Software Requirements Specification

For

Academic Monitoring and Specialization Selection System

Oct 2024

Prepared by

Specialization	SAP ID	Name
BIG DATA	500107769	Dhuruv Kumar
BIG DATA	500102244	Khushi Chauhan
BIG DATA	500105004	Shruti Srivastava

School of Computer Science

UNIVERSITY OF PETROLEUM & ENERGY STUDIES,

DEHRADUN- 248007. Uttarakhand

Table of Contents

Revision History

Date	Change	Reason for Changes	Mentor Signature

1. INTRODCTION

1.1 Purpose of the Project

The purpose of the "Academic Monitoring & Specialization Selection System" project is to develop, design and implement a dual-purpose academic support system that enhances both student guidance and educational management.
The purpose of this project is twofold:

- **Specialization Selection for Students**: Many students struggle with choosing a specialization that aligns with their academic strengths and career aspirations. This often leads to dissatisfaction and reduced performance in their chosen fields.
- **Performance-based Grouping for Educators**: Educators face difficulties in providing targeted support due to the varied academic performance levels of students. Managing these diverse needs in a large class setting can be challenging, leading to inefficient resource allocation and missed opportunities for student improvement.

This project aims to solve these problems by implementing advanced algorithms for specialization recommendation and performance-based student grouping. The motivation for this project stems from the growing need for data-driven decision-making in education and the potential to significantly improve student outcomes and institutional efficiency. By leveraging data-driven techniques such as classification and clustering algorithms, this project seeks to improve educational outcomes and streamline academic management.

1.2 Target Beneficiary

The primary beneficiaries of this system are:

- **Students**: They will receive personalized recommendations for specialization choices, increasing their chances of success in their academic journey.
- **Educators**: The system will help them efficiently identify different levels of student performance, facilitating targeted support for those who need it and offering opportunities for high achievers to advance further
- **Educational Institutions**: Universities and colleges can use the system to enhance their academic planning and performance management processes, potentially improving overall educational quality and student outcomes.

1.3 Project Scope

Area of Application

The software will be applied in higher education settings, particularly in institutions offering multiple specializations or majors. It will cover two main areas:

- **Specialization Guidance**: The system uses a **Decision Tree classification algorithm** to analyse students' academic records, personal preferences, and career goals. It then recommends the most suitable specialization paths, helping students align their aspirations with their academic strengths.
- **Performance-based Grouping**: A **K-Medoids clustering algorithm** is implemented to group students based on performance data such as test scores, attendance, and feedback. This feature enables educators to identify different performance levels and provide customized support accordingly.

Key objectives and benefits include:

- Personalized guidance for students, improving their chances of success in chosen specializations.
- Efficient grouping of students for educators, facilitating targeted teaching strategies.
- Data-driven decision-making for educational institutions to improve academic management.

Project Requirements and Deliverables:

- **Data Storage**: Implementation of a **MongoDB** database to store academic history, preferences, performance data, and clustering/grouping information.
- **Algorithm Development**: Design and implementation of classification and clustering algorithms for student guidance and performance grouping.
- **User Interface**: A **menu-driven interface** that allows users (students and educators) to interact with the system easily and access recommendations or grouping information.
- **Testing and Refinement**: Continuous testing to refine both the algorithms and the user interface based on feedback and performance metrics.

1.4 References

[1] Singh, Samrat, and Vikesh Kumar. "Performance analysis of engineering students for recruitment using classification data mining techniques." *International Journal of Science, Engineering and Computer Technology* 3, no. 2 (2013): 31.

[2] Le Quy, Tai, Gunnar Friege, and Eirini Ntoutsi. "A review of clustering models in educational data science toward fairness-aware learning." *Educational data science: Essentials, approaches, and tendencies: Proactive education based on empirical big data evidence* (2023): 43-94.

[3] Bobâlcă, Claudia, Oana Ţugulea, and Cosmina Bradu. "How are the students selecting their bachelor specialization? A qualitative approach." *Procedia economics and Finance* 15 (2014): 894-902.

[4] Kurniawan, Tri Basuki, and Indah Hidayanti. "Classification Algorithms to Determine Students' Specialization in a Higher Education Institution." *Journal of Data Science* 2023 (2023).

[5] Hamoud, Alaa, Ali Salah Hashim, and Wid Akeel Awadh. "Predicting student performance in higher education institutions using decision tree analysis." *International Journal of Interactive Multimedia and Artificial Intelligence* 5 (2018): 26-31.

[6] Jiawei, Han, and Kamber Micheline. *Data mining: concepts and techniques*. Morgan kaufmann, 2006.

2. PROJECT DESCRIPTION

2.1 Reference Algorithm

2.1.1 Decision Tree Algorithm for Specialization Selection

Algorithm: C4.5 (A variant of ID3)

Input

- A dataset D containing student attributes:
  - **Numerical attributes:** CGPA, grades in various subjects (CPL, DSA, Python, etc.)
  - **Categorical attributes:** Interests in various specializations (ML, Cyber Security, Networking, etc.)
- **Target attribute:** Specialization choice (the class label).

Output

- A decision tree that predicts the best specialization for a student.

Steps

a) **Preprocessing:**

- **Data Cleaning:** Handle missing values by either removing incomplete entries or imputing them with statistical measures (mean, median, or mode).
- **Normalization:** Normalize numerical data to a common scale if necessary (for example, scaling grades between 0 and 1) to prevent attributes with larger ranges from dominating the model.
- **Categorical Encoding:** Convert categorical attributes (like interests) into numerical form using techniques such as one-hot encoding.

b) **Calculate Entropy:**

- **Entropy Formula:**

$$\text{Entropy(S)} = -\sum_{i=0}^{n} p_i log_2(p_i)$$

- Calculate the entropy for the target variable in dataset S, where p_i is the probability of each class.

c) **Calculate Information Gain:**

- For each attribute A:

$$\text{Information Gain(A)} = \text{Entropy(S)} - \sum_{v \in Value(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

- Here, S_v is the subset of S where attribute A has value v. The attribute with the highest information gain is selected for the split.

d) **Select the Best Attribute:**

- Choose the attribute that provides the highest information gain as the decision node for the current node in the tree.

e) **Create Subtrees:**

- For each value of the selected attribute:
  - Create a branch for each value.
  - Recursively repeat steps 2-4 on the subset of data corresponding to that branch (creating subtrees) until all data points in a subset belong to the same class or no attributes are left to split on.

f) **Stopping Criteria:**

- Stop the recursion if:
  - All instances in a subset belong to the same class (pure node).
  - There are no more attributes to split on (leaf node).
  - A predefined maximum depth is reached to prevent overfitting.

g) **Pruning (Optional):**

- After the tree is built, prune the tree by removing branches that provide little predictive power (using methods like reduced error pruning or cost complexity pruning).

h) **Output:**

- A decision tree model that can predict the specialization based on the attributes of incoming students.

2.1.2 K-Medoids Clustering Algorithm for Academic Monitoring

Algorithm: K-Medoids (PAM - Partitioning Around Medoids)

Input

- A dataset containing student performance metrics (grades, CGPA, interests).
- Number of clusters k.

Output

- Clustering of students into k groups based on their performance and interests.

Steps

a) **Initialization:**

- Randomly select k data points from the dataset to serve as initial medoids. These points represent the centers of the clusters.

b) **Assignment Step:**

- For each student $x_i$in the dataset:
  - Calculate the distance from x_i to each medoid m_j using the Euclidean distance:

$$d(x_i, m_j) = \sqrt{\sum_{p=1}^{n} (x_{ip} - m_{jp})^2}$$

  - Assign x_i to the cluster represented by the nearest medoid.

c) **Update Step:**

- For each medoid m_j:
  - For each non-medoid student x_i:

- Calculate the total cost of clustering if x_i were to be the new medoid.
- Cost is the sum of the distances from all points in the cluster to the new medoid.
- If the cost is lower than the current medoid's cost, update m_j to x_i.

d) **Convergence Check:**

- Repeat the assignment and update steps until the medoids no longer change, or a maximum number of iterations is reached.

e) **Output Clusters:**

- Once convergence is achieved, output the final clusters of students. Each student is associated with a medoid, representing their cluster.

2.2 Data/ Data structure

The project uses data categorized into:

- **Student Data**: Includes academic history (GPA, grades, completed courses), personal preferences (interests, career goals), and demographic information (age, major).
- **Performance Data**: Contains test scores and interest scales.
- **Specialization Data**: Lists available specializations and their requirements.
- **Grouping Data**: Uses performance metrics for clustering students into groups based on their academic performance.

Data Sources

- **Primary**: Data collected from forms created by the project team.
- **Secondary**: Synthetic data generated by the project team for testing purposes.

Sampling Technique

To ensure a fair representation of the student population, we'll use a simple random sampling with some basic categorization:

1. Divide students into broad categories:
   - Undergraduate and graduate students
   - Different academic years (e.g., freshmen, sophomores, juniors, seniors)
2. For each category:

- Make a list of all students
- Use a random number generator to select a predetermined number or percentage of students

3. Combine the selected students from each category to form the final sample
    - This approach ensures we have a mix of students from different academic levels while keeping the selection process simple and random within each group.

Statistical Methods

1. **Descriptive Statistics**: Descriptive statistics provide a summary of the overall trends in the student data, offering insights into important metrics. For example:
    - **Average GPA**: This will give you an overall sense of the students' academic performance.
    - **Common Preferences**: You can identify which specializations or career paths are most popular among students.

    These summaries help you understand patterns in the data without diving into the details of individual students. This broad view is valuable for educators and administrators to recognize general academic strengths, areas of interest, and trends in student performance.

2. **Normalization**: When dealing with performance data like test scores, the range of values can vary widely. To ensure that no single metric (like test scores) dominates or skews the results, normalization is applied. This scales all values to a consistent range. For instance:
    - The interest level is scaled from **1 to 5**.
    - Grades are converted into equivalent numerical values.
    - For students who provided "NC" for CGPA, the project substitutes **5** in place of "NC."

    This makes different metrics comparable and prevents any one data point (e.g., a high-test score) from disproportionately influencing the outcome of your algorithms (classification or clustering).

3. **Outlier Detection**: Outliers are data points that are significantly different from the rest of the dataset, such as a student with an exceptionally high or low GPA compared to others. These outliers can distort the results of algorithms if not handled properly. By applying statistical methods for outlier detection, you can:
    - Identify these extreme values.
    - Handle them by either removing them or adjusting their influence (e.g., capping the extremes).

This ensures that outliers don't skew the results or lead to incorrect conclusions in classification or clustering. For example, a student with an unusually low-test score might unduly influence clustering, leading to poor grouping decisions.

2.3 SWOT Analysis

2.3.1 Strengths:

a) **Data-Driven Decision Making**: The project utilizes advanced algorithms (classification and clustering) to provide personalized recommendations, enabling students to make informed choices about their academic paths.

b) **Personalized Learning Recommendations**: By analysing individual academic histories and preferences, the system tailor's specialization suggestions, enhancing student engagement and success.

c) **Enhanced Educator Support**: Educators can leverage the performance-based grouping feature to identify students needing extra help and to recognize high achievers, allowing for targeted interventions and resource allocation.

d) **Automated Performance-Based Grouping**: The use of clustering algorithms for automatic grouping of students based on their performance metrics simplifies the process for educators. This feature saves time and effort, allowing them to focus on teaching rather than manual student assessments, while also ensuring that students receive the appropriate support based on their performance levels

2.3.2 Weaknesses:

a) **Dependence on Accurate Data Input**: The effectiveness of the system relies heavily on the accuracy and completeness of the input data. Inaccurate data can lead to misleading recommendations and groupings.

b) **Complex Implementation**: The integration of classification and clustering algorithms into existing educational systems can be technically challenging, requiring expertise and resources.

c) **Initial Resistance from Stakeholders**: There may be resistance from educators or administration who are accustomed to traditional methods, making it difficult to fully adopt the new system.

2.3.3 Opportunities:

a) **Growing Demand for Personalized Education**: With the increasing emphasis on personalized learning experiences, there is a significant opportunity to expand the system's reach and adoption among educational institutions.

b) **Potential Partnerships with Institutions and EdTech Companies**: Collaborating with educational organizations and technology companies can lead to improved functionality and wider implementation of the system.

c) **Expansion into Other Educational Areas**: The algorithms and methodologies used in this project can be adapted for use in other educational settings, such as career counselling and vocational training.

2.3.4 Threats:

a) **Privacy Concerns Regarding Student Data**: Handling sensitive student information raises concerns about data privacy and security, which could deter institutions from adopting the system.

b) **Changing Educational Policies and Standards**: Evolving regulations in education and data protection may impact the feasibility and design of the system.

c) **Competition from Other Educational Technologies**: The market for educational technology is rapidly growing, with many companies offering similar solutions. Staying ahead of competitors will require continuous innovation and improvement.

2.4 Project Features

The dual-purpose academic support system includes several key features that enhance student guidance and educational management. Below are the major features along with their corresponding functions:

2.4.1 User Registration and Authentication

- Students and educators can create accounts and log in securely.
- Admins can manage user roles and access.

2.4.2 Student Specialization Guidance

- **Submit Preferences**: Students input their academic preferences and career goals.
- **View Specialization Suggestions**: Students receive tailored recommendations based on their academic performance and preferences.

2.4.3 Performance Metrics Management

- **View Performance Metrics**: Educators can review individual and aggregate performance data to track student progress.
- **Feedback Management**: Educators can provide feedback on student performance.

2.4.4 Automated Performance-Based Grouping

- **Group Students**: The system clusters students based on their performance metrics, allowing educators to manage groups efficiently.

2.4.5 Data Management

- **Database Interaction**: The system interacts with the MongoDB database to store and retrieve student data, performance metrics, and specialization requirements.

2.5 User Classes and Characteristics

2.5.1 Students

Description: Students are the primary users of the system. They provide their academic history and preferences through forms to receive personalized recommendations for specialization paths.

Characteristics:

- **Demographics**: Typically, undergraduate or graduate students in engineering or related fields.
- **Technical Proficiency**: Varies from novice to intermediate; familiar with using web applications.
- **Goals**: To receive accurate specialization recommendations based on their academic performance and interests.
- **Tasks**:

  1. Complete input forms detailing academic history and interests.

  2. Review recommendations generated by the system.

2.5.2 Academic Advisors

Description: Academic advisors support students in making informed decisions regarding their specialization and academic paths. They use the system to access insights about student performance and interests.

Characteristics:

- **Demographics**: Faculty members or designated staff with expertise in academic counselling.
- **Technical Proficiency**: Intermediate; familiar with educational software and data analysis tools.
- **Goals**: To assist students effectively in selecting appropriate specializations and provide targeted guidance based on performance data.
- **Tasks**:

1. Access and review reports generated by the system.
2. Guide students based on the recommendations provided.

2.5.3 System Administrators

Description: System administrators manage the overall functionality and maintenance of the system, ensuring data integrity and user access.

Characteristics:

- **Demographics**: IT professionals or designated staff responsible for system operations.
- **Technical Proficiency**: Advanced; knowledgeable in database management and system configuration.
- **Goals**: To maintain system performance, security, and user data management.
- **Tasks**:
  - Manage user accounts and permissions.
  - Oversee system updates and maintenance.
  - Ensure data security and integrity.

2.6 Design and Implementation Constraints

Clustering Component

a) Hardware Boundary Conditions

- **Timing Requirements**:
  - The clustering algorithm should process and group student data within a maximum response time of 10 milliseconds to ensure a responsive user experience.
- **Memory Requirements**:
  - The application must efficiently manage up to 10,000 student records, requiring a minimum of 4 GB of RAM during peak operations.

b) Interfaces to Other Applications

- **Database Management System**:
  - MongoDB for storing and retrieving student data used in clustering.

c) Specific Technologies and Tools to Be Used

- **Programming Language**: Java, utilizing libraries such as OpenCSV for CSV data handling and collections for managing student records.
- **Development Tools**:
    - IDE: IntelliJ IDEA or Eclipse.

d) Parallel Operations

- The clustering process should be capable of executing in parallel for multiple student datasets, using Java's multi-threading capabilities to enhance processing efficiency.

e) Language Requirements

- Primary language: Java for implementation; English for the user interface.

f) Security Considerations

- Implement user authentication to ensure secure access to clustering functionalities.
- Use data encryption for sensitive information to protect student data during transmission and storage.

g) Design Conventions or Programming Standards

- Follow Java naming conventions and modular design practices for maintainable and readable code.
- Include Javadoc comments for documentation of methods and classes.

Classification Component

a) Hardware Boundary Conditions

- **Timing Requirements**:
    - The classification algorithm should generate specialization recommendations within a maximum of 5 seconds to maintain user engagement.
- **Memory Requirements**:
    - The application should efficiently handle and analyse up to 10,000 student profiles, requiring around 4 GB of RAM for optimal performance.

b) Interfaces to Other Applications

- **Database Management System**:
    - MongoDB for storing academic records and user profiles relevant to classification tasks.
- **Integration with External APIs**:
    - Possible integration with academic systems or platforms that provide additional data for improving classification accuracy.

c) Specific Technologies and Tools to Be Used

- **Programming Language**: Java, employing libraries for machine learning (e.g., Weka or Smile) to implement the decision tree algorithm.
- **Development Tools**:
  - IDE: IntelliJ IDEA or Eclipse.

d) Parallel Operations

- The classification process should be capable of handling multiple user requests simultaneously, utilizing multi-threading in Java for efficient processing.

e) Language Requirements

- Primary language: Java; English for the user interface.

f) Security Considerations

- Ensure user authentication for secure access to classification functionalities.
- Encrypt sensitive user data to safeguard personal and academic information during both transmission and storage.

g) Design Conventions or Programming Standards

- Adhere to Java coding standards, including meaningful variable naming and modular code structure for ease of maintenance.
- Implement Javadoc comments for public methods and classes for clear documentation.

h) Data accuracy

- Training Data Quality: The effectiveness of the Decision Tree depends on accurate and representative training data. Biases or inaccuracies can lead to poor specialization recommendations.
- Student Data Accuracy: Ensuring the accuracy of student data (academic history, preferences, demographics) is vital. Errors or missing information can adversely affect classification results.

2.7 Design diagrams

2.7.1 Use case diagram:

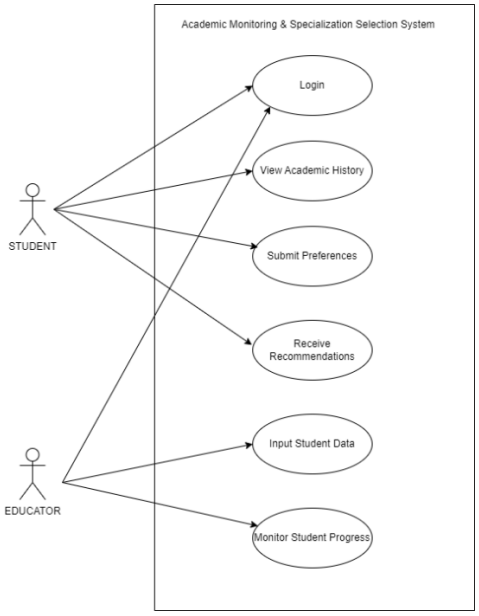

Figure 1: Use case diagram

2.7.2 Class diagram:

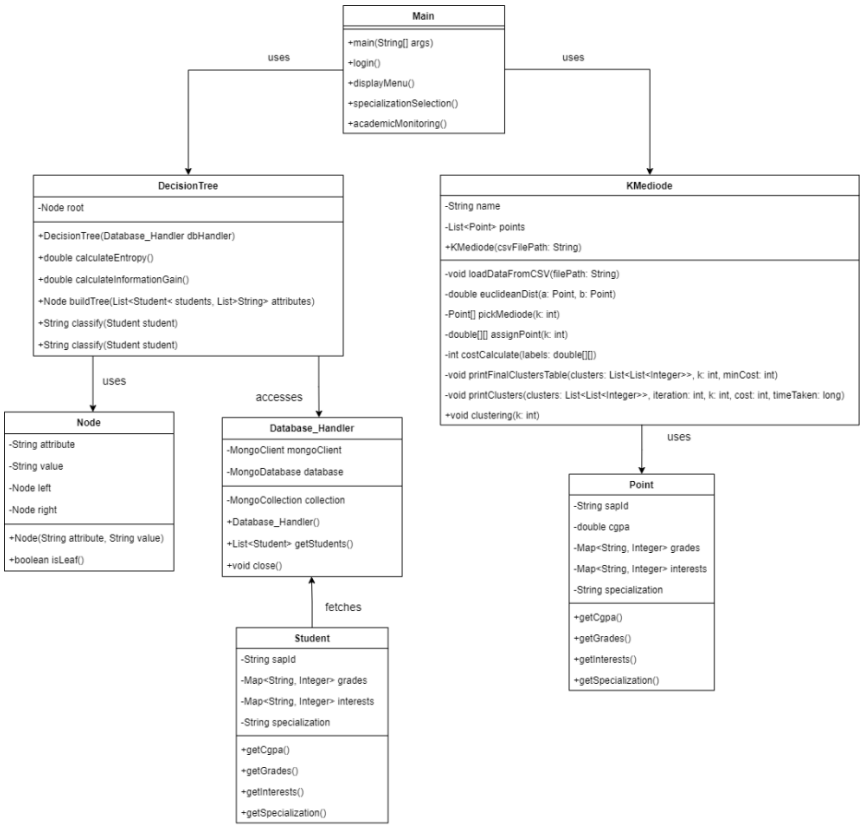

Figure 2: Class diagram

2.7.3 Data flow diagram:

Level - 0 data flow diagram:

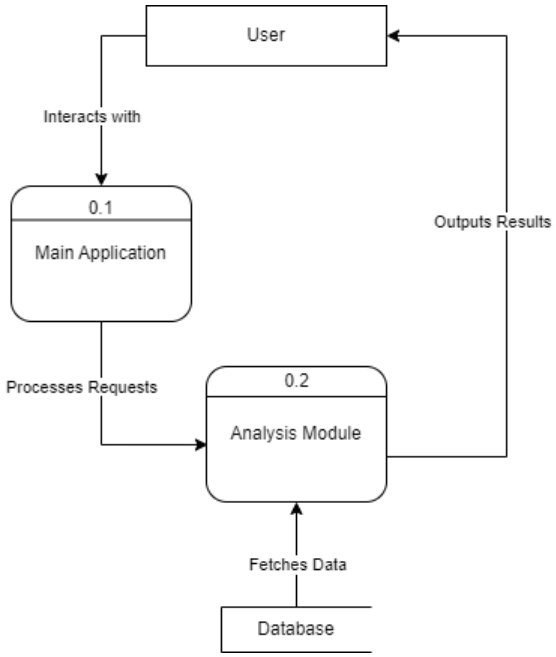

Figure 3-1: Level - 0 data flow diagram

Level - 1 data flow diagram:

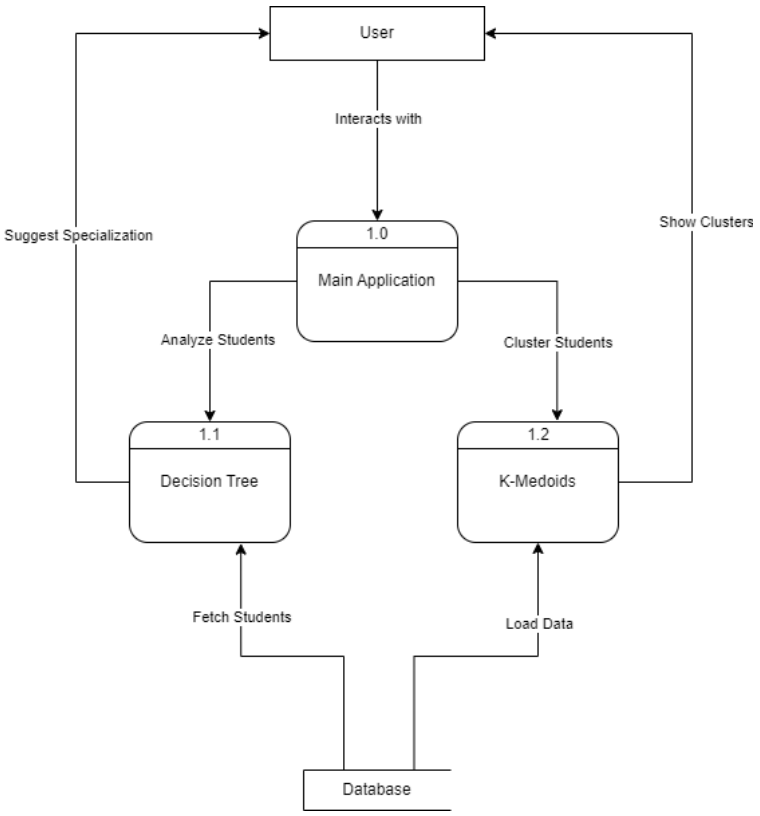

Figure 3-2: Level - 1 data flow diagram

Level - 2 data flow diagram:

Level – 2.1 data flow diagram of Specialization recommender:

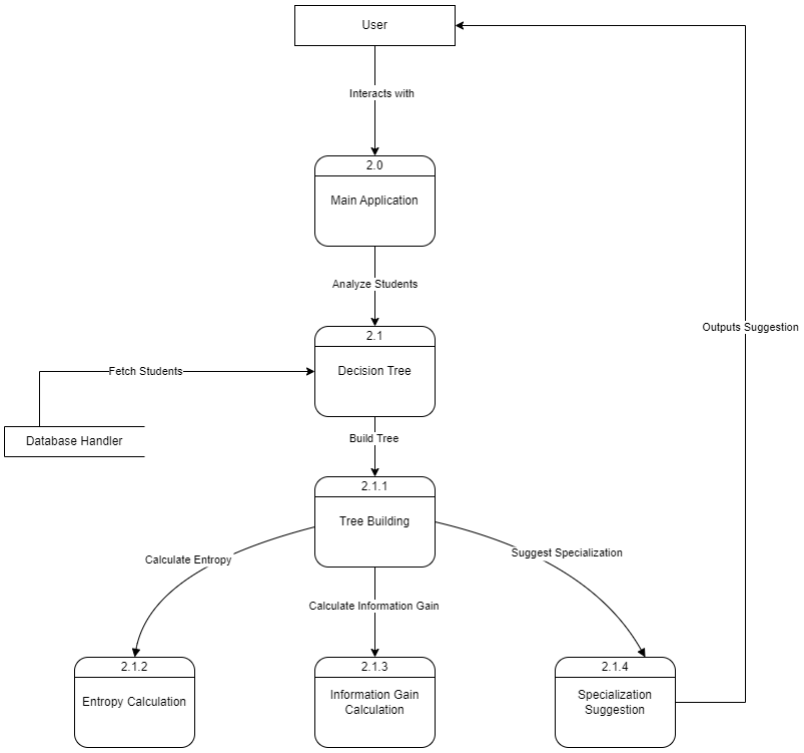

Figure 3-3-1: data flow diagram of Specialization recommender

Level – 2.2 data flow diagram of Academic monitoring:

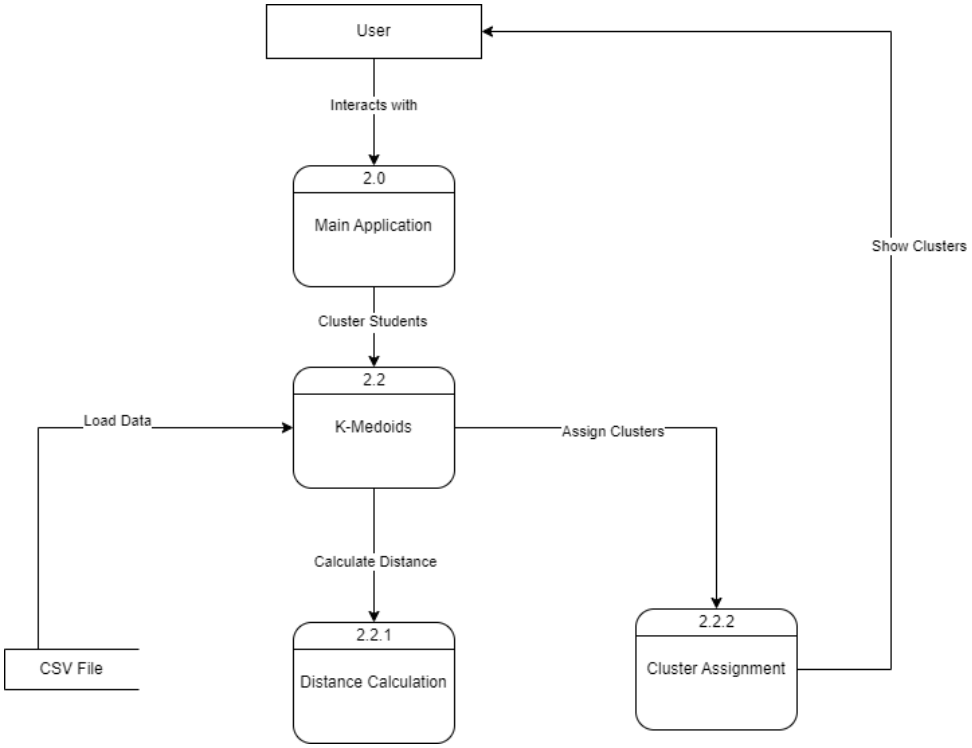

Figure 3-3-2: data flow diagram of Academic monitoring

Level - 3 data flow diagram:

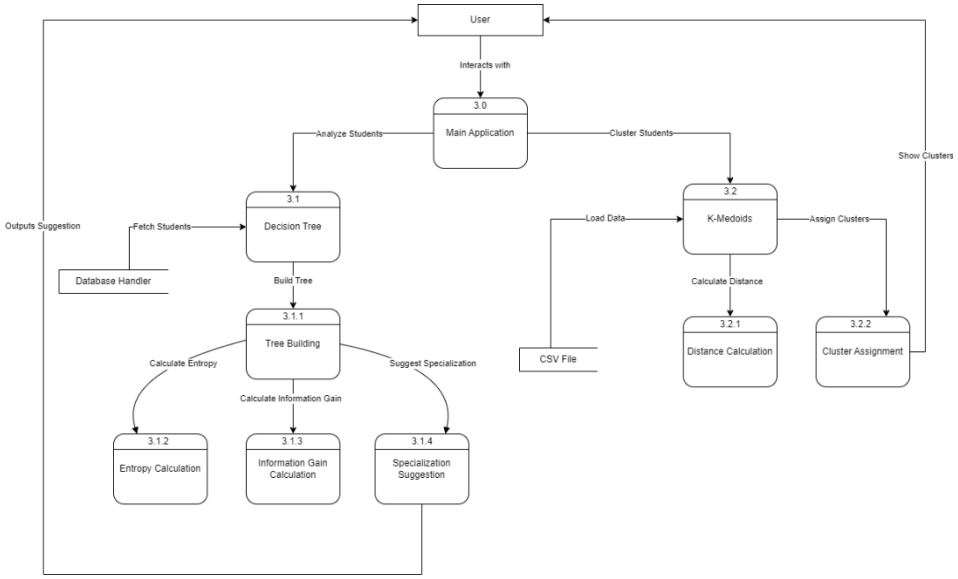

Figure 3-4: Level - 3 data flow diagram

2.7.4 Activity diagram:

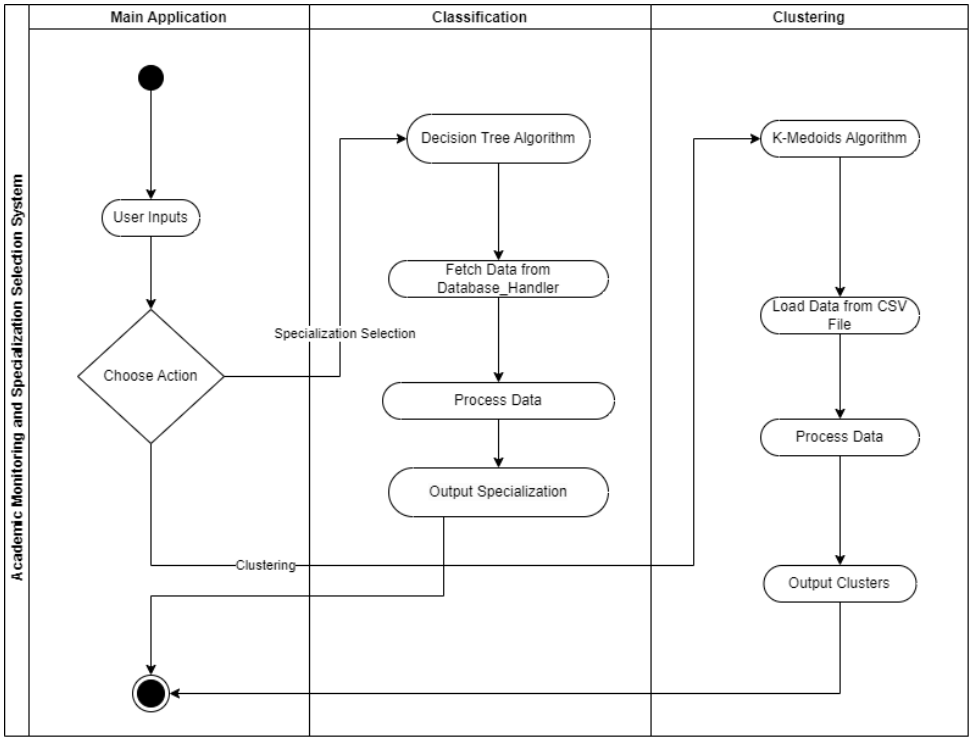

Figure 4: Activity diagram

2.7.5 State diagram:

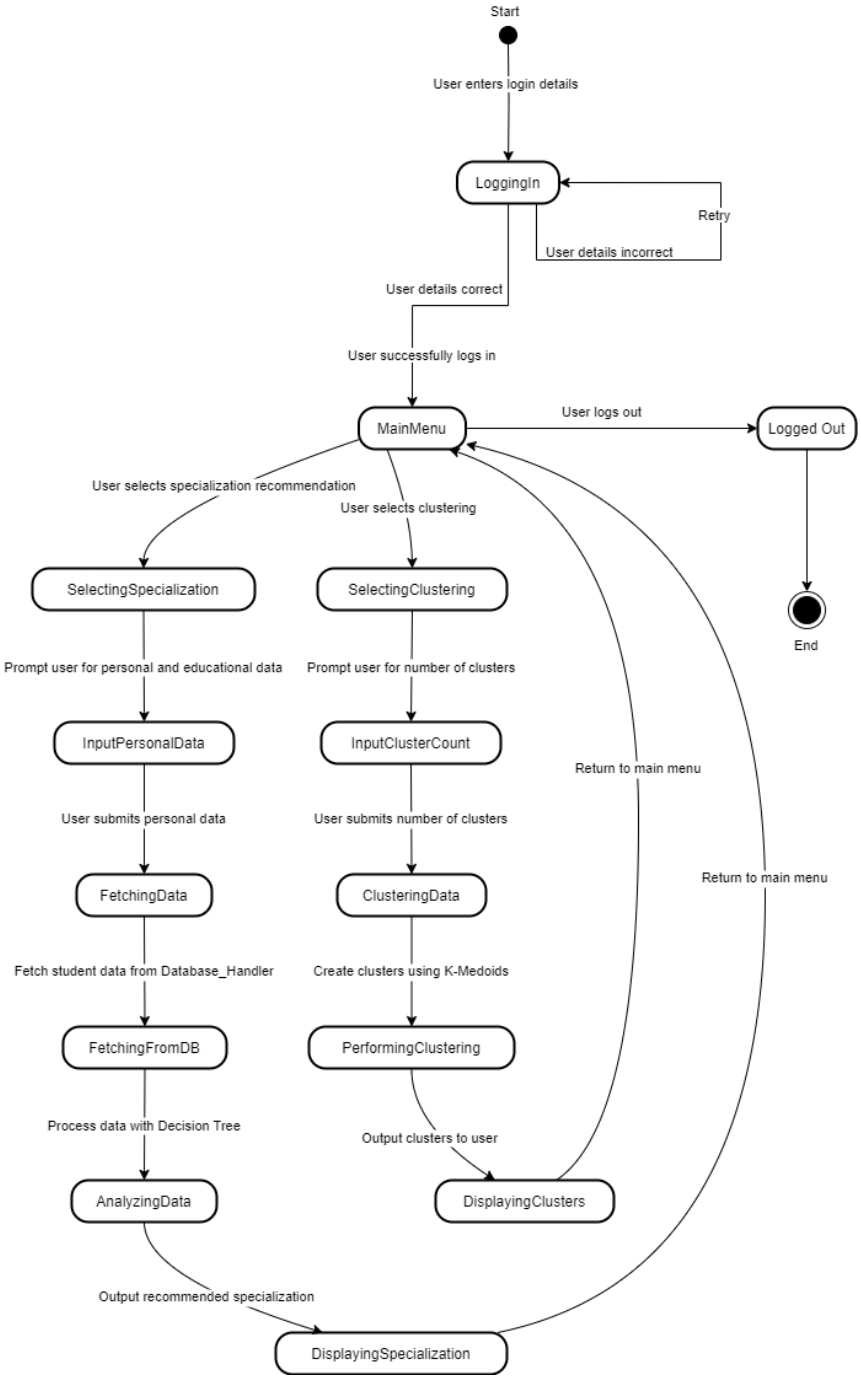

Figure 5: State diagram

2.7.6 Sequence diagram:

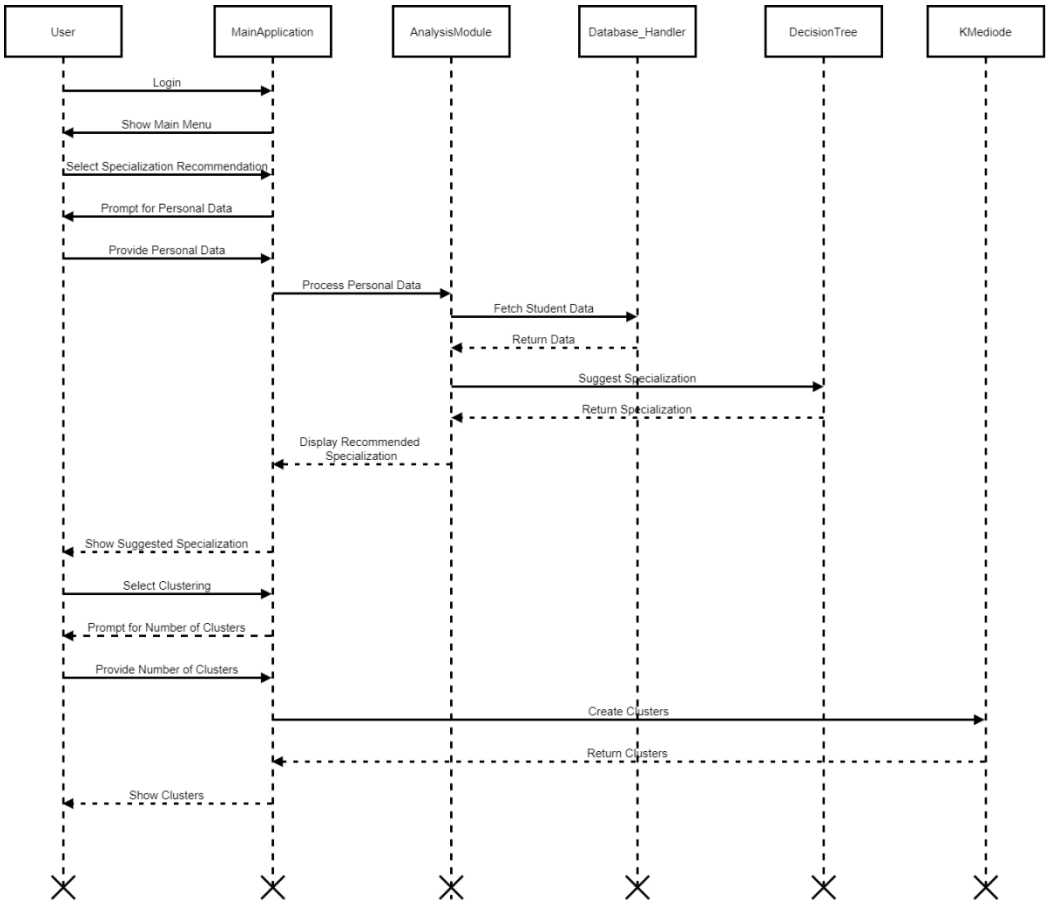

Figure 6: Sequence diagram

2.7.7 Entity relation diagram:

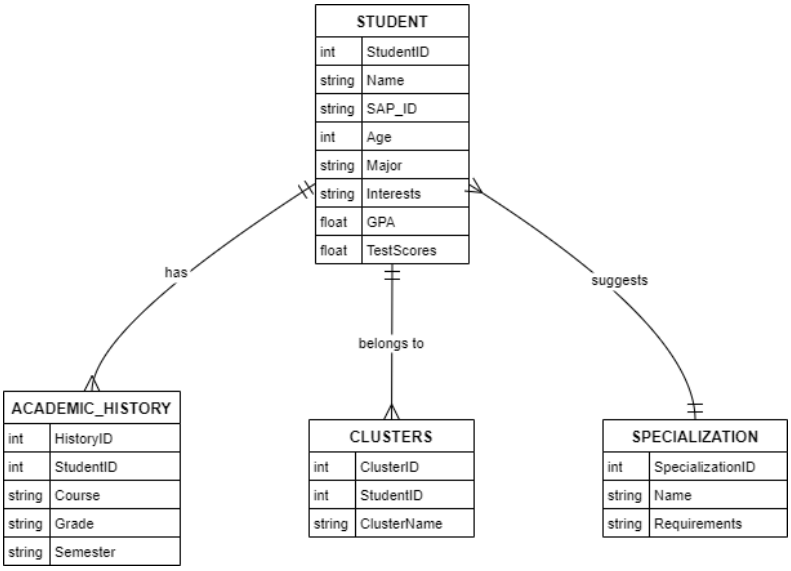

Figure 7: Entity relation diagram

2.8 Assumption and Dependencies

Assumptions

a) **User Familiarity with Technology**:

- It is assumed that users will have a basic understanding of using web applications and will be familiar with navigating user interfaces designed for educational management systems.

b) **Data Quality and Availability**:

- The successful functioning of the clustering and classification algorithms assumes that the student data stored in the MongoDB database is accurate, complete, and up to date.

c) **Performance of Algorithms**:

- It is assumed that the chosen algorithms (Decision Tree for classification and clustering algorithms) will perform effectively on the dataset provided, producing reliable and valid results.

d) **System Resources**:

- It is assumed that the hardware resources (memory, processing power) allocated to the application will be sufficient to handle the expected load of up to 10,000 student records without significant performance degradation.

e) **User Access and Authentication**:

- The system will assume that users will follow proper authentication procedures, ensuring that only authorized personnel access sensitive student data and functionalities.

Dependencies

a) **MongoDB Database**:

- The project relies on MongoDB for data storage and retrieval. Any issues related to database performance, such as slow queries or downtime, could impact the system's overall functionality.

b) **Java Development Environment**:

- The project depends on a stable Java development environment, including the required libraries and tools (e.g., Weka, OpenCSV). Changes or issues in these tools may affect the implementation and performance of the algorithms.

c) **External APIs**:

- If integrated, the project will depend on external APIs for additional data to improve classification accuracy. Any changes or unavailability of these APIs could hinder the system's functionality.

d) **User Hardware and Network**:

- The application assumes users have adequate hardware and internet connectivity to interact seamlessly with the system. Poor network performance or outdated hardware could impact user experience.

3. SYSTEM REQUIREMENTS

3.1 User Interface

The user interface (UI) of the academic monitoring and specialization selection system is designed to be user-friendly and intuitive. It aims to facilitate easy navigation and provide a seamless experience for users, including students and administrators.

User Interface Components

3.1.1 Login Screen

- **Description**: The initial screen for user authentication.
- **Components**:
  - Username Field: Input field for the user's username.
  - Password Field: Input field for the user's password.
  - Login Button: Button to submit login credentials.
  - Forgot Password Link: Link for password recovery.

3.1.2 Main Menu

- **Description**: The main dashboard for navigating system features.
- **Components**:
  - Specialization Recommendation Button: Navigates to the specialization recommendation feature.
  - Academic Monitoring Button: Navigates to the clustering feature for academic monitoring.

- Logout Button: Exits the current session.

3.1.3 Specialization Recommendation Interface

- **Description**: A form for students to input personal and educational data for specialization analysis.
- **Components**:
  - Personal Information Fields: Name, Age, Major, Interests.
  - Educational Background Fields: GPA, Completed Courses, Grades.
  - Submit Button: Submits the data for analysis.
  - Result Display Area: Shows the suggested specialization after processing.

3.1.4 Clustering Interface

- **Description**: A form for inputting parameters for academic monitoring.
- **Components**:
  - Number of Clusters Field: Input field for specifying the number of clusters.
  - Submit Button: Submits the clustering request.
  - Result Display Area: Shows the generated clusters and relevant information.

3.1.5 Results Screen

- **Description**: Displays the results from both specialization recommendations and clustering analysis.
- **Components**:
  - Specialization Results Area: Displays the recommended specialization based on the Decision Tree analysis.
  - Clustering Results Area: Displays the clusters created by the K-Medoids algorithm.
  - Back Button: Returns the user to the main menu.

3.2 Software Interface

The Academic Monitoring & Specialization Selection System consists of two main modules, each with specific roles. This section explains how these modules connect, what services they provide, and how they communicate.

3.2.1 Connections Between Modules

- **Clustering Module**
  - **Input**: Receives student data (like academic history and preferences) for clustering.
  - **Output**: Creates groups of students based on their academic performance.
  - **Database Interaction**: Fetches records from CSV files and saves the clustering results back into the database.
- **Classification Module**
  - **Input**: Takes student profiles and academic records to recommend specializations.
  - **Output**: Suggests specialization paths based on classification methods.
  - **Database Interaction**: Accesses MongoDB to retrieve student profiles and save the recommended specializations.

3.2.2 Services Needed

- **Database Services**:
  - MongoDB provides services for basic data operations (Create, Read, Update, Delete) for student records and clustering results.
- **User Interface Services**:
  - The system needs a user interface that manages user inputs and displays outputs based on user actions.
- **Authentication Services**:
  - Basic authentication is required to ensure secure access to both modules.

3.2.3 Nature of Communications

- **Inter-module Communication**:
  - The Clustering and Classification modules share data through the MongoDB database. Each module works independently but relies on the same data.
- **User Interaction**:
  - Users interact with the system via a menu-driven interface that collects input for clustering and classification.

- The user interface sends commands to the relevant modules and shows results.

3.2.4 Application Programming Interface (API) Protocols

- **Database API**:
  - Uses MongoDB's API for data operations, enabling the application to connect, query, and manage student data.
- **Internal Function Calls**:
  - Each module provides specific functions for its operations:
    - **Clustering Functions**:
      - clustering(): Starts the clustering process.
      - printClusters(): Retrieves and displays the clustering results.
    - **Classification Functions**:
      - getBestAttribute(): Analyzes a student's profile for recommendations.
      - suggestSpecialization(): Returns the suggested specialization.

3.3 Database Interface

Database Management Systems Used

a) **Classification Component**

- **Database Management System**: **MongoDB**
- **Description**:
  - MongoDB is a NoSQL database that stores data in a flexible, JSON-like format. It is designed to manage large volumes of unstructured data, making it ideal for storing student academic profiles and specialization recommendations. The database allows for easy scaling and quick retrieval of documents based on queries.
- **Key Features**:
  - **Document-Oriented Storage**: Stores student records as documents in collections for easy access and manipulation.
  - **Indexing**: Supports various indexing techniques to improve query performance for academic records.

- **Query Language**: Offers a rich query language for performing complex queries and aggregations.

b) **Clustering Component**

- **Data Storage Method**: **CSV Files**
- **Description**:
  - CSV (Comma-Separated Values) files are used to store student data for the clustering component. This simple format allows for easy data manipulation and import/export capabilities. The files contain student records, including academic performance metrics, which are read and processed during clustering operations.
- **Key Features**:
  - **Simplicity**: Easy to read and write, making it accessible for data import and pre-processing.
  - **Portability**: Can be easily transferred and opened in various applications like spreadsheets and data analysis tools.
  - **Flexibility**: Handles different data types, including numeric and categorical data, suitable for clustering algorithms.

c) Interface Specifications

- **Classification Component**:
  - **Data Retrieval**: The system connects to the MongoDB database using a Java MongoDB driver to execute queries for retrieving student profiles and related data.
  - **Data Manipulation**: Implements CRUD (Create, Read, Update, Delete) operations through the MongoDB API to maintain student record integrity.
- **Clustering Component**:
  - **Data Input**: The application reads student data from CSV files using the OpenCSV library, parsing the data into formats suitable for clustering algorithms.
  - **Data Output**: The results of the clustering process can be exported to CSV format for further analysis or reporting.

3.4 Protocols

3.4.1 Data Communication Protocols:

- **MongoDB Protocol:** The application uses the MongoDB driver to establish a connection with the MongoDB database. This allows for data retrieval and manipulation through commands issued from the command-line interface.

3.4.2 File Handling Protocols:

- **CSV File Protocol:** The application reads and writes student data using the CSV file format. It utilizes the OpenCSV library for parsing and managing data in a simple and effective manner.

3.4.3 Command-Line Interface (CLI) Protocols:

- **User Interaction:** Users interact with the system through a menu-driven command-line interface. The interface processes user inputs for selecting actions (e.g., specialization recommendation or clustering).

- **Input/Output Handling:** User inputs are captured via command prompts, and results are displayed directly in the terminal, ensuring a straightforward and efficient user experience.

3.4.4 Error Handling Protocols:

- The application implements basic error handling to manage issues such as invalid inputs or database connection failures. Error messages will be displayed in the command-line interface to inform users of any problems encountered.

4. NON-FUNCTIONAL REQUIREMENTS

4.1 Performance requirements

This section outlines the performance requirements for the Academic Monitoring & Specialization Selection System. These requirements ensure that the system operates efficiently and effectively under expected conditions.

a) **Response Time**

- The system should provide feedback to the user within **2 seconds** for data retrieval and processing actions (e.g., fetching student data, generating clusters, suggesting specializations).

- The response time for clustering operations (K-Medoids) should not exceed **5 seconds** for datasets containing up to **1,000 students**.

a) **Scalability**

- The system must be able to scale efficiently as the number of users and data increases. It should handle an increase of up to **10,000 students** without a significant degradation in performance.

- The K-Medoids clustering algorithm should be able to accommodate up to **500 data points** per cluster without exceeding acceptable response times.

b) **Data Handling Capacity**

- The system should support a minimum of **5,000 records** in the MongoDB database while maintaining optimal query performance.

- CSV file handling for the clustering component should efficiently process files containing up to **5,000 records** without significant latency.

c) **Accuracy**

- The classification and clustering processes should maintain a minimum accuracy of **85%** for specialization suggestions and clustering results based on historical data and user feedback.

d) **Concurrency**

- The system should support a minimum of **10 concurrent users** accessing the system simultaneously without a noticeable decline in performance.

- Database interactions should be optimized for concurrent read and write operations, allowing multiple users to access data without conflicts.

e) **Resource Utilization**

- The system should utilize no more than **70%** of CPU and **75%** of memory resources during peak operation.

- The database should maintain efficient indexing and storage to minimize disk space usage and enhance retrieval times.

f) **Logging and Monitoring**

- The system should log user interactions and errors for performance monitoring and debugging purposes, allowing for real-time analysis without affecting system performance.

4.2 Security requirements

This section outlines the security requirements for the Academic Monitoring & Specialization Selection System. These requirements are essential to protect sensitive data, ensure user privacy, and maintain the integrity of the system.

4.2.1 User Authentication

- The system must implement a secure user authentication mechanism, requiring users to provide valid credentials (username and password) before accessing the main application.
- Passwords should be stored securely using hashing algorithms (e.g., bcrypt) to prevent unauthorized access to user accounts.

4.2.2 Access Control

- Access to different functionalities of the system should be restricted based on user roles (e.g., administrator, student). Users should only have access to features relevant to their role.
- The system should enforce session management to log users out after a period of inactivity (e.g., 15 minutes) to prevent unauthorized access.

4.2.3 Data Encryption

- All sensitive data, including personal information and academic records, should be encrypted both in transit and at rest to protect against data breaches.
- Communication between the client application and the MongoDB database must be secured using SSL/TLS protocols to prevent eavesdropping and man-in-the-middle attacks.

4.2.4 Input Validation

- The system should implement input validation mechanisms to prevent SQL injection, cross-site scripting (XSS), and other injection attacks.
- User inputs must be sanitized and validated to ensure only expected data types and formats are accepted.

4.2.5 Data Integrity

- The system should maintain data integrity by implementing validation checks during data input and processing. This ensures that only valid and accurate data is stored and used in analysis.

- Regular audits and checks should be performed to identify and rectify any data inconsistencies or corruptions.

4.2.6 Error Handling

- The system should implement secure error handling to prevent the exposure of sensitive information in error messages. Errors should be logged internally without displaying detailed information to the user.
- Users should receive generic error messages to avoid revealing system vulnerabilities.

4.2.7 Audit Logging

- The system must maintain an audit log of user activities, including login attempts, data access, and modifications, to ensure accountability and traceability.
- Audit logs should be protected against tampering and should only be accessible to authorized personnel for review.

4.2.8 Backup and Recovery

- Regular backups of the MongoDB database should be scheduled to prevent data loss in case of system failure or cyberattacks.
- A disaster recovery plan should be established to ensure data can be restored quickly and accurately after a breach or loss.

4.2.9 Security Training

- Users and administrators should undergo regular security training to recognize potential threats and adopt best practices for data protection and privacy.

4.3 Software Quality Attributes

4.3.1 Adaptability

- The system would be designed to accommodate changes in educational requirements or user preferences without requiring extensive modifications to the codebase. This includes the ability to integrate new algorithms for clustering or classification as they become available.

4.3.2 Availability

- The application must be accessible to users whenever needed, aiming for a target uptime of at least 99%. Maintenance windows would be planned to minimize user impact.

4.3.3 Correctness

- The system would accurately implement the classification and clustering algorithms to ensure that recommendations and groupings reflect the actual data and user profiles, thus providing reliable outputs.

4.3.4 Flexibility

- The architecture of the application would allow for easy modification and enhancement of features, enabling the addition of new functionalities or the modification of existing ones without significant restructuring.

4.3.5 Interoperability

- The application would support integration with external data sources and services, including educational platforms and APIs, to enhance its functionality and data accuracy.

4.3.6 Maintainability

- The system would be structured in a way that allows developers to easily update, fix, or enhance the application. This includes clear documentation and adherence to coding standards to facilitate understanding and modification of the codebase.

4.3.7 Portability

- The application would be portable across different operating systems and environments, ensuring that it can run on various platforms with minimal adjustments.

4.3.8 Reusability

- Components of the application, such as data handling modules or algorithm implementations, would be designed for reusability in other projects or modules, reducing duplication of effort in future developments.

4.3.9 Robustness

- The application would be able to handle errors gracefully, providing meaningful feedback to users without crashing. This includes validating inputs and managing exceptions effectively.

4.3.10 Testability

- The system would be designed to facilitate testing, with clear interfaces and modular components that allow for unit testing, integration testing, and user acceptance testing to ensure quality.

4.3.11 Usability

- The user interface would be intuitive and user-friendly, allowing users to navigate the menu-driven system easily. Proper user feedback would be provided to enhance the overall user experience.

5. OTHER REQUIREMENTS

5.1 Training Requirements

- Training sessions should be organized for users and administrators to ensure effective use of the system and to familiarize them with its functionalities.

5.2 Backup and Recovery Plans

- A robust backup strategy should be implemented to prevent data loss, with regular backups of the MongoDB database and CSV files. A recovery plan must be in place to restore data in case of system failures.

5.3 Performance Monitoring

- The system should include performance monitoring tools to track application responsiveness, resource utilization, and error rates, allowing for proactive management of performance issues.

5.4 Testing Requirements

- A thorough testing strategy must be established, including unit testing, integration testing, and user acceptance testing, to ensure the reliability and correctness of the application before deployment.

5.5 System Maintenance

- Ongoing maintenance processes should be defined to address bug fixes, updates, and enhancements, ensuring the system remains functional and secure over time.

5.6 Technical Support

- A technical support mechanism should be established to assist users with any issues encountered while using the system, providing timely responses to user inquiries.

5.7 Scalability

- The system should be designed to accommodate future growth, allowing for an increase in the number of users and data volume without significant degradation in performance.

➢ Appendix A: Glossary

Academic Monitoring: The process of tracking and analysing students' academic performance to provide insights and support for their educational journey.

Clustering: A data mining technique that groups a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups. Used for identifying patterns in student performance.

Classification: A machine learning approach that assigns labels to data points based on their attributes, aiming to predict the categorical outcome of a data point.

Decision Tree: A flowchart-like structure used in machine learning for making decisions based on the values of input features, commonly used for classification tasks.

Database Management System (DBMS): Software that interacts with the user, applications, and the database itself to capture and analyse data. In this project, MongoDB is utilized for storing classification data.

MongoDB: A NoSQL database that stores data in a flexible, JSON-like format, allowing for scalability and efficient data retrieval.

CSV (Comma-Separated Values): A simple file format used to store tabular data in plain text, where each line represents a data record and each record consists of fields separated by commas. Used in this project for clustering data storage.

User Authentication: The process of verifying the identity of a user attempting to access the system, ensuring that only authorized individuals can use the application's functionalities.

Menu-Driven Program: An application interface that allows users to interact with the system by selecting options from a list or menu, facilitating easy navigation.

Role-Based Access Control (RBAC): A security approach that restricts system access to authorized users based on their roles, ensuring that users can only perform actions permitted for their role.

Data Protection Regulations: Legal requirements that govern the processing of personal data, ensuring individuals' privacy and rights. Examples include the General Data Protection Regulation (GDPR).

Usability: A measure of how easy and efficient it is for users to interact with the system, emphasizing user satisfaction and the effectiveness of the interface.

Robustness: The ability of the system to handle errors and unexpected inputs gracefully without crashing or producing incorrect results.

Performance Benchmarking: The process of measuring the performance of the system under specific conditions to ensure it meets the required performance standards.

Interoperability: The capability of the system to integrate and work with other external systems or services, enhancing its functionality and data exchange.

➢ Appendix B: Analysis Model

- **Use Case Diagrams**:

  Show interactions between users (administrators and students) and the system.

  Highlight functionalities like viewing recommendations, inputting student data, and generating clustering results.

  Define functional requirements based on user needs.

- **Class Diagrams**:

  Illustrate the structure of the system with different classes, attributes, and methods.

  Show relationships among classes (associations, generalizations, dependencies).

  Serve as a blueprint for future code implementation.

- **Data Flow Diagrams (DFDs)**:

  Visualize how data moves through the system.

  Outline inputs, outputs, data stores, and processes.

  Present in multiple levels:

  Level 0: Context diagram showing the system as a whole.

  Level 1: Breakdown of major processes like data input, classification, and clustering.

- **Activity Diagrams**:

  Represent the sequence of activities or flow of control during operations.

  Show steps involved in processes, such as generating a specialization recommendation based on student data.

  Help visualize how key functions are performed.

- **State Diagrams**:

  Depict the different states an object can be in and transitions between these states.

Outline the lifecycle of important objects, like a student's status changing from "Registered" to "Recommended."

Provide a comprehensive view of object behaviour throughout the system.

- **Sequence Diagrams**:

  Clarify interactions within the system by showing the order of operations during specific use cases.

  Illustrate how different components collaborate to complete tasks, such as generating recommendations.

  Ensure correct sequencing of interactions between objects.

- **Entity-Relationship (ER) Diagram**:

  Represents the data model for the system, showing the entities involved and their relationships.

  Key entities may include:

  Student: Contains attributes like Name, SAP ID, GPA, and Interests.

  Specialization: Contains details about available specializations and requirements.

  Performance: Stores performance metrics such as test scores and interest levels.

  Clustering: Represents clusters formed based on academic performance metrics.

  Shows relationships between entities, such as:

  A **Student** can have multiple **Performance** records.

  A **Specialization** can be recommended to multiple **Students** based on their profiles.

➢ Appendix C: Issues List

The following outlines the current issues identified:

Data Privacy Concerns: There are ongoing discussions regarding compliance with local data protection regulations, particularly concerning the handling and storage of sensitive student information. Further analysis is required to align the system's practices with these regulations.

Integration Challenges: There may be potential integration issues with external educational platforms or APIs that could affect data accuracy and interoperability. A detailed assessment of required APIs and their functionalities is pending.

User Access Control: The role-based access control implementation requires further clarification to ensure that user permissions are appropriately assigned and managed. A comprehensive review of user roles and permissions is needed.

Testing Strategy: A detailed testing strategy that encompasses unit testing, integration testing, and user acceptance testing is yet to be defined. This strategy is crucial for ensuring the system's reliability and usability.