# Project Report: AI Model Registry

**Student Name:** Adarsh Ramakrishna

**Student ID:** A00336108

**Date**: 26 November 2025

## 1. Introduction

The **AI Model Registry** is a custom Java application designed to help organizations track and manage their artificial intelligence software models. In the real world, companies have many different AI models (like text generators or image creators) and need a way to organize them. This application acts as a digital logbook. It allows users to register new models, store important details like who created them and when, and categorize them by type (Text, Image, or Audio).

I built this project to demonstrate my understanding of Object-Oriented Programming (OOP) using the latest features of **Java 21 and Java 22**. The goal was to create a system that is simple to use but technically advanced, ensuring data is safe and organized efficiently.

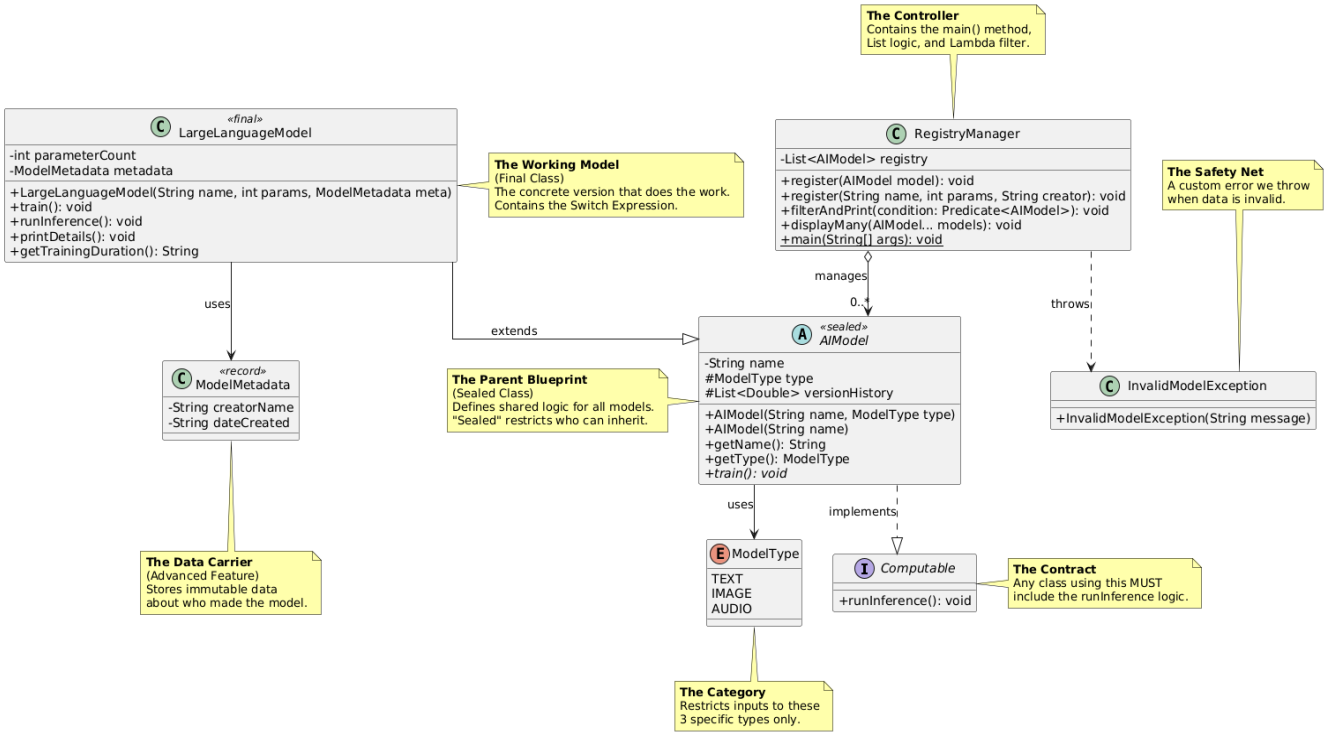
## 2. User Stories

The application was built to satisfy the following user requirements:

- **US-1:** As a user, I want to **register** a new AI model so that it is saved in the system.
- **US-2:** As a system administrator, I want to **prevent** models with invalid data (like negative parameter counts) from being entered.
- **US-3:** As a manager, I want to **view** a list of all models currently in the registry.
- **US-4:** As a developer, I want to **filter** the list of models to find specific ones (e.g., models with names starting with "G").
- **US-5:** As a user, I want to see the **training duration** automatically calculated based on the type of model (e.g., Text models take 2 weeks).

- **US-6:** As a user, I want to see **metadata** (creator name and date) attached to every model.

# 3. Architecture (UML Diagram)




# 4. Evaluation

## Adherence to the Project Brief

I believe this project successfully meets all the requirements set out in the assignment brief. I focused on building a "clean" application that uses modern Java features to solve real problems, rather than just writing code for the sake of it.

**Fundamentals Implementation** I covered all the core OOP concepts required:

- **Inheritance & Polymorphism:** I created a base class called AIModel that defines the shared rules. The specific class LargeLanguageModel extends this parent class. This allowed me to treat different models as the same type in my list.

- **Encapsulation:** All my class fields are private to protect the data. I used "Getters" to allow safe access to this data.

- **Interfaces:** I used the Computable interface to create a contract. This ensures that every model in the system promises to have a runInference() method.

- **Exceptions:** I handled errors in two ways. I created a custom checked exception (InvalidModelException) to stop users from entering negative numbers. I also used a try-catch block in the main method to keep the program running smoothly even if an error occurs.

- **Method Overloading:** In the RegistryManager, I created two versions of the register() method. One takes a full object, and the other takes raw data (strings and ints) and builds the object for you. This makes the code more flexible for the user.

**Advanced Language Features:**

1. **Records:** I used a Java Record for ModelMetadata. This was much simpler than writing a full class because records automatically handle all the boilerplate code (like constructors and toString) for data that doesn't need to change.

2. **Sealed Classes:** I made the AIModel class sealed. This is a security feature that strictly controls inheritance. It ensures that only authorized classes (like LargeLanguageModel) can extend my system, preventing unauthorized or "hacker" classes from breaking the rules.

3. **Switch Expressions:** Instead of a long, clunky switch statement with break keywords, I used the modern Switch Expression in the getTrainingDuration() method. It allows me to return a value directly based on the ModelType Enum, making the code much easier to read.

4. **Lambdas & Predicates:** I implemented a filtering system using Java's Predicate interface. This allows me to pass a small piece of logic (a lambda) into a method, such as asking the system to "find all models that start with the letter G."

**Java 22 Features:** I included the "Unnamed Variable" feature from Java 22. In my catch block, I used the underscore (_) instead of a variable name like e. This tells the compiler that I am catching a generic exception, but I do not intend to use the variable, which keeps the code cleaner.

# 5. Source Code Repository

To demonstrate version control and adherence to the "Extra Marks" criteria, the complete source code and history of updates for this project are hosted publicly.

**GitHub Repository:** https://github.com/adarsh6980/AI-Model-Registry-Using-JAVA/tree/main

## Problems Encountered & Solutions

During the development, I faced a few challenges:

**1. The "Sealed" Class Logic:** At first, I didn't fully understand how Sealed Classes worked. I tried to create a subclass without adding it to the permits list, and the compiler gave me an error. I learned that Sealed Classes act like a "VIP List"—if a class isn't on the list, it can't come in. This helped me understand how to strictly control my application's architecture.

**2. Exception Handling Flow:** I initially placed my "Filter" code after the line that triggers an error (the "BadModel" registration). Because the Exception stops the flow of the program immediately, my filter code never ran, and I couldn't see the output. I realized that the try block stops execution the moment it hits a snag. I fixed this by re-ordering the code in the main method, placing the successful operations (like filtering) *before* the intentional error test.

**Conclusion** Overall: The **AI Model Registry** is a robust, modern Java application. It effectively combines fundamental programming rules with features to create a secure and flexible tool for managing data.