# Angular Fundamentals

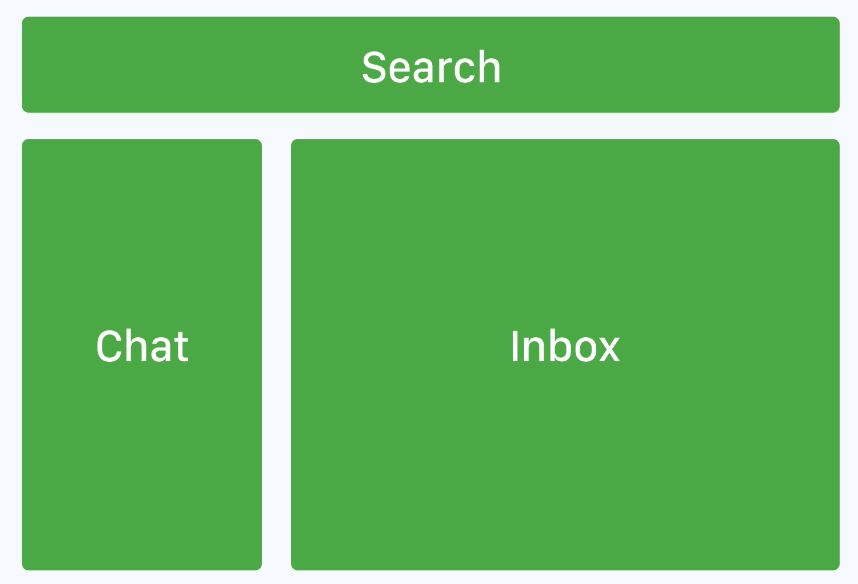
## Angular High-Level Architecture overview

Architecture of angular is made up of 5 building blocks

1. Modules
2. Components
3. Directives
4. Routing
5. Services

### Modules

Is a main building block of any application. We can have multiple modules, single module or depend on other modules. This contains routes, components (features), services and more

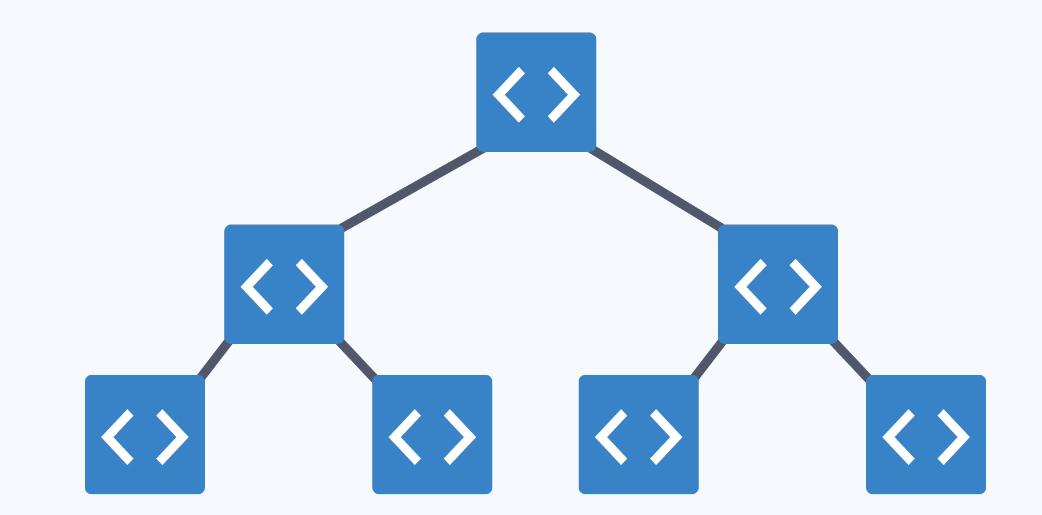


In this single page application, we will have three modules search, chat and inbox modules. Each module will have specific component.

### Components

Contains a template, data and logic, forming part of a DOM tree.

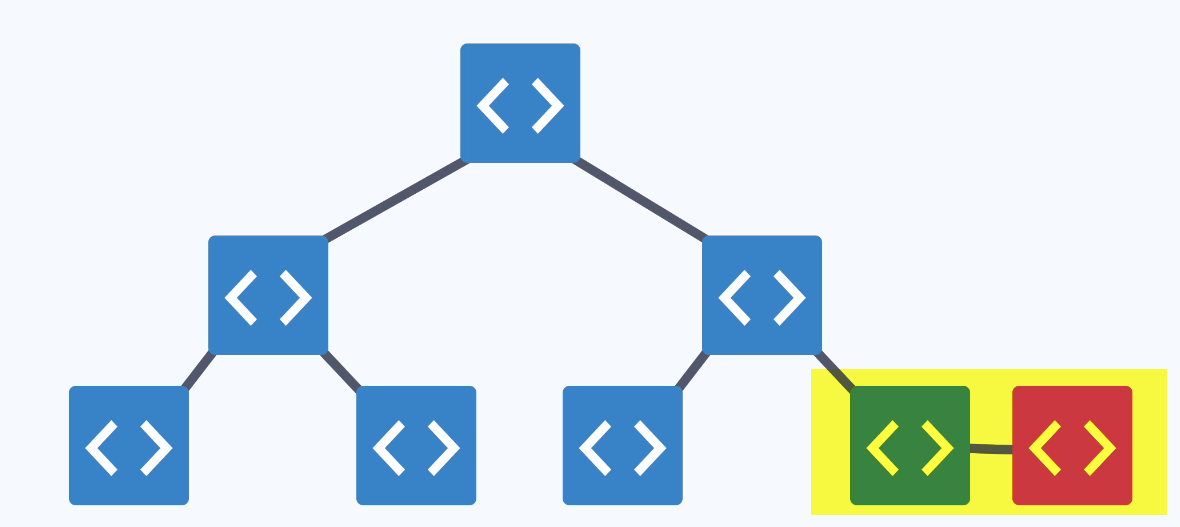
Component is a high-level DOM tree node which will then create sub tree nodes.



This diagram represents what DOM tree of component look like. Each component will render its own HTML inside. We call this as component architecture.

### Directives

Directive can bind to existing DOM element or HTML element or a component or insider template on its own which will extend or transform an element and its children.

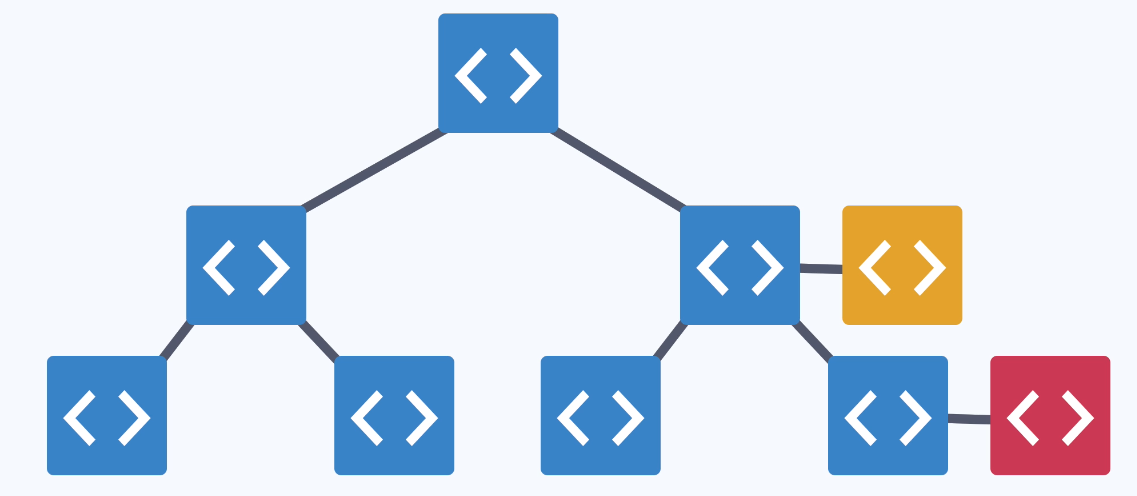


Directive is bonded to a component. Example search directive bonded to search box in a component which will then tie to a behavior when a user types we might want to listen to an event.

### Services

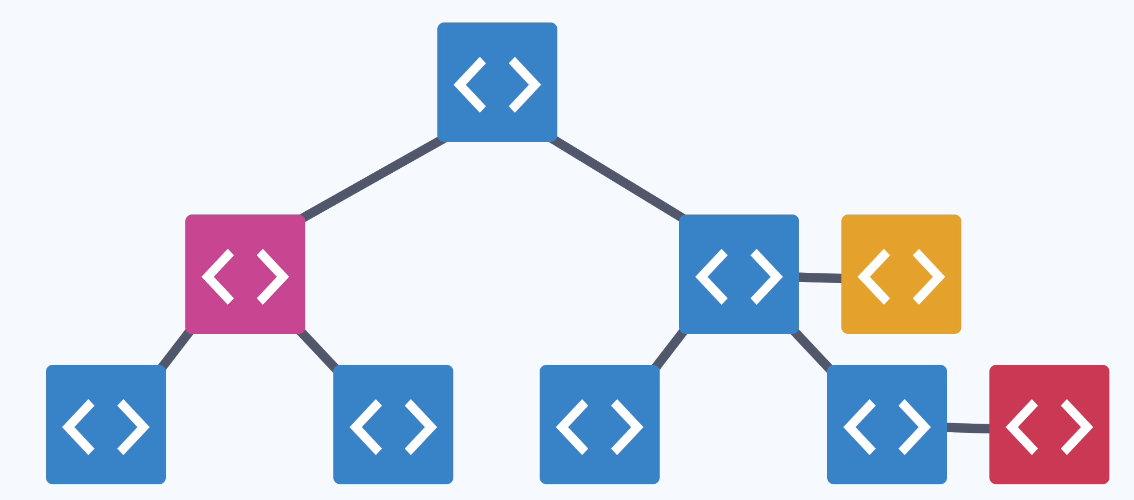
Services is basically data layer, logic that is not component related, such as API requests. This is mainly used to get data into a component.

AJAX request to a API fetch some data back and we pass it into a component, component can render the data and we can make changes in the view and we can update the backend of which all of this logic is then ley in the service.



### Routing

Drives the navigation for the application. Tells the application which component should be rendered based on the URLs. Routing renders a component based on the URL state, drives navigation.



/inbox – left hand component

/chat – right hand component.

Setup Instructions

Step 1: Package Manager

To install the project dependencies, you will need to use either npm or install Yarn.

If you have already installed Yarn, be sure to upgrade to the latest version. If you’d like to install yarn over npm, run the following in your terminal:

npm install -g yarn

Mac users can alternatively use brew to install yarn.

brew update

brew install yarn

If you experience any issues when installing/using yarn you can checkout the installation instructions here. <https://yarnpkg.com/en/docs/install>

Step 2: Project Dependencies

Now that we have a package manager, we can install the project dependencies.

You can do this by running:

yarn install

#OR

npm install

This will install our dependencies for running our Angular application. If you experience any setup issues with Windows, please see this thread

<https://github.com/UltimateAngular/angular-fundamentals-seed/issues/10>

Step 3: Running the project

During development, the project is built using webpack-dev-server. This provides a local development server as well as having webpack recompile our app when a file changes. The project will also automatically refresh the page whenever we make changes.

To start the project in development, run:

yarn start

#OR

npm run start

This will output some information about the project (such as the TypeScript version and build progress). Once you see “build completed”, you are ready to code!

Open your browser to localhost:4000 to start running the code.

## v4-v6 Changelog

Just a few things to take note on that have changed since the course was recorded. These are very brief, and worth addressing after you’ve completed the course as they are just minor changes.

ng-template

<template> is now <ng-template>

Http module

Http is now HttpClient

This change is almost identical but also comes with some additional typing options.

<https://angular.io/guide/http>

RxJS

RxJS v6 imports have changed, read more here. This does not affect behaviour or concepts of the courses, merely an aesthetic change.

<https://github.com/ReactiveX/rxjs/blob/master/docs_app/content/guide/v6/migration.md#howto-convert-to-pipe-syntax>

# ES5 to ES6 and TypeScript refresher

Don’t use the Angular project seed for these next few lessons, we’re using a simple TypeScript setup only – if you use the seed project you’ll likely see “Module parse failed” errors like this.

Module parse failed: angular-fundamentals-seed/main.ts Unexpected token (35:11) You may need an appropriate loader to handle this file type.

Instead, follow along either just by watching or coding on TypeScript Playground.

<https://www.typescriptlang.org/play/>

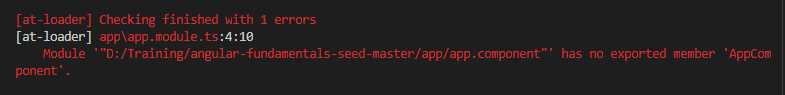
## First component with @Component

In app.component.ts

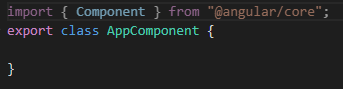
We need to first import Component – Component is a decorator. Decorator is essentially just a function that annotates specific function.

import { Component } from "@angular/core";

In terminal we will get below error

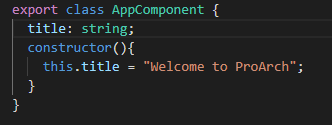


This means that we need to export a class named AppComponent





Below is better code than above.



Component is just a function in - import { Component } from "@angular/core";

However, it’s a decorator we use @Component to use this type script feature.



@Component - This is to register our component with angular.

The selector essentially creates the element in the HTML

All angular projects have a base root component that renders the entire application.

Assign the template to this specific component. There are two ways

1. Template with in-inline string. By using back tick ``
2. templateUrl – we can have the html defined and provide the html path here.

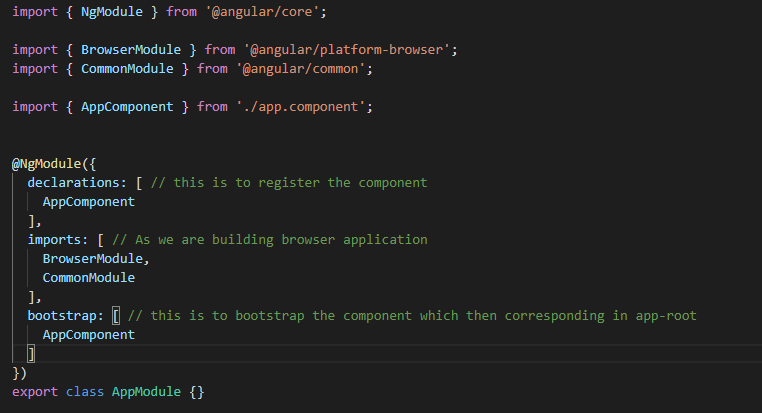
## Root module with @NgModule

Angular is split up into multiple modules. So each piece of angular such as Forms / HTTP lives inside different module.

NgModule & the Component decorator is available in core module.

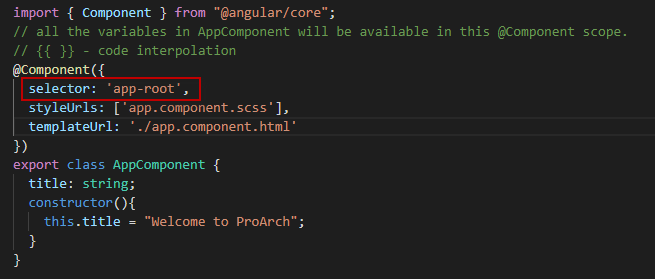
Common decorator is available in common module. When we start working with templates and creating things such as directives. This will come into picture.

BrowserModule decorator is available in platform-browser module.

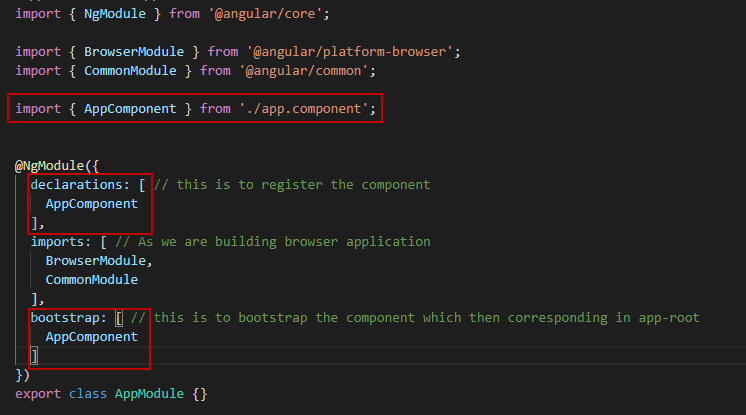


## Bootstrapping Angular

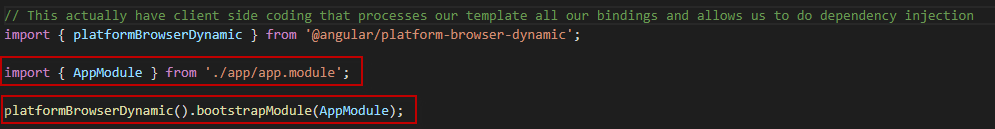
From the top we have app component, register this as app-root, which is placed in index.html



To bootstrap this app-root we must import the component registered in the module we have to tell the module to bootstrap the component



And in our main typescript file we can tell the compiler to bootstrap our application module.



# Template fundamentals

## Interpolation and expressions

Interpolation allows angular to bind specific properties from AppComponent to template.

{{ Title }} – this is actually called expression

Examples of expressions with interpolation:

{{ title }}

<div> {{ title + '!' }} </div>

<div> {{ numberOne }}, {{ numberTwo }} </div>

<div> {{ numberOne + numberTwo }} </div>

<div> {{ isHappy ? ':-)' : ':-(' }} </div>

## Property binding – square bracket notation

How we can pass data from component class to a template, by binding it to a element.

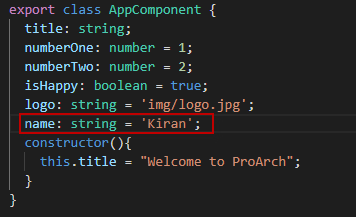
{{ }} – exactly called as sugar syntax. Sugar syntax is easier way of doing something that looks cleaner.

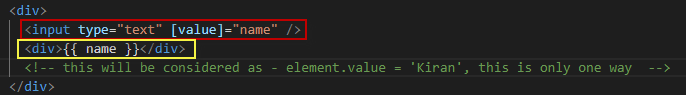
<h1>{{ title }}</h1> - title can fit into h1 element.

<h1 innerHTML=”title”></h1>

To do this angular need to bind to a property. This way we do this is:

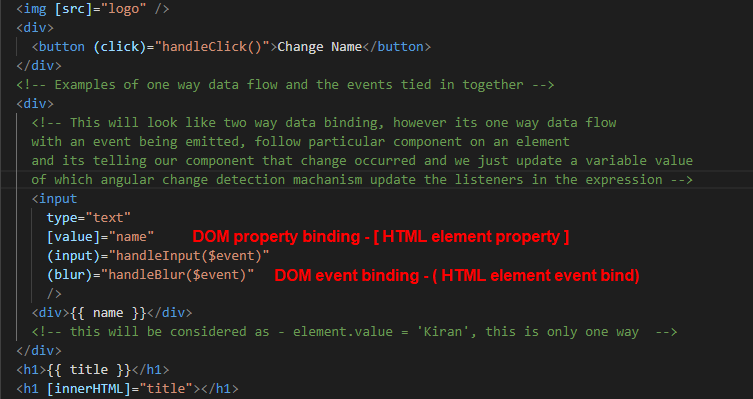
<h1 [innerHTML]=”title”></h1> – one way day flow syntax. Variable defined in component and we using it in template looking for innerHTML property.





If we change name in input, it will not reflect in the expression below.

## Event binding



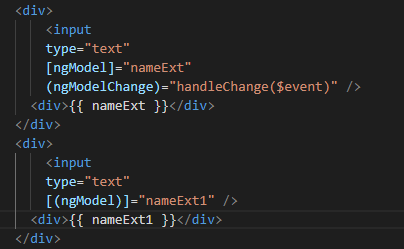


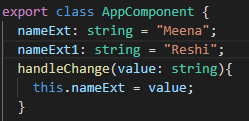
## Two-way binding

FormsModule – from '@angular/forms' – allow us to do two-way day binding

Two-way data binding is okay if we have local component. However, when we want to emit a change, we should always use one way data flow and emit the change with event listener.

Two-way data binding with ngModel – [( )] – property binding – inside that we are listening to event binding. Both the below code in HTML are same. The second part [( )] – angular with take care of that and convert it to the first type of HTML code.

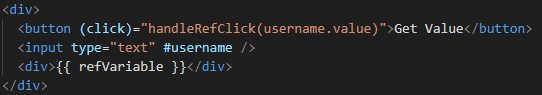


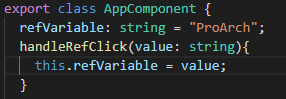


If we go with same variable. Both the code with change the two labels as well as the two text boxes.

## Template #ref variables

Template ref gives access to particular DOM element essentially exports the properties so that we can access anywhere in the template using that #variable.

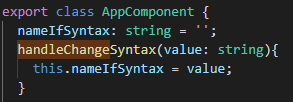


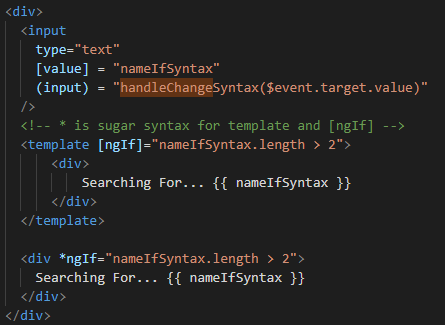


# Rendering flows

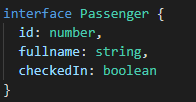
## ngIf, \* syntax and <ng-template>

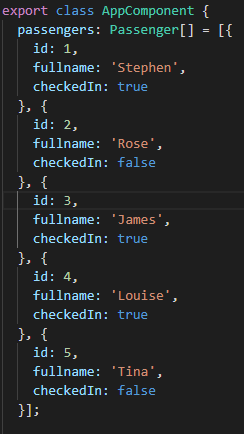
\*ngIf – angular directive - is called structural directive.

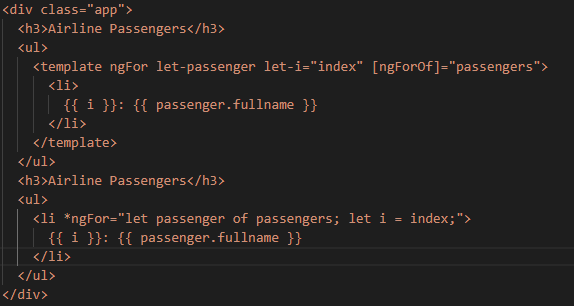




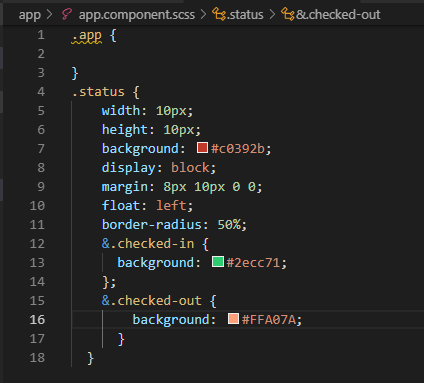
## ngFor and iterating collections

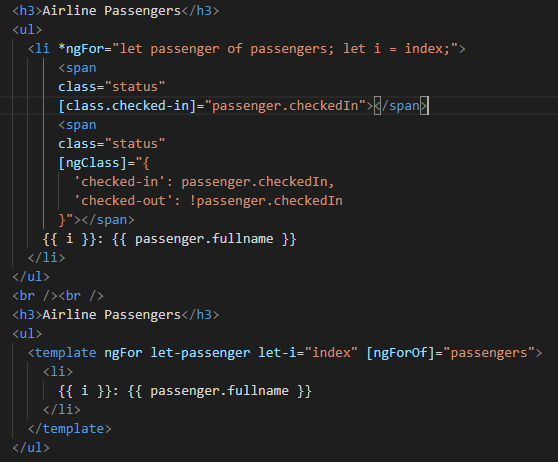




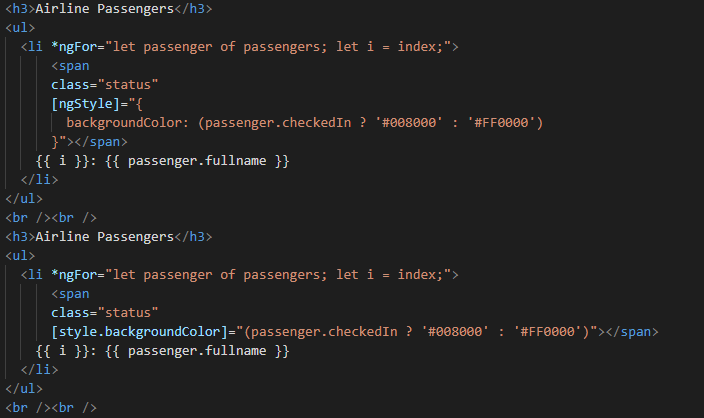


## ngClass and className bindings



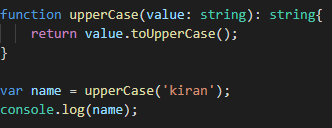


## ngStyle and style bindings

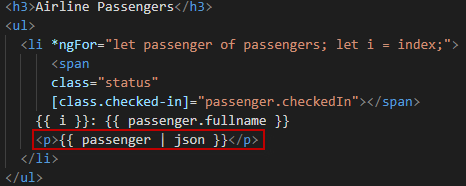


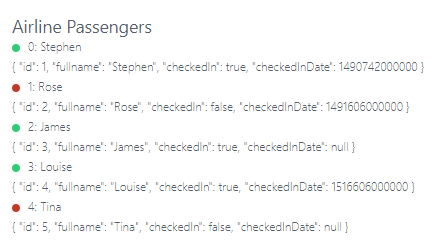
## Pipes for data transformation

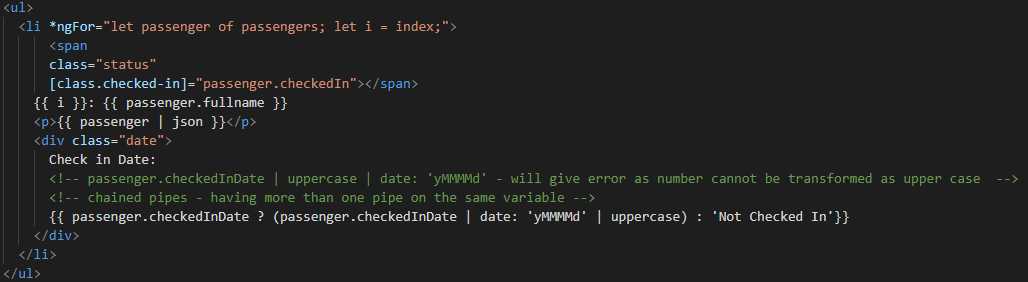
Pipes means functions that return something new. Pipe is a data transformation mechanism.



Json pipe







## Safe navigation operator

