VectraOps

Knowledge Library

TEMPLATE

JWT and Refresh Token Readiness Checklist for .NET APIs

A practical, vendor-neutral PDF checklist for evaluating whether a .NET API authentication design with JWT access tokens and refresh tokens is ready for production.

JWT and Refresh Token Readiness Checklist for .NET APIs

Use this checklist to validate a .NET API authentication design that uses short-lived JWT access tokens plus refresh tokens for session continuity.

---

1) Confirm the architecture matches the use case

- [ ] The client needs persistent sign-in state.
- [ ] The API benefits from short-lived bearer tokens.
- [ ] The authentication model is not being used as a default for machine-to-machine workloads.
- [ ] The refresh token is required only where user or device continuity is needed.
- [ ] The team can clearly state why refresh tokens are better than reauthentication for this scenario.

Notes

- Browser apps, mobile apps, and operator portals are common fits.

- Service-to-service traffic often needs a different flow.

---

2) Validate access token design

- [ ] Access tokens are short-lived.
- [ ] Access tokens are signed with a trusted key.
- [ ] The API validates the token signature on every request.
- [ ] The API validates issuer.
- [ ] The API validates audience.
- [ ] The API validates expiration.
- [ ] The API validates required claims and roles/scopes.
- [ ] The API rejects tokens that do not match the intended trust boundary.

Acceptance criteria

- A stolen access token has a limited useful lifetime.
- A valid token cannot be reused outside its expected issuer and audience.

---

3) Validate refresh token handling

- [ ] Refresh tokens are stored separately from access tokens.
- [ ] Refresh tokens are treated as long-lived credentials.
- [ ] Refresh tokens are never accepted as general API authorization tokens.
- [ ] The token endpoint is the only place refresh tokens are exchanged for new access tokens.
- [ ] Refresh tokens are bound to a clear scope such as user, device, session, or client.

- [ ] Refresh tokens have an explicit expiration policy.
- [ ] Refresh token reuse is detected or prevented.

Acceptance criteria

- A refresh token cannot be used as a bearer token for normal API calls.
- The system can distinguish legitimate renewal from suspicious reuse.

---

4) Confirm rotation and revocation behavior

- [ ] Refresh token rotation is enabled or intentionally disabled with a documented reason.
- [ ] If rotation is enabled, the previous token is invalidated after exchange.
- [ ] Replayed refresh tokens are rejected.
- [ ] The system supports revocation for individual sessions or devices.
- [ ] The system supports bulk revocation when an account is compromised.
- [ ] Revocation events are recorded and measurable.

Questions to answer before release

- How is a stolen refresh token invalidated?
- What happens if the client retries an already-rotated token?
- Can support revoke one session without logging the user out everywhere?

---

5) Review token storage and transport

- [ ] Tokens are transmitted only over TLS.

- [ ] Access tokens are not written to logs.
- [ ] Refresh tokens are not written to logs.
- [ ] Sensitive token values are not included in exception messages.
- [ ] Browser storage choice is intentional and risk-reviewed.
- [ ] Mobile storage uses platform-appropriate secure storage.
- [ ] Server-side storage is protected with access controls and encryption where appropriate.
- [ ] Secrets and signing keys are managed consistently with other sensitive configuration.

Storage guidance

- Prefer the least exposed storage option that still supports the user experience.
- Treat refresh tokens with the same seriousness as other long-lived credentials.

---

6) Verify logging and observability

- [ ] Authentication success and failure events are logged without exposing token contents.
- [ ] Refresh token exchange failures are tracked.
- [ ] Token revocation events are tracked.
- [ ] Suspicious replay patterns are visible to operations or security teams.
- [ ] Logs support incident response without leaking secrets.
- [ ] Metrics exist for token issuance, refresh, rejection, and revocation.

Minimum observability set

- Issued access tokens
- Issued refresh tokens

- Refresh attempts
- Failed refresh attempts
- Revocations
- Replay detections

---

7) Check failure handling

- [ ] Expired access tokens produce controlled responses.
- [ ] Expired or revoked refresh tokens produce controlled responses.
- [ ] Error messages do not reveal internal validation logic.
- [ ] The client knows when to retry, reauthenticate, or stop.
- [ ] Token endpoint failures do not expose stack traces or implementation details.
- [ ] The application degrades safely when refresh is unavailable.

Desired behavior

- Access token expiration should prompt a refresh attempt.
- Refresh token failure should trigger a clean sign-in path.

---

8) Define session policy clearly

- [ ] Session duration is documented.
- [ ] Access token lifetime is shorter than refresh token lifetime.
- [ ] Inactivity and absolute expiration rules are defined.
- [ ] Session scope is documented per client type.
- [ ] Users understand when they will be signed out.

- [ ] Administrators can describe the impact of compromise.

Policy questions

- How long may a user stay signed in?
- Does inactivity end the session?
- Does each device get its own refresh token?
- Can the user remain signed in across browser restarts?

---

9) Confirm production readiness

- [ ] The team can answer where the refresh token is stored.
- [ ] The team can answer how the refresh token is protected.
- [ ] The team can answer how the refresh token is rotated.
- [ ] The team can answer how the refresh token is revoked.
- [ ] The API validation rules are documented.
- [ ] The incident response process includes token revocation.
- [ ] The design has been reviewed for replay and theft risk.
- [ ] The design has been tested with expired, revoked, and rotated tokens.

Go-live gate

Do not ship until every one of the following is true:

- [ ] Access tokens are short-lived.
- [ ] Refresh tokens are protected and trackable.
- [ ] Rotation or replay detection is defined.
- [ ] Revocation is operationally supported.

- [ ] Logs and errors do not leak sensitive details.
- [ ] The client experience is documented for refresh failure.

---

10) Quick decision summary

This model fits when:

- [ ] The client needs persistent sign-in.
- [ ] Short-lived API tokens are preferred.
- [ ] The team can support refresh token storage and revocation.
- [ ] The operational team can monitor token lifecycle events.

This model is a poor fit when:

- [ ] The workload is machine-to-machine and has no user session.
- [ ] Refresh token storage would be hard to secure.
- [ ] The system cannot support server-side tracking or revocation.
- [ ] The team cannot define token lifecycle ownership.

Final review

Before production, make sure you can state the following in one sentence each:

- What the access token is for.
- What the refresh token is for.

- Where the refresh token is stored.
- How the refresh token is protected.
- How rotation works.
- How revocation works.
- What happens when refresh fails.
- How compromise is contained.

If any of those answers are unclear, the design is not ready yet.