1604C054 – Hybrid Mobile Programming

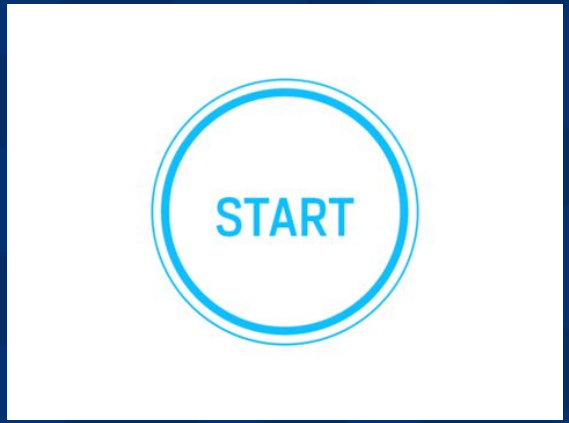
Ionic Basic, Routing, and Navigation

WEEK 02

Informatics Engineering
Universitas Surabaya

Outline

1. Preparation
2. Angular Route
3. Tabs Navigation
4. Drawer Navigation



Preparation

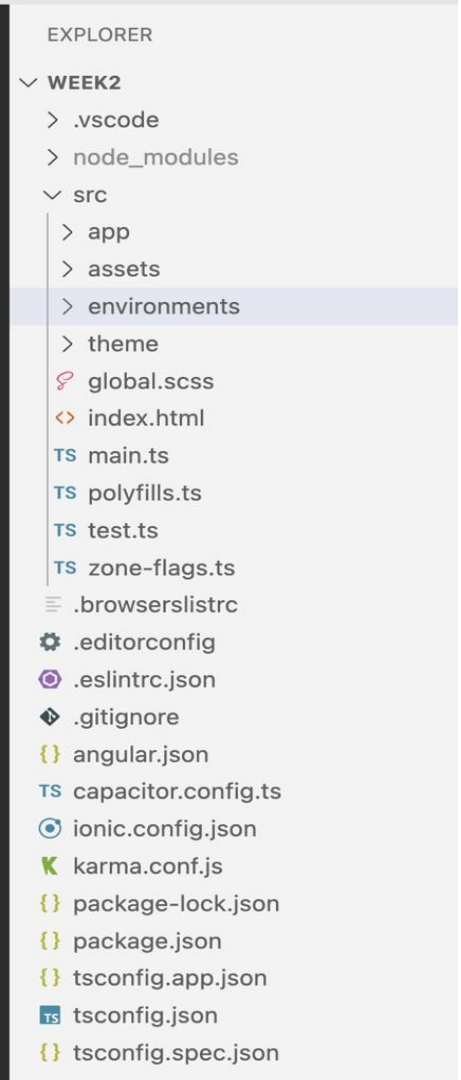
Preparation

Please create a new project with blank template
or copy from **https://ubaya.id/hybrid_week2_source**

In VS Code, add extension :

- Ionic. Official extension for Ionic and Capacitor development from Ionic.co
- Angular Snippet. Angular version 16 snippets by John Papa

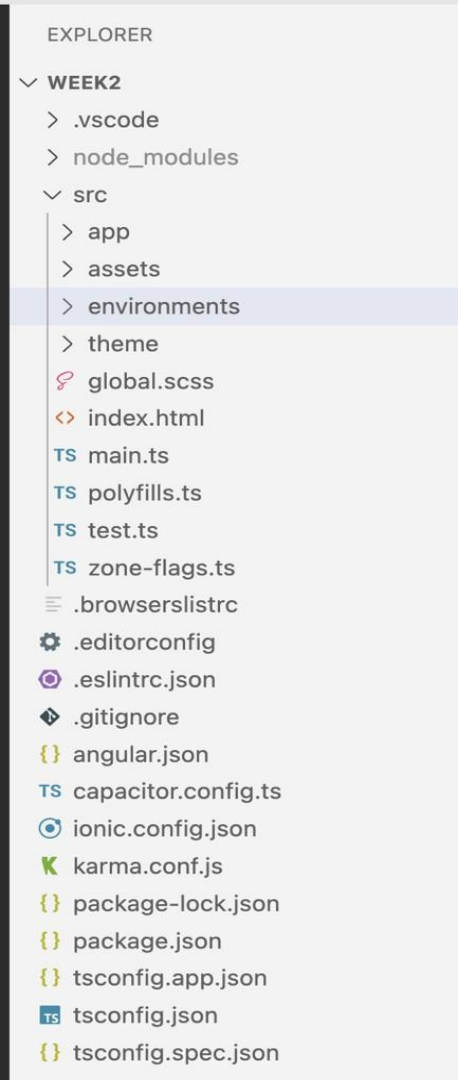
Project Structure



node_modules: This folder contains all the external packages and libraries (dependencies) that our Ionic Angular project relies on. This includes Angular, Ionic components, and any other third-party libraries you may add to our project. This folder is automatically generated and managed by the package manager (npm) when you install project dependencies.

src: The src folder is where you will spend most of your time developing your Ionic app. It contains the source code of your application, including Angular components, services, modules, styles, and assets. You'll create and organize your app's structure and content within this folder.

Project Structure (2)



package.json: This file contains metadata about your project and a list of dependencies (both development and production). It's used by npm or Yarn to install and manage project dependencies. You can also define scripts for common tasks like building, testing, and running your app.

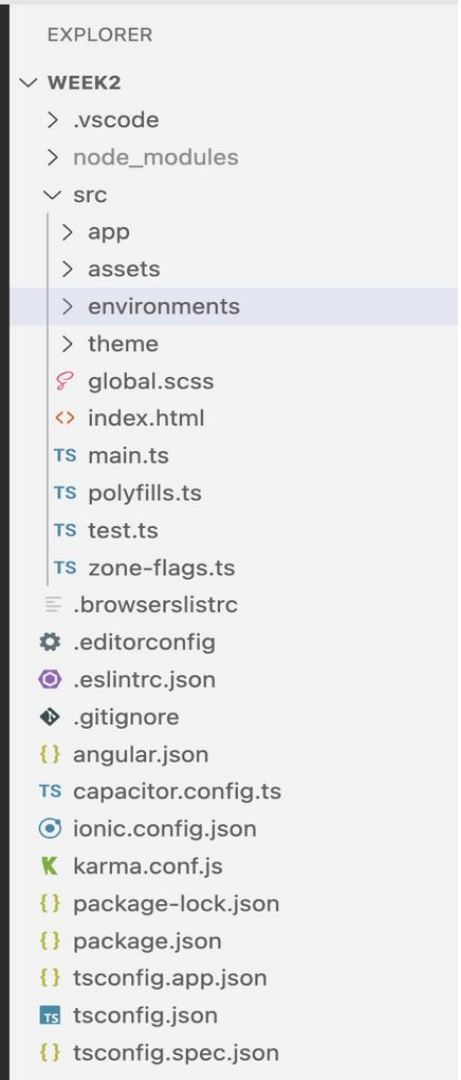
package-lock.json (**or yarn.lock**): These files are automatically generated and help ensure that the exact versions of dependencies are installed across different environments. They are used by npm or Yarn to lock down dependency versions.

tsconfig.json: This file is used to configure TypeScript settings for your project. It defines how TypeScript compiles your code and includes important settings like module resolution, target version, and more.

angular.json: This file is specific to Angular CLI and contains configuration settings for your Angular project. It includes build options, development server configuration, and more.

ionic.config.json (or ionic.config.js): This file contains configuration settings specific to the Ionic framework. It may include settings related to plugins, Cordova, and other Ionic-specific options.

Project Structure (3)



src/app : The app folder is the heart of your Ionic Angular application. It contains the core components, modules, services, and other files that make up your app's functionality.

src/assets : While there's often an assets folder in the root directory of your project for static assets, you might also have an assets folder within the src directory. This can be used for additional assets specific to your app's code.

src/environments : This folder typically contains environment configuration files, such as environment.ts and environment.prod.ts. These files allow you to define environment-specific settings, like API endpoints or debug flags.

src/theme : If you want to customize the styles and theming of your Ionic app, you can create a theme folder to store Sass/SCSS files for styling and theming.

How ionic-angular page work ?

The ionic-angular page start from **app-root** element in **index.html**.

```
22  <body>
23    <app-root></app-root>
24  </body>
25
```

The app-root element not explicitly defined in the HTML file itself. Instead, it is dynamically created and inserted into the DOM by the Angular framework during the bootstrap process.

The bootstraping process call **bootstrapModule** In the main.ts file (located in the src folder). here is code that calls the platformBrowserDynamic().bootstrapModule(AppModule) method to bootstrap your Angular application. The **AppModule** is the main module of your application.

The AppModule located at **app.module.ts** declaring AppComponent as the bootstrap

```
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule],
  providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

...about component (also page)

Before continue to AppComponent, we have to know the structure of component (and also page).

There are 4 files in an angular components :

✔ **.component.ts**
The TypeScript file contains the page component class, which defines the page's behavior and logic.

✔ **.component.html**
This HTML file defines the structure and layout of the component's view.

✔ **Typically .css, .scss, .less, etc.**
This file contains the component-specific styles (CSS or preprocessor syntax) that define the appearance and layout of the component's view.

✔ **.component.spec.ts**
This TypeScript file contains unit tests for the component, ensuring that it behaves correctly under various conditions.

How ionic-angular page work (2)

The **app.component** is the root component of our Angular application. It's defined in the **app.component.ts** file and typically has an associated HTML template (**app.component.html**).

The **app.component** is the first component loaded when your application starts. It's responsible for defining the overall layout and structure of our application's user interface.

In the app.component.html template, in blank starter, we find a **<router-outlet></router-outlet>** element. This is a placeholder that the Angular Router uses to display the content of routed components. It acts like a viewport for the routed views.

src > app > app.component.html > ion-app

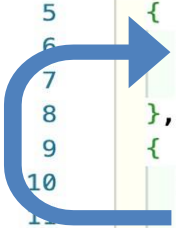
```
1   <ion-app>
2     <ion-router-outlet></ion-router-outlet>
3   </ion-app>
4
```

How ionic-angular page work (3)

In your Angular application, you typically have a routing configuration defined in a module, often named AppRoutingModule (**app-routing.module**).
This routing configuration specifies the mapping between URL routes and the components that should be displayed when those routes are navigated to.

src > app > TS app-routing.module.ts > ...

```
1   import { NgModule } from '@angular/core';
2   import { PreloadAllModules, RouterModule, Routes } from '@angular/router';
3
4   const routes: Routes = [
5     {
6       path: 'home',
7       loadChildren: () => import('./home/home.module').then( m => m.HomePageModule)
8     },
9     {
10      path: '',
11      redirectTo: 'home',
12      pathMatch: 'full'
13    },
14  ];
15
```



How ionic-angular page work (4)

Between app.component and **app-routing.module**

1. When you run your Angular application, the **app.component** is loaded first, and its associated HTML template is displayed.
2. As users navigate to different routes (e.g., by clicking links or entering URLs), the Angular Router intercepts these navigation events and checks the routing configuration to determine which component should be displayed.
3. When a route is matched, the associated component is instantiated and its template is rendered within the **<router-outlet></router-outlet>** of the **app.component**.

In summary, the **app.component** serves as the root of your application's UI and provides a placeholder (**<router-outlet>**) for displaying the content of routed components. Routing configuration determines which component to load and display when specific URLs are accessed. This setup allows you to build a single-page application with multiple views (pages) that are loaded dynamically as users navigate through the application.

Opening the home

based on the route
it will open the **home.module**

```
path: 'home',
loadChildren: () => import('./home/home.module').then( m => m.HomePageModule)
```

Angular encourages the use of feature modules to organize and encapsulate related functionality within your application. Each feature module typically consists of its own components, services, and routing configuration.

home.module.ts (Feature Module):

- The home.module.ts file is a feature module specifically dedicated to the home page. It allows you to encapsulate all the components, services, and other features related to the home page in a self-contained module.
- Within the home.module.ts file, you can declare the HomePage component, import and configure any necessary dependencies, and provide services or other resources specific to the home page.
- Feature modules like this help keep your code organized, maintainable, and easier to scale as your application grows.

Opening the home (2)

home-routing.module.ts (Routing Configuration):

- The home-routing.module.ts file is used to define the routing configuration for the home page. It specifies how the home page is accessed and what component should be displayed when the user navigates to its route.
- This separation of routing configuration into a dedicated module allows for better organization and makes it easier to manage routes for specific parts of your application.
- It's a common practice to create a routing module for each feature module, which keeps your routing logic modular and easier to maintain.

in the routing :

related with component name in home.page

```
const routes: Routes = [
  {
    path: '',
    component: HomePage,
  }
];
```

```
> app > home > TS home.page.ts > ...
import { Component } from '@ang

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html'
  styleUrls: ['home.page.scss']
})
expor    HomePage {

  constructor() {}

}
```

Opening the home (3)

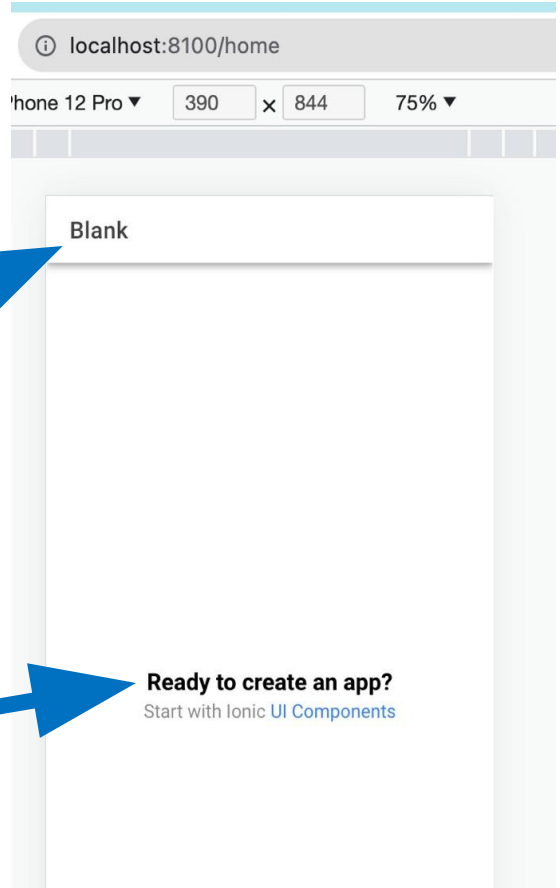
Therefore the User Interface that will display is in **home.page.html**

app > home > home.page.html > ion-header

```
<ion-header [translucent]="true">
  <ion-toolbar>
    <ion-title>
      Blank
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content [fullscreen]="true">
  <ion-header collapse="condense">
    <ion-toolbar>
      <ion-title size="large">Blank</ion-title>
    </ion-toolbar>
  </ion-header>

  <div id="container">
    <strong>Ready to create an app?</strong>
    <p>Start with Ionic <a target="_blank" rel="noc
  </div>
</ion-content>
```



Angular Component vs Page

In Angular, both components and pages are building blocks for creating the user interface of a web application, and they share many similarities. However, there are some key differences between the two:

Role and Purpose:

Component: A component is a reusable, self-contained piece of UI functionality. It can represent various parts of a user interface, such as a button, form, header, or a widget. Components are designed to be highly reusable and can be used in different parts of an application.

Page: A page, on the other hand, typically represents a top-level view or route in an application. It is a specific type of component that is often associated with a particular URL route. Pages are used to structure the application's navigation and present content that is meant to be displayed as a standalone view.

Routing:

Component: Components are not directly associated with routing. They can be used on multiple pages or views, and they don't have their own routes.

Page: Pages are usually associated with specific routes in an application. When you navigate to a particular URL route, the corresponding page component is loaded and displayed.

Usage:

Component: Components can be used anywhere within an application, including within other components, pages, or even other components. They are often used to encapsulate and reuse UI functionality.

Page: Pages are typically used as the main content views that are loaded when you navigate to specific routes. They represent distinct sections or screens of an application.

Angular Component vs Page (2)

Responsibilities:

Component: Components can have various responsibilities, depending on their specific use case. They can handle user input, display data, manage state, and interact with services or other components.

Page: Pages primarily focus on presenting content and managing the high-level structure of a view. They may coordinate the display of multiple components and handle routing-related logic.

Navigation:

Component: Components do not handle navigation directly. They rely on routing and navigation services provided by Angular for navigation between different parts of an application.

Page: Pages are designed to be navigated to directly through routing. They often serve as entry points to specific sections of an application.

Scope:

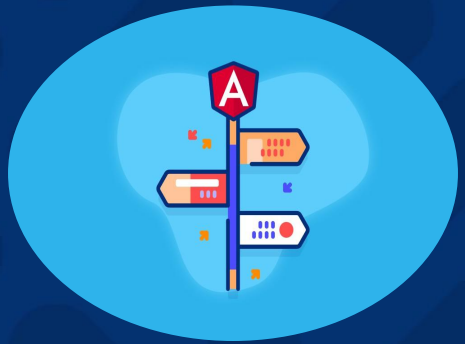
Component: The scope of a component is typically narrower, and it focuses on a specific UI element or feature.

Page: The scope of a page is broader, encompassing a complete view or screen within the application.

Examples:

Component: Examples of components include buttons, input fields, date pickers, dropdown menus, and custom widgets.

Page: Examples of pages include a home page, login page, product details page, and user profile page.



Angular Route

Creating New Page

We will create new page and open it with a link in the home page.

If the ionic is served, stop it by press **Ctrl + C** . then type this to create about page:

ionic generate page about

Folder about is created. 6 files is added in this folder, and app- routing.module.ts is updated

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS

(base) MacDaniel:week2 mac$ ionic generate page about
> ng generate page about --project=app
CREATE src/app/about/about-routing.module.ts (343 bytes)
CREATE src/app/about/about.module.ts (465 bytes)
CREATE src/app/about/about.page.scss (0 bytes)
CREATE src/app/about/about.page.html (300 bytes)
CREATE src/app/about/about.page.spec.ts (446 bytes)
CREATE src/app/about/about.page.ts (252 bytes)
UPDATE src/app/app-routing.module.ts (607 bytes)
[OK] Generated page!
(base) MacDaniel:week2 mac$
```

```
const routes: Routes = [
  {
    path: 'home',
    loadChildren: () => import('./home/home.module').then( m => m.HomePageModule)
  },
  {
    path: '',
    redirectTo: 'home',
    pathMatch: 'full'
  },
  {
    path: 'about',
    loadChildren: () => import('./about/about.module').then( m => m.AboutPageModule)
  },
];
```

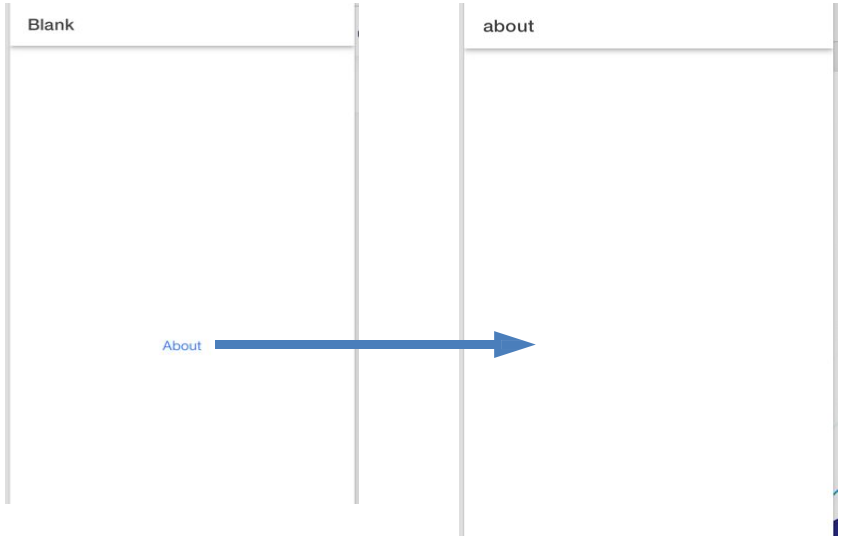
Creating The Link

Open home.page.html . Add :

```
<a routerLink="/about">About</a>
```

and serve again

routerLink is a directive in Angular that is used to create navigation links in your application. It's typically used with anchor (<a>) elements to define clickable links that trigger navigation to different routes within your Angular application. However, you can also use it with other interactive elements like buttons and divs.



Why There Are Many Routing File?

In an Angular application with multiple pages or views, you often configure routing in both the **app- routing.module.ts** (the root routing module) and individual routing modules for each feature or page, such as the **about-routing.module.ts**. Here's a breakdown of the differences and purposes of each:

Root Routing (app-routing.module.ts):

- **Entry Point**: The root routing module (app-routing.module.ts) is the entry point for defining the top-level routing configuration for your entire application. It contains the base routing setup that applies to all parts of your app.
- **Global Routes**: This module typically includes routes that are accessible from any part of your application, such as a home page or a shared error page. These routes are accessible globally and are often defined at the root level.
- **Layout Structure**: It might also define the overall layout structure of your application, such as a common header or footer that appears on all pages.
- **Outlet**: The <router-outlet></router-outlet> directive is often placed in the root component's template (e.g., app.component.html). This serves as the main content area where routed components will be displayed.
- **Fallback Routes**: You can define fallback or wildcard routes in the root routing module to handle cases where the user enters an invalid URL or an unmatched route.

Why There Are Many Routing File? (2)

Feature-Specific Routing (about-routing.module.ts):

- **Scoped Routes**: Feature-specific routing modules, like about-routing.module.ts, are used to define routes specific to a particular feature, section, or page of your application.
- **Isolation**: They encapsulate the routing configuration for the feature, which keeps the code modular and makes it easier to manage routing for that feature.
- **Child Routes**: Feature-specific routing modules can define child routes within the context of the feature. For example, if the "About" page has subpages, you can define those routes here.
- **Route Configuration**: These modules are responsible for specifying the routes that are unique to the feature. They specify the URLs and the associated components that should be displayed when those URLs are accessed.
- **Lazy Loading**: If you have a large application, you might use feature-specific routing modules in combination with lazy loading. This means that the feature's code is loaded only when the user accesses that part of the application, improving initial load times.

Why There Are Many Routing File? (3)

Example Scenario:

Consider a scenario where you have a home page and an "**About**" page in your application. You would typically configure the routes as follows:

- In **app-routing.module.ts**, you define routes that apply globally to your entire application. For example:
  - '**home**' route maps to the home page component.
  - '**about**' route maps to the "About" page component.
- In about-routing.module.ts, you define additional routes specific to the "**About**" page, such as subpages like **'about/team'** and **'about/history**'. These routes are scoped to the "**About**" feature. This separation of routing configuration allows for a modular and organized approach to managing routing in your Angular application, making it easier to maintain as your app grows. The root routing module handles global routing concerns, while feature-specific routing modules handle the routing within their respective features.

Why There Are Many Routing File? (4)

Lets try it !

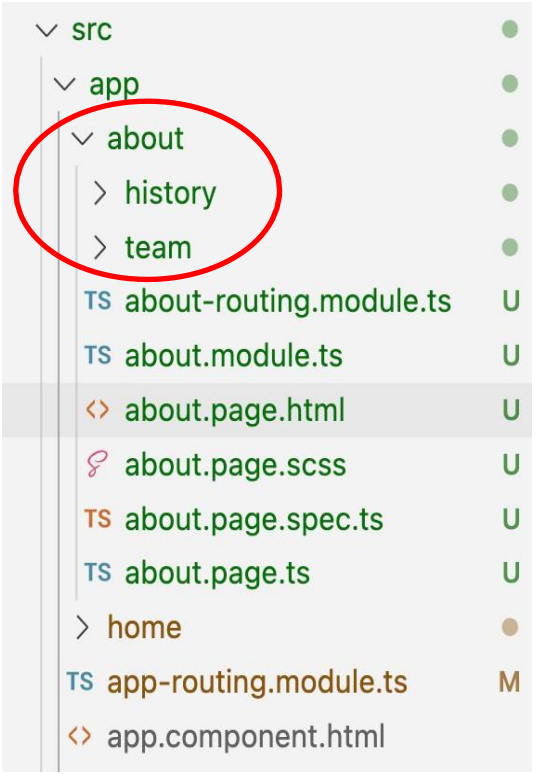
Create new page inside about folder :

ionic generate page about/team

ionic generate page about/history

and add link in about page

```
<a routerLink="team">Team</a><br>
<a routerLink="history">History</a>
```



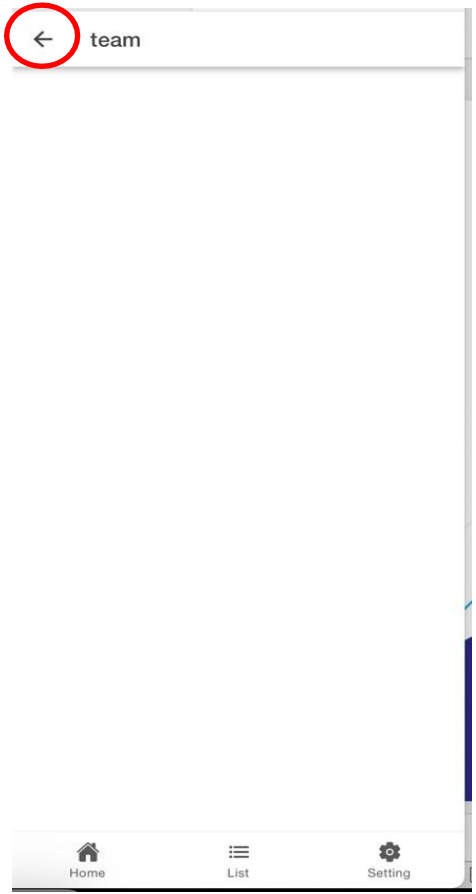
Back Button

If we want to add back button in the tab bar, we can add :

```
<ion-buttons slot="start">
  <ion-back-button></ion-back-button>
</ion-buttons>
```

```
<ion-buttons slot="start">
    <ion-back-button defaultHref="/home"></ion-back-button>
</ion-buttons>
```

Add this attribute if the back button not showing, set the defaultHref value into the desired route.

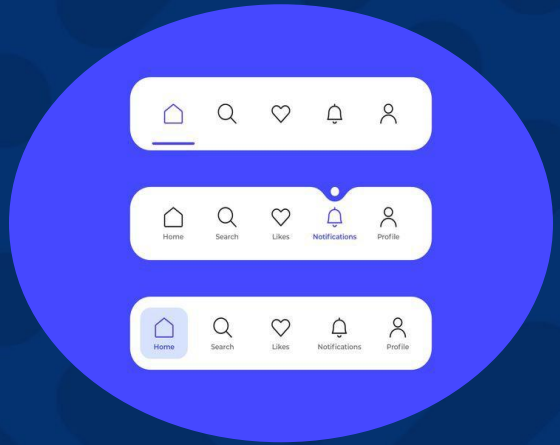


Route With Parameter

Routing with parameters allows you to pass data from one page to another when navigating within your app. This is useful for tasks like displaying specific details for an item, searching for content, or customizing the behavior of a page based on user input. Ionic Angular provides a straightforward way to work with route parameters.

```
const routes: Routes = [
  {
    path: 'details/:id', // :id is a parameter
    loadChildren: () => import('./details/details.module').then(m =>
m.DetailsPageModule)
  },
  // Other routes...
];
```

We keep this 'route with parameter' for a while, and continue it after practicing angular typescript code.



Tabs Navigation

Adding Tabs

Add in app.component.html

```
<!-- tabs.page.html -->
<ion-tabs>
  <ion-tab-bar slot="bottom">
    <ion-tab-button tab="home">
      <ion-icon name="home"></ion-icon>
      <ion-label>Home</ion-label>
    </ion-tab-button>

    <ion-tab-button tab="list">
      <ion-icon name="list"></ion-icon>
      <ion-label>List</ion-label>
    </ion-tab-button>

    <ion-tab-button tab="setting">
      <ion-icon name="settings"></ion-icon>
      <ion-label>Settings</ion-label>
    </ion-tab-button>
  </ion-tab-bar>
</ion-tabs>
```

You should already have list page and setting page.

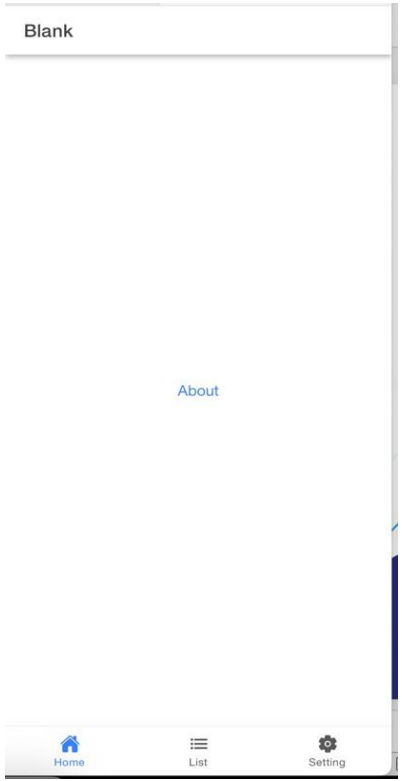
If not, please generate it !

Add Routing

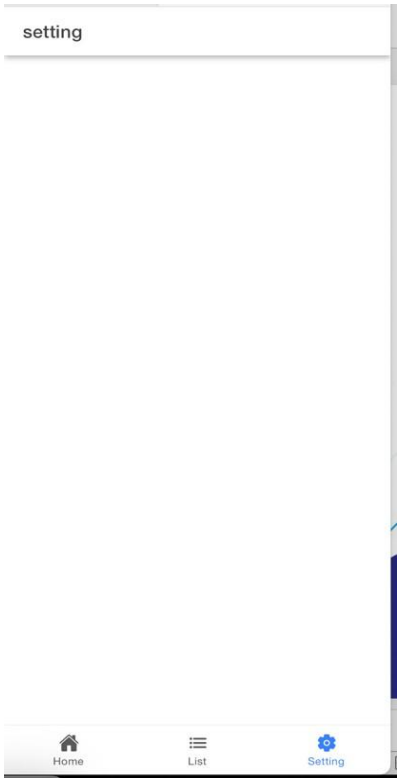
Add in **app-routing.module.ts**

```
{
path:
'tabs',
children: [
{ path: 'home', loadChildren: () => import('./home/home.module').then(m =>
m.HomePageModule) },
{ path: 'list', loadChildren: () => import('./list/list.module').then(m =>
m.ListPageModule) },
{ path: 'setting', loadChildren: () => import('./setting/setting.module').then(m =>
m.SettingPageModule) },
]
},
```

Result

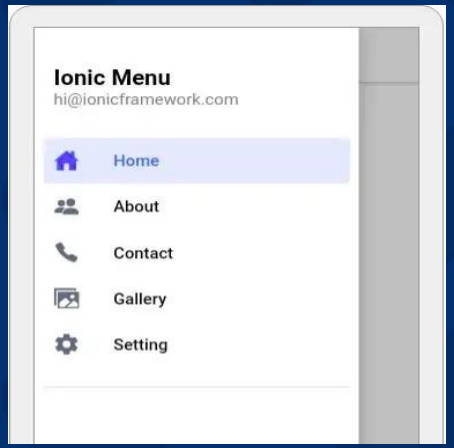

Blank
About
Home
List
Setting


list
Home
List
Setting


setting
Home
List
Setting

Ion Icon

ion-icon list : https://ionic.io/ionicons/



Drawer Navigation

Adding Drawer

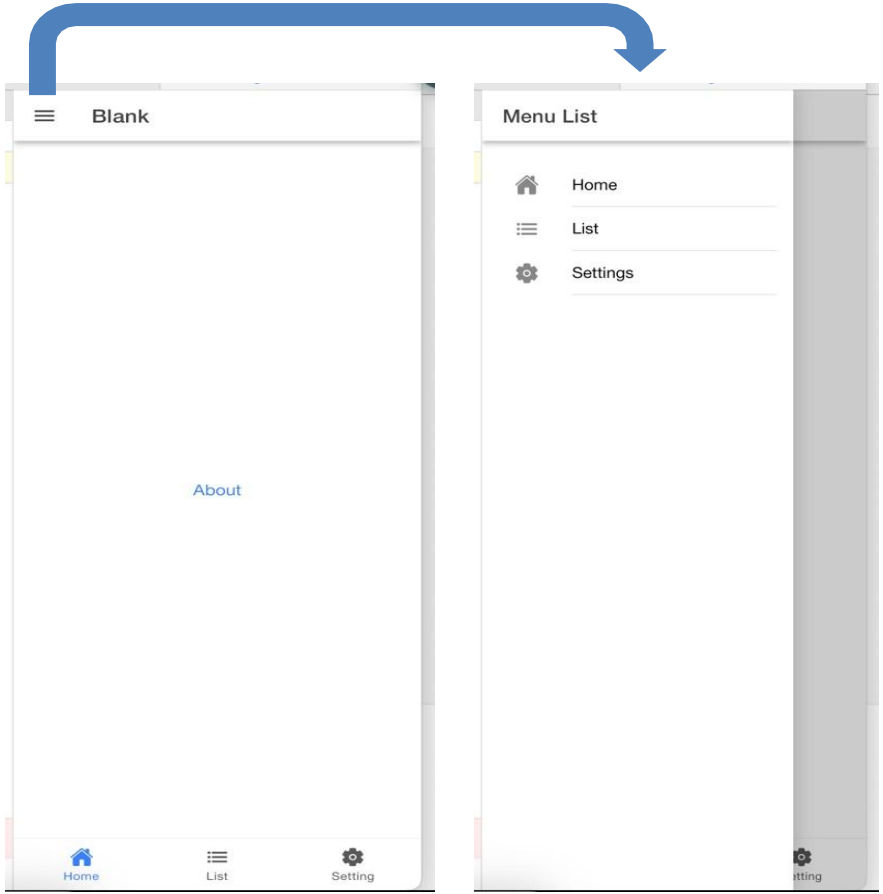
```
<ion-router-outlet id="main-content">
</ion- router-outlet>
```

```
<ion-menu auto-hide="true" contentId="main-content">
   <ion-header>
      <ion-toolbar>
           <ion-title>Menu List</ion-title>
      </ion-toolbar>
   </ion-header>
   <ion-content class="ion-padding">
      <ion-list>
            <ion-item routerLink="/home">
                <ion-icon name="home" slot="start"></ion-icon>
                 <ion-label>Home</ion-label>
            </ion-item>
            <ion-item routerLink="/list">
                 <ion-icon name="list" slot="start"></ion-icon>
                 <ion-label>List</ion-label>
            </ion-item>
            <ion-item routerLink="/setting">
                 <ion-icon name="settings" slot="start"></ion-icon>
                 <ion-label>Settings</ion-label>
            </ion-item>
      </ion-list>
   </ion-content>
</ion-menu>
```

Adding Toggle Button

In the pages that can open the drawer.
Put this, usually before the caption.

```
<ion-buttons slot="start">
    <ion-menu-button></ion-menu-button>
</ion-buttons>
```



Auto Hide When Menu is Clicked

To make the drawer hide again after user click the menu option : add ion-menu-toggle

```
<ion-menu-toggle>
  <ion-item routerLink="/home">
  <ion-icon name="home"
slot="start"></ion-icon>
  <ion-label>Home</ion-label>
  </ion-item>
</ion-menu-toggle>
```

Thanks.

Any Question ?