Day 26

What are the top 5 skills that everyone should have ?

You have a skills based portfolio where you can invest daily, weekly & monthly. This investment strategy changes with time, outcome

Time management
AI & ML
Testing
Grid Computing
Soft skills
Database Skills (SQL, No SQL, Graph DB, cassandra)
Core Programming(C#, Java etc)
Basic Front End Skills (HTML, CSS, JS)
Specialized frame work : Angular, react etc
Master crypto currency
Content development skills
Voice and Accent Skills (VNA)
—---
How to get into C Level roles CFO, CTO, CEO, CLDO
Why project management ?

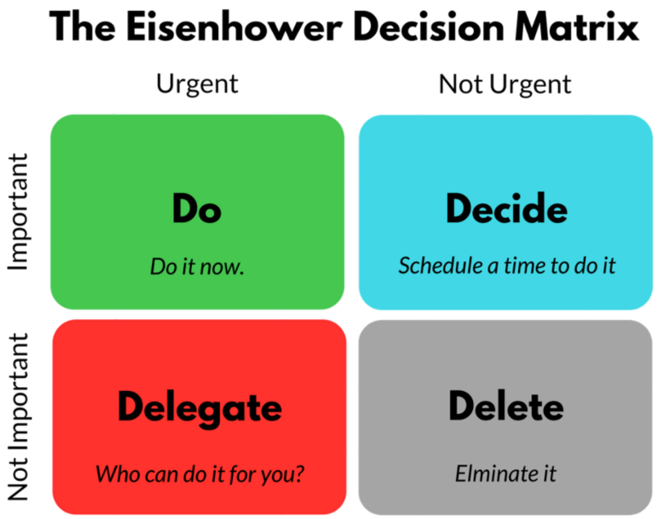
Suppose you have following things with as a resource 👍

1. Time
2. Energy
3. Money

When you are in this age 20-25, you can go bold
When you are in this age 35-40 year

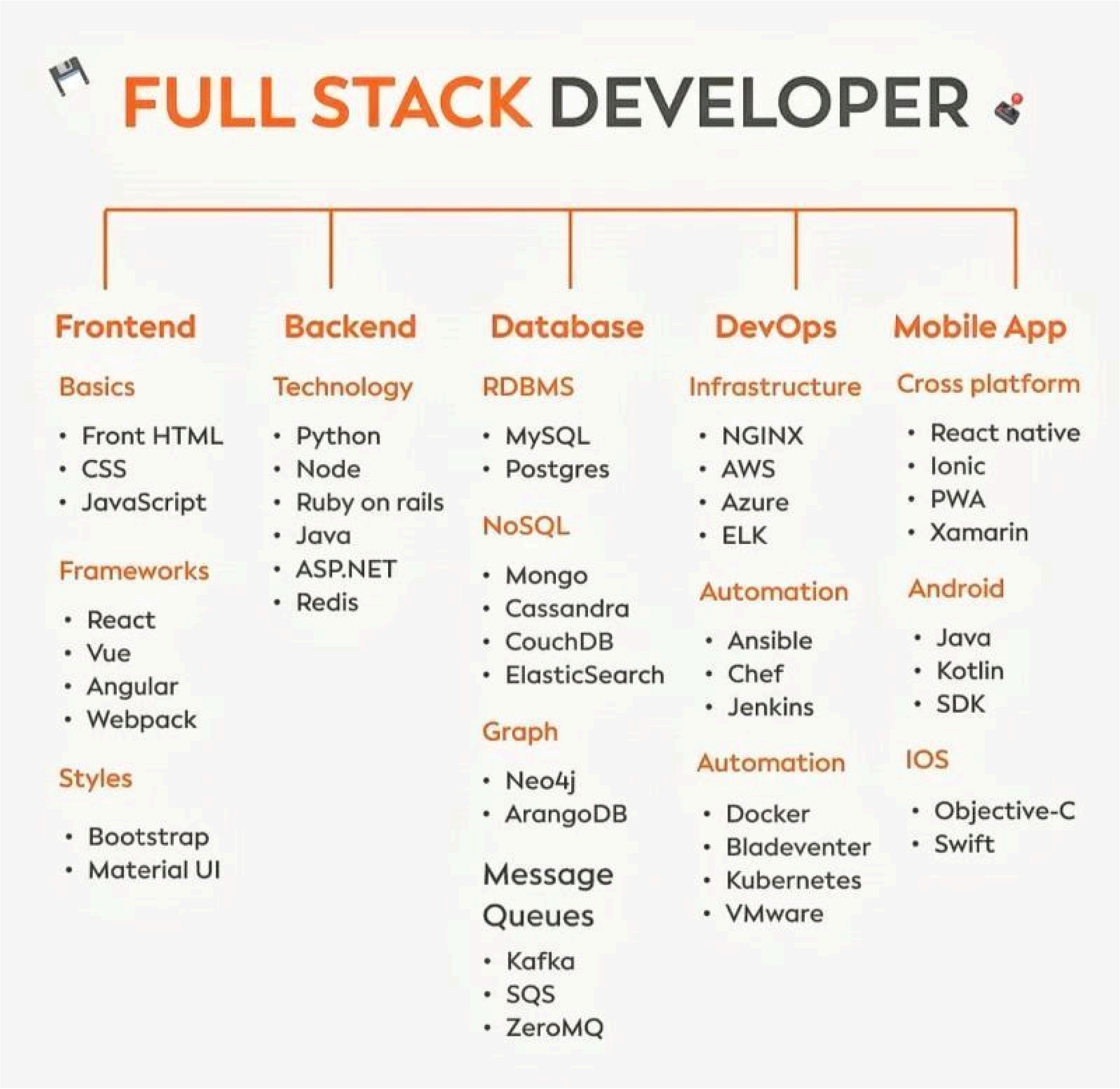
Four Categories:

1. Evergreen(No Loss & Only returns)
2. Less returns
3. High risk high returns : 10-15%
4. Not adapted by market leaders (May give returns in future)



The Eisenhower Decision Matrix

	Urgent	Not Urgent
Important	**Do** *Do it now.*	**Decide** *Schedule a time to do it*
Not Important	**Delegate** *Who can do it for you?*	**Delete** *Elminate it*



FULL STACK DEVELOPER

Frontend

Basics

- Front HTML
- CSS
- JavaScript

Frameworks

- React
- Vue
- Angular
- Webpack

Styles

- Bootstrap
- Material UI

Backend

Technology

- Python
- Node
- Ruby on rails
- Java
- ASP.NET
- Redis

Database

RDBMS

- MySQL
- Postgres

NoSQL

- Mongo
- Cassandra
- CouchDB
- ElasticSearch

Graph

- Neo4j
- ArangoDB

Message Queues

- Kafka
- SQS
- ZeroMQ

DevOps

Infrastructure

- NGINX
- AWS
- Azure
- ELK

Automation

- Ansible
- Chef
- Jenkins

Automation

- Docker
- Bladeventer
- Kubernetes
- VMware

Mobile App

Cross platform

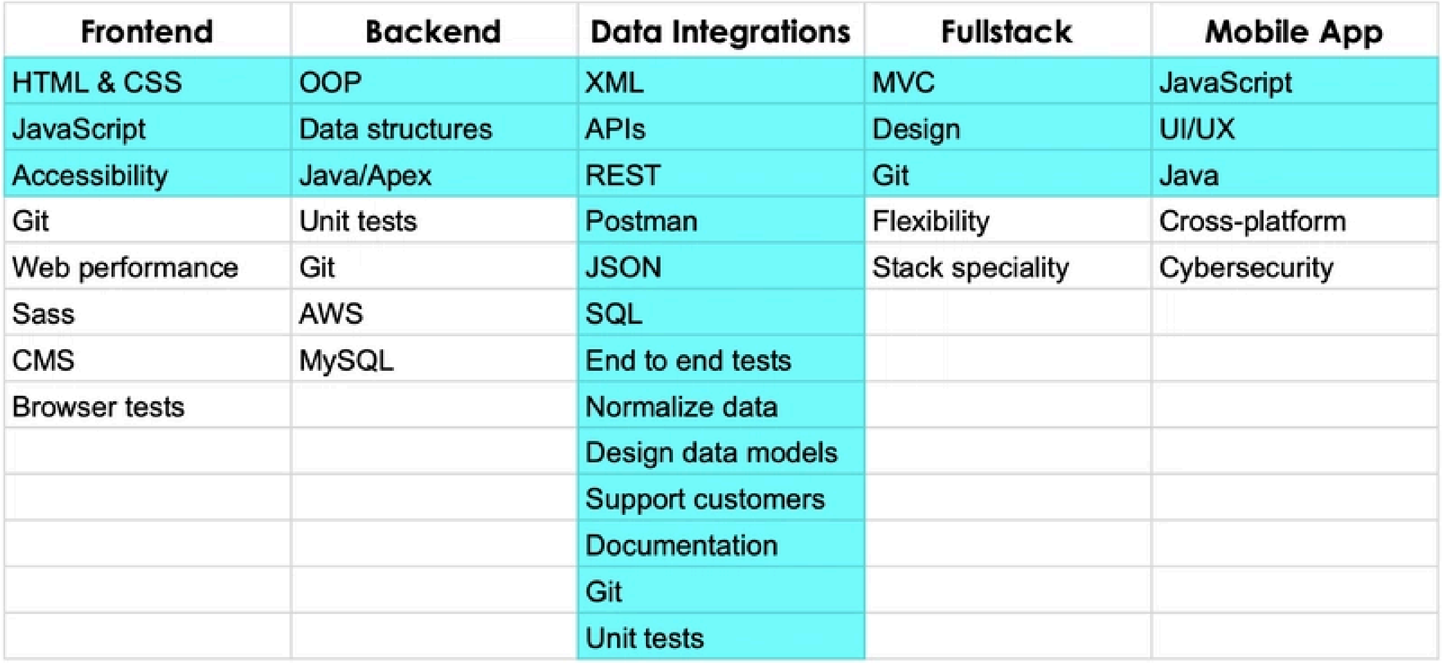
- React native
- Ionic
- PWA
- Xamarin

Android

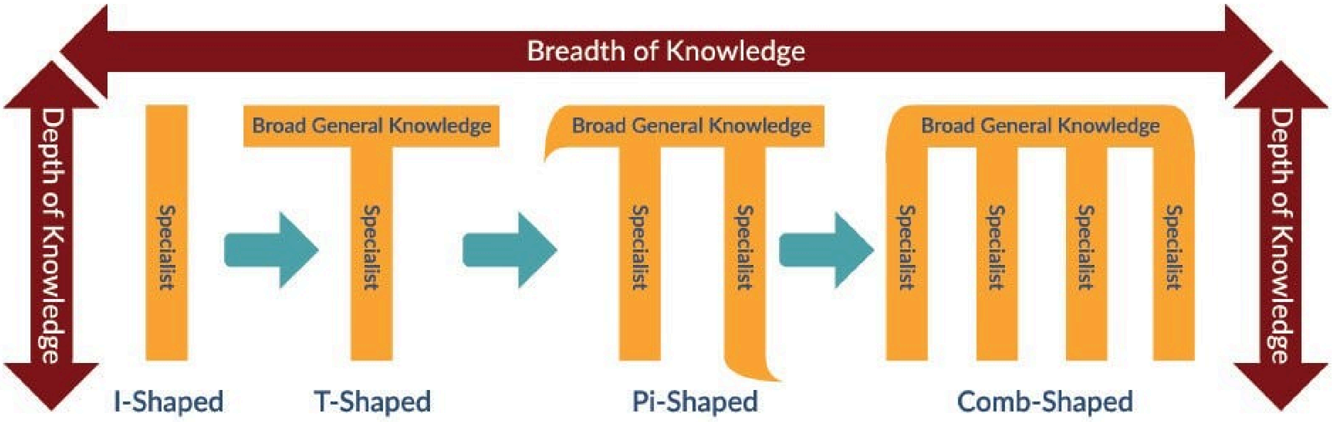
- Java
- Kotlin
- SDK

IOS

- Objective-C
- Swift



Frontend	Backend	Data Integrations	Fullstack	Mobile App
HTML & CSS	OOP	XML	MVC	JavaScript
JavaScript	Data structures	APIs	Design	UI/UX
Accessibility	Java/Apex	REST	Git	Java
Git	Unit tests	Postman	Flexibility	Cross-platform
Web performance	Git	JSON	Stack speciality	Cybersecurity
Sass	AWS	SQL		
CMS	MySQL	End to end tests		
Browser tests		Normalize data		
		Design data models		
		Support customers		
		Documentation		
		Git		
		Unit tests		

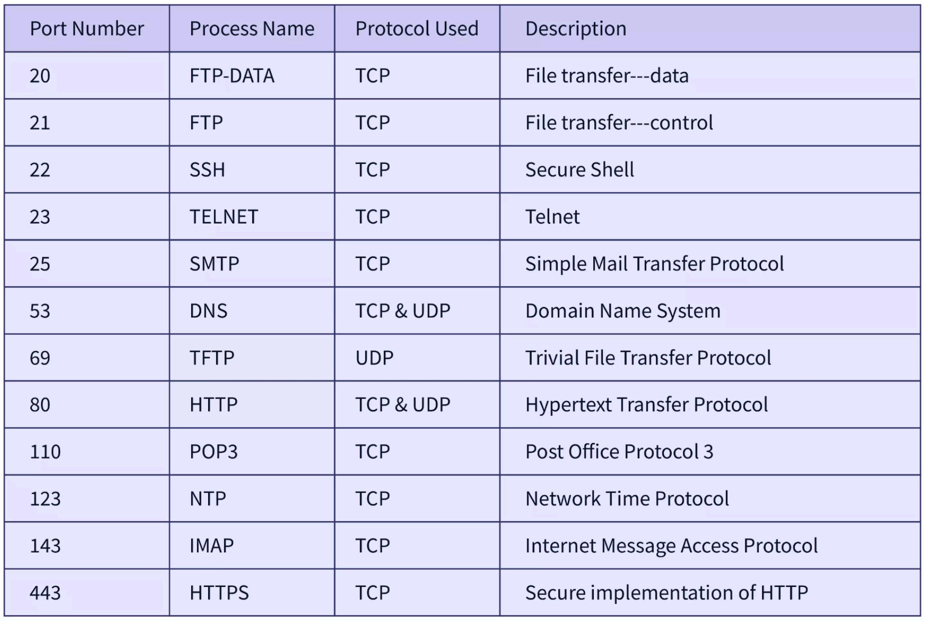


C# + HTML, CSS, JS + MSSSQL + Angular + .NET Core (MVC) Full Stack
C#
Testing in C#
APi development in C#

	.NET Core (Recap)	
Day 26	Middleware and Static Files	Understanding middleware in ASP.NET Core Middlewar are components that inspect or forwards HTTP request/response in a pipeline(authentication, routing, business logic including Transformation)
		Configuring and using middleware components ● It helps us in placing exception handling along with HTTPs redirection ● Put static files before routing and authentication before authorization

		Serving static files (HTML, CSS, JavaScript) ● All static files are served from wwwroot folder via UseStaticFiles() ● We use CDN for large assets. (like npm Packages) Demo : Serving static files via ASP.NET core application Step 1: Creating a project via cmd line ASP.NET Core Web App Step 2: Creating a basic middleware pipeline (program.cs) Step 3: Testing the middleware Step 4: Adding static files Step 5: Testing static files Request -> Middleware 1 -> Middleware 2 -> Static files -> Final response
		Security considerations with static files: 1. No authorization is applied to file under wwwroot(public) 2. Never puts secret/source code in wwwroot 3. To protect a file, do not serve it as static.
	IN ASP.NET we have multiple UI Stack (Razorpages, MVC, Blazor) Introduction to Razor Pages	Overview of Razor Pages architecture ● Razor Page is a page-focused pattern built on the MVC view engine. ● Each URL maps to a page(cs + HTMl -> .cshtml) with an optional PageModel(.cshtml.cs) ● We have handlers like Onget/Onpost process requests. Advantage of Razor page approach: 1. Page centric , less ceremony than controller for simple page flows. 2. Cleaner file co-location(.cshtml next .cshtml.cs) 3. Great for forms, CRUD and website scenarios. MVC is preferred In scenarios like Multi resource APIs, rich controllers, filters and larger app layering.
		Advantages of Razor Pages over traditional MVC
		Creating and configuring Razor Pages in a project Every page start with @page is directive Partial views typically live in Pages/Shared/ and begin with _
		Folder structure and naming conventions
	Razor Syntax and Page Model	Razor syntax basics and directives ● Inline C# : @Datetime.UTCNow ● Code Block : @{ var x = 10 } ● Conditional Loops : @if (...) { ..} @foreach(var i in …) ● Directives : 1. @ page 2. @ model 3. @ using 4. @ inject 5. @ section 6. @ functions(legacy) Demo : Simple Razor page and page Model Step 1: Create a razor project Step 2: Create a razor page Step 3: Adding a tag helper

		Step 4: Adding dependency injection Step 5: Running application
		Mixing HTML and server-side code
		Understanding the PageModel class
		Property binding and handling requests
	Advanced .Net Core & Intro to MVC	
	Deep Dive into Razor Pages	Model binding in Razor Pages
		Binding complex types and collections
		Overview of different view types in Razor Pages
	Partial Views and Routing in Razor Pages	Creating and using partial views for reusability
		Implementing and configuring routing in Razor Pages
		Customizing routes and route parameters
	MVC Overview and Model Binding	Understanding the MVC pattern in ASP.NET Core
		Roles of Models, Views, and Controllers
		Setting up an MVC project
		Model binding in MVC: simple and complex types



Port Number	Process Name	Protocol Used	Description
20	FTP-DATA	TCP	File transfer---data
21	FTP	TCP	File transfer---control
22	SSH	TCP	Secure Shell
23	TELNET	TCP	Telnet
25	SMTP	TCP	Simple Mail Transfer Protocol
53	DNS	TCP & UDP	Domain Name System
69	TFTP	UDP	Trivial File Transfer Protocol
80	HTTP	TCP & UDP	Hypertext Transfer Protocol
110	POP3	TCP	Post Office Protocol 3
123	NTP	TCP	Network Time Protocol
143	IMAP	TCP	Internet Message Access Protocol
443	HTTPS	TCP	Secure implementation of HTTP

Problem Statement: Employee Information Portal

Background

A small company wants a simple web application to display employee information. The HR team needs a Razor Page that shows employee details, filters them by department, and allows basic interactions.

Problem Definition

Create a Razor Page (`Employees.cshtml`) that:

1. Displays Employee Data

- Use Razor syntax to render a list of employees (hardcoded or from `PageModel`).
- Each employee should show: Name, Department, Email, and Join Date.
- Format the Join Date properly (e.g., `@employee.JoinDate.ToShortDateString()`).

2. Implements Conditional Rendering

- If an employee is in the IT department, highlight their row in a different color (e.g., light blue).
- If an employee has been with the company for more than 2 years, display a "⭐ Veteran" badge next to their name.

3. Uses Loops & Directives

- Use `@foreach` to render all employees in an HTML table.
- Use `@if` to apply conditional styling (e.g., background color for IT employees).
- Use `@using` to import necessary namespaces (e.g., `System.Collections.Generic`).

4. Adds Basic Interactivity

- Add a dropdown filter to show employees by department (e.g., "All", "IT", "HR", "Finance").
- Use Tag Helpers (`asp-page-handler`) or JavaScript to filter the list dynamically.

5. Includes Dependency Injection

- Inject a service (e.g., IEmployeeService) to fetch employee data (optional for simplicity).
- Alternatively, hardcode a List<Employee> in the PageModel.

6. Uses @section for Scripts

- Add a simple JavaScript alert when the page loads (e.g., "Employee data loaded!").

Sample Data Structure

```
        public class Employee
        {
            public string Name { get; set; }
            public string Department { get; set; }
            public string Email { get; set; }
            public DateTime JoinDate { get; set; }
}
```

Expected Output

- A table listing employees with proper formatting.
- IT employees highlighted.
- Veteran employees marked with a star (⭐).
- A dropdown filter to toggle departments.
- A JavaScript alert on page load.

Success Criteria

➔ Razor Page correctly uses inline C# (e.g., @DateTime.Now).

➔ Conditionals (@if) and loops (@foreach) work as expected.

➔ Directives (@page, @model, @using, @inject) are properly used.

➔ Filtering works (via form submission or JavaScript).

➔ JavaScript alert appears on page load (@section Scripts).

Bonus (Optional)

- Add a search bar to filter employees by name.
- Use AJAX to avoid full page reload when filtering.

- Add a "Add Employee" form (advanced).

Submission Guidelines

- Provide `Employees.cshtml` and `Employees.cshtml.cs`.
- Ensure the code is clean and readable.
- Explain key logic in comments.

Case study :
SPA based on a smart event planner to manage events, attendees, registrations etc.

Based on topics like Components and services, two data bindings.
Modules :

- Events Module
- Attendees Module

Components

- event-list, event-detail
- Attendee-list, attendee-detail

<u>Steps for implementation : -</u>
Step 1: Creating new project along with building blocks
Step 2: Creating feature Modules
Step 3: Creating core components

Step 4: Data Binding & Directives

- ○ Implementing angular Binding
- ○ Using Directives and Pipes

Step 5: Component styling and communication

Apply styling

- Inline styles for quick demo
- External CSS for event cards
- Scoped styles for attendee list

Lifecycle Hooks

- Using ngOnInit to load mock data.
- Using NgOnchanges to detect parent-> child input

Component Communication

- Parent (event-list) passes event object to child(event-details) using @input
- Here Child emits “register” events back to parent using @output

Step 6: HTTP & Observables

- Setup a MOCK API with data for events and Attendees.
- Setup JSON server
- Fetch data using HTTP
  - ○ Creating a service ie events
  - ○ Using HttpClient to fetch events
  - ○ Subscribe with Observables
  - ○ Add error handling(catch Errors)

Step 7: Implement Forms

- Template driven forms (Attendee registration)
  - ○ Create form with [(ngModel)]
  - ○ Add validations (required email)
  - ○ Display error messages
- Reactive form (Event Creation)
  - ○ Using FormBuilder and FormGroup
  - ○ Adding validations (min date, required tile, max attendees)
  - ○ Shows error messages dynamically.

Step 8: Capstone Integration

1. Event List page -> Fetch events from API, apply directives and pipes.
2. Event Details page -> Show event details, allow attended registrations.
3. Attendee List page -> show registered attendees.
4. Event create page -> add a new event via reactive form
5. Error handling -> shows user friendly messages when API fails

Step flow : Setup -> modules-> Components -> Binding -> Directives -> Forms -> HTTP -> Final demo

Following are the steps for implementing JSON server :

1. Install package using “npm install -g json-server”
2. Create a db.json file with events and attendees.
3. Start server : json-server –watch db.json –port 3000