Agentic AI Solution – Repository Structure, Branching, Deployment, and Versioning Strategy

none

1. Introduction

As part of setting up the **Agentic AI Solution** project to support multiple use cases, we are establishing a **multi-repository model** in GitLab. This approach aligns with MetricStream's current repository structure standards and will allow better modularity, deployment management, and infrastructure scaling.

The following sections describe the repository structure, branching strategy, deployment workflow, and versioning approach. This will also serve as a foundation for setting up CI/CD pipelines via Jenkins.

2. Repository Structure

We will initially create the following GitLab repositories:

Repository Name	Purpose
`admin-frontend`	Frontend UI for administration and configuration
`agentic-core-framework`	Core backend framework, APIs, and orchestration logic
`ai-packages`	AI models, prompt templates, specialized logic packages
`socket-server`	Real-time communication server for live updates, events

> Note:
> Additional repositories might be introduced depending on finalization of pending design decisions. These will be discussed and finalized over the next few days.

3. Branching Strategy

We will maintain **four primary branches** per repository, aligned with environment lifecycles:

Branch Name	Purpose
`development`	Active development and feature integration
`qa`	Code ready for QA testing and internal validation
`uat`	Code ready for User Acceptance Testing (UAT)
`main`	Stable, production-ready code

Important Points:

- All primary branches will be **protected**.
- Code promotion will follow a strict pipeline through the environments.
- Merge from `development` → `qa` will happen only after the dev branch is sufficiently stable.

4. Deployment Strategy

The **code promotion** model will be followed to align with MetricStream's best practices:

From Branch	To Environment	Notes
`development`	Development Environment	Active developer testing
`qa`	QA Environment	After dev stabilization
`uat`	UAT Environment	Pre-production validation
`RELEASE-<version>`	Production Environment	Controlled release deployment

- For **<u>Production</u>**:
    - After UAT sign-off, a **release branch/tag** (`RELEASE-<version>`) will be created from the `uat` branch.
    - Production deployment will be based on this release branch.
    - Post-production deployment, the `RELEASE-<version>` branch will be merged into the `main` branch to keep production state reflected.

  Note:
  Given the fast-paced timelines, we will not introduce an intermediate environment between Dev and QA unless

required later.
However, if it's an enforced MetricStream standard later, we are open to revisiting this.

5. Versioning Strategy

We will adopt a **combined Month-Year and Semantic Versioning model** for better clarity and release traceability.

Format:
`YYYY.MM.Major.Minor.Patch`

Example:

`2025.04.1.0.0` → (Year = 2025, Month = April, Major version 1, Minor version 0, Patch version 0)

This strategy ensures:

- Easy identification of releases by time periods.
- Smooth support for hotfixes, patches, minor upgrades without confusion.

---

6. Summary

Aspect	Strategy
Repository Structure	Multi-repo aligned with product modules
Branching Model	4 Primary Branches (development, qa, uat, main)
Deployment Model	Code promotion through environments, controlled production releases
Versioning	Year.Month.Major.Minor.Patch (Example: 2025.04.1.0.0)

Next Steps:

- Create the initial repositories and protect primary branches.
- Setup Jenkins CI/CD pipelines based on this branching and promotion strategy.
- Finalize pending design decisions to add any further repositories if required.