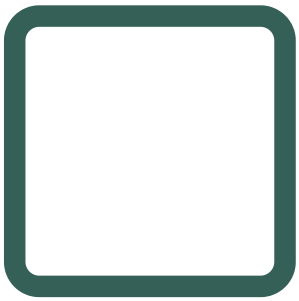
Quuipu

SERIE QUUIPU RESEARCH & ARCHITECTURE

WHITEPAPER 10

Web Protocol

Specifica per sviluppatori e integrazione



Whitepaper tecnico / specifica · Italiano

Quuipu Edizione 2 · v1

Identità della pubblicazione

Pubblicazione
Web Protocol

Serie
Serie Quuipu Research & Architecture

Numero di pubblicazione
Whitepaper 10

Edizione
Quuipu Edizione 2 · v1

Lingua
Italiano

Edizione storica
Cognisphere Edizione 1

Questa pubblicazione succede a Cognisphere Edizione 1. I PDF storici restano immutabili.

Indice

01. Why a protocol

02. Protocol architecture

03. Protocol stack

04. Resource model

05. API conventions

06. Context API

07. Context schemas

08. Experience API

09. Experience Plan schema

10. Semantic API

11. Host API

12. Source API

13. SDK design

14. Authentication

15. Authorization and permissions

16. Policy API

17. Events

18. Event catalog

19. Webhooks

20. Telemetry

21. Audit and decision trace

22. Errors

23. Rate limits and quotas

24. Versioning

25. Schema evolution

26. Pagination, filtering and search

27. Long-running operations

28. Web security

29. Privacy and data boundaries

30. Multi-tenant protocol

31. Web SDK integration pattern

32. Server integration pattern

33. Webhook integration pattern

34. Developer tooling

35. Testing and conformance

36. Developer experience

37. Governance of the protocol

38. Protocol invariants

39. Roadmap to a Developer Ecosystem

40. The protocol principle

A. Illustrative endpoint catalog

B. Canonical error model

C. Webhook verification sequence

D. Protocol invariants

Preamble

Specifica per sviluppatori e integrazione

> Questa pubblicazione succede a Cognisphere Edizione 1. Quuipu Edizione 2 preserva la tesi storica aggiornando brand e identità di pubblicazione attivi.

Preamble

Developer & Integration Specification

PROTOCOL MAP

A stable boundary for the Quuipu ecosystem Il protocollo separa integrazione, cognizione e rendering attraverso API e schema versionati.

Application / Website ↓

SDK or Direct Client ↓

Host API ↓

Context · Semantic · Experience APIs ↓

↓

Sources · State · AI/Model Adapters

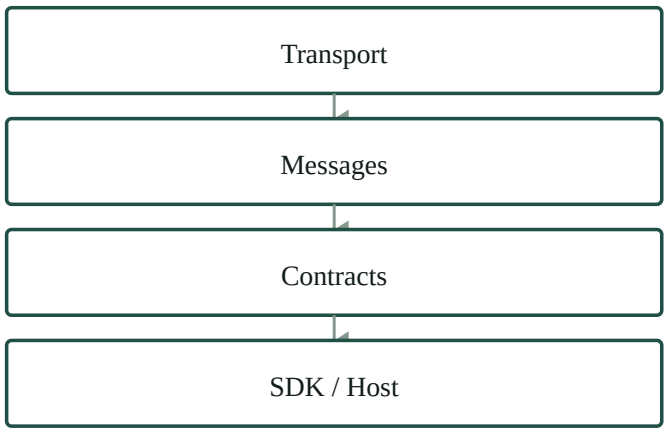


Figure 1 — Stack Quuipu Web Protocol.

API PLANES

Public protocol surfaces Plane Primary resources Purpose

Host Host, Capabilities, Source Bind runtime to a digital surface

Context ContextItem, Intent, Goal, Session Represent and transition bounded context

Semantic Concept, Transformation, SourceIdentity Preserve meaning and provenance

Experience ExperienceRun, ExperiencePlan, Component Compose adaptive presentation

Events Event, Subscription, Delivery Asynchronous integration

Governance Actor, Permission, Policy, Trace Control access and behavior

CAPITOLO 01

Why a protocol

Quuipu diventa un ecosistema sviluppatori solo quando le sue capacità possono essere integrate attraverso contratti stabili, versionati e governati. Il Web Protocol definisce quel confine senza trasformare

Specification thesis Il protocollo descrive oggetti, lifecycle, comandi, query, eventi e policy osservabili. L'implementazione del Core può evolvere purché mantenga i contratti pubblici.

Not an implementation leak Context resolution, semantic algorithms e internal reasoning non vengono esposti come dettagli da orchestrare lato client.

Host independence Un sito, portale, applicazione o SDK può consumare Quuipu attraverso lo stesso modello concettuale.

Draft status Questa V1 substantive edition è una specification architecture: definisce la forma normativa proposta da consolidare quando l'API reale sarà congelata.

CAPITOLO 02

Protocol architecture

Il protocollo organizza l'integrazione in piani separati: Host, Context, Semantic, Experience, Events e Governance.

Host plane Lifecycle, capabilities, tenant binding, source registration e surface integration.

Context plane Declared context, intent, goal, session, state projection, transitions e portability.

Semantic plane Concept resolution, source identity, transformations, provenance e validation.

Experience plane Experience run, modes, plans, components, feedback, fallback e resume.

Event plane Asynchronous lifecycle and domain events with stable envelopes.

Governance plane Authentication, authorization, permissions, policies, telemetry controls, versioning and audit.

Host Plane ↓

Context Plane ↓

Semantic Plane ↓

Experience Plane ↓

Event Plane ↓

Governance Plane

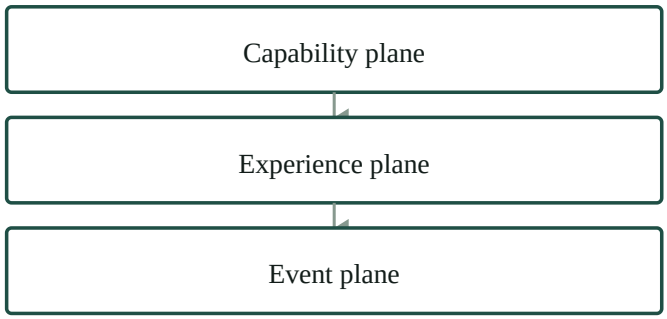


Figure 2 — Piani del protocollo.

CAPITOLO 03

Protocol stack

SDK e API sono due superfici dello stesso protocollo: l'SDK semplifica il consumo, ma non crea semantiche parallele.

Stack Application -> SDK/Client -> Host API -> Context/Semantic/Experience APIs -> Quuipu Runtime -> Source/Model/State adapters.

Thin SDK principle L'SDK gestisce serialization, retries, type safety, event subscriptions e ergonomic helpers; le decisioni cognitive restano server/runtime-side.

Direct API Gli integratori possono usare HTTP/event transports direttamente quando non vogliono un SDK.

Conformance SDK ufficiali e third-party devono superare gli stessi contract tests.

CAPITOLO 04

Resource model

Le API operano su risorse esplicite con identity, tenant scope, version e lifecycle.

Core resources Tenant, Host, Source, Session, ContextItem, Intent, Goal, ExperienceRun, ExperiencePlan, Component, SemanticConcept, Transformation, EventSubscription.

Identifiers Gli ID sono opachi e stabili; non codificano PII o business semantics.

References Le risorse si collegano tramite ref tipizzati invece di incorporare copie non governate.

Lifecycle Ogni risorsa dichiara stati validi e transizioni consentite.

Resource Scope Typical lifecycle

Session tenant/host created -> active -> suspended -> closed

ContextItem session/task/tenant declared/inferred -> valid -> expired

ExperienceRun tenant/host/session started -> composed -> presented -> completed

ExperiencePlan run valid -> superseded/expired

Transformation source/concept candidate -> validated/rejected -> invalidated

Subscription tenant active -> paused -> revoked

CAPITOLO 05

API conventions

Una convenzione uniforme riduce ambiguità e rende gli SDK generabili e testabili.

Base shape /v1/{tenantScope}/{resource} è una forma illustrativa; la URI definitiva dipenderà dal deployment.

Commands vs queries Mutazioni semantiche usano comandi espliciti; le query non producono side effect cognitivi.

Idempotency Comandi retryable accettano Idempotency-Key o equivalente.

Concurrency ETag/resourceVersion consente optimistic concurrency e impedisce lost updates.

Correlation requestId, runId e traceId collegano chiamate, eventi e osservabilità.

CAPITOLO 06

Context API

La Context API espone le operazioni del Context Model senza consentire mutazioni arbitrarie di un blob di sessione.

Commands declareContext, activateIntent, setGoal, transitionContext, expireContext, resetContext, importContextPackage.

Queries getResolvedContext, getActiveIntent, getGoal, getContextTrace, exportContextProjection.

Projection first I consumer ricevono projection autorizzate, non l'intero contesto disponibile.

Precedence Declared context non può essere sovrascritto silenziosamente da inferred context tramite API.

Operation class Examples Invariant

Declare declareContext Origin remains declared

Resolve getResolvedContext Returns authorized projection

Transition transitionContext Version/policy checked

Export exportContextProjection Minimized + audience-bound

Import importContextPackage Candidate until local resolution

Expire expireContext Derived dependencies invalidated

CAPITOLO 07

Context schemas

ContextItem e ContextPackage rendono context interoperabile fra SDK, host e runtime.

ContextItem

`{id, kind, value, origin, scope, confidence?, observedAt, effectiveUntil?, expiresAt?, purpose, sensitivity, version}.`

ContextPackage

`{schemaVersion, packageId, issuer, audience, tenantScope, projection, sourceRefs, issuedAt, expiresAt, policyRef, integrity}.`

Typed values kind determina lo schema del value; free-form metadata non sostituisce i campi normativi.

Validation Unknown fields seguono una forward-compatibility policy esplicita invece di essere interpretati liberamente.

CAPITOLO 08

Experience API

La Experience API gestisce l'intero lifecycle di un adaptive experience run.

Commands startExperience, submitInput, selectMode, chooseNode, submitFeedback, resetExperience, resumeExperience, closeExperience.

Queries getExperienceRun, getExperiencePlan, getAvailableModes, getFallbackState, getExperienceTrace.

Run scope Ogni run appartiene a tenant, host, session e source scope definiti.

Canonical access Anche quando l'enhancement fallisce, l'API deve poter restituire canonicalSourceRef/fallback.

startExperience ↓

submitInput / selectMode ↓

getExperiencePlan ↓

render ↓

submitFeedback ↓

context/state update ↓

complete / continue

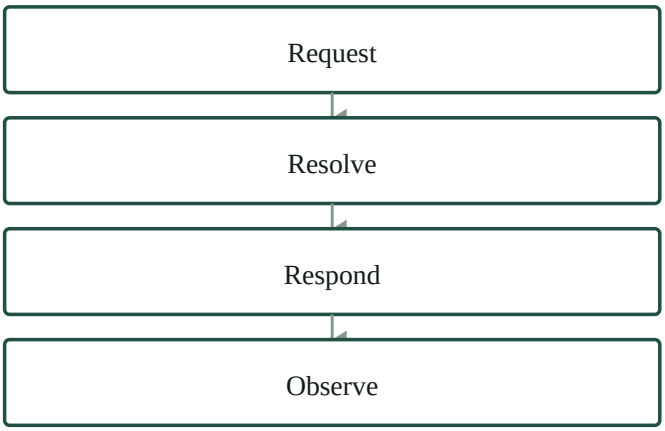


Figure 3 — Ciclo di vita Experience API.

CAPITOLO 09

Experience Plan schema

L'Experience Plan è il principale output pubblico del runtime e separa composizione cognitiva da rendering.

Plan

`{experienceId, runId, mode, resolvedContextRef, intentRef, goalRef?, sourceSet, semanticViewRef, components, provenance, trustState,`

fallbackRef, expiresAt}.

Components Ogni component ha type, purpose, contentRef/payload, sourceRefs, trustState, interactions, guards e fallback.

Graph components possono essere lineari o nodes di un Experience Graph con allowed transitions.

Surface neutrality Il protocollo non impone DOM, React, Astro o altro framework.

CAPITOLO 10

Semantic API

La Semantic API consente di risolvere concetti e validare trasformazioni senza esporre internals proprietari.

Queries resolveConcept, getConcept, getConceptRelations, getSourceIdentity, getTransformationRecord.

Commands validateTransformation, registerSourceMapping, invalidateSemanticArtifact.

Semantic identity Concept ID e source identity rimangono stabili attraverso rappresentazioni derivate.

Boundary La API restituisce esiti, evidence refs e reason codes, non chain-of-thought.

CAPITOLO 11

Host API

La Host API collega Quuipu alla superficie digitale che lo incorpora.

Lifecycle create/configure/initialize/start/ready/shutdown/dispose possono essere rappresentati come capability/lifecycle contract.

Capabilities Host dichiara rendering, navigation, locale, auth delegation, storage, source access e accessibility capabilities.

Host context Il runtime riceve HostCapabilities e HostSecurityContext, non accesso ambientale al browser/server.

Isolation Host adapter non può bypassare tenant, source o policy boundaries.

CAPITOLO 12

Source API

L'integrazione con CMS, knowledge base e altri sistemi avviene attraverso source contracts, non mediante ownership del contenuto.

Registration registerSource descrive authority class, connector type, schema, freshness, canonical URL/ref e tenant scope.

Read contract Il runtime può richiedere content units, metadata, versions e evidence refs attraverso adapter.

Write boundary

Invalidation Source update events invalidano semantic artifacts, caches ed Experience Plans dipendenti.

CAPITOLO 13

SDK design

Gli SDK devono rendere facile fare la cosa corretta senza nascondere le proprietà di trust del protocollo.

Core modules client, context, experience, semantic, events, auth, types e testing utilities.

Typed errors ContextExpiredError, PolicyDeniedError, VersionConflictError, UnsupportedModeError e RateLimitError sono preferibili a errori generici.

Async model Promise/future per commands e queries; async iterator/observable per event streams dove supportato.

No secret state L'SDK non mantiene un secondo Context Model proprietario sul client.

CAPITOLO 14

Authentication

Authentication stabilisce chi sta chiamando il protocollo; non decide da sola cosa il caller può fare.

Actors Service application, end-user delegated session, administrator, connector, webhook receiver.

Credentials OAuth/OIDC, signed service credentials o deployment-specific mechanisms possono implementare il contract.

Token minimization Token dovrebbero contenere identity/scopes minimi e riferimenti, non context payload sensibili.

Rotation Credential rotation e revocation devono essere supportate senza invalidare semantic identities.

CAPITOLO 15

Authorization and permissions

Authorization combina actor, tenant, resource, action, scope e policy.

Permission tuple Permit(actor, action, resource, tenant, purpose, contextClass) -> allow/deny/conditional.

Scopes context:read, context:declare, experience:run, semantic:read, source:register, events:subscribe sono esempi illustrativi.

Fine-grained control Un'app può avviare experience senza poter esportare context o leggere audit trace.

Deny by default Cross-tenant, admin, source-write e sensitive-context permissions non sono implicite.

Scope example Allows Does not imply

experience:run Start/continue runs Context export

context:declare Declare bounded context Read all context

semantic:read Resolve concepts Register sources

events:subscribe Receive permitted events Admin trace access

source:register Register source metadata Write canonical content

CAPITOLO 16

Policy API

Policy non è un header opzionale: è una dipendenza versionata delle decisioni del runtime.

Policy resources TenantPolicy, ExperiencePolicy, DataPolicy, AIUsePolicy, SourcePolicy e TelemetryPolicy.

Evaluation policy/evaluate può restituire allow, qualify, require_confirmation, require_review o deny con reason codes.

Version pinning ExperienceRun registra policyVersion per riproducibilità.

No client override Un client può chiedere una modalità, non disabilitare una policy server-side.

CAPITOLO 17

Events

Gli eventi disaccoppiano runtime, host, analytics consentiti e integrazioni downstream.

Envelope

`{eventId, type, schemaVersion, tenantId, hostId, sessionId?, runId?, timestamp, actorRef, payloadRef/payload, reasonCode?, traceId}.`

Families context.*, intent.*, semantic.*, source.*, experience.*, policy.*, webhook.*.

Delivery semantics At-least-once è compatibile con consumer idempotenti; ordering è garantito solo entro partition dichiarate.

Privacy Event payload segue minimization e non deve diventare una copia del context state.

Event field Role

eventId Idempotency/replay identity

type Namespaced event semantics

schemaVersion Payload compatibility

tenantId / hostId Isolation scope

sessionId / runId Optional lifecycle correlation

timestamp Event time

actorRef Originating actor

payloadRef/payload Minimized event data

reasonCode Operational explanation

traceId Cross-plane correlation

CAPITOLO 18

Event catalog

Un catalogo stabile permette a partner e strumenti di reagire senza polling.

Context context.declared, context.updated, context.expired, context.transferred.

Experience experience.started, experience.composed, experience.presented, experience.fallback, experience.completed.

Semantic semantic.resolved, transformation.validated, transformation.rejected.

Source source.updated, source.invalidated, source.unavailable.

Policy policy.qualified, policy.blocked, review.required.

CAPITOLO 19

Webhooks

I webhook portano eventi Quuipu verso sistemi esterni con un modello di sicurezza esplicito.

Subscription endpoint, event filters, tenant scope, secret/public key, status e delivery policy.

Signing Ogni delivery include timestamp, event ID e signature verificabile.

Replay protection Receiver verifica freshness, signature e eventId già processati.

Retries Backoff, dead-letter state e delivery attempt metadata devono essere osservabili.

No inbound command by default Un webhook è delivery outbound; comandi inbound richiedono API autenticata separata.

Runtime Event ↓

Filter ↓

Signed Webhook Delivery ↓

Receiver Verification ↓

Idempotent Processing ↓

ACK / Retry / Dead Letter

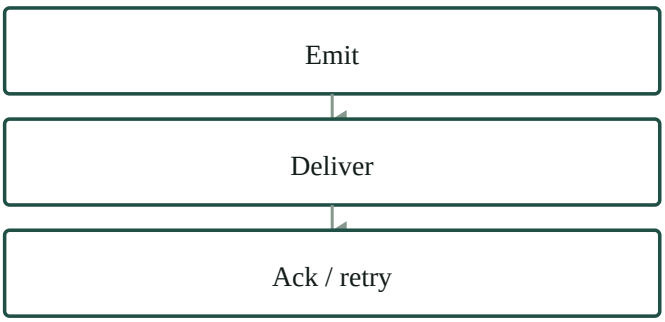


Figure 4 — Modello di consegna webhook.

CAPITOLO 20

Telemetry

Telemetry misura salute e qualità del protocollo senza trasformarsi in sorveglianza dell'utente.

Planes Operational, reliability, quality, security e cost telemetry.

Examples Latency per endpoint/stage, error rate, fallback rate, schema rejection, webhook delivery, AI invocation cost.

Excluded by default Raw prompts, full context, sensitive values e full content bodies non sono metriche.

Tenant controls TelemetryPolicy può disabilitare classi, ridurre retention o imporre regional processing.

CAPITOLO 21

Audit and decision trace

Auditability richiede distinguere access log, protocol trace e cognitive decision trace.

Access log Chi ha chiamato quale endpoint e con quale esito.

Protocol trace Request/event correlation, versions, retries e dependency status.

Decision trace Source refs, policy decisions, reason codes, transformations e fallback usati dal runtime.

No hidden reasoning Nessun trace deve esporre chain-of-thought del modello; registra decisioni verificabili.

CAPITOLO 22

Errors

Gli errori sono parte del protocollo e devono indicare se il caller può retry, correggere o fare fallback.

Error envelope

`{code, message, category, retryable, requestId, traceId, details?, remediation?}.`

Categories AUTH, PERMISSION, VALIDATION, CONFLICT, EXPIRED, POLICY, SOURCE, SEMANTIC, EXPERIENCE, RATE_LIMIT, INTERNAL_SAFE.

Safe details details non includono segreti, raw model prompts o dati di altri tenant.

Fallback hint Per errori cognitivi il server può fornire canonicalSourceRef o safeMode.

CAPITOLO 23

Rate limits and quotas

Limiti proteggono disponibilità, costi e fairness senza cambiare la semantica del protocollo.

Dimensions Requests, concurrent runs, event throughput, AI-assisted operations, source sync e webhook deliveries.

Headers/metadata Remaining, reset e retry-after possono essere esposti in modo standard.

Tenant quotas Quota può differire per plan ma non deve indebolire trust invariants.

Graceful degradation Superare una quota AI può degradare verso deterministic/canonical path quando possibile.

CAPITOLO 24

Versioning

Versioning deve permettere evoluzione pluriennale senza costringere tutti gli integratori a migrazioni simultanee.

Layers Protocol version, resource schemaVersion, event schemaVersion, policyVersion e SDK version sono distinti.

Compatibility Additive optional fields sono preferiti; semantic breaking changes richiedono major version.

Deprecation Announce -> dual support -> migration tooling -> sunset con finestre documentate.

Pinning Critical integrations possono pin versioni compatibili per run o client.

Version layer Example Breaking-change rule

Protocol v1 Major for semantic break

Resource schema schemaVersion=3 Migration/compatibility policy

Event schema event schemaVersion Consumer compatibility

Policy policyVersion Pinned to decision/run

SDK sdk 1.x May advance independently within protocol compatibility

CAPITOLO 25

Schema evolution

Gli schema sono contratti pubblici e devono avere regole di compatibilità machine-testable.

Canonical schemas JSON Schema/OpenAPI/AsyncAPI o equivalenti possono rappresentare REST/event contracts.

Unknown fields Consumer robusti ignorano campi additive non critici, ma non valori enum sconosciuti quando materialmente rilevanti.

Migration Transformers versionati convertono payload persistiti o portable packages.

Examples as tests Gli esempi pubblicati devono essere validati in CI contro lo schema reale.

CAPITOLO 26

Pagination, filtering and search

Le collection API devono essere prevedibili e bounded.

Pagination Cursor-based pagination evita instabilità con dataset dinamici.

Filtering Filtri supportati sono espliciti; non esiste query arbitraria sullo state interno.

Search Semantic search può essere capability separata con source scope e policy.

Limits Server applica max page size e complexity budget.

CAPITOLO 27

Long-running operations

Source ingestion, large semantic validation o export possono richiedere operazioni asincrone.

Operation resource

`{operationId, type, status, progress?, resultRef?, error?, createdAt, expiresAt}.`

States queued, running, succeeded, failed, cancelled.

Polling/events Client può interrogare operation o sottoscrivere operation.* events.

Cancellation Cancel è best-effort e non annulla side effects già committed.

CAPITOLO 28

Web security

Il protocollo deve essere integrabile sul Web senza affidarsi a privilegi ambientali.

Browser clients CORS, CSRF protections, PKCE/delegated tokens e origin restrictions secondo deployment.

Secrets Service secrets non devono essere esposti nel browser.

Content security Rendered components rispettano CSP e safe-link policy dell'host.

Input security Prompt injection e malicious source content sono trattati come untrusted input, non istruzioni di sistema.

CAPITOLO 29

Privacy and data boundaries

Data minimization è incorporata negli schema e nei permission checks.

Classification public, tenant-confidential, user-contextual, sensitive-restricted sono classi illustrative.

Residency Deployment può applicare region/processor constraints tramite policy.

Retention Event, trace, context e telemetry hanno retention class separate.

Deletion Delete/revoke produce invalidation delle future projections e dei derived artifacts dove richiesto.

CAPITOLO 30

Multi-tenant protocol

Tenant scope è presente prima di ogni risoluzione, query, evento o delivery.

Namespace Resource IDs sono risolti nel tenant namespace.

Cross-tenant Nessuna API cross-tenant generica; exchange passa da export/import o gateway autorizzato.

Admin plane Operator administration è separata dalle application APIs.

Testing Conformance suite include negative tests per tenant leakage.

CAPITOLO 31

Web SDK integration pattern

Un'integrazione Web tipica resta progressiva: il sito canonico funziona prima, Quuipu si attiva come enhancement.

Bootstrap Host configura tenant, locale, source scope e capabilities.

Start SDK apre session/run e dichiara context minimo.

Render Experience Plan viene mappato su componenti dell'host.

Events UI actions producono protocol commands; non mutano direttamente il runtime state.

Fallback Se SDK/API non sono disponibili, il sito continua a servire contenuto canonico.

Canonical Website ↓

Host Bootstrap ↓

Context API ↓

Experience API ↓

Experience Plan ↓

Host Components ↓

Protocol Events ↓

Canonical Fallback

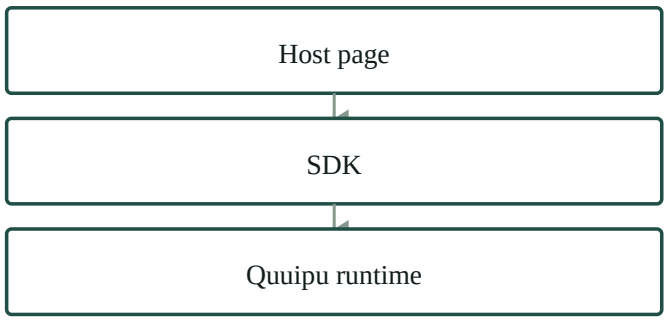


Figure 5 — Integrazione progressiva Web SDK.

CAPITOLO 32

Server integration pattern

Le integrazioni server-side possono orchestrare source access, authorization e rendering senza esporre credenziali al browser.

BFF Backend-for-Frontend può scambiare token e applicare tenant policy.

SSR Experience Plan può essere ottenuto server-side per rendering iniziale, con hydration opzionale.

Connector Source adapters possono vivere nel trust domain del cliente.

Isolation Il server host non riceve internals Quuipu oltre i contratti pubblici.

CAPITOLO 33

Webhook integration pattern

Sistemi enterprise possono reagire a eventi senza entrare nel runtime path sincrono.

Example review.required -> workflow system -> human approval -> authenticated API command -> experience artifact released.

Loose coupling Il consumer può essere indisponibile senza bloccare la presentazione, salvo policy che richiede approvazione.

Replay Event ID e delivery log permettono recupero deterministico.

Boundary Webhook non è una backdoor per modificare context o policy.

CAPITOLO 34

Developer tooling

Un ecosistema sano richiede strumenti che trasformino la specification in feedback immediato per gli integratori.

CLI Validate config, schemas, permissions, webhook signatures e conformance fixtures.

Sandbox Tenant isolato con synthetic sources e deterministic fixtures.

Inspector Visualizza Experience Plan, source refs, reason codes, policy outcomes ed event timeline.

Code generation Typed clients possono essere generati dai canonical schemas senza perdere semantic helpers.

Tool Developer value

CLI Validate config, schemas, permissions, signatures

Sandbox Safe synthetic integration environment

Inspector See plans, sources, policy and event timeline

Schema registry Canonical machine-readable contracts

Conformance runner Prove compatibility against a protocol version

CAPITOLO 35

Testing and conformance

La compatibilità non si dichiara: si prova con una suite pubblica di contract e invariant tests.

Contract tests Request/response/event schemas, auth failures, idempotency e version conflicts.

Invariant tests No expired context, no unauthorized source, provenance requirements, tenant isolation, safe fallback.

Golden fixtures Known context + sources + policy -> expected plan/event sequence.

Certification Un SDK/connector può dichiararsi Quuipu-compatible solo contro una versione di conformance.

CAPITOLO 36

Developer experience

La DX deve rendere visibili i concetti Quuipu invece di nasconderli dietro una singola funzione magica.

Progressive path Start with canonical host -> add Context API -> Experience API -> events -> advanced semantic/source integrations.

Local clarity Errors e reason codes devono indicare il boundary violato.

Docs Reference, guides, recipes e conceptual architecture sono separati ma cross-linked.

Examples Esempi mostrano sempre auth, tenant scope, fallback e cleanup, non solo happy path.

CAPITOLO 37

Governance of the protocol

Un protocollo destinato a partner richiede una governance di evoluzione esplicita.

Change process Proposal -> review -> experimental -> stable -> deprecated.

Normative language MUST/SHOULD/MAY viene usato nella futura specification normativa.

Extension registry Nuovi event types, component types e capabilities usano namespace e registration rules.

Security process Vulnerability handling e security advisories sono parte del developer ecosystem.

CAPITOLO 38

Protocol invariants

Gli invarianti definiscono ciò che nessun SDK, host o versione può aggirare.

P-01 Tenant scope precedes every resource operation.

P-02 Declared context cannot be silently overwritten by inference.

P-03 Experience enhancement never removes canonical fallback.

P-04 Derived representations preserve required provenance.

P-05 Permissions are deny-by-default for sensitive and cross-tenant actions.

P-06 AI/provider state is not canonical Quuipu state.

CAPITOLO 39

Roadmap to a Developer Ecosystem

Il Web Protocol è il punto di partenza per trasformare Quuipu da prodotto integrabile a piattaforma estendibile.

Phase 1 Canonical APIs + official Web SDK + schemas + sandbox.

Phase 2 Events/webhooks + connector SDK + conformance suite.

Phase 3 Component/experience extension registry + partner certification.

Phase 4 Multi-language SDKs, marketplace/registry governance e enterprise developer tooling.

Constraint Ecosystem growth non deve indebolire semantic integrity, context boundaries o human control.

CAPITOLO 40

The protocol principle

Il protocollo deve rendere Quuipu componibile senza rendere il suo comportamento cognitivo arbitrario.

Stable contracts Developers integrate against resources and semantics, not implementation internals.

Bounded power Permissions, policy and schemas constrain every extension.

Observable evolution Versions, events and traces make change inspectable.

Interoperability Context, semantics and experience can cross surfaces through explicit contracts.

Final thesis A developer ecosystem becomes trustworthy when integration is open at the boundary and governed at the core.

CAPITOLO A

Illustrative endpoint catalog

The paths below are architectural examples, not a claim that these endpoints are already frozen or deployed.

Method Illustrative path Purpose

`POST /v1/experiences Start ExperienceRun`

`POST /v1/experiences/{id}/inputs Submit bounded input`

`GET /v1/experiences/{id}/plan Read current ExperiencePlan`

`POST /v1/context/declarations Declare context`

`GET /v1/context/resolved Read authorized resolved projection`

`POST /v1/context/export Create portable projection`

`POST /v1/semantic/resolve Resolve concept/meaning`

`POST /v1/semantic/transformations/validate Validate derived representation`

`POST /v1/sources Register source contract`

`POST /v1/event-subscriptions Create event/webhook subscription`

CAPITOLO B

Canonical error model

Code family Retry? Typical remediation

AUTH_* Usually no Refresh/re-authenticate

PERMISSION_* No Request scope/change policy

VALIDATION_* No Correct request/schema

VERSION_CONFLICT Yes after read Reload resource/version

CONTEXT_EXPIRED Conditional Re-declare/re-resolve

POLICY_DENIED No Use permitted mode/canonical

SOURCE_UNAVAILABLE Yes/conditional Retry or canonical fallback

SEMANTIC_UNCERTAIN No blind retry Qualify/use source

RATE_LIMITED Yes Honor retry-after

INTERNAL_SAFE Conditional Retry/fallback with traceId

CAPITOLO C

Webhook verification sequence

Receive HTTPS request ↓

Read timestamp + eventId ↓

Verify signature ↓

Check timestamp tolerance ↓

Check eventId replay store ↓

Validate schemaVersion ↓

Process idempotently ↓

Return success

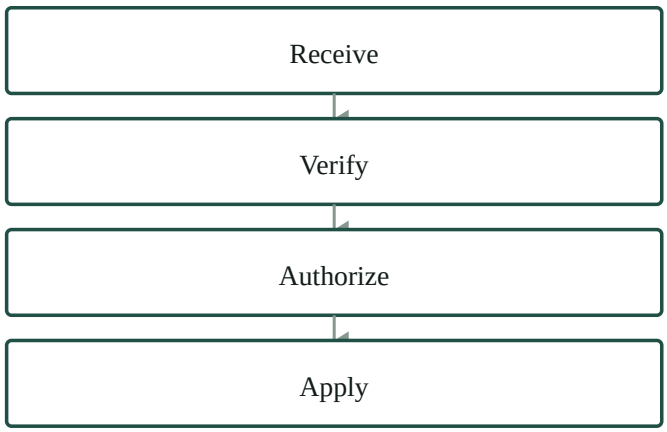


Figure 6 — Sequenza consigliata di verifica del receiver.

Webhook payloads SHOULD be treated as untrusted until signature, freshness and schema validation succeed.

CAPITOLO D

Protocol invariants

P-01 · Tenant first Tenant scope is established before resource resolution or retrieval.

P-02 · Context precedence Declared context is not silently overwritten by inference.

P-03 · Projection by purpose Consumers receive only context required and permitted for the task.

P-04 · Canonical fallback Cognitive enhancement cannot remove access to authoritative content.

P-05 · Provenance survives Required source and transformation references remain attached to derived experience.

P-06 · Policy is server-authoritative Clients cannot bypass policy through request parameters.

P-07 · Permissions default to deny Sensitive, administrative and cross-tenant actions require explicit grants.

P-08 · Events are minimized Event streams are not replicas of full context or content state.

P-09 · AI state is not protocol state Provider conversation memory is not canonical Quuipu state.

P-10 · Version is explicit Resources, events and policies declare compatibility-relevant versions.

P-11 · Webhooks are signed External event delivery is authenticated and replay-resistant.

P-12 · Conformance is testable Compatibility is tied to machine-verifiable contracts and invariants.

APPENDIX E

Conformance profile Profile Required capabilities

HOST-BASE Host lifecycle, tenant binding, canonical fallback

CONTEXT-BASE Declare, resolve, expire, projection, trace

EXPERIENCE-BASE Start run, submit input, plan, mode, feedback, fallback

SEMANTIC-BASE Concept/source identity, transformation validation

EVENTS-BASE Event envelope, idempotent consumer, version handling

WEBHOOK-BASE Signing, replay protection, retry handling

SECURITY-BASE AuthN, AuthZ, tenant isolation, secret handling

OBSERVABILITY-BASE requestId/runId/traceId, safe metrics and errors

APPENDIX F

Relationship to the Quuipu series Whitepaper Protocol relationship

01 · The Cognitive Web Defines the category and human-centered objective.

02 · Architecture Defines Core/Host boundaries exposed through contracts.

03 · Quuipu Web Provides the primary integration surface.

04 · Semantic Integrity Defines provenance and transformation invariants.

05 · Trust Architecture Defines policy, permissions, trace and bounded intelligence.

06 · Quuipu AI Defines AI Gateway as an optional provider boundary.

07 · Applications Supplies domain-specific integration profiles.

08 · Context Model Defines Context API semantics and portability.

09 · Experience Runtime Defines ExperienceRun and ExperiencePlan semantics.

10 · Web Protocol Turns the architecture into a developer-facing contract model.

APPENDIX G

Normative specification roadmap Artifact Future normative role

OpenAPI document HTTP request/response contracts

AsyncAPI/event schemas Events and webhook contracts

JSON Schemas Resource and portable payload validation

SDK reference Language-specific ergonomic binding

Conformance suite Machine-verifiable compatibility

Security profile Auth, signing, token and browser requirements

Extension registry Namespaced components, events and capabilities

Change policy Lifecycle from experimental to deprecated

Questa edizione non dichiara che l'API sia già congelata: prepara la struttura che potrà diventare normativa quando gli endpoint, gli schema e i transport reali saranno implementati e certificati.

Developer & Integration Specification

OPEN AT THE BOUNDARY. GOVERNED AT THE CORE. VERSIONED BY CONTRACT.