Qnnex Project – Technical Documentation

Comprehensive System Design, Protocol Specifications, and Integration Overview

Date: August 2025

1. BECKN Protocol Overview

The BECKN protocol enables standardized communication between buyer and seller applications. It relies on a structured message model comprising a `context` and `message` block. This ensures interoperability across platforms, enabling consistent and scalable API interactions for tasks like search, order, and status updates.

2. Sample Communication Flows

The transaction lifecycle starts with a search from the Buyer App, routed through the Protocol Layer to relevant Seller Apps. Responses are returned and shown to the user for selection and order placement. The seller confirms the order, and status updates are managed asynchronously.

3. Endpoint Responsibilities

* Buyer App Platform (BAP): Sends search, select, order, and status requests. It also handles callbacks for search results and order confirmations.

* Business Platform Provider (BPP): Responds to incoming protocol requests, including catalog data, order acceptance, and status updates.

* Gateway: A middleware routing layer ensuring secure, structured communication between all participating systems.

4. Integration Best Practices

- • Always use consistent `context` and `message` formats.
- • JSON payloads must include correct content types.
- • Use unique identifiers for every transaction.
- • Handle all `/on_*` callbacks asynchronously.
- • Validate payload structure before sending/receiving.

- • Log all request/response cycles for traceability.

5. System Requirements Overview

Registration Requirements

- • Email verification is mandatory for all users.
- • Optional mobile verification for buyers; recommended for sellers.
- • Strong password policies (length, complexity) are encouraged.
- • User interface adapts dynamically based on role.
- • Prevent duplicate registrations via email/phone checks.
- • Sessions should be tracked and managed securely.

Order Requirements

- • Buyers can add items to a cart and proceed through a checkout flow.
- • A short window for editing/canceling orders (e.g., 5 minutes) is suggested.
- • Real-time inventory checks are necessary to avoid overselling.
- • Partial fulfillment logic is proposed to manage limited inventory.
- • Cancellation reasons can be captured for better analytics.

System Features

- • Live order tracking using shipping APIs.
- • Webhooks for real-time payment success/failure updates.
- • Notifications (email, push) for all critical events.
- • Seller dashboards with key business metrics.
- • Admin functionality to manage users, content, and disputes.
- • All data must be secured via HTTPS and encryption protocols.

6. Protocol Team – Technical Stack and Workflow

Technology Stack

- • Node.js + Express: Backend APIs with scalability via clustering.
- • Redis: Fast, in-memory cache for session/state data.
- • MongoDB: NoSQL store for catalog and order records.
- • PostgreSQL: Reliable relational store for logging and transactions.
- • Kafka / NATS: High-speed messaging for async communication.
- • Axios: Lightweight HTTP client for inter-service calls.
- • Security: HMAC for internal integrity; RSA for public trust.
- • Beckn JSON Schema: Ensures standard protocol messaging.

Workflow Architecture

- • Buyer initiates a search.

- • Protocol forwards to sellers.
- • Sellers return their catalog.
- • Buyer selects and places an order.
- • Protocol forwards the order and awaits seller confirmation.
- • All further updates (status, confirmation) are handled asynchronously.

Integration Requirements

- • Buyer App Team: Must implement search, select, order, and status APIs using Beckn-compatible formats.
- • Seller App Team: Must handle order-related callbacks, real-time availability, and confirmation logic.

7. Free vs Paid Technology Options

Technology	Free Use Case	Paid/Enterprise Use Case
Node.js	Open source; scalable apps	Cloud hosting costs (AWS, Heroku, etc.)
Redis	High speed, single-node use	Redis Enterprise: HA, backups, multi-region
MongoDB	Manual sharding in Community Edition	Atlas: auto-scaling, global clusters
PostgreSQL	Full DB engine (self-hosted)	Managed services: backups, replication
Kafka	OSS version, manual operations	Confluent Cloud: schemas, security, auto-scaling
NATS	Lightweight core messaging	JetStream: monitoring, persistence
Axios	Free and fully open source	N/A
Beckn Protocol	Free, open JSON-based standard	N/A

8. System Architecture Overview

The architecture ensures smooth and scalable interactions between Buyer Apps, the central Protocol Layer, and Seller Apps. All communication is standardized, secure, and handled asynchronously to support high-throughput operations.

1. BECKN Protocol Schemas (Simplified Example)

Context Object (required in every request/response):

```
{
 "domain": "nic2004:60232",
 "country": "PKR",
 "city": "std:080",
 "core_version": "0.9.3",
 "transaction_id": "1234567890",
 "message_id": "abcdef123456",
 "timestamp": "2025-08-06T12:00:00Z",
 "bap_id": "buyer-app.qnnex.com",
 "bpp_id": "seller-app.qnnex.com"
}
```

Message Object (example for /search):

```
{
 "intent": {
  "item": {
   "descriptor": { "name": "apples" }
  },
  "fulfillment": {
   "end": { "location": { "gps": "12.9716,77.5946" } }
  }
 }
}
```

2. Sample Payloads

A. /search (Buyer → BAP)

```
{
 "context": { /* see above */ },
 "message": {
  "intent": {
   "item": { "descriptor": { "name": "apples" } }
  }
 }
}
```

B. /on_search (BBP → Gateway/BAP)

```
{
 "context": { /* see above, with bpp_id set */ },
 "message": {
  "catalog": [
   {
    "id": "item-1",
    "descriptor": { "name": "apples" },
    "price": { "currency": "INR", "value": "100" },
    "available": true
   }
  ]
 }
}
```

C. /order (Buyer → BAP)

```
{
 "context": { /* see above */ },
 "message": {
  "order": {
   "item_id": "item-1",
   "quantity": 2,
   "fulfillment": { "end": { "location": { "gps": "12.9716,77.5946" } } }
  }
 }
}
```

D. /on_order (BBP → Gateway/BAP)

```
{
 "context": { /* see above, with bpp_id set */ },
 "message": {
  "order": {
   "id": "order-123",
   "status": "accepted",
   "items": [{ "id": "item-1", "quantity": 2 }]
  }
 }
}
```

3. Endpoint Documentation (QNNEX Prototype)

Buyer App Platform (BAP) Endpoints

Endpoint	Method	Description
/search	POST	Search for items

/select	POST	Select item(s)
/order	POST	Place an order
/status	POST	Get order status
/confirm	POST	Confirm an order
/cancel	POST	Cancel an order
/on_search	POST	Callback for search result
/on_select	POST	Callback for select
/on_order	POST	Callback for order
/on_status	POST	Callback for status

Business Platform Provider (BBP) Endpoints

Endpoint	Method	Description
/on_search	POST	Respond to search
/on_select	POST	Respond to select
/on_order	POST	Respond to order
/on_status	POST	Respond to status
/on_confirm	POST	Respond to confirm
/on_cancel	POST	Respond to cancel

Gateway Endpoints

Endpoint	Method	Description
/search	POST	Route search to BBP
/select	POST	Route select to BBP
/order	POST	Route order to BBP
/status	POST	Route status to BBP
/confirm	POST	Route confirm to BBP
/cancel	POST	Route cancel to BBP

/on_search	POST	Route on_search to BAP
/on_select	POST	Route on_select to BAP
/on_order	POST	Route on_order to BAP

4. Integration Checklist for Teams

- • Use the context and message structure in every request.
- • Always POST JSON with Content-Type: application/json.
- • Use unique transaction_id and message_id for each transaction.
- • Handle all callbacks (/on_*) as asynchronous responses.
- • Validate payloads against the above schemas.
- • Log all requests and responses for debugging.

5. References

• BECKN Protocol Spec

• QNNEX Prototype README