
- for loop
- while loop
- do-while loop



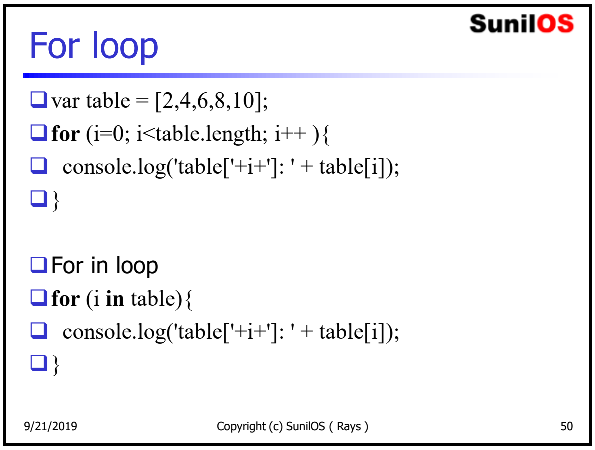
SunilOS

if-else Statement

- You can perform conditional operation using if-then-else statement.
  - o var money = 100;
  - o **if** (money> 100){
  - o   console.log('Wow! I can buy Pizza');
  - o }**else**{
  - o   console.log('Oh! I can not buy Pizza');
  - o }

9/21/2019 Copyright (c) SunilOS (Rays) 49

49



SunilOS

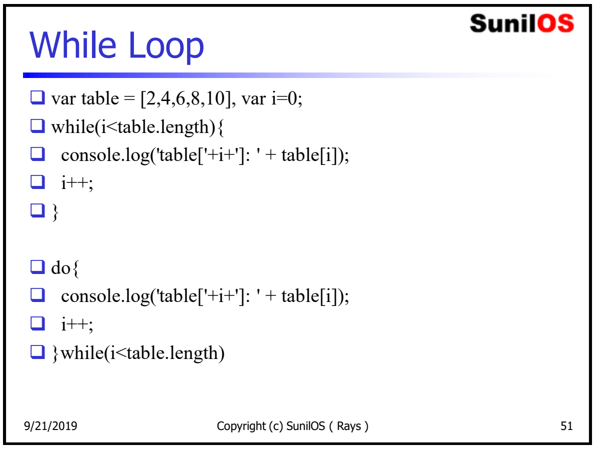
For loop

- var table = [2,4,6,8,10];
- **for** (i=0; i<table.length; i++){
-   console.log('table['+i+']: ' + table[i]);
- }

- For in loop
- **for** (i **in** table){
-   console.log('table['+i+']: ' + table[i]);
- }

9/21/2019 Copyright (c) SunilOS (Rays) 50

50



SunilOS

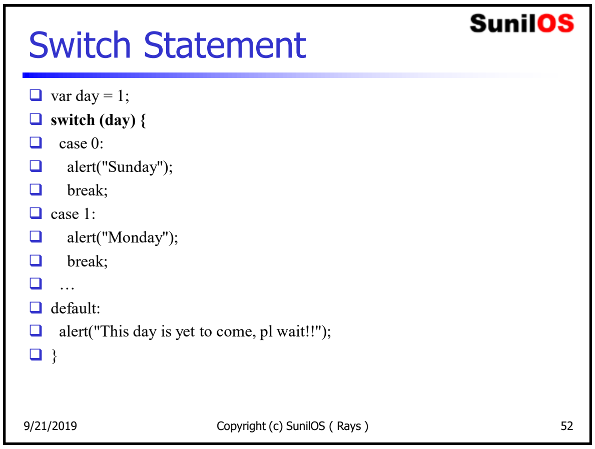
While Loop

- var table = [2,4,6,8,10], var i=0;
- while(i<table.length){
-   console.log('table['+i+']: ' + table[i]);
-   i++;
- }

- do{
-   console.log('table['+i+']: ' + table[i]);
-   i++;
- }while(i<table.length)

9/21/2019 Copyright (c) SunilOS (Rays) 51

51



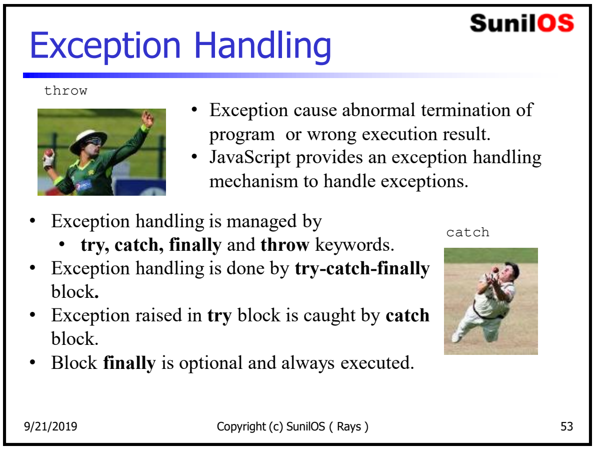
SunilOS

Switch Statement

- var day = 1;
- **switch (day)** {
-   case 0:
-     alert("Sunday");
-     break;
- case 1:
-     alert("Monday");
-     break;
-   …
- default:
-   alert("This day is yet to come, pl wait!!");
- }

9/21/2019 Copyright (c) SunilOS (Rays) 52

52



SunilOS

Exception Handling

throw

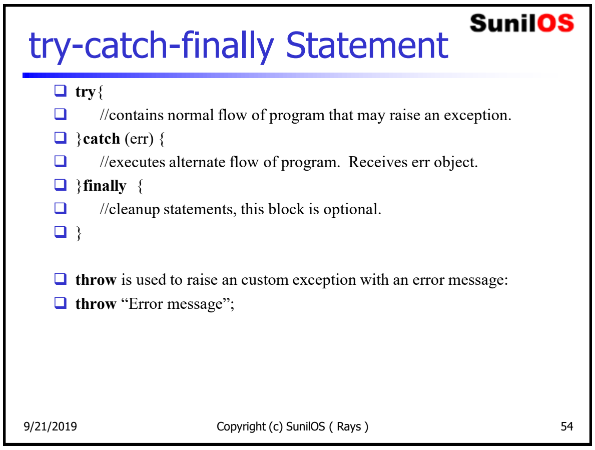
- Exception cause abnormal termination of program or wrong execution result.
- JavaScript provides an exception handling mechanism to handle exceptions.

catch

- Exception handling is managed by
  - **try, catch, finally** and **throw** keywords.
- Exception handling is done by **try-catch-finally** block**.**
- Exception raised in **try** block is caught by **catch** block.
- Block **finally** is optional and always executed.

9/21/2019 Copyright (c) SunilOS (Rays) 53

53



SunilOS

try-catch-finally Statement

- **try**{
-     //contains normal flow of program that may raise an exception.
- }**catch** (err) {
-     //executes alternate flow of program. Receives err object.
- }**finally** {
-     //cleanup statements, this block is optional.
- }

- **throw** is used to raise an custom exception with an error message:
- **throw** “Error message”;

9/21/2019 Copyright (c) SunilOS (Rays) 54

54

SunilOS

Flow of execution

- **try {**
  - **a**
  - **b //Throw Exception**
  - **c**
- **} catch (Exception e) {**
  - **d**
  - **e**
- **} finally {**
  - **f**
- **}**

❑ Normal Flow

❑ a b c f

❑ Exceptional Flow

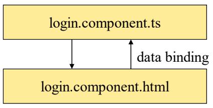
❑ a b d e f

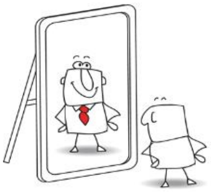
55

SunilOS

Data Binding

❑ It is data synchronization between View and Controller





56

SunilOS

Data Binding (Contd.)

❑ Data Binding can be **One-way**, where data change in controller is reflected at view, or **Two-way**, where data changes are reflected in both directions; controller and view.

❑ The following types of bindings are supported by Angular:

- One-way binding
  - Interpolation - {{attribute-name}}
  - Property Binding - [attribute-name]
- Event Binding - (event)
- Two-way binding - [(attribute-name)]

57

SunilOS

Interpolation

❑ One-way data binding is done by directive {{}}, called interpolation.

❑ Attributes defined in controller can be displayed in html using {{}}.

❑ For example, `message` attribute defined in `LoginComponent` is displayed at login html using interpolation.

❑ `export class LoginComponent implements OnInit {`

❑ `  message = 'Invalid id or password';`

❑ `}`

❑ `<H1>Login</H1>`

❑ `<p style="color:red" >{{message}}</p>`

58

SunilOS

Property Binding

❑ Property binding is used for one-way data binding

❑ It binds controller attribute with DOM property of HTML elements

❑ For example

- `<input type="text" [value]="message" >`
- `<img [src]='imageUrl'>`

- `export class LoginComponent implements OnInit {`
- `  message = 'Invalid id or password';`
- `  imageUrl = 'login.gif';`
- `}`

59

SunilOS

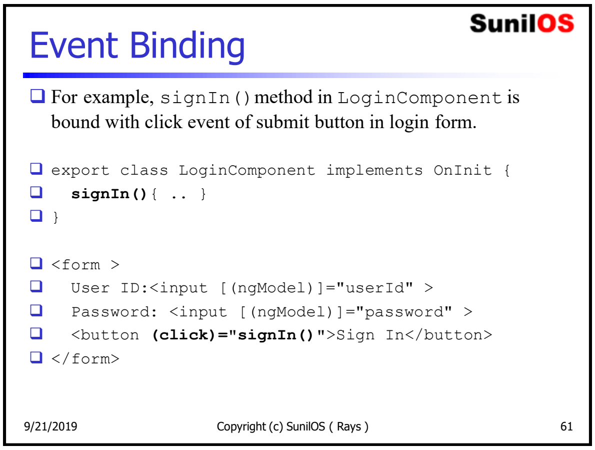
Event Binding

❑ Html form events can be bound with component class methods using **(event)** directive.

❑ Followings are form events to be bind:

- (click) : called on button click event
- (change) : called on value change event
- (keyup) : called on key up event
- (blur) : called on blur event

60



SunilOS

Event Binding

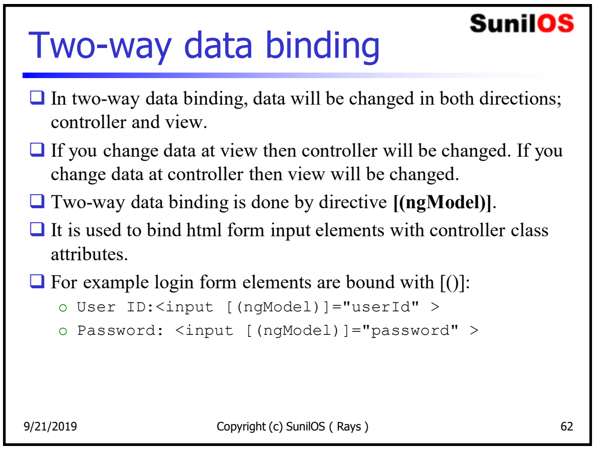
- For example, `signIn()` method in `LoginComponent` is bound with click event of submit button in login form.

```
export class LoginComponent implements OnInit {
  signIn(){ .. }
}

<form >
  User ID:<input [(ngModel)]="userId" >
  Password: <input [(ngModel)]="password" >
  <button (click)="signIn()">Sign In</button>
</form>
```

9/21/2019 Copyright (c) SunilOS (Rays) 61

61



SunilOS

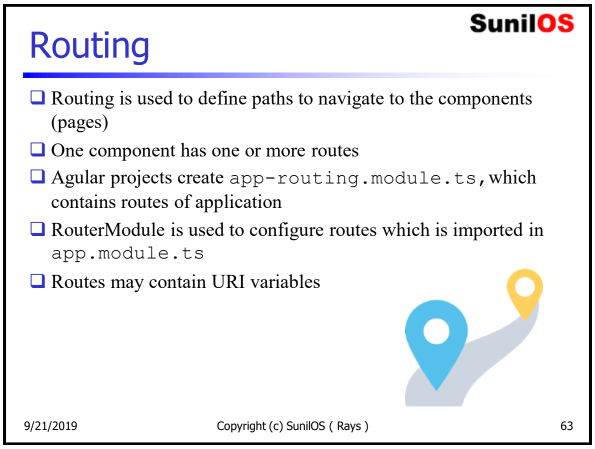
Two-way data binding

- In two-way data binding, data will be changed in both directions; controller and view.
- If you change data at view then controller will be changed. If you change data at controller then view will be changed.
- Two-way data binding is done by directive **[(ngModel)]**.
- It is used to bind html form input elements with controller class attributes.
- For example login form elements are bound with [()]:

```
User ID:<input [(ngModel)]="userId" >
Password: <input [(ngModel)]="password" >
```

9/21/2019 Copyright (c) SunilOS (Rays) 62

62



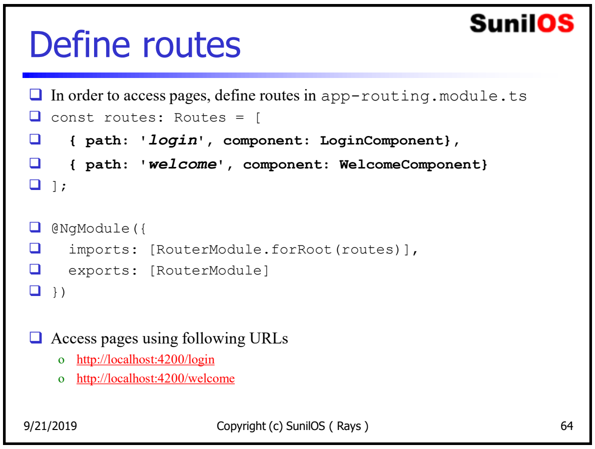
SunilOS

Routing

- Routing is used to define paths to navigate to the components (pages)
- One component has one or more routes
- Agular projects create `app-routing.module.ts`, which contains routes of application
- RouterModule is used to configure routes which is imported in `app.module.ts`
- Routes may contain URI variables

9/21/2019 Copyright (c) SunilOS (Rays) 63

63



SunilOS

Define routes

- In order to access pages, define routes in `app-routing.module.ts`

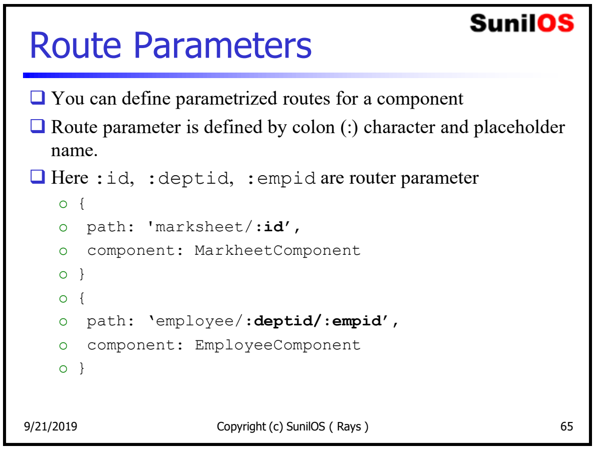
```
const routes: Routes = [
  { path: 'login', component: LoginComponent},
  { path: 'welcome', component: WelcomeComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
```

- Access pages using following URLs
  - http://localhost:4200/login
  - http://localhost:4200/welcome

9/21/2019 Copyright (c) SunilOS (Rays) 64

64



SunilOS

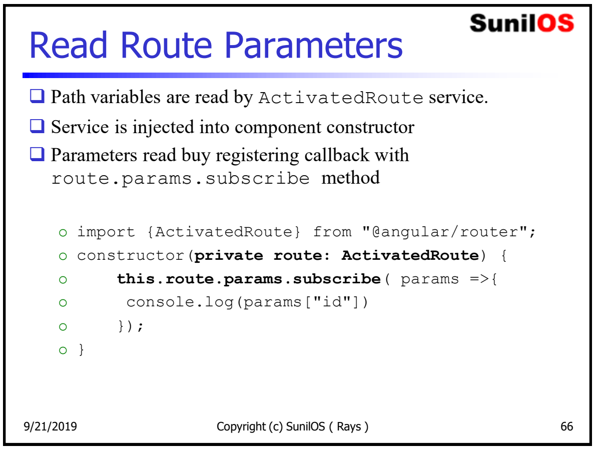
Route Parameters

- You can define parametrized routes for a component
- Route parameter is defined by colon (:) character and placeholder name.
- Here `:id`, `:deptid`, `:empid` are router parameter

```
{
 path: 'marksheet/:id’,
 component: MarkheetComponent
}
{
 path: ‘employee/:deptid/:empid’,
 component: EmployeeComponent
}
```

9/21/2019 Copyright (c) SunilOS (Rays) 65

65



SunilOS

Read Route Parameters

- Path variables are read by `ActivatedRoute` service.
- Service is injected into component constructor
- Parameters read buy registering callback with `route.params.subscribe` method

```
import {ActivatedRoute} from "@angular/router";
constructor(private route: ActivatedRoute) {
    this.route.params.subscribe( params =>{
     console.log(params["id"])
    });
}
```

9/21/2019 Copyright (c) SunilOS (Rays) 66

66

Directives

- Directives are used to change DOM structure of an html page
- Angular has many pre-defined directives such as **`*ngFor`** and **`*ngIf`**
- You can create custom directives with help of `@Directive` decorator
- There are four types of directives:
  - Components directives
  - Structural directives
  - Attribute directives
  - Custom Directive

Component Directive

- A component is also a directive-with-a-template.
- A `@Component` decorator is actually a `@Directive` decorator extended with template-oriented features.

Structural Directive

- A structure directive iterates or conditionally manipulates DOM elements.
- Structural directives have a * sign before the directive name such as **`*ngIf`** and **`*ngFor`**
- Directive **`*ngFor`** is used to iterate and print list in html page.
- Directive **`*ngIf`** is used to conditionally display an html DOM element.

*ngFor

```
export class MarksheetlistComponent implements OnInit {
  list = [
    {"id":1,"rollNo":"A1","name":"Rajesh Verma”},
    {"id":2,"rollNo":"A2","name":"Ashish Nehra"},
    {"id":3,"rollNo":"A3","name":"Manish”}
  ];
}
```

```
<table border="1">
<tr  *ngFor = "let e of list" >
    <td>{{e.id}}</td>
    <td>{{e.rollNo}}</td>
    <td>{{e.name}}</td>
</tr>
</table>
```

*ngIf

```
<p *ngIf="error" style="color:red" >
  {{message}}
</p>
```

- You can use `else` with `if` directive

```
<div *ngIf="success == true; then SUC else FAIL"></div>
<ng-template #SUC >
  <p style="color:green" >{{message}}</p>
</ng-template>
<ng-template #FAIL>
  <p style="color:red">{{message}}</p>
</ng-template>
```

Attribute Directive

- Attribute directive alter the appearance or behavior of an existing HTML element.
- Attribute directive look like regular HTML attributes.
- The `ngModel` directive, which implements two-way data binding, is an example of an attribute directive.
- `ngModel` modifies the behavior of an existing element by setting its display property and responding to the changing events.
  - `<input [(ngModel)]="movie.name">`

SunilOS

Custom Directive

- ❑ You can define your own custom directive using `@Directive` decorator.
- ❑ Custom directive can be generated by CLI command:
  - o `ng generate directive myDir`
- ❑ Above command will generate
  - o `@Directive({`
  - o `  selector: '[appMyDir]'`
  - o `})`
  - o `export class MyDirDirective {..}`

SunilOS

Pipe

- ❑ Pipes are used to format the data.
- ❑ Pipes can be used to change data from one format to another. In Agular JS it used to call filters.
- ❑ Pipe (|) character is used to apply pipe to an attribute.
- ❑ For example
  - o `{{ name | uppercase }}`
  - o `{{ name | lowercase }}`

SunilOS

Pipes

- ❑ Angular have following inbuilt pipe
  - o Lowercasepipe
  - o Uppercasepipe
  - o Datepipe
  - o Currencypipe
  - o Jsonpipe
  - o Percentpipe
  - o Decimalpipe
  - o Slicepipe
- ❑ You can create your own custom pipes

SunilOS

Pipe examples

```
<div style = "width:50%;float:left;border:solid 1px black;">
  <h1>change case pipe</h1>
  <b>{{title | uppercase}}</b><br/>
  <b>{{title | lowercase}}</b>

  <h1>Currency Pipe</h1>
  <b>{{6589.23 | currency:"USD"}}</b><br/>
  <b>{{6589.23 | currency:"USD":true}}</b>

  <h1>Date pipe</h1>
  <b>{{todaydate | date:'d/M/y'}}</b><br/>
  <b>{{todaydate | date:'shortTime'}}</b>

  <h1>Decimal Pipe</h1>
  <b>{{ 454.78787814 | number: '3.4-4' }}</b>
  // 3 is for main integer,4-4 are for integers to be
displayed
</div>
```

SunilOS

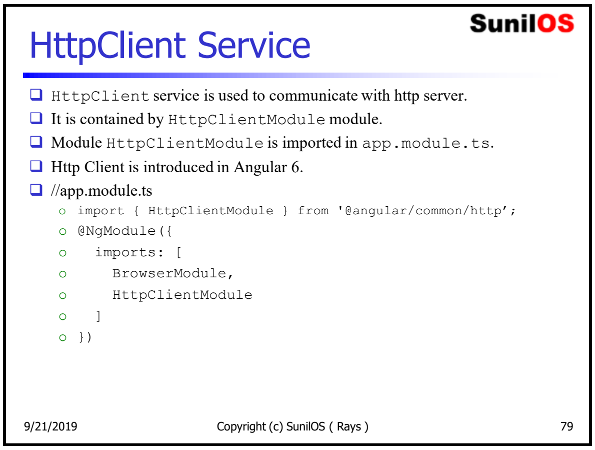
Services

- ❑ Service contains business logics and data, shared by multiple Components
- ❑ In general, services communicate with Rest Web APIs and perform CRUD operations
- ❑ Component’s controller calls service to perform business operations.
- ❑ A service can be created by following CLI command:
  - o `ng generate service UserService`
  - o Or
  - o `ng g s UserService`
- ❑ Service class is decorated by `@Injectable` decorator.
  - o **`@Injectable()`**
  - o `export class UserService {`
  - o `  constructor(private http: HttpClient) { }`
  - o `}`

SunilOS

UserService

- ❑ Lets create user service
  - o `export class` **`UserService`** `{`
  - o `  authenticate(login:string, password:string, response) {`
  - o `  ...`
  - o `  }`
  - o `}`
- ❑ Service is injected to component using constructor
  - o `export class` **`LoginComponent`** `implements OnInit {`
  - o `  public userId:string = 'Enter User ID';`
  - o `  public password:string = '';`
  - o `  constructor(`**`private service:UserService`**`) {`
  - o `  }`
  - o `}`



SunilOS

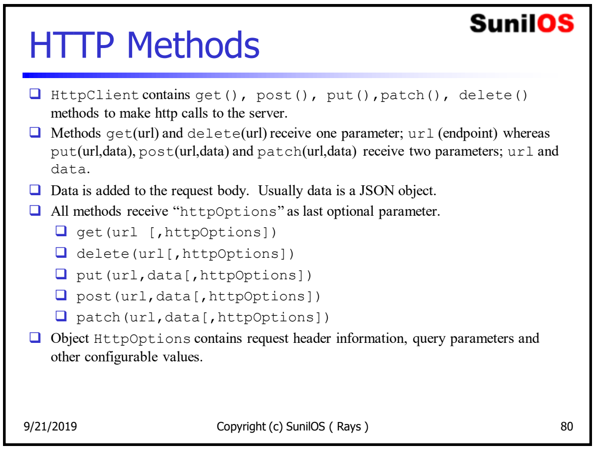
HttpClient Service

- `HttpClient` service is used to communicate with http server.
- It is contained by `HttpClientModule` module.
- Module `HttpClientModule` is imported in `app.module.ts`.
- Http Client is introduced in Angular 6.
- //app.module.ts

```
o import { HttpClientModule } from '@angular/common/http';
o @NgModule({
o   imports: [
o     BrowserModule,
o     HttpClientModule
o   ]
o })
```

9/21/2019 Copyright (c) SunilOS (Rays) 79

79



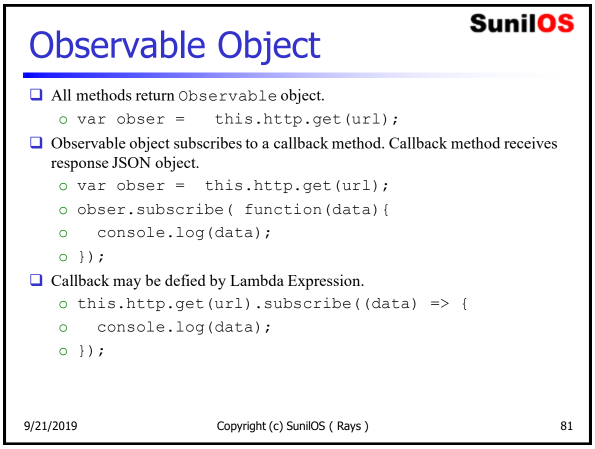
SunilOS

HTTP Methods

- `HttpClient` contains `get(), post(), put(),patch(), delete()` methods to make http calls to the server.
- Methods `get`(url) and `delete`(url) receive one parameter; `url` (endpoint) whereas `put`(url,data), `post`(url,data) and `patch`(url,data) receive two parameters; `url` and `data`.
- Data is added to the request body. Usually data is a JSON object.
- All methods receive "`httpOptions`" as last optional parameter.
  - `get(url [,httpOptions])`
  - `delete(url[,httpOptions])`
  - `put(url,data[,httpOptions])`
  - `post(url,data[,httpOptions])`
  - `patch(url,data[,httpOptions])`
- Object `HttpOptions` contains request header information, query parameters and other configurable values.

9/21/2019 Copyright (c) SunilOS (Rays) 80

80



SunilOS

Observable Object

- All methods return `Observable` object.

```
o var obser =   this.http.get(url);
```

- Observable object subscribes to a callback method. Callback method receives response JSON object.

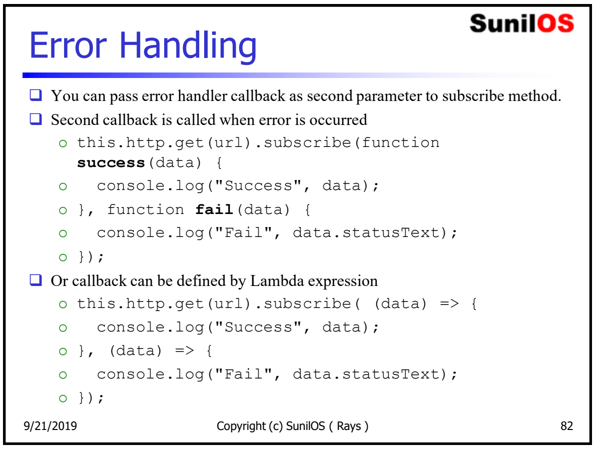
```
o var obser =  this.http.get(url);
o obser.subscribe( function(data){
o   console.log(data);
o });
```

- Callback may be defied by Lambda Expression.

```
o this.http.get(url).subscribe((data) => {
o   console.log(data);
o });
```

9/21/2019 Copyright (c) SunilOS (Rays) 81

81



SunilOS

Error Handling

- You can pass error handler callback as second parameter to subscribe method.
- Second callback is called when error is occurred

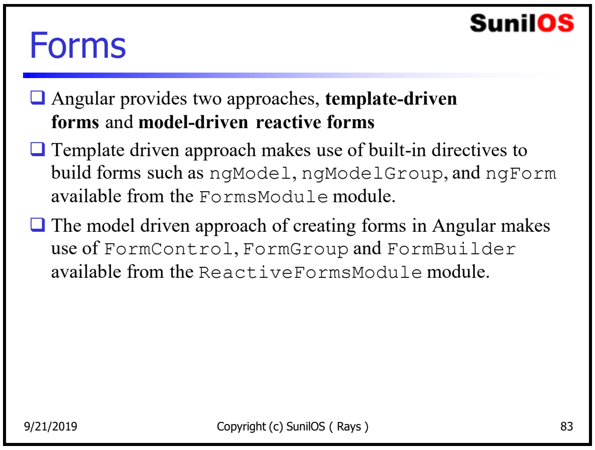
```
o this.http.get(url).subscribe(function
  success(data) {
o   console.log("Success", data);
o }, function fail(data) {
o   console.log("Fail", data.statusText);
o });
```

- Or callback can be defined by Lambda expression

```
o this.http.get(url).subscribe( (data) => {
o   console.log("Success", data);
o }, (data) => {
o   console.log("Fail", data.statusText);
o });
```

9/21/2019 Copyright (c) SunilOS (Rays) 82

82



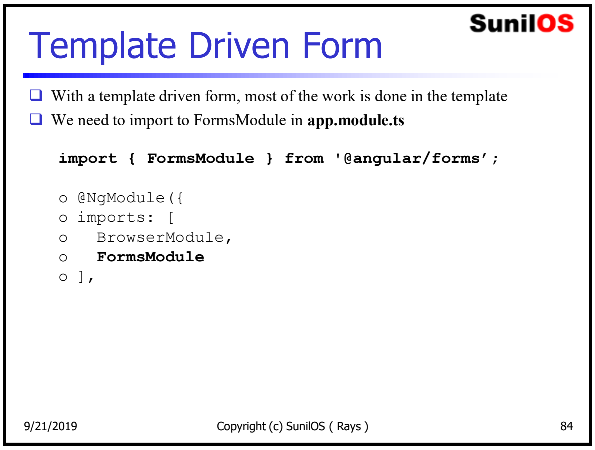
SunilOS

Forms

- Angular provides two approaches, **template-driven forms** and **model-driven reactive forms**
- Template driven approach makes use of built-in directives to build forms such as `ngModel`, `ngModelGroup`, and `ngForm` available from the `FormsModule` module.
- The model driven approach of creating forms in Angular makes use of `FormControl`, `FormGroup` and `FormBuilder` available from the `ReactiveFormsModule` module.

9/21/2019 Copyright (c) SunilOS (Rays) 83

83



SunilOS

Template Driven Form

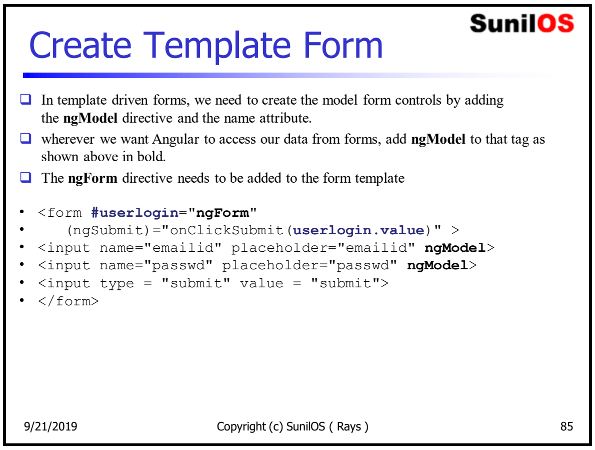
- With a template driven form, most of the work is done in the template
- We need to import to FormsModule in **app.module.ts**

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

o @NgModule({
o imports: [
o   BrowserModule,
o   FormsModule
o ],
```

9/21/2019 Copyright (c) SunilOS (Rays) 84

84



Create Template Form

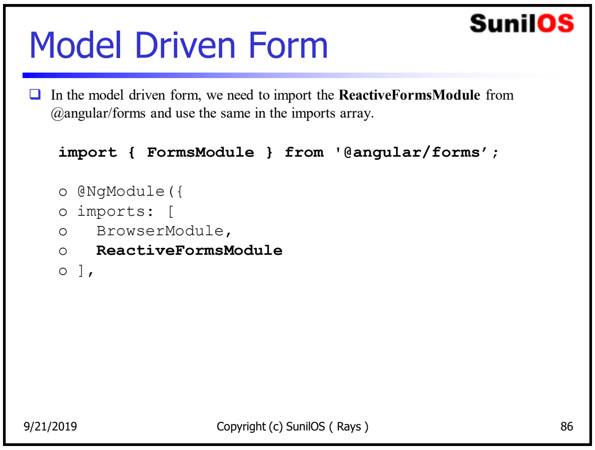
SunilOS

- ❑ In template driven forms, we need to create the model form controls by adding the **ngModel** directive and the name attribute.
- ❑ wherever we want Angular to access our data from forms, add **ngModel** to that tag as shown above in bold.
- ❑ The **ngForm** directive needs to be added to the form template

```
• <form #userlogin="ngForm"
•    (ngSubmit)="onClickSubmit(userlogin.value)" >
• <input name="emailid" placeholder="emailid" ngModel>
• <input name="passwd" placeholder="passwd" ngModel>
• <input type = "submit" value = "submit">
• </form>
```

9/21/2019 Copyright (c) SunilOS (Rays) 85

85



Model Driven Form

SunilOS

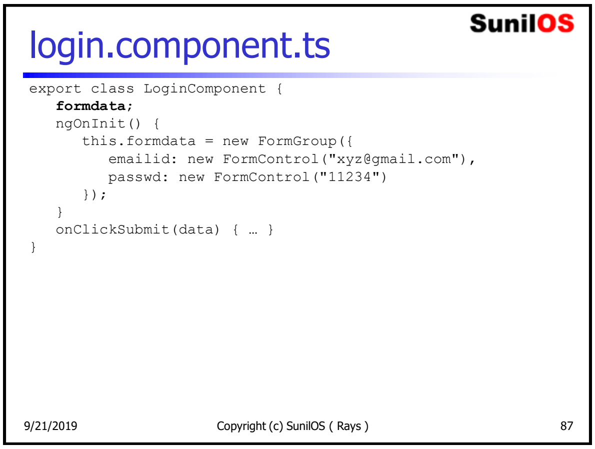
- ❑ In the model driven form, we need to import the **ReactiveFormsModule** from @angular/forms and use the same in the imports array.

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

o @NgModule({
o imports: [
o   BrowserModule,
o   ReactiveFormsModule
o ],
```

9/21/2019 Copyright (c) SunilOS (Rays) 86

86



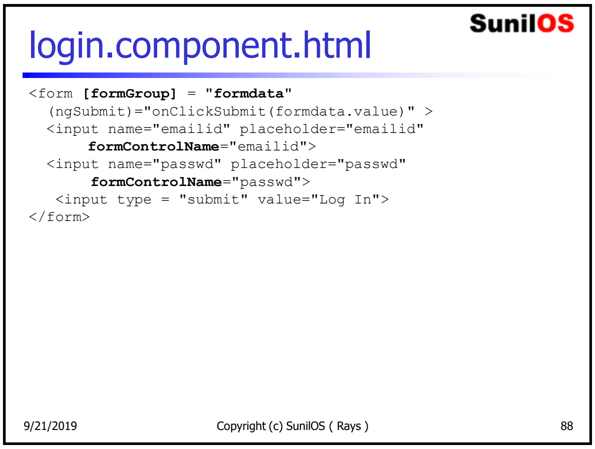
login.component.ts

SunilOS

```
export class LoginComponent {
   formdata;
   ngOnInit() {
      this.formdata = new FormGroup({
         emailid: new FormControl("xyz@gmail.com"),
         passwd: new FormControl("11234")
      });
   }
   onClickSubmit(data) { … }
}
```

9/21/2019 Copyright (c) SunilOS (Rays) 87

87



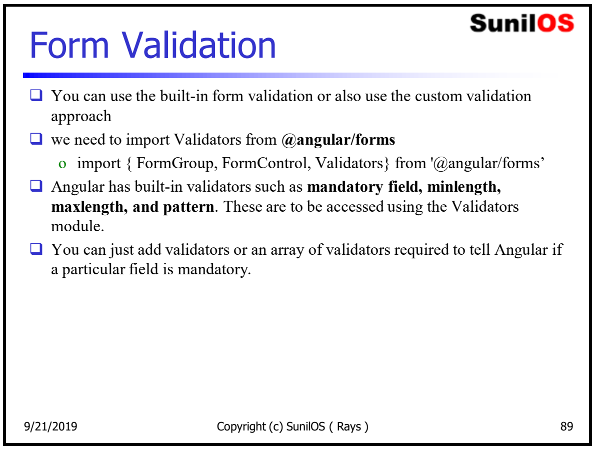
login.component.html

SunilOS

```
<form [formGroup] = "formdata"
  (ngSubmit)="onClickSubmit(formdata.value)" >
  <input name="emailid" placeholder="emailid"
       formControlName="emailid">
  <input name="passwd" placeholder="passwd"
       formControlName="passwd">
   <input type = "submit" value="Log In">
</form>
```

9/21/2019 Copyright (c) SunilOS (Rays) 88

88



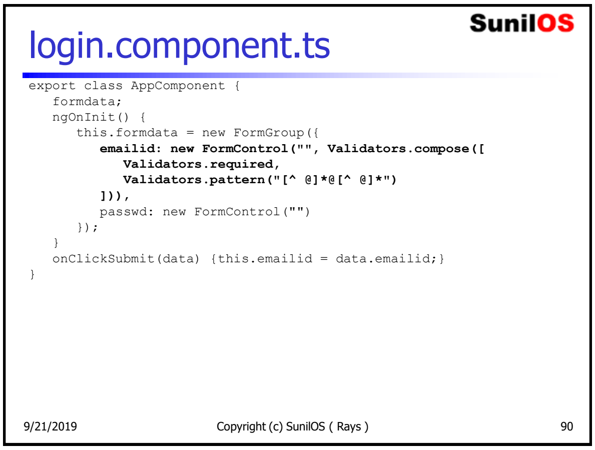
Form Validation

SunilOS

- ❑ You can use the built-in form validation or also use the custom validation approach
- ❑ we need to import Validators from **@angular/forms**
  - o import { FormGroup, FormControl, Validators} from '@angular/forms'
- ❑ Angular has built-in validators such as **mandatory field, minlength, maxlength, and pattern**. These are to be accessed using the Validators module.
- ❑ You can just add validators or an array of validators required to tell Angular if a particular field is mandatory.

9/21/2019 Copyright (c) SunilOS (Rays) 89

89



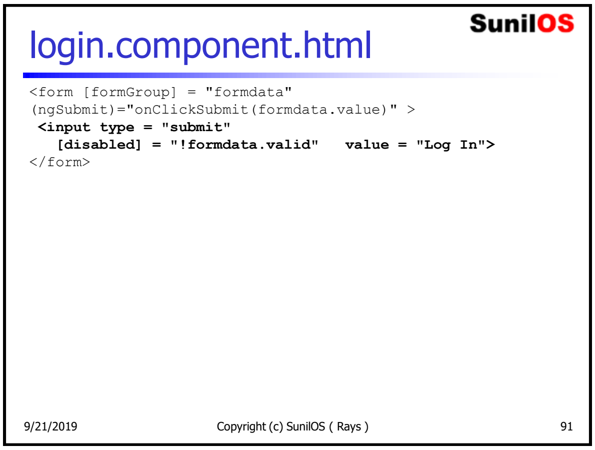
login.component.ts

SunilOS

```
export class AppComponent {
   formdata;
   ngOnInit() {
      this.formdata = new FormGroup({
         emailid: new FormControl("", Validators.compose([
            Validators.required,
            Validators.pattern("[^ @]*@[^ @]*")
         ])),
         passwd: new FormControl("")
      });
   }
   onClickSubmit(data) {this.emailid = data.emailid;}
}
```

9/21/2019 Copyright (c) SunilOS (Rays) 90

90

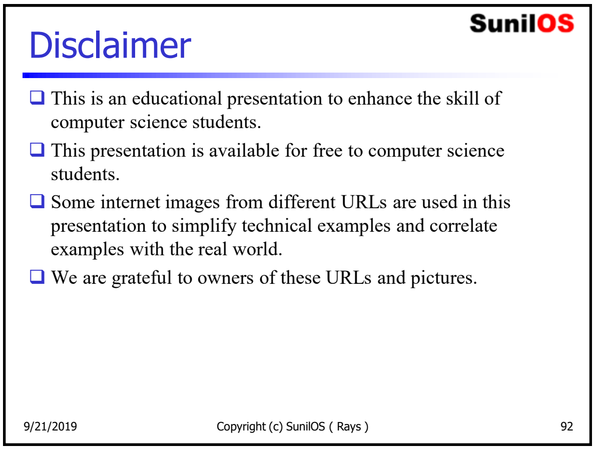


login.component.html

```
<form [formGroup] = "formdata"
(ngSubmit)="onClickSubmit(formdata.value)" >
 <input type = "submit"
   [disabled] = "!formdata.valid"   value = "Log In">
</form>
```

9/21/2019 Copyright (c) SunilOS (Rays) 91

91

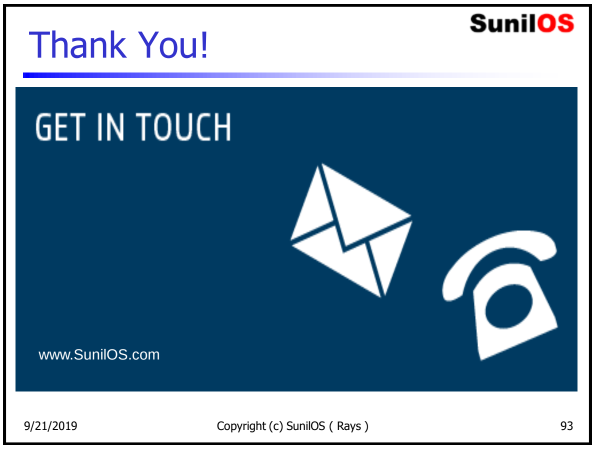


Disclaimer

- This is an educational presentation to enhance the skill of computer science students.
- This presentation is available for free to computer science students.
- Some internet images from different URLs are used in this presentation to simplify technical examples and correlate examples with the real world.
- We are grateful to owners of these URLs and pictures.

9/21/2019 Copyright (c) SunilOS (Rays) 92

92



Thank You!

GET IN TOUCH

www.SunilOS.com

9/21/2019 Copyright (c) SunilOS (Rays) 93

93