SME Interview

Dineshkumar Thangavel

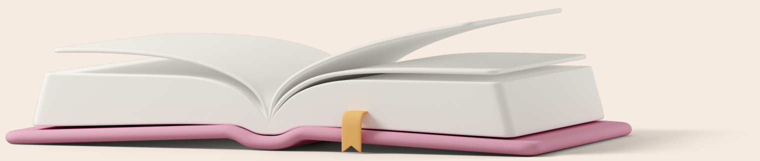
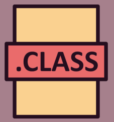


Table of Content

Resume

The Door to Your Interview Success



Resume

🎯 Objective

Create an ATS-friendly technical résumé aligned with GenC skill clusters.

📚 Topics

- Layout: Objective → Projects → Internships → Skills → Education → Achievements.
- Add cluster keywords: *Java* + *SQL* / *Python* + *SQL* / *Full Stack* + *DB* / *C#* . *NET*.
- Use measurable project outcomes.
- Add GitHub / LinkedIn links and tech-toolchain tags.

Why Resume Matters

- 60 % of SME questions come from your resume.
- Interviewer’s first 5 minutes = resume discussion.
- Resume shows clarity + ownership.

Resume Rules

- One Page Only
- Consistent Details (name, marks, dates)
- Projects with Role + Tech + Impact
- Skills you can explain
- Zero grammar errors

Common Red Flags

- 🚫 Vague project lines ('Worked on a college project')
- 🚫 Too many skills / buzzwords
- 🚫 Long paragraphs
- 🚫 Fancy templates / photos
- 🚫 Typos & mixed fonts

Do's and Don'ts

- One page, simple layout.
- Professional email ID.
- Projects = 2 bullets each (Action + Tech + Impact).
- Group skills by type (Languages / Tools / DB).
- Check grammar and spacing.

- Don't add irrelevant details (age, marital status).
- Don't copy friends' projects.
- Don't list every technology you've ever heard of.
- Don't use colored backgrounds or fancy fonts.

Action Verbs

💡 **Formula:**

[Action Verb] + [What You Built] + [Tech Used] + [Your Role] + [Impact]

Action Verbs: Implemented, Designed, Developed, Optimized, Built, Automated, Configured, Created, Debugged, Improved, Reduced, Integrated, Validated.

Weak vs Improved

Weak:

“Worked on attendance system using Python.”

Improved:

“Designed and implemented an automated attendance system using Python (Flask) and SQLite; implemented QR-based check-in that reduced manual entry errors by 95 % in pilot tests.”

Weak vs Improved

Weak:

"Did a web project."

Improved:

"Developed a responsive web application using React and Node.js for event management; implemented REST APIs and MongoDB-based persistence; handled user authentication and role-based access (admin/attendee)."

Education & Skills

Education

B.Sc. Computer Science — XYZ College, City — 2024 — CGPA: 7.6 / 10 (or 72 %)

Skills

Languages: Java, Python, C

Databases: MySQL, SQLite

Web: HTML, CSS, JavaScript, Flask

Tools: Git, VS Code

Recommended

Certifications:

- Python Programming – NPTEL (2024)
- MySQL Essentials – Infosys Springboard

Achievements:

- Class Topper in DBMS (2023)
- Presented project in National Level Symposium

Hobbies / Interests (optional):

- Tech blogging • Graphic design • Reading business books

Tell me about yourself

Your Interview Starts Here

Tell me about yourself

🎯 Objective

Project a confident, technically aligned professional identity.

📚 Topics

- Personal + Academic + Project overview.
- Skill layering: “I’m skilled in … currently improving …”.
- Core interest + career goal + company fit.
- Hobbies as light closure.

Why This Question Matters

- 🎯 Sets first impression
- 🎯 Checks your communication & fluency
- 🎯 Judges clarity of thought
- 🎯 Gives interviewers direction for follow-up

Common Mistakes

- 🚫 Too long or too short or memorized
- 🚫 Irrelevant personal details
- 🚫 No mention of project or skills
- 🚫 Poor structure or flow

Preferred Pattern

Section	Purpose
🧍 Personal	Warm intro: name, degree, background
💼 Work Experience	Highlight real or mini work exposure
💻 Project	Show practical application of learning
🧑‍💻 Internship	Prove real-world adaptability
⚙️ Skills	Layered skill story (Core → Tool → Add-ons)
💡 Company Fit	Show awareness & interest in Cognizant
❤️ Closing Statement	Personal touch: goals, hobbies, learning focus

Personal

🎙️ Start with a simple, confident opening.

Structure:

Name

Degree + Specialization

College + Year

City or background (optional, short)

Example:

“Good morning, I’m Priya Sharma, pursuing B.Sc. Computer Science (2024) from ABC College, Chennai. I have a strong interest in solving logical problems and building simple tech solutions that make daily life easier.”

Work Experience

“Highlight relevant or part-time exposure — even freelance, training assistant, or mini projects count.”

Example:

“I’ve worked part-time as a lab assistant, helping juniors debug Python programs and handle college-level automation scripts. That experience improved my communication and team coordination skills.”

Project Discussion

Structure:

1️⃣ Project Name & Objective - 2️⃣ Role / Contribution - 3️⃣ Tech Stack - 4️⃣ Impact / Outcome -

5️⃣ Challenge Faced & Fix

Example (eCommerce context):

"My major project was an E-Commerce Product Management System built using Python (Flask) and SQLite.

I handled the backend logic — managing product listings, user authentication, and database queries.

We reduced data redundancy by 40% through normalization.

One major challenge was duplicate product entries, which I fixed using unique constraints and validation."

Intership

"Link classroom learning with workplace exposure."

Example:

"During my internship at TechNova Solutions, I worked on data cleaning using Python and automated Excel-based reports.

It helped me understand how corporate teams collaborate and meet tight deadlines."

Skills

"Skills – Your Strengths + Your Growth Story"

Layer 1 – Skilled In	**Layer 2 – Improving / Learning Now**
Technologies, languages, or tools you are already confident with.	Topics or frameworks you're currently enhancing.
Example → "I'm skilled in Python, SQL, and HTML/CSS."	Example → "I'm currently improving my OOP and database skills."
✅ Shows current capability	✅ Shows curiosity & learning mindset

Skills

Python Stack Student

“I’m skilled in Python, SQL, and HTML/CSS, and I’m currently strengthening my OOP and database design skills.”

Java Stack Student

“I’m skilled in Core Java and MySQL, and I’m currently enhancing my Spring Boot and DSA skills.”

MERN Stack Student

“I’m skilled in HTML, CSS, and JavaScript, and I’m currently building projects to improve my React and Node.js skills.”

Full Stack or Hybrid

“My core skills include Java, Python, and SQL, and I’m continuously improving my OOP and data-handling capabilities.”

Company Fit

“Show why you belong at Cognizant — be specific.”

Example:

“I’m particularly drawn to Cognizant for its learning-focused culture and opportunities for technology upskilling.

I like that freshers here work on real client projects early, which perfectly matches my learning goals.”

Closing Statement

“End on a confident, personal tone.”

Example:

“My career goal is to become a full-stack developer who designs scalable and user-friendly applications.

In my free time, I enjoy UI sketching and reading about new tech trends — they keep me creative and curious.”

Sample Answer

"I'm Dinesh Kumar, a B.Tech. Computer Science graduate from XYZ College (2024).

In my final year, I developed a Student Attendance Automation System using Python (Flask) and SQLite, where my role was backend development and QR-based data validation.

The system reduced manual entry errors by 95% during testing.

I'm skilled in Python, SQL, and HTML/CSS, and I'm currently improving my OOP and database skills.

I'm looking forward to starting my career at Cognizant, where I can contribute to quality-driven software projects while continuously learning new technologies.

Outside academics, I enjoy UI design and tech blogging – I love learning how design and code work together."

Feedback Tips

- ✅ Speak with energy, not speed
- ✅ Maintain eye contact / camera focus
- ✅ Smile naturally
- ✅ Use gestures subtly
- ✅ End with confidence ("Thank you")

Project Discussion

Show What You Built, Not What You Memorized

- 🎯 Validates your technical skills
- 🎯 Tests clarity & ownership
- 🎯 Reveals your problem-solving mindset
- 🎯 Checks teamwork & communication

The 5-Block Structure

💡 Project → Role → Tech → Impact → Challenge Fixed

1️⃣ Name & purpose

2️⃣ Your role

3️⃣ Technology stack

4️⃣ Impact / result

5️⃣ Challenge you fixed (very important)."

Example

"I developed a Student Attendance Automation System using Python (Flask) and SQLite.

I handled backend APIs for QR-based attendance capture.

The system reduced manual entry errors by 95 %.

A challenge I faced was data duplication — I fixed it using unique constraints and input validation."

Follow-up Questions

- **Code & Logic:** How does your main module work? Can you write that function?
- **Database:** Show your table structure / how you wrote the query.
- **Tech Reasoning:** Why did you choose Python over Java?
- **Debugging:** What error did you face and how did you solve it?
- **Improvement:** If given more time, what would you add?

How to Prepare?

✅ Know your architecture (front → back)

✅ Understand the flow of data

✅ Memorize one core function logic

✅ Have one SQL query ready

✅ Prepare one bug you fixed

Mistakes to Avoid

- 🚫 Copying projects you can’t explain
- 🚫 Saying ‘my team did that’ (no ownership)
- 🚫 Forgetting to mention your tech stack
- 🚫 No quantified impact
- 🚫 No ‘challenge fixed’ story

OOPs

Technical Concept Check — Think, Don’t Memorize

OOPs

🎯 Objective

Master object-oriented design and clean-code thinking.

📚 Topics

- Class | Object | Constructor | Inheritance | Encapsulation | Polymorphism | Abstraction.
- Interface vs Abstract Class (e-commerce example).
- SOLID Principles: SRP | OCP | LSP | ISP | DIP.
- Exception handling best practices.
- Design Patterns (NEW):
- Singleton – one instance (DB connection).
- Factory – object creation logic.
- Observer – event notification.
- MVC – separation of layers.
- DAO – data-access abstraction.
- C# vs Java syntax parallels.

Why OOPs Matters

- 🎯 Helps model real-world systems (like eCommerce)
- 🎯 Promotes reusability & maintainability
- 🎯 Provides scalability for large projects
- 🎯 Reduces redundancy

Example: (Customer → Cart → Order → Product)

The Core OOPs Terms

Concept	Definition	Keyword
Class	Blueprint or template that defines data and behavior.	“Design”
Object	Instance of a class that holds real data.	“Reality”
Constructor	Special method that initializes object state automatically.	“Initialization”
super()	Keyword to access parent class constructor or methods.	“Parent link”

The Core OOPs Terms

```
class Product:
    def __init__(self, product_id, name, price):
        self.product_id = product_id
        self.name = name
        self.price = price

# Object creation
p1 = Product(101, "Wireless Mouse", 799)
print(p1.name, "₹", p1.price)
```

The 4 Pillars Overview

Concept	Meaning	Keyword
Encapsulation	Bundling data + methods	Security
Inheritance	Reuse parent class	Reusability
Polymorphism	One action, many forms	Flexibility
Abstraction	Hide details, show essentials	Simplicity

.CLASS

Encapsulation Example

“Encapsulation means binding data (variables) and methods (functions) together and restricting direct access to them.”

Encapsulation Example

```
class Customer:
    def __init__(self, name, email):
        self.__name = name
        self.__email = email

    def get_email(self):
        return self.__email
```

.CLASS

Inheritance Example

“Inheritance allows one class to reuse properties and methods of another.

Child class object inherits all the properties of parent class object”

Inheritance Example

```
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class ElectronicProduct(Product):
    def __init__(self, name, price, warranty):
        super().__init__(name, price)
        self.warranty = warranty
```

Polymorphism Example

“Polymorphism means same action, different behavior — one interface, many implementations.”

1. Method Ovecrloading - same method, different behavior
2. Method Overriding - child class overrides the behavior of parent class method

Polymorphism Example

```
class Payment:
    def pay(self):
        pass

class CardPayment(Payment):
    def pay(self):
        return "Paid via Card"

class UpiPayment(Payment):
    def pay(self):
        return "Paid via UPI"
```

Abstraction Example

“Abstraction hides complexity and shows only the essential features to the user.”

Abstraction Example

```
from abc import ABC, abstractmethod

class Payment(ABC):
    @abstractmethod
    def pay(self):
        pass

class WalletPayment(Payment):
    def pay(self):
        print("Payment done using wallet balance")
```

Interface vs Abstract Class

Feature	Abstract Class	Interface
Purpose	To provide *partial abstraction* (some logic + some rules)	To provide *full abstraction* (only rules, no logic)
Keyword	`abstract`	`interface`
Methods	Can have abstract & non-abstract methods	All methods are abstract by default (in Python/Java)
Variables	Can have variables (instance/static)	Only constants (public static final in Java)
Constructor	Can have constructor	Cannot have constructor
Inheritance	Class → extends abstract class	Class → implements interface
Multiple Inheritance	Single inheritance only	Multiple interfaces can be implemented
Usage	When subclasses share common base logic	When classes need to follow a contract
Example Use Case	`abstract class Payment {}` defines shared validation logic	`interface Payable {}` defines `makePayment()` signature

Common Mistakes Students Make

- 🚫 Rote definitions
- 🚫 No code example
- 🚫 No project connection
- 🚫 Confusing terms (abstraction vs encapsulation)

Simple Coding Logics

Problem understanding → approach explanation → correct output reasoning

Approach Explanation

Step	Layer	Purpose
1 **Algorithm (Concept)**	Write down what the problem needs and how you plan to solve it logically.	Defines your thought flow before coding.
2 **Algorithm with Clear Steps**	Break your logic into numbered, simple, executable steps.	Ensures clarity before implementation.
3 **Pseudo Code**	Represent logic using structured English + simple code notation.	Acts as a language-independent plan.
4 **Code (Java / Python)**	Convert pseudo code into syntactically correct code.	Implementation of logic.
5 **Dry Run (Logic Validation)**	Manually trace with sample input and verify output.	Checks correctness and helps in debugging.

Example

Find the Second Largest Element

Algorithm:

1. Read the array.
2. Initialize first and second as minimum values.
3. Traverse array:
    - If arr[i] > first, update second = first, first = arr[i].
    - Else if arr[i] > second and arr[i] != first, update second = arr[i].
4. Print second.

Example

Find the Second Largest Element

Pseudo Code:

```
START
 INPUT array A of size N
 first ← second ← -∞
 FOR each element x in A:
   IF x > first:
     second ← first
     first ← x
   ELSE IF x > second AND x != first:
     second ← x
 PRINT second
END
```

Example

Find the Second Largest Element

Python Code:

```
arr = [10, 45, 12, 50, 22]
first = second = float('-inf')
for x in arr:
    if x > first:
        second = first
        first = x
    elif x > second and x != first:
        second = x
print("Second largest:", second)
```

Example

Find the Second Largest Element

Java Code:

```
int[] arr = {10, 45, 12, 50, 22};
int first = Integer.MIN_VALUE, second = Integer.MIN_VALUE;
for (int x : arr) {
    if (x > first) {
        second = first;
        first = x;
    } else if (x > second && x != first) {
        second = x;
    }
}
System.out.println("Second largest: " + second);
```

Example

Find the Second Largest Element

Dry Run:

i	arr[i]	first	second
0	10	10	-∞
1	45	45	10
2	12	45	12
3	50	50	45
4	22	50	45

Basic DSA

Think in Structures, Not Just Steps

DSA

🎯 Objective

Strengthen logic building, data structures, and I/O handling.

📚 Topics

- Arrays | Strings | Recursion | Sorting | Searching.
- Linked List | Stack | Queue | Tree | Graph.
- HashMap | Set | Prefix Sum | Sliding Window.
- **Java I/O Streams:** FileInputStream, BufferedReader, FileWriter.
- Time & Space Complexity analysis.

What Is a Data Structure?

A Data Structure is a way of organizing and storing data so it can be used efficiently.

Examples:

Array, Linked List, Stack, Queue, Tree, Graph, HashMap

Why DSA in Interviews?

Interviewer Perspective

- Checks your logical organization skills
- Predicts coding maturity
- Evaluates problem-solving clarity

✨ **Visual:** Code + Flowchart diagram

Array

Step	Explanation
Definition	A linear data structure that stores elements in **contiguous memory locations**, accessed using an **index**.
Analogy	🗃️ Like a **row of lockers** — each locker has a unique number (index) to access items instantly.
Syntax	**Java:** `java int[] arr = {10, 20, 30, 40}; System.out.println(arr[2]);` **Python:** `arr = [10, 20, 30, 40]; print(arr[2])`
Applications	Searching, Sorting, Matrix Representation, Static Data Storage.
Traversals	**Java:** `for(int i=0;i<arr.length;i++){System.out.print(arr[i]+" ");}` **Python:** `for i in range(len(arr)): print(arr[i], end=" ")`
Operations + Time Complexity	Access O(1) • Search O(n) • Insert/Delete O(n) • Traverse O(n)

Linked List

Step	Explanation
Definition	A dynamic linear data structure made of **nodes**, where each node contains data and a pointer to the next node.
Analogy	🚆 Like a **train** — each coach (node) is connected to the next coach.
Syntax	**Java:** `java class Node { int data; Node next; Node(int d){ data=d; next=null; } }` **Python:** `class Node: def __init__(self, data): self.data=data; self.next=None`
Applications	Dynamic Memory Allocation, Stack & Queue Implementation, Undo Operations.
Traversals	**Java:** `Node temp = head; while(temp != null){ System.out.print(temp.data+" "); temp = temp.next;}` **Python:** `temp = head; while temp: print(temp.data, end=" "); temp = temp.next`
Operations + Time Complexity	Access O(n) • Insert O(1)* • Delete O(1)* • Search O(n) • Traverse O(n) *if pointer/reference known

Stack

Step	Explanation
Definition	A linear data structure that follows the **LIFO (Last In First Out)** principle.
Analogy	🍽️ Like a **stack of plates** — last plate kept is the first one removed.
Syntax	**Java:** `java import java.util.*; Stack<Integer> s = new Stack<>(); s.push(10); s.push(20); System.out.println(s.pop());` **Python:** `stack = []; stack.append(10); stack.append(20); print(stack.pop())`
Applications	Function Calls (Recursion), Undo/Redo, Expression Evaluation, Syntax Checking.
Traversals	**Java:** `for(Integer x : s){ System.out.print(x+" "); }` **Python:** `for x in stack[::-1]: print(x, end=" ")`
Operations + Time Complexity	Push O(1) • Pop O(1) • Peek O(1) • Traverse O(n)

Queue

Step	Explanation
Definition	A linear structure that follows **FIFO (First In First Out)** order.
Analogy	🎟️ Like a **ticket queue** — first person in line gets served first.
Syntax	**Java:** `java import java.util.*; Queue<Integer> q = new LinkedList<>(); q.add(10); q.add(20); System.out.println(q.remove());` **Python:** `from collections import deque; q = deque(); q.append(10); q.append(20); print(q.popleft())`
Applications	Task Scheduling, CPU Job Queue, Messaging Systems.
Traversals	**Java:** `for(Integer x : q){ System.out.print(x+" "); }` **Python:** `for x in q: print(x, end=" ")`
Operations + Time Complexity	Enqueue O(1) • Dequeue O(1) • Peek O(1) • Traverse O(n)

Overview

Structure	Order	Memory Type	Key Principle	Example Use	Insert	Delete	Access	Search
Array	Indexed	Contiguous	Random Access	Static data	O(n)	O(n)	O(1)	O(n)
Linked List	Sequential	Dynamic	Pointer-based	Dynamic data	O(1)*	O(1)*	O(n)	O(n)
Stack	Linear	Dynamic	LIFO	Undo, Function calls	O(1)	O(1)	O(1)	O(n)
Queue	Linear	Dynamic	FIFO	Scheduling	O(1)	O(1)	O(1)	O(n)

Tree

Step	Explanation
Definition	A **hierarchical** data structure consisting of nodes connected by edges. The top node is the **root**, and every node can have child nodes.
Analogy	🌴 Like a **family tree** — parent at top, children branching below.
Syntax	**Java:** `java class Node { int data; Node left, right; Node(int d){ data=d; left=right=null; } }` **Python:** `class Node: def __init__(self, data): self.data=data; self.left=None; self.right=None`
Applications	Hierarchical data (XML/HTML), File Systems, Databases (B-trees), Expression Trees, AI Decision Trees.
Traversals	**DFS Types:** InOrder (L–Root–R), PreOrder(Root-L-R), PostOrder(L-R-Root) BFS Type: Level Order(L1, L2…….Ln)
Operations + Time Complexity	Search O(h)

Graph

Step	Explanation
Definition	A collection of **vertices (nodes)** connected by **edges (links)**; can be **directed/undirected**, **weighted/unweighted**.
Analogy	🗺️ Like a **city map** — intersections are nodes, roads are edges.
Syntax	**Java (Adjacency List):** `Map<Integer,List<Integer>> graph=new HashMap<>(); graph.put(0,List.of(1,2));` **Python:** `graph={0:[1,2],1:[2],2:[0,3],3:[3]}`
Applications	Social Networks, Maps & Routing, Compilers (Dependency Graphs), Networking (Shortest Path).
Traversals	**DFS (Depth-First Search):** via stack/recursion **BFS (Breadth-First Search):** via queue
Operations + Time Complexity	Add Vertex O(1)

Overview

Structure	Order	Connection	Traversal	Use Case	Avg Search	Avg Insert
Tree	Hierarchical	Parent-child	DFS / BFS	File Systems	O(log n)	O(log n)
BST	Hierarchical	Ordered L<R<Rt	Inorder (sorted)	Searching / Sorting	O(log n)	O(log n)
Graph	Network	Arbitrary links	DFS / BFS	Maps / Social Networks	O(V + E)	O(1)

SQL

Speak to the Database

SQL

🎯 Objective

Develop query writing + database design integration.

📚 Topics

- ERD & Normalization (1NF → 3NF).
- Constraints & Integrity (Domain, Entity, Referential).
- CRUD | Joins | Subqueries | Window functions.
- Transactions & ACID.
- Integration: React form → Spring Boot API → MySQL.

ER Diagram & Cardinality

Entity Relationship Diagram (ERD) shows how tables are connected.

Components:

- **Entity:** Real-world object (Customer, Product, Order)
- **Attribute:** Property (Name, Amount, City)
- **Relationship:** How entities connect (One-to-One, One-to-Many, Many-to-Many)

Cardinality Examples:

- Customer → Order = 1:N
- Product → Order = 1:N
- Student ↔ Course = M:N

✨ Visual Suggestion: Simple eCommerce ERD diagram (Customer–Order–Product).

Normalization

Purpose: Reduce redundancy, improve data consistency.
Forms:

1️⃣ **1NF:** Atomic values only.

2️⃣ **2NF:** Remove partial dependencies.

3️⃣ **3NF:** Remove transitive dependencies.

Example:

```
| Order_ID | Customer_Name | Customer_City | Product | → Split into
→ Customer & Order tables |
```

Database Integrity Rules

Type	Meaning	Example
Domain Integrity	Valid data type & format	Age > 0
Entity Integrity	Each row uniquely identified	Primary Key
Referential Integrity	Valid link between tables	Foreign Key (Order → Customer)

Reading Queries

```
SELECT column_name(s)
FROM table_name
WHERE condition
JOIN other_table ON condition
GROUP BY column_name
HAVING aggregate_condition
ORDER BY column_name ASC|DESC
LIMIT n OFFSET m;
```

Join

Join Type	Definition	Output
INNER JOIN	Common rows in both	Returns only rows that have matching values in both tables.
LEFT JOIN	All from left + matching right	All customers, even those without orders
RIGHT JOIN	All from right + matching left	Returns all records from the left table (Customer), and the matching rows from the right table (Orders). If no match → NULL on right side.
FULL JOIN	All from both	Returns all records when there's a match in either left or right table. Non-matching rows show NULLs on respective sides.
Cross Join	Combination of both table	Returns the Cartesian product of both tables — each row from left table combines with all rows from right table.

Sub-Query

A subquery (or inner query) is a query nested inside another query.

It allows you to perform complex operations by combining multiple steps into one SQL statement.

Think of it like this:

"The inner query gives me a list;
The outer query uses that list to decide what to do."

Example:

```
SELECT column_list

FROM table_name

WHERE column operator (SELECT column FROM table WHERE condition);
```

CS Basics

Reinforce Domain

CS

🎯 **Objective**

Reinforce foundational CS theory to tackle SME concept questions.

📚 **Topics**

- Operating Systems: process vs thread, scheduling, deadlock, paging.
- Computer Networks: OSI/TCP-IP, IP addressing, HTTP methods, DNS, REST API.
- DBMS Theory: keys, normalization, indexing, DBMS vs file system.
- Software Engineering: SDLC, Agile, Scrum roles, Testing types, Version Control.

Cluster
Specialization Tracks

Resume

Cluster

Cluster	Stack	Deliverables
Java Full Stack	Java Core, Spring Boot, MySQL, React, HTML/CSS/JS	CRUD + API + React UI
Python Full Stack / Data	Python, Flask, SQL, Pandas / NumPy	Analytics Dashboard
C# /.NET	C#, ASP.NET, SQL Server, Bootstrap	REST API + DB Integration
Cloud / DevOps	AWS/GCP, Docker, CI/CD	Cloud deployment + GitFlow

THANK YOU

Any questions ?