Day 19	Introduction to Angular as a Framework	
	Introduction to Angular as a Framework	Introduction to Angular Angular is Typescript based open source front end framework developed by Google for building SPA with rich user interface.
		Key features and advantages 1. Component based architecture : Easy to maintain and scale 2. Two -way data binding : sync between Model and view automatically. 3. Dependency injection : Improves Modularity and testability. 4. Directives : Extend HTML with custom behaviors 5. Routing : Create SPA navigation without page loads. 6. Crosplatform : Works for web, mobile(Ionic) and desktop
		Use cases where Angular is preferred over other frameworks : ● Large enterprise applications ● Applications require real time updates. ● Complex SPA with multiple dynamic views. ● App with high maintainability needs.
	Getting Started with Angular	**Installing Angular CLI** **Npm install -g @angular/cli**
		Understanding Just-In-Time (JIT) and Ahead-of-Time (AOT) compilation
		Creating a basic Angular application **Ng serve**
	Building Blocks of Angular Applications	Angular Modules: Container for components , directives, pipes and services. Ex Root Module : AppModule(declared inside app.module.ts) Components: Basic building blocks in angular Logic : -> .ts Template: -> .html style:-> .css Templates : inline or external
		Creating and organizing Angular modules
		Creating components and their templates
	Angular Modules, Components, and Templates	Role of components in SPA (Single Page Application) development
		Creating and organizing Angular modules
		Creating components and their templates
		Role of components in SPA (Single Page Application) development
	Angular Binding	Property binding, event binding, and two-way binding
		Practical exercises: Building dynamic templates with data binding

What is NPM , node CLI?

Installing, Updating and Removing Packages using NPM ?

Actions	Command syntax	Description	Example
Install a package locally	npm install <package-name>	Installs the package in the node_modules folder and adds it to dependencies in package.json	npm install loadash
Install a package globally	npm install -g <package-name>	Install the package systemwide so i can be used in terminal directly	npm install -g nodemon
Installing a specific version	npm insall<package-name >@ <version>	Install the specified version of the package	npm install loadash@4.17. 15
Updating a package	npm update <Package-name>	Updates the package to the latest version allowed by package.json	npm updated nodeash
Update all packages	npm update	Update all packages listed in dependencies	npm update
Remove a package	Npm uninstall <package -name>	Removes the package from node_modules and from package.json	Npm uninstall loadash
List installed packages	Npm list	Shows all installed packages and their versions	
List globally installed packages	Npm list -g –depth=0	Shows top level global packages.	
Reinstall from package.json	Npm install	Install all packages listed in package.json	

Creating a simple Node.js application ?

Implementing Async/Await/ promises in Node.Js ?

Getting started with Express js:

1. **Create a folder -> npm init -y**
2. **Install Express** npm install express

Here We create a basic server in Express and can compare it with a plain Node.js HTTP server(performance and simplicity)

Basic HTTP server using HTTP module:

Const http = require(http) // nodejs builtin http module | As it will all us to create a webserver
Why the above step ?
Here we are using HTTP for handling request and response

```
Cons server = http.createServer((req,res) =>
req.writeHead(200,{‘Contet_Type’: ‘text/plain’});
res.end(‘Hello World from HTTP Server !!’);

server.listen(3000,() => {
console.log(“server running at http://localhost:3000/”);
});
```

Feature	Node.js HTTP Server	Express.js Server	Key Difference
Importing Module	const http = require('http');	import express from "express"; *(or)* const express = require('express');	Express is an **external library** installed via npm install express, while http is built into Node.js.
Creating Server/App	const server = http.createServer((req, res) => {...});	const app = express();	In Node.js, you manually create the server; in Express, app handles it internally.
Handling Requests	server callback handles requests: (req, res) => { ... }	Route methods like app.get('/', (req, res) => {...})	Express provides **route methods** (get, post, etc.) instead of handling all requests manually.

Sending Response	res.writeHead(200, {'Content-Type': 'text/plain'}); res.end('Hello World');	res.send('Hello! Your Express server is running 🚀');	Express automatically sets headers like Content-Type for you.
Listening on Port	server.listen(3000, () => {...});	app.listen(3000, () => {...});	Both use .listen() but Express ties it directly to the app object.
Ease of Use	More manual code: handle headers, parse URLs, etc. yourself.	Minimal code: routing, middleware, JSON parsing are built-in.	Express is **simpler and more readable** for larger projects.
Dependencies	No external installation needed.	Requires installing Express via NPM.	Express adds extra features but needs installation.

Basic routing in Express Js:
GET route for / that returns " Welcome to Express"
POST route that accepts JSON data
PUT and DELETE routes

Basic routing in Express.Js

Do

---

1. **What are the steps for creating sample API using Express.Js(Backend design using Express.Js)**
2. **Getting started with API testing using Postman.**
3. **How to implement Jquery ?**
4. **Demo based on Design patterns using C# and comparing it with javascript.**

Header

Quik Links	Body

Footer

Each component in angular consist of :

1. Typescript file(.ts) with decorator @component with properties defined.
    a. Selectors - to get a css selector for the components.
    b. Template URL/template - to define the HML template of the component.
    c. styleURLs/style - to define styling of the component

Steps for creating a component :

Step 1: Generate a Component with CLI : `ng generate component mycomponent`

Or ng g c my-component // short hand

Step 2: Adding my-component to the main file ie app.component.html

```
<div>
        <app-my-component> </app-my-component>

</div>
```

Step 3: Adding content to my component

ng g c my-component	
Inline Template @component({ selector: 'app-my–component', Template: `  <div> Here is my inline template for component </div>` })	Separate file @Component({ selector:'app-my-component', templateUrl:'myComponent.component.html' }) Here is external template

Step 4: Declaring component styles

ng g c my-component --inline-style=true	
Inline - Inside .ts file **@Component({** **selector:'app-my-component',** **:`<div class="main">** **Hello World!** **</div>`,** **styles: [** **`.main{** **background: red;** **}`** **]** **})**	**In Separate File** **@Component({** **selector:'app-my-component',** **template:`<div class="main">** **Hello World!** **</div>`,** **styleUrls: ['./myComponent.component.css']** **})** **.main{** **background: red;** **}**

Node.js and angular 👍

- **Angular CLI runs on Node.js**
- **Package management is done via npm**
- **Efficient build process for frontend project**

Case study : Simple product card app
Scenario:
Imagine you are working for an e-commerce startup. The product page needs a simple card that shows the product name and price, and a button to mark it as "Added to Cart".

Step 1: Create a new angular Project (ng version), choose css for styling
Step 2: Create a component name product-card
Step 3: Add component to app ie src/app/app.component.html

```
<h1>My Store</h1>
<app-product-card></app-product-card>
```

Step 4: Implement property and Event binding
src/app/product-card/product-card.component.ts

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

@Component({
  selector: 'app-product-card',
  templateUrl: './product-card.component.html',
  styleUrls: ['./product-card.component.css']
})
export class ProductCardComponent {
  // Property Binding Example
  productName: string = 'Wireless Headphones';
  productPrice: number = 2999;

  // Event Binding Example
  addToCart() {
    alert(`${this.productName} has been added to cart!`);
  }
}
```

Step 5: Create Template with Bindings

Open `src/app/product-card/product-card.component.html`:

```
<div style="border:1px solid #ccc; padding:10px; width:250px;">
  <h2>{{ productName }}</h2> <!-- Interpolation -->
  <p>Price: ₹{{ productPrice }}</p>

  <!-- Property Binding -->
  <button [disabled]="false" (click)="addToCart()">Add to Cart</button>
</div>
```

Step 6: Run the app
Ng serve