Unit Testing, Mocking, and Dependency Injection with Moq

# Objectives

## Understand how Mocking can enhance Test-Driven Development (TDD)

* Mocking: Creating simulated objects that mimic the behavior of real objects.
* Isolation: Testing components in isolation from external dependencies.
* Test Doubles: General term for mocks, stubs, fakes, etc.
* Mock vs Fake vs Stub:
* - Mock: Verifies interactions.
* - Fake: Working implementation, not production-ready.
* - Stub: Returns hardcoded responses.
* Key Advantages of TDD: Improved design, maintainable code, faster debugging, and reduced regression errors.

## Explain the meaning of Mocking in Unit Testing and why use mocks in Unit Testing

* Mocking helps simulate complex or slow systems (e.g., databases, APIs).
* Improves test execution speed and reliability.
* Allows developers to focus on logic rather than integration during unit tests.
* Isolation of Dependencies using Mocks and Stubs.

## Understand the basics of Dependency Injection (DI) and how DI helps unit testing

* Dependency Injection: A technique to pass dependencies from outside.
* Constructor Injection: Pass dependencies via the class constructor.
* Method Injection: Pass dependencies via method parameters.
* Helps in mocking and improving testability by removing hardcoded dependencies.

## Demonstrate how to create a testable code with Moq

* Write interfaces for external systems.
* Use constructor injection to inject dependencies.
* Mock the interface using Moq in the test project.
* Verify the interactions and test behavior using mock return values.

## Demonstrate how to create a mock object that accesses a database for unit tests

* Create an interface for the database access layer (e.g., IRepository).
* Mock the interface using Moq to simulate DB behavior.
* Use `Setup()` to simulate return values from DB methods (e.g., GetById, Add).
* Test the logic layer by injecting the mock repository.

## Demonstrate how to mock objects that access the file system for unit tests

* Abstract file operations into an interface (e.g., IFileService).
* Create a mock of the interface using Moq.
* Set up expected behavior like file exists, read, or write.
* Inject and test the class without accessing the real file system.