# Knowledge-Based System for AIU CSE Course Registration Advising

## Overview

The Knowledge-Based System (KBS) for AIU CSE Course Registration Advising is designed to assist Computer Science and Engineering (CSE) students at Alamein International University (AIU) in selecting courses for registration. The system analyzes a student's semester, Cumulative Grade Point Average (CGPA), and course history (passed and failed courses) to recommend a tailored course load aligned with the Artificial Intelligence Science track. It ensures compliance with university policies, such as credit limits and prerequisites, and provides clear, user-friendly explanations for its recommendations.

## Team Formation

- **Size**: Teams can consist of up to 5 students. Each member must contribute to one or more project components, with responsibilities clearly divided based on team size.
- **Track Selection**: Each team must select one CSE track from the following list:
    1. Software Engineering
    2. Big Data Analytics
    3. Computer Vision
    4. Artificial Intelligence Science (chosen for this specification)
    5. Embedded Systems
    6. Cloud Computing
    7. High Performance Computing
    8. Cyber Security
    9. Artificial Intelligence Engineering
- **Registration**: Teams must register their members and selected track by the deadline specified in the course schedule. Late registration incurs a 5% deduction from the final project grade.
- **Uniqueness**: No two teams can select the same track, ensuring diversity across projects.

## Project Requirements

The KBS must integrate the following components seamlessly:

1. **Knowledge Base**
    - **Content**: Stores comprehensive course information and university policies in a structured format (e.g., CSV or JSON). Required fields include:
        - Course Code (e.g., CSE101)
        - Course Name (e.g., "Introduction to Programming")
        - Description (brief summary)
        - Prerequisites (comma-separated course codes, e.g., MATH101)
        - Co-requisites (comma-separated course codes, if any)
        - Credit Hours (e.g., 3)
        - Semester Offered (e.g., Fall, Spring, or Both)
    - **Policies**: Includes rules such as:

        - Credit limits based on CGPA (e.g., CGPA < 2.0: max 12 credits; 2.0 ≤ CGPA < 3.0: max 15 credits; CGPA ≥ 3.0: max 18 credits)
        - Prioritization of failed courses for retaking.

2. **Knowledge Base Editor**
    - **Functionality**: A tool for administrators to manage the knowledge base without altering the source code. It must support:
        - Adding a new course with all required fields.
        - Editing an existing course's details.
        - Deleting a course.
        - Viewing the current knowledge base (e.g., list all courses).
    - **Validation**: Ensures prerequisites and co-requisites reference existing courses and that credit hours are positive integers.
    - **Interface**: Can be command-line based or integrated into the Streamlit app as an admin section.

3. **Inference Engine**
    - **Purpose**: Generates course recommendations using rule-based reasoning. Implemented using Python's Experta library (or similar).
    - **Rules**: Must include at least the following:
        - **Credit Limit Rule**: Caps total credits based on CGPA (see above).
        - **Prerequisite Rule**: Only recommends courses if all prerequisites are passed.
        - **Co-requisite Rule**: Ensures co-requisites are either previously passed or included in the current semester's recommendations.
        - **Failed Course Priority Rule**: Prioritizes failed courses if prerequisites are met.
        - **Semester Availability Rule**: Recommends only courses offered in the upcoming semester (e.g., Fall or Spring).
        - **Track Requirement Rule**: Ensures recommendations align with the Artificial Intelligence Science track's required and elective courses.
    - **Output**: A list of recommended courses with total credit hours not exceeding the CGPA-based limit.

4. **User Interaction Module**
    - **Requirement**: A **mandatory web-based interface** built using Streamlit, a Python library for creating interactive web applications.
    - **Input Features**:
        - Text field or dropdown for the student's current semester (e.g., Fall 2024).
        - Numeric input for CGPA (0.0–4.0, with validation).
        - Multi-select dropdowns or text input to list passed and failed courses (e.g., from the knowledge base).
    - **Output Features**:
        - A table displaying recommended courses, including:
            - Course Code

- Course Name
- Credit Hours
- Total credit hours for the semester.

- Error messages for invalid inputs (e.g., "CGPA must be between 0.0 and 4.0").

- **Design**: Intuitive and visually appealing, with clear labels, instructions, and feedback (e.g., success messages after submitting inputs).
- **Integration**: Connects to the inference engine to process inputs and display results dynamically.

5. **Explanation System**
    - **Goal**: Provides transparent reasoning for recommendations and restrictions. Examples:
        - "CSE301 is recommended because you passed CSE201, its prerequisite."
        - "CSE402 is unavailable due to an unmet prerequisite, MATH204."
        - "CSE101 is prioritized because you failed it previously."
    - **Implementation**: Embedded in the inference process, with explanations linked to specific rules and knowledge base facts.
    - **Display**: Shown in the Streamlit interface as a dedicated section (e.g., a collapsible list or popup) alongside the recommendations.

## Technical Requirements

- **Programming Language**: Python, leveraging its compatibility with Experta (for the inference engine) and Streamlit (for the interface).
- **Dependencies**: Include required libraries (e.g., streamlit, experta, pandas for CSV handling) in a requirements.txt file.
- **Compatibility**: Must run on standard lab computers with clear setup instructions if additional dependencies are needed.
- **Error Handling**: Handle edge cases, such as:
    - Invalid CGPA input.
    - No courses available due to unmet prerequisites or credit limits.
    - Missing or malformed knowledge base data.

## Collaboration and GitHub Contribution Guidelines

- **Objective**: Ensure effective teamwork and teach version control skills using GitHub. Individual contributions are a key evaluation criterion.

- **Requirements**:
    - Each team must maintain a GitHub repository for the project.
    - Commits must reflect meaningful contributions (e.g., no empty commits or bulk file uploads without context).
    - Suggested component assignments (adjust based on team size):
        - Student 1: Knowledge Base and Editor
        - Student 2: Inference Engine (rules and logic)
        - Student 3: Streamlit Interface (input/output design)
        - Student 4: Explanation System
        - Student 5: Integration, testing, and documentation
    - Each student must commit their work regularly, with descriptive messages (e.g., “Added prerequisite rule to inference engine”).

- **Best Practices**:
    - Use branches for feature development (e.g., feature/knowledge-base, feature/streamlit-ui).
    - Merge via pull requests with team reviews to ensure quality.
    - Avoid large, monolithic commits; break work into smaller, logical updates.

- **Evaluation**: Individual grades will partly depend on GitHub commit history, assessing:
    - Frequency and consistency of contributions.
    - Relevance and quality of work committed.
    - Collaboration (e.g., pull request comments, issue tracking).

## Deliverables

Submit all components as a single package by [insert submission date]:

- **Final Report**: A detailed document including:
    - Project title, team members, and their assigned roles.
    - Application Description: Overview of the system, its functionality, and the domain modeled.
    - Rules: List and explain all implemented rules with examples.
    - Knowledge Base: Describe content and editor functionality, including sample entries.
    - Inference Engine: Detail the reasoning process and implementation.
    - User Interaction: Simulate a full Streamlit interaction (inputs, outputs, screenshots).
    - Explanation System: Explain how explanations are generated and displayed.
    - Streamlit Interface: Describe design, features, and implementation.

  - GitHub Collaboration: Summarize team workflow and link to the repository.
- **Source Code**: All files, including:
  - Knowledge base file (e.g., courses.csv).
  - Python scripts for the KBS components.
  - Streamlit app code (e.g., app.py).
  - Well-commented code following good practices.
- **Demonstration**: A video (max 5 minutes) showing:
  - Launching the Streamlit app.
  - Entering sample student data.
  - Displaying recommendations and explanations.
  - Handling an edge case (e.g., low CGPA or failed course).

## Grading Rubric (100 Points)

- **Requirements and Domain Understanding (15%)**: Clarity and completeness of the application description, rules, and interaction simulation.
- **Knowledge Base and Editor (20%)**: Completeness and functionality of the knowledge base and editor.
- **Rules and Inference Engine (25%)**: Correctness and effectiveness of rules and reasoning.
- **User Interaction and Explanation (20%)**: Quality of the Streamlit interface and clarity of explanations.
- **Report and Documentation (10%)**: Coherence, completeness, and adherence to standards.
- **Demonstration and Overall Integration (10%)**: Functionality and usability in the video.
- **Bonus Consideration**: Up to 5% extra for exceptional UI design or advanced features (e.g., alternative course suggestions), at the instructor’s discretion.

## Additional Notes

- **Late Submission**: Incurs a 10% deduction from the final grade.
- **Demo Expectations**: Be prepared to run the Streamlit app, show source code, and justify design decisions during a review.
- **Resources**: Use AIU advising documents for policy details. Streamlit tutorials (e.g., official documentation) and GitHub guides are recommended.
- **Academic Integrity**: All work must be original and cited properly. Plagiarism or cheating results in a zero grade.