1. Model Registry – Technical Specification

Purpose

Central repository to register, version, and manage all AI/ML and SLM models used in **Quarlets**, including Molecular Transformers, GNNs, property predictors, and text-based SLMs for chemistry/NLP tasks.
It ensures **traceability**, **reproducibility**, **governance**, and **compliance** across the model lifecycle (development → staging → production → deprecation).

Core Technologies

- **Backend:** FastAPI (Python 3.11)
- **Metadata Store: PostgreSQL** ✅ (recommended for relational, schema-based metadata with JSON support for flexible fields)
- **Model Artifacts Storage:** AWS S3 / MinIO
- **Model Versioning:** MLflow / Hugging Face Hub integration
- **Authentication:** OAuth2 + API key per model
- **Monitoring:** Prometheus + Grafana (model status, performance metrics)

🧩 Model Metadata Schema

Below is a detailed list of **metadata fields** you should store per model in the registry:

Category	Field	Description	Data Type	Example
Identity	`model_id`	Unique UUID for model	UUID	`d34e5b71-3f1d-4d19-9f0f-1b5b7e1a3eaf`
	`model_name`	Descriptive model name	VARCHAR(150)	`molecular-transformer-v1`
	`display_name`	Friendly name shown in marketplace	VARCHAR(200)	`Molecular Transformer for Reaction Prediction`
Versioning	`version`	Semantic version (major.minor.patch)	VARCHAR(20)	`1.0.3`
	`parent_model_id`	Optional – base model from which this was derived	UUID (nullable)	
	`source_repo`	Source repository or model hub link	VARCHAR(255)	`https://huggingface.co/quarlets/moltransformer`
Model Type & Category	`model_type`	Core type (Transformer / GNN / SLM / Regression / Ensemble)	VARCHAR(50)	`Transformer`

	domain	Application area	VARCHAR(50)	Retrosynthesis
	tags	Comma-separated keywords	TEXT	reaction, synthesis, transformer
Artifact Details	artifact_path	S3/MinIO path for model weights	VARCHAR(255)	s3://quarlets-models/transformer/v1/model.pt
	model_format	Serialization format	VARCHAR(50)	PyTorch, ONNX, TensorFlow
	input_schema	Expected input format (JSON)	JSONB	{"type":"SMILES","shape":"string"}
	output_schema	Output data structure	JSONB	{"type":"reaction_prediction","format":"JSON"}
	dependencies	Required libraries & versions	JSONB	{"rdkit":"2024.03","torch":"2.3.0"}
Training Details	dataset_name	Training dataset used	VARCHAR(150)	USPTO-50k
	dataset_version	Dataset version	VARCHAR(50)	v2.1
	training_parameters	Hyperparameters	JSONB	{"lr":0.0001,"batch_size":32}
	framework	ML framework	VARCHAR(50)	PyTorch

	`hardware_used`	GPU/CPU details	VARCHAR(100)	`A100-40GB`
Performance Metrics	`metrics`	Evaluation results	JSONB	`{"accuracy":0.92,"f1_score":0.89}`
	`benchmark_dataset`	Dataset used for benchmark testing	VARCHAR(150)	`ChEMBL-Bench`
Lifecycle & Governance	`status`	Model lifecycle stage	ENUM(`development`, `staging`, `production`, `deprecated`)	`production`
	`created_by`	User who registered the model	VARCHAR(100)	`ramesh.naidu@quarlets.ai`
	`created_at`	Timestamp	TIMESTAMP	`2025-10-07T12:34:00Z`
	`last_updated_at`	Timestamp	TIMESTAMP	
	`reviewer`	Reviewer/Approver	VARCHAR(100)	`qa-team@quarlets.ai`
	`approval_notes`	Comments from reviewer	TEXT	`Validated for FDA submission use.`
Security & Integrity	`checksum`	SHA256 checksum of model file	CHAR(64)	`7c222fb2927d828af22f592134e8932480637c0d`

	`encryption_status`	Boolean if encrypted	BOOLEAN	`true`
	`signed_by`	Digital signature identity	VARCHAR(100)	`quarlets-signing-service`
	`access_policy_id`	Reference to policy in MCP Server	UUID	
Runtime Details	`inference_endpoint`	URL for deployed model	VARCHAR(255)	`https://api.quarlets.ai/inference/moltransformer-v1`
	`resource_requirements`	CPU/GPU/memory requirements	JSONB	`{"gpu":1,"memory":"16GB"}`
Audit & Logs	`last_accessed`	Timestamp of last usage	TIMESTAMP	
	`access_count`	Total number of inference calls	INTEGER	`4502`
	`usage_stats`	Aggregate usage metrics	JSONB	`{"calls_last_7_days":502,"avg_latency":0.34}`
Env	`Env_type`	This model is used for Dev/staggi ng/production	text	

🧱 Database Design Choice

✅ **PostgreSQL** is the best fit because:

- Strong ACID guarantees and schema enforcement for model metadata.
- **JSONB columns** allow flexible storage for evolving metadata like hyperparameters and metrics.
- Supports **advanced indexing (GIN)** for JSON fields and text search (for tags and model names).
- Integrates easily with **MLflow** and **FastAPI**.
- Can scale horizontally via read replicas or **Citus** extension.

Optional secondary systems:

- **Redis** for caching frequently requested metadata (latest models, popular tags).
- **Elasticsearch** for advanced search and filtering by tags, metrics, or domains.

APIs (unchanged + internally extended)

- `POST /models/register` → Register new model with metadata + upload file to S3.
- `GET /models/{id}` → Fetch model details, metadata, metrics.
- `GET /models/latest?type=gnn` → Fetch latest active version by type.
- `POST /models/promote/{id}` → Move model from staging → production (with policy link).
- `GET /models/list` → Paginated list by type, tags, or status.
- `GET /models/search?q=transformer&domain=synthesis` → Metadata-based search.
- `GET /models/metrics/{id}` → Retrieve current metrics snapshot.

Infra / Deployment

- **Containerized** via Docker, deployed on **Kubernetes (EKS / AKS)**.
- Persistent Volume for PostgreSQL (AWS RDS preferred for managed setup).
- S3 bucket for model artifacts with IAM-based access.
- **GPU model serving** via Triton Inference Server integration.

Security

- **Role-based Access Control (RBAC)**: Model Owner / Reviewer / Consumer
- **AES-256 encryption** for model weights
- **SHA256 checksum validation** on upload
- **Audit trails** for every update or promotion event
- Integration with **MCP Server** for fine-grained access policies