OBJECT ORIENTED SOFTWARE DESIGN PROJECT
ITERATION-1
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.

CourseSection: Particular example of a semester-long course offered.

Grade: A student's academic assessment for a particular course.

Registration: The course enrollment process.

Transcript: A student's academic performance record.

CourseRegistrationSystem: 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.

Functional and Non-Functional Requirements

a-)Functional Requirements

1. **User Login**
    a. The system should allow users to log in with assigned usernames and passwords.
    b. Priority: Very High
    c. Criticality: Very Critic
    d. Risks: User authentication errors, lack of password security.
    e. Reduce Risks: Strong security policies.
2. **User Roles**
    a. The system should define different user types and manage them with role-based authorizations within the system.

b. Priority: High
c. Criticality: High
d. Risks: Incorrect assignment of user roles or faulty authorizations
e. Reduce risks: Minimizing errors by establishing well-defined user roles and authorizations.

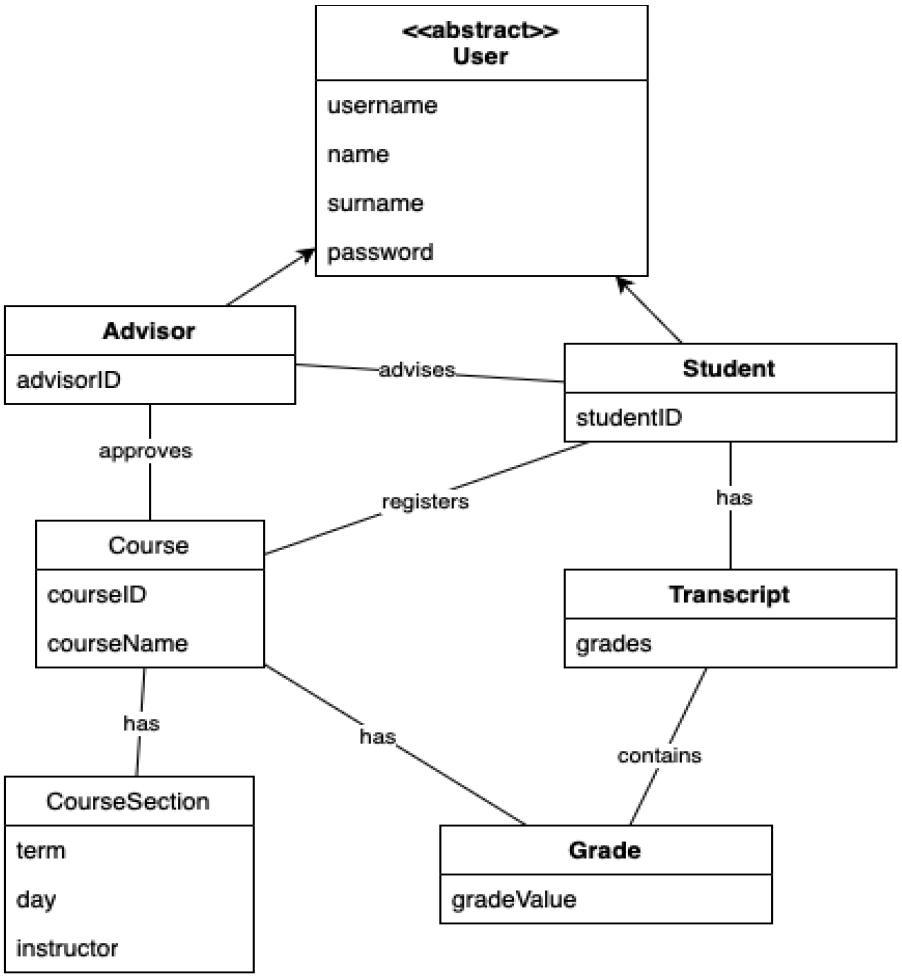
3. **Course Registration**
   a. The system should conduct the course registration process in compliance with the department's rules.
   b. Priority: High
   c. Criticality: High
   d. Risks: Violation of course registration rules
   e. Reduce risks: Designing and verifying the registration process in accordance with department policies.
4. **Registering for Courses**
   a. Users should be able to view courses and request for each of them.
   b. Priority: High
   c. Criticality: Very Critic
   d. Risks: Users can be requested for the course which user should not register.
   e. Reduce risks: Preventing errors by using a user-friendly interface.
5. **User Authorization**
   a. The system should perform the necessary processes for user authorization.
   b. Priority: High
   c. Criticality: Medium
   d. Risks: Users who log in the system with wrong information access the system.
   e. Reduce risks: Ensuring security operations work perfectly and complete implementation of authentication processes.
6. **Course Statistics**
   a. The system should generate and display statistics for courses to users.
   b. Priority: Low
   c. Criticality: Very Low
   d. Risks: Inaccurate or inappropriate statistical representations.
   e. Reduce risks: Recheck the statistics of courses and ensure the values are correct.

b-)Non-Functional Requirements

1. **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.
2. **Security:**
    - The application must be safeguarded against unauthorized access and security breaches.
3. **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.

4. **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.
5. **Security:**
    - The application must be secured to prevent unauthorized access and protect against data breaches.
6. **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 advisor or student.
- The “Student” and “Advisor” classes represent students and advisors, 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.