Object Oriented Software Design Group Project
RAD Report

GROUP NUMBER 15	
NAME SURNAME	NO
Muhammed Enes Gündüz	150120038
Mustafa Tolga Akbaba	150120001
Kadir Pekdemir	150121069
Murat Albayrak	150120025
Necati Koçak	150120053
Taha Enes Karaduman	150120530

Brief Description of the Project

In this project, we're embarking on the creation of a robust Java-based course registration system for a university department. Our primary focus is to provide a seamless registration process for both students and advisors, with the potential for future expansion to accommodate roles like department heads or administrators.

We're developing a variety of classes, including Course, Student, Advisor, and more. These classes are not mere data containers; they come equipped with meaningful methods to enforce the department's business logic. We'll use inheritance and polymorphism to improve code flexibility, with clear aggregation relationships between classes to define how they interact.

Additionally, the project features the ability to read and update student transcripts stored in JSON files and manage system parameters through a separate parameters.json file. The clean separation of domain logic from the user interface is another key aspect of our design, with a dedicated controller class ensuring a clear boundary between the two.

In essence, this project is all about building an adaptable and secure course registration system that adheres to the department's rules, incorporates object-oriented principles to make the code maintainable and efficient, and ensures smooth academic record management for both students and advisors. It's a comprehensive solution that addresses the core needs of the department and sets the stage for potential future enhancements.

Glossary

- **Student:** A registered student in the department.
- **Personal:** A collective term for university staff members. It

  consists of academic staff, support staff, and other workers.

- **Advisor:** Academic advisors helping students choose their courses.
- **Course:** An instructional module provided by the department.

- **Lecturer:** the teacher in charge of instructing classes.
- **CourseSecttion:** 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

Functional Requirements

1. **User Login**
    - The system should allow users to log in with assigned usernames and passwords.
    - Priority: High
    - Criticality: Very Critical
    - Risks: User authentication errors, password security breaches.
    - Reduce risks: Strong password policies.
2. **User Roles**
    - The system should define different user types (e.g., students and advisors) and manage them with role-based authorizations within the system.
    - Priority: High
    - Criticality: High
    - Risks: Incorrect assignment of user roles or faulty authorizations
    - Reduce risks: Minimizing errors by establishing well-defined user roles and authorizations.
3. **Course Registration**
    - The system should conduct the course registration process in compliance with the department's rules.
    - Priority: High
    - Criticality: High
    - Risks: Violation of course registration rules
    - Reduce risks: Designing and verifying the registration process in accordance with department policies.
4. **Viewing and Registering for Courses**
    - Users should be able to view courses and register for them.
    - Priority: High
    - Criticality: Medium
    - Risks: Users registering for the wrong courses or being inadequately informed.
    - Reduce risks: Minimizing errors by using a user-friendly interface and authentication processes.
5. **User Authorization**
    - The system should perform the necessary processes for user authorization.
    - Priority: High

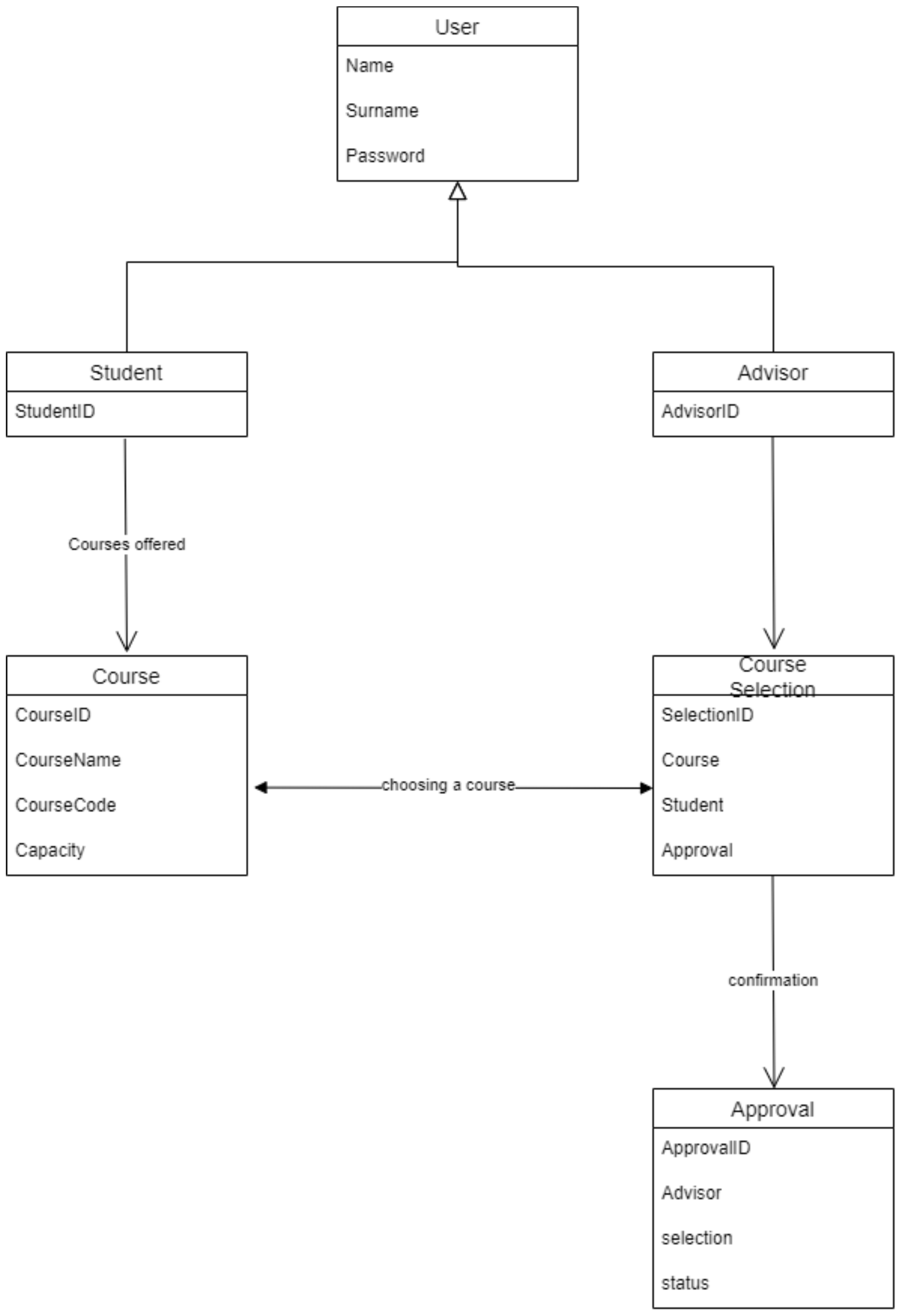
- Criticality: Medium
- Risks: Incorrect users gaining system access.
- Reduce risks: Ensuring secure operations and complete implementation of authentication processes.

6. **Course Change**
    - Students should be able to change the courses they have registered for within a specific time frame.
    - Priority: Medium
    - Criticality: Medium
    - Risks: Incorrect management of course change processes.
    - Reduce risks: Automating and verifying the change processes.
7. **Course Statistics**
    - The system should generate and display statistics for courses to users.
    - Priority: Low
    - Criticality: Medium
    - Risks: Inaccurate or incomplete statistical calculations.
    - Reduce risks: Automating and verifying statistical calculations.

Non-Functional Requirements

1. **Performance:**
    - The performance of the application is crucial, especially to ensure that users can perform operations in the system swiftly. Metrics such as response times, processing speeds, and startup times can be used to measure performance. Techniques like code optimization and database optimization can enhance performance.
2. **Portability:**
    - The application should function seamlessly on different platforms, such as Windows, Linux, macOS. This entails writing platform-independent code, using appropriate tools and frameworks, and avoiding platform-specific code.
3. **Availability:**
    - The availability of the application should be ensured with uninterrupted operation and minimal downtime. Downtime should be maintained at an acceptable level, and backup systems for failovers within a specific time frame may be implemented.
4. **Security:**
    - The application must be protected against unauthorized access and breaches.
5. **Maintainability:**
    - Your code should be sustainable in the long term. Adhere to object-oriented programming (OOP) principles to make your code modular and comprehensible. This enables you to easily expand and reuse your code.

Domain Model



In this model:

- The "Student" and "Advisor" classes represent students and advisors, respectively.
- The "Course" class represents individual courses, and the "Course Selection" class represents the selections made by students for specific courses.
- The "Approval" class represents the approval status of course selections made by students, with the advisor's involvement in the approval process.
- This model allows students to make course selections, advisors to review and approve these selections, and tracks the approval status for each selection.

Use Cases

Use Case 1: User Authentication

Use Case Name: User Authentication
Purpose: To allow users to log into the system.
Actors:

- User (Student, Advisor)

Preconditions:

- The user must have a valid username and password assigned.

Main Flow:

1. User requests access to the system.
2. The system prompts the user for their username and password.
3. User enters the correct username and password.
4. The system validates the entered username and password.
5. If authentication is successful, the user logs into the system.

Alternative Flows:

- *Step 4:* If the entered username or password is incorrect, the system notifies the user of the incorrect login information and prompts the user to enter the correct details.

Use Case 2: Course Registration

Use Case Name: Course Registration
Purpose: To enable students to register for courses.
Actors:

- Student (Student)

Preconditions:

- The user must have successfully logged into the system.
- The user must have a valid student ID.

- Courses must have available and empty slots.

Main Flow:

1. User requests course registration.
2. The system displays a list of available courses and their vacancies.
3. User selects the desired course to enroll in.
4. The system checks the vacancy status of the selected course.
5. If there are available slots, the system registers the user for the course and updates the vacancy.
6. The system notifies the user of the successful registration.

Alternative Flows:

- *Step 5:* If the course is full, the system informs the user about the full capacity and allows them to choose another course.

Use Case 3: Transcript Update

Use Case Name: Transcript Update
Purpose: To allow students to update their transcripts.
Actors:

- Student (Student)

Preconditions:

- The user must have successfully logged into the system.
- The user must have a valid student ID.
- The transcript file belonging to the student (e.g., "150119055.json") must be readable.

Main Flow:

1. The system reads the student's transcript file.
2. The user updates past course and grade information (e.g., new courses taken in the current semester, pass/fail status).
3. The system writes the updated transcript information to the transcript file.
4. The system validates the updated transcript information and confirms the file update to the user.

Alternative Flows:

- *Step 2:* The user can add or modify course information as needed. The system can offer verification after each update. If the user declines verification, the system preserves the previous transcript information, canceling the operation.

Use Case 4: Course Creation and Management

Use Case Name: Course Creation and Management
Purpose: To enable staff members (such as lecturers and advisors) to create new courses, update course information, and manage course details.
Actors:

- Staff (Lecturer, Advisor)

Preconditions:

- Staff members must be authenticated and have appropriate permissions to create or update courses.
- The system should have a predefined list of available courses.

Main Flow:

1. Staff member selects the option to create or update a course.
2. System displays a form for entering course details (course code, title, description, maximum capacity, schedule, etc.).
3. Staff member enters the required course information.
4. System validates the entered information and checks for any conflicts with existing courses.
5. If the information is valid and there are no conflicts, the system creates or updates the course.
6. System notifies the staff member of the successful course creation or update.

Alternative Flows:

- *Step 4:* If the entered information is invalid or conflicts with existing courses (e.g., duplicate course code), the system displays an error message and prompts the staff member to correct the information.
- *Step 5:* If the maximum capacity of the course is exceeded, the system notifies the staff member about the capacity limit and prompts them to adjust the maximum capacity or schedule.

Use Case 5: User Role Management

Use Case Name: User Role Management
Purpose: To allow administrators to manage user roles and permissions within the system.
Actors:

- Administrator

Preconditions:

- Administrator must be authenticated and have appropriate permissions for user role management.

Main Flow:

1. Administrator selects the option to manage user roles.

2. System displays a list of existing users along with their roles.
3. Administrator selects a user and chooses to update their role.
4. System presents a list of available roles (e.g., student, advisor, lecturer) for selection.
5. Administrator selects the new role for the user.
6. System updates the user's role and notifies the administrator of the successful role change.

Alternative Flows:

- *Step 3:* If the administrator chooses to add a new user, the system prompts them to enter the user's details and assign a role.
- *Step 5:* If the selected role is not valid or conflicts with existing roles, the system displays an error message and prompts the administrator to select a valid role.

These additional use cases expand the functionality of the course registration system by incorporating features for course management and user role management.

Use Case 6: Grade Submission

Use Case Name: Grade Submission
Purpose: To allow lecturers to submit grades for students.
Actors:

- Lecturer (Staff)

Preconditions:

- The lecturer must be authenticated and have the necessary permissions.
- The course must have ended, and all assignments/exams must be graded.

Main Flow:

1. Lecturer selects the option to submit grades.
2. The system displays a list of students enrolled in the course and their current grades.
3. Lecturer enters the final grades for each student.
4. The system validates the entered grades and checks for any errors.
5. If grades are valid, the system updates the students' records with the new grades.
6. The system notifies the lecturer of the successful grade submission.

Alternative Flows:

- *Step 4:* If there are errors in the entered grades (e.g., out-of-range grades), the system displays an error message and prompts the lecturer to correct the grades.

Use Case 7: Course Withdrawal

Use Case Name: Course Withdrawal

Purpose: To enable students to withdraw from a course they have previously registered for.
Actors:

- Student (Student)

Preconditions:

- The user must have successfully logged into the system.
- The course withdrawal deadline has not passed.

Main Flow:

1. Student selects the option to withdraw from a course.
2. The system displays a list of the student's enrolled courses.
3. Student selects the course they want to withdraw from.
4. The system confirms the withdrawal request with the student.
5. If the student confirms, the system updates the course enrollment status, removes the student from the course, and updates the vacancy.
6. The system notifies the student of the successful course withdrawal.

Alternative Flows:

- *Step 4:* If the student decides not to withdraw, the system cancels the operation and returns to the course selection screen.

Use Case 8 Schedule Generation

Use Case Name: Schedule Generation
Purpose: To generate class schedules for the upcoming semester based on course offerings and faculty availability.
Actors:

- Administrator (Staff)

Preconditions:

- The administrator must be authenticated and have the necessary permissions.
- Course offerings and faculty availability must be available in the system.

Main Flow:

1. Administrator selects the option to generate schedules.
2. The system fetches the list of available courses and faculty schedules.
3. Administrator configures schedule parameters (e.g., class times, days, maximum number of classes per day).
4. The system generates tentative schedules based on the configured parameters.
5. Administrator reviews the generated schedules and makes adjustments if necessary.

6. Once satisfied, the system finalizes the schedules and assigns classrooms.
7. The system notifies the faculty and students about their class schedules.

Alternative Flows:

- *Step 5:* If the generated schedules have conflicts or do not meet the requirements, the administrator can manually edit the schedules before finalizing them.

Use Case 9: Course Evaluation

Use Case Name: Course Evaluation
Purpose: To allow students to provide feedback and evaluate courses at the end of the semester.
Actors:

- Student (Student)

Preconditions:

- The student must have successfully logged into the system.
- The course must have ended, and the evaluation period must be open.

Main Flow:

1. Student selects the option to evaluate a course.
2. The system displays a list of courses the student was enrolled in during the semester.
3. Student selects a course to evaluate.
4. The system presents a questionnaire or feedback form related to the course.
5. Student provides ratings and comments based on their experience.
6. The system stores the evaluation responses anonymously.
7. The system notifies the student that the evaluation has been successfully submitted.

Alternative Flows:

- *Step 5:* If the student chooses not to provide feedback on a specific question, they can skip it and proceed with the rest of the evaluation form.

Use Case 10: User Account Management

Use Case Name: User Account Management
Purpose: To allow administrators to manage user accounts, including creation, modification, and deletion.
Actors:

- Administrator (Staff)

Preconditions:

- The administrator must be authenticated and have the necessary permissions.
- User account details must be available and accessible.

Main Flow:

1. Administrator selects the option to manage user accounts.
2. The system displays a list of existing user accounts.
3. Administrator selects a user account to manage (create, edit, or delete).
4. If creating a new account, the system prompts the administrator to enter user details (username, password, role, etc.).
5. If editing an existing account, the system allows the administrator to modify user details.
6. If deleting an account, the system prompts the administrator for confirmation.
7. The system updates the user account information based on the administrator's actions.
8. The system notifies the administrator of the successful account management operation.

Alternative Flows:

- *Step 4:* If creating a new account, the system can enforce password complexity rules and username uniqueness. If the entered details do not meet the requirements, the system displays an error message and prompts the administrator to correct the information.

System Sequence Diagram (SSD)

Use Case Name: Course Registration

Purpose: To enable students to register for courses.

Actors:

- Student (Student)

Preconditions:

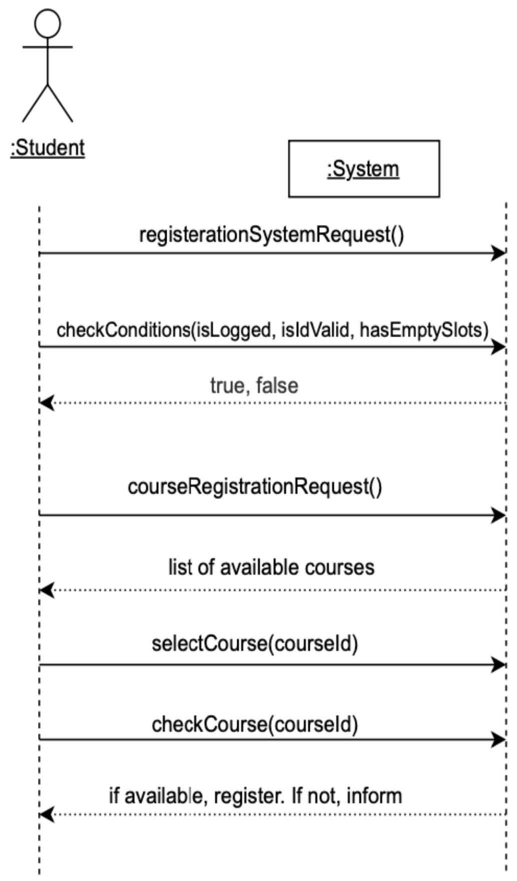
- The user must have successfully logged into the system.
- The user must have a valid student ID.
- Courses must have available and empty slots.

Main Flow:

1. User requests course registration.
2. The system displays a list of available courses and their vacancies.
3. User selects the desired course to enroll in.
4. The system checks the vacancy status of the selected course.
5. If there are available slots, the system registers the user for the course and updates the vacancy.
6. The system notifies the user of the successful registration.

Alternative Flows:

- *Step 5:* If the course is full, the system informs the user about the full capacity and allows them to choose another course.


:Student
:System
registerationSystemRequest()
checkConditions(isLogged, isIdValid, hasEmptySlots)
true, false
courseRegistrationRequest()
list of available courses
selectCourse(courseId)
checkCourse(courseId)
if available, register. If not, inform