

SYLLABUS

Introduction to the Specialization

This specialization is designed for students who want to build a strong foundation in **backend development using C#**.

Students begin with core IT fundamentals, develop logical and analytical skills, learn programming from zero, and then progress into professional backend engineering topics used in real-world software development.

The specialization combines:

- **Theory + Hands-on practice**
- **Guided labs + “You Try It” coding tasks**
- **Monthly assessments**
- **One final backend project**

By the end, students will confidently understand how backend systems work and how to create projects fundamentals.

Graded Practice Quiz

Homework Tasks:

- 30 knowledge-based questions
- Assigned as at-home work
- Each student’s score is recorded and included in their monthly performance review

Guided Lab

Practice Session:

- Homework questions and tasks are reviewed in detail
- Instructor provides individual explanations to clarify difficult or unclear topics
- Students continue practicing until they fully understand the concepts

MODULE 1 — IT FOUNDATIONS (Computer Science Basics)

In this module students will understand the fundamental knowledge of operating systems, software tools, internet technologies, and networking.

Skills Gained: IT fundamentals, system usage, browser security, cloud basics, information security.

TOPICS:

1. Operating Systems:

- What is an OS?
- Microsoft Windows — basics and settings
- Apple macOS — basics and settings
- GNU/Linux — basics, terminal intro, common distributions (Ubuntu), when to use Linux
- Software Applications (browsers, settings, security)
- Networking & Cloud Basics (DNS, basic protocols, intro to cloud)

2. Software Applications:

- Understanding software usage requirements
- Basic browser security & privacy settings
- Safe browsing practices for developers

2. Computer Networks:

- Key networking protocols (HTTP, HTTPS, TCP/IP, DNS)
- Client–server architecture & how backend apps communicate

MODULE 2 — Foundation of Backend Development

In this module, students will learn the fundamental principles of backend development and understand the responsibilities of backend engineers. By the end of the module, students will be able to create a simple project plan, set up a version control repository on GitHub, and apply essential backend concepts in practical scenarios.

Skills Gained: Git & GitHub for version control, Debugging and logic error analysis using Microsoft Copilot, Core backend development principles, Practical debugging techniques, Understanding how the internet and web requests work.

TOPICS:

1. Introduction to Backend Development:

- What is Backend Development?
- Core Principles of Backend Development
- Roles & Responsibilities of a Backend Engineer
- Essential Skills, Technologies & Tools
- Backend Development Workflow
- Project Planning Fundamentals
- Resource Management & Documentation
- Tool Integration for Effective Backend Planning

2. Project Development Essentials

- Integrated Coding Practice
- Introduction to Debugging
- Debugging Techniques
- Using Microsoft Copilot for Debugging & Logic Error Analysis
- Git Basics for Backend Development
- How the Internet Works
- DNS Basics
- Hosting & Deployment Basics
- How a Browser Works
- HTTP vs HTTPS (Security & Data Transfer Basics)

MODULE 3 — Algorithms

In this module, students will build strong problem-solving skills through data structures, algorithms, and performance optimization. By the end, students will be able to design efficient solutions, write algorithmic logic, and analyze performance using Big-O notation.

Skills Gained: Fundamental data structures, Sorting and searching techniques, Algorithmic strategies, Hands-on implementation, Performance analysis

TOPICS:

1. Logical Thinking & Problem-Solving Foundations:

- Types of logic used in programming
- Deductive reasoning in problem-solving
- Steps in deductive reasoning
- Problem decomposition techniques
- Top-down vs bottom-up approaches
- Introduction to pseudocode

2. Flowcharts & Basic Data Types:

- Introduction to algorithms
- Flowcharts and algorithm structures
- Practical flowchart analysis
- Data types & type conversion
- Introduction to variables
- Declaration, initialization & assignment
- Basics of algorithm design

3. Control Structures, Loops & Functions:

- If-else statements (basic to advanced)
- Switch statements & complex cases
- Decision-making in programs
- For, while, do-while loops
- Loop optimization techniques
- Integrating loops with conditional logic
- Introduction to functions & methods
- Method syntax, parameters & return values
- Combining functions/methods for problem-solving
- Developing structured programs using functions

4. Control Structures, Loops & Functions:

- Characteristics of arrays & linked lists
- Use cases for stacks & queues
- Choosing the right data structure

- Implementation of arrays & linked lists
- Introduction to Big-O notation
- Applying Big-O to data structures

5. Sorting & Searching Algorithms:

- Introduction to sorting & searching
- Bubble sort implementation
- Quicksort & merge sort (mechanics + implementation)
- Testing & analyzing sorting algorithms
- Linear search & its applications
- Binary search in sorted structures
- Binary search implementation
- Using binary search in backend systems
- Optimization using search algorithms
- Selecting the appropriate sorting method for backend tasks

6. Trees & Graphs:

- Introduction to trees & graphs
- Practical use cases in real systems
- Tree traversal techniques
- Directed vs undirected graphs
- Graph traversal algorithms
- Introduction to load balancing algorithms & strategies

6. Advanced Problem-Solving:

- Introduction to advanced algorithms
- Overview of dynamic programming
- Greedy algorithms
- Hands-on dynamic programming exercises
- Implementing DP in scheduling applications
- Introduction to hashing
- Demonstrations of hashing techniques

MODULE 4 — Introduction to Programming in C#

In this module, students will explore both the foundational and advanced concepts of C# programming. They will learn how the .NET runtime works, develop strong understanding of memory management, error handling, asynchronous programming, LINQ, OOP principles, and modern C# features (C# 7–13).

Skills Gained: CLR, Garbage Collector & memory types, Object-Oriented Programming, Collections & Generics, LINQ & Lambda Expressions, Delegates & Events, Exception handling, Multithreading, Tasks, Async/Await, Reflection & Streams, Using new C# features (7–13)

TOPICS:

1. Fundamentals of C# and .NET:

- C# vs C programming
- Basic C# syntax: statements, comments, namespaces
- Top-level statements & global using directives
- Directives vs statements
- Object class overview
- CLR, CTS, CLS
- Managed vs unmanaged code
- Assemblies (EXE, DLL), GAC, NuGet packages
- Application Domain vs Process vs Thread
- Inter-process communication (pipes, shared memory, sockets)
- Intra-process communication (locks, mutexes, events, conditions)

Garbage Collector (GC):

- Dispose, Finalize/Destructor
- GC generations
- Using statement
- Memory cleanup strategies

2. Data Types & Operators:

- var vs object vs dynamic

- Implicit vs primitive types
- Float vs double vs decimal
- Assignment, arithmetic, increment/decrement
- Relational, logical, bitwise operators
- `Equals()` vs `==`
- Unary operators

3. Data Types & Operators:

- Value vs reference types
- Stack vs heap
- `string` vs `StringBuilder`
- String formatting & interpolation
- Substring & string manipulation
- Boxing & unboxing
- Implicit vs explicit casting
- `Convert` vs `Parse` vs `TryParse`
- Type checking: `is` and `as`
- Nullable value types
- Null-forgiving operator
- Null-coalescing (`??`)
- `ref`, `out`, `ref local`, `ref return`, `in` parameters

4. Control Statements:

- Selection statements
- Iteration statements
- Ternary operator
- `const` vs `readonly`

5. Methods:

- What is a method?
- Extension methods
- Expression-bodied methods
- Tuples
- Nested methods

- Method overloading & overriding

6. Object-Oriented Programming (OOP):

- What is OOP?
- Classes, records, structs, enums
- Fields & properties (get/set/init)
- Constructors (private/static/chaining)
- Private vs static constructor
- Deconstructors
- base and this keywords
- Static classes
- Anonymous classes
- Sealed classes
- Access modifiers
- Polymorphism & encapsulation
- Virtual methods & method hiding
- Abstract classes & methods
- Inheritance vs interface (including changes in C# 8 & 11)
- Abstract vs interface comparison

7. Arrays:

- Single-dimensional & multidimensional arrays
- Jagged arrays
- Object arrays
- Dynamic arrays
- Indexers
- Partial classes

8. Generics & Collections:

Generics

- Generic classes, methods & interfaces
- Advantages & disadvantages

Collections

- Generic vs non-generic collections
- Non-generic: ArrayList, Hashtable, SortedList, Stack, Queue
- Generic:
 - List<T>
 - Dictionary<TKey, TValue>
 - SortedDictionary<TKey, TValue>
 - Stack<T>, Queue<T>
 - HashSet<T>
 - LinkedList<T>
- DictionaryEntry vs KeyValuePair
- List<T> vs LinkedList<T>
- Collection initializers
- IEnumerable vs ICollection vs IQueryable

9. LINQ & Lambda Expressions:

LINQ Basics

- What is LINQ?
- Query syntax vs method syntax
- LINQ operators:
 - select, selectMany, orderBy
 - First, Single, Last
 - All, Any
 - ThenBy
 - Skip/SkipWhile
 - Take/TakeWhile
- TakeWhile vs Where
- Where vs Let
- AsEnumerable vs AsQueryable

LINQ and Databases

- DataContext role
- Why SELECT comes after FROM in LINQ
- LINQ vs stored procedures
- LINQ providers
- Types of LINQ (Objects, XML, SQL, DataSet, Entities)

- PLINQ (parallel LINQ)
- Advantages & disadvantages

Lambda Expressions

- What is a lambda expression?
- Where and why we use lambda
- Important rules of lambda
- Expression vs statement lambda
- Type inference
- Does lambda have a type?

10. Errors & Exceptions:

- What is an error?
- Types: compile-time, runtime, logical
- What is an exception?
- Exception classes (DivideByZero, SQL, Format, etc.)
- throw vs throw ex
- Exception handling: try/catch/finally
- System.Exception vs ApplicationException
- Inner exceptions
- AggregateException

11. Asynchronous Programming:

Threads

- What is a thread?
- Thread types
- Thread vs process
- Thread pool
- Multithreading
- Thread methods: Sleep, Join, Abort, IsAlive
- Locks, Monitor, Mutex
- Deadlock & race conditions

Tasks

- Task.Run vs Task.FromResult
- Task.WhenAll
- Task.Delay
- Task.WaitAll
- Task.ContinueWith

Async/Await

- Async vs sync
- IO-bound vs CPU-bound
- Context switching
- Interlocked

Parallel Programming

- Parallel.For
- Parallel.ForEach
- Parallel.Invoke
- Async vs parallel
- Multithreading vs parallel

12. Delegates & Events:

Delegates

- What is a delegate & why it's used
- Multicast delegates
- Delegate invocation
- Callback functions
- Delegate vs interface
- Handling exceptions in multicast delegates
- BeginInvoke vs EndInvoke
- Built-in delegates
- Generic vs custom delegates
- Method group conversion
- Anonymous methods vs lambdas

Events

- What is an event?
- Publisher/subscriber
- Event handling
- EventHandler
- Async event handling

13. Reflection:

- What is reflection
- `typeof()` vs `GetType()`
- When reflection is useful
- Risks of reflection
- Binding types & binding flags
- `MethodInfo` & `PropertyInfo`
- Late binding vs early binding

14. Streams & File Handling

Stream Class

- Stream properties: `CanRead`, `CanWrite`, `CanSeek`, `Length`, `Position`
- Stream methods: `Read`, `Write`, `Seek`, `Close`, `Flush`

Derived Classes

- `FileStream`
- `StreamReader/StreamWriter`
- `BinaryWriter`
- `BufferedStream`

Async Operations

- `ReadAsync` & `WriteAsync`
- Buffering & chunk reading
- Handling exceptions in streams
- Managing resources

- Access control & security

File & Directory

- File vs FileInfo
- Directory vs DirectoryInfo