CHECKLIST

ASP.NET Programming Production Readiness Checklist

A practical operational checklist for reviewing ASP.NET applications before production, focused on request flow, security, validation, observability, and deployment readiness.

ASP.NET Programming Production Readiness Checklist

Use this checklist to review an ASP.NET application from an operational perspective before production release.

1) Request flow and application structure

- [ ] Routes are defined clearly and do not overlap unintentionally.
- [ ] Middleware order is documented and reviewed.
- [ ] Authentication runs before authorization where required.
- [ ] Validation occurs before business logic executes.
- [ ] Controllers or endpoints remain thin and delegate work to services.
- [ ] Cross-cutting concerns are handled in middleware, filters, or policies rather than duplicated in handlers.
- [ ] Error handling is centralized and produces consistent responses.

2) Authentication and authorization

- [ ] The authentication scheme is selected and configured explicitly.

- [ ] Anonymous access is allowed only where intended.
- [ ] Authorization policies are defined centrally.
- [ ] Role-based and claim-based access rules are reviewed for sensitive actions.
- [ ] Privileged actions are not protected only by client-side checks.
- [ ] Access-denied and unauthenticated responses are consistent and auditable.
- [ ] Service-to-service access is handled with an appropriate trust model.

3) Input validation and request handling

- [ ] Request models validate required fields, formats, ranges, and lengths.
- [ ] Invalid payloads are rejected before downstream processing.
- [ ] Validation errors are returned in a predictable format.
- [ ] Nullable and optional fields are intentionally designed.
- [ ] Large payloads, unexpected types, and malformed requests are handled safely.
- [ ] Model binding behavior is understood for each endpoint.
- [ ] Dangerous defaults are not relied on silently.

4) Security controls

- [ ] TLS is enforced at the edge and for required backend hops.
- [ ] Secrets are stored outside source code and configuration is environment-aware.
- [ ] Sensitive data is not logged in plaintext.
- [ ] CSRF protections are configured where browser-based sessions are used.
- [ ] CORS rules are narrowly scoped for the intended clients.
- [ ] Rate limiting or throttling is in place for shared or sensitive endpoints.
- [ ] Security headers and other edge protections are reviewed where relevant.
- [ ] Input is treated as untrusted even for internal systems.

5) Observability and diagnostics

- [ ] Requests include correlation or trace identifiers.
- [ ] Logs are structured and searchable.
- [ ] Error logs distinguish validation, authorization, dependency, and system failures.
- [ ] Metrics exist for latency, throughput, error rate, and saturation.
- [ ] Health checks reflect real dependency status where appropriate.
- [ ] Alerts are defined for actionable operational thresholds.
- [ ] Diagnostic output is sufficient to troubleshoot incidents without exposing sensitive data.

6) Reliability and performance

- [ ] Expensive work is not performed in middleware unless necessary.
- [ ] Dependency calls have timeouts and failure handling.
- [ ] Retries are controlled and do not amplify load during outages.
- [ ] Caching is intentional and safe for the data being served.
- [ ] Blocking operations are reviewed for impact on throughput.
- [ ] Memory-heavy or long-running requests are identified and controlled.
- [ ] Concurrency and async behavior are reviewed for correctness.

7) Deployment and environment readiness

- [ ] Environment-specific settings are separated from code.
- [ ] Production configuration is validated before release.
- [ ] Feature flags or rollout controls are used where needed.
- [ ] Database migrations and schema changes are planned and reversible where possible.

- [ ] Rollback steps are documented.
- [ ] Application startup failure modes are tested.
- [ ] Runtime dependencies are documented and available in the target environment.

8) Maintainability and governance

- [ ] Framework conventions are documented for the team.
- [ ] The location of routing, policies, validation, and middleware is easy to find.
- [ ] Shared patterns are used consistently across endpoints.
- [ ] Exceptions to standard patterns are justified and documented.
- [ ] Code reviews include operational checks, not just functional correctness.
- [ ] Ownership of controllers, services, and policies is clear.
- [ ] Technical debt in request handling is tracked and visible.

9) Production verification questions

Before go-live, answer these questions clearly:

- [ ] What happens to an unauthenticated request?
- [ ] What happens to an authenticated but unauthorized request?
- [ ] What happens to an invalid request body?
- [ ] What happens when a downstream dependency is slow or unavailable?
- [ ] Where are request failures logged?
- [ ] How do operators trace one request across the system?
- [ ] What controls prevent abuse or accidental overload?
- [ ] Which parts of the behavior are handled by framework features versus application code?

10) Final release check

- [ ] Security review completed.
- [ ] Validation paths tested with valid and invalid inputs.
- [ ] Authorization rules verified for intended user groups.
- [ ] Observability confirmed in a production-like environment.
- [ ] Failure modes tested and understood.
- [ ] Configuration checked for the target environment.
- [ ] Rollback and support procedures are ready.

Notes

Use this checklist as a release gate, not a one-time review. ASP.NET programming is operationally successful when request handling is predictable, secure, observable, and easy to support under real-world load.