Angular Basics and MVC Architecture

KOENIG
step forward

Introduction to Angular and Its Ecosystem:

- Overview of Angular’s features and benefits
- Key concepts: Components, Directives, Services, and Modules
- Setting up an Angular project with the Angular CLI

Overview of Angular’s features and benefits

Angular is an application design framework and development platform for creating efficient and sophisticated single page application.

Angular is a popular open-source web application framework developed by Google and a community of individual developers and corporations.

It is designed to simplify the process of building dynamic, single page web applications and provides a comprehensive set of tools and features for front-end web development.

Angular is a framework for building client applications using HTML, CSS, and JavaScript / TypeScript.

Overview of Angular’s features and benefits

In short, Angular is…

- TypeScript-based, open-source framework developed and maintained by Google for building scalable and dynamic web applications.
- Complete front-end framework with built-in features like component-based architecture, dependency injection, routing, and state management.

Overview of Angular’s features and benefits

Key Features of Angular

- Component-Based Architecture

  Angular applications are built using reusable components, making the code modular and maintainable.

- Two-Way Data Binding

  Synchronizes data between the UI and the component class, reducing boilerplate code.

- Dependency Injection (DI)

  Enhances modularity and testability by managing service dependencies efficiently.

- Directives & Templates

  Built-in and custom directives allow dynamic UI manipulation and code reuse.

- Routing & Navigation

  The Angular Router enables single-page applications (SPA) with lazy loading to improve performance.

Overview of Angular's features and benefits

Key Features of Angular

- Reactive Forms & Template-Driven Forms

  Supports form validation and dynamic form controls with powerful form-handling techniques.

- HTTP Client Module

  Built-in module to handle API requests with Observables (RxJS).

- RxJS & State Management

  Uses Reactive Extensions (RxJS) for handling asynchronous operations like HTTP calls and event handling.

  State management solutions like NgRx, Akita, and NGXS provide centralized state control.

- Built-in Security & Performance Enhancements

  Sanitization for security against XSS (Cross-Site Scripting) attacks.

  Ahead-of-Time (AOT) Compilation improves loading speed.

Overview of Angular’s features and benefits

Angular Ecosystem

- The Angular ecosystem consists of tools and libraries that enhance development efficiency

Tool/Library	Purpose
Angular CLI	Automates project setup, builds, and testing.
RxJS	Handles reactive programming and asynchronous operations.
Angular Material	Pre-built UI components for faster development.
NgRx	State management using Redux principles.
Jasmine & Karma	Unit testing framework and test runner.

Installation Procedure of Angular

The Angular installation process involves a few key steps, primarily centered around Node.js, npm (or yarn), and the Angular CLI.

1. Install Node.js and npm:

Node.js provides the JavaScript runtime environment.

npm (Node Package Manager) is used to install and manage Angular and its dependencies.

Procedure:

- Go to the official Node.js website: https://nodejs.org/
- Download the LTS (Long Term Support) version, which is generally recommended for stability.
- Run the installer and follow the on-screen instructions.
- Verify the installation by opening your terminal or command prompt and running the following commands:
    - node -v (to check the Node.js version)
    - npm -v (to check the npm version)

Installation Procedure of Angular

The Angular installation process involves a few key steps, primarily centered around Node.js, npm (or yarn), and the Angular CLI.

2. Install the Angular CLI:

The Angular CLI (Command Line Interface) is a tool that simplifies Angular development.

It allows you to create new Angular projects, generate components, run tests, and more.

Procedure:

- Open your terminal or command prompt.
- Run the following command to install the Angular CLI globally:
- npm install -g @angular/cli
- This command installs the latest stable version of the Angular CLI.
- Verify the installation by running:
- ng --version

Installation Procedure of Angular

The Angular installation process involves a few key steps, primarily centered around Node.js, npm (or yarn), and the Angular CLI.

3. Create a New Angular Project:

- In your terminal or command prompt, navigate to the directory where you want to create your project.
- Run the following command, replacing "my-app" with your desired project name:

```
ng new my-app
```

- The CLI will prompt you with a few questions:
  - "Would you like to add Angular routing?" (Choose "y" for yes or "n" for no)
  - "Which stylesheet format would you like to use?" (Choose your preferred format, such as CSS, SCSS, or Sass)
- The CLI will then create a new Angular project with the specified name and configuration.

Installation Procedure of Angular

The Angular installation process involves a few key steps, primarily centered around Node.js, npm (or yarn), and the Angular CLI.

4. Run the Angular Application:

- Navigate to your project directory:

```
cd my-app
```

- Run the following command to start the development server:

```
ng serve --open or ng serve -o
```

- This command will build your application and open it in your default web browser at http://localhost:4200.
- Any changes you make to your code will be automatically reflected in the browser.

Key concepts: Components, Directives, Services, and Modules

- Angular is built on a modular and component-based architecture, allowing developers to create scalable applications.
- The four fundamental concepts in Angular are Components, Directives, Services, and Modules.
- Components
  - A Component is the basic building block of an Angular application. It controls a part of the UI and consists of:
    - Template (HTML): Defines the UI.
    - Class (TypeScript): Contains business logic and data.
    - Styles (CSS/SCSS): Defines the appearance.

Key concepts: Components, Directives, Services, and Modules

- Directives
    - Directives modify the behavior or appearance of elements in the DOM. They are classified into:
    - Structural Directives (Modify DOM structure)
        - *ngIf – Conditionally renders an element.
        - *ngFor – Loops through an array to generate elements.
        - *ngSwitch – Implements switch-case logic in templates.
    - Attribute Directives (Modify element appearance or behavior)
        - ngClass – Adds classes dynamically.
        - ngStyle – Applies styles dynamically.

Key concepts: Components, Directives, Services, and Modules

- Services
  - Services are used to share logic across components, such as data fetching, business logic, or API calls.
  - They are injected using Dependency Injection (DI).
- Modules
  - Modules help organize the application by grouping related components, directives, and services.
  - Every Angular app has at least one module: AppModule (app.module.ts).

Understanding MVC in Angular

Understanding MVC in Angular

- The role of Model, View, and Controller in Angular applications
- Structuring components and services to align with MVC principles
- Benefits of using MVC for clean and scalable applications

The role of Model, View, and Controller in Angular applications

- Angular follows a component-based architecture, but it also aligns with the Model-View-Controller (MVC) pattern for organizing applications efficiently.
- Model (M) – Data & Business Logic
  - The Model represents the data and business logic of the application.
  - It defines the structure of the data and interacts with APIs or services to fetch, update, or delete data.
- View (V) – UI & Presentation Layer
  - The View is responsible for displaying data to the user. It includes HTML templates and Angular directives to render dynamic content.

The role of Model, View, and Controller in Angular applications

- Controller (C) – Handles Logic & Communication
  - In Angular, the Component (TypeScript Class) acts as the Controller.
  - It manages data, user interactions, and communication with services.

Structuring components and services to align with MVC principles

- To follow MVC principles in Angular, we need to separate concerns effectively. In Angular:
  - Model (M) → Represents the data structure. (Eg. Product.ts)
  - View (V) → Handles the UI. (Eg. product.component.html)
  - Controller (C) → Component (Manages data and user interactions).
- To enhance maintainability, services should handle data operations separately from components.
- This ensures a clean MVC structure.

Benefits of using MVC for clean and scalable applications

- The Model-View-Controller (MVC) pattern is a widely used software design architecture that helps build clean, maintainable, and scalable applications.
- In Angular, implementing MVC principles leads to a structured approach for handling data, UI, and logic separately.
- Separation of Concerns (SoC)
  - Each part of the application (Model, View, and Controller) has a clear responsibility, reducing code complexity.
- Scalability
  - MVC makes it easier to extend the application as requirements grow.

Benefits of using MVC for clean and scalable applications

- Reusability & Modular Design
  - With MVC, different components and services are reusable across the application.
- Maintainability & Testability
  - Applications built with MVC are easier to test and maintain because:
  - The Model and Service handle data and logic separately from UI.
  - Unit testing can be done independently on each layer.
  - Bugs and issues are isolated, making debugging easier.
- Improved Collaboration & Development Speed
  - Teams can work on different MVC layers independently, increasing productivity.

Benefits of using MVC for clean and scalable applications

- Consistent and Predictable Code Structure
  - With MVC, the project follows a well-defined structure, making it easy for new developers to understand.
- Easier Integration with APIs & Databases
  - Since the Model handles data structure, it can easily be mapped to APIs or databases.

Designing a Code Verification Application

Designing a Code Verification Application

- Gathering requirements: Expected inputs and verification criteria
- Creating a user-friendly interface for code input and results display
- Developing business logic for validation and error handling

Gathering requirements: Expected inputs and verification criteria

- Expected Inputs
  - User Identifier – Email, phone number, or username.
  - Verification Code – A one-time passcode (OTP) or security code.
  - Request Type – Registration, password reset, transaction approval, etc.
- Verification Criteria
  - Correctness – The code must match the one generated.
  - Expiration Time – The code should expire after a set time (e.g., 5 minutes).
  - Attempts Limit – Restrict invalid attempts to prevent brute-force attacks.
  - Single Use – Ensure each code is used only once.

Creating a user-friendly interface for code input and results display

- A well-designed code verification UI ensures that users can easily enter the code and see clear feedback on the verification process.
- UI Components Overview
  - Input Field for Code – Users enter the verification code.
  - Submit Button – To verify the entered code.
  - Countdown Timer – Shows expiration time for the code.
  - Resend Code Button – Allows users to request a new code after some delay.
  - Success/Error Messages – Provides feedback based on verification results.

Developing business logic for validation and error handling

- Code Validation Logic
  - We will validate the 6-digit verification code based on:
  - Correctness – The code must match the one stored in the database.
  - Expiration – The code must be valid within a specific time (e.g., 5 minutes).
  - Attempts Limit – Restrict the number of incorrect attempts to prevent brute-force attacks.
  - One-Time Use – Ensure the code can be used only once.

Developing business logic for validation and error handling

- Common Error Cases & Responses

Error Case	Response
Invalid Code	"Incorrect code. Try again."
Code Expired	"Your code has expired. Request a new one."
Too Many Attempts	"Too many incorrect attempts. Try again later."
Code Already Used	"This code has already been used."
System Error	"Something went wrong. Please try again."

Testing and Debugging

Testing and Debugging

- Writing unit tests for components and services using Jasmine and Karma
- Debugging and optimizing the application for better performance
- Ensuring error handling and validation are robust and user-friendly

Testing and Debugging

Angular Unit Testing

Angular unit testing is the process of testing Angular applications to ensure that they work in the desired manner and to get optimum solutions.

Testing in an Angular project typically involves writing and running various types of tests to verify the functionality, performance, and reliability of Angular components.

It can also test services, directives, pipes, and other parts of the application using angular testing. Integration testing angular applications to test frameworks ensures desired results.

Testing and Debugging

Significance of Angular Unit Testing

- Maintaining Quality
- Regression Testing
- Refactoring Support
- Documentation and Specification
- Facilitating Collaboration

Testing and Debugging

Arrange
Act
Assert

Principles of Angular Unit Testing

- **Arrange**

  It involves arranging data from selected test groups or environments for testing.

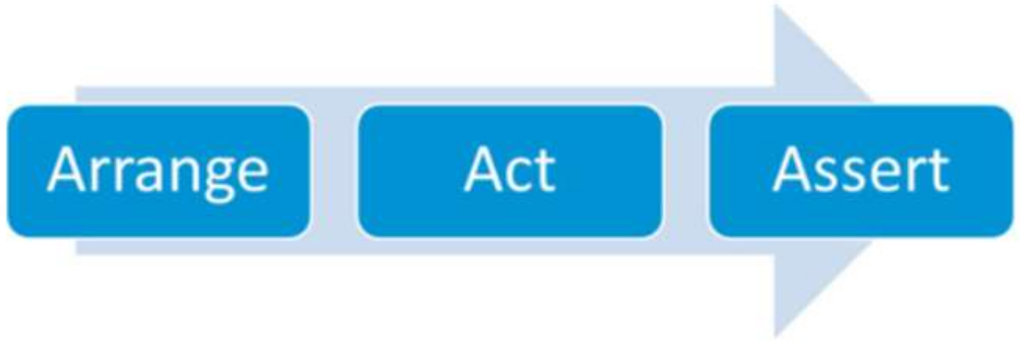
- **Act**

  It involves performing actual logic on arranged data.

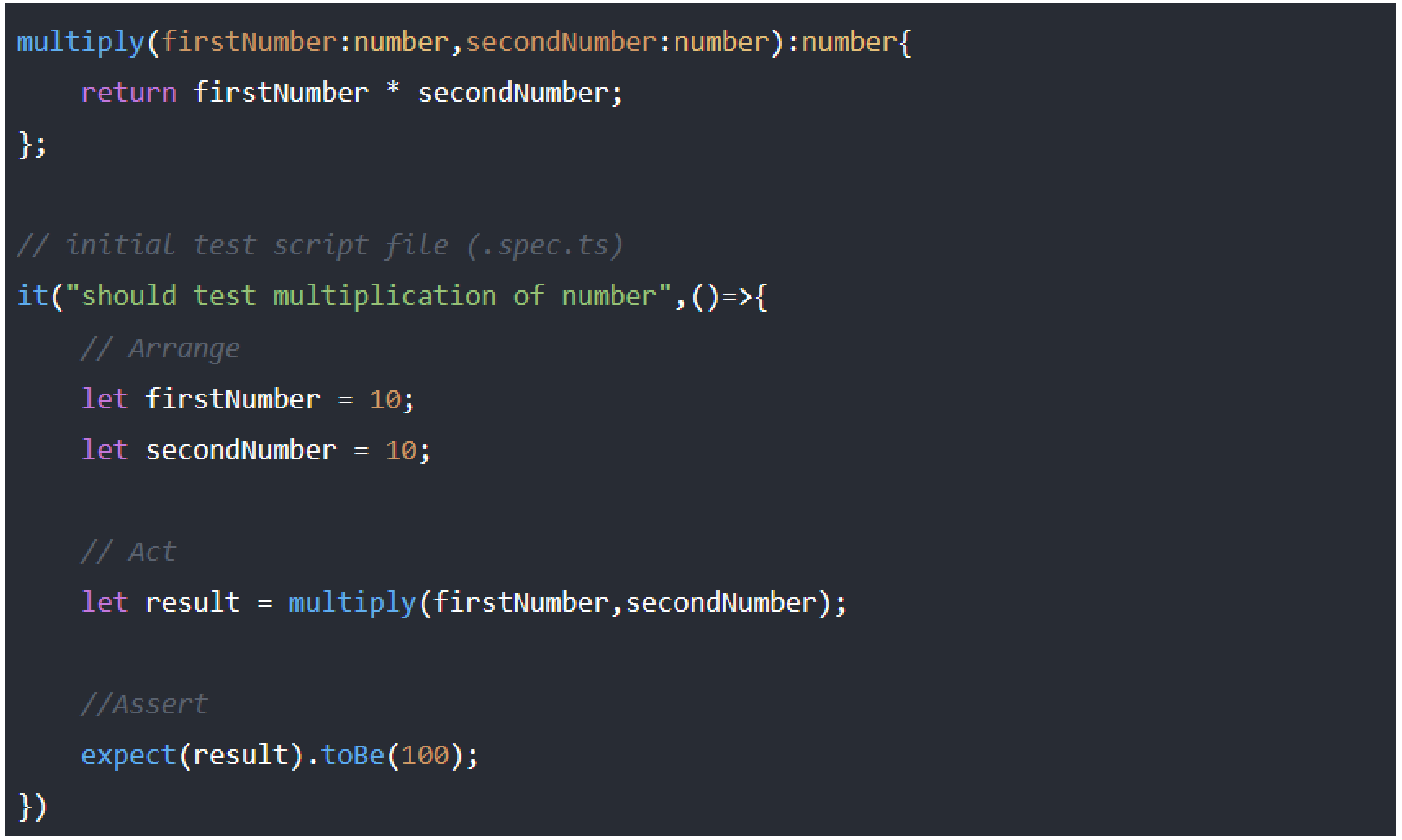
- **Assert**

  It involves the verification of test results.

Testing and Debugging



The below code shows a simple example of the AAA process**Arrange**



```
multiply(firstNumber:number,secondNumber:number):number{
    return firstNumber * secondNumber;
};

// initial test script file (.spec.ts)
it("should test multiplication of number",()=>{
    // Arrange
    let firstNumber = 10;
    let secondNumber = 10;

    // Act
    let result = multiply(firstNumber,secondNumber);

    //Assert
    expect(result).toBe(100);
})
```

Testing and Debugging

Angular uses Jasmine and Karma for unit testing purposes.

Jasmine :

Jasmine is a behavior-driven development (BDD) testing framework. It provides a test suite and helps you to write unit test cases in a human-readable format. You simply write test cases for your code to achieve a requirement.

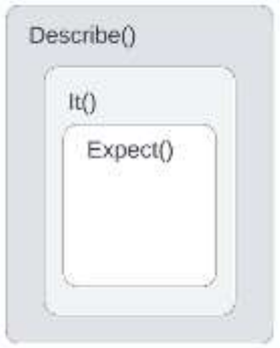
Karma :

Karma is a test runner that runs test cases that are written by using the Jasmine framework. It provides some features like code coverage, live reloading test files, and integration with continuous integration (CI) tools.

Writing First Angular Unit Test

- To write unit test cases we need a .spec.ts file. There are two ways to create a .spec test file extension is,
- It can create it by using angular cli or we can create our own test file with extension .spec.ts.
- Generally angular creates this file for you by default unless we tell the angular to don't create this file.
- When the spec.ts file is created by angular it provides us the basic skeleton of test cases.

Writing First Angular Unit Test



Describe

Describe the block used to create a group of test cases. It has two arguments first one is the identifier or name of the first test case and the second one is a function that contains test cases.

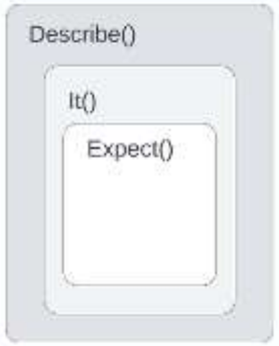
It

It block is used to write individual test cases for each functionality. It also has two arguments identifier or name of the test case and function which contains assertion.

Expect

It is used to create expectations. It takes a single value and a matcher function which tests the value against the expected value.

Writing First Angular Unit Test



```
// app.component.spec.ts test file
import { TestBed } from '@angular/core/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(() => TestBed.configureTestingModule({
    imports: [RouterTestingModule],
    declarations: [AppComponent]
  }));

  it(`should return app title`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    let result = app.getTitle();
    expect(result).toEqual('App');
  });

});
```

Building and Testing the Code Verification Application

Implementing the Angular Application:

- Working with Angular Forms for user input and validation
- Building reusable components for modularity and flexibility
- Integrating services for business logic and backend communication

Hands-On Project

Hands-On Project

- Developing a complete Angular application to verify generated codes
- Incorporating real-time feedback and validation messages
- Testing the application in different scenarios to ensure reliability