Contents

index

Pinocchio webchat: refactor getOrCreateConv to EngineBuilder pattern

Overview

Key Links

- **Related Files**: See frontmatter RelatedFiles field
- **External Sources**: See frontmatter ExternalSources field

Status

Current status: **active**

Topics

- pinocchio
- webchat
- refactor
- architecture

Tasks

See tasks.md for the current task list.

Changelog

See changelog.md for recent changes and decisions.

Structure

- design/ - Architecture and design documents
- reference/ - Prompt packs, API contracts, context summaries
- playbooks/ - Command sequences and test procedures
- scripts/ - Temporary code and tooling
- various/ - Working notes and research
- archive/ - Deprecated or reference-only artifacts

tasks

Tasks

TODO

- ☐ Pinocchio: design `EngineConfig` + `Signature()` for webchat composition
- ☐ Pinocchio: introduce go-go-mento-style `EngineBuilder` (`BuildConfig` / `BuildFromConfig`)
- ☐ Pinocchio: introduce `SubscriberFactory` (Redis vs in-memory)
- ☐ Pinocchio: refactor `getOrCreateConv` into a manager that rebuilds on signature change
- ☐ Pinocchio: remove per-route `build := func() ...` closures from `pinocchio/pkg/webchat/router.go`
- ☐ Pinocchio: switch `Conversation.Sink` to `events.EventSink` (so sinks can be wrapped)
- ☐ Tests: signature stability + rebuild behavior (profile change, override change)
- ☐ Moments: write a follow-up migration sketch (how Moments could adopt the same interfaces)

01-simplify-getorcreateconv-via-enginebuilder-pinocchio-webchat

Simplify `getOrCreateConv` via EngineBuilder (Pinocchio webchat)

Problem Statement

Pinocchio's webchat currently wires conversation creation with **per-callsite** `build := func() (...)` closures that construct:

- a provider `engine.Engine`
- a `WatermillSink` bound to the conversation topic
- a `message.Subscriber` (Redis group subscriber vs in-memory)

`getOrCreateConv` then **caches the first result** it sees for a `conv_id` and returns it on all subsequent calls.

This is hard to maintain because:

1. The closure logic is duplicated across multiple routes (`/ws`, `/chat`, `/chat/{profile}`).
2. Policy decisions (profile defaults, overrides parsing) are mixed with transport wiring (Redis subscriber setup).
3. Once a conversation exists, **profile/override changes are ignored**, which can lead to subtle "engine doesn't match request" behavior.

The goal of PI-001 is to remove these ad-hoc closures and replace them with a reusable **EngineBuilder + config signature** pattern inspired by go-go-mento (and compatible with Moments' heavier needs).

Scope / Non-goals

In scope (this ticket's refactor target):

- Replace per-callsite `build()` closures with a centralized builder abstraction.
- Make rebuild decisions deterministic (config signature).
- Introduce a reusable pattern that Moments can adopt later, without immediately porting Moments.

Out of scope for PI-001 (but the design should not block these):

- Full Moments migration / consolidation (that's MO-001).
- DB persistence/hydration logic.
- Step controller integration (separate workstream).

Current Pinocchio Architecture (As-Is)

Call graph (simplified)

Pinocchio has three major callsites that all construct a `build()` closure:

- WS connect (`/ws`)
- Start run (`/chat`) with cookie/default profile + overrides

- Start run (`/chat/{profile}`) with explicit profile + overrides

Each callsite calls:

```
conv, err := r.getOrCreateConv(convID, build)
```

Responsibility split today

`getOrCreateConv` (pinocchio) File: `pinocchio/pkg/webchat/conversation.go`

Responsibilities:

- "Create once per conv_id" cache (no rebuild logic).
- Assign a generated `RunID` (effectively the session id).
- Attach:
  - connection pool
  - stream coordinator
  - session (`*session.Session`) seeded with a first turn and `ToolLoopEngineBuilder` (base engine + event sinks)
- Start the stream coordinator immediately.

`build()` closures (callsite-local) File: `pinocchio/pkg/webchat/router.go`

Duplicated responsibilities:

1. **Transport wiring**
   - `EnsureGroupAtTail` (Redis)
   - build group subscriber (Redis) vs use in-memory subscriber (router.Subscriber)
2. **Sink creation**
   - `middleware.NewWatermillSink(r.router.Publisher, topicForConv(convID))`
3. **Settings + policy application**
   - StepSettings from `ParsedLayers`
   - profile default system prompt and middleware list
   - apply overrides (system_prompt, middlewares)
4. **Engine composition**
   - `composeEngineFromSettings(stepSettings, sys, uses, r.mwFactories)`

Observed issues

A) "Create-once cache" ignores profile/override changes Because `getOrCreateConv` does not compare a config signature, the first route to create a conversation "wins". For example:

- A client joins via WS using profile A (creates conv + engine A).
- Later, a client POSTs `/chat` with overrides (expects engine B).
- `getOrCreateConv` returns the existing conversation, and the overrides are silently ignored for the base engine.

Moments partially addresses this by rebuilding on a signature change, but Pinocchio does not today.

B) The closure return type forces `*middleware.WatermillSink` Pinocchio stores `Sink *middleware.WatermillSink` on the conversation, not `events.EventSink`. That makes it awkward to introduce sink wrapper pipelines (extractors, filtering, fanout) without changing types.

go-go-mento treats WatermillSink as the base sink and returns `events.EventSink` from its builder.

C) Policy and transport are entangled Redis group management and subscriber creation are embedded in the same closure as profile parsing and engine composition. This is exactly the kind of "wiring glue" that becomes brittle as the code grows.

Reference Patterns (What We Want To Reuse)

go-go-mento: EngineBuilder + EngineConfig + ConversationManager

Files:

- `go-go-mento/go/pkg/webchat/engine_builder.go`
- `go-go-mento/go/pkg/webchat/engine_config.go`
- `go-go-mento/go/pkg/webchat/conversation_manager.go`

Key ideas:

1. **EngineConfig as a first-class object**
   - Capture all inputs that influence engine composition.
   - Serialize to JSON and use the JSON string as a debuggable signature.
2. **EngineBuilder centralizes composition**
   - `BuildConfig(profileSlug, overrides) -> EngineConfig`
   - `BuildFromConfig(convID, EngineConfig) -> (engine, eventsink)`
3. **Subscriber creation is independent**
   - `SubscriberFactory(convID) -> subscriber`
4. **ConversationManager owns rebuild decisions**
   - Rebuild engine/sink/subscriber if profile or signature changes.

This pattern is the closest direct match for what Pinocchio needs.

Moments: similar needs, but "ad-hoc closure" wiring remained

Files:

- `moments/backend/pkg/webchat/router.go`
- `moments/backend/pkg/webchat/conversation.go`

Moments already has:

- a `getOrCreateConv` that can rebuild on "signature changes"
- a concept of profile slug and a per-conversation engine config signature field
- more state (identity session refresh, step controller, doc lens extractors, etc.)

But Moments still relies on:

- callsite-local `build := func() (engine, sink, subscriber)` closures

- a weak signature: `profileSlug + constantSuffix`

Pinocchio can provide a cleaner reusable pattern that Moments can later adopt.

Geppetto: `session.EngineBuilder` (runner builder) is related but distinct

File: `geppetto/pkg/inference/session/builder.go`

Geppetto's "builder" builds a **runner** (`InferenceRunner`) for a session, not an engine/sink/subscriber triple. It's still relevant because Pinocchio webchat ultimately needs to run inference via `session.Session.StartInference`.

But the immediate PI-001 refactor target is the webchat "composition glue" around `getOrCreateConv`.

Proposed Design (To-Be)

Design Principle

Replace:

- "per-callsite closure that returns engine/sink/subscriber"

With:

- "centralized EngineBuilder (engine + sink)"
- "centralized SubscriberFactory (subscriber)"
- "ConversationManager (get-or-create + rebuild-on-signature)"

Diagram: To-Be components

```
                     +------------------------------+
HTTP/WS handlers --> | ConversationManager          |
                     | - GetOrCreate(conv_id, ...)  |
                     | - rebuild on signature       |
                     +---------------+--------------+
                                     |
                  +------------------+------------------+
                  |                                     |
         +--------v---------+                  +-------v--------+
         | EngineBuilder    |                  | Subscriber      |
         | - BuildConfig    |                  | Factory         |
         | - BuildFromConfig|                  | - Build(conv_id)|
         +--------+---------+                  +-------+--------+
                  |                                     |
                  v                                     v
          (engine, eventsink)                     (subscriber)
                  |
                  v
         +--------+---------+
         | Conversation     |
         | - engine         |
         | - sink           |
         | - subscriber     |
         | - stream         |
         | - session          |
         +-------------------+
```

Proposed Pinocchio EngineConfig (minimal common denominator)

We want something that matches go-go-mento closely enough that migration is obvious:

```go
type EngineConfig struct {
    ProfileSlug  string                 `json:"profile_slug"`
    SystemPrompt string                 `json:"system_prompt"`
    Middlewares  []MiddlewareUse        `json:"middlewares"`
    StepSettings *settings.StepSettings `json:"step_settings"`
    // Optional: Tools []string (Pinocchio tool registry is currently built per-run)
}

func (c EngineConfig) Signature() string // returns JSON string
```

Notes:

- StepSettings must be included (or a stable derivative of it), otherwise the signature is incomplete.
- Tools are currently assembled per run from `Router.toolFactories`; we can add them later if we want rebuilds to occur when tool configuration changes.

Proposed EngineBuilder interface (Pinocchio)

This is intentionally isomorphic to go-go-mento:

```go
type EngineBuilder interface {
    BuildConfig(profileSlug string, overrides map[string]any) (EngineConfig, error)
    BuildFromConfig(convID string, cfg EngineConfig) (engine.Engine, events.EventSink, err
}
```

Implementation dependencies likely include:

- `parsed *layers.ParsedLayers`
- `profiles ProfileRegistry`
- `mwFactories map[string]MiddlewareFactory`
- `publisher message.Publisher` (to build WatermillSink)
- optional sink wrapping hooks (for future Moments reuse)

Proposed SubscriberFactory (Pinocchio)

Keep transport details out of the EngineBuilder:

```go
type SubscriberFactory func(convID string) (message.Subscriber, error)
```

Pinocchio already has two distinct strategies:

- Redis: ensure group + build group subscriber with consumer name `ws-forwarder:<conv_id>`
- In-memory: reuse `router.Subscriber`

Proposed ConversationManager (Pinocchio)

This can start as a simplified port of go-go-mento's manager:

```go
type ConversationManager struct {
    mu            sync.Mutex
    conversations map[string]*Conversation
    builder       EngineBuilder
    subscriber    SubscriberFactory
    baseCtx       context.Context
    idleTimeout   time.Duration
}

func (cm *ConversationManager) GetOrCreate(
    ctx context.Context,
    convID string,
    profileSlug string,
    overrides map[string]any,
) (*Conversation, error)
```

Behavior:

- Build EngineConfig from profile + overrides.
- Compute signature.
- If conversation exists:
  - If profile changed OR signature changed: rebuild engine/sink, rebuild subscriber, reattach stream coordinator.
  - Else: reuse.
- If conversation does not exist:
  - create new conversation, attach pool/stream/session.

"Sink type" choice: store `events.EventSink` not `*WatermillSink`

Recommendation:

- Conversation should store `events.EventSink` (which can be a WatermillSink or a wrapped sink).
- The base Watermill sink should be an internal detail of the EngineBuilder.

This aligns with go-go-mento and makes Moments adoption easier (Moments uses sink wrapper pipelines heavily).

How This Helps Moments Later

Moments has the same refactor pain, but amplified:

- more sink wrapping
- more "profile resolution" complexity
- identity session refresh
- step controller
- persistence/hydration hooks

If Pinocchio adopts the go-go-mento-style builder/manager split, Moments can migrate incrementally:

1. Introduce a Moments EngineBuilder that implements the same interface:

   - BuildConfig(profileSlug, overrides) returns a JSON-signatured config including resolved prompt/middleware/tool lists.
   - BuildFromConfig(convID, cfg) composes engine + sink pipeline over a base Watermill sink.
2. Swap Moments' `getOrCreateConv` to call the builder, and delete ad-hoc callsite closures.
3. Keep Moments-only concerns (identity, step mode) in the conversation manager layer, not in the builder.

This allows a "clean core" to exist even if Moments' package remains messy in the short term.

Migration Plan (Implementation Checklist for PI-001)

No code is written in this ticket yet; this is an implementation plan for the next phase.

1. Define `EngineConfig` + `Signature()` in Pinocchio webchat (or a shared pinocchio package).
2. Introduce `EngineBuilder` interface matching go-go-mento's API.
3. Implement a Pinocchio `EngineBuilder` using:
   - `settings.NewStepSettingsFromParsedLayers(r.parsed)`
   - profile default prompt/middlewares
   - override parsing logic currently in `/chat` handlers
   - `composeEngineFromSettings(...)`
   - `middleware.NewWatermillSink(r.router.Publisher, topicForConv(convID))`
4. Introduce `SubscriberFactory` for Redis vs in-memory subscriber creation.
5. Replace `Router.getOrCreateConv(convID, build)` with `ConversationManager.GetOrCreate(...)`.
6. Remove the duplicated `build := func() ...` blocks from `pinocchio/pkg/webchat/router.go`.
7. Add tests for:
   - config signature stability
   - rebuild on signature change
   - no rebuild on identical inputs

Open Questions

1. Should Pinocchio rebuild conversations on:
   - WS join profile (cookie) changes?
   - `/chat` overrides changes mid-session? The go-go-mento answer is "yes, rebuild when signature changes".
2. Do we want a "session manager" layer (go-go-mento style) in Pinocchio soon anyway? If yes, PI-001 should structure code to make that addition natural.

Appendix: Where to look in code (As-Is)

- Pinocchio:
  - `pinocchio/pkg/webchat/router.go` (three duplicated build closures)
  - `pinocchio/pkg/webchat/conversation.go` (`getOrCreateConv` cache semantics)
  - `pinocchio/pkg/webchat/engine.go` (`composeEngineFromSettings`)
  - `pinocchio/pkg/inference/enginebuilder/parsed_layers.go` (existing minimal builder)
- go-go-mento:
  - `go-go-mento/go/pkg/webchat/engine_builder.go`
  - `go-go-mento/go/pkg/webchat/engine_config.go`

  - `go-go-mento/go/pkg/webchat/conversation_manager.go`

- Moments:
  - `moments/backend/pkg/webchat/router.go`
  - `moments/backend/pkg/webchat/conversation.go`

01-diary

Diary

Goal

Track the investigation and design work for refactoring Pinocchio's webchat `getOrCreateConv` to stop using ad-hoc `build()` closures and instead use a reusable EngineBuilder-style pattern inspired by go-go-mento (and compatible with Moments' more complex needs), without writing code yet.

Step 1: Create Ticket, Seed Docs, And Gather References

This step created the PI-001 ticket workspace and established the documentation scaffolding for an in-depth analysis. The immediate goal was to "freeze" the problem statement and identify the most load-bearing files in Pinocchio, go-go-mento, and Moments before proposing a refactor plan.

The main outcome is a new analysis document with a curated set of related source files so that future steps (including eventual implementation) can proceed systematically and review can focus on a small number of core artifacts.

What I did

- Created the docmgr ticket `PI-001-WEBCHAT-ENGINEBUILDER`.
- Added an analysis doc and a diary doc for the ticket.
- Related the most relevant Pinocchio/go-go-mento/Moments/Geppetto files to the analysis doc.

Why

- The current Pinocchio webchat creates engines/sinks/subscribers via per-callsite closures; this is hard to reason about, hard to reuse, and diverges from the stronger "builder/manager" patterns we already have in go-go-mento (and partially in Moments).
- Establishing a ticket + doc trail early keeps the refactor design reviewable and makes later implementation less error-prone.

What worked

- `docmgr ticket create-ticket` created the workspace under `geppetto/ttmp/2026/01/22/`.
- `docmgr doc add` produced the analysis + diary docs with the expected frontmatter.
- `docmgr doc relate` successfully attached the intended cross-repo file references.

What didn't work

- N/A (analysis-only step; no code changes and no tests required).

What I learned

- The "EngineBuilder" pattern in go-go-mento is explicitly designed to keep Router handlers lean by centralizing engine + sink composition, and it cleanly separates "subscriber creation" as another dependency (SubscriberFactory). That split looks directly applicable to Pinocchio.

- Pinocchio already has a small `ParsedLayersEngineBuilder` abstraction (`pinocchio/pkg/inference/engin`
  but it currently doesn't cover subscriber creation or sink wrapping and therefore can't replace the `build()` closures in webchat yet.

What was tricky to build

- N/A (no implementation in this step), but the main "tricky" aspect is scoping: we need to design a builder API that is reusable across Pinocchio and later Moments without prematurely importing all of Moments' complexity (profiles, sink pipelines, persistence, step controller).

What warrants a second pair of eyes

- Ensure the ticket scope is correct: PI-001 should focus on refactoring Pinocchio's webchat builder plumbing (removing closures) and not accidentally turn into the broader "MO-001 port moments webchat" consolidation work.

What should be done in the future

- Continue with a deep comparative analysis of:
  - Pinocchio's current `getOrCreateConv` call graph and responsibilities.
  - go-go-mento's `EngineBuilder` + `ConversationManager` responsibilities split.
  - Moments' current build closure patterns (where it got more complex and why).

Code review instructions

- Start at `geppetto/ttmp/2026/01/22/PI-001-WEBCHAT-ENGINEBUILDER--pinocchio-webchat-refactor-getorcreateconv-to-enginebuilder-pattern/analysis/01-simplify-getorcreateconv-via-enginebuilder-pinocchio-webchat.md`.
- Validate that `RelatedFiles` includes the intended core sources (Pinocchio webchat, go-go-mento builder/manager, Moments router, Geppetto session builder).

Technical details

- Ticket created: `PI-001-WEBCHAT-ENGINEBUILDER`
- Docs created:
  - `geppetto/ttmp/2026/01/22/PI-001-WEBCHAT-ENGINEBUILDER--pinocchio-webchat-refactor-getorcreateconv-to-enginebuilder-pattern/analysis/01-simplify-getorcreateconv-via-enginebuilder-pinocchio-webchat.md`
  - `geppetto/ttmp/2026/01/22/PI-001-WEBCHAT-ENGINEBUILDER--pinocchio-webchat-refactor-getorcreateconv-to-enginebuilder-pattern/reference/01-diary.md`

Step 2: Map Pinocchio's Current getOrCreateConv Responsibilities

This step closely read Pinocchio's current webchat router and conversation lifecycle to understand exactly what the `build()` closures are responsible for today. The key outcome is a sharper problem statement: `getOrCreateConv` is only a "create-once cache", while the call-sites' closures contain policy (profiles/overrides) and transport wiring (subscriber setup) that should be centralized.

This is important because Moments (and go-go-mento before it) already solved this class of problem by separating "engine/sink composition" from "subscriber creation" and then letting a manager decide when to rebuild (based on deterministic config signatures). Pinocchio is currently missing that abstraction boundary.

What I did

- Read Pinocchio webchat lifecycle:
  - `pinocchio/pkg/webchat/router.go` (WS join + /chat handlers).
  - `pinocchio/pkg/webchat/conversation.go` (`getOrCreateConv`, connection pool, stream startup).
  - `pinocchio/pkg/webchat/engine.go` (engine composition policy; middleware order).
- Identified all `build := func() (engine.Engine, *middleware.WatermillSink, message.Subscriber, error)` callsites.
- Noted exactly which responsibilities are duplicated and which are "policy decisions".

Why

- We need to remove `build()` closures from callsites without losing configurability (profiles, overrides).
- We need a design that can later be reused by Moments, where the same pattern exists but with more layers (sink pipelines, profile resolution, persistence, step controller).

What worked

- The Pinocchio webchat code is compact and readable enough that responsibilities can be listed explicitly:
  - transport plumbing (Redis group/subscriber creation),
  - sink creation (Watermill sink bound to topic),
  - settings extraction (StepSettings from ParsedLayers),
  - policy application (profile defaults + request overrides),
  - engine composition (system prompt + middleware uses),
  - session creation (ToolLoopEngineBuilder + seed turn).

What didn't work

- N/A (analysis-only).

What I learned

- `getOrCreateConv` currently ignores profile/override changes once a conversation exists. That means a "WS join" can silently lock in engine settings, and later `/chat` requests with overrides will reuse the existing engine. This is the exact class of issue that go-go-mento's `ConversationManager` signature check avoids.
- The current closure type forces `*middleware.WatermillSink` (not `events.EventSink`), which makes it harder to introduce sink wrapper pipelines without touching core types. go-go-mento returns `events.EventSink` and treats WatermillSink as just the base sink.

What was tricky to build

- The biggest "sharp edge" is disentangling responsibilities without changing behavior:
  - WS join wants "default profile engine" even if no run is started yet.
  - /chat wants to apply overrides (system prompt, middleware list) for engine composition.
  - Both paths want consistent subscriber creation behavior under Redis vs in-memory transport.

What warrants a second pair of eyes

- The conclusion that Pinocchio needs signature-based rebuild semantics (not just "create-once") is a functional behavior question: if we introduce rebuilds, we must ensure this doesn't break UI/client assumptions about session stability.

What should be done in the future

- Compare Pinocchio's closure responsibilities against go-go-mento and Moments, and propose a minimal "common denominator" interface set (EngineBuilder + SubscriberFactory + optional SinkBuilder).

Code review instructions

- Read these files in order:
  - `pinocchio/pkg/webchat/router.go` (find the three `build := func() ...` blocks).
  - `pinocchio/pkg/webchat/conversation.go` (see how `buildEng()` is only used on create).
  - `pinocchio/pkg/webchat/engine.go` (policy for middleware application order).

Technical details

- Pinocchio closure signature today:
  - `func() (engine.Engine, *middleware.WatermillSink, message.Subscriber, error)`
- Pinocchio conversation cache semantics:
  - "create once per conv_id; return existing conversation thereafter"

Step 3: Compare Against go-go-mento and Moments (Builder + Manager Patterns)

This step examined the more mature patterns in go-go-mento and the "evolved but messy" reality in Moments. The goal was to extract a reusable pattern that Pinocchio can adopt now (to remove the closures) while staying compatible with Moments' needs later (sink pipelines, identity/session refresh, step controller, persistence/hydration).

The key conclusion is that go-go-mento already provides the right architectural split: Engine composition is centralized (EngineBuilder + EngineConfig signatures) while subscriber creation is a separate dependency (SubscriberFactory). Moments partially re-implemented this, but still wires composition via ad-hoc closures and uses a weak "signature" check.

What I did

- Read go-go-mento's engine composition pipeline:
  - `go-go-mento/go/pkg/webchat/engine_builder.go`
  - `go-go-mento/go/pkg/webchat/engine_config.go`
- Read go-go-mento's lifecycle orchestration:
  - `go-go-mento/go/pkg/webchat/conversation_manager.go`
- Read Moments' current (messier) version of the same idea:
  - `moments/backend/pkg/webchat/router.go` (inline `build := func() ...` closures)

  - `moments/backend/pkg/webchat/conversation.go` (`getOrCreateConv` rebuild-on-profile change)
- Re-read Pinocchio's existing `ParsedLayersEngineBuilder`:
  - `pinocchio/pkg/inference/enginebuilder/parsed_layers.go`

Why

- Pinocchio should adopt the "good parts" (centralized build logic + signature-based rebuild) instead of expanding the per-callsite closure approach.
- If Pinocchio's pattern is close enough to go-go-mento's, Moments can later reuse it (or migrate gradually) instead of remaining a parallel, bespoke implementation.

What worked

- go-go-mento's design is explicit and debuggable:
  - `EngineConfig` is JSON-serializable and its `Signature()` is intentionally the JSON string (not a hash).
  - `ConversationManager.GetOrCreate` compares signatures and rebuilds engine/sink/subscriber when needed.
  - Subscriber creation is pluggable via `SubscriberFactory`, keeping the engine builder transport-agnostic.

What didn't work

- Moments' current rebuild signature is too weak:
  - `newSig := profileSlug + engineBuildSignatureSuffix` ignores most composition inputs and doesn't directly encode overrides. It helps force recomposition in some scenarios, but it isn't a principled "config signature" like go-go-mento's.
- Moments still relies on ad-hoc closures at callsites (despite having logic inside `getOrCreateConv` to rebuild on signature changes).

What I learned

- The "right" unit of reuse is not "a closure that returns (engine,sink,subscriber)"; it's a small set of composable interfaces:
  - `EngineBuilder` (engine + sink composition, driven by a serializable config)
  - `SubscriberFactory` (transport-specific subscription strategy)
  - optionally `SinkBuilder` (Moments-style sink pipelines over a base Watermill sink)
- Pinocchio's current `ParsedLayersEngineBuilder` is too narrow to replace the webchat closures because:
  - it doesn't build a sink per conversation/topic,
  - it doesn't support config signatures/rebuild decisions,
  - it doesn't help with subscriber creation.

What was tricky to build

- Reconciling terminology and scope:
  - go-go-mento's "EngineBuilder" builds `(engine, eventsink)` and returns `EngineConfig`.
  - Geppetto's `session.EngineBuilder` builds a runner for a session.

  - Pinocchio currently uses “builder” to mean “build engine + sink + subscriber closure”. We need to choose names/interfaces in Pinocchio that are unambiguous and align with existing usage.

What warrants a second pair of eyes

- Whether Pinocchio should copy go-go-mento’s exact API surface (BuildConfig/BuildFromConfig) or adapt it:
  - Copying makes future Moments migration easier.
  - Adapting might better align with Geppetto’s `session.EngineBuilder` but risks divergence.

What should be done in the future

- Write the PI-001 analysis doc with:
  - A proposed interface set for Pinocchio (mirroring go-go-mento where possible).
  - A migration plan that removes closures from `pinocchio/pkg/webchat/router.go`.
  - A “Moments adoption plan” describing how Moments could eventually swap its closures for the same builder/manager.

Code review instructions

- Compare these implementations directly:
  - go-go-mento: `go-go-mento/go/pkg/webchat/engine_builder.go`
  - go-go-mento: `go-go-mento/go/pkg/webchat/conversation_manager.go`
  - moments: `moments/backend/pkg/webchat/conversation.go`
  - pinocchio: `pinocchio/pkg/webchat/router.go` and `pinocchio/pkg/webchat/conversation.go`

Technical details

- go-go-mento: `EngineConfig.Signature()` returns JSON (human-debuggable).
- moments: signature is effectively `profileSlug + const` today (not input-complete).

Step 4: Write The Refactor Analysis (Proposed To-Be Design)

This step distilled the findings from Pinocchio/go-go-mento/Moments into a concrete “To-Be” design: Pinocchio should adopt a go-go-mento-style `EngineBuilder` and `EngineConfig` signature mechanism, and move “get-or-create + rebuild-on-signature” into a manager layer. The analysis doc is intended to be implementation-ready while still being reusable as a design reference for Moments migration.

The main deliverable is the PI-001 analysis doc with explicit responsibilities, diagrams, interfaces, and a migration checklist.

What I did

- Wrote the analysis doc:
  - `geppetto/ttmp/2026/01/22/PI-001-WEBCHAT-ENGINEBUILDER--pinocchio-webchat-refactor-getorcreateconv-to-enginebuilder-pattern/analysis/01-simplify-getorcreateconv-via-enginebuilder-pinocchio-webchat.md`
- Updated the ticket task list to reflect the proposed implementation checklist.

Why

- We need a shared "north star" doc before writing code so that:
  - Pinocchio's refactor doesn't drift into unrelated webchat changes,
  - and Moments can later adopt the same pattern without yet another re-implementation.

What worked

- The doc now captures:
  - the "as-is" responsibilities split (closures vs cache),
  - the "to-be" split (EngineBuilder + SubscriberFactory + ConversationManager),
  - and the rationale for storing `events.EventSink` instead of `*WatermillSink` on the conversation.

What didn't work

- N/A (analysis-only; no implementation yet).

What I learned

- The minimum reusable pattern for Pinocchio↔Moments is interface-level:
  - adopt the same conceptual split as go-go-mento (builder + manager + subscriber factory),
  - and keep Moments-only complexity in the manager layer, not inside engine composition.

What was tricky to build

- The main tricky part is naming and compatibility:
  - "EngineBuilder" means different things in Geppetto (runner builder) vs go-go-mento (engine+sinks). The doc deliberately mirrors go-go-mento's API shape for easier migration, while calling out that Geppetto's session builder is still a separate concept.

What warrants a second pair of eyes

- Confirm that the recommended change "store `events.EventSink` in Conversation" is acceptable in Pinocchio and doesn't break any assumptions in the current webchat run loop wiring.

What should be done in the future

- Upload the analysis doc (and diary) to reMarkable for review.
- Only after review, start implementing PI-001 tasks in small, testable slices.

Code review instructions

- Read the analysis doc top-to-bottom and sanity check:
  - the identified Pinocchio bug/behavior (create-once ignores overrides),
  - the proposed interface set,
  - and the migration plan sequencing.

Technical details

- Analysis doc path:
  - `geppetto/ttmp/2026/01/22/PI-001-WEBCHAT-ENGINEBUILDER--pinocchio-webchat-refactor-getorcreateconv-to-enginebuilder-pattern/analysis/01-simplify-getorcreateconv-via-enginebuilder-pinocchio-webchat.md`