**Title: AI-Enhanced Plagiarism Detection Using Adaptive Neural Networks**

**UML Diagrams**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**GOALS:**

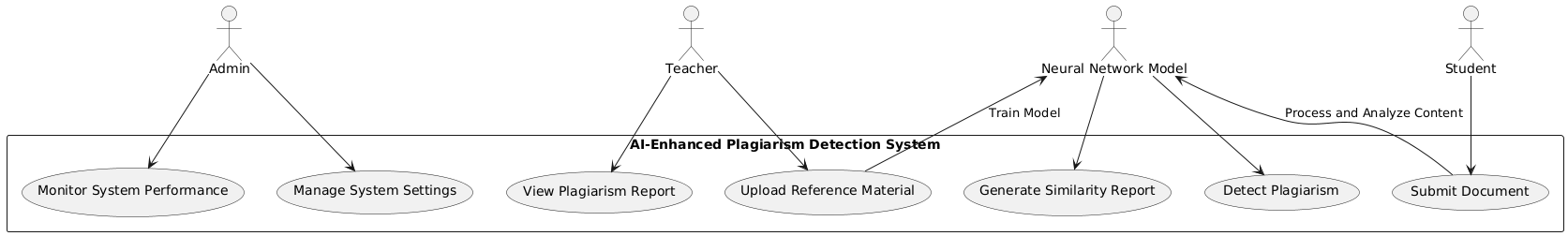
The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.

7. Integrate best practices.

**USE CASE DIAGRAM**

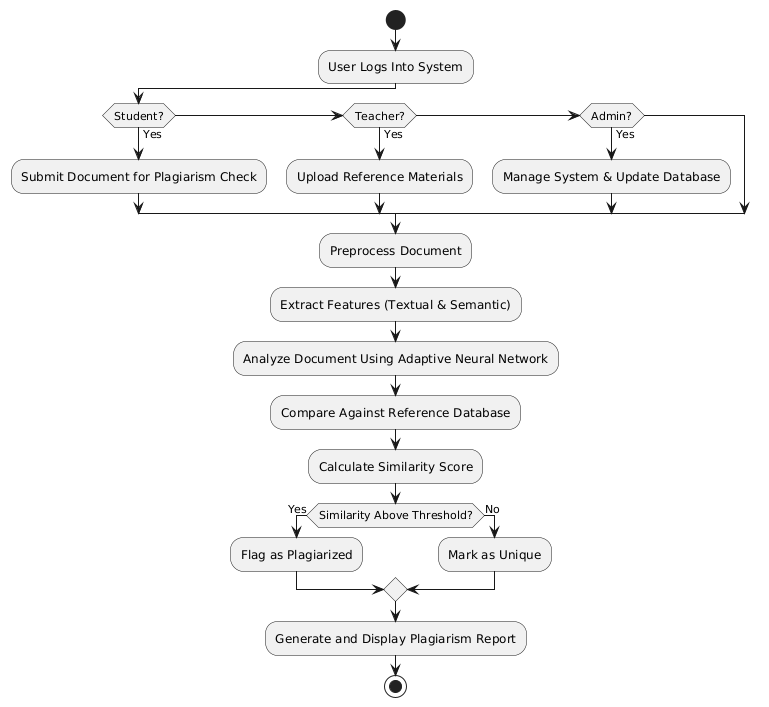
A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



**FIG: Use case Diagram**

**Activity diagrams**

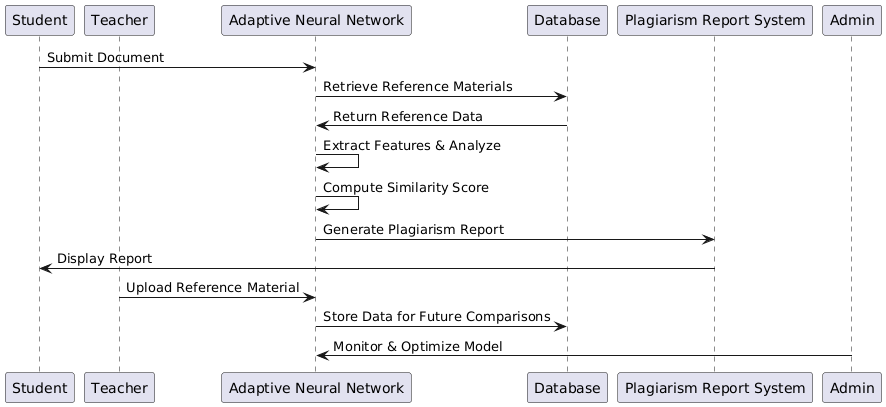
Activity diagrams are graphical representations of workflows of stepwise activities and actions[1] with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities.[2][3] Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores.[citation needed]Activity diagrams are graphical representations of workflows of stepwise activities and actions[1] with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities.[2][3] Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores.[citation needed]



**FIG: Activity Diagram**

**SEQUENCE DIAGRAM**

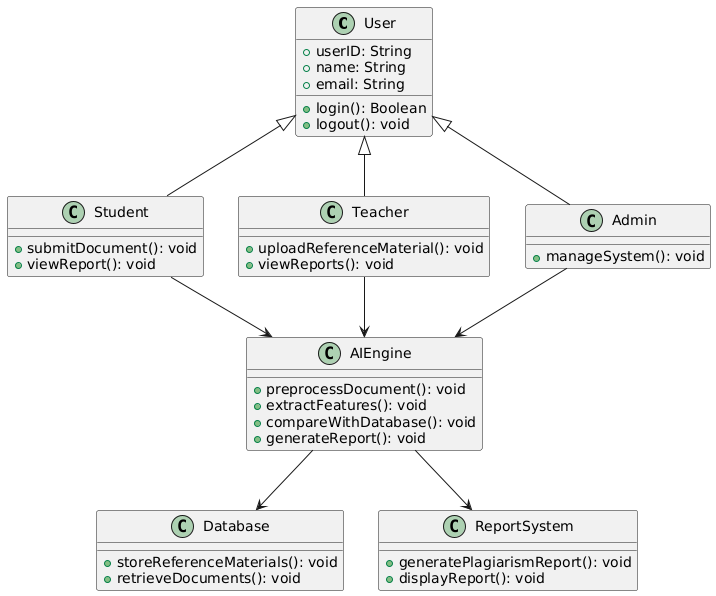
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



**FIG: Sequence Diagram**

**CLASS DIAGRAM**

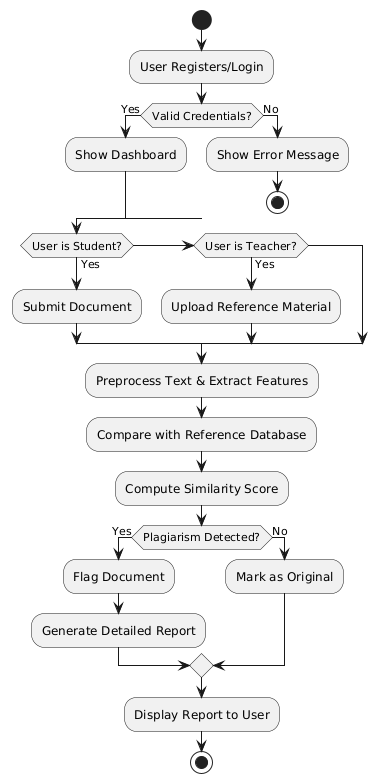
In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.



**FIG: Class Diagram**

**Flowchart diagram:**

A flowchart is a diagram depicting a process, a system or a computer algorithm. It is a diagrammatic representation of the solution to a given problem but, more importantly, it provides a breakdown of the essential steps to solving the problem. When designing and planning a process, flowcharts can help you identify its essential steps and simultaneously offer the bigger picture of the process. It organizes the tasks in chronological order and identify them by type, e.g. process, decision, data, etc. Each step is independent of implementation as the flowchart only describes what should happen at that step, what input is needed and what the output of the step is but it says nothing about how to implement the step.



**FIG: Flowchart Diagram**

**Component Diagram**

The component diagram extends the information given in a component notation element. One way of illustrating the provided and required interfaces by the specified component is in the form of a rectangular compartment attached to the component element.[2] Another accepted way of presenting the interfaces is to use the ball-and-socket graphic convention. A provided dependency from a component to an interface is illustrated with a solid line to the component using the interface from a "lollipop", or ball, labelled with the name of the interface. A required usage dependency from a component to an interface is illustrated by a half-circle, or socket, labelled with the name of the interface, attached by a solid line to the component that requires this interface. Inherited interfaces may be shown with a lollipop, preceding the name label with a caret symbol. To illustrate dependencies between the two, use a solid line with a plain arrowhead joining the socket to the lollipop.[3]

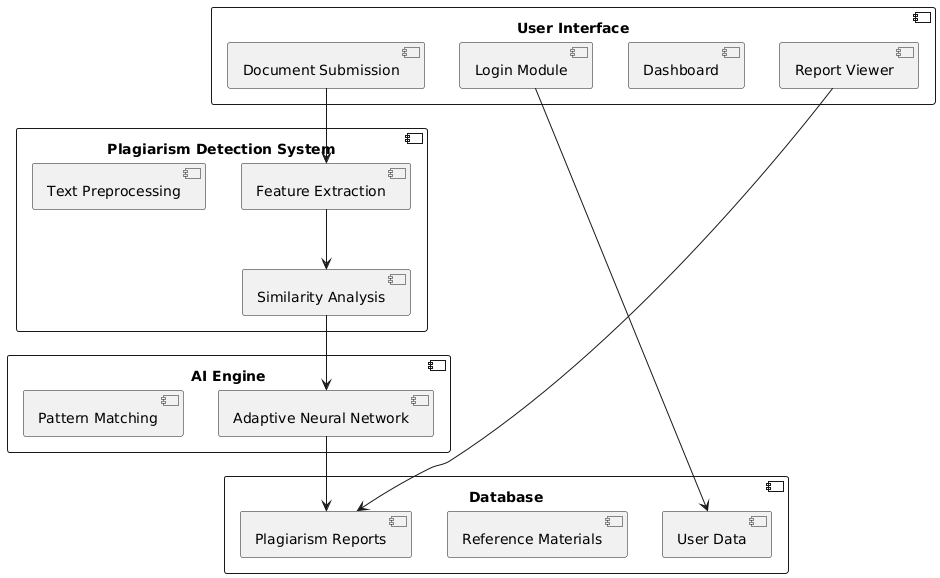
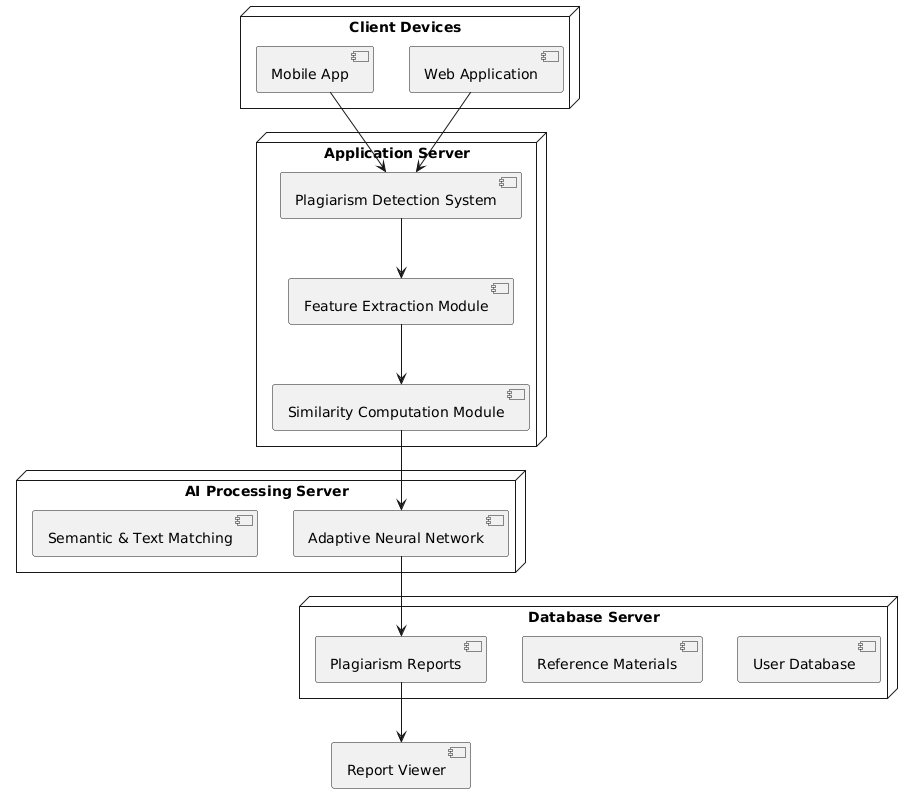


FIG: Component Diagram

**System Architecture**



**FIG: System Architecture**