OBJECT ORIENTED SOFTWARE DESIGN PROJECT

ITERATION-2

Requirement Analysis Document (RAD)

|  |  |
| --- | --- |
| GROUP 14 | |
| Name Surname | No |
| Umut Çağlar ÇELİK | 150121003 |
| Safa BAKIRCIOĞLU | 150122515 |
| Hikmet TOPAK | 150121047 |
| Efe YALIM | 150121074 |
| Emirhan ÜNSAL | 150121011 |
| Esra UĞURBAŞ | 150121056 |

# Project Description

In this project, we're developing a comprehensive Java-based course registration system for a university department. The main goal is to streamline the registration process for students and advisors, with a design that allows for future expansion to support roles like department heads and administrators.

We’re building a set of classes like Course, Student, and Advisor, each designed with purposeful methods to enforce the department’s business logic. These classes go beyond simple data storage, utilizing inheritance and encapsulation to enhance code flexibility, with well-defined aggregation relationships illustrating their interactions.

A key design element is the separation of domain logic from the user interface, maintained by a dedicated controller class that establishes a clear boundary between them.

This project is fundamentally focused on creating a flexible and secure course registration system that complies with department regulations, leverages object-oriented principles for maintainable and efficient code, and facilitates effective academic record management for students and advisors. It’s a complete solution that meets the department’s essential needs and allows for future improvements.

# Glossary

* Student: A registered student in the system.
* Advisor: Academic advisors helping students choose their courses.
* Course: An instructional module provided by the department.
* Department Scheduler: Schedules the course sections (e.g. place, hour, day).
* Course Section: Some information about related course (e.g. place, time).
* Grade: A student's academic assessment for a particular course.
* Registration: The course enrollment process.
* Transcript: A student's academic performance record.
* Course Registration System: Control class overseeing communication.
* Username and Password: Use data to verify users' identities within the system. To access the system, users and students enter their passwords and usernames.
* Wait List: Waiting list for students who try to request a course which has no space for them.
* Prerequisite Course: The course which is required to pass to request another course.
* Status: A feature for a course to be active or passive.
* Credit: The unit that represents the contribution of a specific course to a student's overall grade point average.
* Enrollment Capacity: Max number of students that can be capable of enrolling on that specific course.
* Current Capacity: The current number of students who enrolled on that course.

# Functional and Non-Functional Requirements

a-) Functional Requirements

1. **User Login**
   1. The system should allow users to log in with assigned usernames and passwords.
   2. Priority: Very High
   3. Criticality: Very Critic
   4. Risks: User authentication errors, lack of password security.
   5. Reduce Risks: Strong security policies.

## **User Roles**

* 1. The system should define different user types and manage them with role-based authorizations within the system.
  2. Priority: High
  3. Criticality: High
  4. Risks: Incorrect assignment of user roles or faulty authorizations
  5. Reduce risks: Minimizing errors by establishing well-defined user roles and authorizations.

## **Course Registration Process**

* 1. The system should conduct the course registration process in compliance with the department's rules.
  2. Priority: High
  3. Criticality: High
  4. Risks: Violation of course registration rules
  5. Reduce risks: Designing and verifying the registration process in accordance with department policies.

## **Registering for Courses**

* 1. Users should be able to view courses and request for each of them.
  2. Priority: High
  3. Criticality: Very Critic
  4. Risks: Users can be requested for the course which the users should not register for.
  5. Reduce risks: Preventing errors by using a user friendly interface.

## **Scheduling Course Sections**

1. The department scheduler should be able to schedule course sections according to his/her wish.
2. Priority: High
3. Criticality: High
4. Risks:Conflicts between course timings. Overlapping schedules for faculty or classrooms.
5. Reduce Risks: Develop a conflict detection mechanism for schedules.

## **Approving or Rejecting Course**

* 1. The advisors should be able to approve or reject the students’ requests.
  2. Priority: High
  3. Criticality: Medium
  4. Risks: Advisors may accidentally approve or reject requests.
  5. Reduce risks: Provide a confirmation step before finalizing decisions.

## **Conflict Detection**

1. The information of course sections which were chosen by the same student should not be the same.
2. Priority: Medium
3. Criticality: High
4. Risks: Cannot get the information about course sections.
5. Reduce Risks: Information of course sections should write the cleaner and more readable.

## **Course Statistics**

* 1. The system should generate and display statistics for courses to users.
  2. Priority: Low
  3. Criticality: Low
  4. Risks: Inaccurate or inappropriate statistical representations.
  5. Reduce risks: Recheck the statistics of courses and ensure the values are correct.

b-) Non-Functional Requirements

## **Availability:**

* + The application should be highly available, with continuous operation and minimal downtime. Downtime should remain within acceptable limits, and backup systems for timely failover may be implemented to support this.

## **Security:**

* + The application must be safeguarded against unauthorized access and security breaches.

## **Portability:**

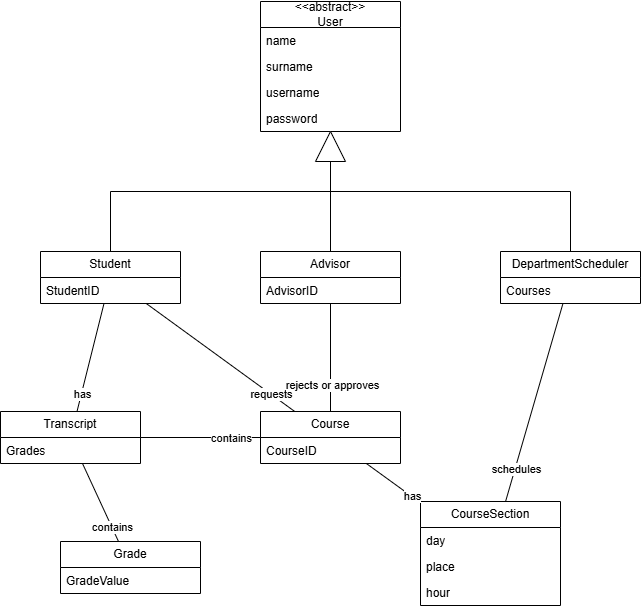
* + The application should operate smoothly across various platforms, including Windows, Linux, and macOS. This requires writing platform-independent code, leveraging suitable tools and frameworks, and avoiding platform-specific dependencies.

1. **Performance:**
   * The application's performance is essential to ensure that users can complete operations quickly. Key performance metrics include response times and processing speeds.Techniques like code and database optimization can be applied to boost overall performance.

## **Maintainability:**

* + The code should be maintainable over the long term by adhering to object-oriented programming (OOP) principles, making it modular and easy to understand. This approach facilitates code expansion and reuse.

Domain Model



In this model:

* The “User” class represents the user of the system. It can be an advisor, student or department scheduler.
* The “Student”, “Advisor” and “DepartmentScheduler” classes represent students, advisors and department scheduler, respectively.
* The “Course” class represents individual courses.
* The “Transcript” class represents the transcript of the student who is log in the system.
* The “Grade” class represents the grade taken by the student for a particular course.
* The “CourseSection” class represents some information related to the course it is associated with.

Use Cases

Use Case 1: Request Course

* Purpose: to establish a system where students request a course
* Actor: Student
* Preconditions:

-The users must login successfully to their registration system according to the true inputs.

-The system must have shown the course list which includes the courses available to request.

* Main Flow:

1. The student chooses the request course operation.
2. The system shows the list of the available courses.
3. The student requests the course for enrollment.
4. The system adds the course to the required course list of the student.

Use Case 2: Approve Course

* Purpose: To enroll students for requested courses and verify the course registration for the students
* Actor: Advisor
* Preconditions:

-The advisor must have logged in the system successfully.

-The system must have shown requests of students to the advisor correctly.

-The student must have requested the course.

-The student must be enrolled in fewer than 5 courses.

* Main Flow:

1. The system shows the requested course to the advisor.
2. The advisor approves if the preconditions are provided.
3. The course is added to enroll course list of the student.
4. The course is removed to request course list of the student.

Use Case 3: Display Enrolled Course List

● Purpose: to display enrolled courses to the student

● Actor: Student

● Preconditions:

-The student must have logged into the system successfully.

● Main Flow:

1. The student chooses the “Enrolled Courses” operation.
2. The system gets the list of the enrolled courses of the student.
3. The system displays that list to the student.

Use Case 4: Display Transcript

● Purpose: to display transcript to the student

● Actor: Student

● Preconditions:

-The student must have logged into the system successfully.

● Main Flow:

1. The student chooses the “View Transcript” operation.
2. The system gets the list of the grades taken and names for each course of the student.
3. The system displays that transcript to the student.

Use Case 5: Conflict While Requesting a Course

* Purpose: to enable the students to choose a course without conflict with their schedules.
* Actors: Student, Department Scheduler
* Preconditions:

-The student must be logged into the system.

-The student has access to their list of enrolled courses, requested courses, and course schedule.

-The Department Scheduler has ensured that course schedules and places are updated and conflict-free within the department.

-There must be some overlap between the courses that students will take.

* Main Flow:

1. The Department Scheduler chooses the course section details.
2. The student logins the system and selects a course to request.
3. If there is no conflict based on his/her schedule, the student can request the course.
4. If a conflict is detected based on his/her schedule:

-The system notifies the student about the conflict and asks the student to choose one of the courses.

Use Case 6: Adding and Removing From Waitlist a Student

* Purpose: to not be able to request a course for a student which has a chance to exceed its enrollment capacity.
* Actor: Student, Advisor
* Preconditions:

-The student must be logged into the system.

-The system must have displayed the list of available courses correctly.

-The sum of the number of current capacity of the course and the number of students who have requested that course.

-The waitlist of that course must be empty.

* Main Flow:

1. The student requests the course which has a chance to exceed its enrollment capacity.
2. The system adds the student to the waitlist of that course.
3. The system sends a message to inform the student about that situation.
4. The student logs out the system.
5. An advisor who advises that student logs into the system.
6. The advisor chooses the option: “See the requests.”
7. The system displays the requests of the students advised by that advisor.
8. The advisor rejects the request of a student.
9. The system adds the course to the requested course list of the student who is the first member of the waitlist of that course.
10. The system removes the student from the waitlist of that course.
11. The advisor chooses the option: “See the requests.”
12. The system displays the requests of the students advised by that advisor.
13. The advisor can see the request of the student who waited in the waitlist of the course.