CHECKLIST

ASP.NET Core Request Validation with Data Annotations Checklist

A practical checklist for validating ASP.NET Core request models with Data Annotations, helping teams fail fast on malformed input and verify production readiness.

ASP.NET Core Request Validation with Data Annotations

Production Checklist

Use this checklist to review request validation on ASP.NET Core APIs before release.

1) Define the validation boundary

- [ ] Confirm the DTO is the request boundary, not a domain model.
- [ ] Validate only the fields that come from the wire.
- [ ] Keep business rules out of Data Annotations when they require state, lookups, or authorization.
- [ ] Treat validation as separate from authentication and authorization.

2) Apply the right attributes

- [ ] Mark required fields with [Required].
- [ ] Limit string size with [StringLength] or [MaxLength].

- [ ] Restrict numeric input with [Range].
- [ ] Use [EmailAddress] only for basic format checks.
- [ ] Use [RegularExpression] only for simple, maintainable patterns.
- [ ] Use [Compare] only for straightforward equality checks.

3) Prevent bad input early

- [ ] Validate incoming models before controller or endpoint logic runs.
- [ ] Ensure invalid requests return a predictable 400 Bad Request response.
- [ ] Verify ModelState behavior if you do not use [ApiController].
- [ ] Confirm invalid payloads do not reach downstream services or persistence layers.

4) Verify controller and endpoint behavior

- [ ] If using [ApiController], confirm automatic validation responses are enabled.
- [ ] If using minimal APIs or custom pipelines, confirm validation is explicit and consistent.
- [ ] Check that validation failures include useful error details for clients.
- [ ] Ensure the success path assumes the model has already passed validation.

5) Review common request rules

- [ ] Required fields are enforced for mandatory inputs.
- [ ] Length limits match storage, downstream API, and security expectations.
- [ ] Ranges match valid business bounds.
- [ ] Formats are strict enough to block obvious junk input but not overly restrictive.
- [ ] Cross-field dependencies are handled outside basic annotations when needed.

6) Check for operational safety

- [ ] Reject malformed payloads before expensive business logic runs.
- [ ] Reduce noisy logs from null, empty, or malformed values.
- [ ] Confirm validation helps protect downstream dependencies from avoidable calls.
- [ ] Pair validation with rate limiting for abusive traffic.
- [ ] Pair validation with authorization policies for protected actions.

7) Look for limitations and edge cases

- [ ] Test cross-property rules such as date ordering separately.
- [ ] Test tenant-specific or role-specific rules separately.
- [ ] Confirm annotations do not create a false sense of full business validation.
- [ ] Review any custom validation attributes for clarity and maintainability.
- [ ] Avoid encoding complex domain logic into regular expressions.

8) Validate error handling and client experience

- [ ] Confirm response bodies are consistent across endpoints.
- [ ] Confirm client teams can understand and act on validation messages.
- [ ] Verify error formatting in the target ASP.NET Core version.
- [ ] Test malformed requests in integration and contract tests.
- [ ] Ensure invalid requests are rejected before side effects occur.

9) Production readiness review

- [ ] Every public request DTO has appropriate validation attributes.

- [ ] No endpoint depends on manual, duplicated validation checks alone.
- [ ] Validation behavior is documented for API consumers.
- [ ] Security controls and validation controls are both in place.
- [ ] Common malformed payloads are covered by automated tests.

Quick Go-Live Questions

- [ ] Does the DTO reject missing required values?
- [ ] Are size and range limits reasonable for real clients?
- [ ] Are validation errors predictable and consistent?
- [ ] Are authorization and validation handled as separate concerns?
- [ ] Are domain rules validated outside Data Annotations when necessary?

Minimal review example

Review note

Data Annotations are best for simple, structural validation at the request boundary. Use them to fail fast on malformed input, then apply separate checks for authorization, domain logic, and abuse protection.