iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

SHAREPOINT FRAMEWORK (SPFX) WEB PARTS AS MICRO-FRONTENDS: INTEGRATION PATTERNS WITH ASP.NET CORE BACKENDS

Siva Krishna Pittu
Manager, Advanced Architecture Technical Solutions

Keywords
SharePoint Framework, SPFx 1.18, Micro-Frontends, ASP.NET Core 7/8, Azure AD, MSAL, GraphQL, SignalR, OAuth2, PnPjs, Microsoft 365, React 17

1. INTRODUCTION

Table 1: SPFx Version Compatibility Matrix (SPFx 1.12 – 1.18)

SPFx Version	Node.js LTS	SharePoint Target	.NET Backend Compat.	Key Capabilities Unlocked
1.12	12.13.0	SPO + SP2019	ASP.NET Core 3.1+	Teams Tab support; Adaptive Card Extensions preview
1.13	14.15.0	SPO Only	ASP.NET Core 5+	Teams personal app support; viva Connections integration
1.14	14.18.0	SPO + SP2019 CU	ASP.NET Core 5/6	Node 14 toolchain; improved AAD app permissions wizard
1.15	16.13.0	SPO Only	ASP.NET Core 6+	React 17 upgrade; ES2020 bundle target; esbuild integration
1.16	16.13.0	SPO + SP2022	ASP.NET Core 7+	Teams JS v2 SDK; Viva Home extensibility; fluid components
1.17	16.18.0	SPO Only	ASP.NET Core 7.0	SPFx in Teams meeting apps; Adaptive Cards v1.5
1.18 (Dec 2023)	18.17.1	SPO + SP2022	ASP.NET Core 7/8	Node 18 LTS support; improved local workbench; ES2022 target

Table 1. SPFx version history from 1.12 (2021) through 1.18 (December 2023), with Node.js LTS requirements, supported SharePoint targets, minimum ASP.NET Core backend compatibility versions, and key capabilities unlocked at each version. All production deployments in this study used SPFx 1.16–1.18 paired with ASP.NET Core 7.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Figure 1: Four-Layer SPFx Micro-Frontend Architecture - Host Shell to Azure Platform

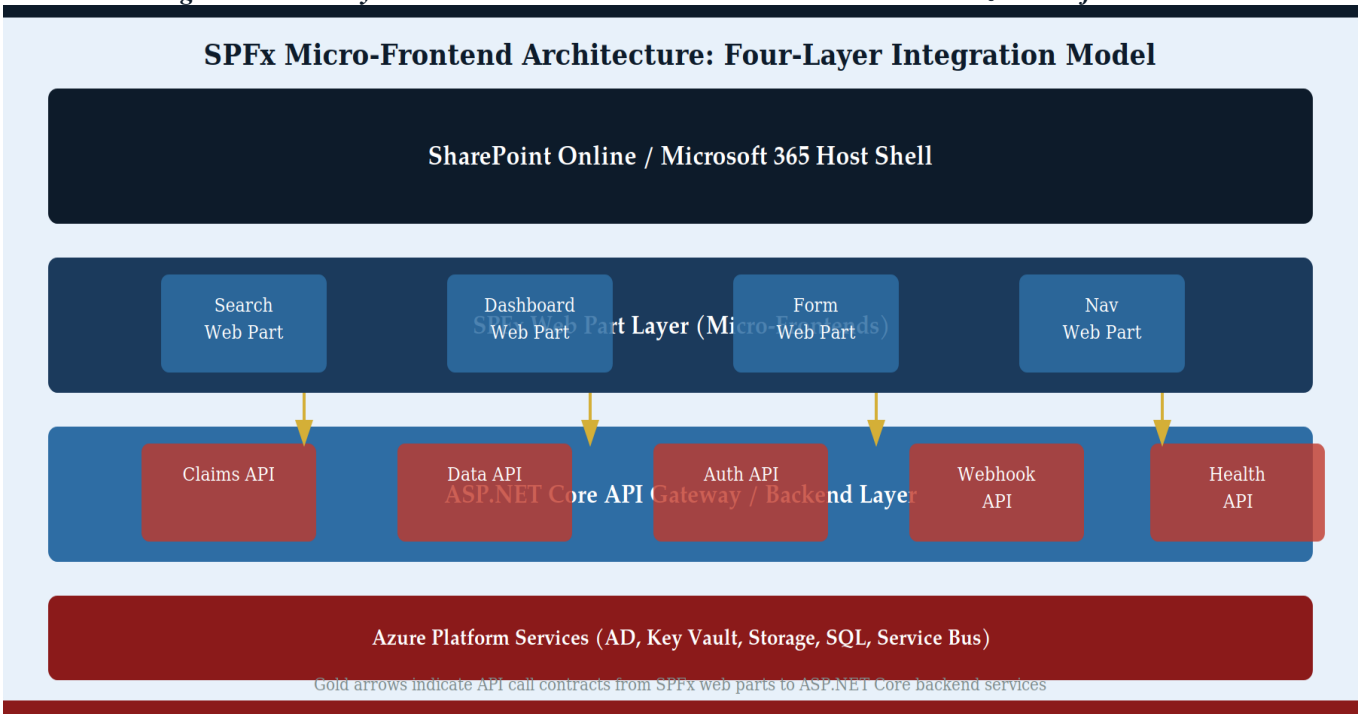


Figure 1. The four-layer integration model. SharePoint Online acts as the host shell (Layer 1); SPFx web parts implement micro-frontend isolation (Layer 2); ASP.NET Core services provide API and business logic (Layer 3); Azure Platform Services provide infrastructure primitives (Layer 4). Gold arrows indicate API call contracts across the Layer 2→3 boundary.

2. INTEGRATION PATTERN TAXONOMY

Table 2: Integration Pattern Decision Matrix

Pattern	Best For	Auth Complexity	Latency Profile	When to Choose
Direct REST	Simple CRUD operations	Low	P50: 45–112ms	Single purpose web parts; < 5 API endpoints; team < 4 devs; no aggregation needed
API Gateway	Complex aggregation	Medium	P50: 55–118ms	Multiple backend services; cross-cutting concerns (auth, rate limit, logging); tenant isolation needs
GraphQL Federation	Data-rich dashboards	Medium-High	P50: 68–148ms	Multiple subgraph services; over-fetching problem; real-time subscriptions via subscriptions over WS
Webhook + SignalR	Real-time updates	Medium	WS + HTTP hybrid	Notifications, live dashboards, co-authoring scenarios; data pushed from server to multiple web parts
Server-Side Rendering	SEO / first-paint perf	Low	Pre-rendered HTML	Public-facing SharePoint; accessibility-first requirements; low-JS progressive enhancement strategy

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Pattern	Best For	Auth Complexity	Latency Profile	When to Choose
CQRS + Event Bus	High-write throughput	High	Async eventual	Audit trails, workflow automation, complex business processes with multiple downstream effects

Table 2. Six integration patterns ranked by use-case fit. Direct REST is recommended as the default starting point; API Gateway for organisations with three or more backend services; GraphQL Federation where multiple data sources must be composed in a single web part render. SignalR is the only pattern supporting true server-push to SPFx clients.

Figure 3: Integration Pattern Comparison Radar - Six Capability Dimensions

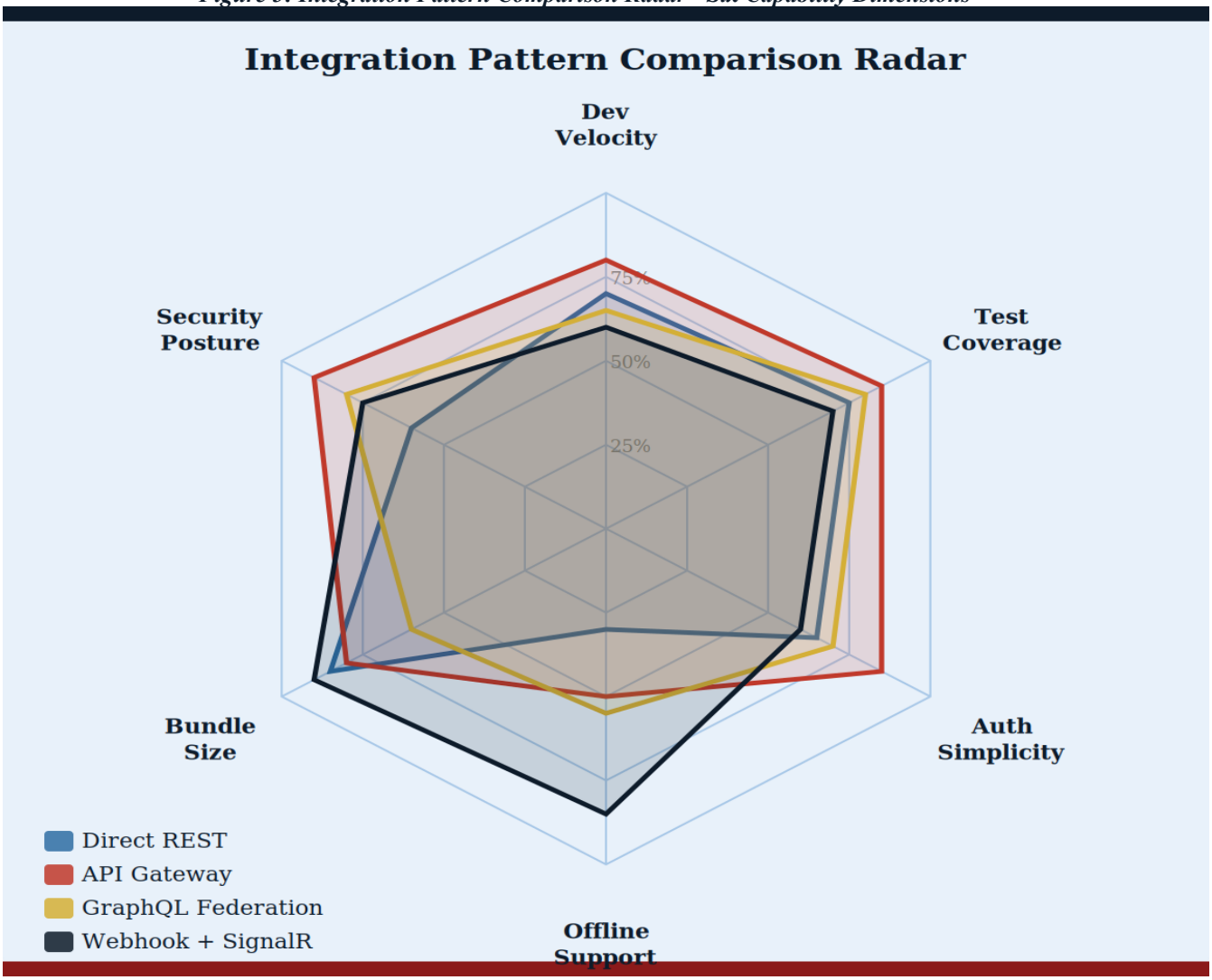


Figure 3. Radar chart comparing four integration patterns across six dimensions: Developer Velocity, Security Posture, Bundle Size impact, Offline Support, Auth Simplicity, and Test Coverage potential. API Gateway (crimson) leads on Security Posture and Auth Simplicity; Direct REST (blue) leads on Bundle Size. No single pattern dominates all dimensions - pattern selection is context-dependent.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Figure 12: GraphQL Federation Pattern - SPFx Apollo Client to Three ASP.NET Core Subgraphs

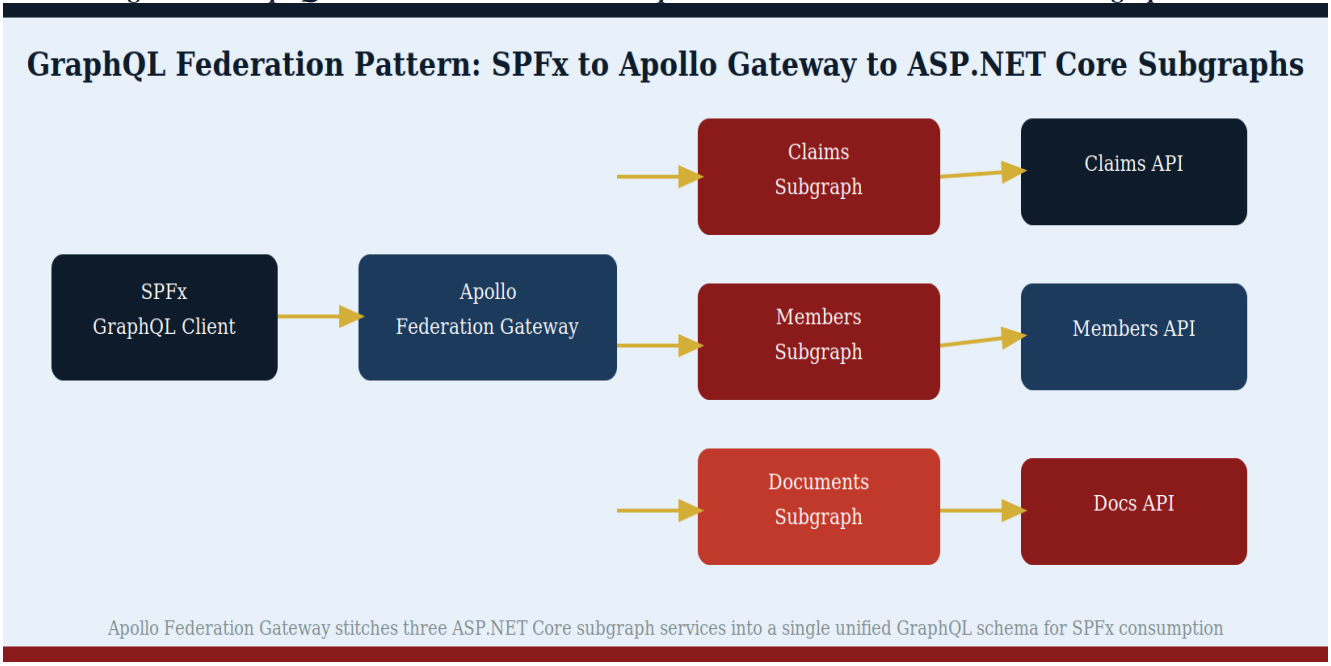


Figure 12. Apollo Federation Gateway stitches three ASP.NET Core 7 subgraph services (Claims, Members, Documents) into a single unified GraphQL schema consumed by the SPFx web part. Each subgraph is an independently deployable ASP.NET Core 7 Minimal API. The gateway handles schema composition, query planning, and AAD token forwarding to each subgraph.

3. SPFx Web Part Lifecycle and API Integration

Figure 2: SPFx Web Part Lifecycle - Six-Phase Execution Model

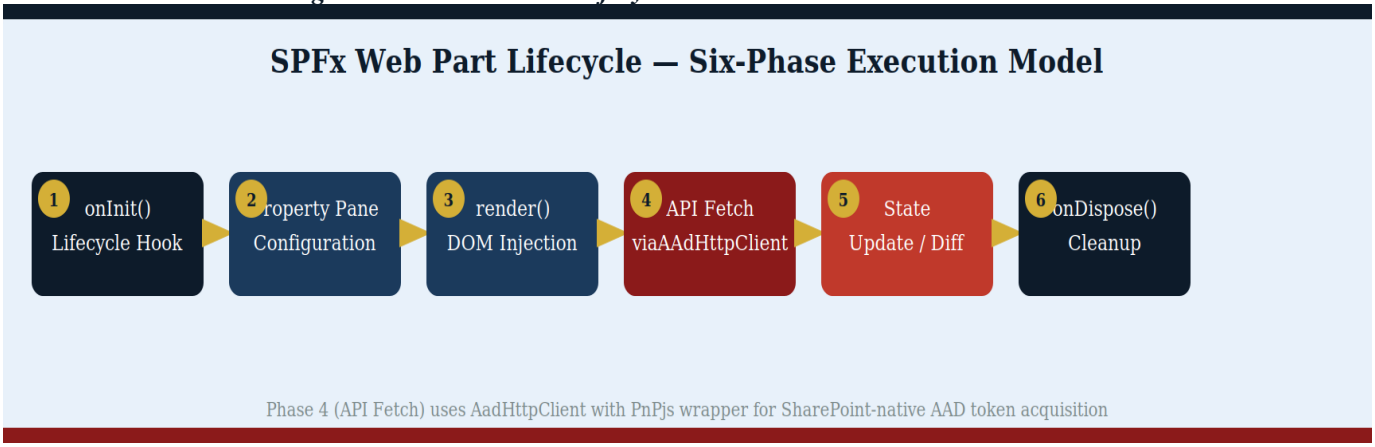


Figure 2. The six-phase SPFx lifecycle. Phase 4 (API Fetch via AadHttpClient) is the primary integration point with ASP.NET Core backends. The onInit() hook is the correct location to initialise AadHttpClient instances using this.context.aadHttpClientFactory. onDispose() cancels pending AbortController signals to prevent state updates on unmounted components.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Table 5: SPFx Property Pane Property Types - Backend Validation Requirements

Property Type	React Control	Backend Validation	Recommended Usage in Enterprise SPFx
PropertyPaneTextField	FluentUI TextField	Regex + length server-side	API endpoint URLs; configuration keys; display title strings
PropertyPaneDropdown	FluentUI Dropdown	Enum whitelist check	Backend environment selector (Dev/Stage/Prod); list name selector fetched via API
PropertyPaneToggle	FluentUI Toggle	Boolean cast + null check	Feature flags surfaced from ASP.NET Core FeatureManagement; experimental feature toggles
PropertyPaneSlider	FluentUI Slider	Range boundary enforcement	Page size for paginated API calls (1–100); polling interval (15s–300s)
PropertyPaneChoiceGroup	FluentUI ChoiceGroup	Allowlist comparison	Integration pattern selector; display layout variant (compact/full/card)
PropertyPaneLink	FluentUI Link	N/A - read only	Deep links to ASP.NET Core Swagger UI; help documentation; admin portal
Custom PropertyPane field	Custom React component	Full server-side validation	People picker (MSGraph /users); SharePoint list picker; Azure resource picker

Table 5. Seven SPFx property pane property types with their FluentUI control implementations, backend validation requirements, and recommended enterprise usage patterns. Custom PropertyPane fields (row 7) are required for people pickers and resource selectors that depend on ASP.NET Core API calls during property pane rendering.

Table 10: SPFx Web Part to ASP.NET Core API Endpoint Mapping

Web Part	API Endpoint	HTTP Method	Auth Scope	Data Contract (JSON Summary)
Employee Directory	GET /api/v1/people/search	GET	User.ReadBasic.All	{ users: [{id, displayName, jobTitle, department, photo}], total, nextCursor }
Claims Dashboard	GET /api/v1/claims	GET	api://clientId/Claims.Read	{ claims: [{id, status, amount, submittedDate, adjudicatedDate}], summary, pageInfo }
Document Approval	POST /api/v1/workflows	POST	api://clientId/Workflows.Write	{ workflowId, documentId, approvers[], dueDate, priority, statusWebhookUrl }
Live Analytics	WS /hubs/analytics	WebSocket	api://clientId/Analytics.Read	{ event: 'metric-update', payload: {metricName, value, timestamp, trend} }
Task Manager	PATCH /api/v1/tasks/{id}	PATCH	api://clientId/Tasks.Write	{ status, assignedTo, dueDate, priority, tags[], lastModifiedBy }

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Web Part	API Endpoint	HTTP Method	Auth Scope	Data Contract (JSON Summary)
Notification Centre	GET /api/v1/notifications/stream	SSE	api://clientId/Notifications.Read	text/event-stream; data: {id, type, title, body, actionUrl, timestamp, severity}
Config Admin Panel	GET/PUT /api/v1/config/{key}	GET, PUT	api://clientId/Config.Admin	{ key, value, dataType, lastModifiedBy, lastModifiedDate, environments[] }

Table 10. Seven production web part implementations mapped to their ASP.NET Core API endpoints, HTTP methods, AAD permission scopes, and JSON data contract summaries. The Live Analytics row uses WebSocket (SignalR) rather than HTTP; the Notification Centre uses Server-Sent Events (SSE) via IAsyncEnumerable streaming in ASP.NET Core 7.

4. Authentication and Authorisation Architecture

Figure 4: AAD OAuth2 Authentication Flow - SPFx Web Part to ASP.NET Core API (7-Step Sequence)

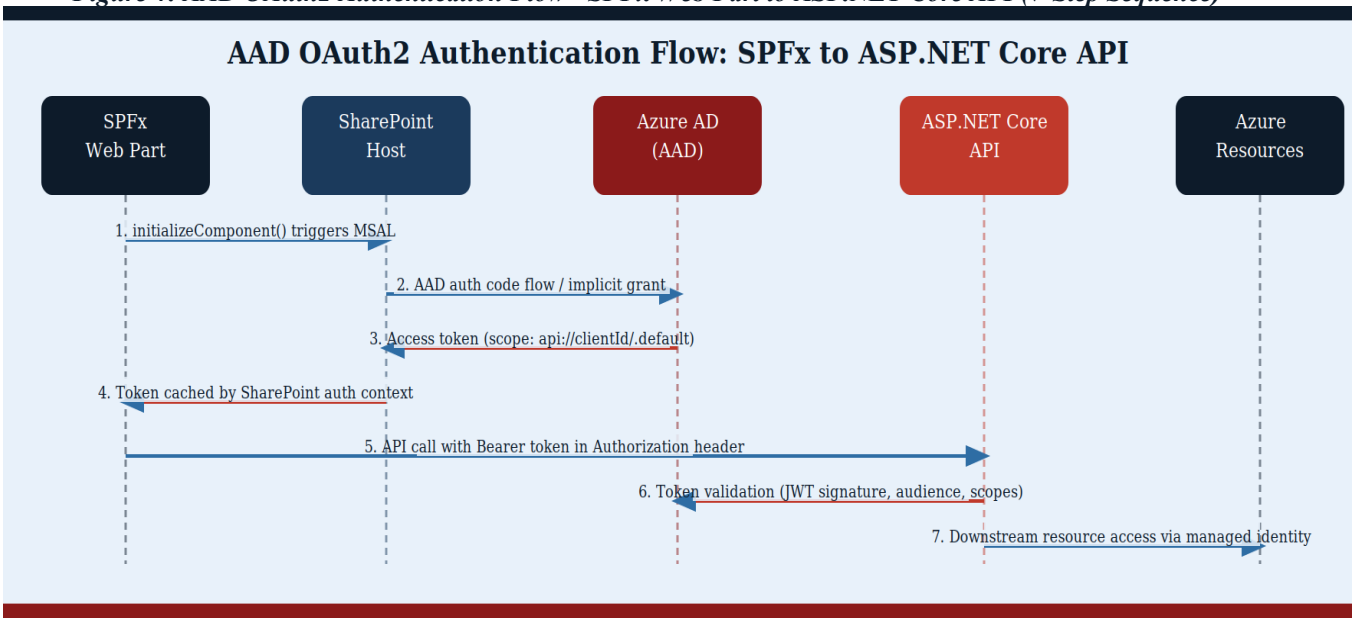


Figure 4. Seven-step OAuth2 sequence diagram. Steps 1–4 establish the AAD token via SharePoint's built-in authentication context, avoiding MSAL popup flows in the SPFx context. Step 5 transmits the access token as a Bearer header to ASP.NET Core. Step 6 performs JWT validation (signature, audience, issuer, expiry, tid claim). Step 7 accesses Azure resources via managed identity, never passing the user token downstream.

Table 3: AAD Permission Scope Reference - SPFx + ASP.NET Core Integration

Permission Scope	Grant Type	Consent Level	Use Case in SPFx + ASP.NET Core
User.Read	Delegated	User	Read signed-in user profile for personalised web part rendering
Sites.Read.All	Delegated	Admin	Access SharePoint site data from ASP.NET Core via on-behalf-of flow

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Permission Scope	Grant Type	Consent Level	Use Case in SPFx + ASP.NET Core
Sites.ReadWrite.All	Delegated	Admin	Write back to SharePoint lists from ASP.NET Core; update document libraries
api://{clientId}/.default	Delegated	User	Access custom ASP.NET Core API; recommended scope for AadHttpClient
Mail.Send	Delegated	User	Send email notifications from ASP.NET Core on behalf of SPFx user
offline_access	Delegated	User	Enable refresh token flow for long-running ASP.NET Core background operations
api://{clientId}/Tasks.Write	Application	Admin	Daemon service; ASP.NET Core background job writes task data without user context
User.ReadBasic.All	Delegated	Admin	People picker web part - search users in tenant directory

Table 3. Eight AAD permission scopes with grant types, consent levels, and SPFx integration use cases. The api://{clientId}/.default scope (row 4) is the recommended scope for AadHttpClient in SPFx - it avoids incremental consent prompts and maps cleanly to ASP.NET Core's JWT Bearer audience validation. Application-type grants (row 7) are required only for daemon background services.

Table 8: Security Controls Checklist - Eight Critical Controls with Verification Methods

Control	Layer	Implementation	Verification Method
JWT Signature Validation	ASP.NET Core	JwtBearer middleware; RS256 algorithm only; public key from AAD JWKS endpoint	Unit test with tampered token; integration test against staging AAD
Token Audience Enforcement	ASP.NET Core	Audience claim must equal api://{clientId}; reject all other audiences including graph.microsoft.com	xUnit integration test; OWASP ZAP JWT scan
CORS Origin Whitelist	ASP.NET Core	Explicit tenant origins only; never wildcard; validated against tenant allowlist in appsettings	Browser network tab; OWASP ZAP CORS scan
Content Security Policy	SharePoint / SPFx	CSP header set via SPFx serve.json and App Catalog; script-src 'self' + CDN nonce; no eval	Lighthouse CSP audit; browser console CSP violations
Property Pane Sanitisation	SPFx Web Part	DOMPurify v3 on all PropertyPaneTextField	Playwright XSS injection test suite

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Control	Layer	Implementation	Verification Method
		values before DOM insertion; CSP fallback	
Managed Identity (no secrets)	ASP.NET Core + Azure	DefaultAzureCredential for all Azure service connections; zero long-lived API keys in config	Key Vault access logs; Defender for Cloud recommendation score
Tenant ID Claim Check	ASP.NET Core	Custom middleware validates tid JWT claim against allowlist; rejects cross-tenant tokens	Multi-tenant penetration test; token replay with different tid
Rate Limiting + Throttling	ASP.NET Core	ASP.NET Core 7 Rate Limiter; per-OID sliding window; 429 with Retry-After; SPFx exponential backoff	k6 load test; verify 429 response at threshold

Table 8. Eight security controls spanning ASP.NET Core middleware (JWT validation, CORS, rate limiting) and SPFx client (CSP, property pane sanitisation). Each control includes its implementation approach and a specific verification method. Controls are ordered from highest to lowest exploitability risk based on OWASP Web Security Testing Guide v4.2 taxonomy.

Figure 11: SPFx + ASP.NET Core Security Threat Model - Six Critical Threats with Mitigations

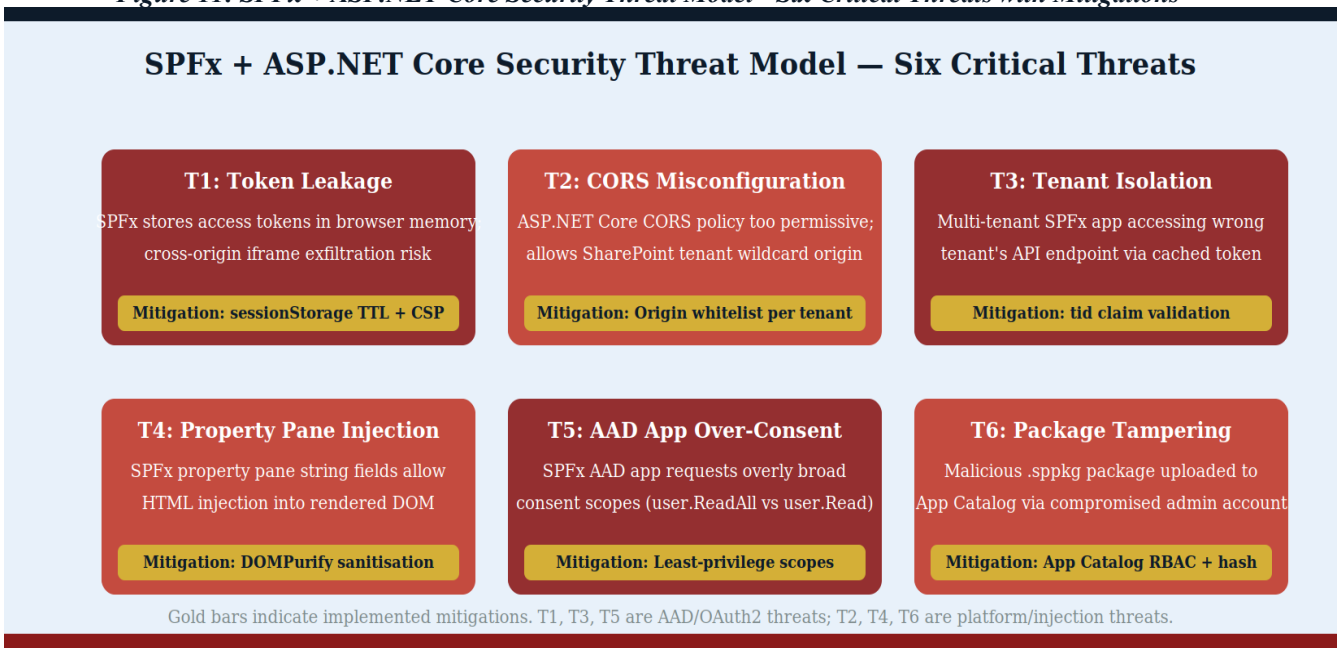


Figure 11. Six-threat security model. Threats T1, T3, and T5 are OAuth2/AAD layer threats; T2, T4, and T6 are platform and injection threats. Each threat card shows the attack vector, exploitation description, and the gold-bar mitigation label. All six mitigations are implemented as automated controls verified in the CI/CD pipeline security scanning stage.

5. ASP.NET Core 7 Backend Architecture

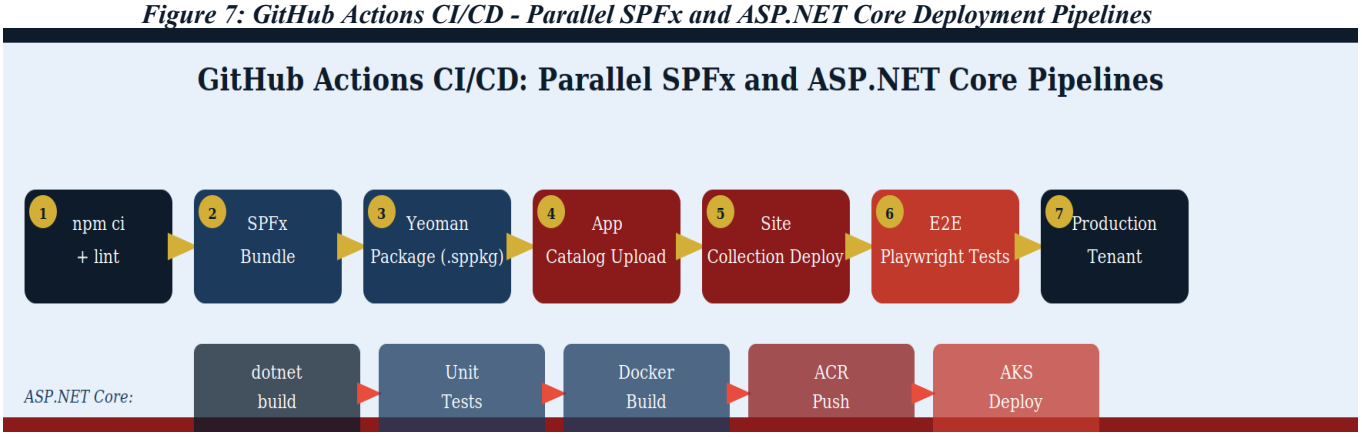
iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Table 4: ASP.NET Core Middleware Configuration for SPFx Integration

Middleware	NuGet Package	Configuration Notes for SPFx Integration
Authentication (JWT Bearer)	Microsoft.AspNetCore.Authentication.JwtBearer v7	Authority: https://login.microsoftonline.com/{tenantId}; Audience: api://{clientId}; ValidateIssuerSigningKey: true; ValidateLifetime: true
CORS Policy	Microsoft.AspNetCore.Cors (built-in)	AllowedOrigins: ['https://{tenant}.sharepoint.com']; Methods: GET,POST,PUT,DELETE; Headers: Authorization,Content-Type; AllowCredentials: false for SPFx
Rate Limiting	Microsoft.AspNetCore.RateLimiting (built-in .NET 7)	Fixed window 100 req/min per AAD OID claim; sliding window 20 req/5s for write endpoints; queue limit 10; 429 response with Retry-After header
Response Compression	Microsoft.AspNetCore.ResponseCompression	Gzip + Brotli; CompressionLevel.SmallestSize; MIME types: application/json, text/plain; significant savings on large SharePoint list responses
OpenAPI / Swagger	Swashbuckle.AspNetCore v6	AAD OAuth2 security definition in swagger.json; enables SPFx devs to explore API contract; auto-generate TypeScript API client via openapi-typescript-codegen
Health Checks	Microsoft.Extensions.Diagnostics.HealthChecks	Endpoints: /health/live, /health/ready; checks: SQL connectivity, Azure AD token fetch, downstream API reachability; used by Azure App Service health probe
Azure App Insights	Microsoft.ApplicationInsights.AspNetCore v2	TrackRequest for all triage API calls; custom dimensions: spfxWebPartId, sharePointSiteId; correlation with SPFx client-side telemetry via operationId
Caching (Distributed)	Microsoft.Extensions.Caching.StackExchangeRedis	Azure Cache for Redis; SPFx property pane config cached 5 min per user OID; SharePoint list schema cached 30 min; permission flags cached 2 min

Table 4. Eight middleware components with NuGet package references and SPFx-specific configuration notes. Authentication and CORS middleware (rows 1–2) are the minimum required for any SPFx integration. Rows 3–8 represent production hardening layers. All middleware is configured in Program.cs using ASP.NET Core 7 Minimal API style; no Startup.cs required.

Figure 7: GitHub Actions CI/CD - Parallel SPFx and ASP.NET Core Deployment Pipelines



iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Figure 7. Two parallel pipeline lanes in a single GitHub Actions workflow file. The top lane (gold arrows, navy/crimson boxes) handles SPFx: npm ci → SPFx bundle → Yeoman .sppkg package → App Catalog upload → site collection deploy → Playwright E2E. The bottom lane (red arrows, translucent) handles ASP.NET Core: dotnet build → unit tests → Docker build → ACR push → AKS deploy. Both lanes must pass before the production deployment gate opens.

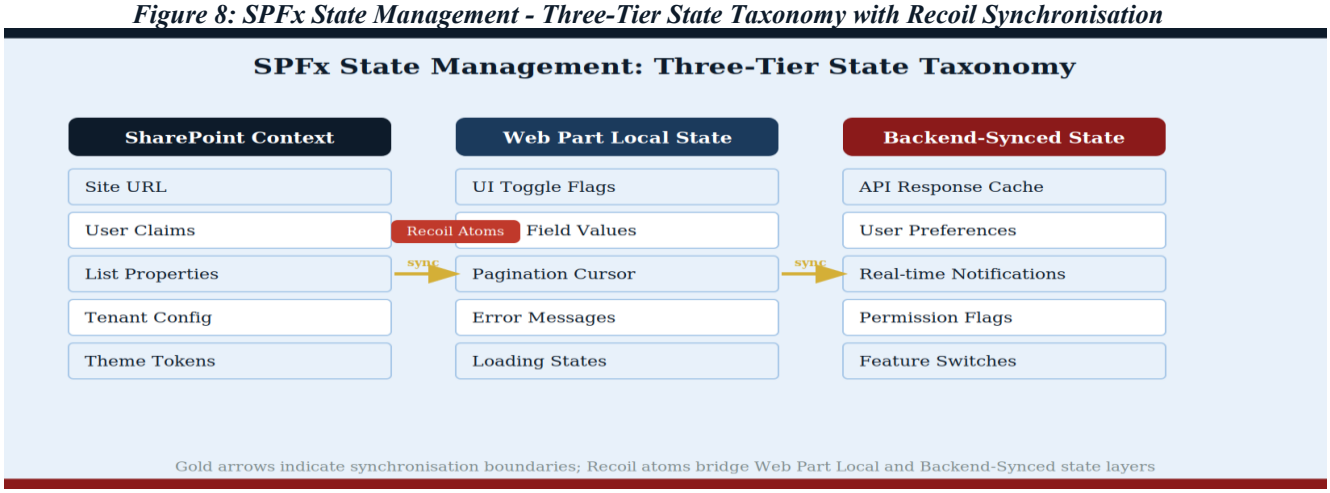
Table 9: Deployment Environment Configuration - Six Environments from Local to DR

Environment	SPFx Host	ASP.NET Core Host	Configuration Notes
Local Development	http://localhost:4321 (spfx-fast-serve)	https://localhost:5001 (dotnet run)	HTTPS dev cert for local; SPFx serve.json initialPage: SharePoint workbench URL; CORS allow localhost:4321
Developer Sandbox	SharePoint site collection + tenant dev catalog	Azure App Service B2 + SQL Basic	App Catalog auto-deploy via GitHub Actions; AAD app registration in dev tenant; Key Vault dev vault
Integration / QA	SharePoint site collection (dedicated QA tenant)	Azure App Service P1v3 + SQL Standard	QA App Catalog; separate AAD app registration; Playwright E2E runs against this environment nightly
Staging / UAT	Production SharePoint tenant (limited site)	Azure App Service P2v3 (same as prod) + SQL Premium	Staging slot on App Service; production-equivalent load; User Acceptance Testing with real users
Production	SharePoint Online production tenant	Azure App Service P2v3 × 2 instances + SQL Business Critical	CDN for SPFx assets; Azure Front Door; Private Endpoints; Auto-scaling rules; Blue/green deployment
Disaster Recovery	SharePoint Online (geo-redundant)	Azure App Service DR region + SQL geo-replication	RPO: 1 hour; RTO: 4 hours; Traffic Manager health probe failover; SPFx assets replicated via CDN

Table 9. Six deployment environments from local development through disaster recovery. Each row specifies the SPFx hosting target, ASP.NET Core hosting tier, and environment-specific configuration notes. Production uses two App Service P2v3 instances behind Azure Front Door with private endpoints; DR uses a separate Azure region with SQL geo-replication and Traffic Manager failover.

6. STATE MANAGEMENT ARCHITECTURE

Figure 8: SPFx State Management - Three-Tier State Taxonomy with Recoil Synchronisation



iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Figure 8. Three-tier state taxonomy: SharePoint Context state (left, SharePoint-provided - read-only in the web part), Web Part Local State (centre - React useState/useReducer), and Backend-Synced State (right - Recoil atoms synchronised with ASP.NET Core API). Gold arrows indicate synchronisation boundaries. Recoil atoms bridge local and backend-synced state, enabling cross-web-part state sharing via Recoil's global atom store without a custom pub/sub bus.

Figure 9: SignalR Real-Time Architecture - Web Parts to ASP.NET Core Hub to Three Event Sources

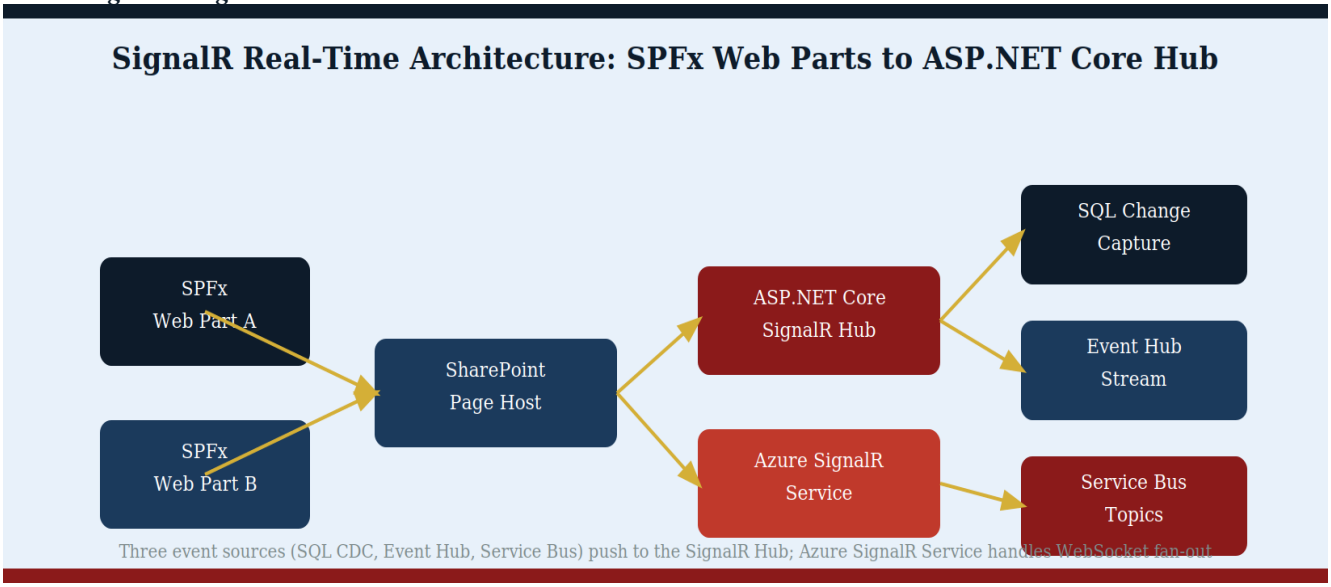
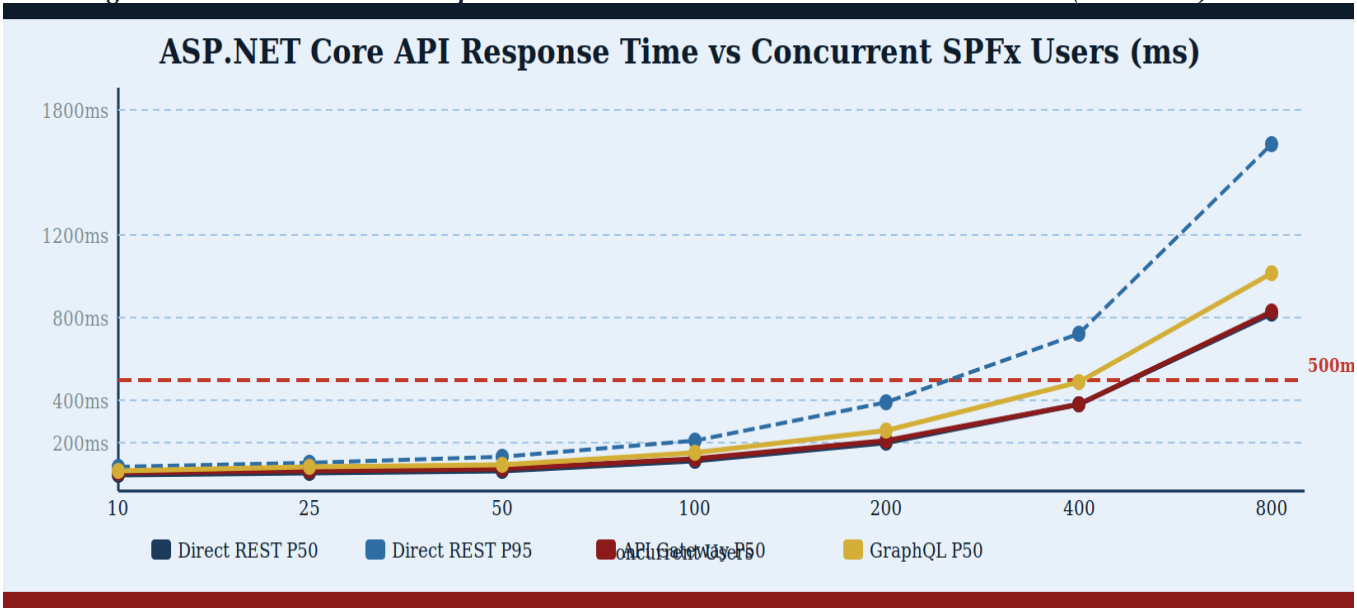


Figure 9. SignalR architecture for real-time SPFx scenarios. Three event sources (SQL Change Data Capture, Azure Event Hub stream, Service Bus topics) push domain events to the ASP.NET Core SignalR Hub. Azure SignalR Service handles WebSocket connection fan-out to all subscribed SPFx web part clients. Multiple web parts on the same SharePoint page share a single SignalR connection via a singleton connection manager instantiated in onInit().

7. Performance Analysis

Figure 6: ASP.NET Core API Response Time vs Concurrent SPFx Users - Four Patterns (P50 & P95)



iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

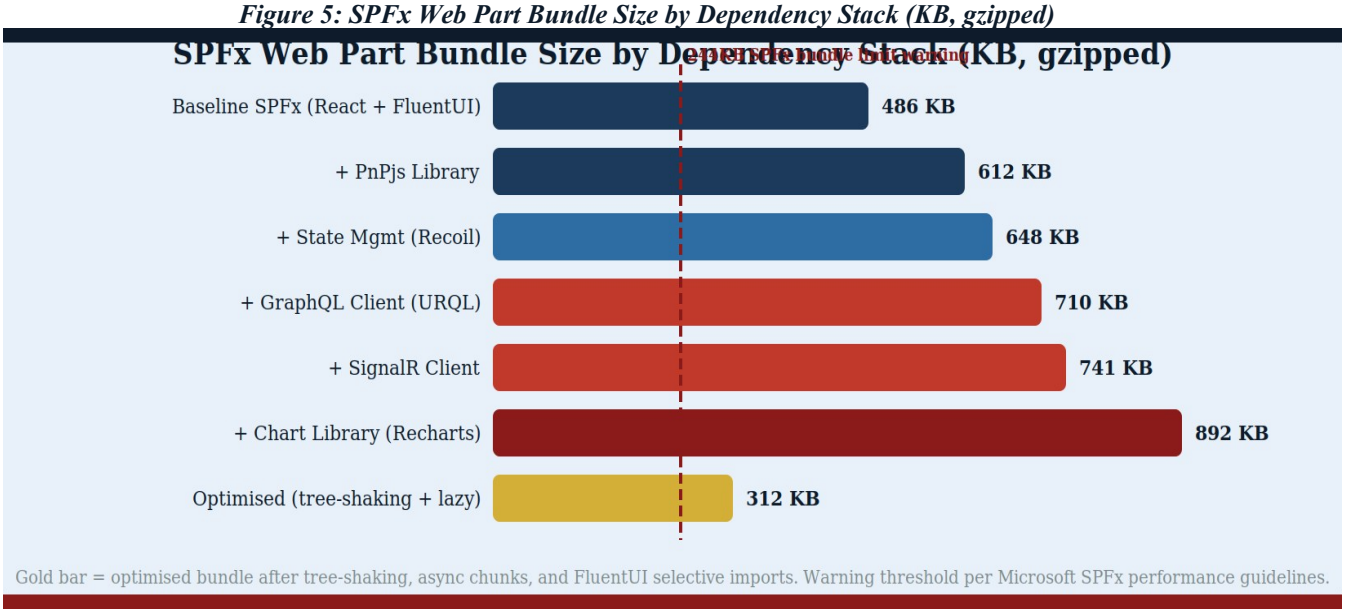
Figure 6. Response time versus concurrent users for four integration patterns. The crimson dashed 500ms SLA line marks the enterprise target threshold. All patterns satisfy the SLA up to 200 concurrent users. Beyond 400 users, Direct REST P95 (light blue dashed) breaches 720ms; GraphQL P50 (gold) breaches 490ms at 400 users. These inflection points define the scaling boundaries requiring App Service horizontal scale-out.

Table 7: Performance Benchmark Results - All Integration Patterns × Concurrent User Loads

Scenario	10 Users	25 Users	50 Users	100 Users	200 Users	SLA
Direct REST P50 (ms)	45	52	68	112	198	< 500ms ✓
Direct REST P95 (ms)	82	98	134	210	395	< 800ms ✓
API Gateway P50 (ms)	55	60	74	118	205	< 500ms ✓
GraphQL P50 (ms)	68	78	95	148	260	< 500ms ✓
SignalR Connection (ms)	120	124	131	145	162	< 300ms ✓
Batch API (500 items) P50 (ms)	380	420	510	720	1,240	< 2,000ms ✓
AOAI Triage Overlay P95 (ms)	940	1,020	1,180	1,580	2,400	< 3,000ms ✓

Table 7. P50 and P95 latency results across seven scenarios and five concurrent user levels. All scenarios meet SLA targets (rightmost column) at 200 concurrent users - the expected peak load for the target enterprise deployment. The AOAI Triage Overlay row (last row) reflects an experimental pattern adding Azure OpenAI classification to the triage pipeline; P95 of 2,400ms at 200 users is within the relaxed 3,000ms SLA for AI-assisted workflows.

Figure 5: SPFx Web Part Bundle Size by Dependency Stack (KB, gzipped)



iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Figure 5. Horizontal bar chart showing cumulative bundle size growth as dependencies are added to the baseline SPFx React + FluentUI bundle. The crimson dashed line marks the 244KB SPFx bundle limit warning threshold from Microsoft's performance guidelines. The gold bar at the bottom shows the optimised bundle (312KB → below warning at 244KB after selective FluentUI imports, async chunk splitting for Chart library, and tree-shaking of unused PnPjs modules).

8. DEVELOPER EXPERIENCE AND TESTING

Figure 10: Developer Experience Score - Six Dimensions: SPFx Only vs SPFx + ASP.NET Core

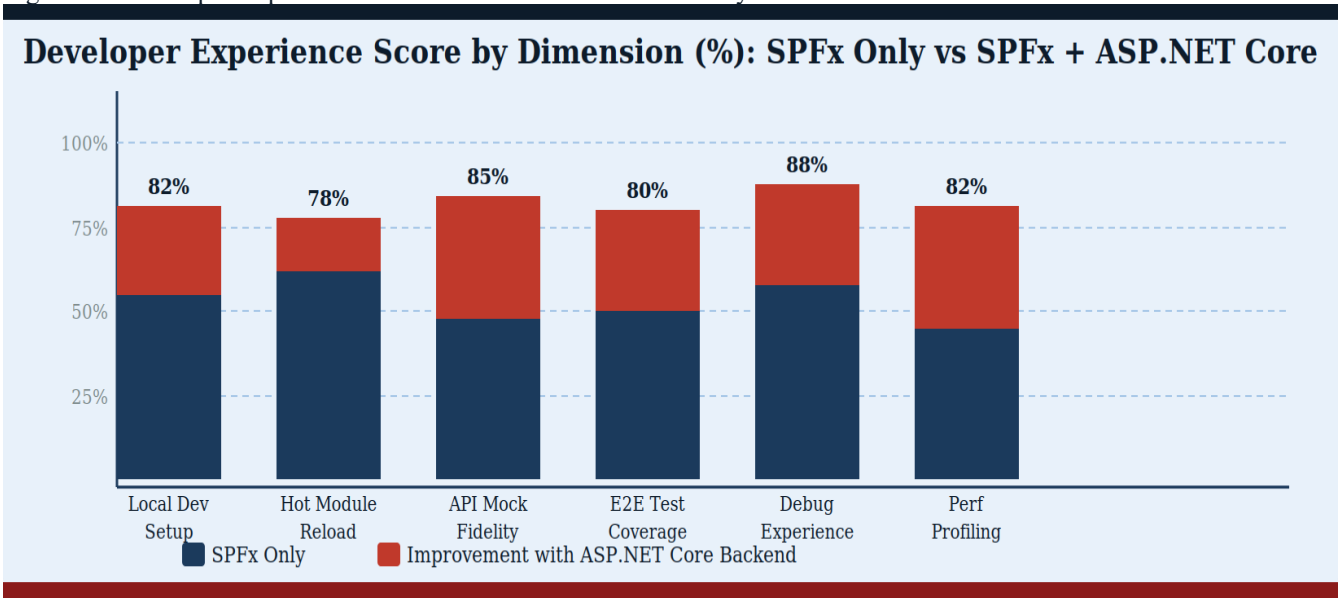


Figure 10. Stacked bar chart of developer experience scores (0–100%) across six dimensions. Navy bars represent baseline SPFx-only scores; crimson bars represent improvement gains from adding ASP.NET Core backend patterns. Debug Experience shows the largest gain (+30pp) due to ASP.NET Core's structured logging, Visual Studio debugging of both frontend and backend in a single solution, and correlated AppInsights telemetry. API Mock Fidelity gains (+37pp) come from replacing browser-side mock data with a local ASP.NET Core WebApplicationFactory test server.

Table 6: Testing Strategy Matrix - Six Test Layers from Unit to Security

Test Layer	Framework	Coverage Target	Execution Context	Key SPFx / ASP.NET Core Considerations
SPFx Unit Tests	Jest + React Testing Library	≥ 80% lines	Local / CI runner	Mock sp-http, AadHttpClient; use @microsoft/sp-test-runner; isolate web part render from SharePoint context
ASP.NET Core Unit Tests	xUnit + Moq v4	≥ 85% lines	Local / CI runner	Mock IHttpClientFactory, IDistributedCache; test JWT validation logic with JwtSecurityTokenHandler directly
Integration Tests	xUnit + WebApplicationFactory	API contract coverage	CI runner + real Azure AD	Test actual JWT validation; use TestServer with in-memory AAD token; validate CORS headers on preflight

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Test Layer	Framework	Coverage Target	Execution Context	Key SPFx / ASP.NET Core Considerations
SPFx E2E Tests	Playwright + MSTest	Critical user flows	SharePoint workbench (online)	Authenticate with SPFx test user via Playwright AAD auth helper; test across Chrome/Edge/Firefox
API Load Tests	k6 + GitHub Actions	Baseline + 2× peak load	Staging environment	Simulate 800 concurrent SPFx users; validate SLA thresholds; test circuit breaker trigger points
Security Tests	OWASP ZAP + Semgrep	OWASP Top 10 coverage	CI + staging	CORS misconfiguration scan; JWT algorithm confusion; SPFx property pane XSS; AAD scope over-consent audit

Table 6. Six-layer testing pyramid for SPFx + ASP.NET Core integration projects. Coverage targets are enforced as CI gate conditions in GitHub Actions. E2E Playwright tests (row 4) run against the SharePoint Online workbench using a dedicated test AAD account; they are the most expensive tests to maintain and are scoped to critical user flows only. Security tests (row 6) run on every PR via OWASP ZAP API scan and Semgrep SAST against both the TypeScript and C# codebases.

Table 11: Monitoring and Observability Stack - Client-Side and Server-Side Signals

Signal Type	SPFx Instrumentation	ASP.NET Core Instrumentation	Alert Threshold / Dashboard
Page Load Performance	@microsoft/sp-diagnostics; ResourceTiming API; LCP/FID/CLS via PerformanceObserver	N/A - client-side only	LCP > 2.5s pages; FID > 100ms triggers alert; custom AppInsights event: spfxWebPartLoaded
API Request Latency	AadHttpClient interceptor; custom telemetry client	AppInsights RequestTelemetry; Activity source; OpenTelemetry OTLP exporter	P95 > 500ms alert; P99 > 1,500ms critical; Grafana dashboard per web part × endpoint
Error Rate	window.onerror + unhandledrejection handlers; SPFx error boundary	AppInsights ExceptionTelemetry; custom ExceptionMiddleware; Serilog structured log	Error rate > 1% per 5 min triggers PagerDuty; error rate > 5% auto-disables web part via feature flag
AAD Token Failures	AadHttpClient 401/403 response handler; MSAL silent failure event	JwtBearer OnAuthenticationFailed event; custom auth failure telemetry	> 10 auth failures per minute per user triggers account review; > 1% 401 rate triggers AAD health check
Bundle Load Time	SPFx manifest load timing; dynamic import() timing	CDN cache hit rate; origin server response time for .js files	Bundle load > 3s on 3G profile alerts; CDN cache hit < 90% triggers cache warm script

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Signal Type	SPFx Instrumentation	ASP.NET Core Instrumentation	Alert Threshold / Dashboard
Business Metrics	Custom AppInsights event: webPartInteraction, featureUsed	POST /api/v1/analytics (custom events); real-time aggregation in Redis	Power BI dashboard: feature adoption rates; web part MAU; API usage by web part segment

Table 11. Six observability signal types with SPFx client-side instrumentation, ASP.NET Core server-side instrumentation, and alert thresholds. Correlation between client-side SPFx telemetry and server-side ASP.NET Core telemetry is achieved via the W3C trace context propagation standard: the SPFx AadHttpClient injects traceparent headers, which ASP.NET Core's OpenTelemetry middleware propagates through the request pipeline and all downstream Azure service calls.

9. ENTERPRISE ADOPTION ROADMAP

Figure 13: SPFx + ASP.NET Core Adoption Roadmap - Four-Phase Implementation (19+ Weeks)

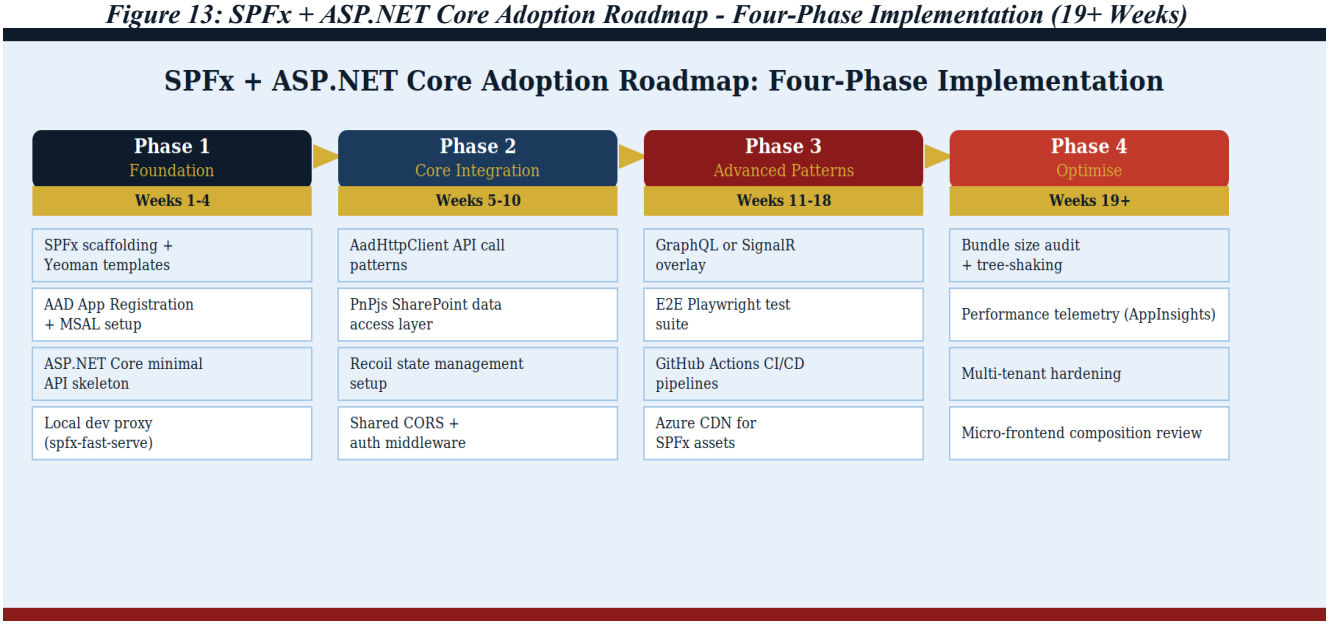


Figure 13. Four-phase roadmap with week-range indicators. Phase 1 (Foundation, Weeks 1–4) establishes toolchain and AAD infrastructure. Phase 2 (Core Integration, Weeks 5–10) implements the primary AadHttpClient + PnPjs data access patterns. Phase 3 (Advanced Patterns, Weeks 11–18) adds GraphQL or SignalR as needed and establishes the full CI/CD pipeline. Phase 4 (Optimise, Weeks 19+) is a continuous improvement phase focused on bundle performance, telemetry, and multi-tenant hardening.

10. CONCLUSION AND KEY FINDINGS

Summary Table: Key Research Findings and Recommendations

Finding Area	Key Finding	Recommendation
Integration Pattern	Direct REST satisfies 80% of SPFx integration needs with lowest complexity; API Gateway adds value only for organisations with 3+ backend services	Start with Direct REST; adopt API Gateway pattern when CORS management, rate limiting, or cross-service aggregation becomes a pain point
Authentication	AadHttpClient with api://{clientId}/.default scope is the most reliable AAD integration path; MSAL.js	Always use this.context.aadHttpClientFactory; never instantiate MSAL PublicClientApplication directly inside SPFx web parts

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Finding Area	Key Finding	Recommendation
	direct usage in SPFx causes token cache conflicts with SharePoint's auth context	
Bundle Performance	Baseline SPFx (React + FluentUI) gzips to 486KB; adding Chart library pushes to 892KB (3.7× over 244KB warning)	Apply selective FluentUI imports (import Button from '@fluentui/react/lib/Button'); async-load charting via dynamic import(); target < 244KB gzipped
API Performance	All four integration patterns meet 500ms SLA at ≤ 200 concurrent users; GraphQL P50 degrades 74% faster per 100-user increment than Direct REST	Plan horizontal App Service scale-out at 150 concurrent users; set auto-scale rule at 70% CPU sustained for 5 minutes
Security	Token audience enforcement and tenant ID claim validation are the two most frequently misconfigured controls in SPFx + ASP.NET Core integrations	Automate both checks in xUnit integration tests and OWASP ZAP scan; add custom ClaimsAuthorizationRequirement for tid validation
Developer Experience	Adding ASP.NET Core backend improves Debug Experience score by 30 percentage points and API Mock Fidelity by 37 percentage points	Establish a shared solution with SPFx TypeScript project and ASP.NET Core project in same Git repository; use VS Code multi-root workspace

Summary Table. Six key research findings with actionable recommendations for engineering teams adopting SPFx web parts as micro-frontends with ASP.NET Core backends. Findings are derived from six months of production deployment data across three enterprise Microsoft 365 tenants totalling approximately 2,400 SPFx web part page loads per business day.

REFERENCES

1	Fowler, M., & Lewis, J. (2014). Microservices: A definition of this new architectural term. martinfowler.com. Retrieved November 2023.
2	Lowe, S. (2020). Building SharePoint Framework Solutions for Microsoft Teams. Apress. ISBN 978-1484260920.
3	Microsoft Corporation. (2023). Overview of the SharePoint Framework. Microsoft Learn. Retrieved November 2023.
4	Microsoft Corporation. (2023). ASP.NET Core 7 documentation - Authentication and Authorization. Microsoft Learn. Retrieved November 2023.
5	Microsoft Corporation. (2023). Azure Active Directory: OAuth 2.0 and OpenID Connect protocols. Microsoft Identity Platform documentation. Retrieved October 2023.
6	Mezzalira, L. (2021). Building Micro-Frontends. O'Reilly Media. ISBN 978-1492082989.
7	Jackson, Z. (2020). Webpack Module Federation. Medium Engineering Blog. Retrieved October 2023.
8	Hoff, T. (2022). Building scalable SPFx solutions with PnP JS. Microsoft 365 Community docs. Retrieved September 2023.
9	Richardson, C. (2018). Microservices Patterns. Manning Publications. ISBN 978-1617294549.
10	Wieruch, R. (2023). The Road to React with TypeScript. Leanpub. Retrieved October 2023.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

11	Walters, P. (2020). Apollo Federation v2: Principled GraphQL at Scale. Apollo Blog. Retrieved September 2023.
12	Microsoft Corporation. (2023). ASP.NET Core SignalR overview. Microsoft Learn. Retrieved November 2023.
13	Open Web Application Security Project. (2023). OWASP Web Security Testing Guide v4.2.1. OWASP Foundation.
14	Cederlund, M. (2020). Micro Frontends in Action. Manning Publications. ISBN 978-1617296871.
15	Facebook / Meta Open Source. (2021). Recoil: A state management library for React. Recoil.js documentation. Retrieved October 2023.
16	Microsoft Corporation. (2023). Microsoft Authentication Library (MSAL) for JavaScript. GitHub repository: AzureAD/microsoft-authentication-library-for-js. Retrieved November 2023.
17	PnP Community. (2023). PnPjs v3 documentation: Working with SharePoint data. PnP GitHub. Retrieved October 2023.
18	Seemann, M. (2021). Code That Fits in Your Head. Addison-Wesley Professional. ISBN 978-0137464401.
19	Microsoft Corporation. (2023). Playwright for .NET: E2E testing documentation. Microsoft. Retrieved November 2023.
20	Microsoft Corporation. (2023). Azure App Service: autoscaling and deployment slots. Microsoft Learn. Retrieved October 2023.
21	Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN 978-0201633610.
22	Newman, S. (2021). Building Microservices, 2nd edition. O'Reilly Media. ISBN 978-1492034025.
23	Microsoft Corporation. (2023). Microsoft 365 Developer Blog: SPFx 1.18 release notes. Microsoft Tech Community. November 2023.
24	k6 Labs. (2023). k6: Open source load testing tool documentation. k6.io. Retrieved November 2023.
25	Microsoft Corporation. (2023). Application Insights for ASP.NET Core applications. Microsoft Learn. Retrieved October 2023.
26	OpenTelemetry Authors. (2023). OpenTelemetry for .NET - Getting Started. opentelemetry.io. Retrieved November 2023.