MBSPro System Implementation – Technical Overview

Deployed on Cloud (render):

https://mbspro-web.onrender.com

Introduction and Case Analysis

Medicare billing in general practice is notoriously complex[1]. A GP might, for example, attempt to bill a chronic disease management plan together with a standard consultation for the same patient on the same day – only to encounter a mutual exclusivity error. This is because Medicare's co-claiming rules forbid charging certain chronic care items alongside general attendance items in a single day[2]. Such scenarios are common: the Medicare Benefits Schedule (MBS) includes around **12,500** item numbers historically, with roughly **6,000** currently active[3]. Each item has detailed criteria and restrictions, making it easy for practitioners to miss billable services or make compliance mistakes[1]. The **MBSPro Challenge** addresses this problem by envisioning a system that enables **safe, ethical, and efficient** billing for GPs. The goal is to guide practitioners in real-time, ensuring they capture all eligible services without violating MBS rules – ultimately supporting both better care and practice sustainability.

System Overview and Architecture

MBSPro is implemented as a modern web application composed of a **Next.js 14** frontend and a **NestJS** backend, supported by a cloud database and several AI services. The architecture is modular, with clear separation of concerns between the user interface, core logic, and external integrations. Below is a text-based architecture diagram illustrating the main components and data flow:

Browser Client (GP user)
↳ Frontend: Next.js 14 (React) application
↳ REST API calls to backend (HTTPS)
↳ Backend: NestJS server (Node.js)
↳ Supabase (PostgreSQL) – primary data store for MBS items, rules, users
↳ Pinecone Vector DB – semantic index of MBS text (OpenAI embeddings)
↳ OpenAI GPT-4 API – document generation (referrals, care plans)
↳ Cohere Rerank API – optional result reranking
↳ HAPI FHIR server – local validation of FHIR resources

Communication and Integration: The Next.js frontend communicates with the NestJS backend via HTTP REST endpoints (configured via environment variables such as the API base URL). The frontend provides an interactive UI for searching and selecting MBS items, and it can capture voice input (via the Web Speech API) to enhance usability. The NestJS backend contains the application logic and connects to external services through well-defined interfaces. For example, it uses the Pinecone API (with

credentials like PINECONE_API_KEY, index name, environment region from configuration) to perform vector similarity searches. It leverages the OpenAI API (with an OPENAI_API_KEY) to access GPT-4 for generating clinical documents. If a Cohere API key is provided, the backend will call Cohere's rerank endpoint to refine search result ordering. The Supabase database (PostgreSQL) is used to store persistent data – including the MBS item catalog and policy rules – and is accessed from NestJS (using secure keys in environment variables for the connection). Finally, a local **HAPI FHIR** server (accessed via its base URL in config) is integrated for validation of FHIR resources. This server runs in the background and the backend sends it FHIR bundles (containing claims/encounters) to ensure they conform to standards before any submission.

Overall, the architecture is designed for **modularity and clarity**. The frontend is decoupled from backend logic (communicating only through APIs), and the backend in turn treats external services (database, vector store, LLMs, etc.) as pluggable components configured via environment variables. This separation allows each part of the system to be developed and scaled independently, and it simplifies debugging (for instance, the backend can be tested with or without the AI services enabled by toggling API keys in the .env configuration).

Core Technical Components

Retrieval & Ranking Engine (Hybrid Search Fusion)

To assist GPs in finding the right MBS items during a consultation, MBSPro employs a **hybrid retrieval engine** that combines semantic search with traditional lexical search. When the user searches for a service (e.g. by keyword or voice input), the backend performs two parallel queries: a **dense vector search** and a **textual SQL search**. The **Pinecone vector store** holds embeddings of MBS item descriptions and related notes, generated using OpenAI's embedding model. This allows semantic matches – for example, a query for "diabetes care plan" will retrieve items like *GP Management Plan* even if the exact words differ. Simultaneously, the system runs a **lexical query** on the Supabase (PostgreSQL) database (for instance, a full-text search or a direct code match) to catch explicit keyword hits or item number queries.

The results from these two methods are then **fused** in a ranker. Each result is scored and combined with weighted heuristics. If an item appears in both the semantic and lexical results (indicating high relevance agreement), it gets a **boost** in ranking. Tunable weights determine the balance between semantic similarity and textual relevancy. The engine may also incorporate domain-specific tweaks – for example, items that match the patient's context or the GP's past usage could be elevated. Optionally, a **Cohere ReRank** API call can be applied to the merged results: the top N candidate items (e.g. 10 or 20) along with the original query are sent to Cohere's reranker model, which returns a refined ordering. This reranking step, if enabled, can further improve precision by using a large language model to judge relevance in context. The outcome is a list of suggested MBS items ordered by relevance/confidence, delivered back to the frontend for the GP to review.

Confidence Scoring Algorithm

Each suggested item is accompanied by a confidence score that indicates how likely it is to be relevant and allowable. The scoring algorithm takes the raw similarity scores from the retrieval phase and **shapes** them into a 0–100% scale for user-friendly display. This is done by applying a sigmoid function to the combined score, which normalizes outliers and yields a more calibrated confidence measure. The system then adjusts these scores based on rule checks and multi-source agreement. For example, if an item was found via both vector and lexical search and also does not conflict with any billing rules in the current context, the system increases its score slightly (reflecting higher confidence in its suitability). Conversely, if an item is semantically relevant to the query but would violate a billing policy when combined with already-selected items, the score might be dampened or the item might be flagged (so the GP knows its applicability is limited). This dynamic scoring mechanism ensures that the top suggestions are not only relevant to the query but also **practically valid** given Medicare's rules. The confidence score thus helps practitioners quickly gauge which suggestions are most likely correct, reducing the cognitive burden and uncertainty in decision-making.

Policy Rules Engine

At the heart of MBSPro's safe billing guidance is a **rules engine** that enforces Medicare's billing policies. The system encodes billing rules in a structured JSON format, making the ruleset easy to update as policies change. These **policy rules** cover various constraints: mutual exclusivity (certain items cannot be co-claimed together), prerequisites (an item may require another service first or certain conditions), frequency limits (an item can only be claimed once in a time period), and other eligibility criteria (e.g. patient age or chronic condition requirements).

Rules are evaluated at two levels: - **Per Item:** As soon as a specific item is selected, the engine checks any item-specific rules. For instance, if an item requires a service referral or has a restriction like "not claimable in a home visit", the UI can warn the user or require confirmation. - **Per Selection (Combination):** Whenever the set of selected items is updated, the engine evaluates rules that involve multiple items. This catches conflicts such as the earlier example of attempting to combine a general consultation with a care plan on the same day[2]. If a forbidden combination is detected, the system immediately flags it and explains the issue (via the "Explain" feature, described later). The rules engine operates in real-time, meaning as the GP adds or removes items, feedback is provided instantly about any compliance problems or special conditions.

The JSON-based rules make it straightforward to represent complex logic. For example, a mutual exclusivity rule might be represented as an array of item codes that cannot coincide, along with a message to display if they do. A frequency rule might include an item code plus a timeframe (e.g. "only once per 12 months per patient"). By externalizing this logic from code into data, MBSPro can **easily expand or modify** rules without changing the core application – a key advantage given the frequent updates to MBS guidelines. The rules engine thus acts as a real-time safety net, ensuring that any combination of items a practitioner selects remains **compliant with Medicare policy**.

FHIR Builders & Validation Module

To integrate with healthcare systems and potentially submit claims electronically, MBSPro constructs standard HL7 FHIR resources for billing events. When a GP finalizes a set of MBS items for a consultation, the backend assembles a **FHIR Claim** resource (representing the billable services provided) along with related resources like **Encounter** (the clinical encounter context) and possibly **Patient** and **Practitioner** references if needed. These resources are packaged into a FHIR **Bundle**. The use of FHIR (Fast Healthcare Interoperability Resources) ensures the data is in an interoperable format that can be sent to other systems (e.g. Practice Management Software or Medicare's systems) in a standard way.

Once the FHIR Bundle is built, the system performs a validation step using a local **HAPI FHIR** server. The bundle (or individual resources) is submitted to HAPI's $validate operation, which checks the resources against the FHIR R4 specification and any relevant profiles. This validation catches any structural errors or missing required fields early in the process. For instance, if a required coding system or element is missing in the Claim resource, the HAPI FHIR server will return an error or warning. By leveraging this, the MBSPro backend can ensure the generated claim data is **well-formed and standards-compliant** before it ever leaves the system.

This FHIR builder/validator component not only helps in ensuring correctness, but it also lays the groundwork for integration with external health systems. In future, the validated Claim could be transmitted directly to Medicare or to a practice's billing system via FHIR APIs. For now, the presence of the HAPI FHIR validation step provides developers and users confidence that the data output (the claim for the services rendered) meets the formal requirements of the FHIR standard and the Australian Medicare specification.

Document Generation (Referral & Care Plan Automation)

Beyond billing codes, MBSPro assists with clinical documentation by auto-generating common documents like specialist referrals and chronic care plans. This feature is powered by **OpenAI's GPT-4**, orchestrated with carefully crafted prompts. The backend uses a combination of a **system prompt** (which sets the role and instructions for the model) and dynamic user-specific data to guide the document creation. Critically, the prompt includes a **JSON schema or structured template** that the model must follow, rather than relying on few-shot examples. This means GPT-4 is instructed to output the document content in a predefined JSON structure – for example, fields for patient details, clinical summary, plan actions, and so on.

When a GP requests a generated document (say, a care plan for a patient with diabetes), the system compiles the relevant context: this could include the transcribed notes from the consultation, the selected MBS items (which imply what was done), and any patient data available. The system prompt might say: *"You are an assistant that drafts professional medical documents. Generate a referral letter in JSON format with the following fields: ..."* followed by the required schema. The **user prompt** portion then supplies the specifics (e.g. patient age, conditions, purpose of referral, etc.). GPT-4 then produces a draft document adhering to this structure.

Because the output is JSON, the application can reliably parse it and display it in a user-friendly format (converting to a nicely formatted letter or form). No few-shot examples are provided – the model works off the explicit instructions and schema, leveraging GPT-4's capabilities to generate coherent, contextually relevant text. This approach avoids the need to maintain example documents and ensures consistency in structure. The generated content still requires the GP's review (to ensure clinical accuracy), but it can save significant time by providing a solid first draft. By automating referrals, care plans, and other paperwork, MBSPro reduces administrative burden, allowing doctors to focus more on patient interaction and less on typing up documents.

Supporting Features

Beyond its core logic, MBSPro includes a set of user-centric features to improve the experience and ensure transparency:

- **Voice Input:** On the item suggestion interface, practitioners can use voice dictation to search for MBS items. Leveraging the browser's Web Speech API, the frontend allows the GP to simply speak (e.g. "heart health assessment") and the spoken words are transcribed into the search query. This hands-free input can speed up workflows, especially during a busy consult when typing is inconvenient. The voice feature is built entirely in the frontend – once transcribed, the query is sent to the backend retrieval engine just like a typed query.

- **"Explain" Mode:** For any suggested or selected item, the UI can display an explanation of relevant rules or flags. This is powered by the rules engine's outputs. If an item is flagged by a policy (say it's not claimable due to a conflict or missing criteria), the Explain panel will list the reason (e.g. "Item X cannot be claimed with item Y in the same visit – these two services are mutually exclusive under MBS rules."). Even when there's no conflict, the user can request to see the billing criteria for an item (for example, "Requires a minimum of 20 minutes face-to-face time" or "Only for patients with chronic conditions present for >6 months"), helping them understand **why** an item is suggested and what is required to bill it. This feature promotes transparency and education, so users not only see *what* the system recommends, but also *why* – building trust and billing knowledge.

- **Swap Suggestions:** When a selected item triggers a rule disallowing it, MBSPro helps the user find an alternative. The **Swap** feature suggests rule-valid alternatives – for instance, if item A and B conflict, and A is selected, the system might suggest swapping B for another related item C that is permissible. Similarly, if an item is simply not the optimal choice (maybe a higher-value but equivalent item exists), the system can point that out. These suggestions are generated by searching within the same "family" of items or category. The system leverages the structured hierarchy of MBS (categories, groups, subgroups) to identify items with similar clinical intent. For example, if a certain health assessment item is not applicable due to age restrictions, the Swap might find a similar assessment code for a different age group. By providing alternatives, MBSPro ensures that a rule conflict or ineligible item doesn't dead-end the user – there's guidance on how to achieve the needed billing in a compliant way.

Limitations and Future Work

Current Limitations

While the MBSPro system provides a robust framework, there are some limitations in its current implementation. First, the **ruleset coverage** is not yet exhaustive – the JSON policy rules include many common GP scenarios (like co-claim exclusions and care plan rules) but may not cover every niche MBS item or latest policy tweak. As a result, some billing combinations might slip through without a flag, or certain suggestion nuances might be missed. Secondly, the reliance on external **LLM APIs** introduces latency; calls to GPT-4 for document generation or to Cohere for reranking add a few seconds to the workflow. In a live clinic environment, this delay could be noticeable. There's also dependency on internet connectivity and API availability for these features to work. Another limitation is **data freshness**: the MBS data and FHIR profiles need periodic updating. If the MBS is updated or new item numbers come into effect, there could be a version drift until the system's database and rules are updated accordingly. Similarly, the FHIR validation uses a snapshot of standards (e.g. a particular HAPI FHIR version with HL7 AU profiles); changes in standards or Medicare integration requirements might not instantly reflect in the system.

Integration with existing systems is limited in the current standalone version. For instance, while we output FHIR claims, we haven't fully integrated MBSPro into a live claiming system or a specific Electronic Medical Record (EMR) platform yet – meaning there's still a **gap to actual submission** (the GP or staff may need to manually enter the suggested codes into their billing software). Lastly, because GPT-4 is used without few-shot examples, the content of generated documents might vary slightly in style or completeness, and always needs human verification. The system avoids critical errors by design, but there is no on-device model running – it fully relies on the cloud AI, which raises considerations around **data privacy** (though all patient data in prompts are handled according to privacy rules and not stored on the LLM side).

Future Work

There are several avenues planned to enhance MBSPro beyond the challenge implementation. A top priority is **expanding the ruleset** – continuously updating and adding to the JSON rules to cover more MBS items and scenarios (for example, including specialist consultation rules, procedural items, etc.). We aim to incorporate feedback from users and domain experts to codify more complex policies, ensuring the engine's advice remains comprehensive and up-to-date with Medicare's changes.

Another area is deeper **integration with Practice Management Systems (PMS) and EMRs**. In the future, MBSPro could plug directly into popular GP software (e.g. Best Practice, MedicalDirector) so that suggestions and compliance checks happen seamlessly within the practitioner's existing workflow. This might involve using FHIR APIs or vendor-specific SDKs to pull patient data (with consent) and push billing claims or consult notes directly, eliminating duplicate data entry. Tied to this is potential integration with Medicare's online claiming (PRODA): an end-to-end solution could take a GP from suggestion to claim lodgment in one interface.

We also plan to utilize **feedback-driven tuning** of the AI components. By tracking which suggestions users accept or reject and which false positives occur, we can adjust the retrieval fusion weights or add training data for the reranker. For example, if certain

irrelevant items are frequently shown and ignored, the system can learn to suppress them. User feedback on generated documents will similarly guide prompt improvements or the introduction of templates to ensure the output consistently meets clinical expectations.

Finally, exploring **local LLM support** is on the roadmap. As on-premise or smaller-scale language models become more feasible, we could incorporate a local model for tasks like reranking or even document generation. This would reduce dependence on external APIs, cut latency, and alleviate privacy concerns (since data wouldn't leave the local environment). A local model fine-tuned on de-identified Australian medical data could also potentially improve the relevance of suggestions and explanations. While GPT-4 offers excellent performance, the future might see a hybrid approach where MBSPro uses cloud AI for heavy tasks when available, but can fall back to a capable local model if needed (for offline mode or faster responses).

In summary, the MBSPro system developed for the Challenge demonstrates a cohesive solution to the Medicare billing puzzle – but it's just the beginning. Ongoing work will focus on broadening its knowledge, tightening its integration with real-world GP systems, and leveraging AI advancements to make Medicare billing **safer, smarter, and smoother** for all practitioners.

Sources:

1. Services Australia – *MBS billing rules for chronic condition plans* (example of co-claim restrictions)[2]
2. MBS Online – *Medicare Item Information* (count of MBS items)[3]
3. Best Practice Software – *Smarter Billing, Better Care with MBSPro* (MBS complexity and challenges)[1]

---

[1] Smarter Billing, Better Care with MBSPro - Best Practice

https://bestpracticesoftware.com/blog/smarter-billing-better-care-with-mbspro/

[2] MBS billing rules for GP chronic condition management plans - Health professionals - Services Australia

https://www.servicesaustralia.gov.au/mbs-billing-rules-for-gp-chronic-condition-management-plans?context=20

[3] mbsonline.gov.au

https://www.mbsonline.gov.au/internet/mbsonline/publishing.nsf/Content/1BC94358D4F276D3CA257CCF0000AA73/$File/MBS%20Item%20Information.pdf