What are the main building blocks of an Angular Application?

Angular is a widely used open-source front-end framework used to build dynamic web applications. It consists of several key building blocks that work together to create scalable, and maintainable applications. In this article, we will explore all the main building blocks of an Angular application and we will explore how they contribute to framework architecture.

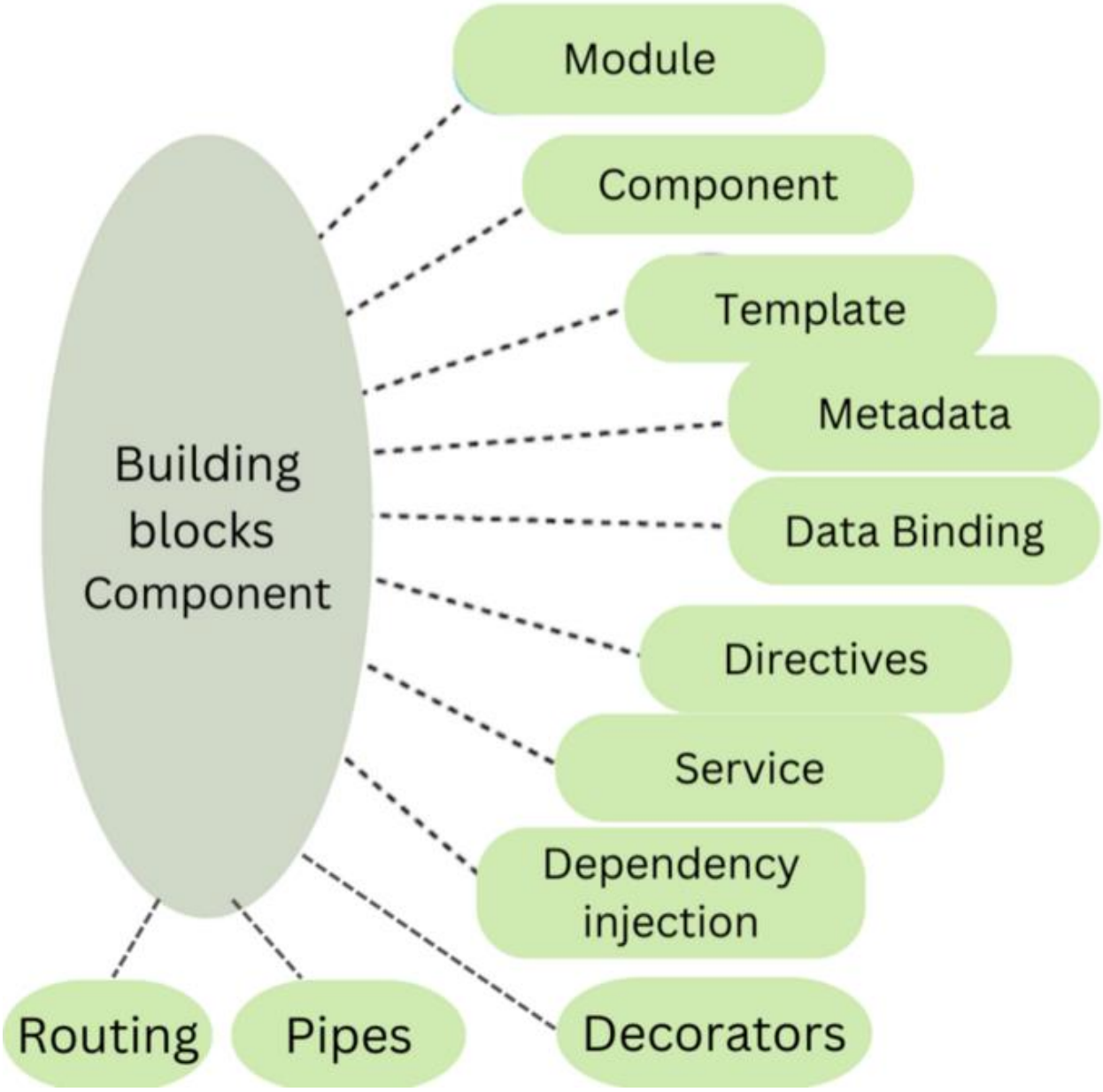
Angular Application Architecture

An Angular application follows the Model-View-Controller (MVC) and Model-View-ViewModel (MVVM) principle design patterns. The Model View Controller (MVC) design pattern specifies that an application consists of a data model, presentation information, and control information. In the Model-View-ViewModel (MVVM) design pattern, the Model component resembles the same concepts as in the traditional Model-View-Controller (MVC) pattern, along with facilitating the extended and adapted to better support the requirements for data binding and the ViewModel.

Building Blocks of an Angular Application

The main building block of an Angular application is as follows:

- Module
- Component
- Templates
- Metadata
- Data Binding
- Directives
- Service
- Dependency Injection
- Decorators
- Pipes
- Routing



Building Blocks of an Angular Application

Module

Modules are like containers for components, directives, services, etc. They help to organize an Angular application which makes it easier to manage or scale application. Modules also support lazy loading means it is used to improve application performance by loading specific parts of the application only when they are required. The root module also known as App Module provides a starting point for the application.

- Javascript

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule }
  from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Components

Components are the fundamental block for the Angular application interface. They combine the presentation layer (template) and component class. They also enable the creation of custom HTML elements and help in transforming the user interface into a collection of self-contained and easily maintainable units. Each component is recognized by means of a completely unique selector, which acts as a custom HTML detail, enabling you to embed it within other components or templates. Components are the visible entities in which users interact with it without any delay, making it a crucial part of the user experience.

- Javascript

```
// app.component.ts
import { Component }
  from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'My Angular App';
}
```

Templates

Templates define the structure of components using Angular templating syntax. They display the data and provide the structure for the user interface. Angular template syntax is also used to combine HTML with Angular-precise directives and bindings. Directives work as markers in the template that teach Angular how they can transform the DOM earlier than rendering it.

- HTML

```
<!DOCTYPE html>
<html>

<head>
  <title>Main Building Block</title>
</head>

<body>
  <!-- app.component.html -->
```

```
  <h1>{{ title }}</h1>
  <p>Welcome to {{ title }}</p>
</body>


</html>
```

Metadata

Metadata refers to the facts that are associated with various elements in an Angular application like components, modules, directives, and so forth. Metadata is furnished using decorators, which can be special capabilities that attach extra statistics to the elements. This metadata facilitates Angular to apprehend the method and use these factors. They also are used to configure and define numerous factors of application elements such as their behavior, look, and relationships with other elements.

- Javascript

```
// app.component.ts
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
}
```

Data Binding

Data binding is responsible for connecting the component's logic with its template through which data can be updated automatically. Angular supports various types of data binding like interpolation, property binding, event binding, and two-way binding which help to develop a responsive experience for the user.

- Javascript

```
// app.component.ts
```

```
@Component({
 // ...
})
export class AppComponent {
 title = 'My Angular App';

 changeTitle() {
  this.title = 'New Title';
 }
}
```

Angular supports multiple kinds of Data Binding:

- **Interpolation**: Embedding expressions inside double curly braces (expression) in the template.
- **Property Binding**: Binding a component property to an element's property with the usage of square brackets.
- **Event Binding**: Binding an element's occasion to an issue technique using parentheses.
- **Two-Way Binding**: Combining property and event binding to achieve bidirectional data synchronization using the [(ngModel)] directive.

Directives

Directives are the special kind of markers that tell Angular to attach specific behavior to elements. Angular has three types of directives:

- **Component Directives**: They are used to create reusable UI components.
- **Attribute Directives**: They are used to change the appearance or behavior of an element like changing its color depending on the condition.
- **Structural Directives**: They are used to modify the structure of the DOM, like adding or removing elements depending on the condition. for example ngIf and ngFor.
- Javascript

```
// highlight.directive.ts

import { Directive, ElementRef, HostListener }
```

```
  from '@angular/core';


@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor(private el: ElementRef) { }

  @HostListener('mouseenter') onMouseEnter() {
    this.highlight('yellow');
  }

  @HostListener('mouseleave') onMouseLeave() {
    this.highlight(null);
  }

  private highlight(color: string) {
    this.el.nativeElement.style.backgroundColor = color;
  }
}
```

Services

Services refer to those classes that provide functionality. They are designed to be embedded into components, services, or modules that assist in sharing statistics, imposing enterprise good judgment, and handling communication with API. This makes it less complicated to reuse the code again and additionally allows for higher testing and preservation.

- Javascript

```
// data.service.ts
```

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

@Injectable({
  providedIn: 'root'
})
export class DataService {
  getData() {
    return 'Data from DataService';
  }
}
```

Dependency Injection

An Angular-Dependency Injection system could be a very beneficial function that allows the control of dependencies internally on the software itself. DI encourages unfastened coupling by using decoupling additives and services from the context of ways their dependencies are mapped. It lets you declare the dependencies of a factor or service. This enhances modularity, testability, and code reusability and additionally makes code bendy and smooth to use.

- Javascript

```
// app.component.ts
import { Component } from '@angular/core';
import { DataService } from './data.service';

@Component({
  // ...
})
export class AppComponent {
  constructor(private dataService: DataService) { }
```

```
  getDataFromService() {
    const data = this.dataService.getData();
    console.log(data);
  }
}
```

Decorators

Decorators in Angular are special sorts of capabilities that may be used to modify or add metadata to classes, methods, properties, or parameters. Angular decorators play a critical function in configuring and enhancing numerous elements of an Angular application. Decorators in Angular offer an easy and declarative way to configure and increase the behavior of various elements of your software. They are a necessary part of the Angular metadata system, allowing you to outline how additives, modules, and other factors ought to behave and interact inside your utility.

- Javascript

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

@Component({
  selector: 'app-example',
  templateUrl: './example.component.html',
  styleUrls: ['./example.component.css']
})
export class ExampleComponent {
  // Component logic goes here
}
```

Pipes

Pipes are a function in Angular that allows you to transform and format data without delay inside the template. They are used to modify the appearance of statistics before it is exhibited to the user. Angular affords built-in pipes for formatting dates, numbers, and text, and you may also create custom pipes to shape unique software desires. Pipes assist in maintaining clean and readable template code by means of abstracting statistics transformation common sense.

- Javascript

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


@Component({
  selector: 'app-date-example',
  template: '<p>Today is {{ today | date }}</p>'
})
export class DateExampleComponent {
  today: Date = new Date();
}
```

Routing

Angular router allows the advent of single-page packages (SPAs) by permitting customers to navigate between distinct views or additives without complete web page reloads. The router maps URLs to perspectives and manages the nation of the utility's UI. It supports capabilities like route parameters, slow loading, guards (for defensive routes), and more.

- Javascript

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes }
  from '@angular/router';
import { HomeComponent }
  from './home.component';
import { AboutComponent }
  from './about.component';


const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent }
```

```
];


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

(This article is sourced from: https://www.geeksforgeeks.org/what-are-the-main-building-blocks-of-an-angular-application/ and the content of the article is here in case of link breakage.)