Angular 14 introduces the standalone component

Components, directives, and pipes can now be declared as standalone

Standalone components, directives, and pipes aim to streamline the authoring experience by reducing the need for [NgModule](https://angular.io/api/core/NgModule)s.

Existing applications can optionally and incrementally adopt the new standalone style without any breaking changes.

A standalone component is a type of component which is not part of any Angular module. Prior to Angular 14, usually when you would create a component, you’d pass it inside the declarations array of a module. If you would not do that, Angular would complain about it and not compile. However, as of Angular 14, you can create a component that is not part of any ngModule, and that component is known as a standalone component.

You can use a standalone component with:

* Module-based components
* Other standalone components
* Loading routes
* Lazy loading

Using a Standalone Component

You can use a standalone component, directive or pipe in either of two ways:

1. Inside another standalone component
2. Inside a module

For both the options, pass it inside the imports array, and also keep in mind that you don’t pass standalone components in the **declaration** array of the modules.

1. 1. Inside a module

So to use it inside AppComponent, which is part of AppModule, you can pass it to the imports array as shown below:

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

AppRoutingModule,

LoginComponent

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

JavaScript

Now you can use it on the AppComponent as below:

<h1>App</h1>

<app-login></app-login>

HTML

You can use a standalone component in another standalone component by passing it to the imports property of that standalone component as shown below:

@Component({

selector: 'app-product',

standalone: true,

imports: [CommonModule, LoginComponent],

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

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

})

export class ProductComponent implements OnInit {

Besides standalone components, in Angular 14, you can also create:

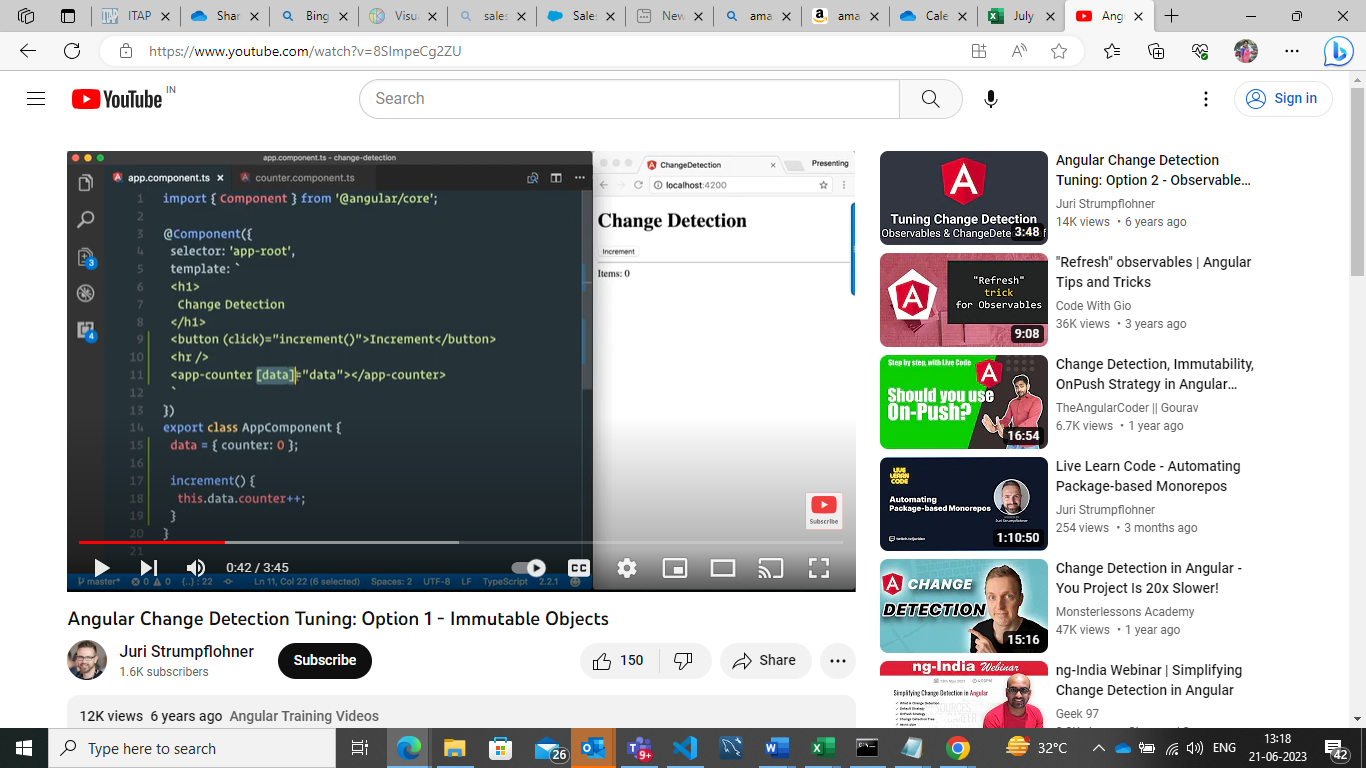
* Standalone directives
* Standalone pipes

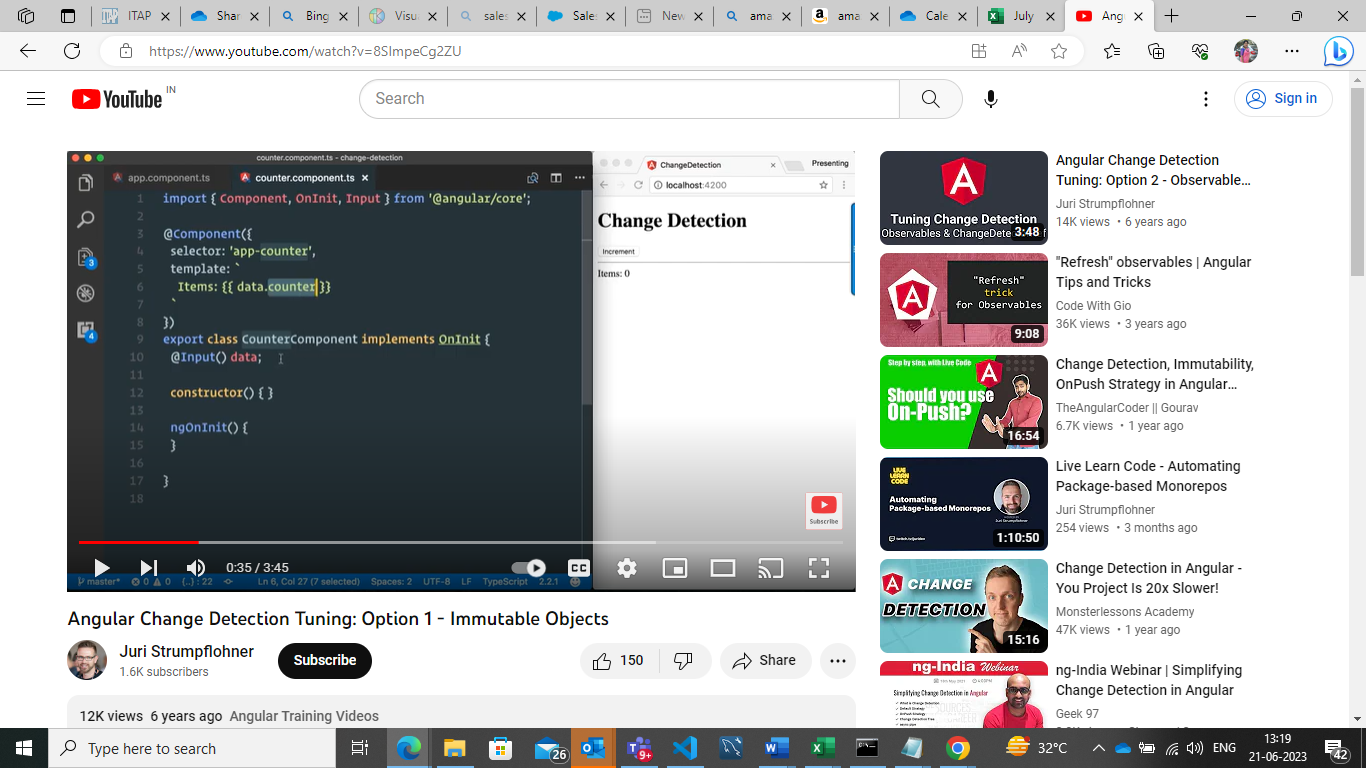
Component

You can create a standalone component, pipe or directive by using the --standalone flag in the ng generate component command:

* ng g p search --standalone
* ng g d credit-card --standalone
* ng g c login --standalone
* Standalone directive Demo:
* Ng g d red
* import { Directive,ElementRef } from '@angular/core';
* @Directive({
* selector: '[appRed]',
* standalone: true
* })
* export class RedDirective {
* constructor(elRef: ElementRef) {
* elRef.nativeElement.style.color = 'red';
* }
* }
* Use red directive in comp1:
* Comp1.ts
* import { Component } from '@angular/core';
* import { CommonModule } from '@angular/common';
* import { RedDirective } from '../red.directive';
* @Component({
* standalone:true,
* selector: 'app-comp1',
* imports:[CommonModule,RedDirective],// to use other module ngif,ngfor
* templateUrl: './comp1.component.html',
* styleUrls: ['./comp1.component.css'],
* })
* export class Comp1Component {
* title="welcome to stanalone component demo!!!!!"
* visible = false
* }
* Comp1.html
* <p>comp1 works!</p>
* {{title}}
* <button (click)="visible = !visible">Toggle</button>
* <div \*ngIf="visible">Hello Angular 15</div>
* <p>directive-demo works!</p>
* <p appRed> myRed Directive Demo!!!!</p>
* Stanalone pipe
* Ng g p gender –standalone

Change Management:





From parent object passed to child and received in object with @Input () property.

Since there is a change in data object of parent component the same change gets propagate to the child and sub-child’s .

To speed the object tell angular only to change the subsequent component not from the root component . change detection tells angular to change when reference to the object change not only the property changes

Onclick of button increment method is invoked . which changes counter property of data object , as we are paaing this object to child . child will also get changed . it will be form top to bottom .

If we change property of an object it will get reflected in all components from top to bottom

To use immutable object to speed up the whole change detection thing so what we are telling angular is basically only update our sub component here the only trigger changes action when the object reference changes and not only basically property within that object.

In the context of Angular, the "onPush" strategy is a change detection strategy that can be used for optimizing the performance of components.

By default, Angular uses the "Default" change detection strategy, which checks for changes in component properties and updates the view accordingly. However, in certain scenarios where the component's state changes infrequently, the "onPush" change detection strategy can be used to improve performance.

When the "onPush" change detection strategy is applied to a component, Angular only checks for changes in the component's input properties or if an event from the component or its children is emitted. It skips the change detection process if there are no changes in the input properties or events. This can result in significant performance improvements, especially for complex components or large component trees.

To use the "onPush" change detection strategy for a component in Angular, you need to do the following:

1. Set the **changeDetection** property of the component decorator to **ChangeDetectionStrategy.OnPush**, like this:

typescriptCopy code

import { Component, ChangeDetectionStrategy } from '@angular/core'; @Component({ selector: 'app-example', templateUrl: './example.component.html', changeDetection: ChangeDetectionStrategy.OnPush }) export class ExampleComponent { // Component logic and properties }

1. Make sure that the component relies on input properties or events to trigger changes. This means that the component should primarily use **@Input()** properties for receiving data and **@Output()** properties for emitting events.

By using the "onPush" change detection strategy, Angular reduces the number of change detection checks and updates, resulting in improved performance for the component.

It's important to note that the "onPush" strategy requires careful management of component state and data flow to ensure that changes are properly propagated and detected. It may require explicit handling of change detection using techniques like **ChangeDetectorRef** and **Observable** data streams.

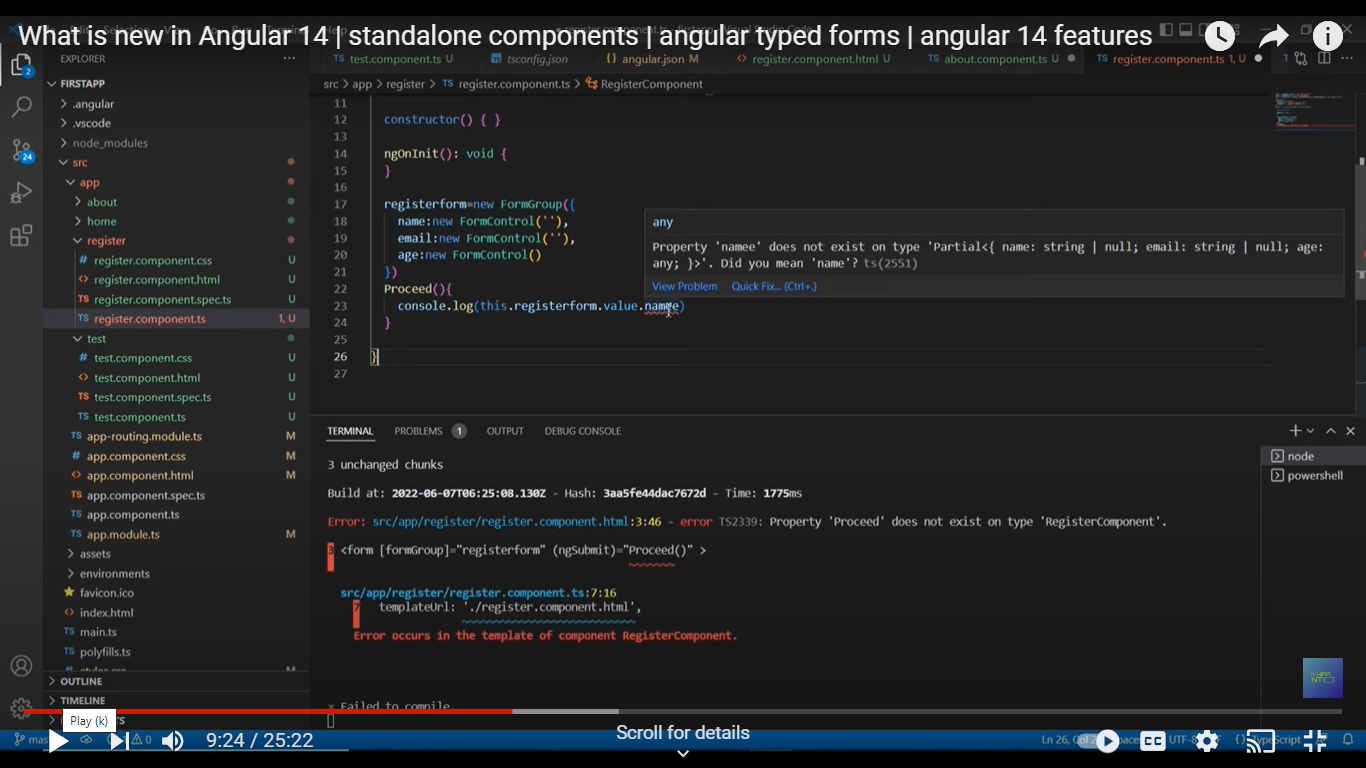
Top of Form

Regenerate response

Bottom of Form

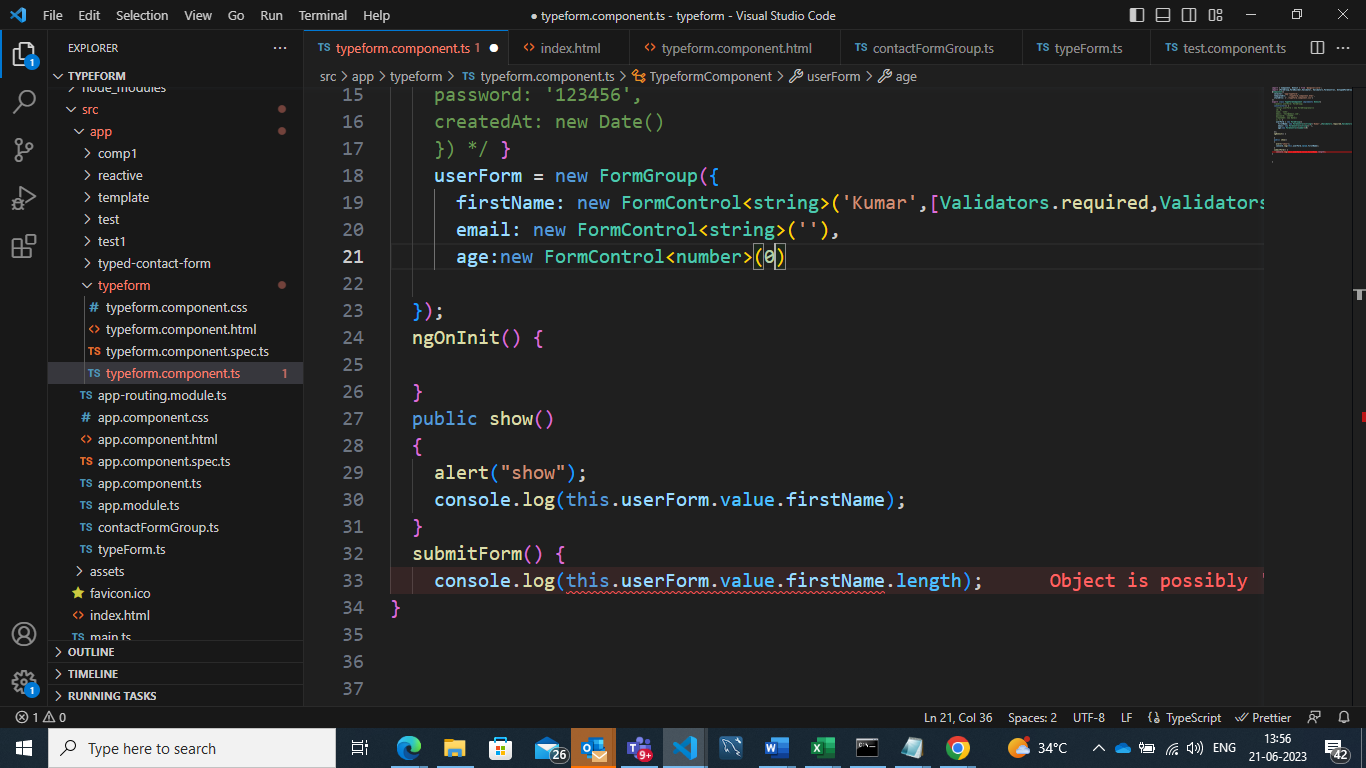
changeDetection:ChangeDetectionStrategy.OnPush tell angular ,Only perform change detection when the reference changes not the property.

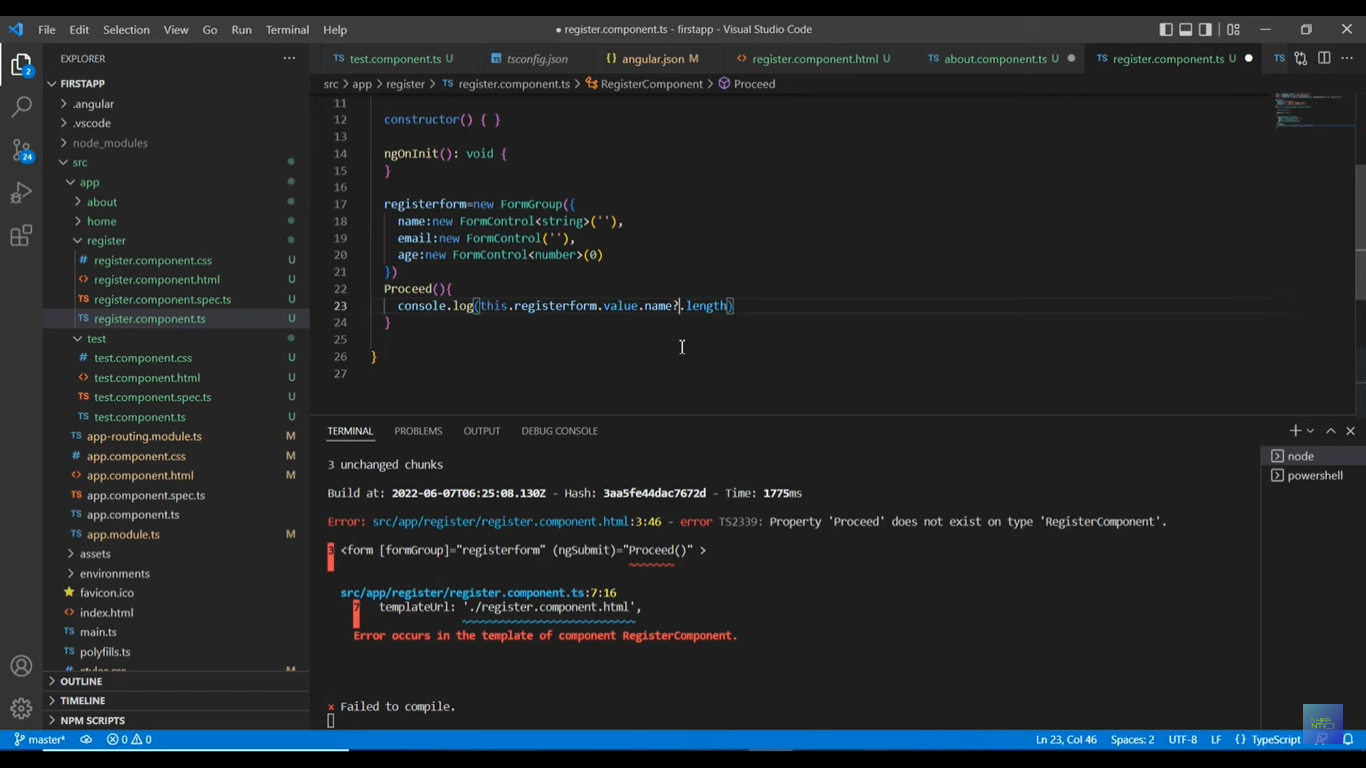
Typed forms:



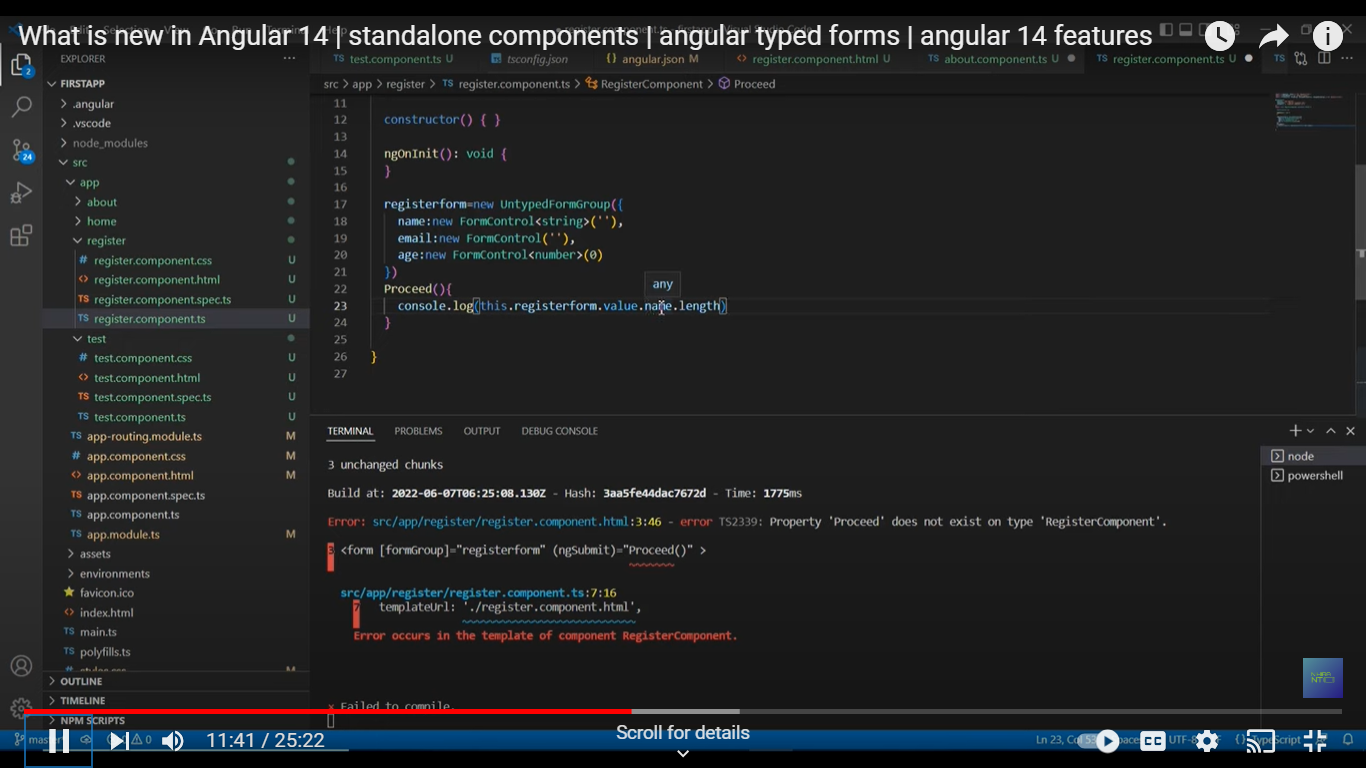
Autofix

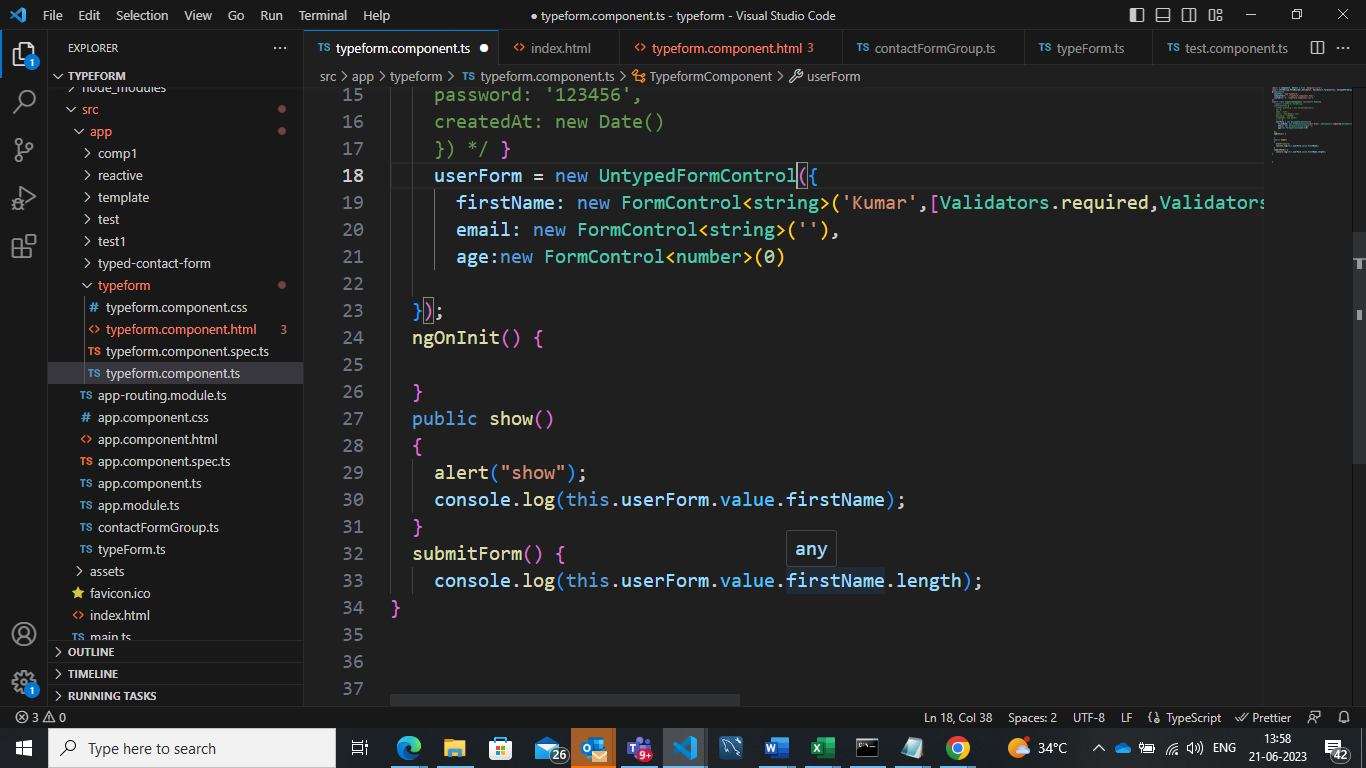
In lower version this features was not added .





Instead of ? in older version we use untypedformgroup and untypedformcontrol





Data type is any.

Tile Strategy:

[Setting Page Titles Natively With The Angular Router 🔥 - DEV Community](https://dev.to/brandontroberts/setting-page-titles-natively-with-the-angular-router-393j)

Extended developer diagnostics

[Angular - ng completion](https://angular.io/cli/completion)