UML DIAGRAM

UML (Unified Modeling Language) is a well standardized language for specifying, visualizing, constructing, and documenting the elements of software systems. UML [1], the Universal Modeling Language, was created by Object Management Group (OMG) and the first draft of UML 1.0 specification was submitted to OMG in January 1997. UML is not a programming language (just as C++, Java or COBOL, not what you program with) compared to regular programming languages. Rather, it is a visual language to use in defining Software design sketches.

The UML is generally a versatile and systematic visual modeling language for the purpose of systematizing the visualization, specification construction and documentation of software systems. While its primary use is in modeling software systems, UML's applications are not limited to this domain. It can also model and reason about processes in a wide range of situations including non-software such as manufacturing unit processes fora.

Although UML is not a programming language, it can be used with tools that generate code in several other programming languages from the UML diagrams. The UML consists of nine core diagrams to represent different facets of a system.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Activity diagram
- State chart diagram
- Deployment diagram
- Component diagram

4.2.1 USE CASE DIAGRAM

A use case is a tool you use to make sense out requirements in to something more manageable for example product delivery website or creating one. The above tools are visualized using Unified Modelling Language (UML) Represented by use case diagrams for an established way to model real-world things and solutions.

A Use Case Diagram has these major elements:

- The boundary that separates the system from its environment.
- Actors — human actors playing different roles in the system.

- Inter actions of different parties or factors in certain situations or problems.
- The main goal of use case diagrams is to clarify functional specifications of a system.

Few Tips that Can Be Followed While Making Use Case Diagram as Effective:

- To give clear, understandable names to Use Cases and Actors.
- Ensuring Well Defined Relationships and Dependencies.
- Include only the necessary relationships to keep the diagram sane.
- Use explanatory notes when needed for the important details

HL: use explanatory notes to explain details required.

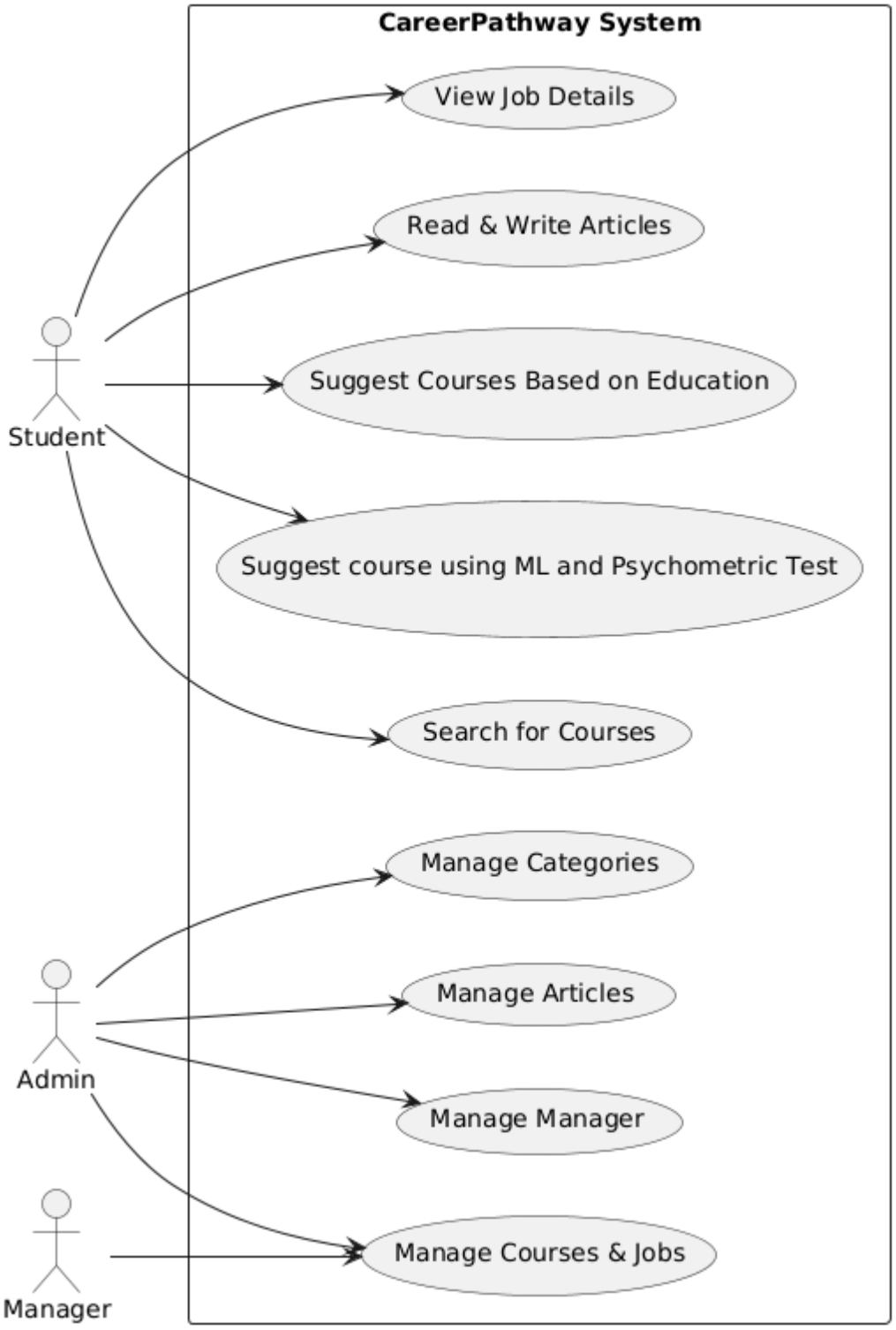
CareerPathway System
View Job Details
Read & Write Articles
Suggest Courses Based on Education
Suggest course using ML and Psychometric Test
Search for Courses
Manage Categories
Manage Articles
Manage Manager
Manage Courses & Jobs
Student
Admin
Manager

Fig 4.2.1

4.2.2 SEQUENCE DIAGRAM

A sequence diagram depicts the order in which objects talk to each other (which shows a progression of events from top to bottom). This diagram is also called an event diagram or event scenarios. Sequence diagrams aims to clarify with the interrelationship of many parts in the system, and their precise sequence of execution. Business, as well software professionals use these diagrams to both scratch out requirements of new and existing systems in explanation and visual form.

Sequence Diagram Notations:

i. Actors — Actors in UML diagram, are people who use a system, and its parts. The actors are not (represented in a UML diagram, as they are exterior to the modeled system). Also they are peeps whose story it will be since they are role-players in the story, i.e people and external entities. In UML diagram the actors are represented by whatever simple stick figures. You can have multiple people in a diagram showing the flow of events but as one goes to one specific incident.

ii. Lifelines — lifeline to each element of a sequence diagram is represented on diagram by a lifeline with the lifeline components in the top of figure.

iii. Messages – Messages exchanged between objects are organized on the lifeline as messages take place one after another in sequence. Messages are represented by Arrows that form the basis structure of a Sequence diagram.

iv. Guards: Guards (also within the UML) are used to represent different types of conditions On them messages may be restricted if certain conditions hold and give you hints about the rules which are in a system / process.

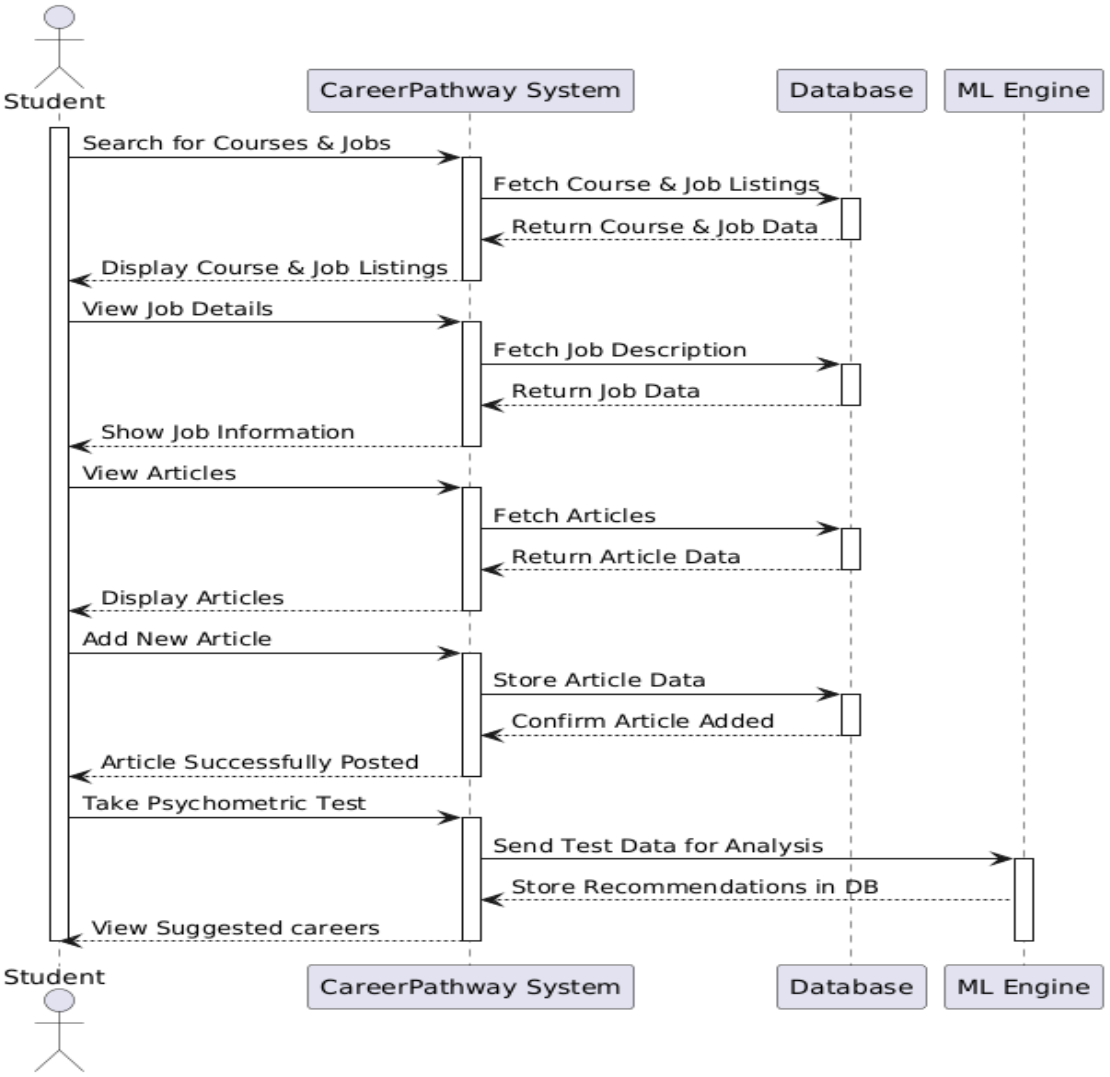

Fig 4.2.2

4.2.2 State Chart Diagram

A State Machine diagram (also called state chart) being a pictorial representation of the states an object can be into within a system and the path through these states. It is a trace of the behavior of the system, showing the shared behaviors of a group of entities (be it team, students in one class, group audience or even the whole organization). State machine diagrams are a helpful means for illustrating interactions of different participating components within a system detailing how objects change in response to events and make clear different states in which each and every entity or component can be.

A state machine diagram covers the following Notations within:

- Initial State : Black Circle initially: Initial state-demonstrated through black circle as an umbrella state for process.
- Result state: This is shown with a filled-in circle within a circle to represent the end of a process.
- Decision box: A diamond shaped element that provides a decision to make based on guard's

evaluation.

- Transition: Transition is when the switch in authority/state due an event Transitions are shown as labeled circling arrows that indicate the event triggering the transition.
- State box: Represents the element within a group of that state at a particular instant. They are usually rectangles with rounded corners such as.

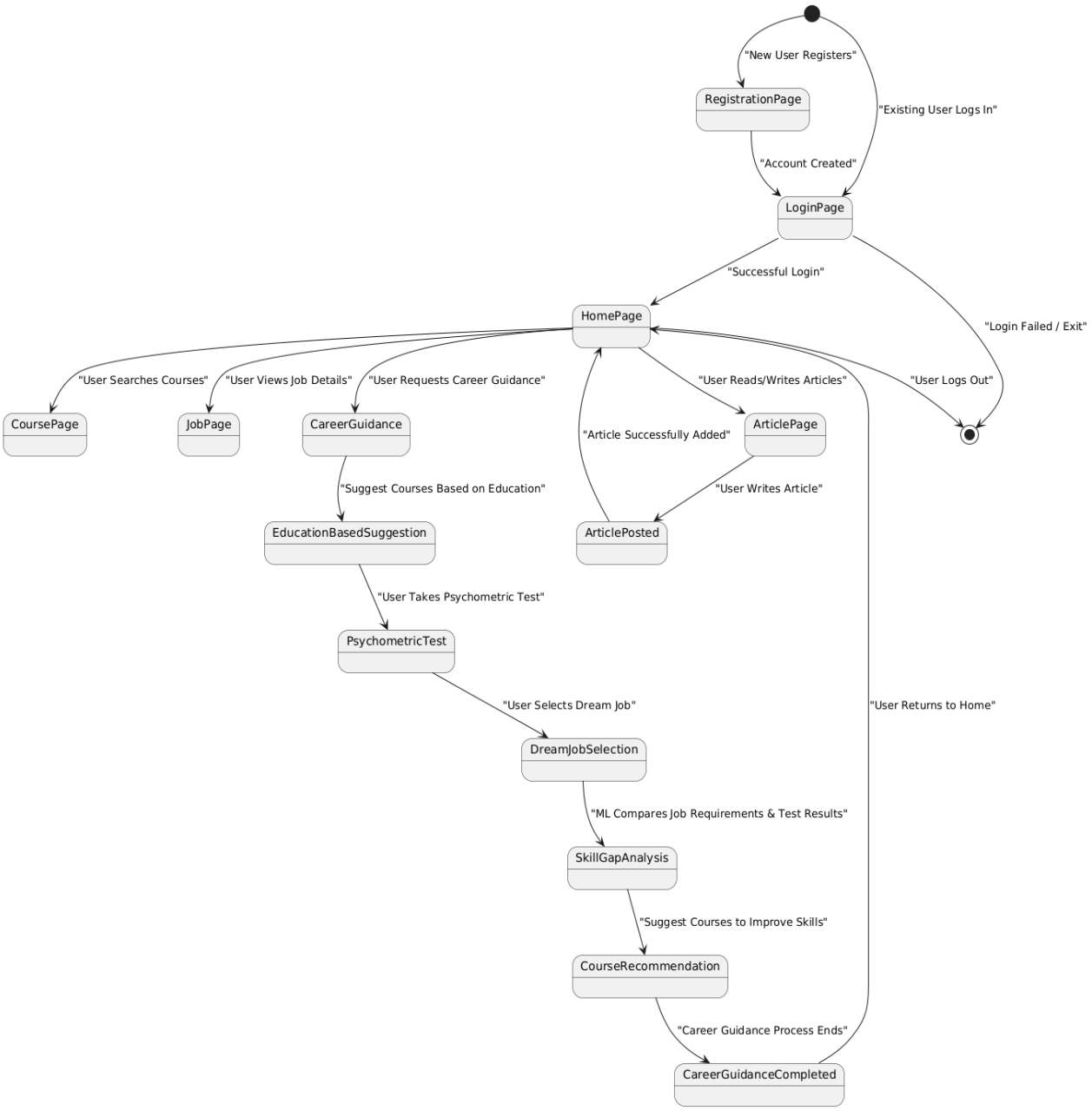

Fig 4.2.3

4.2.4Activity Diagram

An activity diagram illustrates the shared and sequential progression of events in system lifecycles. Helps to know the activities happening, making it clear one task going to another activity Task

sequences is the main theme of activity diagrams and it can model different kinds of flows such as sequential, parallel and alternative paths for this. To support these flows in activity diagrams, forks and join nodes are used which also complements the view to show workload of a system that the system is designed to illustrate more precisely.

Activity Diagram Key Components are:

a) Activities: Activities aggregate behaviors into one or more actions, so we have the network of different with steps interconnected. Actions that connect these steps are lines, the occurrences at that point are all the step-by-step breaks. Actions includes tasks, driving factors and process resources.

b) Activity Partition/Swim Lane :In the swim lane of an activity diagram similar tasks are categorized into rows or columns, but can only be arranged into columns not mandatory for every activity diaram. They can oriented in vertical or horizontal, although they are not all require in every activity diagram from mandatory.

c) Forks: Fork nodes allows to run of various branches of a partical task. They are the place, where one input turns into several outputs which is analogous to different factors in influencing a decision.

d) Join Nodes → Another kind of fork node which uses a Logical AND where all incoming data will converge to one place for synchronization is different from fork node. ..

Activity Diagram Notations are as follows

- Initial State — the starting or 1st step in a process
- Final State: Denotes the end of actions none further progress.
- Decision Box: To assure that our activities are routed correctly.
- An action: Task or activities to be executed on in the process.

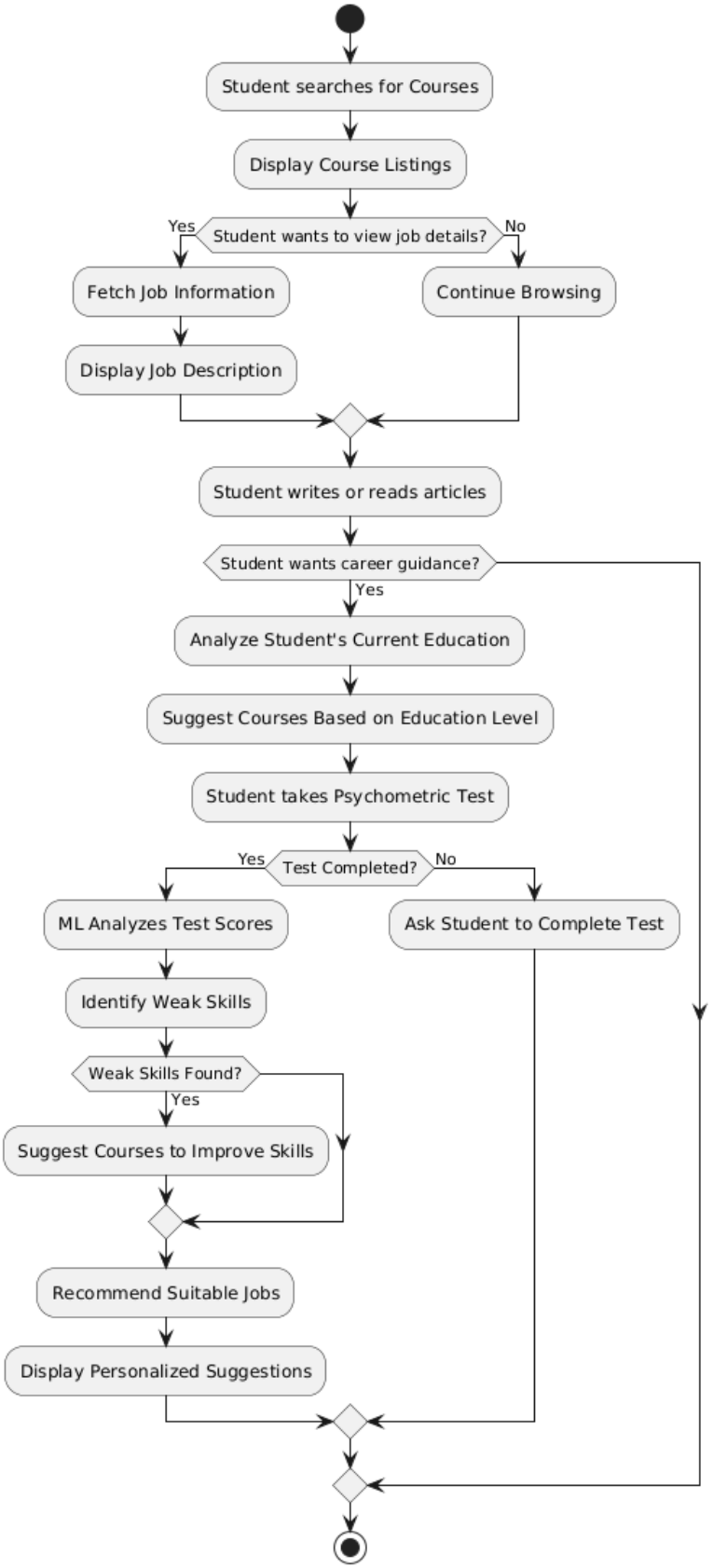
Student searches for Courses
Display Course Listings
Student wants to view job details?
Yes
No
Fetch Job Information
Continue Browsing
Display Job Description
Student writes or reads articles
Student wants career guidance?
Yes
Analyze Student's Current Education
Suggest Courses Based on Education Level
Student takes Psychometric Test
Test Completed?
Yes
No
ML Analyzes Test Scores
Ask Student to Complete Test
Identify Weak Skills
Weak Skills Found?
Yes
Suggest Courses to Improve Skills
Recommend Suitable Jobs
Display Personalized Suggestions

Fig 4.2.5

4.2.5Class Diagram

A class diagram is meant as a blueprint for the Application at Static State, depicting what components it will have and their relationship when system is actually down. It dives into the system design, displaying the different pieces the system is made from and their relationship. A class diagram, essentially is a recipe for developers as they build functional applications in software.

Important Elements of Class Diagram:

- System Overview: A class diagram is a high level representation of software system further defines its compile pieces, relationship and process collaborations. A good organizational structure for names, attributes and methods in software development.
- Structural visualization: The class diagram provides the structural view of system architecture which combines classes, associations and constraints.

Components of a Class Diagram:

A class diagram consisted partly of three major areas:

- Upper Section:Class name, a collection of objects that have similar properties, attributes and behavior. Capitalize initial letter of class name and place it at centre show abstract class title in slanted writing style are some guidelines to demonstrate groups of objects.
- Middle Section: This portion details the attributes of the class and their visibilities denoted public (+), private (-), protected (#), ~ as package.
- Lower Section: Lower segment goes into the methods or operations of a class, with each method in the list starting on a new line Of Class demonstrate its relationship with data.

In UML, relationships within a class diagram fall into three categories:

- Dependency: Signifying dependence of change in one element on change in another.
- Generalization :Representing a hierarchical relationship One class as general and another as specific.
- Association: Form evident connections between elements.
- Multiplicity: One is the default value for the number of instances which may

have certain characteristics develop a la list.

- Aggregation: An aggregation is a collection with a relationship available as association.
- Composition: Composition is a sort of aggregation, sort of It tells the parent needs its child to exist or rather one cannot work without the other.

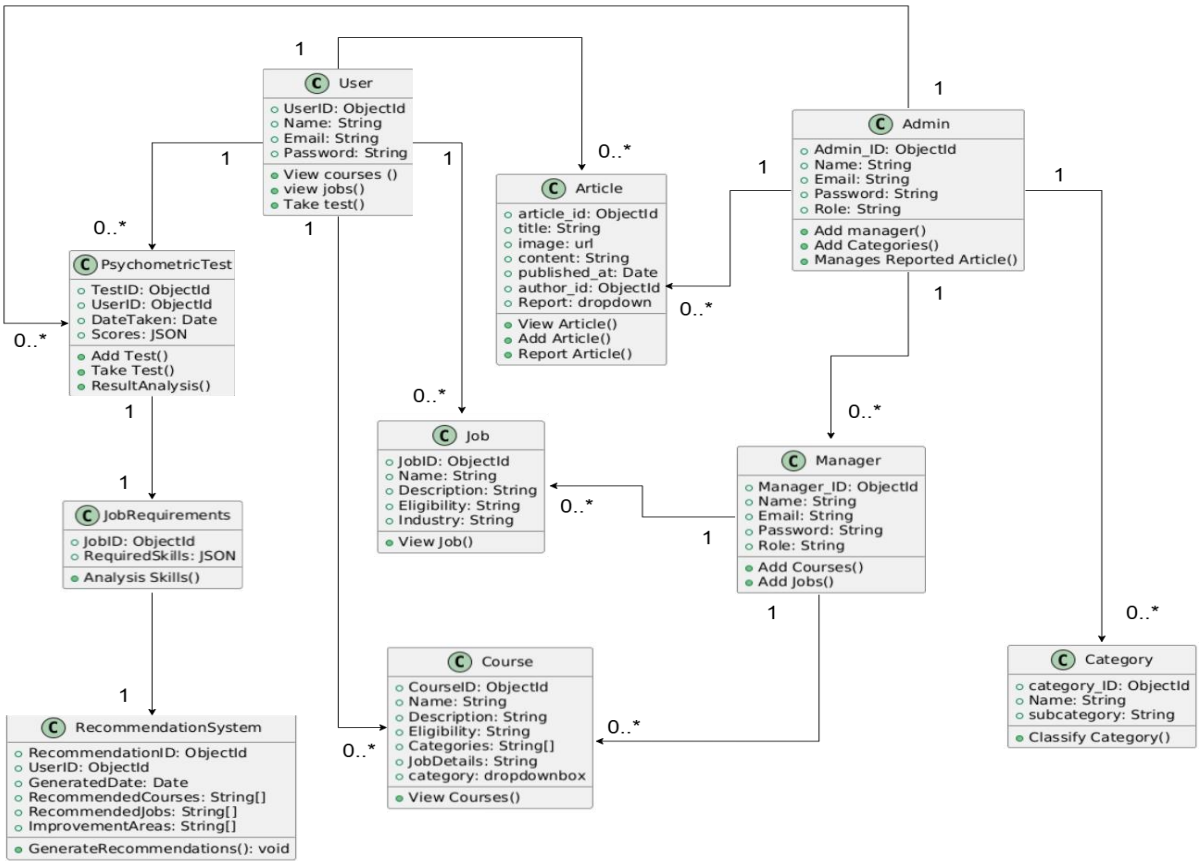

Fig 4.2.5

4.2.6 Object Diagram

Object diagrams are generated from class diagrams and depend on them for their visual. They give a picture of a set of objects in association to some class. An object diagram is simply a point in time snapshot of a object-oriented system. The differences and similarities between object diagrams and class diagrams While true, these have distinctions as well. Whereas a Class diagram is really pretty high level where it shows the classes as is. Class diagrams provide an easy to understand abstraction of system functionality and architecture that the workings of individual objects are hidden.

.

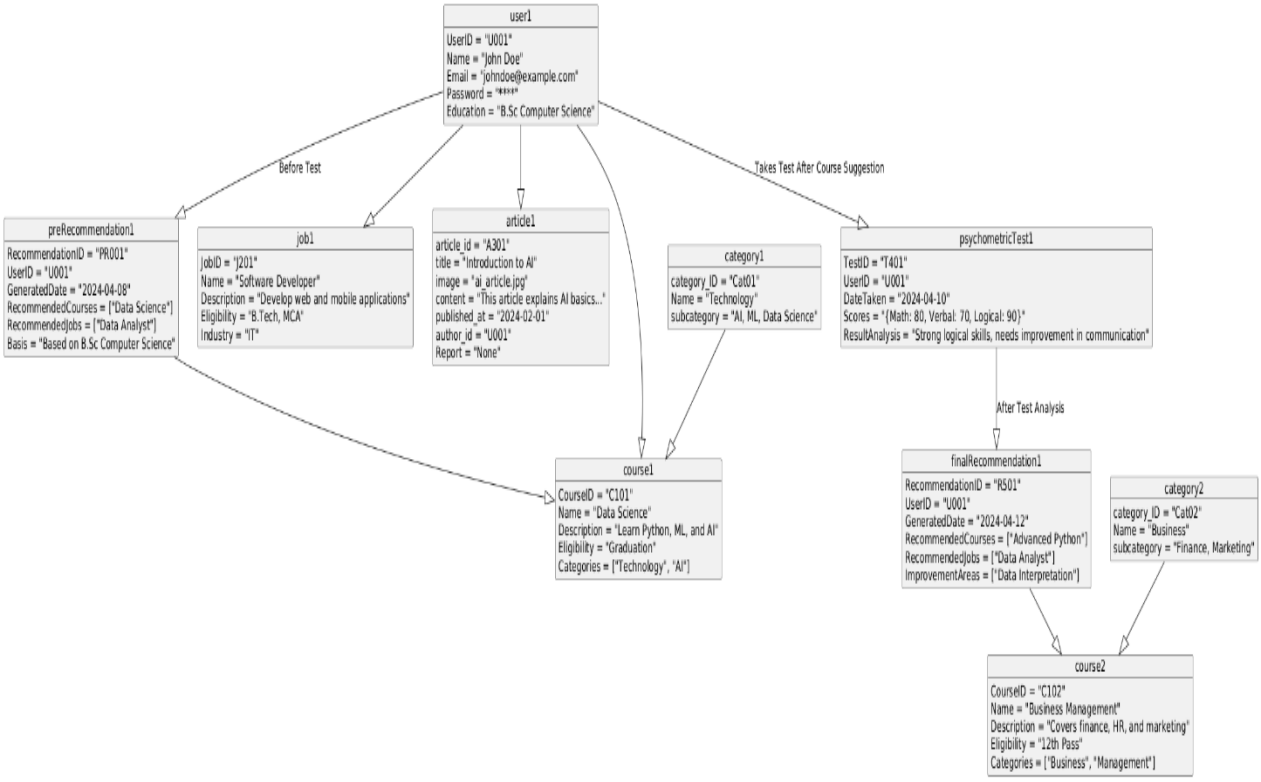

Fig 4.2.6

4.2.7Component Diagram

A component diagram is a tool to componentize large objects of a complex system. It provides a visual depiction of the system, in terms of internal to nodes systems like programs, documents and tools called the system.The associations and organization of the elements of system diagram gives working system Graphic.

Within a component diagram, the component is an interchangeable system unit part that can be varied and can function on its own. It keeps the curtain drawn on its inner workings and needs a special way to do its job, much like a magic concealed box that really only works properly when operated.

Related Notations for a Component Diagram:

- A component
- A node

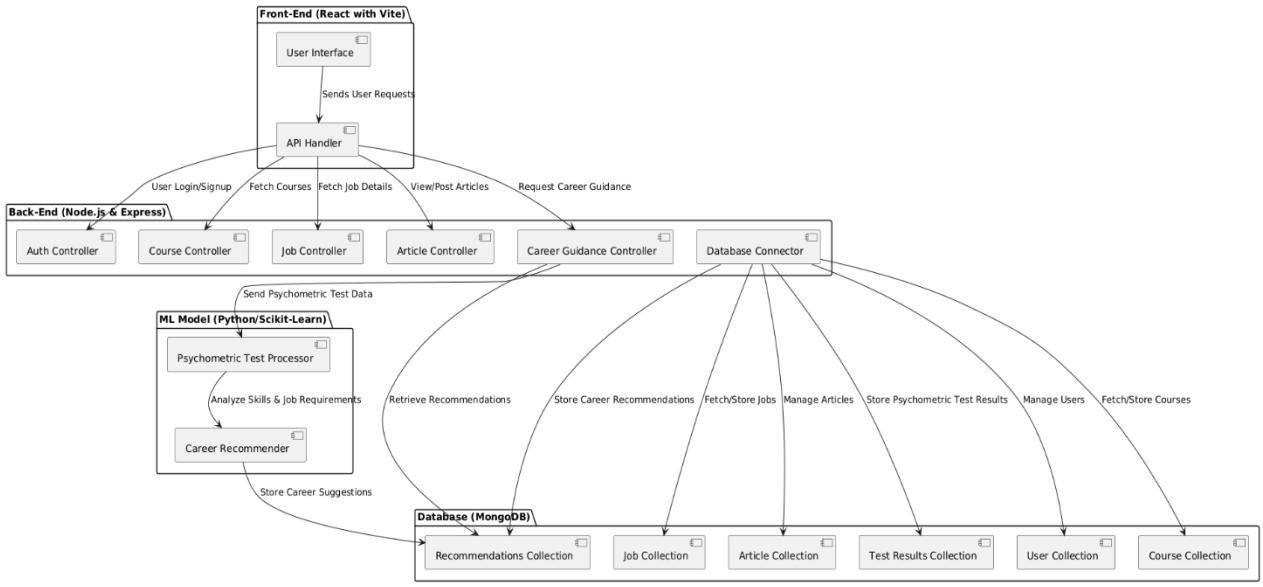

Fig 4.2.7

4.2.8 Collaboration Diagram

A collaboration diagram is a collaboration tool to depict the relationships between objects to a system. It is another way of saying the same thing than a sequence diagram, though. A collaboration diagram creates a snapshot of the object structure within system, rather than show the flow of messages between objects. These are based on the notion of object-oriented programming where object have attributes and are related to each other. Collaboration diagrams more or less represent the object architecture of system in a visual format.

Key elements in a collaboration diagram include:

- Objects: An object is devoted to their (s) names and class(s) (under-one), separated by a colon. Object: in a collaboration diagrams the objects are instances of a class; they will specify both the name and class of that instance. Not all classes are going to need an object representation and in fact, a class can have many different instances.
- Actors: These are pretty important in collaboration diagrams as they are the ones that initiate interactions.
- Links: Links are associations between object and actors, in the form of an instance It shows the relationships between the objects through which messages are passed between objects. An object navigates through links (solid lines) to reach other objects.
- Messages: Objects communicate via messages (object-to-object communication including an information carried in it and have a sequence number to uniquely identify them.) Such messages are then shown as labeled arrows next to the corresponding links, from sender to

receiver. The messages must be directed and the receiver must know that specific message.

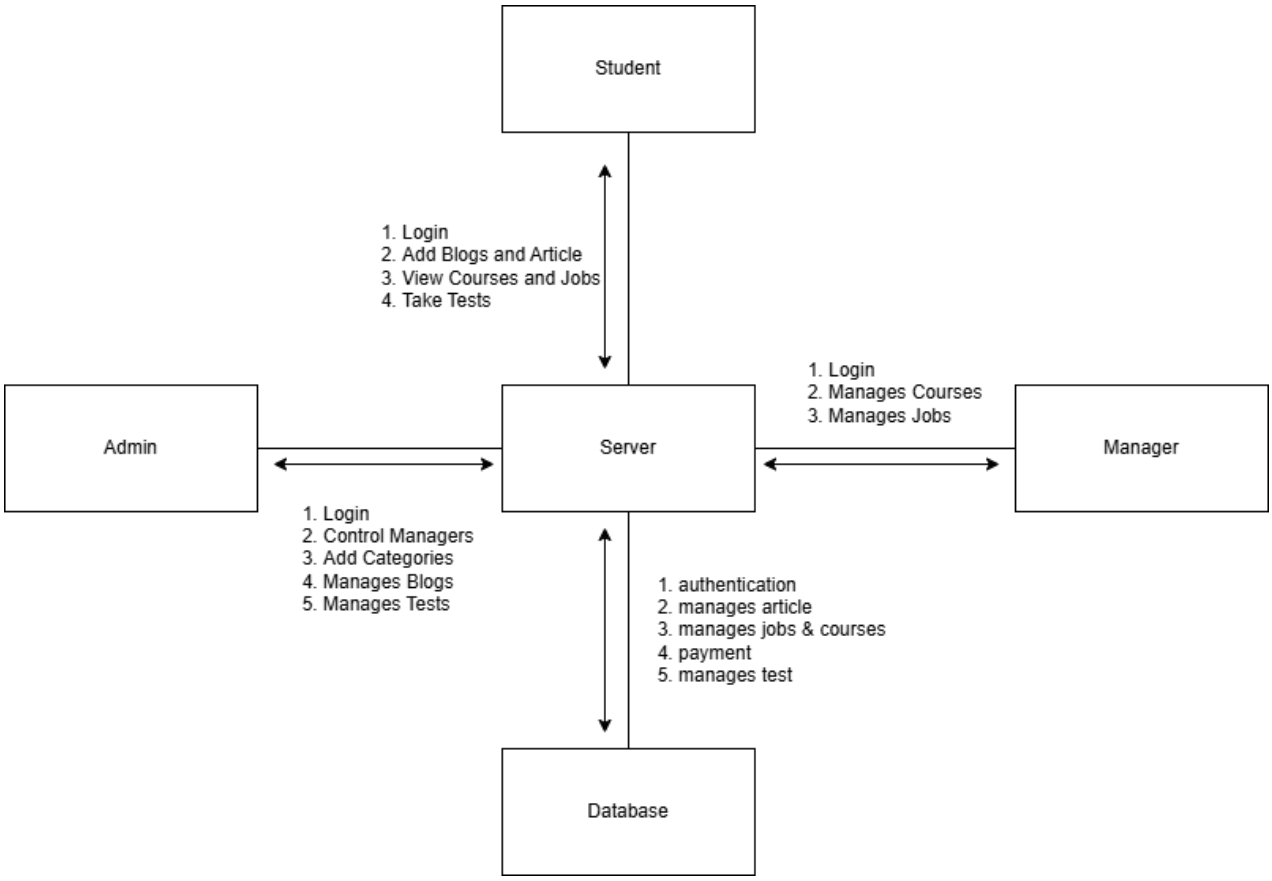

Fig 4.2.8

4.2.9Deployment Diagram

The deployment diagram shows where software goes in the physical machines or servers as a diagram. It provides the static image of the system in the nodes and its connectivity. This diagram delves into the mechanics of programs being put on computers, offering what the software is built this way to sync up with physical machine system.

Deployments Diagram Notations are the likes:

- A component
- An artifact
- An interface
- A node

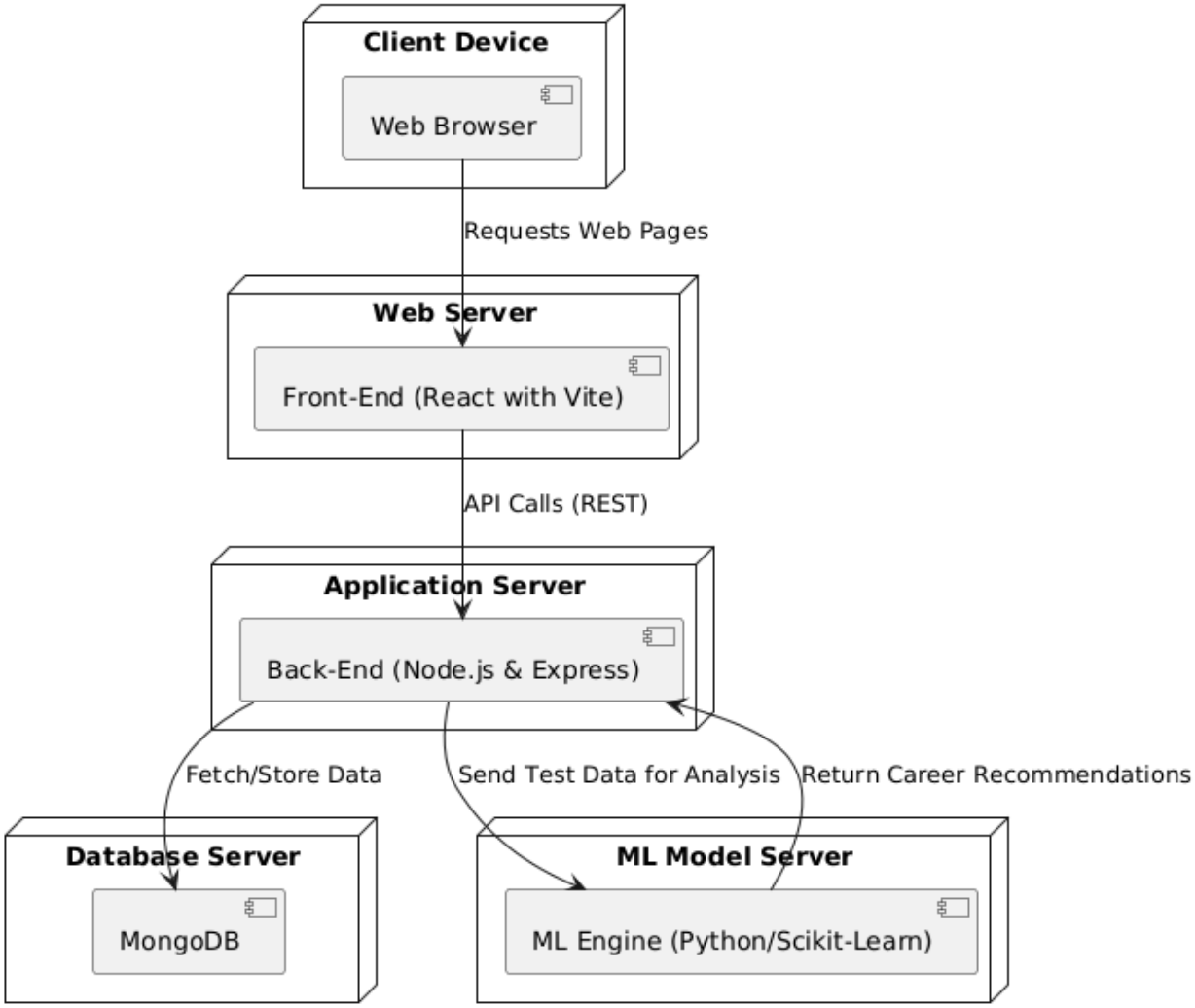
Client Device
Web Browser
Requests Web Pages
Web Server
Front-End (React with Vite)
API Calls (REST)
Application Server
Back-End (Node.js & Express)
Fetch/Store Data
Send Test Data for Analysis
Return Career Recommendations
Database Server
MongoDB
ML Model Server
ML Engine (Python/Scikit-Learn)

Fig 4.2.9