1. Why Testing Is Critical in Blazor Applications

Blazor applications are **component-based UI systems**. Unlike traditional MVC, **logic, UI, and state live together** inside components. This makes testing essential for:

- Verifying **UI rendering**
- Validating **user interactions**
- Ensuring **state changes propagate correctly**
- Protecting against **regressions** as components evolve

Testing in Blazor typically covers **three layers**:

1. **Unit Testing (logic & services)**
2. **Component Testing (Razor components)**
3. **End-to-End (E2E) Testing (full browser automation)**

2. Types of Testing in Blazor

Test Type	**What It Tests**	**Tools**
Unit Tests	Business logic, services	xUnit / NUnit
Component Tests	Razor components rendering & behavior	**bUnit**
Integration Tests	Component + services	bUnit + DI
E2E Tests	Full user flow in browser	Playwright

3. Unit Testing in Blazor (Logic & Services)

Rule

Never put complex logic inside Razor components.
Move it to services → easier to test.

4. Component Testing in Blazor using bUnit

What is bUnit?

bUnit is a Blazor component testing framework that allows you to:

- **Render Blazor components without running the full app**

- **Interact with the rendered markup**
- **Assert UI behavior, state changes, and events**
- **Mock services, authentication, JS interop, and APIs**

Key Characteristics

- **Runs fast (in-memory rendering)**
- **Works with xUnit, NUnit, or MSTest**
- **Ideal for component-level testing, not E2E**

Why bUnit?

Blazor components:

- Render HTML
- Respond to events
- Depend on DI

bUnit provides a **test renderer** for Razor components.

When to Use bUnit

Scenario	**Use bUnit**
Component rendering	✅
UI behavior testing	✅
Event handling	✅
Form validation	✅
API mocking	✅
Full browser automation	❌ (Use Playwright)

4. **Mocking Dependencies in Blazor Tests**

Why Mock?

- Avoid API calls
- Control test data
- Improve reliability

5. **End-to-End (E2E) Testing with Playwright**

Use E2E tests when:

- You need **real browser behavior**
- Navigation & authentication must be verified

Testing Best Practices in Blazor

Architecture Guidelines

- Keep **logic out of Razor**
- Prefer **services over code-behind**
- Use **DI everywhere**

6. **1. What is Blazm Extension?**

Blazm Extension is a **utility layer on top of bUnit** that focuses on:

- Cleaner assertions
- Reduced boilerplate
- More readable test intent
- UI-centric assertions (what the user sees)

Think of it as:

FluentAssertions for Blazor UI tests

Without Blazm (Plain bUnit)

Assert.Contains("Save", cut.Markup);

With Blazm

cut.Should().ContainText("Save");

Much easier to **read, teach, and maintain**.