Author: Priyankesh | priyankeshom@gmail.com
https://github.com/priyankeshh
https://www.linkedin.com/in/priyankesh/

No-Code AI-Enabled Catalog Generator for Beckn Commerce

Project Requirements for No-Code AI-Enabled Catalog Generator for Beckn-Compliant Commerce · Issue #3801 · beckn/beckn-ui-workspace

Project Vision and Goals

Beckn is a decentralized open e-commerce protocol that enables local businesses to publish and transact products/services across independent applications. This project's vision is to empower non-technical sellers (e.g. neighborhood shops) to easily create Beckn-compliant product or service catalogs by conversing in natural language. Using Google's Gemini API for AI-driven conversation and generation, the tool will translate a seller's descriptions into a fully-formed Beckn **Catalog** JSON. By simplifying catalog creation, we aim to accelerate Beckn adoption in accessible domains (like retail and mobility) and demonstrate Beckn's capabilities in demos and pilot programs. The goals include:

- **Ease of Use:** Enable sellers to build and edit catalogs without coding knowledge, via a chat-like interface or guided forms.
- **Standards Compliance:** Produce catalogs that conform to the Beckn core schema (domain-agnostic specification), and relevant extensions (e.g. retail or mobility).
- **Rapid Generation:** Leverage Gemini's NLP to interpret user input into structured catalog entries (items, categories, descriptors, etc.) with minimal manual effort.
- **Validation & Export:** Validate generated catalogs against Beckn JSON schemas and allow export in standard Beckn format for use in buyer/seller apps or gateways.

The project scope will cover retail and mobility use-cases (e.g. stores listing products, taxi operators listing ride options) and omit highly-regulated domains (like healthcare). We will focus on discovery-stage use (search-and-browse) rather than checkout/payment. A pilot implementation might integrate with an example Beckn Provider Platform (BPP) sandbox or registry to showcase end-to-end flow.

User Personas

- **Non-Technical Seller (Merchant):** A local store owner or small service provider with products/services to offer. This user has limited IT skills and needs an intuitive way to publish their offerings on Beckn. They may answer prompts like “describe your business” or “list your items” in plain language. Their goal is to create or update a product catalog without writing JSON or code.

- **Solution Demonstrator (BAP Developer/BP Network Integrator):** A technical user who builds or demos Beckn applications for clients or stakeholders. This persona will use the tool to generate example catalogs for pilot projects or presentations quickly. They care about accuracy to the Beckn spec and might refine prompts to produce specific scenarios.

Each persona benefits from the same core features: a guided conversational UI, AI-assisted completion of catalog fields, and schema validation to ensure Beckn compliance.

Functional Requirements

- **Conversational UI:** A chat-like interface (or a question-driven form) where users can describe their business and products. For example, the tool may ask, “What category of products do you sell?”, “Describe an item (name, price, quantity, etc.)”, or allow freeform prompts like “Add 5 items in *Fruits* category with prices”. The Gemini API processes these inputs to suggest or generate catalog entries.

- **Domain Selection & Templates:** Allow selection of domain (e.g. retail, mobility) so that prompts and default fields match expected categories (e.g. grocery items vs. ride fares). Provide sample templates (e.g. “Bookstore Catalog” or “Taxi Fleet”) to help demo users start.

- **AI-Driven Generation:** Use Gemini to interpret user prompts and generate structured catalog JSON that follows the Beckn **Catalog** schema. For instance, on a “search” intent, the BPP will send a Catalog object as the response. The AI should populate `descriptor`, `categories`, `items`, `fulfillments`, `payments`, etc., according to Beckn’s Catalog definition

- **Schema Validation:** After generation, validate the catalog JSON against the official Beckn schemas. Highlight missing or invalid fields. (For example, every item should include fields like name, price, quantity as per Beckn’s **Item** schema, and categories should have an `id` and `descriptor`.) Use existing Beckn schema definitions (from the protocol-specifications repo) or a library (e.g. AJV for JSON Schema) to enforce correctness. Display validation feedback so the user can fix any issues.

- **Interactive Editing:** Allow the user to iteratively refine the catalog. For example, after initial generation, the user might say “Change the price of Banana to 10” or “Add a new category *Beverages* with two items.” The system re-runs the AI or updates the JSON accordingly. Maintain conversation context so the AI can continue building the

same catalog.

- **Visualization & Preview:** Render a human-readable preview of the catalog (e.g. a list of categories and items with details) alongside the raw JSON. This helps users verify that the generated catalog matches their intent. The preview should update as the user refines inputs.

- **Export Options:** Provide ways to export or share the final catalog JSON. Options might include downloading a `.json` file, copying JSON to clipboard, or sending it to a Beckn Provider Platform (BPP) endpoint via API (for advanced users). Ensure the exported JSON conforms exactly to the Beckn Catalog schema so it can be used by any Beckn-compliant BAP/BPP implementation.

Technical Architecture

We propose a modern web stack with a JavaScript front-end and a lightweight backend/API layer:

- **Front-End (UI):** Next.js (React) – Next.js offers a reactive UI with SEO-friendly server-side rendering for documentation pages, and built-in API routes for backend functionality. It allows seamless integration of React components for chat and form interfaces. (A pure React app is also viable, but Next.js simplifies routing and data fetching.) The UI will include a chat component (for conversation), forms for any structured input, and a JSON preview pane.

- **Back-End (API):** Node.js server (could be built into Next.js API routes) – This layer will handle calls to the Gemini API. When the user submits a prompt or request, the front end sends it to our backend, which relays it to Gemini and returns the generated text/JSON. Using Next.js API routes or an Express server, we keep all logic in JavaScript for simplicity. We will secure the Gemini API key on the backend.

- **Gemini Integration:** The backend will orchestrate prompts and calls to Google Gemini (via REST API or gRPC). We will define structured system prompts to guide Gemini to output valid JSON following the Beckn schema. For example, we might instruct: “Generate a Beckn Catalog JSON for a grocery store with these items…”. The response (text or code) will be parsed into JSON. We will use Gemini’s advanced capabilities (chain-of-thought, JSON parsing) to ensure high-quality output.

- **Data Layer:** If persistence is needed (to save user sessions or catalogs), we can use a NoSQL database (e.g. MongoDB or Firebase) or cloud storage. For a minimal viable product, catalogs could be transient (in-memory or in-browser) with optional download.

- **Schema Validation:** Use a JSON Schema validator library (e.g. AJV) loaded with Beckn’s official schemas (from https://.com/beckn/protocol-specifications/schema). Whenever a catalog JSON is produced or edited, validate it against `Catalog.yaml`,

`Category.yaml`, `Item.yaml`, etc. This ensures compliance with the standard Beckn API (similar to how the Beckn protocol server validates messages).

Overall, the architecture flow is: **User (browser)** → Frontend UI → Backend API (Node/Next) → Gemini API → Backend processing → Frontend JSON/preview. All Beckn schema rules run on the backend (or a shared library) to keep output consistent and secure.

UI/UX Flows and User Journey

1. **Landing & Domain Selection:** The user opens the app and chooses a domain (e.g. “Retail” or “Mobility”). The app briefly explains that Beckn catalogs list products or services offered by a seller.

2. **Introduction Prompt:** The interface (possibly a chatbot bubble or guided form) asks: “Tell me about your business.” The seller might type: “I run a fruit and vegetable store in Delhi.” The system uses this to set context.

3. **Category Generation:** The AI suggests categories based on the business. E.g., “I suggest categories: *Fruits*, *Vegetables*, *Dairy Products*. Would you like to use these?” The user can confirm or edit. (We rely on Beckn’s Category schema – categories group related items.)

4. **Item Addition:** The assistant then guides the user to add items under each category. The user might say: “Under Fruits, add Apple, Banana, Orange.” The AI replies by asking details: “Please specify price, unit (kg/each), and description for each item.” The user answers (or uses quick replies), and the AI constructs item entries. For example, “Apple” becomes an Item with name “Apple”, descriptor “Fresh red apple”, price 120 (per dozen), etc. Internally, this populates an `Item` JSON object (which Beckn defines as a product/service; the schema notes it can represent a grocery item).

5. **Preview and Validation:** As items are added, the app updates a preview list and the underlying JSON. If any required field is missing, a validation message appears (“Price is required for all items”). The user corrects input as needed.

6. **Iterative Refinement:** The user can make higher-level edits by chatting. For example: “Change all prices in Fruits by +10%.” Gemini generates updated values and the catalog updates. Or “Remove category Dairy Products.” The system applies the change. Each time, the JSON is re-validated.

7. **Finalization & Export:** Once the catalog looks correct, the user clicks “Export”. Options may include downloading the JSON or sending it to a Beckn demo BPP. The JSON will match the Beckn format (with `context`, `message/catalog`, etc.) so that, for instance, a BAP could send a `search` intent and a BPP could send this catalog in

an `on_search` response.

Throughout this journey, the UI remains conversational and user-friendly. Key Beckn concepts (Context, Descriptor, Fulfillment modes, etc.) can be abstracted or hidden from the user. For example, default fulfillment modes or payment options can be auto-filled based on domain (e.g. “Home delivery available” for retail). The user primarily deals with familiar terms like items, prices, quantities, while the AI ensures the final output respects the Beckn Catalog schema.

Gemini API Integration Plan

To integrate Google Gemini for natural language understanding and generation:

- **Prompt Engineering:** Develop system prompts that define the format (Beckn Catalog JSON). For example, “You are an assistant that creates JSON catalogs in the Beckn protocol format. Each response should be valid JSON following this schema: { … }.” We will feed Gemini the seller’s latest input (e.g. “Add [Item Name] with [details]”) plus the current catalog JSON so it can update or append entries.

- **Parsing & Safety:** Gemini may output JSON or near-JSON text. We will implement parsing logic to handle minor formatting issues and ensure the result is parseable. We’ll include sanity checks (e.g. using regex or a JSON linter) before accepting Gemini’s output.

- **Conversation Context:** Maintain a structured context with previous user inputs, system prompts, and the evolving catalog state. This helps Gemini produce consistent updates rather than treating each prompt independently. E.g., the system prompt will say “We have already added these categories/items…” and list the partial catalog, so Gemini knows what’s already included.

- **Fallback Handling:** In case Gemini’s response is invalid or incomplete, the UI will notify the user and possibly ask for clarification. For example, if a user request is unclear, the system can respond: “I didn’t understand. Could you please rephrase?”

- **Latency & Performance:** Gemini API calls will be asynchronous. The UI will show a loading indicator (“Generating catalog…”). We expect responses in a few seconds. If needed, we can batch multiple operations (e.g. adding several items at once) into one prompt to minimize round trips.

- **Cost Management:** Since Gemini API usage may incur cost, we will design prompts to be efficient (use few-shot examples rather than long context) and cache common operations if applicable.

This integration plan leverages Gemini's strength in understanding free text and generating structured output. It also incorporates Beckn's schema knowledge at the system prompt level, so that the generated catalog is immediately close to final form.

Timeline and Milestones

1. **Weeks 1-2 – Requirements & Design:** Finalize requirements (this PRD), gather Beckn schema files for retail/mobility catalogs, and design UI mockups. Define data models (Catalog, Category, Item) based on the schemas.

2. **Weeks 3-5 – Front-End Prototype:** Develop the initial React/Next.js UI components for conversation and preview. Implement a basic form/chat interface.

3. **Weeks 6-8 – Gemini Integration:** Obtain access to the Gemini API. Build backend API routes to send prompts and receive responses. Prototype simple interactions (e.g. "Add an item" → update JSON).

4. **Weeks 9-10 – Schema Validation & Editing:** Integrate JSON schema validation using Beckn's official definitions. Implement validation error display. Add UI for manual editing of fields and re-validation.

5. **Weeks 11-12 – Export and Packaging:** Enable exporting the final catalog (download or API call). Polish UI/UX. Test with example scenarios (e.g. generating a grocery catalog or a taxi service catalog) to ensure correctness.

6. **Weeks 13-14 – Testing & Refinement:** Conduct user testing with sample non-technical users or internal team. Fix usability issues, handle edge cases, and ensure stability. Prepare documentation and a demo walkthrough.

7. **Week 15 – Deployment & Demo:** Deploy the app (e.g. on Vercel or a cloud platform) and demonstrate to Beckn mentors. Prepare a recorded demo or live session to show end-to-end Beckn flow using the generated catalog.