csharp

Copy

public class RegisterUserModel

{

[Required(ErrorMessage = "Email is required.")]

[Remote(action: "CheckEmail", controller: "Account", ErrorMessage = "Custom error message.")]

public string Email { get; set; }

}

**Step 2: Implement the Combined Validation Logic**

csharp

Copy

public class AccountController : Controller

{

private readonly IDbConnection \_connection;

public AccountController(IDbConnection connection)

{

\_connection = connection; // Injected via DI

}

[HttpGet]

public async Task<IActionResult> CheckEmail(string email)

{

// Check 1: Is email already registered?

bool emailExists = await \_connection.ExecuteScalarAsync<bool>(

"SELECT COUNT(1) FROM Users WHERE Email = @Email",

new { Email = email }

);

if (emailExists)

{

return Json("Email is already registered.");

}

// Check 2: Is the domain valid?

string domain = email.Split('@').LastOrDefault();

if (domain != "allowed-domain.com")

{

return Json("Email domain is not allowed.");

}

// If all validations pass

return Json(true);

}

}

**Key Points**

1. **Single Action, Multiple Checks**:
   * The CheckEmail action performs both validations (database existence and domain check).
   * Return Json(true) if valid, or a custom error message string if invalid.
2. **Dynamic Error Messages**:
   * The ErrorMessage in the [Remote] attribute is **ignored** if you return a custom message (e.g., return Json("Email domain is not allowed.")).
   * To use a fixed error message, return Json(false) instead of a string.
3. **Async Support**:
   * Use async/await for database calls to avoid blocking threads.

**Alternative: Separate Properties**

If you need **separate validations for different rules**, split the email into two properties (e.g., Email and Domain), each with its own [Remote] attribute. This is uncommon but possible:

csharp

Copy

public class RegisterUserModel

{

[Remote(action: "CheckEmailExists", controller: "Account")]

public string Email { get; set; }

[Remote(action: "CheckEmailDomain", controller: "Account")]

public string Domain => Email?.Split('@').LastOrDefault();

}

**Limitations**

* **Single Error at a Time**: Client-side validation will only show one error message for the field.
* **Complexity**: Combining validations into one action can make the code less modular.

Use this approach when validations are closely related. For independent validations, prefer separate actions or server-side validation in the controller.

New chat