Angular Interview Questions and Answers

A developer's guide to interview success from Angular 2 to 20

2nd Edition

Anil Singh

www.bpbonline.com

Second Revised and Updated Edition 2026

First Edition 2018

ISBN: 978-93-65899-528

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true and correct to the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but the publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Dedicated to

This book is dedicated to all the passionate learners, aspiring developers, and seasoned professionals who strive to grow, innovate, and make a difference through technology.

To my family and mentors—thank you for your unwavering support, encouragement, and belief in my journey. Your guidance has been the foundation of my success. In moments of doubt, your faith reminded me why I started, and in moments of triumph,your presence made it meaningful.

To the global developer community—may this book serve as a stepping stone in your path to mastering Angular and achieving your career goals. May it inspire confidence, spark curiosity, and remind you that every challenge you overcome is a testament to your resilience and passion.

About the Author

Anil Singh is a seasoned solution architect with over 15+ years of experience in the software industry, currently working with a leading U.S. MNC. His expertise spans enterprise architecture, cloud solutions, and full-stack development, with a strong focus on modern web technologies.

Anil holds a B.Sc. in mathematics, a **master of computer applications** (**MCA**), and is a **Microsoft Certified Professional** (**MCP**). His academic foundation and professional journey have equipped him with a deep understanding of both theoretical and practical aspects of software engineering.

He is the author of the book Angular Interview Questions and Answers, a comprehensive guide designed for students, software engineers, tech leaders, and architects. The book distills years of hands-on experience and technical insight into a practical resource for mastering Angular and excelling in technical interviews.

Anil is also the founder of Code-Sample.com, a popular platform where he shares tutorials, code snippets, and architectural best practices to help developers grow their skills and stay current with industry trends.

Whether you are preparing for an interview or looking to deepen your understanding of Angular, Anil's work offers clarity, relevance, and actionable knowledge.

About the Reviewers

- **Akash Chourasia** is an IT professional with over 8 years of experience, specializing in frontend development, particularly with Angular. He has a strong background in building scalable, high-performance web applications, and is passionate about using modern frontend frameworks to create seamless user experiences.

  Akash is dedicated to best practices in UI/UX design, clean code, and performance optimization. He enjoys exploring new technologies, sharing knowledge with the community, and reading up on the latest trends in software development.

- **Chandrani Mukherjee** is a senior enterprise architect at Mphasis, where she designs AI applications that enhance productivity and reduce latency. Since 2024, she has also mentored interns and colleagues while representing the company at technology forums and events. Her expertise spans Python, LangChain, generative AI, vector databases, and large language models, including Google Gemini and AWS Bedrock LLAMA.

  Before joining Mphasis, Chandrani worked as an AI full-stack architect at McKesson in 2024 and as a data analytics and AI consultant at First Abu Dhabi Bank from 2022 to 2023. She previously contributed as an application and data engineer at Etisalat (2018–2022), a platform security developer with OSN (2018), a senior software engineer at Hewlett-Packard Enterprise (2016–2017), and a systems engineer at Tata Consultancy Services (2011–2016).

  Chandrani earned her B.Tech. in information technology from Netaji Subhash Engineering College and an MSc in machine learning and AI from Liverpool John Moores University. She also holds certifications in generative AI fundamentals.

  A senior member of the Society of Women Engineers, Chandrani actively promotes women's advancement in technology. Recognized with an Award for Excellence, she aspires to grow into leadership roles while continuing to contribute research and thought leadership in AI.

- **Neha Bhargava** is a full-stack engineer with over 10 years of experience building scalable web applications and developer tools. She has deep expertise in modern frontend frameworks, including Angular and React, and works extensively across backend systems using Node.js and cloud platforms. Her focus areas include performance optimization, developer experience, and clean architecture.

  Neha is passionate about creating tools and experiences that help developers build faster and more confidently. She has led multiple cross-functional projects aimed at simplifying complex workflows and improving integration experiences. As a technical reviewer, she brings a sharp eye for clarity, accuracy, and practical relevance. Outside of work, she mentors engineers and actively contributes to the developer community.

Acknowledgement

Writing this book has been a journey of reflection, learning, and growth, and it would not have been possible without the support of many incredible people.

First and foremost, I would like to thank my family for their endless patience, love, and encouragement. Your belief in me gave me the strength to keep going, even during the most challenging moments.

To my mentors and colleagues, thank you for sharing your wisdom, challenging my thinking, and inspiring me to strive for excellence. Your insights and feedback have been instrumental in shaping the content and direction of this book.

A special thanks to the developer community—your questions, discussions, and shared knowledge have been a constant source of inspiration. This book is a reflection of the collaborative spirit that drives our industry forward.

To the readers, learners, and professionals who pick up this book—thank you for trusting me to be a part of your journey. I hope this book serves as a valuable resource in your preparation and growth.

Lastly, I am grateful to the team at Code-Sample.com and everyone who contributed directly or indirectly to this project. Your support made this vision a reality.

Preface

In the dynamic world of software development, staying current with frameworks like Angular is more than a technical requirement; it is a gateway to building scalable, efficient, and modern web applications. Angular has become a cornerstone in front-end development, and with its growing adoption, the demand for skilled Angular developers continues to rise.

Over the course of my 15+ years in the software industry, I have had the opportunity to work with diverse technologies, lead architectural initiatives, and mentor developers across various stages of their careers. One consistent challenge I have observed is the pressure professionals face when preparing for technical interviews—especially when trying to quickly review and consolidate their knowledge. This book was created to address that challenge.

This book is designed as a practical, easy-to-navigate guide for students, software engineers, tech leaders, and software architects. Whether you are preparing for your first interview or brushing up before a senior-level discussion, this book offers a structured way to review key Angular concepts, understand common interview patterns, and build confidence.

Each question is crafted to reflect real-world scenarios and interview expectations. The answers are concise yet comprehensive, aiming to not only help you recall information but also understand the reasoning behind it. The goal is to make your preparation efficient, insightful, and empowering.

Also, this book introduces AI-enabled concepts and practices that are becoming increasingly relevant in modern development workflows, helping you stay ahead in a rapidly evolving tech landscape.

I also want this book to serve as a reminder: interviews are not just about technical correctness; they are about clarity, communication, and confidence. With the right preparation and mindset, you can turn every interview into an opportunity to showcase your skills and passion.

Thank you for choosing this book as part of your journey. I hope it becomes a valuable companion in your growth as a developer.

Chapter 1: **The Basic Concepts of Angular** - Explores the essential concepts of Angular, laying the foundation for building dynamic web applications. You will learn about Angular's core architecture, including modules, components, and templates, and how they work together to create a seamless user experience. We will introduce TypeScript, which powers Angular with statically typed and advanced tooling. It will guide you through the installation and setup of Angular using the Angular **Command Line Interface** (**CLI**) and explain how Angular's modules and components help structure your application.

Chapter 2: **Concepts of Components** - Explores the fundamental concept of components in Angular, which serve as the building blocks of any Angular application. Components encapsulate UI elements, manage data, and define application behavior, making Angular a powerful framework for developing dynamic and interactive web applications.

Chapter 3: **Concepts of Template** - Looks into Angular templates, the declarative HTML-based syntax that defines the structure and behavior of the user interface. You will learn how to use interpolation, property binding, and event binding to create dynamic views, and how structural directives help control rendering logic.

Chapter 4: **Concepts of Directives** - Angular directives allow developers to extend HTML with custom behavior. This chapter explains built-in directives such as ngClass, ngStyle, and ngModel, and guides you through creating custom directives to encapsulate reusable logic and manipulate the DOM efficiently.

Chapter 5: **Concepts of Signals**- Shows how signals enable fine-grained reactivity, automatically tracking dependencies and updating the UI with precision. The chapter also explores computed signals, which derive values from other signals, and effects, which react to signal changes without cluttering component logic.

Chapter 6: **Concepts of Dependency Injection** - Covers providers, injectors, and hierarchical DI, showing how services are injected into components and other services to promote loose coupling.

Chapter 7: **Concepts of Routing** - Explains how to configure routes, use route parameters, and implement guards to control access based on user roles or authentication status. You will also learn how to leverage lazy loading to split your application into smaller modules, improving load time and scalability. It covers nested routes, route resolvers, and dynamic navigation, helping you build complex, multi-view applications with clean and maintainable routing logic.

Chapter 8: **Concepts of Forms** – Compares template-driven and reactive forms, explores form validation, dynamic form controls, and best practices for managing user input and form state.

Chapter 9: **Concepts of HTTP Client** - Introduces Angular's HttpClient module, demonstrating how to perform CRUD operations, handle errors, use interceptors, and work with observables for asynchronous data.

Chapter 10: **Concepts of SSR and Hybrid Rendering** - Introduces Angular Universal, the official tool for SSR, and explains how it enables pre-rendering of pages on the server before sending them to the browser. You will also explore hydration strategies, which allow client-side interactivity to take over after server-rendered content is loaded seamlessly. The chapter discusses partial hydration, deferred loading, and how to balance rendering between server and client for optimal user experience.

Chapter 11: **Concepts of Pipes** - Covers built-in pipes like date, currency, and async, and shows how to create custom pipes for advanced formatting and filtering.

Chapter 12: **Concepts of NgModules** - Explains how to define root and feature modules, manage imports and exports, and use shared modules to promote code reuse and maintainability. However, it is important to note that in the latest versions of Angular, the framework has introduced standalone components and optional NgModules, offering a more streamlined and flexible approach to application architecture. While NgModules are still supported, developers are encouraged to explore these newer patterns for simpler and more modern Angular development.

Chapter 13: **Concepts of Internationalization** - Introduces Angular's i18n capabilities, including translation files, locale data, and tools for managing multilingual content and formatting.

Chapter 14: **Angular Security** - Discusses common vulnerabilities like XSS and CSRF, Angular's built-in protections, and best practices for authentication, authorization, and secure data handling.

Chapter 15: **RxJS Concepts with Angular** - Introduces observables, Subjects, and key operators like map, filter, and switchMap, showing how they simplify complex workflows. You will learn how RxJS integrates with Angular's forms, HTTP client, and component lifecycle to build responsive, scalable applications.

Chapter 16: **AI Experimental Features** - Explores cutting-edge features such as predictive UI behavior, where interfaces adapt intelligently to user actions, and smart data handling, which leverages AI to optimize data flow and decision-making. You will also discover tools that offer automated code suggestions, error prediction, and performance tuning, helping developers write cleaner, faster code with less effort.

Chapter 17: **Compiler and Build Tools** - Covers **ahead-of-time** (**AOT**) compilation, Webpack, Vite, and build configurations that streamline deployment and improve load times.

Chapter 18: **Developer Tools** - Introduces Angular DevTools, CLI utilities, and browser extensions that help developers inspect component trees, monitor performance, and troubleshoot issues.

Chapter 19: **Angular Best Practices** - Shares best practices for architecture, naming conventions, folder structure, performance optimization, and maintainability to ensure long-term success.

Chapter 20: **Angular Testing** - Covers unit testing with Jasmine, end-to-end testing with Protractor or Cypress, mocking dependencies, and strategies for achieving high test coverage.

Chapter 21: **Angular Material** - Explores layout systems, responsive design, accessibility features, and how to build visually appealing interfaces with minimal effort.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/b7e401

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/Angular-Interview-Questions-and-Answers-2nd-Edition**. In case there's an update to the code, it will be updated on the existing GitHub repository.
We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at: **errata@bpbonline.com**

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our Discord space

Join our Discord workspace for latest updates, offers, tech happenings around the world, new releases, and sessions with the authors:

https://discord.bpbonline.com

Table of Contents

Chapter 1
The Basic Concepts of Angular

Introduction

In this chapter, we will explore the essential concepts of Angular, laying the foundation for building dynamic web applications. You will learn about Angular's core architecture, including modules, components, and templates, and how they work together to create a seamless user experience. We will introduce TypeScript, which powers Angular with a statically typed language and advanced tooling.

Additionally, this chapter will guide you through the installation and setup of Angular using the Angular **Command Line Interface** (**CLI**) and explain how Angular's modules and components help structure your application. By the end of this chapter, you will have a comprehensive understanding of Angular's basic concepts and be prepared to explore more advanced topics in Angular, such as routing, state management, and testing, in the upcoming chapters.

Structure

This chapter covers the following topics:

- High-level architecture of Angular applications
- Basic Angular
- Installation of Angular
- Basic TypeScript
- Angular architecture

Objectives

The primary objective of this chapter is to offer a comprehensive introduction to Angular, a widely adopted framework for building dynamic, modern web applications. You will begin by understanding what Angular is and why it stands out as a preferred choice for developers worldwide. We will walk you through installing Angular using the Angular CLI and setting up your first project. Alongside, you will be introduced to TypeScript, the language that powers Angular, highlighting its key features, such as statically typed and object-

oriented programming benefits. This chapter also explores Angular's architecture, including core building blocks like modules, components, templates, and services, and how they collaborate to form a functional application. Furthermore, you will gain insights into the structure of an Angular project, learning the purpose and organization of essential files and directories. By the end of this chapter, you will have a strong grasp of Angular's fundamental concepts, setting the stage for deeper exploration in the upcoming chapters.

High-level architecture of Angular applications

The following figure illustrates the high-level architecture of an Angular application, showcasing how its core building blocks, such as components, modules, services, and routing, interact within the framework:

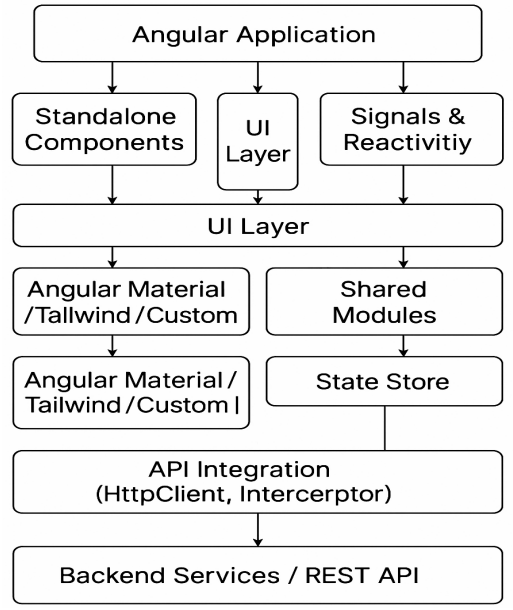

***Figure 1.1**: Architecture of Angular application*

Basic Angular

Question 1: **What is Angular?**

Answer: Angular is a powerful and versatile framework for building modern web applications. It provides a structured, scalable, and high-performance development environment. Some of its key features include:

- **Component-based architecture**: Promotes modularity and reusability by organizing the UI into cohesive, independent components.
- **Data binding**: Enables dynamic interaction between the component logic and the user interface.
- **Dependency injection (DI)**: Simplifies service management and promotes clean, testable code.
- **Routing**: Facilitates navigation and the creation of single-page applications with multiple views.

It is a TypeScript-based, open-source web application framework developed by *Google*. It is designed to build dynamic **single-page applications** (**SPAs**) with a component-based architecture. Angular offers a complete solution for developing scalable applications by providing built-in support for features like DI, data binding, routing, form management, and HTTP services. Its ecosystem includes the Angular CLI, which simplifies project setup, scaffolding, testing, and deployment.

Consider an example. When you run the following command, Angular CLI creates a new project with a standard file structure:

```
ng new my-angular-app
```

In Angular versions prior to v14, this command generated a root module (AppModule) and a root component (AppComponent), along with configurations for TypeScript, Webpack, and other tooling.

However, starting with Angular 14, and more prominently from Angular 17 onward, Angular introduced standalone components. By Angular 19, standalone components became the default, and creating a root module (AppModule) is no longer necessary unless explicitly specified.

Now, the project scaffold includes a standalone root component instead of a module-based setup. This shift simplifies Angular's architecture and aligns it more closely with modern frontend trends, making it easier for developers to start building applications without boilerplate module declarations.

It remains one of the most popular frameworks for front-end development. As of 2025, the latest stable version has been released, which introduces several of the following enhancements:

- **Incremental hydration**: Improves server-side rendering by allowing parts of the page to be hydrated incrementally, enhancing performance.
- **Route-level render mode**: Provides more granular control over how routes are rendered, allowing for optimized loading strategies.
- **Linked signals**: Introduces a new reactive programming model, enabling more efficient state management and data flow.
- **Event replay**: Enhances debugging capabilities by allowing developers to replay events in the application, aiding in troubleshooting.
- **Enhanced server-side rendering (SSR) with incremental hydration**: Angular 19 introduces incremental hydration, allowing parts of the page to be hydrated incrementally, enhancing performance. This approach enables faster initial page loads and smoother interactions by progressively activating server-rendered content on the client side.
- **Modernizing code with language service**: Improves the development experience by providing better code suggestions and error checking through the Angular Language Service.
- **Hot module replacement (HMR)**: Allows faster development cycles by enabling modules to be replaced without a full page reload.
- **Simplified management of micro frontend architectures**: Angular 19 simplifies the management of micro frontend architectures, allowing more efficient integration and development workflows. This enhancement facilitates the creation of scalable and maintainable applications by promoting modular development practices.
- **Standalone components by default**: A standalone component can exist without being part of any NgModule. Typically, in Angular prior to version 19, every component, directive, and pipe had to be declared inside an NgModule before it could be used. Standalone components eliminate the need for this structure and make the development process more flexible. The following are the benefits of standalone components:
    - **Reduced boilerplate code**: You no longer need to declare components in an NgModule to use them in other parts of the application.
    - **Improved flexibility**: Components, directives, and pipes can be used more independently, especially in smaller applications or micro-frontend architectures.
    - **Easier testing**: Standalone components can be tested individually without worrying about module dependencies.
    - **Cleaner and simpler application structure**: Developers can write cleaner code and structure smaller applications more easily.

  Above Angular 18, there is no need for the standalone as true declaration of the components is naturally standalone.

Example of a simple Angular component: The following is a basic example of an Angular component:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `
    <h1>{{ title }}</h1>
    <button (click)="changeTitle()">Change Title</button>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Hello, Angular 19!';
  changeTitle() {
    this.title = 'Title Changed!';
  }
}
```

In the preceding example, the **`AppComponent`** class defines a title property and a **`changeTitle()`** method. The component's template binds the title property to an **`<h1>`** element using **interpolation** (**`{{ title }}`**), and sets up a **click event binding** (**`(click)="changeTitle()"`**) on a button to invoke the method.

This demonstrates Angular's declarative approach to building user interfaces, where the view is automatically updated when the underlying data (model) changes. It also highlights two of Angular's core binding techniques: interpolation for displaying data and event binding for responding to user interactions.

Question 2: **What are the main advantages of using Angular?**

Answer: Angular's advantages include the following:

- **Modularity**: Code is organized into modules (NgModules) that group related functionality. Thus, it is easy to maintain and scale applications.
- **Component-based architecture**: **User interface** (**UI**) elements are encapsulated as components, making them reusable and testable.
- **Two-way data binding**: Synchronizes the model and view, reducing boilerplate code.
- **DI**: Improves modularity and makes components easier to test by providing required services.
- **Built-in tools**: Angular CLI, RxJS integration, and AOT compilation improve developer productivity and performance.
- **Seamless integration**: Works well with RxJS, Firebase, GraphQL.
- **Improved performance**: Faster rendering and change detection. Optimized change detection, lazy loading, AOT compilation
- **Better DI**: More efficient service management.
- **Enhanced forms application programming interface (API)**: Improved form handling and validation.
- **Angular signals**: A new way to manage reactive state.
- **Strong community support**: Backed by *Google* with frequent updates.
- **Security**: Built-in protections against **cross-site scripting** (**XSS**)

For example: Using DI, you can inject a service into a component without manually instantiating it by running the following code:

```
@Injectable({ providedIn: 'root' })
export class DataService {
  getData() {
    return [1, 2, 3];
  }
}
@Component({
  selector: 'app-numbers',
  template: `<div *ngFor="let num of numbers">{{ num }}</div>`
})
export class NumbersComponent implements OnInit {
  numbers: number[];
  constructor(private dataService: DataService) {}  // Dependency injected
  ngOnInit() {
    // Angular automatically calls ngOnInit when the component is initialized.
    // It is typically used to perform initialization logic like fetching data.
    this.numbers = this.dataService.getData();
  }
}
```

Question 3: **What are standalone components in Angular?**

Answer: Standalone components allow you to create Angular components without needing an NgModule.

Consider the following example:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  standalone: true,
  template: `<h1>Hello Angular!</h1>`,
})
export class AppComponent {}
```

Note: The standalone: true property marks the component as a standalone component, meaning it does not need to be declared within an NgModule. This simplifies the application structure and is the recommended approach starting from Angular 17 and above.

Installation of Angular

Question 4: **How do you install Angular?**

Answer: You can get started with Angular quickly with online starters or locally with your terminal. If you are starting a new project, you will most likely want to create a local project so that you can use tooling such as Git.

The following prerequisites must be met before installing Angular on your device:

- **Node.js**: V18.19.1 or later (compatible with Angular 19 and above).
- **Text editor**: Visual Studio Code is recommended.
- **Terminal**: Required for running Angular CLI commands.

- **Development tool**: To improve your development workflow, Angular Language Service is recommended.

The following is the step-by-step installation guide for the latest version of Angular:

1. **Install Node.js and npm**: Angular requires Node.js and **Node Package Manager** (**npm**). To check if Node.js is installed, open a terminal and run the following:

   ```
   node -v
   ```

   ```
   npm -v
   ```

   If they are installed, you will see the version numbers on your screen.

   To install or update Node.js, perform the following:

   a. Download and install the **long-term support** (**LTS**) version from the official website of Node.js (**https://nodejs.org/en**).

   b. Restart the terminal after installation.

2. **Install Angular CLI**: Angular CLI helps in creating and managing Angular projects. To install and verify the installation, perform the following:

   a. Install Angular CLI globally by running the command:

   ```
   npm install -g @angular/cli
   ```

   b. Verify Angular CLI installation by running the following command, which displays the installed versions of the Angular CLI, Angular packages, and your development environment:

   ```
   ng version
   ```

3. **Create a new Angular project**: Once the CLI is installed, you can create a new Angular project by using the command:

   ```
   ng new my-angular-app
   ```

   Replace `my-angular-app` with your project name.

   It will prompt you to select additional options like routing and styling preferences (CSS, SCSS, etc.).

4. **Navigate to the project folder**: Run the command `cd my-angular-app`.

5. **Serve the Angular application**: To start a local development server, run the command:

   ```
   ng serve
   ```

   The default server runs at **http://localhost:4200/**. Open this URL in a web browser to view your Angular application.

6. **Upgrade to the latest Angular version**: This ensures compatibility with the newest features, performance improvements, and security updates.

   If Angular new version is released and you want to upgrade your project, run:

   ```
   ng update @angular/cli @angular/core
   ```

   This will update your Angular CLI and core packages to the latest version.

7. **Build the project for production**: To build an optimized version of your application for deployment, run:

   ```
   ng build --configuration=production
   ```

   This generates the `dist/` folder, containing the optimized production-ready files.

The following are optional steps you can perform:

1. **Installing additional dependencies**: For example, if your project needs Angular Material, run:

   ```
   ng add @angular/material
   ```

2. **Enabling standalone components**: If you are using standalone components instead of NgModule, ensure your component is marked with:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: ‘./app.component.html’,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-angular-app';
}
```

Question 5: **How do you check the installed Angular version?**

Answer: To check the version, run the command **`ng version`**. This will display the Angular CLI version and project dependencies.

Question 6: **What is the difference between ng serve and ng build?**

Answer: **`ng serve`** runs the application in development mode with live reloading. On the other hand, **`ng build`** creates an optimized production build inside the **`dist/`** folder.

Basic TypeScript

Question 7: **What is TypeScript, and why is it used in Angular?**

Answer: TypeScript is a superset of JavaScript that adds static typing, classes, interfaces, and other features. TypeScript compiles into standard JavaScript that is compatible with all browsers. Angular uses TypeScript because its static typing and advanced tooling (such as IntelliSense, compile time checking, and refactoring support) improve code maintainability and catch errors early during development.

TypeScript is used in Angular because it provides the following:

- **Static typing**: Helps catch errors at compile-time rather than runtime.
- **Better code maintainability**: Interfaces, classes, and strong typing improve code structure.
- **Object-oriented programming (OOP) features**: Supports classes, interfaces, and inheritance, making Angular more structured.
- **Improved tooling and IDE support**: It has features like autocompletion, refactoring, and IntelliSense.
- **ES6+ features**: TypeScript supports modern JavaScript features, making Angular development easier.

An example of a simple TypeScript class is as follows:

```
export class Person {
  constructor(public name: string, public age: number) {}
  greet(): string {
    return `Hello, my name is ${this.name}`;
  }
}
```

In Angular, you write components, services, and modules in TypeScript for a better developer experience.

Question 8: **What are the key features of TypeScript that benefit Angular?**

Answer: The following are some key TypeScript features that benefit Angular:

- **Interfaces**: Helps define object structures for better code consistency.
- **Strong typing**: Reduces runtime errors by enforcing variable types.
- **Decorators**: Used to define Angular components, services, and directives.
- **Classes and inheritance**: Supports OOP concepts.
- **Generics**: Allows reusable, type-safe functions and classes.
- **Modules and namespaces**: Helps organize code and avoid naming conflicts.

Question 9: **What are decorators in TypeScript, and how are they used in Angular?**

Answer: Decorators are special TypeScript functions used to modify classes, properties, or methods. In Angular, decorators are used for defining components, services, and directives.

For example:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-example',
  template: `<h1>Hello, Angular!</h1>`,
  styleUrls: ['./example.component.css']
})
export class ExampleComponent {}
```

Here, **`@Component`** is a decorator that marks **`ExampleComponent`** as an Angular component.

Question 10: **Can we use JavaScript instead of TypeScript in Angular?**

Answer: Technically, yes. However, Angular is built with TypeScript, and using JavaScript would mean losing the following:

- Type safety
- Decorators (like **`@Component`**, **`@Injectable`**)
- Better code organization (interfaces, modules)
- Enhanced development experience (**integrated development environment** (**IDE**) support, autocompletion)

For these reasons, TypeScript is the recommended language for Angular applications.

Question 11: **How does TypeScript improve performance in Angular applications?**

Answer: TypeScript improves performance in Angular applications by doing the following:

- **Compile-time error checking**: Detects issues before execution, reducing runtime crashes.
- **Code optimization**: Helps generate efficient JavaScript for better performance.
- **Predictable code execution**: Static typing ensures fewer unexpected errors.
- **Tree-shaking and dead code elimination**: Removes unused code, making the final bundle smaller.

Question 12: **What is the difference between TypeScript and JavaScript in Angular?**

Answer: TypeScript and JavaScript have the following differences, especially in the context of Angular development:

- **Typing**: TypeScript uses **static typing**, meaning variable types are defined at compile time, which helps catch errors early. JavaScript, on the other hand, is **dynamically typed**, where types are determined at runtime, making debugging harder.

- **OOP support**: TypeScript has built-in support for classes, interfaces, and inheritance, making it easier to write structured and maintainable code. JavaScript provides limited OOP support, relying mostly on prototypes.
- **Error detection**: TypeScript detects errors at compile-time, preventing many common bugs before execution. JavaScript detects errors only at runtime, which can lead to unexpected crashes.
- **ES6+ features**: TypeScript includes modern JavaScript features like arrow functions, async/await, and destructuring, even before they are natively supported in browsers. JavaScript often requires transpilers like Babel to use newer features.
- **Angular support**: TypeScript is officially used in Angular, leveraging decorators like **`@Component`** and **`@Injectable`**. While JavaScript can technically be used, it lacks support for these features, making Angular development less efficient.

In summary, TypeScript is preferred for Angular development due to its strong typing, better tooling, error prevention, and enhanced OOP capabilities, whereas JavaScript is more prone to runtime errors and lacks many advanced features required for large-scale applications.

Question 13: **What is Angular used for?**

Answer: Angular is a TypeScript-based front-end framework used for building SPAs and **progressive web applications** (**PWAs**). It provides a component-based architecture, enabling developers to build scalable, maintainable, and modular applications.

The following are a few example use cases:

- **Enterprise web apps**: Large-scale business applications like dashboards (e.g., banking applications).
- **PWAs**: Web apps that behave like mobile apps (e.g., Twitter Lite).
- **E-commerce applications**: Online stores with dynamic product listings (e.g., Amazon, Flipkart).
- **Content management systems (CMS)**: Platforms like *WordPress* alternatives built using Angular.

Question 14: **What is the difference between Angular and other frontend frameworks like React or Vue?**

Answer: Angular is a full-fledged front-end framework developed by Google, while React and Vue are primarily libraries for building user interfaces. The key difference lies in their architecture, approach to state management, and learning curve, shown as follows:

- **Angular** is built using TypeScript and follows a component-based architecture with built-in DI, form validation, and routing. It provides everything required to develop a large-scale enterprise application without needing external libraries.
- **React**, developed by *Facebook*, is a UI library that focuses on building components using **JavaScript** (**JSX**). It requires third-party libraries for features like routing and state management (e.g., React Router, Redux). React is more flexible but requires additional setup for a complete application.
- **Vue.js** is a progressive framework that is lightweight and easy to integrate. It allows developers to adopt its features gradually, making it beginner-friendly. Vue's templating syntax is similar to Angular's but simpler, and it provides built-in state management through Vuex.

The following are some use cases:

- Use **Angular** when building large-scale applications requiring built-in routing, DI, and form handling.
- Use **React** if you need a lightweight UI library with flexible state management.
- Use **Vue.js** for a simpler, faster development experience with minimal configuration.

In summary, Angular is a structured, opinionated framework best suited for large, complex applications, whereas React and Vue offer more flexibility and simplicity, making them popular for smaller, fast-paced projects. Here is a side-by-side visual comparison of the architectures of React and Vue to help you understand their core building blocks and design philosophies:

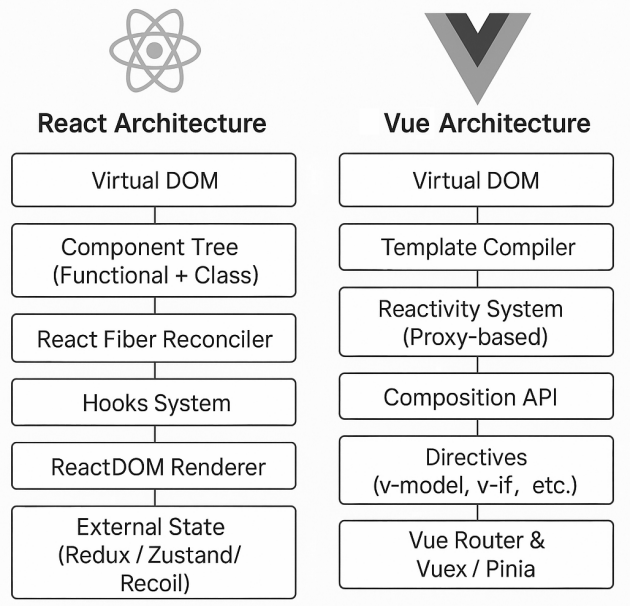

***Figure 1.2**: Architecture of React and Vue*

Angular architecture

Question 15: **What is the architecture of an Angular application?**

Answer: Angular follows a modular architecture consisting of the following:

- **Modules (NgModule)**: Defines a cohesive block of functionality (e.g., **`AppModule`**).
- **Components**: UI building blocks with a template, logic, and styling.
- **Templates**: Define HTML structure using Angular directives and bindings.
- **Directives**: Modify the behavior of DOM elements (**`ngIf`**, **`ngFor`**).
- **Services and DI**: Reusable business logic and data management.
- **Routing**: Enables navigation across different views (**`RouterModule`**).
- **Pipes**: Format and transform data (e.g., date, currency).

For example:

```
@Component({
  selector: 'app-hello',
  template: `<h1>Hello, {{ name }}</h1>`,
})
export class HelloComponent {
  name = 'Angular';
}
```

Question 16: **What is the role of Angular CLI?**

Answer: Angular CLI simplifies project setup, development, and maintenance. It provides commands for the following:

- **Create a new project**: **`ng new my-angular-app`**
- **Serve the application**: **`ng serve`**
- **Generate a new component**: **`ng generate component my-component`**