**EXP.NO:4**

**DATE:**

**Identify The Conceptual Classes and Develop Domain Model and Also**

**Derive a Class Diagram for Intelligent Team Formation System**

**AIM:**

To identify the conceptual classes, associations, and multiplicities of classes, and to develop a domain model using a UML class diagram for the Intelligent Team Formation System.

**PROCEDURE:**

The domain model is created by identifying the key concepts (classes) involved in the system and their relationships. The conceptual classes are derived from actors, use-cases, and important nouns found in the problem description. A UML class diagram is used to visualize the system structure.

**CLASSES:**

The following table lists the identified conceptual classes along with their attributes and functions:

|  |  |  |
| --- | --- | --- |
| **Class** | **Attributes** | **Functions** |
| **User** | userId, name, email, password, role | login(), logout(), viewProfile() |
| **Student** | userId, skills[], interests[], academicScore, availability | updateProfile(), joinTeam(), viewSuggestions() |
| **Mentor** | userId, expertise[], assignedTeams[] | assignTeam(), guideTeam(), viewTeams() |
| **Team** | teamId, teamName, members[], projectTitle, performanceScore | addMember(), removeMember(), evaluateTeam() |
| **Project** | projectId, title, description, requiredSkills[], domain | assignTeam(), updateStatus(), viewProgress() |
| **Skill** | skillId, skillName, proficiencyLevel | updateProficiency(), viewSkillLevel() |
| **TeamRequest** | requestId, studentId, teamId, status | sendRequest(), updateStatus(), cancelRequest() |
| **Feedback** | feedbackId, teamId, mentorId, comments, rating | submitFeedback(), viewFeedback() |
| **MatchingEngine** | students[], projects[], teams[] | suggestTeams(), matchSkills(), generateTeams() |

**RELATIONSHIPS**

**Dependency:**

* TeamRequest depends on Student and Team.
* Feedback depends on Mentor and Team.
* MatchingEngine depends on Student, Project, and Team data.

**Generalization:**

* Student and Mentor are subclasses derived from the User superclass.

**Association:**

* A Student can belong to one or more Teams.
* A Mentor guides multiple Teams.
* A Team works on one Project.
* A Project can be assigned to one or more Teams.
* A Team has multiple Feedback entries from Mentors.

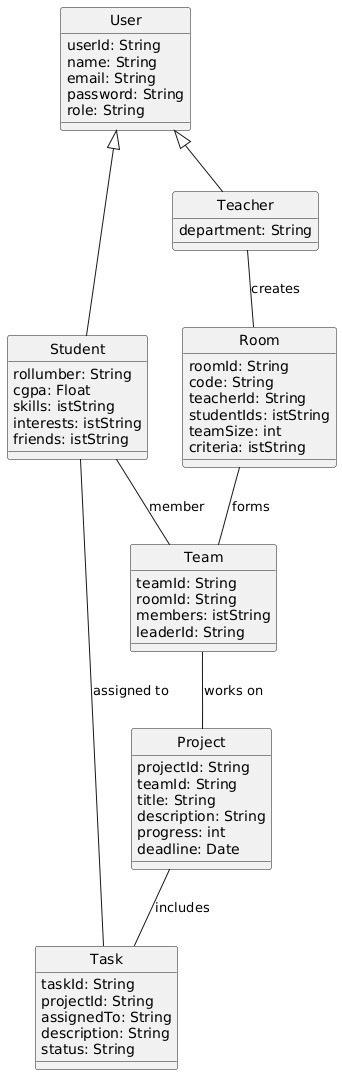
**Multiplicity:**

* One Student → many TeamRequests
* One Mentor → many Teams
* One Project → many Teams (collaborative)
* One Team → many Students
* One Feedback → one Mentor and one Team **Aggregation:**
* A Team is an aggregation of multiple Students.
* Project is associated with multiple required Skills.

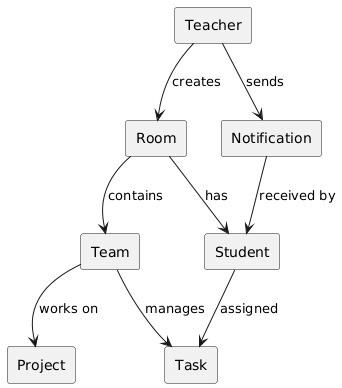
**Composition:**

* A Team cannot exist without being part of a Project.
* Deleting a Team removes its associated TeamRequests and Feedback records.

**CLASS DIAGRAM:**



**DOMAIN MODEL:**



**RESULT:**

The conceptual classes, associations, multiplicities, and relationships of the Intelligent Team Formation System were successfully identified. A domain model and corresponding UML class diagram were developed to clearly represent the system’s structure and behaviour.

**EXP.NO:5**

**DATE:**

**Using The Identified Scenarios, Find The Interaction Between Objects And**

**Represent Them Using Uml Sequence And Collaboration Diagrams**

**AIM:**

To draw UML Sequence and Collaboration diagrams for different scenarios of the Intelligent Team Formation System.

**PROCEDURE:**

**SEQUENCE DIAGRAM:**

A sequence diagram visually represents the flow of messages between various objects over time. It demonstrates how functionalities like team suggestion or mentor assignment occur step-by-step.

**ELEMENTS OF SEQUENCE DIAGRAM:**

* **Lifeline**: Denotes the participating objects such as Student, System, Matching Engine, Mentor, etc.
* **Actor**: A role initiating the interaction (e.g., Student or Mentor).
* **Activation**: A narrow rectangle that shows when an object is performing an operation.
* **Messages**: Arrows showing communication between objects in sequence.

**COLLABORATION DIAGRAM:**

A collaboration (or communication) diagram shows how objects are connected and interact to perform a task. Message numbers indicate the order of execution.

**COMPONENTS OF COLLABORATION DIAGRAM:**

* **Objects**: Named as objectName:ClassName (e.g., student:Student).
* **Links**: Solid lines connecting objects to show relationships.
* **Actors**: Initiators like Student, Mentor, or Admin.
* **Messages**: Numbered arrows indicating the sequence of interactions.

**COMPONENTS OF INTELLIGENT TEAM FORMATION SYSTEM:**

* **Student**: User who signs in, updates profile, and views suggested teams.
* **Mentor**: Reviews and guides assigned teams.
* **MatchingEngine**: Core logic that processes student data and forms teams.
* **Project**: Holds project details and required skills.
* **Team**: A group of students working on a specific project.
* **Admin**: Manages users, assigns mentors, and oversees system activity.
* **Database**: Stores users, teams, projects, and feedback.

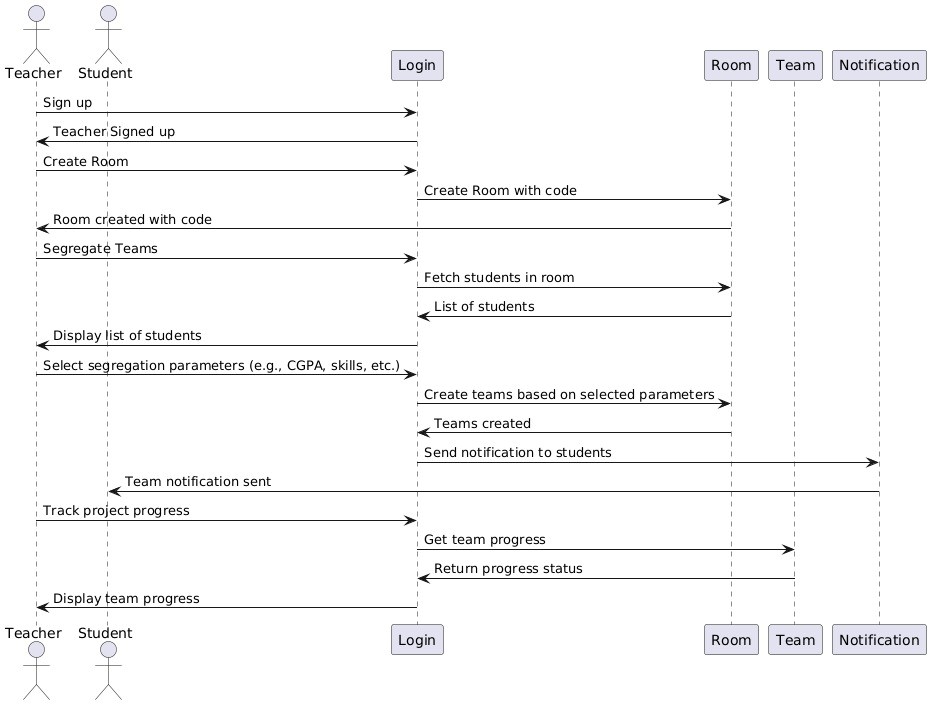
**WORKFLOW OF INTELLIGENT TEAM FORMATION SYSTEM:**

1. **Student logs in** to the system using valid credentials.
2. The **dashboard** displays options: Update Profile, View Suggestions, Join Team.
3. The **student updates profile** with skills, interests, and availability.
4. The **MatchingEngine** processes the data and **suggests compatible teams or**

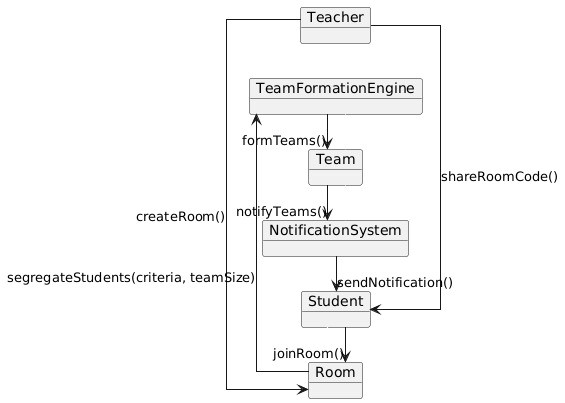
**forms new ones**.

1. Student **joins a team** or **sends a request** to join.
2. The **Mentor reviews and approves** teams or guides ongoing ones.
3. **Team details are updated**, and assigned **project information is displayed**.
4. Feedback is given at intervals by mentors.
5. The student logs out after completing actions.

**SEQUENCE DIAGRAM:**



**COLLABORATION DIAGRAM:**



**RESULT:**

The sequence and collaboration diagrams for the Intelligent Team Formation System were successfully developed. These diagrams visualize how key components like students, mentors, and the MatchingEngine interact to perform use cases such as team formation and project assignment.

**EXP.NO:6**

**DATE :**

**DRAW RELEVANT STATE CHART AND ACTIVITY DIAGRAM**

**AIM:**

To draw State Chart and Activity Diagrams for different scenarios of the Intelligent Team Formation System to understand the flow and behavior of the system during execution.

**PROCEDURE:**

**ACTIVITY DIAGRAM:**

Activity diagrams model the flow of control and logic within a system. They help in visualizing how users interact with the system and how tasks are executed.

**COMPONENTS OF ACTIVITY DIAGRAM:**

**Activities and Edges:**

* Activities represent user or system actions (e.g., login, update profile, form team).
* Edges show the flow between those actions.

**Activity Partition / Swimlane:**

Used to divide responsibilities among actors/components. Swimlanes used in this system:

* **Student**
* **Mentor**
* **Admin**
* **System/Database**

**Control Nodes:**

* **Fork Node**: Splits control into parallel actions (e.g., notify mentor and update DB).
* **Join Node**: Merges parallel flows.
* **Decision Node**: Represents conditional flow (e.g., team found or not).

**ACTIVITY DIAGRAM SCENARIO – Team Formation Workflow:**

**Flow:**

1. Student logs into the system
2. Views dashboard
3. Updates profile (skills, interests, availability)
4. Chooses "Find Team" option
5. System checks for compatible teams
6. [Decision] → If team exists:
   1. Show team suggestion
   2. Student joins team
   3. Mentor is notified
7. [Else] → System initiates new team formation
8. Matching Engine forms a team
9. Mentor is assigned and notified
10. Team information is stored in the database
11. Confirmation displayed to student
12. Student logs out
13. End of activity

**STATE CHART DIAGRAM:**

A state chart diagram shows the different states of an object (such as a student or team) and the transitions caused by events.

**STATE CHART COMPONENTS FOR INTELLIGENT TEAM FORMATION SYSTEM:**

**States:**

* **Idle State**: Initial state before login

* **Logged In**: After successful authentication

* **Profile Updated**: After user updates their data

* **Searching for Team**: System is processing compatibility

* **Team Suggested**: Team options shown to user

* **Team Formed**: New or existing team is joined

* **Mentor Assigned**: Mentor linked to the team

* **Logged Out**: Session ends

**Transitions:**

* **Idle → Logged In** (on successful login)

* **Logged In → Profile Updated** (on update profile action)

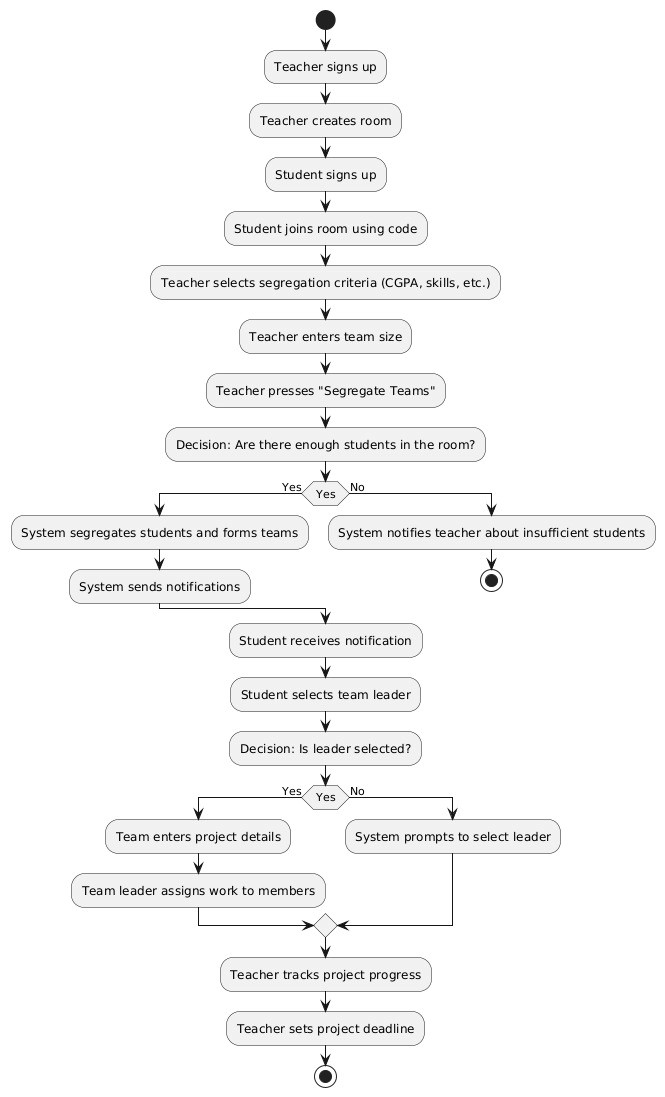
* **Profile Updated → Searching for Team** (on team search click)

* **Searching → Team Suggested** (if match found)

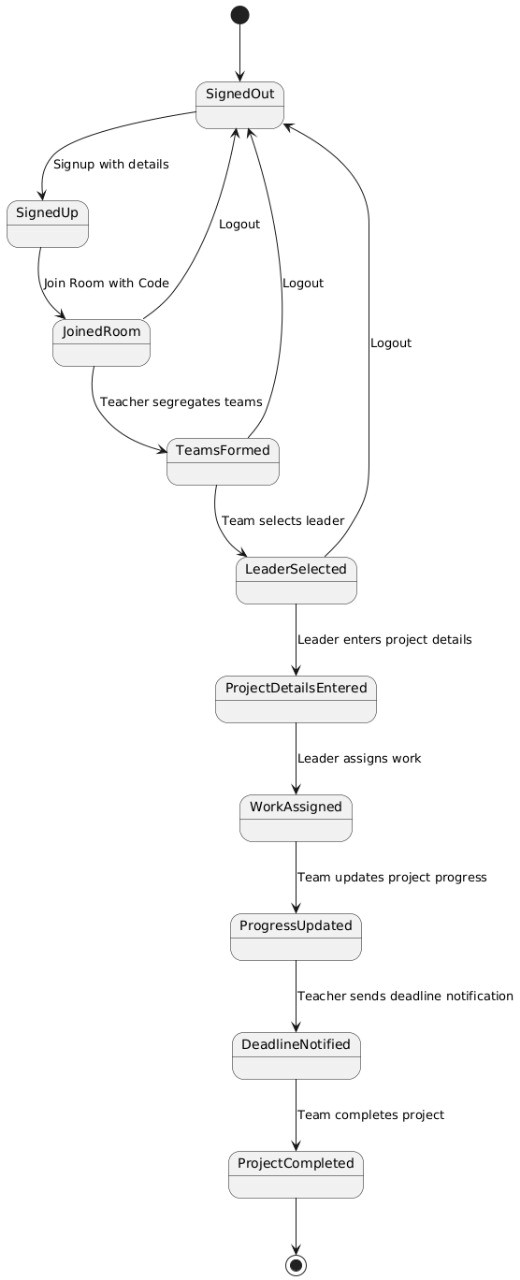
* **Team Suggested → Team Formed** (on student acceptance)

* **Team Formed → Mentor Assigned** (mentor added to team)

**ACTIVITY DIAGRAM:**



**STATE CHART DIAGRAM:**



**RESULT:**

The state chart and activity diagrams for the *Intelligent Team Formation System* were successfully developed .

**EXP.NO:7**

**DATE:**

**IMPLEMENT THE SYSTEM AS PER THE DETAILED DESIGN**

**AIM:**

To draw the **Package Diagram** and implement the system based on the detailed design for the **Intelligent Team Formation System**.

**PROCEDURE:**

**PACKAGE DIAGRAM:**

A package diagram is used to modularize and manage large systems. It depicts the dependencies and responsibilities among grouped classes, helping to maintain system clarity and reduce complexity.

**ELEMENTS OF PACKAGE DIAGRAM:**

● **Package:**

A logical grouping of related classes. Represented as a rectangle with a tab.

● **Class:**

A structure that defines the state and behavior of objects.

● **Interface:**

A collection of method signatures that define expected behaviors. Helps in abstraction.

● **Object:**

An instance of a class used during runtime or testing.

**PACKAGES IN INTELLIGENT TEAM FORMATION SYSTEM:**

1. **User Management Package**

○ **Classes:** User, Student, Mentor, Admin

○ Handles authentication, registration, and role-specific functionality

1. **Profile Management Package**

○ **Classes:** Profile, Skill, Interest

○ Manages user profiles including personal and skill details

1. **Team Formation Package**

○ **Classes:** Team, TeamMatcher, TeamBuilder

○ Responsible for intelligent team matching and formation logic

1. **Mentor Assignment Package**

○ **Classes:** MentorAssigner, MentorHandler

○ Manages mentor-team assignments and notifications

1. **Communication Package**

○ **Classes:** Notification, MessageService

○ Handles communication between students, mentors, and admins

1. **Database/Storage Package**

○ **Classes:** Database, DataAccessLayer

○ Manages data storage, retrieval, and persistence

**CLASSES USED IN INTELLIGENT TEAM FORMATION SYSTEM:**

* User, Student, Mentor, Admin
* Profile, Skill, Interest
* Team, TeamMatcher, TeamBuilder
* MentorAssigner, MentorHandler
* Notification, MessageService
* Database, DataAccessLayer

**INTERFACES:**

* **ITeamMatcher** – Defines contract for team matching algorithms
* **INotificationSender** – Defines behavior for sending alerts/notifications

●  **UserRoleManager** – Interface for handling user roles and access

**LAYERED ARCHITECTURE (Optional Visual Breakdown):**

1. **UI Layer:**

React.js interface for Student, Mentor, Admin dashboards

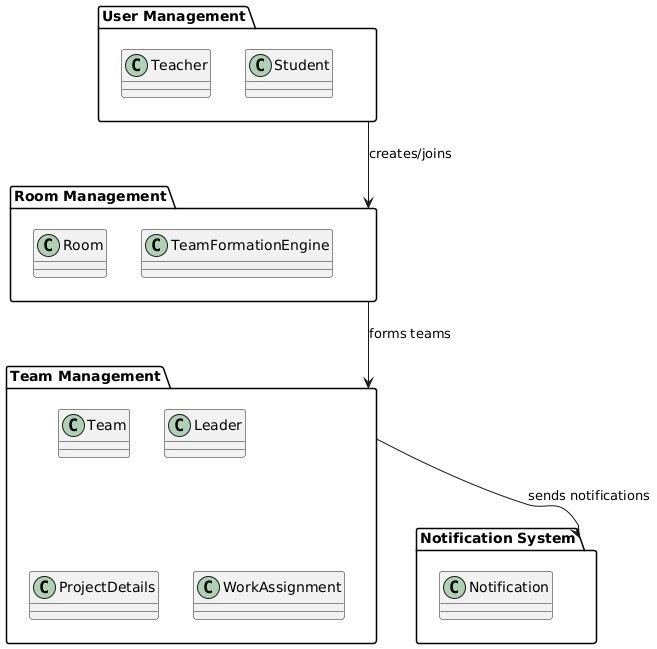
1. **Application/Domain Layer:**

Handles team formation, mentor assignment, and profile evaluation logic

1. **Data Access Layer:**

Implements Express.js APIs, MongoDB operations, and JWT authentication

**PACKAGE DIAGRAM:**



**RESULT:**

The **package diagram** and system design for the *Intelligent Team Formation System* were successfully created and implemented.

**EX. NO: 8**

**DATE:**

**TEST THE SOFTWARE SYSTEM FOR ALL THE SCENARIOS IDENTIFIED AS PER THE USECASE DIAGRAM**

**AIM:**

To develop code for the **Intelligent Team Formation System** and prepare for testing based on the identified use cases.

**PROCEDURE:**

Various test cases were designed and executed to verify the behavior of the Intelligent Team Formation System under different input conditions. Both **positive (success)** and **negative (failure)** scenarios were included to ensure robustness.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case**  **Name** | **Test Case ID** | **Description** | **Pre-Condi tion (Input**  **&**  **Conditions**  **)** | **Post-Condition (Actual Output)** | **Result**  **(Success / Failure)** |
| Login Case 1 | Login\_TC\_01 | To verify  credentials and redirect user to appropriate dashboard | Correct username and password provided | User is logged in and redirected based on role  (Student/Mentor/Admin  ) | Success |
| Login Case 2 | Login\_TC\_02 | To handle invalid login attempt | Incorrect username or password | Error message  displayed, login denied | Failure |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Signup  1 | Case | Signup\_TC\_0  1 | To create a new account with  valid details | Unique username, email, and valid password provided | Account created successfully and  redirected to login page | Success |
| Signup  2 | Case | Signup\_TC\_0  2 | To handle signup with duplicate or missing details | Existing username/e  mail or  incomplete fields | Error message shown, account not created | Failure |
| Profile  Update 1 | | Profile\_TC\_0  1 | To allow a user to update their profile with valid skills and interests | Logged-in user provides valid data | Profile updated and saved to database | Success |
| Profile  Update 2 | | Profile\_TC\_0  2 | To handle update with incomplete profile data | User leaves required fields blank | Validation error shown, update blocked | Failure |
| Team  Formation 1 | | Team\_TC\_01 | To form a team based on  matching algorithm | Multiple students with compatible skills and interests exist | System successfully forms optimal teams | Success |
| Team  Formation 2 | | Team\_TC\_02 | To test behavior when  insufficient  members are  available | Less than required number of students in pool | Team formation not completed, user shown message | Failure |
| Mentor  Assignment 1 | | Mentor\_TC\_0  1 | To assign a  mentor to a team | Team created and mentor available | Mentor successfully assigned to team | Success |
| Mentor  Assignment 2 | | Mentor\_TC\_0  2 | To check behavior when no mentor is  available | No mentors available in system | System shows notification, assignment deferred | Failure |
| Notification  Case 1 | | Notif\_TC\_01 | To notify students after successful team formation | Team formed successfull  y | Students receive team formation notification | Success |
| Notification  Case 2 | | Notif\_TC\_02 | To check behavior when notification fails due to invalid  recipient | Invalid or missing user ID | Error logged,  notification not sent | Failure |

**RESULT:**

The Intelligent Team Formation System was tested with various inputs. All major features worked as expected, with success and failure cases handled correctly.

**EXP.NO:9**

**DATE:**

**Improve The Reusability And Maintainability Of The Software System By Applying Appropriate Design Patterns**

**AIM:**

To redesign the code for the Intelligent Team Formation System by applying suitable design patterns to improve reusability and maintainability.

**PROCEDURE:**

**Model-View-Controller (MVC) Design Pattern**

* **Model:**

Represents the data and business logic of the application (e.g., user data, team preferences, skill sets).

* **View:**

Displays data to the user and sends user interactions to the controller (e.g., team creation interface, dashboards).

* **Controller:**

Handles user input, communicates with the model, and updates the view accordingly (e.g., validating input, forming teams).

**//App (Controller Layer) - App.jsx** import React from 'react'; import ReactDOM from 'react-dom/client'; import { BrowserRouter as Router, Routes, Route } from 'react-router-dom'; import './App.css';

import Homebar from './components/Homebar.jsx'; import Services from './components/User/Services.jsx'; import Food from './components/User/Food.jsx'; import Orders from './components/User/Orders.jsx'; import Complaints from './components/User/Complaints.jsx'; import Home from './components/Home'; import Dashboard from './components/User/Dashboard.jsx'; import Login from './components/Login.jsx'; import Register from './components/Register.jsx'; import RequireAuth from './components/RequireAuth.jsx'; import Unauthorised from './components/unauthorised.jsx'; import Manageusers from './components/Admin/Manageusers.jsx'; import PersistLogin from './components/PersistLogin.jsx'; import Managerooms from './components/Admin/Managerooms.jsx'; import Roomallocation from './components/Manager/Roomallocation.jsx'; import Waitingroom from './components/waitingroom.jsx'; import Manageservices from './components/Manager/Manageservices.jsx'; import Managecomplaints from './components/Admin/Managecomplaints.jsx'; import Managefood from './components/Manager/Managefood.jsx'; import Manageorders from './components/Manager/Manageorders.jsx'; const Roles = { user: 'customer', manager: 'manager', admin: 'admin'

}; function App() { return (

<Router>

<Homebar />

<Routes>

<Route path="/login" element={<Login />} />

<Route path="/register" element={<Register />} />

<Route path="/unauthorised" element={<Unauthorised />} />

<Route element={<PersistLogin />}>

<Route path="/" element={<Home />} />

<Route element={<RequireAuth allowedroles={[Roles.user]} />}>

<Route path="/services" element={<Services />} />

<Route path="/dashboard" element={<Dashboard />} />

<Route path="/food" element={<Food />} />

<Route path="/orders" element={<Orders />} />

<Route path="/complaints" element={<Complaints />} />

</Route>

<Route element={<RequireAuth allowedroles={[Roles.admin]} />}>

<Route path="/Managecomplaints" element={<Managecomplaints />} />

<Route path="/Manageusers" element={<Manageusers />} />

<Route path="/Managerooms" element={<Managerooms />} />

</Route>

<Route element={<RequireAuth allowedroles={[Roles.manager]} />}>

<Route path="/Roomallocation" element={<Roomallocation />} />

<Route path="/Manageservices" element={<Manageservices />} />

<Route path="/Managefood" element={<Managefood />} />

<Route path="/Manageorders" element={<Manageorders />} />

</Route>

<Route path="/waitingroom" element={<Waitingroom />} />

</Route>

</Routes>

</Router>

);

}

export default App; **// servicesModel.js** const services = [

{ id: 1, name: "Mentorship", available: true },

{ id: 2, name: "Coding Practice", available: false }

];

export function getAllServices() { return services;

}

export function toggleAvailability(id) { const service = services.find(s => s.id === id); if (service) service.available = !service.available;

}

**//services.jsx**

import React, { useState, useEffect } from 'react'; import { getAllServices, toggleAvailability } from '../../models/servicesModel';

const Services = () => { const [services, setServices] = useState([]);

useEffect(() => { setServices(getAllServices());

}, []);

const handleToggle = (id) => { toggleAvailability(id); setServices([...getAllServices()]);

}; return (

<div>

<h2>Available Services</h2>

<ul>

{services.map(service => (

<li key={service.id}>

{service.name} - {service.available ? "Available" : "Unavailable"}

<button onClick={() => handleToggle(service.id)}>Toggle</button>

</li>

))}

</ul>

</div>

); };

export default Services;

**RESULT:**

Thus, the code for the Intelligent Team Formation System was successfully redesigned using the MVC design pattern, enhancing its modularity, reusability, and maintainability.