# Angular Material – Owner\_Account

# Angular Material Installation

In this article, we are going to show you how to prepare our Angular project and how to install Angular Material in a few simple steps. But first thing first. Before we start with the Angular Material features, we need to create our project first.

So let’s dive right into it.

## Project Creation

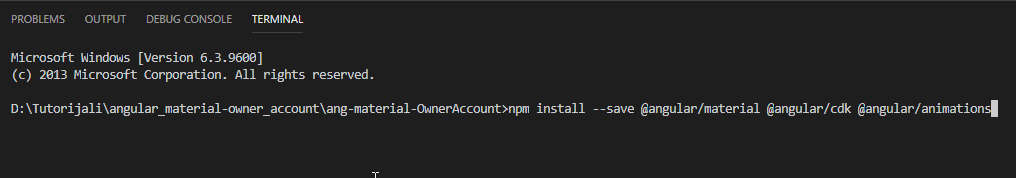
We are going to use Angular CLI through the entire project (and we strongly advise you to do the same), thus creating our project is no exception to that. So, let’s open the command prompt window and create our Angular project:

ng new ang-material-OwnerAccount

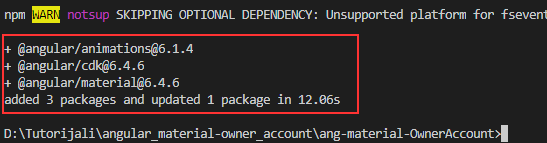
Once the installation is done, we are going to open the Visual Studio Code editor and open our project.

## Angular Material Installation

We are going to use npm to install required packages. Installing just Angular Material is not enough, we need to install CDK and Animations as well. So let’s do that first:



After installation finishes, we should see this result:



Now, we need to configure our animations, by importing BrowserAnimationsModule into the app.module.ts file:

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import {BrowserAnimationsModule} from '@angular/platform-browser/animations';

import { AppComponent } from './app.component';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

BrowserAnimationsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

To continue, let’s include the prebuild theme for Angular Material. The theme is required and we can choose one of the available pre-built themes:

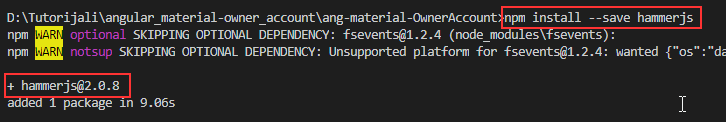
* deeppurple-amber.css
* indigo-pink.css
* pink-blugrey.css
* purple-green.css

To include a theme, we need to open the styles.css file and include the following line:

@import "~@angular/material/prebuilt-themes/indigo-pink.css";

The next step is to install the hammerjs library for the gesture support. In order to have a full feature of some components, we need to install it:

npm install --save hammerjs



After installation, we are going to import it as a first line in the maint.ts file:

import 'hammerjs';

And the last step is to add Material Icons if we want to. This is an optional step, but since we are going to use those icons, we are going to add them as well in the index.html file:

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>AngMaterialOwnerAccount</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">

</head>

<body>

<app-root></app-root>

</body>

</html>

## Creating Material Module

Even though we can import all the required components into the app.module.ts file, this is not recommended. A better solution is to create a separate module with all the required material imports, and then import that separate module into the app.module.ts file. That being said, it is time to do that:

ng g module material --spec false

This command will create a new folder material with the material.module.ts file inside. But this file is missing one thing and that’s the exports array. So, let’s add it:

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

@NgModule({

imports: [

CommonModule

],

exports: [

],

declarations: []

})

export class MaterialModule { }

Finally, we need to import this MaterialModule int the app.module.ts file:

import { MaterialModule } from './material/material.module';

imports: [

BrowserModule,

BrowserAnimationsModule,

MaterialModule

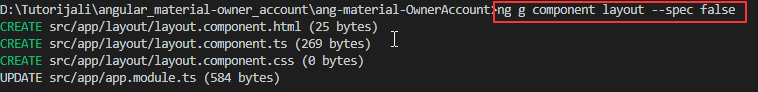
],

That is it. We have prepared everything we need to use the Material components, and without further ado, we are going to start using them.

## Creating the Layout Component

This component is going to be an entry point for our entire application, so let’s create it and import its selector inside the app.component.ts file:

ng g component layout --spec false

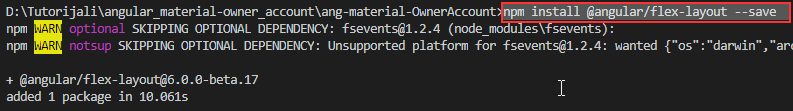


This command will create our component files and import it into the app.module.ts file.

### Angular Flax Layout

Before we modify the html component file, we need to install one more library @angular/flex-layout. We want our application to be responsive and for that purpose, we will use the mentioned library. So, let’s install it:

npm install @angular/flex-layout –save



And we need to register it inside the app.module.ts file:

import {FlexLayoutModule} from '@angular/flex-layout';

imports: [

BrowserModule,

BrowserAnimationsModule,

MaterialModule,

FlexLayoutModule

],

Now we can modify our layout.component.html file:

<div fxLayout="row wrap" fxLayoutAlign="center center" class="layout-wrapper">

<div fxFlex="80%" fxFlex.lt-md="100%" class="flex-wrapper">

<ng-content></ng-content>

</div>

</div>

As we can see, we use some angular/flex directives to create a responsive wrapper around our content. With the fxLayout element, we define the flow order of the child elements inside the container. The fxLayoutAlign will position children according to both main-axis and the cross-axis.

The fxFlex element resizes the child element to 80% of its parent, and if the screen goes below the medium than the child will take a 100% of its parent. If you want to read more about flex-layout, you can do that on this link: [Flex-Layout-Documentation.](https://github.com/angular/flex-layout/wiki/API-Documentation)

With the <ng-content> element, we are using angular content projection.

We have two more classes: layout-wrapper and flex-wrapper, so let’s implement them inside the layout.component.css file:

.layout-wrapper{

height: 100%;

}

.flex-wrapper{

height: 100%;

}

Excellent.

All we have to do is to remove all the content from the app.component.html file and introduce this component by using its selector:

<app-layout>

Application works.

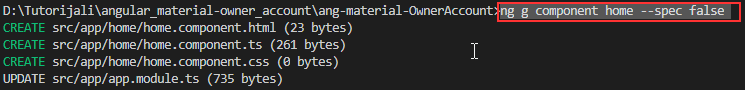
</app-layout>

We can start our application by typing ng serve and see that application truly works.

## Creating Home Component and Using Material Tabs

Let’s create the Home component file structure first:

ng g component home --spec false



Now, let’s modify the home.component.html file:

<section fxLayout="column wrap">

<div fxFlexAlign="center">

<p>Welcome to the Material Angular OwnerAccount Application</p>

</div>

<p>In this applicatoin we are going to work with:</p>

</section>

We need to modify the app.component.html file:

<app-layout>

<app-home></app-home>

</app-layout>

And we need to modify the home.component.css file:

section div p{

color: #1466d0;

font-size: 30px;

text-shadow: 1px 1px 3px grey;

margin: 30px 0;

}

section div + p{

color: #1466d0;

font-weight: bold;

font-size: 20px;

padding-bottom: 20px;

}

To use our first material component, the mat-tab component, we need to register it inside the material.module.ts file:

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { MatTabsModule } from '@angular/material';

@NgModule({

imports: [

CommonModule,

MatTabsModule

],

exports: [

MatTabsModule

],

declarations: []

})

export class MaterialModule { }

And then to modify the home.component.html file:

<section fxLayout="column wrap">

<div fxFlexAlign="center">

<p>Welcome to the Material Angular OwnerAccount Application</p>

</div>

<p>In this application we are going to work with:</p>

<mat-tab-group>

<mat-tab label="Material Components">

<p>We are going to use different material components to create nice looking angular app.</p>

</mat-tab>

<mat-tab label="Consume .NET Core Web API">

<p>We will consume our .NET Core applicatoin. Basicaly, we will create complete CRUD client app.</p>

</mat-tab>

<mat-tab label="Fully responsive navigation menu">

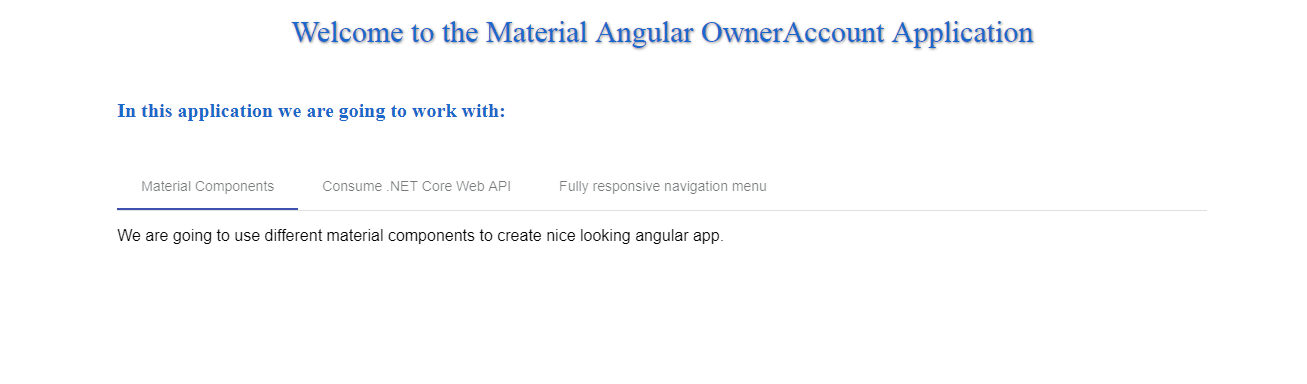
<p>By using material components, we are going to create fully responsive navigation menu, with its side-bar as well.</p>

</mat-tab>

</mat-tab-group>

</section>

Now, we can inspect our result:



## Additional Mat-Tab Features

First of all, let’s style our tab content a little more, to center our text inside every tab:

mat-tab-group {

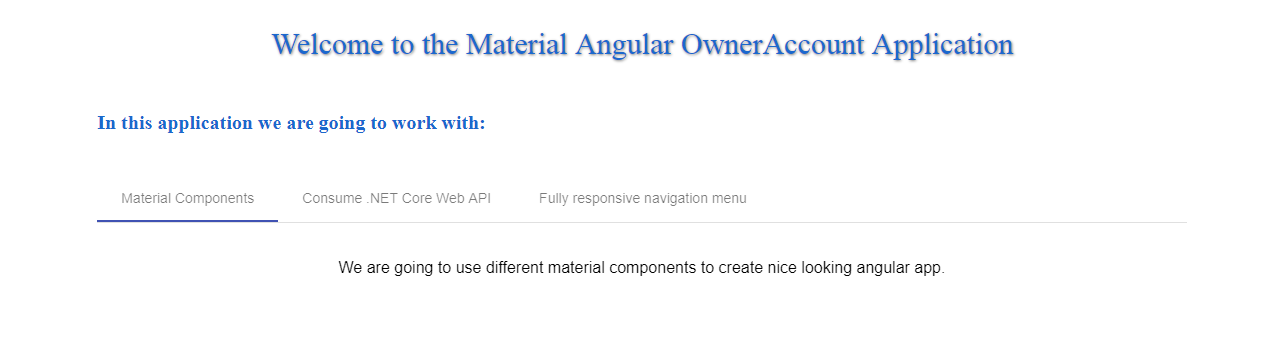
text-align: center;

}

mat-tab-group p {

padding-top: 20px;

}



This control has its own events. The selectedTabChange event is emmited when the active tab changes. The focusChange event is emitted when the user navigates through tabs with keyboard navigation.

So, let’s use the selectedTabChange event:

<mat-tab-group (selectedTabChange)="executeSelectedChange($event)">

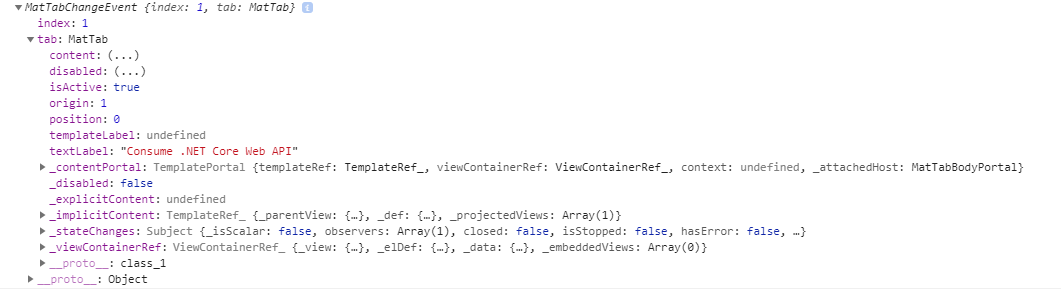
And we need to modify the home.component.ts file:

public executeSelectedChange = (event) => {

console.log(event);

}

Right now, as soon as we switch our tabs, we will see the event object logged into the console window:



## Conclusion

Everything looks great. We have our first material component and in the following articles, we will work with other material components as well.

In this article we have learned:

* How to prepare the Angular project,
* How to install Angular Material, CDK, and Animations
* The way to install and use the @angular/flex-layout library
* To work with the Tab Material component

In the next article, we are going to learn more about creating a navigation menu and menu side-bar with Angular routing.

# Angular Material Navigation Manu

Every application needs to have some sort of navigation, to enable users better navigation experience. Creating such a navigation menu will be our goal in this article. We have to use angular routing as well, and we are going to use it, but we won’t dive deep inside the routing concept. If you want to learn in more detail about angular routing, you can read this article [Angular Navigation And Routing](https://code-maze.com/net-core-web-development-part8/).

Because this series is all about angular material, this article won’t be an exception. We will mainly focus on creating a navigation menu by using different material components. Once we are done, we will have a fully responsive and functional navigation menu with the routing logic to support the complete process.

## Creating Routes

Let’s start with creating a new routing module:

ng g module routing --spec false --module app



Now, let’s modify the routing.module.ts file:

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from '../home/home.component';

const routes: Routes = [

{ path: 'home', component: HomeComponent},

{ path: '', redirectTo: '/home', pathMatch: 'full' }

];

@NgModule({

imports: [

CommonModule,

RouterModule.forRoot(routes)

],

exports: [

RouterModule

],

declarations: []

})

export class RoutingModule { }

Finally, let’s modify the app.compnent.html file to complete routing part for now:

<app-layout>

<main>

<router-outlet></router-outlet>

</main>

</app-layout>

We should be able to see our home component again, but this time it is served on /home route.

## Starting Navigation Development

Angular Material provides us with different components which we can use to create nicely styled, responsive and effective navigation in our app. But we need to start with something, aren’t we? So, let’s start with app.component.html modification by using the mat-sidenav-container component:

<app-layout>

<mat-sidenav-container>

<mat-sidenav #sidenav role="navigation">

<!--this is a place for us to add side-nav code-->

</mat-sidenav>

<mat-sidenav-content>

<!--in here all the content must reside. We will add a navigation header as well-->

<main>

<router-outlet></router-outlet>

</main>

</mat-sidenav-content>

</mat-sidenav-container>

</app-layout>

So, what we do is creating a container for our side navigation bar and specifying the part for our content. As we can see the <mat-sidenav> defines a place for a side navigation and the <mat-sidenav-content> defines a place for our content. We need to use the local reference #sidenav, and a little bit later, we will see why.

Of course, this won’t work. What we are missing is the module registration in the material.module.ts file:

import { MatTabsModule, MatSidenavModule } from '@angular/material';

@NgModule({

imports: [

CommonModule,

MatTabsModule,

MatSidenavModule

],

exports: [

MatTabsModule,

MatSidenavModule

],

Now, we should have a working application again with some grayish background. Let’s style this a bit in the app.component.css file:

mat-sidenav-container, mat-sidenav-content, mat-sidenav {

height: 100%;

}

mat-sidenav {

width: 250px;

}

main {

padding: 10px;

}

And let’s modify the styles.css file:

/\* for sidenav to take whole page \*/

html, body {

margin: 0;

height: 100%;

}

That is it. We have all prepared and it is time to start working on our navigation header component.

## Creating Navigation Header

To create a navigation header, we need to use the mat-toolbar element. But first thing first.

This component has its own module, so we need to register that module inside the material.module.ts file:

import { ..., MatToolbarModule } from '@angular/material';

imports: [

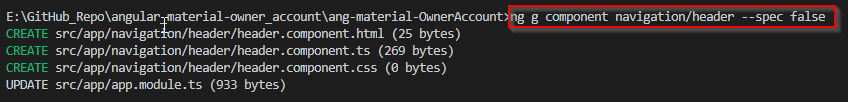
MatToolbarModule,

exports: [

MatToolbarModule,

After that, we are going to create a new header component:

ng g component navigation/header --spec false



Now it is time to include this component inside the app.component.html file, right above the <main> tag:

<mat-sidenav-content>

<app-header></app-header>

<main>

<router-outlet></router-outlet>

</main>

</mat-sidenav-content>

Then, let’s modify the header.component.html file:

<mat-toolbar color="primary">

<div fxHide.gt-xs>

<button mat-icon-button (click)="onToggleSidenav()">

<mat-icon>menu</mat-icon>

</button>

</div>

<div>

<a routerLink="/home">Owner-Account</a>

</div>

<div fxFlex fxLayout fxLayoutAlign="end" fxHide.xs>

<ul fxLayout fxLayoutGap="15px" class="navigation-items">

<li>

<a routerLink="/owner">Owner Actions</a>

</li>

<li>

<a routerLink="/account">Account Actions</a>

</li>

</ul>

</div>

</mat-toolbar>

Basically, we create our navigation with the menu icon (we still need to register its own module inside the material module), and the Owner-Account part that leads to the home component. As we can see, we use the fxHide.gt-xs directive, which states that this part should be hidden only on the screen that is greater than extra small.

We have another part of navigation which is positioned on the end of the navbar and hidden only for the extra small screen.

To continue, let’s register the MatIconModule and MatButtonModule inside the material module file:

import { ... MatIconModule, MatButtonModule } from '@angular/material';

imports: [

MatButtonModule,

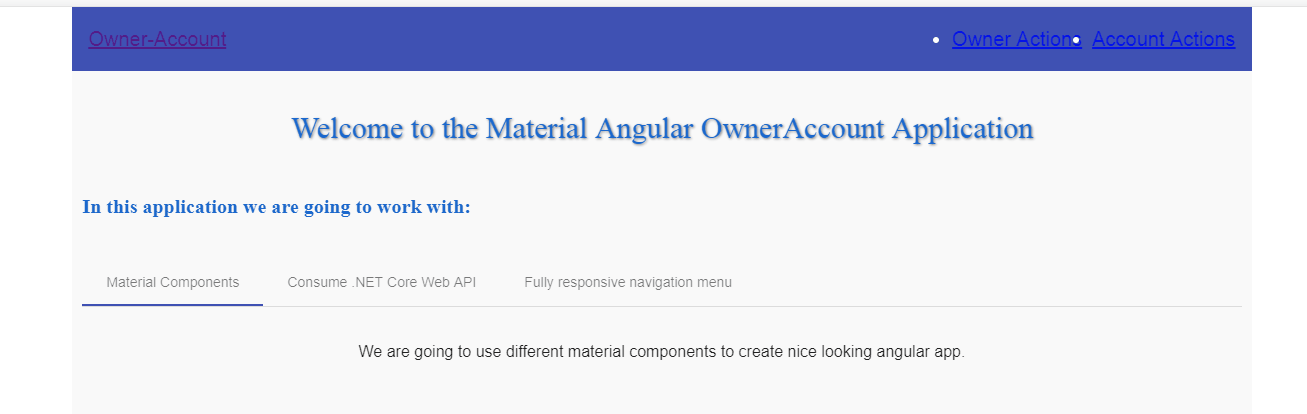
MatIconModule,

exports: [

MatButtonModule,

MatIconModule,

Right now, we have our menu that looks like this:



Looking beautiful right? :D :D Of course not, but we have our starting functionality in place and we are going to make it much nicer.

So, let’s continue and modify the header.component.css file:

a {

text-decoration: none;

color: white;

}

a:hover, a:active{

color: lightgray;

}

.navigation-items{

list-style-type: none;

padding: 0;

margin: 0;

}

mat-toolbar{

border-radius: 3px;

}

@media(max-width: 959px){

mat-toolbar{

border-radius: 0px;

}

}

Now if we look at our menu, it looks much nicer, isn’t it?



If we take a look at our icon button, we are going to see the onToggleSidenav()" event. We need to implement that inside the header.component.ts file:

import { Component, OnInit, Output, EventEmitter } from '@angular/core';

@Component({

selector: 'app-header',

templateUrl: './header.component.html',

styleUrls: ['./header.component.css']

})

export class HeaderComponent implements OnInit {

@Output() public sidenavToggle = new EventEmitter();

constructor() { }

ngOnInit() {

}

public onToggleSidenav = () => {

this.sidenavToggle.emit();

}

}

Finally, we have to react on this event emitter inside our app.component.html file:

<mat-sidenav-content>

<app-header (sidenavToggle)="sidenav.toggle()"></app-header>

<main>

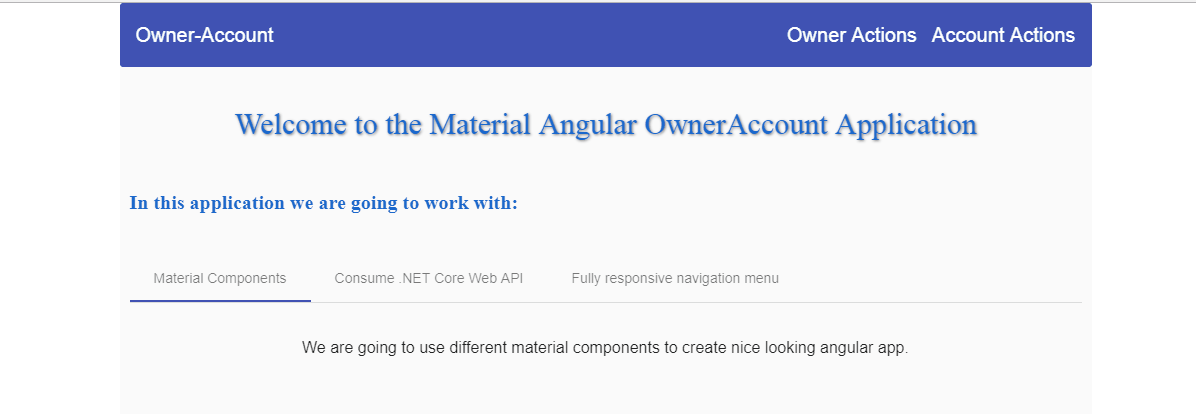
<router-outlet></router-outlet>

</main>

</mat-sidenav-content>

Now we see why we need the #sideNav local reference inside the mat-sidenav component.

Now let’s see how this looks like:



Excellent. Now its time for us to start working on the side-nav.